Deliverable D3.1

Early SmartCLIDE Cloud IDE Design

WP 3

Project Acronym & Number:	SmartCLIDE – GA 871177
Project Title:	Smart Cloud Integrated Development Environment supporting the full-stack implementation, composition and deployment of data-centered services and applications in the cloud
Status:	Draft for the internal review
Dissemination Level:	Public
Authors:	CERTH
Contributors:	All Partners
Document Identifier:	D3.1 Early SmartCLIDE Cloud IDE Design
Date:	29.09.2021
Revision:	1.0
Project website address:	www.smartclide.eu

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No *871177*

Project Partners

Institut für angewandte Systemtechnik Bremen GmbH (ATB), Germany
Intrasoft International SA (INTRA), Luxembourg
Fundacion Instituto Internacionale de Investigacion en Intelligencia Artificial y Ciencias de la Computacion (AIR), Spain
University of Macedonia (UoM), Greece
Ethniko Kentro Erevnas Kai Technologikis Anaptyxis (CERTH), Greece
X/OPEN Company Limited (TOG), United Kingdom
Eclipse Foundation Europe GMBH (ECLIPSE), Germany
Wellness Telecom SL (WT), Spain
Unparallel Innovation LDA (UNP), Portugal
CONTACT Software GmbH (CONTACT), Germany
Kairos Digital, Analytics and Big
Data Solutions SL (KAIROS DS), Spain

Dissemination Level

PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document Control

Version	Notes	Date
0.1	Creation of the document	25.03.2021
0.2	End-to-end demonstration use case	15.04.2021
0.3	First inputs from technology providers	10.06.2021
0.4	Components diagrams and corresponding documentation templates	20.07.2021
0.5	Design Approach Completed	25.08.2021
0.6	Runtime Monitoring and Simulation Updates	26.08.2021
0.7	DLE, Smart Assistant and Service Discovery Update	07.09.2021
0.8	Document circulated for the internal review	08.09.2021
0.9	Service Deployments and Runtime Simulation and Monitoring Console input included	25.09.2021
1.0	Review comments addressed and document finalized for submission	29.09.2021

Abbreviations

AA	Audit Agent
ABRV	Assumption-based Runtime Verification
ACID	Atomicity, Consistency, Isolation and Durability
AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interfaces
AWS	Amazon Web Service
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
BoW	Bag of Words
BPMN	Business Process Model and Notation
CD	Continuous Delivery
CI	Continuous Integration
CLI	Command Line Interface
CM	Create Model
CNN	Convolutional Neural Network
CPS	Create Property Specification
CSV	Comma-separated Values
DL	Deep Learning
DLE	Deep Learning Engine
DoA	Description of Action
e.g	exempli gratia = for example
EPP	Event Processing Point
ERP	Event-Response Packages
FN	False Negative
FP	False Positive
GA	Grant Agreement
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IAM	Identity and Access Management
IDE	Integrated Development Environment
ISS	Import Service Specification
JSON	JavaScript Object Notation
LA	Logging Agent
LSP	Language Server Protocol
LSTM	Long Short-Term Memory
LTL	Linear Temporal Logic
M	Month
MC	Monitor Creation
MCAPI	Monitor Creation API
MDA	Model-Driven Architecture
MEAPI	Monitor Event API
MF	Monitoring Framework
ML	Machine Learning
MOM	Message-Oriented Middleware
MQTT	Message Queuing Telemetry Transport
MRAAPI	Monitor Request and Administration API
MT	SVM Model Templates

MVC	Model-View-Controller
NA	Notification Agent
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NN	Neural Network
OSGi	Open Service Gateway Initiative
PaaS	Platform as a Service
PST	Property Specification Templates
QoS	Quality of Service
QSA	Quantitative Security Assessment
R	Recall
REST	Representational State Transfer
RMI	Remote Method Invocation
RMV	Runtime Monitoring and Verification
RNN	Recurrent neural Network
SaaS	Software as a Service
SDLC	Software Development Life Cycle
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SSAS	Security-related Static Analysis Subcomponent
SSH	Secure Shell
STOMP	Streaming Text Oriented Messaging Protocol
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
UI	User Interface
UML	Unified Modelling Language
URL	Uniform Resource Locator
VP	Vulnerability Assessment
VPM	Vulnerability Assessment Models
VS	Visual Studio
WSDL	Web Services Description Language
XML	Extensible Markup Language

Executive Summary

Deliverable "D3.1 Early SmartCLIDE IDE Design" is part of WP3 and is produced as the main outcome of Task 3.1 "Design, development and unit testing of the User Interface", Task 3.2 "Design, development and unit testing of the Deep Learning Engine", and Task 3.3 "Design, development and unit testing of the Backend Services". The main objective of WP3 is to design, develop, and unit test the three main SmartCLIDE framework components, namely User Interface (UI), Deep Learning Engine (DLE), and the Backend Components. The purpose of this deliverable is to report the early design approach and technical progress that has been conducted until M20. The emphasis is given on the front and backend of the SmartCLIDE Integrated Development Environment (IDE). This deliverable is based on the outcome of previous WP1 deliverables, namely "D1.4 the SmartCLIDE Concept" and "D1.5 Design of SmartCLIDE Architecture". Besides, there is a strong link with WP2, where technology providers have performed research-oriented tasks, which have been documented in "D2.1 SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition, and Deployment". This is the first version of deliverable, which is going to be updated in M30.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Document Purpose

Deliverable "D3.1 Early SmartCLIDE IDE Design" is part of WP3, and is produced as the main outcome of:

- Task 3.1 "Design, development and unit testing of the User Interface",
- Task 3.2 "Design, development and unit testing of the Deep Learning Engine", and
- Task 3.3 "Design, development and unit testing of the Backend Services".

The main objective of WP3 is to design, develop, and unit test the three main SmartCLIDE framework components, namely User Interface (UI), Deep Learning Engine (DLE), and the Backend Components. The purpose of this deliverable is to report the early design approach and technical progress that has been conducted until M20. The emphasis is given on the front and backend of the SmartCLIDE Integrated Development Environment (IDE). This deliverable is based on the outcome of previous WP1 deliverables, namely "D1.4 the SmartCLIDE Concept"[1] and "D1.5 Design of SmartCLIDE Architecture"[2]. Besides, there is a strong link with WP2, where technology providers have performed research-oriented tasks, which have been documented in "D2.1 SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition, and Deployment"[3].

1.2 Approach

The early design of SmartCLIDE components presented in this deliverable is a result of the process already started in task "T1.4 Design of SmartCLIDE System Concept", and task "T1.5 Design of SmartCLIDE Architecture" as illustrated in Figure 1 below.

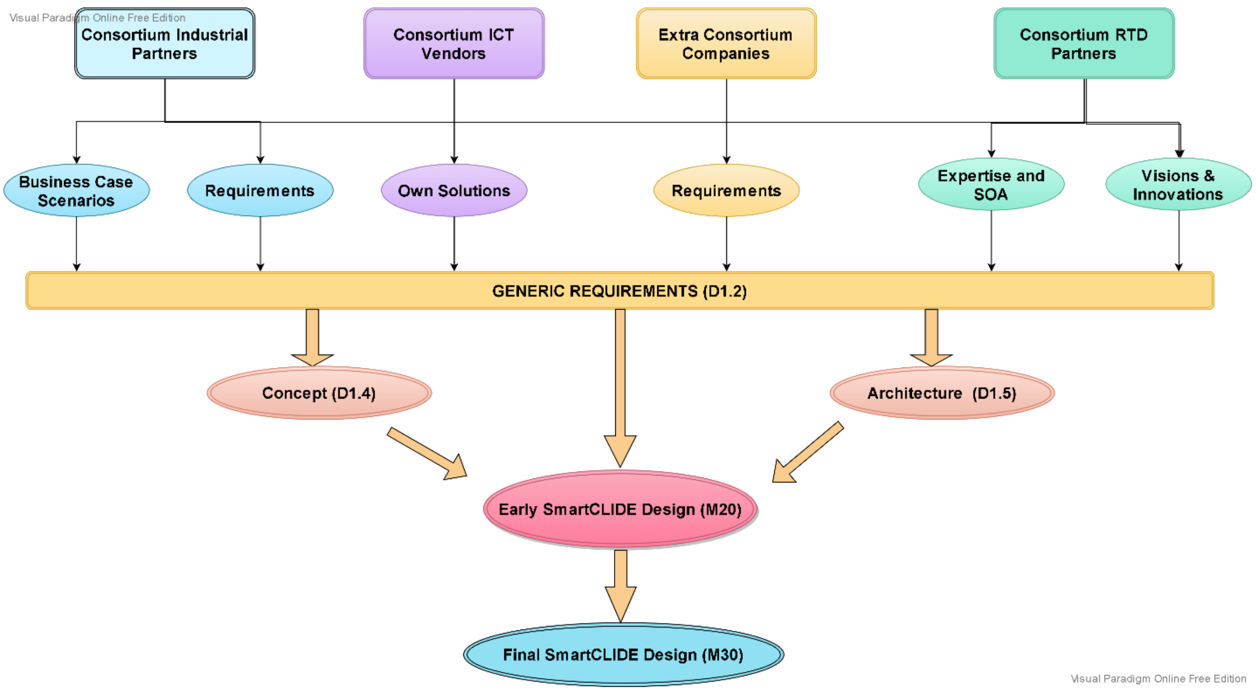

Visual Paradigm Online Free Edition
Consortium Industrial Partners
Consortium ICT Vendors
Extra Consortium Companies
Consortium RTD Partners
Business Case Scenarios
Requirements
Own Solutions
Requirements
Expertise and SOA
Visions & Innovations
GENERIC REQUIREMENTS (D1.2)
Concept (D1.4)
Architecture (D1.5)
Early SmartCLIDE Design (M20)
Final SmartCLIDE Design (M30)
Visual Paradigm Online Free Edition

Figure 1: Approach applied for early SmartCLIDE design

The approach we followed in this document is based on the foundation laid out in D1.5 and the introduced agile development approach [4]. Following a SCRUM-like process, we have organized regular bi-weekly Sprints between all developers to discuss the most important ongoing research and implementation issues. The scope of each Sprint was planned, prioritizing the most critical issues according to the initial timeline and microplanning performed at the beginning of the WP3 (i.e., M10). Initial microplanning was performed in collaboration with the WP2 leader to synchronize research-related tasks and efforts in WP2 and complementary design and implementation activity in WP3. Regular monthly WP2-WP3 calls were held to closely track the progress.

Following the good practice applied in D1.2, D1.3, D1.4, and D1.5, this document has been structured according to the concept and terms of the *ISO/IEC/IEEE 42010:2011* [5] . The overview of architectural views which are addressed in D1.2, D1.3, D1.4, and D1.5 is presented in Table 1.

Table 1: SmartCLIDE Architectural Views

	Architectural Views	Corresponding Deliverable
1	Functional Requirements View	Functional Requirements have been documented in D1.2 and mapped to SmartCLIDE system use cases in D1.3
2	Components Specification View	This view presents the detailed description of each component of the system. This view has been presented in D1.4 for each individual component focusing on description of core functionalities.
3	Operational Environment View	This view describes the conceptual way the system operates in its context, components that describe the system and user groups with which they interact. These topics have been discussed in D1.3 and D1.4. Besides, the Component Architecture Diagram of the overall SmartCLIDE system is included in D1.5.
4	Information Flow View	This view defines information entities and corresponding relationships between them. This view is one of the key outputs of the D1.5.
5	System Deployment View	This view will present the deployment view of the integrated system.

Keeping in mind that various design and architecture specifications have already been presented in WP1 SmartCLIDE deliverables, we focused on the detailed elaboration of *Components Specification*, *Operational environment*, and *Information Flow views*. The starting point of our analysis was the overall SmartCLIDE component diagram documented in the *D1.5 The SmartCLIDE Architecture* (see page 12, figure 2) [2]. Considering specific WP3 objectives, we have performed the following steps for the SmartCLIDE concept elaboration and early design specification:

- **Step 1: UI Design**
    - **Step 1.1:** Detailed analysis of UI Mock-ups from D1.4
    - **Step 1.2:** Specification of the end-to-end demonstration use case (in terms of Mock-ups)
    - **Step 1.3:** Analysis of the required external tools and plugins
    - **Step 1.4:** Theia design and small-scale prototype implementation (proof of concept)
- **Step 2: Detailed Specification of the Frontend & Backend Components**
    - **Step 2.1:** Detailed analysis of the D1.5 with particular focus on overall component and information flow diagrams
    - **Step 2.2:** Creation of the instructions and templates for the elaboration and design of individual components
    - **Step 2.3:** Collection of the relevant contributions from the following component providers
        - Run-time Simulation and Monitoring Console (WT)

        - Smart Assistant (AIR)
        - Deep Learning Engine (AIR)
        - Context Handling (ATB)
        - Source Code Repository (INTRA)
        - Discovery of Services and Resources (AIR)
        - Service Creation, Composition and Testing (UoM)
        - Run-time Monitoring and Verification (TOG)
        - Security (CERTH)
        - Message-Oriented-Middleware (CERTH)
        - Service Deployment (WT)
        - CI/CD Component (INTRA)
    - **Step 2.4:** Design of the overall SmartCLIDE detailed component diagram.

1.3 Document Structure

The document consists of the following sections:

- **Section 1** includes a concise overview of the overall content of the document
- **Section 2** presents User Interface design approach and gives an overview of the Run-time Simulation and Monitoring Console and Smart Assistant frontend components
- **Section 3** outlines Deep Learning Engine design approach with a particular focus on presentation of machine learning and deep learning algorithms and support for service classification. Besides, it presents updated design approach for the Context Handling component.
- **Section 4** presents the design approach and key features on the SmartCLIDE backend components.
- **Section 5** describes the overall unit testing approach
- **Section 6** provides the concluding remarks and outlines future steps.

1.4 Relation to other deliverables

The present deliverable completes the design and specification of the SmartCLIDE system together with D1.2, D1.3, D1.4, and D1.5. It receives input from all these deliverables and primarily from D1.5 using the overall SmartCLIDE component and information flow diagrams as a starting point for the analysis. One of the key outputs of the present deliverable is elaborated component diagrams based SmartCLIDE architecture, including the specification of provided/required interfaces. Besides, this deliverable is strongly linked to D2.1, where underlying research approaches for relevant components are discussed.

1.5 Contributors

All project partners have substantially contributed to this deliverable. In particular, the RTD partners and technology providers have provided detailed designs of their components and underlying services. UNP has led the overall UI design, while KAIROS has performed the Theia analysis and implemented a small-scale prototype. AIR has led the design effort of the DLE together with ATB, who supported the Context Handling part. CERTH has acted as overall editor in preparing each version of the document using a collaborative and iterative approach.

2 User Interface

This section describes the design progress of the main components of the User Interface of SmartCLIDE IDE integrating all the functionalities provided by SmartCLIDE technologies and exposing them as a development experience to developers.

2.1 Frontend User Interface

The specification and development of IDE Front-End is composed 2 processes**: Design Lifecycle** and **Project Implementation.**

The Design Lifecycle corresponds to the steps followed for the design of User Interfaces. The design life cycle started with the specification of the graphical elements that each SmartCLIDE technology will require to interact with users, which resulted in a set of wireframes (detailed description of the wireframes produced in this analysis are provided in Annex A). The purpose of the wireframes is to communicate the order, structure, layout, navigation, and organization of content rather than defining the final the graphical aspects. The next steps will consist of enhancing the wireframes with more detailed graphical elements to create the mock-ups. Mock-ups communicate the visual design aspects that wireframes do not. These include imagery, colour, and typography, giving a sense of what the design will look like and provide guidelines for the implementation of the visual aspects, as represented in Figure 2.

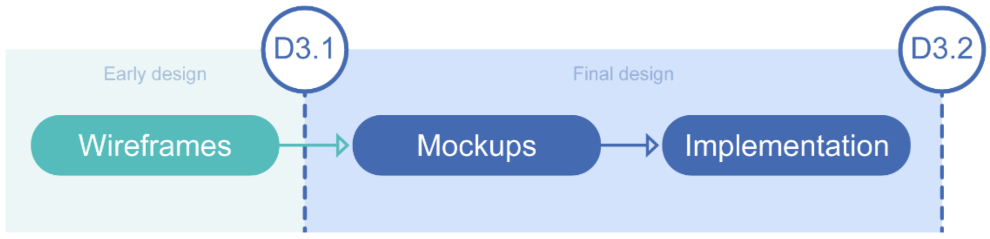


Figure 2: Design approach procedure

The Project Implementation process corresponds to the creation of the software project that will be used for the implementation of the IDE front-end and will be used by SmartCLIDE partners for the implementation of the base infrastructure of the IDE and all the functionalities that will be provided by the SmartCLIDE tools on the backend. This process also includes the definition of the IDE's image were the graphical tools used on the project were configured to present the defined image.

2.1.1 Design

2.1.1.1 Layout

The pages of SmartCLIDE IDE follow a common structure as represented in Figure 3. On the top of the page there are some graphical elements to provide a quick access to specific IDE functionalities. From left to right are presented the Logo of the IDE that redirects to the Dashboard, the Settings icon that provides access IDE settings and the User icon to provide access to user-specific settings.

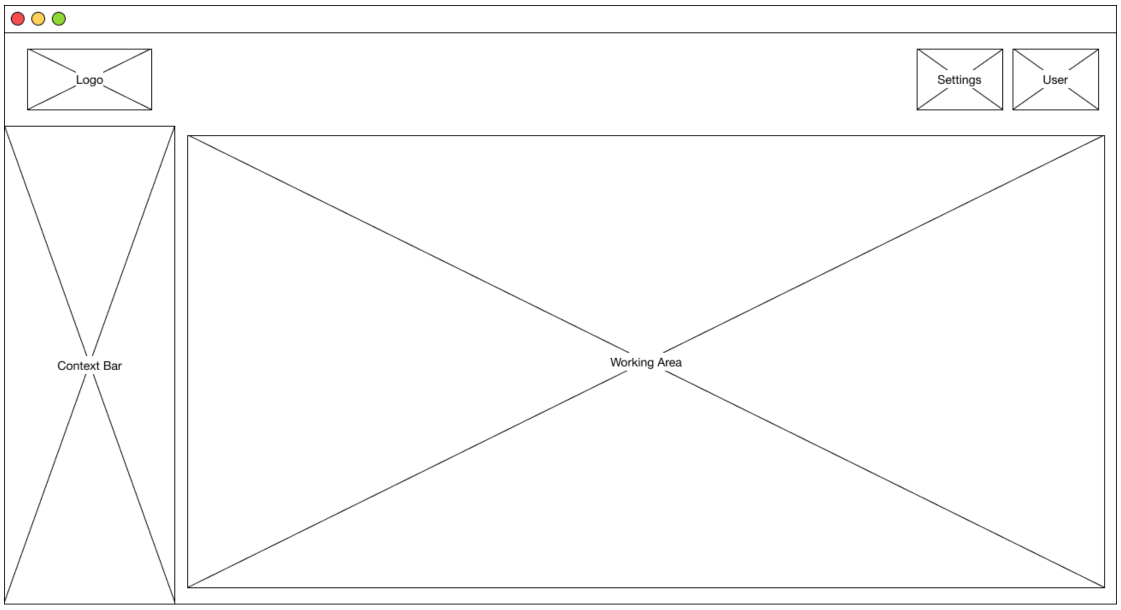


Figure 3: SmartCLIDE IDE base page

Below those elements, we can find 2 main areas: The Context Bar and the Working Area. Those elements are related in the sense that the Working Area presents the main content of the different pages of the SmartCLIDE IDE and the Context Bar adapts to the content presented in the Working area, showing a set of options that allow to navigate/initiate operations over the content shown.

On specific cases an extended view of this structure can be used. This structure, represented in Figure 4, add a new area – the Auxiliary Tools – that can be used to provide the user with specific user interfaces that allow to execute operations related with the content on the Working Area.

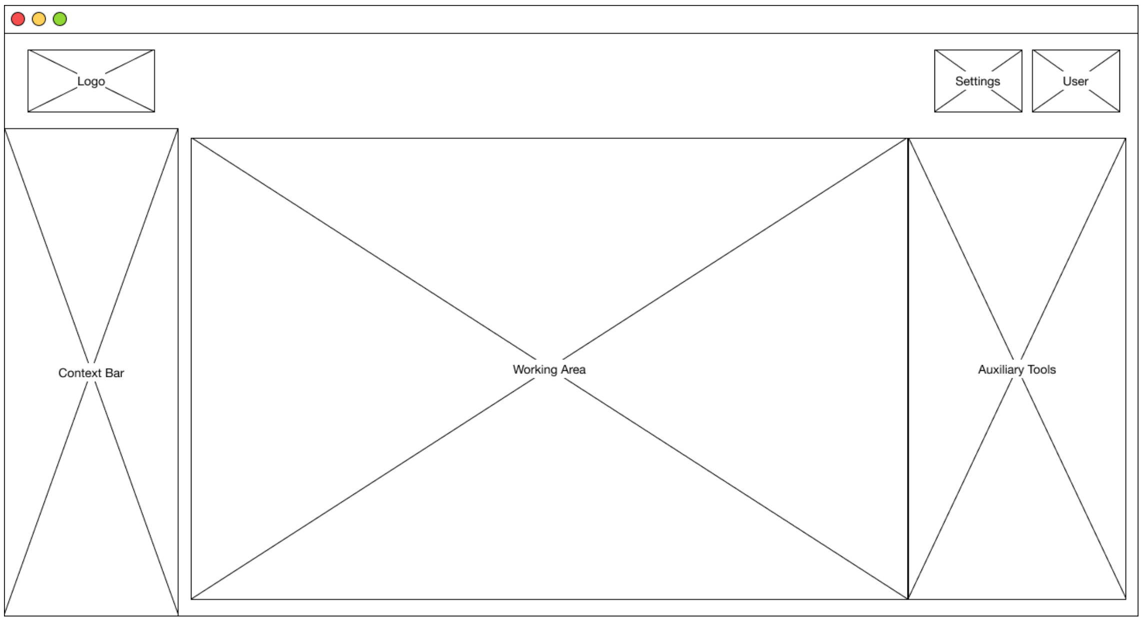


Figure 4: SmartCLIDE IDE base page with auxiliary tools

The concept of Workbench initial defined matches the mainly the concept of Working Area of the layouts, as it is the main area where developers are expected to develop their applications. The concept of Toolbox, in the sense that it provides to the user the resources and main functionalities of the SmartCLIDE tools, is spread across several areas, resulting from the integration of SmartCLIDE backend functionalities on different areas and element of the IDE frontend.

2.1.1.2 Wireframes

The front-end user interface was structured considering the three main pillars for the SmartCLIDE IDE front-end which represent the main types of projects that can be created:

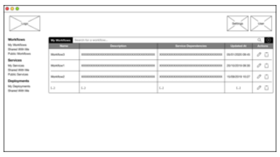
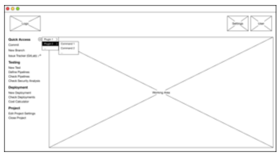
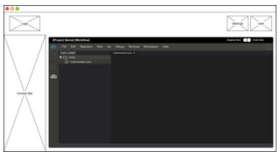
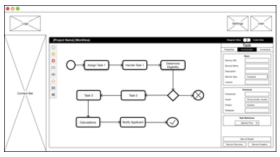
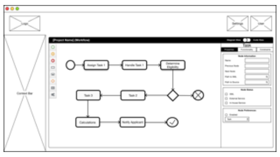
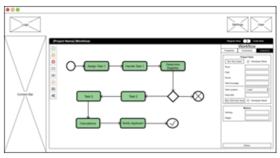
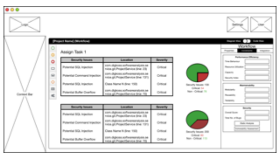
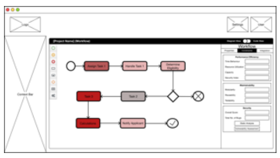
- Services – applications with a well-defined interface developed independently of external entities (e.g., other services).
- Workflows – consist of the graphical representation of a system through diagrams, where each node of the diagram represents a task than can be performed by a service.
- Deployments – configurations of deployments of services or workflows that run on remote environments (e.g., Amazon Web Services).

Aside these 3 pillars, SmartCLIDE IDE must also provide other functionalities that will support the activities and experience to the users. Those functionalities correspond to the:

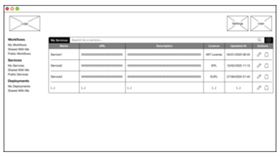
- Dashboard – Customizable welcome page that the IDE user can organise to show him/her relevant information.
- IDE settings – Set of options to customize the IDE settings and to monitor the runtime conditions of the SmartCLIDE IDE
- User settings -Set of option to defined user-specific settings and configuration of aspects of the user development space.

An overview of wireframes associated to the previous functionalities are shown in Table 2. More details about each wireframe can be found in Annex A.

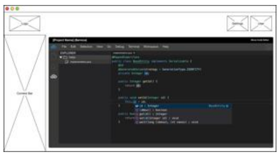
Table 2: An overview on the wireframes of the SmartCLIDE IDE

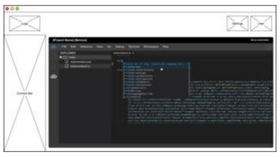
Workflows			
			
The main page of the workflows	**Workflow configuration page**	**An instance of the diagram editor**	**Adding a new shortcut to "Quick Access"**
			
An instance of the Theia code editor	**Task Functionality within the diagram editor**	**Task Properties within the diagram editor**	**Service discovery configuration page**
			
No results found by the service discovery	**Workflow integration in the diagram editor**	**Security analysis page**	**Vulnerability assessment page**

Services


The main page of the services

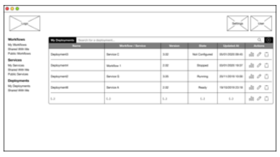
The main configuration page of a service


Smart Assistant services: Code autocompletion

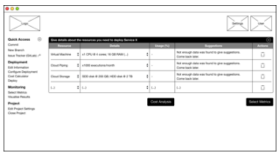

Smart Assistant services: Live template recommendation

Smart Assistant services: Comments generation

Deployments


The main page of the deployments

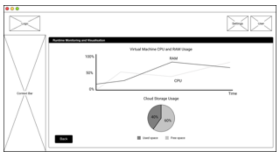
The main configuration page of a deployment


The deployment configuration page

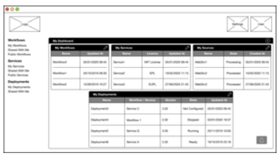
Main page of the cost comparison service

The deployment configuration suggestions

Runtime metrics selection page

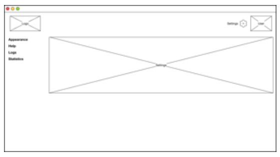

Runtime metrics monitoring and visualization page

Dashboard


The main page of the SmartCLIDE platform

The content configuration page of the dashboard

IDE Settings

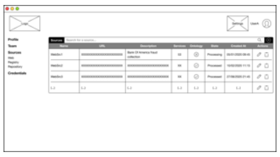


Settings Page

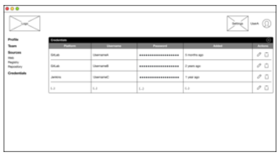
User Settings

The user profile page

The team configuration page



Sources for services



User credentials management page

As part of the validation process of the behaviour and navigation described on the wireframes, a potential usage scenario in which the user (a developer) takes advantage of a wide range of SmartCLIDE functionalities to implement test and deploy a workflow was defined and analysed against the wireframes. This potential usage scenario is detailed in Annex B.

2.1.2 Technical approach

In this section are identified the technical approaches and technologies selected for to start of the development of the SmartCLIDE IDE frontend.

2.1.2.1 Modular Development Approach

The functionalities will be added to SmartCLIDE IDE using a plugin-like approach, enabling functionalities to be developed independently. With this modular, each plugin will use specific interfaces provided by SmartCLIDE IDE, as well as implement interfaces defined by SmartCLIDE IDE to support the correct integration of the development functionalities into the IDE.

The main interfaces that the developed modules have access to expose their functionalities are:

- Working Area interfaces – interface to render the specific content on the working area and listen to user input events.
- Interfaces Context bar integration – Expose endpoints that are connected to the options on the context bar defining the behaviour of the options listed.
- Interface for Auxiliary tools – Define the user interfaces for the extra tools that will be used by the IDE users.
- Interface for Home Dashboard card integration – Interface the describe the widgets/cards that can be visualised in the Dashboard.
- Interface with Quick Access shortcut – Interface to describe the commands that can be inserted in the Quick Access menu, providing shortcuts that the developer can use to optimise his development flow.

All the functionalities that require the definition of the User Interfaces should have a harmonised style to provide the IDE user with a smother experience. To achieving this goal, developers of modules that provide user interfaces are recommended to use SmartCLIDE style theme.

2.1.2.2 Support Technologies

The objective for SmartCLIDE Cloud IDE is to create a website intuitive for any user, regardless his/her experience and role. The IDE should also present an appealing look and feel and extensible to support the usage and integration with external frameworks and libraries that can enhance the IDE functionalities and performance. As a cloud IDE, SmartCLIDE IDE must allow multiple users to use it on their preferred internet browsers, allowing users to use remotly the typical IDE functionalities, supported and enhanced by SmartCLIDE tools. With that in mind, the architecture from Figure 5 was designed.

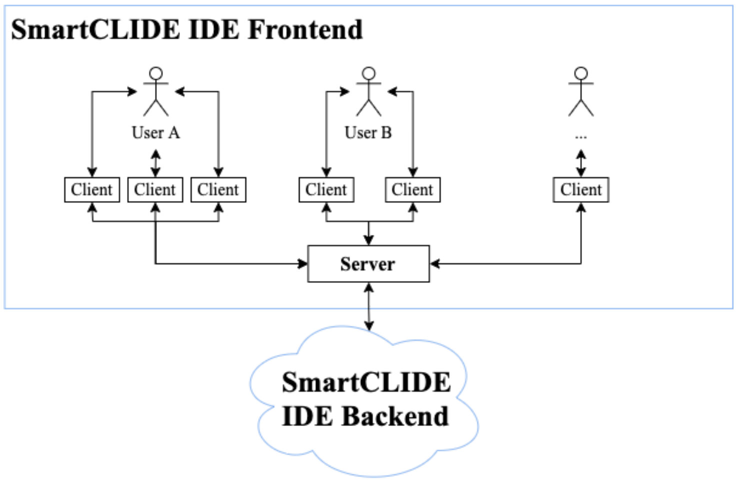


Figure 5: The architecture of the SmartCLIDE IDE

This diagram focuses on the frontend of the SmartCLIDE IDE, which needs to support the management several user authentications simultaneously (e.g., User A, User B, etc.). Moreover, each user may access the IDE in more than one browser tab at the same time, as represented in the cases of User A and User B where these users may be working multiple projects at the same time. All these clients are connected to the same server, requiring the server to be able to associate each client instance (each browser tab) to different sessions and allowing to separate the context across the different client instances to provide them with the required independency. From a higher-level perspective, the frontend directly communicates with the backend of the IDE, which consists of all the SmartCLIDE tools that support the functionalities of the IDE and are deployed somewhere in the Cloud.

For the implementation of the Server component of the Frontend it was decided to use the Meteor[1] framework. This full-stack framework provides mechanisms that runs on the Client-side and on Server-side of a Web application that supports the management of authentication process of multiple users, and automatically creates contextualised sessions for each client instance where the same user is authenticated.

Moreover, Meteor also allow to expose a set of methods to the clients that are executed on the server-side. This mechanism can be used on expose an API to all clients for the interactions with the SmartCLIDE IDE backend. This mechanism supports development of a modular approach by providing a common API that all the modules can use for providing their functionalities to the IDE frontend, while abstracting their code from the management of the users and sessions.

Meteor can be complemented with the usage of frontend framework to increase the power on implementation of the User Interface. The framework Vue.js[2] was selected to assume the role. Vue.js provides many modern concepts like code encapsulation and reuse thought the usage of components and a reactivity system to optimise the re-rendering of objects. Vue.js distinguish between most of its peers by allowing supporting the creation of Vue components using mostly technologies commonly used on Web development: HTML, CSS and JavaScript. This reduces learning curve to start developing components using Vue.js, which is an important requirement for ensuring a smooth development on a team where members have different backgrounds.

SmartCLIDE IDE will require a code editor to support the activities to the developers. Eclipse Theia was select to play this role, providing not only a modern text editor but also an ecosystem of plugins that can be

[1] https://www.meteor.com
[2] https://vuejs.org

used to improve the whole developing lifecycle. In annex C is presented a study about Eclipse Theia main features and approaches to develop plugins and extensions.

To support the development of responsive and visual harmonised interfaces a Bootstrap[3] theme was configured and added to the IDE project.

2.1.3 Image

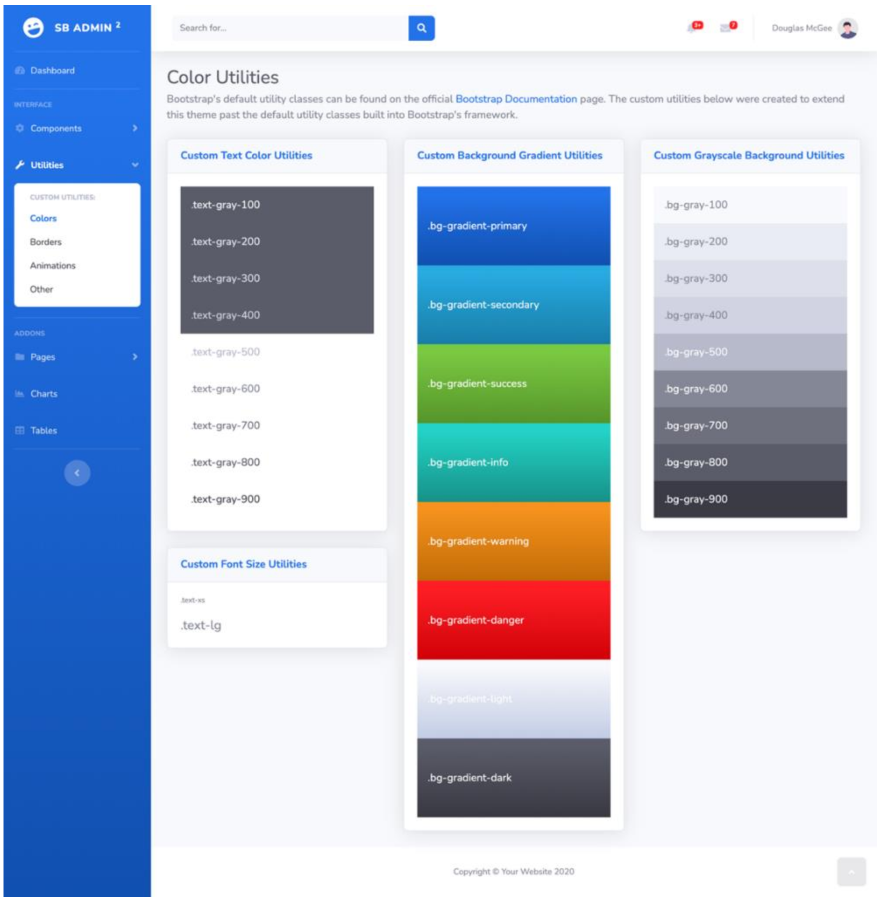
The SmartCLIDE project image was the inspiration for the identification the IDE image. The colour scheme used in the logo and project website and the cloud icon with the round look was used to define the colour and graphical aspect of the element in the IDE.

2.1.3.1 IDE style

2.1.3.1.1 Colours

The SmartCLIDE font colour (blue) and the second colour in the icon (aqua) were the starter in the theme building. They were stablished as the main and one of the secondary colours. We added some more colour pallet so it can be used as the functions needed (e.g., red for delete, orange for warning, etc.).

To test, we implemented the theme style into a GitHub Project Demo[4], where we can see how the behaviours and colours of the several elements play along. The aspect is shown in Figure 6.



Bootstrap's default utility classes can be found on the official Bootstrap Documentation page. The custom utilities below were created to extend this theme past the default utility classes built into Bootstrap's framework.

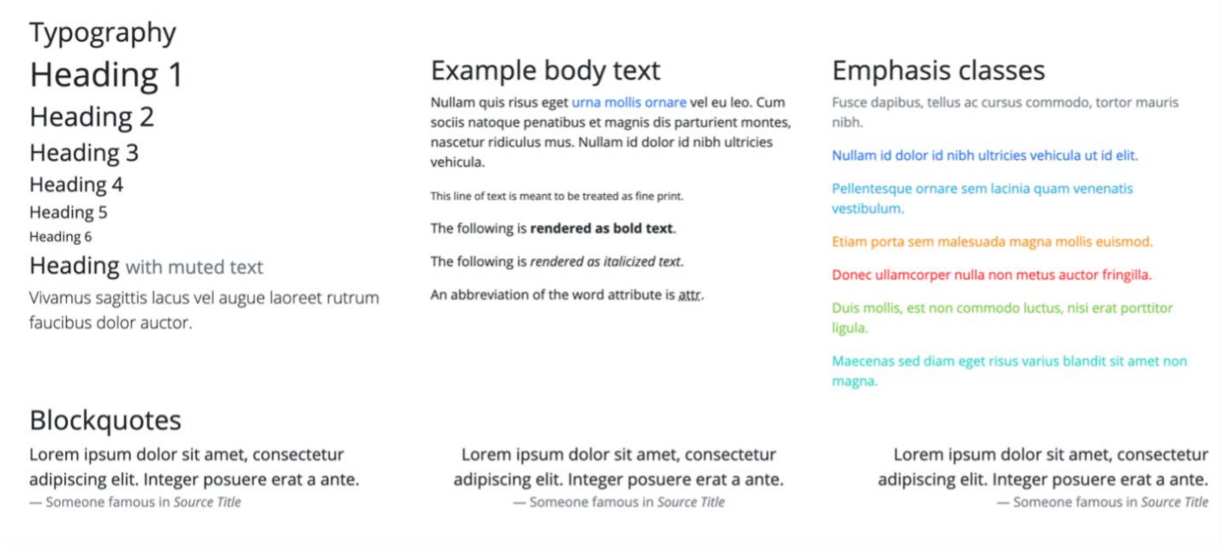
Figure 6: The color pallet of the SmartCLIDE IDE

---

[3] https://getbootstrap.com
[4] https://github.com/StartBootstrap/startbootstrap-sb-admin-2

2.1.3.1.2 Fonts

For the font, the Open Sans family was select to harmonise with the font used in the project website. The default font sizes were kept.



Typography

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Heading with muted text

Vivamus sagittis lacus vel augue laoreet rutrum faucibus dolor auctor.

Example body text

Nullam quis risus eget urna mollis ornare vel eu leo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam id dolor id nibh ultricies vehicula.

This line of text is meant to be treated as fine print.

The following is **rendered as bold text**.

The following is *rendered as italicized text*.

An abbreviation of the word attribute is attr.

Emphasis classes

Fusce dapibus, tellus ac cursus commodo, tortor mauris nibh.

Nullam id dolor id nibh ultricies vehicula ut id elit.

Pellentesque ornare sem lacinia quam venenatis vestibulum.

Etiam porta sem malesuada magna mollis euismod.

Donec ullamcorper nulla non metus auctor fringilla.

Duis mollis, est non commodo luctus, nisi erat porttitor ligula.

Maecenas sed diam eget risus varius blandit sit amet non magna.

Blockquotes

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.
— Someone famous in *Source Title*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.
— Someone famous in *Source Title*

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.
— Someone famous in *Source Title*

Figure 7: Fonts of the SmartCLIDE IDE

2.1.3.1.3 Buttons

The overall aspect of the buttons are aligned with the default settings however, a customisation was made to increase the round aspect in the corners of the buttons.

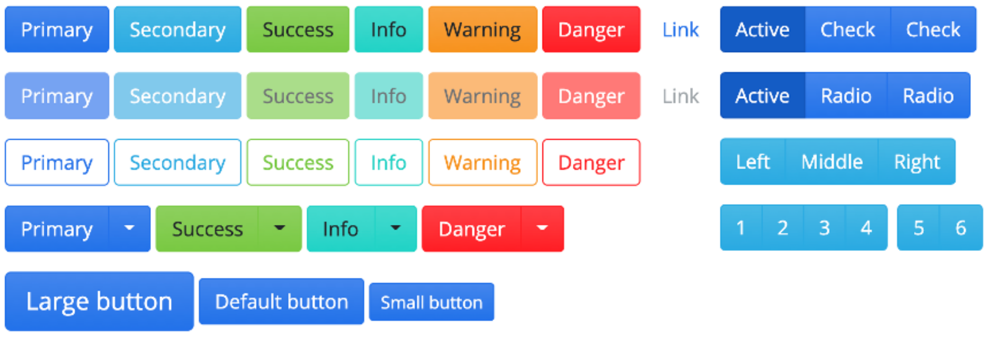
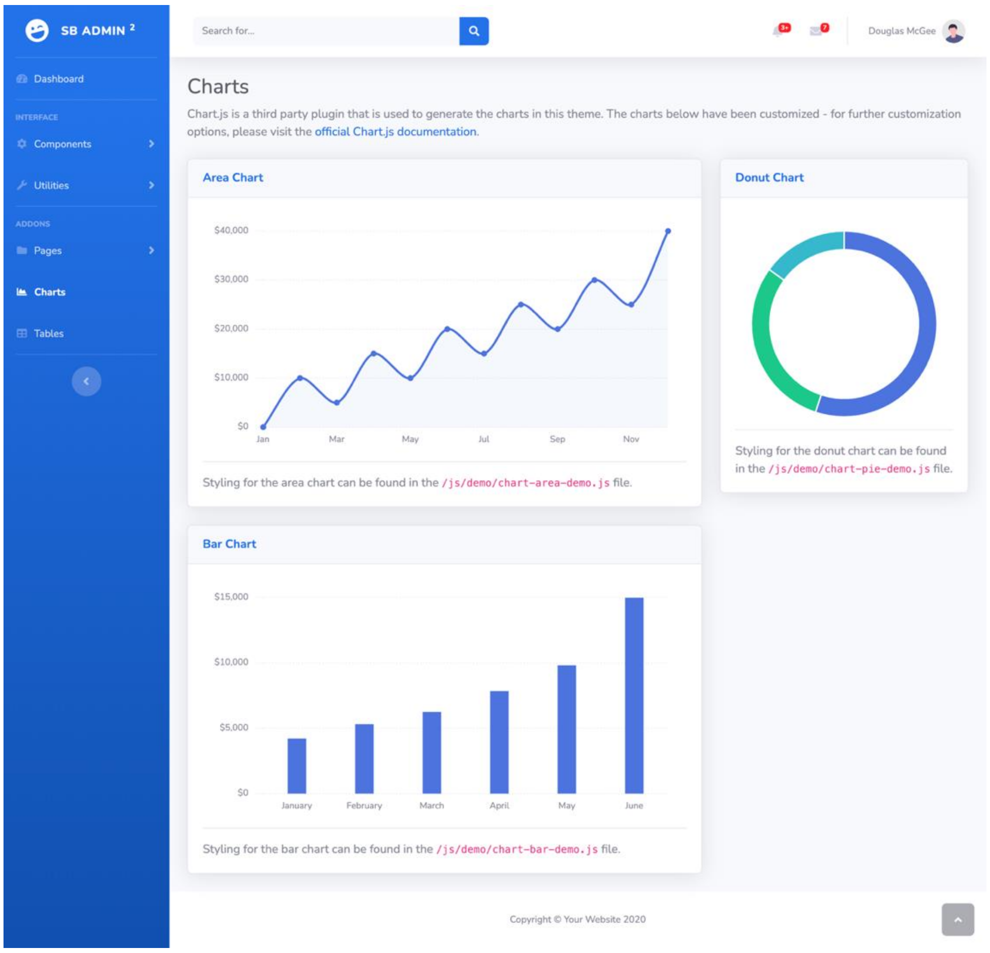


Figure 8: Buttons of the SmartCLIDE IDE

2.1.3.1.4 Guidelines for Graphs & Charts

In the graphs and charts, the recommended colours are two colours present in the logo (blue and aqua). If the chart needs more colours, the secondary colours can be used. Some examples are presented in Figure 9.



Charts

Chart.js is a third party plugin that is used to generate the charts in this theme. The charts below have been customized - for further customization options, please visit the official Chart.js documentation.

Styling for the area chart can be found in the `/js/demo/chart-area-demo.js` file.

Styling for the donut chart can be found in the `/js/demo/chart-pie-demo.js` file.

Styling for the bar chart can be found in the `/js/demo/chart-bar-demo.js` file.

Copyright © Your Website 2020

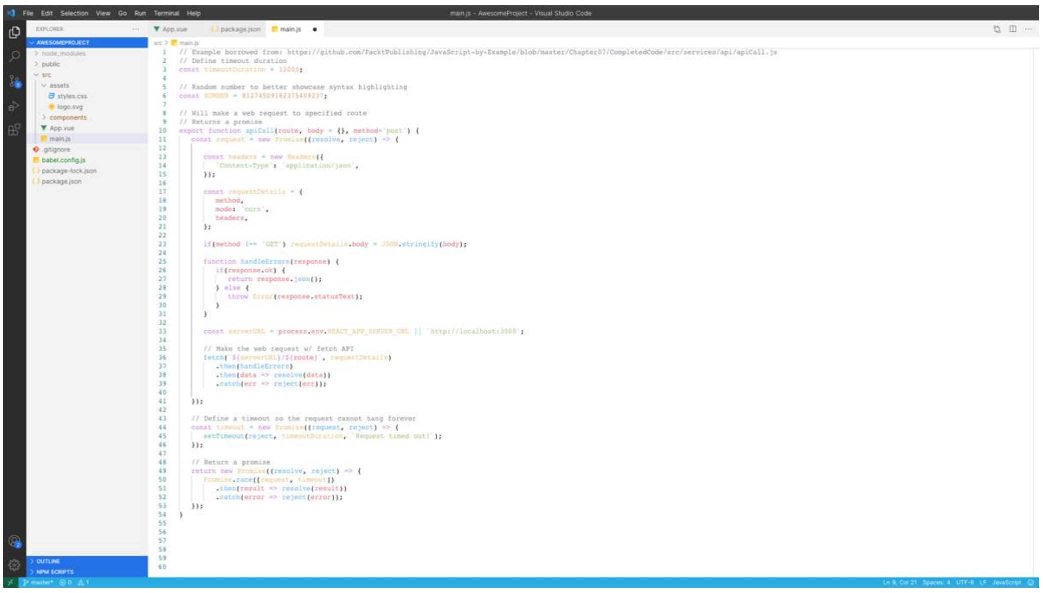
Figure 9: Graphs and charts of the SmartCLIDE IDE

2.1.3.2 Integrated Code editor

The same image concept was applied to the Theia editor through the definition of a Theia theme. Since Theia themes are compatible with the themes of VS Code, a VS Code theme editor[5] was used to define the SmartCLIDE Theia theme.

The colours of All Visual Studio Code sections (e.g., General, Activity Bar, Side Bar, Status Bar, Buttons, Widgets, etc.) where customised to align with the colours defined in the Bootstrap theme. The result of the SmartCLIDE Theia theme is presented in Figure 10.

[5] https://themes.vscode.one



```
// Example borrowed from: https://github.com/PacktPublishing/JavaScript-by-Example/blob/master/Chapter07/CompletedCode/src/services/api/apiCall.js
// Define timeout duration
const timeoutDuration = 12000;

// Random number to better showcase syntax highlighting
const NUMBER = 81274509182375409237;

// Will make a web request to specified route
// Returns a promise
export function apiCall(route, body = {}, method='post') {
  const request = new Promise((resolve, reject) => {

    const headers = new Headers({
      'Content-Type': 'application/json',
    });

    const requestDetails = {
      method,
      mode: 'cors',
      headers,
    };

    if(method !== 'GET') requestDetails.body = JSON.stringify(body);

    function handleErrors(response) {
      if(response.ok) {
        return response.json();
      } else {
        throw Error(response.statusText);
      }
    }

    const serverURL = process.env.REACT_APP_SERVER_URL || `http://localhost:3000`;

    // Make the web request w/ fetch API
    fetch(`${serverURL}/${route}`, requestDetails)
      .then(handleErrors)
      .then(data => resolve(data))
      .catch(err => reject(err));

  });

  // Define a timeout so the request cannot hang forever
  const timeout = new Promise((request, reject) => {
    setTimeout(reject, timeoutDuration, `Request timed out!`);
  });

  // Return a promise
  return new Promise((resolve, reject) => {
    Promise.race([request, timeout])
      .then(result => resolve(result))
      .catch(error => reject(error));
  });
}
```

Figure 10: Theia theme of the SmartCLIDE I

2.2 Run-time Simulation and Monitoring Console

2.2.1 Design Approach

Developing, deploying, and monitoring complex systems such as the real-time communication platform we previously described using containers is a tough task, but thanks to SmartCLIDE, developers will be able not only to deploy it, but also to get:

- A better comprehension of real-time QoS constraints.
- A deeper insight into QoS, by providing monitoring tools that will allow the developer to track quality of service and quality of experience.
- Easier management of deployed services.

In conclusion, SmartCLIDE will aid the developer throughout all the phases, such as development, testing, and deployment. Then, once deployed, the IDE (Integrated Development Environment) will supply visual monitoring tools to manage and extend running capabilities, also providing a detailed analysis about QoS.

The "Run-time Simulation & Monitoring / Visualization" component will provide a front-end for monitoring any of the containers, services and complex systems deployed with SmartCLIDE. Notice that a complex system may be composed by several containers, and each container might include more than one services that cooperate to solve a greater problem.

This interface will allow the developer to explore and visualize the status of deployed services through visual and text-based elements by using a console that will form part of the SmartCLIDE UI. To do so, it retrieves the monitoring data from the Run-time Monitoring & Verification component by using a Pub-Sub paradigm.

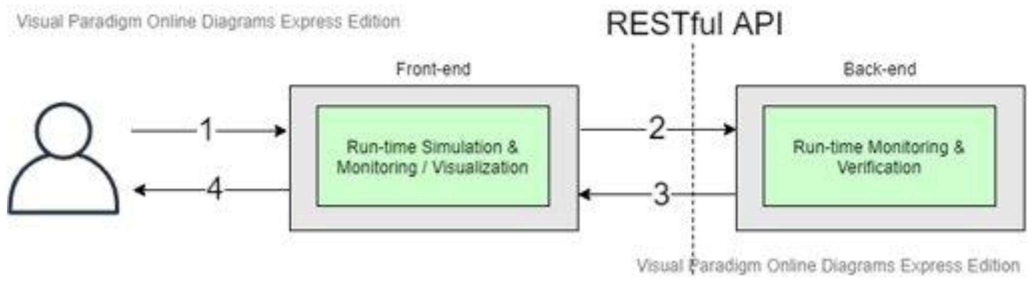

Visual Paradigm Online Diagrams Express Edition
RESTful API
Front-end
Back-end
1
2
3
4
Run-time Simulation & Monitoring / Visualization
Run-time Monitoring & Verification
Visual Paradigm Online Diagrams Express Edition

Figure 11: UML Run-Time Simulation & Monitoring / Visualization Component Diagram

For example, the figure in the slide represents a generic use case for this component:

- First, the developer selects the container he wants to monitor by using the "Run-time Monitoring and Simulation / Visualization" module in the SmartCLIDE UI.
- Then, the "Run-time Monitoring and Simulation / Visualization" component subscribes to the "Run-time Monitoring & Verification" component by using an API.
- Then, the "Run-time Monitoring & Verification" component reports monitoring data from that container.
- Finally, the "Run-time Monitoring and Simulation / Visualization" component represents received run-time monitoring data using both, visual and text-based techniques.

SmartCLIDE provides the capability for non-programmers to construct applications and new services using smart automation. Quality assurance of the constructed applications involves both design time and runtime assurance. Assurance of the runtime behaviour is addressed by validating that the application exhibits behaviour consistent with its specifications, and that the assumptions made about the runtime environment, made at the time of the design of the construction approach, and thus built into the construction of the application, continue to be valid as the application executes.

The intended behaviour of the application may involve both functional and non-functional properties. Key characteristics of the application, such as security, safety, privacy, resilience and reliability are general categories of runtime quality attributes that may be required.

Validity of environmental assumptions involves necessary conditions, under which the construction and execution of the application are expected to be correct. The assumptions made by individual components or services from which an application is composed, and the consequent assumptions of their composition, need to be established and maintained during the execution of the application.

These concerns occur in statically constructed applications as well as those that are dynamically constructed and executed on demand. However, in the dynamic case some runtime quality attributes arise that do not exist in the static case. For example, in the dynamic case the identification of the components or services to be composed, the correctness of the service composition, and testing of the composition arise during construction runtime rather than during a conventional design, implementation, test and deployment cycle.

When an application is dynamically composed for a specific purpose, known properties of the component services can be used to find composition solutions that yield a final property that suits the purpose. However, because the dynamic case does not afford the same opportunity to apply the conventional disciplines for assurance during design and implementation, or for conventional testing, the defining property of the application as well as the assumptions should continue to hold during runtime.

The Runtime Monitoring and Verification (RMV) framework is intended to provide the capability for constructed applications to be automatically monitored at runtime for their validity of their properties. If the SmartCLIDE services creation, services discovery, services composition, and services deployment functionalities are thought of as "programming for non-programmers", then the runtime monitoring and verification features provide integrated "runtime quality assurance for non-programmers".

Though there is some technical risk of falling short of the most ambitious objectives due to these uncertainties, the resulting monitoring framework will nonetheless provide several useful benefits:

- The Runtime monitoring and verification components are used by the Context Handling system to subscribe to get monitored data about runtime values and events that it needs to accomplish its purpose.
- Developers using the service composition capabilities of SmartCLIDE will be able to develop monitoring applications using the monitoring and sensor services that can detect sequences of events specified by patterns specified by the monitoring application. These may be used to detect exceptional conditions or may be integrated into the design of the application to enable a response to monitored conditions external to the application.
- The security component can construct a monitoring application to monitor security relevant events that other application services are designed to generate. These may be stored in the Log as an audit trail for post-execution forensic investigations.
- Other subsystems can construct monitoring applications to assist with instrumentation, testing, or debugging of applications or the SmartCLIDE platform itself.

The Runtime Monitoring and Verification (RMV) framework provides a flexible and configurable framework with programmable interfaces to permit the construction of a monitoring application, using standard or bespoke monitoring components and sensors, in parallel with the construction of a service-oriented application. The framework will also provide for the capture and retention of data gathered by sensors and monitoring applications in a log according to configurable criteria. The technical elements of the approach include:

- A library of monitoring logic components for constructing monitoring applications.
- A library of sensor types that can be instantiated and installed within composed SOA applications and infrastructure components to gather the data/events needed by monitoring applications.
- A mechanism for composing monitoring applications in parallel with the SOA application.
- An SOA execution framework for executing SOA applications, monitoring applications, the monitor framework, and the agents.
- A notification mechanism that can transmit alerts to registered participants when monitored events meet a specified condition.
- A method and mechanism for selecting monitoring components and sensors and constructing appropriate monitoring applications according to the construction of a corresponding SOA application.
- A mechanism for the collection and persistent storage of the Log of monitored data and notifications.
- A mechanism for the configuration of log collection and storage, including where the Log is to be stored, how large logs should be archived, etc.

The starting point for defining the concept of the “Runtime Monitoring and Verification” component is the requirements analysis document. SmartCLIDE “Runtime Monitoring and Verification” component shall be able to support:

- A flexible and configurable framework for the construction and deployment of diverse monitoring applications.
- Programmable responses according to the findings of the monitoring applications, including notifications to the application execution framework and activation of predefined responses.
- Programmer-directed explicit construction of customised monitoring applications.
- Cooperative monitoring, that is, passive monitors that are called by the application and/or the execution framework to report various predefined conditions.

SmartCLIDE “Runtime Monitoring and Verification” component should be able to support:

- Capture and storage and/or forwarding, according to configurable filters, of monitoring data to other components.

SmartCLIDE “Runtime Monitoring and Verification” component may be able to:

- Leverage the SmartCLIDE application service composition to automatically construct monitoring applications to run in tandem with the composed applications and to detect in the applications’ behaviour violations of the applications’ specifications.

2.2.2 Cost-analysis Tool

The functionality of the Cost-Analysis tool is that of a predictor of the real costs of the facilities where the developer's application is deployed. They will be able to use the methods described in the tool to replicate the cost study in different server options (AWS, AZURE, GC, etc.) and reach their own conclusions about the various alternatives and their potential use in the CI / CD stages.

2.3 Smart Assistant

The main goal of SmartCLIDE is to increase development speed, reduce errors, and improve the accuracy of the programming process. In general, most IDEs include several tools to cover most aspects of software development like analysing, designing, implementing, testing, and deploying. However, IDEs could have more automation and intelligence to help developers. These features can be obtained by using the Smart Assistant component, which uses artificial intelligence functionalities through the Deep Learning Engine (DLE) component for its purposes. In this section smart assistant, functionality will be discussed. The Smart Assistant and DLE have been specified as distinct components in previous versions of SmartCLIDE documents [1] [2]. However, the linkage of both components is high and can lead to remarkable overlap.

Many smart functionalities can be found in extension marketplaces [6], like Kite [7] extension for completion during source code writing. However, SmartCLIDE needs more guarantees to maintain and version smart extensions in its own models. This is why SmartCLIDE proposed that the responsibility for the Smart Assistant's intelligent behaviours should reside with the DLE. In this way, the Smart Assistant component is simplified as a simple gateway between the DLE's internal models and the clients that will use those behaviours through the Restful API.

The intelligent behaviours that the Smart Assistant provides to assist and guide the user while development in different phases of software development are:

- One line code generation
- Context monitoring specification to provide suggestions.
  - Code repository suggestions
  - Recommender system for the sizing of the deployment environment
- Get suggestions for workflow autocompletion as Items recommender in a BPMN workflow
- Get a set of suggestions for acceptance test based on Gherkin

2.3.1 Design Approach

The UML diagram in Figure 12 shows the five responsibilities of the Smart Assistant. The different subcomponents of the Smart Assistant are intimately related to the IA models of the DLE component, which is why some subcomponents are simple gateways to the IA models.

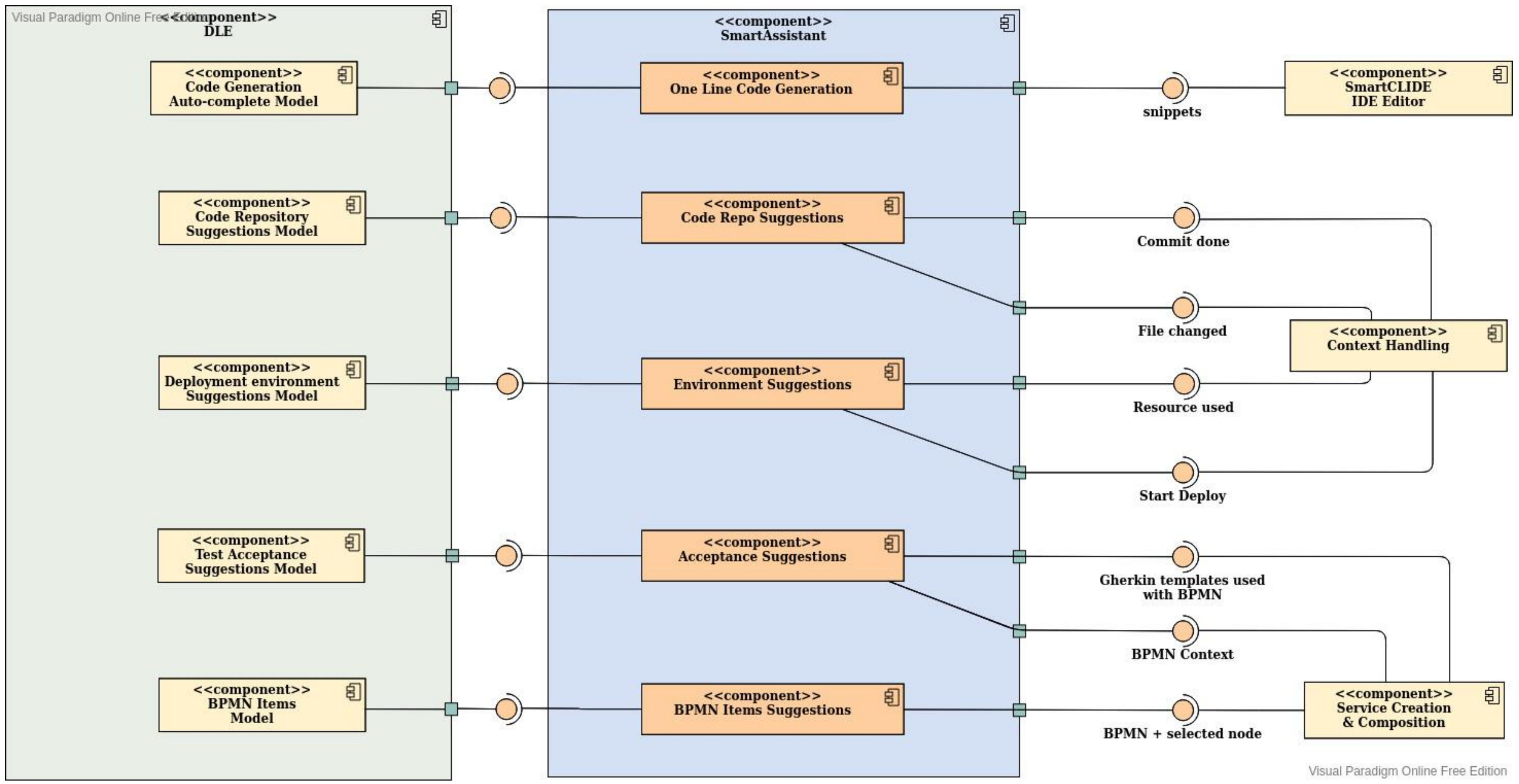


Figure 12: Smart Assistant Component Diagram as DLE interface

The key functionalities of the subcomponents of the Smart Assistant are presented in Table 3 -

Table 7.

Table 3:Smart Assistant One Line Code Generation

Name	One Line Code Generation
Functionality	One Line Code Generation API subcomponent is an API Restful interface to the Code Generation Auto-complete Model subcomponent of the DLE component and is defined in the DLE section.
Relevant Use Cases (D1.3)	UC-0006
Functional Requirements	D85, D88, D89, D90, D105

Table 4: Smart Assistant Code Repository

Name	Code Repository
Functionality	This wizard is responsible for generating suggestions to the user to facilitate commits to the git repository. Receiving information from the monitoring system and with the help of the DLE, it will determine the best time to commit to the git repository.
Relevant Use Cases (D1.3)	UC-0006, UC-0015
Functional Requirements	D85, D88, D89, D90, D105

Table 5: Smart Assistant Environment Component

Name	Environment
Functionality	Responsible for generating suggestions for the metrics of the deployment environment
Relevant Use Cases (D1.3)	UC-0006, UC-0009
Functional Requirements	D85, D89, D90, D105

Table 6: Smart Assistant Acceptance Test Suggestions Subcomponent

Name	Acceptance test Suggestions
Functionality	The acceptance test set suggestion system, based on collaborative filtering techniques, is responsible for providing the user with a set of tests defined in Gherkin format to be applied to the workflow defined in the BPMN and help verify if the expectations are met
Relevant Use Cases (D1.3)	UC-0006, UC-0008
Functional Requirements	D85, D87, D88, D90

Table 7: Smart Assistant BPMN Items Suggestions Subcomponent

Name	**BPMN Items suggestions**
Functionality	The BPMN Items suggestion system consists of automatically selecting the next node/item in the workflow being modelled during service composition in BPMN format.
Relevant Use Cases (D1.3)	UC-0006, UC-0007, UC-0011, UC-0012
Functional Requirements	D84, D85, D86, D88, D90

2.3.2 Interface Specification

The communication interfaces provided and required by Smart Assistant and shown in Figure 12are detailed in Table 8.

Table 8: Smart Assistant Interface Specification

No	**Interface (/API)**	**Description**	**Type**	
			Provided	**Required**
1	**One line Code Generation API**	The Code generation API is a subcomponent of the DLE module. This subcomponent is responsible for generating code based on the user search query. The API requires the user's queries to be passed it. • **code_input**: A sequence of chars or words in text format. • Method: this parameter can include the "Default" or "GPT2" value which specifies the DL algorithms in the backend for generating code. If developers don't pass the arguments into the function, this parameter will have the "Default" value. • Language: this parameter specifies the target programming language. • code_sugg_len: This parameter specifies the length of code that will be generated, which can accept a numeric value. • code_sugg_lines: This parameter specifies the lines of codes suggested by the code generator, which can accept a numeric value. The API returns related code snippets based on user usage history and quality parameters in the collected dataset. The first version of this API is designed for finding Java codes.	*	
2	**Code Repository API File changed**	The Code Repository API component offers an API endpoint in order to accept requests from the Context Handling component to make suggestions to the user	*	

No	Interface (/API)	Description	Type	
			Provided	Required
		when determining the best time to commit to the git repository. It needs as input monitoring information each time a change is saved to a file locally: - user. To be able to make custom suggestions - branch. In order to be able to make custom suggestions - pending files. The number of modified files pending upload to the code repository. - compilation errors. The number of compilation errors. It is a recommended data to have in order not to make suggestions of commits having compilation errors. This event will trigger a hint about the need to commit to the code repository if applicable based on the context data and the suggestion model.		
3	**Code Repository API Commit**	The Code Repository API commit component offers an API endpoint in order to accept requests from the Context Handling component to determine the best time to commit to the git repository. It needs as input monitoring information each time a commit is made to the code repository: - user. To be able to make custom suggestions - branch. In order to be able to make custom suggestions - number of files. The number of files uploaded to the code repository.	*	
4	**Environment API Resource used**	The Environment API component offers API endpoints to accept requests from the Context Handling component to make suggestions to the user about the sizing of the deployment environment. It needs as input tracking information each time a workflow execution is performed: - Unique identifier of the workflow that allows to relate it to the tracking sizing information provided through the Environment API Resource used. - maximum resources used, such: - Mb of ram - CPU Hz - Concurrent threads - Execution time duration - disk space	*	

No	Interface (/API)	Description	Type	
			Provided	Required
5	**Environment API Start Deploy**	The Environment API component offers API endpoints in order to accept requests from the Context Handling component to make suggestions to the user about sizing of the deployment environment This API is the entry point of the recommendation model. Context handling should use this API to get a recommendation when the user wants it (typically when all tests have passed successfully). It needs as input tracking information each time a workflow execution is performed: - Unique identifier of the workflow that allows to relate it to the previous sizing information provided through the Environment API Resource used. This event will trigger a deployment environment sizing hint.	*	
6	**Acceptance API Gherkin templates + BPMN**	The Acceptance API component provides API endpoints for the purpose of accepting requests for providing the user with a set of tests defined in Gherkin format to be applied to the workflow defined in the BPMN and help verify if the expectations are met. This is the entry point for improving the recommender system. Here are the gherkin templates that have been used as an acceptance test for a BPMN workflow. - Finalised BPMN - Set of gherkins used as acceptance test of the BPMN	*	
7	**Acceptance API BPMN Context**	The Acceptance API component provides API endpoints for the purpose of accepting requests for providing the user with a set of tests defined in Gherkin format to be applied to the workflow defined in the BPMN and help verify if the expectations are met. In this API, you receive the BPMN finalized when the user wants to get recommended acceptance tests. The input data is: - Finalised BPMN The response is: - Set of gherkins is recommended as acceptance tests for the BPMN.	*	

No	Interface (/API)	Description	Type	
			Provided	Required
8	**BPMN Items API**	The BPMN Items API suggestion system consists of automatically selecting the next node/item in the workflow being modelled during service composition in BPMN format. When the user wants a recommendation during the editing of the BPMN, user should indicate: - BPMN in XML format in edit - selected node of the BPMN And the output of the recommender system will be: - New item/task to be included in the XML of the BPMN	*	
9	**One Line code Generation to DLE Model**	The One Line code generation API use DLE subcomponent Code Generation Auto-Complete Model as recommendation engine		*
10	**Code repo to DLE Model**	The Environment API subcomponent use DLE Deployment environment Suggestions Model subcomponent as recommendation engine		*
11	**Environment API to DLE Model**	The Environment API subcomponent use DLE Deployment environment Suggestions Model subcomponent as recommendation engine		*
12	**Acceptance API to DLE Model**	The Acceptance API subcomponent uses the DLE Test Acceptance suggestions model subcomponent as a recommendation engine.		*
13	**BPMN items API to DLE Model**	The BPMN Items API subcomponent uses the DLE BPMN Items Model subcomponent in order to get recommendations results.		*

2.3.2.1 Smart assistant functionalities

Smart assistant needs provide functionality which facilitates the development process. Most of this functionality is the result of merging artificial intelligence with existing IDE functionality. this section reviews the available smart assistant functionality:

One line code generation

In IDEs, code completion can suggest lists of API/function/Class/variable. Several studies are focused on code-completion tools that can improve the list of API suggestions [8] [9] have presented a learning model for API parameter suggestions in IDEs. Besides, [10] proposed the statistical learning models that learn from code changes. They believe the code changes are accrued for one method; their dataset is based on code changes snippets. Also, in some practical works [7] Language modelling has been utilized for autocompletion tasks that have successful results. Accordingly, the DLE subcomponent has provided automatic code generation based on the generative models. Smart assistant subcomponent is responsible for provide interface of DLE autocomplete model. This interface needs to use context awareness of what developers are writing and provide proper inputs for the DLE model.

Context monitoring

The context monitor is designed to solve two responsibilities of the Smart Assistant.

- Guidance through the process of pushing changes into a version control repository.
- Suggestions or guidance in the deployment stage on the sizing of the deployment environment

We have defined two operating scenarios that will allow the Smart Assistant, with the help of the DLE, to make suggestions that will facilitate the SmartCLIDE user's work.

For the commit-based recommender system, a model based on a combination of statistical methods and computational techniques has been employed. The fundamental idea of the method is that the conditional probability density of random variables such as the number of changes between commits is approximated using numerical methods. There is a number of changes made for a programmer with the characteristics of such a random variable. Based on this information, the system calculates how common or infrequent that situation is with respect to the approximate distribution, thus producing a suggestion.

The resource estimation system has been designed within the formal framework of fuzzy inference systems (FIS). In this case, the aim is to capture the common relationships, understandable to human experts in this field, in rules of a mathematical nature that consider the uncertainty intrinsic to the natural language. The mathematical realization of these rules is done with functional forms commonly used in the domain of fuzzy logic, thus allowing the operation of the system to be understandable to human experts, as well as allowing fine tuning in the case that a dataset with metrics on the resource allocation problem becomes available in the future.

Acceptance test based on Gherkin

The model for acceptance test recommendation is based on the construction of a history that relates the BPNM diagrams available in the database to the existing acceptance test cases to which they have been linked. the subcomponent of smart assistant needs to link user and AI based model in DLE.

This subcomponent is responsible for receiving two types of requests:

- *Model feedback*. BPMN file in XML format and a set of acceptance test templates in Gherkins format that the user has decided to use for testing. This data will help the model explained in the DLE section to progress by learning in order to be able to make better recommendations.
- *Context BPMN file*. This is the acceptance test set request. The BPMN file is used by the CBR model (detailed in the DLE) as input. The response is the acceptance test suite in gherkin format.

In both types of requests, the subcomponent in the Smart Assistant is in charge of communicating the user context with the IA model of the DLE subcomponent. It performs the validation and checking of the input on the different fields, both in terms of format and required fields. And managing the result of the model to make the most efficient recommendation to the user.

Items recommender in a BPMN workflow

Service composition needs efficiently exploring for services from service repositories. SmartCLIDE aims to provide suggestions for the next node/item in the BPMN workflow. Accordingly, The AI-based model has been proposed to make service suggestions while service composition. This smart assistant subcomponent tries to provide a user interface for interacting with the DLE model.

The goal of the proposed model is to suggest a list of candidate Web services while developers are composting services in BPMN workflows. The adopted solution is embedded in a DLE subcomponent which provides the suggestions using API. The input of the model is a suggestion request which can be a one-off signal from the SmartCLIDE monitoring system. SmartCLIDE has several monitoring modules which are continuously working in the background. For example, while developers are building services workflow, when a step in the workflow diagram drawing is completed, SmartCLIDE creates a signal to the backend to receive service suggestions for the next step. The signal must contain the identification of the last node so that suggestions are made based on it. The output signal, in JSON format, will contain the recommended service to link to the node indicated in the request. The resulting list may have no recommendations for all requests which depend on the DLE decision.

The actual BPMN context has a significant role in providing smart suggestions which provide required information for proposed models. The service suggestions module in the BPMN has been exposed as a web

API and will be implemented in Python. Both request and response are in Json format. The module has four main steps which are: 1) Query Compositor 2) Current BPMN Extractor 3) BPMN semantic identifier 4) Numerical vector transformer.

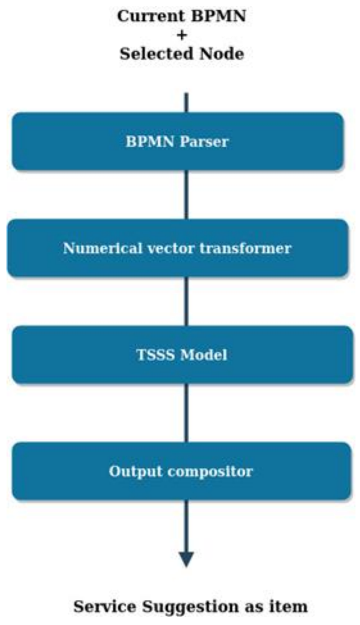


Figure 13: BPMN Suggestions Architecture

The different stages of the recommendation model are detailed below:

- *BPMN Parser*. This stage is responsible for extracting information from the incomplete XML file that is received as input. The current study introduces semantic BPMN identifiers based on network techniques to transform the incomplete BPMN into a text sequence.
- *Numerical vector Transformer*. This stage has been designed based on NLP techniques which use word embedding text representation. The main idea is to transform the incomplete BPMN into a numerical vector. The well-known Python libraries for implementing this submodule have been utilized including Spacy and Gensim.
- *Transformed space similarity search (TSSS).* Based on similarity metrics defined in the transformed vector space created in the previous stage, tools are designed to locate the most suitable services for the BPMN design context in which the user is located. The last nodes introduced determine a textual pattern to which corresponds a representation in the transformed space that can be used for recommendation. This stage is at the heart of the success and failure of the recommendations. The aim is to suggest services from the internal SmartCLIDE registry during the composition of the workflow in BPMN, so as to avoid having to do a search to complete each node individually.
- *Output compositor.* The output compositor is responsible for defining, validating service suggestions. Based on different request, service suggestion can offer 3 types of JSON response:
    - To improve the results obtained from the model by applying a system of penalties on the recommendation rating to services already present in the BPMN diagram to avoid making recommendations of nodes already present.
    - Decide whether the results obtained from the model allow a recommendation to be made. The transformed vector similarity model results in the ordered list of services most similar to the context.
    - Compose the recommendation in JSON format

3 Deep Learning Engine

3.1 DLE Problem Specification

The rapid rate of technological and digital advancement requires the building of related software, which is a time-consuming process. SmartCLIDE includes the advantages of Artificial Intelligence (AI) and Cloud Computing. These technologies can help the developer overcome the complexities associated with multi-platform software products. Concerning Intelligent software engineering, theoretical [11] [12] and empirical [13] [14] works have shown that software intelligence has widely used in software development. Accordingly, SmartCLIDE has proposed the DLE component, which is responsible for feeding smart Assistants by intelligent models. DLE subcomponent responsibility can fall into the following categories:

- Context monitoring specification in order to provide suggestions
- AI code completion for generating one-line code using language modelling
- The acceptance test set suggestion for giving the user a set of tests defined in Gherkin format.
- Classification of Web services based on their meta-data in order to reduce service selection search space
- Code repository suggestion is responsible for making suggestions for the user to facilitate commits to the git repository.
- Service deployment environment recommendations in order to produce suggestions for the sizing of the deployment environment.
- BPMN Items suggestions aim to help automation in selecting the next node/item in the BPMN workflow

3.1.1 Design Approach

DLE subcomponents are responsible for supporting AI-based smart assistant features. Figure 14 demonstrates the DLE component diagram, including components, subcomponents, interfaces, ports, and their relationships. The DLE component of SmartCLIDE aims to provide some intelligent features directly in the service classification, code generation and predictive modelling wizard subcomponents. But in addition, it also contains subcomponents responsible for AI models to service the Smart Assistant responsibilities.

The key functionalities of the subcomponents of the DLE are presented in Table 9 - Table 16.

Table 9: Service Classification Model

Name	**Service Classification Model**
Functionality	The Service classification subcomponent is responsible for classifying new services. There are two important resources for new services. First, the newly created services are created by SmartCLIDE users using the service creation module. Second, the new observation by service discovery module.
Relevant Use Cases (D1.3)	UC-0006, UC-0012, UC-0028
Functional Requirements	D85, D88, D89, D105

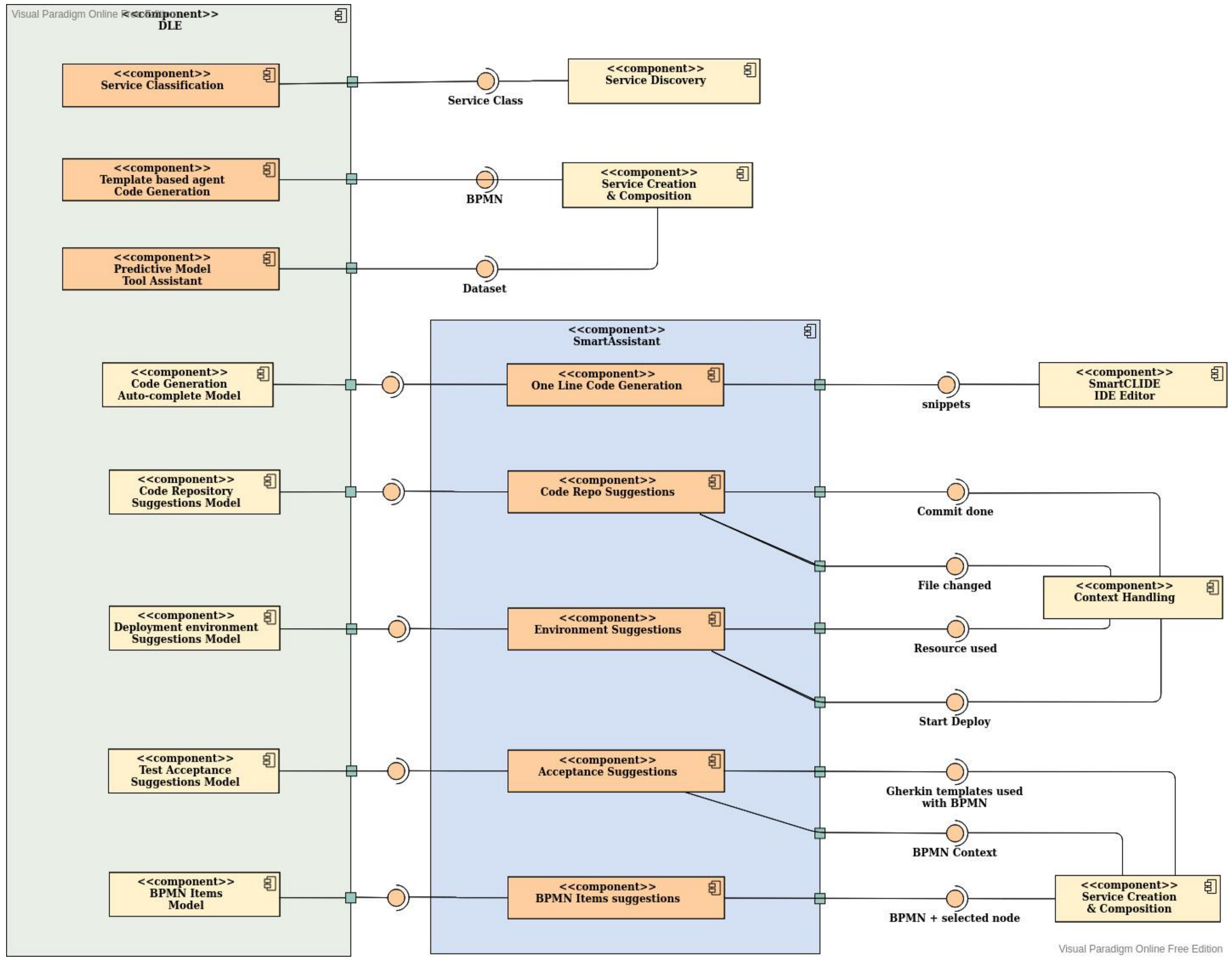

Visual Paradigm Online Free Edition
<<component>> DLE
<<component>> Service Classification
<<component>> Template based agent Code Generation
<<component>> Predictive Model Tool Assistant
<<component>> Code Generation Auto-complete Model
<<component>> Code Repository Suggestions Model
<<component>> Deployment environment Suggestions Model
<<component>> Test Acceptance Suggestions Model
<<component>> BPMN Items Model
Service Class
<<component>> Service Discovery
BPMN
<<component>> Service Creation & Composition
Dataset
<<component>> SmartAssistant
<<component>> One Line Code Generation
<<component>> Code Repo Suggestions
<<component>> Environment Suggestions
<<component>> Acceptance Suggestions
<<component>> BPMN Items suggestions
snippets
<<component>> SmartCLIDE IDE Editor
Commit done
File changed
<<component>> Context Handling
Resource used
Start Deploy
Gherkin templates used with BPMN
BPMN Context
BPMN + selected node
<<component>> Service Creation & Composition
Visual Paradigm Online Free Edition

Figure 14 DLE Component Diagram

Table 10: Template based agent Code Generation

Name	Template based agent Code Generation
Functionality	This subcomponent is responsible for generating code based on internal templates. The API returns related code snippets based on templates to implement the workflow represented in BPMN in low code. The first version of this API is designed for finding Java codes.
Relevant Use Cases (D1.3)	UC-0006
Functional Requirements	D103

Table 11: Code Generation Auto-complete Model

Name	Code Generation Auto-complete Model
Functionality	This subcomponent is responsible for one line automatic code generation based on DL learning model which is trained by external and internal available source codes
Relevant Use Cases (D1.3)	UC-0006
Functional Requirements	D85, D88, D89, D105

Table 12: Code Repository Suggestions Model

Name	Code Repository Suggestions Model
Functionality	This wizard is in responsible for generating suggestions to the user to facilitate commits to the git repository. Receiving information from the monitoring system, and with the help of the DLE, it will determine the best time to commit to the git repository.
Relevant Use Cases (D1.3)	UC-0006, UC-0015
Functional Requirements	D85, D88, D89, D105

Table 13: Deployment environment suggestions Model

Name	Deployment environment suggestions Model
Functionality	This subcomponent is responsible for generate suggestions for the sizing of the deployment environment
Relevant Use Cases (D1.3)	UC-0006, UC-0009
Functional Requirements	D85, D89, D90, D105

Table 14: Acceptance test Suggestions Model

Name	Acceptance test Suggestions Model
Functionality	The acceptance test set suggestion system, based on collaborative filtering techniques, is responsible for providing the user with a set of tests defined in Gherkin format to be applied to the workflow defined in the BPMN and help verify if the expectations are met
Relevant Use Cases (D1.3)	UC-0006, UC-0009
Functional Requirements	D85, D87, D88, D90

Table 15: BPMN Items suggestions

Name	BPMN Items suggestions
Functionality	The BPMN Items suggestion system consists of automatically selecting the next node/item in the workflow being modelled during service composition in BPMN format.
Relevant Use Cases (D1.3)	UC-0006, UC-0009
Functional Requirements	D85, D87, D88, D90

Table 16: Predictive Model tool API

Name	Predictive Model tool API
Functionality	This wizard, as a subcomponent of the DLE in SmartCLIDE, is accessible through a RESTful API in several stages, structured in an ideally linear flow that in practice allows iterative backtracking. Its objective is to guide the user in the creation of a predictive model.
Relevant Use Cases (D1.3)	UC-0006 (P65 associated requirements)
Functional Requirements	D70, D100, D101, D102

3.1.2 Interface Specification

No	Interface (/API)	Description	Type	
			Provided	Required
1	Service Classification API	The Service classification API is a subcomponent of the DLE module. This subcomponent is responsible for classifying new services. There are two important resources for new services. First, the newly created services are created by SmartCLIDE users using the service creation module. Second, the new observation by service discovery module. Both services need to have required fields	*	

No	Interface (/API)	Description	Type	
			Provided	Required
		that are defined in the service specification schema. The following parameter is mandatory to route the requests to this implemented API: • **Service Name**: A string including web service name. • **Service Description**: a service description in string format. • **service_id**: Service Id in numeric format. This API determines one pre-defined category in an available dataset and returns a JSON file containing the service class. The API has the ability of batch services classification.		
2	Template Code Generation API	The Code generation API is a interface to the subcomponent Template Code Generation of the DLE module. This subcomponent is responsible for generating code based on internal templates. The API requires the BPMN in XML format. **Current BPMN workflow**: A xml format. The API returns related code snippets based on templates to implement in low code the workflow represented in BPMN. The first version of this API is designed for finding Java codes.	*	
3	Predictive Model tool API	This wizard, as a subcomponent of the DLE in SmartCLIDE, is accessible through a RESTful API in several stages, structured in an ideally linear flow that in practice allows iterative backtracking. Its objective is to guide the user in the creation of a predictive model. The stages of the wizard are: 1) Functionality related to dataset analysis. It receives as input the dataset in csv format, analyses it and returns in JSON format the names of the columns, suggested types and an example of target so that the user can modify this configuration if necessary, in later stages. 2) Optionally, simple graphical representations of the attributes in the dataset can be obtained, considering the type (quantitative, qualitative, ordinal...) assigned to them. This can be used for the	*	

No	Interface (/API)	Description	Type	
			Provided	Required
		user to decide which attributes to consider or ignore in the construction of the model. 3) Having as input the JSON-like information with the chosen attributes, their assigned data types, the target column (in the case of supervised algorithms), the algorithm and its configuration selected for training and the configuration of the policy to split between training and testing, the model is adjusted and trained. 4) The metrics and data of the resulting model can be checked with dedicated endpoints. The status of the training process can be monitored 5) The final model is exported, including example code to allow it to be deployed as a stand-alone service.		
4	One Line code Generation to Smart Assistant	Provide a generated one-line code by DL model for Smart Assistant	*	
5	Code repo to DLE Model	Provide a generated one-line code by DL model for Smart Assistant	*	
6	Environment API to DLE Model	Provide service deployment environment recommendations for Smart Assistant	*	
7	Acceptance API to DLE Model	Provide acceptance test recommendation results for Smart Assistant	*	
8	BPMN items API to DLE Model	Provides suggestions for new items during workflow editing in BPMN format.	*	

3.1.3 Service Classification

Web services include a wide range of software, applications, or cloud technologies that use standardized protocols to communicate and exchange data messaging. Therefore, service has different meanings based on its applications. Since some service classification approaches tend to be more efficient in some service applications, it is necessary to first define the terms before reviewing the technical architecture.

Online services are mostly a marketing expression that can fall into three main categories: 1) Infrastructure as a Service (IaaS), 2) Platform as a service (PaaS), 3) SaaS Software as a Service (SaaS). IaaS offers services such as virtual hosts, networking, and virtualization like Amazon Web Services (AWS) for systems administrators. PaaS can offer tools or some Web services for developers, which are available on a cloud structure. SaaS platforms make software available to even non-developer users on the Internet. Given these categories, our work focuses on classifying web services or APIs (PAAS).

The service classification subcomponent of the DLE attempts to categorize discovered service data into a predefined number of classes. First, the component discovers services from different resources and repositories. Most of the web service repositories have a category field to identify their service functionality. To this end, the component has used the category field as a dataset label. However, the main problem is that those categories have too much variety, which provides class diversity and decreases classification performance. For this reason, the service classification component has used a hybrid AI model by combining unsupervised and supervised learning techniques. In particular, clustering approaches are employed to cluster service categories. Afterward, ML and DL learning models have been examined for service classification on clustered service data. A Long Short Term Network (LSTM) was trained, with the text processed using Fasttext [15], which is an extension of the word2vec [16] model for text representation. Also, machine learning (ML) algorithms such as MultinomialNB classifier, GradientBoosting, and Support Vector Classifier (SVC) use the TF-IDF vectorizer to process the text. This work evaluates the mentioned algorithms based on accuracy, precision, recall, and f1 score metrics.

3.1.4 Code Generation

Automatic code generation is a solution that accelerates software development. The code generation sub-component tries to provide code autocompletion which can offer a line of codes during programming. Autocomplete IDE functionality through AI can be divided into two groups: 1) API/function/Class/variable suggestions were improved by recommender systems 2) Autocomplete approaches applied language modelling [17] [18]. The first category can improve the accuracy of suggestions and provide an efficient list of API/function/variable suggestions. The second group is for the code completion approach in which AI can be used, such as NLP techniques for language modelling. These suggestions can consider local codes and external features in the current IDE tab.

The Code generation subcomponent needs to develop a promising approach. In practice, there are successful efforts such as Tabninea [19] [20], Kite [21], which provide one-line code generation using language models. Consequently, this subcomponent, which needs to develop a promising approach, has proposed automatic code generation based on LSTM and pretrained GPT2 models for generating one line code.

3.1.5 Datasets

Service classification and code generation needs a dataset for providing learning models. This section will describe the dataset resources and processing data steps.

3.1.5.1 Dataset for Code Autocompletion

In traditional template-based code generation, the focus has mostly been on user input and available designed code template. However, automatic code generator generation approaches that used AI models need users' input and require training datasets. These datasets can have different representations of code (e.g., code snippets, images, abstract syntax trees, etc.).

Pre-processing dataset depends on the level of code like function, class levels that the automatic code generator will generate. SmartCLIDE code generation aims to assist developers while writing codes. Therefore, the provided dataset includes lines of codes in text format. Although the local project codes or codes in the running IDE tab can provide valuable code corpus for training a deep learning algorithm, external public datasets can be utilized. GitHub Java corpus [22] is a collection of Java code on a large scale divided into training and test datasets. The training dataset includes 10,968 java projects and 264,225,189 lines of codes. Also, the test dataset consists of 3,817 java projects and 88,087,507 lines of codes. However, the model is under development, and selecting the right dataset is investigating, as well.

SmartCLIDE autocompletion aims to assist developers while programming with one-line code completion. To this end, datasets include lines of codes in text format. Although, the local project codes or codes in the running IDE tab can provide valuable code corpus, in order to train deep learning algorithm, external public datasets have utilized, as well.

3.1.5.2 Dataset for Web Service

Several public service registries [23] [24] have collected web services. Current work uses a dataset that is collected from Programmable web [23]. This public service dataset includes thousands of services from popular providers. ProgrammableWeb was established in 2005, and it has collected 24100 web APIs by September 2021. The collected data includes detailed information such as description, category, versions, and response formats. The category field of this public dataset is assumed as a service data label. However, service categories have more than 500 different values. In general, class diversity is a challenging issue in building successful classifiers. Therefore, DLE provides a pipeline for clustering datasets before classification. Moreover, a significant part of web service API is Web/Internet, which is network APIs that are uniquely addressable across the Internet.

3.1.5.3 Dataset for Acceptance Test

The model for acceptance test recommendation is based on the construction of a history that relates the BPNM diagrams available in the database to the existing acceptance test cases to which they have been linked. In order to design the AI model to achieve the desired recommendation system, two datasets have been obtained:

- A total of 3479 .feature files in Gherkin format.
- A total of 2268 .bpmn files with the XML

The fact that the collected Gherkins datasets contain a large number of examples of a purely didactic nature and in several languages means that one cannot expect any kind of modeling around them to result in a tool that is applicable to real projects. To solve this problem, a Case-Based Reasoning (CBR) approach is proposed that allows the incremental construction of a set of Gherkins associated with realistic projects, with the prior consent of the users, who, by granting it, will be able to benefit from the recommendation system. As a basis for the algorithm, it is proposed that, based on a correspondence of BPNMs and Gherkins associated with them, for a new BPMN in the context of the IDE, the most similar BPMNs in the knowledge base are searched (using notions of textual similarity as a criterion for this search) and then retrieving the Gherkins associated with them, offering them to the user for possible incorporation.

In order to allow the system to be put into operation, the establishment of a pilot case is considered using the dataset collected which, although due to the nature introduced above, it is not possible to generate a model that can be extrapolated to real scenarios, it will allow the behaviour of the recommender system to be validated qualitatively.

3.1.6 Data Processing

In AI-based models, the results are highly dependent on the data quality. Therefore, data processing can impact the adequate performance of any algorithm. The majority of the collected data in SmartCLIDE, especially in service classification and code generation, is text. Given that the majority of the data is text, text pre-processing is required prior to clustering and classification. In fact, pre-processing text transforms text from human language to machine-readable format. Text pre-processing includes some well-known NLP techniques: tokenization, normalization, stop words removal, and lemmatization/stemming. Tokenization splits a sequence of text into units with semantic meaning. Normalization includes introductory text pre-processing, including removing punctuations, transforming to lower case, and removing numbers from the document. Stop words removal is responsible for removing generic words of the English language such as determiners, conjunctions, and other parts. Lemmatization or stemming is a process that reduces each word to its root (e.g., using Porter's stemming algorithm [25]).

SmartCLIDE text processing tries to cover levels of linguistic processing which includes syntax, semantics and pragmatics analysist. Moreover, data pre-processing class diagrams aims to support both NL and structured codes pre-processing. Figure 16 demonstrates data pre-processing class diagrams that support both NL and structured codes pre-processing.

3.1.6.1 Data Preprocessing for source code dataset

Regarding source code preprocessing, NLP suffers from ignoring the concept of code structure. However, recently researchers provide learnable probabilistic models of source code that consider code structure [26]. Hence, there is a strong need for a tool that helps develop quality software by automatically pre-processing source codes. SmartCLIDE preprocessing uses the Codeprep [27] tool. Table 17 demonstrates some functions which Codeprep supports.

Table 17 Example of functions for preprocessing source code corpora

Function Name	Desc
RemoveComments	This function replaces comments with specified character
CamelCaseSpeliting	Function names such as "sendEmail" will split based on Java function naming
LineLevelSpliting	This function splits lines of codes in a file
FunctionLevelSplitting	One of important code mining task is, the Tokenization of code is an important stage, the common method for tokenizing source code during data mining can be tokenizing based functions. SmartCLIDE preprocessing use Tree-sitter for function level Tokenization

However, most of the mining approaches are based on well-commented, well-naming functions, variables, and classes. Thus, these approaches may provide weak results in some real-world source codes.

3.1.6.2 Web Service Data Processing

The Service classification sub-component utilizes real-world web service API. Collected datasets suffer from a famous classification problem, which is imbalanced classes. Therefore, we have conducted possible solutions for learning from imbalanced data, including data-level and algorithm-level methods. Moreover, the collected dataset has high-class diversity, which impacts classification accuracy. To address this issue, our approach applied some filtering and HAC clustering to the service categories.

In order to cluster service data, two popular clustering algorithms have been investigated: 1) K-Means clustering, 2) Agglomerative hierarchical clustering. K-Means clustering attempts to partition the dataset into K predefined groups, which do not have overlap. However, agglomerative hierarchical clustering does not need a predefined number of groups. Instead, it uses a recursive concept, which can be applied bottom-up or top-down. Both the K-Means and hierarchical agglomerative clustering have been implemented. Finally, hierarchical agglomerative clustering has shown better results; for this reason, it has been selected. Afterward, two methods for dealing with imbalanced data have been utilized. First, the synthetic minority oversampling technique **(**SMOTE) has been applied to the clustered dataset. SMOTE is an oversampling technique selecting examples that are close in the feature space. The results demonstrate that using the SMOTE technique provides some improvements; however, synonyms-based augmentation offers a significant improvement. shows Each of the stages for data processing.

Table 18: Process Stage Details for Providing Training Dataset for Web Service Classification

Records	Rows	Reprocess Steps
Row Dataset	~24000	-
Clustered Dataset	~21000	Select categories with more than 10-rows data cluster data by hierarchical clustering
Augmented Dataset	~30000	Augment categories less than 400 services

3.2 Machine Learning and Deep Learning Algorithms

Nowadays, Artificial intelligence (AI) is a well-known research area that builds intelligent programs and machines that can improve automation tasks. This section will discuss service classification and code generation, which mainly includes AI-based text processing techniques.

3.2.1 Service classification

Over the past decades, several approaches have been adopted to determine the category of a web service from several pre-defined categories [28] [29] [30] [31] [32] [33] [34] [35] [21]. The early approaches used manual defining services and keyword-based service discovery. Afterwards, semantic-based approaches [28] [29] were introduced to overcome keyword search limitations. In recent years, studies have mostly focused on using AI-based techniques [30] [31] [32]. SmartCLIDE service classification needs to identify challenges and solutions in both practical and theoretical aspects.

Two major evolutions have had a significant impact on the service classification research area. First, the evolution of service implementation, which is transformed from traditional SOAP services to RESTful services. Second, the evolution of in-text processing by switching from traditional bag of words (BOW) feature engineering to utilizing word-embedding text representation and deep learning.

Although most earlier approaches have used information extraction for extracting service features from WSDL [33] [34] [35] [21], these days, RESTful is the prevalent solution for providing web services/APIs. In fact, RESTful service descriptions are merely in the free text, which is less machine-readable than structured WSDL. Consequently, some automatic processes, such as automatic data collection, service specification and discovery are facing some challenges.

Considering RESTful services evolution, service description has become a significant feature in service classification. Accordingly, most of the text classification approaches have been applied in this research area. The emergence of word-embedding techniques has improved keyword-based feature engineering. Additionally, the increasing word embedding open-source projects such as Glove [36] or word2vec [15](fixed embedding) helps the fast and efficient low-dimensional representation of web service data. These features has led to better results in service classification [28] [30] [32]. Therefore, a series of recent studies in text processing has indicated that the combination of feature engineering and learning model was switched from traditional BOW/ML algorithms to word-embedding/deep learning. Consequently, the service classification approaches have impacted the mentioned text processing evolution. The earlier approaches mostly combined BOW and traditional ML. Up to now, LSTM and BiLSTM models have shown proper accuracy in service classification [32] [28]. BiLSTM-based modeling offers better predictions than regular LSTM-based models. However, BiLSTM models suffer from high computational processes, which can be challenging in practice.

3.2.2 Code Generation

Automatic code generation has important applications in software development tools, the most famous of which are integrated development environments (IDEs). These tools have features such as template suggestion and automatic code completion, which can use automatic code generation techniques. Moreover, merging artificial intelligence (AI) can bring new opportunities in most of the mentioned research areas. AI offers the ability to predict [37] [38] [39] [40], recognize [41] [42], and generate [43] [44] on the basis of existing data; this ability can be used for automatic code generation.

The literature has shown that automatic code generation methods can be categorized into 1) software engineering [45], 2) combination of software engineering and AI [37] [46], and 3) sole AI approaches [21] [41]. The selection of automatic code generation methods depends on factors like output scale (e.g., full software code, function, one line of code) or automation level. The first category includes promising techniques for both partial or full software source code generation, which can help non-developer users. The major topics in the first category include model-driven, template-based code generation, and visual programming. The second category has mostly improved code generation applications (e.g., code completion) in tools (e.g., IDEs) that have recently been proven to behave more intelligently. Additionally, recommender systems are extensively used in this category. In the third category, most methods are driven by text mining or image textual representation techniques. GUI2code AI approaches show promising results for fully automated code generation from an image [31]. However, fully automated multi-lines code generation through generative AI models still needs human supervision [47]. Recently, attention-based neural networks have been investigated to solve this issue. Besides, some works used code search instead of writing the program directly. Fully automated code generation through generative AI models is still an unpromising approach since it cannot keep track of structural code dependencies [48]. Balog, Matej, et al. in [43] have introduced the "DeepCoder" model, which can write code after searching a large code database. It resorts to inductive programming and search techniques to generate codes in its domain-specific language (DSL). However, their approach is limited to writing programs in just five lines of code. Regarding this limitation, some works used AI models to search existing code instead of directly writing limited code by deep learning. The hierarchy of automatic code generation categories is illustrated in Figure 36.

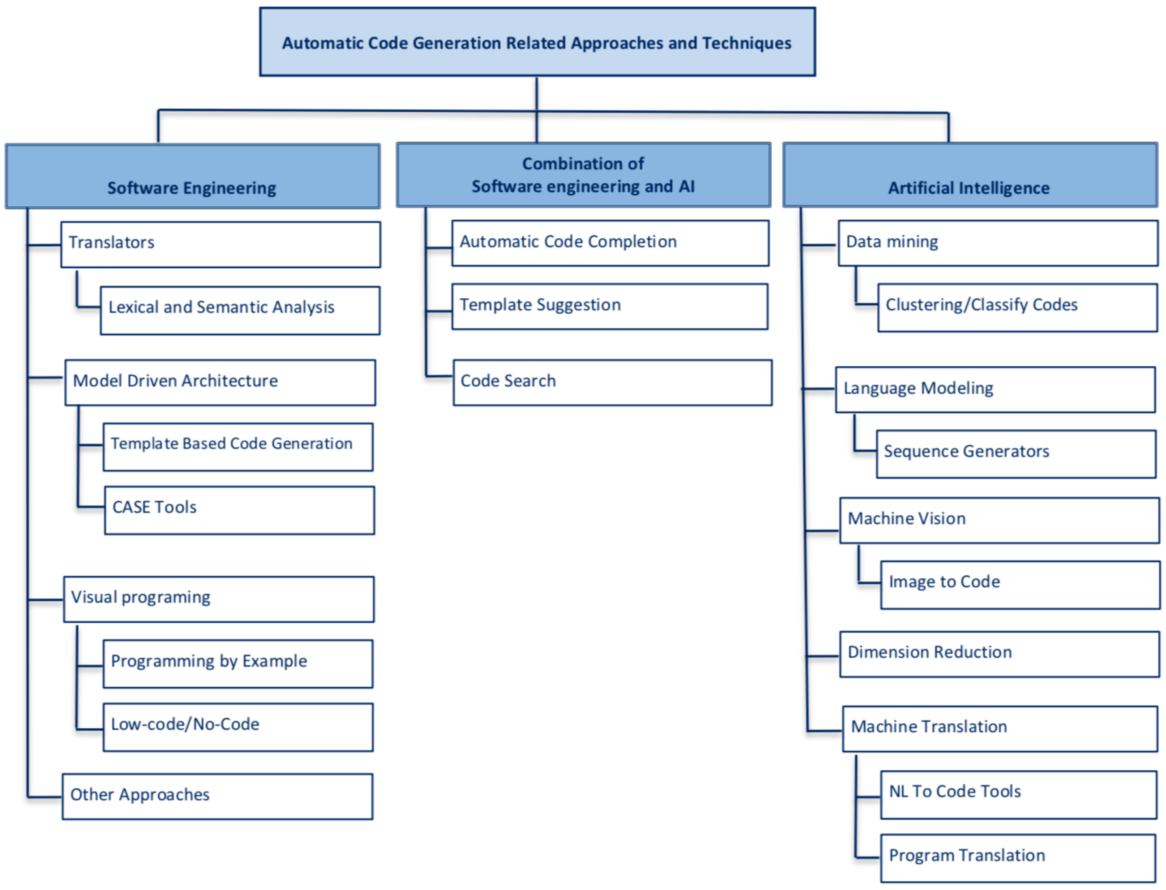


Figure 15 Automatic Code Generation Related Approaches Classification in SE and AI

Literature have shown merging artificial intelligence with existing IDE functionality can bring new opportunities in most involved area in software development tool (see Table 19). In general, the most promising approaches have improved current functionalities such as code autocomplete, code search in IDEs. These functionalities have been improved extensively by AI algorithms, especially through recommendation system models. To this end, SmartCLIDE aim to provide semantic code auto-completion powered by AI which suggest more relative code snippets for users while programming.

Table 19: Literature Review

Category	Research area	Application	Ref
Combination of	Recommendation model+ Code search	Code Suggestion NL-to-code search Code-to-code search	[35] [47] [44] [49]

Category	Research area	Application	Ref
Software Engineering and AI Approaches	Recommendation model + Code completion	API parameter suggestion in IDEs API method recommendation	[50] [51]
	Recommendation model + Template based code generation	Template Suggestion	[52]
AI Approaches	Language Modeling • Sequence Generative Model(e.g.RNN)	Code Completion Automatically write code API Method Recommendation	[53] [41]
	Machine Vision Model • CNN Neural network	GUI2Code Visual programming Automatically Handwrite to code	[38] [42]
	Dimension Reduction • NN embedding feature	Contextual Embedding of Source Code Source Code Representation	[54] [54]
	Machine Translation: • language modelling (e.g.RNN)	Natural language to code Generate code summarize Translate one language to another Abstract Syntax Tree	[55] [42]

3.3 Implementation

SmartCLIDE aims to provide some intelligent features for developers using AI-Based techniques. Figure 16 shows the class diagrams of service classification and code generation pipeline.

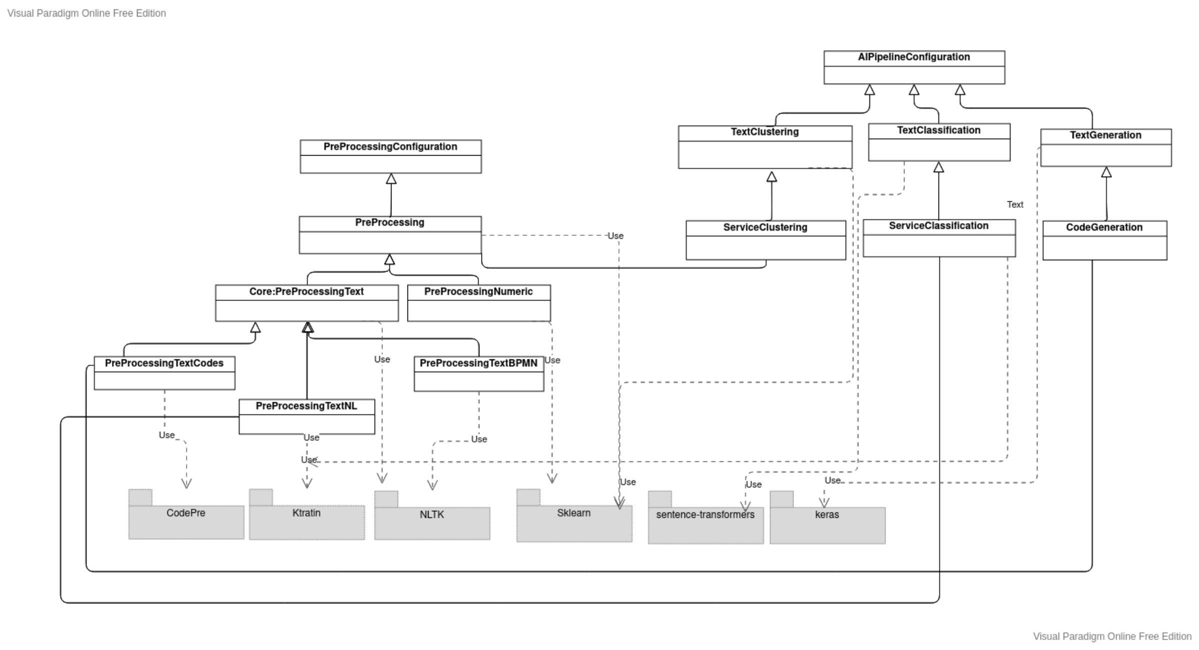


Figure 16 The scheme of Classes and used packages in DLE component

3.3.1 Service Classification

A hybrid unsupervised and supervised model was proposed for service classification. This model has trained based on real-world web services. The recent studies [30] [31] have shown that Programableweb [23] is the most popular resource for training the service classification models. This public service registry includes thousands of services from popular providers like Google APIs [23]. To this end, the model has been trained based on this public dataset, facilitating a comparison of the results.

The proposed model includes the following steps: 1) Providing dataset,2) Pre-processing data by NLP techniques, 3)Text representation by Distilbert transformer 4) Clustering dataset, 5) Dealing with imbalanced using data augmentation, 6) Text representation by BOW/Fast text 7) Classification using ML/DL algorithms 8)Save trained models.

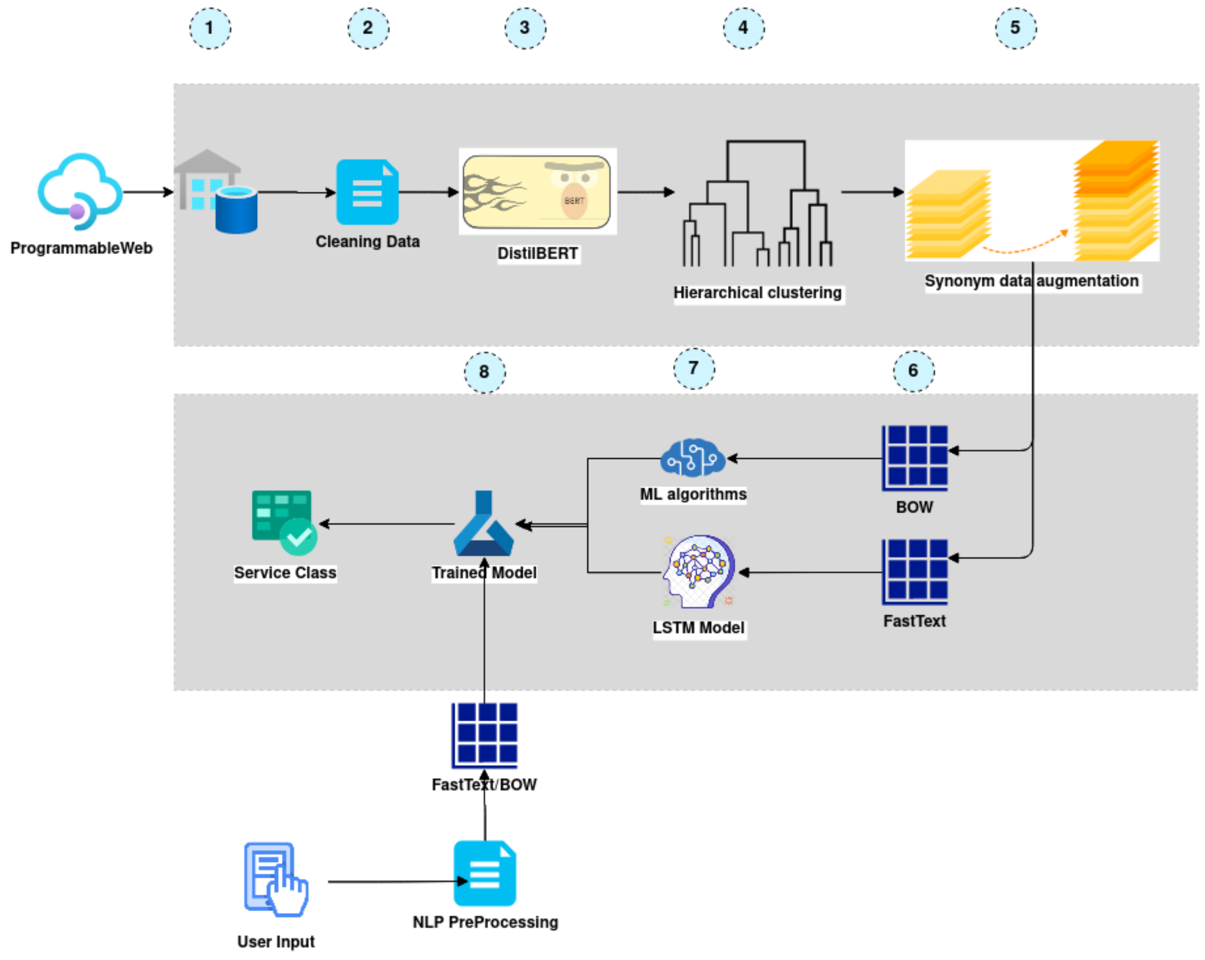
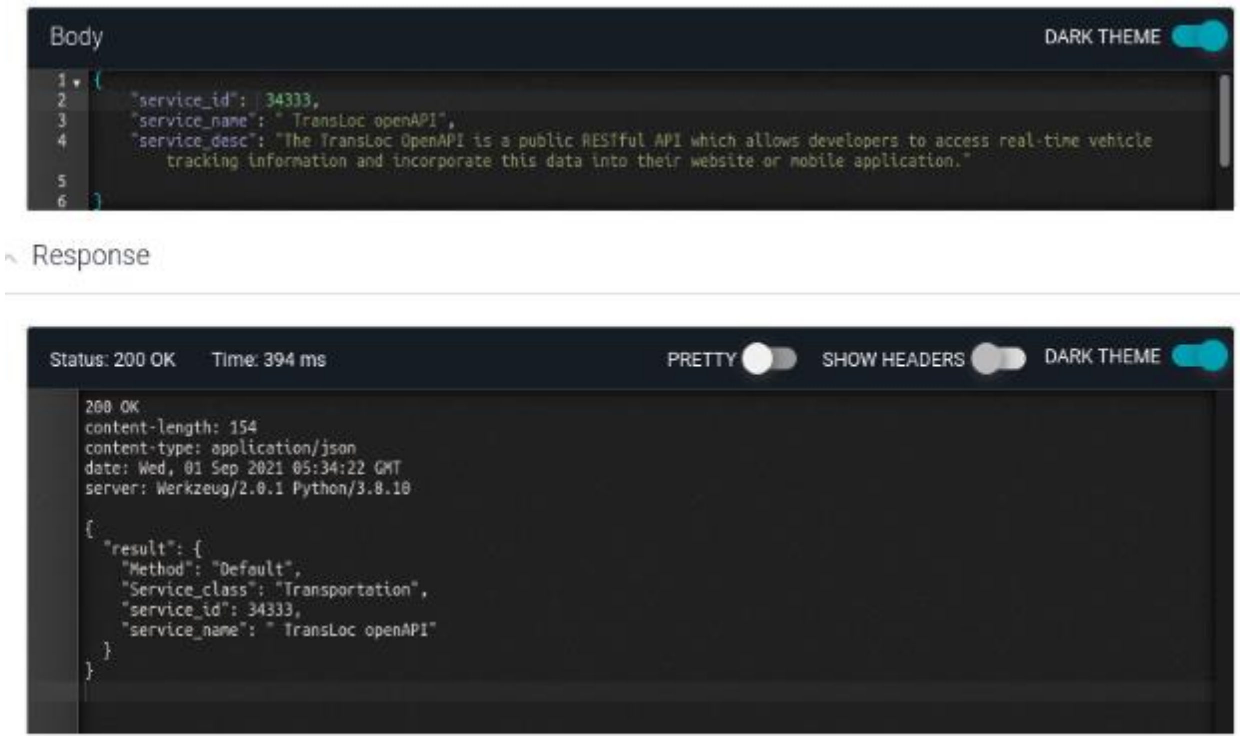


Figure 17 Hybrid supervised/unsupervised approach to classify web services

In this work, service classification takes advantage of text classification trends and deep learning methods. After processing the collected dataset, techniques and algorithms have been applied to solve the service classification problem. A long short-term network (LSTM) was trained by the processed text using Fasttext [16], which is an extension of the word2vec [15]. Additionally, Machine Learning (ML) algorithms such as MultinomialNB classifier, GradientBoosting, and Support Vector Classifier (SVC) use the TF-IDF vectorizer to process the text. This work evaluates the mentioned algorithms based on accuracy, precision, recall, and f1 score metrics. Regarding individual classifiers, results show that weighted SVM has better accuracy and performance. The model is deployed as a REST API, in which a sample request and response is shown in Figure 18.



```
{
    "service_id":  34333,
    "service_name": " TransLoc openAPI",
    "service_desc": "The TransLoc OpenAPI is a public RESTful API which allows developers to access real-time vehicle
        tracking information and incorporate this data into their website or mobile application."

}
```

Status: 200 OK   Time: 394 ms

```
200 OK
content-length: 154
content-type: application/json
date: Wed, 01 Sep 2021 05:34:22 GMT
server: Werkzeug/2.0.1 Python/3.8.10

{
  "result": {
    "Method": "Default",
    "Service_class": "Transportation",
    "service_id": 34333,
    "service_name": " TransLoc openAPI"
  }
}
```

Figure 18 HTTP request and response example for service classification sub-component API

3.3.2 Code Generation

Generative models and language modeling present a powerful role in some IDE features; for example, [21] [41] have shown promising results in practice by using language models. SmartCLIDE tries to use a language modeling pipeline to generate one-line codes based on existing internal and external source codes.

Language modeling is an active area in NLP, which uses different neural network architectures and Transformers [56]. In this technique, the language model is built first, and then the sample is generated with the help of learned models.

In this regard, a large amount of research has been conducted on the use of generative models for generating texts, voices, and images. In other words, a sequence generator can get large text files as an input and can train a model to predict the next character in a sequence. For example, The set of textbooks by an author (e.g.shakespeare) can be used as a data-set. Then, generative models can produce a text similar to the author's writing style.Since recurrent neural networks RNNs (e.g., LSTMs) such as LSTM have hidden states which remember sequence information, they can be useful in language modeling. These hidden states allow LSTM to generate text by character or word to the original training data. However, lack of dependency and parallel computation lead to researchers use transformers. Recently state-of-the-art text generation uses popular auto-regressive models like Generative Pre-trained Transformer (GPT) models, GPT-2 [57] and GPT3 [58] by OpenAI.

The proposed model is trained to generate code suggestions while developers are writing codes. Also, latency while suggesting codes is an issue in this component. Consequently, the proposed pipeline has used DistilGPT2 [59], which is two times faster than GPT2. In terms of training data. The best scenario is to learn from the internal SmartCLIDE codes project. However, deep learning algorithms train well with large data. Therefore, the proposed model needs to use external source codes. The collected source code has been filtered based on quality parameters such as forks and stars in GitHub.

DL algorithms can learn better by massive data, which includes repetitive patterns. Therefore, source codes that have a variety of programmed tasks can decrease the performance [25].

3.3.3 Acceptance test based on Gherkin

One of SmartAssistant's purposes is to invite the user to test the resulting code in a test environment. For this purpose, the SmartAssistant will provide a set of acceptance tests in Gherkin [60] specifications. Acceptance tests are those designed to determine whether the requirements of the defined functionalities have been met for the end-user without having to worry about the details of the implementation. They are usually performed in the final phases of the test stage to verify compliance with the requirements. Gherkin is a logical language with a BDD approach that allows defining behaviours in a way that customers can understand. In this way, Gherkin does not specify how to test software units or classes but defines the behaviour of the software in order to fulfil the proposed scenarios.

SmartCLIDE has defined the model to ask the user to define behaviors in Gherkin format. This definition assumes as an initial phase to define the workflow programmed in BPMN format. Such a workflow in BPMN format defines the information flows through the different tasks of the diagram it represents.

The model for acceptance test recommendation is based on the construction of a history that relates the BPNM diagrams available in the database to the existing acceptance test cases to which they have been linked. This interrelation is done through their history itself, combined with textual similarity techniques and semantic content extraction mechanisms in the case of BPNMs.

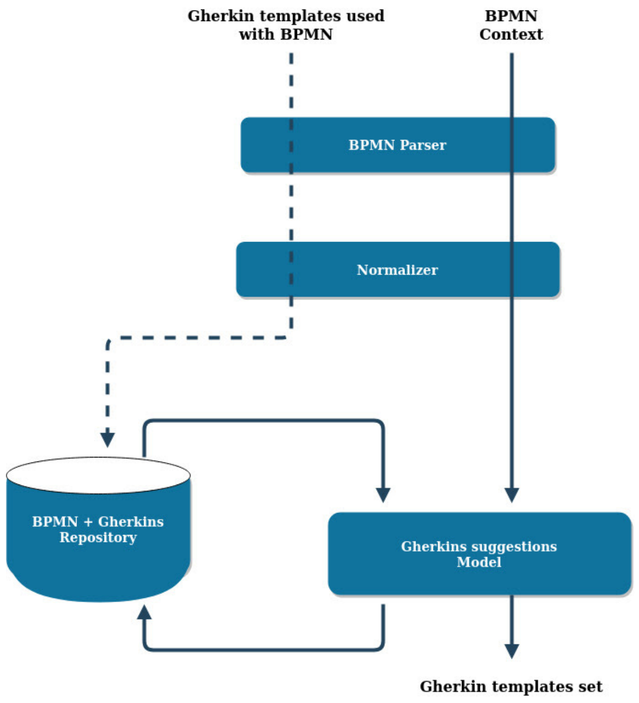

Gherkin templates used with BPMN
BPMN Context
BPMN Parser
Normalizer
BPMN + Gherkins Repository
Gherkins suggestions Model
Gherkin templates set

Figure 19 Acceptance Test Suggestion Model

The different stages of the model are:

- BPMN Context. (Input). The BPMN file in XML format that you are working with in the SmartCLIDE workspace is the starting point of this process. This process should start when the BPMN is completed to avoid making suggestions on how to test an incomplete BPMN.
- BPMN Parser. It extracts the relevant information from the XML format of the different tasks.
- Normalizer. Transforms the information extracted from the BPMN and normalises it so that it can be used as input to the AI model.
- BPMN + Gherkins Repository. This repository stores the history of Gherkins files associated with the BPMNs in which they have been used as acceptance tests.
- Gherkin templates used with BPMN. (Input). This point allows the repository to be continuously fed with the gherkins files that have finally been used as acceptance test of a certain BPMN and that can be used later to improve the recommender system.
- Gherkins suggestions Model. Based on the repository of historical BPMNs + Gherkins, the AI model will obtain as a result a list of Gherkin templates that have been used in previous BPMNs.

- Gherkin templates set. This is the result of the process. This list of gherkins will be offered to the user so that he/she can decide whether to use it as an acceptance test of the flow defined in the BPMN he/she is working with. And here it is necessary to collect the user's feedback so that the Ghekins that are used as acceptance test can be returned as input to the process of acceptance test suggestions in Gherkin format.

As previously indicated, a Case-Based Reasoning approach will be used. This approach is composed of four stages:

- Retrieve: Given a target problem, retrieve from memory the relevant cases to solve it. A case consists of a problem, its solution and, usually, annotations on how the solution was obtained. In our problem, a case consists of a BPMN (XML file) and the acceptance test used for it in Gherkin notation. A Gherkin is retrieved from the database using the KNN algorithm. Given each training algorithm (x, f(x)) and xq as query instances that need to be classified. Also, if assume x1,...,xk stands for k nearest neighbours of xq, if discrete-valued target function:

$$\hat{f}(x_q) \rightarrow \arg\max_{v \in V} \sum_{k\, i=1} \delta(v, f(x_i))$$

  Where δ(a,b) =1 if a=b and where δ(a,b) =0 otherwise and also take mean of f values of k nearest neighbours, if the target function is real-valued.

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

- Reuse: Adapting the solution of the previous case to the target problem. This may involve adapting the solution as necessary to fit the new situation. The user will receive a Gherkin and will be able to modify it for use.
- Revise: Once the previous solution (retrieved Gherkin) has been adapted to the target situation (input BPMN), test the new solution (modified Gherkin) and, if necessary, revise it.
- Retain: Once the solution has been successfully adapted to the target problem, save the resulting case as a new case in the database. The input BPMN and the acceptance tees that have been used (Gherkin) are stored.

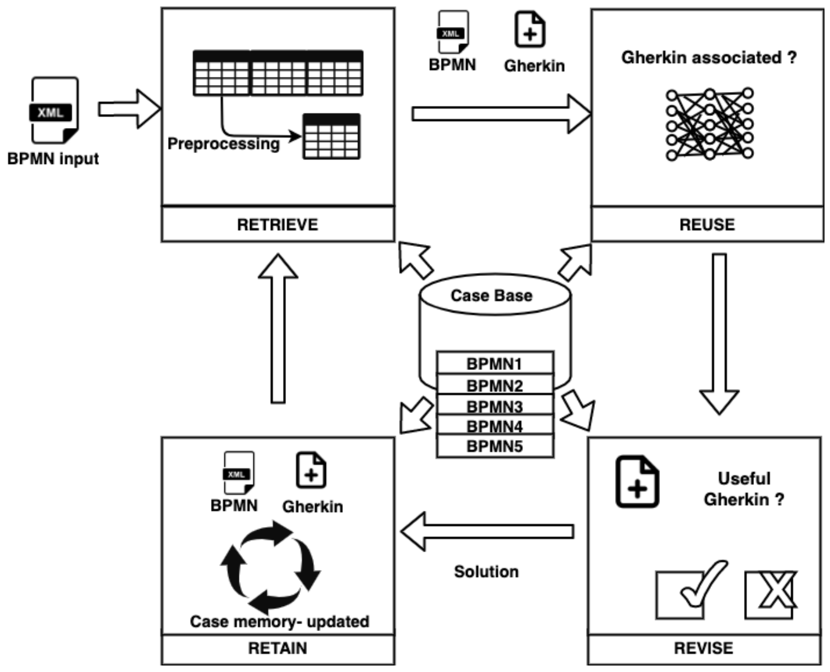


Figure 20 CBR cycle to obtain the acceptance test for an input BPMN

3.3.4 Items Recommender in a BPMN Workflow

This AI-based approach provides recommendations during service composition. The suggestions are based on a selected service composition approach by (BPMN-based work-flow) data representation, existing/history BPMN work-flows, and provided service specification information.

During the composition of the user's workflow in SmartCLIDE represented in BPMN format, the user can receive suggestions for new items to be included next. The adopted solution is executed as back-end component in SmartCLIDE. It receives as input a recommendation request. This request can be a one-off signal from the SmartCLIDE monitoring system when a step in the workflow diagram drawing is completed. The signal must contain the identification of the last node so that suggestions are made based on it. The output signal, in JSON format, will contain the recommended service to link to the node indicated in the request.

3.3.5 Tools and Infrastructure

The following frameworks have been used for the implementation of the service classification and code generation component of the DLE:

Scikit-learn/TensorFlow/Keras: Scikit-learn [61] has a rich history in the official Python general machine learning framework. Also, TensorFlow [62] is a well-known library for deep learning, which Google introduced. . TensorFlow is an open-source library that has both high-level and low-level APIs. Also, Keras [63] provides high-level APIs based on TensorFlow. This can help developers to achieve their goals by less programming.

NLTK/spaCy: The Natural Language Toolkit called NLTK [64] is a library for working with Natural languages in Python. spaCy [65] is another natural language processing library. One of the main differences between NLTK and spaCy is that spaCy uses an object-oriented approach.

Pre-trained word embedding is an example of transfer learning, which helps to use public embeddings. Therefore, it can aid in leverage training quality and also decrease time. Some of the popular pre-trained models are listed below:

- **Word2Vec** [16]**:** (by Google): This model is provided by Google which is trained on Google News data.
- **GloVe** [36]**:** (by Stanford): This model is provided by Standford, which is called Global Vectors (GloVe). This model includes various models from 25, 50, 100, 200, to 300 dimensions based on 2, 6, 42, 840 billion tokens.

Pre-training transformers for language understanding:

- **DistilBERT** [66]**:** DistilBERT is a fast and light Transformer model trained by distilling BERT[65] (Bidirectional Encoder Representations from Transformers). Bert is a transformer that Google has introduced. The significant feature of BERT is using BiLSTM, which is the most promising model for learning long-term dependencies.
- **DistilGPT2** [59]**:** DistilGPT2is a Transformer that is two times faster than GPT2.
- **Tree-sitter** [67] : The tokenization of code is a critical stage that can be based on functions (method, API). Tree-sitter is a library for tokenizing source codes that can parse most programming languages.

3.4 Context Handling

Context Handling UML Component diagram is presented in Figure 21. The Context Monitoring and Context Extraction subcomponents are documented in Table 20 and Table 21, respectively.

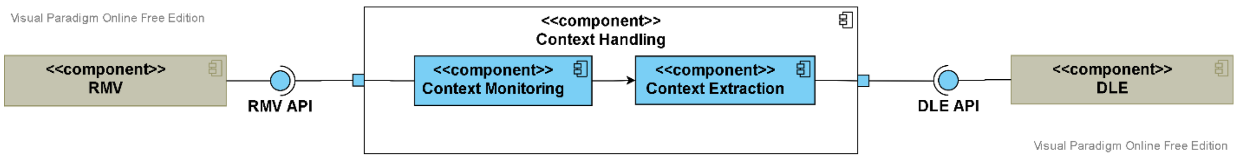


Figure 21: Context Handling Component

Table 20: Context Monitoring Subcomponent

Name	**Context Monitoring**
Functionality	See Section 3.4.1.1
Relevant Use Cases (D1.3)	This module has no direct reference to D1.3. It provides background functionality to support the DLE.
Functional Requirements	See Section 3.4.1.2

Table 21: Context Extraction Subcomponent

Name	**Context Extraction**
Functionality	See Section 3.4.2.1
Relevant Use Cases (D1.3)	This module has no direct reference to D1.3. It provides background functionality to support the DLE.
Functional Requirements	See Section 3.4.2.2

3.4.1 Context Monitoring Specification

The objective of the Context Monitor service is to receive raw data and provide aggregated context data. It is a generic solution for monitoring data sources, which is customizable for different communication protocols and data structures. It enables data pre-processing or data aggregation.

3.4.1.1 Design Approach

The main objective of the Context Monitoring component is to receive raw sensor data and provide aggregated context data. To this end, it allows monitoring of legacy systems in enterprises and products via different interfaces from the Data Access Layer. Therefore, it is able to standardize and correlate data from distinct systems (e.g., map actions from file systems and web services), which later serve as a basis for identification and extraction of context. The main component of the Context Monitoring is the modular monitoring process, used for all monitoring features with an extendable and configurable standardized process (see Figure 22).

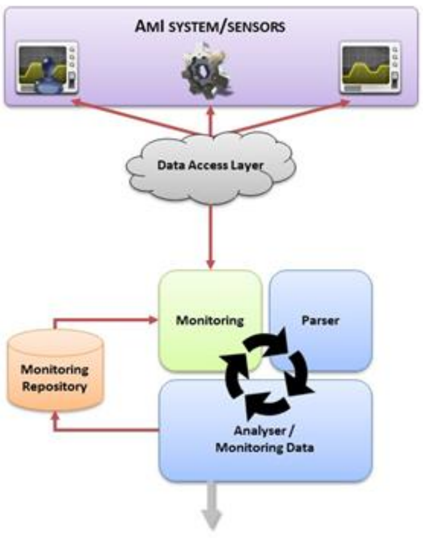


Figure 22: Recurrent monitoring process

The process consists of three parts:

- **Monitoring system/sensor/behaviours** service, which contains all features to monitor legacy systems and devices in enterprises via the Data Access Layer. The distributed monitoring services also call back to this module with their gathered information. The monitoring features can be extended and configured for different systems and do not need to comply with other modules.
- **Parser module**, which contains content parser for the different possible data captured by the monitoring module. The parser offers access to the diverse data possible interacted and therefore monitored with. It provides access to the analyser and may parse available environment properties.
- **Analyzer / Monitoring Data Builder** module, which correlates the monitored content (and maybe environment properties) and constructs the standardized monitoring data to be stored and handed over to the Context Monitoring / Determination service or any other service that needs this information.

For each data source to be monitored an index can be specified, which will hold the environment properties of the resources monitored. Depending on the actual resources to be parsed and analyzed (e.g., XML data from web services), several resource specific plugins can used by the generic Context Monitoring framework. Each of these plugins allows inclusion of parsers and analyzers, which are specified for additional external and legacy systems.

3.4.1.2 Technical Specification

The following section describes the specific technical specifications of the Context Monitoring component.

3.4.1.2.1 Context Monitoring Service

The main classes of the Context Monitoring Service are (Figure 24):

- **IAmIMonitoringService:** The interface defining the System Monitor web service and its public classes. The type of the web service (i.e., SOAP) is defined via Java annotations.
- **AmIMonitoringService:** The main class of the System Monitor. Each configured Monitor is started in a separate thread to allow parallel continuous monitoring of configured data sources. Monitored information is stored in the monitoring data repository, which is made accessible via this service for each monitor.
- **IAmIMonitoringDataRepositoryService:** The interface defining the monitoring repository service. A reference to this service is hold in the System Monitor to allow access to the concrete implementation of the monitoring repository.

- **MonitoringConfiguration:** It holds the configuration of the system monitor during runtime. This configuration object reads the XML configuration file that configures the system monitor.

```xml
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="http://www.atb-bremen.de"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.atb-bremen.de monitoring-config.xsd">

    <indexes>
        <index id="index-git" location="target/indexes/git"/>
    </indexes>

    <datasources>
        <datasource id="datasource-git" type="messagebroker"
                    monitor="de.atb.context.monitoring.monitors.GitMonitor"
                    uri=""
                    options="server=localhost&amp;port=5672&amp;exchange=smartclide-monitoring&amp;topic=monitoring.git.commits.*"
                    class="de.atb.context.monitoring.config.models.datasources.MessageBrokerDataSource"/>
    </datasources>

    <interpreters>
        <interpreter id="interpreter-git">
            <configuration type="*"
                           parser="de.atb.context.monitoring.parser.GitParser"
                           analyser="de.atb.context.monitoring.analyser.GitAnalyser"/>
        </interpreter>
    </interpreters>

    <monitors>
        <monitor id="monitor-git" datasource="datasource-git" interpreter="interpreter-git" index="index-git"/>
    </monitors>

</config>
```

Figure 23: Example of Monitoring Configuration

- **ThreadedMonitor:** The ThreadedMonitor object is responsible for starting and stopping all configured monitoring plugins (Monitors). During runtime, it holds a list of all defined monitors and manages their states.

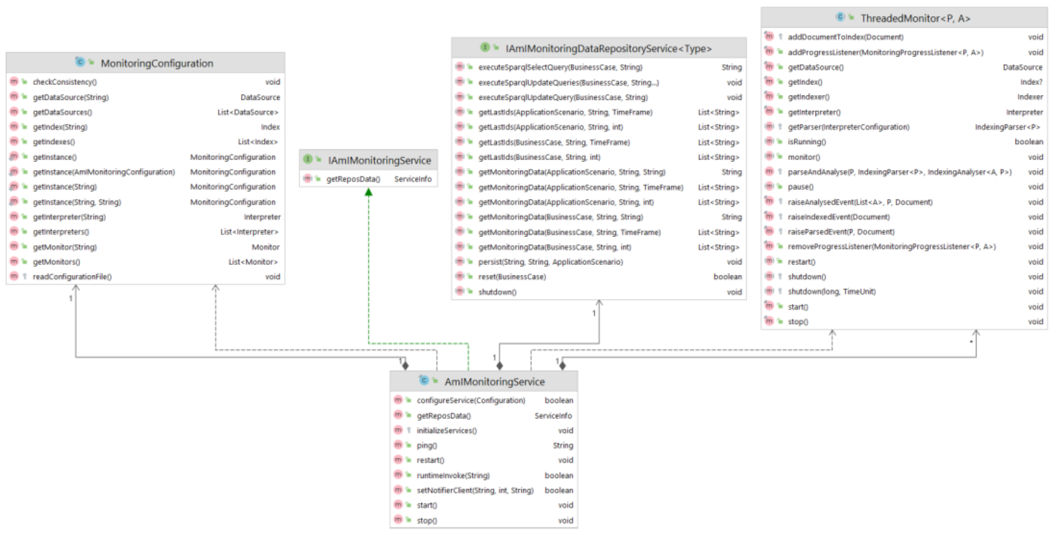

MonitoringConfiguration
checkConsistency() void
getDataSource(String) DataSource
getDataSources() List<DataSource>
getIndex(String) Index
getIndexes() List<Index>
getInstance() MonitoringConfiguration
getInstance(AmIMonitoringConfiguration) MonitoringConfiguration
getInstance(String) MonitoringConfiguration
getInstance(String, String) MonitoringConfiguration
getInterpreter(String) Interpreter
getInterpreters() List<Interpreter>
getMonitor(String) Monitor
getMonitors() List<Monitor>
readConfigurationFile() void
IAmIMonitoringService
getReposData() ServiceInfo
IAmIMonitoringDataRepositoryService<Type>
executeSparqlSelectQuery(BusinessCase, String) String
executeSparqlUpdateQueries(BusinessCase, String...) void
executeSparqlUpdateQuery(BusinessCase, String) void
getLastIds(ApplicationScenario, String, TimeFrame) List<String>
getLastIds(ApplicationScenario, String, int) List<String>
getLastIds(BusinessCase, String, TimeFrame) List<String>
getLastIds(BusinessCase, String, int) List<String>
getMonitoringData(ApplicationScenario, String, String) String
getMonitoringData(ApplicationScenario, String, TimeFrame) List<String>
getMonitoringData(ApplicationScenario, String, int) List<String>
getMonitoringData(BusinessCase, String, String) String
getMonitoringData(BusinessCase, String, TimeFrame) List<String>
getMonitoringData(BusinessCase, String, int) List<String>
persist(String, String, ApplicationScenario) void
reset(BusinessCase) boolean
shutdown() void
ThreadedMonitor<P, A>
addDocumentToIndex(Document) void
addProgressListener(MonitoringProgressListener<P, A>) void
getDataSource() DataSource
getIndex() Index?
getIndexer() Indexer
getInterpreter() Interpreter
getParser(InterpreterConfiguration) IndexingParser<P>
isRunning() boolean
monitor() void
parseAndAnalyse(P, IndexingParser<P>, IndexingAnalyser<A, P>) void
pause() void
raiseAnalysedEvent(List<A>, P, Document) void
raiseIndexedEvent(Document) void
raiseParsedEvent(P, Document) void
removeProgressListener(MonitoringProgressListener<P, A>) void
restart() void
shutdown() void
shutdown(long, TimeUnit) void
start() void
stop() void
AmIMonitoringService
configureService(Configuration) boolean
getReposData() ServiceInfo
initializeServices() void
ping() String
restart() void
runtimeInvoke(String) boolean
setNotifierClient(String, int, String) boolean
start() void
stop() void
1
1
1
1
1
1
*

Figure 24: Class diagram of the Context Monitoring Service

3.4.1.2.2 Monitoring Repository Service

The main classes of the Service are:

- **IAmIMonitoringDataRepositoryService**: Interface defining the monitoring repository service. The type of the web service (i.e., SOAP) is defined via Java annotations.
- **AmIMonitoringDataRepositoryService**: The main class of the Monitoring Repository service. Each configured Monitor is started in a separate thread to allow parallel continuous monitoring

of configured data sources. Monitored information are stored in the monitoring data repository that is made accessible via this service for each monitor.

- **PersistenceUnitService**: An abstract class, where functionalities related to persistence are implemented.
- **MonitoringDataRepository:** This class implements all required functionalities to persist and load data to and from the monitoring repository.The data is stored in an SDB[6] repository.

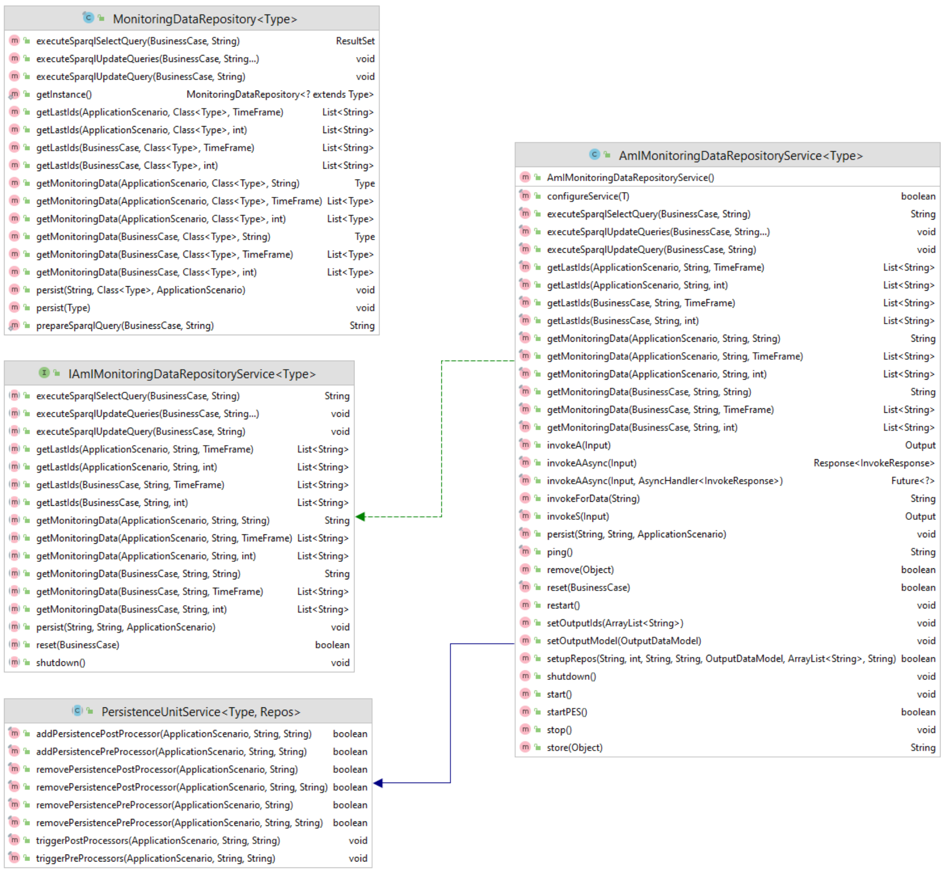

MonitoringDataRepository<Type>
executeSparqlSelectQuery(BusinessCase, String) ResultSet
executeSparqlUpdateQueries(BusinessCase, String...) void
executeSparqlUpdateQuery(BusinessCase, String) void
getInstance() MonitoringDataRepository<? extends Type>
getLastIds(ApplicationScenario, Class<Type>, TimeFrame) List<String>
getLastIds(ApplicationScenario, Class<Type>, int) List<String>
getLastIds(BusinessCase, Class<Type>, TimeFrame) List<String>
getLastIds(BusinessCase, Class<Type>, int) List<String>
getMonitoringData(ApplicationScenario, Class<Type>, String) Type
getMonitoringData(ApplicationScenario, Class<Type>, TimeFrame) List<Type>
getMonitoringData(ApplicationScenario, Class<Type>, int) List<Type>
getMonitoringData(BusinessCase, Class<Type>, String) Type
getMonitoringData(BusinessCase, Class<Type>, TimeFrame) List<Type>
getMonitoringData(BusinessCase, Class<Type>, int) List<Type>
persist(String, Class<Type>, ApplicationScenario) void
persist(Type) void
prepareSparqlQuery(BusinessCase, String) String
IAmlMonitoringDataRepositoryService<Type>
executeSparqlSelectQuery(BusinessCase, String) String
executeSparqlUpdateQueries(BusinessCase, String...) void
executeSparqlUpdateQuery(BusinessCase, String) void
getLastIds(ApplicationScenario, String, TimeFrame) List<String>
getLastIds(ApplicationScenario, String, int) List<String>
getLastIds(BusinessCase, String, TimeFrame) List<String>
getLastIds(BusinessCase, String, int) List<String>
getMonitoringData(ApplicationScenario, String, String) String
getMonitoringData(ApplicationScenario, String, TimeFrame) List<String>
getMonitoringData(ApplicationScenario, String, int) List<String>
getMonitoringData(BusinessCase, String, String) String
getMonitoringData(BusinessCase, String, TimeFrame) List<String>
getMonitoringData(BusinessCase, String, int) List<String>
persist(String, String, ApplicationScenario) void
reset(BusinessCase) boolean
shutdown() void
PersistenceUnitService<Type, Repos>
addPersistencePostProcessor(ApplicationScenario, String, String) boolean
addPersistencePreProcessor(ApplicationScenario, String, String) boolean
removePersistencePostProcessor(ApplicationScenario, String) boolean
removePersistencePostProcessor(ApplicationScenario, String, String) boolean
removePersistencePreProcessor(ApplicationScenario, String) boolean
removePersistencePreProcessor(ApplicationScenario, String, String) boolean
triggerPostProcessors(ApplicationScenario, String, String) void
triggerPreProcessors(ApplicationScenario, String, String) void
AmlMonitoringDataRepositoryService<Type>
AmlMonitoringDataRepositoryService()
configureService(T) boolean
executeSparqlSelectQuery(BusinessCase, String) String
executeSparqlUpdateQueries(BusinessCase, String...) void
executeSparqlUpdateQuery(BusinessCase, String) void
getLastIds(ApplicationScenario, String, TimeFrame) List<String>
getLastIds(ApplicationScenario, String, int) List<String>
getLastIds(BusinessCase, String, TimeFrame) List<String>
getLastIds(BusinessCase, String, int) List<String>
getMonitoringData(ApplicationScenario, String, String) String
getMonitoringData(ApplicationScenario, String, TimeFrame) List<String>
getMonitoringData(ApplicationScenario, String, int) List<String>
getMonitoringData(BusinessCase, String, String) String
getMonitoringData(BusinessCase, String, TimeFrame) List<String>
getMonitoringData(BusinessCase, String, int) List<String>
invokeA(Input) Output
invokeAAsync(Input) Response<InvokeResponse>
invokeAAsync(Input, AsyncHandler<InvokeResponse>) Future<?>
invokeForData(String) String
invokeS(Input) Output
persist(String, String, ApplicationScenario) void
ping() String
remove(Object) boolean
reset(BusinessCase) boolean
restart() void
setOutputIds(ArrayList<String>) void
setOutputModel(OutputDataModel) void
setupRepos(String, int, String, String, OutputDataModel, ArrayList<String>, String) boolean
shutdown() void
start() void
startPES() boolean
stop() void
store(Object) String

Figure 25: Class diagram of the Monitoring Repository Service

3.4.1.2.3 Monitors

The System Monitor has several generic monitors that were implemented in the current prototype. Each of the monitors is implemented as a software service. These monitors are:

- a Monitoring Service for the SmartCLIDE message broker
- a Monitoring Service for SOAP-based web services,
- a Monitoring Service for legacy file systems and
- a Monitoring Service for relational databases

These monitors were implemented in the Early Prototype:

- **ThreadedMonitor:** The ThreadedMonitor object is responsible for starting and stopping all configured monitoring plugins (Monitors). During runtime, it holds a list of all defined monitors and manages their states.
- **FileSystemMonitor:** This generic monitor checks files in a specific folder of a filesystem for changes. For example, a source code files in a file system.

[6] SDB Triple Store - https://jena.apache.org/documentation/sdb/

- **DatabaseMonitor:** This generic monitor allows to observe a database for changes in its schema. This monitor was not used in a validation scenario, but for some unit test cases during the development.
- **MessageBrokerMonitor:** This generic monitor checks for events coming from a message broker. For example, a monitor of the RMV sends information about new monitored information via a message broker to the Context Handling.
- **WebServiceMonitor:** This generic monitor retrieves data from a system that makes observable data available via a web service. This monitor was not used in a validation scenario, but for some unit test cases during the development.

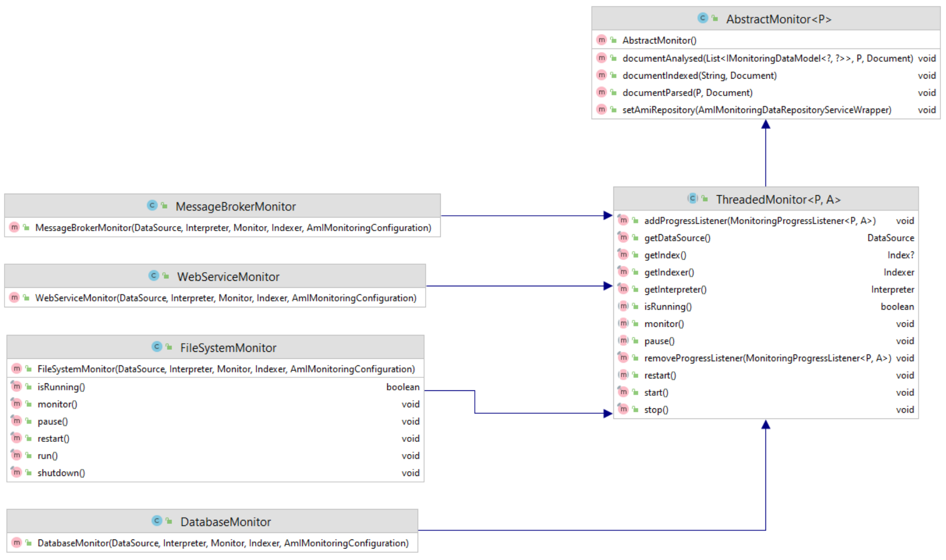

AbstractMonitor<P>
AbstractMonitor()
documentAnalysed(List<IMonitoringDataModel<?, ?>>, P, Document) void
documentIndexed(String, Document) void
documentParsed(P, Document) void
setAmiRepository(AmIMonitoringDataRepositoryServiceWrapper) void
ThreadedMonitor<P, A>
addProgressListener(MonitoringProgressListener<P, A>) void
getDataSource() DataSource
getIndex() Index?
getIndexer() Indexer
getInterpreter() Interpreter
isRunning() boolean
monitor() void
pause() void
removeProgressListener(MonitoringProgressListener<P, A>) void
restart() void
start() void
stop() void
MessageBrokerMonitor
MessageBrokerMonitor(DataSource, Interpreter, Monitor, Indexer, AmIMonitoringConfiguration)
WebServiceMonitor
WebServiceMonitor(DataSource, Interpreter, Monitor, Indexer, AmIMonitoringConfiguration)
FileSystemMonitor
FileSystemMonitor(DataSource, Interpreter, Monitor, Indexer, AmIMonitoringConfiguration)
isRunning() boolean
monitor() void
pause() void
restart() void
run() void
shutdown() void
DatabaseMonitor
DatabaseMonitor(DataSource, Interpreter, Monitor, Indexer, AmIMonitoringConfiguration)

Figure 26: Class diagram of the System Monitors

3.4.1.2.4 Parser

The main class for all parsers is the IndexingParser. The following briefly describes the implemented parsers:

- **IndexingParser:** This is an abstract monitoring parser. The corresponding analyzer for each parser is created during initialization. The IndexingParser holds a reference to the corresponding analyzer, which is executed after successful parsing of data.
- **FileParser:** Abstract parser that parses data from flat files. This class contains the generic functionality of the parser for files located in a file system. The parsing process itself has to be implemented by each concrete implementation.
- **DatabaseParser:** Abstract parser that parses data from a database. This class contains the generic functionality of the parser for data sets located in a database that can be accessed via jdbc. The parsing process itself has to be implemented by each concrete implementation.
- **MessageBrokerParser:** Abstract parser that parses data received from a message broker. This class contains the generic functionality of the parser for data sets received via a message broker. In the current implementation the message broker is based JRabbitMQ (see Section MoM). The parsing process itself has to be implemented by each concrete implementation.
- **WebServiceParser:** Abstract parser that parses data from a web service. This class contains the generic functionality of the parser for data set accessible via SOAP based web services. The parsing process itself has to be implemented by each concrete implementation.

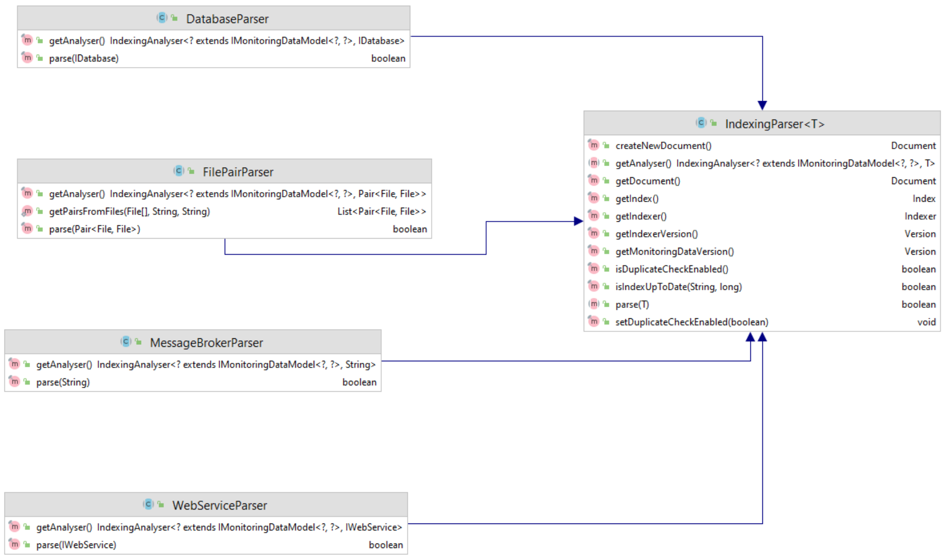

DatabaseParser
getAnalyser() IndexingAnalyser<? extends IMonitoringDataModel<?, ?>, IDatabase>
parse(IDatabase) boolean
IndexingParser<T>
createNewDocument() Document
getAnalyser() IndexingAnalyser<? extends IMonitoringDataModel<?, ?>, T>
getDocument() Document
getIndex() Index
getIndexer() Indexer
getIndexerVersion() Version
getMonitoringDataVersion() Version
isDuplicateCheckEnabled() boolean
isIndexUpToDate(String, long) boolean
parse(T) boolean
setDuplicateCheckEnabled(boolean) void
FilePairParser
getAnalyser() IndexingAnalyser<? extends IMonitoringDataModel<?, ?>, Pair<File, File>>
getPairsFromFiles(File[], String, String) List<Pair<File, File>>
parse(Pair<File, File>) boolean
MessageBrokerParser
getAnalyser() IndexingAnalyser<? extends IMonitoringDataModel<?, ?>, String>
parse(String) boolean
WebServiceParser
getAnalyser() IndexingAnalyser<? extends IMonitoringDataModel<?, ?>, IWebService>
parse(IWebService) boolean

Figure 27: Class Diagram of the Parsers

3.4.1.2.5 Analyzer

The main class for all analysers is the IndexingAnalyser. The following briefly describes the implemented analysers:

- **IndexingAnalyser:** This is an abstract monitoring analyser.
- **FileAnalyser:** Abstract analyser that processes files from a file system. This class contains the generic functionality for analysing data files located in a file system. The analysing process itself has to be implemented by each concrete implementation.
- **DatabaseAnalyser:** Abstract analyser that processes data from a database. This class contains the generic functionality for analysing data sets located in a database that can be accessed via jdbc. The analysing process itself has to be implemented by each concrete implementation.
- **MessageBrokerAnalyser:** Abstract analyser that processes data from a database. This class contains the generic functionality for analysing data received from a message broker. The analysing process itself has to be implemented by each concrete implementation.
- **WebServiceAnalyser:** Abstract analyser that processes data from a web service. This class contains the generic functionality for analysing data sets that can be accessed via SOAP based web services. The analysing process itself has to be implemented by each concrete implementation.

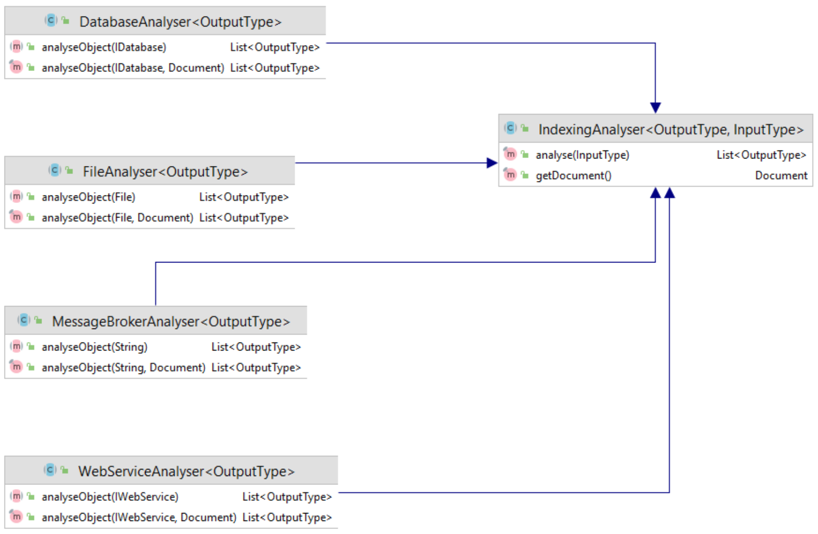

DatabaseAnalyser<OutputType>
analyseObject(IDatabase) List<OutputType>
analyseObject(IDatabase, Document) List<OutputType>
IndexingAnalyser<OutputType, InputType>
analyse(InputType) List<OutputType>
getDocument() Document
FileAnalyser<OutputType>
analyseObject(File) List<OutputType>
analyseObject(File, Document) List<OutputType>
MessageBrokerAnalyser<OutputType>
analyseObject(String) List<OutputType>
analyseObject(String, Document) List<OutputType>
WebServiceAnalyser<OutputType>
analyseObject(IWebService) List<OutputType>
analyseObject(IWebService, Document) List<OutputType>

Figure 28: Class Diagram of the Analysers

3.4.1.2.6 Monitoring Data Models

The main class for all monitoring data models is the MonitoringDataModel. The following briefly describes the monitoring data models:

- **IMonitoringDataModel:** Interface defining the data model for monitored data. The monitoring data model will be created by the analyzer and persisted into the monitoring repository.
- **IMonitoringData:** Interface defining which methods have to be implemented for a concrete implementation of monitoring data.

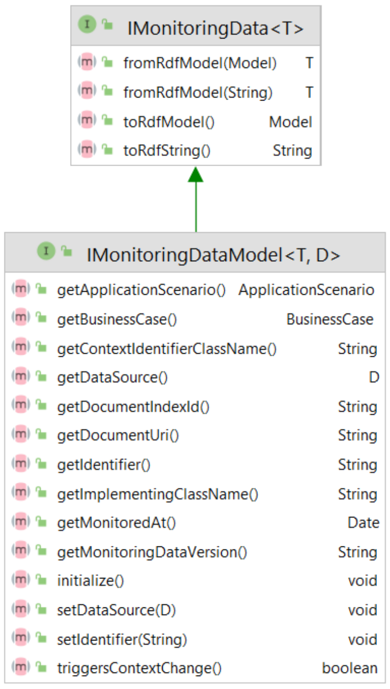

IMonitoringData<T>
fromRdfModel(Model) T
fromRdfModel(String) T
toRdfModel() Model
toRdfString() String
IMonitoringDataModel<T, D>
getApplicationScenario() ApplicationScenario
getBusinessCase() BusinessCase
getContextIdentifierClassName() String
getDataSource() D
getDocumentIndexId() String
getDocumentUri() String
getIdentifier() String
getImplementingClassName() String
getMonitoredAt() Date
getMonitoringDataVersion() String
initialize() void
setDataSource(D) void
setIdentifier(String) void
triggersContextChange() boolean

Figure 29: Class Diagram – Monitoring Data Model

3.4.1.3 Interface Specification

Table 22: Context Monitoring Interface

No	Interface (/API)	Description	Type	
			Provided	Required
1	**MonitoringRequest/ list_available_monitors**	Interface provided by RMV to obtain a list of all available monitors		*
2	**MonitoringRequest/ register_for_notification**	Interface provided by RMV to subscribe to the monitored values for a given monitor and sensor		*
3	**ContextMonitor/ listen_to_notification**	Notification invoked by MoM about new data available for Context Monitoring	*	
4	**ContextMonitoring/ notifyExtraction**	Notifies the Context Extraction about new data available in the monitoring repository to be processed by Context Extraction	*	

3.4.2 Context Extraction Specification

The objective of the Context Extractor service is to identify the context of development processes or specified systems and to provide it for further use within the SmartCLIDE solution to other modules or external systems.

3.4.2.1 Design Approach

This section provides the specification of the parts of which the Context Extraction component consists. The service uses a context model for an integrated representation of the observed environment (e.g., products, systems, machines, processes, or users).

The Context Extraction component consists of the following services:

- **Context identification:** It determines the current context of products, machines and production processes using monitored raw data from the Context Monitoring component and historic context information stored in a context repository, based on the context model.
- **Context reasoning**: It uses reasoning rules on the context provided by the context identification part, and determines more accurate context, which cannot be directly identified during the process in the context identification part. Fully elaborated specification of the context reasoning will be provided in D3.2.
- **Context provision**: It provides detected context to the other services.

As shown in the figure above, the Context Extraction component identifies context based on monitoring data, provided by the monitoring module, enhances it through different types of reasoning techniques (i.e., context reasoning), and provides the refined context for further exploitation to modules (or external systems).

The Context Extraction module is based on the services developed within the projects Self-Learning[7] and SAFIRE[8]. The module will be extended to address the specific needs of the SmartCLIDE solution, as well as the particularities of the three business cases. The collection of input data will be conducted by the monitoring module, and the Context Extraction part will analyse structured and unstructured data in order to specify the current context and identify the parameters of the product/machine or process environment could affect its performance. The identification of the respective context is carried out based on the context models, which define each time what information is relevant to the observed context. The identified context information is being stored in the context repository as annotation to the content that is used in the observed context.

3.4.2.2 Technical Specification

The following section describes the specific technical specifications of the Context Extraction component.

3.4.2.2.1 Context Extraction Service

The main classes of the Context Extraction are:

- **IContextExtractionService:** The interface defining the Context Extractor service and its public classes. The type of the web service (i.e., SOAP) is defined via Java annotations.
- **ContextExtractionService:** The main class of the Context Extraction component.
- **IMonitoringDataModel:** A reference to the monitoring data model, which contains the current monitoring data to be used for Context Extraction.

[7] https://cordis.europa.eu/project/id/228857
[8] https://www.safire-factories.org/

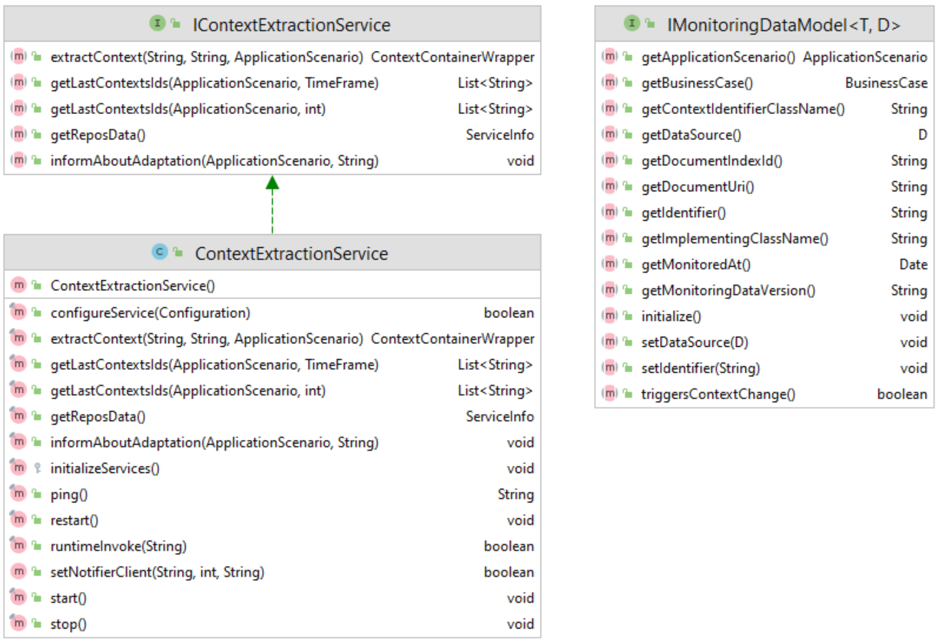

IContextExtractionService
extractContext(String, String, ApplicationScenario) ContextContainerWrapper
getLastContextsIds(ApplicationScenario, TimeFrame) List<String>
getLastContextsIds(ApplicationScenario, int) List<String>
getReposData() ServiceInfo
informAboutAdaptation(ApplicationScenario, String) void
ContextExtractionService
ContextExtractionService()
configureService(Configuration) boolean
extractContext(String, String, ApplicationScenario) ContextContainerWrapper
getLastContextsIds(ApplicationScenario, TimeFrame) List<String>
getLastContextsIds(ApplicationScenario, int) List<String>
getReposData() ServiceInfo
informAboutAdaptation(ApplicationScenario, String) void
initializeServices() void
ping() String
restart() void
runtimeInvoke(String) boolean
setNotifierClient(String, int, String) boolean
start() void
stop() void
IMonitoringDataModel<T, D>
getApplicationScenario() ApplicationScenario
getBusinessCase() BusinessCase
getContextIdentifierClassName() String
getDataSource() D
getDocumentIndexId() String
getDocumentUri() String
getIdentifier() String
getImplementingClassName() String
getMonitoredAt() Date
getMonitoringDataVersion() String
initialize() void
setDataSource(D) void
setIdentifier(String) void
triggersContextChange() boolean

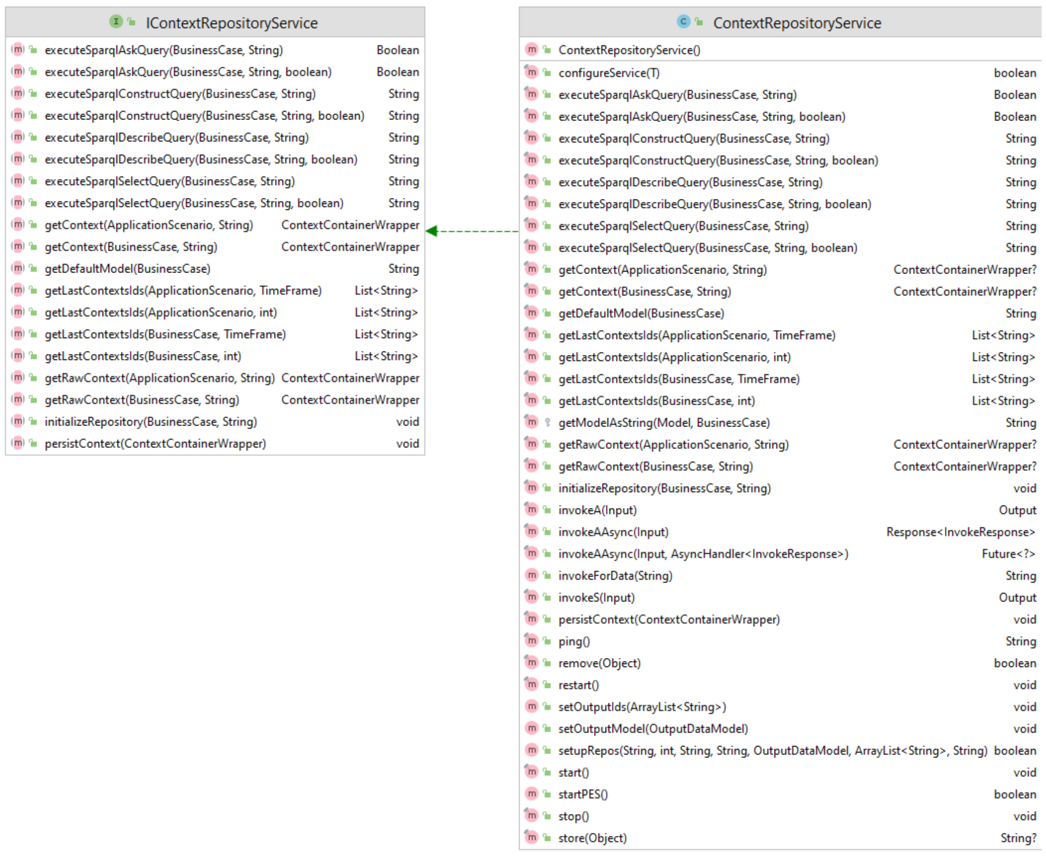
Figure 30: Class diagram of the Context Extraction Service

3.4.2.2.2 Context Repository Service

The main classes of the Context Repository Service are:

- **IContextRepositoryService:** An interface defining the context repository service. The type of the web service (i.e., SOAP) is defined via Java annotations.
- **ContextRepositoryService:** The main class of the context repository service. The identified context are stored in the context repository. Via the configuration of the Context Extraction service, the usage of a reasoner can be configured. Reasoning functionality is included in this object, because it is responsible to query the context repository. In the full prototype implementation, the Pellet[9] reasoner will be used.

[9] https://github.com/stardog-union/pellet



IContextRepositoryService	
executeSparqlAskQuery(BusinessCase, String)	Boolean
executeSparqlAskQuery(BusinessCase, String, boolean)	Boolean
executeSparqlConstructQuery(BusinessCase, String)	String
executeSparqlConstructQuery(BusinessCase, String, boolean)	String
executeSparqlDescribeQuery(BusinessCase, String)	String
executeSparqlDescribeQuery(BusinessCase, String, boolean)	String
executeSparqlSelectQuery(BusinessCase, String)	String
executeSparqlSelectQuery(BusinessCase, String, boolean)	String
getContext(ApplicationScenario, String)	ContextContainerWrapper
getContext(BusinessCase, String)	ContextContainerWrapper
getDefaultModel(BusinessCase)	String
getLastContextsIds(ApplicationScenario, TimeFrame)	List<String>
getLastContextsIds(ApplicationScenario, int)	List<String>
getLastContextsIds(BusinessCase, TimeFrame)	List<String>
getLastContextsIds(BusinessCase, int)	List<String>
getRawContext(ApplicationScenario, String)	ContextContainerWrapper
getRawContext(BusinessCase, String)	ContextContainerWrapper
initializeRepository(BusinessCase, String)	void
persistContext(ContextContainerWrapper)	void

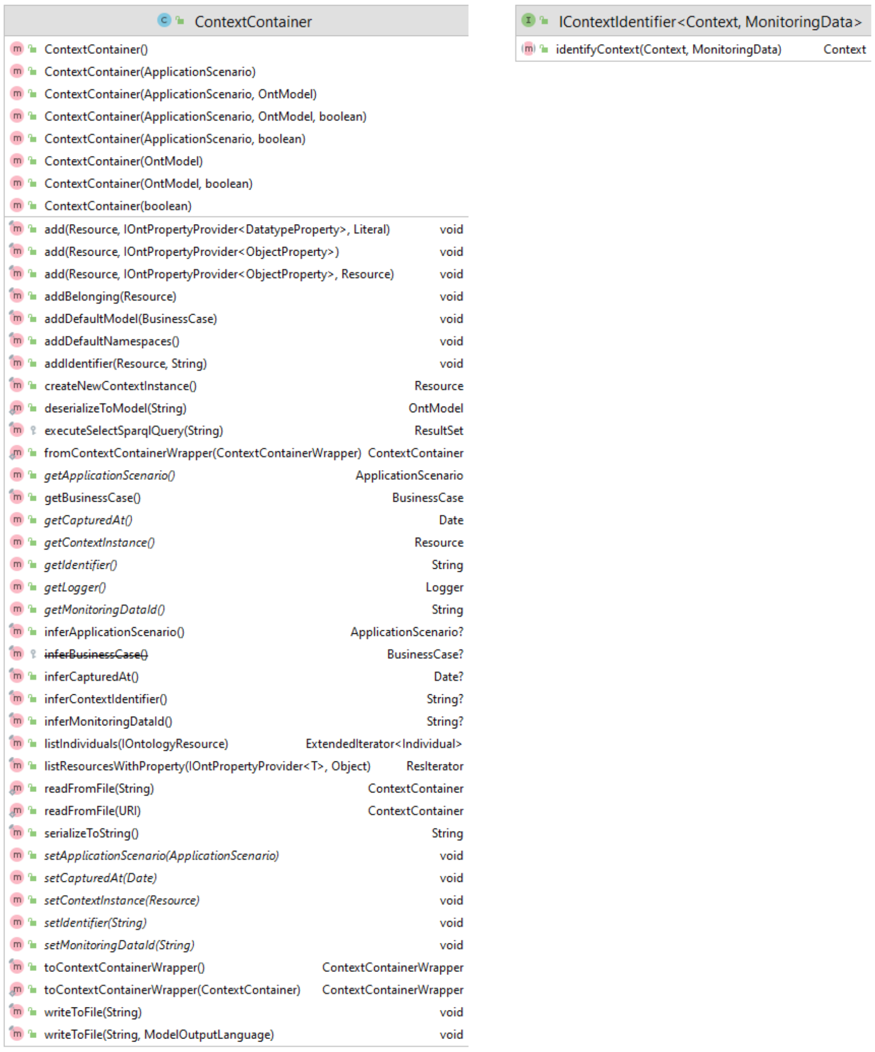
ContextRepositoryService	
ContextRepositoryService()	
configureService(T)	boolean
executeSparqlAskQuery(BusinessCase, String)	Boolean
executeSparqlAskQuery(BusinessCase, String, boolean)	Boolean
executeSparqlConstructQuery(BusinessCase, String)	String
executeSparqlConstructQuery(BusinessCase, String, boolean)	String
executeSparqlDescribeQuery(BusinessCase, String)	String
executeSparqlDescribeQuery(BusinessCase, String, boolean)	String
executeSparqlSelectQuery(BusinessCase, String)	String
executeSparqlSelectQuery(BusinessCase, String, boolean)	String
getContext(ApplicationScenario, String)	ContextContainerWrapper?
getContext(BusinessCase, String)	ContextContainerWrapper?
getDefaultModel(BusinessCase)	String
getLastContextsIds(ApplicationScenario, TimeFrame)	List<String>
getLastContextsIds(ApplicationScenario, int)	List<String>
getLastContextsIds(BusinessCase, TimeFrame)	List<String>
getLastContextsIds(BusinessCase, int)	List<String>
getModelAsString(Model, BusinessCase)	String
getRawContext(ApplicationScenario, String)	ContextContainerWrapper?
getRawContext(BusinessCase, String)	ContextContainerWrapper?
initializeRepository(BusinessCase, String)	void
invokeA(Input)	Output
invokeAAsync(Input)	Response<InvokeResponse>
invokeAAsync(Input, AsyncHandler<InvokeResponse>)	Future<?>
invokeForData(String)	String
invokeS(Input)	Output
persistContext(ContextContainerWrapper)	void
ping()	String
remove(Object)	boolean
restart()	void
setOutputIds(ArrayList<String>)	void
setOutputModel(OutputDataModel)	void
setupRepos(String, int, String, String, OutputDataModel, ArrayList<String>, String)	boolean
start()	void
startPES()	boolean
stop()	void
store(Object)	String?

Figure 31: Class diagram of the Context Repository Service

3.4.2.2.3 Context Identifiers

The main classes of the Context Identifiers are:

- **IContextIdentifier:** The interface defining a context identifier. A context identifier is a wrapper used to identify the context based on monitored data and the context model. In each concrete implementation of a context identifier the usage of a reasoner can be defined.
- **ContextContainer:** This class is a wrapper object that holds an identified context during runtime.



ContextContainer	
ContextContainer()	
ContextContainer(ApplicationScenario)	
ContextContainer(ApplicationScenario, OntModel)	
ContextContainer(ApplicationScenario, OntModel, boolean)	
ContextContainer(ApplicationScenario, boolean)	
ContextContainer(OntModel)	
ContextContainer(OntModel, boolean)	
ContextContainer(boolean)	
add(Resource, IOntPropertyProvider<DatatypeProperty>, Literal)	void
add(Resource, IOntPropertyProvider<ObjectProperty>)	void
add(Resource, IOntPropertyProvider<ObjectProperty>, Resource)	void
addBelonging(Resource)	void
addDefaultModel(BusinessCase)	void
addDefaultNamespaces()	void
addIdentifier(Resource, String)	void
createNewContextInstance()	Resource
deserializeToModel(String)	OntModel
executeSelectSparqlQuery(String)	ResultSet
fromContextContainerWrapper(ContextContainerWrapper)	ContextContainer
getApplicationScenario()	ApplicationScenario
getBusinessCase()	BusinessCase
getCapturedAt()	Date
getContextInstance()	Resource
getIdentifier()	String
getLogger()	Logger
getMonitoringDataId()	String
inferApplicationScenario()	ApplicationScenario?
~~inferBusinessCase()~~	BusinessCase?
inferCapturedAt()	Date?
inferContextIdentifier()	String?
inferMonitoringDataId()	String?
listIndividuals(IOntologyResource)	ExtendedIterator<Individual>
listResourcesWithProperty(IOntPropertyProvider<T>, Object)	ResIterator
readFromFile(String)	ContextContainer
readFromFile(URI)	ContextContainer
serializeToString()	String
setApplicationScenario(ApplicationScenario)	void
setCapturedAt(Date)	void
setContextInstance(Resource)	void
setIdentifier(String)	void
setMonitoringDataId(String)	void
toContextContainerWrapper()	ContextContainerWrapper
toContextContainerWrapper(ContextContainer)	ContextContainerWrapper
writeToFile(String)	void
writeToFile(String, ModelOutputLanguage)	void

IContextIdentifier<Context, MonitoringData>	
identifyContext(Context, MonitoringData)	Context

Figure 32: Class Diagram – Context Identifiers

3.4.2.3 Interface Specification

Table 23: Context Extraction Subcomponent Interface

No	Interface (/API)	Description	Type	
			Provided	Required
1	**ContextMonitoring/ listen_to_notification**	Notification invoked by Context Monitoring about new data available for Context Extraction		*
2	**ContextExtraction/ notifyDLE**	Notifies the DLE about new identified context. The Context Monitoring send a notification including the identified context to the MoM.	*	

4 Backend Services

This section presents the early design approach of the core backend components. All technology providers have described their individual design approaches from the technological perspective. A complimentary description of the research-oriented approaches is described in "*D2.1 SmartCLIDE Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment*". The following artifacts have been specified for each component:

- Component diagram using UML 2.5 Class diagram notation
- Specification table for each subcomponent including a brief description of its functionality and links to the functional requirements (from D1.2) and corresponding use case (from D1.3)
- Interface Specification table describing provided and required interfaces as illustrated on the component diagram.

The overall SmartCLIDE component diagram is presented in Figure 33. The diagram in Figure 33 includes only high-level components. Detailed component diagrams, including subcomponents and their relationships, are presented and discussed in in Subsection 4.1 - 4.9.

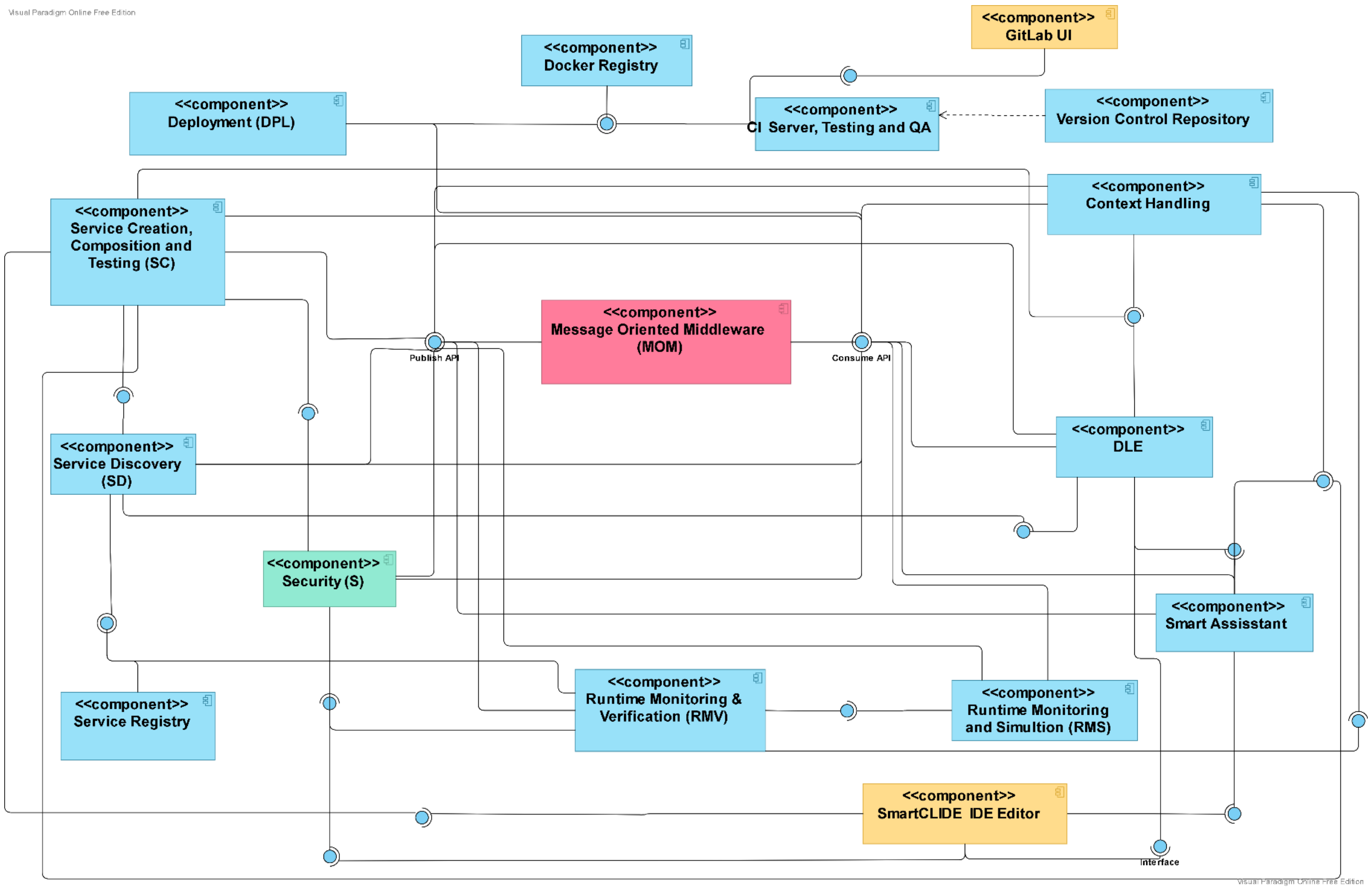

Visual Paradigm Online Free Edition
<<component>> GitLab UI
<<component>> Docker Registry
<<component>> Deployment (DPL)
<<component>> CI Server, Testing and QA
<<component>> Version Control Repository
<<component>> Context Handling
<<component>> Service Creation, Composition and Testing (SC)
<<component>> Message Oriented Middleware (MOM)
Publish API
Consume API
<<component>> DLE
<<component>> Service Discovery (SD)
<<component>> Security (S)
<<component>> Smart Assisstant
<<component>> Service Registry
<<component>> Runtime Monitoring & Verification (RMV)
<<component>> Runtime Monitoring and Simultion (RMS)
<<component>> SmartCLIDE IDE Editor
Interface
Visual Paradigm Online Free Edition

Figure 33: Component - based SmartCLIDE architecture update

4.1 Source code repository

4.1.1 Technology Choice

At the time of writing this document, the predominant version control system used for both open-source and proprietary software is Git[10]. Git is a distributed version control system where the complete version history of a codebase resides on every copy, or clone, of a git repository. The convenience of this approach is that a developer may make changes to a local copy of the codebase, create new branches, commit changes, and alter the local commit history without being connected to a central repository. Once satisfied with the local changes, the developer can synchronise them with a remote repository using the "push" mechanism, while changes made by other developers can be obtained from a remote repository using git's "pull" mechanism. The remote repository is usually hosted on a central git server, shared by a team who co-ordinate their development effort by pushing and pulling project updates to and from the central server.

A consequence of the dominance of Git in the market is that the majority of development tools have excellent support for it as a version control system, either built-in or available as a plugin. Regarding the tools selected to be reused by SmartCLIDE, Elcipse Theia includes built-in git support, while workflows defined in jBPM[11] are stored internally in git and can be synchronised to an external git repository.

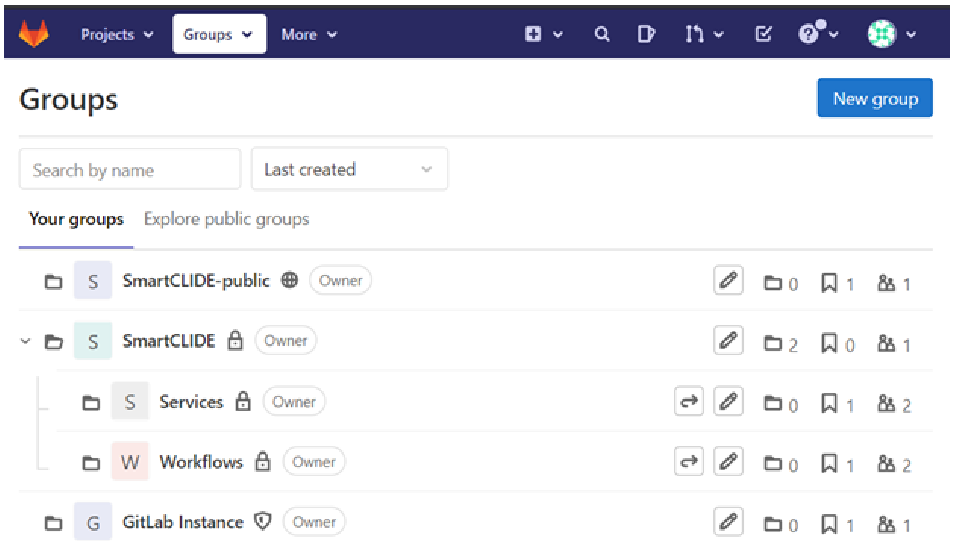
While Git on its own provides excellent support for version control, there are several services that provide additional functionality on top, including GitHub[12], GitLab[13] and Atlassian Bitbucket[14]. Some common additional features are:

- A web-based user interface to support git repository configuration as well as other value-added services
- Support for code review
- Support for CI/CD pipelines

For SmartCLIDE, the consortium has chosen to use GitLab since it is available as a pre-packaged Docker image that can be deployed either on-premises or in the cloud and has a number of other features that make it useful for integration in the SmartCLIDE environment, such as:

- Integration with external security providers for access management, and Keycloak[15] in particular, which is the chosen User Access Management platform
- RESTful and GraphQL APIs that can be used by other components of SmartCLIDE
- Native support for CI/CD
- Hierarchical organisation of Git repositories using "Groups" and "Subgroups", as shown in Figure 34.

[10] https://git-scm.com/
[11] https://www.jbpm.org/
[12] https://github.com/
[13] https://about.gitlab.com/
[14] https://bitbucket.org/
[15] https://www.keycloak.org/



Groups

SmartCLIDE-public	Owner	0	1	1
SmartCLIDE	Owner	2	0	1
Services	Owner	0	1	2
Workflows	Owner	0	1	2
GitLab Instance	Owner	0	1	1

Figure 34: Hierarchical Group Structure in GitLab

As a proposed structure, the top-level SmartCLIDE group contains sub-groups named “Services” and “Workflows”, each of which contain GitLab projects that contain a Git Repository plus other data for handling additional features such as merge requests and CI/CD.

4.1.2 Items to be Stored in Source Code Repository

There are a number of source-code artifacts required to implement a workflow in the SmartCLIDE environment, which should all be version-controlled:

- **Workflow definitions and metadata:** Each workflow definition has its own repository on the GitLab server. Depending on the use case for the environment, the workflow repositories may be grouped according to different criteria. For instance, all workflow repositories may belong to a single group if the SmartCLIDE environment is used for just one project, or they may be further divided into subgroups to allow more refined access control by multiple project teams. Data stored in version control for the workflow includes:
  - **Metadata regarding the workflow** (e.g., name, description, and service dependencies)
  - **Gherkin description of the workflow**
  - **BPMN model of the workflow.** Note that the selected workflow engine, jBPM, contains its own embedded git server. There are two options proposed for the SmartCLIDE IDE:
    - Synchronize the workflow definition between the jBPM embedded git instance and GitLab
    - Use only the embedded Git instance in the jBPM server
- **Service Source Code:** In cases where a service is written from scratch, assembled using a template, or otherwise modified from existing source code, the service should have its own Git repository on the GitLab server. It is proposed to build a library of service definitions, grouped separately from the workflow definitions, since each service has the potential to be re-used in different workflows.

4.1.3 Branching Strategies

The flexibility of Git as a version control system, the ease with which branches can be created and merged, and its use in many different contexts has led to the existence of many branching and release strategies. Suitability of a strategy depends on many factors, including the size of a team, the complexity of the software, whether multiple releases need to be maintained concurrently, etc.

While there is nothing to prevent a team choosing from any of the strategies, in the context of SmartCLIDE the simplest strategy for the circumstances is probably the best. This could be simply using the master (or main) branch directly or, if code review practices are to be used, using short-lived feature branches branched off master and the Merge Request feature of GitLab.

4.1.4 GitLab APIs

Aside from direct access to a git repository via the usual git HTTP(S) or SSH protocol, GitLab also provides APIs that can be used to obtain and update information on the GitLab server. The most useful for the SmartCLIDE are:

- **REST API** [68]
- **GraphQL API** [69]

At the time of writing this document, the RESTful API has more complete functionality and documentation than the GraphQL, although the aim of the GitLab developers is to move over to GraphQL for the additional benefits of being able to specify exactly what is required in response to a request and being able to request multiple data items in a single request.

Some examples of tasks that can be performed over the RESTful API are:

- List the projects that the user has access to, in a specific project group/namespace
- Create a new project in a given group/namespace

These API endpoints are particularly useful for listing and creation of workflows and services by various components of SmartCLIDE, so that it is not necessary for a user to log in to the GitLab UI to be able to use the value-added features of GitLab. An example of creating a new project from the REST API is shown in Figure 35.

```
curl --request POST \
  --url http://gitlab.localhost.private:20080/api/v4/projects \
  --header 'Authorization: Bearer E2Mr_rz3WCsDE_WabYBz' \
  --header 'Content-Type: application/json' \
  --data '{
      "name": "new-workflow-example2",
      "namespace_id": 5
}'
{
  "id": 4,
  "description": null,
  "name": "created-with-api",
  "name_with_namespace": "SmartCLIDE / workflows / created-with-api",
  "path": "created-with-api",
  "path_with_namespace": "smartclide/workflows/created-with-api",
  "created_at": "2021-07-12T15:02:13.929Z",
  "default_branch": null,
  "tag_list": [],
…
```

Figure 35: Example request to GitLab REST API to create a new Project

The token used in the authorization header has been added manually to the user's profile via the GitLab UI. The response output is truncated for brevity.

4.2 Discovery of Services and Resources

The Discovery of Services and Resources backend module is responsible for collecting data on services discovered through the use of crawlers, maintaining an internal registry of services, as well as serving queries/requests for services based on service usage details and service code requests. This component communicates with the DLE to classify services and receive updates.

4.2.1 Design Approach

The Service Discovery component will have five sub-components that will implement the described logic, depicted in light yellow in the Figure 36.

The workflow for a search for services request would be:

1. An internal service registry/database (implemented using the Elasticsearch [70], technology) containing classified services will be in place.
2. Query search input: A service description, along with its tags and category (optional).
3. If category is not in user input, we need to classify user input based in the description using the DLE Service Classification API.
4. The search is established through the Internal Service Index Manager sub-component, first it will search in the service registry, if it does not find any candidate to return, the crawler sub-component will perform a search of the online repositories for the user input and store new repositories if any in the service registry.
5. All services in a category if any, will be ranked based on downloads, followers, stars and other additional information.
6. Those services if any with all available fields in the service registry (URL, name, follower, code…) will be a returned to the user.

Table 24: Discovery of Services & Resources Crawlers

Name	**Crawlers**
Functionality	Collect information from web service listings, service code repositories, service registries and provide data ready to be stored in the registry.
Relevant Use Cases (D1.3)	UC-0002, UC-0004, UC-0028,
Functional Requirements	D1, D2, D3,D4,D5,D6,D7

Table 25: Discovery of Services & Resources Internal Services Index

Name	**Internal Services Index Manager**
Functionality	Allows to store, search and classify both discovered and new created services. This component communicates with the DLE classifier and uses the Elasticsearch API to perform searches, along with its internal SQL service registry.
Relevant Use Cases (D1.3)	UC-0002, UC-0004, UC-0028
Functional Requirements	D3, D4, D6

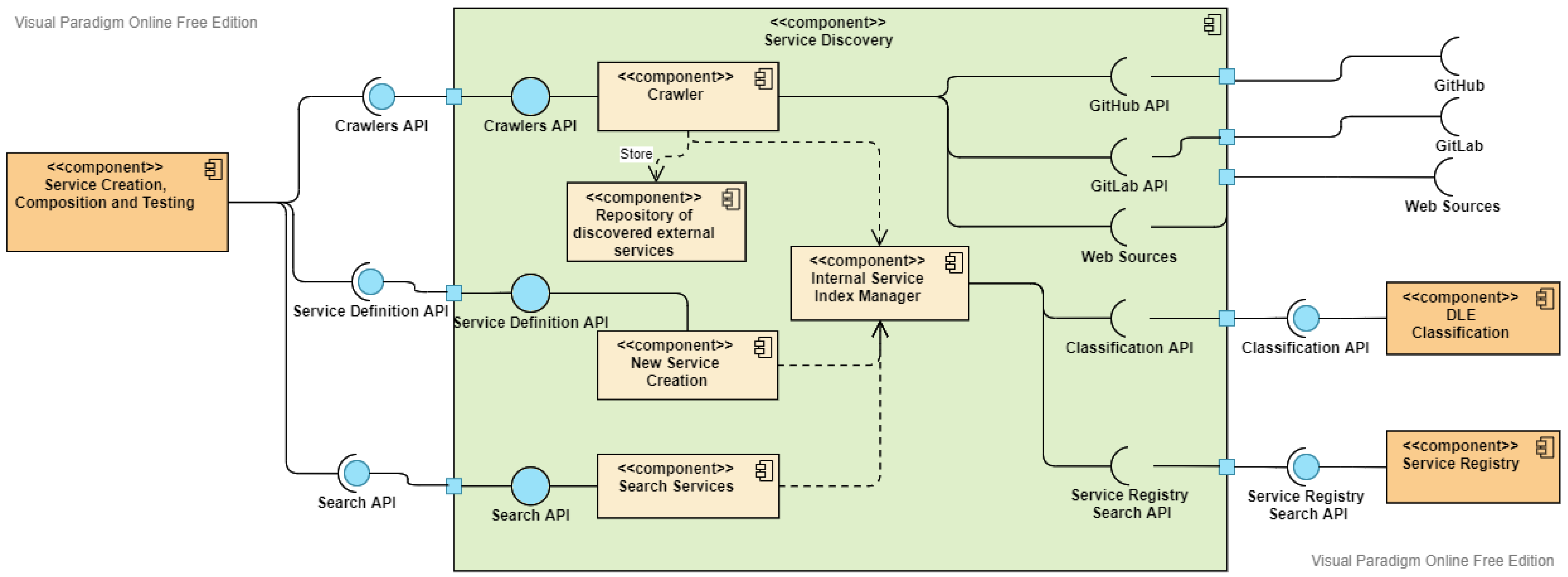


Figure 36: Discovery of Services and Resources Component Diagram

Table 26: Discovery of Services & Resources - Repository of discovered services

Name	Repository of discovered services
Functionality	Store the records discovered by the crawler tool in .csv files while executing the retrieval requests, serving as a backup of the discovered services until they are uploaded to the internal database.
Relevant Use Cases (D1.3)	UC-0002, UC-0004, UC-0028
Functional Requirements	D1, D2, D3, D4

Table 27: Discovery of Services & Resources New Services Creation

Name	New service Creation
Functionality	Allows new services, created from the Service, Composition and Testing component to be stored and classified in the service registry index.
Relevant Use Cases (D1.3)	UC-0002, UC-0004, UC-0028
Functional Requirements	D3, D4, D5

Table 28: Discovery of Services & Resources Search Services

Name	Search Services
Functionality	Using an internal SQL record and the Elasticsearch API, this component will accept search requests that it will delegate to the internal registry and then to Elasticsearch, returning the ranked services to the user based on the search.
Relevant Use Cases (D1.3)	UC-0002, UC-0004, UC-0005, UC-0011, UC-0012, UC-0028
Functional Requirements	D4, D5, D6, D7

4.2.2 Interface Specification

Table 29: Discovery of Services and Resources Interface Specification

No	Interface (/API)	Description	Type	
			Provided	Required
1	**Crawlers API**	The Crawlers API, which is a subcomponent of Discovery of Services and Resources, provides a system of search requests with two parameters, where one at a time is mandatory. • **from_url:**a plain-text URL of a github//gitlab/bitbucket organization or user. • **from_keywords:** one or a series of keywords separated by comma related to what topics of services to search for Returns the number of repositories found added to the index and the delay in search.	*	
2	**Search API**	Although this functionality has been delegated to the Elasticsearch API [71] as it has been decided to use a high-level solution, the component will expose a custom API to cater for different types of searches. By making use of an internal SQL registry and the Elasticsearch API this component will accept search requests which it will delegate to the internal registry and then to Elasticsearch, returning to the user the ranked services. • **service_to_search:** a JSON file containing the minimum data to search: keyword (optional), name and description. The more parameters the better the search will be. See Fig X, for more parameters. Returns a JSON containing the services ranked according to search criteria.	*	
3	**Service Definition API**	Used for storing new services created from the Service, composition and testing component. Accepts a parameter: • **service_to_add:**a JSON file containing the service minimum required fields as described in Figure X Returns a JSON containing the response message from the request to insert that service into the index.	*	
4	**Service Classification API**	This component is used to classify the discovered services, the API is called with the following parameters: • **Service Name** • **Service Description** • **Tags** Gets the returned JSON file containing service class to be stored in the service registry.		*
5	**Service Registry API**	This component uses the Elasticsearch stack, its API will be exposed [71] to communicate the internal service index manager subcomponent with the service registry		*

4.2.3 Crawlers

The main objective of the crawlers is to create an internal database of services and resources for the Service Discovery component. It supervises collecting the information generated from the crawlers that generate lists of services from web service listings, service code repositories and service registries, either using scraping in the first case or through an API in the rest.

The creation of the data frames will be detailed below based on the information that can be obtained from each method. The main workflow is shown in Figure 37.

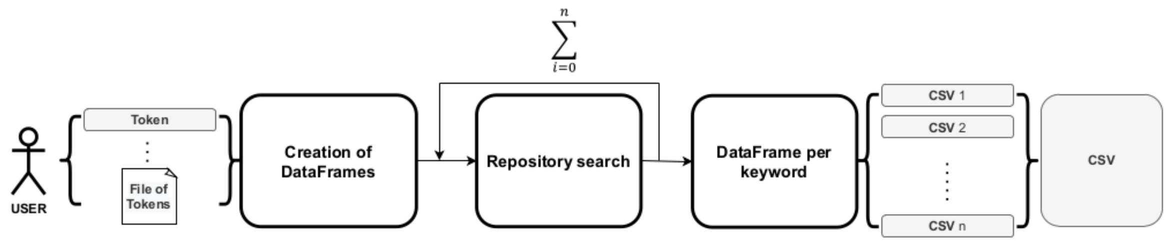


Figure 37: Pipeline of the data extraction process

A user provides a set of tokens as an input, the crawler logic sets up a Dataframe for the search, the search in performed iteratively, for each keyword a CSV file is exported to have a copy of the process. When all the keywords have been collected, the data is merged and exported under the same file. This file is stored in the internal SQL registry then it will be automatically synchronised with the Elastic registry.

It also provides service extraction and several file collection components to meet the needs of the code generation and recommender component, and for other sources, such as web Programmableweb. On a recurring basis, the content of the websites will be crawled to gather new information which will be updated in the Service Register. In this way, the entire content of the websites is periodically downloaded and processed.

4.2.4 Internal Services Index

To manage the internal index, several requirements must be considered given the type of response expected to cover the following needs:

1. **Fast reacting**: a system that allows fast text search, insertions, deletions, and updates is needed to improve the user experience.
2. **Stable and scalable**: a system that is consistent, where backups are frequent, as well as a system that allows the insertion of new data in an efficient way is needed.
3. **Simple but effective**: it is required that the system allows quick search, allowing for a multitude of filters to refine searches while maintaining a simple complexity.

For the internal registration of the services, it has been decided to use Elascticsearch [1] as a search server hosting a dataset of services with an intermediate register detailed in Figure 38 the fields marked as required will be the minimum required information about a service to be classified.

Elascticsearch is an open-source distributed search engine and serves as an analysis database, it was developed in Apache Lucene using Java. It allows to store, search, and analyse a large volume of data in a matter of seconds and to get fast search answers through search indexes.

In contrast to SQL, a relational model, Elascticsearch is based on a search engine system, making it easier to ingest new data by scaling horizontally. It therefore fulfils the need of rapid reaction and scalability, resulting easily optimise it on various servers.

The most important point is that it serves as a REST API for data updates, builds and searches. Allowing highly customized searches [40], for example, if we search for a term, it searches all fields until it finds exactly that term. Additionally, we can do fuzzy searches -similar-, prefixes, ranges of values, regular expressions, data type, wildcard searches with * and match searches - match certain terms and not all- among others. Through the Service Discovery Component this Elascticsearch index will receive Service queries as well as new Services to be saved and respond with services that meet the query. With its API, it meets the need to be simple but effective.

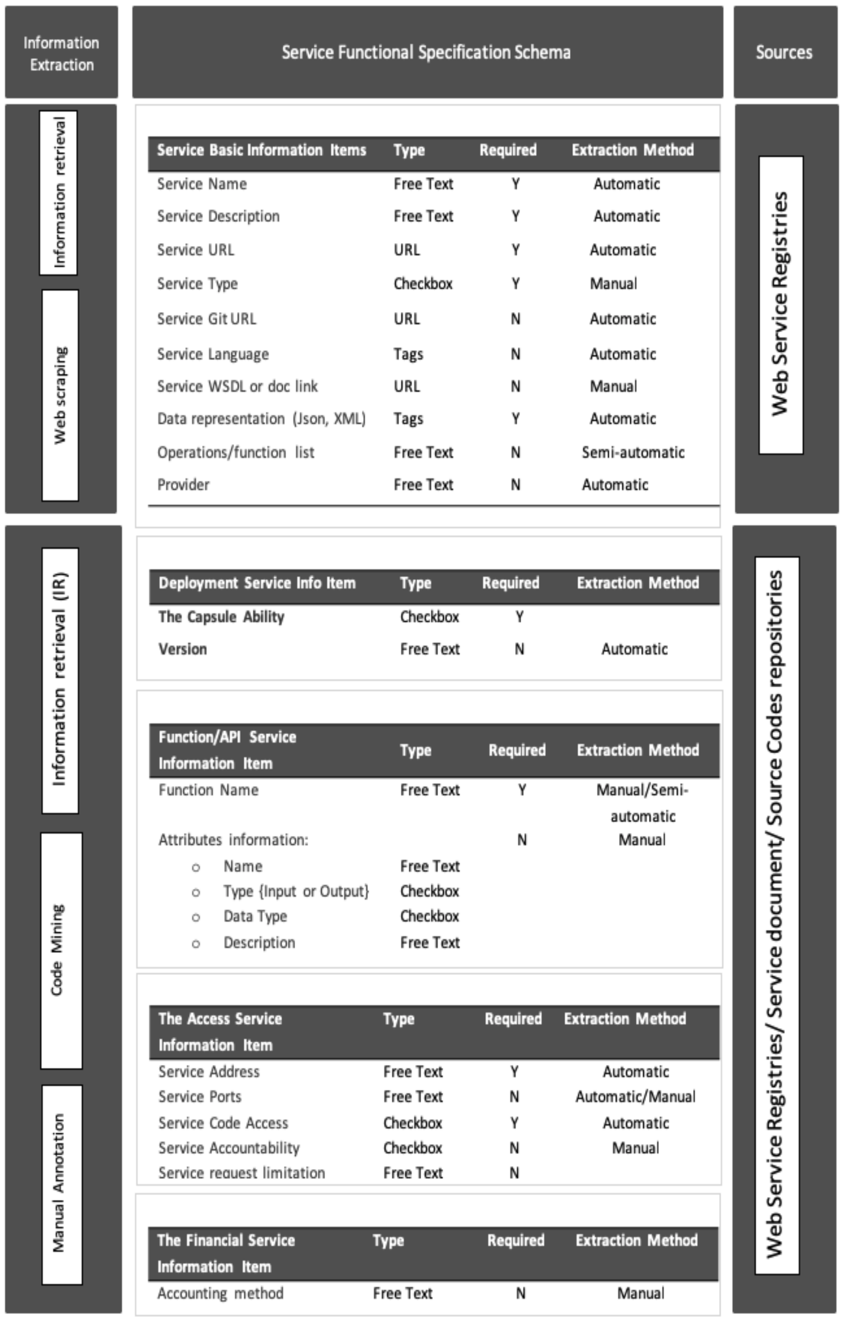
The required attribute in the service specification schema (Figure 63) is essential to perform searches. It should be noted that this component is perfectly extensible and easy to add new attributes to the services. However, it does not support ACID properties (Elasticsearch search engine model constraint). New data will be attached with a version number, it will increase monotonically, but when two write calls come, both will write concurrently and keep only the latest version.

An internal SQL registry will be maintained to serve as a database for the Crawler sub-component, this database will be synchronised with Elasticsearch to provide the types of searches described above. Therefore, the internal registry will have two databases, one to support the crawler (SQL) and one for searches (Elasticsearch).

Its function is mainly to be an intermediary between searches and data insertions into the Elasticsearch registry, when a new service is created its data will be collected by the New Service Creation sub-component then passed to this module, the data is sorted using the DLE and then uploaded to the registry.

On the other hand, it also accepts insertions, when the crawler performs service searches the output data will be passed to this submodule which will again use the DLE to classify the new services and insert them into Elasticsearch. The duplicate control is done in the internal SQL record, when a component is newly discovered and already exists in the SQL record no further action will be taken, if it is new, it will execute the flow as described.

It also accepts search parameters from the Search Services component, implements the search logic described in the Design Approach.


Information Extraction
Service Functional Specification Schema
Sources
Information retrieval
Web scraping
Web Service Registries
Information retrieval (IR)
Code Mining
Manual Annotation
Web Service Registries/ Service document/ Source Codes repositories

Service Basic Information Items	Type	Required	Extraction Method
Service Name	Free Text	Y	Automatic
Service Description	Free Text	Y	Automatic
Service URL	URL	Y	Automatic
Service Type	Checkbox	Y	Manual
Service Git URL	URL	N	Automatic
Service Language	Tags	N	Automatic
Service WSDL or doc link	URL	N	Manual
Data representation (Json, XML)	Tags	Y	Automatic
Operations/function list	Free Text	N	Semi-automatic
Provider	Free Text	N	Automatic

Deployment Service Info Item	Type	Required	Extraction Method
The Capsule Ability	Checkbox	Y	
Version	Free Text	N	Automatic

Function/API Service Information Item	Type	Required	Extraction Method
Function Name	Free Text	Y	Manual/Semi-automatic
Attributes information:		N	Manual
○ Name	Free Text		
○ Type {Input or Output}	Checkbox		
○ Data Type	Checkbox		
○ Description	Free Text		

The Access Service Information Item	Type	Required	Extraction Method
Service Address	Free Text	Y	Automatic
Service Ports	Free Text	N	Automatic/Manual
Service Code Access	Checkbox	Y	Automatic
Service Accountability	Checkbox	N	Manual
Service request limitation	Free Text	N	

The Financial Service Information Item	Type	Required	Extraction Method
Accounting method	Free Text	N	Manual

Figure 38: Required Attributes

4.2.5 New service creation

This subcomponent is limited to collect and check the new services created by the tool. The information about the service will be passed on to the Internal Index Services Manager to be classified and stored in the register.

4.2.6 Search services

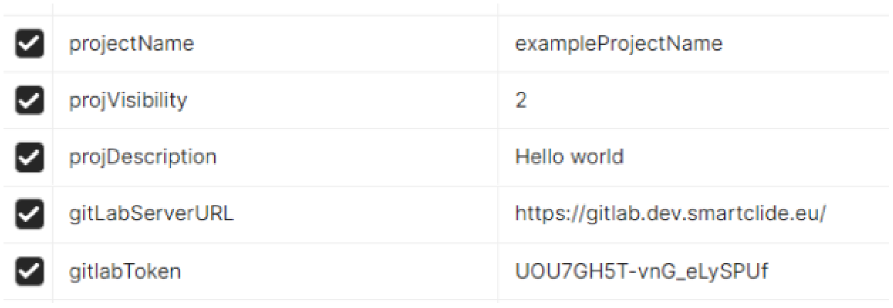
This sub-component is limited to collecting search information from the Service, composition and testing component, creates the query requests to be passed to the Internal Services Index Manager submodule to execute the search in Elasticsearch or to search for new services if there are no records for it, the search logic is delegated to that component

4.3 Service Creation, Composition and Testing

4.3.1 Design Approach

The Service Creation subcomponent will be responsible for handling the creation of a new service. The component will create the required infrastructure for the development process by automatically creating and configuring functions such as version control and continuous integration. The above will be achieved by leveraging the already existing GitLab API. Apart from aiding with the creation process, the component will accompany the user through it by providing useful functionalities through interaction with other components and external tools. The user will have the ability to request functionalities by notifying Deep Learning Engine or Software Security. Furthermore, the user will be able to fetch analysis data from the Reusability Index and TD Principal and Interest subcomponents. Finally, an API will be exposed, that will allow the usage of certain functionalities when called by either the Eclipse Theia extension, JBPM Workbench or any other type of UI. The overall purpose of the component is to aid during the development process and ultimately lead to a better implemented final result.

The functionality provided by the Service Creation backend service, is accessible via a REST API. The service, as of now, offers an endpoint that accepts request containing the appropriate parameters. For testing and presentation purposes, POSTMAN was used to make requests, but also to evaluate their response.



✓	projectName	exampleProjectName
✓	projVisibility	2
✓	projDescription	Hello world
✓	gitLabServerURL	https://gitlab.dev.smartclide.eu/
✓	gitlabToken	UOU7GH5T-vnG_eLySPUf

Figure 39: Service Creation end point

The endpoint (/createStructure) requires four parameters:

- projectName: The name for the new GitLab project.
- projVisibility: The new project's visibility (0:public, 1:internal, 2:private).
- projDescription: A description for the new project.
- gitLabServerURL: The url of the hosted GitLab instance.
- gitlabToken: The access token of the user that will own the project.

The service receives a request and checks it's parameters for null, empty and invalid values. Assuming everything is in order, a new GitLab project is created and the appropriate response is returned, containing a status of 0 and a new project url. In case there is any sort of error, the response contains a status of 1 and a message explaining the error.

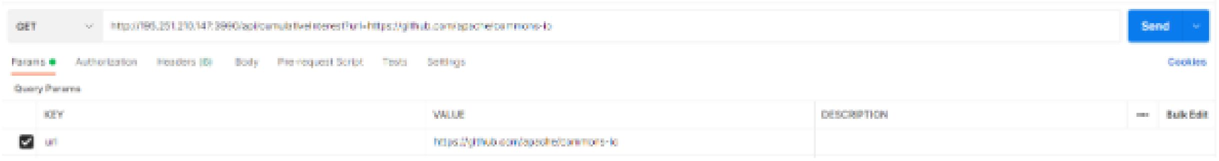
The TD Principal and Interest subcomponent is responsible for calculating the main aspects of the TD concept and it consists of two separate subcomponents. These subcomponents are responsible for the assessment of the two main pillars of TD, namely "Principal" and "Interest".

<u>TD Interest Component</u>

The results of the TD Interest Component are provided via REST API, through multiple endpoints. In this section, we will demonstrate three API requests by providing each time the appropriate query parameters and we will present the exact form of the request's response.

<u>Cumulative Interest</u>

A screenshot of the POSTMAN request is shown in the following Figure. The client basically asks for the project's total TD Interest value (in monetary & time units) in each revision.


```
GET  http://195.251.210.147:3990/api/cumulativeInterest?url=https://github.com/apache/commons-io
```

KEY	VALUE	DESCRIPTION
url	https://github.com/apache/commons-io	

Figure 40: The Cumulative Interest Service Request

As it can be observed, the client must provide only one mandatory query parameters: the repository's URL

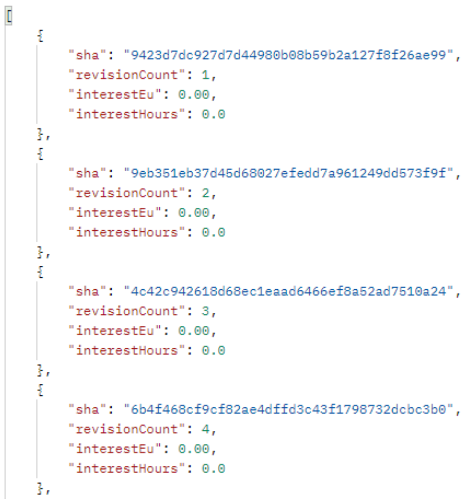
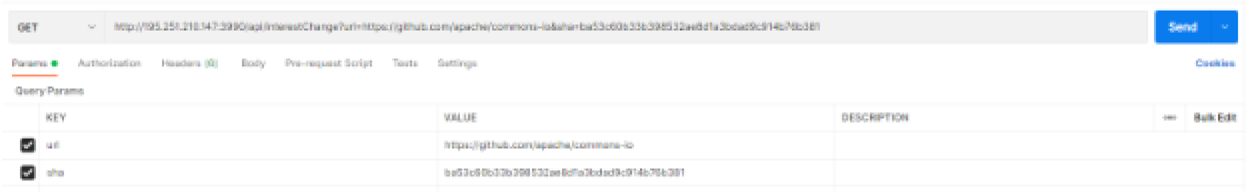

```
[
    {
        "sha": "9423d7dc927d7d44980b08b59b2a127f8f26ae99",
        "revisionCount": 1,
        "interestEu": 0.00,
        "interestHours": 0.0
    },
    {
        "sha": "9eb351eb37d45d68027efedd7a961249dd573f9f",
        "revisionCount": 2,
        "interestEu": 0.00,
        "interestHours": 0.0
    },
    {
        "sha": "4c42c942618d68ec1eaad6466ef8a52ad7510a24",
        "revisionCount": 3,
        "interestEu": 0.00,
        "interestHours": 0.0
    },
    {
        "sha": "6b4f468cf9cf82ae4dffd3c43f1798732dcbc3b0",
        "revisionCount": 4,
        "interestEu": 0.00,
        "interestHours": 0.0
    },
```

Figure 41: Cumulative Interest Service Response

Once the request is sent, the service delivers a JSON file, whose form is shown in the above Figure. The API response consists of a JSON array, within of which there are JSON objects, with 4 attributes: (a) the commit id (SHA), (b) the incrementing revision number (revisionCount), (c) the value of the project's total interest in euros and (d) the value of the project's total interest in terms of time (hours). Interest values are calculated with the reported methodology.

<u>Interest Change</u>

A screenshot of the POSTMAN request is shown in the Figure below. The client asks for the change in a project's total interest compared to the previous revision.


```
GET  http://195.251.210.147:3990/api/interestChange?url=https://github.com/apache/commons-io&sha=ba53c60b33b398532ae8d1a3bdad9c914b76b381
```

KEY	VALUE	DESCRIPTION
url	https://github.com/apache/commons-io	
sha	ba53c60b33b398532ae8d1a3bdad9c914b76b381	

Figure 42: Interest Change Request

In this case, the client must provide two mandatory query parameters: (a) the repository's URL and (b) the commit id (SHA).

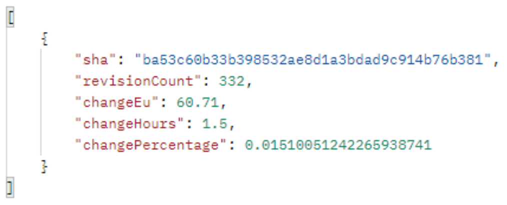
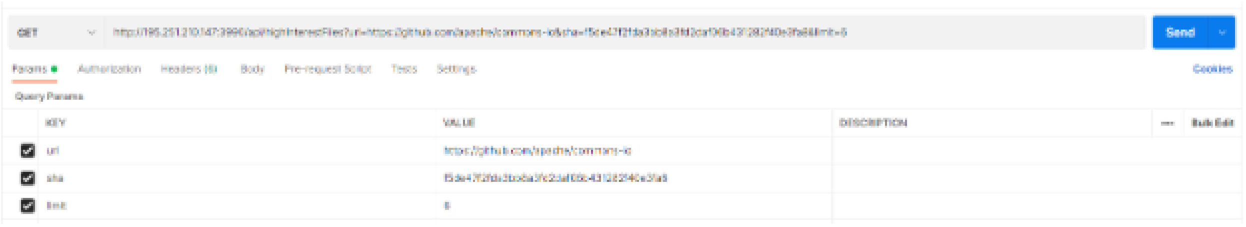

```
[
    {
        "sha": "ba53c60b33b398532ae8d1a3bdad9c914b76b381",
        "revisionCount": 332,
        "changeEu": 60.71,
        "changeHours": 1.5,
        "changePercentage": 0.01510051242265938741
    }
]
```

Figure 43: Interest Change Response

The API response consists of a JSON array, within of which there are multiple JSON objects, with 5 attributes: (a) the commit id (SHA), (b) the incrementing revision number (revisionCount),(c) the change of project's total interest in terms of money (euros), (d) the change of project's total interest in terms of time (hours) and (e) the change percentage of project's total interest (compared to the previous revision).

Files with High Interest

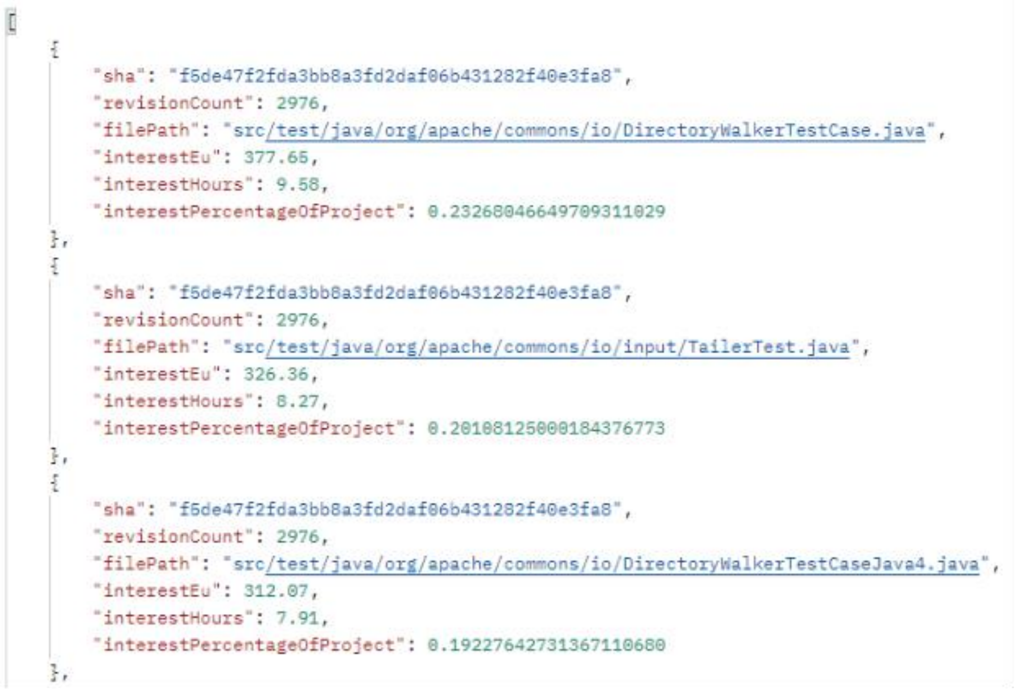
Using this endpoint, the client asks for the files with the highest TD interest value, in a specific revision.


GET http://195.251.210.147:3990/api/highInterestFiles?url=https://github.com/apache/commons-io&sha=f5de47f2fda3bb8a3fd2daf06b431282f40e3fa8&limit=5

	KEY	VALUE	DESCRIPTION
✓	url	https://github.com/apache/commons-io	
✓	sha	f5de47f2fda3bb8a3fd2daf06b431282f40e3fa8	
✓	limit	5	

Figure 44: Files and High Interest Request

As it can be observed, the client must provide two mandatory query parameters and one optional: (a) the repository's URL and (b) the commit id (SHA) and (c) the maximum number (limit) of returned results (optional).



```
[
    {
        "sha": "f5de47f2fda3bb8a3fd2daf06b431282f40e3fa8",
        "revisionCount": 2976,
        "filePath": "src/test/java/org/apache/commons/io/DirectoryWalkerTestCase.java",
        "interestEu": 377.65,
        "interestHours": 9.58,
        "interestPercentageOfProject": 0.23268046649709311029
    },
    {
        "sha": "f5de47f2fda3bb8a3fd2daf06b431282f40e3fa8",
        "revisionCount": 2976,
        "filePath": "src/test/java/org/apache/commons/io/input/TailerTest.java",
        "interestEu": 326.36,
        "interestHours": 8.27,
        "interestPercentageOfProject": 0.20108125000184376773
    },
    {
        "sha": "f5de47f2fda3bb8a3fd2daf06b431282f40e3fa8",
        "revisionCount": 2976,
        "filePath": "src/test/java/org/apache/commons/io/DirectoryWalkerTestCaseJava4.java",
        "interestEu": 312.07,
        "interestHours": 7.91,
        "interestPercentageOfProject": 0.19227642731367110680
    },
```

Figure 45: Files and High Interest Response

Once the request is sent, the service responds with a JSON file that includes JSON array. The arreay consists of multiple JSON objects, with 6 attributes: (a) the commit id (SHA), (b) the incrementing revision number (revisionCount),(c) the file's path, (d) the value of the project's total interest in euros and (e) the value of the project's total interest in terms of time (hours) and (f) the file's interest as a proportion of the project's total interest (percentage).

The TD Principal subcomponent is going to be in charge of the assessment of the time or money needed to bring an individual component or even the whole system, to an optimum structural quality state. Moreover, this component has the intention to provide the developer with ways in which the system can become better in the aforementioned respect. To achieve these goals an external tool named SonarQube is going to be used which quantifies the TD Principal through the number of code inefficiencies according to a set of rules. The connection of this tool with our component is going to be established through SonarQube's API.

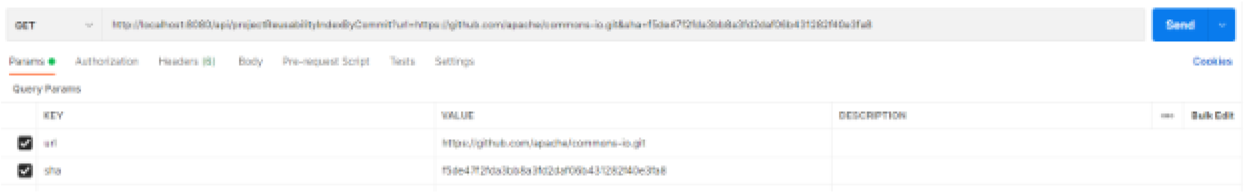
The TD Interest subcomponent is responsible for measuring the interest both of the system as a whole and its software components individually. TD Interest component is able to measure interest both in time unit (minutes, hours etc.) and as a monetary unit (euros or dollars). This component has the intention to provide the developer a clear perspective of the evolutionary and current system's code quality state. Along with the measurements of interest mentioned, the component will produce a set of basic software quality metrics, in order to comprehend the system's size, complexity, cohesion and coupling. This set of metrics will also be used as input by the ***Reusability Index subcomponent.***

Reusability Index Component

The results of the Reusability Index Component are provided via REST API. In this section, we will demonstrate two API requests by providing each time the appropriate query parameters and we will present the exact form of request's response.

Reusability Index by Commit

The client makes a request for the project's reusability index value in a specific revision.


GET http://localhost:8080/api/projectReusabilityIndexByCommit?url=https://github.com/apache/commons-io.git&sha=f5de47f2fda3bb8a3fd2daf06b431282f40e3fa8

Query Params

	KEY	VALUE	DESCRIPTION
✓	url	https://github.com/apache/commons-io.git	
✓	sha	f5de47f2fda3bb8a3fd2daf06b431282f40e3fa8	

Figure 46: Reusability Index by Commit Request

As it can be observed, the client must provide 2 mandatory query parameters: (a) the repository's URL and (b) the commit id (SHA).

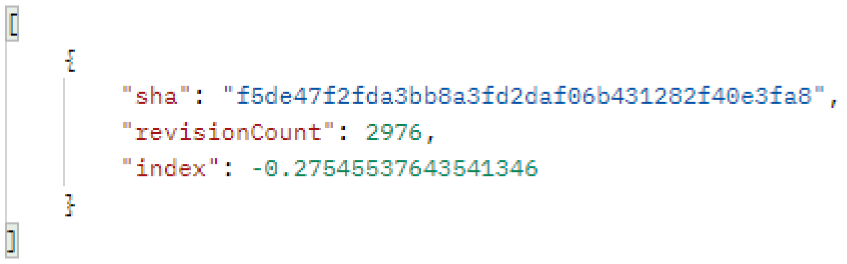
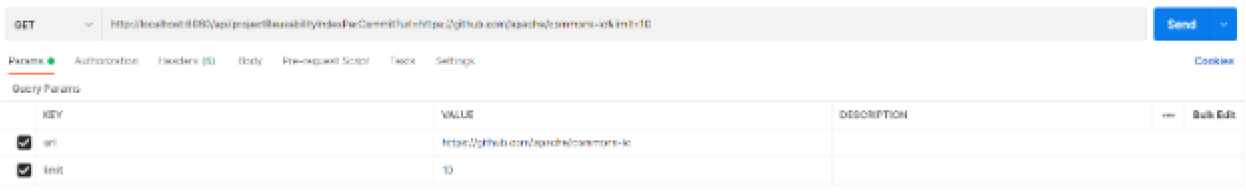

```
[
    {
        "sha": "f5de47f2fda3bb8a3fd2daf06b431282f40e3fa8",
        "revisionCount": 2976,
        "index": -0.2754553764354134б
    }
]
```

Figure 47: Reusability Index by Commit Response

The API response consists of a JSON array, within of which there is a JSON object, with 3 attributes: (a) the commit id (SHA), (b) the incrementing revision number (revisionCount) and (c) the value of the Reusability Index, calculated with the reported methodology.

Reusability Index per Commit

Using this endpoint, the client asks for the project's reusability index value in each revision.

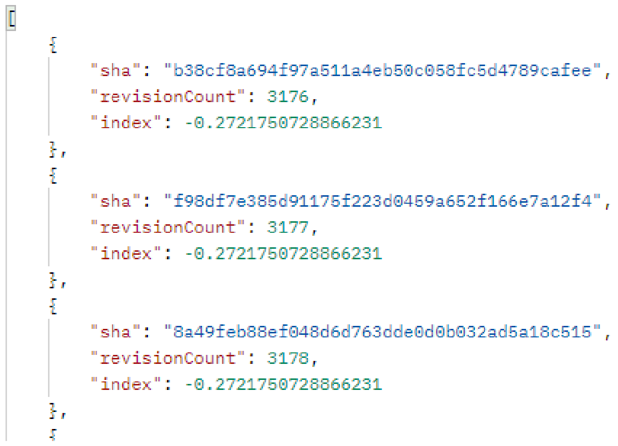

GET http://localhost:8080/api/projectReusabilityIndexPerCommit?url=https://github.com/apache/commons-io&limit=10

Query Params

	KEY	VALUE	DESCRIPTION
✓	url	https://github.com/apache/commons-io	
✓	limit	10	

Figure 48: Reusability Index by Commit Request

As it can be observed, the client must provide one mandatory query parameter and one optional: (a) the repository’s URL (mandatory) and (b) the maximum number (limit) of returned results (optional).



```
[
    {
        "sha": "b38cf8a694f97a511a4eb50c058fc5d4789cafee",
        "revisionCount": 3176,
        "index": -0.2721750728866231
    },
    {
        "sha": "f98df7e385d91175f223d0459a652f166e7a12f4",
        "revisionCount": 3177,
        "index": -0.2721750728866231
    },
    {
        "sha": "8a49feb88ef048d6d763dde0d0b032ad5a18c515",
        "revisionCount": 3178,
        "index": -0.2721750728866231
    },
    {
```

Figure 49: Reusability Index per Commit Response

The API response consists of a JSON array, within of which there are multiple JSON objects, with 3 attributes: (a) the commit id (SHA), (b) the incrementing revision number (revisionCount) and (c) the value of the Reusability Index, calculated with the reported methodology.

Both the ***TD Principal and Interest subcomponents*** will be accessible through RESTful web services, to facilitate the process of integrating and invoking the aforementioned components both from the inside and outside of SmartCLIDE’s ecosystem.

The analysis results of both components will be presented to the developer in an individual panel inside the *Eclipse Theia* instance, along with the computed results of the ***Reusability Index component***, which is closely related to the ***TD Interest subcomponent***, as we already mentioned.

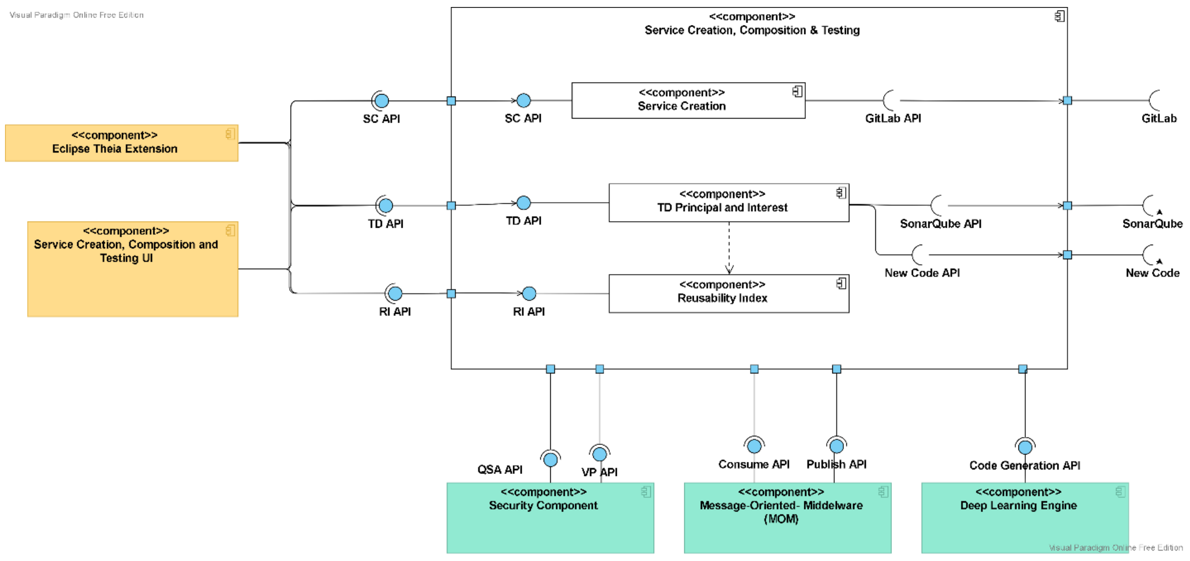


Figure 50: Service Creation, Composition and Testing Component Diagram

Table 30: Service Creation, Composition & Testing Subcomponent

Name	**Service Creation**
Functionality	***The Service Creation subcomponent*** is responsible for creating the required infrastructure for the creation of a new service and aiding the user in it’s implementation

Relevant Use Cases (D1.3)	UC-0001, UC-0003, UC-0014
Functional Requirements	D9

Table 31: TD Principal and Interest Subcomponent

Name	**TD Principal and Interest**
Functionality	***The TD Interest subcomponent*** is responsible for measuring the interest of a system. ***TD Interest*** is able to measure interest both in time unit (minutes, hours etc.) and as a monetary unit (euros or dollars).
Relevant Use Cases (D1.3)	UC-0001
Functional Requirements	D22, D24

Table 32: Reusability Index Subcomponent

Name	**Reusability Index**
Functionality	***Reusability Index component*** is responsible for calculating and providing the reusability index, which describes the degree to which a software component can be reused in different systems.
Relevant Use Cases (D1.3)	UC-0001
Functional Requirements	D24

4.3.2 Interface Specification

Table 33: Service Creation, Composition and Testing Subcomponent

No	**Interface (/API)**	**Description**	**Type**	
			Provided	**Required**
1	**Service Creation (SC) API**	***The Service Creation*** component offers API endpoints, in order to accept requests for creating the structure for the development of a new service. Each request should contain: • **projectName**: a name for the new repository. • **gitLabServerURL**: a URL defining the location of the GitLab server, that is going to host the repository. • **gitlabToken**: the access token of the User that is going to own the repository. • **Additional configuration options** for the GitLab repository	*	

No	Interface (/API)	Description	Type	
			Provided	**Required**
		Service Creation returns a JSON string containing the exit code of the request, along with an appropriate error message (in case of failure) or a GitLab project URL (in case of success).		
2	**TD Principal and Interest (TD) API**	***The TD Interest API*** is a software component that provides a web service, whose functionality is invoked via API and is responsible for providing the interest both of the system as a whole and its software components individually in time & monetary units. Each request should contain: • **repository:** a URL referring to project's repository (e.g., GitHub, GitLab, Bitbucket, etc.) • **gitLabServerURL**: a URL defining the location of the GitLab server, that is going to host the repository. • **gitlabToken**: the access token of the User that owns the repository ***The TD Interest API*** returns a JSON file containing the measurements of interest mentioned above, along with a set of basic software quality metrics, such as size, complexity, cohesion and coupling. The level of the aforementioned metrics' granularity will stop at file level. ***The TD Principal API*** will provide a web service which will be exposed via API calls.Each request will have the following mandatory parameters: • **sonarqube:** the information of the SonarQube instance where the project is analyzed. • **repository**: a URL referring to project's repository. **The TD Principal API** will return a JSON file containing the measurements of Principal.	*	

No	Interface (/API)	Description	Type	
			Provided	Required
3	**Reusability Index (RI) API**	***The Reusability API*** is a software component that provides a web service, whose functionality is invoked via API and is responsible for calculating and providing the reusability index, which describes the degree to which a software component can be reused in other systems. Each request should contain: • **repository:** a URL referring to project's repository (e.g., GitHub, GitLab, Bitbucket, etc.) • **gitLabServerURL**: a URL defining the location of the GitLab server, that is going to host the repository. • **gitlabToken**: the access token of the User tha owns the repository ***The Reusability API*** returns a JSON file containing the measurements of the reusability index. The level of the aforementioned index granularity will stop at file level.	*	
4	**GitLab API**	***GitLab*** is used as the source repository for newly developed services. Other than storing the code of a service, it also provides version control and continuous integration capabilities. It provides an API, through which its functionalities are accessible. We use GitLab's API to create, configure and update the source code repositories.		*
5	**New Code API**	***New Code*** is an external software component that provides a web service, whose functionality is invoked via API. New Code provides historical data about the project's source code additions and modifications, whose presence is mandatory in order the ***TD Interest subcomponent*** to be able to calculate and export its results. Each request has two mandatory parameters: • **repository:** a URL referring to project's repository (e.g., GitHub, GitLab, Bitbucket, etc.) • **commitId:**a string value indicating the distinct SHA value of a project's commit, in which file changes occurred and must be taken into consideration within the analysis		*

No	Interface (/API)	Description	Type	
			Provided	Required
		The New Code API returns a JSON file containing all the meta-data (name of file(s) modified, start line of modification(s), etc.) of file changes that occurred in the commit, whose ID (commitId) was passed as a parameter to the request.		
6	**SonarQube API**	***SonarQube*** is an external tool that analyzes a source code and its able to return it's results through a web API. The results of the API are in a JSON format and for the request we will need: • For the end point **api/issues/search** is required the component key along with the parameter type=CODE_SMELL, in order to retrieve the issues that the developer needs to resolve. For the end point api/measures/component we will need the component along with the metric name, in order to get metrics or even the TD Principal of a file or the whole system.		*
7	**QSA API**	Interface provided by Software Security to request a security assessment by providing either a GitLab URL or a project's files.		*
8	**VP API**	Interface provided by Software Security to request a vulnerability assessment by providing either a GitLab URL or a project's files.		*
9	**Code Generation API**	Interface provided by Deep Learning Engine to request code suggestions.		*
10	**Consume API**	Service Creation, Composition and Testing component will use Consume API to send asynchronous messages to Service Discovery, Security, DLE, and Smart Assistant components.		*
11	**Publish API**	Service Creation, Composition and Testing component will use the Publish API to receive messages from Service Discovery, Security, DLE, and Smart Assistant components.		*

4.4 Run-time Monitoring & Verification

4.4.1 Support for services developed with SmartCLIDE

When services are created in SmartCLIDE the RMV may be employed to generate a runtime monitor for the service. Runtime monitors supplement, at run time, other quality assurance measures such as testing and verification that are employed at development time. Particularly in SmartCLIDE, when automated methods used to generate, or assist in the generation of, code for applications with minimal human intervention, it is possible for there to be semantic "misunderstandings" between component services composed to create the new service, or unexpected interactions of features of the components, resulting in unexpected and undesirable behaviours. Runtime verification may be able to quickly intercept misbehaving services and take a predetermined defensive or remedial action.

The monitors created for a service may be reviewed by the user and disabled or additional monitors specified and generated to be run with the service. In addition, custom monitoring services may be created as SmartCLIDE generated services, to serve other user applications or SmartCLIDE components.

4.4.2 Support to other SmartCLIDE components

Runtime monitoring and Verification (RMV) has explicit support for security and context-sensitivity SmartCLIDE component in addition to synthesized monitoring for created services and the creation of bespoke monitoring services.

4.4.2.1 Support for Security

In support of SmartCLIDE's Security component the RMV system provides security auditing services. The Audit module can generate its own audit logs consisting of audit records generated at the request of any component in the system that possesses a valid audit token. The audit service can add audit records to a file created by the audit service, or it can be directed to an interface to an existing audit or logging service provided by the underlying platform. Auditing is configured through the Audit API, which also provides APIs to generate audit records.

4.4.2.2 Support for Context Handling

For context-sensitivity the RMV system provides context reporting services comprising of the abilities to:

- Collect events from any service that incorporates an RMV monitor
- Create bespoke monitoring applications to monitor and report on conditions of interest to the context system

The Context Handling component interacts with RMV through the Monitoring Request and Configuration API and may also interact to create bespoke monitoring applications through the Monitor Creation API.

4.4.3 Design Approach

4.4.3.1 Overview

The architecture of the Runtime Monitoring & Verification (RMV) system is shown in Figure 2. The RMV will incorporate and build upon several existing technologies as well as implementing new features and integrating them in a novel way to support the runtime quality assurance goal of SmartCLIDE.

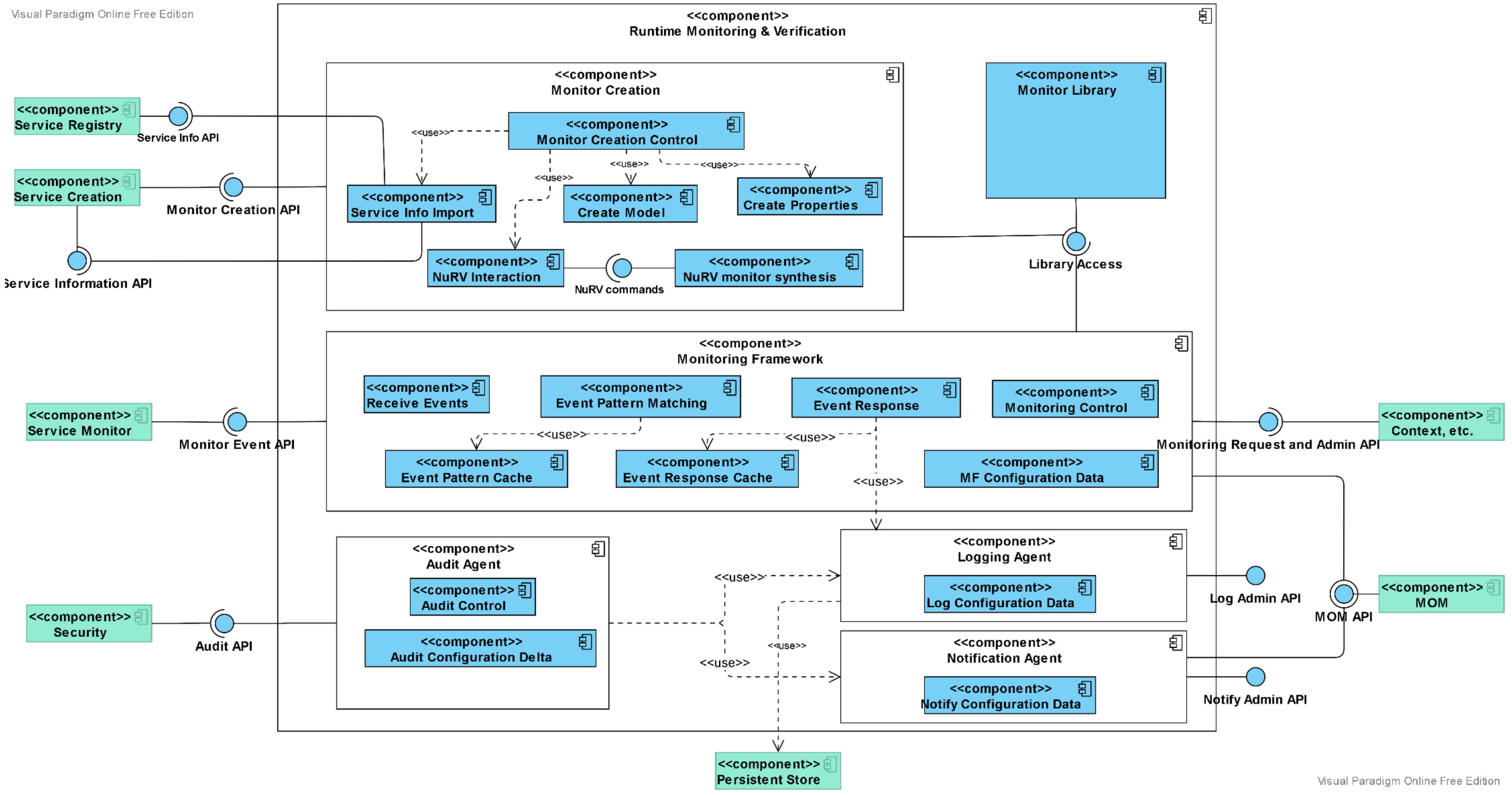

Visual Paradigm Online Free Edition
<<component>> Runtime Monitoring & Verification
<<component>> Monitor Creation
<<component>> Monitor Creation Control
<<component>> Service Info Import
<<component>> Create Model
<<component>> Create Properties
<<component>> NuRV Interaction
<<component>> NuRV monitor synthesis
NuRV commands
<<use>>
<<component>> Monitor Library
Library Access
<<component>> Service Registry
Service Info API
<<component>> Service Creation
Monitor Creation API
Service Information API
<<component>> Monitoring Framework
<<component>> Receive Events
<<component>> Event Pattern Matching
<<component>> Event Response
<<component>> Monitoring Control
<<component>> Event Pattern Cache
<<component>> Event Response Cache
<<component>> MF Configuration Data
<<component>> Service Monitor
Monitor Event API
<<component>> Context, etc.
Monitoring Request and Admin API
<<component>> Audit Agent
<<component>> Audit Control
<<component>> Audit Configuration Delta
<<component>> Logging Agent
<<component>> Log Configuration Data
<<component>> Notification Agent
<<component>> Notify Configuration Data
<<component>> Security
Audit API
Log Admin API
Notify Admin API
<<component>> MOM
MOM API
<<component>> Persistent Store
Visual Paradigm Online Free Edition

Figure 51: Run-time Monitoring and Verification Component Diagram

RMV works hand-in-hand with SmartCLIDE Service Creation and is dependent upon the information used by Service Creation to compose the functionality needed for a specified purpose. RMV creates a monitor to run alongside and share certain variables $\boldsymbol{\mathcal{V}}$ with the created service. These variables provide the "observables" $\mathrm{O} \subseteq \boldsymbol{\mathcal{V}}$ needed for the function of the monitors. While running the monitors report their output to the Monitoring Framework (MF) as a monitor event at each "step" of the monitored service. The MF is initialized with Event-Response Packages (ERP) that associate a sequence of actions with an event pattern. The set of potential actions is customizable and may include both internal actions and actions outside of RMV. The most common actions are to log and/or notify other components of the event match.

A flexible and configurable Audit Agent (AA) receives notice of security-relevant events from RMV and other SmartCLIDE components. The set of such *auditable events* is customizable. The AA is configurable to select a subset of the auditable events, the current *audit selection*, to process through the Logging Agent (LA) to create a persistent record of the event, and/or through the Notification Agent (NA) to send notification of the event to other SmartCLIDE components.

4.4.3.2 Technology

Among the incorporated extant technologies are:

- the overall architectural framework of command processor, server structure, RESTful APIs, and testing from an implementation of the Next Generation Access Control standard [72] by The Open Group [73].
- the Event Processing Point (EPP) from TOG-NGAC an adaptation and extension of the EPP will form the core of the Monitoring Framework component
- the runtime verification extension [74] to the symbolic model checker [75] will form the core of the Monitor Creation component
- a recent audit API addition to TOG-NGAC will be adapted for the Audit Agent.

The RMV will include newly developed components for SmartCLIDE including:

- Create SMV model *K* for the service to be monitored from service specifications used by SmartCLIDE service creation
- Create property specifications in Linear Temporal Logic (LTL) from service specifications
- Logging Agent for flexible and configurable logging
- Notification Agent to provide flexible and configurable notification to SmartCLIDE components
- Monitor Library to hold various facts and rules needed by other components and modules of RMV

The top-level module of the Runtime Monitoring and Verification component is RMV. The names of the subcomponent modules and their constituents are formed from "RMV" plus a suffix for the sub- or sub sub-module. We provide a brief description of the function of each of the main subcomponents in the following tables and a more detailed decomposition, corresponding to Figure 51, of the modules and submodules in Section 4.4.3.

Table 34: RMV subcomponent Monitor Creation (RMV-MC)

Name	**Monitor Creation**
Functionality	Monitor creation utilizes service specifications of the service to be monitored to build a behavioral model and a set of essential properties. These artifacts are built from service specification information obtained from the Service Registry and from Service Creation, and become input to NuRV, an implementation of an advanced monitor synthesis algorithm that operates within a framework for Assumption-Based Runtime Verification (ABRV) with Partial Observability and Resets [76]. NuRV is an extension

	of the model checker nuXmv [74] [75] [16]. This technique allows monitoring even in situations in which not all of the desired variables may be observable. It uses the model of the system being monitored as an assumption on its behavior that enables reasoning about the non-observables. The model serves as an assumption of how the service will behave; it can be detailed, providing a presumably accurate prediction, or it can be coarse and still provide benefit; it can even be empty, except for a declaration of the variables in terms of which the properties are specified. In addition to strictly monitoring for the specified properties, the monitor will provide a distinct output if the observed behavior deviates from the model. Service specifications used by SmartCLIDE Service Creation to create services for the user are also passed to Monitor Creation as coordinated by an interaction between Service Creation and Monitor Creation. For the created service the Monitor Creation creates a monitor to run as part of the service and sharing a chosen set of "Observable" variables with the service. The monitor is a combination of individual monitors for properties of interest. To create a monitor, first the service specifications are used to construct a behavioral model $\boldsymbol{K}$ of the service and the shared variables $\boldsymbol{\mathcal{V}}$, in SMV notation, to be used as an assumption by the monitor. The properties that the monitor is to verify are constructed as LTL formulae $\varphi_1, \varphi_2, ..., \varphi_n(\boldsymbol{\mathcal{V}})$. The resulting $\boldsymbol{K}$ and $\varphi_1, \varphi_2, ..., \varphi_n$ are submitted to NuRV/nuXmv for the synthesis of the monitors $\mathfrak{M}_i$. Internally, RMV-MC uses NuRV (with nuXmv) for monitor synthesis. MC provides the resulting monitors to Service Creation and Deployment to be run with their associated service.
Relevant Use Cases (D1.3)	UC-0025
Functional Requirements	Using patterns and tactics stored in the Monitor Library (RMV-ML), the Create Model and Create Properties modules use SmartCLIDE service specifications to prepare input to NuRV and provides the resulting monitors to SmartCLIDE Service Deployment. RMV-MC exposes a Monitor Creation API (MCAPI) outside of the RMV subsystem. This API is a subset of the MC control functions used by RMV-MF to control and coordinate the operation of MC. The internal API includes: • load_service_spec(service_spec) • service_spec2smv(service_spec_id, SMV_model) • service_spec2ltl(service_spec_id, LTL_properties) • service_spec2nurv(service_spec_id, NuRVcmds) • create_monitor(model, properties) • graph_monitor(model, properties) • export_monitor(monitor_id) External MCAPI exposes functions to control the building of an SMV model K, selection of the shared variables $\boldsymbol{\mathcal{V}}$, the specification of LTL properties $\varphi_1, \varphi_2, ..., \varphi_n(\boldsymbol{\mathcal{V}})$, and the creation of monitors $\mathfrak{M}_i$. Certain NuRV and nuXmv functions, which are also used internally within RMV-MC are also exposed in the MCAPI, including: • add_property • show_property • build_monitor • generate_monitor

[16] nuXmv work carried out within EU Framework Programme 7, D-MILS Project; ABRV and NuRV work received funding from EU Horizon 2020, CITADEL Project.

	These may be used by a UI that allows a user to examine, enable/disable, and add properties to be monitored (D60).

Table 35: RMV subcomponent Monitor Library (RMV-ML)

Name	Monitor Library
Functionality	The RMV-ML is a database of various facts and rules used by other modules of the RMV system. Information contained in the ML includes: • SMV specification patterns • LTL property patterns
Relevant Use Cases (D1.3)	UC-0025
Functional Requirements	The Monitor Library provides functions to retrieve information contained in the library and to update information in the library. These functions are exposed through an RMV internal API (D60, D61).

Table 36: RMV subcomponent Monitoring Framework (RMV-MF)

Name	Monitoring Framework
Functionality	RMV-MF is the hub and control system of RMV. It contains the Event Processing Point (EPP) which receives events and current property verdicts from the monitors running with their associated services. Received events are processed according to the Monitoring Framework Configuration Data which includes Event-Response Packages (ERP) that define event patterns and associated responses. Received events are checked against cached event patterns. When a pattern match occurs the associated response from the event response cache is executed by the Event Response Execution function.
Relevant Use Cases (D1.3)	UC-0025
Functional Requirements	The RMV-MF provides a Monitor Event API (MEAPI) that is used by the runtime monitors to report events and monitor verdicts. Users of the AAAPI must authenticate with an EPP token. The RMV-MF also provides a Monitoring Request and Administration API (MRAAPI) that is used by other SmartCLIDE subsystems and components to configure and to request monitoring services (D62, D63).

Table 37: RMV subcomponent Audit Agent (RMV-AA)

Name	Audit Agent
Functionality	Security auditing services comprising the abilities to: • Define a set of auditable events • Select a subset of the auditable events to be collected in the audit log

	• Define the format of the audit log record • Generate an audit log record in response to a reported event • Store audit records in a persistent audit log file through the Logging Agent • Manage the audit service and audit log files • Generate audit alarms to be delivered through the Notification Agent
Relevant Use Cases (D1.3)	UC-0025
Functional Requirements	RMV-AA provides an API (AAAPI) visible outside of the RMV to be used to administer operations of the AA and to report audit events. Users of the AAAPI must authenticate with an audit token. Auditing is configured through the Audit API, which also provides APIs to provide the abilities listed above. The APIs provide that ability to control audit functions, such as establishing the set of auditable events, and the format of audit records; select the set auditable events to subsequently audit; manipulate audit log files; and generate audit records (D60, D61).

Table 38: RMV subcomponent Logging Agent (RMV-LA)

Name	Logging Agent
Functionality	Filter and store log messages in a persistent log storage according to the Logging Configuration Data
Relevant Use Cases (D1.3)	UC-0025
Functional Requirements	The RMV-LA provides an API (LAAPI) visible outside of the RMV to be used to administer the LA's configuration data, which controls LA operation. RMV-LA may be called by RMV-MF as part of an event response, or by RMV-AA to commit log entries to the Persistent Log Storage. The logging is done according to the Logging Configuration Data which may be modified by calls to the Log/Notify Administration API. According to the configuration, part of the handling of a logging operation can be to forward the logged event to the Notification Agent for direct notification to SmartCLIDE components (D60, D61).

Table 39: RMV subcomponent Notification Agent (RMV-NA)

Name	Notification Agent
Functionality	Send direct notifications to SmartCLIDE components according to the Notification Configuration Data
Relevant Use Cases (D1.3)	UC-0025

Functional Requirements	The RMV-NA provides an API (NAAPI) visible outside of the RMV to be used to administer the NA's configuration data, which controls NA operation. RMV-NA may be called by RMV-LA as part of a logging operation, or by RMV-AA to raise audit alarms to subscribed SmartCLIDE components. The notification is done directly over a committed link to another SmartCLIDE component or over the MoM according to the Logging Configuration Data which is modified by calls to the Log/Notify Administration API. According to the Notification Configuration Data the manner of the notification is determined (D61).

The Logging Agent is concerned with creating persistent stores of log entries, while the Notification Agent is concerned with communicating directly to SmartCLIDE components the occurrence of events of interest to those components (such events may also be logged and/or entered in an audit trail).

4.4.4 Interface Specification

Table 40: Run-time Monitoring and Verification Subcomponent Interface

No	Interface (/API)	Description	Type	
			Provided	Required
1	ServiceInfo(Creation)/ getServiceSpec	Interface provided by Service Creation to obtain information about a new service to be created		*
2	ServiceInfo(Registry)/ getServiceInfo	Interface provided by Service Registry to obtain information about an existing service		*
3	NuRV Commands (accessed via the NuRV Interaction module)	Interface to use NuRV functions, including: ▪ *NuRV/ build_monitor* Build the symbolic monitor for a given LTL property ▪ *NuRV/ generate_monitor* Generate standalone monitor ▪ *NuRV/ heartbeat* Verify one input state in the symbolic monitor ▪ *NuRV/ read_model* Reads SMV model file ▪ *NuRV/ flatten_hierarchy* Flattens hierarchy of modules ▪ *NuRV/ encode_variables* Builds the BDD variables necessary to compile the model into a BDD ▪ *NuRV/ build_flat_model* Compiles flattened hierarchy into a Scalar FSM ▪ *NuRV/ build_model* Compiles the flattened hierarchy into a BDD ▪ *NuRV/ add_property* Adds an LTL property to the list of properties		*

		▪ *NuRV/ read_trace* Loads a previously saved trace		
4	Monitor Creation API	Monitor creation functions, including: • *MonitorCreation/ createMonitor* Monitor Creation invoked by Service Creation or SmartCLIDE orchestration. • *MonitorCreation/graphMonitor* Provide a visual graph of the monitor for display by the UI	*	
5	Monitor Event API	Use monitoring functions, including: • *MonitorEvent/ reportEvent* Used by monitors (and potentially other SmartCLIDE components) to incrementally submit events for processing	*	
6	Monitoring Request and Admin API	Control Monitoring functions, including: • *MonitoringRequest/ listAvailableMonitors* Used by SmartCLIDE components to discover available monitoring data channels • *MonitoringRequest/ registerForNotification* Register for notification of monitoring data • *MonitoringRequest/ loadiERP* Load immediate Event-Response Package (E-R package expressed in Event-Response Language) without activating it • *MonitoringRequest /unloadERP* Delete and ERP. Deactivate the ERP first if the ERP is currently active • *MonitoringRequest/ activateERP* Activate a loaded ERP—begin looking for event patterns specified in the ERP • *MonitoringRequest/ deactivateERP* Deactivate an ERP but do not unload it. It can be re-activated without reloading • *MonitoringRequest/ currentERP* Get name(s) of currently active ER package(s)	*	
7	Log Admin API	Control Logging functions, including: • *LogAdmin/ logControl* Name and location of the log file(s); Open a new log file	*	
8	Notify Admin API	Control Notification functions, including: • *NotifyAdmin/ notifyControl*	*	

		Change Notify Configuration Data to add or remove notifications and to change notification parameters		
9	Audit API	Control auditing functions, including: • *Audit/ auditControl* Set audit record format; Get/Set list of all auditable events; Choose whether audit records are stored in a separate audit log file or in the regular log • *Audit/ auditSelect* Select among auditable events to log; The current audit selection may have the provided selection added to it or it may be reset to the provided selection • *Audit/ auditLogfile* Manipulate audit log files; create a new audit log file; safely close previous file • *Audit/ auditGen* Generate and Audit Record with the specified audit event identifier and audit data according to the format specified in the Audit Configuration Data	*	
10	MOM API	MOM provides asynchronous communication by the Monitoring Framework and the Notification Agent with other SmartCLIDE components. The Monitoring Framework will publish lists of subscribe-able monitor information and monitoring results to subscribing SmartCLIDE components, will accept subscriptions, and provide notifications to subscribers		*

4.4.5 RMV Component, Modules and Submodules

Runtime Monitoring & Verification is considered a SmartCLIDE component and is designated “RMV.” The subcomponents that make it up are designated as modules “RMV-x” and their submodules are designated “RMV-x-y” in documentation. The source code modules are named “rmv” for the top-level component, “rmv_x” for modules, and “rmv_x_y” for submodules, where “x” and “y” are replaced by the module and submodule labels. Following are the currently defined modules:

- rmv_mc - monitor creation
  - rmv_mc_cm - create model
  - rmv_mc_cps - create property specs
  - rmv_mc_nui - interface to NuRV/nuXmv/NuSMV
  - rmv_mc_iss - import service specification
  - rmv_mcapi - monitor creation API
- rmv_mf - monitoring framework
  - rmv_mf_ctl - monitoring framework control module
  - rmv_mf_epp - event processing point
  - eppapi - event processing point API
  - rmv_mrapi - monitoring request API

- rmv_ml - monitor library
    - rmv_ml_mt - model templates
    - rmv_ml_pst - property specification templates
- rmv_aa - auditing agent
    - audit - generic audit manager
    - auditapi - auditing API
- rmv_la - logging agent
    - rmv_lnapi - logging and notification API
- rmv_na - notification agent
- cme_sim - simulation of the context system
- exec_sim - simulation of service/monitor execution
- miscellaneous and utility modules
    - command - generic command interpreter
    - command_rmv - definiton of commands specific to rmv tool
    - erl - Event-Response Language of epp
    - jsonresp - JSON responses for Web APIs
    - param - global parameters
    - procs - stored command procedures
    - sessions - dynamic session information
    - test - self test framework
    - ui - simple console I/O utilities

Currently the entire RMV, including its modules and submodules, will run as a single process with multiple threads. However, external tools, such as NuRV and graphViz are executed as separate processes.

The following subsections describe the key source code modules and submodules.

4.4.5.1 Runtime Monitoring & Verification (RMV) Module

This is the top-level module of the Runtime Monitoring & Verification component. It contains all of the RMV modules including one that makes the various RMV functions available as a server with Web interfaces.

In the assumption-based runtime verification (ABRV) monitoring paradigm the model of the system is treated as an *assumption* on the behaviour the system being monitored. It can serve as predictive future behaviour of the system being monitored but it is acknowledged that the actual behaviour of the system may differ from that predicted by the model. This divergence between the model and the implementation could happen in several ways: the code or the model may be “incorrect,” which could be the result of subtle semantic differences such as from a misunderstanding of the semantics of the constructs of the implementation language and the model specification language, from a difference in the level of abstraction of the expressions, from incorrect translation of the model to the code (or vice versa depending on the actual sequence of steps of development and validation), from a “bug” in tools such as a programming language compiler or in the specification language interpreter, or unaccounted for dependencies or influences from outside of the service.

4.4.5.2 Monitor Creation (RMV-MC) Module

This module works with Service Creation and Service Discovery to create monitors for services. It performs the following functions:

- receives requests from SmartCLIDE service creation to construct a monitor for a service
- retrieves from service creation and other SmartCLIDE sources information about the service and packages it as a service specification from which a monitor can be synthesized
- calls upon MC submodules to generate and compile the information needed to use the ABRV method of monitor synthesis embodied in the NuRV tool
- invokes NuRV to synthesize a monitor, or, depending on an option, to determine that a monitor can be synthesized

- store information about the constructed monitor in the monitoring library
- notify the requester of the identifier of the generated monitor
- provide a function whereby the deployment service can use the monitor identifier to retrieve the portions of the monitor that must run with the monitored service.

4.4.5.2.1 Create Model (RMV-MC-CM) Module

This module takes a service specification as input and returns a model of the service. The format of the created structure is defined by the monitor library module. The substance of the model is a transition system expressed in the SMV language. Information about the service to be monitored is to be obtained from other SmartCLIDE components and placed into a service specification structure by the RMV-MC-ISS module.

The method of the module uses the service information to drive the application of a grammar for a fragment of the SMV language as a generator for sentences corresponding to the service specification.

4.4.5.2.2 Create Property Specification (RMV-MC-CPS) Module

This module takes a service specification as input and returns a list of properties of the service to be monitored. The format of the created properties is defined by the monitor library module. The substance of a property is an expression in linear temporal logic (LTL). The monitor library contains a sub-library of LTL patterns that can be used to express common properties or property fragments and composed to create encoded properties of the service returned by this module. The properties, gleaned from information about the service, may relate to correctness, constraints, exception conditions, timing, or anything consistent with the available observables and the collection method.

4.4.5.2.3 Import Service Specification (RMV-MC-ISS) Module

This module is responsible for importing information about a service and building a service specification in the form utilized by the other modules of Monitor Creation. The information may arrive as parameters of an external request, i.e., by service creation, to the monitor creation API, or by this module directly consulting other SmartCLIDE components for information about the service or other general information common to all created services. The information to be collected must be sufficient to construct the model and the properties of the service to be monitored. In some cases, not all of the information may be used.

Necessary information includes the variables, or a subset thereof, that define the state of the service, definition of functions or operations that alter the values of the state variables, identification of the execution model, and the conditions and/or timing of the steps for synchronization of variable value capture. The nature and format of the information available from other SmartCLIDE components is a product of their respective design efforts and is being reviewed as part of this design effort. Negotiations between the design efforts will be undertaken as needed to obtain sufficient information for monitor synthesis. The information gathered will be put into a defined *service specification* structure for consumption by the RMV-MC-CM and RMV-MC-CPS modules.

4.4.5.2.4 NuRV Interface (RMV-MC-NUI) Module

This module provides an interface to the NuRV runtime verification tool. APIs are provided to create an instance of NuRV and to establish a communication session with it. The NuRV Interface is able to direct NuRV to import the model and properties for a service to be monitored and to generate online or offline monitors for that service. When a monitored service is run the monitoring framework establishes a session with NuRV and with the monitor sensor deployed with the service to step the monitor using heartbeat messages that communicate to the monitor the value of state predicates.

4.4.5.3 Monitoring Framework (RMV-MF) Module

This module, the top-level module of monitoring framework, works with Monitor Creation and the Monitor Library modules to collect events from monitored services, to respond to patterns of such events with associated with actions as defined by *event response packages* expressed in the Event-Response Language (ERL), and to process requests from clients within SmartCLIDE to receive monitoring results. The Monitoring Framework performs the following functions:

- accept events from each active monitor and process according to the steps defined for that monitor
- store and interpret event response packages to execute defined responses for matched event patterns
- forward requests for logging or notification generated by event responses
- accept requests from client processes to receive information related to ongoing monitoring activities
- process the loading/unloading and activation/deactivation of event response packages according to the ongoing monitoring activities.

4.4.5.3.1 RMV Control (RMV-MF-CTL) Module

Overall coordination, as necessary, of the execution of the functions of the Monitoring Framework listed above is provided by this module. The module may simply comprise code in the top-level RMV-MF module.

4.4.5.3.2 Event Processing Point (RMV-MF-EPP) Module

The EPP provides programmed responses to event patterns. The event patterns and responses are encoded in the Event-Response Language (ERL). The response is implemented as a sequence of primitive action commands that are independently and modularly definable. The EPP through its API can be instructed to load Event-Response Packages (ERP) coded in ERL, and to activate/deactivate loaded ERPs. A distinct API is provided for reporting events to the EPP which are then matched against event patterns in an ERP.

4.4.5.4 Monitor Library (RMV-ML) Module

The Monitor Library provides storage and access to various information needed within the RMV subsystem. In particular, templates for creation of the service model in SMV, templates for the creation of property specifications in LTL, and composition rules are used by the logic of the RMV-MC-CM and RMV-MC-CPS modules to create the model and properties of a service.

4.4.5.4.1 SMV Model Templates (RMV-ML-MT) Module

This is a library of patterns for the construction of a model of the service to be monitored, expressed in SMV. The patterns are a fragment of SMV expressed as attributed grammar rules in the Definite Clause Translation Grammar (DCTG) logic programming formalism. The grammar rules are used as a generator, rather than as a recognizer, that is guided through the grammar's semantic attributes in the construction of an SMV model using the available service specification information.

4.4.5.4.2 Property Specification Templates (RMV-ML-PST) Module

This is a library of parameterized Linear Temporal Logic (LTL) patterns taken from Dwyer's LTL patterns [77] that is usable by the property composition mechanisms in Create Property Specification module, and ultimately exported in the LTL syntax defined by NuSMV.

4.4.5.5 Audit Agent (RMV-AA) Module

The Audit Agent encompasses an API module and an audit function module. The audit API provides interfaces to administer the auditing functions. The audit functions include:

- changing the set of auditable events
- selecting the set of events to be audited
- managing the audit log storage
- managing audit alarms.

4.4.5.6 Logging Agent (RMV-LA) Module

The Logging Agent is responsible for creating a persistent storage log of events that are chosen to be so logged. It can create a single integrated log or several distinct logs. Logging is used by the Audit Agent for persistent audit log storage, by the Event Processing Point for logging event responses, and it may be used by any other SmartCLIDE subsystem for centralized logging.

4.4.5.7 Notification Agent (RMV-NA) Module

The Notification Agent is able to communicate to other SmartCLIDE components either through synchronous direct point-to-point connections or asynchronously through the MoM. Notification Agent actions are performed according to configuration information maintained within the module by notification administration functions.

4.5 Security

4.5.1 Vulnerability Assessment

Software security is a critical consideration for software development companies who want to provide safe and dependable software to their clients [78]. Modern software applications are typically accessible through the internet and handle sensitive data. As a result, they are continually vulnerable to harmful assaults. Exploiting a single vulnerability can have far-reaching repercussions for both the end user (e.g., information leakage) and the organization that owns the affected software (e.g., financial losses and reputation damages) [79]. As a result, the software industry has shifted its focus towards creating proactive approaches that may give developers with suggestive information about the security quality of their program by detecting susceptible hotspots in the source code.

The Vulnerability Prediction (VP) mechanism is one such system that enables the prediction and mitigation of software vulnerabilities early in the development cycle. By assigning limited test resources to potentially risky items, VP models (VPM) can be utilized to prioritize testing and inspection efforts. Several VPMs have been developed throughout the years that use a variety of software elements as inputs, such as software metrics, static analysis warnings, and a text mining approach known as bag of words (BoW) [78], [80]. Although these models have shown encouraging outcomes, there is still room for improvement. Static analysis warnings contain a high number of false positives in addition to severe alarms. The BoW technique appears to produce better results than static analysis alerts and the usage of software metrics; however it is overly reliant on the software project used for model training. As a result, current research has switched its attention to more complex approaches for detecting patterns in source code that signal the presence of a vulnerability. They concentrate on collecting information from a specific software application's raw source code or from abstract representations of its source code, such as their Abstract Syntax Tree.

Using the raw text of the source code in the form of sequences of instructions, this work creates deep-learning (DL) models capable of predicting whether a software component is susceptible or not, employing approaches from the fields of natural language processing (NLP) and text classification. We used approaches from the NLP discipline for this aim. The source code is seen as text, and the vulnerability assessment work, like sentiment analysis, is regarded as a text classification problem. So, using NLP techniques such as Bidirectional Encoder Representations from Transformers (BERT) [81], data pre-processing and transformation to sequences, and training DL models (e.g., recurrent neural networks) suitable for analysing sequential data, we detect potentially vulnerable components using a binary classifier trained primarily on text token sequences from the source code. Furthermore, software measurements acquired by static code analysers, in conjunction with text mining approaches, might be utilized to improve the prediction performance of the models.

4.5.1.1 Text Mining

The process of tokenizing a text, such as the source code, is known as text mining. The simplest text mining approach is BoW [82]. In BoW, the code is divided into text tokens, each of which is accompanied by the number of times it appears in the code. So, each word corresponds to a feature, and the frequency of that feature in a component adds up to the value of that feature for that component. Aside from BoW, text mining employs natural language processing techniques such as word2vec[17] pre-trained embedding vectors to extract semantic information from tokens.

4.5.1.2 Datasets

As part of the current effort, we created two distinct VPMs, one for each of two commonly used programming languages, C/C++ and Java. We used a vulnerability dataset generated from two National Institute of Standards and Technology (NIST) data sources in the case of C/C++: the National Vulnerability Database (NVD) and the Software Assurance Reference Dataset (SARD) . This dataset contains 7651 class files, 3438 of which are classified as susceptible and the remaining 4213 as clean. In the instance of the Java model, a dataset given by OWASP is used to train and assess the model. We gathered source code files written in both Java and C++ programming languages and used a variety of pre-processing approaches to convert the datasets into a series of words (i.e.tokens). All comments, as well as the header/import instructions that specify the usage of certain libraries in the class, were deleted from the dataset. The numeric values were then changed with a unique identifier "numId$," while the text values and characters were replaced with another unique identifier "stride$." All blank lines are also deleted before the text is converted into a list of code tokens (such as new, char, strlen, and so on) in the order they occur in the source code.

Because ML models understand numerical values, these generated tokens are replaced by an integer (integer encoding4) after data purification. These numbers are then converted into vectors known as embedding vectors. These vectors are numerical representations of text tokens and have a specified size. Embeddings are essential because they can better represent a token than basic integer values.

4.5.1.3 Performance Metrics

Several performance metrics are available in the literature and are widely used to assess the prediction performance of ML models. The number of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) produced by the models is generally used to compute these performance metrics. In the case of vulnerability assessment, we lay a specific focus, as in earlier studies, on the Recall (R) of the produced models, because the greater the Recall of the model, the more real vulnerabilities it predicts. Aside from the capacity of the created models to identify the great majority of potentially vulnerable files present in a software product, the volume of the produced FPs (i.e., neutral files identified as vulnerable by the models) is crucial to examine because it is known to impact the models' practicability. Due to the high number of FPs, developers must check a non-trivial number of neutral files in order to discover a vulnerable file. As a result, the number of FP is related to the amount of human effort required by developers to detect files containing vulnerabilities. The lower the number of FPs, the better the accuracy of the model. As a result, we must consider both recall and precision. This finding emphasizes the significance of the f1-score, which represents the balance of precision and recall. However, because it is more essential in VP to detect susceptible (i.e., vulnerable) files at the price of creating FPs (but not too many), we used f2-score as our assessment metric to adjust our models and evaluate them on testing datasets. The f2-score is a weighted average of precision and recall, with recall being more important than precision. It is equivalent to:

$$F_2 = \frac{\left((1 + 2^2) * Precision * Recall\right)}{(2^2 * Precision + Recall)}$$

[17] https://www.tensorflow.org/tutorials/text/word2vec

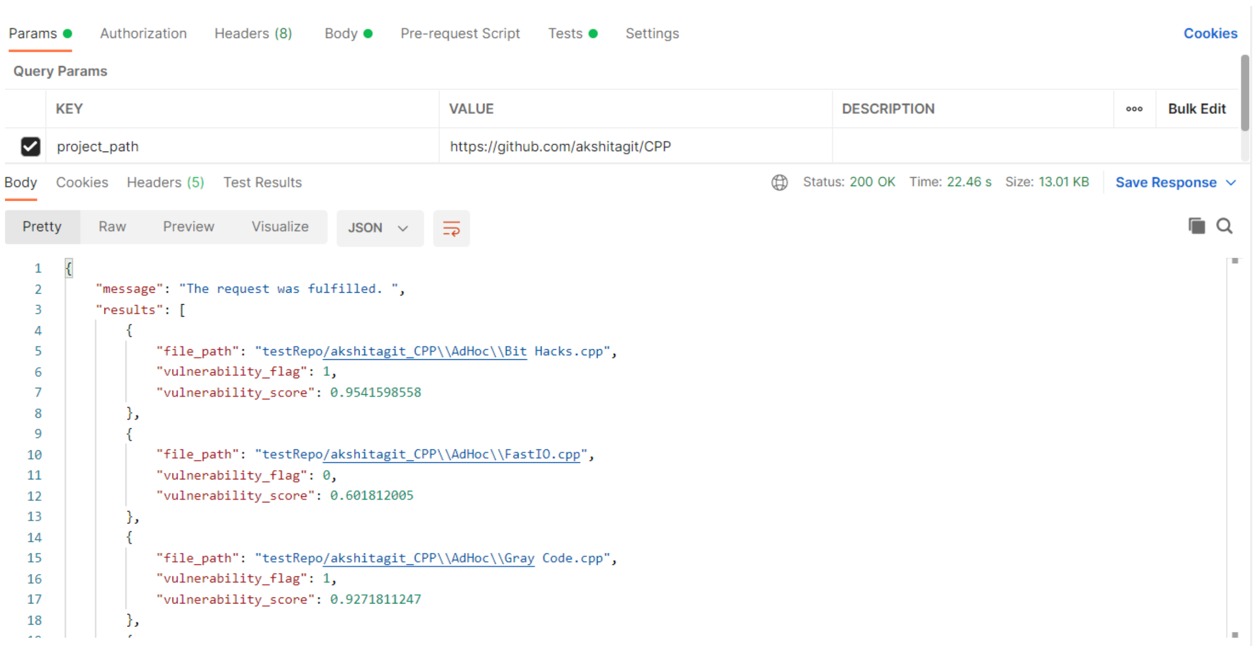
4.5.1.4 Results

A pre-trained BERT [81] model is employed. It is, in fact, the pre-trained BERT model for sequence categorization. In terms of size, it is classified as a BERT base model. This implies that the model possesses the following characteristics:

Table 41: Hyper-parameters of BERT base model

Number of layers	12
Hidden size	768
Total parameters	110M

Two case studies are presented here, one for a C++ project and one for a Java project. The major objective of these case studies is to assist the reader in understanding how the suggested solution works and what advantages are expected. The project that was used for the C++ case study is stored at GitHub, and its URL is: https://github.com/akshitagit/CPP. Below we provide screenshots from the request and the response using POSTMAN:



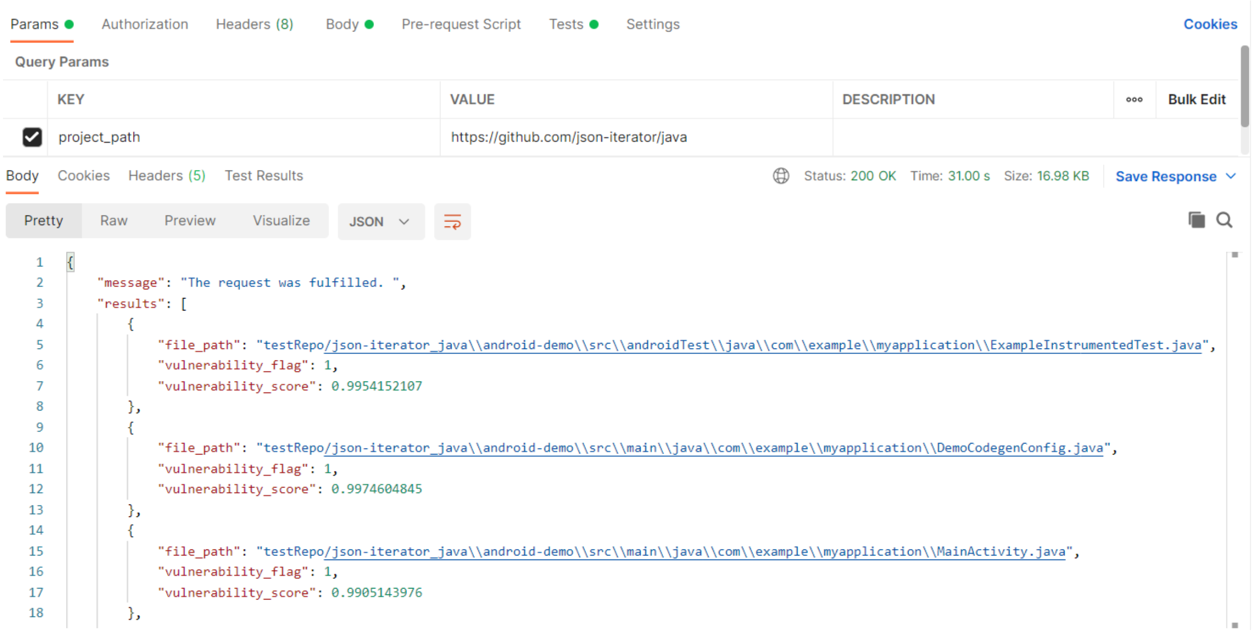
KEY	VALUE	DESCRIPTION
☑ project_path	https://github.com/akshitagit/CPP	

Status: 200 OK Time: 22.46 s Size: 13.01 KB

```
{
    "message": "The request was fulfilled. ",
    "results": [
        {
            "file_path": "testRepo/akshitagit_CPP\\AdHoc\\Bit Hacks.cpp",
            "vulnerability_flag": 1,
            "vulnerability_score": 0.9541598558
        },
        {
            "file_path": "testRepo/akshitagit_CPP\\AdHoc\\FastIO.cpp",
            "vulnerability_flag": 0,
            "vulnerability_score": 0.601812005
        },
        {
            "file_path": "testRepo/akshitagit_CPP\\AdHoc\\Gray Code.cpp",
            "vulnerability_flag": 1,
            "vulnerability_score": 0.9271811247
        },
```

Figure 52: The request and response of the Vulnerability Assessment service for the C++ Use Case

The same process is followed for the Java case study where we analysed the GitHub project with url https://github.com/json-iterator/java:



KEY	VALUE	DESCRIPTION
project_path	https://github.com/json-iterator/java	

Status: 200 OK Time: 31.00 s Size: 16.98 KB

```
{
    "message": "The request was fulfilled. ",
    "results": [
        {
            "file_path": "testRepo/json-iterator_java\\android-demo\\src\\androidTest\\java\\com\\example\\myapplication\\ExampleInstrumentedTest.java",
            "vulnerability_flag": 1,
            "vulnerability_score": 0.9954152107
        },
        {
            "file_path": "testRepo/json-iterator_java\\android-demo\\src\\main\\java\\com\\example\\myapplication\\DemoCodegenConfig.java",
            "vulnerability_flag": 1,
            "vulnerability_score": 0.9974604845
        },
        {
            "file_path": "testRepo/json-iterator_java\\android-demo\\src\\main\\java\\com\\example\\myapplication\\MainActivity.java",
            "vulnerability_flag": 1,
            "vulnerability_score": 0.9905143976
        },
```

Figure 53: The request and response of the Vulnerability Assessment service of the Java Use Case

The useful information extracted by this model is that a vulnerability flag and a vulnerability score is assigned to each source code file (in case of C++) or class file (in case of Java) in the project. The vulnerability flag indicates the potential existence of vulnerabilities in the file and it can be equal either to zero (i.e. indicating that the file is not likely to be vulnerable) or to one (i.e. indicating that the file is likely to be vulnerable). The vulnerability score is the likelihood with which the model assigns these flags to the files. In other words, the score denotes how sure the model is about its predictions.

4.5.2 Security related Static Analysis

A software vulnerability is a defect in the design, development, or configuration of software that in case of exploitation it can violate a security policy [83]. The vast majority of software vulnerabilities are produced by a small number of common programming errors [84],[85]. These errors are made by developers throughout the coding process, usually due to a lack of security expertise or to the rapid production cycles. It is, however, unrealistic to expect them to recollect hundreds of security-related issue patterns and bad behaviors that they should avoid. As a result, effective tools are required to help them avoid introducing such security issues and, as a result, write more safe code [85], [86].

Static analysis has been demonstrated to be effective in discovering security issues early enough in the software development process. Their primary differentiating characteristic is that they are applied immediately to the system's source or compiled code (or a representation of the source code), without the requiring its execution. Indeed, static analysis has been shown to be useful in detecting security flaws early in the software development process. It is regarded as an essential approach for enhancing security during the software development process as a whole. This belief is shared by several software security experts, while well-established secure software development lifecycles (SDLCs), such as Microsoft's SDL[18], OWASP's OpenSAAM[19], and Cigital's Touchpoints[20], advocate for the use of static analysis as the primary mechanism for adding security during the SDLC's coding (i.e., implementation) phase. Furthermore, according to the BSIMM[21] initiative, ASA is a security activity frequently used by big technology businesses such as Google, Microsoft, Adobe, and Intel. As a result, static analysis should be incorporated

[18] https://www.microsoft.com/en-us/securityengineering/sdl
[19] https://www.opensamm.org/
[20] http://www.swsec.com/resources/touchpoints/
[21] https://www.bsimm.com/

into the software development process for security-critical software systems such as web apps, IoT software, etc.

Besides their capacity to discover possible security concerns (i.e., vulnerabilities) in a software product's source code, static analysis techniques may be used as the foundation for obtaining more complex and abstract information about the examined software's overall security. In reality, because static code analyser outputs are large lists of raw warnings (i.e. alerts), they do not provide easy-to-understand insight to software product stakeholders (e.g., developers, engineers, project managers, etc.). As a result, appropriate knowledge extraction techniques should be utilized to extract security-related information from these raw data and provide it to the user in a more natural and readily comprehensible manner (e.g., through aggregation, visualization, etc.). To that aim, static analysis tool findings have recently begun to be utilized as the foundation for the development of models and methodologies capable of aggregating the raw warnings produced by the tools in order to give quantitative representations of critical security features of software programs. The low-level alerts generated by static code analysers, as well as the high-level metrics derived from their smart aggregation, are intended to facilitate the development of more safe software.

To that end, the SmartCLIDE project aims to provide solutions for monitoring and enhancing the security of software applications (task- or workflow-level) based on static analysis, correctly configured to detect security problems. In particular, we conducted a review of the literature to identify the best static code analysers for detecting security problems. Then, we created the Security-related Static Analysis Subcomponent (SSAS), a static analysis framework/platform that combines popular static code analysers that are appropriately configured to identify security vulnerabilities. The framework (i.e., subcomponent) is extremely configurable, enabling the user to specify which sorts of security vulnerabilities (e.g., Buffer Overflow, Cross-site Scripting, Denial of Service, etc.) it should look for while evaluating a given software. The platform is also highly extensible, allowing for the incorporation of new static code analysers to address different demands, domains, and programming languages. Aside from the static analysis itself, an aggregator module is used for advanced aggregation of the static analysis findings to offer additional high-level insights (e.g., at the workflow level) and more abstract information with regard to the security level of the examined program. State-of-the-art models and aggregation approaches were used and extended for the design of the aggregator module to fit in the framework of the SmartCLIDE project. The quantitative aggregated expressions are anticipated to give helpful insights to non-technical stakeholders such as project managers, facilitating decision making across the whole software development process. A REST API is provided to access the SSAS subcomponent. It is illustrated in two case studies, which focus on real-world open-source applications.

4.5.2.1 Security Analysis Platform/Framework

One critical step in our study was to conduct a survey of current static code analysers that might be used to check and validate the security level of software products. There are several static analysis tools available, each of which covers a single language or a group of languages. Research has been performed on the right tools to incorporate into the model, which must address a wide variety of security concerns while also being extendable and relevant to other programming languages in the future. The first focus of SmartCLIDE is on evaluating software products built in the Java and Python programming languages. The tools that will be chosen for integration into the framework should be able to detect errors in the source code, offer a comprehensive analysis of the code and the error, propose an indicated solution to the problem, and detect a wide variety of security risks.

To comply with the project's open-source philosophy, we opted to employ open-source technologies. Furthermore, this would provide the community with free access to our ideas, improving their usefulness. The tools described above were deliberately picked since each one covers a different field, and combining them would result in greater overall coverage of security concerns. CK[22] offers a number of metrics for Java code, measurements that are directly applicable to code characteristics and may be incorporated in security models. Furthermore, CK generates thorough csv files that list all of the errors discovered by the tool as well as their location (file/class, line of code, directory). We chose to allow users to calculate

[22] https://github.com/mauricioaniche/ck.

software metrics using our security analysis methodology, because there are several studies that indicate a tight link between software metrics and software vulnerabilities [87], [88], [89], [90].

PMD[23] is a static code analyser that identifies infractions to specified rules based on best practices. PMD defines a collection of rules from which one may construct appropriate rulesets addressing security problems. PMD comprises exception handling rules, abused functionality rules, null pointer rules, and other rules that are combined together to build a ruleset for particular situations (Null Pointer ruleset, Exception Handling ruleset, etc.). Furthermore, PMD offers a thorough csv file (NullPointer.csv, ExceptionHandling.csv) addressing the concerns and rules of each ruleset inside the source code for each individual ruleset. We have used PMD as the foundation of security models in the literature [91].

SonarQube[24] is a platform that currently supports 27 programming languages. It comes with a range of security profiles by default, as well as the ability to construct profiles that address particular security concerns that the user may choose to include in a security profile. By providing an API, SonarQube exposes the proper endpoints to retrieve the analysis scores in JSON form, after completing the analysis. Additionally, by visiting the SonarQube site, a detailed analysis of the project analysed with all the metrics and issues raised is presented in Sonar GUI, giving the user the ability to examine everything in detail. Pylint and Bandit are two static analysers designed particularly for Python. Both improve Sonarqube's security coverage and round out the entire security coverage for Python applications.

Following the selection of the most appropriate static code analysers, the service that matches to our security analysis framework/platform is built. The framework's functionality is accessible via a RESTful API. In detail, the user can access the service using an HTTP request. This request should include the url of the project to be examined (e.g., GitHub, GitLab, Bitbucket, etc.), the language of the project to be analysed (currently Python or Java), and a JSON including the desired configuration of the selected tools as well as of the aggregator. The entire architecture is suited to scalability. Specifically, our service is built on platform logic, which means that the user will be able to access it via HTTP requests, it will be easy to include in a variety of projects because its functionalities are exposed via a RESTful API, and it will be easy to extend and support more programming languages and tools included, because it is built in MVC logic, which means that there is a different sub-segment for each tool.

When the analysis is finished, the service returns a JSON file containing the values of each property picked by each independent tool, as well as the property scores, characteristic scores, and overall security index. Furthermore, by invoking a different URL, the user has the opportunity to save locally a full CSV report from all the tools used, detailing errors in the source code, their specific location, and, in some cases, suggestions for how to address such issues. The security analysis platform is highly customizable.

Aside from specifying the tools and security concerns the framework should look for, the user may customize the aggregator via the submitted request. More specifically, the user may choose the security model's properties and characteristics, as well as the thresholds for assessing the properties and the weights for computing the scores of the characteristics and the overall security score of the system. This is significant because it enables the user to build their own models for examining the security level of software, since it is well recognized in the literature that no one model is enough for evaluating all types of software [85]. It should be mentioned, however, that predefined security models have been developed based on our security expertise as default security profiles to be used by users either for assessing their software projects or as a blueprint on how to construct their own models.

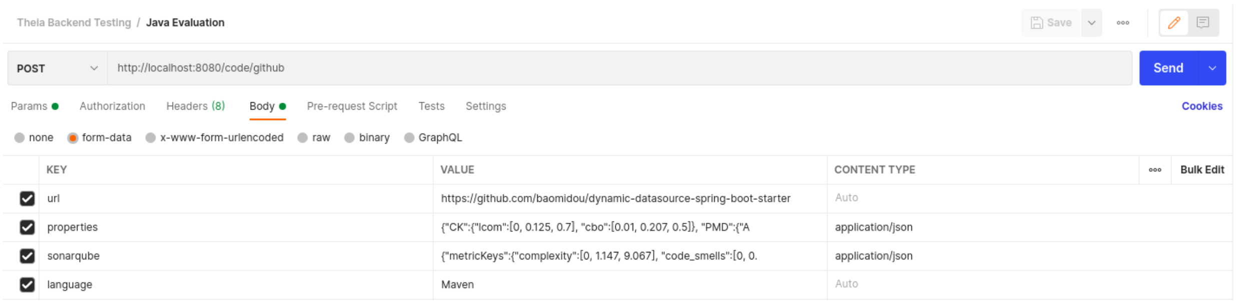
4.5.2.2 Results

The suggested security analysis framework/platform is presented in this section through two case studies on real-world open-source software applications. In particular, two examples are presented, one for Java and one for Python programming language. The primary objective of these case studies is to assist the reader in understanding how the suggested solution works and what advantages are anticipated. As previously stated, the functionality of our model is provided via a REST API. We will illustrate how to

[23] https://pmd.github.io/.
[24] https://www.sonarqube.org/.

correctly call our services using POSTMAN, what sort of request our endpoint allows, the response to the request, and the functionality behind the calls.

A screenshot of the POSTMAN request for evaluating a Java project using the default Java model is shown below. This is a tutorial for anyone interested in learning about the structure of a request and using our platform.



Theia Backend Testing / Java Evaluation

POST http://localhost:8080/code/github

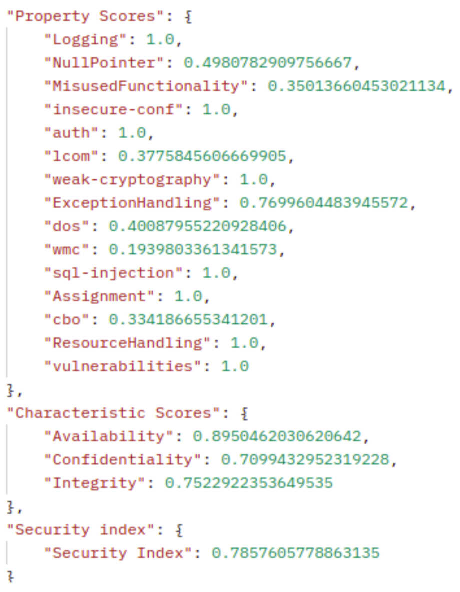
Params ● Authorization Headers (8) Body ● Pre-request Script Tests Settings

none ● form-data x-www-form-urlencoded raw binary GraphQL

	KEY	VALUE	CONTENT TYPE
✓	url	https://github.com/baomidou/dynamic-datasource-spring-boot-starter	Auto
✓	properties	{"CK":{"lcom":[0, 0.125, 0.7], "cbo":[0.01, 0.207, 0.5]}, "PMD":{"A	application/json
✓	sonarqube	{"metricKeys":{"complexity":[0, 1.147, 9.067], "code_smells":[0, 0.	application/json
✓	language	Maven	Auto

Figure 54: The request that needs to be submitted for analysing a Java project using the SSAS service

When the analysis is finished, the service delivers a JSON file containing the findings of the analysis. The SSAS response for the software repository utilized in the current scenario is shown below.

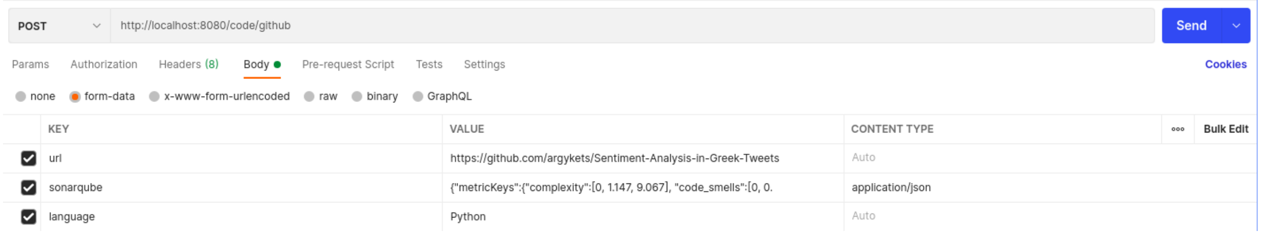


```
"Property Scores": {
    "Logging": 1.0,
    "NullPointer": 0.4980782909756667,
    "MisusedFunctionality": 0.35013660453021134,
    "insecure-conf": 1.0,
    "auth": 1.0,
    "lcom": 0.3775845606669905,
    "weak-cryptography": 1.0,
    "ExceptionHandling": 0.7699604483945572,
    "dos": 0.40087955220928406,
    "wmc": 0.1939803361341573,
    "sql-injection": 1.0,
    "Assignment": 1.0,
    "cbo": 0.334186655341201,
    "ResourceHandling": 1.0,
    "vulnerabilities": 1.0
},
"Characteristic Scores": {
    "Availability": 0.8950462030620642,
    "Confidentiality": 0.7099432952319228,
    "Integrity": 0.7522922353649535
},
"Security index": {
    "Security Index": 0.7857605778863135
}
```

Figure 55: The response (i.e., evaluation report) of the SSAS service for the Java use case

The first section of the JSON contains the normalized values of the attributes chosen from each tool. Following that, the "Property Scores" section provides the utility function-calculated property scores as well as the thresholds for each property. Following that, "Characteristics Scores" were incorporated, which were computed by multiplying the weight by the property score for each characteristic. Finally, the overall "Security Index" of the project under consideration is given, which includes the security score as determined by the characteristic scores.

The similar method is used for Python project analysis. The only parameter that changes are the properties, which include the appropriate tools for Python analysis as well as the project's url and the language, which should be defined as Python. A screenshot of the POSTMAN request is shown below. This is a tutorial for anyone interested in learning about the structure of a request and using our platform.



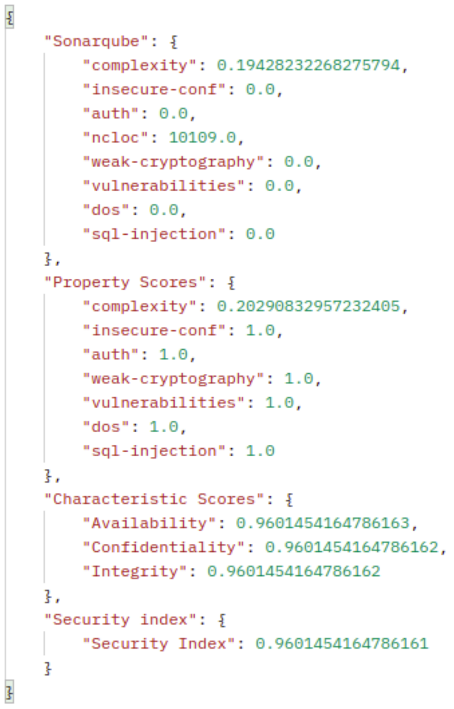
POST http://localhost:8080/code/github

Params · Authorization · Headers (8) · Body · Pre-request Script · Tests · Settings · Cookies

none · [✓ form-data] · x-www-form-urlencoded · raw · binary · GraphQL

	KEY	VALUE	CONTENT TYPE
✓	url	https://github.com/argykets/Sentiment-Analysis-in-Greek-Tweets	Auto
✓	sonarqube	{"metricKeys":{"complexity":[0, 1.147, 9.067], "code_smells":[0, 0.	application/json
✓	language	Python	Auto

Figure 56: The request submitted for analysing a Python project using the SSAS service



```
{
    "Sonarqube": {
        "complexity": 0.19428232268275794,
        "insecure-conf": 0.0,
        "auth": 0.0,
        "ncloc": 10109.0,
        "weak-cryptography": 0.0,
        "vulnerabilities": 0.0,
        "dos": 0.0,
        "sql-injection": 0.0
    },
    "Property Scores": {
        "complexity": 0.20290832957232405,
        "insecure-conf": 1.0,
        "auth": 1.0,
        "weak-cryptography": 1.0,
        "vulnerabilities": 1.0,
        "dos": 1.0,
        "sql-injection": 1.0
    },
    "Characteristic Scores": {
        "Availability": 0.9601454164786163,
        "Confidentiality": 0.9601454164786162,
        "Integrity": 0.9601454164786162
    },
    "Security index": {
        "Security Index": 0.9601454164786161
    }
}
```

Figure 57: The response of the SSAS service using Python project

For the time being, the idle Python security model supports SonarQube characteristics. In the future, open-source tools such as Bandit and Pylint will be introduced to improve Python's security. Again, as indicated in the response, the first portion of the JSON contains the normalized values of the SonarQube attributes. Following that, the “Property Scores” section provides the utility function-calculated property scores as well as the thresholds for each property. Following that, “Characteristics Scores” were incorporated, which were computed by multiplying the weight by the property score for each characteristic. Finally, the overall “Security Index” of the project under consideration is given, which includes the security score as determined by the characteristic scores.

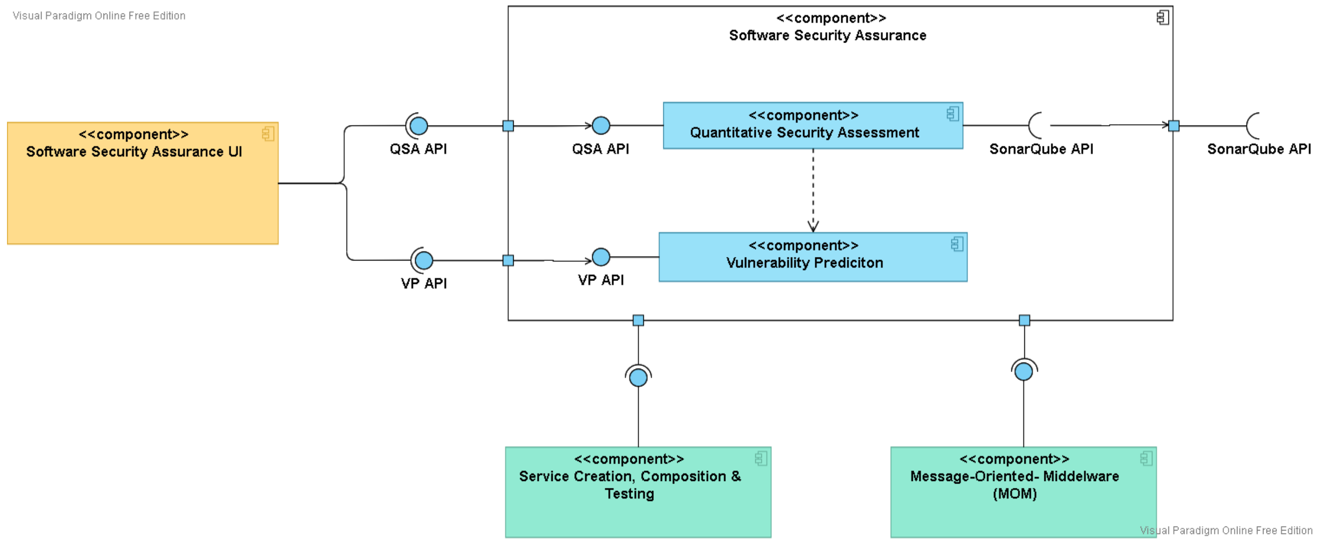

Visual Paradigm Online Free Edition
<<component>>
Software Security Assurance
<<component>>
Software Security Assurance UI
QSA API
QSA API
<<component>>
Quantitative Security Assessment
SonarQube API
SonarQube API
<<component>>
Vulnerability Prediciton
VP API
VP API
<<component>>
Service Creation, Composition & Testing
<<component>>
Message-Oriented- Middelware (MOM)
Visual Paradigm Online Free Edition

Figure 58: Software Security Assurance Module

Figure 59: Quantitative Security Assessment Subcomponent

Name	**Quantitative Security Assessment**
Functionality	Responsible for assessing security level of software applications based on Security Assessment Model
Relevant Use Cases (D1.3)	-
Functional Requirements	D47, D48, D49

Figure 60: 2Vulnerability Prediction Subcomponent

Name	**Vulnerability Prediction**
Functionality	Is responsible for predicting security issues (i.e., vulnerabilities)
Relevant Use Cases (D1.3)	-
Functional Requirements	D50, D51

4.5.3 Interface Specification

Table 42: Software Security Assurance Component Interfaces

No	API	Description	Type	
			Provided	**Required**
1	Security-related Static Analysis (SSA) API	The SSA, which is a subcomponent of the Software Security module, provides a web service which is exposed via API. Each request has mandatory parameters which are: • **repository:** Either a URL referring to project's repository (e.g., GitHub, GitLab, Bitbucket, etc.) or a zipped file containing the project from the local directory. • **properties:** a JSON file containing the metrics alongside with the thresholds of each metric for the model extraction. It also contains the characteristics with the weights for the model extraction. • **sonarqube:** a JSON file containing the selected SonarQube metrics alongside with the thresholds of each metric for the model extraction. • **language:** A value indicating the core language in which the analysed software project has been implemented (e.g., "java" if the selected software project is written in Java programming language). SSA Web service returns a JSON file containing the security assessment report, which includes the *security index* and the *security scores* of the model *attributes* (e.g., properties, characteristics, etc.), and the detailed low-level static analysis results (optional).	*	
2	Vulnerability Assessment (VA) API	The VA, which is a subcomponent of the Software Security component, provides a web service, which is exposed via API. Each request has mandatory parameters which are: • **project**: Either a URL referring to project's repository (e.g., GitHub, GitLab, Bitbucket, etc.) or a zipped file containing the project from the local directory. • **language**: A value indicating the core language in which the analysed software project has been implemented (e.g., "java" if the	*	

No	API	Description	Type	
			Provided	Required
		selected software project is written in Java programming language). VA Web service returns a JSON file containing the vulnerability prediction report. This report actually contains the names of the application files that are identified as vulnerable by the model, along with relevant information (e.g., each file's probability of containing vulne rabilities, a flag indicating whether they are vulnerable or not, etc.).		
3	SonarQube API	SonarQube provides a web API to expose its functionalities to other applications and services. SonarQube platform provides a set of tools to analyze source code and return the results in a JSON format through HTTP calls. We use SonarQube platform to enhance our security model with properties provided by the platform as well as language extension. • **api/issues/** call accepts as a parameter the vulnerability (buffer-overflow, csrf, etc.) and returns an extensive JSON report of the vulnerability selected regarding the user's project. • **api/measures/** accepts a String parameter with comma separated metrics (e.g., code_smells, bugs, ncloc) and returns results of the parameters requested.		*
4	Publish API	Security Components requires the Publish API exposed by the SmartCLIDE Message-Oriented-Middleware (MOM) to send message to the Service Creation, Composition and Testing Component.		*
5	Consume API	Security Components requires the Consume API exposed by the SmartCLIDE Message-Oriented-Middleware (MOM) to receive messages from Service Creation, Composition and Testing Component, Service Discovery, and RMV component.		*

4.6 Intercommunication

Message Oriented Middleware (MOM) component is responsible for the inter-component communication with the SmartCLIDE platform. The MOM is designed and implemented as a message broker which is a piece of software that enables applications, services, and systems to communicate with one another to exchange information[92] [93]. This communication is achieved by translating messages between formal messaging protocols, allowing independent services written in different languages or implemented in various platforms to interact with each other.

SmartCLIDE's MOM component offers standardized means of handling the data flow between the components of the SmartCLIDE platform. Thus, developers using the SmartCLIDE platform can focus on the on the core logic of the application. MOM can validate, route, store, and deliver messages to the appropriate destinations, allowing senders to issue messages without knowing where the receivers are and whether they are active or not, thus facilitating the decoupling of services and processes within systems.

There are several message broker's available, with popular choices being Apache Kafka and RabbitMQ. In following section briefly introduce key technologies used for the design and implementation of the MOM component, followed by presentation of our actual design approach.

4.6.1 Background Information

4.6.1.1 Message brokers

A message broker often relies on a message queue, a component that orders and stores messages until the receiving applications can process them, in order to guarantee message delivery and reliable message storage. Messages within a message queue are stored in the same order as they were transmitted and remained in the queue until receipt is confirmed.

A message broker offers two basic message distribution patterns:

- **Point-to-point messaging**: In this pattern, there is a one-to-one relationship between the sender and the receiver of the message. Each message is sent and consumed only once. This pattern is usually used when some action needs to be performed only once. When multiple consumers are used, each consumer typically receives a portion of the message stream to allow for concurrent processing.
- **Publish/subscribe messaging**: This is a broadcast-style distribution method, often referred to as "pub/sub" where producers send messages on a topic. In this model, the producer is known as a *publisher*, and the consumer is known as a *subscriber*. One or many publishers can publish on the same topic, and a message from one or many publishers can be received by many subscribers. Subscribers subscribe to topics, and all subscribers on the topic receive all messages published to that topic.

4.6.1.2 Apache Kafka

Apache Kafka[25] is an open-source stream-processing software platform aiming to provide a unified, high-throughput, low-latency framework for handling real-time data feeds. It has been developed by the Apache Software Foundation and is written in Java and Scala. Kafka does not implement the notion of a queue. Instead, Kafka stores collections of records in categories called topics. For each topic, Kafka maintains a partitioned log of messages. Each partition is an ordered, immutable sequence of records, where messages are continually appended. Kafka is distributed; therefore, topic partitions are replicated across various nodes.

Producers, client applications that publish messages to Kafka, create the data within the topics and consumers, client applications that subscribe to topics, read from those topics. In Kafka, messages are just simple byte arrays; the developers can utilize them in order to store any object in any format that they wish (e.g., JSON, plain text etc.). Developers can also opt to attach a key to a message, guaranteeing that all messages with that specific key will get to the same partition. During consumption from a topic, a group with multiple consumers can also be configured. Each of the consumers in a specific group will access messages from a particular subset of partitions within the topics they subscribe to. This will ensure that

[25] https://kafka.apache.org/

every message is delivered to one consumer in the group, and all of the messages that carry the same key are sent to the same consumer.

Figure 61 shows the architecture of Apache Kafka:

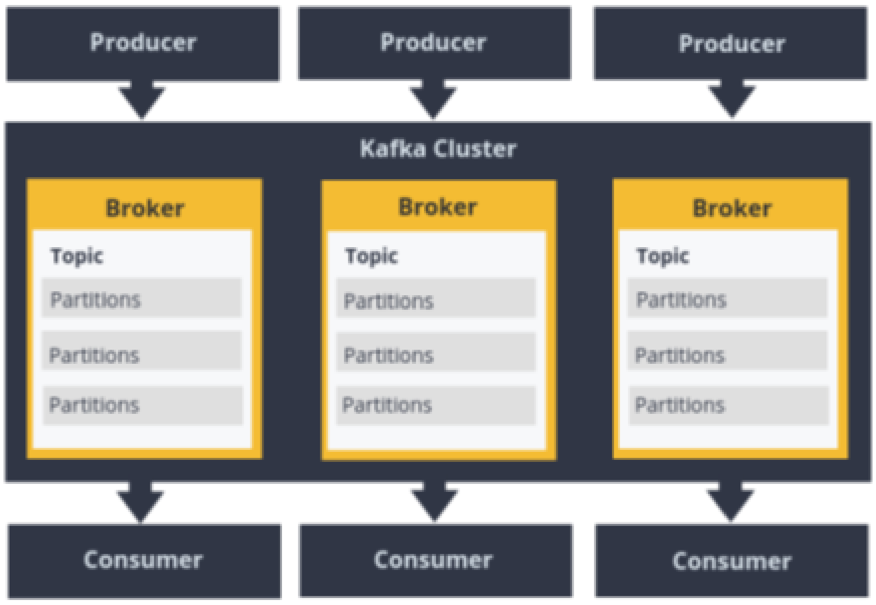


Figure 61: Apache Kafka Architecture

Apache Kafka supports use cases such as metrics, activity tracking, log aggregation, stream processing, commit logs and event sourcing. The following messaging scenarios are especially suited for Kafka:

- Streams with complex routing, throughput of 100K/sec events or more, with "at least once" partitioned ordering
- Applications requiring a stream history, delivered in "at least once" partitioned ordering. Clients can see a 'replay' of the event stream
- Event sourcing, modelling changes to a system as a sequence of events
- Stream processing data in multi-stage pipelines. The pipelines generate graphs of real-time data flows.

Kafka is a distributed system consisting of servers and clients that communicate via a high-performance TCP network protocol. It can be deployed on bare-metal hardware, virtual machines and containers in on-premises as well as cloud environments. Moreover, the Kafka community has built optimized client libraries for many programming languages, such as Python, PHP, Node.js, .NET and more. However, since Kafka is written in Java, the native Java client library delivers the best possible performance.

4.6.1.3 RabbitMQ

RabbitMQ[26] is one of the most widely used open-source message brokers [94]. It was originally based on the Advanced Message Queuing Protocol (AMQP), but later has been modified to support other messaging protocols as well, like Message Queuing Telemetry Transport (MQTT) and Streaming Text Oriented Messaging Protocol (STOMP). It is implemented in Erlang OTP, a technology tailored for building reliable, fault tolerant, stable, and highly scalable systems capable of handling very large numbers of concurrent operations [95] [96].

RabbitMQ is a general-purpose message broker that has been designed to be used for a variety of messaging scenarios and provides client libraries for several programming languages such as Java, Python, PHP and

[26] https://www.rabbitmq.com/

.NET. It uses a smart broker/dumb consumer model focused on delivering messages to consumers constantly and supports several variations of pub-sub, point to point and request-replaying messaging techniques. Moreover, it offers both synchronous and asynchronous modes of communication.

A really strong asset of RabbitMQ is the power routing capabilities it provides. The routing logic in RabbitMQ is implemented in different (so-called) exchange types. An exchange is responsible for routing the messages to different queues. Messages are not published directly to a queue; instead, the producer sends messages to an exchange which forwards the message to the appropriate queue(s) with the help of bindings and routing keys. A binding is a kind of link between a queue and an exchange. There are four exchange types:

- **Direct exchanges** route messages according to the routing key that the message carries. The routing key is a string of words, separated by periods, that has some relevance to the message.
- **Fanout exchanges** route messages to all available queues. In this broadcasting type of exchange, the routing key is ignored.
- **Topic exchanges** route messages to one or more queues according to a complete or partial match with the routing key.
- **Header exchanges** route messages based on the message headers, which can contain more attributes than a routing key.

Figure 62 presents a typical message flow in RabbitMQ:

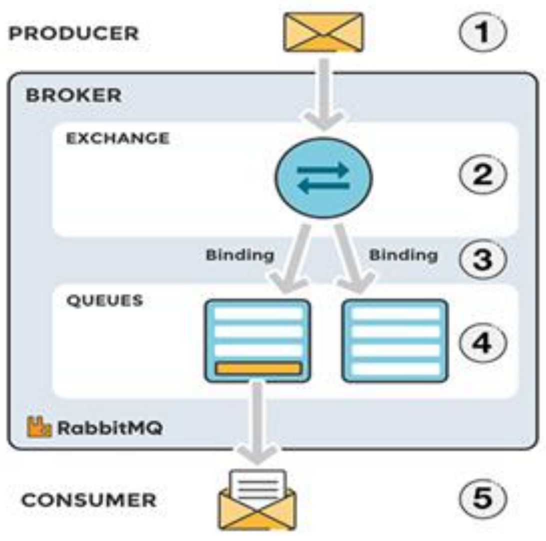


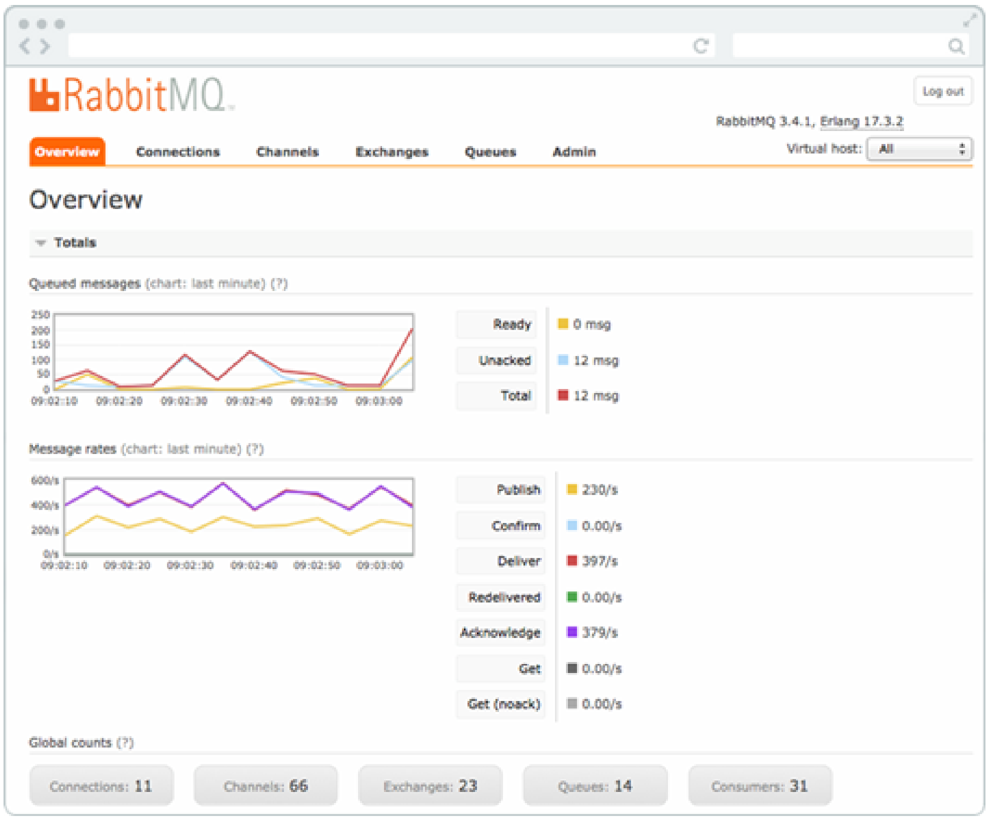
Figure 62: RabbitMQ message flow

As it can be seen, the producer publishes a message to an exchange. The exchange receives the message and routes it taking into account different message attributes, such as the routing key. In order for a queue to receive a message from an exchange, bindings have to be created between the queue and the exchange. In the specific case depicted in figure n. there are two bindings to two different queues from the exchange. The exchange routes the message into the queues depending on message attributes. Consequently, the message is received by the queue and it remains in the queue until a consumer retrieves it. Lastly, the consumer receives and handles the message.

RabbitMQ offers several plugins that can be added to extend use cases and integration scenarios. These plugins extend the core broker functionality in various ways such as, system state monitoring, support for more protocols, node federation and more. A very useful plugin is the RabbitMQ Management plugin which is included in the *RabbitMQ* distribution and can be enabled by the RabbitMQ server administrators in order to provide a GUI for managing and monitoring RabbitMQ. The Management plugin offers a user-friendly interface that allows the system administrators to handle and monitor the RabbitMQ server from a web browser. The plugin periodically collects and aggregates data about many aspects of the system and the management UI is implemented as a single page application which relies on the RabbitMQ HTTP API. Some of the core features of the plugin are:

- Declaring, listing and deleting exchanges, queues, bindings, users and user permissions
- Monitoring queue length, message rates (globally and per queue, exchange or channel) and resource usage of queue
- Managing users (provided administrative permissions of the current user).
- Managing policies and runtime parameters (provided sufficient permissions of the current user).
- Forcing to close client connections and purge queues
- Sending and receiving messages (useful in development environments and for troubleshooting).

Additionally, the UI application supports recent versions of Google Chrome, Safari, Firefox, and Microsoft Edge browsers. Figure 63, Figure 64, and Figure 65 below show some tabs of the Management UI menu. Figure 64 presents charts for queued messages and message rates, while Figure 64 and Figure 65 show the declared exchanges and queues respectively.

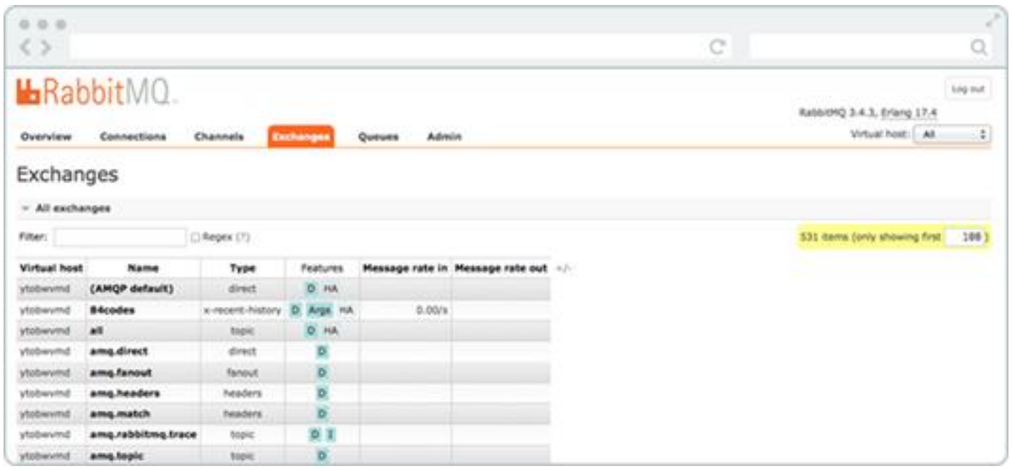


Ready	🟨 0 msg
Unacked	🟦 12 msg
Total	🟥 12 msg

Publish	🟨 230/s
Confirm	🟦 0.00/s
Deliver	🟥 397/s
Redelivered	🟩 0.00/s
Acknowledge	🟪 379/s
Get	■ 0.00/s
Get (noack)	■ 0.00/s

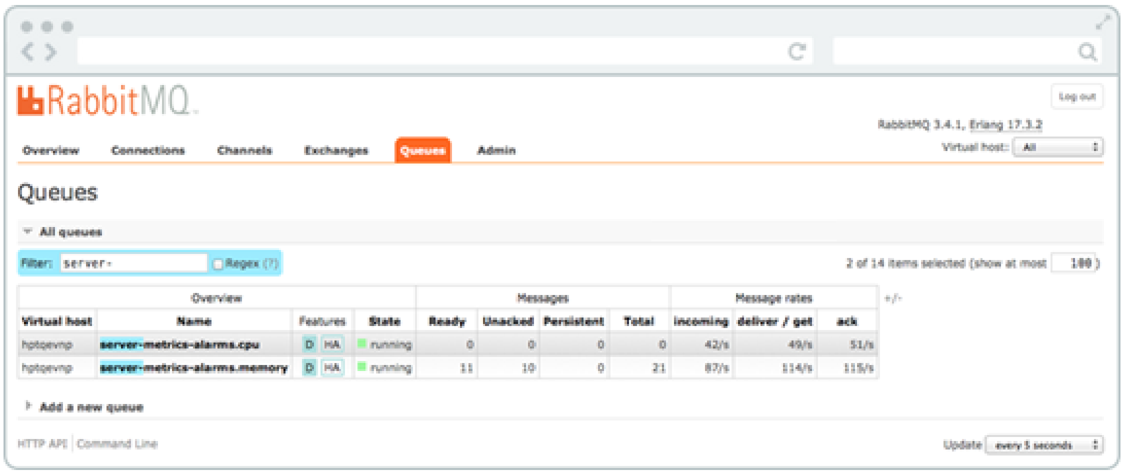
Connections: 11 Channels: 66 Exchanges: 23 Queues: 14 Consumers: 31

Figure 63: Message queues and rates



Virtual host	Name	Type	Features	Message rate in	Message rate out
ytobwvmd	**(AMQP default)**	direct	D HA		
ytobwvmd	**84codes**	x-recent-history	D Args HA	0.00/s	
ytobwvmd	**all**	topic	D HA		
ytobwvmd	**amq.direct**	direct	D		
ytobwvmd	**amq.fanout**	fanout	D		
ytobwvmd	**amq.headers**	headers	D		
ytobwvmd	**amq.match**	headers	D		
ytobwvmd	**amq.rabbitmq.trace**	topic	D I		
ytobwvmd	**amq.topic**	topic	D		

Figure 64: Declared message exchanges



RabbitMQ 3.4.1, Erlang 17.3.2

Virtual host: All

Queues

All queues

Filter: server- Regex (?)

2 of 14 items selected (show at most 100)

Overview				Messages				Message rates		
Virtual host	Name	Features	State	Ready	Unacked	Persistent	Total	incoming	deliver / get	ack
hptqevnp	server-metrics-alarms.cpu	D HA	running	0	0	0	0	42/s	49/s	51/s
hptqevnp	server-metrics-alarms.memory	D HA	running	11	10	0	21	87/s	114/s	115/s

Add a new queue

Update every 5 seconds

Figure 65: Declared queues

4.6.1.4 Spring, Spring Boot and Spring AMQP

Spring[27] is a popular, open-source framework that provides comprehensive infrastructure support for developing Java applications. Spring is based on dependency injection and Inversion of Control[28] , two key principles that enable developers to create modular applications consisting of loosely coupled components. It also offers built-in support for typical application requirements, such as validation, data binding, exception handling, resource, and event management etc. and integrates with various Java EE technologies[29] like RMI (Remote Method Invocation)[30] and Java Web Services[31]. In general, Spring Framework provides all the tools and features developers need in order to create, in a loosely coupled way, cross-platform Java EE applications capable of running in any environment [32].

Spring Boot[33] is an extension of the Spring framework, which eliminates the boilerplate configurations required for setting up a Spring application. Although Spring is a capable and comprehensive framework, a significant amount of time and knowledge is still necessary in order to configure, set up and deploy a Spring application. Spring Boot mitigates the effort required from a developer's perspective and simplifies the development of web applications and microservices with Java Spring Framework, by providing the following capabilities:

- Auto-configuration of Spring and 3rd party libraries whenever possible
- Creation of stand-alone Spring applications by embedding directly web servers, such as Tomcat, Jetty or Undertow (no need to deploy WAR files)
- Opinionated 'starter' dependencies to simplify the build and application configuration
- Provision of production-ready features such as metrics, health checks, and externalized configuration
- No requirement for XML configuration

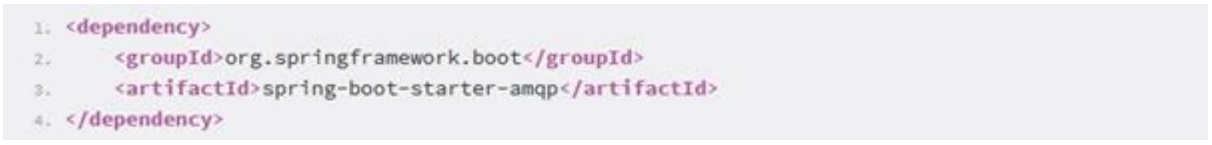
Spring AMQP[34] is the Spring implementation of AMQP-based messaging solutions and applies the spring core dependency injection to the AMQP-based messaging application programming. Spring AMQP consists of two modules, spring-amqp and spring-rabbit. The 'spring-amqp' module is the base abstraction for the AMQP Protocol implementation and represents the core AMQP "model", whereas the 'spring-rabbit'

[27] https://spring.io/
[28] https://www.educative.io/edpresso/what-is-inversion-of-control
[29] https://www.oracle.com/java/technologies/java-ee-glance.html
[30] https://www.ibm.com/docs/en/sdk-java-technology/7?topic=uc-java-remote-method-invocation-2
[31] https://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html
[32] https://www.ibm.com/cloud/learn/java-spring-boot
[33] https://spring.io/projects/spring-boot
[34] https://spring.io/projects/spring-amqp

module is the RabbitMQ implementation which provides broker-specific implementations for the abstractions offered by spring-amqp. These two modules combined provide features such as:

- Connection management of the RabbitMQ broker
- Automatic declaration of queues, exchanges and bindings
- RabbitTemplate, a class for sending and receiving messages easily
- Listener container for asynchronous processing of inbound messages

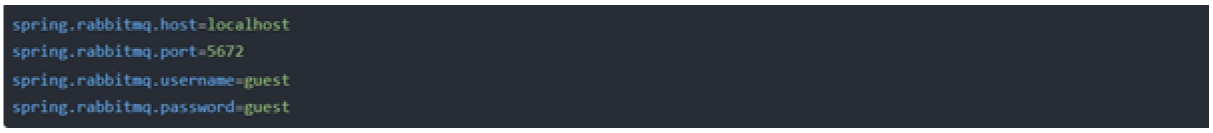
The Spring AMQP project can be easily integrated into a Spring boot application by simply adding the *spring-boot-starter-amqp* dependency to the application's *pom.xml*, as shown in Figure 66. Spring Boot Starters[35] [16] are dependency descriptors that contain a lot of predefined dependencies under a single name, decreasing the total number of dependencies to be added and the configuration time for developers.



```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

Figure 66: Integration of Spring AMQP into Spring boot application

Another advantage of using Spring AMQP along with Spring Boot is that several configurations of the RabbitMQ server, such as host name and port number, can be easily performed through the use of a simple properties file, as described in Figure 67.



```
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
```

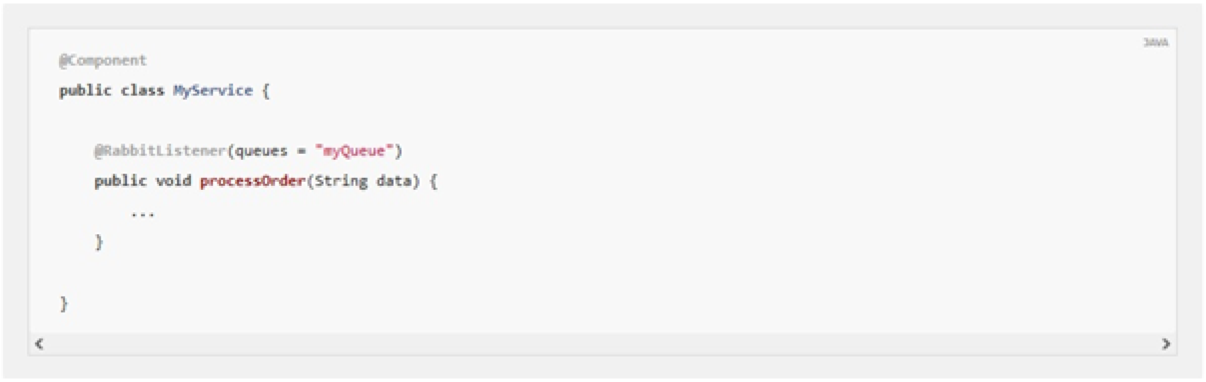
Figure 67: Configuration of the RabbitMQ server

There are two ways to receive a message from the broker in Spring AMQP. The simplest option is to poll for one message at a time with a polling method call. The more complicated yet more common approach is to register a listener that receives messages on-demand, asynchronously.

In the polling consumer scenario, the *AmqpTemplate*[36], [17] the core interface for sending and receiving messages in Spring AMQP, can be used for polled message reception. The *AmqpTemplate* takes care of the message receive and reply phases and includes several convenient methods for synchronously receiving, processing and replying to messages. Therefore, in most cases, developers should provide only an implementation of a callback function to perform some business logic for the received message and build a reply object or message, if needed[37] [18].

In the asynchronous consumer scenario, the easiest way to receive a message asynchronously is by using the annotated listener endpoint infrastructure which exposes a method of a managed bean as a Rabbit listener endpoint. Figure 68 shows an example of the *@RabbitListener* annotation.

[35] https://github.com/spring-projects/spring-boot/tree/main/spring-boot-project/spring-boot-starters
[36] https://docs.spring.io/spring-amqp/docs/current/reference/html/#amqp-template
[37] https://docs.spring.io/spring-amqp/docs/current/reference/html/#polling-consumer



```java
@Component
public class MyService {

    @RabbitListener(queues = "myQueue")
    public void processOrder(String data) {
        ...
    }

}
```

Figure 68: An example of the *@RabbitListener* annotation

In the above example, every time a message is available on the queue named *myQueue*, the *processOrder* method is invoked accordingly (in this case, with the payload of the message). The annotated endpoint infrastructure creates for each annotated method a message listener container behind the scenes, which is constantly listening for new messages arriving to the queue(s)[38] [19].

4.6.1.5 RabbitMQ vs Apache Kafka

A lot of discussion has been made over the years about which is the best message broker between Apache Kafka and RabbitMQ. The truth is that deciding whether to use Kafka or RabbitMQ was never easy [93] and the decision is highly dependent on the individual use case scenario. Furthermore, with both technologies improving day by day, the margins of advantage of the one tool over the other have become narrower.

In the context of SmartCLIDE, the main reason we opted in favor of RabbitMQ is that Kafka is not just a message broker. It is a streaming platform which excels in storing and analyzing streaming data and is very useful for systems that need to gain insight into the data. However, this kind of functionality is excessive for the needs of the SmartCLIDE project where the main role of the MOM component is to serve just as a means of communication between the different applications. RabbitMQ has been used extensively in the industry as the middleman between microservices and can scale more than enough, if necessary, to cover the needs of the SmartCLIDE project.

Additionally, there are some more reasons [92] for choosing RabbitMQ instead of Kafka, which are briefly described below:

- RabbitMQ has a more flexible routing mechanism and supports four different routing options, while Kafka has only one very simple routing approach
- RabbitMQ can support message priority by using queues called priority queues, which can deliver messages based on the priority that was set on the message. This can be very helpful in cases where some important messages must be delivered faster than other messages of less significance that have been placed earlier in the queue. Kafka on the other hand does not provide such kind of functionality
- The architecture of Kafka is quite complicated, and a developer should be familiar with concepts such as message offsets, partitions, and consumer groups. On the contrary, the way RabbitMQ works is simpler and straighter forward
- RabbitMQ has a built-in, user-friendly graphical interface for monitoring the RabbitMQ server via a web browser. For Kafka, there are also monitoring tools, both commercial and open source, but none of them comprises a built-in feature of the Kafka platform and their integration requires extra amount of time and effort.

[38] https://docs.spring.io/spring-amqp/docs/current/reference/html/#async-consumer

4.6.2 Design Approach

For the implementation of the MOM component, we have used the official RabbitMQ Docker image [97] in order to run the RabbitMQ server inside a Docker container. This is the easiest way to have a RabbitMQ instance up and running and enhances portability. For making RabbitMQ available to the other SmartCLIDE components, we have set up a RESTful API with the help of Spring Boot, thus offering HTTP access to the message broker. The API is briefly described in 4.6.3 and provides functionality for publishing messages to queues and consuming messages from queues as well. At first, we implemented some test queues and tried to send/receive test messages to/from these queues using the REST API to verify that the defined interfaces/endpoints work as expected.

Although a component can use the REST API to send a message to a queue instantly, the consumption of a message from a queue cannot take place immediately just by using the HTTP protocol, since the server (message broker) is not able to initiate a communication to the client (consumer). Instead, the message broker can send the message to the consumer only after receiving an HTTP GET request first. This means that a component should periodically send requests to the broker to check if there are any incoming messages in the corresponding queues, which is not very efficient from a performance perspective. To avoid this issue, we have investigated WebSockets [98], a technology that establishes a bidirectional, full-duplex communications channel between a client and a server, which operates over HTTP through a single TCP socket connection. This means that both parties can start sending data at any time, so the server can independently send data to the client without the client having to request it.

Apart from the REST API and the WebSockets, there is also another way for the SmartCLIDE components to access the message broker. Since all the components will be located in the same cluster and they will be able to communicate internally via the cluster network, the RabbitMQ AMQP-based API [99] can be utilized directly by the different components. RabbitMQ uses the AMQP protocol by default and offers numerous AMQP-based clients supporting the most popular programming languages. Each component can integrate the corresponding client library in its application code and communicate with the broker directly to connect to the appropriate queues and retrieve the desired messages. Although this type of communication is faster compared to the HTTP-based communication offered by the REST API, the use of the REST API can be still very useful in case there is no RabbitMQ client library available for the programming languages used for developing a component or if for some reason the message broker should be accessed from the outside world, since the RabbitMQ AMQP-based API can be utilized directly only inside the SmartCLIDE cluster.

Once we finalized the design and proceed with initial implementation, we focused on the specification of the publishing and subscribe ques as outlined in Table 43. To increase the readability, we have introduced the following abbreviations:

- *For the components:* Service Discovery (SD), Service Composition (SC), Security (S), Run-time Monitoring & Verification (RMV), Run-time Monitoring & Simulations (RMS), Deep Learning Engine (DLE), Context Handling (CH), Smart Assistant (SA), Deployment (DPL)
- *For the roles:* Producer (P) and Consumer (C).

Table 43: MOM publish/subscribe messaging ques

	SD(AIR)	SC (UoM)	S (CERTH)	RMV (TOG)	RMS (WT)	DLE (AIR)	CH (ATB)	SA (AIR)	DPL (WT)
SD (AIR)		Direct: ☒	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☒	Direct: ☐	Direct: ☐	Direct: ☐
		MoM: ☐ P: ☒ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐
SC (UoM)	Direct: ☐		Direct: ☐	Direct: ☒	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☐

	SD(AIR)	SC (UoM)	S (CERTH)	RMV (TOG)	RMS (WT)	DLE (AIR)	CH (ATB)	SA (AIR)	DPL (WT)
	MoM: ☒ P: ☒ C: ☒		MoM: ☒ P: ☒ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☒ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☒ C: ☒	MoM: ☐ P: ☐ C: ☐
S(CERTH)	Direct: ☐	Direct: ☐		Direct: ☒	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☐
	MoM: ☒ P: ☐ C: ☒	MoM: ☒ P: ☒ C: ☒		MoM: ☒ P: ☐ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐
RMV (TOG)	Direct: ☐	Direct: ☐	Direct: ☐		Direct: ☒	Direct: ☐	Direct: ☒	Direct: ☐	Direct: ☒
	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☒ C: ☐		MoM: ☒ P: ☒ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☒ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☒ C: ☐
RMS (WT)	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☒		Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☐
	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☐ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☐ C: ☒		MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☒ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☐ C: ☒
DLE (AIR)	Direct: ☒	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☐		Direct: ☐	Direct: ☒	Direct: ☐
	MoM: ☐ P: ☐ Cr: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐		MoM: ☒ P: ☐ C: ☒	MoM: ☐ P: ☒ C: ☐	MoM: ☐ P: ☐ C: ☐
CH (ATB)	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☒	Direct: ☐	Direct: ☐		Direct: ☐	Direct: ☐
	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☐ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☒ C: ☐		MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐
SA (AIR)	Direct: ☐	Direct: ☒	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☒	Direct: ☐		Direct: ☒
	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☒ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☐ C: ☒	MoM: ☐ P: ☐ C: ☐		MoM: ☒ P: ☐ C: ☒
DPL (WT)	Direct: ☒	Direct: ☒	Direct: ☐	Direct: ☒	Direct: ☐	Direct: ☐	Direct: ☐	Direct: ☐	
	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☐ C: ☒	MoM: ☒ P: ☐ C: ☒	MoM: ☐ P: ☐ C: ☐	MoM: ☒ P: ☐ C: ☒	MoM: ☐ P: ☐ C: ☐	

MOM component diagram is presented in Figure 69. As shown in Figure 69 below, the MOM component resides in the center of the system architecture and comprises three sub-components, namely the Message Checker, the Message Transformer and the Message Router. The Message Checker is the first point of interaction when communicating with the MOM component and verifies the validity of the incoming/outcoming messages while the actual routing of the messages is implemented by the Message Router sub-component. The Message Transformer modifies each message accordingly so that it can be parsed at both ends (publisher and consumer).

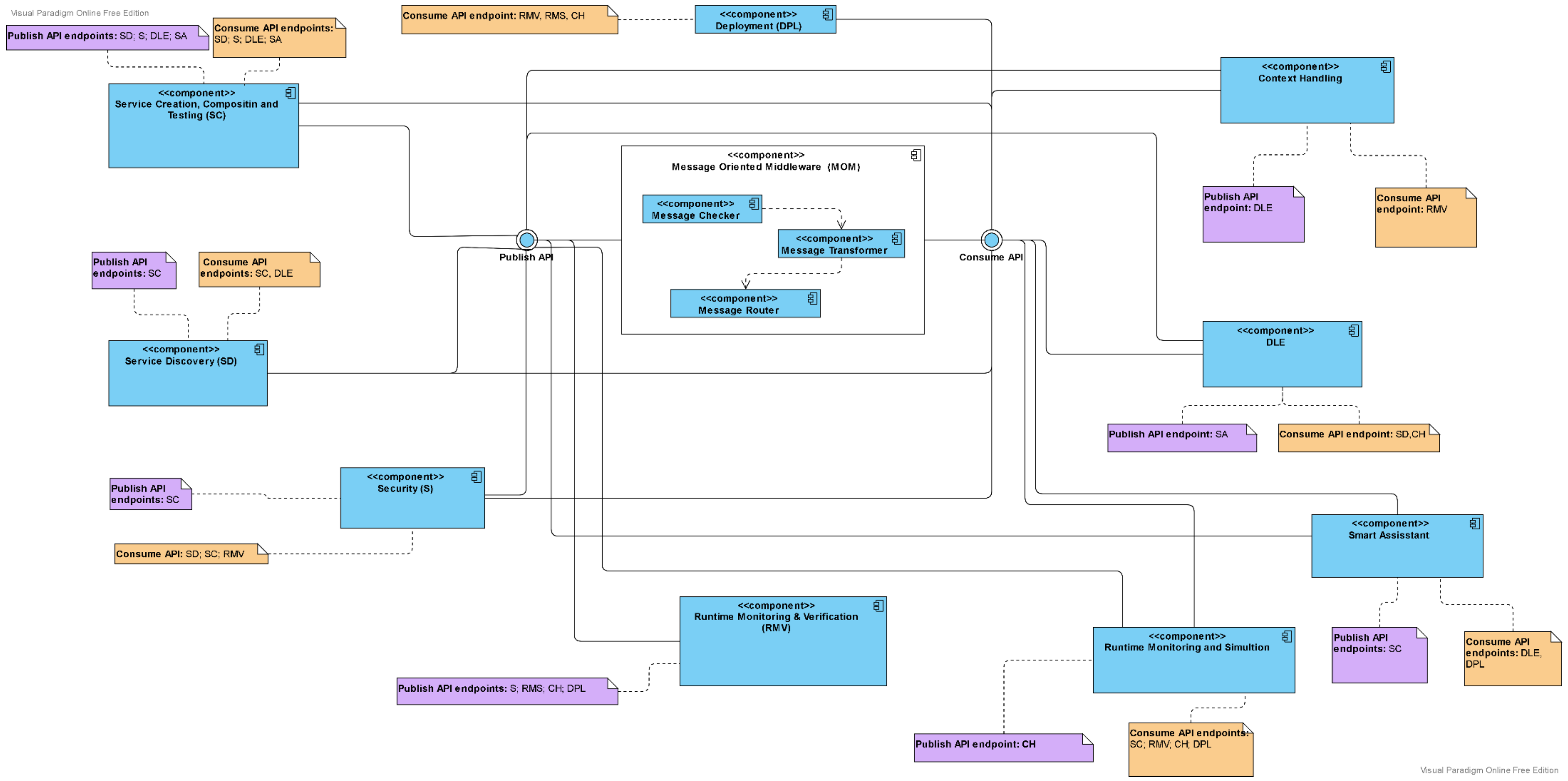

Visual Paradigm Online Free Edition
Publish API endpoints: SD; S; DLE; SA
Consume API endpoints: SD; S; DLE; SA
Consume API endpoint: RMV, RMS, CH
<<component>> Deployment (DPL)
<<component>> Context Handling
<<component>> Service Creation, Compositin and Testing (SC)
<<component>> Message Oriented Middleware (MOM)
<<component>> Message Checker
<<component>> Message Transformer
<<component>> Message Router
Publish API
Consume API
Publish API endpoint: DLE
Consume API endpoint: RMV
Publish API endpoints: SC
Consume API endpoints: SC, DLE
<<component>> Service Discovery (SD)
<<component>> DLE
Publish API endpoint: SA
Consume API endpoint: SD,CH
Publish API endpoints: SC
<<component>> Security (S)
Consume API: SD; SC; RMV
<<component>> Smart Assisstant
<<component>> Runtime Monitoring & Verification (RMV)
<<component>> Runtime Monitoring and Simultion
Publish API endpoints: SC
Consume API endpoints: DLE, DPL
Publish API endpoints: S; RMS; CH; DPL
Publish API endpoint: CH
Consume API endpoints: SC; RMV; CH; DPL
Visual Paradigm Online Free Edition

Figure 69: MOM Component Diagram

Table 44: MOM Message Checker Subcomponent

Name	Subcomponent 1
Functionality	MoM will be able to check if the exchanged messages, either at the sender's or at the receiver's end, comply with a specific format.
Relevant Use Cases (D1.3)	-
Functional Requirements	-

Table 45: MOM Message Transformer Subcomponent

Name	Subcomponent 1
Functionality	MoM will transform the data/messages from the sender's native format to the receiver's native format.
Relevant Use Cases (D1.3)	-
Functional Requirements	-

Table 46: MOM Message Router Subcomponent

Name	Subcomponent 1
Functionality	MoM should support several message routing policies and message delivery guarantees (e.g., at-most-once, and exactly-once).
Relevant Use Cases (D1.3)	-
Functional Requirements	-

4.6.3 Interface Specification

The Message Broker is accessible via a REST API. The base URL is:

http://160.40.53.126:8080/broker/api/v1

MOM component has 2 provided interfaces which are described in Table 47.

Table 47: MOM Component Interfaces

No	API	Description	Type	
			Provided	**Required**
1	**Publish API**	This API shall be used by a component that wants to send a message to another component. In order to send a message to the queue of the receiver, the producer of the message would make the following HTTP call: http://160.40.53.126:8080/broker/api/v1/publisher/{publisherName}/{consumerName} For example, if *Component A* wants to send a message to *Component B*, the following endpoint should be used: http://160.40.53.126:8080/broker/api/v1/publisher/componentA/componentB In case the name of the consumer is not specified in the request path, the message will be broadcasted to all the components that can get messages from the specific producer. For example, the HTTP call: http://160.40.53.126:8080/broker/api/v1/publisher/componentA would broadcast a message to all the components that have been defined as consumers for *Component A*.	*	
2	**Consume API**	This API provides access to the queue a component is listening to. All the other components, that want to communicate with the specific component, will send their messages to that queue. Then, the receiving component can get these messages by accessing the following endpoint: http://160.40.53.126:8080/broker/api/v1/consumer/{componentName} For example, if *Component A* wants to get the incoming messages from its queue, the following endpoint should be used: http://160.40.53.126:8080/broker/api/v1/consumer/componentA	*	

Publishing messages example:

According to the MoM Specification Table, the Service Creation component can send messages to the Service Discovery, Security, Deep Learning and Smart Assistant components. The following HTTP call can be used by the Service Creation component to broadcast a message to all the aforementioned components:

POST	http://160.40.53.126:8080/broker/api/v1/publisher/serviceCreation

Body request data format example:

```
{
  "serviceId ": " ff098b1b-e218-410d-a0a1-a486fdda3bc6",
  "serviceName ": " Test Service 1",
  "description ": " An example test service ",
  "owner": "7ac10294-159e-4153-8da2-ec5e5d2f2e58"
  }
```

Consuming messages example:

The Service Discovery component can perform the following HTTP call in order to retrieve an incoming message from its queue:

GET	http://160.40.53.126:8080/broker/api/consumer/serviceDiscovery

Response body data format example:

```
{
  "serviceId ": " ff098b1b-e218-410d-a0a1-a486fdda3bc6",
  "serviceName ": " Test Service 1",
  "description ": " An example test service ",
  "date ": "2021-06-08 12:42:18",
  "owner": "7ac10294-159e-4153-8da2-ec5e5d2f2e58"
  }
```

4.7 User Access Management

User Access Management (UAM), also known as identity and access management (IAM), is the act of defining and managing the roles and access privileges of individual network entities (users and devices) to a variety of cloud and on-premises applications. Users include customers, partners and employees, while devices include computers, smartphones, routers, servers, controllers and sensors. The core objective of an IAM solution is to assign one digital identity to each individual or a device. Once that digital identity has been established, the IAM solution maintains, modifies, and monitors access levels and privileges through each user's or device's access life cycle.

In today's complex compute environments, IT departments are under increased regulatory and organizational pressure to protect access to corporate resources. As a result, they can no longer rely on manual and error-prone processes to assign and track user privileges. IAM automates these tasks and provides a seamless way to manage user identities and access all in one place. The core responsibilities of an IAM system are:

- Verification and authentication of users based on their roles and contextual information such as geography, time of day, or (trusted) networks
- Capturing and recording of user login events
- Managing and provision of visibility of the business's user identity database
- Managing the assignment and removal of users' access privileges
- Allowing system administrators to manage and restrict user access and monitor changes in user privileges

The adoption of an Identity Management system provides a wide range of benefits to organizations, such as:

- Secure access: access privileges are granted according to the selected policy, and all individuals and services are properly authenticated, authorized and audited
- Reduced risk: companies have greater control of user access, which reduces the risk of internal and external data breaches
- Ease of use: the use of an IAM framework can make it easier to enforce policies around user authentication, validation and privileges
- Reduced IT costs: businesses can operate more efficiently by decreasing the effort, time and money that would be required to manually manage access to their networks
- Meeting compliance: an effective IAM system facilitates businesses to confirm compliance with critical privacy regulations such as HIPAA, the Sarbanes-Oxley Act and GDPR

4.7.1 Keycloak

Keycloak [100] is an identity and access management solution (IAM) [101] [102] for web apps and RESTful web services. It has been developed by RedHat, one of the world's biggest Open Source software producers, and comprises all the important features an IAM solution needs to provide [103]. The goal of Keycloak is to facilitate application developers to secure the apps and services they have deployed in their organization by providing out of the box easily tailorable security features that developers normally have to implement on their own in order to meet individual requirements of their organization. In that context, Keycloak offers central management of admins, users, user rights, system access processes, password rules and passwords.

Figure 70 below provides a high-level description of how Keycloak works [104]:

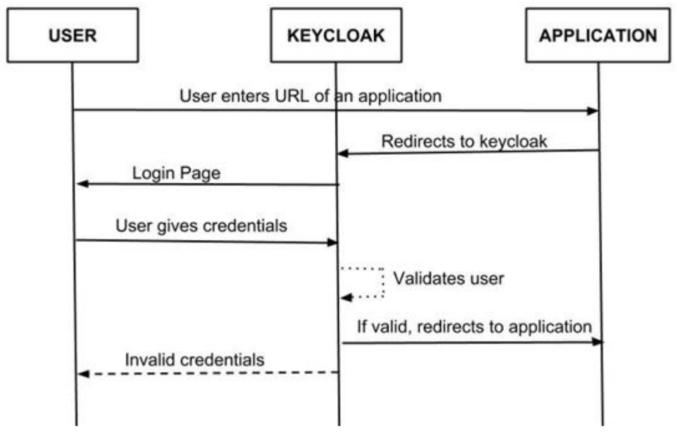


Figure 70: Keycloak Sequence Diagram

Browser applications redirect a user's browser from the application to the Keycloak authentication server where they enter their credentials. This redirection is important because users are completely isolated from applications and applications never see a user's credentials. Keycloak checks the validity of the provided user credentials and decides whether or not the user will be granted access to the requested protected URL(s) of the application.

The key benefits offered by Keycloak are the following [105] [106]:

- Multiple Protocols Support: Keycloak supports three different authentication protocols, namely OpenID Connect, OAuth 2.0 and SAML 2.0
- Authorization & Authentication: Keycloak supports system logon with one account or one single virtual identity
- Identity Brokering: Keycloak can serve as a proxy between the users of an application and external identity providers.

- SSO: Keycloak provides full support for Single Sign-On and Single Sign-Out
- Scalability: Keycloak is capable of managing a nearly limitless number of accounts, adaptable to different needs
- Social Identity Providers: Keycloak allows the use of Social Identity Providers, providing built-in support for Google, Facebook, Twitter, Stack Overflow and more
- Performance: Keycloak is powerful, future-proof and suitable for enterprise applications
- Active Community: there is an active community that provides continuous and customer-oriented development and keeps Keycloak up-to-date with regular releases
- Open Source: Keycloak is completely open source and free in contrast with most of the tools that offer similar features such as Okta and AuthO

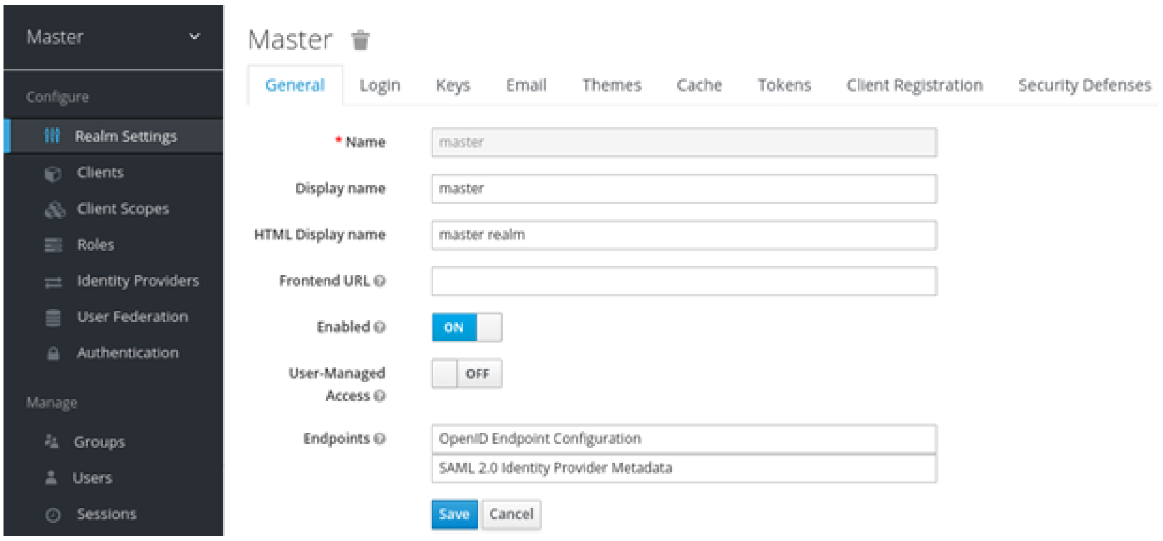
Additionally, Keycloak provides the Admin Console, an extensive and friendly interface which is accessible via a web browser, for administrators and developers to configure and manage Keycloak [107] [108]. There are three main entities in Keycloak:

1. realm: A realm secures and manages security metadata for a set of users, applications, and registered auth clients. Realms are fully isolated from one another and each realm has its own configuration and its own set of applications and users.

2. client: Clients are entities that can request authentication of a user within a realm. Every application that interacts with Keycloak is considered to be a client.

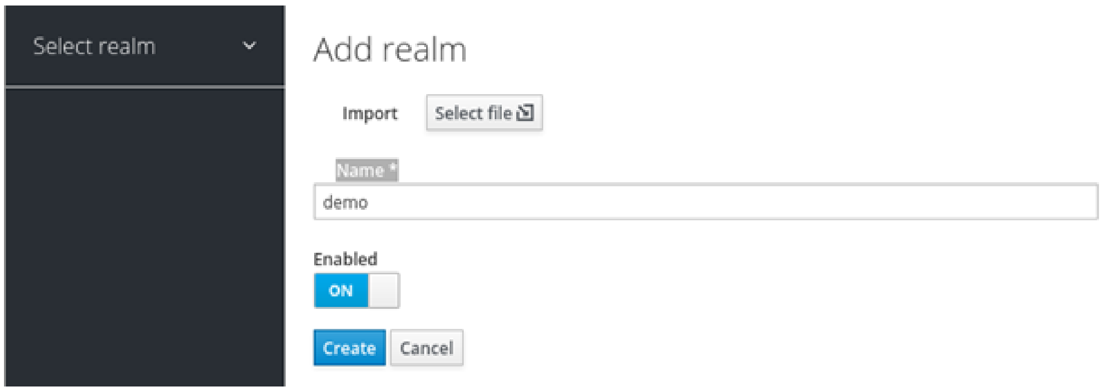
3. role: Roles identify a type or category of user. Keycloak often assigns access and permissions to specific roles rather than individual users for a fine-grained access control

In order to use the admin console, a system administrator will need to create an initial admin account first. This account will allow the admin to log into the *master* realm's administration console where he/she can start creating realms as well as users and applications for these realms. Figure 71 below depicts an overview of the master realm, while Figure 72 and Figure 73 show how to create a new realm from the master realm.



* Name	master
Display name	master
HTML Display name	master realm
Frontend URL	
Enabled	ON
User-Managed Access	OFF
Endpoints	OpenID Endpoint Configuration
	SAML 2.0 Identity Provider Metadata

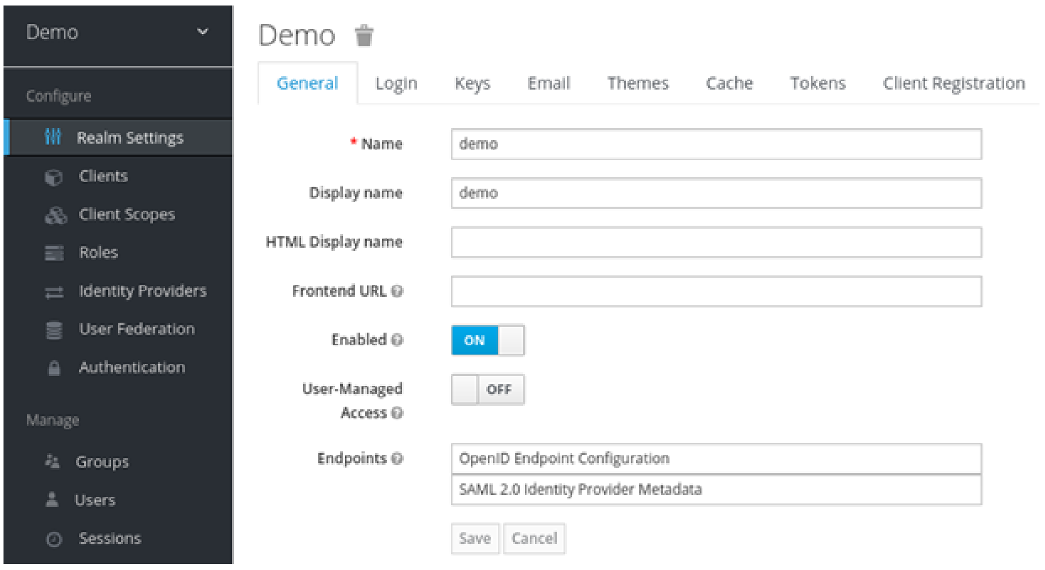
Figure 71: Keycloak Master Realm



Name: demo

Enabled: ON

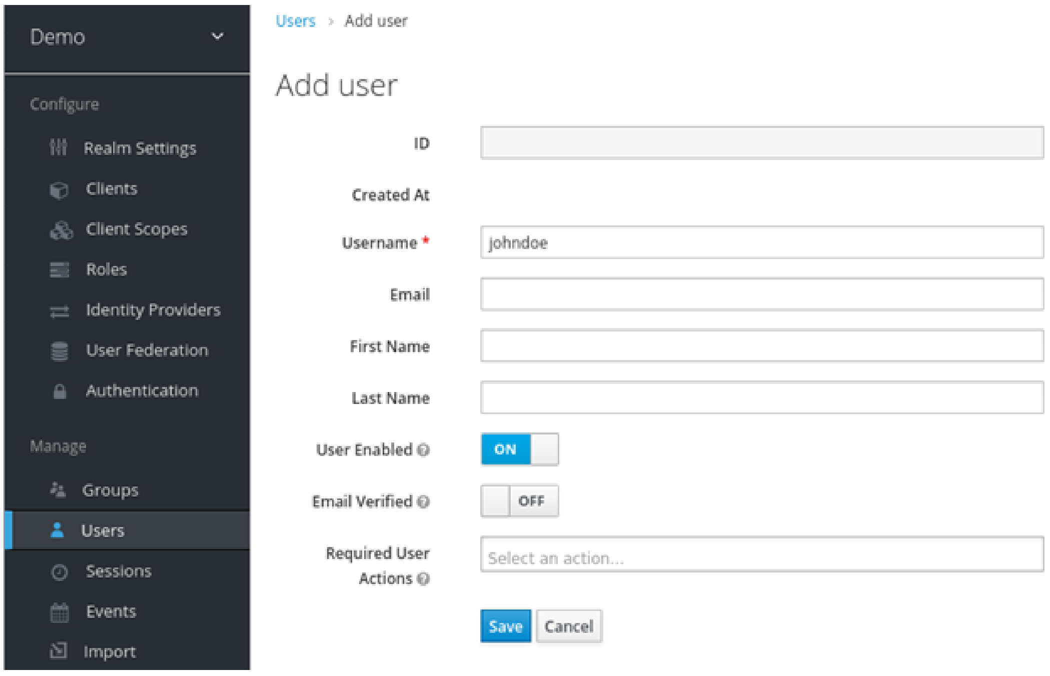
Figure 72: Creating a new realm (1/2)



* Name	demo
Display name	demo
HTML Display name	
Frontend URL	
Enabled	ON
User-Managed Access	OFF
Endpoints	OpenID Endpoint Configuration SAML 2.0 Identity Provider Metadata

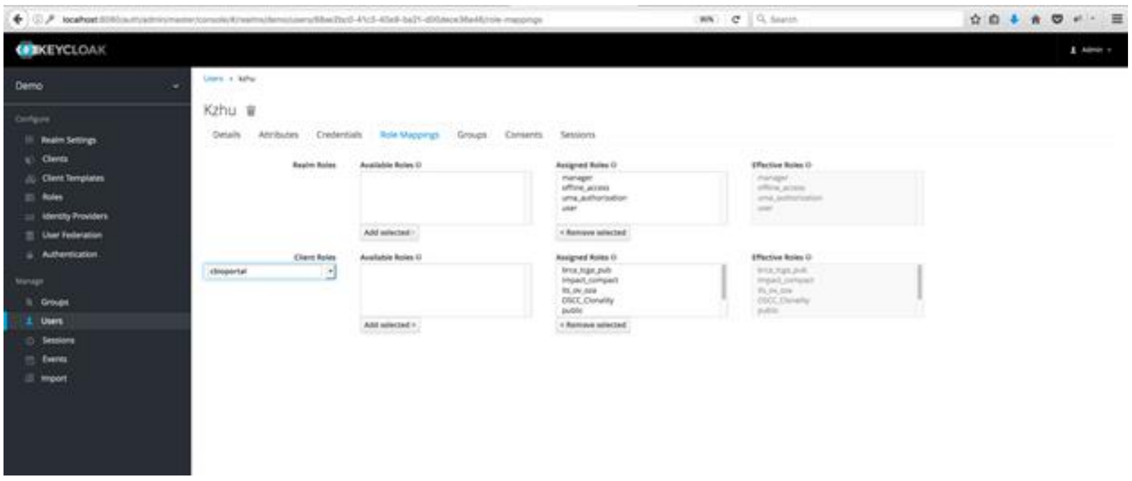
Figure 73: Creating a new realm (2/2)

As soon as the realm has been created, users can be added within that realm as shown in Figure 74. The admin can also define appropriate roles and assign them to the created users as it can be seen in Figure 75.



ID	
Created At	
Username *	johndoe
Email	
First Name	
Last Name	
User Enabled	ON
Email Verified	OFF
Required User Actions	Select an action...

Figure 74: New user



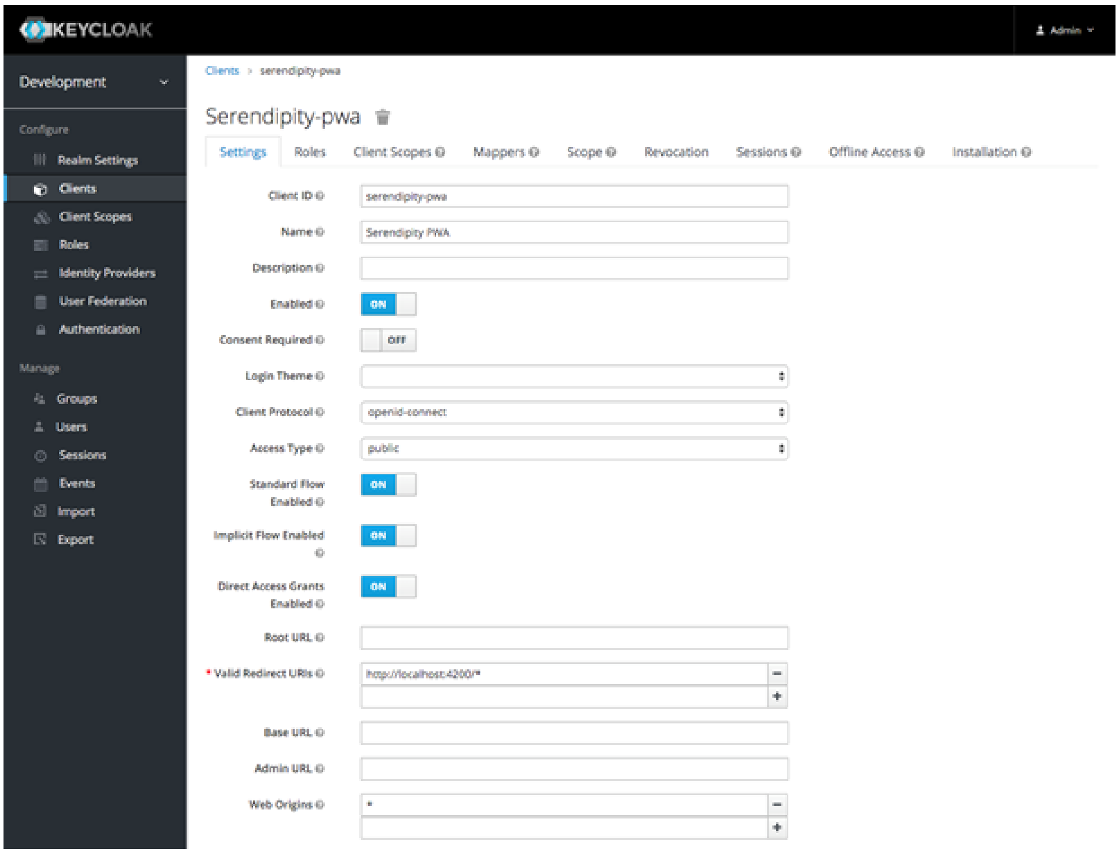
Kzhu

Realm Roles	Available Roles	Assigned Roles	Effective Roles
		manager offline_access uma_authorization user	manager offline_access uma_authorization user

Client Roles	Available Roles	Assigned Roles	Effective Roles
clinoportal		lmca_logo_pub impact_compact its_ov_ssa OSCC_Clonality public	lmca_logo_pub impact_compact its_ov_ssa OSCC_Clonality public

Figure 75: Role Assignment

Figure 76 presents how a sample application (client) that needs to be secured with Keycloak can be configured within a realm.



Serendipity-pwa

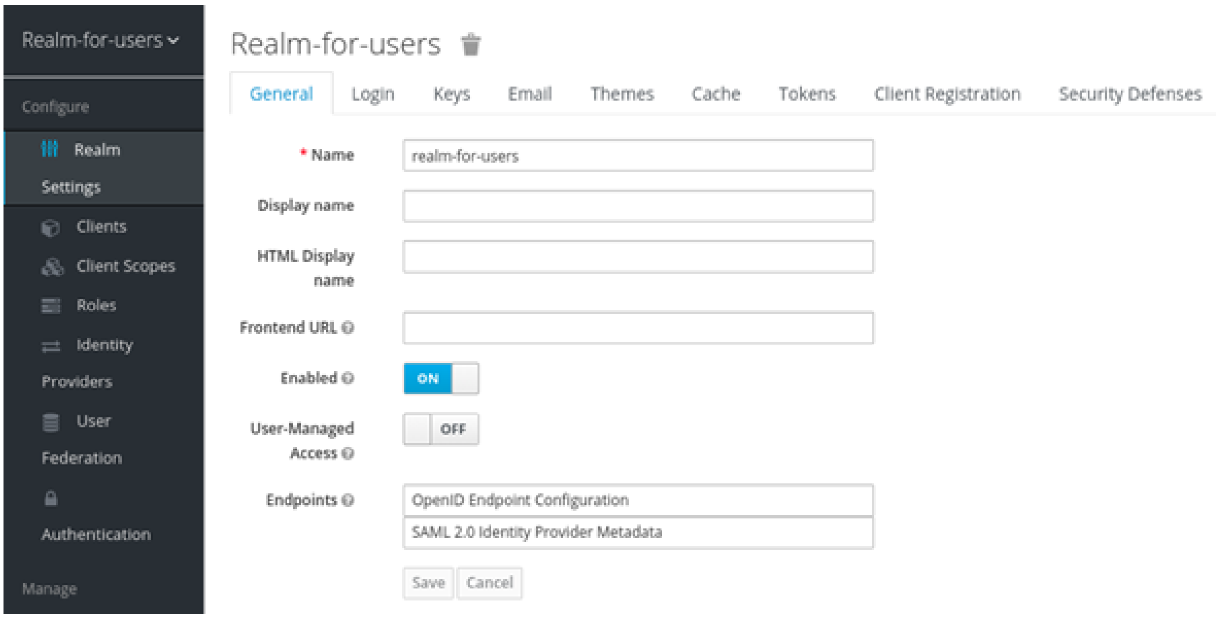
Client ID	serendipity-pwa
Name	Serendipity PWA
Description	
Enabled	ON
Consent Required	OFF
Login Theme	
Client Protocol	openid-connect
Access Type	public
Standard Flow Enabled	ON
Implicit Flow Enabled	ON
Direct Access Grants Enabled	ON
Root URL	
* Valid Redirect URIs	http://localhost:4200/*
Base URL	
Admin URL	
Web Origins	*

Figure 76: Client application configuration

Apart from the Admin Web Console, Keycloak also offers the Admin REST API which is capable of performing all the operating tasks that are necessary to manage the Keycloak system. In fact, the web console uses this REST API under the hood [109], therefore all the functionality and features of the web console are provided by the Keycloak Admin REST API as well. More information about this REST API can be found in https://www.keycloak.org/docs-api/14.0/rest-api/index.html.

4.7.2 Keyclock and Eclipse Che

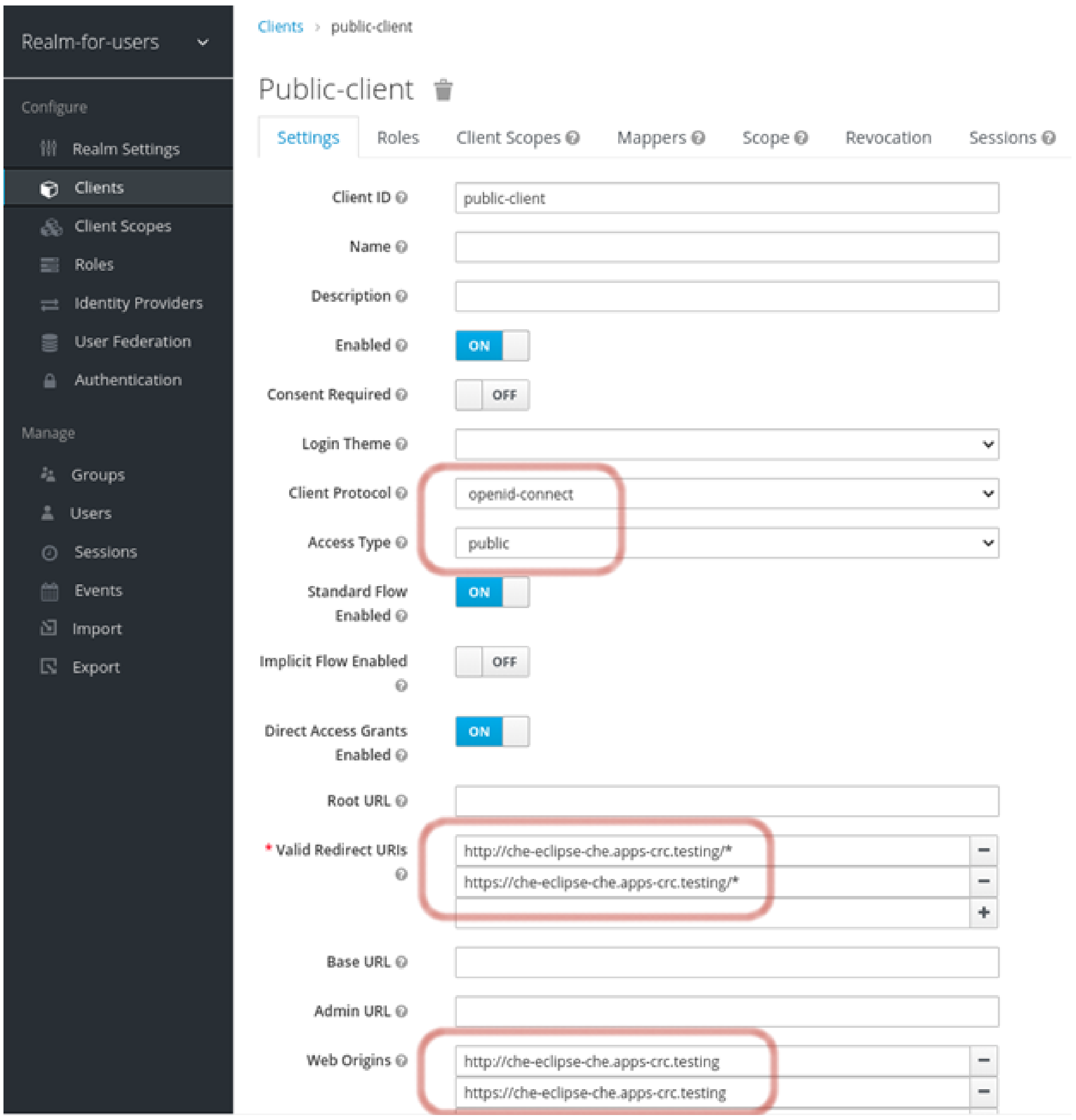
Eclipse Che utilizes by default the multiuser mode which uses Keycloak to authenticate users. Therefore, Eclipse Che can take advantage of all the capabilities Keycloak may offer in creating, importing, managing, deleting and authenticating users. By default, an Eclipse Che installation includes the deployment of a dedicated Keycloak instance. However, Che is capable of using an external Keycloak instance as well, which can be very useful in cases where there is an existing Keycloak instance with already-defined users, for example, a company-wide Keycloak server used by several applications. In that case, a realm containing the users intended to connect to Che should be defined in the external Keycloak server, as shown in Figure 77. In that realm, an OpenID Connect client needs to be defined that Che will use to authenticate the users. The Client Protocol must be set to openid-connect and the Access Type must be public, as it can be seen in Figure 78. Furthermore, the URIs that contain the base URL of the Che server must be defined.



Realm-for-users

* Name	realm-for-users
Display name	
HTML Display name	
Frontend URL	
Enabled	ON
User-Managed Access	OFF
Endpoints	OpenID Endpoint Configuration SAML 2.0 Identity Provider Metadata

Figure 77: Configuring Eclipse Che to work with an existing Keycloak server (1/2)



Public-client

Client ID	public-client
Name	
Description	
Enabled	ON
Consent Required	OFF
Login Theme	
Client Protocol	openid-connect
Access Type	public
Standard Flow Enabled	ON
Implicit Flow Enabled	OFF
Direct Access Grants Enabled	ON
Root URL	
* Valid Redirect URIs	http://che-eclipse-che.apps-crc.testing/*
	https://che-eclipse-che.apps-crc.testing/*
Base URL	
Admin URL	
Web Origins	http://che-eclipse-che.apps-crc.testing
	https://che-eclipse-che.apps-crc.testing

Figure 78: Configuring Eclipse Che to work with an existing Keycloak server (2/2)

4.8 Deployment [WT]

4.8.1 Design Approach

Deployment and deployment monitoring service. This is a very first approach where we will only consider: GitLab + “SmartCLIDE Interpreter + Jenkins + Docker + Kubernetes/AWS

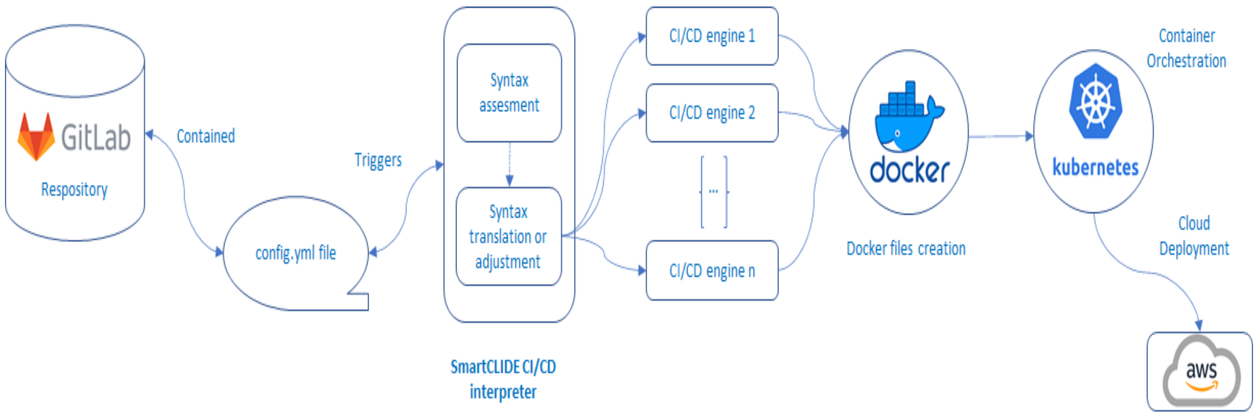


Figure 79: Set of Applications Diagram. Workflow

4.8.2 Third-party services

The deployment and deployment monitoring microservice makes use of the following elements to perform the deployment and monitoring tasks, from the gitlab-ci pipeline file until the deployment is monitored as it runs on the kubernetes cluster, be it on any cloud infrastructure.

4.8.2.1 Kairos interpreter

The Kairos interpreter is a microservice developed by kairos whose main function is, from a gitlab-ci file, to obtain a jenkins pipeline file.

4.8.2.2 Jenkins

Jenkins was chosen as the CD/CI engine since it is the main target of the Kairos interpreter as a CD/CI engine. In addition, since it is open source software, it can be deployed on the developer's machine in the development and testing tasks of the deployment microservice. As more and more organizations are using Docker to unify their build and test environments for their applications, Jenkins allows us to interact with Docker through default Docker support in its pipelines. On the other hand, Jenkins pipelines allow images to be built from the Dockerfile found in the main folder of the software project.

4.8.2.3 Docker registry

Docker Registry is an application that manages storing and delivering Docker container images. Registries centralize container images and reduce build times for developers. Docker images guarantee the same runtime environment through virtualization, but building an image can involve a significant time investment. Docker registry will be used as a central repository of images once they are built. It will be from this service from where the Kubernetes deployment will obtain the image of the containers to be deployed in the cluster.

4.8.2.4 Kubernetes cluster

Because Kubernetes is an open-source project, you can use it to run containerized applications in any environment without having to change your operational tools. A large community of volunteers maintains and improves Kubernetes software. In addition, many other vendors and open-source projects create and maintain Kubernetes-compatible software that you can use to enhance and extend your application infrastructure. The scope of the service also includes the use case of obtaining information about the status of the service while it is running in Kubernetes. Some of this data can be RAM memory in use, network information or CPU usage. It is assumed that a Kubernetes cluster is running on any of the clouds, such as AWS, Azure or Google Cloud Platform.

4.8.3 Workflow

4.8.3.1 Gitlab CI/CD pipeline to Jenkins job

For the early prototyping, it is assumed that the build stage and the push stage are done using docker, so the resulting jenkins pipeline should also be of these characteristics. It is also assumed that the software project to be deployed contains in its root directory a Dockerfile to build the docker image that will later be built with jenkins.

4.8.3.2 Using Jenkins to publish docker images to a docker registry server

With the Jenkins pipeline obtained from the previous step, a jenkins job is launched that builds the docker container. If the job is successful, the resulting image is uploaded to the docker registry server. This way the built image will be available to be consumed by the kubernetes cluster.

4.8.3.3 Deploying applications on kubernetes from docker images

Once the jenkins job has successfully completed and the image has been uploaded to the docker image registry, the next step is to deploy the built image(s) to the Jenkins cluster.

4.8.3.4 Kubernetes monitoring service

The scope of the deployment service also includes the use case of obtaining information about the status of the service while it is running in Kubernetes. Some of this data can be RAM memory in use, network information or CPU usage.

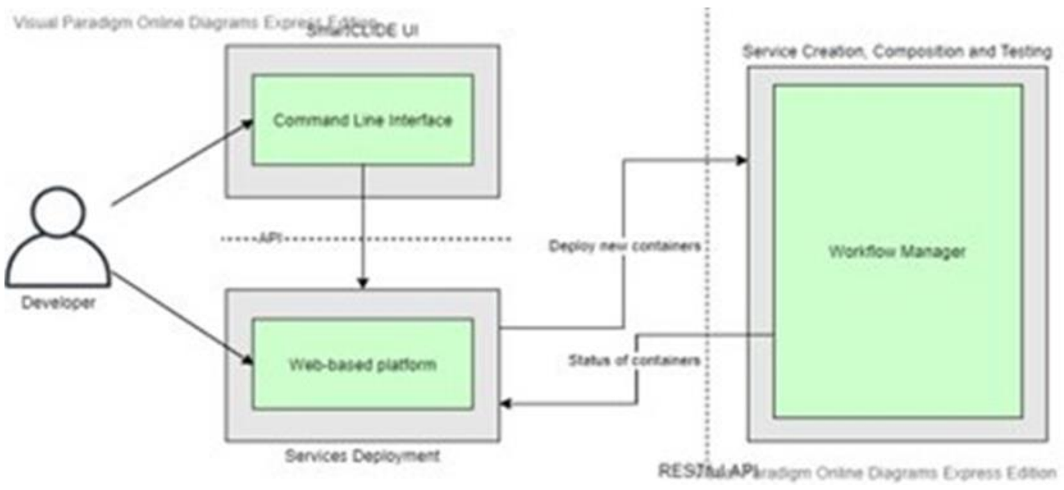


Figure 2: Deployment Component Diagram

4.8.4 Docker-based deployment

Services can be deployed as isolated instances using a well-known technology such as dockers. Dockers is an open-source project that automates the deployment of software applications inside containers by providing an additional layer of abstraction and automation of OS-level virtualization on Linux. Opposite to the Virtual Machine paradigm, containerized applications do execute over the host operating system, just using a docker layer between.

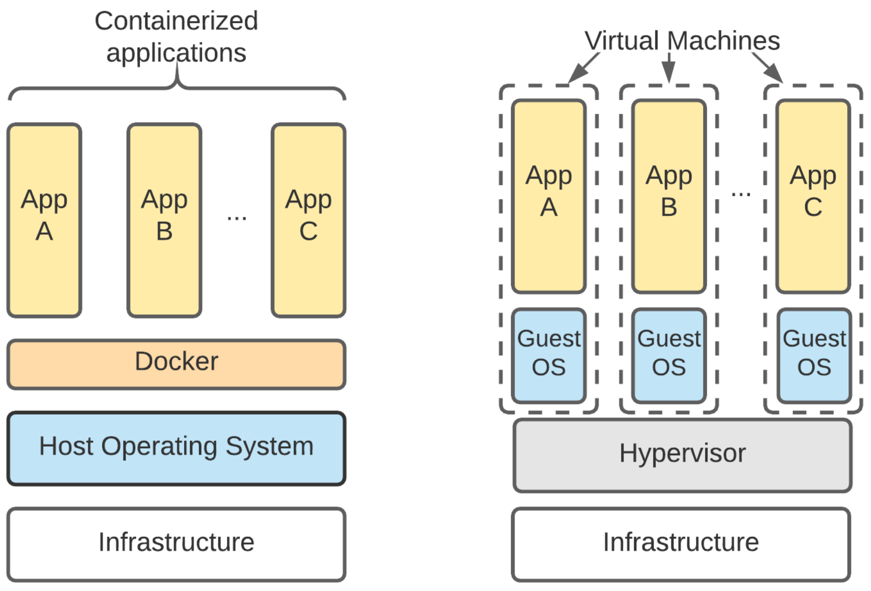


Figure 3: Docker based Deployment Architecture

4.8.4.1 Communication with Cloud Providers

Application integration on AWS is a set of services that enable communication between decoupled components in microservices, distributed systems, and serverless applications. Decoupling applications at any scale can reduce the impact of changes, making updates easier and new deployments and new features released faster.

4.9 SmartCLIDE CI/CD

4.9.1 Design Approach

The proposed basis for the SmartCLIDE CI/CD infrastructure is the built-in CI/CD capability provided natively by the chosen version control system, GitLab. For the Continuous Integration (CI) component of this, there are several areas to consider:

- What elements of the system are subject to CI
- How CI integrates with the development strategy

When considered at the level of individual services, CI is a requirement for those services which are defined within the SmartCLIDE source code repository, i.e., services which are developed from scratch, modified from a template or generated as source code with SmartCLIDE tooling. CI on GitLab is generally configured by means of a configuration file, namely the *gitlab-ci.yml* file, at the root of the corresponding source repository. Within this configuration file, the various stages of the build pipeline are defined. A build pipeline might typically involve the following stages:

- Build – compile the code in the repository
- Unit test – run unit tests
- Package – package the service into a deployable unit
- Integration test – run integration tests
- Deploy – deploy to an environment

Note that the stages of the build pipeline are flexible and may be defined according to the needs of the project and the technology used. For instance, pipeline stages for a service written in Java would invoke

Maven[39] for compilation and unit testing, whereas a service written in Ruby[40] would require invocation of the Rake[41] build tool. In the case of SmartCLIDE, the end-result of a successful build is a Docker image definition, ready to be used in the BPMN-defined workflows.

For Continuous Delivery (CD) at the workflow definition level, the flexibility afforded by the GitLab CD functionality, with built-in support for Docker and Kubernetes deployments and the ability to run arbitrary scripts, may serve as the basis for deployment of the composed service, as described in Section 2.4 of this document.

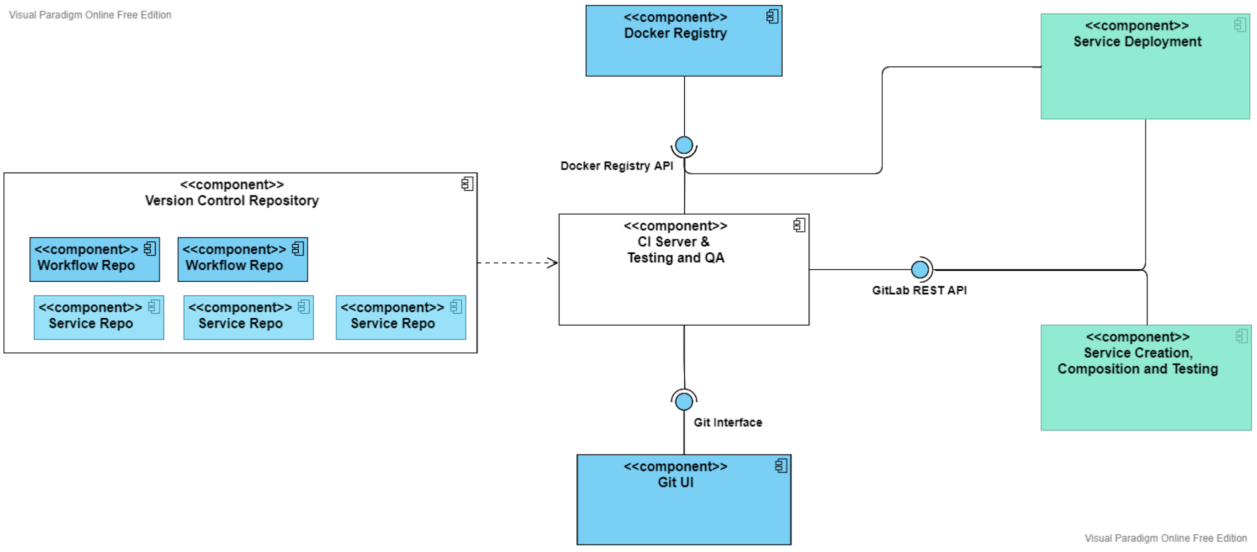

Visual Paradigm Online Free Edition
<<component>> Docker Registry
<<component>> Service Deployment
Docker Registry API
<<component>> Version Control Repository
<<component>> Workflow Repo
<<component>> Workflow Repo
<<component>> Service Repo
<<component>> Service Repo
<<component>> Service Repo
<<component>> CI Server & Testing and QA
GitLab REST API
<<component>> Service Creation, Composition and Testing
Git Interface
<<component>> Git UI
Visual Paradigm Online Free Edition

Figure 80: CI Server & Testing and QA Component Diagram

Table 48: CI Server & Testing

Name	**CI Server & testing Component**
Functionality	Perform automated build, test, and packaging of services from source code.
Relevant Use Cases (D1.3)	Provide basis for Services Deployment component
Functional Requirements	P69, P70, P72

[39] https://maven.apache.org/
[40] https://www.ruby-lang.org/en/
[41] https://github.com/ruby/rake

4.9.2 Interface Specification

Table 49: CI Server & Testing and QA Component Interface Specification

No	Interface (/API)	Description	Type	
			Provided	Required
1	GitLab REST API	GitLab's REST API provides a rich set of capabilities for controlling GitLab. Full documentation can be found at https://docs.gitlab.com/ee/api/api_resources.html	*	
2	Git interface	The standard Git interface, over ssh or https connection, accessible with git client or git library built into SmartCLIDE components. The configuration file, .gitlab-ci.yml, is included in the directory root of a project to provide configuration of the CI/CD pipeline.		*

5 SmartCLIDE Testing Approach

5.1 Test levels

Testing and verification in SmartCLIDE will be conducted in multiple levels namely unit testing, integration testing, API testing, performance testing, usability testing and system testing.

In the unit testing level [110], the individual components of the application are tested. The aim is to ensure that each component works as expected. Unit testing requires a detailed knowledge of the source code and are executed during the development (coding phase) of an application by the developers. Unit tests isolate a section of code and verify its correctness. This section of code is called unit and can be an individual method, function, procedure, object or module. Unit tests are in general quite cheap to automate, and errors found at this level can be fixed quickly. Moreover, unit testing results in more readable and less complex source code and simplifies the integration of a unit into an application.

Integration testing [111] is a type of testing meant to check interconnection and interoperability between the different software components, ensuring that the integrated units operate well together. Integration tests normally and logically follow unit tests and focus on exposing defects at the time of interaction between the different software modules when they are integrated and data/information flow among them. Integration testing can be very useful especially when requirements can be changed or enhanced at the time of module development. In such cases, these new requirements may not be tested appropriately at the unit testing level and create problems in the communication between the system components. Integration tests can prevent such flaws and verify that the different modules work properly as a unit. In the context of SmartCLIDE, integration testing will play a vital role in the software development process since there are many different components that need to communicate with each other.

Application Programming Interface (API) testing [112] is part of the integration testing and is performed in the business layer where all the business logic processing is carried out. Each SmartCLIDE component will have its own API to interact and communicate with the other components. This API will define what requests can be made, which data formats can be used, what kind of responses are expected etc. for each component. API testing will ensure that all the APIs meet the required expectations regarding functionality, performance, reliability and security. An API that is not tested properly may cause problems not only to the primary application that exposes that API but also to other applications it integrates with.

Smoke tests [113] will run right after a new software build is deployed to assure that the core features of the system are still working properly. Smoke tests run quickly, are relatively easy to perform and help identifying flaws in early stages. Furthermore, smoke tests can save test effort and time by detecting early defects in a software product that would make the use of further and more expensive tests a waste of time and resources.

Performance tests [114] will be conducted to check how the system behaves under different workloads. This testing method is mostly used to identify performance issues and bottlenecks and determines the system performance in terms of speed, stability, reliability and scalability. Performance testing can be either quantitative or qualitative and can be divided into different sub-types such as Load testing and Stress testing. Load testing is carried out to check how an application behaves under anticipated user loads and is usually performed with the help of automated testing tools that simulate real-world usage. It intends to detect errors that can occur under different load variations and provides an estimation of the maximum capacity of the system before performance is affected, thus providing valuable insight into performance bottlenecks which can improve the scalability of the system. Stress testing is a software testing technique that verifies the stability and reliability of the system. It emphasizes on robustness and checks the system behaviour under abnormal conditions, e.g., by applying loads beyond the actual load limit or taking away some of the resources, to identify the breaking point. Stress testing ensures that the system can operate in an appropriate way in abnormal conditions and recover quickly after a failure.

Usability testing [115] refers to evaluating a product or service by testing it with representative users. It usually involves observing users as they attempt to complete typical tasks and mainly focuses on the effectiveness, efficiency, accuracy and user friendliness of the system. Through usability testing, the design and development teams can evaluate how satisfied the participants are with the system and identify changes required to improve user performance and satisfaction. Usability tests are often conducted repeatedly, from early development until a product's release.

The testing of the overall SmartCLIDE platform in the acceptance testing level will be performed after the completion of the overall system integration, according to the defined acceptance level test cases.

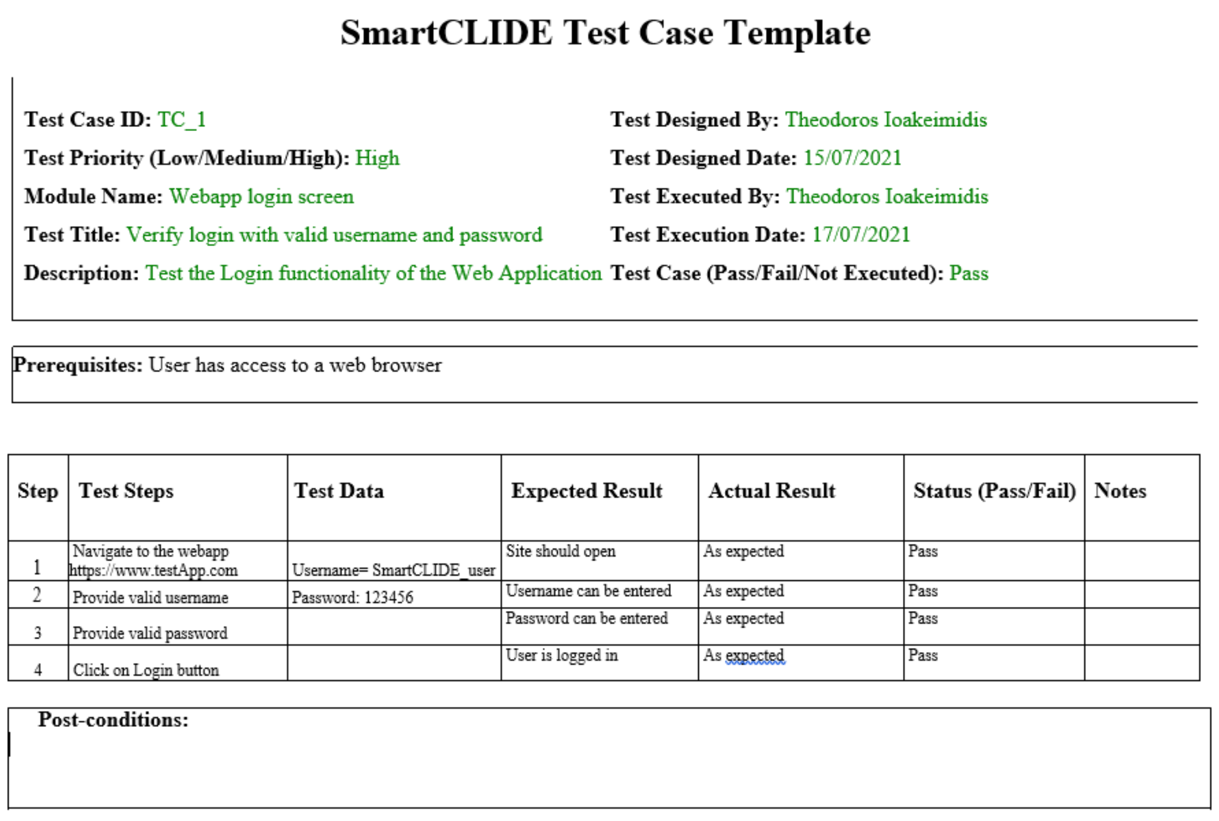
5.2 Testing tools and frameworks

In order to technically design and implement the several test cases, the SmartCLIDE development teams utilized a set of tools and frameworks. One popular testing tool used by the development teams is Junit [116]. Junit is one of the most popular unit-testing frameworks in the Java ecosystem. It is open source and helps Java developers to write and run repeatable tests. JUnit offers several functionalities for writing and executing high-quality unit tests like annotations to identify test methods, assertions for testing expected results and test runners for running tests. JUnit is supported by popular IDEs and build tools and it is elegantly simple, enabling developers to write codes faster. Another testing tool that is often utilized in conjunction with Junit is the Maven Surefire Plugin [117] which is used during the test phase of the build lifecycle to execute the unit tests of an application. This plugin generates test reports in two different file formats, plain text files (*.txt) and XML files (*.xml).

An additional tool that was utilized by the SmartCLIDE development teams is PlUnit [118]. It is a Prolog unit-test framework initially developed for SWI-Prolog. Tests are written in pure Prolog and can be embedded inside a normal source module or be placed in a separate test-file that loads the files to be tested.

5.3 Testing Documentation Template

In the context of the SmartCLIDE verification and testing strategy, a common format has been defined for reporting the several test cases of the SmartCLIDE components. This format is entitled SmartCLIDE Test Case Template and the figure below presents a test case example reported with this template.



SmartCLIDE Test Case Template

Test Case ID: TC_1	**Test Designed By:** Theodoros Ioakeimidis
Test Priority (Low/Medium/High): High	**Test Designed Date:** 15/07/2021
Module Name: Webapp login screen	**Test Executed By:** Theodoros Ioakeimidis
Test Title: Verify login with valid username and password	**Test Execution Date:** 17/07/2021
Description: Test the Login functionality of the Web Application	**Test Case (Pass/Fail/Not Executed):** Pass

Prerequisites: User has access to a web browser

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Navigate to the webapp https://www.testApp.com	Username= SmartCLIDE_user	Site should open	As expected	Pass	
2	Provide valid username	Password: 123456	Username can be entered	As expected	Pass	
3	Provide valid password		Password can be entered	As expected	Pass	
4	Click on Login button		User is logged in	As expected	Pass	

Post-conditions:

Figure 81: Example test case reported using the SmartCLIDE Test Case Template

As shown in the figure 1 above, the SmartCLIDE Test Case Template includes the following fields:

- Test Case ID: The ID of the test
- Test Priority: The priority of the test
- Module Name: The name of the module that is being tested
- Test Title: The title of the test
- Description: A short textual description of the test
- Test Designed By: The name of the tester/developer who created the test
- Test Designed Date: The date the test was created
- Test Executed By: The name of the tester/developer who executed the test
- Test execution Date: The date the test was executed
- Test Case: The test result
- Prerequisites: The software/hardware prerequisites for the correct execution of the test
- Step #: The serial number of the step
- Test Step: A short textual description of the test step
- Test Data: The data required for the correct execution of the test
- Expected Result: The expected result from the execution of the test step
- Actual Result: The actual result obtained from the execution of the test step
- Status: The test step status
- Notes: Any additional notes regarding the execution of test steps
- Post-conditions: The changes to the system after the completion of the Test Case

6 Conclusions

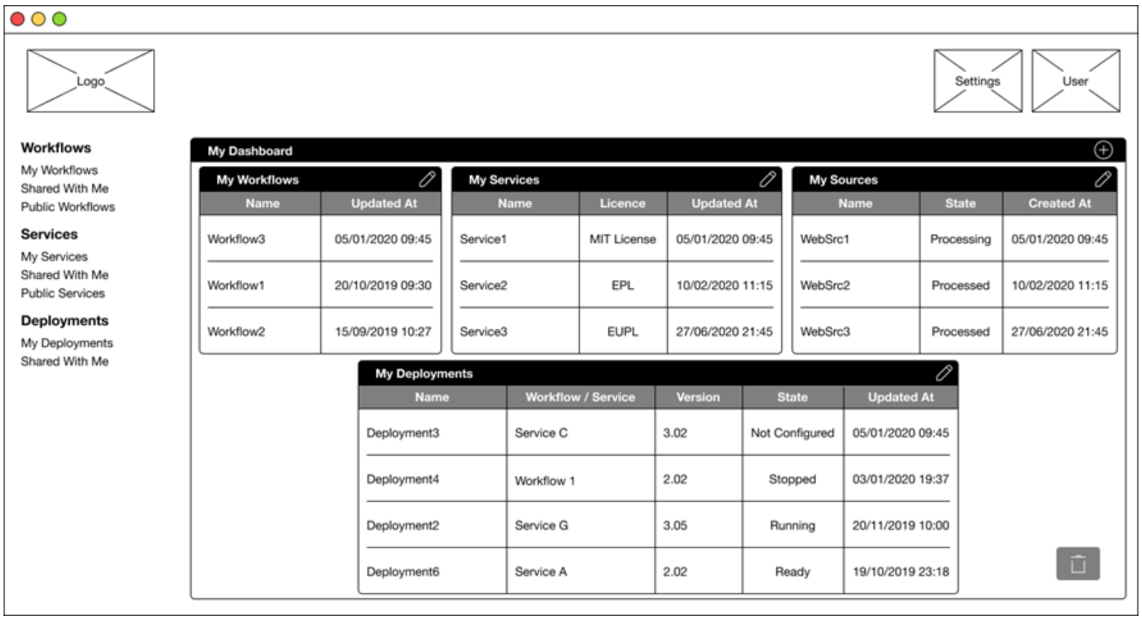
This document presented early SmartCLIDE frontend and backend design approach. The adopted early design approach followed the conceptualization and architecture design outcomes conducted in WP1 and documented in D1.2, D1.3, D1.4, and D1.5. The work was organized into two parallel branches: (i) user interface design, and (ii) elaboration of backend and frontend components.

An end-to-end demonstration use case was presented to explain how the various SmartCLIDE functionalities can be accessed from the envisioned SmartCLIDE IDE. Besides, the overall UI design approach and key implementation decisions are discussed. This deliverable formalizes the early design of SmartCLIDE components, focusing on specification of the core frontend (i.e., *Run-time Monitoring and Simulation Console* and *Smart Assistant*), and backend components (i.e., *User Access & Management*, *Discovery of Services and Resources*, *Service Creation*, *Composition and Testing*, *Discovery of Services and Resources*, *Run-time Monitoring and Verification*, *Security*, *MOM*, *Deployment*, and *CI/CD component*). All component diagrams are specified using UML 2.5 Component Diagram notation and clearly indicate provided and required APIs.

The next update of this document is planned for the M30. Hence, each technical partner will be responsible for an additional in more-detail description of the components. We expect that the proposed specification approach will provide efficient support for the ongoing development and integration effort. However, minor refinements might be necessary to properly document and formalized interface specifications (e.g., elaborated API documentation template). Finally, in the next version of this deliverable, we will focus on the detailed specification and the System Deployment view update.

7 Annex A: Wireframes of the IDE

The first page that the user sees when they enter the IDE is their homepage (Figure 82). It consists of a dashboard that presents a summary of information regarding the state of activities related with services, deployments, or other user-related processes monitored by the IDE. Additionally, the context bar provides direct access to the main pages of the workflows, services, or deployments. This page can be accessed by clicking the logo on the upper-left corner of the page.



My Workflows

Name	Updated At
Workflow3	05/01/2020 09:45
Workflow1	20/10/2019 09:30
Workflow2	15/09/2019 10:27

My Services

Name	Licence	Updated At
Service1	MIT License	05/01/2020 09:45
Service2	EPL	10/02/2020 11:15
Service3	EUPL	27/06/2020 21:45

My Sources

Name	State	Created At
WebSrc1	Processing	05/01/2020 09:45
WebSrc2	Processed	10/02/2020 11:15
WebSrc3	Processed	27/06/2020 21:45

My Deployments

Name	Workflow / Service	Version	State	Updated At
Deployment3	Service C	3.02	Not Configured	05/01/2020 09:45
Deployment4	Workflow 1	2.02	Stopped	03/01/2020 19:37
Deployment2	Service G	3.05	Running	20/11/2019 10:00
Deployment6	Service A	2.02	Ready	19/10/2019 23:18

Figure 82: The main page of the SmartCLIDE platform

The dashboard consists of a customisable area with resizable cards that provide an overview of the user's work. The plus button on the upper-right corner redirects the developer to the configuration page Figure 83 where a new card can be added to the dashboard. Dragging a card and dropping it in the bin on the lower-right corner removes it from the dashboard.

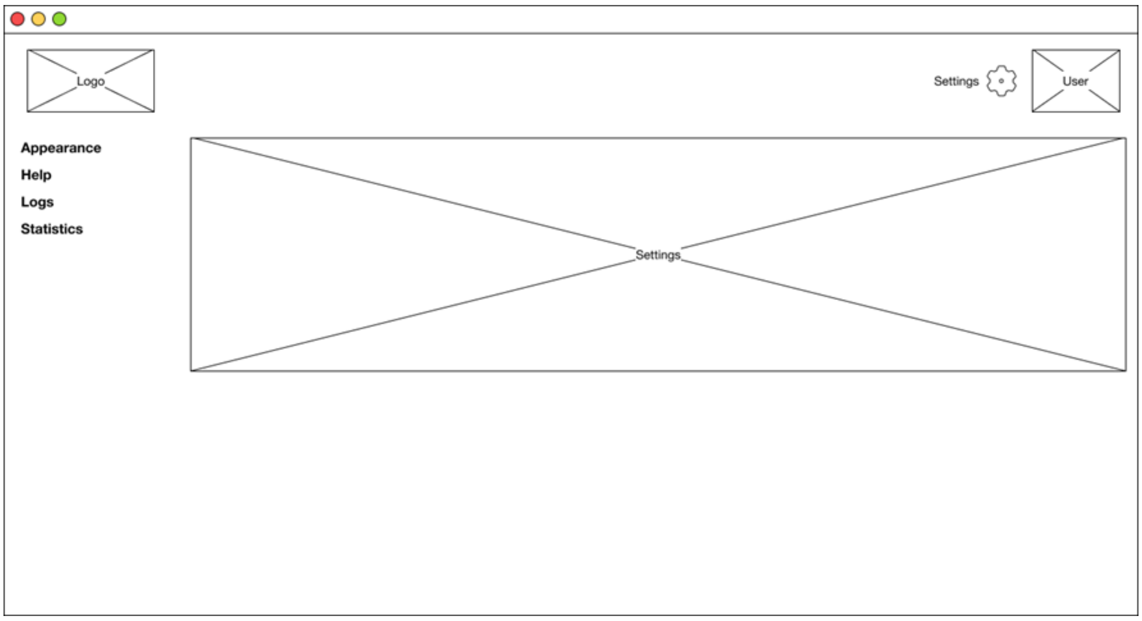


Figure 83: Settings Page

The page header contains two icons, on the upper-right corner: (i) “Settings” and (ii) “User”. When the settings icon is clicked, the user is presented IDE settings-related options, in the context bar (Figure 83):

- Appearance: Allows the user to configure SmartCLIDE IDE’s graphical theme, i.e., the graphical element’s colour and shape.
- Help: Contains the documentation of the IDE and a FAQ section.
- Logs: Shows the logs associated to the running of the tools in the SmartCLIDE ecosystem.
- Statistics: Provides resource usage statistics of the SmartCLIDE IDE.

When adding a card, the user is prompted to insert its category (Workflows, Services, Deployments and Sources), which defines the set of columns that can be viewed. Thus, the user must order the columns according to their priority because, if the card is set to a small size, not all columns will be displayed.

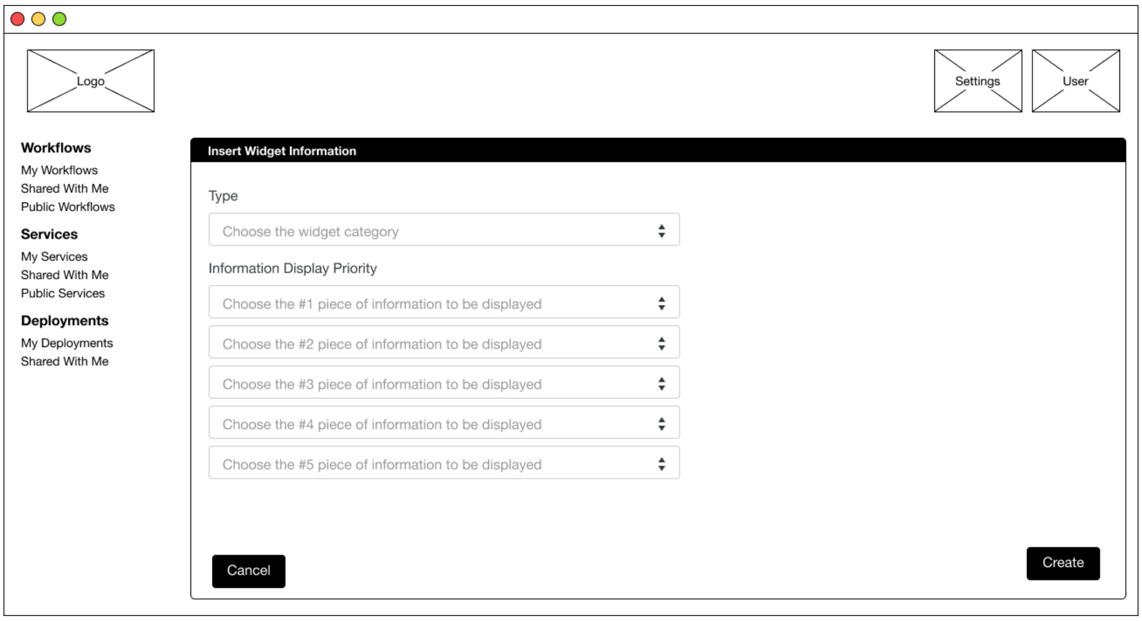
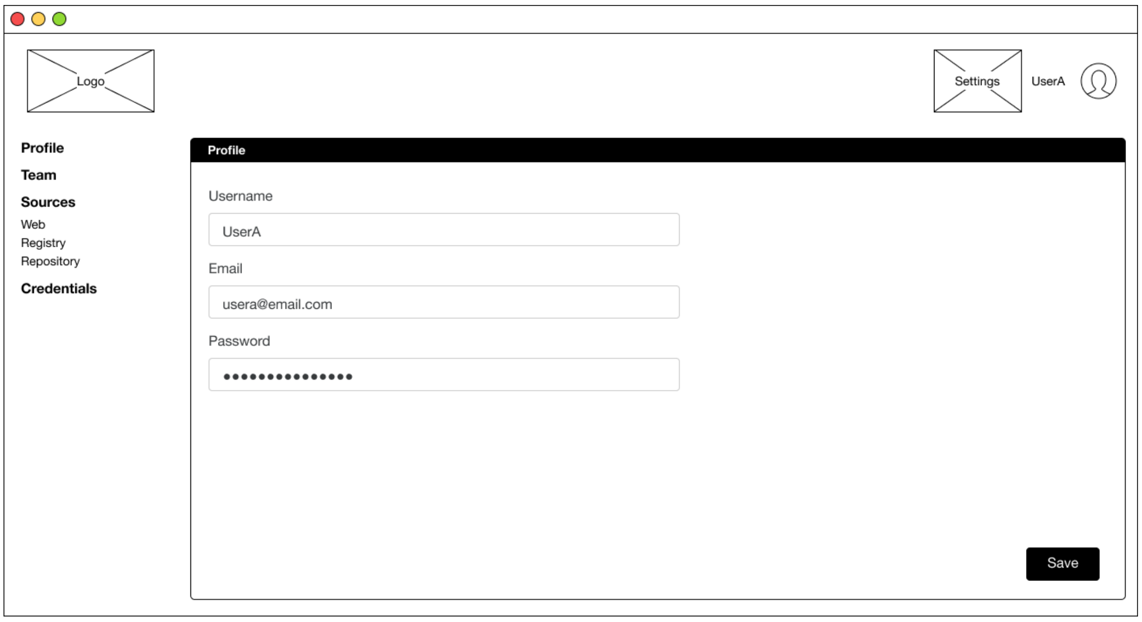

Logo
Settings
User
Workflows
My Workflows
Shared With Me
Public Workflows
Services
My Services
Shared With Me
Public Services
Deployments
My Deployments
Shared With Me
Insert Widget Information
Type
Choose the widget category
Information Display Priority
Choose the #1 piece of information to be displayed
Choose the #2 piece of information to be displayed
Choose the #3 piece of information to be displayed
Choose the #4 piece of information to be displayed
Choose the #5 piece of information to be displayed
Cancel
Create

Figure 84: The content configuration page of the dashboard

On the other hand, clicking the avatar icon makes the context bar display the account-related options:

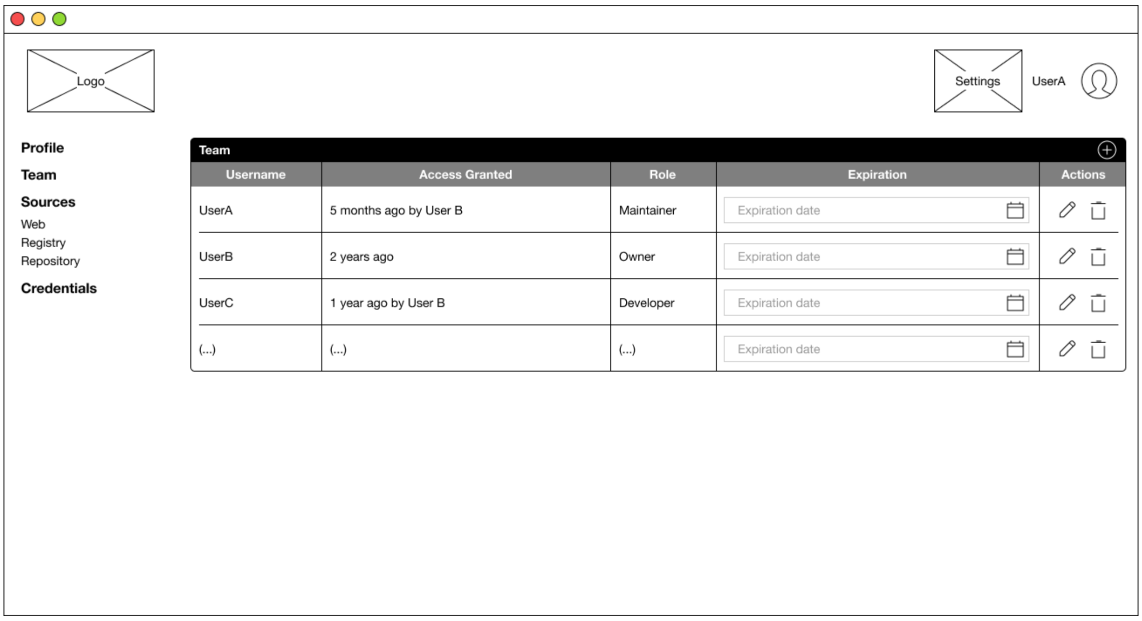
- **Profile:** page where the following profile settings can be changed (Figure 85):
  - Username: the name by which the user is known within the SmartCLIDE IDE
  - Email: the email address of the user
  - Password: the password used to log into the IDE



Username
UserA
Email
usera@email.com
Password
•••••••••••••••

Figure 85: The user profile page

- **Team:** Within the SmartCLIDE IDE, a team is a group of users that have access to the same projects, although with different permission levels. To become part of a team, the username inserted by the team owner must exist. In the page represented in Figure **86**, the user can manage its team members and access the following team-related information:
  - Username: the name by which the user is known within the SmartCLIDE IDE
  - Access Granted: when was the user granted access to the team and by who
  - Role: the role of the user within the team
  - Expiration: when will the user stop being part of the team



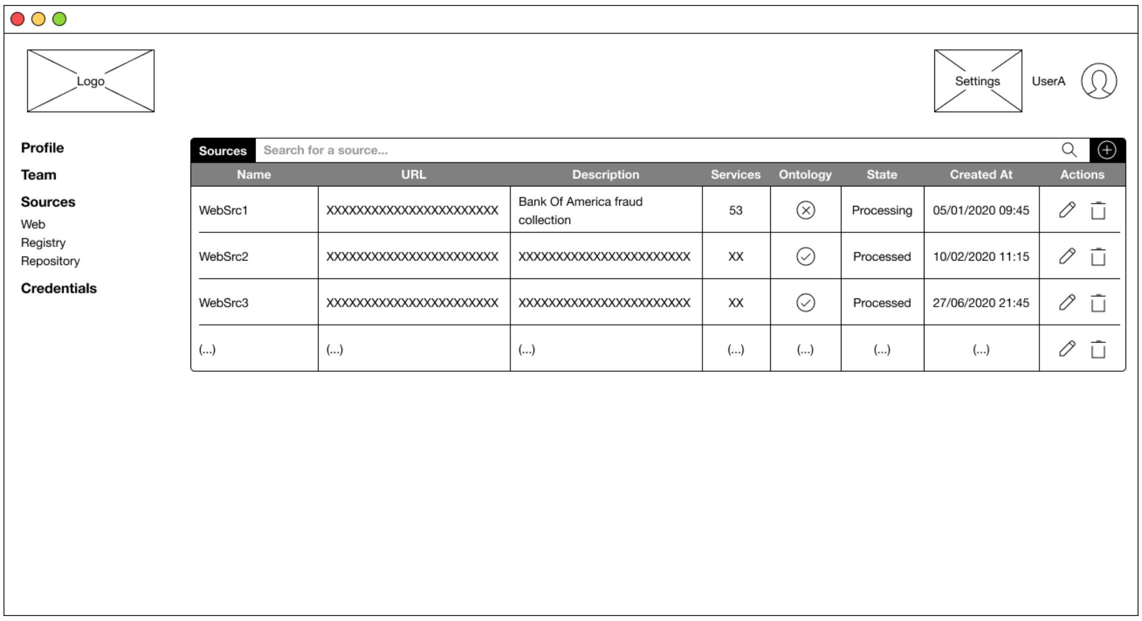
Username	Access Granted	Role	Expiration	Actions
UserA	5 months ago by User B	Maintainer	Expiration date	
UserB	2 years ago	Owner	Expiration date	
UserC	1 year ago by User B	Developer	Expiration date	
(...)	(...)	(...)	Expiration date	

Figure 86: The team configuration page

- **Sources:** Contains an overview of the sources where the IDE looks for services to be used and allows their management, which includes searching for, adding, editing, removing, and checking their status. According to **Figure 87**, this page displays the following information:
  - Name: name of the source
  - URL: web address of the source
  - Description: short description of the source
  - Services: number of services contained within the source
  - Ontology: indicates whether an ontology is available in the source
  - State: indicates whether the IDE has finished indexing the services from the source
  - Created At: when the source was added to SmartCLIDE

  This functionality supports the following use cases [119]:

  - UC-0004 Discover Resources and Services
  - UC-0028 Indexation and Classification of services and resources from an external repository



Name	URL	Description	Services	Ontology	State	Created At	Actions
WebSrc1	XXXXXXXXXXXXXXXXXXXXXXX	Bank Of America fraud collection	53	⊗	Processing	05/01/2020 09:45	
WebSrc2	XXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXX	XX	✓	Processed	10/02/2020 11:15	
WebSrc3	XXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXX	XX	✓	Processed	27/06/2020 21:45	
(...)	(...)	(...)	(...)	(...)	(...)	(...)	

Figure 87: Sources for services

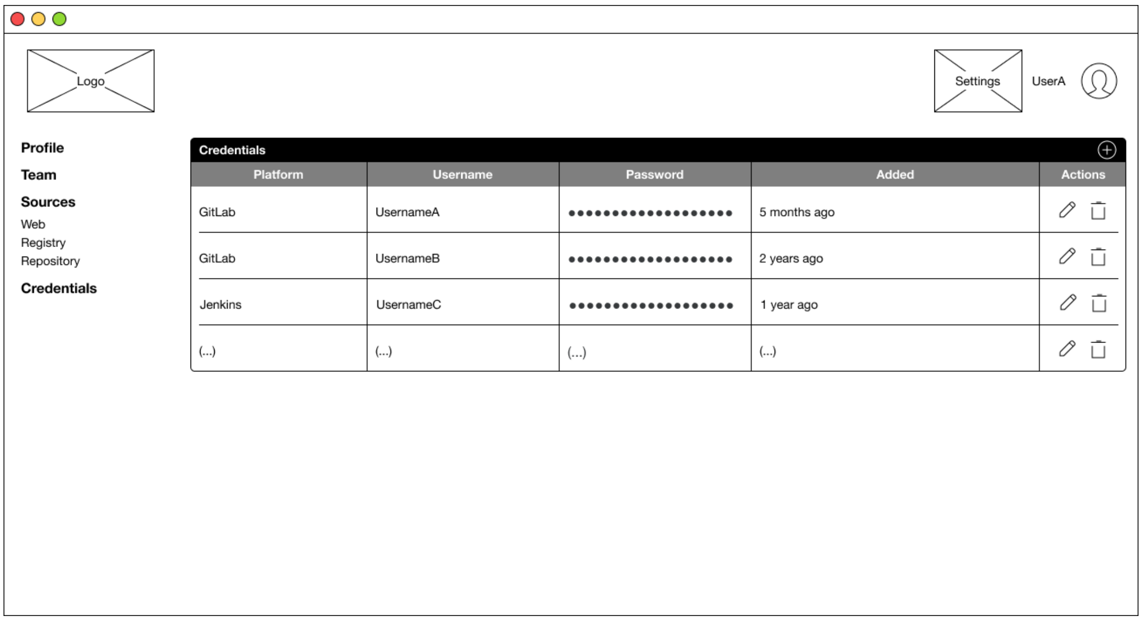
1 **Credentials:** Allows the user to manage their credentials for accessing development support tools like GitLab and Jenkins[42], which includes adding, editing, and removing credentials. Figure **88** shows the page that allows the user to manage such credentials and provides the following information:

- Platform: the platform where the credentials are to be used
- Username: the username to log in the platform
- Password: the password to log in the platform
- Added: date when the credentials were registered in the SmartCLIDE IDE

This functionality supports the use cases below [119]:

- UC-0001 Creation of a service from a template
- UC-0002 Create services with data abstraction levels
- UC-0003 Create and deploy a service from the IDE
- UC-0006 A non-expert user creates a new service with assistance
- UC-0010 Deploy a service from the CLI within SmartCLIDE
- UC-0011 Create a system using low-code programming
- UC-0012 Creating a complex scenario from templates
- UC-0015 Accessing Git repositories
- UC-0016 Change something on SmartCLIDE itself

[42] https://jenkins.io



Credentials

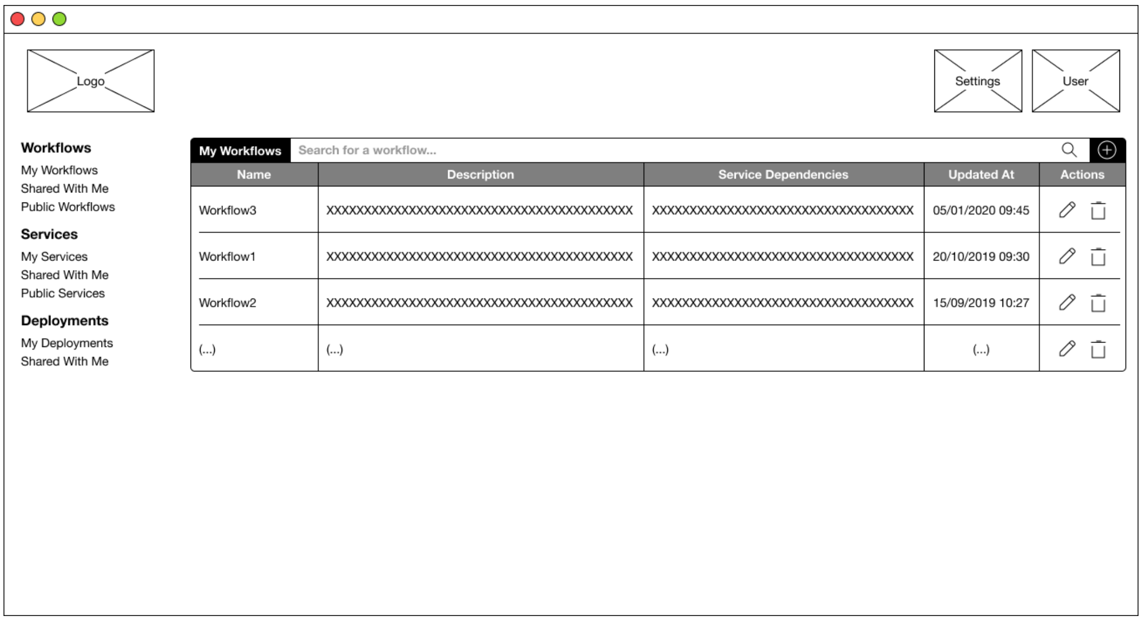
Platform	Username	Password	Added	Actions
GitLab	UsernameA	●●●●●●●●●●●●●●●●●●	5 months ago	
GitLab	UsernameB	●●●●●●●●●●●●●●●●●●	2 years ago	
Jenkins	UsernameC	●●●●●●●●●●●●●●●●●●	1 year ago	
(...)	(...)	(...)	(...)	

Figure 88: User credentials management page

A.1 Search for, add, edit, or remove workflows [Workflow Functionality]

Figure **89** shows the main page for workflows. Through the context bar, the user can choose to display one out of three groups of workflows: (i) “My workflows”, (ii) “Shared with me”, and (iii) “Public Workflows”. These groups allow the developer to filter the workflows being shown in the table (i.e., only workflows developed by the logged in user, workflows that were shared with the user or public workflows developed within the SmartCLIDE platform). The results can also be filtered by any keyword (name or description) or value (update date) that the user inserts into the search bar. In addition, the sorting order of the results may be swapped by clicking the titles of the columns.

This page is the starting point for creating, editing, or removing workflows, by clicking the plus, pencil or bin symbols, respectively.



My Workflows | Search for a workflow...

Name	Description	Service Dependencies	Updated At	Actions
Workflow3	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	05/01/2020 09:45	
Workflow1	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	20/10/2019 09:30	
Workflow2	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	15/09/2019 10:27	
(...)	(...)	(...)	(...)	

Figure 89: The main page of the workflows

This functionality supports the use cases below [119]:

- UC-0007 Decompose complex systems into smaller pieces
- UC-0011 Create a system using low-code programming
- UC-0012 Creating a complex scenario from templates
- UC-0014 Visualizing services and data flows

A.2 Collect or change the details of a workflow [Workflow Functionality]

When the user decides to either create or edit a workflow, the page from Figure **90** shows up (blank, if creating, or filled in, if editing a workflow).

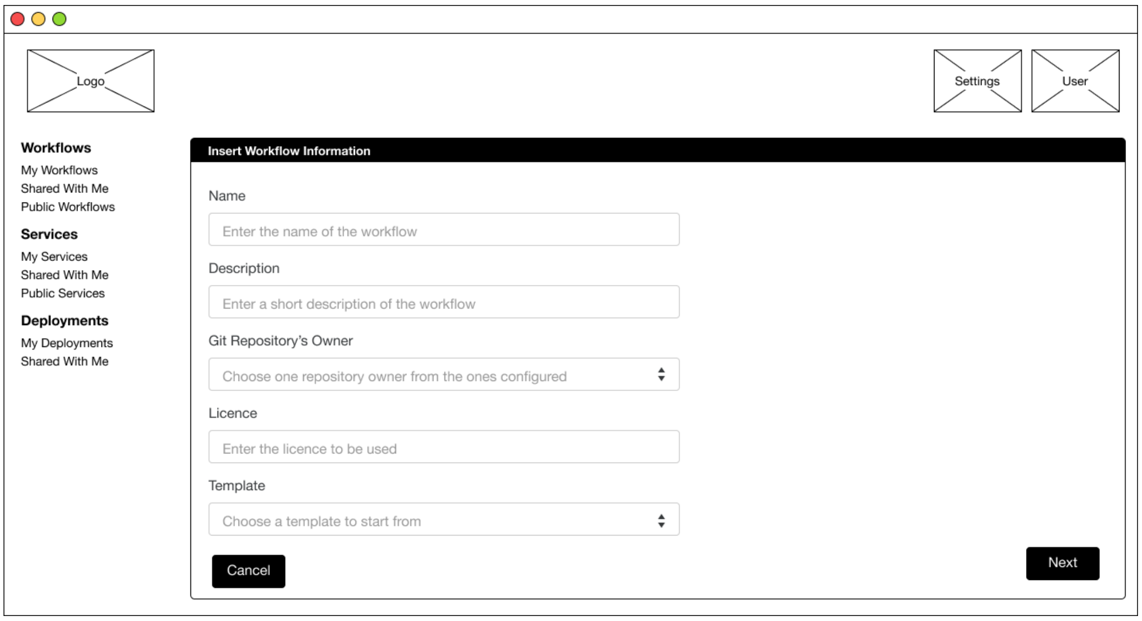


Figure 90: Workflow configuration page

Here, the user inserts the information of the workflow and chooses a diagram template to start designing from. The repository where the workflow will be stored is only created after the “Next” button is pressed. In case the user wants to abort the workflow creation, the navigation bar can be used to go to another page. This functionality supports the use cases below [119]:

- UC-0007 Decompose complex systems into smaller pieces
- UC-0011 Create a system using low-code programming
- UC-0012 Creating a complex scenario from templates
- UC-0014 Visualizing services and data flows

A.3 Draw or edit workflows [Workflow Functionality]

In order to design a workflow, a diagram editor is required. In the beginning, the behaviour of the workflow must be specified, in the "Properties" tab (Figure **91**). For this purpose, a diagram editor will be extended to provide the right-panel properties' window, which shows different options depending on the type of element in focus ("Workflow" if nothing is selected or "Task" if a node is selected).

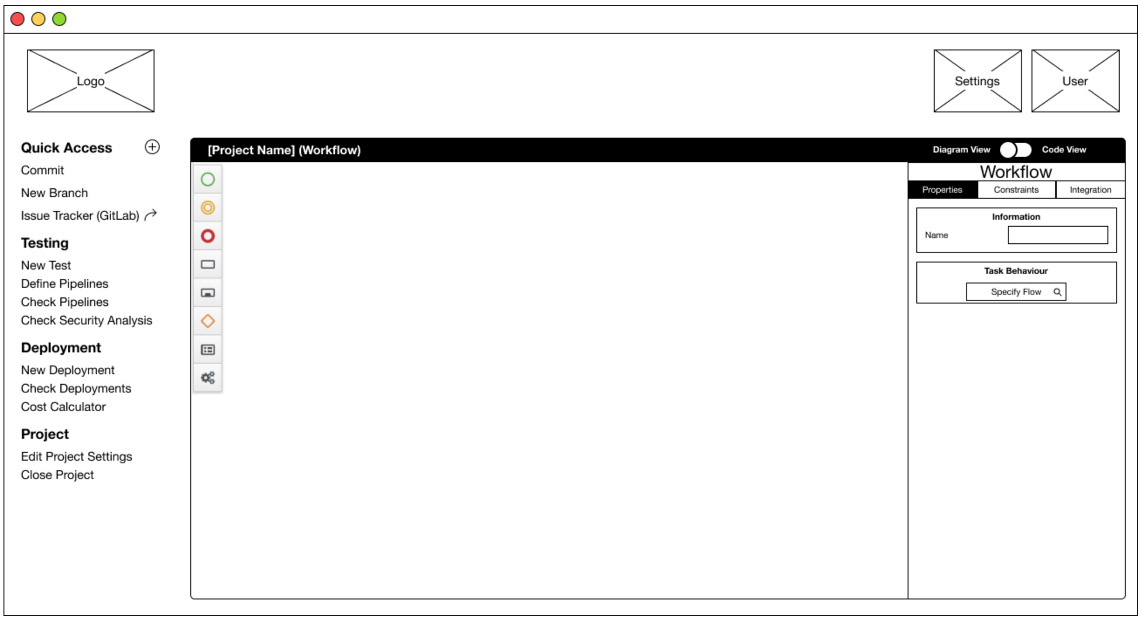


Figure 91: An instance of the diagram editor

After that, the elements are dragged onto the draw area and the fields from the "Properties" and "Functionality" tabs of each node/task must be completed (Figure **92** and Figure **93**). Throughout this process, the Smart Assistant aids the developer by suggesting nodes as the workflow is being designed.

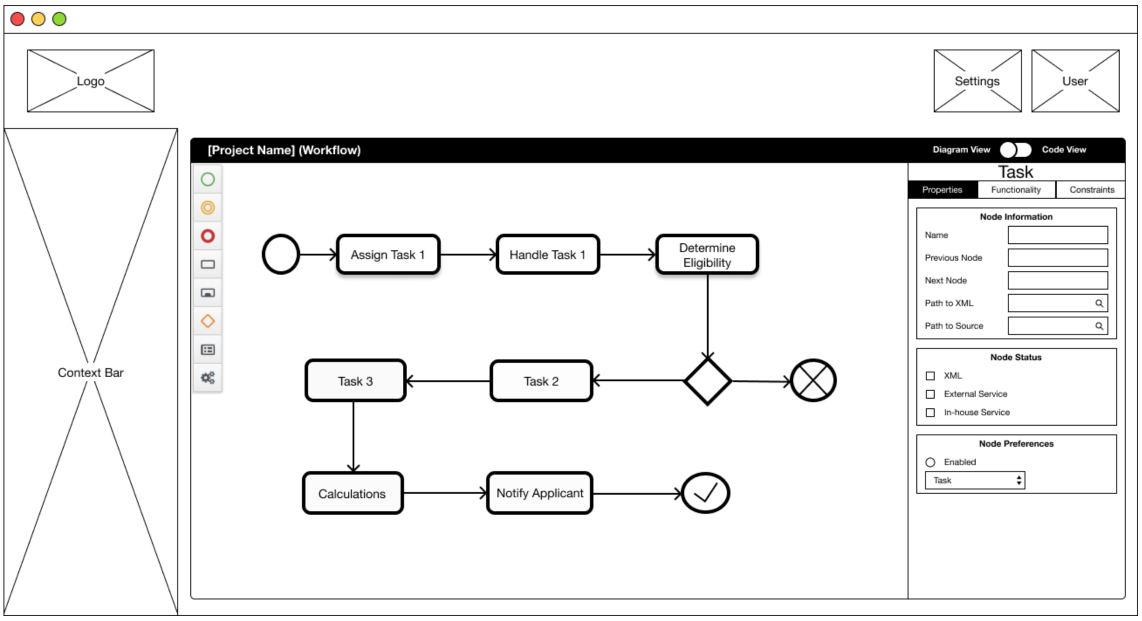


Figure 92: Task Properties within the diagram editor

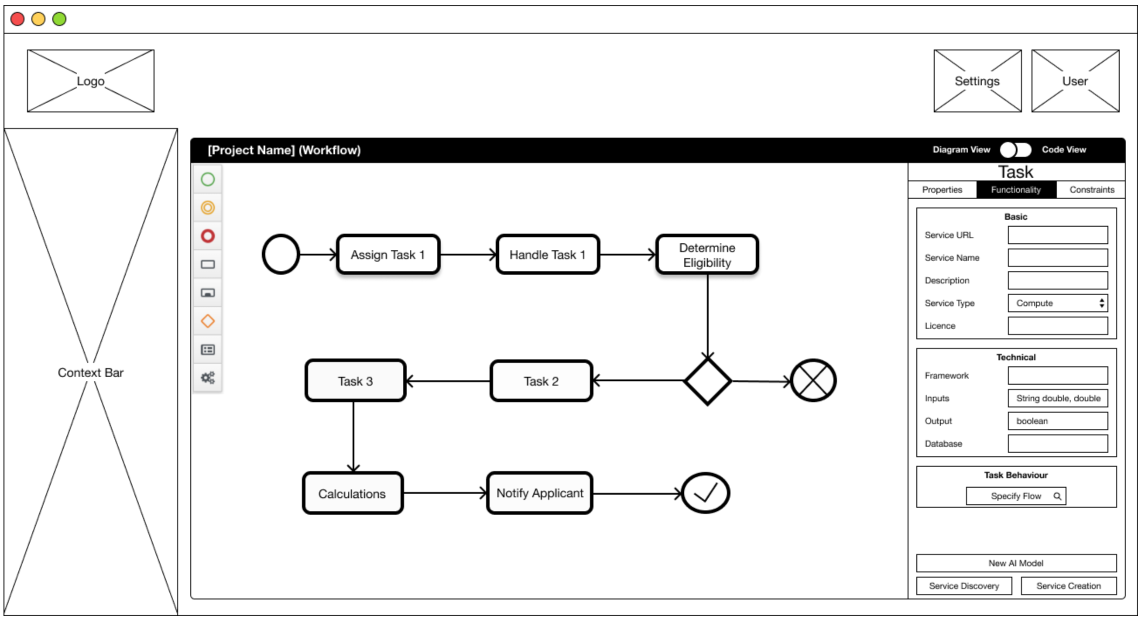


Figure 93: Task Functionality within the diagram editor

At any time, the user can change to the "Code Editor" tab, inspect, and manually edit the code being generated by SmartCLIDE (Figure **94**), using an instance of the Eclipse Theia IDE[43].

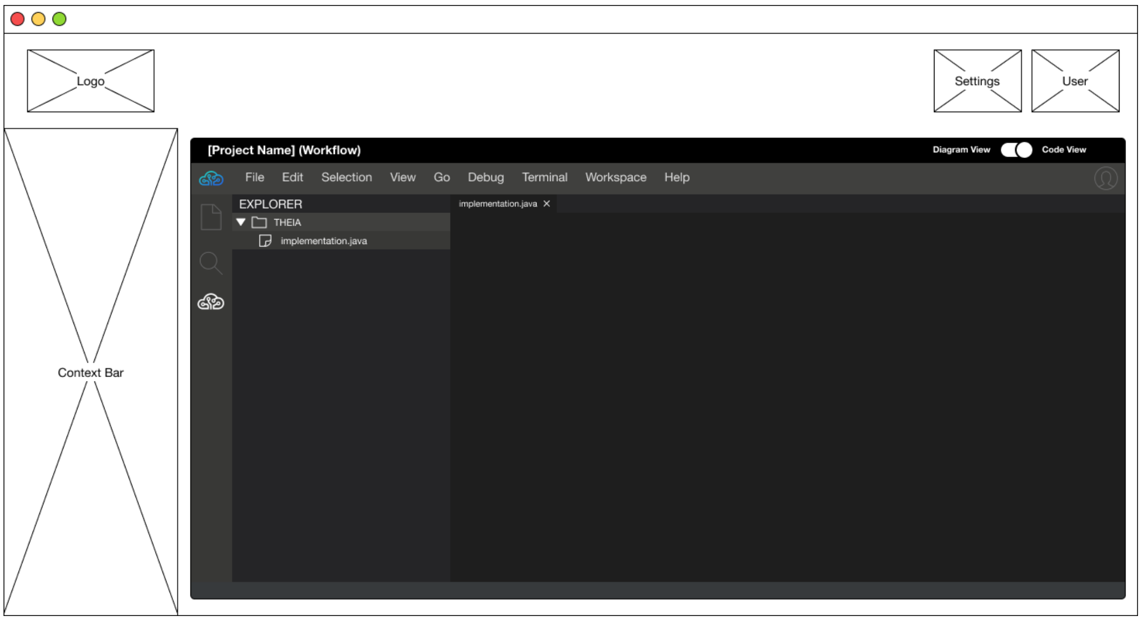


Figure 94: An instance of the Theia code editor

Notice that, once the user arrives at the development stage of a project, the context bar provides access to a set of functionalities grouped in accordance with the stages of the software development life cycle, namely:

- New Test: where the user can specify a new test for the workflow
- Define Pipelines: allows the developer to set a pipeline that automates the development process
- Check Pipelines: provides the user with the state of their pipelines
- Check Security Analysis: displays the results of the security analysis on the workflow
- New Deployment: triggers the creation of a new deployment of the workflow
- Check Deployments: shows the list of deployments of the same workflow to the user
- Cost Calculator: gives access to a comparison of costs between different Cloud providers

[43] https://theia-ide.org

Additionally, a “Quick Access” area is provided, where the user can add custom shortcuts to other functionalities (e.g., Git management), as shown below.

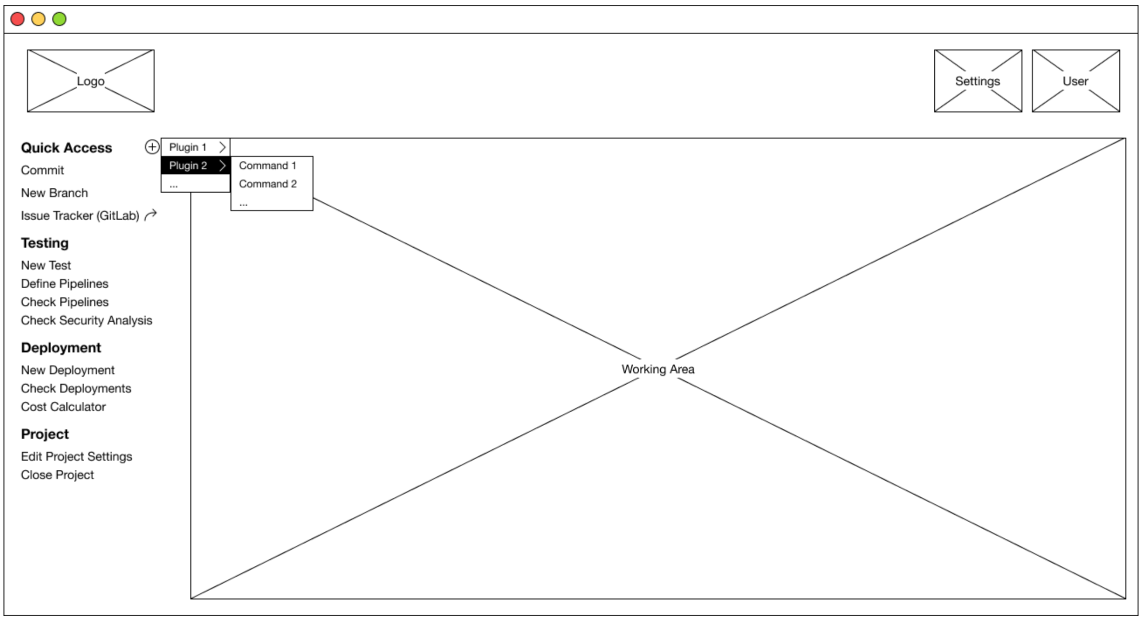


Figure 95: Adding a new shortcut to "Quick Access"

Finally, the “Project” category displays an option to edit the settings of the project or to close it. Once any of these options is clicked, the corresponding window replaces the editor.

Moreover, the title of the window indicates the name of the project as well as its type (workflow, service or deployment). Finally, the toggle button on the upper-right corner allows the developer to select one of the views (diagram – only available for workflows – or code – available for every type of project) at any time.

These functionalities support the use cases below [119]:

- UC-0007 Decompose complex systems into smaller pieces
- UC-0011 Create a system using low-code programming
- UC-0012 Creating a complex scenario from templates
- UC-0014 Visualizing services and data flows
- UC-0022 Usage of a BPMN editor

A.4 Find a suitable service for a task [Workflow Functionality]

A task can be easily implemented using an existing service. For that, SmartCLIDE provides the Service Discovery tool (Figure **96**) which is accessible through the “Functionality” tab. It receives details of new registries and analyses them to be able to suggest the most suitable services, which can be deployable versions, services connected to the web or services in source code.

When using the Service Discovery, the user is asked which task they need a service for, and is given a list of services that match the meta-data previously inserted in the “Functionality” tab. The developer is shown some details about the selected service (e.g., URL, license, framework, etc.) that help them making the best choice. When the “Next” button is pressed, the selected service is bound to the task.

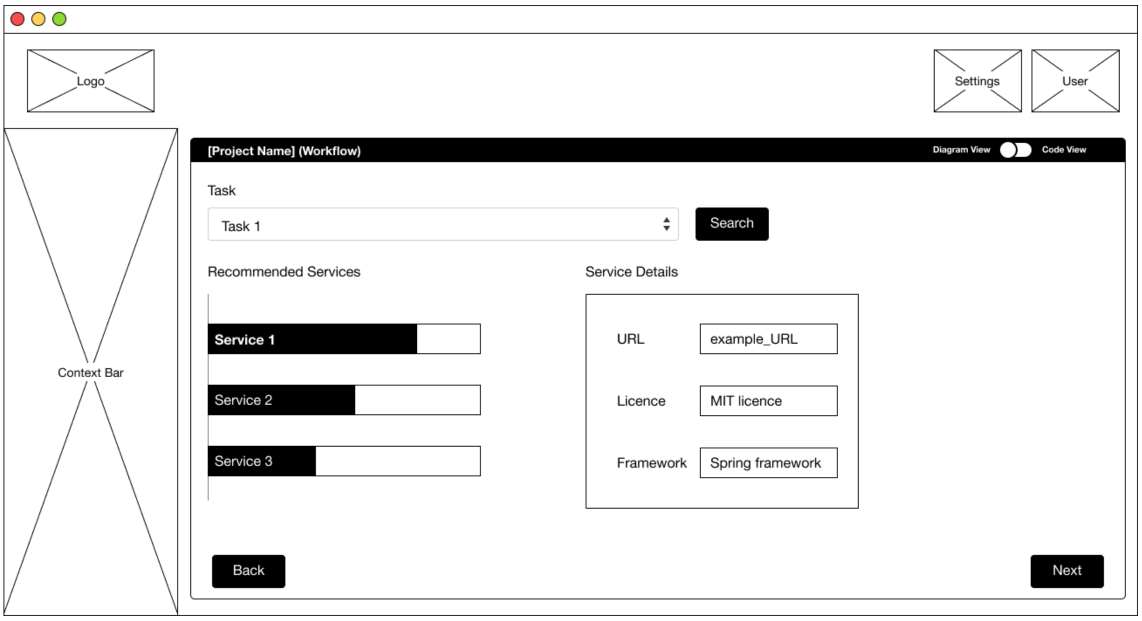


Figure 96: Service discovery configuration page

In case this tool cannot find any results, the message from Figure **97** is displayed.

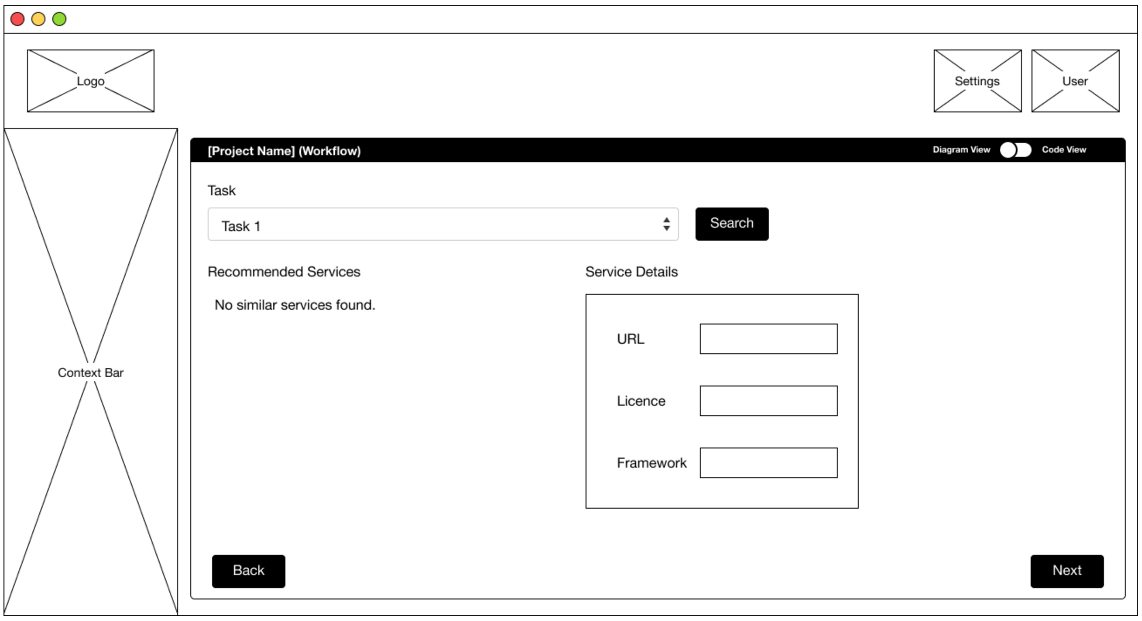


Figure 97: No results found by the service discovery

These functionalities support the use cases below [119]:

- UC-0004 Discover resources and services
- UC-0007 Decompose complex systems into smaller pieces
- UC-0011 Create a system using low-code programming
- UC-0012 Creating a complex scenario from templates

A.5 Test the workflow [Workflow Functionality]

When the workflow is completed, the developer is expected to click a blank area in the canvas, go to the "Integration" tab of the workflow and test it (Figure **98**).

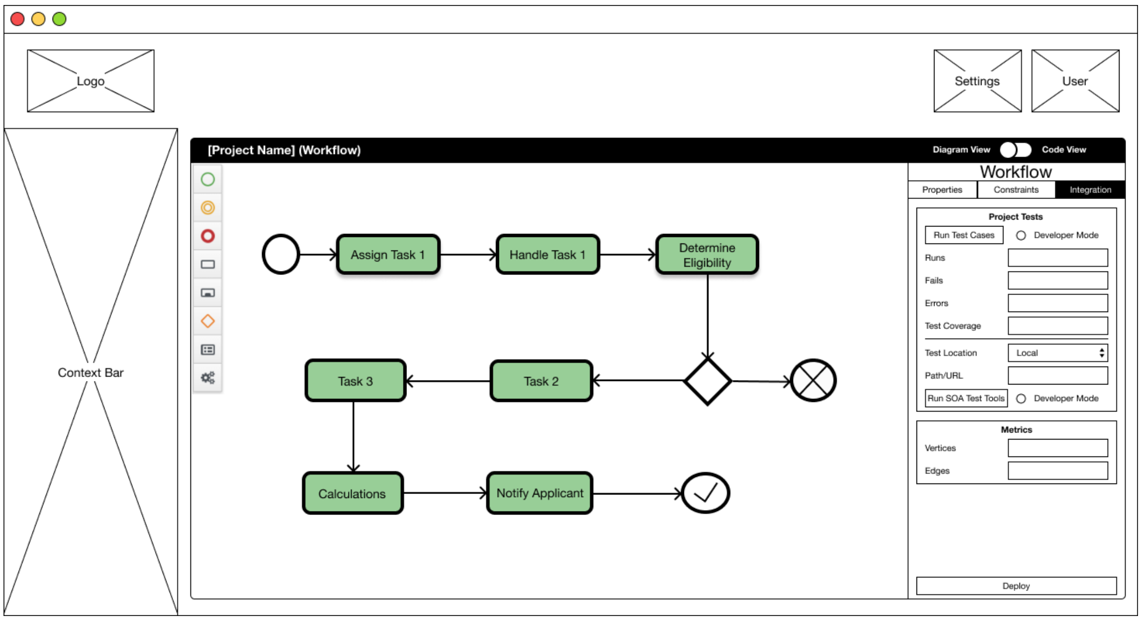


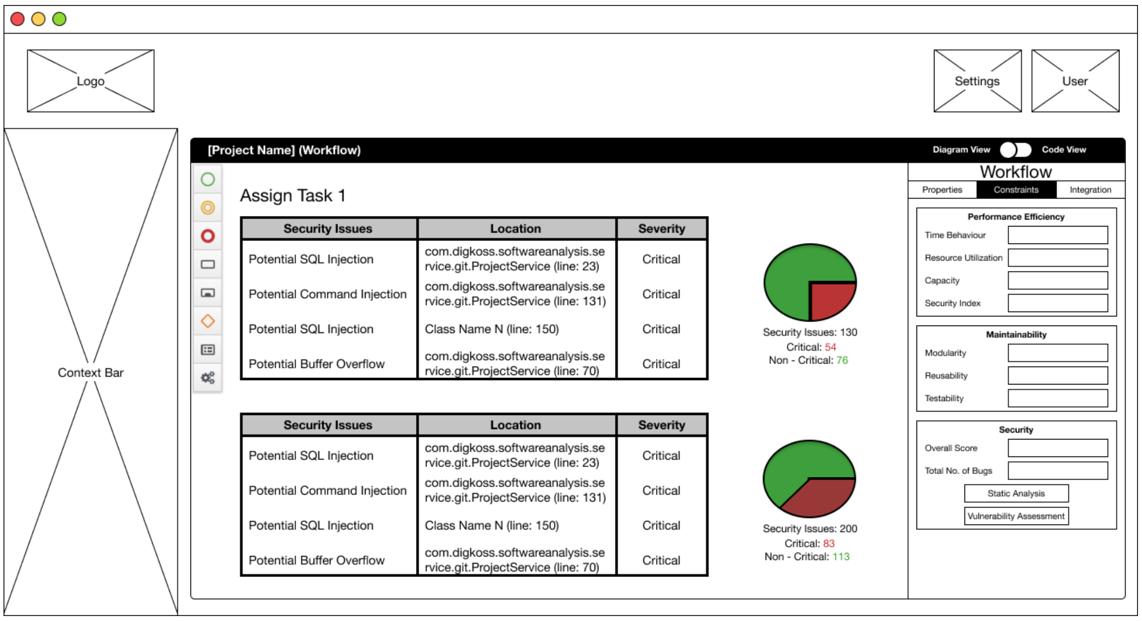
Figure 98: Workflow integration in the diagram editor

This functionality supports the following use cases [119]:

- UC-0006 A non-expert user creates a new service with assistance
- UC-0008 Test services from within SmartCLIDE

A.6 Run a security analysis on the workflow [Workflow Functionality]

SmartCLIDE allows the user to run in the background a security analysis on the workflow, as well as assessing its vulnerability, namely identify potential security breaches. Such options are available in the "Constraints" tab of the workflow, according to Figure **99** and Figure **100**.



Assign Task 1

Security Issues	Location	Severity
Potential SQL Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 23)	Critical
Potential Command Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 131)	Critical
Potential SQL Injection	Class Name N (line: 150)	Critical
Potential Buffer Overflow	com.digkoss.softwareanalysis.service.git.ProjectService (line: 70)	Critical

Security Issues: 130
Critical: 54
Non - Critical: 76

Security Issues	Location	Severity
Potential SQL Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 23)	Critical
Potential Command Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 131)	Critical
Potential SQL Injection	Class Name N (line: 150)	Critical
Potential Buffer Overflow	com.digkoss.softwareanalysis.service.git.ProjectService (line: 70)	Critical

Security Issues: 200
Critical: 83
Non - Critical: 113

Figure 99: Security analysis page

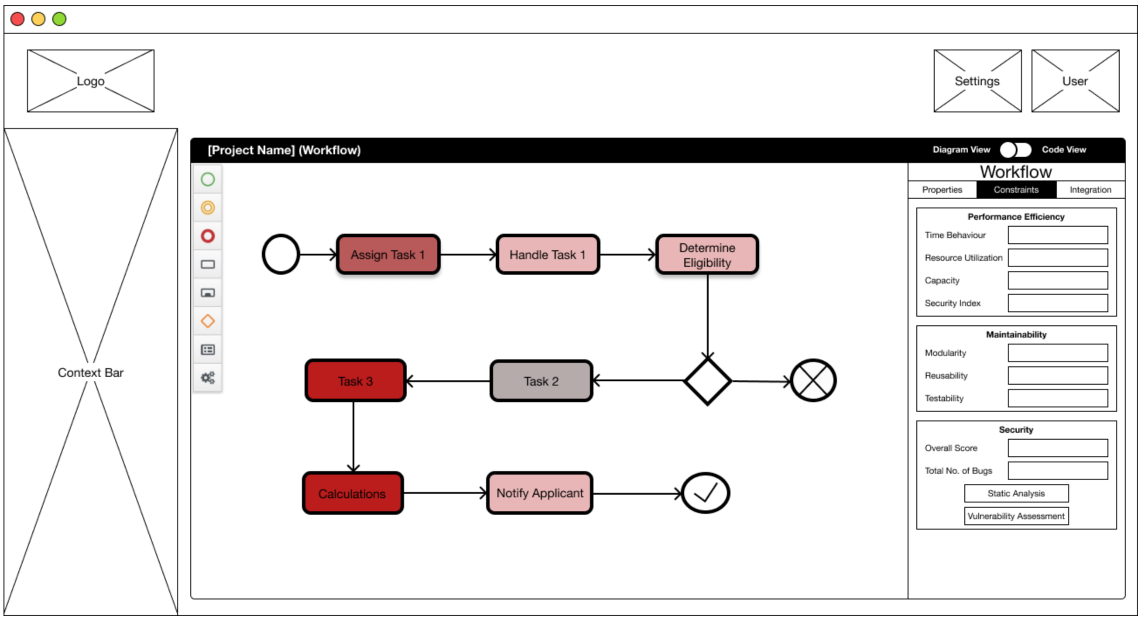


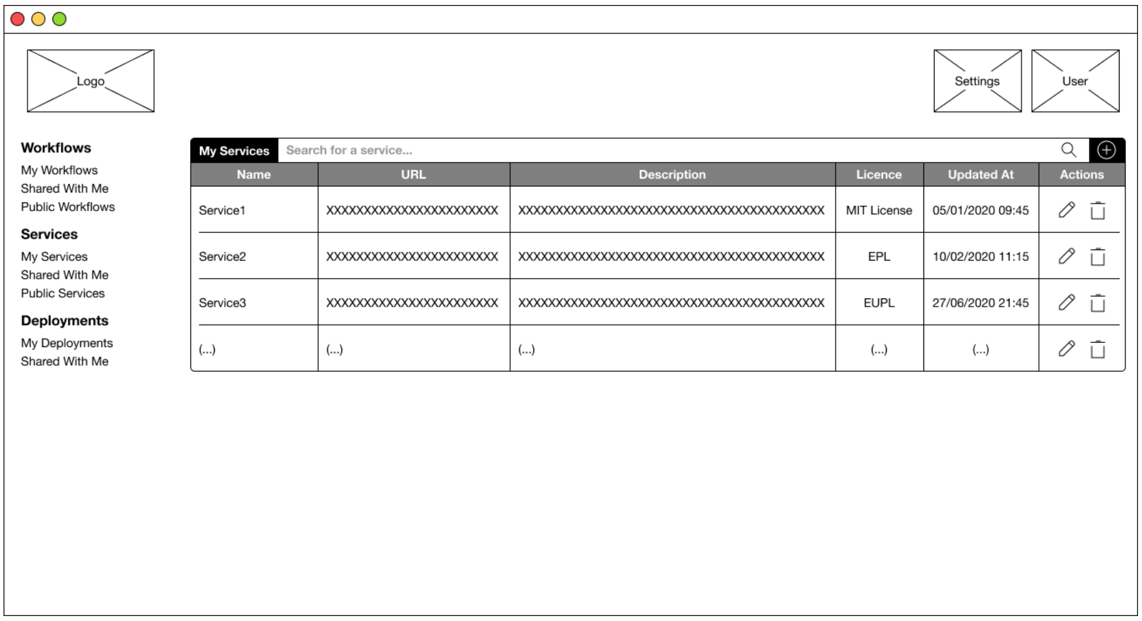
Figure 100: Vulnerability assessment page

This functionality supports the use cases below [119]:

- UC-0026 Create secure services with authentication

A.7 Search for, add, edit or remove services [Service Functionality]

Figure **101** portrays the main page of the services. As in the case of the workflows, the user may filter the services by developer ("My Services", "Shared with Me" or "Public Services"), in the context bar, or any keyword (i.e., name, URL, description, or licence) or value (update date) written in the search bar. Additionally, the sorting order can be changed by clicking the title of the columns. Finally, this page is the starting point for creating, editing, or removing services as well.



Name	URL	Description	Licence	Updated At	Actions
Service1	XXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	MIT License	05/01/2020 09:45	
Service2	XXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	EPL	10/02/2020 11:15	
Service3	XXXXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	EUPL	27/06/2020 21:45	
(...)	(...)	(...)	(...)	(...)	

Figure 101: The main page of the services

This functionality supports the use cases below [119]:

- UC-0001 Creation of a service from a template
- UC-0002 Create services with data abstraction levels
- UC-0003 Create and deploy a service from the IDE
- UC-0004 Discover resources and services
- UC-0006 A non-expert user creates a new service with assistance
- UC-0008 Test services from within SmartCLIDE
- UC-0012 Creating a complex scenario from templates
- UC-0015 Accessing Git repositories
- UC-0020 Specify the licence of a service
- UC-0026 Create secure services with authentication

A.8 Edit the details of a service [Service Functionality]

When the user decides to either create or edit a service, the page presented in Figure **102** shows up (blank, if creating, or filled in, if editing). In this page the user can insert or edit the main details about the service:

- Name: the name of the service
- Description: a brief description of the service
- License: the license used to protect the service distribution
- Framework: the framework used to develop the service

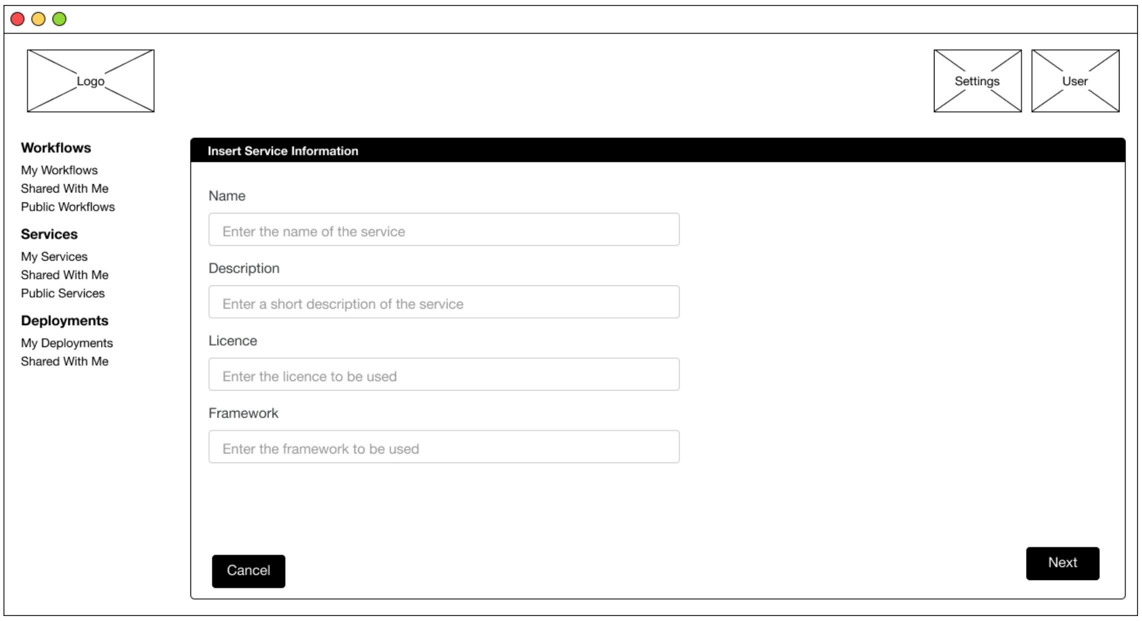


Figure 102: The main configuration page of a service

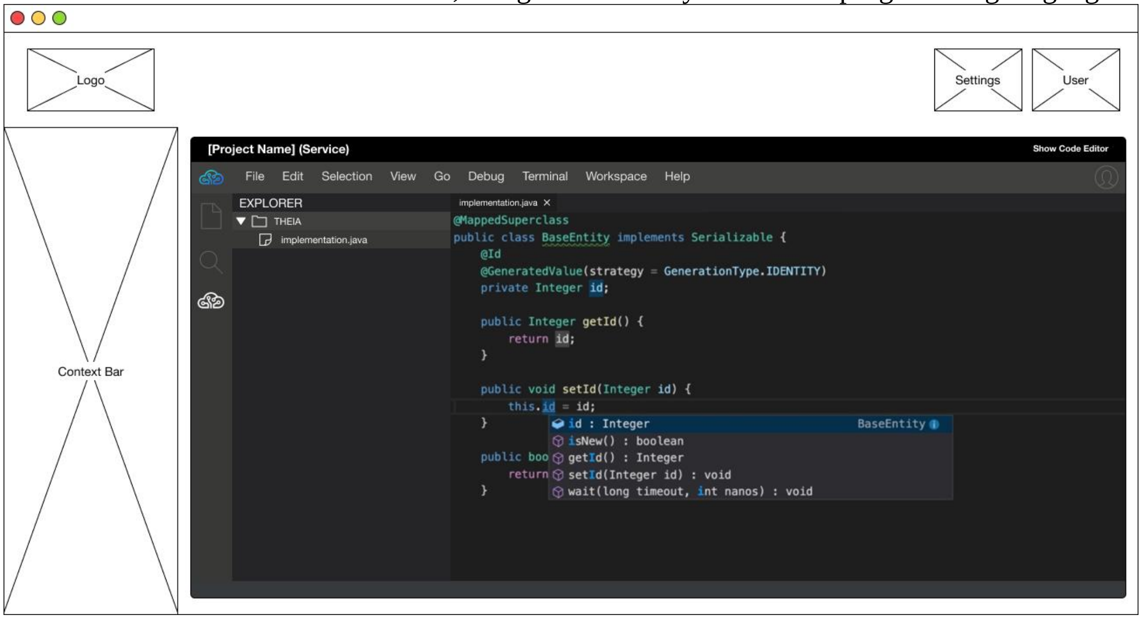
After pressing “Next”, a GitLab repository and a Jenkins continuous integration (CI) pipeline are configured. These functionalities support [119]:

- UC-0001 Creation of a service from a template
- UC-0002 Create services with data abstraction levels
- UC-0003 Create and deploy a service from the IDE
- UC-0004 Discover resources and services
- UC-0006 A non-expert user creates a new service with assistance
- UC-0008 Test services from within SmartCLIDE
- UC-0012 Creating a complex scenario from templates
- UC-0015 Accessing Git repositories
- UC-0020 Specify the licence of a service
- UC-0026 Create secure services with authentication

A.9 Smart Assistant for code edition [Service Functionality]

SmartCLIDE also provides an Eclipse Theia[44] instance, as source code editor. Throughout the implementation stage, the Smart Assistant helps the developer with:

- Code autocompletion (Figure **103**): the user gets suggestions of variables, function names, etc. as they type.
- Live template recommendations (Figure **104**): the syntax of functions, assignments, etc. is automatically completed, preventing the user from having to type every single character.
- Comments generation (Figure **105**): right-clicking on the code provides an option to generate comments on the written code, using the correct syntax for the programming language in use.

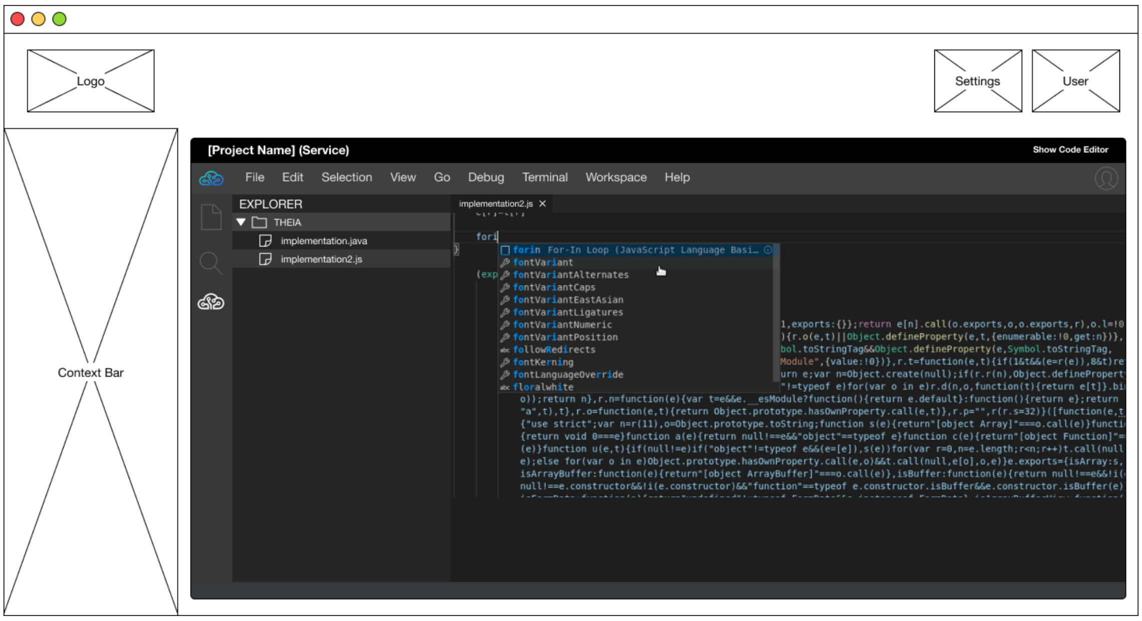


```
@MappedSuperclass
public class BaseEntity implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }        id : Integer                               BaseEntity
             isNew() : boolean
    public boo getId() : Integer
        return setId(Integer id) : void
    }        wait(long timeout, int nanos) : void
```

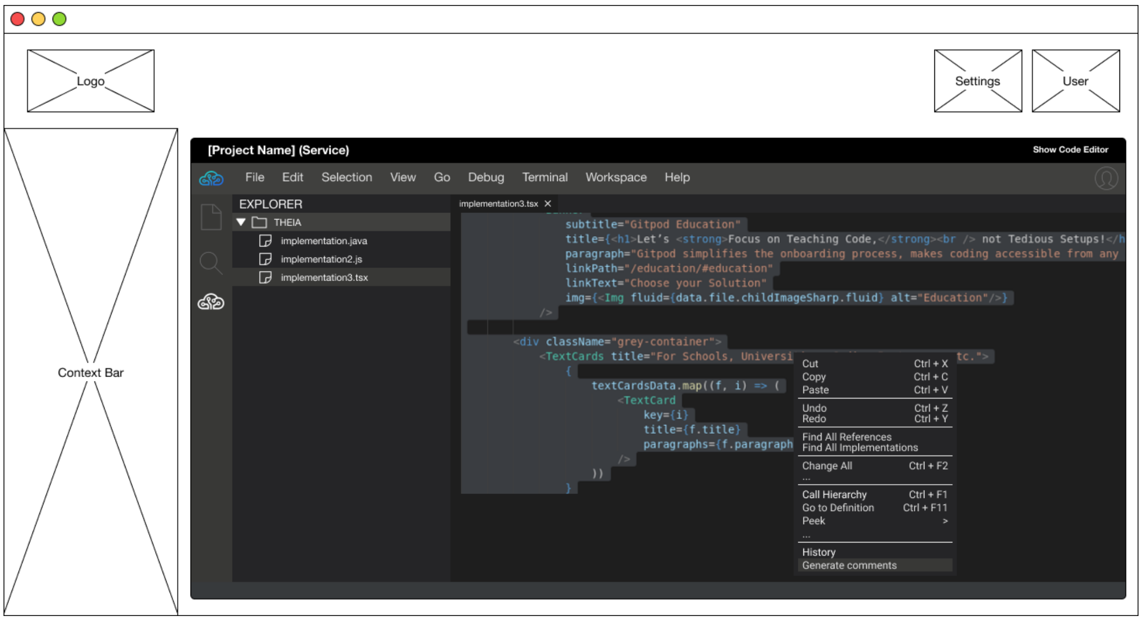
Figure 103: Smart Assistant services: Code autocompletion



```
    fori
}       forin For-In Loop (JavaScript Language Basi…
        fontVariant
    (exp fontVariantAlternates
        fontVariantCaps
        fontVariantEastAsian
        fontVariantLigatures
        fontVariantNumeric                  1,exports:{}};return e[n].call(o.exports,o,o.exports,r),o.l=!0
        fontVariantPosition                 ){r.o(e,t)||Object.defineProperty(e,t,{enumerable:!0,get:n}}},
        followRedirects                     bol.toStringTag&&Object.defineProperty(e,Symbol.toStringTag,
        fontKerning                         Module",{value:!0})},r.t=function(e,t){if(1&t&&(e=r(e)),8&t)re
        fontLanguageOverride                urn e;var n=Object.create(null);if(r.r(n),Object.definePropert
        floralwhite                         "!=typeof e)for(var o in e)r.d(n,o,function(t){return e[t]}.bi
          o));return n},r.n=function(e){var t=e&&e.__esModule?function(){return e.default}:function(){return e};return
          "a",t),t},r.o=function(e,t){return Object.prototype.hasOwnProperty.call(e,t)},r.p="",r(r.s=32)}([function(e,t
          {"use strict";var n=r(11),o=Object.prototype.toString;function s(e){return"[object Array]"===o.call(e)}functi
          {return void 0===e}function a(e){return null!==e&&"object"==typeof e}function c(e){return"[object Function]"=
          (e)}function u(e,t){if(null!=e)if("object"!=typeof e&&(e=[e]),s(e))for(var r=0,n=e.length;r<n;r++)t.call(null
          e);else for(var o in e)Object.prototype.hasOwnProperty.call(e,o)&&t.call(null,e[o],o,e)}e.exports={isArray:s,
          isArrayBuffer:function(e){return"[object ArrayBuffer]"===o.call(e)},isBuffer:function(e){return null!==e&&!i(
          null!==e.constructor&&!i(e.constructor)&&"function"==typeof e.constructor.isBuffer&&e.constructor.isBuffer(e)
```

Figure 104: Smart Assistant services: Live template recommendation

[44] https://theia-ide.org



implementation3.tsx

```
            subtitle="Gitpod Education"
            title={<h1>Let's <strong>Focus on Teaching Code,</strong><br /> not Tedious Setups!</h
            paragraph="Gitpod simplifies the onboarding process, makes coding accessible from any
            linkPath="/education/#education"
            linkText="Choose your Solution"
            img={<Img fluid={data.file.childImageSharp.fluid} alt="Education"/>}
        />

    <div className="grey-container">
        <TextCards title="For Schools, Universi
            {
                textCardsData.map((f, i) => (
                    <TextCard
                        key={i}
                        title={f.title}
                        paragraphs={f.paragraph
                    />
                ))
            }
```

Figure 105: Smart Assistant services: Comments generation

These functionalities support [119]:

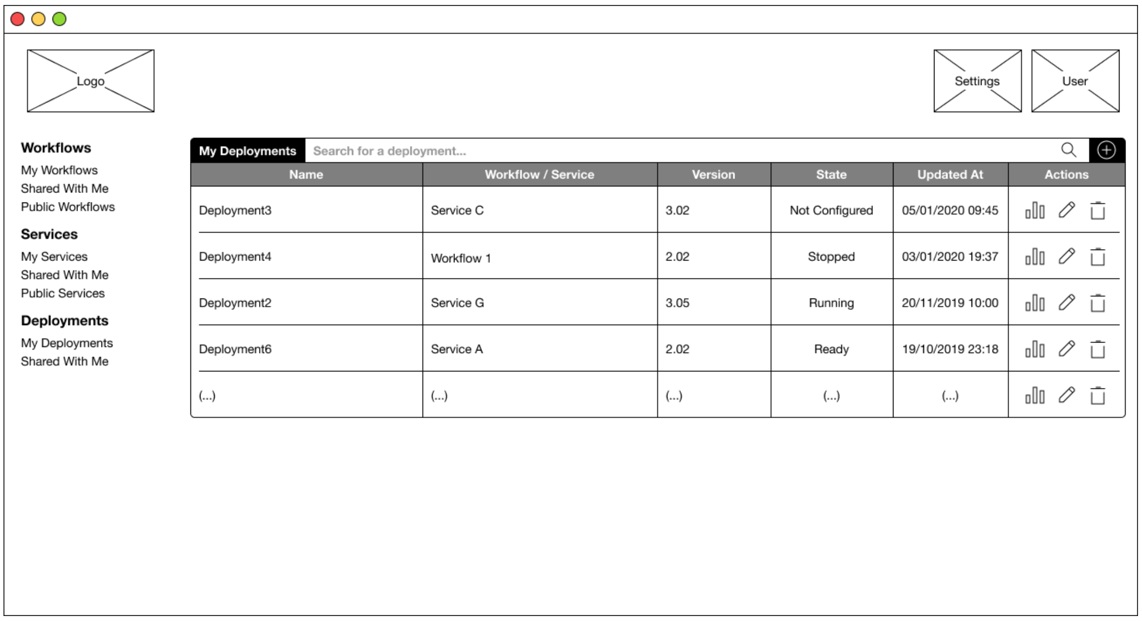
- UC-0001 Creation of a service from a template
- UC-0002 Create services with data abstraction levels
- UC-0003 Create and deploy a service from the IDE
- UC-0004 Discover resources and services
- UC-0006 A non-expert user creates a new service with assistance
- UC-0008 Test services from within SmartCLIDE
- UC-0012 Creating a complex scenario from templates
- UC-0015 Accessing Git repositories
- UC-0020 Specify the licence of a service
- UC-0026 Create secure services with authentication

A.10 Search for, add, edit or remove deployments [Deployment Functionality]

Figure **106** illustrates the main page of the deployment. Similarly, the context bar gives access to both the user's own deployments and shared with them, including the following details:

- Name: name of the deployment
- Workflow/Service: name of the deployed workflow or service
- Version: version of the deployed workflow or service
- State: state of the deployment (*not configured*, *ready*, *running*, *stopped*)
- Updated At: date when the deployment was last edited

The searching mechanisms work in the same way as in the previous cases. In addition, through this page the users can create new and edit or remove past deployments.



Name	Workflow / Service	Version	State	Updated At	Actions
Deployment3	Service C	3.02	Not Configured	05/01/2020 09:45	
Deployment4	Workflow 1	2.02	Stopped	03/01/2020 19:37	
Deployment2	Service G	3.05	Running	20/11/2019 10:00	
Deployment6	Service A	2.02	Ready	19/10/2019 23:18	
(...)	(...)	(...)	(...)	(...)	

Figure 106: The main page of the deployments

These functionalities support [119]:

- UC-0003 Create and deploy a service from the IDE
- UC-0005 Search for deployed services
- UC-0009 Conduct a cost analysis
- UC-0010 Deploy a service from the CLI within SmartCLIDE

A.11 Edit the details of a deployment [Deployment Functionality]

When the user decides to either create or edit a workflow, the page presented in Figure **107** shows up (blank, if creating, or filled in, if editing). In this page the user can insert or edit the main details about the deployment:

- Name: the name of the service
- Workflow/Service: indicates what is going to be deployed
- Version: the version of the workflow or service to be deployed

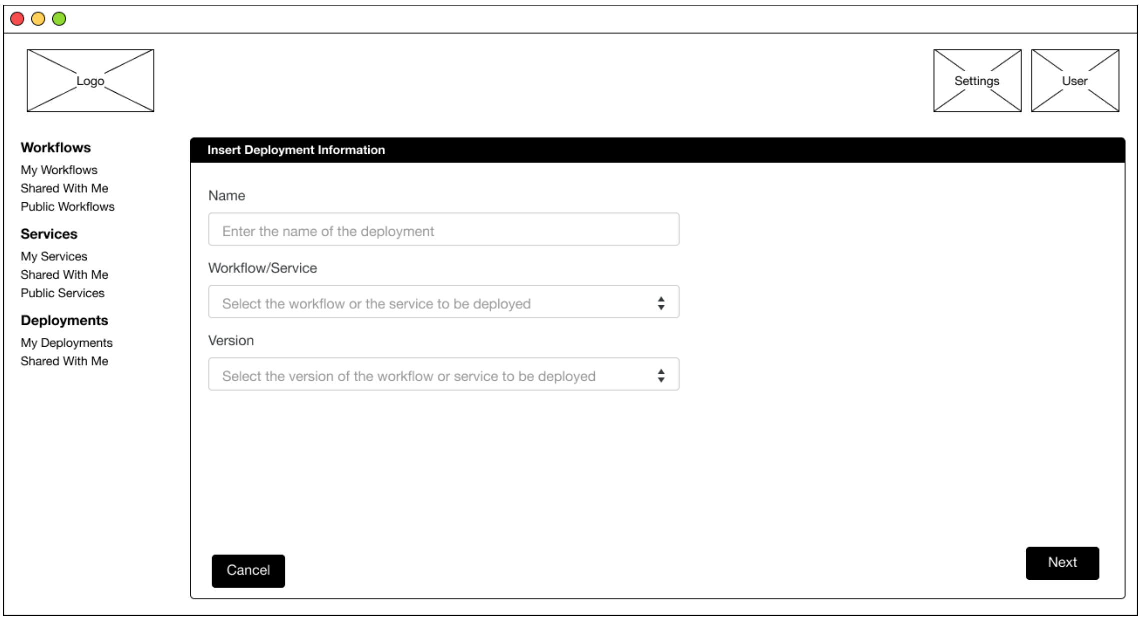


Figure 107: The main configuration page of a deployment

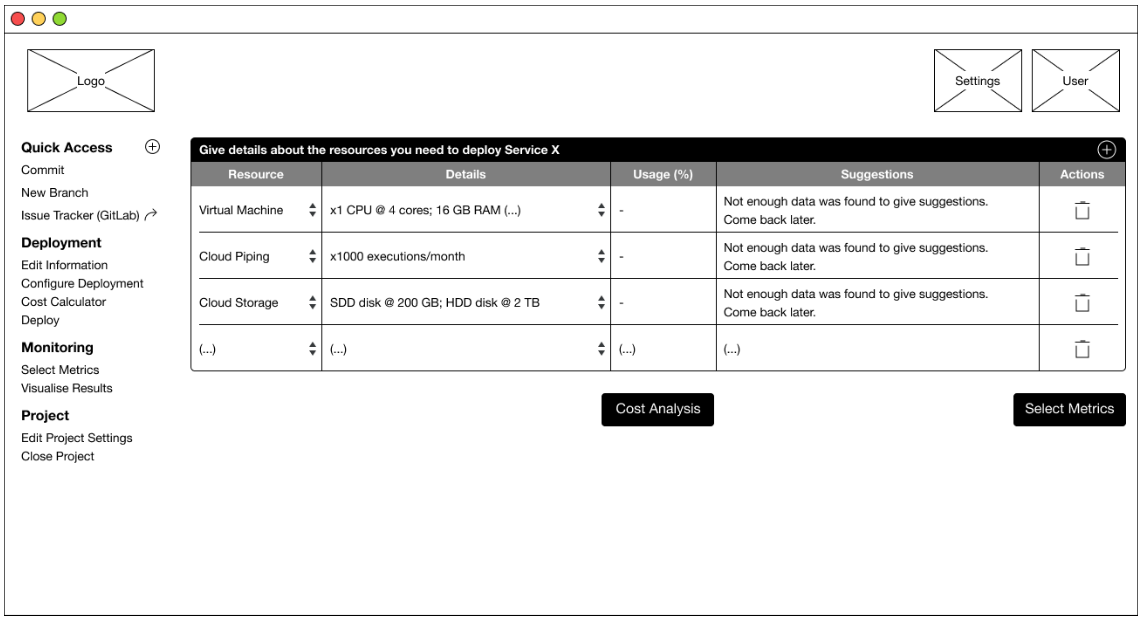
This functionality supports [119]:

- UC-0003 Create and deploy a service from the IDE
- UC-0005 Search for deployed services
- UC-0009 Conduct a cost analysis
- UC-0010 Deploy a service from the CLI within SmartCLIDE

A.12 Resource configuration and optimisation [Deployment Functionality]

To configure a deployment, the user must click the pencil symbol, by which time they are redirected to the deployment configuration page (Figure **108**), where they insert the computing resources required for the deployment and, after a while, fine-tune them according to the IDE suggestions (Figure **109**). In this page, the following columns are presented:

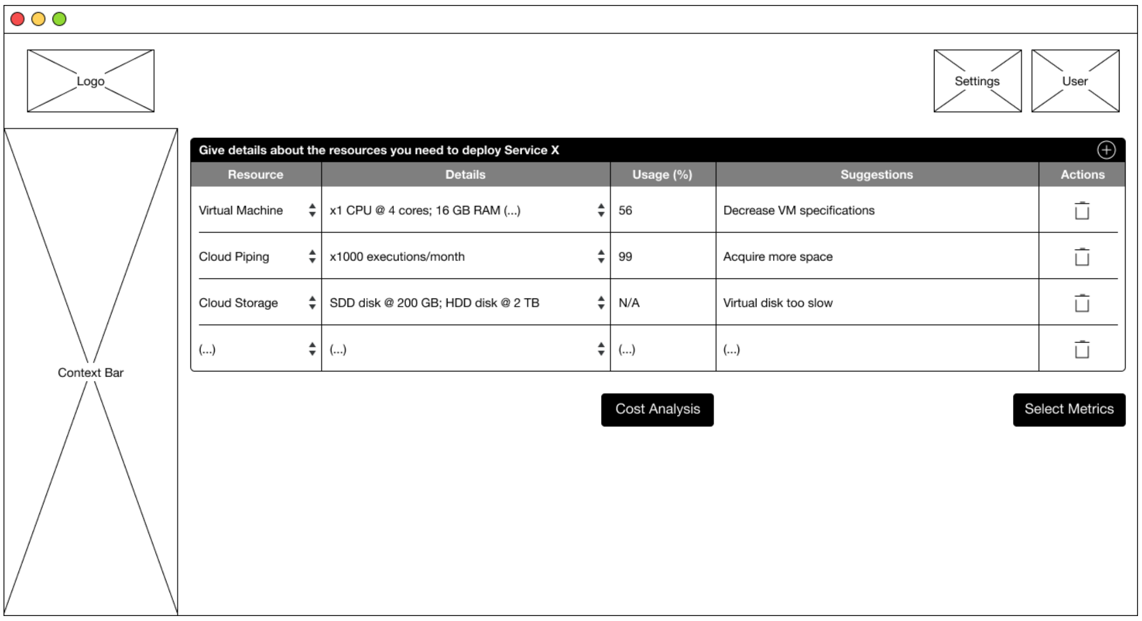
- Resource: the type of resource needed for the deployment
- Details: the resource specifications
- Usage (%): usage percentage, that indicates how optimised the configuration of that resource is.
- Suggestions: recommendations on how to optimise the configuration



Give details about the resources you need to deploy Service X

Resource	Details	Usage (%)	Suggestions	Actions
Virtual Machine	x1 CPU @ 4 cores; 16 GB RAM (...)	-	Not enough data was found to give suggestions. Come back later.	
Cloud Piping	x1000 executions/month	-	Not enough data was found to give suggestions. Come back later.	
Cloud Storage	SDD disk @ 200 GB; HDD disk @ 2 TB	-	Not enough data was found to give suggestions. Come back later.	
(...)	(...)	(...)	(...)	

Figure 108: The deployment configuration page



Give details about the resources you need to deploy Service X

Resource	Details	Usage (%)	Suggestions	Actions
Virtual Machine	x1 CPU @ 4 cores; 16 GB RAM (...)	56	Decrease VM specifications	
Cloud Piping	x1000 executions/month	99	Acquire more space	
Cloud Storage	SDD disk @ 200 GB; HDD disk @ 2 TB	N/A	Virtual disk too slow	
(...)	(...)	(...)	(...)	

Figure 109: The deployment configuration suggestions

As in the case of the workflows and services, when configuring a deployment, a context bar is displayed on the left-hand side of the page. The options it provides, however, are only related with the deployment or monitoring stages, namely:

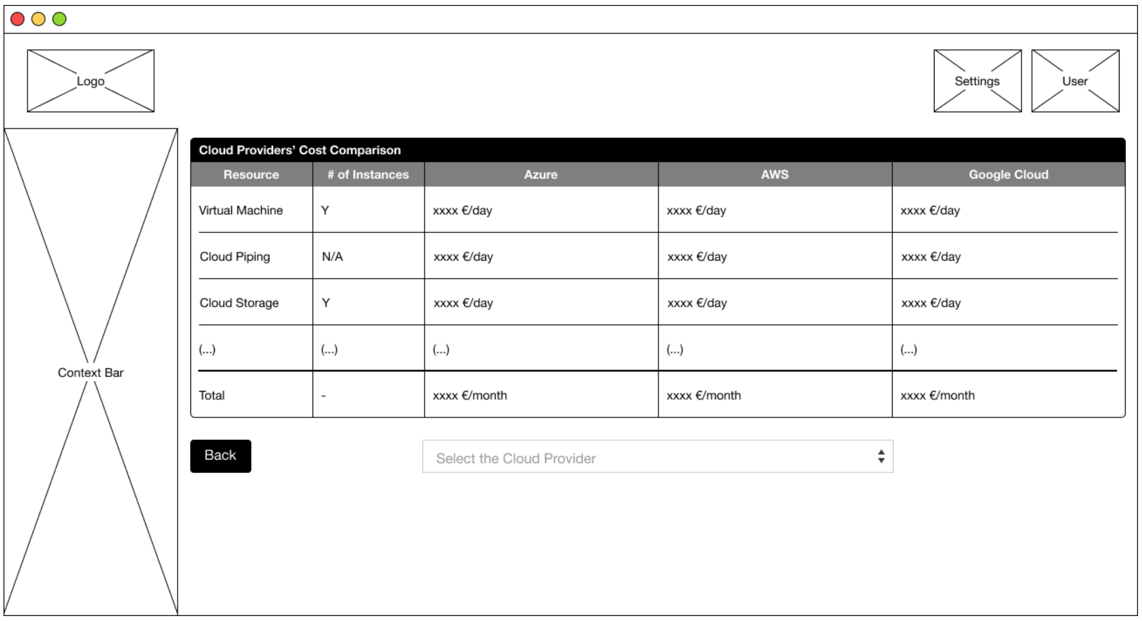
- Edit Information: redirects the user to the page with the information about the deployment
- Configure Deployment: allows choosing the resources needed for the deployment
- Cost Calculator: presents a cost comparison between different Cloud providers
- Deploy: where the developer can deploy the workflow/service
- Select Metrics: allows the user to choose which parameters must be monitored during runtime
- Visualise Results: shows the values for the previously selected metrics

These functionalities support [119]:

- UC-0003 Create and deploy a service from the IDE
- UC-0005 Search for deployed services
- UC-0009 Conduct a cost analysis
- UC-0010 Deploy a service from the CLI within SmartCLIDE

A.13 Run a cost analysis [Deployment Functionality]

Before deploying the workflow/service, the user can see a cost comparison that assists in choosing the best cloud provider (Figure **110Figure 110**) and then, go back to the deployment configuration page. It is worth mentioning that the cost simulation service will only be able to provide accurate values at the component level.



Cloud Providers' Cost Comparison

Resource	# of Instances	Azure	AWS	Google Cloud
Virtual Machine	Y	xxxx €/day	xxxx €/day	xxxx €/day
Cloud Piping	N/A	xxxx €/day	xxxx €/day	xxxx €/day
Cloud Storage	Y	xxxx €/day	xxxx €/day	xxxx €/day
(...)	(...)	(...)	(...)	(...)
Total	-	xxxx €/month	xxxx €/month	xxxx €/month

Figure 110: Main page of the cost comparison service

This functionality supports [119]:

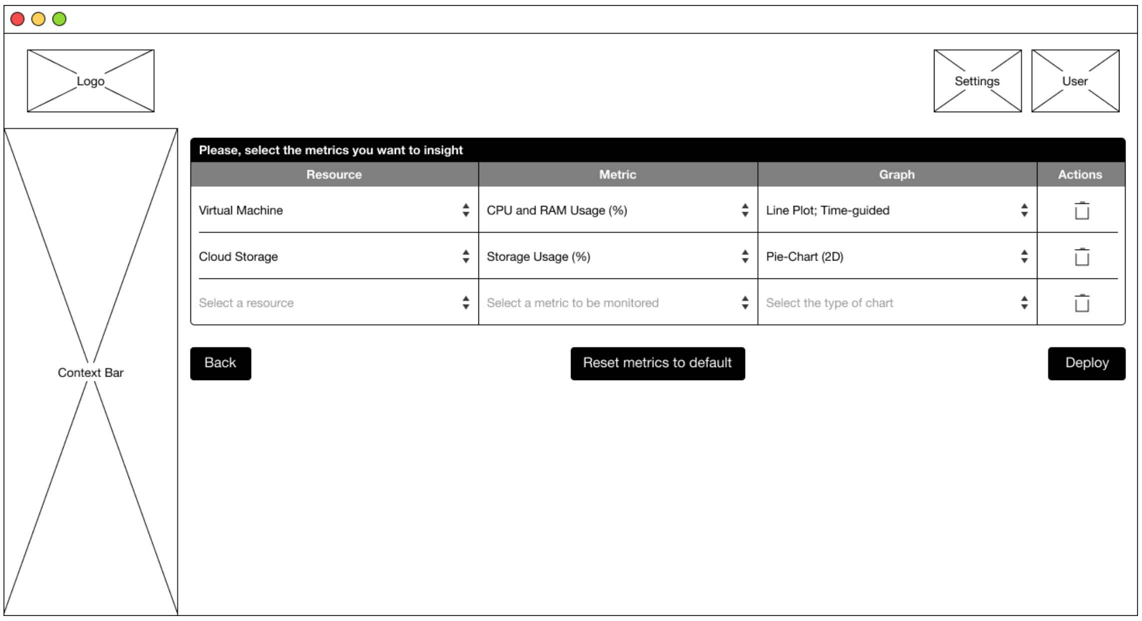
- UC-0009 Conduct a cost analysis

A.14 Choose the metrics to be monitored [Deployment Functionality]

The user can choose the metrics to be automatically monitored during runtime by clicking "Select Metrics" in the deployment configuration page, or through the context bar. By doing so, the user is redirected to the runtime metrics selection page presented in Figure **111**. There, the developer chooses as many resource metrics as they want to monitor, indicating:

- Resource: one resource from the set inserted in the deployment configuration
- Metric: the parameter to be monitored from the indicated resource
- Graph: the type of graphical representation for the information

After deploying the workflow/service, the user is redirected to the main page of the deployments.



Please, select the metrics you want to insight

Resource	Metric	Graph	Actions
Virtual Machine	CPU and RAM Usage (%)	Line Plot; Time-guided	🗑
Cloud Storage	Storage Usage (%)	Pie-Chart (2D)	🗑
Select a resource	Select a metric to be monitored	Select the type of chart	🗑

Figure 111: Runtime metrics selection page

This functionality supports [119]:

- UC-0025 Monitor and verify services

A.15 Monitor the metrics selected for a deployment [Deployment Functionality]

The developer clicks the bars symbol in the main page of deployments and monitors the selected metrics, as presented in Figure **112**. The displayed data is collected by the Runtime Monitoring & Verification and the Context Handling components.

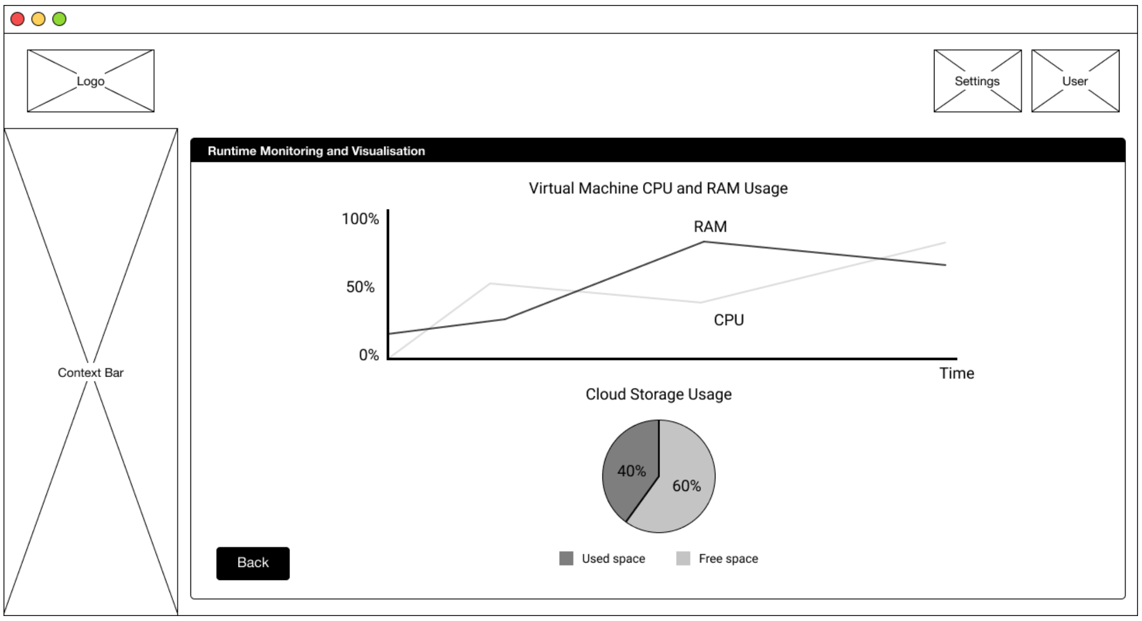


Figure 112: Runtime metrics monitoring and visualization page

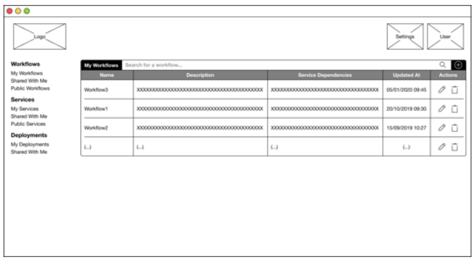
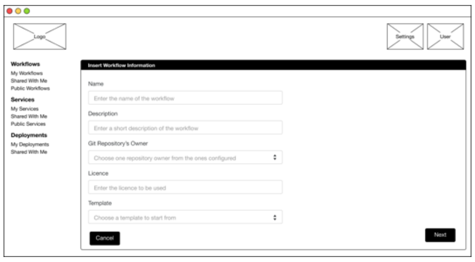
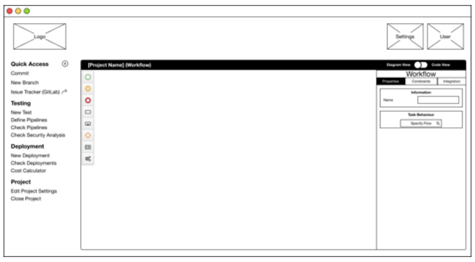
This functionality supports [119]:

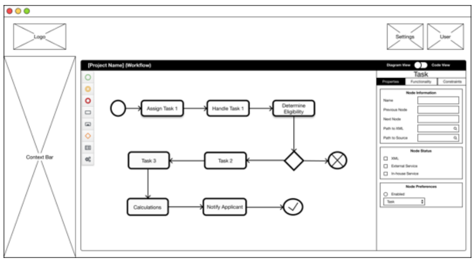
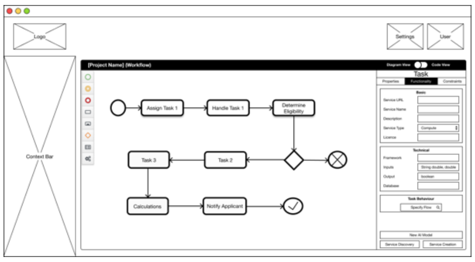
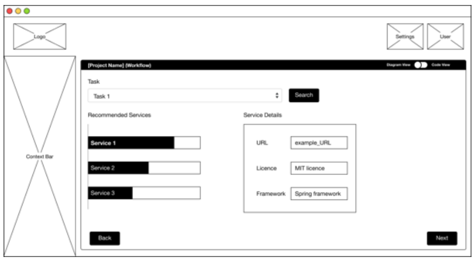
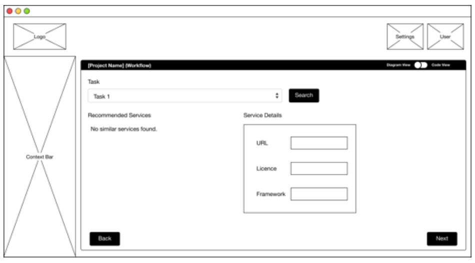
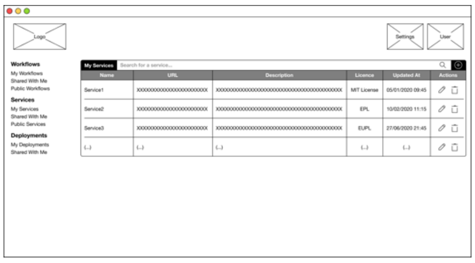
- UC-0025 Monitor and verify services

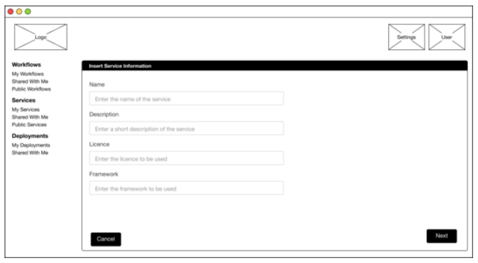
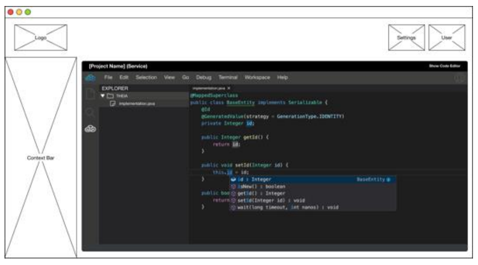
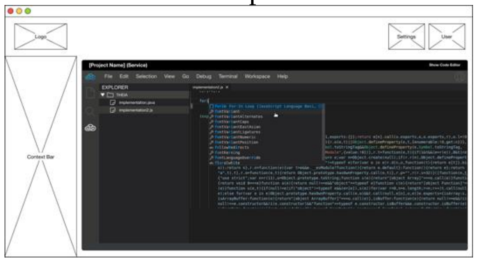
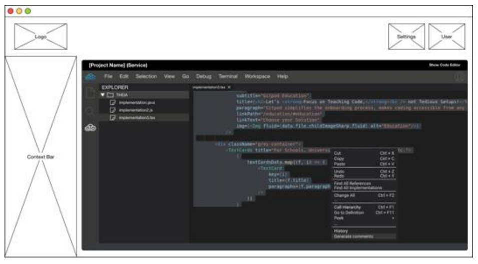
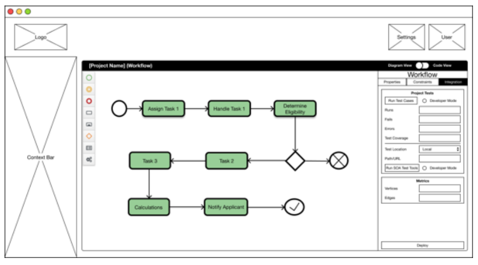
8 Annex B: End-to-End Scenario Use Case

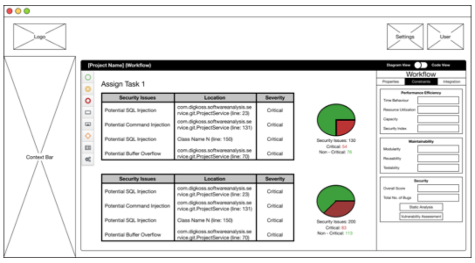
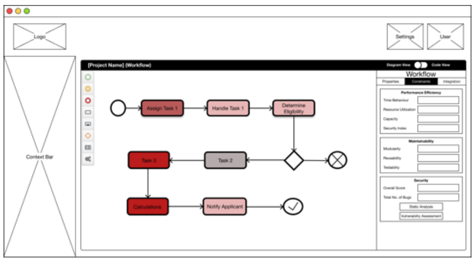
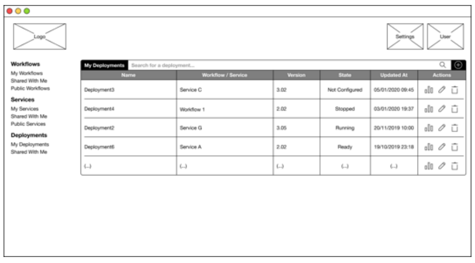
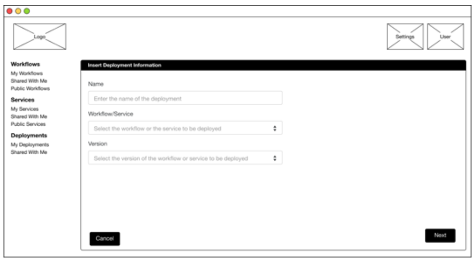
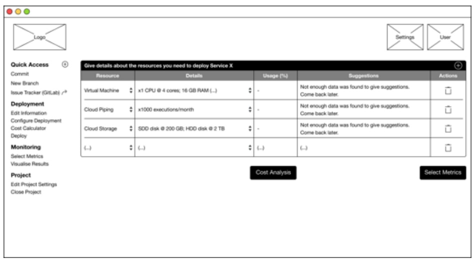
With the aim of showing how the tools and functionalities presented in the previous section can be used within the same scenario, an 18-steps end-to-end demonstration Use Case was designed, including as many SmartCLIDE components as possible.

Table 50: End-to-end demonstration Use Case

Step	Description	Example
1	The user is at the main page of the workflows and decides to create a new one.	 Logo Settings User Workflows My Workflows Shared With Me Public Workflows Services My Services Shared With Me Public Services Deployments My Deployments Shared With Me My Workflows Search for a workflow... Name Description Service Dependencies Updated At Actions Workflow3 05/01/2020 09:45 Workflow1 20/10/2019 09:30 Workflow2 15/09/2019 10:27 The main page of the workflows
2	The user is prompted to insert the details of the workflow (e.g., name, description, etc.) and to choose a workflow template (if any).	 Logo Settings User Workflows My Workflows Shared With Me Public Workflows Services My Services Shared With Me Public Services Deployments My Deployments Shared With Me Insert Workflow Information Name Enter the name of the workflow Description Enter a short description of the workflow Git Repository's Owner Choose one repository owner from the ones configured Licence Enter the licence to be used Template Choose a template to start from Cancel Next Workflow configuration page
3	In the diagram editor, the user designs the workflow (assisted by the Smart Assistant).	 Logo Settings User Quick Access Commit New Branch Issue Tracker (GitLab) Testing New Test Define Pipelines Check Pipelines Check Security Analysis Deployment New Deployment Check Deployments Cost Calculator Project Edit Project Settings Close Project [Project Name] (Workflow) Diagram View Code View Workflow Properties Constraints Integration Information Name Task Behaviour Specify Flow An instance of the diagram editor

Step	Description	Example
4	For each node, the user must fill in the details from the tabs on the right-panel.	 Logo Settings User Context Bar [Project Name] (Workflow) Diagram View Code View Task Properties Functionality Constraints Assign Task 1 Handle Task 1 Determine Eligibility Task 3 Task 2 Calculations Notify Applicant Task Properties within the diagram editor  Logo Settings User Context Bar [Project Name] (Workflow) Task Properties Functionality Constraints Assign Task 1 Handle Task 1 Determine Eligibility Task 3 Task 2 Calculations Notify Applicant Task Functionality within the diagram editor
5	In order to fully describe the behaviour of each node, a service may be bound to a task. For that, the user can use the Service Discovery tool to find the most suitable service.	 Logo Settings User Context Bar [Project Name] (Workflow) Diagram View Code View Task Task 1 Search Recommended Services Service 1 Service 2 Service 3 Service Details URL example_URL Licence MIT licence Framework Spring framework Back Next Service discovery configuration page
6	In case the Service Discovery tool fails to find an appropriate service, the user must develop a new one.	 Logo Settings User Context Bar [Project Name] (Workflow) Task Task 1 Search Recommended Services No similar services found. Service Details URL Licence Framework Back Next No results found by the service discovery
7	The developer is thus directed to the main page of the services and clicks the plus button.	 Logo Settings User Workflows My Workflows Shared With Me Public Workflows Services My Services Shared With Me Public Services Deployments My Deployments Shared With Me My Services Search for a service... Name / URL / Description / Licence / Updated At / Actions Service1 / XXXXXXXXXXXXXXXXXXXXXXX / XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX / MIT License / 05/01/2020 09:45 Service2 / XXXXXXXXXXXXXXXXXXXXXXX / XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX / EPL / 10/02/2020 11:15 Service3 / XXXXXXXXXXXXXXXXXXXXXXX / XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX / EUPL / 27/06/2020 21:45 (...) / (...) / (...) / (...) / (...) The main page of the services

Step	Description	Example
8	The user is prompted to insert the details of the service. When the form is completed, SmartCLIDE creates a GitLab repository to host the source code and a Jenkins CI pipeline to automate the different stages of the development.	 Logo Settings User Workflows My Workflows Shared With Me Public Workflows Services My Services Shared With Me Public Services Deployments My Deployments Shared With Me Insert Service Information Name Enter the name of the service Description Enter a short description of the service Licence Enter the licence to be used Framework Enter the framework to be used Cancel Next The main configuration page of a service
9	An Eclipse Theia instance is launched, where the user develops the source code of the service (either from scratch or from a template, generated by the Smart Assistant). In addition, the Smart Assistant provides code autocompletion, live template recommendations (including suggestions of arguments) and comments generation. The service can then be assigned to the task, back in the diagram editor.	 Logo Settings User Context Bar [Project Name] (Service) Smart Assistant services: Code autocompletion  Logo Settings User Context Bar [Project Name] (Service) Smart Assistant services: Live template recommendation  Logo Settings User Context Bar [Project Name] (Service) Smart Assistant services: Comments generation
10	When the workflow is complete, it is tested through the "Integration" tab in the diagram editor.	 Logo Settings User Context Bar [Project Name] (Workflow) Workflow Assign Task 1 Handle Task 1 Determine Eligibility Task 3 Task 2 Calculations Notify Applicant Workflow integration in the diagram editor

Step	Description	Example
11	Additionally, a security analysis and a vulnerability assessment of the workflow can be executed, through the “Constraints” tab.	 Security analysis page  Vulnerability assessment page
12	Once the workflow is ready for deployment, the user goes to the main page of the deployments and clicks the plus button.	 The main page of the deployments
13	The user is prompted to insert the details of the deployment (e.g., name, workflow/service to be deployed, version of the workflow/service to be deployed, etc.).	 The main configuration page of a deployment
14	The user clicks the pencil button and starts configuring the deployment by choosing the list of resources and corresponding specifications.	 The deployment configuration page

Assign Task 1

Security Issues	Location	Severity
Potential SQL Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 23)	Critical
Potential Command Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 131)	Critical
Potential SQL Injection	Class Name N (line: 150)	Critical
Potential Buffer Overflow	com.digkoss.softwareanalysis.service.git.ProjectService (line: 70)	Critical

Security Issues: 130
Critical: 54
Non - Critical: 76

Security Issues	Location	Severity
Potential SQL Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 23)	Critical
Potential Command Injection	com.digkoss.softwareanalysis.service.git.ProjectService (line: 131)	Critical
Potential SQL Injection	Class Name N (line: 150)	Critical
Potential Buffer Overflow	com.digkoss.softwareanalysis.service.git.ProjectService (line: 70)	Critical

Security Issues: 200
Critical: 83
Non - Critical: 113

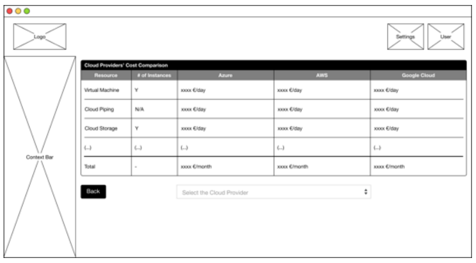
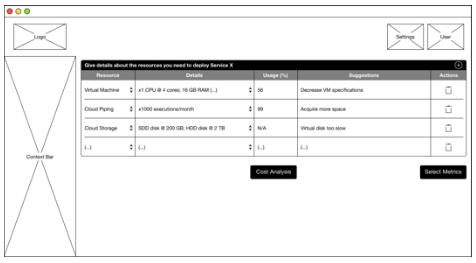
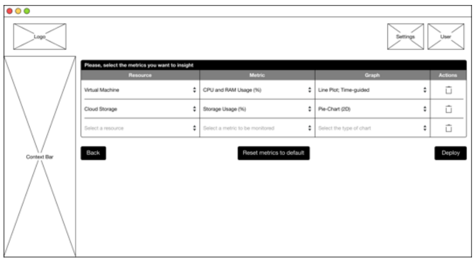
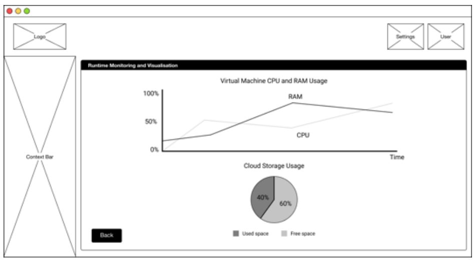
My Deployments

Name	Workflow / Service	Version	State	Updated At	Actions
Deployment3	Service C	3.02	Not Configured	05/01/2020 09:45	
Deployment4	Workflow 1	2.02	Stopped	03/01/2020 19:37	
Deployment2	Service G	3.05	Running	20/11/2019 10:00	
Deployment6	Service A	2.02	Ready	19/10/2019 23:18	
(...)	(...)	(...)	(...)	(...)	

Insert Deployment Information
Name: Enter the name of the deployment
Workflow/Service: Select the workflow or the service to be deployed
Version: Select the version of the workflow or service to be deployed

Give details about the resources you need to deploy Service X

Resource	Details	Usage (%)	Suggestions	Actions
Virtual Machine	x1 CPU @ 4 cores; 16 GB RAM (...)	-	Not enough data was found to give suggestions. Come back later.	
Cloud Piping	x1000 executions/month	-	Not enough data was found to give suggestions. Come back later.	
Cloud Storage	SDD disk @ 200 GB; HDD disk @ 2 TB	-	Not enough data was found to give suggestions. Come back later.	
(...)	(...)	(...)	(...)	

Step	Description	Example
15	The user goes to the cost analysis page and selects the most suitable cloud provider.	 Logo Settings User Context Bar Back Select the Cloud ProviderCloud Providers' Cost Comparison Resource · # of Instances · Azure · AWS · Google Cloud Virtual Machine · Y · xxxx €/day · xxxx €/day · xxxx €/day Cloud Piping · N/A · xxxx €/day · xxxx €/day · xxxx €/day Cloud Storage · Y · xxxx €/day · xxxx €/day · xxxx €/day (...) · (...) · (...) · (...) · (...) Total · - · xxxx €/month · xxxx €/month · xxxx €/month Main page of the cost comparison service
16	Meanwhile, the Smart Assistant's suggestions on how to optimise the resources configurations become available and the user can edit them.	 Logo Settings User Context Bar Cost Analysis Select MetricsGive details about the resources you need to deploy Service X Resource · Details · Usage (%) · Suggestions · Actions Virtual Machine · x1 CPU @ 4 cores; 16 GB RAM (...) · 56 · Decrease VM specifications Cloud Piping · x1000 executions/month · 99 · Acquire more space Cloud Storage · SDD disk @ 200 GB; HDD disk @ 2 TB · N/A · Virtual disk too slow (...) · (...) · (...) · (...) The deployment configuration suggestions
17	The user chooses the runtime metrics to be collected and deploys the workflow.	 Logo Settings User Context Bar Back Reset metrics to default DeployPlease, select the metrics you want to insight Resource · Metric · Graph · Actions Virtual Machine · CPU and RAM Usage (%) · Line Plot; Time-guided Cloud Storage · Storage Usage (%) · Pie-Chart (2D) Select a resource · Select a metric to be monitored · Select the type of chart Runtime metrics selection page
18	The user can monitor the chosen metrics by clicking the bar charts symbol in the main page of the deployments.	 Logo Settings User Context Bar Runtime Monitoring and Visualisation Virtual Machine CPU and RAM Usage 100% 50% 0% RAM CPU Time Cloud Storage Usage 40% 60% Used space Free space BackRuntime metrics monitoring and visualization page

9 Annex C: Eclipse Theia Analysis

Eclipse Theia is an open-source extensible platform for building web- and cloud-based tools and IDE-like products, that can run as desktop applications or in the browser [120] [121]. It is implemented in TypeScript, is based on Microsoft's Visual Studio Code, and emphasizes customizability and extensibility. The project was incepted in 2017, developed by TypeFox and Ericsson, with additional contributions from Red Hat, IBM, Google and Arm Holdings. It was first launched in March 2017. Since May 2018, Theia has been a project of the Eclipse Foundation [122].

C1 Main features

The following are main Eclipse Theia features [123] [124] [125]:

Same interface and capabilities on web and desktop

Theia is a unique IDE platform that supports building desktop and web-based IDEs from the same codebase. From a high-level view, Theia consists of two parts: a frontend, running in a browser, and a backend server running on any host. In a typical web-based scenario, the browser renders and handles all the UI, while the backend is served from anywhere (just as a typical web/cloud application). In a local, desktop scenario, the server part is deployed locally, and Electron is used as a replacement for the Browser. But in both cases, the functionality offered is exactly the same.

Usability

Theia is focused on code, and so it is very slim in terms of footprint of features in the UI. Its usability concept is centered around using the keyboard rather than the mouse. Many features are available via CLI or the command palette only. Some claim it is a hybrid between an IDE and a simple code editor, reflecting the preference of many developers nowadays, especially for web development. However, if the provided usability concept does not fit, thanks to the extensibility and adaptability of the platform one can implement a totally different approach with the same level of support.

Extensibility and adaptability

Theia is meant as a platform, not as a tool itself. This leads to a consistent “extension first” approach, which basically means that everything is an extension, even the core features, which are provided by the project itself. As a result, you can customize almost everything within Theia and even replace core features, if needed. Moreover, Theia offers yet another mechanism for implementing custom functionalities. Through a defined API, Theia allows developers to create so called “Theia plugins”, which are a more limited but also more flexible way of contributing to the richfulness of the end product.

VS Code extension support

Theia is based on Microsoft Visual Studio Code, which makes both share a lot of commonalities, ideas and concepts such as extensibility. VS Code extensions make use of a defined API that can be used for extending VS Code. Theia implements the same API, so many VS Code extensions can also be used in Theia. In a nutshell, VS Code extensions and Theia plugins are conceptually almost the same. Unfortunately, Microsoft prohibits non-Visual Studio products from accessing and/or downloading any binaries from their marketplace. This limitation not only affects Theia and all of its downstream adopters, but also releases based on the open-source code of VS Code, such as VS Codium. For that reason, OpenVSX was created.

Open VSX

Open VSX is an open-source implementation of a VS Code extension registry that has been developed under the umbrella of the Eclipse Foundation. A publicly hosted version of it is available at open-vsx.org. The intention is to create a public registry for open-source VS Code extensions, accessible for everyone. VS Code extension developers are encouraged to push their extensions to Open VSX in addition to Microsoft's marketplace in order to gain better public reach. The registry can also be deployed and adjusted by organizations, to host their own registries within their private networks.

Vendor neutral

VS Code, base of Theia, is hosted by Microsoft; they can define and change the rules of the project at will and at any point in time. They have control over the infrastructure, over contributions and over future directions of the project. On the other hand, Eclipse Theia is hosted at the Eclipse Foundation, a vendor-neutral organization controlled by its members. There are defined rules for governance, e.g., how to elect new committers to the project. There is even a working group focused on cloud-based tools, the "Eclipse Cloud Development Tools" working group, where strategic directions of the ecosystem as well as a branding and marketing strategy are developed among its members, making Theia clearly more "vendor-neutral" compared to VS Code.

C2 Technical view

How to run/launch Theia

- **Option 1:** Download and launch the Theia Blueprint

If you are interested in trying Theia on the desktop, this is the way to go. The Theia project provides a template desktop product called Eclipse Theia Blueprint [122]. It comes with installers for all major operating systems so it is very easy to consume. It assembles a subset of existing Theia features and extensions and is a good way to showcase the capabilities of the platform.Option 1: Launch it from a package.json (plain and simple)

As Eclipse Theia and its extensions are node.js packages and a Theia-based application is a collection of packages, one very simple way to launch Theia plus selected extensions is to create a package.json listing all extensions you want to include and launch it using node CLI. This essentially means you define your own product based on Theia and run it. The advantage of this way of launching Theia is that it is very simple and does not require more than yarn and npm. It also allows you to very easily add and remove extensions.

- **Option 2:** Use a preconfigured docker image

Containerized images already contain any dependencies needed to run the product, which is faster and convenient, especially for using "exotic" extensions of Theia, such as Rust or Go support. Further, if you want to launch Theia on a server, you might prefer to have it encapsulated in a container. A list of preconfigured images can be found on the theiaide dockerhub page targeting specific use cases, e.g. Python or Ruby development, although the most famous one at the moment is definitely the Theia IDE for web developers.

- **Option 3:** Clone, build and run from the sources

This option is very similar to option 1, but instead of getting pre-built Theia packages from npm package registry (npmjs.com), you directly build them from the sources. The required effort and complexity to launch Theia this way is very similar to option 1, but the benefit is that you can now modify the original sources of Theia and immediately see any changes you apply. This option is interesting if you want to have a deeper look at the internals of Theia and extend/fix/contribute something.

- **Option 4:** Use Eclipse Che (host runtimes and workspaces)

Eclipse Che provides a workspace server, meaning a server which can host the development environment of one or several developers. When you move your development environment to the cloud, you will either need to set up those developer runtimes yourself or use something like Eclipse Che. Eclipse Theia is the default IDE for Che since version 7, which means that Che defines an IDE product based on Theia (called Che-Theia). The simplest way to use Theia in Che is by using an openshift.io account and use or create a stack with Eclipse Theia.

- **Option 5:** Use Gitpod (also host runtimes and workspaces)

Gitpod is another solution to host online workspaces including a browser IDE based on Theia. It was one of the first products adopting Theia. Gitpod allows you to describe the required development environment as a script, and will set-up a fresh development environment including the Theia IDE in an amazingly short time. It can even pre build projects to get you ready to code with almost no waiting time, so that you can have a fresh working environment for every new task. Gitpod has been published as an open source project in 2020 and can be self-hosted, but it also offers online workspaces as SaaS.

C3 Theia: Main elements

What you see after starting a Theia application is called the application shell [126]. Similarly, to the Eclipse Workbench, it provides the means to layout the contained building blocks either programmatically or through user interaction, i.e. drag and drop. Compared to the regular Eclipse IDE for establishing a reference, the main elements in the shell are:

- **Panels**

A standard Theia shell is composed of a main area, which is similar to the editor area in Eclipse. Additionally, the shell has three collapsible panels: left, right and bottom. These panels can contain any number of widgets, ordered in tabs with support for splitting them.

- **No Perspectives**

Theia doesn't support the notion of perspectives, although support for it could be added with relative ease. Theia stores the layout of the shell between sessions per workspace. If no previous layout exists, extensions can contribute to the *'initialLayout'*.

- **Widgets**

The elements contained in the panels and the main area are called Widgets. Theia doesn't treat editors differently than views from the application shell perspective, so you can freely arrange editors and views in the panels and main area.

- **Singleton Widgets or Views**

Widgets that should only be opened at most once, like the file navigator or the git widget, are called singleton widgets or simply views as in Eclipse. Implementing such widgets is very easy and Theia provides a convenient base class for that called '*AbstractViewContribution*'. The UI in such widgets is usually implemented using React. But Theia doesn't impose any special JavaScript Framework which allows for reusing every JavaScript library out there.

- **Editors**

Theia embeds VS Code's Monaco editor and wraps it in a thin API, so other web-based editor widgets could possibly be used as well. Language support is usually provided through the Language Server Protocol (LSP) but additional API is available. One can, for instance, decorate an editor with complex inline widgets.

- **Commands**

A command in Theia is a means for users to run some code in the IDE. There are no things like categories, and there is always only one single handler that implements a command. The rare cases (like copy/paste) where multiple different implementations are needed for the same command depending on the context where the command is invoked, are solved through specific contribution points within those handlers.

- **Keybindings**

Keybindings are very important, specifically for developer tools. Theia provides a keybinding registry, where bindings for a command can be registered. Keybindings can be reconfigured by the users either per workspace or per user. The KeybindingContext is the equivalent to what is a Context in the Eclipse IDE. It allows for enabling or disabling keybindings in certain situations. A KeybindingContext can be reused and consists of a single function implementing whether the context is active or not. Unlike the Eclipse IDE, there is no hierarchy of contexts in its declaration as this can easily be modeled by calling a more coarse-grained context from within another one.

- **Menus**

Menus are registered in a central menu registry using paths. A path consists of a list of IDs (strings), where the first segment indicates what menu the children are part of. For instance, the main menu bar is defined with the id 'menubar'. Or if you want to contribute a menu entry to e.g. the editor's context menu, you would start your menu contribution path with the id 'editor_context_menu'. Any view can add new top-level menu paths and thus allow other extensions to contribute to that menu.

The other segments are pointing to groups contained in those menus. The ordering is based on the textual order of the ids on the same level. Which is why group ids usually start with a number. For instance, if you wanted to contribute an item to the open section in the main menu's file menu, the path would be:

```
export const MY_PATH = ['menubar', '1_file', '3_close', 'my_item'];
```

A menu item points to a command id and optionally has callback functions for indicating its enablement status and whether it is visible at all.

C4 Customizing Theia: Plugins

Plugins are pieces of custom functionality that interact with Theia platform using a well-defined API [127] [128] [129]. Plugins are encapsulated in a dedicated process and cannot "harm" the stability of the hosting product. Moreover, plugins can be installed at runtime. However, the encapsulation has one drawback: you can only extend or adapt what is exposed in the API, which means that plugins are significantly "less powerful" in terms of what they can do.

Probably the most important thing to know about developing a plugin for Eclipse Theia is that the plugin mechanism and the API available for plugins are almost identical to VS Code extensions. In fact, VS Code extensions can usually be used in Theia. Therefore, most documentation and tutorials about extending VS Code can be applied to Theia too.

Theia provides some tooling around developing its own plugins and is, therefore, the best tool for doing so. The plugin tooling within Theia allows directly running an instance of a plugin under development. As a plugin cannot exist on its own, the command "Hosted Plugin: Start Instance" will start a new Theia instance, which is identical to your development environment, but that additionally hosts the deployed plugin under development. To not get confused by the two Theia instances running in this scenario, the status bar at the bottom indicates which Theia instance you are looking at. Other commands allow you to stop the plugin instance, to restart it and to debug it, which enables breakpoints and source code inspection of a plugin under development.

Plugin APIs

The Eclipse Theia plugin API consists of two conceptual parts, the programmatic TypeScript API and a declarative API, to be used via the package.json file [129].

TypeScript API

The TypeScript API is very easy to use and is typically exposed under a single namespace via the module import '@theia/plugin' (typically bound to a variable "theia"). The following is an overview of its sub-namespaces:

- commands: To add executable commands to the workbench and editors, which can be triggered via the command palette, menu items or key bindings
- comment: To create a CommentController allowing more sophisticated commenting support in the code editor
- debug: Anything related to extending the debug capabilities, but also to interact with breakpoints and debug sessions
- env: Provides access to the environment that Theia is running in, e.g. the clipboard or the user-specific language setting.
- languageServer: To register a custom language server.
- languages: Everything else language-related, e.g. to implement a custom "Call hierarchy provider" or a custom formatter.

- plugins: To interact with other installed plugins. Plugin can expose an API to be used by others.
- scm: To add support for a source control system
- tasks: To contribute or interact with tasks, e.g. being notified if a task has been completed. Tasks are like jobs, e.g. a compilation could be a task
- window: To interact and contribute to the workbench, e.g. to show any kind of dialogue, change the status bar or react to an opened terminal
- workspace: To interact with the workspace, e.g. to react on save or browsing folders.

There are two ways of using this API. The first one is to call a specific function to trigger an action. A good example of this is triggering the notification in the 'hello world' example before. The second type of API call is to register a "contribution" to Theia. This basically means something is added to Theia, typically a callback, that is triggered by "some other action" in Theia. "Some other action" in this context can mean the direct invocation of a command by the user (as in the 'hello world' example), but also more indirect use cases, e.g. if the user triggers "call hierarchy" in the code editor. The second type typically uses functions in the API starting with "register…", e.g. the "registerCommand" call in the hello world example.

Declarative API

Besides the programmatic Typescript API, Theia provides a declarative API for plugins via the package.json, called "contribution points". Again, this is similar to VS Code [130], so you can refer to the VS Code documentation about contribution points. Declarative contribution points make some use cases simpler compared to code, especially configurations and "wiring things together". Technically even more important, these contributions can be simply parsed by Theia, without executing any code. This way is faster and more robust from the viewpoint of Theia. As can be seen, the plugin API of Eclipse Theia is a mix of declarative contributions (in the package.json), use case specific files (i.e. for configuration) and code using the TypeScript API (for the implementation of the new features).

Customizing Theia: Extensions

Theia Extensions, in contrast to Plugins [127], are pieces of functionality that are built into the Theia platform itself by means of dependency injection, and allow modifying, adapting or extending whatever needed [128]. Needless to say, extensions are much more powerful in terms of what can be done, but that also means it is much easier to break the platform itself. An extension is technically a node package, so it contains a package.json in its root folder, and so the IDE platform being created can (must?) include it in its dependencies.

A Theia extension consist of three important pieces:

- The package.json, which additionally to its regular role in Node.js, such as its name and dependencies, contains some Theia-specific metadata:
  - the keyword "theia-extension", which marks the node package as a Theia extension. Theia will run through all packages on start-up and process those that are marked as an extension.
  - a list of "theiaExtensions", enumerating TypeScript files (modules) that act as entry points in the extension project.
- The actual "modules", which wire your extensions to the Theia framework and other extensions.
- The implementation of any features that are provided by your extension

The responsibility of a Theia extension module is to wire the extension features with the underlying framework and/or other extensions. In fact, as the underlying framework also just consists of extensions, these two cases are actually the same.

Dependency injection

To achieve communication between extensions, Theia uses dependency injection, implemented on the library "inversify.js". If you are familiar with the Eclipse Platform, inversify replaces extension points and OSGi declarative services. If you are familiar with Google Guice, inversify is pretty much the same with only small differences [131].

From a high-level point of view, you can think of this mechanism as a global bulletin board. Any extension can place or retrieve items from this board. As an example, an extension could place a "menu contribution" on the board, expressing that it wants to add a menu item to a specific menu. The Theia-internal extension, which is responsible for creating the menu at runtime can browse the board, pick up all contributed menu items and render the menu accordingly. The technical term for this "board" using dependency injection is a "context". To be able to retrieve the correct objects from within the context, they are registered under a specific "key", which is conceptually a TypeScript interface (technically a "symbol"). The diagram below shows the registration of a contribution along a menu contribution example. The "extension-frontend-module" registers an object "ExtensionMenuContribution" in the context using the interface "MenuContribution" as a key. The Theia core extension picks up those menu contributions, also using the interface as a key.

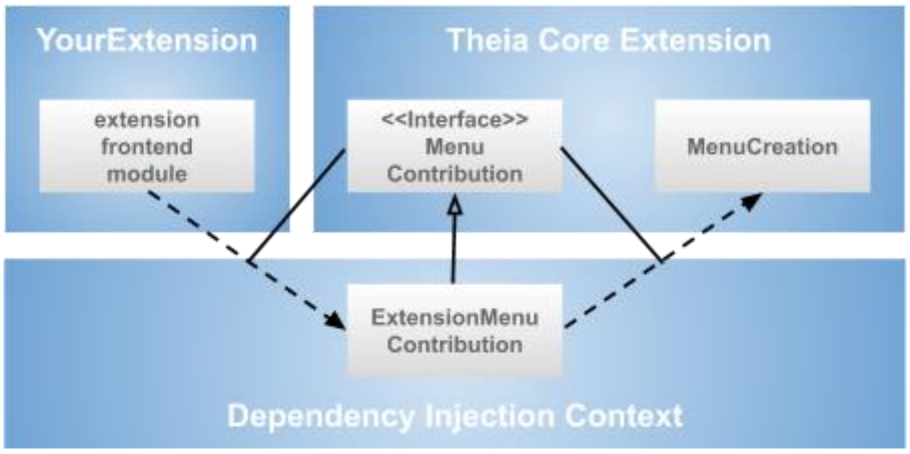

YourExtension
Theia Core Extension
extension frontend module
<<Interface>> Menu Contribution
MenuCreation
ExtensionMenu Contribution
Dependency Injection Context

Figure 113: Theia extension mechanism

Let's look at how this "wiring" looks like in code. If you open the "extension-frontend-module" of the example extension, you will see the registration of two contributions, a CommandContribution and a MenuContribution:

```
bind(CommandContribution).to(ExtensionCommandContribution);

bind(MenuContribution).to(ExtensionMenuContribution);
```

As can be seen in the code listing, the registration points to two extension-specific classes, the "ExtensionCommandContribution" and the "ExtensionMenuContribution". These actually implement the desired behavior; in the example, a "hello world" action.

Contributions

The implementation of contributions is use case specific and usually specified by the contribution interface, e.g. "MenuContribution". There, you implement the functions required by the Contribution interface, and link to the actual behavior and functionality of your extension.

C5 Why Eclipse Theia

The consortium was faced at the very beginning, with the need to decide whether to use a full-fledged IDE or a web-embeddable code editor with syntax highlighting and some other features.

Shortly after starting to investigate the matter, it became clear that an embeddable WYSIWYG text editor with syntax highlighting would not be suitable for the job, since we wanted to integrate a bunch of functionalities that would simply not be possible to integrate in such kind of editors. So, for example, while code completion based on snippets, syntax highlighting, or version control integration is quite well supported on existing web-based code editors, like Codiad[45] or IceCoder[46] among others, other more

[45] http://codiad.com/
[46] https://icecoder.net/

advanced features like refactoring support, debugging, or code generation are either not supported at all or have limited support. Also, extended functionalities for those editors are relatively scarce, with a community of users mainly focused on web development.

On the other hand, consortium members didn´t want to deal with the complexities that come with the usage of a complete, full-blown desktop IDE, since the base idea was to provide a cloud-based working environment, ready to be used, not requiring anything to be downloaded or installed locally, so platforms like Eclipse RCP[47] or Netbeans RCP and similar got discarded as well.

Microsoft's VS Code, base of Eclipse Theia, was also considered as an option, as it provides most of the features that the consortium was looking for, as well as a massive extension marketplace, and a big and well established community of developers and adopters. But the fact that it is closed source and under control of Microsoft makes it a not viable solution [132] [123].

Eclipse Theia came out to be the best option, combining both worlds. It is lightweight and simple enough to be run on a browser, but can also be downloaded and run locally. It is highly customizable by its plugin and extension mechanisms, it is supported and backed by several important organizations, and offers a big amount of ready made extensions, developed open source by a very active community, with focus on development in general. The fact that it shares the same API as VS Code makes it possible to use any of the existing extensions created, although the developer must publish it first on the Eclipse Theia marketplace in order for it to be freely redistributable.

Moreover, its election as default IDE for some workspace-handling solutions like Eclipse Che, its adaptability and extensibility made it the perfect choice for the development of the SmartCLIDE project. We can extract the following list of advantages and disadvantages of Theia as platform:

Table 51: Advantages and disadvantages of Theia

Advantages	Disadvantages
▪ Fully customizable and adaptable. Being a platform intended to serve as base for development products, Theia makes an excellent job on offering the most flexible base. ▪ Branding is perfectly supported, as can be seen on the products from companies like Arduino, ARM, or Gitpod ▪ Perfectly embeddable into other web-based environment ▪ Easy to add new functionalities that could be further extended with plugins ▪ Support for graphical modelling and BPMN descriptions via Eclipse GLSP	▪ Relatively steep learning curve. ▪ The extension mechanism is poorly documented. The extension points have to be looked for by actually digging into the platform's own source code, which typically implies spending time debugging and in trial-and-error. ▪ Lack of out-of-the-box graphical components for UI interactions which makes it hard to share the look and feel among extensions. ▪ Many of the existing extensions are published for VS Code only, which in practice means that they would not be directly usable in a commercial/redistributable Theia-based product. ▪ Theia has its own plugin API, which extends and attempts to override that of VS Code, causing that eventually some of the existing extensions might not run in Theia

[47] https://wiki.eclipse.org/Rich_Client_Platform

10 References

[1] "D1.4 The SmartCLIDE Concept," Public Report, Oct. 2020.

[2] "D1.5 The SmartCLIDE Architecture," Public Report, Oct. 2020.

[3] "D2.1 Innovative Approaches and Features on Services Discovery, Creation, Composition and Deployment," Public Deliverable, Aug. 2021.

[4] "Manifesto for Agile Software Development." https://agilemanifesto.org/ (accessed Sep. 08, 2021).

[5] 14:00-17:00, "ISO/IEC/IEEE 42010:2011," *ISO*. https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/05/50508.html (accessed Sep. 08, 2021).

[6] "Extensions for Visual Studio family of products | Visual Studio Marketplace." https://marketplace.visualstudio.com/vscode (accessed Sep. 07, 2021).

[7] "Kite - Free AI Coding Assistant and Code Auto-Complete Plugin," *Code Faster with Kite*. https://www.kite.com/ (accessed Sep. 07, 2021).

[8] "Alizadehsani, Z., Gomez, E. G., Ghaemi, H., González, S. R., Jordan, J., Fernández, A., & Pérez-Lancho, B. (2021, April). Modern Integrated Development Environment (IDEs). In Sustainable Smart Cities and Territories International Conference (pp. 274-288). Springer, Cham."

[9] M. Asaduzzaman, C. K. Roy, and K. A. Schneider, "PARC: Recommending API methods parameters," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2015, pp. 330–332. doi: 10.1109/ICSM.2015.7332481.

[10] A. Ohiomah, P. Andreev, and M. Benyoucef, "A Review of Big Data Predictive Analytics in Information Systems Research," *Information Systems*, p. 23, 2017.

[11] S. E. El-Sayyad, A. I. Saleh, and H. A. Ali, "A new semantic web service classification (SWSC) strategy," *Cluster Comput*, vol. 21, no. 3, pp. 1639–1665, Sep. 2018, doi: 10.1007/s10586-018-2367-9.

[12] "Alizadehsani, Z., Gomez, E. G., Ghaemi, H., González, S. R., Jordan, J., Fernández, A., & Pérez-Lancho, B. (2021, April). Modern Integrated Development Environment (IDEs). In Sustainable Smart Cities and Territories International Conference(pp. 274-288). Springer, Cham.".

[13] "Dynatrace | The Leader in Automatic and Intelligent Observability," *Dynatrace*. https://www.dynatrace.com/ (accessed Sep. 21, 2021).

[14] "Cloud Computing Services | Microsoft Azure." https://azure.microsoft.com/en-us/ (accessed Sep. 21, 2021).

[15] K. W. Church, "Emerging trends: I did it, I did it, I did it, but. . .," *Natural Language Engineering*, vol. 23, no. 3, pp. 473–480, May 2017, doi: 10.1017/S1351324917000067.

[16] "fastText." https://fasttext.cc/index.html (accessed Sep. 21, 2021).

[17] "RapidAPI - The Next Generation API Platform," *RapidAPI*. https://rapidapi.com/ (accessed Aug. 29, 2021).

[18] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A Neural Probabilistic Language Model," *Journal of Machine Learning Research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[19] D. D. Rathod, "PERFORMANCE EVALUATION OF RESTFUL WEB SERVICES AND SOAP / WSDL WEB SERVICES," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 7, Art. no. 7, Aug. 2017, doi: 10.26483/ijarcs.v8i7.4349.

[20] "Code Faster with AI Code Completions." https://www.tabnine.com/about?utm_term=&utm_campaign=&utm_source=adwords&utm_medium=ppc&hsa_acc=4311736126&hsa_cam=14293437790&hsa_grp=&hsa_ad=&hsa_src=x&hsa_tgt=&

hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gclid=CjwKCAjwhaaKBhBcEiwA8acsHB3g8f wVmy2V3_zfAV-4H4tknbqBz8EJqrKyxr4H7Zy7GdnviQBG3RoCx5MQAvD_BwE (accessed Sep. 21, 2021).

[21]“Kite - Free AI Coding Assistant and Code Auto-Complete Plugin,” *Code Faster with Kite*. https://www.kite.com/ (accessed Aug. 29, 2021).

[22]M. Allamanis and C. Sutton, “Mining source code repositories at massive scale using language modeling,” in *2013 10th Working Conference on Mining Software Repositories (MSR)*, May 2013, pp. 207–216. doi: 10.1109/MSR.2013.6624029.

[23]“ProgrammableWeb,” *ProgrammableWeb*. https://www.programmableweb.com/ (accessed Sep. 10, 2021).

[24]“RapidAPI - The Next Generation API Platform,” *RapidAPI*. https://rapidapi.com/ (accessed Sep. 10, 2021).

[25]P. Willett, “The Porter stemming algorithm: then and now,” *Program*, vol. 40, no. 3, pp. 219–223, Jul. 2006, doi: 10.1108/00330330610681295.

[26]M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, “A Survey of Machine Learning for Big Code and Naturalness,” *ACM Comput. Surv.*, vol. 51, no. 4, p. 81:1-81:37, Jul. 2018, doi: 10.1145/3212695.

[27]*Codeprep*. giganticode, 2021. Accessed: Sep. 21, 2021. [Online]. Available: https://github.com/giganticode/codeprep

[28]W. Li *et al.*, “Semantic-based web service discovery and chaining for building an Arctic spatial data infrastructure,” *Computers & Geosciences*, vol. 37, no. 11, pp. 1752–1762, Nov. 2011, doi: 10.1016/j.cageo.2011.06.024.

[29]“ERGOT: A Semantic-Based System for Service Discovery in Distributed Infrastructures.” https://ieeexplore.ieee.org/document/5493471 (accessed Aug. 29, 2021).

[30]Y. Yang *et al.*, “ServeNet: A Deep Neural Network for Web Services Classification,” *2020 IEEE International Conference on Web Services (ICWS)*, pp. 168–175, Oct. 2020, doi: 10.1109/ICWS49710.2020.00029.

[31]K. Punitha, “A Novel Mixed Wide and PSO-Bi-LSTM-CNN Model for the Effective Web Services Classification,” *WEB*, vol. 17, no. 2, pp. 218–237, Dec. 2020, doi: 10.14704/WEB/V17I2/WEB17026.

[32]R. Nisa and U. Qamar, “A text mining based approach for web service classification,” *Information systems and e-business management : ISeB*, vol. 13, no. 4, Nov. 2015.

[33]M. Crasso, A. Zunino, and M. Campo, “AWSC: An approach to Web service classification based on machine learning techniques,” *Int. Artif.*, vol. 12, no. 37, p. 561, Mar. 2008, doi: 10.4114/ia.v12i37.955.

[34]C. Sanchez Sanchez, E. Villatoro Tello, A. G. Ramirez De La Rosa, H. Jimenez Salazar, and D. E. Pinto Avendaño, “WSDL Information selection for improving web service classification,” 2017, Accessed: Aug. 29, 2021. [Online]. Available: http://ilitia.cua.uam.mx:8080/jspui/handle/123456789/503

[35]M. M. Rahman, C. K. Roy, and D. Lo, “Automatic query reformulation for code search using crowdsourced knowledge,” *Empirical Softw. Engg.*, vol. 24, no. 4, pp. 1869–1924, Aug. 2019, doi: 10.1007/s10664-018-9671-0.

[36]“GloVe: Global Vectors for Word Representation.” https://nlp.stanford.edu/projects/glove/ (accessed Sep. 21, 2021).

[37]C. Ragkhitwetsagul and J. Krinke, “Siamese: scalable and incremental code clone search via multiple code representations,” *Empir Software Eng*, vol. 24, no. 4, pp. 2236–2284, Aug. 2019, doi: 10.1007/s10664-019-09697-7.

[38] T. Beltramelli, “pix2code: Generating Code from a Graphical User Interface Screenshot,” in *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, New York, NY, USA, Jun. 2018, pp. 1–6. doi: 10.1145/3220134.3220135.

[39] S. E. V. and P. Samuel, “Automatic Code Generation From UML State Chart Diagrams,” *IEEE Access*, vol. 7, pp. 8591–8608, 2019, doi: 10.1109/ACCESS.2018.2890791.

[40] P. I and J. Paulose, “Prediction of Answer Keywords using Char-RNN,” *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 3, Art. no. 3, Jun. 2019, doi: 10.11591/ijece.v9i3.pp2164-2176.

[41] “Autocompletion with deep learning.” https://www.tabnine.com/ (accessed Aug. 29, 2021).

[42] V. Jain, P. Agrawal, S. Banga, R. Kapoor, and S. Gulyani, “Sketch2Code: Transformation of Sketches to UI in Real-time Using Deep Neural Network,” *arXiv:1910.08930 [cs, eess]*, Oct. 2019, Accessed: Aug. 29, 2021. [Online]. Available: http://arxiv.org/abs/1910.08930

[43] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, “DeepCoder: Learning to Write Programs,” *arXiv:1611.01989 [cs]*, Mar. 2017, Accessed: Aug. 29, 2021. [Online]. Available: http://arxiv.org/abs/1611.01989

[44] R. F. G. Silva, C. K. Roy, M. M. Rahman, K. A. Schneider, K. Paixao, and M. de Almeida Maia, “Recommending comprehensive solutions for programming tasks by mining crowd knowledge,” in *Proceedings of the 27th International Conference on Program Comprehension*, Montreal, Quebec, Canada, May 2019, pp. 358–368. doi: 10.1109/ICPC.2019.00054.

[45] S. Jörges, Ed., *Construction and Evolution of Code Generators: A Model-Driven and Service-Oriented Approach*. Berlin Heidelberg: Springer-Verlag, 2013. doi: 10.1007/978-3-642-36127-2.

[46] M. M. Rahman, C. Roy, and D. Lo, “RACK: Automatic API Recommendation Using Crowdsourced Knowledge,” *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, doi: 10.1109/SANER.2016.80.

[47] “Deep Code Search | IEEE Conference Publication | IEEE Xplore.” https://ieeexplore.ieee.org/abstract/document/8453172 (accessed Sep. 21, 2021).

[48] X. Gu, H. Zhang, and S. Kim, “Deep code search,” in *Proceedings of the 40th International Conference on Software Engineering*, Gothenburg Sweden, May 2018, pp. 933–944. doi: 10.1145/3180155.3180167.

[49] “Codota - AI Code Completions for your IDE, Avalible Search for Java Code on 6.” https://www.codota.com (accessed Aug. 29, 2021).

[50] C. Zhang *et al.*, “Automatic parameter recommendation for practical API usage,” in *2012 34th International Conference on Software Engineering (ICSE)*, Jun. 2012, pp. 826–836. doi: 10.1109/ICSE.2012.6227136.

[51] A. T. Nguyen *et al.*, “API code recommendation using statistical learning from fine-grained changes,” in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, New York, NY, USA, Nov. 2016, pp. 511–522. doi: 10.1145/2950290.2950333.

[52] B. Uyanik and V. H. Şahin, “A template-based code generator for web applications,” 2020, doi: 10.3906/elk-1910-44.

[53] S. Proksch, J. Lerch, and M. Mezini, “Intelligent Code Completion with Bayesian Networks,” *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 1, p. 3:1-3:31, Dec. 2015, doi: 10.1145/2744200.

[54] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, “When deep learning met code search,” in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, New York, NY, USA, Aug. 2019, pp. 964–974. doi: 10.1145/3338906.3340458.

[55] B. A. Campbell and C. Treude, “NLP2Code: Code Snippet Content Assist via Natural Language Tasks,” *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, doi: 10.1109/ICSME.2017.56.

[56] A. Vaswani *et al.*, “Attention Is All You Need,” *arXiv:1706.03762 [cs]*, Dec. 2017, Accessed: Sep. 21, 2021. [Online]. Available: http://arxiv.org/abs/1706.03762

[57] “Better Language Models and Their Implications,” *OpenAI*, Feb. 14, 2019. https://openai.com/blog/better-language-models/ (accessed Sep. 21, 2021).

[58] “GPT-3: Its Nature, Scope, Limits, and Consequences | SpringerLink.” https://link.springer.com/article/10.1007/s11023-020-09548-1 (accessed Sep. 21, 2021).

[59] “distilgpt2 · Hugging Face.” https://huggingface.co/distilgpt2 (accessed Sep. 21, 2021).

[60] “QWS Dataset.” https://qwsdata.github.io/citations.html (accessed Aug. 29, 2021).

[61] “scikit-learn: machine learning in Python — scikit-learn 0.24.2 documentation.” https://scikit-learn.org/stable/ (accessed Sep. 21, 2021).

[62] “TensorFlow,” *TensorFlow*. https://www.tensorflow.org/ (accessed Sep. 21, 2021).

[63] “Keras: the Python deep learning API.” https://keras.io/ (accessed Sep. 21, 2021).

[64] “Natural Language Toolkit — NLTK 3.6.3 documentation.” https://www.nltk.org/ (accessed Sep. 21, 2021).

[65] “spaCy · Industrial-strength Natural Language Processing in Python.” https://spacy.io/ (accessed Sep. 21, 2021).

[66] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” *arXiv:1910.01108 [cs]*, Feb. 2020, Accessed: Sep. 21, 2021. [Online]. Available: http://arxiv.org/abs/1910.01108

[67] *tree-sitter*. tree-sitter, 2021. Accessed: Sep. 21, 2021. [Online]. Available: https://github.com/tree-sitter/tree-sitter

[68] GitLab Docs, “REST API resources.” [Online]. Available: https://docs.gitlab.com/ee/api/api_resources.html

[69] GitLab Docs, “GraphQL API.” [Online]. Available: https://docs.gitlab.com/ee/api/graphql/

[70] “Search API | Elasticsearch Guide [7.14] | Elastic.” https://www.elastic.co/guide/en/elasticsearch/reference/current/search-search.html (accessed Aug. 29, 2021).

[71] “Document APIs | Elasticsearch Guide [7.14] | Elastic.” https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html (accessed Sep. 08, 2021).

[72] “International Committee for Information for Information Technology Standards - Cyber security technical committee,” 1. American National Standard for Information Technology Next Generation Access Control (NGAC).” ANSI, INCITS 565-2020, April 2020.

[73] “NGAC policy tool, policy server, and EPP Release note for v0.4.7 development version,” Rance DeLong, The Open Group, July 2021.

[74] A. Cimatti, C. Tian, and S. Tonetta, “NuRV: A nuXmv Extension for Runtime Verification,” in *Runtime Verification*, Cham, 2019, pp. 382–392. doi: 10.1007/978-3-030-32079-9_23.

[75] R. Cavada *et al.*, “The nuXmv Symbolic Model Checker,” in *Computer Aided Verification*, Cham, 2014, pp. 334–342. doi: 10.1007/978-3-319-08867-9_22.

[76] A. Cimatti, C. Tian, and S. Tonetta, “Assumption-Based Runtime Verification with Partial Observability and Resets,” in *Runtime Verification*, Cham, 2019, pp. 165–184. doi: 10.1007/978-3-030-32079-9_10.

[77] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, “Patterns in property specifications for finite-state verification,” in *Proceedings of the 21st international conference on Software engineering*, New York, NY, USA, May 1999, pp. 411–420. doi: 10.1145/302405.302672.

[78] M. Siavvas, E. Gelenbe, D. Kehagias, and D. Tzovaras, “Static Analysis-Based Approaches for Secure Software Development,” in *Security in Computer and Information Sciences*, Cham, 2018, pp. 142–157. doi: 10.1007/978-3-319-95189-8_13.

[79] E. Gelenbe *et al.*, “NEMESYS: Enhanced Network Security for Seamless Service Provisioning in the Smart Mobile Ecosystem,” in *Information Sciences and Systems 2013*, Cham, 2013, pp. 369–378. doi: 10.1007/978-3-319-01604-7_36.

[80] S. M. Ghaffarian and H. R. Shahriari, “Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey,” *ACM Comput. Surv.*, vol. 50, no. 4, p. 56:1-56:36, Aug. 2017, doi: 10.1145/3092566.

[81] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota, Jun. 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423.

[82] H. Avram, S. Riccardo, J. Wouter, and W. James, “Software Vulnerability Prediction using Text Analysis Techniques,” InProceedings of the 4th international workshop on Security measurements and metrics 2012 Sep 21 (pp. 7-10).

[83] “Krsul IV. Software vulnerability analysis.,” Purdue University; 1998.

[84] M. Howard, D. LeBlanc, and J. Viega, *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*, 1st edition. New York: McGraw-Hill Education, 2009.

[85] P. Morrison, D. Moye, R. Pandita, and L. Williams, “Mapping the field of software life cycle security metrics,” *Information and Software Technology*, vol. 102, pp. 146–159, Oct. 2018, doi: 10.1016/j.infsof.2018.05.011.

[86] M. Green and M. Smith, “Developers are Not the Enemy!: The Need for Usable Security APIs,” *IEEE Security Privacy*, vol. 14, no. 5, pp. 40–46, Sep. 2016, doi: 10.1109/MSP.2016.111.

[87] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, “Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities,” *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 772–787, Nov. 2011, doi: 10.1109/TSE.2010.81.

[88] I. Chowdhury and M. Zulkernine, “Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities,” *Journal of Systems Architecture*, vol. 57, no. 3, pp. 294–313, Mar. 2011, doi: 10.1016/j.sysarc.2010.06.003.

[89] M. Zhang, X. de Carné de Carnavalet, L. Wang, and A. Ragab, “Large-Scale Empirical Study of Important Features Indicative of Discovered Vulnerabilities to Assess Application Security,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 9, pp. 2315–2330, Sep. 2019, doi: 10.1109/TIFS.2019.2895963.

[90] M. Siavvas, D. Tsoukalas, M. Jankovic, D. Kehagias, and D. Tzovaras, “Technical debt as an indicator of software security risk: a machine learning approach for software development enterprises,” *Enterprise Information Systems*, vol. 0, no. 0, pp. 1–43, Sep. 2020, doi: 10.1080/17517575.2020.1824017.

[91] “A hierarchical model for quantifying software security based on static analysis alerts and software metrics | SpringerLink.” https://link.springer.com/article/10.1007/s11219-021-09555-0 (accessed Sep. 04, 2021).

[92] “What are Message Brokers?,” Aug. 11, 2021. https://www.ibm.com/cloud/learn/message-brokers (accessed Sep. 04, 2021).

[93] "What are Message Brokers? | IBM." https://www.ibm.com/cloud/learn/message-brokers (accessed Sep. 04, 2021).

[94] "What is RabbitMQ? | A Quick Glance of What is RabbitMQ?," *EDUCBA*, Feb. 22, 2020. https://www.educba.com/what-is-rabbitmq/ (accessed Sep. 04, 2021).

[95] "Management Plugin — RabbitMQ." https://www.rabbitmq.com/management.html (accessed Sep. 04, 2021).

[96] "Part 1: RabbitMQ for beginners - What is RabbitMQ? - CloudAMQP." https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html (accessed Sep. 04, 2021).

[97] "Rabbitmq - Official Image | Docker Hub." https://hub.docker.com/_/rabbitmq (accessed Sep. 06, 2021).

[98] "Using WebSocket Requests," *Postman Learning Center*. https://learning.postman.com (accessed Sep. 06, 2021).

[99] "RabbitMQ Tutorials — RabbitMQ." https://www.rabbitmq.com/getstarted.html (accessed Sep. 06, 2021).

[100] "Server Administration Guide." https://www.keycloak.org/docs/latest/server_admin/index.html (accessed Sep. 04, 2021).

[101] "What Is Identity and Access Management? Guide to IAM," *SearchSecurity*. https://searchsecurity.techtarget.com/definition/identity-access-management-IAM-system (accessed Sep. 04, 2021).

[102] D. Strom, "What is IAM? Identity and access management explained," *CSO Online*, Apr. 08, 2021. https://www.csoonline.com/article/2120384/what-is-iam-identity-and-access-management-explained.html (accessed Sep. 04, 2021).

[103] "Identity and Access Management (IAM)," *Fortinet*. https://www.fortinet.com/resources/cyberglossary/identity-and-access-management (accessed Sep. 04, 2021).

[104] "What Keycloak Is and What It Does? - DZone Security," *dzone.com*. https://dzone.com/articles/what-is-keycloak-and-when-it-may-help-you (accessed Sep. 04, 2021).

[105] CoMakeIT, "A Quick Guide To Using Keycloak For Identity And Access Management," Aug. 02, 2018. https://www.comakeit.com/blog/quick-guide-using-keycloak-identity-access-management/ (accessed Sep. 04, 2021).

[106] "Keycloak: Secure & easy Identity & Access Management | Single Sign-On & IAM | AOE." https://www.aoe.com/en/expertise/keycloak.html (accessed Sep. 04, 2021).

[107] N. Köbler, "Keycloak Admin Client(s) - multiple ways to manage your SSO system | Niko Köbler - Keycloak Expert, Software-Architect & Trainer." https://www.n-k.de/2016/08/keycloak-admin-client.html (accessed Sep. 04, 2021).

[108] "Managing identities and authorizations :: Eclipse Che Documentation." https://www.eclipse.org/che/docs/che-7/administration-guide/managing-identities-and-authorizations/ (accessed Sep. 04, 2021).

[109] "Getting Started Guide." https://www.keycloak.org/docs/latest/getting_started/ (accessed Sep. 04, 2021).

[110] "Unit Testing Best Practices: 9 to Ensure You Do It Right," *AI-driven E2E automation with code-like flexibility for your most resilient tests*, Mar. 11, 2021. https://www.testim.io/blog/unit-testing-best-practices/ (accessed Sep. 07, 2021).

[111] "What is Integration Testing (Tutorial with Integration Testing Example)." https://www.softwaretestinghelp.com/what-is-integration-testing/ (accessed Sep. 07, 2021).

[112] "What is API Testing? | Definition, Benefits, Types & Tool." https://www.katalon.com/api-testing/ (accessed Sep. 07, 2021).

[113] Atlassian, "The different types of testing in software," *Atlassian*. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing (accessed Sep. 07, 2021).

[114] "Software Testing - Levels." https://www.tutorialspoint.com/software_testing/software_testing_levels.htm (accessed Sep. 07, 2021).

[115] "What is Usability Testing?," *The Interaction Design Foundation*. https://www.interaction-design.org/literature/topics/usability-testing (accessed Sep. 07, 2021).

[116] "JUnit 5." https://junit.org/junit5/ (accessed Sep. 07, 2021).

[117] "Maven Surefire Plugin – Introduction." https://maven.apache.org/surefire/maven-surefire-plugin/ (accessed Sep. 07, 2021).

[118] "Prolog Unit Tests." https://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/plunit.html%27) (accessed Sep. 07, 2021).

[119] SmartCLIDE Confidential Report, "D1.3 Use Case Scenarios," Oct. 2020.

[120] Wikipedia, "Eclipse Theia." [Online]. Available: https://en.wikipedia.org/wiki/Eclipse_Theia

[121] Eclipse Foundation, "Theia - An Open, Flexible and Extensible Cloud & Desktop IDE Platform." [Online]. Available: https://theia-ide.org/

[122] Eclipse Foundation, "Eclipse Theia Project." [Online]. Available: https://projects.eclipse.org/projects/ecd.theia

[123] "Eclipse Source, Eclipse Theia FAQ." [Online]. Available: https://eclipsesource.com/blogs/2019/12/24/eclipse-theia-ide-faq/

[124] "Eclipse Source, Eclipse Theia vs. VS Code." [Online]. Available: https://eclipsesource.com/blogs/2019/12/06/the-eclipse-theia-ide-vs-vs-code/

[125] "dev.to, Theia 1.0 - Finally a good browser IDE." [Online]. Available: https://dev.to/svenefftinge/theia-1-0-finally-a-good-browser-ide-3ok0

[126] Eclipse Foundation, "The busy RCP's developer guide to Eclipse Theia." [Online]. Available: https://www.eclipse.org/community/eclipse_newsletter/2018/october/theia.php

[127] Eclipse Source, "Eclipse Theia extensions vs. plugins vs. Che-Theia plugins." [Online]. Available: https://eclipsesource.com/blogs/2019/10/10/eclipse-theia-extensions-vs-plugins-vs-che-theia-plugins/

[128] J. Helming *et al.*, "How to create/develop an Eclipse Theia IDE extension," *EclipseSource*, Nov. 21, 2019. https://eclipsesource.com/blogs/2019/11/21/how-to-create-develop-an-eclipse-theia-ide-extension/ (accessed Sep. 07, 2021).

[129] Eclipse Source, "How to create/develop an Eclipse Theia IDE plugin." [Online]. Available: https://eclipsesource.com/blogs/2019/10/17/how-to-add-extensions-and-plugins-to-eclipse-theia/

[130] Microsoft, "Visual Studio Code extension API." [Online]. Available: https://code.visualstudio.com/api

[131] Eclipse Source, "How to inversify() in Eclipse Theia." [Online]. Available: https://eclipsesource.com/blogs/2018/11/28/how-to-inversify-in-eclipse-theia/

[132] "JAXEnter, What Theia is all about." [Online]. Available: https://jaxenter.com/theia-ide-efftinge-interview-134467.html