Development of Service Workbench for Software Delivery Platform

Development of Service Workbench for Software Delivery Platform

Mayank Sharma

October 2023

Eindhoven University of Technology
Stan Ackermans Institute – Software Technology

EngD Report: 2023/091

Confidentiality Status:
Not Confidential

Partners	Thermo Fisher Scientific	Eindhoven University of Technology
Steering Group	ir. E. Algra, EngD ir. H.T.G. Weffers, EngD	
Date	October 2023	

Eindhoven University of Technology

Composition of the Thesis Evaluation Committee:

Chair:	Prof. Dr. M.G.J. van den Brand
Members:	ir. E. Algra, EngD
	ir. H.T.G. Weffers, EngD
	T.E. Molina, MSc
	Dr. ir. L.G.W.A. Cleophas

The design that is described in this report has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct.

Contact Address
Eindhoven University of Technology
Department of Mathematics and Computer Science
MF 5.080A, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
+31 402474334

Partnership
This project was supported by Eindhoven University of Technology and Thermo Fisher Scientific.

Published by
Eindhoven University of Technology
Stan Ackermans Institute

EngD-report
2023/091

Preferred reference
Development of Service Workbench for Software Delivery Platform Eindhoven University of Technology, EngD Report, October 2023

Abstract
Thermo Fisher Scientific offers various scientific research services and products, including analytical testing, clinical laboratory services, and pharmaceutical research and development. Their Transmission Electron Microscope is a noteworthy product that the semiconductor, life sciences, and material sciences industries use. It also helps semiconductor manufacturers identify defects in semiconductor chips. Thermo Fisher Scientific also provides complex instruments that require professional installation, troubleshooting, and upgrades to function efficiently. These instruments also host a software hosting infrastructure that Thermo Fisher Scientific Digital Service Engineers manage. To address the challenges of installing and configuring this instrument, we developed a new tool, *Workbench*, which the Field Service Engineers can use to install and update the infrastructure at the customer's location. Additionally, it provides access to external tools that support troubleshooting and other application installations.

Keywords
EngD, Software Technology, Workbench, Graphical User Interface, Automation, Audits, Manage Applications

Foreword

The MSD business of Thermo Fisher Scientific (formerly known as FEI) has traditionally been a global leader in the innovation of electron microscopes (EM). Software has played an increasing role in the delivery of the innovations, yet mainly focused on the instruments themselves.

The new area the company is moving towards is to deliver solutions that support the workflow of the customer using various instruments, covering data management as well as data post processing. To be successful, we need ability to deliver (pure) software solutions as managed service, interfacing with customer infrastructure, and due to the nature of the customer and instruments, deployed and managed within the premises of the customer. Our Software Delivery Platform (SDP) infrastructure services the needs of the on-premise software as a service delivery with local tools and automation. The current configuration management interface of this infrastructure is limited in its usage due to the command line nature of the management toolsets, where an easy to use, intuitive and foolproof solution is needed that can be used by lesser trained IT service engineers also.

Mayank has done a good job in filling this gap. He interacted intensely with various stakeholders outside R/D resulting in a feature set that meets end user needs. As part of the implementation, he introduced a number of new libraries and mechanisms in the SDP configuration GUI sphere and nicely integrated with an existing GUI app, with good decoupling mechanisms with the other system parts.

Due to his rigid time and scope management, Mayank was able to deliver the most important parts of the scope into a release candidate of the actual product.

Mayank demonstrated visible growth during his project in his personal and professional skills, and we wish him all the best on his next endeavors.

Egbert Algra, MSc, PDEng
11 October 2023

Preface

The Engineering Doctorate (EngD) Software Technology (ST) program at Eindhoven University of Technology is a two-year postmaster program focused on training individuals for a career as a technological designer in the industrial sector. The program consists of 14 months of advanced training in technical and professional subjects and three in-house projects driven by industry Needs. Following this, trainees complete a ten-month final project in a company and are awarded the EngD degree upon completion.

This final technical report details the *Development of Service Workbench for Software Delivery Platform* project, which Mayank Sharma completed during his ten-month final project. The project aimed to create a user-friendly, web-based application for Field Service Engineers (FSEs) at Thermo Fisher Scientific with varying skills and backgrounds to install, upgrade, and troubleshoot the software hosting infrastructure called Software Delivery Platform (SDP).

The report includes an introduction to Thermo Fisher Scientific, the project goal, an explanation of critical concepts in the domain analysis, and a description of core needs and requirements. The report then delves into the detailed project solution, followed by verification and validation for software quality. Finally, the report provides recommendations for future work, project management, and career-oriented learning through project retrospectives.

Mayank Sharma
October 2023

Acknowledgements

As an Engineering Doctorate (EngD) Software Technology (ST) trainee working on the EngD project, I had the opportunity to learn and grow professionally. I am grateful for the support and contributions of my supervisors, colleagues, and friends throughout the project.

I sincerely thank Egbert Algra from Thermo Fisher Scientific and Harold Weffers from the Eindhoven University of Technology, who served as my company and university supervisor, respectively. Their invaluable guidance, unwavering support, and insightful feedback were integral to the success of my ten-month-long project, *Development of Service Workbench for Software Delivery Platform*. They aided with project management, documentation, risk management, stakeholder communication, and stakeholder management, as well as valuable insights on architecture and critical pointers that helped me navigate the project successfully. Their willingness to help with every aspect of the project ensured its success.

I am also grateful to my colleagues Respa A. Putra, Giovanni de Almeida Calheiros, Chen Gang, Tor Halsan, Bart van Knippenberg, and Bob Peeters from the Research & Development Team for their camaraderie and stimulating discussions. Their diverse perspectives were invaluable in refining the software design and implementation of the project.

I express my deepest gratitude to Tarkan Akcay, Jordy Plug, Kieran Ham, and Ataur Rahman from the Service Organization for their valuable feedback during the project.

Yanja Dajsuren and Karin Majoor provided invaluable support and guidance during my two years in the EngD program, and I am thankful for their contributions. I would also like to thank my friends from the EngD ST 2023 generation for the joy, laughter, and memorable moments we shared.

I want to thank my parents, Arvind Prakash Sharma and Sarita Sharma, and my sister, Mrinal Sharma, for their understanding, encouragement, and occasional distractions that provided moments of respite during the intense phases of my project. I would also like to thank my spiritual guide, Daaji, for always supporting my endeavors.

Lastly, I would like to thank all whom I have missed and who have helped in my EngD journey.

Mayank Sharma
October 2023

Executive Summary

Thermo Fisher Scientific offers various scientific research services and products, including analytical testing and research, clinical laboratory services, and pharmaceutical research and development. One of their noteworthy products is the Transmission Electron Microscope, which helps manufacturers identify defects in semiconductor chips.

Thermo Fisher Scientific provides complex instruments housing a software hosting infrastructure that requires professional installation, troubleshooting, and upgrades to function efficiently. The Service Organization sends Field Service Engineers to customers to perform these tasks. However, the complex instruments Thermo Fisher Scientific Research and Development Engineers developed require more demanding installation and configuration, resulting in a new tool called *Workbench*. This tool would enable Field Services Engineers to install, upgrade, and troubleshoot the instruments independently at Thermo Fisher Scientific customers' locations.

We developed a solution during the project to help with installation, upgradation, and troubleshooting. Workbench provides a Graphical User Interface that interacts with the software hosting infrastructure to install, update, and troubleshoot itself. This software hosting infrastructure is the Software Delivery Platform, which can be installed or updated by executing automation files. Additionally, Workbench provides the opportunity to view output generated by the automation files in the form of history. It is a centralized place showing each automation file's complete output trace. To support troubleshooting, Workbench provides access to other applications that help to troubleshoot the Software Delivery Platform and install other applications.

As an experimental version, Workbench has become part of the Thermo Fisher Scientific Q3 software release. This release will enable all Service Engineers to test the application and provide further feedback to improve the application.

Table of Contents

List of Figures

List of Tables

Glossary

TU/e	Eindhoven University of Technology
DMP	Data Management Platform
SDP	Software Delivery Platform
DSE	Digital Service Engineer
DSO	Digital Service Organization
FSE	Field Service Engineer
GTS	Global Technical Service
OE	Operations Engineer
RNDE	Research and Development Engineer
EngD	Engineering Doctorate
SPC	Support PC
MPC	Microscope PC
MTA	Milestone Trend Analysis
CI/CD	Continuous Integration/ Continuous Delivery
ST	Software Technology
TEM	Transmission Electron Microscope
SCA	Software Configurator Application
SCI	Software Configurator Installer
SCIA	Software Configurator Installer Application
SLMT	Software Lifecycle Management Tools
SWAP	WDP Web-installer Application Program
GUI	Graphic User Interface
VM	Virtual Machine
MoSCoW	Must have, should have, could have, won't have
SO	Service Organization
SE	Service Engineer
SSH	Secure Shell
PDCA	Plan Do Check Act
PMP	Project Management Plan
SEM	Scanning Electron Microscope
VM	Virtual Machine

1. Introduction

1.1 Context

Thermo Fisher Scientific is a global leader in scientific research services and products. Their services include analytical testing and research, clinical laboratory services, and pharmaceutical research and development. They serve many customers, including academic institutions, government agencies, biotechnology companies and semiconductor manufacturers [1].

One notable product that Thermo Fisher Scientific produces is their Transmission Electron Microscope (TEM) which has proven an invaluable tool for semiconductor manufacturers. With the ability to image materials at the nanoscale, these microscopes can help identify defects in semiconductor chips [2]. This saves time and money and ensures a higher-quality final product. The electron and light microscopy can be seen further in Appendix A.

1.2 Business Problem & Goal

Along with the TEM, Thermo Fisher Scientific provides customers with complex instruments housing a software hosting infrastructure that requires professional installation, upgradation, and troubleshooting to work effectively and efficiently. Since many customers need skilled professionals, the Service Organization (SO), part of Thermo Fisher Scientific, sends their Field Service Engineers (FSEs) to perform these on-site tasks. These engineers are also responsible for any necessary hardware or software upgrades during the operation and maintenance of the instrument. It is important to note that these instruments are often connected to the internet.

Thermo Fisher Scientific has recently developed a new range of instruments that use virtualization and container-based design to provide faster software upgrades. However, this new range requires a more demanding installation and configuration, necessitating Digital Service Engineers (DSEs) support. DSEs are software skilled and understand these complex instruments. To provide support, they connect to the instruments online or visit the customer site. While the SO has a few DSEs to manage the instruments and the software hosting infrastructure, creating a new tool was necessary to enable the FSEs to install, update, and troubleshoot it independently on-site at the customer's location.

1.3 Project Problem & Goal

For this project, we worked with the Data Management Platform (DMP) as the complex instrument that houses the Software Delivery Platform (SDP) as its software hosting infrastructure. To understand DMP, SDP, and other concepts, please refer to Appendix B. Most of the SDP installation and upgradation process occurs within the terminal [3] software. However, the *SDP Configuration* is created and updated through another Software Configurator Application (SCA) [4]. SCA is a web-based Graphical User Interface (GUI) tool designed to be simple and easy to use, enabling FSEs to configure the SDP. It generates an SDP Configuration that contains SDP resource information. SCA contains two main functionalities. It allows the generation of SDP Configuration from scratch. This generation is known as a *Complete Action*. Besides configuring a Complete Action, SCA also allows FSEs to update SDP by updating specific parts of the SDP Configuration through a series of forms after the Complete Action is successful. These are known as *Quick Actions* within SCA.

DSEs from the Digital Service Organization (DSO) within Thermo Fisher Scientific manage most tasks. These tasks are simple, automated, and easy to use. However, installing SDP using automation scripts is still an error-prone and daunting task for FSEs. SCA does not implement this installation through its web-based GUI. This installation using automation scripts is termed as *Applying An Action* in the following chapters. Moreover, straightforward access through a single place to other applications for troubleshooting and application installation needs to be created, as accessing them is a tedious task for FSEs.

Therefore, this project presented an opportunity to develop a simple and easy-to-use web-based GUI application called *Workbench* that bridges the gap between FSEs with different skills and backgrounds to install, upgrade, and troubleshoot SDP independently. This includes the following objectives:

- Applying the Complete Action for full SDP installation after the SDP Configuration is generated
- Applying Quick Actions to update specific parts of SDP after the SDP Configuration is updated
- Providing access to output traces in the form of automation audits after actions have been applied in a centralized place called *Action History*
- Providing access to SWAP and other service applications such as Grafana [5], Kibana [6], and Kubernetes [7] dashboards for troubleshooting and application installation

Even though Workbench was developed for FSEs, we considered both DSEs and FSEs end users since both will use the application to manage SDP. These engineers will be collectively called Service Engineers (SEs) in the following chapters. We designed a web-based GUI application to provide access through browsers and mitigate the error-prone nature of the terminal. Moreover, web-based GUI allowed the application to simplify and improve ease of use.

1.4 Project Scope

Figure 1 shows the scope of the project. Since generating a configuration goes hand in hand, applying an action became an extension to SCA known as Software Configurator Installer (SCI). The SCI will apply the actions through the new web-based GUI. The combined application became a Software configurator installer application (SCIA). Since SWAP is responsible for application installation on SDP, it is part of The Software Lifecycle Management Tools (SLMT) and SCA. SDP contains a Kubernetes cluster [7] where SWAP installs these applications. Workbench combines SLMT with maintenance applications and acts as a unification point for all installation and troubleshooting of SDP. We will collectively address SWAP and service applications as *Manage Applications* in the remaining sections for simplicity.

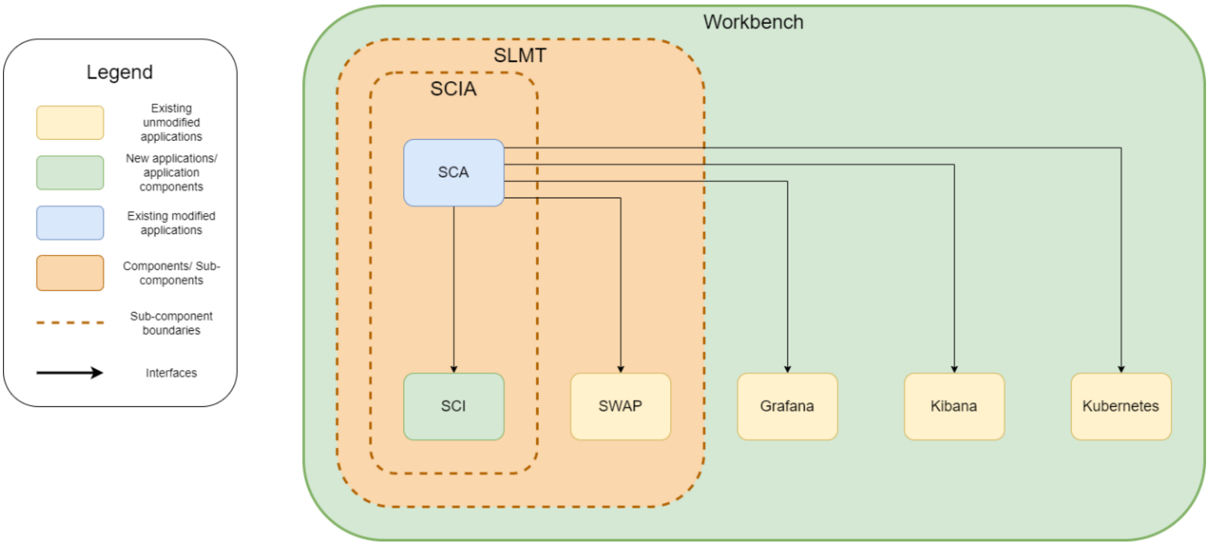


Figure 1. Diagram showing the workbench scope

2.Needs & Requirements

2.1 Overview

To manage this project, we defined a timeline containing four phases: planning, requirements elicitation and elaboration, solution elaboration, and finalization. For a detailed understanding of these phases, please refer to Section 6.2. For the requirements elicitation and elaboration phase, we captured many user *Needs* [8] from stakeholder interviews. These Needs helped to formulate the functional and non-functional requirements for the project. To see a detailed explanation of how Needs and *System Requirements* [9] were collected and documented, please refer to Section 6.6.

2.2 Needs

We captured 91 raw Needs that were formatted and formalized into 54 Needs in a *Needs Register*. The Needs Register contained a list of user Needs requested by various stakeholders. Table 1 shows a list of core Needs (addressing the main functionality or the system's quality) elicited from the elicitation process. These Needs align with the project goal defined in Section 1.3.

Table 1. Core Needs elicited

Needs ID	Needs
N1	Apply the complete action for full SDP installation
N2	Apply quick actions to update specific parts of SDP
N3	Show line-by-line, verbose, output, progress, and exit status of the action applied
N4	View action history of all automation audits in a centralized place
N5	Provide access to manage applications for troubleshooting and installing applications
N6	Simple and easy-to-use for SEs
N7	Provide extensibility to add more actions
N8	Provide modularity for code reusability and maintainability
N9	Allow only one SE to apply an action at one time, while others can view the progress of that action

2.3 System Requirements

The System Requirements contained a combination of functional (solution functionality) and non-functional (solution quality). In the *Requirements Register*, we formulated a total of 22 requirements. The Requirements Register contained a list of System Requirements that addressed the user Needs and helped to develop Workbench. Out of these, ten were core as they address the core Needs.

2.3.1. Functional

Out of the ten requirements, six were functional requirements. These requirements addressed the main functionality of the system's Needs. Table 2 shows the core functional requirements and the Needs they address. We also added the associated *User Stories* [10] to the requirements.

Table 2. Functional requirements and the Needs they address

Requirements ID	System Requirements	User Story	Needs ID
F1	The system shall apply the complete action for full SDP installation.	As an SE, I want to apply the complete action so that I can install the full SDP by myself.	N1
F2	The system shall apply quick actions to update specific parts of SDP.	As an SE, I want to apply quick actions to update specific parts of SDP so that I update specific parts of SDP by myself.	N2
F3	The system shall show line-by-line, verbose output of the action applied.	As an SE, I want to see line-by-line, verbose output of the action applied so that I can check the progress of the action.	N3
F4	The system shall notify the success or failure of the action applied.	As an SE, I want to see the success and failure of the action applied so that I can check if the action was a success or a failure and notify others of any problems.	N3
F5	The system shall show the action history of all automation audits in a centralized place.	As an SE, I want to view the action history of all automation audits in a centralized place so that I can check the exit status and audit content and notify others of any problems.	N4
F6	The system shall provide access to manage applications for troubleshooting and installing applications.	As an SE, I want to access manage applications for troubleshooting and installing applications so that I can troubleshoot SDP and install applications by myself.	N5

2.3.2. Non – Functional

The other remaining four requirements were non-functional and addressed the software's quality. These can be seen in Table 3. We used the ISO 25010 [11] reference framework standard list of non-functional requirements to elicit them, as in Table 16, Appendix C.

Table 3. Non-functional requirements and the Needs they address

Requirements ID	System Requirements	User Story	Needs ID
NF1	The system shall be simple and easy to use.	As an SE, I want to use the workbench with ease so that I can use it without any user manual or help.	N6
NF2	The system shall be extensible with more quick actions and manage applications.	As an RNDE, I want to extend the workbench with more quick actions and manage applications so that an SE can perform more tasks independently.	N7
NF3	The system shall be easy to modify to fix issues quickly and reuse components.	As an RNDE, I want to modify the workbench easily so that I can fix issues quickly, provide uninterrupted access to the workbench for the SEs, and reuse components.	N8
NF4	The system shall allow only one action to be applied at a time.	As an SE, I want only one action to be applied at a time and redirect other SEs to view the action in progress.	N9

2.4 Use Cases

We also created a list of *Use Cases* [12] from the requirements that describe how SEs will use Workbench. These were created from various consultation sessions throughout the project. Table 4 shows these Use Cases. To view detailed Use Cases, please refer to Appendix G.

Table 4. Use Cases and the requirements they address

Requirements ID	Use cases ID	Use cases	Description
F1	UC 1	Apply complete action	This use case involves starting automation script execution of the complete action through the web-based GUI.

Requirements ID	Use cases ID	Use cases	Description
F2	UC 2	Apply quick actions	This use case involves starting automation script execution of the quick actions through the web-based GUI.
F3	UC 3	View output	This use case involves viewing the output and progress of the actions through the web-based GUI.
F4	UC 4	View success or failure	This use case involves viewing the success and failure of the actions through a web-based GUI.
F5	UC 5	View action history	This use case involves viewing the action history in a single place. It also allows viewing audits of each action through the history.
F6	UC 6	Access manage applications	This use case involves accessing service applications and SWAP for troubleshooting and application installation.

3.Solution

3.1 Overview

This chapter explains our high-level view of our application, the part-by-part design and implementation for each core component, and the requirements it addresses. The chapter follows this approach to show how we designed and implemented our application part-by-part. We will delve into implementation notes for each component, such as the final UI[1], development tools and libraries, and how we developed specific components. For a complete understanding of how we documented the *Solution Architecture*, please see Section 6.7. To view all the Solution Architecture diagrams, please see Appendix G.

3.2 High-Level View

Figure 2 shows the high-level view of the application. The application consists of the Client, a Webserver, and SDP. It follows a client-server architecture [13]. The arrows represent the flow. Connections without arrows represent the bi-directional flow. The diagram uses color coding, as shown in Figure 2. Table 5 explains the high-level view components in detail. The Client connects with the Workbench Webserver application and displays the rendered template. The Webserver connects with SDP resources and performs the necessary tasks. The SDP provides the necessary resources for Workbench.

Table 5. Description of high-level view of solution components

Component	Description	Interfaces
Client	A Web browser to view rendered Workbench UI templates	Accesses the Workbench Webserver to perform tasks
Webserver	Contains the main business logic and application entry points to perform tasks	Accesses the SDP resources to generate SDP configuration, execute automation scripts, parse automation audits, and access service applications and SWAP
SDP	Software hosting infrastructure within DMP	Stores or provides the necessary resources to Webserver to perform tasks

The Webserver can be divided into further sub-components and modules.

Table 6 explains these subcomponents, their interfaces, and the requirements they fulfill. The Webserver consists of three sub-components. The main is Workbench, which contains both SCI and SCA. SCI contains both the Action and History as they go hand in hand. The Manage Applications provide access to the service applications and SWAP. Workbench also contains *Form Flows* that integrate SCI and SCA. SCA is the existing module that generates the SDP Configuration.

[1] The Final UI may contain some inconsistent naming.

Table 6. Traceability between the solution components and requirements

Sub-component	Requirements fulfilled	Module	Description	Interfaces
SCI	F1 - F4	Action	Contains the functionality to apply complete and quick actions	Executes automation scripts to install and update SDP
	F5	History	Contains the functionality that shows action history, their exit status, and their audit trail	Parses automation audits generated by automation scripts within SDP
Workbench	F6	Manage Applications	Contains the functionality that integrates SWAP, SDP Health traffic light and service applications redirect links	Accesses the service applications and SWAP within SDP
		Form Flows	Contains flow information that defines the sequence of forms to generate and update SDP configuration	Integrates SCI and SCA
SCA		SCA	Existing module that contains forms for different SDP VMs, storage, and other resources	Interacts with SDP to generate SDP Configuration

In the following sections, we will discuss in detail the different modules[2] of the Webserver and how Form Flows integrate SCI and SCA. Only the SCA components we added or modified will be discussed in the design and implementation.

[2] It contains some sub-modules implemented in class-based approach. Thes include the sub-modules part of the business logic. This approach was used to improve code modularity and fulfil NF3.

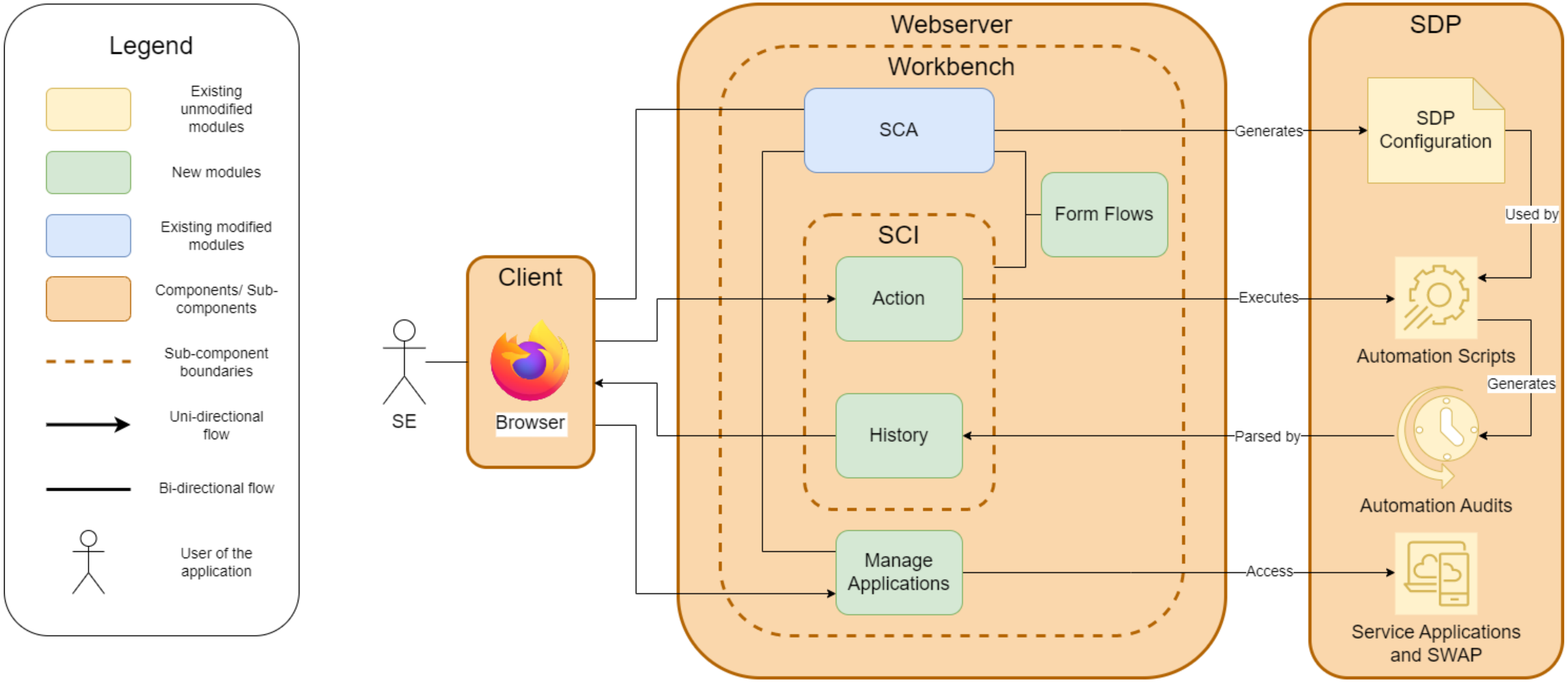


Figure 2. Diagram showing the high-level view of the solution

3.3 SCI – Action

This module addresses applying Complete and Quick Actions. SEs can apply these actions once they have completed forms to create or update the SDP Configuration. Since the system had to be simple, easy to use, and self-guided, it had to display output in a format that is easy to understand or resemble existing tools used by SEs such as the terminal. In addition, we made it a single-click installer as requested by them. Finally, we made it simple to extend for an RNDE if a need in the future arises to support more Quick Actions.

3.3.1. Applying an action

Design

Applying an action can be explained in detail in Figure 6. Its sub-modules can be explained in Table 7.

Table 7. Description of sub-modules within Action

Sub-module	Description	Interfaces
Action API	Contains endpoints for accessing action application and showing the rendered action template	Triggers Operation to update Webserver state and start an action
Web Socket	Contains WebSocket [14] endpoints implementation to broadcast line-by-line output to the Client	Provides line-by-line output to multiple clients
Operation	Contains business logic for managing the Webserver state	Triggers Script Executor to execute Bash Scripts
Script Executor	Contains business logic for executing a Bash Script in SDP	Signal Bash Scripts to start or stop the execution in the background and collect output and exit code
Bash Scripts	Contains commands to install or update SDP and execute necessary Ansible Scripts	Executes Ansible Scripts and sends output or exit code to Script Executor
Ansible Scripts	Contains commands to install or update SDP based on SDP Configuration	Sends the Ansible exit code and output to Bash Scripts
Form Flow File	Contains Bash Scripts and Actions mapping	Parsed by Operation to select the appropriate Bash Script for Action applied.

In this functionality, an SE can apply an action from the Client. The Action API receives the action request and initializes the Operation sub-module to start an action. Since Operation sub-module uses a singleton design pattern [15], the application contains only a single instance and, therefore, always ensures a single Bash Script [16] is executed by Script Executor. This fulfils the non-functional requirement NF4. As the executor executes the script and gathers line-by-line output, the output is cascaded to the Web Socket sub-module, sending the output to the browser where the SE can view the ongoing action. Once the action is successful or fails, the Script Executor captures the exit code [17] with which the execution ends. An exit code is a numerical value used by computer programs to denote success or failure. This is sent to the rendered template in the Client as success or failure to notify the SE.

The Operation sub-module acts as a single-entry point to the business logic that manages the internal Webserver state and passes information to the Script Executor based on a mapping present in a JSON file [18]. This JSON file, known as *Form Flow File*, contains a mapping for all Complete and Quick Actions as shown in Figure 20.

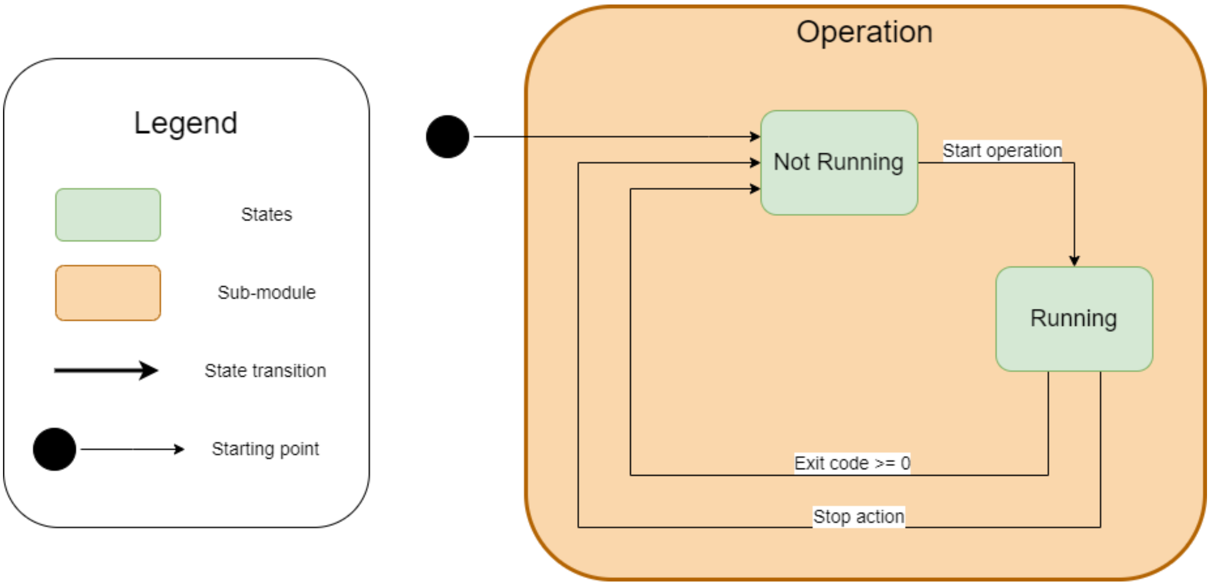


Figure 3. Diagram showing how operation manages Webserver state

To further explain, how Operation manages the Webserver state, we developed a mechanism that stores existing in-progress action information. This mechanism was developed based on decision D12 in Table 18, Appendix E. Figure 3 show the internal state within Operation. The states could be either running or not running. When an action is applied, the Operation enters a running state. Due to the singleton pattern, every Client accessing the SCI, accesses the same state and therefore gets the same in-progress action information. This ensures, all Clients and therefore SEs get to check the progress of the same action applied. The state of the server resets when the action is cancelled, or it completes either with success or failure.

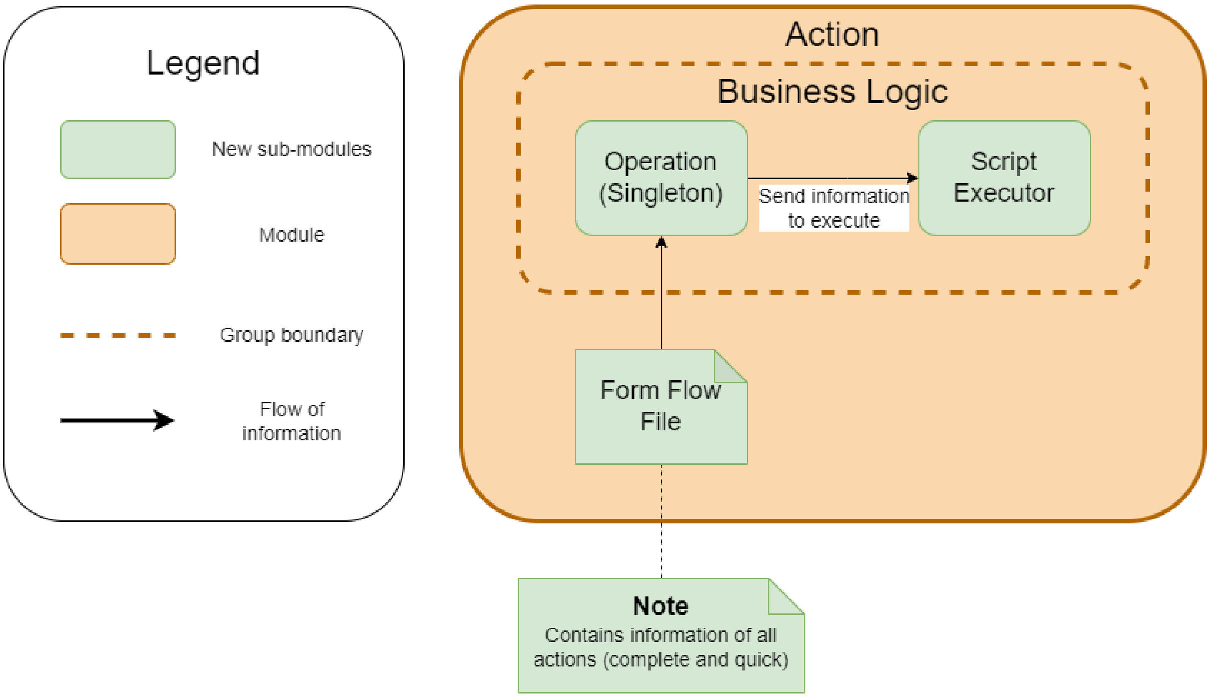


Figure 4. Diagram showing the flow of information from Form Flow File to Script Executor

Additionally, we decided to group Complete and Quick Actions to make the system modular. When an action is applied, the Operation obtains the mapping to run the automation script present in Form Flow File. Figure 4 shows this behavior. The file contains the mapping of all the actions and automation scripts. When an action is applied, the operation obtains the name of the Bash Script based on the mapping. It then passes this name to the Script Executor. The Script executor then looks for the Script in the SDP and then executes it to create or update SDP resources. Since any action can be executed from the same Script Executor sub-module, more actions can be added in the future by the RNDEs. All the RNDEs must do is to extend the Form Flow File with new automation mapping which can then applied by SEs from the client. This fulfills NF2 and NF3 since the action sub-module does not change due to any new addition of actions.

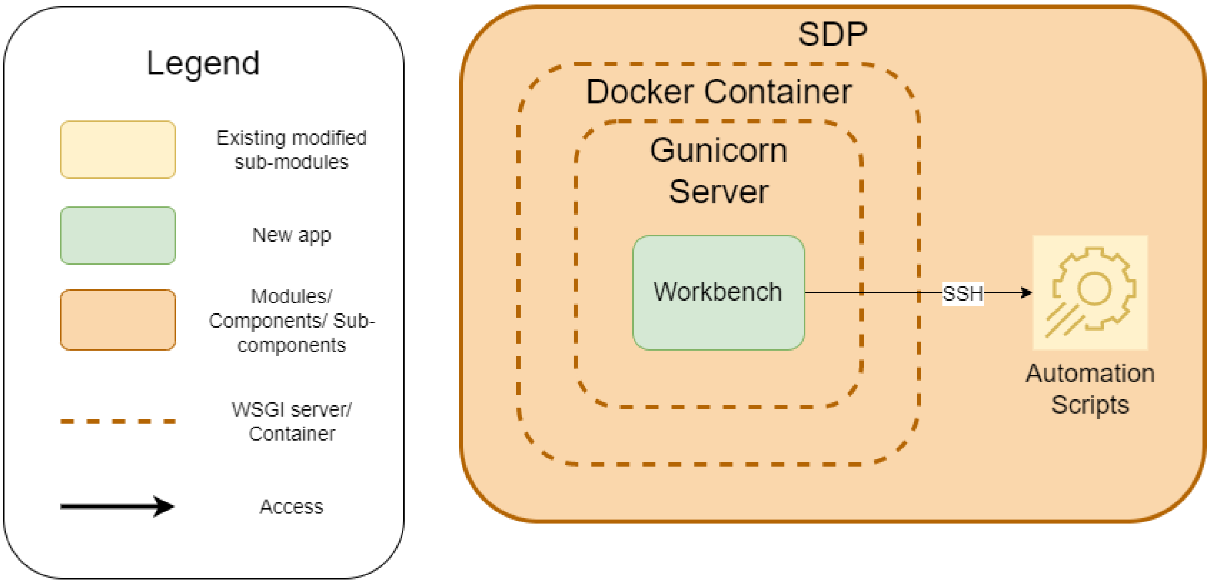


Figure 5. Diagram showing workbench accessing automation scripts through SSH

Implementation

Figure 36 shows the final web page of the application. It consists of the following elements:

- **Status panel** – The status panel shows the status of the action applied, such as in progress, success, error, or action cancelled. It turns green when the action is successful or red when it fails. Otherwise, it remains yellow when the action is in progress or cancelled.
- **Progress bar** – This shows the progress of the action and turns green when the action is successful or red when the action has failed. When the action is completed, the progress bar fills up to 100% of the bar. Otherwise, it is not complete.
- **Output panel** – It shows line–by–line output. Any output line containing color is also shown with color information in the panel. When an action succeeds, a final output line is displayed on the panel, denoting the success of the action. Otherwise, the line denotes failure of the action.
- **Cancel button** – It allows an SE to cancel an applied action. It is enabled by default, and the action is cancelled when the SE clicks it, and this button gets disabled.
- **Close button** – It allows an SE to close the action page and move to Action History. It is turned off by default and becomes enabled when an action is cancelled, succeeded, or failed.

To implement the functionality at the code level, we chose several libraries. Table 17 in Appendix D shows the tools and technologies used and their purpose. Since the automation scripts are present in the SDP and need to run on it natively, the application must connect with the SDP. To perform this, the subprocess [19] library must connect with SDP using a Secure Shell (SSH) [20] connection. This is shown in Figure 5. Workbench is deployed in SDP itself. It is served using the gunicorn [21] server with the gevent-websocket [22] library to support the Web Socket asynchronous [23] behavior. It is then further wrapped within a Docker Container [24] to run on SDP.

To execute scripts, we implemented our own single thread [25] model. Workbench uses the subprocess library to execute Bash Scripts. The flask-socketio [26] creates a thread of execution to broadcast which is passed as a callback to the Script Executor. The execution command is provided by the Script Executor sub-module. However, since the execution needs to happen natively on SDP, the execution command is wrapped within the SSH command that helps to connect to SDP. This creates a channel for communication between Workbench and SDP and execution begins. The line-by-line output is then sent back to the subprocess library. The output is then fed to the Web Socket callback which then broadcasts the output to the Client. The output can be viewed on the browser in color for easier interpretation of the content and to comply with NF1. This was possible because the output is displayed using the xterm.js [27] library that resembles the terminal used by SEs.

To manage the internal state, we used Boolean value which is set to true when an action is applied and running. When not running, the same value is set to false. If another action is applied, while an action is in progress, the SE is redirected to the existing action in progress.

3.3.2. Cancel an action

Design

The SE can also cancel an action when something goes wrong during the whole process. Figure 7 shows that the SE can cancel the action through the browser, interrupting the SDP's automation scripts. This sends an output to the Client rendered template where the SE sees the action cancelled.

Implementation

The cancel button can be seen in the final web page in Figure 36. To cancel a running action, the Script Executor sends an interrupt signal to the running process. This terminates subprocess library execution and it receives an error exit code.

3.3.3. Auto – reconnect to action

Design

Workbench also supports auto-reconnection in case an SE closes the browser on his computer or wants to check the progress from another computer. Figure 8 shows the auto-reconnect scenario. The number in the circles show the steps of reconnection. As the first Client disconnects in step 1, the action in progress action continues to run till it either succeeds or fails. As shown in step 2, when the SE accesses the workbench application again, the Client starts receiving output information from the Web Socket sub-module. The home also shows the progress of the action, and the SE can then choose to go to action web page to see the latest progress.

Implementation

Figure 35 shows the final web page of home, which shows a running action. The page shows the action in progress and the button to go to the action page. When clicked on the *to installer* button, the SE gets redirected to the action page shown in Figure 36. This is possible because each Client connects to socketio [28] receivers, one at home and one in action that receive output. When loading the home page, the home socketio receiver connects with the Web Socket sub-module, and it starts receiving output. When redirected to the action page, the Client connects with the second receiver and receives information in detail.

Since the socketio and flask-socketio support auto-reconnect, it was easy to implement this feature. The socketio receiver keeps checking for a flask-socketio connection provided by Web socket sub-module and connects when found.

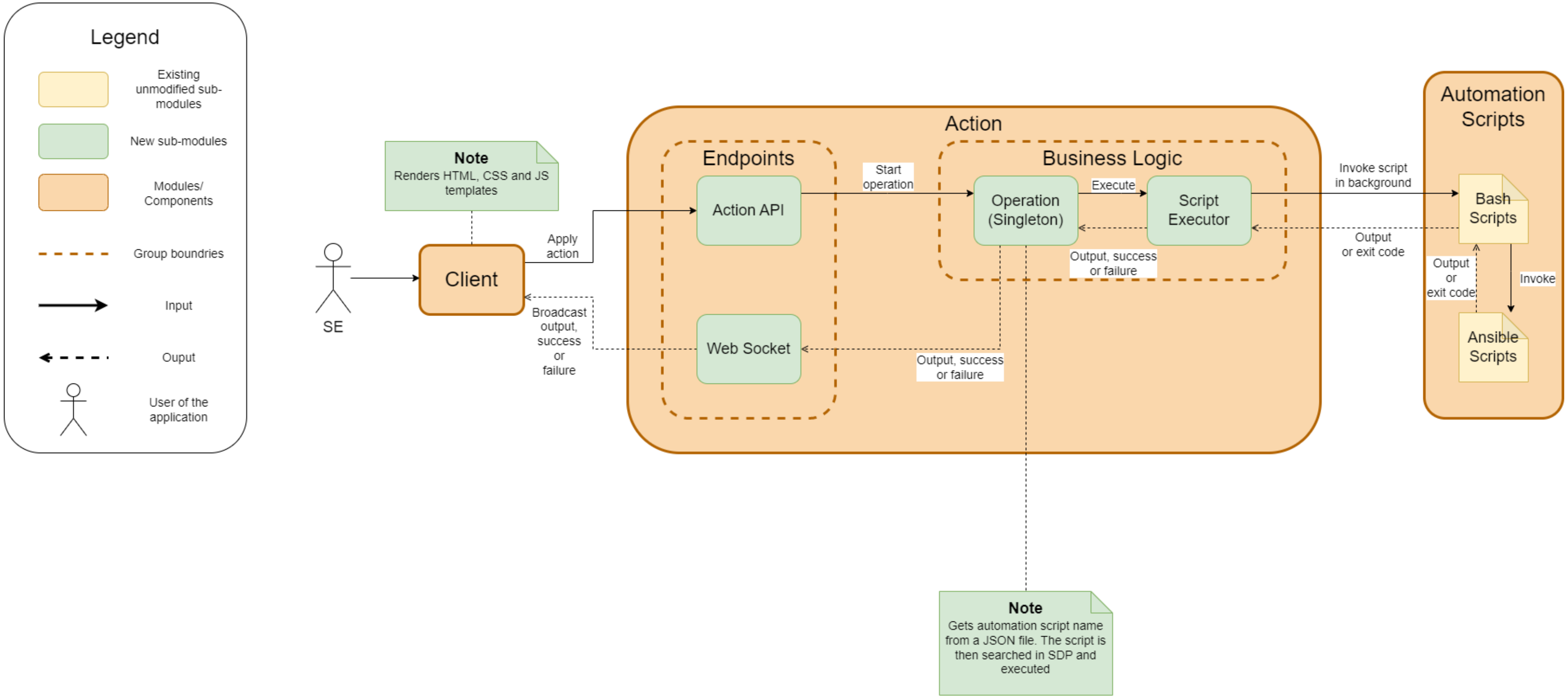


Figure 6. Diagram showing applying an action

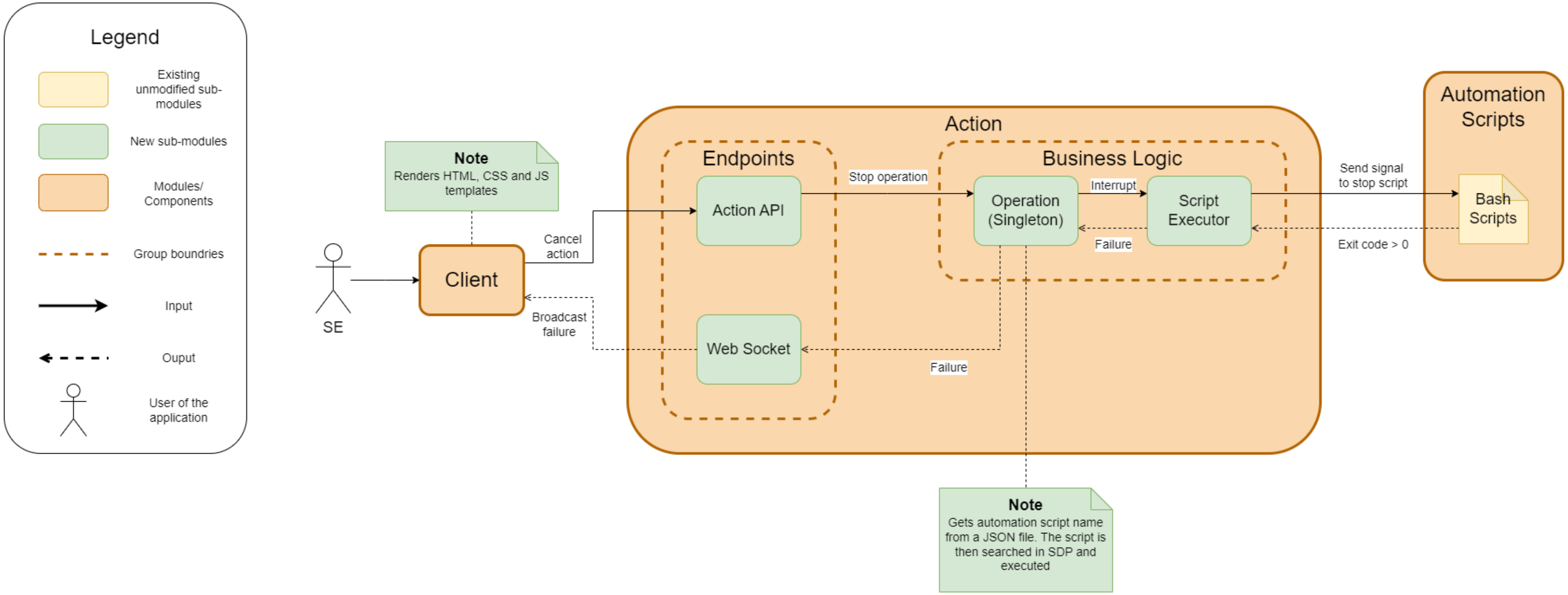


Figure 7. Diagram showing cancelling an action

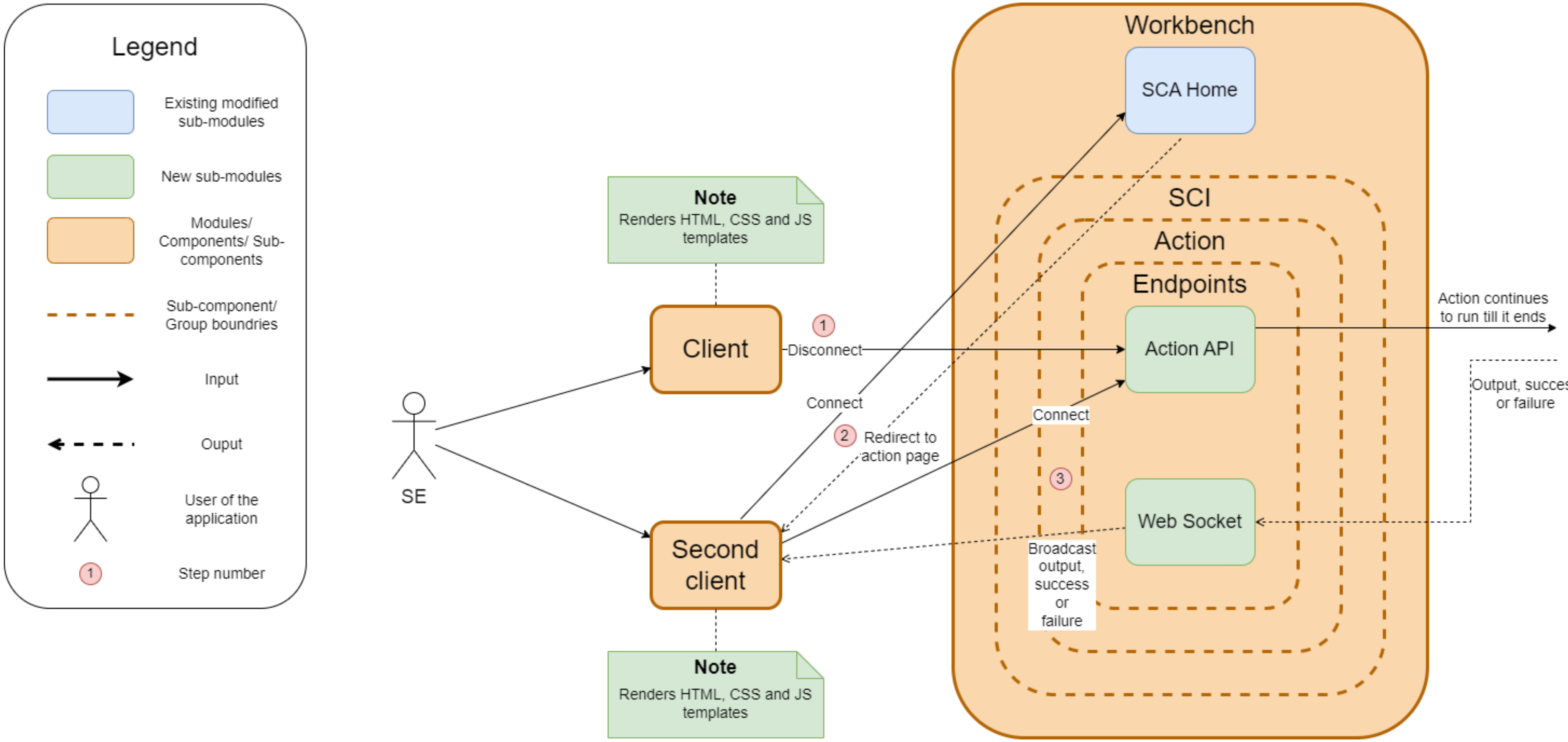


Figure 8. Diagram showing client reconnection

3.4 SCI – Action History

The second module of SCI includes viewing Action History. In this part, an SE can view all the Action History from the most recent to the oldest. Additionally, it shows the action name, exit status, audit creation date and the entire audit content for each action.

Design

The design of the Action History can be seen in Figure 11. It is still within the SCI as it shows the history of all the audits of actions. The audits are generated from automation scripts that install or update SDP. Table 8 describes the elements of Action History.

Table 8. Description of sub-modules within History

Sub-module	Description	Interfaces
History API	Contains endpoints for accessing the entire action history and audit contents	Interacts with Audit Extractor to retrieve extracted and sorted audit content
Audit Extractor	Contains business logic to extract the action name, audit creation date, exit status, and audit content	Interacts with Automation Audits within SDP to extract their contents
Audit	File containing entire action output and exit status of the automation scripts	Generated by automation scripts within SDP and their content is parsed by Audit Extractor

The SE can access Action History through the History API. The History API then interacts with Audit Extractor which extracts information from the Audits. The Audit name contains detailed information of the action name and creation date. Inside the Audit is the entire action output and exit status. The Audit Extractor parses, sorts, and compiles all this information and then sends it as a response to the Client through the History API. The SE can then view the entire history with all the compiled information on the Client rendered template.

Implementation

The history consists of two web pages. These can be seen in Figure 39 and Figure 40. Figure 39 shows the final history web page while Figure 40 shows the final audit content web page. It also shows the audit details, such as the action name, exit status, and the creation date. To view the audit content, the SE can select the *View Content* button to see the entire action output.
The web pages contain the following elements:

- **Homepage button** – The homepage button allows an SE to move back to the Workbench home from the action history web page
- **Filter selector** – This allows SE to filter the actions by action name and exit status
- **View Content button** – This allows SEs to access the audit content web page to view the audit
- **History page button** – The history page button allows an SE to move back to the action history web page
- **Audit panel** – This panel shows the entire output of the automation audit for an action
- **Status** – This shows the exit status of the applied action
- **Action name** – This is the name of the action applied
- **Date** – Date of creation for the automation audit

We took decision D9 in Table 18, Appendix E to establish a standard between the audits and Workbench. Through this, we agreed that the final exit code of the automation script would be printed as the last line in the audit content. This can be seen in Figure 41 where the exit code is printed in the last line. This can then be parsed by the Audit Extractor, which can be shown as a success or failure icon on the history web page. This can be seen in Figure 39, which shows the red and green icons to denote failure and success, respectively.

Figure 9. Final UI showing filter button to filter action history

An SE can also use the filter button to filter actions by action name or exit status. Figure 9 shows the filter selector. This makes it easier for SEs to sort and view specific action information. To view complete audit content, we designed another web page shown in Figure 40. It shows the entire audit content, the action name and the date created.

At the implementation level, we used several libraries. The libraries can be seen in Table 17, Appendix D. We used HTML tables to design the action history. To filter the audits, we took a decision D22 in Table 18, Appendix E to decide the library for filtering. We chose a custom implementation [29]. The implementation was derived from an existing library. To parse the action name and date of creation from audit file name, we used the re [30] library for regular expressions. The action name and creation date can be seen in Figure 10 that the Audit Extractor parses to display on action history. We used the font awesome icons as it was already integrated within SCA. Colors were enabled for audit content for easier interpretation and to comply with NF1. This was possible because the audit content is displayed using the xterm.js library that resembles the terminal used by SEs.

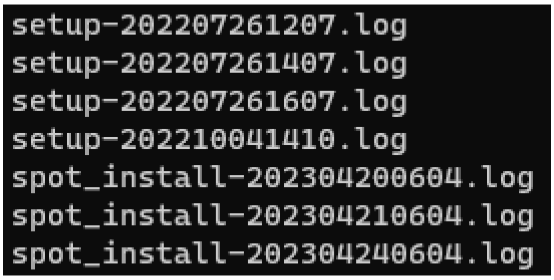


Figure 10. Image showing the name and creation date of automation audits

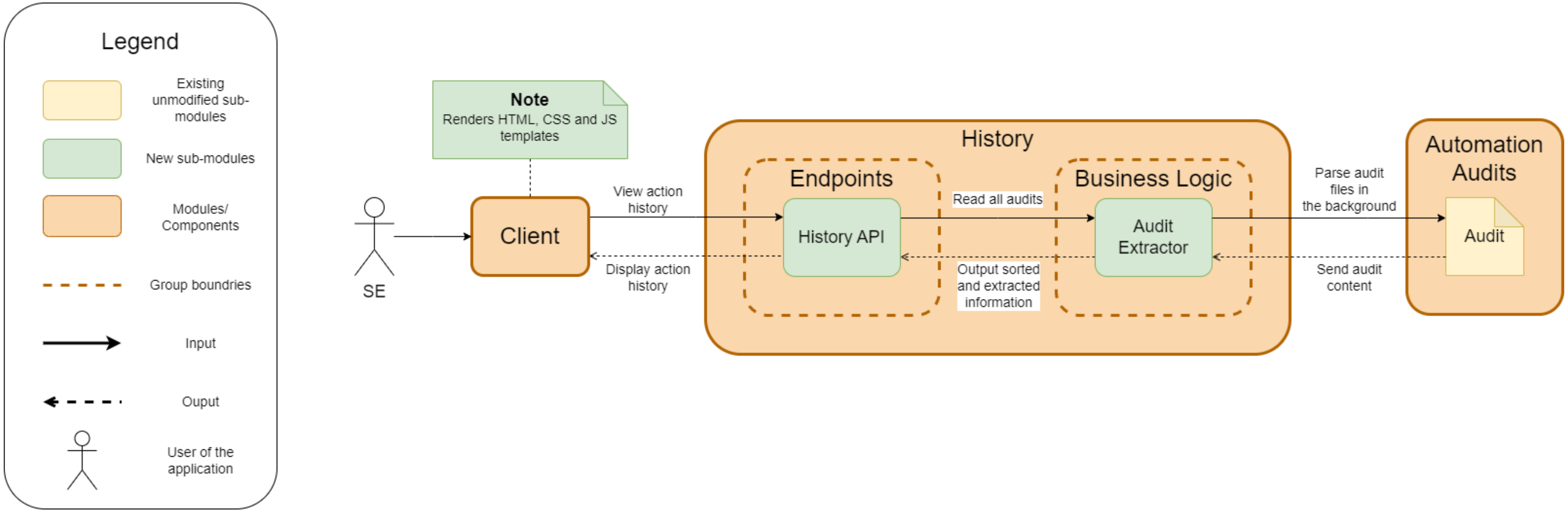


Figure 11. Diagram showing internal sub-modules of history

3.5 Workbench – Manage Applications

Workbench also provides access to other apps that help to monitor and manage SDP. These include the service applications and SWAP. SEs are redirected to these applications on a new Web browser tab. It also provides SDP Health alerts as a traffic light and embeds SWAP.

3.5.1. Accessing service applications

Design

Figure 16 shows the design of accessing service applications. Its sub-modules are explained in Table 9. The design was according to decision D14 in Table 18, Appendix E. The numbers in the design show the steps of the access. The arrows in the diagram show the inputs and outputs. As shown in step 1, when an SE accesses Workbench, the SE gets access to the SCA home. SCA home was an existing SCA sub-module modified and reused as the home for Workbench. As the SCA home loads, the URL Builder builds service applications URLs based on a mapping present in a Manage Applications File as shown in Figure 12 and application hostname, as shown in step 2. Finally, in step 3, when an SE accesses a specific service application from SCA home, the SE gets redirected to that application based on the URL in a new tab. For example, when an SE clicks on *View SDP Health*, the SE gets redirected to the Grafana overview page in another browser tab.

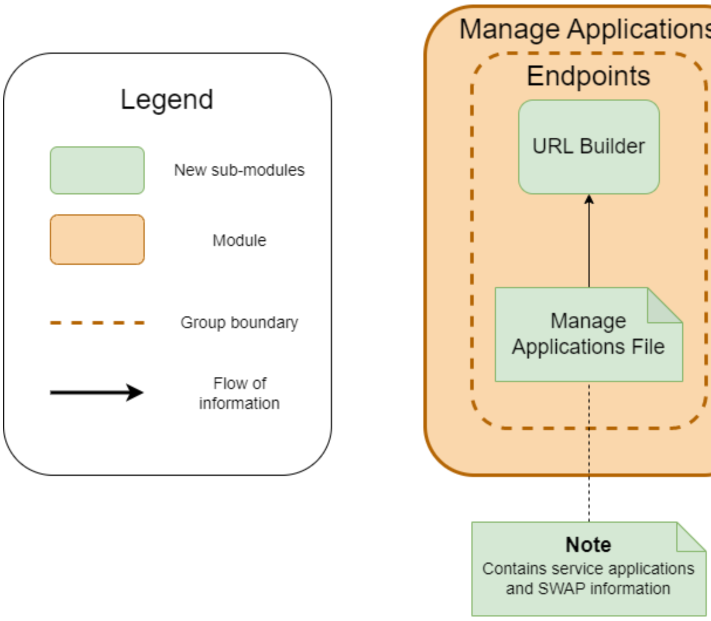


Figure 12. Diagram showing how URL builder obtains Manage Applications information

Since Workbench is also deployed in the SDP, Workbench, service applications, and SWAP contain the same hostname. The URLs are built dynamically when the Workbench is operational since SEs cannot update the application URLs as they cannot access the code. This dynamic building of URLs fulfills NF1 and makes it simpler for SEs to access the service applications with a single click.

Table 9. Description of Manage Applications internal sub-modules

Sub-module	Description	Interfaces
URL Builder	Contains the logic to build URLs for service applications and SWAP	Interacts with hostname and manage applications JSON File to provide application URLs
SCA Home	Contains the logic to show service applications and SWAP on the Client rendered template	Provides Client rendered template with a list of URLs to redirect SEs to service applications and SWAP
Grafana	Service application for monitoring SDP health and providing alerts for problems within SDP	Accessed through SCA home and provide alerts to SDP health traffic light
Kibana	Service application for diagnostics and log trace present in Elasticsearch	Accessed through SCA home
Kubernetes Dashboard	Service application to manage applications running on the Kubernetes Cluster and check application health	Accessed through SCA home
SWAP	SLMT application that installs applications on the Kubernetes Cluster	Accessed through SCA home and embedded within Workbench
Manage Applications API	Contains endpoints to render SWAP Iframe template and filter severity level from SDP health alerts	Provides Client rendered template with SWAP embedded and interacts with Grafana to fetch SDP health alerts

Implementation

Figure 43 shows the final UI of Manage Applications. We added a new button, Manage Applications, which shows a list of service applications and SWAP. These applications can be seen further in Figure 13. When an SE clicks on View SDP Health, *View Kibana logs* and *View Kubernetes dashboard*, the SE gets directed to the specific service application in a new tab.

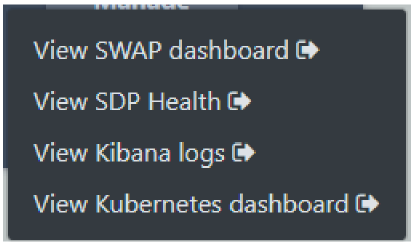


Figure 13. Final UI showing list of Manage Applications

In the background, the URL Builder sub-module builds the URL using the hostname obtained from the `request` [31] library and Manage Applications File. The request library is part of the Flask application framework [32] used for building Workbench. The Manage Applications File can be seen in Figure 14. It shows mapping for different service applications. Since the mapping is decoupled into this JSON file, it can be extended further with new applications fulfilling NF2 and NF3.

```
{       You, 3 months ago • Update tests …
    "swap": {
        "name": "View SWAP dashboard",
        "port": 30004,
        "type": "iframe",
        "protocol": "https",
        "test_host_name": "config.minions.acht.athena.zone"
    },
    "sdp_health": {
        "name": "View SDP Health",
        "port": 30011,
        "type": "alert",
        "protocol": "http",
        "test_host_name": "config.minions.acht.athena.zone",
        "alert_api": "/api/alertmanager/grafana/api/v2/alerts",
        "username": "CustomerService",
        "password": "feico95-001",
        "polling_time": 60000,
        "test_polling_time": 3000
    },
    "kibana": {
        "name": "View Kibana logs",
        "port": 30002,
        "type": "redirect",
        "protocol": "http",
        "test_host_name": "config.minions.acht.athena.zone"
    },
```

Figure 14. Image showing the structure of the Manage Applications file

3.5.2. Accessing SWAP within Workbench

Design

Since the DSEs wanted the SWAP tool to be embedded within the Workbench, we took decision D15 in Table 18, Appendix E to show it through an Iframe [33]. An Iframe is like a mini window on a web page that shows another web page inside it.

Figure 17 shows how an SE can view the SWAP tool through an Iframe. Like Section 3.5.1, the URL builder sub-module builds the URL for SWAP. However, it returns a SWAP endpoint rather than the URL. The manage application sub-module also contains a Manage Applications API that acts as a wrapper for SWAP. When an SE clicks on *View SWAP dashboard*, as shown in Figure 13, the SE gets directed to another web page containing an Iframe through the manage application sub-module where SWAP is embedded.

Implementation

Figure 42 shows the final Iframe web page of SWAP within Workbench. The Iframe contains a sidebar to navigate back to the homepage. Additionally, the SE can also access SWAP functionality from within Workbench.

To implement it, we used HTML Iframe element described in Table 17, Appendix D to wrap the built URL within Workbench. This way, the SEs can install applications from a single unified application. Just like accessing service application information from Manage Applications File and hostname, SWAP also uses the same functionality. The Manage Applications File can be seen in Figure 14.

3.5.3. Viewing SDP Health alerts

Design

In addition to giving access to Grafana, the Workbench also fetches alerts from Grafana at a fixed regular interval. Figure 18. shows how the alerts are gathered and filtered based on severity level. A traffic light that shows colored output is present in the Client rendered template. The Manage Applications API sub-module requests for alerts from Grafana. Once the alerts are gathered, the sub-module checks for severity levels in the alert output. The severity levels were decided as a standard interface between SDP and Workbench for denoting alerts. They are of three types:

- **Critical** – This is for resources that are very crucial and require immediate troubleshooting.
- **Warning** – This is for resources that do not require immediate troubleshooting.
- **Ok** – This denotes that all SDP resources are healthy.

The color enforces an SE to check Grafana application for troubleshooting. Another functionality the traffic light provides is accessing Grafana when clicked. This uses the same mechanism mentioned in Section 3.5.1.

Implementation

Figure 35 shows the final design of the Traffic light [34] on the home page. The traffic light shows three colors. These colors can be seen in Figure 15, which signify the severity levels.

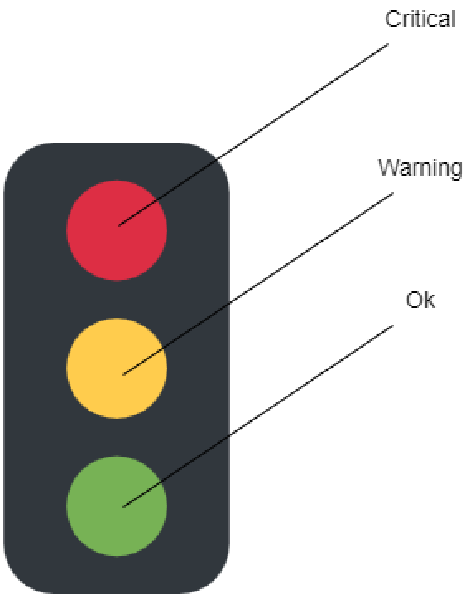


Figure 15. SDP Health traffic light used in Workbench

To implement the traffic light, we used a combination of the http.client [35] library and Grafana alert API [36] in the Manage Applications API sub-module. The http.client library helps fetch a response from Grafana Alert API, which is filtered for severity level. This mechanism fetches the response at regular intervals to notify SEs about the status of SDP Health. The Grafana alert fetch API was decided based on decision D16 in Table 18, Appendix E.

To use the Grafana alert API, an authentication strategy is required for access. Based on decision D18 in Table 18, Appendix E, we decided to use a read-only service account consisting of a username and password for authentication that will only be dedicated for SEs. The SEs do not need to provide these details and are instead parsed from Manage Applications File as shown in Figure 14 under sdp_health, username, and password. This keeps the Workbench simple and complies with NF1.

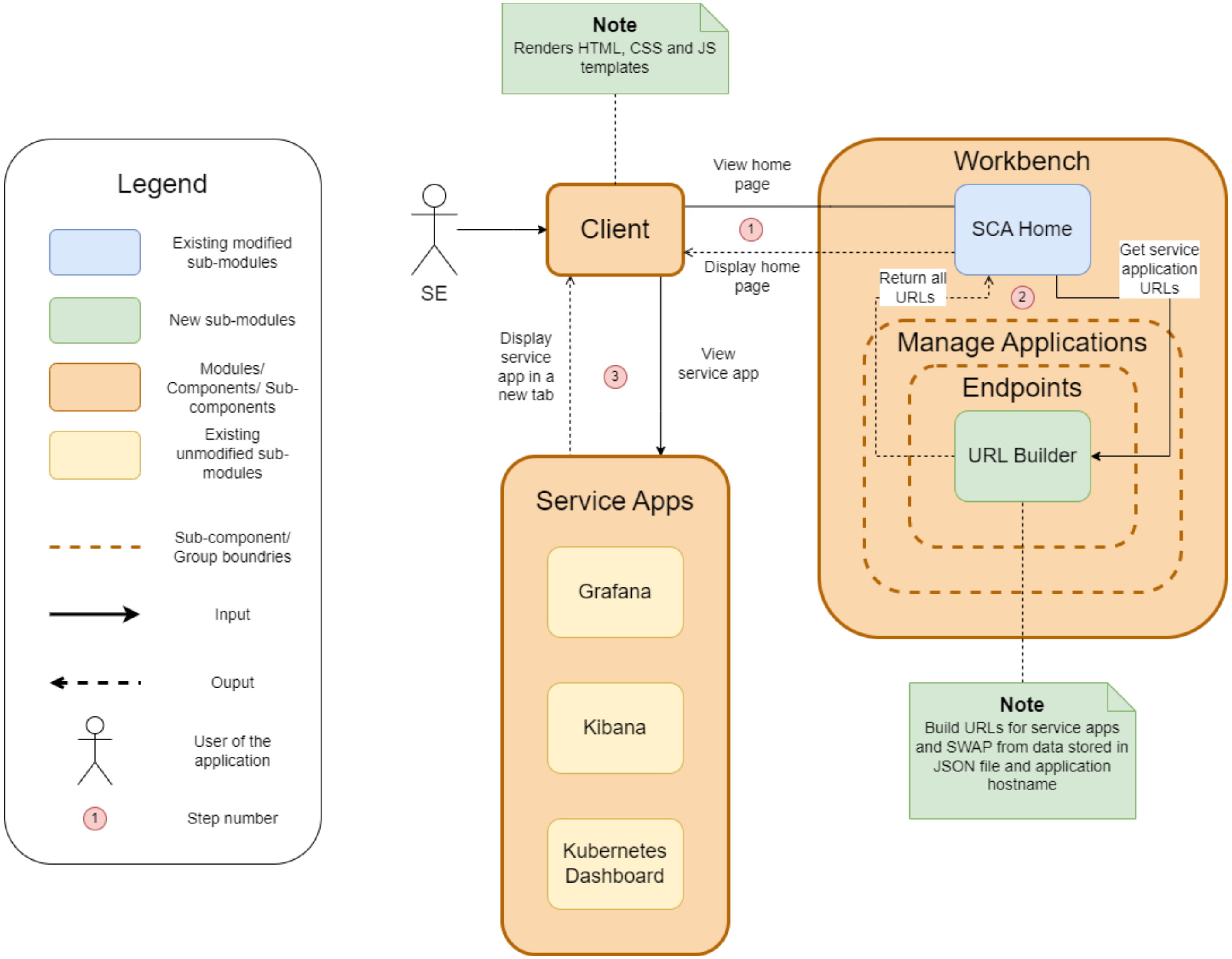


Figure 16. Diagram showing access to service applications

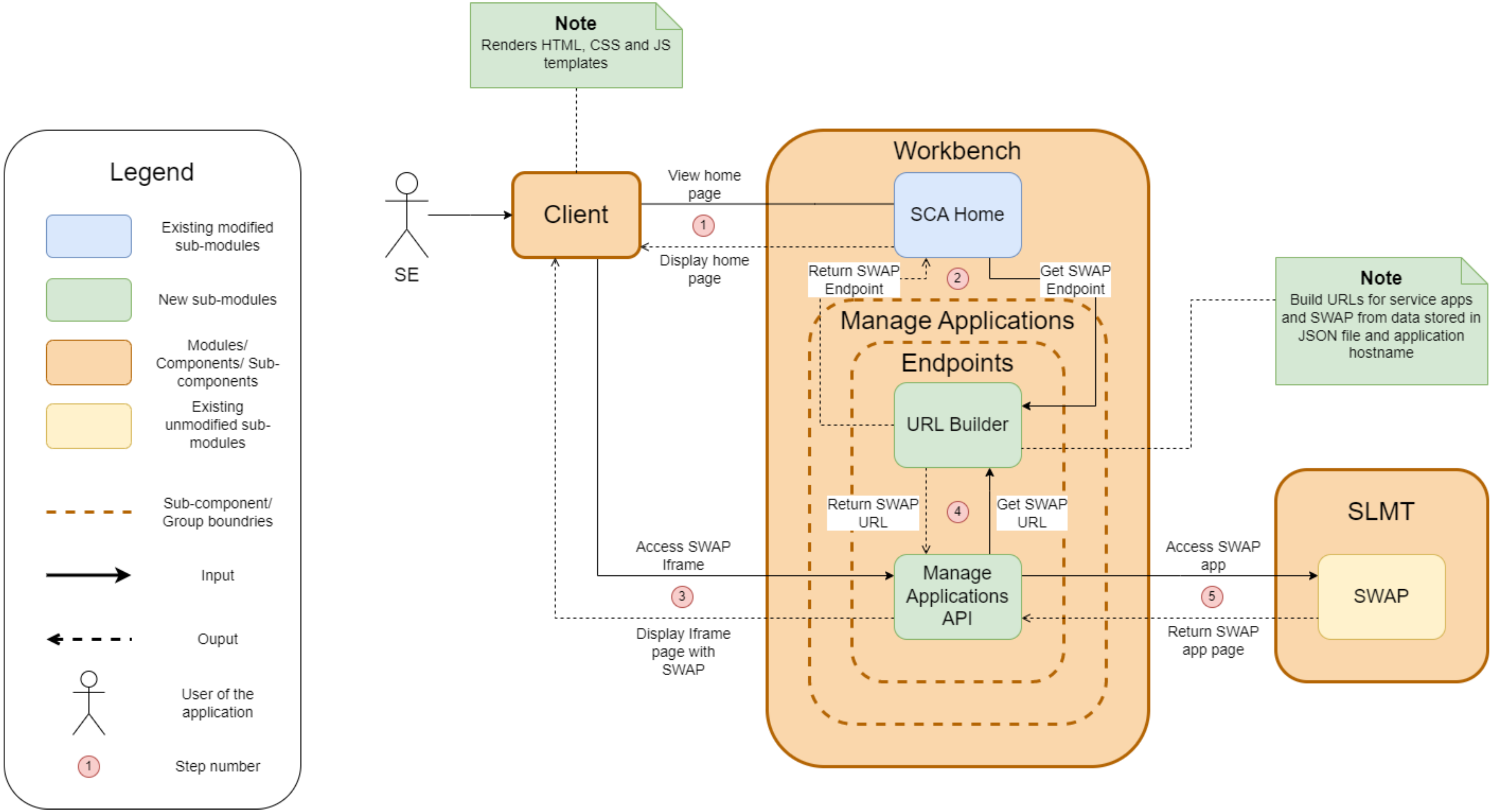


Figure 17. Diagram showing access to SWAP embedded within Workbench

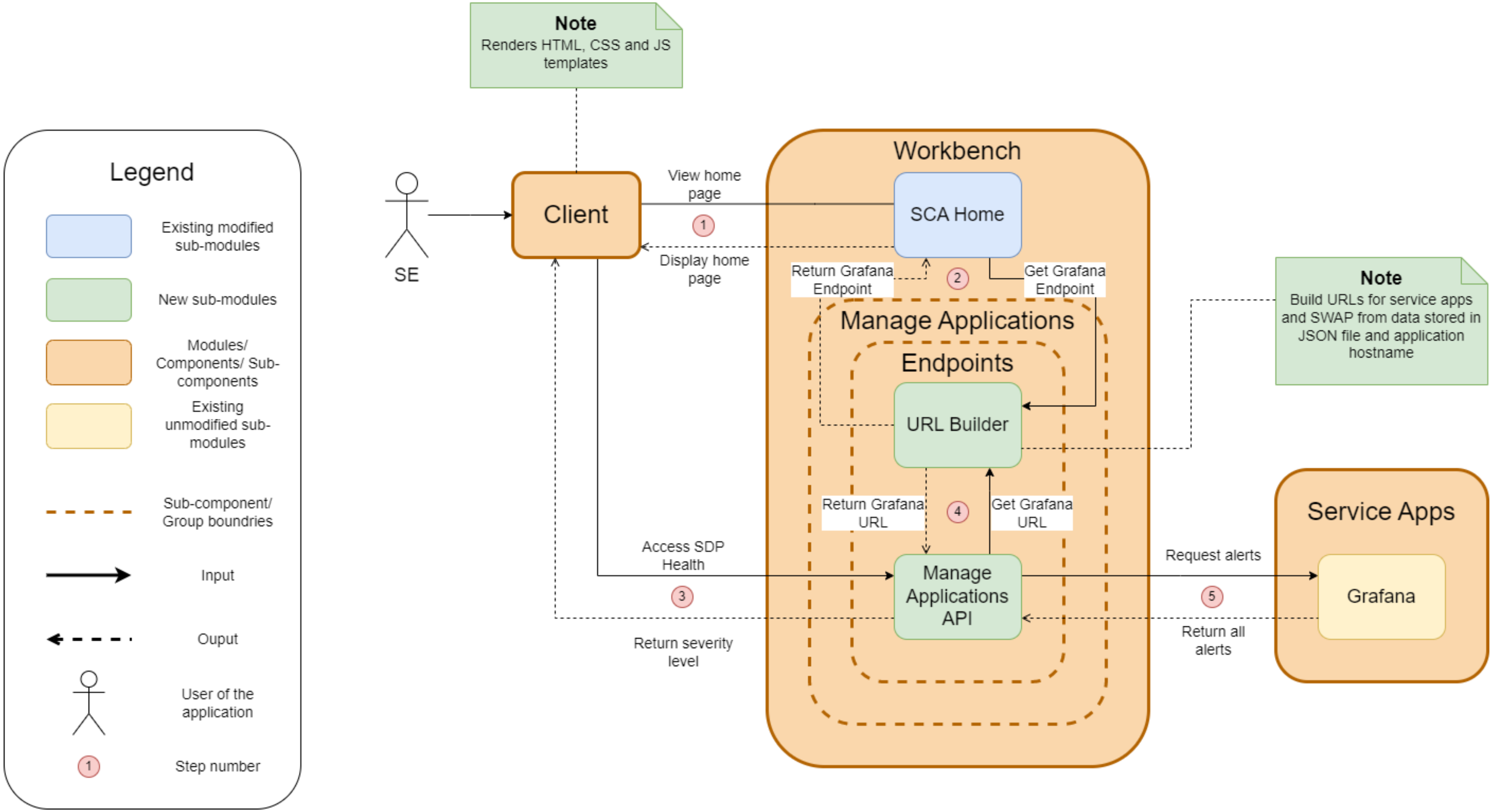


Figure 18. Diagram showing Workbench accessing SDP Health alerts

3.6 Integration – Form Flow

The Form Flow module is the final introduction to the Workbench application. The Form Flow helps integrate SCA with SCI, ensuring the appropriate scripts are installed when an SE applies an action.

Design

Figure 21 shows the integration between SCI and SCA using Form Flow. The integration contains several elements described in Table 10. For simplicity, the components of SCA are showing at a high-level abstraction.

Table 10. Description of internal sub-modules of Form Flows

Sub-module	Description	Interfaces
Other SCA Modules	Contains logic for filling forms and generating the SDP Configuration	Interacts with Form Flow Mapper to retrieve the next and previous form mapping
Pre-Install	SCA module containing business logic to review form filed information before starting an action. It is also the starting point of executing an automation script	Interacts with Form Flow Mapper to retrieve the previous form mapping or transition to applying an action
Action	SCI module containing the business logic to executing an automation script	Interacts with Form Flow Mapper to transition to viewing action history after the action is complete.
Form Flow Mapper	It contains the business logic to traverse through next and previous forms. It also integrates SCI by traversing from SCA forms to applying an action and viewing action history	Interacts with Form Flow File to parse the flow mapping and provide URLs to different SCI and SCA modules
Form Flow File	Contains flow mapping to traverse through next and previous forms. It also contains mappings to traverse from SCA to applying an action and viewing action history in SCI	Provides the Form Flow File Mapper with flow mapping

To explain this module, we need to understand a Form Flow. An SE first accesses the series of forms through Other SCA Modules, as shown in Figure 21, Step 1. As the forms are traversed, the Form Flow Mapper provides the web page URLs in the background from the Form Flow File. The mapper is connected to the existing next and previous functionality in SCA Forms. After filling out all the forms, the SE reaches the Pre-Install Module, where the SE can review the contents of the forms entered. If some information is entered incorrectly, the SE can go to previous or go transition to action web page to execute the script in the background. Finally, in step 3, after the automation script is complete, the SE can view the complete Action History, the current action `exit status`, and its audit content by closing the action page.

This can be further explained using Figure 22. It shows an example of a Form Flow for Complete Action. The Form Flow starts from the home page, where the SE selects the Complete Action to be applied. It then traverses through a series of forms using the next or previous URLs to add or update form information. This generates the SDP Configuration which can then be viewed on the Pre-Install page. The SE can then apply the Complete Action and the automation script for the Complete Action begins to execute in the background. The SE is transitioned to the action web page and the SE can close the action page and view Action History after the action is completed.

Implementation

We reused the existing next and previous buttons in Form Flows to implement the functionality. Figure 44 shows an example of these buttons that help an SE traverse the forms. To bridge the SCI and SCA, we added the Pre-Install page as a new component that helps SE review the SDP Configuration before applying an action. This is depicted by the final Pre-install web page in Figure 45. The web page also shows the previous and install button to update the details or apply an action. Once an action is applied and completed, the SE can then view the Action History by selecting the close button, as shown in Figure 36.

Figure 19. Image showing the flow parameter in the URL that Form Flow Mapper uses

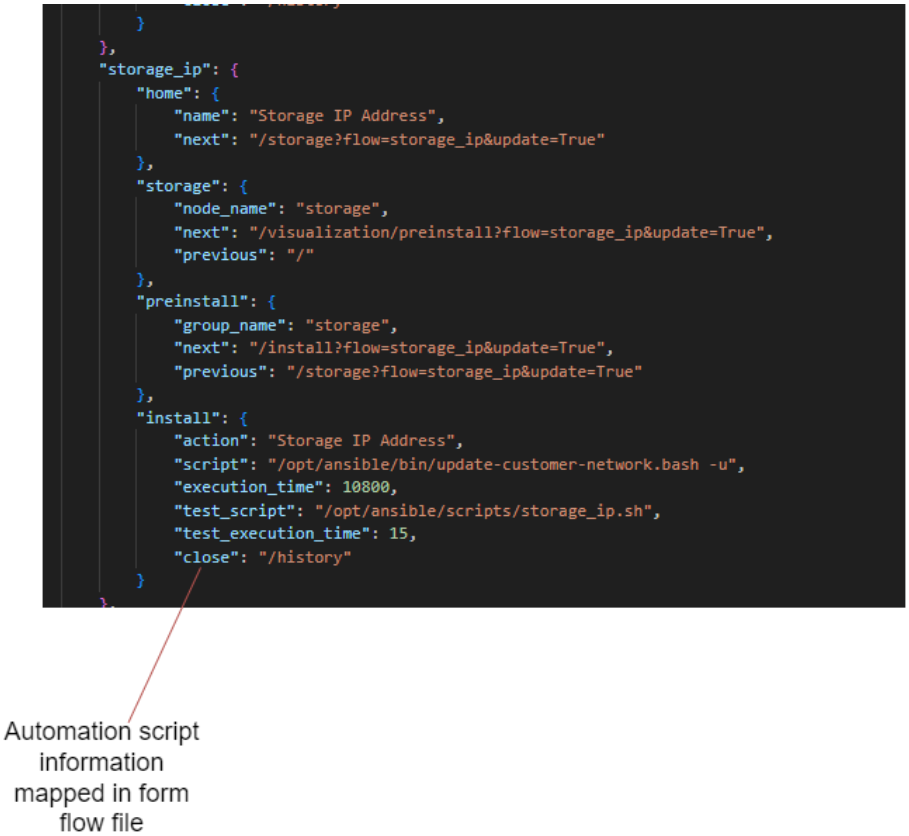


Figure 20. Image showing the structure of the Form Flow File

The implementation depends on a flow value as shown in Figure 19. This flow value is used across actions to determine the type of action being updated or applied and determine the next and the previous forms. Figure 20 shows the structure of a specific Form Flow in the Form Flow File. The Form Flow File is implemented using JSON. The flow value is the first key of the JSON, which is storage_ip in this example. The second-order keys, home, storage, pre-install and install (action), represent the forms for this flow. Each form contains the URLs to traverse through the flow. To comply with NF3, we combined the automation script information with the flow file in the action page denoted by install.

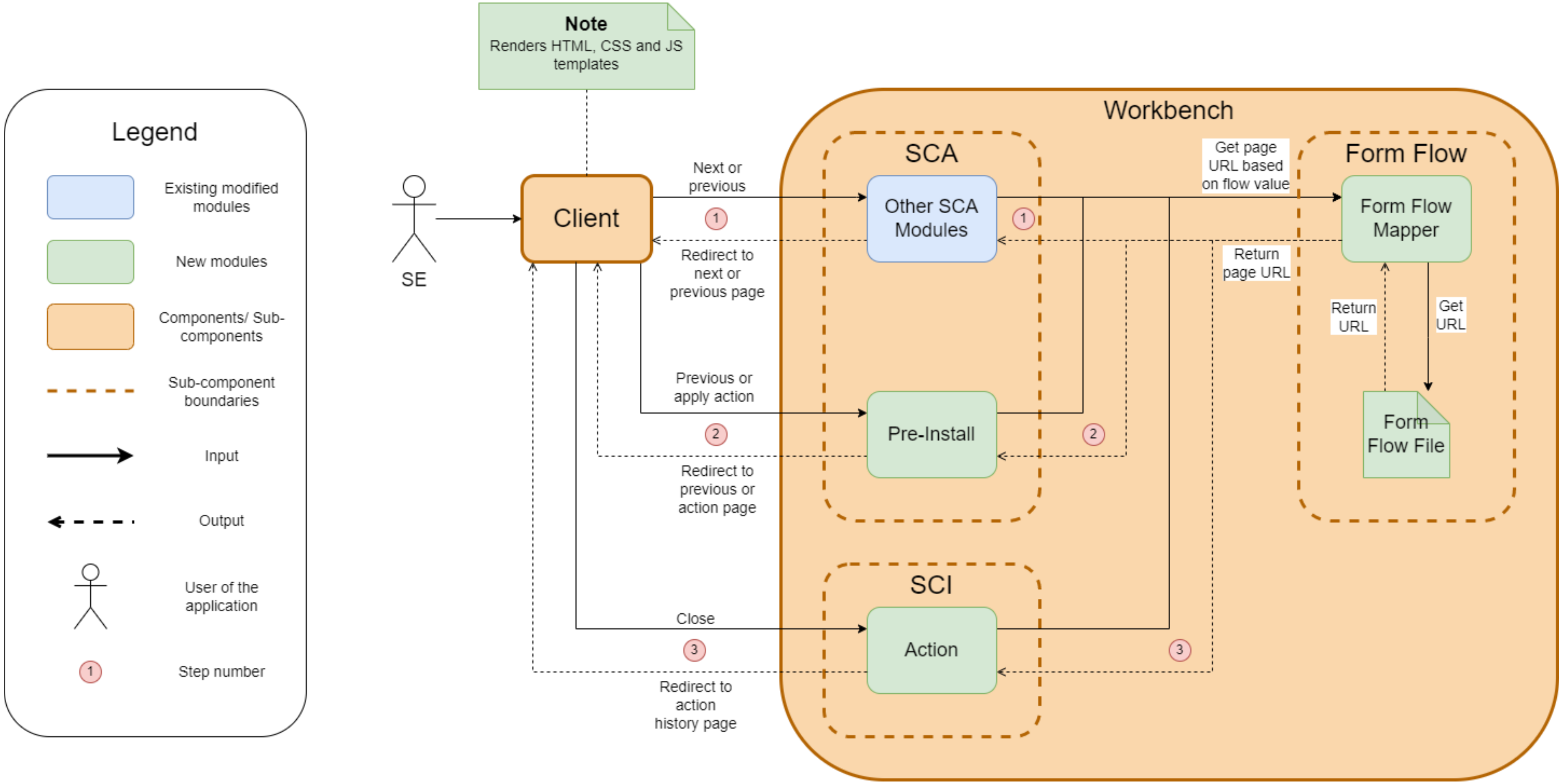


Figure 21. Diagram showing the steps to transition from SCI to SCA

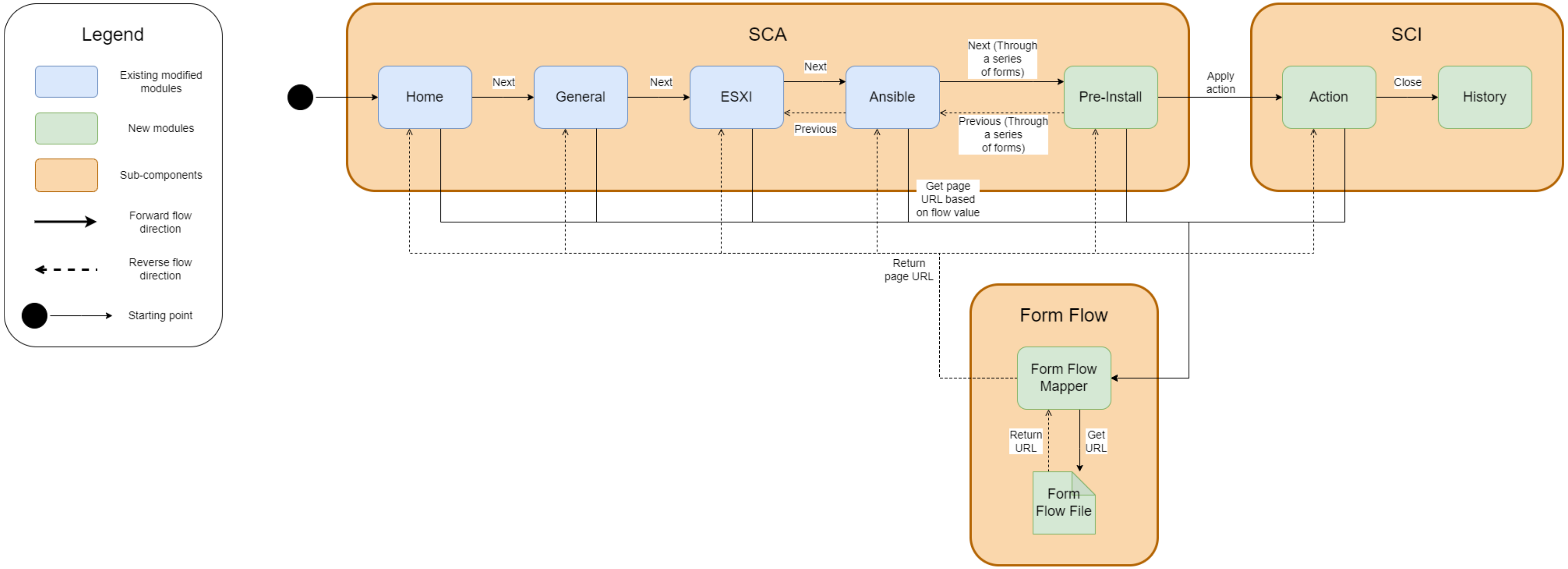


Figure 22. Diagram showing an example of complete Form Flow

4. Verification & Validation

4.1 Overview

This chapter covered various techniques and methods for verifying and validating software. In the end, we also discuss the feedback from the stakeholders that can be implemented for further improvement of Workbench.

4.2 Verification

We performed verification through several tests. These tests were automated and integrated into the GitLab CI [37] pipeline for the testing environment and Make file [38] for the local environment. Table 17 in Appendix D shows the tools and technologies used for verification.

4.2.1. End-to-end testing

We developed these tests to check the end-to-end behavior of the application. We created new end-to-end tests to test the behavior and modified some old tests for our application. Table 11 shows the test specifications and the requirements associated with the tests. We developed 39 test cases, of which only core test cases have been shown. The test cases have been simplified for readability. Additionally, we combined test case T1 with previous SCA test cases to test complete Form Flows.

Table 11. Traceability between requirements and test cases

Requirement ID	Test ID	Test Description	Test Specification
F1 – F2	T1	Applying a complete action successfully	**Given** I am on the pre-install page with the SDP Configuration **When** I click on install on the pre-install page **Then** I should see action output on the action page **And** I should see success on the action page **And** I should see a success audit on the history page
	T2	Applying an action which fails	**Given** I am on the pre-install page with the SDP Configuration **When** I click on install on the pre-install page **Then** I should see action output on the action page **And** I should see error on the action page. **And** I should see a failed audit on the history page
	T3	Cancelling an action in progress	**Given** I am on the pre-install page with the SDP Configuration and an action in progress **When** I click on the cancel button **Then** I should see action cancelled on the action page **And** I should see a failed audit on the history page

Requirement ID	Test ID	Test Description	Test Specification
	T4	Auto reconnect to an action in progress	**Given** I am on the pre-install page with the SDP Configuration and an action in progress **When** I close Workbench **And** reopen it to access the home page **Then** I should see the action in progress on the home page **And** I can click on "to installer" button to check the progress of action
F3	Fulfilled by T1 – T4		
F4	Fulfilled by T1 – T4		
F5	T5	Viewing action history	**Given** I am on the home page with the SDP Configuration and some automation audits present **When** I access the action history page **Then** I should see a list of automation audits on action history
	T6	Viewing an audit content	**Given** I am on the action history page **When** I click on the view content button for an action **Then** I should see the audit content in the audit content page
F6	T7	Viewing the traffic light	**Given** I am on the home page with the SDP Configuration and the complete action applied **When** I look at the SDP health traffic light **Then** I should see a color on the traffic light denoting the status of the SDP infrastructure
	T8	Accessing SWAP	**Given** I am on the home page with the SDP Configuration and the complete action applied **When** I click on "view SWAP dashboard" button within "Manage Applications" option **Then** I should see the SWAP Iframe page with SWAP embedded within Workbench
	T9	Accessing a service application	**Given** I am on the home page with the SDP Configuration and the complete action applied **When** I click on service application link from the "Manage Applications" option other than SWAP **Then** I should see the service application in another browser tab

To test these end-to-end test cases, we also created a new strategy for testing the functional requirements for local and testing environments. In this strategy, we created mock automation scripts to test the functionality and execute them in an isolated mock automation scripts container since the original scripts required the SDP infrastructure resources. These resources were not available in the local testing environment. This strategy can be explained in Figure 23, which shows three docker containers and two docker volumes. The arrows represent the input and output between containers. The black lines denote the connection between the volumes and the containers. The elements and their interfaces are described further in Table 12.

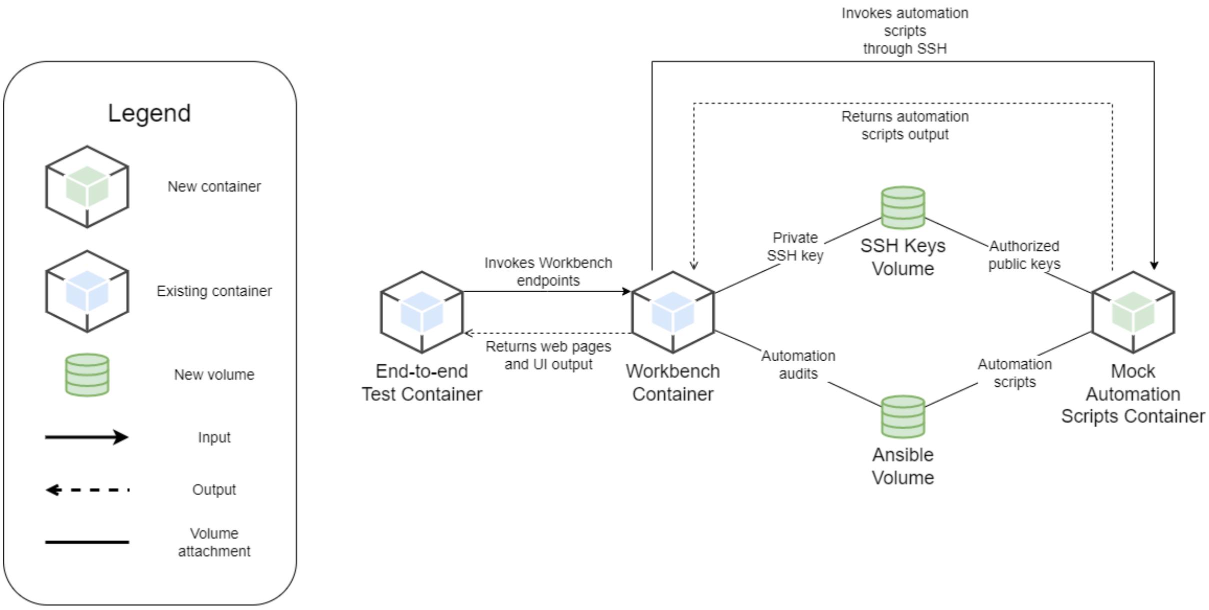


Figure 23. Diagram showing the interactions between containers for testing

Table 12. Description of components used for End-to-end testing

Component	Description	Interfaces
Ansible volume	A docker volume that contains mock automation scripts and their automation audits	Provides mock automation scripts container with automation scripts, and Workbench with automation audits
SSH keys volume	A docker volume that contains SSH keys for connecting to mock automation scripts container	Provides mock automation scripts container with authorized public key and Workbench with private SSH key
Mock automation scripts container	A docker container that behaves like an Open-SSH [20] server	Sends automation scripts output to Workbench
Workbench container	A docker container running the Workbench application	Invokes automation scripts in the script's container through SSH and sends the output to Cypress tests
End-to-end test container	A docker container that executes Cypress [39] tests	Interacts with Workbench endpoints and verifies UI output

Instead of connecting the SE to the VM described in 3.3.1, the application is connected to the mock automation scripts container in the local and testing environment through SSH protocol. The application invokes the mock automation scripts by using the private SSH key present in the SSH keys volume. This is validated by the authorized key file containing the public SSH key. The script is executed once the connection is established, and the output is sent to the Workbench. The mapping to mock scripts is also mapped in the Form Flow File as test_script, as shown in Figure 20, for RNDE to update them easily. This strategy helps with the following:

- Test the SSH behavior of the application to automate scripts
- Test requirements F1- F6
- Give access to mock automation audits generated by the application
- Decouple and isolate the testing of applying an action from the local and testing environment, making it reusable

4.2.2. Unit testing

We also added new tests and modified some old tests. The old tests that we modified were testing endpoints for different SCA and SCI components. The modification introduced the flow value mentioned in Section 3.6 to test for Form Flow support. In addition, we introduced tests for pre-install and action endpoints to test the Form Flows further.

4.2.3. Static analysis

We also integrated a Pylint [40] static analyzer to analyze code. It helped to check the Python code quality and ensured it followed the PEP-8 [41] style guide. It was also integrated into GitLab CI, Make file, and pre-commit hooks to ensure the committed followed the style guide.

4.3 Validation

We performed validation through demo sessions, consultations, dry runs, and code reviews with stakeholders to ensure the application was designed according to the requirements. These sessions also validated the non-functional requirements with crucial stakeholders, NF1 – NF4. All the sessions were organized based on the project working process, described further in Section 6.3. Toward the end of the project, we also organized a final feedback session to gather feedback from stakeholders.

4.4 Final Feedback

During the final feedback session, the stakeholders gave several feedbacks. The feedback is shown in Table 13.

Table 13. Detailed feedback for different elements of Workbench

Feedback
UI
• Some graphic elements, which include the web pages, can be made more self-guided, such as providing additional information when hovering over buttons, such as those shown in Figure 35, Figure 37, and Figure 43. • Some graphic elements can be replaced by elements provided by Thermo Fisher Scientific design guidelines so that Workbench becomes more coherent with other applications, making it easier for SEs to recognize them. • The names of the buttons shown in Figure 37 could be improved further to make the application more self-guided. • Certain graphic elements can be improved further to have a more polished look. • It is essential to provide a graphical popup on the action page when an action is completed to make it more evident to the SEs.
Functionality
• The functionality of flows, SWAP, and accessing other service applications looks good. • The actions in Figure 37 could be disabled when an action is in progress to prevent other SEs from starting another action.

In general, Workbench looks good to the stakeholders and fulfills the core Needs of the SEs. It is part of SDP release 2.14.0 Q3 as an experimental version for SEs for experimentation. This will further help to improve Workbench and stability the product.

5.Conclusion

5.1 Overview

To summarize, this project presented an opportunity to develop a simple and easy-to-use web-based GUI application called Workbench that bridges the gap between FSEs with different skills and backgrounds to install, upgrade, and troubleshoot SDP independently.

To solve this, we interviewed various stakeholders to understand the business and project problem as described in Chapter 1 and the Needs described in Chapter 2. These helped to elicit our requirements and develop Use Cases, such as Applying An Action, Viewing Action Output, accessing the Action History, and accessing Manage Applications.

We developed a solution for Workbench from the requirements mentioned in Chapter 3. It contains SCI that allows only a single action to be applied. The SCI also uses Web Sockets, which support auto reconnection if the SE closes the browser. SCI is integrated with SCA through Form Flows. Additionally, our application allows access to Action History to show the automation audits generated during automation script execution. It also shows the exit status of the action. The application also contains access to Manage Applications for which the URLs are added dynamically so that the SEs can access them without additional configuration.

To verify and validate the solution, we used several ways. Additionally, we devised a new strategy to test mock scripts in the local and test environment, as mentioned in Chapter 4. Toward the end of the project, we organized a final feedback session where the stakeholders tried the application, and we collected valuable feedback. This feedback will help to improve the Workbench further and make it a usable and stable product.

Workbench is now part of SDP release 2.14.0 Q3 as an experimental version for SEs, which will further help in its stabilization.

5.2 Recommendation for Future Work

From the feedback collected in Section 4.4, we recommend that the RNDE improve some aspects of the application to stabilize Workbench further. The improvements are further described in the following section.

5.2.1. UI

The UI could be made more self-explanatory. As suggested by stakeholders in the final feedback session, this could be achieved by showing more information when hovering over elements. Figure 24 shows an example of this behavior.

Figure 24. Image showing extra information for an element

Turning off the Quick Action button shown in Figure 37 would be better when applying an action. This behavior would ensure that SEs could not start another action while an action is in progress. Before applying an action, the SEs can review the form information on the pre-install page. However, the UI of the pre-install page should be improved for readability.

The progress bar progresses to 100% only when an action is completed. This behavior should be extended to show the progress bar updates as the action page shows output, giving a more realistic feel of how much the automation script has progressed in the background. Moreover, the action page could also show how much time the execution of the automation script would take in the background.

Additionally, certain small UI elements, such as texts, could be changed to make the UI more readable and understandable to SEs.

5.2.2. Functionality

The RNDEs could add an extra column in Action History showing the version number of the Complete Action applied. A standard could be set up between the audits generated and Workbench. The audit could show the Complete Action version being applied as the first line, which the Workbench could parse. Once successful, the Action History could also show this information.

Additionally, the RNDEs could also enable downloading audits from the Action History. This would ensure the SE could also save the audit to show others the automation problems and attach them as evidence for further technical support. This feature should be accompanied by audit size as an additional column on the Action History to show how extensive the audit is and indicate SE the time it could take to load and download it.

The production automation scripts do not contain the interface to generate audits when their execution is completed. In the future, the RNDEs must add this behavior for Action History. This functionality could be done in two ways:

- Interface through automation scripts
  - **Pro** – Adding to automation scripts would ensure any script could generate the audits
  - **Con** – Require modifications to automation scripts
- Interface through Workbench
  - **Pro** – Workbench would generate the audit and store the automation output, preventing modifications of automation scripts
  - **Con** – Automation scripts not executed through the Workbench would not generate any audits and, therefore, would not show on the Action History

This choice must be made by the RNDEs for audit generation and viewing audits in history.

Besides the Workbench, SWAP is a tool that installs and updates applications on the Kubernetes Cluster, as explained in Appendix B. When Workbench becomes operational alongside SWAP, both could perform execution simultaneously. This could result in SDP entering a failed state. This could be prevented by adding an API [42] in Workbench that SWAP could use to check if an action is in progress.

To improve usability further, Workbench could also manage auto sign-on for service applications to simplify access for SEs and not be concerned with managing external credentials. This behavior would further improve NF1.

5.2.3. Testing

Further testing is needed for Workbench. This includes adding more unit tests to the current test suite and resolving bugs found in dry runs. Even though the application was tested on the production server, it is yet to be tested with the entire three-step SDP installation explained in Appendix B. This will help test the Workbench*'s* stability and resolve further bugs that could improve it.

Additionally, we recommend that the RNDEs use BDD test specifications [43] for unit tests. This can be achieved through Behave [44], which supports this approach with Python. Through this, test cases will become human-readable, and it will be easier to communicate and validate tests with SEs.

5.2.4. Continuous Integration

Since SCA is also evolving independently, the Workbench contains a slightly outdated version of SCA. The Workbench features exist in a separate branch in the current SCA Gitlab code repository. It should be merged with the main SCA code to keep a single version moving forward.

6.Project Management

6.1 Overview

This chapter explains the Project Management and the processes used during the project. It also addresses formats and conventions used for documenting the project. The overall project was executed using agile [45] evolutionarily. Smaller processes were developed based on the Plan Do Check Act (PDCA) cycle [46]. These were modified throughout the project to fit our working style. The modifications were through trial and error.

6.2 Project Planning

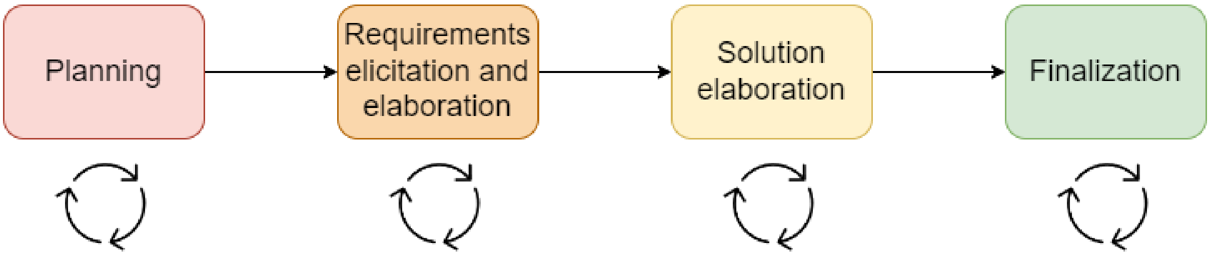


Figure 25. Image explaining the overall project timeline

The project started on 2 January 2023 till 31 October 2023. To manage the project schedule, we created an overall project timeline. The project timeline was based on agile, where it was managed evolutionarily. Figure 25 shows the main phases of the timeline. The descriptions are as follows:

- **Planning –** During this phase, we developed a Project Management Plan (PMP) [47] using ISO 16326:2019 standard [48] outlining the basic processes. We also created a *Risk* and *Stakeholder Register* to manage the project.
- **Requirements elicitation and elaboration –** During this phase, we identified the Needs and elicited the core System Requirements from the stakeholders. The requirements were elaborated in the Requirements Register and consulted with stakeholders.
- **Solution elaboration –** During this phase, we developed the architecture and experimented with different tools and technology before implementing Workbench. We also verified and validated Workbench through end-to-end tests and demo sessions.
- **Finalization –** During this phase, we tested the application in the production server to find production bugs and the product's stability. We also finalized the documents for handover such as the confluence page with updated project information

Granular planning was using Milestone Trend Analysis (MTA) [49]. An MTA is a method to track the project process. It helps in measuring and checking deviations between the planned and the actual dates of a milestone achieved. A milestone is essential in the project's progress when a deliverable is delivered to the client. These milestones were used to measure and show the progress of the project.

6.3 Process Management

We used an extended version of our working process. Figure 26 shows the stages of our process based on PDCA. The stages are explained as follows:

- Plan
  - **Decide Tasks** – In this stage, tasks such as making a document or implementing a feature were decided to be completed.
- Do

-
  - **Experiment** – In this stage, the solution is experimented to decide the appropriate approach for a problem and identify trade-offs.
  - **Implement** – Stage where the solution is implemented and tested for the End-to-end behavior of the application using a series of test cases.
  - **Draft** – Stage where a draft is created for a document
- Check
  - **Consult** – Stage where the draft document or code was reviewed.
  - **Demo** – Stage where the implemented feature or mock-ups were demonstrated to the Service Engineers
- Act
  - **Update** – Stage where changes were updated based on the feedback received
  - **Refactor** – Stage where code was updated based on the feedback received

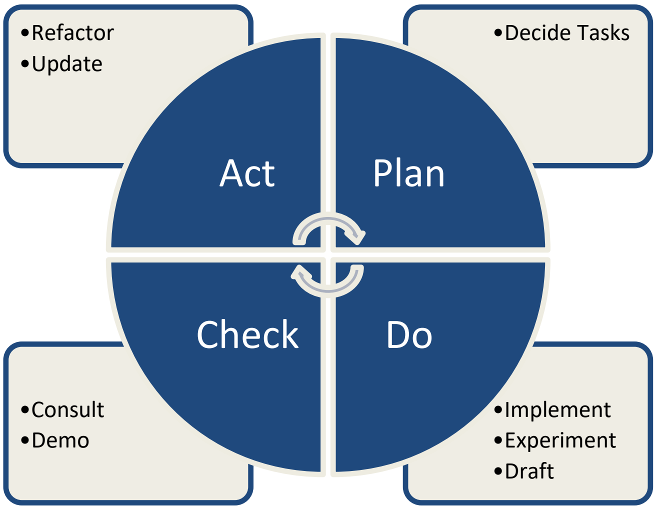


Figure 26. Diagram showing the working process

6.4 Risk Management

Figure 27. Diagram showing the risk management process

We managed *Risk* in a *Risk Register*. The top two Risks can be seen in Table 23, Appendix H. The register contains the following information:

- **ID** – ID of the Risk
- **Risk Description** – Explains what the Risk is about
- **Risk Category** – Shows the category of the Risk identified
- **Impact Level** – Explains how strong its impact will be on the project
- **Probability Level** – Shows how likely the Risk is going to happen
- **Severity Level** – Product of Impact and Probability and shows how severe the Risk is
- **Mitigation** – Strategy to prevent the Risk from happening
- **Contingency** – Strategy to use in case the Risk occurs
- **Status** – Show the status of the Risk

The Risk Register provided an opportunity to decide the critical mitigation strategy to avoid the Risk from taking place or a contingency plan in case the Risk occurs. It was updated monthly or whenever an uncontrolled event occurred. The most critical Risks were then discussed with the supervisors to decide a possible way forward based on the mitigation and contingency plan. Figure 27 shows the cycle of the process. The stages are explained as follows:

- **Identification** – Identifying risk which is something out of control
- **Mitigation or Contingency** – Identify strategies to prevent the risk from happening or steps to reduce if it has already occurred
- **Consultation** – If the risk has a high impact on the project and high occurrence, then consult with Supervisors to decide a way forward
- **Action** – Act based on the feedback received

6.5 Change Management

Changes were common during the project. To accommodate changes, we followed a process as shown in Figure 28.

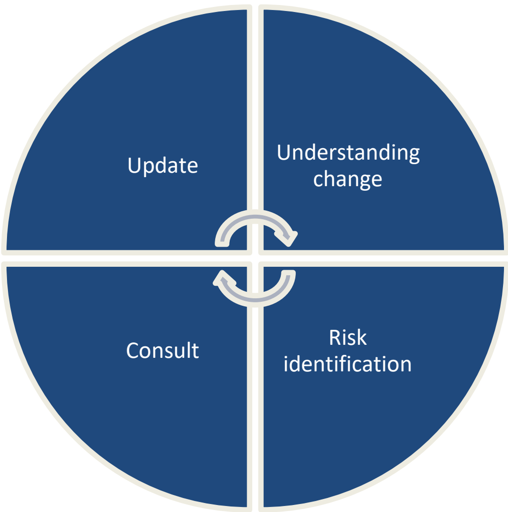


Figure 28. Diagram showing the change management process

The stages are explained as follows:

- **Understanding change** – This step explores changes and adds them to the Needs Register when requested by stakeholders.
- **Risk identification** – This step identifies risks and their impact based on remaining time available and feasibility. Trade-offs were also developed in this phase, noting the pros and cons of approaches.
- **Consult** – This step is where Supervisors were consulted with the change request and possible solutions moving forward.
- **Update** – This step is where the change was either implemented based on priority or termed as out of scope for the project

6.6 Needs & Requirements Management

Another critical aspect of the project was managing the Needs and the requirements. These were maintained in the Needs and Requirements Registers, respectively. The Needs Register contained a list of Needs based on their priority. The Needs Register acted as a document for consultation whenever new Needs were identified, or a change occurred. If a Need was crucial for Workbench, it was then moved to the Requirements Register, where it was formally written. We used Must have, Should have, Could have, and Won't have (MoSCoW) [50] to define priorities for the requirements. We followed an elicitation process to gather and update the Needs and requirements. Figure 29 shows the process in detail.

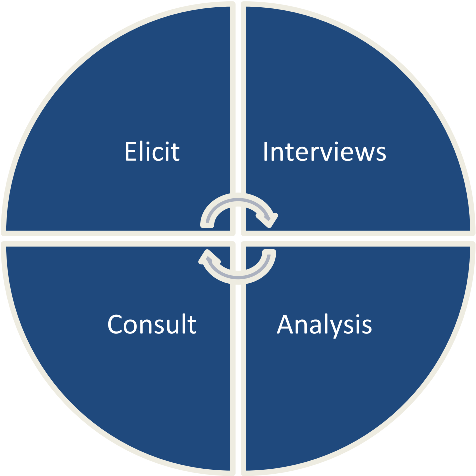


Figure 29. Diagram showing the interview to requirements process

The stages are explained as follows:

- **Interviews** – This step is where interviews were planned and conducted with stakeholders
- **Analysis** – This step is where interviews were analyzed, and the Needs Register was updated
- **Consult** – This step is where Supervisors were consulted for the Needs
- **Elicit** – This step is where requirements were formulated, and a core set of Needs were elicited

To document the Needs, we developed a simple format for the register., Table 14 show the headings of the Needs Register.

Table 14. Headings used for documenting Needs in the Needs Register

Headings	Purpose
Needs ID	Denotes the ID of the Needs
Needs	Denotes the description of the Needs
Considered	Denotes if the need was considered for the project
Requester	Denotes the stakeholder who requested the need
Remarks	Denotes extra description of the Needs or additional notes derived from the interviews

6.7 Architecture & Decision Management

We documented our architecture using the 4 + 1 view model [51]. It provided several views to our application design. These can be seen in Appendix G. Additionally, to analyze alternatives for library implementation and making architectural choices, we used decision matrices to analyze the pros and cons. These helped to make an informed decision for our architecture and implementation. These decisions can be explored further in Appendix E.

6.8 Quality Management

For quality management, we had to organize various sessions. These were either through consultations, demos, and dry runs. These sessions were opportunities to gather valuable feedback and improve our documentation or implementation.

6.8.1. Consultations

Consultations were used to validate documents. These documents include the Needs Register, requirement register, architecture and design diagrams, mock-ups, and decision matrix. Various stakeholders were invited during this session to validate the documents. The DSEs, Operations Engineer (OE), and Global Technical Service (GTS) were consulted for mock-ups, and RNDEs were consulted for technical documents such as architecture and design diagrams. Feedback from these sessions helped us to better understand the Needs, clarify requirements, improve the architecture, design, and mock-ups, and make better decisions.

6.8.2. Demo Sessions

We also organized demo sessions with critical stakeholders. These sessions helped to validate the UI and the end-to-end behavior of the application. In these sessions, the SEs obtained the opportunity to see and try the application and give feedback. The feedback from these sessions helped improve the UI and functionality of the application to meet the stakeholders' Needs.

6.8.3. Dry Run

Dry runs were organized towards the end of the project. These were organized to run the Workbench application in a production server to check if it fulfilled the functionality, such as applying an action. Additionally, these helped to check if the application was running without any bugs or errors in the production server.

7.Project Retrospective

7.1 Overview

Since I was the primary owner of the project, It was a new experience, and I learned various lessons during its course. This chapter will discuss the key lessons learned from those challenges.

7.2 Lessons Learned

Accepting the unknown – This is the more critical learning. During the project, when I did not know the answer to a question, I used to present my assumptions. I was provided with feedback from my supervisors that rather than assuming things, it is essential to accept that I do not know the answer and ask for clarification. Doing so will help me to be more explicit in my understanding.

Asking the question why – Asking the question why had been uneasy for me. During the project, there were moments when I was unclear about the domain, and therefore, I had to communicate with the stakeholders multiple times. I was provided feedback from my Company Supervisor that asking such questions should become the norm to ensure that I do things with a purpose and a reason and understand the stakeholder requirements better.

Understanding priorities – During the project, there were many instances when I was pushing myself to complete multiple requirements simultaneously. However, I realized that since the project time was limited, I could only complete specific requirements requested by the stakeholders. I felt that rather than accepting everything, selecting tasks based on priorities and addressing them for delivery was crucial.

Testing along with implementation – One of the biggest lessons learned was writing tests as part of the implementation. I remember when my Company Supervisor recommended that I implement tests before writing the implementation. When I applied this approach, I noticed that thinking about tests first also enforced considering edge cases and handling them accordingly. Moreover, the cycle for implementing and refactoring was also reduced. Therefore, I will prefer using this approach in my future career.

Always consult if faced with blockers – I habitually worked alone before the project. During the project, when I was stuck on a problem for some time, I decided to ask for some implementations help from RNDEs. The problem was resolved within 30 minutes, and I then understood the benefits of communicating blocks with the team as it solved my problem faster.

Always think of the production environment from the beginning – Towards the end of the project, some issues appeared when I deployed my code to the production environment for the dry run. Though we solved the major issues, I realized I could have detected this problem earlier if I had more concretely considered CI/CD and the production environment. Therefore, I will prefer using this approach in my future career for application stability.

Overall, the project boosted my professional career as a Software Engineer and Technical Designer. The project helped develop my technical and communication skills and better software solutions.

References

[1] Thermo Fisher Scientific, "About," Thermo Fisher Scientific, 2023. [Online]. Available: https://corporate.thermofisher.com/us/en/index/about.html.

[2] P. Kirby, "Transmission Electron Microscopy in Semiconductors: Generating Ground Truth Insights," Thermo Fisher Scientific, 05 08 2023. [Online]. Available: https://www.thermofisher.com/blog/semiconductors/tem-analysis-semiconductor-development/.

[3] peppertop, "The Linux command line for beginners," 2023. [Online]. Available: https://ubuntu.com/tutorials/command-line-for-beginners#1-overview.

[4] K. A. R. Putra, "A solution for configuring an Infrastructure-as-a-Service," Technische Universiteit Eindhoven, Eindhoven, 2022.

[5] Grafana Labs, "Grafana: The open observability platform," Grafana Labs, 2023. [Online]. Available: https://grafana.com/.

[6] Elasticsearch B.V., "Kibana: Explore, Visualize, Discover Data," 2023. [Online]. Available: https://www.elastic.co/kibana.

[7] The Kubernetes Authors, "What is a Kubernetes cluster?," VMware. Inc., 2023. [Online]. Available: https://www.vmware.com/nl/topics/glossary/content/kubernetes-cluster.html.

[8] S. Piechottka, "Template: User Needs List," 27 June 2022. [Online]. Available: https://openregulatory.com/user-needs-list-template-iec-62304/.

[9] O. Eidel, "Writing Software Requirements Based on the IEC 62304," OpenRegulatory, 24 May 2023. [Online]. Available: https://openregulatory.com/software-requirements-iec-62304/.

[10] M. Rehkopf, "User stories with examples and a template," Atlassian, 2023. [Online]. Available: https://www.atlassian.com/agile/project-management/user-stories.

[11] International Organization for Standardization, "ISO/IEC 25010:2011," March 2011. [Online]. Available: https://www.iso.org/standard/35733.html.

[12] IBM Corporation, "Defining use cases," 26 April 2023. [Online]. Available: https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/lifecycle-management/7.0.3?topic=SSYMRC_7.0.3/com.ibm.rational.rrm.help.doc/topics/c_uc.htm.

[13] Britannica, T. Editors of Encyclopaedia, "Client-server architecture," Encyclopedia Britannica, 17 September 2023. [Online]. Available: https://www.britannica.com/technology/client-server-architecture.

[14] I. Fette and A. Melnikov, "The WebSocket Protocol," Internet Engineering Task Force, December 2011. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6455.

[15] G. E., V. J., H. R. and J. R., "Obeject Creational: Singleton," in *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 2009, pp. 127-134.

[16] Free Software Foundation, Inc., "GNU Bash," Free Software Foundation, Inc., 22 September 2020. [Online]. Available: https://www.gnu.org/software/bash/.

[17] Free Software Foundation, Inc., "The GNU C Library," Free Software Foundation, Inc., 2023. [Online]. Available: https://www.gnu.org/software/libc/manual/html_node/Exit-Status.html.

[18] "ECMA-404," ECMA International, December 2017. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/.

[19] Python Software Foundation, "subprocess - Subprocess management," Python Software Foundation, 2023 October 2023. [Online]. Available: https://docs.python.org/3/library/subprocess.html.

[20] OpenBSD Foundation, "OpenSSH," OpenBSD Foundation, 10 August 2023. [Online]. Available: https://www.openssh.com/.

[21] B. Chesneau, "Gunicorn - WSGI server," 2023. [Online]. Available: https://docs.gunicorn.org/en/latest/index.html.

[22] J. Gelens, "gevent-websocket," Python Software Foundation, 12 March 2017. [Online]. Available: https://pypi.org/project/gevent-websocket/.

[23] D. Bevans, "Asynchronous vs. Synchronous Programming: Key Similarities and Differences," Mendix Technology BV, 19 September 2023. [Online]. Available: https://www.mendix.com/blog/asynchronous-vs-synchronous-programming/.

[24] Docker Inc., "What is a Container?," Docker Inc., 2023. [Online]. Available: https://www.docker.com/resources/what-container/.

[25] J. Brownlee, "Python Threading: The Complete Guide," Super Fast Python, 09 April 2022. [Online]. Available: https://superfastpython.com/threading-in-python/.

[26] M. Grinberg, "Flask-SocketIO," 2018. [Online]. Available: https://flask-socketio.readthedocs.io/en/latest/.

[27] "xterm.js documentation," 2019. [Online]. Available: http://xtermjs.org/.

[28] Socket.IO, "Socket.IO," Socket.IO, 2023. [Online]. Available: https://socket.io/.

[29] "GitHub dropdown-filter," 9 November 2021. [Online]. Available: https://github.com/zangetsu-isshin/dropdown-filter/tree/main.

[30] Python Software Foundation, "Regular expression operations," 23 September 2023. [Online]. Available: https://docs.python.org/3/library/re.html.

[31] "Quickstart," 2023. [Online]. Available: https://flask.palletsprojects.com/en/2.3.x/quickstart/#accessing-request-data.

[32] "Welcome to Flask," 2023. [Online]. Available: https://flask.palletsprojects.com/en/2.3.x/.

[33] Mozilla Foundation, "The Inline Frame element," 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe.

[34] Twitter, Inc, "Vertical Traffic Light SVG Vector," 2019. [Online]. Available: https://www.svgrepo.com/svg/407686/vertical-traffic-light.

[35] Python Software Foundation, "HTTP protocol client," 2023. [Online]. Available: https://docs.python.org/3/library/http.client.html.

[36] Grafana Labs, "HTTP API reference," 2023. [Online]. Available: https://editor.swagger.io/?url=https://raw.githubusercontent.com/grafana/grafana/main/pkg/services/ngalert/api/tooling/post.json.

[37] GitLab B.V., "Get started with GitLab CI/CD," 2023. [Online]. Available: https://docs.gitlab.com/ee/ci/.

[38] Free Software Foundation, Inc., "GNU make," 26 February 2023. [Online]. Available: https://www.gnu.org/software/make/manual/make.html.

[39] "JavaScript Component Testing and E2E Testing Framework," Cypress, [Online]. Available: https://www.cypress.io/. [Accessed May 2023].

[40] "Pylint," 2 October 2023. [Online]. Available: https://www.pylint.org/.

[41] G. v. Rossum, B. Warsaw and N. Coghlan, "PEP 8 - Style Guide for Python Code," 1 August 2013. [Online]. Available: https://peps.python.org/pep-0008/.

[42] "What is an API?," Red Hat, Inc., 2 June 2022. [Online]. Available: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces.

[43] M. Fowler, "GivenWhenThen," 21 August 2013. [Online]. Available: https://martinfowler.com/bliki/GivenWhenThen.html.

[44] J. Engel, B. Rice and R. Jones, "Welcome to behave!," 12 September 2023. [Online]. Available: https://behave.readthedocs.io/en/latest/.

[45] "Principles behind the Agile Manifesto," [Online]. Available: https://agilemanifesto.org/principles.html. [Accessed February 2023].

[46] Lean Enterprise Institute, Inc., "Plan, Do, Check, Act (PDCA)," 2023. [Online]. Available: https://www.lean.org/lexicon-terms/pdca/.

[47] B. Roseke, "Project Management Plan," 21 October 2021. [Online]. Available: https://www.projectengineer.net/project-management-plan-the-12-core-components/.

[48] "ISO/IEC/IEEE International Standard - Systems and software engineering - Life cycle processes - Project management - Redline," *ISO/IEC/IEEE 16326:2019(E) - Redline,* pp. 1-83, 2019.

[49] M. Waida, "What is a Milestone Trend Analysis," 17 July 2021. [Online]. Available: https://www.wrike.com/blog/what-milestone-trend-analysis/.

[50] "Chapter 10: MoSCoW Prioritisation," Agile Business Consortium Limited, 2023. [Online]. Available: https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html.

[51] P. B. Kruchten, "The 4+1 View Model of architecture," *IEEE Software, vol. 12, no. 6,* pp. 42-50, 1995.

[52] A. Ilitchev, "Transmission (TEM) vs. Scanning (SEM) Electron Microscopes: What's the Difference?," 11 November 2019. [Online]. Available: https://www.thermofisher.com/blog/materials/tem-vs-sem-whats-the-difference/.

[53] A. Ilitchev, "Seeing with Electrons: The Anatomy of an Electron Microscope," 11 April 2020. [Online]. Available: https://www.thermofisher.com/blog/atomic-resolution/seeing-with-electrons-the-anatomy-of-an-electron-microscope/.

[54] VMware, Inc., "What is Virtualization?," 2023. [Online]. Available: https://www.vmware.com/nl/solutions/virtualization.html.

[55] Red Hat, Inc., "Ansible is Simple IT Automation," 2023. [Online]. Available: https://www.ansible.com/.

[56] J. Forcier, "Welcome to Paramiko!," 2023. [Online]. Available: https://www.paramiko.org/.

[57] "About ANSI," American National Standards Institute (ANSI), 2023. [Online]. Available: https://ansi.org/about/introduction.

[58] "Modern UI for Ansible," [Online]. Available: https://www.ansible-semaphore.com/. [Accessed 12 September 2023].

[59] "OMG Unified Modeling Language (OMG UML)," Object Management Group, Inc., 05 December 2017. [Online]. Available: https://www.omg.org/spec/UML/2.5.1/PDF.

[60] "Electron Microscopy," 2023. [Online].

[61] "BDD Testing & Collaboration Tools for Teams," SmartBear Software, [Online]. Available: https://cucumber.io/. [Accessed May 2023].

[62] J. Engel, B. Rice and R. Jones, "Welcome to behave!," 12 September 2023. [Online]. Available: https://behave.readthedocs.io/en/latest/.

Appendix A. Light & Electron Microscopy

There are two different types of microscopies. The first is light microscopy [52], which uses visible light to view small objects. It uses glass lenses to focus the light on specimens to magnify and produce the image. The specimens are placed close to the microscope lenses, and the magnification depends on the type and number of lenses used in the microscope. Figure 30 [52] shows an example of a light microscope. It uses a focused light source that passes through the specimen using the condenser lens. The light passes through the lens, which our eyes can then view.

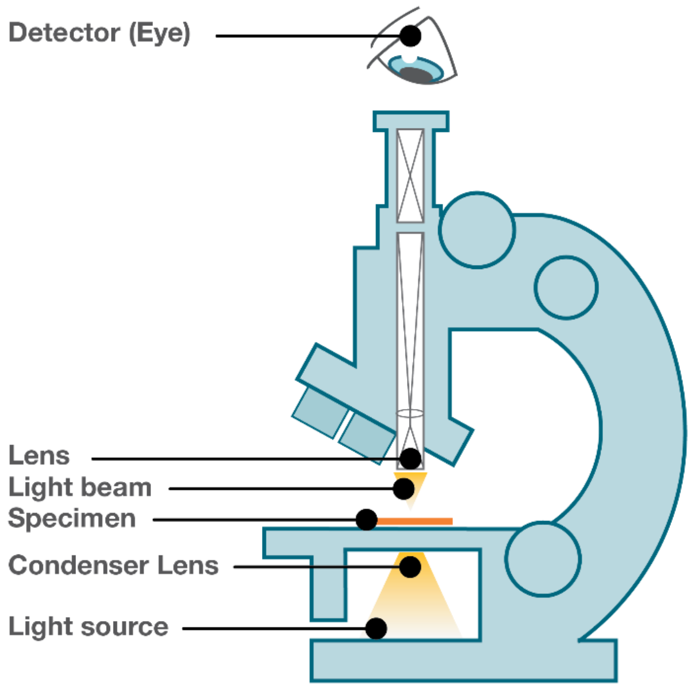


Figure 30. Image showing a cross section of a light microscope [53]

The second type of microscopy uses electrons as a source to see small objects instead of visible light [52]. It allows us to see tiny specimens that are not visible through a light source, as a beam of electrons possesses a wavelength even smaller than visible light. Such specimens included cell internal structures, protein structures, and individual atoms. In this approach, the microscope fires electrons to the specimen through electromagnetic or electrostatic lenses. These lenses focus the electrons through the specimen, which is then captured through an electron imaging device as a detector, as our eyes cannot view electrons. Figure 31 [53] shows a side-by-side comparison between light and Transmission Electron Microscopes (TEM).

There exist two types of electron microscopes [53]. The first is TEM, and the second is the Scanning Electron Microscope (SEM). In TEM, the beam of light is passed through a thin specimen film showing the internal structures in detail. However, the drawback is that it requires the specimen to be very thin, which could allow electrons to pass through. In SEM, the beam of light is swept across the specimen, recording the bounced electron. This technique scans the surfaces of the specimen. It is faster than TEMs. However, it does not show the internal structures in detail. Figure 32 [53] shows the comparison of the images generated. The SEM on the left shows the surface of the specimen, showing multiple bacteria. The TEM on the right shows the internal structure of a single bacteria.

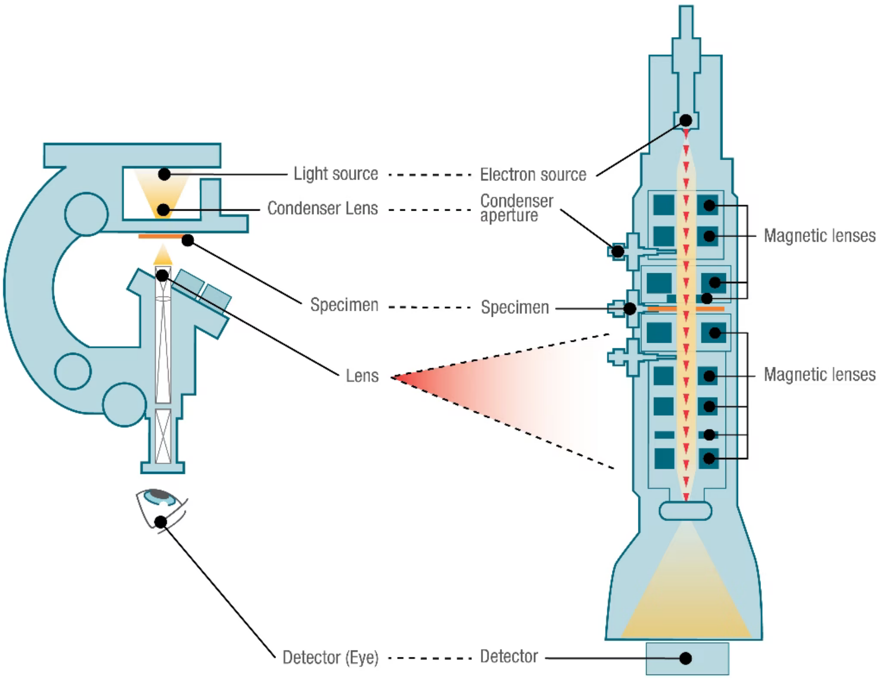


Figure 31. Side-by-side comparison between light microscope and TEM [53]

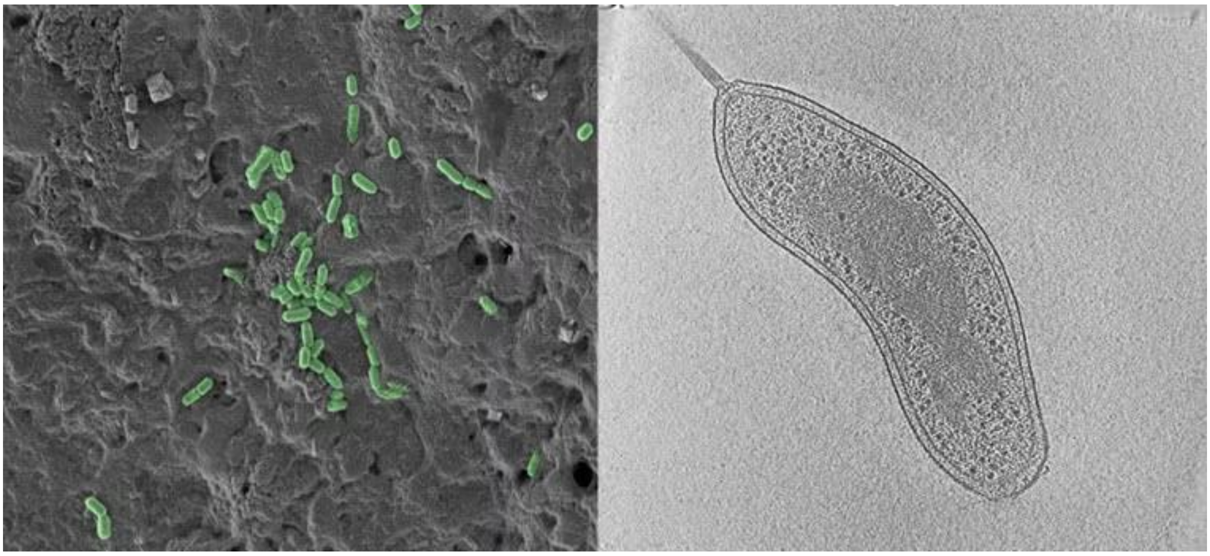


Figure 32. Side by side comparison of SEM and TEM images [52]

Appendix B. Domain Analysis

Domain

Before describing one of the new complex instruments used by Thermo Fisher Scientific customers, we will explain certain technologies and tools used by these instruments. These include technologies like virtualization, containerization, and other tools like Ansible, terminal, Grafana, Kibana and Kubernetes.

Virtualization

Virtualization is a technology that enables a single physical computer to function as multiple virtual computers known as Virtual Machines (VMs) by creating virtualized versions of computer resources such as processors, memory, storage, and networking components. These VMs operate independently, each running its operating system and applications [54].

Containerization

Containerization is another technology that allows software applications and their dependencies to be packaged and run in isolation. This packaging is called a container, and it is lightweight and portable. The container can run consistently across different computing environments. One such tool is Docker, which facilitates this technology for applications [24].

Ansible

Ansible is an open-source automation tool to simplify the management and orchestration of complex tasks across multiple servers or devices. It allows the automation of tasks related to configuration management, application deployment, cloud provisioning, and more. Ansible uses scripts known as playbooks to install modules and tools. The playbooks use a configuration file containing the information for VMs, storage and other resources [55].

Terminal

A *terminal* is a text-based interface that allows users to interact with a computer's operating system using text commands. It supports the execution of bash scripts, which are plain text files containing a series of commands written in the Bash [16] scripting language. These scripts are executed by the Bash interpreter, allowing users to automate tasks, perform complex operations, and streamline repetitive processes [3].

Grafana

Grafana is an open-source analytics and visualization platform that allows users to create, explore, and share interactive dashboards for monitoring and analyzing data from various sources, such as databases, cloud services, and applications. It also helps show alerts for different resources to see if something has gone wrong [5].

Kubernetes

Kubernetes, often abbreviated as K8s, is an open-source container orchestration platform for automating containerized applications' deployment, scaling, and management. It allows users to efficiently manage clusters of containers, ensuring that applications run reliably and scale seamlessly across various environments. It contains a web-based user interface that provides a visual representation and control panel for managing Kubernetes Clusters. It allows users to monitor the health and performance of their cluster, deploy and manage applications, inspect resources, and troubleshoot issues. The dashboard offers a convenient way to interact with and manage the various components of a Kubernetes cluster, making it more accessible to users who prefer a graphical interface over command-line interactions [7].

Kibana

Kibana is an open-source data visualization and exploration platform designed for Elasticsearch. It provides powerful search and visualization capabilities, allowing users to analyze and interact with their data stored in Elasticsearch indices. Kibana is often used for log and event data analysis, business intelligence, and other data-driven applications [6].

SDP Overview

Data Management Platform (DMP) is a new complex instrument. It is a PC platform that supports using the TEM. The DMP is shown in Figure 33. It contains a software hosting infrastructure called the Software Delivery Platform (SDP) that supports microscope applications. It also supports applications for managing software installation. These applications include Software Configurator Application (SCA) and SDP Web-installer Application Program (SWAP). The SDP also contains service applications that include Grafana, Kibana, and Kubernetes Dashboard. Table 15 shows the elements of DMP and their purpose.

SDP utilizes virtualization technology to manage virtual machines that operate independently. Within SDP, applications use containerization through docker to run multiple applications independently. SCA helps to generate a configuration that helps to define SDP. This configuration lists devices and their properties Ansible uses to install or upgrade SDP. The Research and Development Engineers (RNDEs) design and develop SDP and its applications.

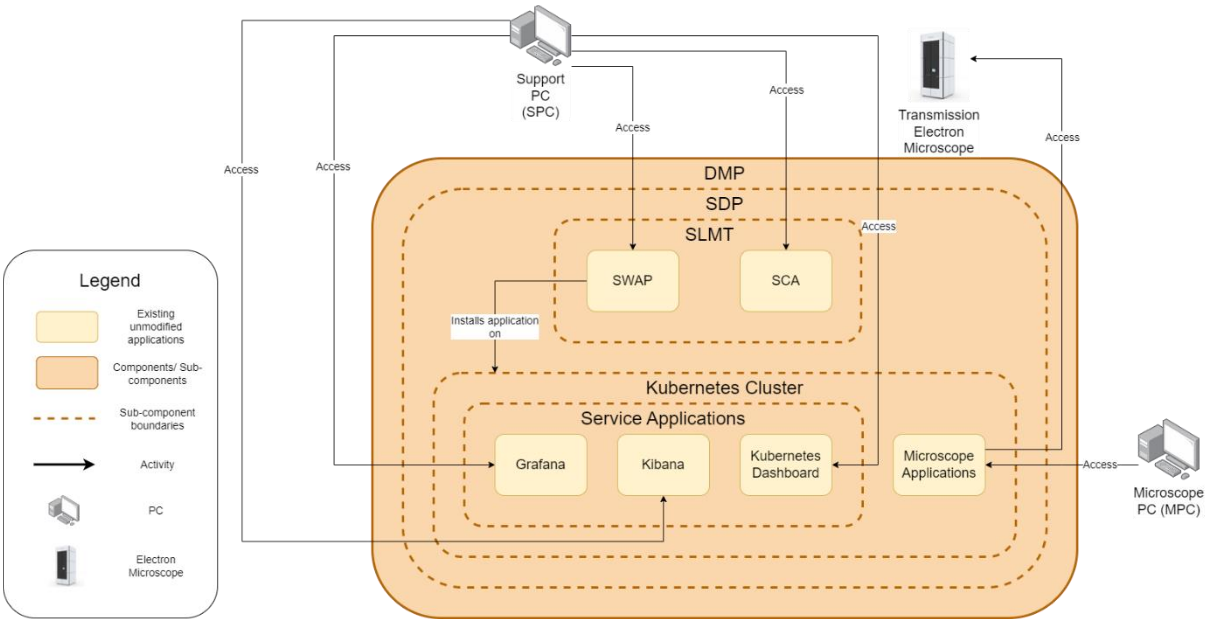


Figure 33. Diagram showing the internal components of SDP

Table 15. Internal components of SDP and their purpose

Component	Purpose
DMP	PC platform that connects TEM, MPC, SPC, microscope applications, and service applications
SDP	Software hosting infrastructure within DMP that hosts applications
Software Lifecycle Management Tools (SLMT)	SLMT used for application and infrastructure installation and upgrades
SPC	Used by DSEs to access SDP, SCA, SWAP and service applications
MPC	Used by researchers to access microscope applications
Kubernetes Cluster	A collection of VMs that help to run and manage applications
Microscope Applications	A set of applications for the microscope
Service Applications	A set of applications that help to troubleshoot SDP and applications running on the Kubernetes Cluster
Grafana	An application used for monitoring and alerts
Kibana	An application used for diagnostics and log trace
Kubernetes Dashboard	An application used for identifying application health
SCA	Web-based GUI application that helps to define an SDP specification for resource allocation
SWAP	Web-based GUI application that helps to install applications on the Kubernetes Cluster
Transmission Electron Microscope (TEM)	Transmission Electron Microscope used by Microscope Applications

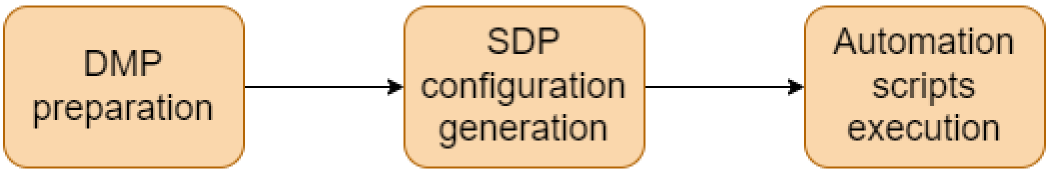


Figure 34. Steps showing the installation process of SDP

SDP Installation

SDP installation involves a three-step process, as shown in Figure 34. These steps help to prepare DMP, generate SDP configuration, and execute Automation Scripts. These scripts are a combination of bash scripts and ansible playbooks. The steps are as follows:

- The first step involves an automated process of setting up the entry VM with modules and packages necessary for SDP installation with DMP. The entry VM comes installed with DMP. It is straightforward, requiring no input from a DSE for the execution. To invoke the process, a DSE must upload executable files into the system and execute through the VM terminal. An operating system is the software that manages hardware resources and provides services for DMP to run effectively.

- Once the VM is ready, the SDP Configuration is generated using SCA. Since SCA is a web-based GUI application, it can be accessed through a web browser like Firefox. The DSEs then must fill out forms that generate a configuration from the information. This configuration generation is known as a Complete Action where the SDP Configuration is generated for the first time.

- Once the configuration is generated, the Complete Action can be applied by executing automation scripts through the operating system terminal. During the process, a DSE is asked several inputs to configure the SDP correctly. Once the execution is completed, an automation audit is generated that can be viewed by DSEs to check if the SDP was installed correctly.

SDP Troubleshooting & Application Installation

To troubleshoot SDP, the DSEs use a few service applications. These include Grafana, Kibana, and Kubernetes. Grafana helps monitor SDP and notify DSEs through alerts if there are issues with SDP resources. Kibana is used for diagnostics and log trace, where logs collected from different applications can be sliced and diced to show readable information for troubleshooting. Kubernetes checks for application health installed on the Kubernetes cluster. To install other applications on the Kubernetes cluster, DSEs use SWAP. These applications include both microscope and service applications.

Appendix C. NFR Reference List

Below is a standard list of product qualities used to elicit the non-functional requirements for the project. The standard list is from ISO 25010:2011 Section 4.2 product quality model. The list is shown in Table 16. The list was used during interviews to ask questions related to non-functional requirements.

Table 16. List of nonfunctional requirements derived from ISO 25010:2011

Product Qualities	Sub Characteristics
Functional Suitability	Functional completeness
	Functional correctness
	Functional appropriateness
Performance Efficiency	Time behavior
	Resource utilization
	Capacity
Compatibility	Co-existence
	Interoperability
Usability	Appropriateness recognizability
	Learnability
	Operability
	User error protection
	User interface aesthetics
	Accessibility
Reliability	Maturity
	Availability
	Fault tolerance
	Recoverability
Security	Confidentiality
	Integrity
	Non-repudiation
	Accountability
	Authenticity
Maintainability	Modularity
	Reusability
	Analyzability
	Modifiability
	Testability
Portability	Adaptability
	Installability
	Replaceability

Appendix D. Tools & Technologies

Table 17. List of tools and technologies used for the project

Tools & Technologies	Type	Purpose
Socketio	Implementation	The client uses this library to connect to the web server. It uses Web Socket for communication and uses HTTP long-polling as a fallback. This library is part of the presentation component.
flask-socketio	Implementation	This library connects with the socketio library for communication. It is an extension of the Flask framework. This library is part of the Web Socket component in the web server.
subprocess	Implementation	The library used to run a script in the SDP. The action module uses this library to execute an automation script.
gunicorn	Implementation	This library is a WSGI server used to run the web server in a production environment.
gevent-websocket	Implementation	This library is used with gunicorn to support the asynchronous nature of Web Socket communication. When present, the socketio and flask-socketio libraries upgrade to a WebSocket connection.
xterm.js	Implementation	This library implements the output panel that shows the automation script output and the audit panel to view audit contents on UI. It replicates a terminal software used by the SEs.
bootstrap	Implementation	This library implements the progress bar, which is already present in the existing SCA.
JSON	Implementation	This file format stores action and automation script mapping. The operation module can obtain the appropriate automation script based on the action.
Iframe	Implementation	This HTML template element embeds external content, such as other web pages or videos, within a web page.
HTTP client	Implementation	This built-in Python module provides low-level access to the HTTP protocol, allowing you to create and send HTTP requests directly. It is part of the standard library that

Tools & Technologies	Type	Purpose
		can make HTTP requests, handle responses, and work with HTTP headers.
Grafana alert API	Implementation	This API manages alerts programmatically. It enables standard HTTP methods (GET, POST, PUT, DELETE) to list, create, update, and delete alerts. Authentication is required to access the alerts.
re	Implementation	This library helps to parse the audit name and creation date from the audit files.
Font awesome	Implementation	This font library provides icons for action success and failure.
Make file	Verification	This automation script uses the Make automation tool to build and test the workbench application in the local environment.
Pre-commit hooks	Verification	This automation script triggers when code is committed to the GitLab repository. It runs Pylint to check if Python code meets the PEP8 coding standard.
GitLab CI	Verification	This Continuous Integration (CI) feature of GitLab helps to set up a pipeline to build, test, and deploy the application to the production environment.
Cypress	Verification	This end-to-end testing framework tests UI interactions with a web browser.
Cucumber	Verification	This BDD tool describes test specifications for end-to-end tests. It integrates with Cypress to create automated tests that are easy to understand.
Pylint	Verification	This is a static code analyzer that checks if the Python code meets the PEP8 coding standard.
Pytest	Verification	This unit-testing framework tests Python modules and components.
GitLab	Configuration Management	This repository stores the solution's source code and runs the CI/CD pipeline.
Jira	Project Management	Jira is an issue-tracking software for managing sprints, epics, user stories, and tasks.
Confluence	Project Management	Confluence is a document wiki for the project.
MS teams	Stakeholder Communication	MS teams is a video conferencing tool for communication with all stakeholders.

Tools & Technologies	Type	Purpose
Excel	Document Management	Excel is a documenting tool for project management-related tables and graphs for TU/e Supervisor.
Word	Document Management	Word is a documenting and reporting tool for sharing meeting minutes with the TU/e Supervisor.
Email	Stakeholder Communication	Email is a communication tool used to communicate with all stakeholders.
One drive	Document Management	One drive is a document repository that contains all project-related documents.

Appendix E. Decisions

Table 18. List of decisions made during the project

ID	Description	Outcome	Number of Alternatives	Number of criteria
D1	To decide if SCI will be part of SCA or a separate application	We decided to go with a single application and use the existing stack. SCA and SCI will then need to be integrated.	2	1
D2	To find a suitable real-time technology for communication between the SCI and the web browser	We decided to go with Web Sockets as I was more aware of how to use them rather than polling. Moreover, its functionality was more commonly used in examples available online. It also replicated terminal behavior. This was further discussed and decided during Egbert's system design whiteboarding session.	4	11
D3	To decide the design pattern for SCI for applying a single action at a time	We decided to use singleton pattern over python module to achieve class base approach, modularity and apply single action at a time.	2	1
D4	To decide to change bash scripts to python or ansible or keep using bash scripts.	We decided to go with bash scripts as it is a single-entry point, and it can be decoupled from SCI and workbench through a JSON file.	3	2
D5	To decide the appropriate library for executing a shell command on a remote host and retrieve, line-by-line, and verbose output	We decided to use subprocess library as paramiko [56] can only run with gunicorn sync worker. subprocess is also compatible with gunicorn async gevent-websocket worker.	3	9
D6	To decide an existing library that can show colored and verbose output of terminal	We decided to use xterm.js as it is used more than terminal.js and supports ANSI [57] terminal color format.	2	2

ID	Description	Outcome	Number of Alternatives	Number of criteria
D7	To decide a specific framework for workbench application	We decided to add workbench features to SCA application and therefore a new framework was not required. It will be easier for SEs to use a single app.	3	11
D8	To decide separate app or single app (combined with SCA) for workbench	We decided to add workbench features to SCA application as it will be easy for SEs to access the application and RNDEs to maintain the application.	3	2
D9	To decide the suitable way to capture exit status of actions applied from action history	We agreed to print exit code as last line of automation audits. The history in workbench can then capture the exit status from the audit content and display on the browser as success or failure.	6	3
D10	To decide a suitable approach to apply actions in unattended mode	We agreed to use defaults and pass them as parameters to scripts. Workbench will use file with values temporary till the scripts have been changed.	4	3
D11	To decide the integration strategy of SCI and SCA	We used JSON file strategy to store next and previous values of form flows.	3	2
D12	To decide the internal state management of SCI	We decided to use a Boolean value rather than a state design pattern to denote if the SCI server is applying an action automation or is idle.	2	1
D13	To decide whether to use a single action class rather than inheritance relationship for mapping all actions	We decided to use a single script executor that will be unaware of the automation scripts. The action information will be decoupled in the JSON file which can be modified and extended by RNDEs independently.	2	1
D14	To decide if the maintenance applications will be	We decided to add Grafana, Kibana and Kubernetes as redirect links.	2	1

ID	Description	Outcome	Number of Alternatives	Number of criteria
	redirecting links through the workbench.			
D15	To decide if SWAP will be embedded within workbench rather than a redirect link	We decided to use Iframe to access SWAP from within workbench rather than redirecting SEs to the application.	2	1
D16	To decide the library to fetch Grafana alerts for the traffic light	We decided to use Grafana alert API to fetch Grafana alerts and filter based on severity levels.	3	2
D17	To decide the library to show a progress bar in the browser	We decided to use bootstrap library progress bar as bootstrap library is already imported in the existing SCA code.	2	2
D18	To decide if a new account should be used for SEs	We decided to use a new read only account for SEs.	2	1
D19	To decide the strategy for testing mock automation scripts during end-to-end testing in local and testing environment	We decided to use a separate mock automation scripts docker container for end-to-end testing.	2	2
D20	To decide the appropriate frontend and backend library for Web Sockets	We decided to use socketio (frontend) and flask-socketio (backend) as it supports advanced features such as broadcasting to multiple clients, supports fallback mechanism to long polling if the production environment does not support Web Socket connection. It also supports reconnection if browser is closed, or connection is interrupted.	4 (Templates) 4 (Webserver)	2 (Templates) 2 (Webserver)
D21	To decide a custom solution for applying actions or using an existing open-source solution	We decided to create a custom solution as the ansible-semaphore [58] open-source solution uses ansible scripts for automation rather than bash scripts. Bash scripts act as single-entry point to the automation process. The UI of the ansible-	2	2

ID	Description	Outcome	Number of Alternatives	Number of criteria
		semaphore tool is also difficult to use for SEs.		
D22	To decide a library to filter the action history table	We decided to use a custom solution available online.	3	2
D23	To use appropriate worker class library for Gunicorn WSGI production server	We decided to use gevent-websocket library to support flask-socketio asynchronous behavior.	6	2

Table 19. Decisions associated to different modules

Module	Decision ID
Actions	D2, D3, D4, D5, D6, D10, D12, D13, D17, D20, D21, D23
History	D9, D22
Managing Applications	D14, D15, D16, D18
Form Flows	D1, D7, D8, D11
End-to-end Testing	D19

Table 20. Stakeholders consulted for different decisions

Stakeholders	Role	Decisions Consulted
Egbert Algra	Company Supervisor, Staff Architect (RNDE)	All decisions
Giovanni de Almeida Calheiros	Software Design Engineer III (SWAP, SDP Infrastructure) (RNDE)	All decisions
Tor Halsan	Software Design Engineer IV (SDP Infrastructure) (RNDE)	D2, D4
Respa A. Putra	Software Design Engineer II (SCA) (RNDE)	All decisions
Chen Gang	DevOps Engineer (RNDE)	D10
Harold Weffers	TU/e Supervisor	D5, D7, D8
Jordy Plug	DSE	D10, D14, D15, D17, D22
Tarkan Akcay	DSE	D10, D14, D15, D17, D22
Kieran Ham	GTS	D17
Ataur Rahman	OE	D14, D15, D17, D22

Table 21. An example of a decision taken to choose a suitable real-time technology

Status	**Final**
Impact	**High**
Informed	**Egbert, Respa, Giovanni, Tor**
Last Updated	**18/08/2023**
Purpose	To find a suitable real-time technology for communication between the SCI and the web browser

Criteria	WebSocket (WS)	Long Polling (HTTP)	AJAX Short Polling (HTTP)	Server-Sent Events (HTTP)
Real-time communication	Yes, it communicates in real-time by creating a single connection.	Yes, it does it partially with multiple requests and responses. It makes a request, and the server holds the request till it has data, keeping the connection open long enough for the client to receive the response.	No, it does it partially with multiple requests and responses. It makes a request to the server every fixed interval.	Yes, only from the server side. To initiate this, the client will need to send a request to the server. After this the server will send the client with data continuously.
Existing support	No existing support.	Yes, there is existing support and therefore it can be managed by RNDEs.	Yes, there is existing support and therefore it can be managed by RNDEs.	No existing support.
Maturity & browser support	Yes, it is highly mature and supported by all modern browsers. (Edge, chrome, Firefox, opera, safari)	Yes, it is highly mature and supported by most modern and old browsers.	Yes, it is highly mature and supported by most modern and old browsers.	Yes, it is highly mature and supported by all modern browsers. (Edge, chrome, Firefox, opera, safari)
Familiarity	Experienced in certain web socket libraries.	No experience so far but willing to learn.	No experience so far but willing to learn.	No experience so far but willing to learn.

Criteria	Web-Socket (WS)	Long Polling (HTTP)	AJAX Short Polling (HTTP)	Server-Sent Events (HTTP)
Auto-reconnect	⊖ Depends on the library used for implementation.	⊖ Must be implemented manually.	⊖ Must be implemented manually.	⊕ Supports automatic tracking of last seen message and auto reconnect.
Fallback mechanism	⊖ Needed if not supported natively by old browsers and production environment.	⊕ Natively supported by old browsers.	⊕ Natively supported by old browsers.	⊖ Needed if not supported natively by old browsers and production environment.
Bi-directional communication	⊕ Yes, it is bi-directional	⊖ No, it is unidirectional	⊖ No, it is unidirectional	⊖ No, it is unidirectional
Message ordering	⊕ Ordered sequence since the underlying protocol is TCP which presents bytes in order as they are sent.	⊖ Can be an issue since multiple HTTP requests from the same client can be in transmission simultaneously. Some can arrive before others. They can also be duplicate if two or more browser tabs are open. It could also happen if there are more than one or more connection at a time.	⊖ Can be an issue since multiple HTTP requests from the same client can be in transmission simultaneously. Some can arrive before others.	⊕ Ordered sequence since it transfers data though a single long-lived HTTP connection.
Latency	⊕ Low latency as the data is sent immediately once available.	⊕ Low latency as the response is only sent when the data is available. Till then, the request is held by the server.	⊖ High latency due to message delays. Some responses could be empty. Since the request is periodic, any data that comes in between the HTTP requests will have to wait for the next request. This causes delays.	⊕ Low latency as the data is sent immediately once available through a single, long-lived connection.

Criteria	Web-Socket (WS)	Long Polling (HTTP)	AJAX Short Polling (HTTP)	Server-Sent Events (HTTP)
Message overhead	+ Low message overhead as headers are not sent on each server request.	− High message overhead as each request carries the HTTP header.	− High message overhead as each request carries the HTTP header.	+ Low message overhead
Firewall compatible	− Can be blocked by firewall that perform application-level packet inspection.	+ Firewall compatible since the underlying protocol is HTTP.	+ Firewall compatible since the underlying protocol is HTTP.	+ Firewall compatible since the underlying protocol is HTTP.
Score	**3**	**2**	**-2**	**4**
Discussion	**Discussed with Giovanni, Tor and Respa:** Recommended to use the already existing libraries that are part of SCA.			
Discussion	**Discussed with Egbert:** Recommended to use either Web Sockets or polling (long and short).			
Outcome	Decided to go with Web Sockets as I was more aware of how to use them rather than polling. Moreover, it has more real time in nature. This was further discussed and decided during system design whiteboarding session with Egbert.			
Reference	High Performance Browser Networking by Oreilly chapter 14, https://codeburst.io/polling-vs-sse-vs-websocket-how-to-choose-the-right-one-1859e4e13bd9			

Table 22. Legend for the example decision

Icon	Value
+	+1
−	-1
	0 (zero)

Appendix F. Final UI

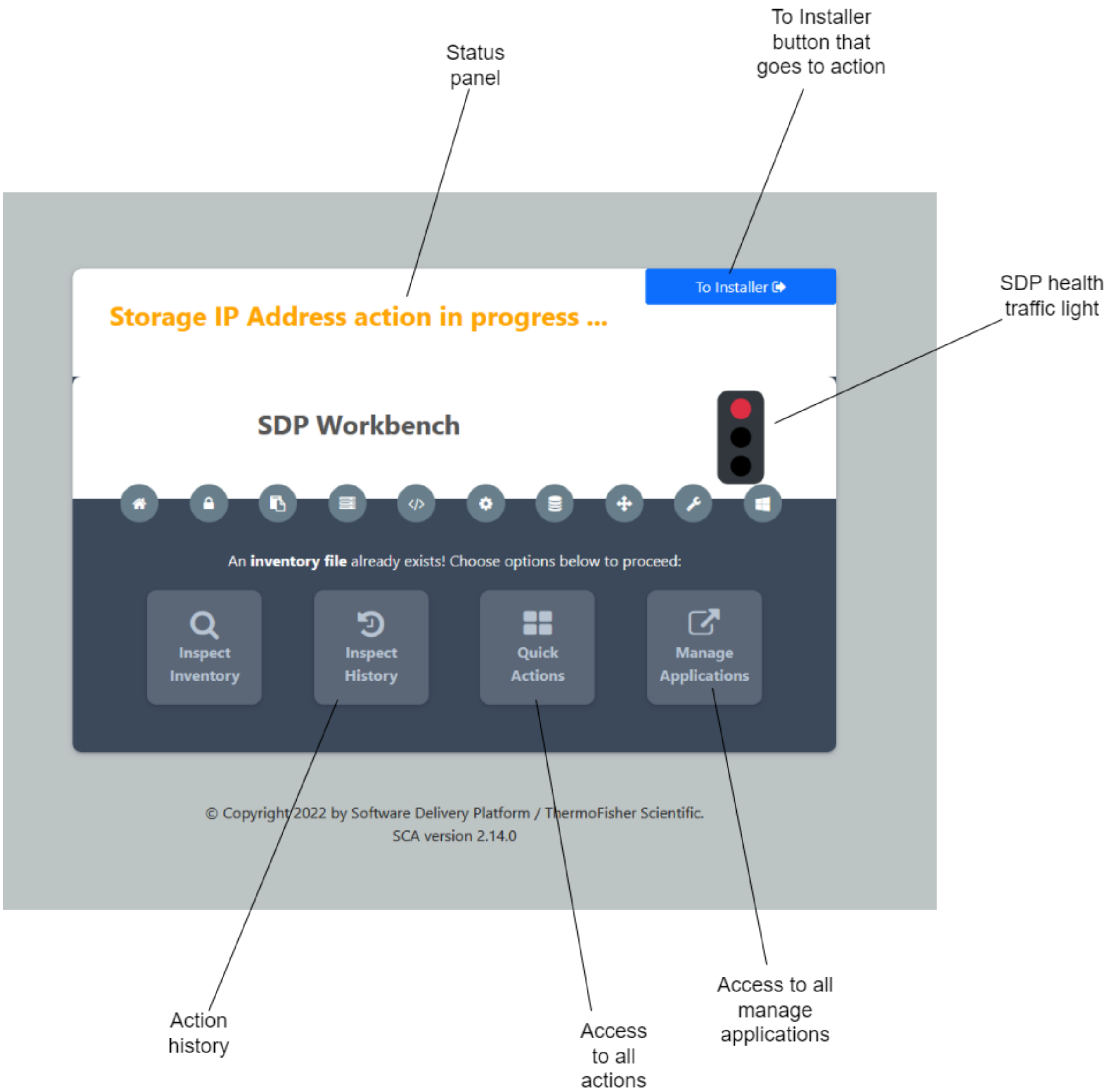


Figure 35. Final UI of the workbench homepage

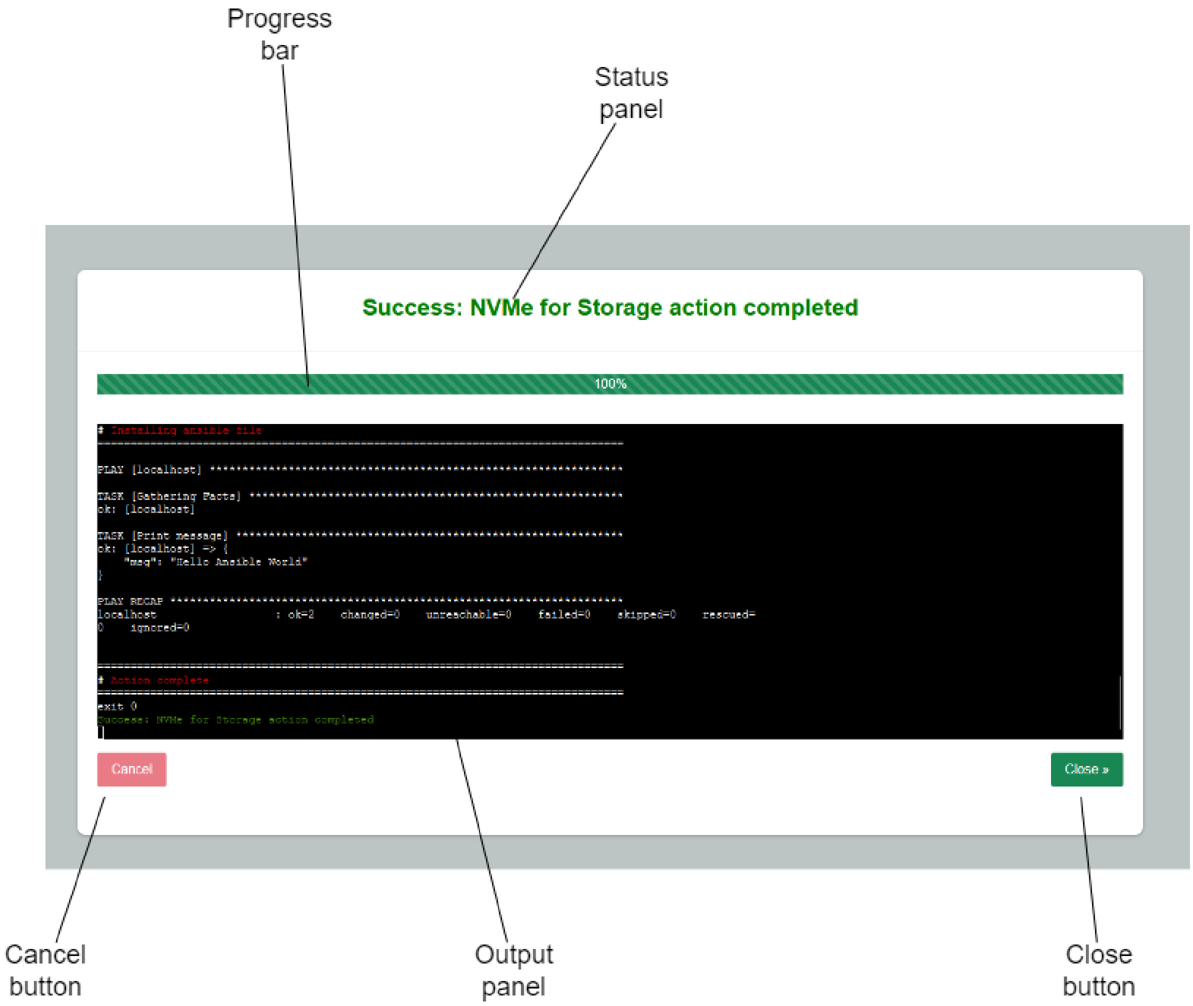


Figure 36. Final UI for viewing action output

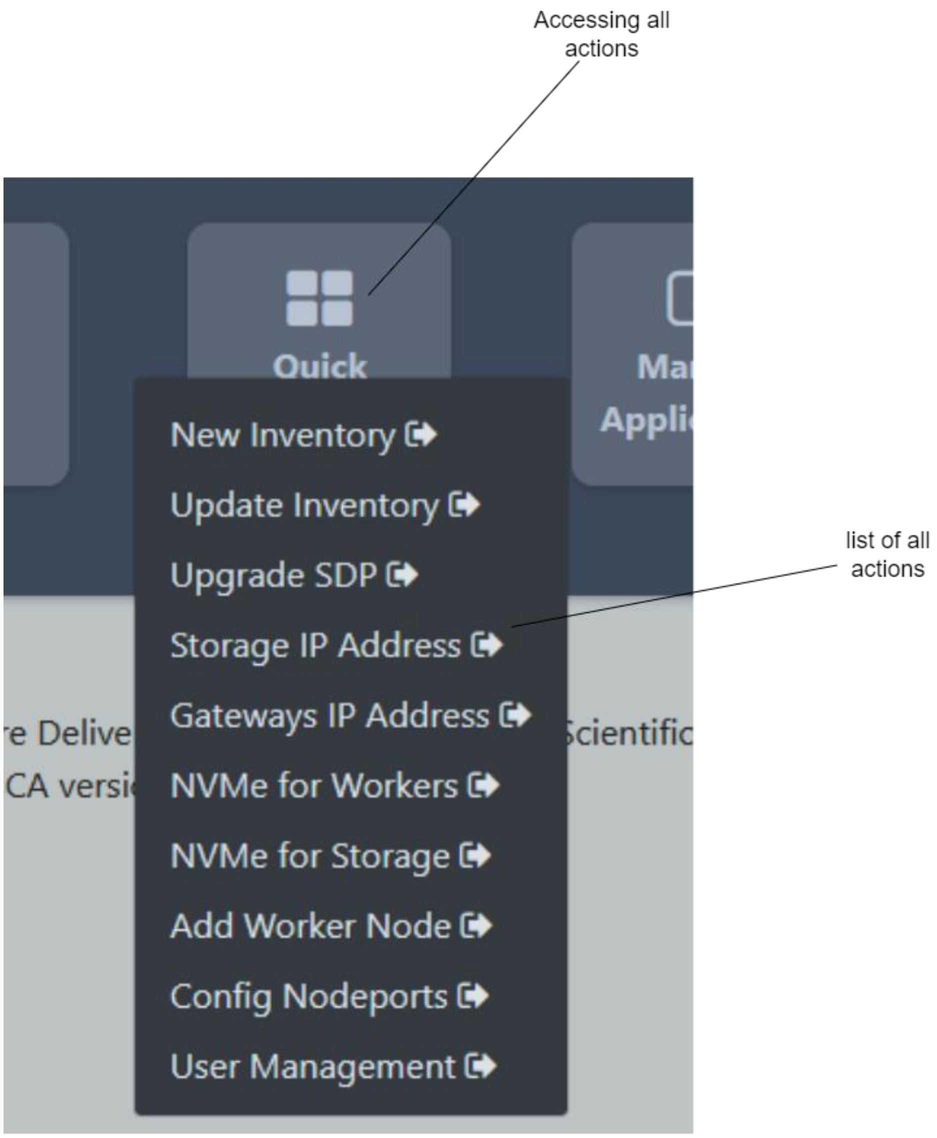


Figure 37. Final UI showing a list of all actions

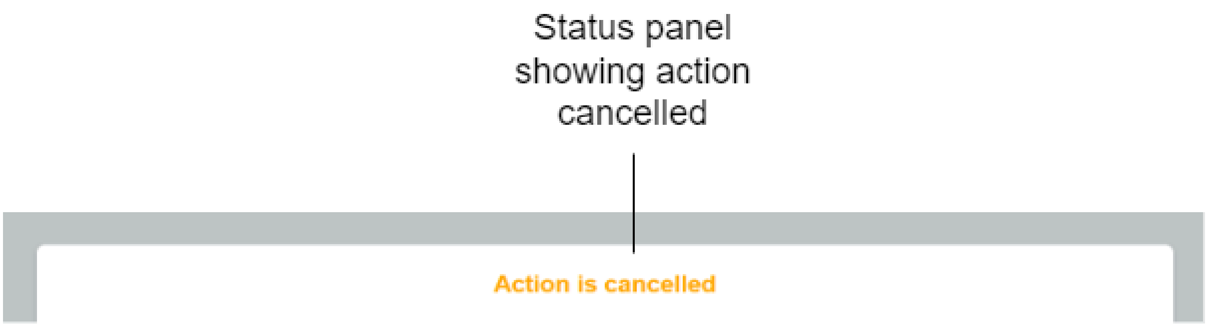


Figure 38. Final UI showing action cancelled on the action page

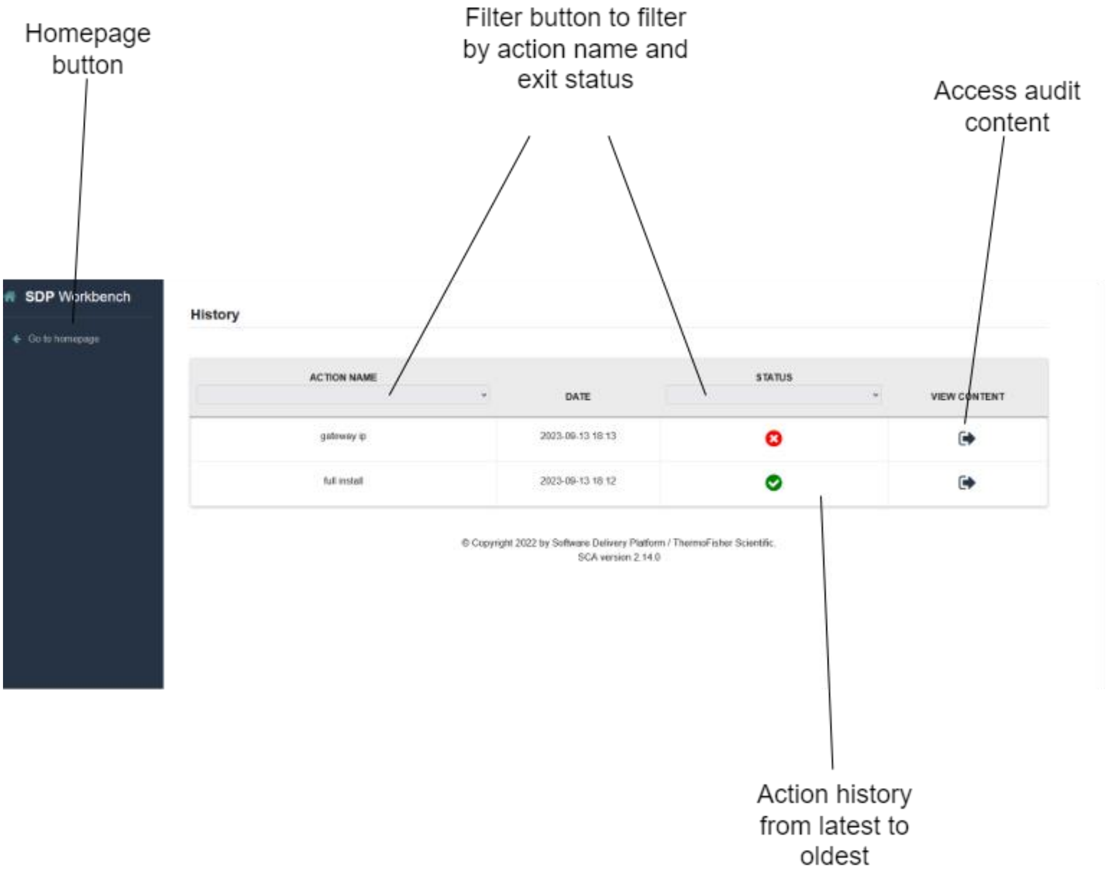


Figure 39. Final UI of action history

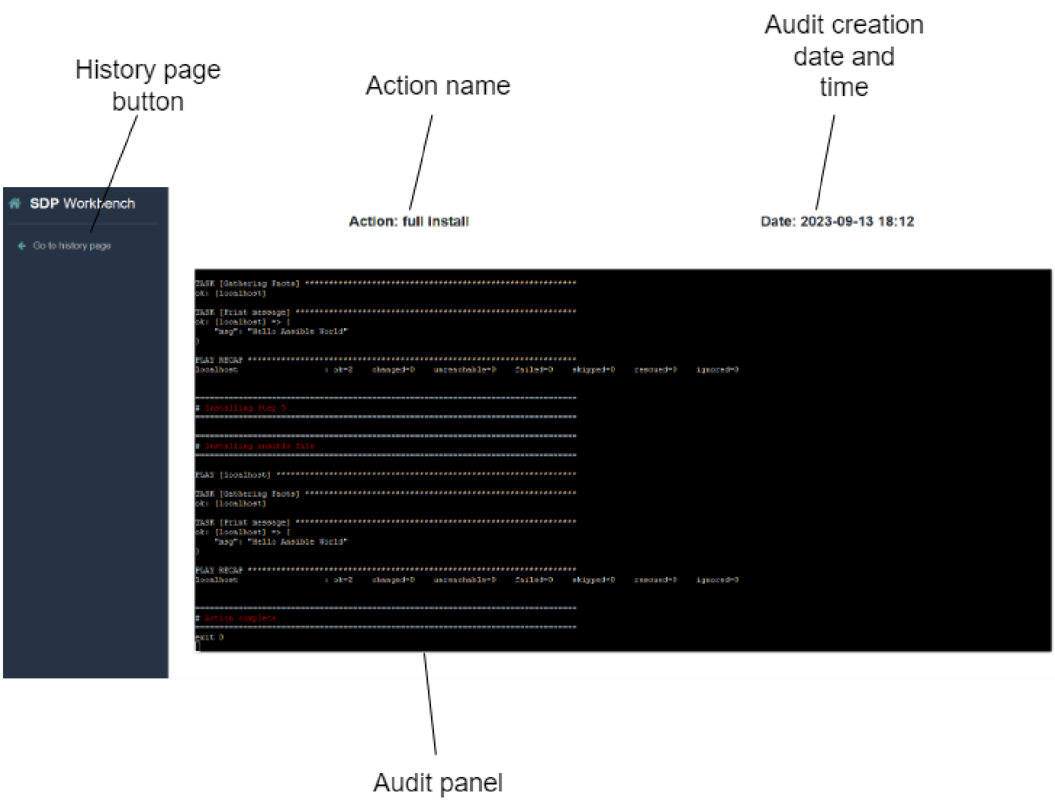


Figure 40. Final UI of viewing an audit content for an action

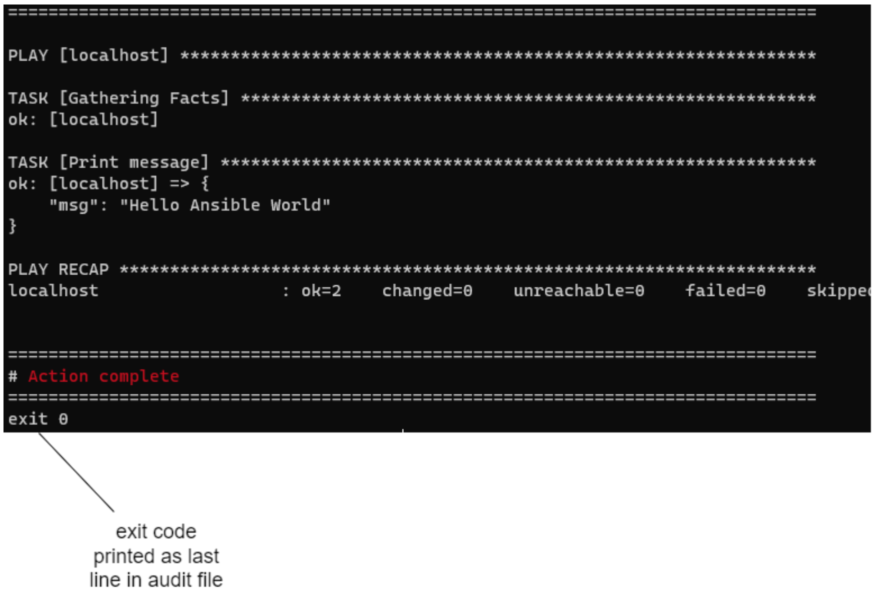


Figure 41. Final UI showing the list line of audit

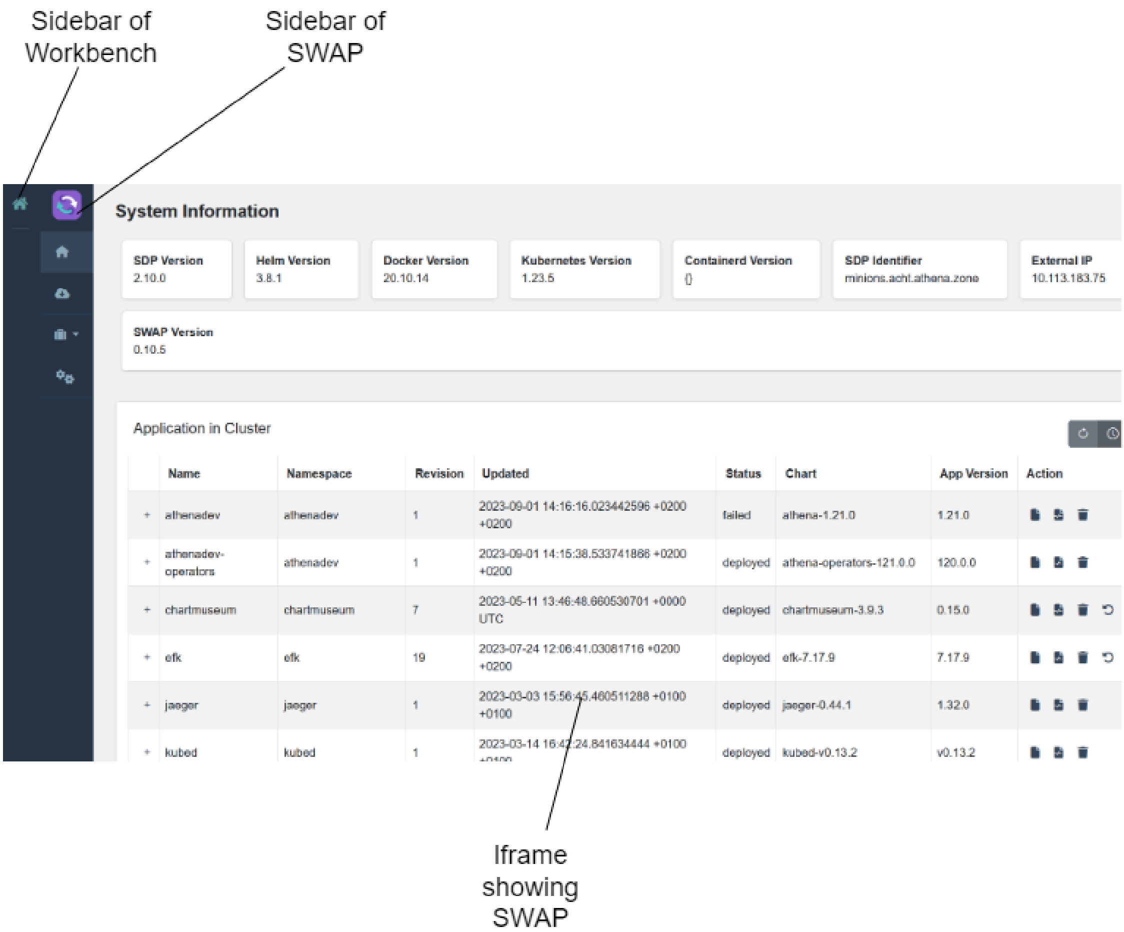


Figure 42. Final UI showing SWAP within Iframe

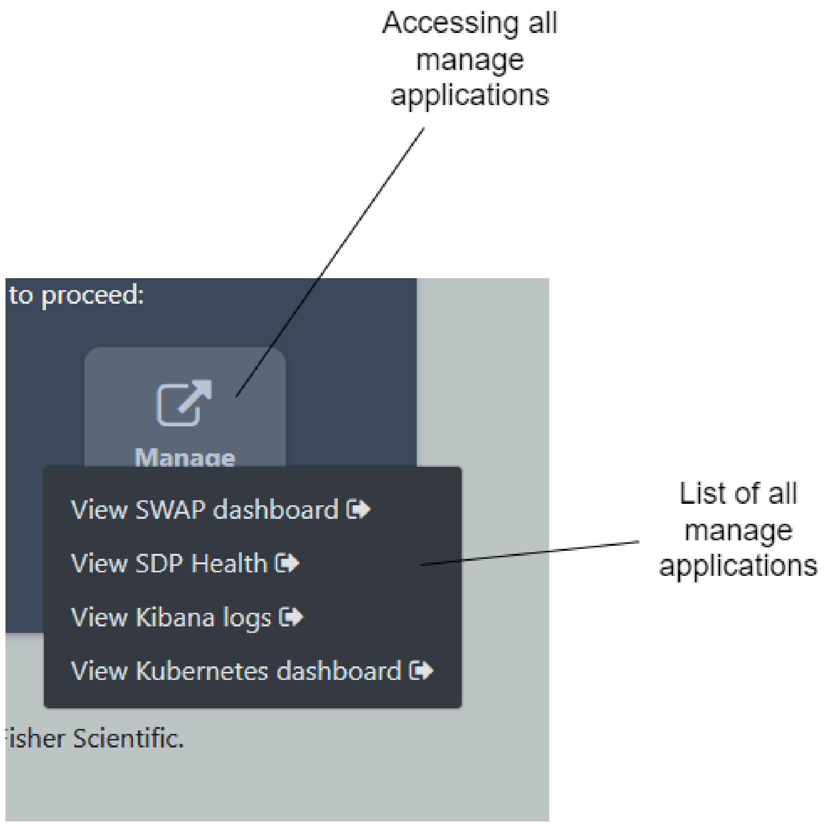


Figure 43. Final UI showing the list of Manage Applications

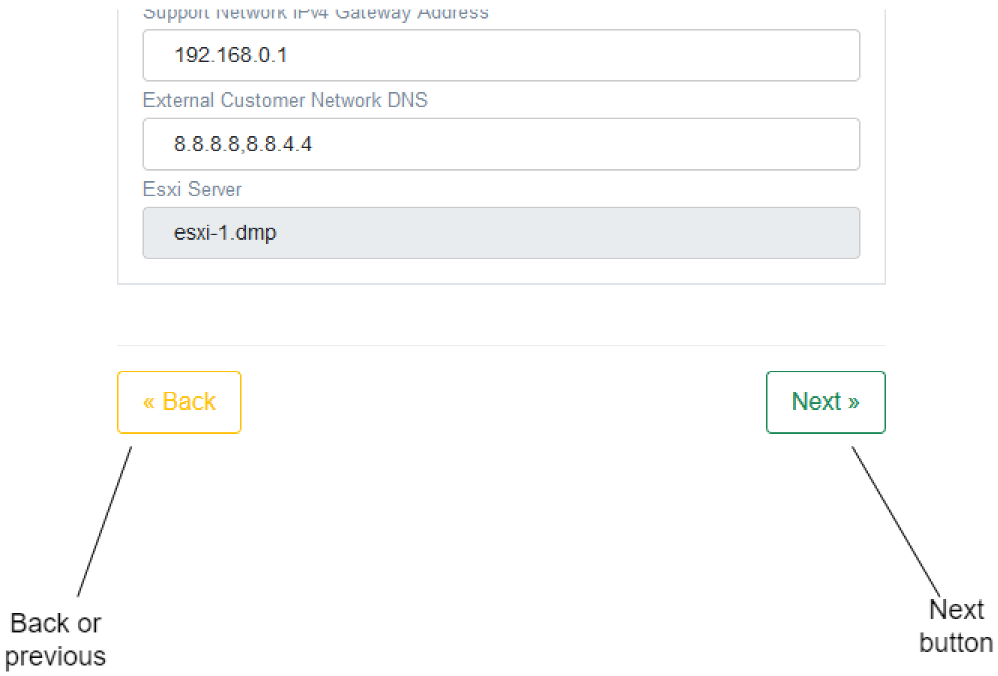


Figure 44. Final UI showing the next and previous buttons of SCA

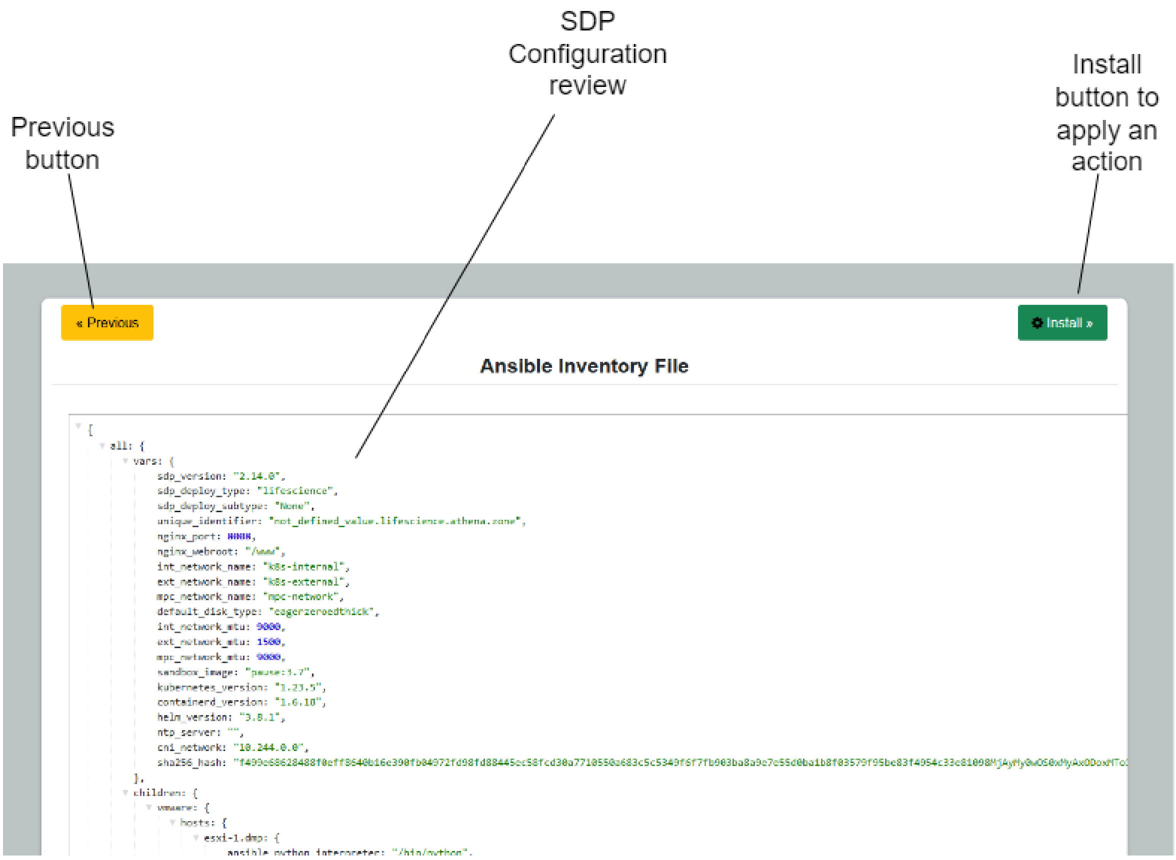


Figure 45. Final UI showing the pre-install web page and its elements

Appendix G. Solution Architecture

Overview

We documented our architecture using 4 + 1 view model. It was implemented using the UML [59] notation This model contains the following views and the associated diagrams[3]:

- Development View – This view shows the implementation view of Workbench. It is shown through the package diagram as shown in Figure 46. We chose the package diagram to show the layered architecture and modularity of the implementation and it becomes easier to communicate with RNDEs to implementation changes.
- Physical View – This view shows the deployment of Workbench. We considered the deployment diagram to denote how Workbench operates in testing and production environment. This can be seen in Figure 47 and Figure 48.
- Process View – This view shows how the sub-modules/classes[4] interacts with one another. We used the sequence diagrams to show the behavior. This can be seen in Figure 49.
- Logical View – This view denotes the functionality of Workbench. We used class and state diagrams to describe the functionality. These can be seen in Figure 50 and Figure 51.
- Scenarios – This describes the Use Cases the SEs interact with Workbench. These can be seen in Figure 52.

[3] There may be some differences between the diagrams in the Solutions Chapter and Appendix G. Diagrams in Appendix G are more detailed.
[4] Even though the business logic was developed using class-based approach, we used sub-module to simplify explanation.

Development View

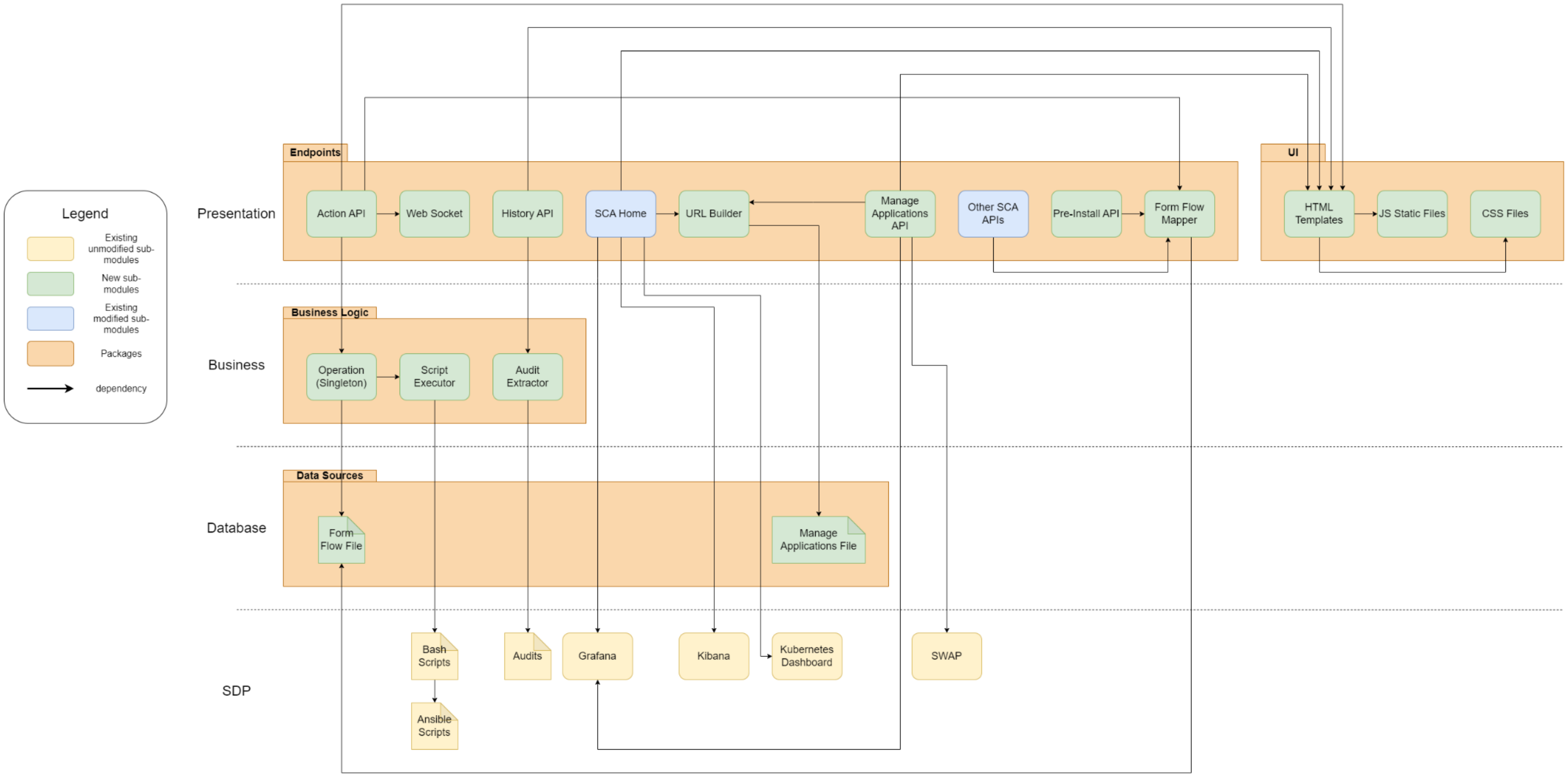


Figure 46. Diagram showing the development view of Workbench

Physical View Production

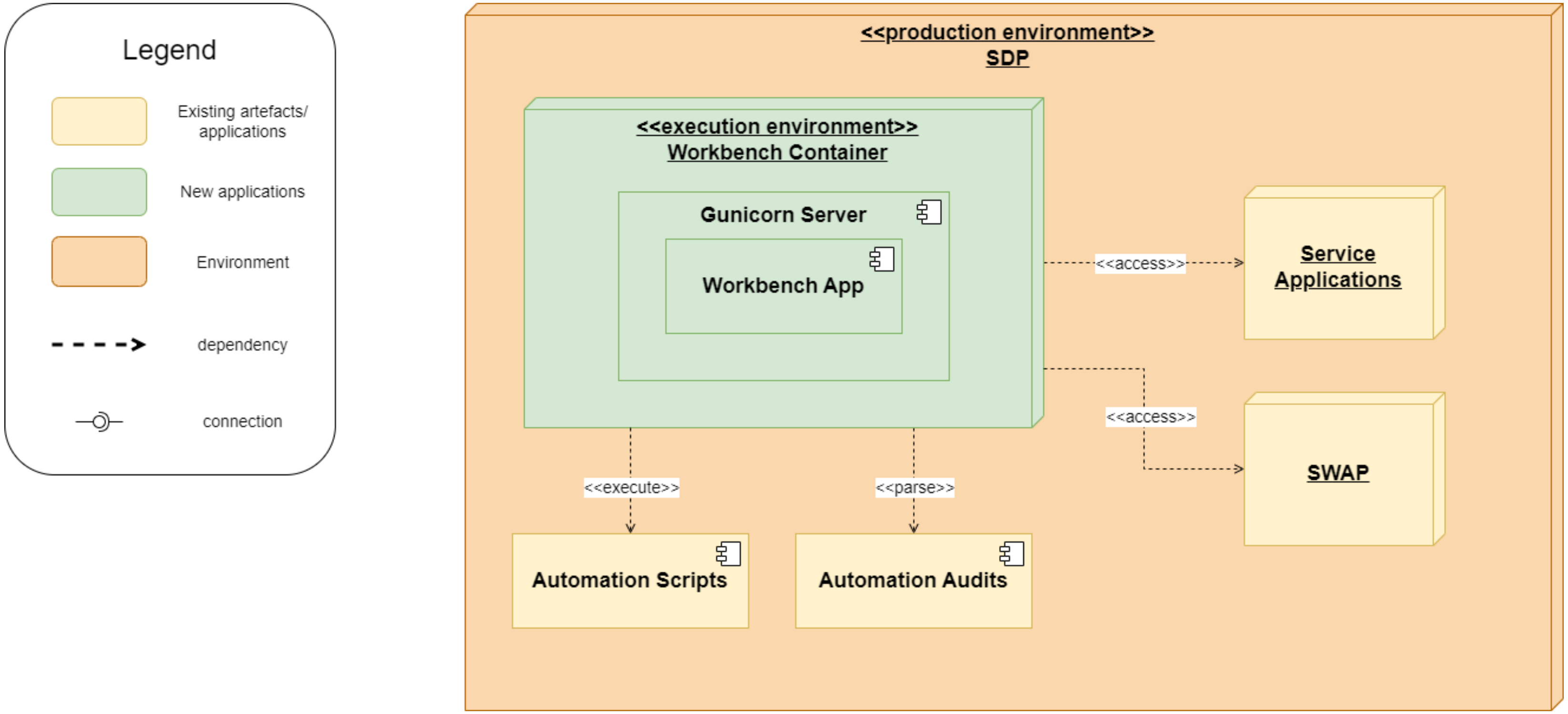


Figure 47. Diagram showing physical view for the production environment

Physical View Testing

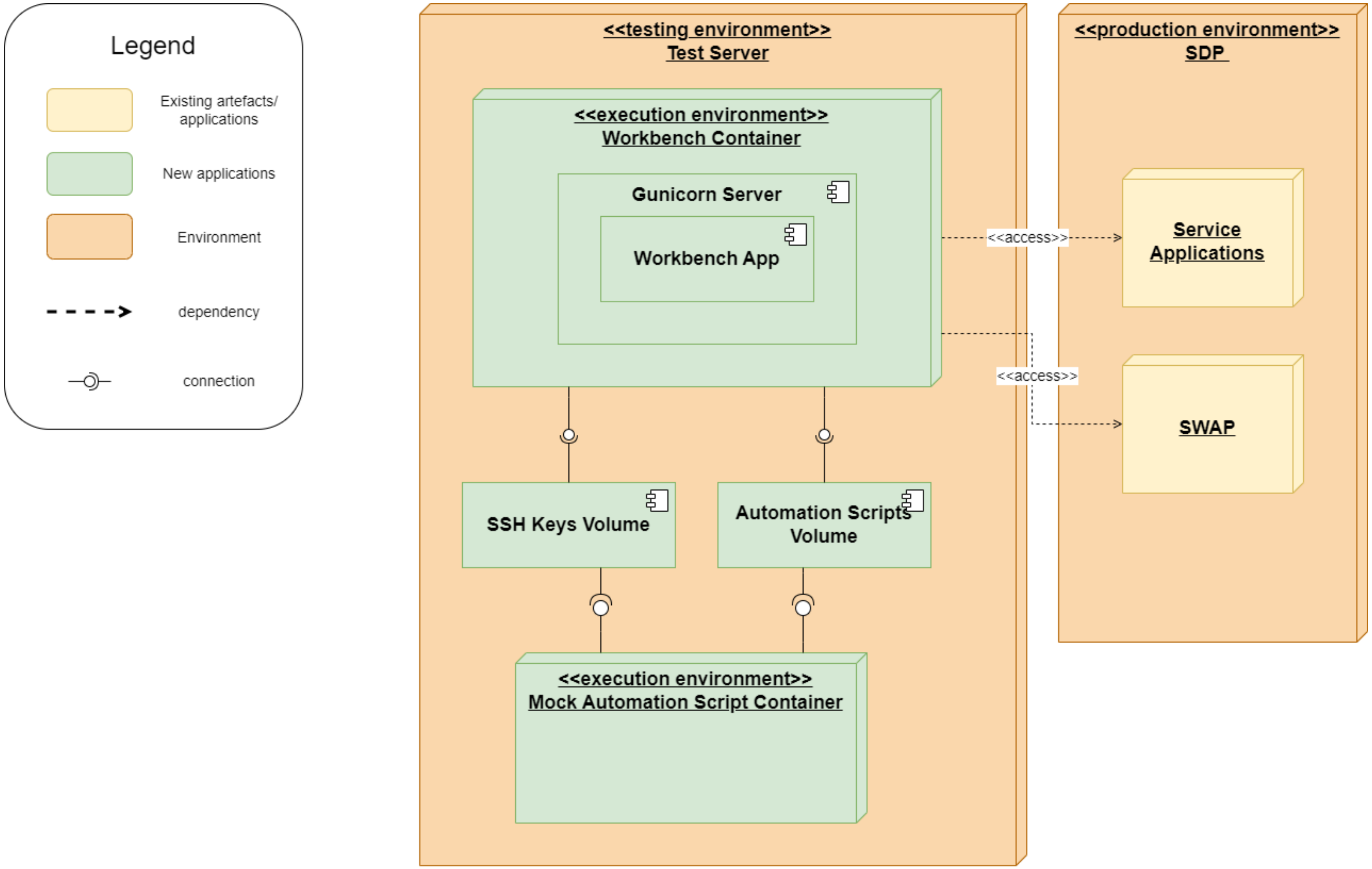


Figure 48. Diagram showing physical view for the testing environment

Process View

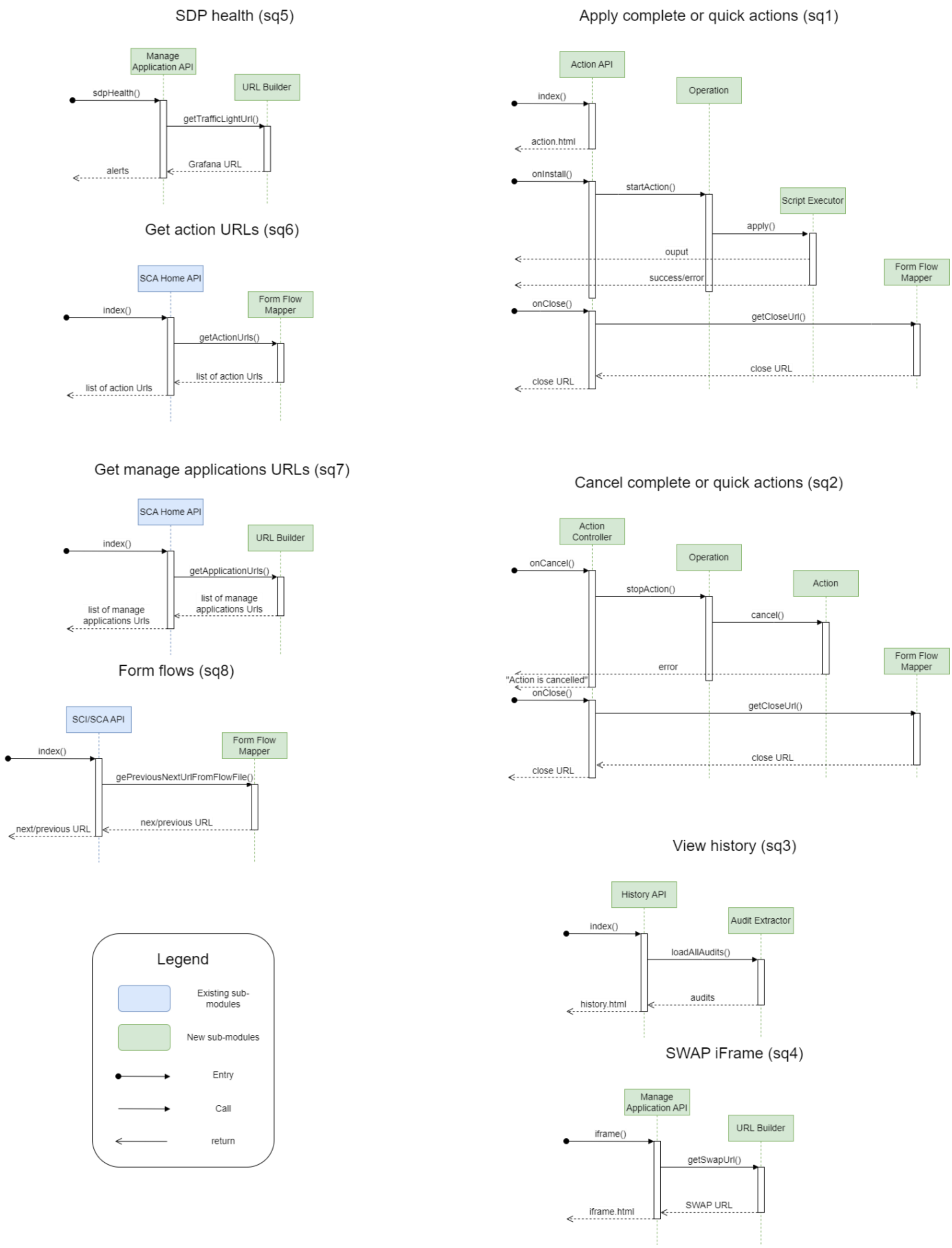
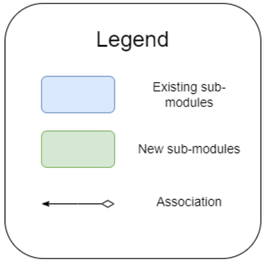


Figure 49. Sequence diagrams of workbench

Logical View (Class/Sub-module diagram)



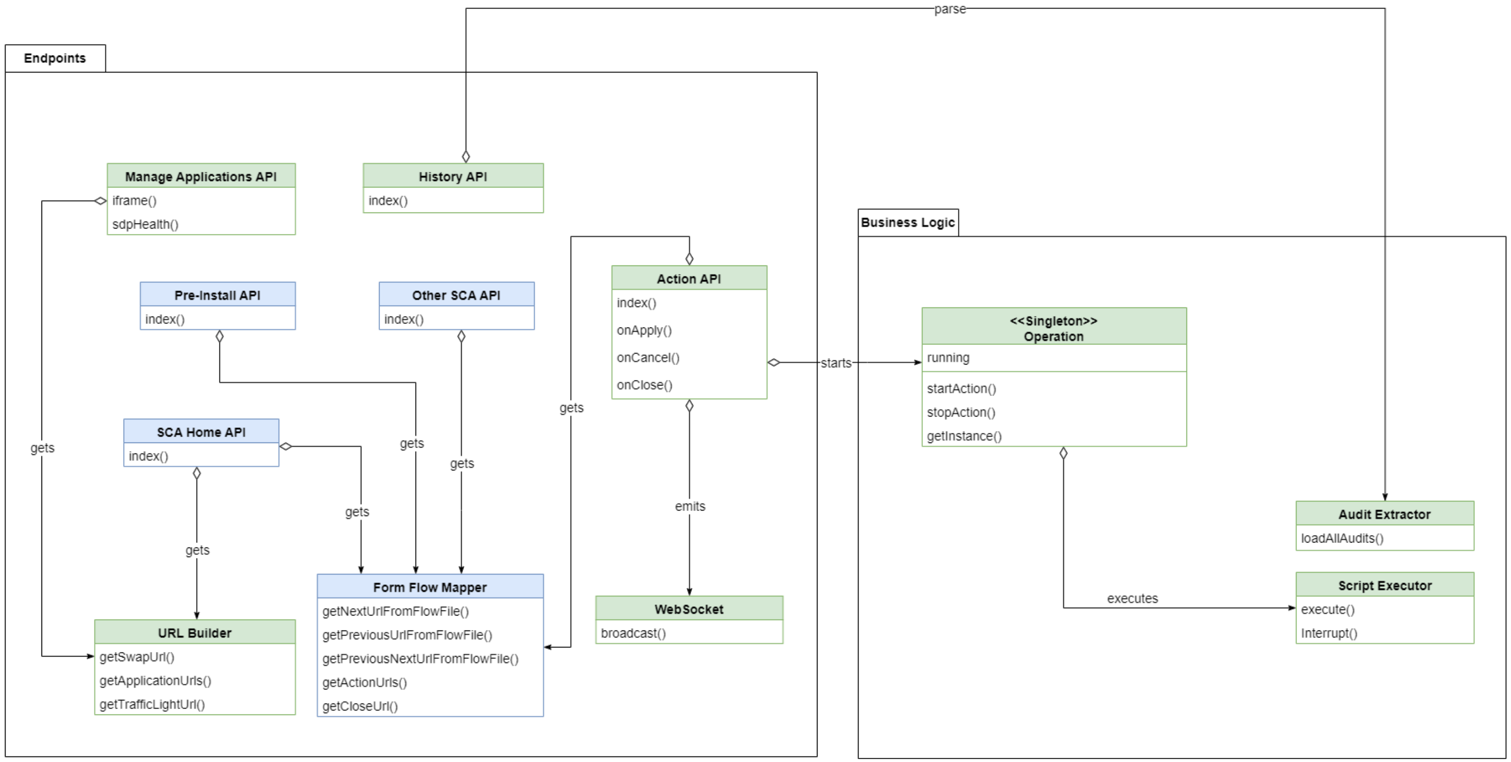


Figure 50. Diagram showing the sub-modules/classes and their relations

Logical View (State Machine Diagram)

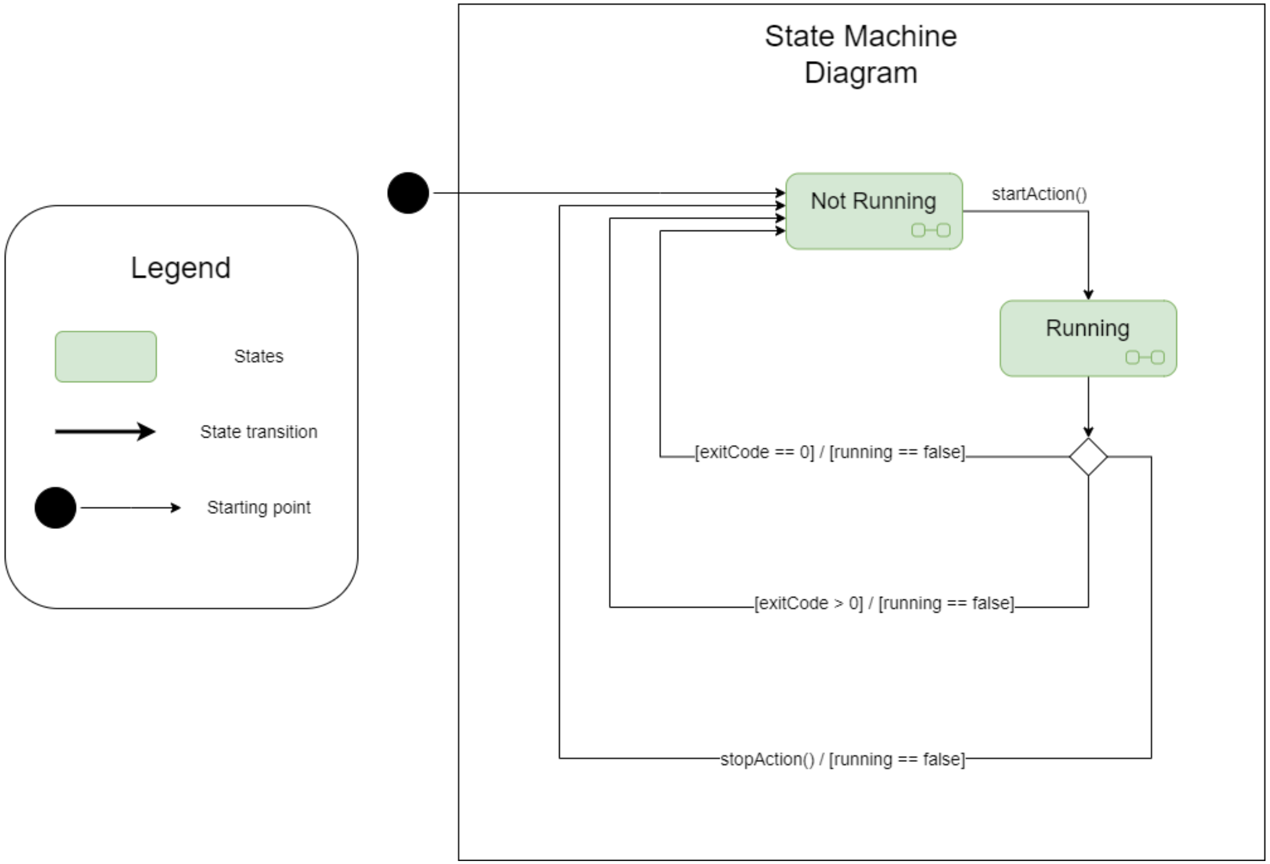


Figure 51. Diagram showing the state machine diagram of operation

Use Cases

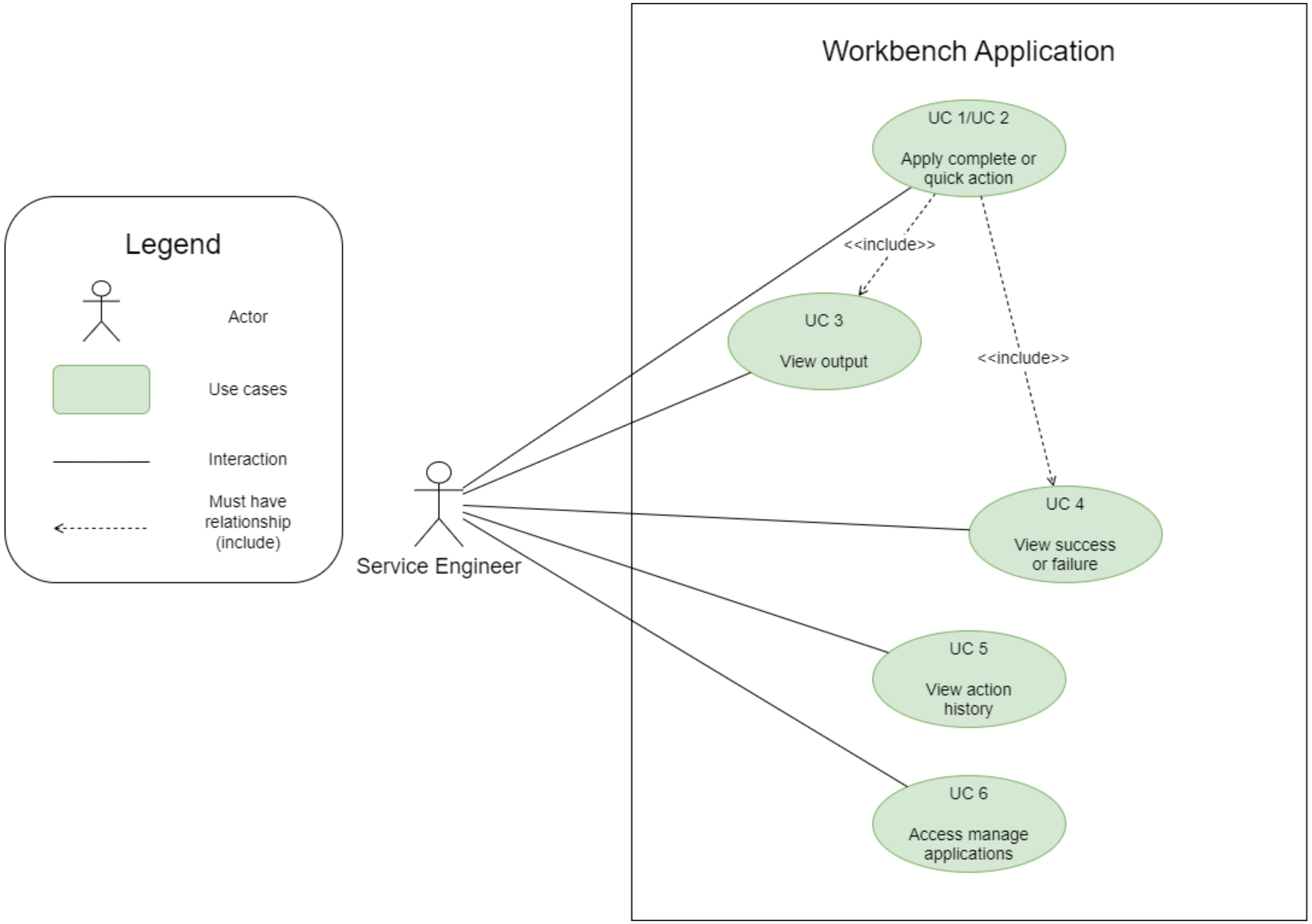


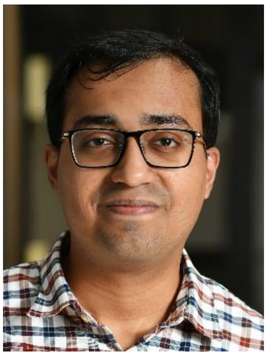
Figure 52. Uses cases for workbench application

Appendix H. Risks

Table 23. Top two risks that occurred during the project

ID	Risk Description	Risk Category	Impact Level (1 - 5)	Probability Level (1 - 5)	Severity Level	Mitigation	Contingency	Status
1	The project may get delayed as SCA and automation scripts are also evolving along with Workbench.	Product	3	3	9	Discuss monthly with RNDEs to know what has changed. Discuss the priorities based on available time.	Discuss the risk with the company and TU/e supervisor during one-to-one or PSG with alternative strategies.	Decided on common interfaces between SCA and automation scripts to reduce dependency.
2	The project may get delayed due to application failure in the production environment.	Product	3	4	12	Organize more dry runs to test the application in the production environment. Understand the nature of the problem and show possible alternatives to the RNDEs. Discuss the priorities based on available time.	Discuss the risk with the company and TU/e supervisor during one-to-one or PSG with alternative strategies.	Changed the paramiko library to subprocess with SSH to support gunicorn gevent-websocket and web socket asynchronous behavior

About the Author



Mayank Sharma received his bachelor's degree in Electronics and Communication Engineering from Manipal University Jaipur, India, in 2017. After that, he completed his master's degree in software engineering at the University of Melbourne, Australia, in 2020. He completed his final year team project with the University of Melbourne Audiology Department titled *Pediatric speech, spatial and qualities (SSQ) of hearing scale*. It involved developing a web application for administering questionnaires and evaluating hearing disabilities in children using the SSQ scale, replacing the old paper-based concept. He and his team came runners-up in the University of Melbourne Endeavour Discipline Award for the team project. Additionally, he worked as a student supervisor at the University of Melbourne, supervising and guiding student teams for their final IT projects. His interests include software architecture and design, multi-domain systems, and entrepreneurship.