UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

**Laima Anna Dalbina**

# Design and Development of an Automated Website Test Management Tool at Rocketlab OÜ

**Bachelor's Thesis (9 ECTS)**

Supervisor(s):
Prof. Dietmar Alfred Paul Kurt Pfahl
Institute of Computer Science
University of Tartu
Estonia

Armin Haller
Chief Executive Officer
Rocketlab OÜ

Tartu 2021

## Design and Development of an Automated Website Test Management Tool at Rocketlab OÜ

**Abstract:**

Rigorous testing of web applications is one of the most important activities ensuring that all components perform as expected. This can be achieved by regression testing where the existing software components are tested when a change is introduced. Manual testing-based practices where scripts and the generated reports are stored in different locations can be tedious and may introduce errors. With the increasing amount of web applications being developed, it is essential that a test management platform exists, that centralizes test execution and reporting. The current thesis addresses the design and development of an automated test management tool within Rocketlab OÜ. The tool is designed as a web application to introduce automated testing within a unified environment. The frontend, which is the primary focus of this work, is developed in React and the backend is handled by Python. The tool environment enables testers to plan and automate the execution and reporting of the tests. The tool also aids in mitigating human-induced errors that might be introduced during extensive manual testing. In addition, it provides a platform for all test projects and the possibility for the clients and inexperienced testers, to execute the tests as well. This leverages transparency between the client and the company. The test management tool bears the potential to be integrated within Rocketlab OÜ to manage multiple testing projects and ensure time-efficient reporting. The tool can also be commercialized by Rocketlab OÜ.

**Keywords:**

Web application, software testing, test management, frontend, automated

**CERCS: P170, P175**

**Lühikokkuvõte:**

Ulatuslik veebirakenduste testimine on oluline, kuna see tagab, et kõik osad töötavad ettearvatult. Seda on võimalik saavutada regressioonitestidega, mis suudavad kontrollida eksisteerivaid tarkvara komponente pärast muudatuste toimumist. Testimislahendused, kus kasutatakse manuaalselt kirjutatud testimisskripte ja kus genereeritud raportid salvestatakse erinevatesse asukohtadesse, võivad olla tülikad hallata ning tekitada vigu. Järjest suureneva veebirakenduste hulga tõttu on vaja, et loodaks testide haldamise platvorm, mis koondaks kokku nii testide käitumise kui ka raportite genereerimise. Selles bakalaureusetöös käsitletakse Rocketlab OÜs loodava automatiseeritud testide haldustarkvara projekteerimist ning arendust. See tööriist on loodud veebirakenduse kujul, et saavutada automatiseeritud testimine ühtses keskkonnas. Töö fookuses oleva toote frontend on loodud React raamistikuga ja backend Pythoniga. See tarkvara keskkond võimaldab testijatel planeerida ja automatiseerida testide käitust ja aruandlust. Ühtlasi aitab loodud toode vältida põhjalikul käsitsi testimisel tekkivaid inimlikke vigu. Pikemas perspektiivis saab loodud platvormil hallata igasuguseid testimisprojekte ning ka klientidel ja vähese kogemusega testijatel oleks võimalik teste käivitada. See võimaldaks hoida klientide ja ettevõtte vahelist läbipaistvust. Testide haldustarkvaral on ka potentsiaal olla integreeritud Rocketlab OÜ tööprotsessi, et hallata mitut testimisprojekti korraga ja võimaldada õigeaegset tulemuste raporteerimist. Lisaks on loodud tarkvaral võimalik saada Rocketlab OÜ pakutavaks kommertstooteks.

**Võtmesõnad:**

Veebirakendus, tarkvara testimine, testide haldamine, frontend, automatiseerimine

**CERCS: P170, P175**

# Table of Contents

# List of Tables

## List of Figures

## Acronyms

| | |
|---|---|
| TaaS | Testing-as-a-Service |
| UI | User Interface |
| BDD | Behaviour Driven Development |
| DOM | Document Object Model |
| ALM | Application Lifecycle Management |
| OÜ | Osaühing (Ltd – private limited company) |
| CEO | Chief Executive Officer |
| OS | Operating System |

# 1. Introduction

Software testing is a crucial part of software development. It not only defines the premise of testing the developed software but also is able to scope major improvements within the same. This includes determining the correctness and quality of the software at different stages while comparing them to the requirements defined by the customer [1]. Therefore, testing is a necessity for any software development. As the size of the development grows, the testing takes up a major proportion of development efforts. With increasing size, keeping track of all the tests becomes tedious. Hence, testing also needs to be well managed.

Parveen et al. [2] state that "test management is a method of organizing test assets and artifacts such as test requirements, test plans, test cases, testscripts, and test results to enable easy accessibility and reusability". A test management system in the form of a software tool brings about a more efficient approach by eliminating human-induced errors within the tests and reports.

Seeing the advantages test management gives, a test management tool Q.tool was developed. It allows to manage automated tests, gather them under suites and group them by projects. It is possible to schedule and execute pipelines and get detailed reports afterwards. In addition, the dashboard displays the statistics regarding test executions.

In Rocketlab OÜ, the current solution for test management is not sufficient. It lacks an overview and unified location of test results and test cases from different projects. This has been an issue as the number of tests from the clients is increasing and it is getting difficult to track them. By implementing the tool within the company, it would improve the current test management process. The tool bears the potential to be utilised by test automation engineers, testers, clients, and admins.

## 1.1. Problem Statement and Goal

Rocketlab OÜ is focusing on test automation which includes regression testing on websites. While executing test automation scripts, the results are shown using reports. They are not only crucial for testers but developers and clients as well. Current solutions have basic reporting using the ExtentReports library[3] however that is not adequate for the company's current needs. While using this library, it is possible to visualize the data only for single test procedures. Although that does not provide efficient and long-term solutions for generating test execution summary. Thus there exists the need for an application that not only consolidates the test summaries but also provides central control of upcoming regression tests. That is the main motivation for the development of a new tool.

The tool, we intend to develop, is a web-based test management system where the user can create new test cases, gather them under test suites, and execute them. The suite can be executed either manually or using test schedulers. After the execution on a server, a report would be automatically generated and saved in the cloud. The tool would have cloud-based storage that would be accessible to users where they can manage their test cases, test suites, and reports. The goal of the thesis is to implement an elaborate reporting system that would suit the company's needs and vision of leveraging regression testing as a service.

## 1.2. Structure of the Thesis

The thesis is structured as follows. Section 1 introduces software testing and the current statement of the problem (section 1.1).

Section 2 talks about the practices and approaches in software testing (section 2.1). It illustrates the different test management tools (section 2.2). and mentions the prospects of software testing in Rocketlab OÜ.

The methodology and development of the proposed test management tool are elaborated in Section 3 including the requirements, design, implementation, verification, and validation.

The results are compiled in Section 4 which portray the final product i.e. the test management tool. It illustrates the design and development along with requirements, design, implementation, verification, and validation.

Prevailing limitations within the tool are sighted in Section 5 which is followed by the conclusion and future work in Section 6 of the thesis.

# 2. Background

Before the tool is designed, it is essential that in-depth information is known regarding the current testing practices and the test management tools that exist within the software testing community.

## 2.1. Testing

In recent times, testing has played a major role in software and web application development. Trivedi [4] states that testing typically takes 40% to 50% of development efforts. This means that it requires extensive and varied testing.

As the web application expands, manual test execution can take a lot of time depending on the size of the application. This is the reason why automated tests are introduced. They can be run with little supervision providing time efficiency for the tester. Software is tested using different approaches and logic. A few testing practices are as follows:

- Unit Testing
- Integration Testing
- System Testing
- Sanity Testing
- Smoke Testing
- Interface Testing
- Regression Testing
- Beta/Acceptance Testing

Moussa et al.[5] writes that regression testing is a type of software testing which is performed when a change in the application has occurred. Such kind of testing makes sure that the existing software parts act as expected when a change in the system is introduced. Web application development requires each release to be tested as new features are implemented with time. These releases can be tested using regression testing which is an integral part of its development.

## 2.2. Test management and tools

With an increase in project size, testing activities increase likewise and the tests get stored at various locations. Once the project size grows, then the testing activities increase as well. Hence, it is much harder to maintain a good overview of all the tests. This is the main reason why test management is introduced. Parveen et al.[2] mentions that test management is a method of organizing test assets to enable accessibility and reuse.

For testing purposes, a tool is introduced which provides the capability to keep all tests from different projects in one place.

Commercial tools such as ALM Octane, Xray, PractiTest, TestMonitor, and TestBench offer testing platforms for manual and automated tests. However, the need to have a modular platform for both testers and clients is necessary. The clients have specific and discrete requirements. For every client, the tests and reports may require special functionality that can be implemented or activated within a modular testing framework.

## 2.3. Rocketlab

Rocketlab OÜ[1] is a company that provides Testing-as-a-Service (TaaS). It means that we have multiple projects from various clients where each has its regression test suite. These projects are needed to be tested, reported, and overviewed professionally while mitigating the scope of errors and missing out on essential information. In addition, to be able to handle the project from a management tool, providing an approach to execute the tests in parallel and storing relevant information within the same tool, can be advantageous.

Based on the regression testing tasks at Rocketlab OÜ, the need for a test management tool has been realized which would solve the following:

- lack of unified location for storing test results
- lack of overview of all projects in one location
- lack of overview of test cases and test suites for a particular project
- lack of statistics about previous pipeline runs

In Figure 1, the current test execution flow used in Rocketlab is shown. Once a test request is received from the client, the appropriate test cases are chosen according to the request. The tests are then individually chosen and executed by the tester on their local machine. After the execution, the result is evaluated. If the test passes, then it is reported as so. On the other hand, if it fails, then the reason is evaluated and the action is according.

If the failure is caused by an error in the testing target, then it is reported as a failure. Although if it is caused by inaccurate test steps, the code is refactored and the scripts are run again.

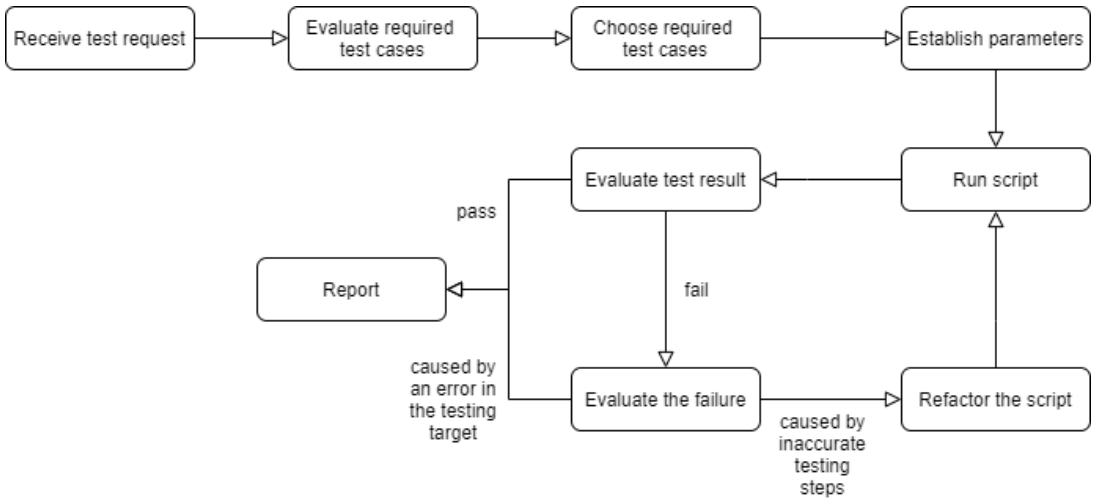


Figure 1. Flow graph for test execution manually

Currently, test execution is handled within an integrated development environment (IDE) by a tester and reports are saved on a local computer. This gives an additional task to the tester where the reports have to be sent to the supervisor. This does not provide automated upload to cloud-based storage and can introduce miscommunication. As the tester is using a local machine, for many testers the computer differs which can introduce certain inherent

[1] Rocketlab OÜ is a trademark company registered in Estonia (https://www.rocketlab.ch/en/)

errors. For example, the environment in the computer is set differently or the computer operating system differs from one tester to another.

These issues led the company to realize that a tool is necessary to solve them. First of all, the web application would be modular and easily adaptable. As the application is made within the company, necessary features can be added quickly, without any issues. This enables crucial feature implementation. In addition, this tool satisfies the current vision of the company, which is to have a test management tool. The tool is owned by the company and eventually can be monetized.

When working with clients, transparency of results is required. A client can have a simpler user interface (UI) while using the application rather than an IDE. This is crucial as the client can be certain of the tests and test results.

# 3. Method

This section describes the methodology of requirements gathering, design, and implementation for the test management tool.

## 3.1. Requirements Gathering

The requirements are gathered by interviewing the CEO of the company and colleagues who actively work on executing the tests. There are multiple ways how to gather requirements, although an interview is chosen because it is one of the most common ways for it. Also, it is one of the primary sources of requirements. Discussion and brainstorming sessions led to realize the potential for the tool to make it user intuitive and work efficiently at the same time. Essential features were also highlighted that provided a cutting-edge approach to the tool. Supervisor and colleagues are required to interview because they give a viewpoint which is being developed by their personal experience. Every person has worked on a different part of testing, therefore, provided a unique view of the problem.

The requirements are captured in the form of user stories. They have become the standard to provide a summary of the functionality for both technical and non-technical members. Also, user stories help to define the entire product and its functionality. This allows to define and understand the requirements easily across all of the stakeholders. During early discussions, the stakeholders and their needs are identified. The first stakeholder is a client who hires Rocketlab to test and provide test results for their websites. Another stakeholder is the CEO and the manager who are responsible for overlooking the testing tasks. They are given the role of admin. A test automation engineer is an additional stakeholder who is responsible for running and maintaining the tests. A tester as a stakeholder is responsible for running the tests.

## 3.2. Design

Regarding the design of the tool, it is necessary that the platform, upon which the tool is built, is decided. Based on the company's policies which prefers server-based tool over intranet-based, it is a prerequisite that the tool is made as a web application.

It is required to define the various components of the web applications:

- Frontend
- Backend
    - Web framework
    - Database
    - Task execution

The decisions for the design of the tool are based on gathered requirements, existing tools in the company, the company's policies, and personal experience.

## 3.3. Implementation

For the implementation of the web application, it is necessary that it contains the components that execute all desired tasks in both frontend and backend.

One of the widely used tools for this purpose is Docker[6]. Other options are rkt[7], LXD [8], and Windows Hyper-V[9].

For the frontend, there are many choices. Most popular ones are React[10], Angular[11], and Vue[12]. All of them have their advantages and disadvantages, although React has the advantage of the author's previous experience with the framework.

For the backend, the foundation of the component is based on a web framework that enables the development, creation, and publishing of web applications. Depending on the chosen language, there are many possibilities. Python is relatively easier language to learn where the author has adequate experience. For backend development, Python is widely used. There are various frameworks based on Python: CherryPy[13], FastAPI[14], Falcon[15], Bottle[16], Flask[17], Django[18] and others.

There are two possibilities for a database: relational or non-relational. Relational databases are more suitable for highly structured data which is easily stored and retrieved and is suited for varying sure permissions[19]. A non-relational database is more used when the data is unstructured and it is necessary to scale horizontally[19].

To support task execution which is necessary for a more complex web application, there are few solutions based on Python: APScheduler[20], Crontab[21], Celery[22], and others.

## 3.4. Verification and Validation

Since a new tool is developed, it must be tested. The main functionality of the tool is tested automatically. All of the user stories are covered by either automatic or manual tests.

To implement integration testing, test cases are developed to test the main functionality of the tool.

After the main functionality is implemented, to gain feedback, an interview is conducted with a colleague experienced in the test automation field. The interview consists of 5 main parts:

- Introduction and schedule of the interview
- Problem statement and goal
- Architecture explanation
- Demo of the tool
- Questions and feedback

The questions cover topics of usability, user interface, user experience, the defined problems, additional functionality, and other test management tools.

# 4. Results

The section deals with the results of requirements gathering, design, and development of the test management tool.

## 4.1. Requirements Gathering

As mentioned in section 2.3, the current testing approach is manual and lacks the unattended execution that regression tests require. When comparing Figure 1 and Figure 2, we observe how the test execution changes once the tool is introduced. A large part of the tasks is handled by the web application. Compared to IDEs, the user experience is made easier by using the application. The user does not have to be a programmer to plan and execute the tests. Running the tests is handled by a server and they are run in parallel which drastically increases the total execution time for one pipeline. Evaluation of the test status is handled within the application as well.

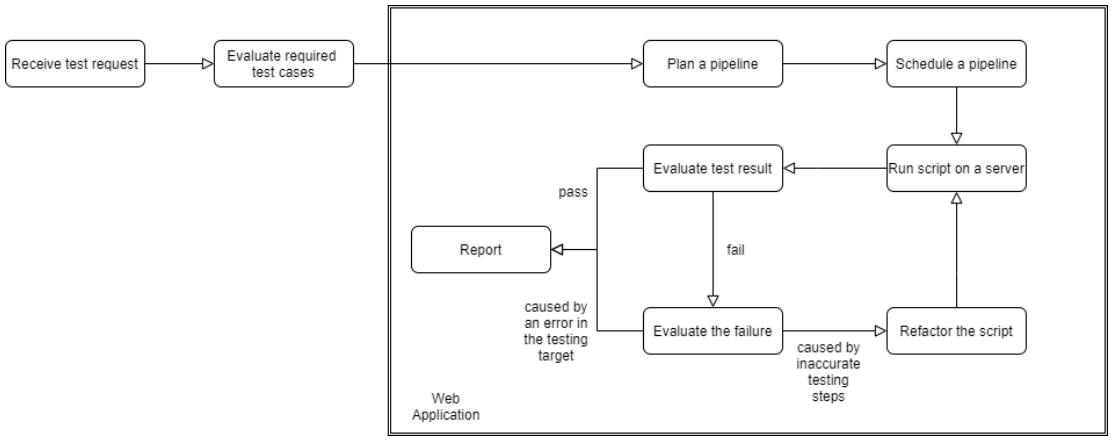


Figure 2. Flow graph for test execution within the QTool web application

To satisfy all the requirements, it was decided to develop a web application. Various components of the web application are identified as the following: architecture, authentication, code design, design, database, task execution, and monitoring and integration.

The requirements are composed of user stories that include both functional and non-functional requirements.

### 4.1.1. Functional Requirements

There are 11 user stories identified for 3 roles: tester, test automation engineer, and admin. The user stories are shown in table 1 with their respective IDs.

| ID | User story |
|---|---|
| US1 | As a tester or test automation engineer or admin, I am able to log in to the tool. |
| US2 | As a tester or test automation engineer, I am able to access the dashboard view. |
| US3 | As a tester or test automation engineer, I am able to access the execution view. |

| | |
|---|---|
| US4 | As a tester or test automation engineer, I am able to access the report view. |
| US5 | As a tester or test automation engineer, I am able to view projects and pipelines in execution view. |
| US6 | As a tester or test automation engineer, I am able to schedule a pipeline in execution view. |
| US7 | As a tester or test automation engineer, I am able to view scheduled and finished pipelines in execution view. |
| US8 | As a tester or test automation engineer, I am able to view finished pipelines and its reports in report view. |
| US9 | As a test automation engineer, I am able to access the planning view. |
| US10 | As a test automation engineer, I am able to view, add, edit and delete projects, test suites, pipelines, and test cases in the planning view. |
| US11 | As an admin, I am able to see the admin view. |
| US12 | As an admin, I am able to see all users, edit or delete them in the admin view. |

Table 1. Functional requirements depicted as user stories

### 4.1.2. Non-functional Requirements

The non-functional requirements for the QTool are as follows:

| ID | User story |
|---|---|
| US12 | As a tester, test automation engineer or admin, I want to be able to run the tool on 5 latest versions of Chrome, Firefox, and Edge by having full functionality available when using. |
| US13 | As a tester, test automation engineer or admin, while using the application, the size of elements and their location on the page responds to the changes in the window size. |
| US14 | The application scales with desktop, tablet, and mobile screen resolutions where the smallest resolution is 360px × 640px and the largest is 2560px × 1440px |
| US15 | As the CEO, I want the system to use our existing servers to deploy the application |

Table 2. Non-functional requirements depicted as user stories

### 4.1.3. Interpretation of collected requirements

The requirements mentioned in Tables 1 and 2 are identified as potential requirements that are essentially required for a coherent usage scenario for different actors working on the web application. They define the premise of using the tool to perform different actions that would yield desirable outputs within the environment. The 3 actors - tester, test automation engineer, and admin - have different accessibility within the environment which enables them to use the tool in different aspects. These requirements are functional and non-functional. Functional requirements are the ones that describe the functionality of the software, and non-functional requirements define the system attributes which constrain the design of the system. Based on the identified requirements the technical know-how has been discussed below. These constitute the knowledge base that a person needs to possess to use the tool to its complete extent.

#### *4.1.3.1. Client Projects*

As mentioned in section 2.3, in a company, which provides TaaS, the clients give projects which need to be tested. The projects in the company only consist of the required tests for a particular client.

Test execution is a crucial part of the web application. To make it possible, it is necessary to have a code that is comprised of tests. The code gathered under a Java project is called a *client project*. In the client project, we have all the classes and tests necessary for test execution.

#### *4.1.3.2. Project Creation*

There are two cases of how a client project is implemented. The user can start a new project in the application or can import an already existing project.

If the user wants to start a new project from scratch, then it is possible to add it using the application. This way the tool will add a project with the name and description mentioned by the user to the client project directory. It is being done by taking a template project and adding the necessary details to it using a Cookiecutter[23] library.

If a project is already created, it is possible to import the whole project into the application and it will detect and parse using a script all the test suites and test cases that are already defined and add to the project.

#### *4.1.3.3. Task Execution*

The task is defined as an execution of a test case or a test suite. To execute any test, we need to plan them accordingly. Once a test is planned, it can be added to a test suite. Once all the necessary test suites are planned then it is possible to plan a pipeline by assigning test suites to it. Once a pipeline is planned, it can be scheduled and executed. Only a pipeline can be executed with the application. With Celery, it is possible to schedule the pipelines and obtain the results afterward.

#### *4.1.3.4. Result Acquisition*

After a pipeline is executed, every test case is assigned a status: passed or failed. From these results, it is possible to view the status of the test suite. If all passed, then the test suite status is passed. However, the failure of even a single test result in the fail status of the entire test suite. The same logic applied to pipelines where we see the status of all test suites. Every test case has its details report with test steps. These are mainly meant for the test automation engineer who can see the detailed logs and errors if such occur.

#### *4.1.3.5. Hierarchy of Collection*

All the elements which are possible to add in the application together are termed as a 'collection'. The hierarchy of the collection is depicted in Figure 3. The project is where all the collection elements are gathered under. It is possible to add pipelines and test suites to a project. Test cases are gathered under test suites. Test suites can be assigned to a pipeline as well.

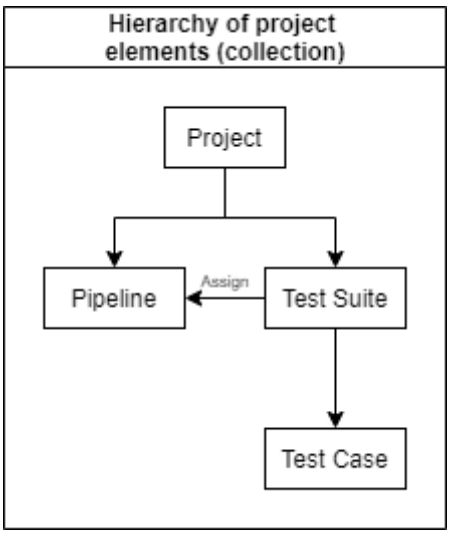


Figure 3. Hierarchy of collection

## 4.2. Design

The design aspects of the tool were considered while keeping in mind, the concepts that contributed to smoother development along with providing a intuitive user experience. This involved the use of a comprehensive tool framework that hosts a detailed architecture along with the frontend and the database. The ability to format the code and provide authentication lead to a suitable tool design.

### 4.2.1. Architecture

Docker holds the application and isolates the development of a web application; hence the internal conflicts are evaded. As Docker provides an environment that does not depend on the computer's OS, it makes it easier to avoid unnecessary OS-dependent issues. In addition, Docker provided a modular approach that allows smoother and quicker development.

Nginx[24] web server is used to serve the application and process web requests. It is one of the most used web servers in web development and provides all the necessary features for the tool.

In Figure 4, the architecture of the web application is shown. The actor[2] interacts with the frontend which is served using Nginx. Both frontend and Nginx communicate with the backend to get all the necessary information. Backend communicates with the database, client projects, and task queue, Celery. Flower monitors tasks within Celery and Redis acts as a message broker. Redis creates workers that execute the tasks scheduled by Celery.

[2] Actor can be a tester, test automation engineer or admin. Test automation engineer can access all 4 views, although tester cannot see the planning view. Admin has separate admin view with data from the database.

Workers and backend communicate with Client projects. Client projects is a separate folder that holds the imported projects or projects made by the tool. The projects are developed in Java using a testing framework, TestNG.

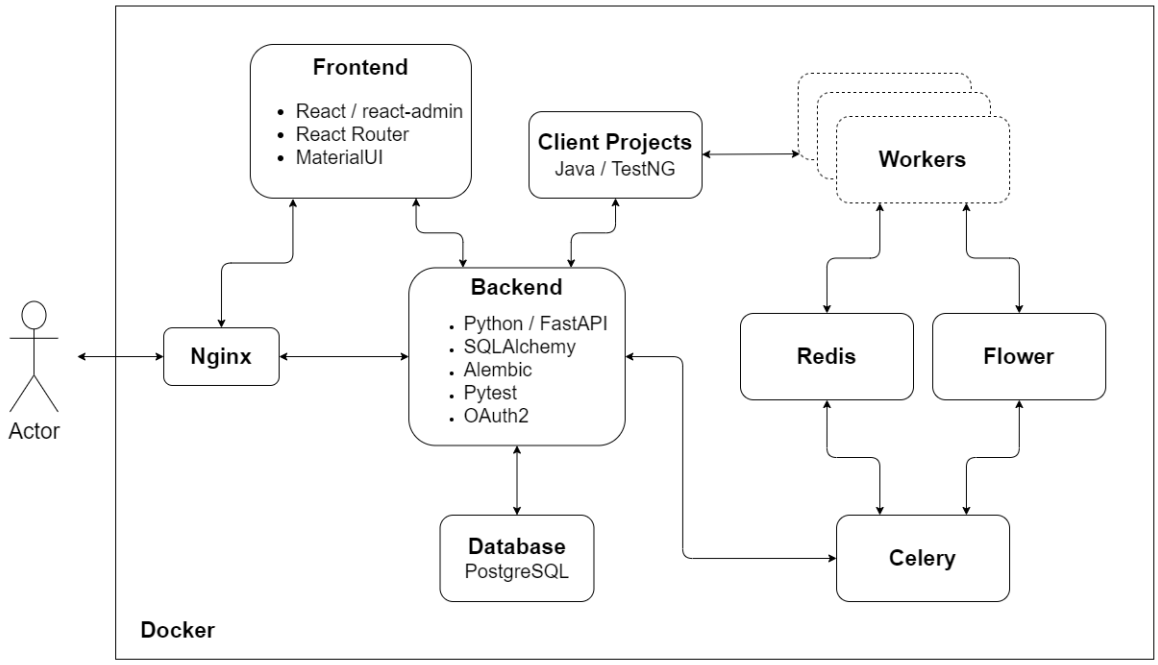


Figure 4. The architecture of the web application

### 4.2.2. Web Framework

FastAPI is chosen to be the web framework for the application. It provides quick responses as compared to other major Python frameworks like Flask and Django. It supports asynchronous code and has a very short development time. Also, excellent documentation and easy deployment ease the development stage for the application.

### 4.2.3. Authentication

FastAPI has the functionality to support login. It uses OAuth2[25] to enable a secure login. This feature helps to provide secure access to only those users who have been registered by the admin.

### 4.2.4. Code Formatting

Additionally, Prettier[26] and ESLint[27] provide automatic feedback about problems in code and provide the possibility to automatically fix problems and formatting. This enables to strengthen efficient coding practices.

### 4.2.5. Frontend

For the frontend, the JavaScript library React with MaterialUI[28] and react-admin are used. This aids in making a highly dynamic, responsive, and interactive design. React introduces numerous advantages such as virtual document object model (DOM) which increases the speed of updates enabling to make highly dynamic UI. React is well-known for its reusability enabled by components. This undoubtedly saves time for the developers which is crucial in the fast-paced web development world.

With React it is possible to enable Redux[29] - a convenient state container that allowed the state of the objects to be stored and passed to other components easily. React Hooks improves state management even more. It helps to increase the reusability of components' logic and adds lifecycle methods features to functional components. In addition, React is an open-source library that gives developers the support, applications, and additional tools from other developers.

### 4.2.6. Database

To make the frontend functional, it is necessary to implement working functionality. In web development, 'backend' is the term used to describe the part of the application which enables the logic and functionality between components. This is enabled by object-relational database PostgreSQL[30] with the addition of database toolkit SqlAlchemy[31] and lightweight database migration tool Alembic[32].

This trio enabled by Python is capable of taking charge of database manipulation to store the data coming from the application's frontend. PostgreSQL gives a way to store the data in an object-relational database, whereas SqlAlchemy with the integration of Alembic gives an intuitive way how to access the database using Python.

### 4.2.7. Tasks Execution and Monitoring

For the test management tool, the execution of tasks is very crucial. Celery[22] is a very popular Python-based task queue that enables asynchronous task scheduling and execution. Flower[33], a web-based tool, provides the possibility to monitor all of the tasks and task queue in a web-based tool for the administrators. As Celery is best used with a message broker, Redis, an extension to Celery, is used[34]. Redis is used to store messages describing the task and to store results from Celery queues.

### 4.2.8. Tool Framework

During the research, it was found that a template framework from an open-source project 'fastapi-react'[35] suits the web application's needs and provides all the necessary features. It uses such technologies as FastAPI, React, PostgreSQL, SqlAlchemy, Celery, Flower, Alembic, Pytest[36], Prettier, ESLint, Docker Compose, Nginx, MaterialUI, react-admin[37]. The decision to use a template project was decided due to 2 main reasons:

- Time Efficiency

  Approximately 90% of the time is spent in setting up the project framework[35]. it is essential to develop a working tool to respect internal company timelines and deadlines. Hence, a premade framework was utilized to develop the tool citing Rocketlab OÜ project deadlines.

- Contribution to Open Source projects

  Open source projects not only gather feedback from a wider community but also enthusiasts can contribute to open projects that might provide essential elements to be added to the tools.

## 4.3. Implementation

Implementing the tool design required thorough understanding of the structure. This was implemented within the fastapi-react open-source project along with a frontend to be able to perform the desired tasks. The structure and the frontend design implementation of the tool are discussed in the following sections.

### 4.3.1. Tool Structure

The project structure is shown in Figure 5.

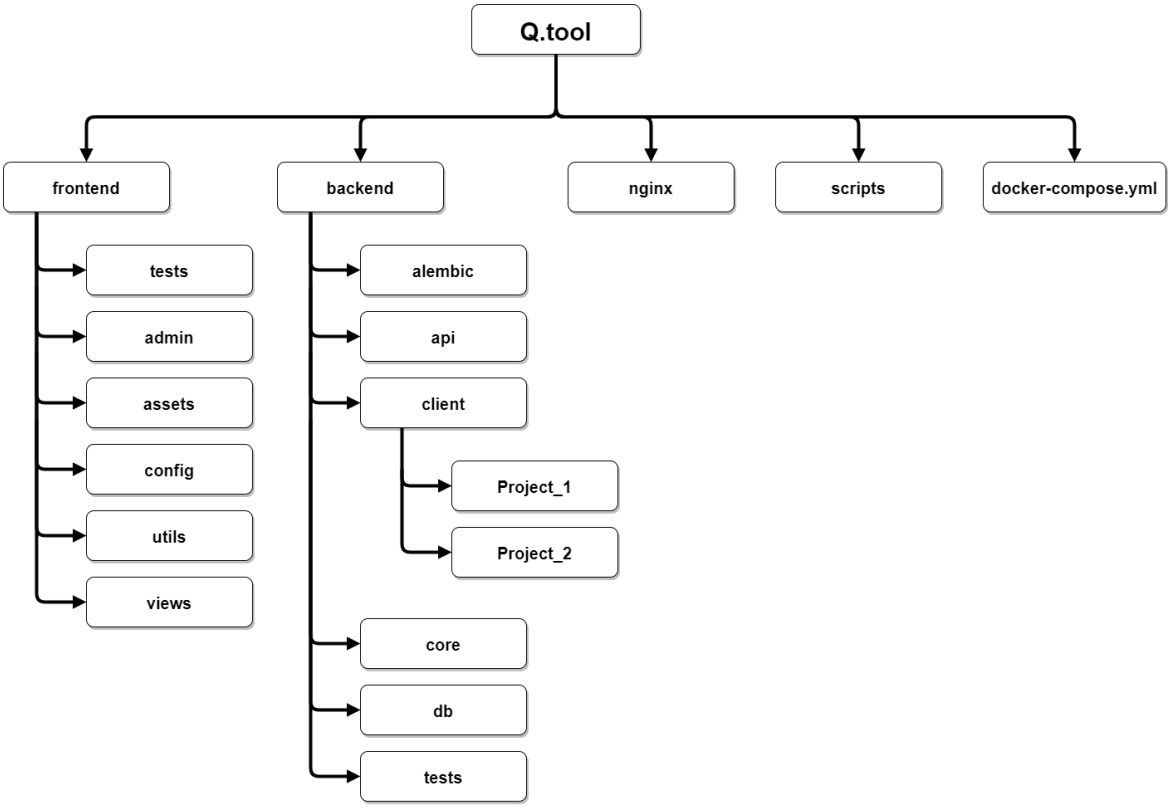


Figure 5. Project directory structure of the web application

The project is structured in 4 main folders: frontend, backend, nginx, and scripts. The frontend has 6 subfolders which include tests, admin view, assets, configurations, utilities, and views. The backend has 6 subfolders: alembic, api, client projects, core, database, and tests. Nginx folder has the configuration for Nginx and scripts consist of shell scripts to run the Docker files. The file docker-compose.yml has the configurations for Docker images.

The access to the repository can be granted by requesting the author (laima.anna.d@gmail.com).

### 4.3.2. Frontend Design Implementation

The web application is implemented in 4 main parts (Figure 6):

- dashboard
- planning
- execution
- reporting.

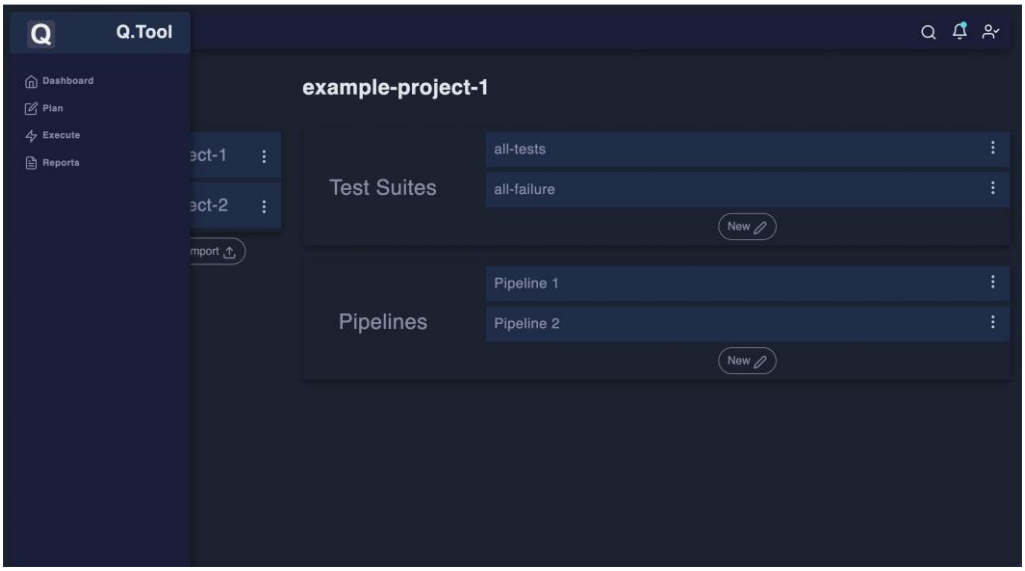


Figure 6. The navigational panel

The dashboard is the first section in the navigational panel (Figure 6). As shown in Figures 7 and 8, currently 3 graphs are shown. The pie chart shows passed and failed test cases. The line graph shows the amount of time taken by the number of test cases, test suites, or pipelines. It is plotted for the time in minutes against the number of test cases in a particular period. The third graph shows the reason of failures and how often they happen.


Figure 7. The dashboard displaying graphs ‘Passed/Failed Test Cases’ and ‘Time Taken’. On the left, there is a filter bar left

On the left panel, a filter menu is shown. It is possible to filter out the data that is plotted with the graphs. Initially, all projects within the tool are shown. Once a project is selected then all pipelines within the project are displayed. The same logic is applied when we choose a pipeline and later a test suite.

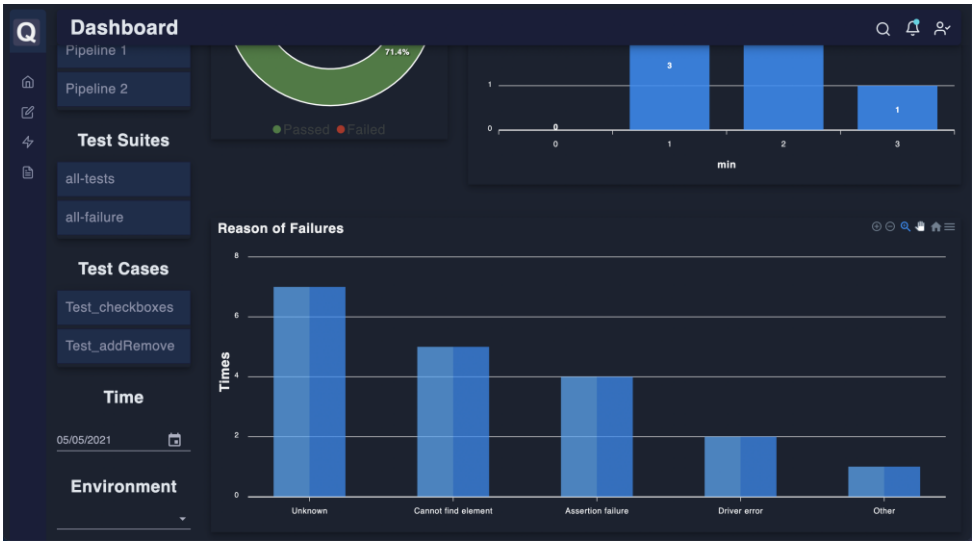


Figure 8. The dashboard displaying graph ‘Reason of Failures’ and a filter bar

The next element in the side menu is the planning section which is shown in Figure 9. The planning section allows to add new, edit or delete projects, pipelines, test suites, and test cases. A modal window is shown is used to add or edit any of the collection elements. When choosing a project, then pipelines and test suites that belong to the current project are shown.

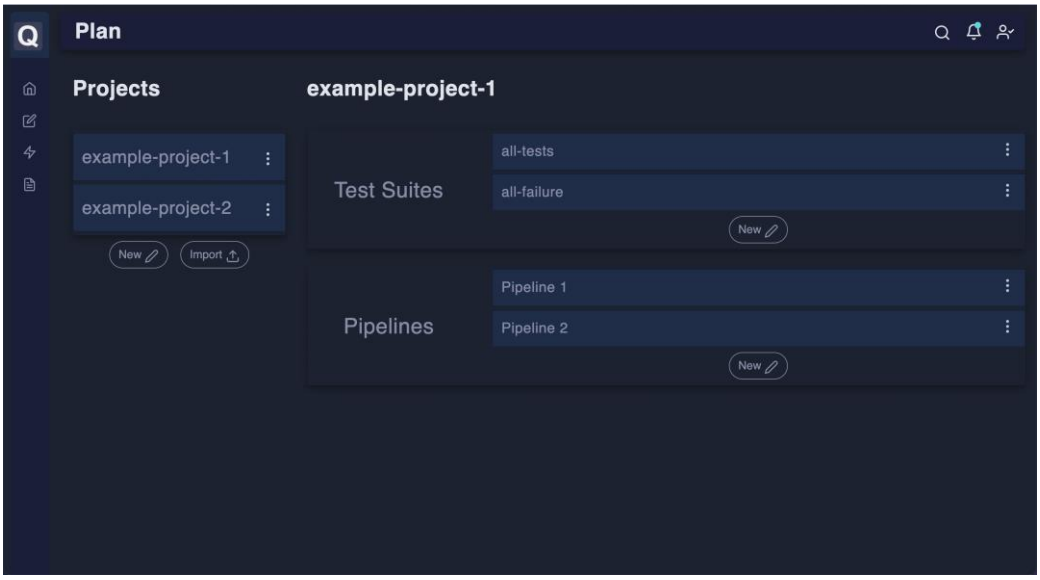


Figure 9. The planning view

By clicking on a particular test suite, a detailed view of all the test cases assigned to the test suite is shown (Figure 10).

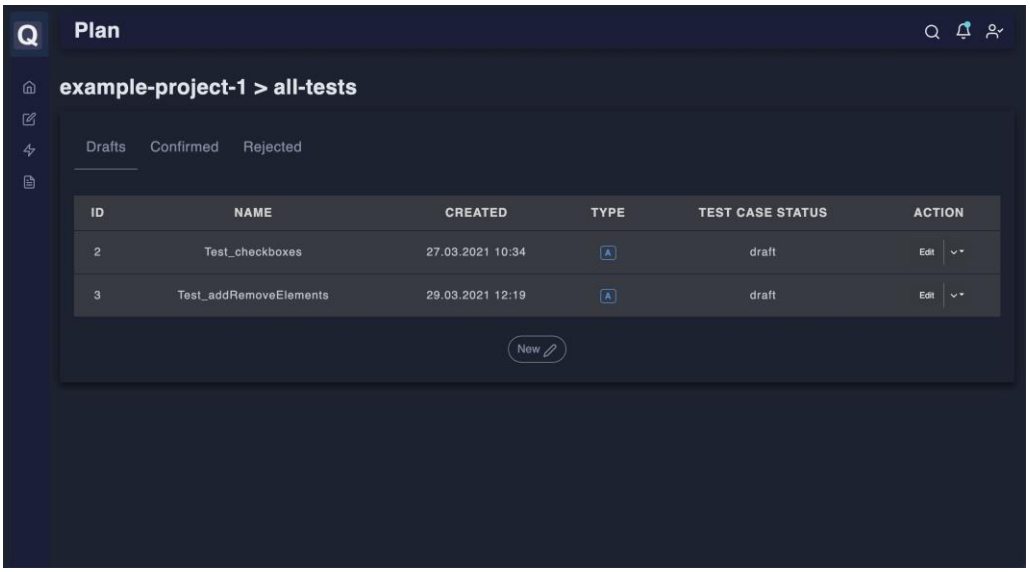


Figure 10. The planning view where test cases under a particular test suite are shown

Under the next section, the execution view is visible. Here it is possible to schedule pipelines and see the status of scheduled and finished pipelines.

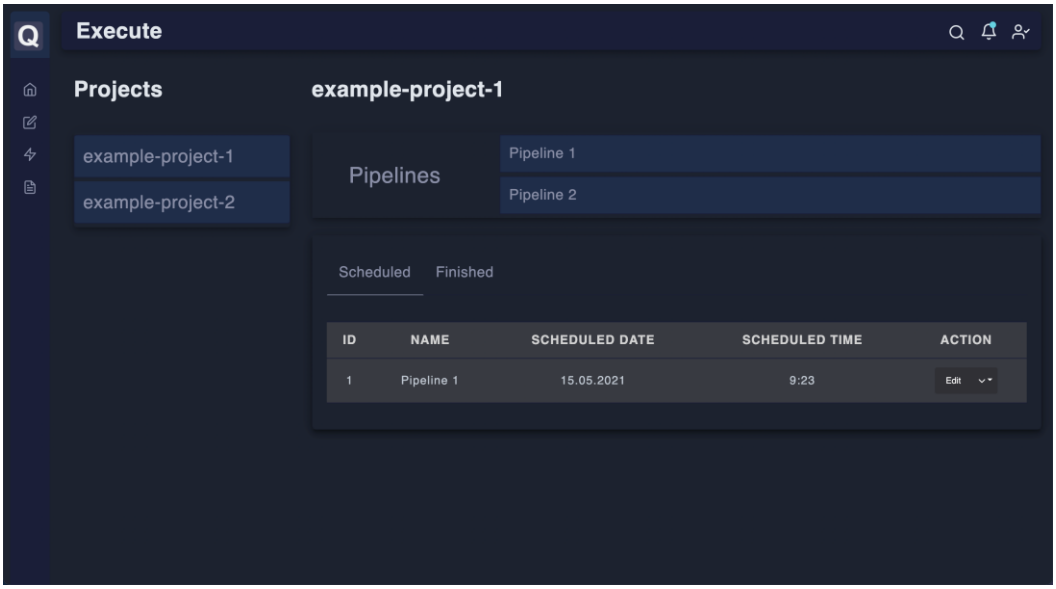


Figure 11. The execution view where a table of scheduled pipelines is shown

When choosing a project, only planned pipelines are shown in this view. On clicking a pipeline, it is possible to schedule it. By using the simple scheduling tab, it is possible to schedule the pipeline to run once at a particular date and time.

Once a pipeline is scheduled, it appears in the scheduled pipeline tab as shown in Figure 11. The scheduled time and date are shown in the table. It is possible to edit and cancel the pipeline execution. Once the pipeline is running or finished running, it is added to the finished pipeline tab (Figure 12). Once open is clicked, it goes to the report view where the report for this particular pipeline execution is shown.

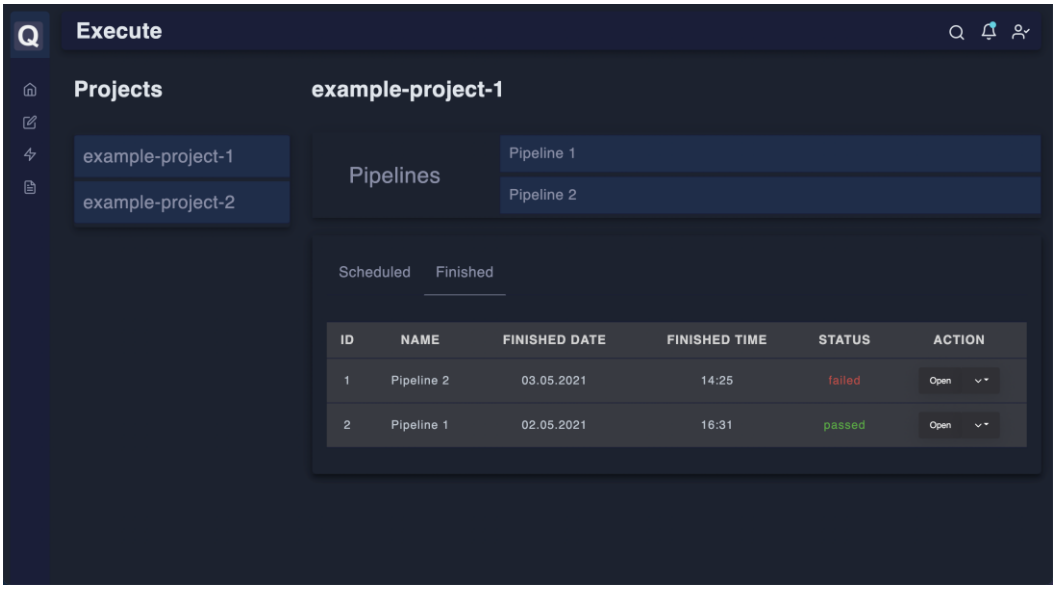


Figure 12. The execution view where a table of finished pipelines is shown

The last part is the reporting view (Figure 13). Here it is possible to choose a project and all of the executed pipelines are displayed. The details such as id, name, finished date and time, and status are shown in the table. Once the open button is pressed, a detailed view of the report for a particular pipeline is shown, as shown in Figure 14.

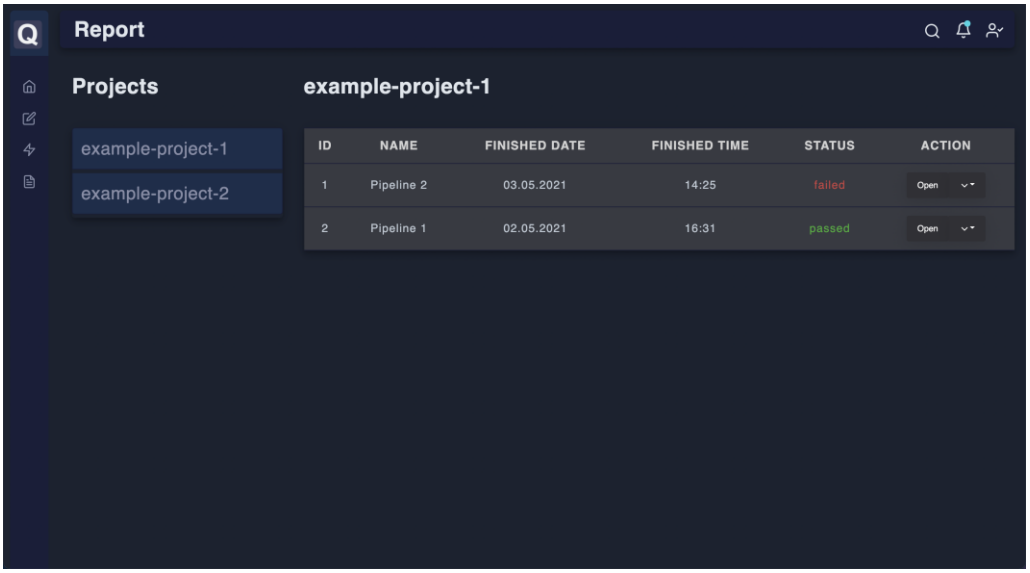


Figure 13. Reporting view

In the detailed report view (Figure 14), it is possible to see all the test suites executed. On clicking a test suite, a table with its test cases is displayed. It is possible to click ‘open’ which displays a detailed HTML report regarding the particular test case. This report file is produced from the test run and attached to the test case.

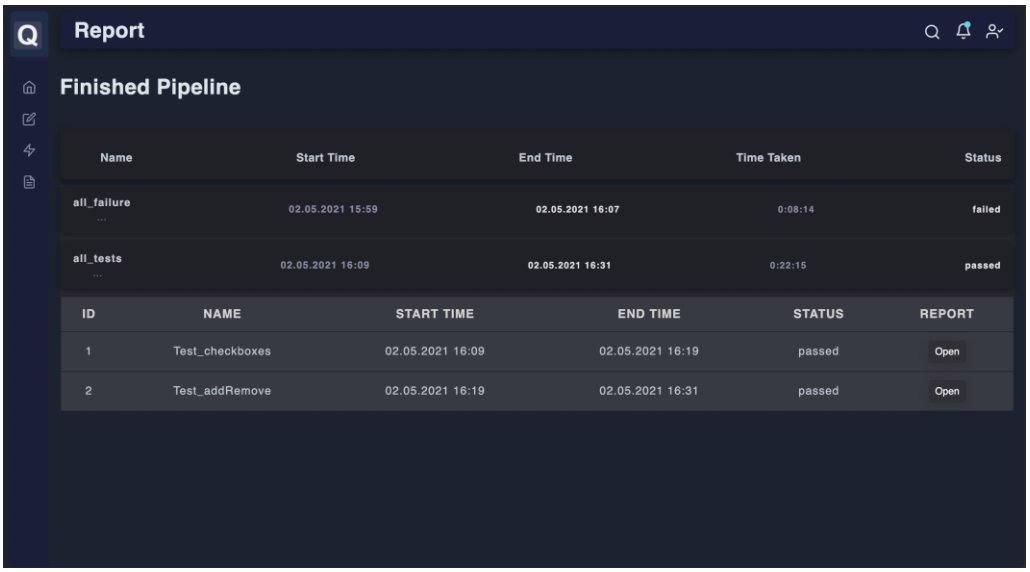


Figure 14. Detailed report view

A user manual is mentioned in the appendix I which shows the complete functionality of the tool.

## 4.4. Verification and Validation

To verify and validate the developed tool, it was tested and analysed. The methods are elaborated in sections 4.5.1, 4.5.2, and 4.5.3.

### 4.4.1. System testing

The web application is tested manually and by automated test scripts. To verify that the application behaves as intended, Pytest is used for the project. It enables backend unit testing using Python. Pytest is executed after any major or minor update to avoid any unintentional mistakes. Postman[38] is used for testing the API which provides the link between the frontend and the backend. For frontend automated smoke tests are implemented. They inspect whether all components load without issues.

To verify the tool functioning correctly, manual tests were performed. The tests were planned according to the user journey in section 3.2. To test the functionality of the tool, a Java-based test project has been created, named ’example-project-1’. It consists of 3 test cases and 2 test suites. All 3 test cases assess some part of the functionality of a website “The Internet”[39].

A test case is developed where a project is added and for the project, a test suite, a pipeline, and a test case are added. Test steps and results are as follows are mentioned in appendix II table 1. The screenshots of the actual result in the tool are attached after the test steps.

Another test case is made that tests the test execution and result acquisition. The test steps and results are mentioned in appendix II table 2. The screenshots of the actual result in the tool are attached after the test steps.

### 4.4.2. End-user testing

The end-user testing included 2 different verification procedures: usage scenario and feedback.

Both these cases contributed to the validation of the tool based on guidelines to evaluate the system and an interview with an experienced colleague, either of which provides insights to the lessons learned for designing the test management tool

#### *4.4.2.1. Usage scenario*

The usage scenario gives guidelines on how to evaluate the system. In Figure 15, it describes the main steps a user is required to follow to use the main feature of the application. There are 2 user roles for this flow: Test Engineer and Tester.

First, a user accesses the web application and logs in. To log in, the user has to be registered by an admin. A dashboard appears. There are 4 views in total:

- dashboard
- plan
- execute
- report

Test Engineer can access all, although Tester is not capable of accessing the planning view. Under the planning view, the user can plan new, edit, or delete any of the collection elements. Under the execute view, it is possible to schedule any planned pipelines. Also, scheduled pipelines and the last 10 executions are shown. After a pipeline is finished, it is possible to see it in the finished pipeline tab. When clicking on it report view opens and shows a report for the pipeline. Once clicking on a test case, a detailed test report with steps appears.

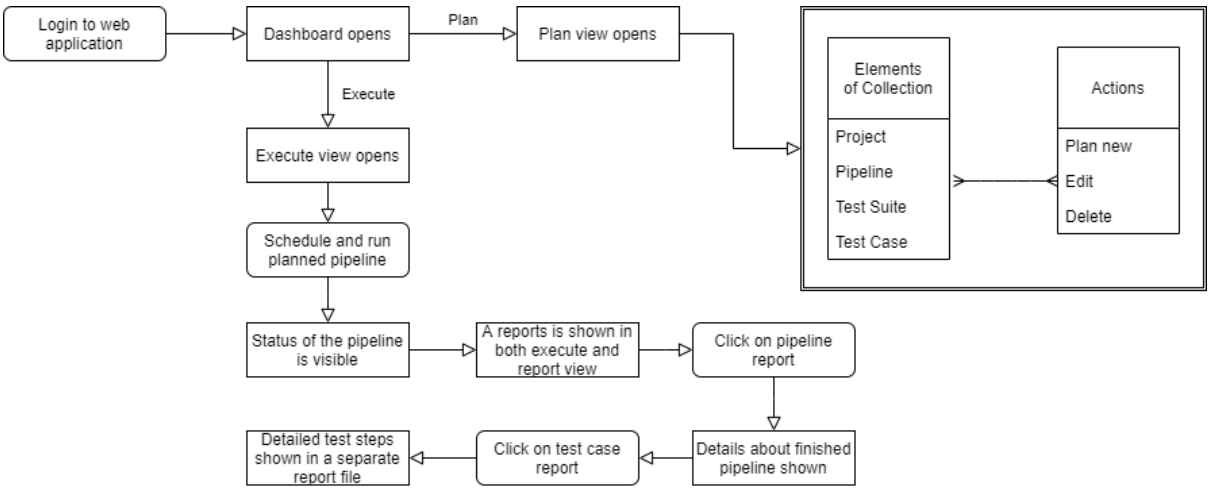


Figure 15. Usage scenario within the web application for planning, executing, and reporting the tests

#### *4.4.2.2. Feedback*

An interview was conducted with a colleague from the company who is experienced with software testing. The interviewee mentioned that the problems stated in section 2.3, are addressed by developing the tool. Overall the feedback is positive. Work accomplished with the QTool within 3 months of development was well appreciated. However, there is still some work to do to have the test management tool fully implemented within the company. Some advantages that were brought out are that it is possible to easily modify the in-house tool as necessary. There is no block from a 3rd party solution where some of the features are

not implemented. Having one tool for both test management and test execution is a good solution and not many products are supporting this. A disadvantage to the tool is that it will take more time and effort to maintain tool.

Some suggestions mentioned during the discussions are as follows.

The dashboard could be modified so that it includes not only overall results for the test run but also detailed results about the previous pipeline run. The results can be compared to the average for the same pipeline. A toggle switch would be used to change these views. The reason of failure can be shown as a table with the frequency for each reason shown. Also, a total number of filtered elements of the collection should be shown. Under the overall dashboard, there could be a graph that shows the past execution status for the past week.

One more suggestion that was made is to have more supported frameworks and languages for the tool. Once the tool is ready with one framework, then it was planned that more are added. In the future, it is a possibility to have the integrations with Jenkins and Jira.

Regarding the user interface, it was advised to colour code the different views so the user can easily distinguish between them.

Another suggestion was to add a widget that has the most crucial information and actions on it for quick access

When asked about whether it is possible to commercialize the tool, it was answered that wider market research is necessary to answer this question.

### 4.4.3. Lessons learned

The testing reveals that the tool is capable of providing an overview of the projects and test suites and test cases under them. As the tool can manage multiple test cases which is useful not only for the company but for the client as well. If the client is provided access to the tool, then it improves the transparency of the results where the client himself is capable of seeing the test result and overview them. The tool is capable of executing pipelines, which consist of several test suites and test cases. It provides a unified place for test results along with an overview of the same.

Taking consideration from the feedback, the tool could have had more features, but based on company timelines and the development period, the main functionality of the tool was developed. With more discussions to follow, new features and additional functionality shall be implemented which will lead to full integration of the tool within the company. As discussed in the interview, the tool has the potential to be launched as a commercial product.

# 5. Limitations

As the tool is importing new projects which are developed in Java, it is required that the project is configured properly. The Java project is required to be configured with Maven[40] to execute the tests. There are several libraries that the project requires. While importing the names of the test cases and test suites have to have a particular keyword to be detected by the tool.

The tests are executed using Chromedriver [41]. As Chrome releases a new version almost every month, it is required that the latest Chromedriver is added to the tool. Otherwise, it might break the tests and provide incorrect results.

# 6. Conclusions and Future Work

The current thesis addresses the design and development of a test management tool at Rocketlab OÜ. It aims to provide an overview of automated tests and projects and a unified location to keep track of test results. In addition, it offers inexperienced testers the possibility to execute test pipelines and obtain results. The tool facilitates a singular location to plan, execute, schedule, and report the tests as requested by the client. It consolidates the reporting of past and failed test cases, suites, and pipelines that are essential for developing better applications with time. Using software development tools like React, FastAPI as a framework, an extensive frontend has been developed within the premise of this work.

The tool was developed considering the project deadlines and the internal requirements of the company. It was presented as a web-based application and after further discussions with colleagues and the CEO, the tool was verified and has been accepted within Rocketlab. The tool is aspired not only to ease the use of testers and test automation engineers but also to provide transparent feedback to the clients along with mitigating human errors regarding the requested tests. The tool also serves as a platform to expand and grow within the TaaS community. Currently, the tool is in the process of being integrated with previous test pipelines and including new test cases within the projects.

In the future it is planned to expand the tool with additional features:

- Add support for Cucumber-based projects in Java.
- Add additional graphs in the dashboard.
- Add support for manual tests.
- Link users to tests and tasks.
- Add version control to test cases, suites, pipelines.
- Implement the feedback from the colleague

As discussions progress regarding the company and client requirements, additional requirements will be sighted.

The tool is looked forward to being implemented as a unified solution for executing all future test pipelines from the client. It is envisioned to be used across the company by all employees to execute project tests and eventually be commercialized and marketed as a test management tool within the software testing industry.

## Acknowledgments

The author would like to acknowledge Rocketlab OÜ that provided the opportunity and the resources for the development of this project.

This project was carried out within the Bachelors' curriculum offered by the institute of Computer Science at the University of Tartu. The author expresses her gratitude to her supervisors Mr. Armin Haller, Rocketlab OÜ, and prof. Dietmar Alfred Paul Kurt Pfahl, University of Tartu, whose constant guidance and direction resulted in Q.tool and the compilation of this thesis.

The author also would like to thank her family and friends who have been a constant help and support throughout this journey.

# 7. References


[1] Perry WE. Effective methods for software testing. 3rd ed. Indianapolis, IN: Wiley; 2006. 973 p.

[2] Parveen T, Tilley S, Gonzalez G. A case study in test management. In: Proceedings of the 45th annual southeast regional conference [Online]. New York, NY, USA: Association for Computing Machinery; 2007 [cited 2021 Mar 28]. p. 82–7. (ACM-SE 45). Available from: http://doi.org/10.1145/1233341.1233357

[3] Extent Reporting Framework [Online]. [cited 2021 Apr 20]. Available from: https://www.extentreports.com/

[4] Trivedi SH. Software Testing Techniques. Int J Adv Res Comput Sci Softw Eng. 2012 Oct;2(10):7.

[5] Moussa S, Kandil P, Badr N. A Study for Regression Testing Techniques and Tools. Int J Soft Comput Softw Eng JSCSE. 2015 Jan 1;5:64–84.

[6] Overview of Docker Compose [Online]. Docker Documentation. 2021 [cited 2021 Apr 20]. Available from: https://docs.docker.com/compose/

[7] rkt Topic Page [Online]. [cited 2021 May 6]. Available from: https://www.openshift.com/learn/topics/rkt

[8] Linux Containers [Online]. [cited 2021 May 6]. Available from: https://linuxcontainers.org/

[9] scooley. Introduction to Hyper-V on Windows 10 [Online]. [cited 2021 May 6]. Available from: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/

[10] React – A JavaScript library for building user interfaces [Online]. [cited 2021 Apr 20]. Available from: https://reactjs.org/

[11] Angular [Online]. [cited 2021 May 6]. Available from: https://angular.io/

[12] Vue.js [Online]. [cited 2021 May 6]. Available from: https://vuejs.org/

[13] CherryPy — A Minimalist Python Web Framework [Online]. [cited 2021 May 6]. Available from: https://cherrypy.org/

[14] FastAPI [Online]. [cited 2021 Apr 20]. Available from: https://fastapi.tiangolo.com/

[15] Falcon | The minimal, fast, and secure web framework for Python [Online]. [cited 2021 May 6]. Available from: https://falconframework.org/

[16] Bottle: Python Web Framework — Bottle 0.13-dev documentation [Online]. [cited 2021 May 6]. Available from: http://bottlepy.org/docs/dev/

[17] Welcome to Flask — Flask Documentation (1.1.x) [Online]. [cited 2021 May 6]. Available from: https://flask.palletsprojects.com/en/1.1.x/

[18] The Web framework for perfectionists with deadlines | Django [Online]. [cited 2021 May 6]. Available from: https://www.djangoproject.com/

[19] Choosing a Database for Your Web Application [Online]. Future Hosting. 2019 [cited 2021 May 6]. Available from: https://www.futurehosting.com/blog/choosing-a-database-for-your-web-application/

[20] Advanced Python Scheduler — APScheduler 3.7.0 documentation [Online]. [cited 2021 May 6]. Available from: https://apscheduler.readthedocs.io/en/stable/

[21] Owens M. python-crontab: Python Crontab API [Online]. [cited 2021 May 6]. Available from: https://gitlab.com/doctormo/python-crontab/

[22] Celery - Distributed Task Queue — Celery 5.0.5 documentation [Online]. [cited 2021 Apr 20]. Available from: https://docs.celeryproject.org/en/stable/#

[23] cookiecutter/cookiecutter [Online]. cookiecutter; 2021 [cited 2021 May 3]. Available from: https://github.com/cookiecutter/cookiecutter

[24] NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy [Online]. NGINX. [cited 2021 Apr 20]. Available from: https://www.nginx.com/

[25] OAuth 2.0 — OAuth [Online]. [cited 2021 Apr 20]. Available from: https://oauth.net/2/

[26] Prettier · Opinionated Code Formatter [Online]. [cited 2021 Apr 20]. Available from: https://prettier.io/index.html

[27] ESLint - Pluggable JavaScript linter [Online]. ESLint - Pluggable JavaScript linter. [cited 2021 Apr 20]. Available from: /

[28] Material-UI: A popular React UI framework [Online]. [cited 2021 Apr 20]. Available from: https://material-ui.com/

[29] Redux - A predictable state container for JavaScript apps. | Redux [Online]. [cited 2021 May 6]. Available from: https://redux.js.org/

[30] Group PGD. PostgreSQL [Online]. PostgreSQL. 2021 [cited 2021 Apr 20]. Available from: https://www.postgresql.org/

[31] SQLAlchemy - The Database Toolkit for Python [Online]. [cited 2021 Apr 20]. Available from: https://www.sqlalchemy.org/

[32] Welcome to Alembic’s documentation! — Alembic 1.5.8 documentation [Online]. [cited 2021 Apr 20]. Available from: https://alembic.sqlalchemy.org/en/latest/

[33] Flower - Celery monitoring tool — Flower 1.0.0 documentation [Online]. [cited 2021 Apr 20]. Available from: https://flower.readthedocs.io/en/latest/

[34] Asynchronous Tasks in Django with Redis and Celery [Online]. Stack Abuse. [cited 2021 Apr 26]. Available from: https://stackabuse.com/asynchronous-tasks-in-django-with-redis-and-celery/

[35] Abud G. Buuntu/fastapi-react [Online]. 2021 [cited 2021 Apr 20]. Available from: https://github.com/Buuntu/fastapi-react

[36] pytest: helps you write better programs — pytest documentation [Online]. [cited 2021 Apr 20]. Available from: https://docs.pytest.org/en/latest/

[37] marmelab/react-admin [Online]. marmelab; 2021 [cited 2021 Apr 20]. Available from: https://github.com/marmelab/react-admin

[38] Postman | The Collaboration Platform for API Development [Online]. Postman. [cited 2021 May 7]. Available from: https://www.postman.com/

[39] The Internet [Online]. [cited 2021 May 3]. Available from: http://the-internet.herokuapp.com/

[40] Maven – Welcome to Apache Maven [Online]. [cited 2021 May 2]. Available from: https://maven.apache.org/

[41] ChromeDriver - WebDriver for Chrome [Online]. [cited 2021 May 2]. Available from: https://chromedriver.chromium.org/home

# Appendix

## I. User manual

The web application is designed in 4 main parts: dashboard, planning, execution, and reporting. (Figure 16)

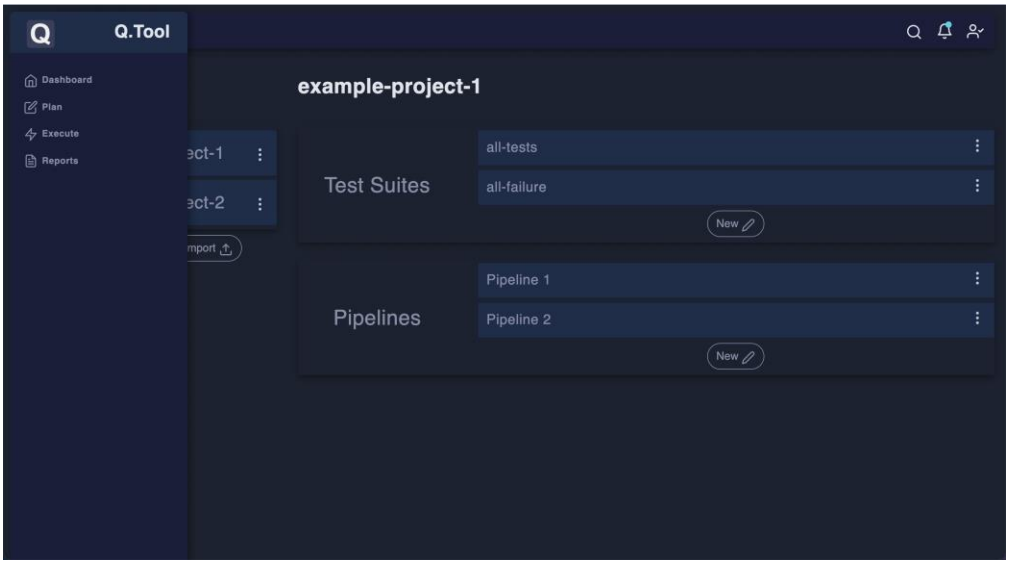


Figure 16. The navigational panel

The dashboard is the first section in the navigational panel (Figure 16). As shown in Figures 17 and 18, currently 3 graphs are shown. The pie chart shows passed and failed test cases. The line graph shows the amount of time taken by the number of test cases, test suites, or pipelines. It is plotted for the time in minutes against the number of test cases in a particular period. This way it is possible to see a possible execution time for any element of collection if it has been executed at least once. The third graph shows the reason of failures and how often they happen. This data is gathered by the test engineer manually assigning the issue that was present if a test case failed. In the report section, it is possible to choose a reason with a drop-down for each failed test case.


Figure 17. The dashboard displaying graphs ‘Passed/Failed Test Cases’ and ‘Time Taken’ where on the left side there is a filter bar

On the left side, a filter menu is shown. It is possible to filter out the data that is shown with the graphs. At first, only projects are shown. Once a project is selected then all pipelines under the project are displayed. The same logic is applied when we choose a pipeline and afterward the test suite.

For viewing the number of passed and failed test cases on the graph, the filtration works as follows. When opening a dashboard view, the first project is always selected. All the test case data from all executed pipelines is shown. After choosing a pipeline, the test cases only for this particular pipeline are shown. It is possible to narrow down the data to a particular test suite and test case. It was chosen to show passed and failed status for test cases under a particular collection element, otherwise the data is not displayed correctly. If the passed or failed status is shown for a pipeline, then the following logic applies. All of the test suites have to pass for a pipeline to have a passed status. In case even one test suite fails, the entire pipeline, fails. In case such a strategy is applied, it would introduce an inconsistency in the data which would not depict the real situation.

To display the time taken by a particular element of the collection, the logic has been discussed further. When choosing a project, all of its pipelines executed time is displayed. When choosing a particular element of the collection, all previous execution times for this particular pipeline are shown. The same applies to test suites and test cases.

The third graph showing the reason of failure for a test case has a similar logic implemented as the chart showing the number of passed and failed test cases. The reason of a failure is applied to a particular test case, hence when filtering out the data for test cases under the chosen collection element is shown.

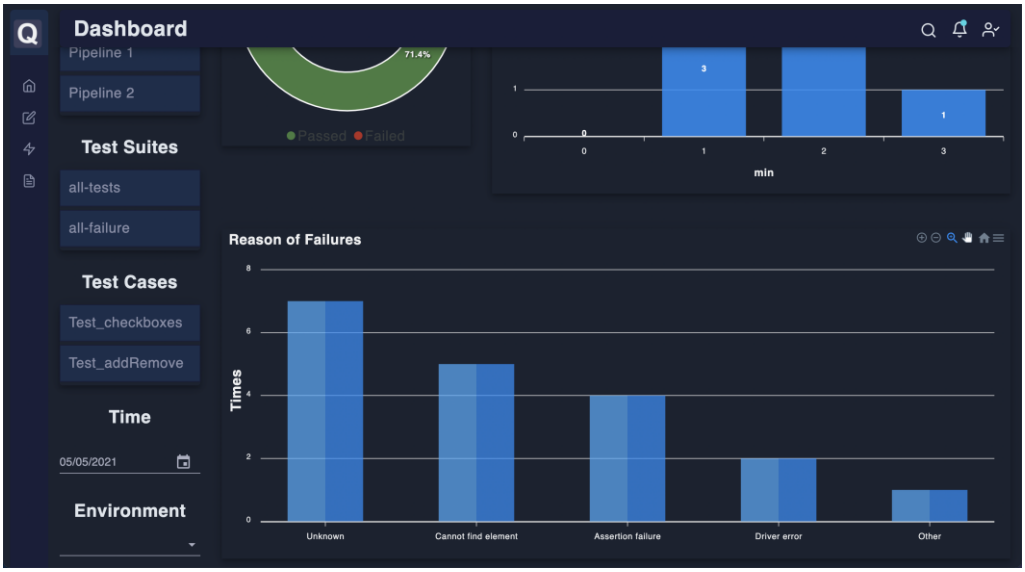


Figure 18. The dashboard displaying graph ‘Reason of Failures’ and a filter bar

The next element in the side menu is the planning section. As shown in Figure 20, the planning view provides the possibility to add new, edit or delete projects, pipelines, test suites, and test cases. To add or edit any of the collection elements, a modal window is shown. As shown in Figure 21, it is possible to add or edit its name and description. As shown in Figure 19, it is possible to get info, edit or delete any of the collection element. When choosing a project, then pipelines and test suites that belong to the current project are shown.

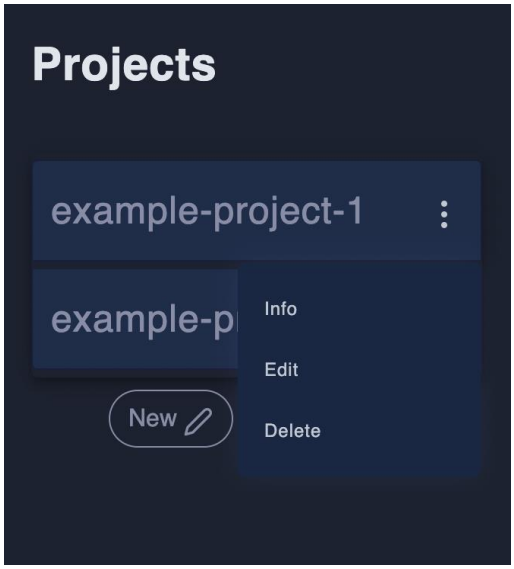


Figure 19. The submenu for a project

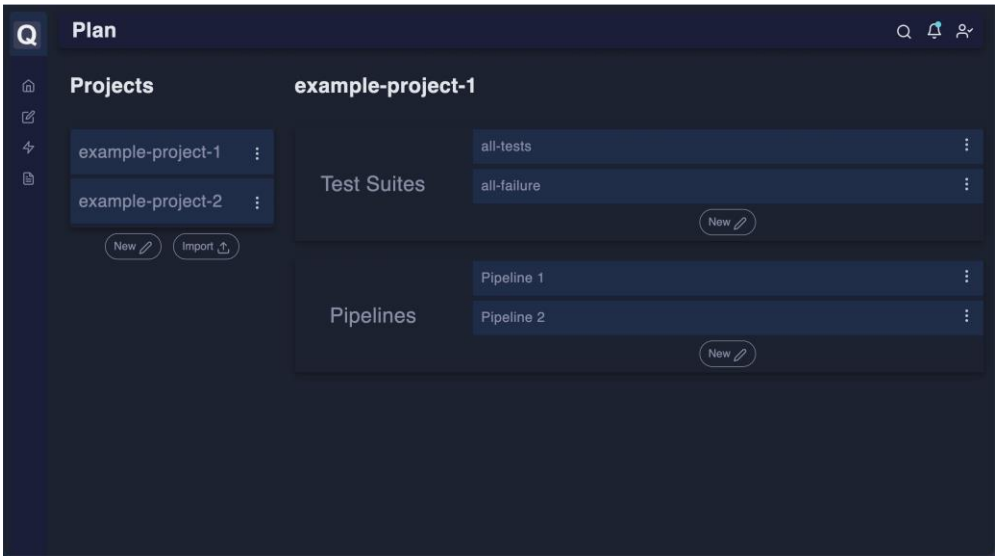


Figure 20. The planning view

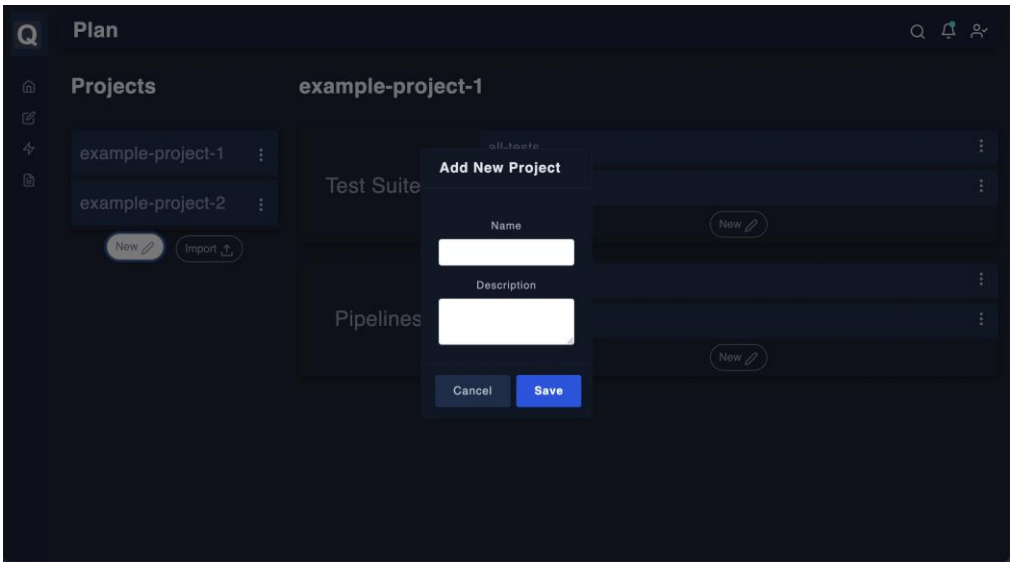


Figure 21. The modal window when adding or editing a collection element

In Figure 21, the test cases which belong to a certain test suite are shown. The details such as tester, creation date, type, tags, test case status about each test case are shown in the table. When making a new test case or editing an existing one, it is possible to add test steps for the test automation engineer to implement. It will be shown as comments under the test case code. Under the 3 dots, a dropdown menu with 3 options – info, edit and delete – will appear.

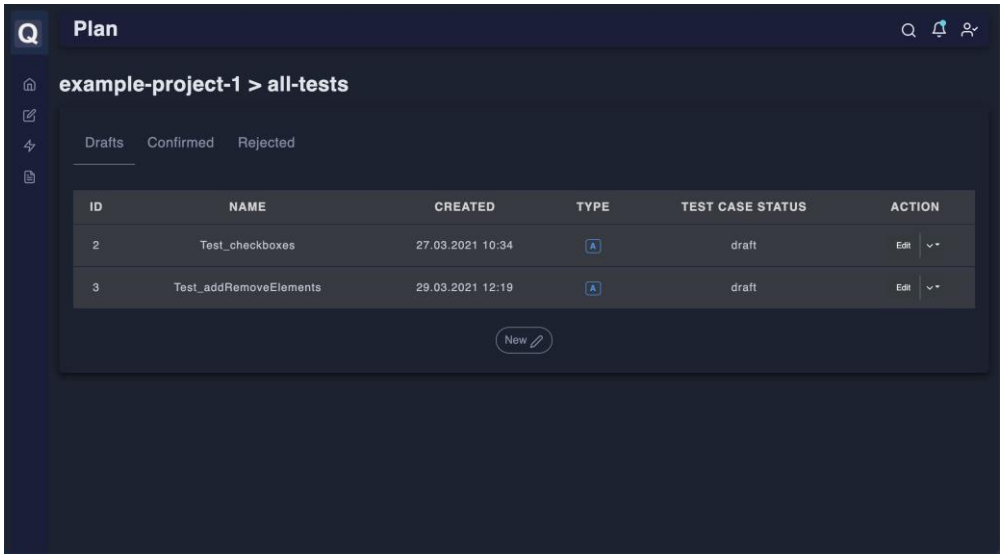


Figure 22. The planning view where test cases under a test suite are shown

In addition, it is possible to import an existing project. This way the application is going to detect already existing test suites and pipelines and add them to the project. The import button is visible in Figure 20.

Under the next section, the execution view is visible. Here it is possible to schedule pipelines and see the status of scheduled and finished pipelines.

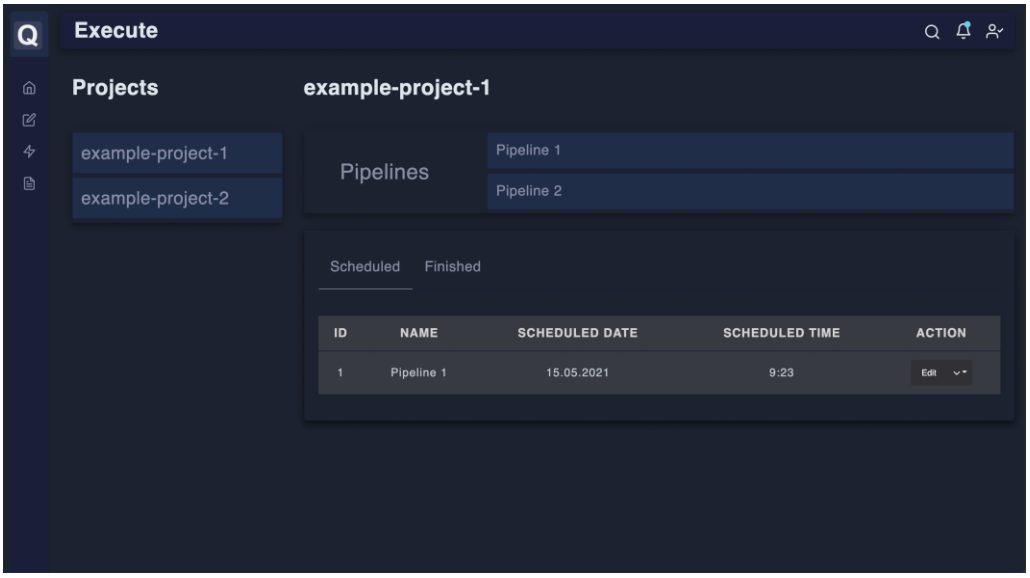


Figure 23. The execution view (scheduled view)

When choosing a project, then only planned pipelines are shown in this view. When clicking on a pipeline, it is possible to schedule it. In Figure 24, a modal window is shown which appears after clicking on a pipeline. By using the simple scheduling tab, it is possible to schedule the pipeline to run once at a particular date and time. In Figure 25, when choosing the custom option, it is possible to schedule a pipeline to run repeatedly.

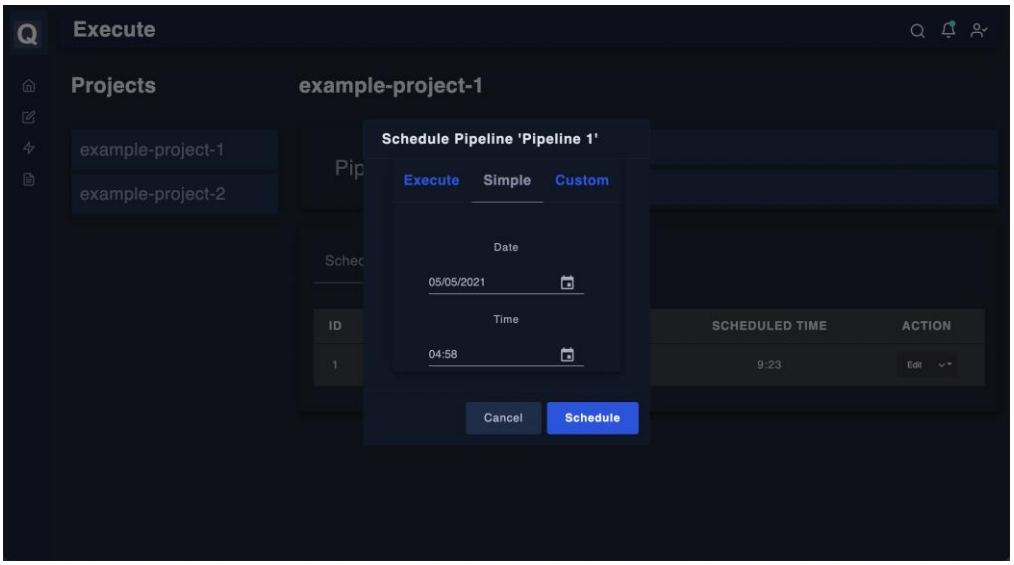


Figure 24. Schedule a pipeline modal window (simple option)

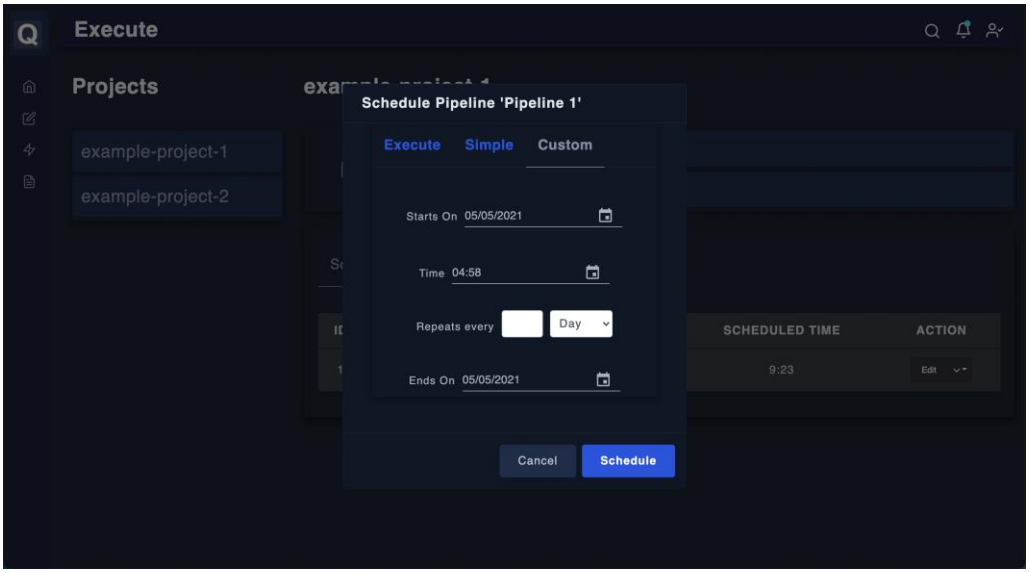


Figure 25. Schedule a pipeline modal window (custom option)

Once a pipeline is scheduled, then it appears in the scheduled pipeline tab as shown in Figure 23. The scheduled time and date are shown in the table. It is possible to edit and cancel the pipeline execution. As seen in Figure 27, once a pipeline is running or finished run, then it is added to the finished pipeline tab. There only the last 10 finished pipelines with their details are shown. The details include is name finished date and time, and status. Once open is clicked, it goes to the report view where the report for this particular pipeline execution is shown.

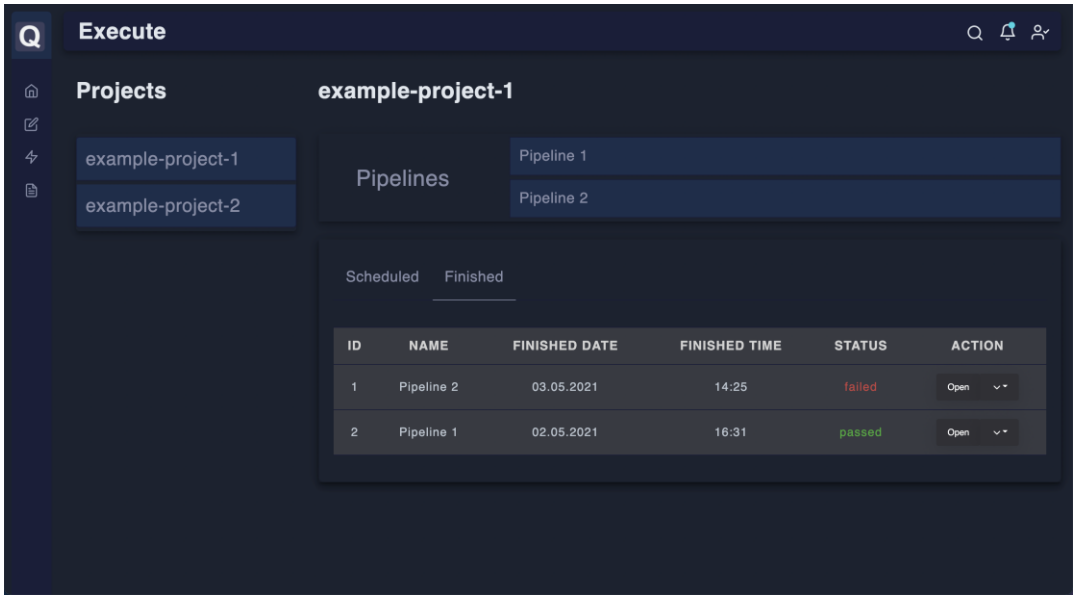


Figure 27. The execution view (finished view)

The last part is the reporting view. Here it is possible to choose a project and all of the executed pipelines are displayed. The details such as id, name, finished date and time, and status are shown in the table. Once the open button is pressed, a detailed view of the report for a particular pipeline is shown, as shown in Figure 28.

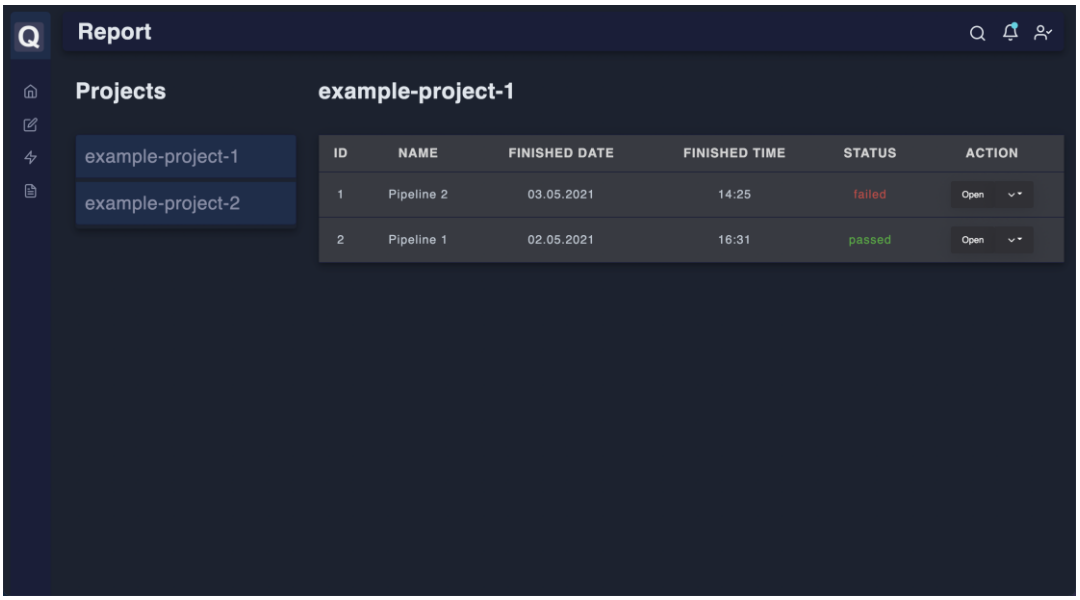


Figure 28. Reporting view

In the detailed report view (Figure 29), it is possible to see all the test suites executed. Details such as pipeline name, start and end time, time taken, and status are shown in the table. Once clicking on a test suite, a table with its test cases is displayed. The details such as test case ID and name, start and end time, and status are shown. It is possible to click open and it opens a detailed HTML report regarding the particular test case. This report file is produced from the test run and attached to the test case.

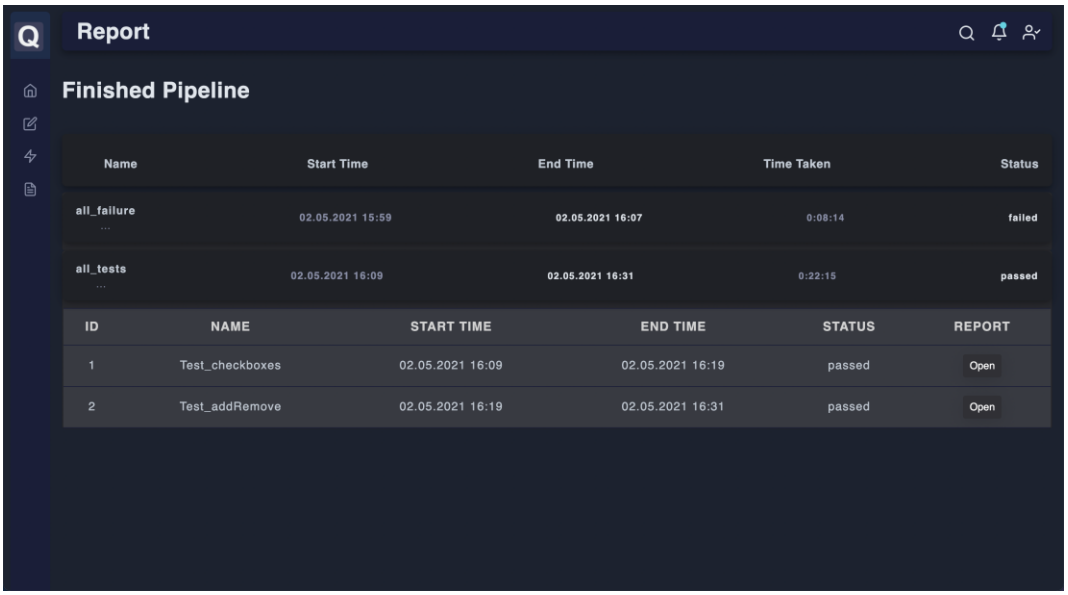


Figure 29. Detailed report view

## II. Manual Testing of the Web Application

| | Test Steps | Expected Results | Actual Results |
|---|---|---|---|
| 1. | Log in to the tool | Dashboard opens | Passed (Figure 30) |
| 2. | Go to planning view using the navigational panel on the left | Plan view opens | Passed (Figure 31) |
| 3. | Click on the add button (+) under projects | Add new project modal appears | Passed (Figure 32) |
| 4. | Insert name and description of the new project and click on the button "Add" | A new project appears under the list of projects | Passed (Figure 33) |
| 5. | Click on the add button (+) under test suites | Add new test suite modal appears | Passed |
| 6. | Insert name and description of the new test suite and click on the button "Add" | A new test suite appears under the list of projects | Passed (Figure 34) |
| 7. | Click on the just added test suite | Test case view opens | Passed (Figure 35) |
| 8. | Click on the add button (+) under the test case table | Add new test case modal appears | Passed (Figure 36) |
| 9. | Enter the details about the test case, mark the checkbox confirmed, and click the button "Add" | A new test case appears under confirmed test cases | Passed (Figure 37) |
| 10. | Go back to the planning view using the navigational panel on the left | Plan view opens | Passed (Figure 38) |
| 11. | Click on the add button (+) under pipelines | Add new pipeline modal appears | Passed |
| 12. | Insert name and description of the new pipeline and click on the button "Add" | A new pipeline appears under the list of projects | Passed (Figure 39) |
| 13. | Click on the pipeline made in step 12 | Assign test suites to pipeline modal appears | Passed (Figure 40) |
| 14. | Click on the checkbox with test suite name made in step 6 to assign test suite to a pipeline and click the button "Save" | The test suite is assigned to a pipeline | Passed |

Table 2. Test case 1 test steps. Plan a new project with a test suite, test case, and a pipeline


Figure 30. Dashboard view

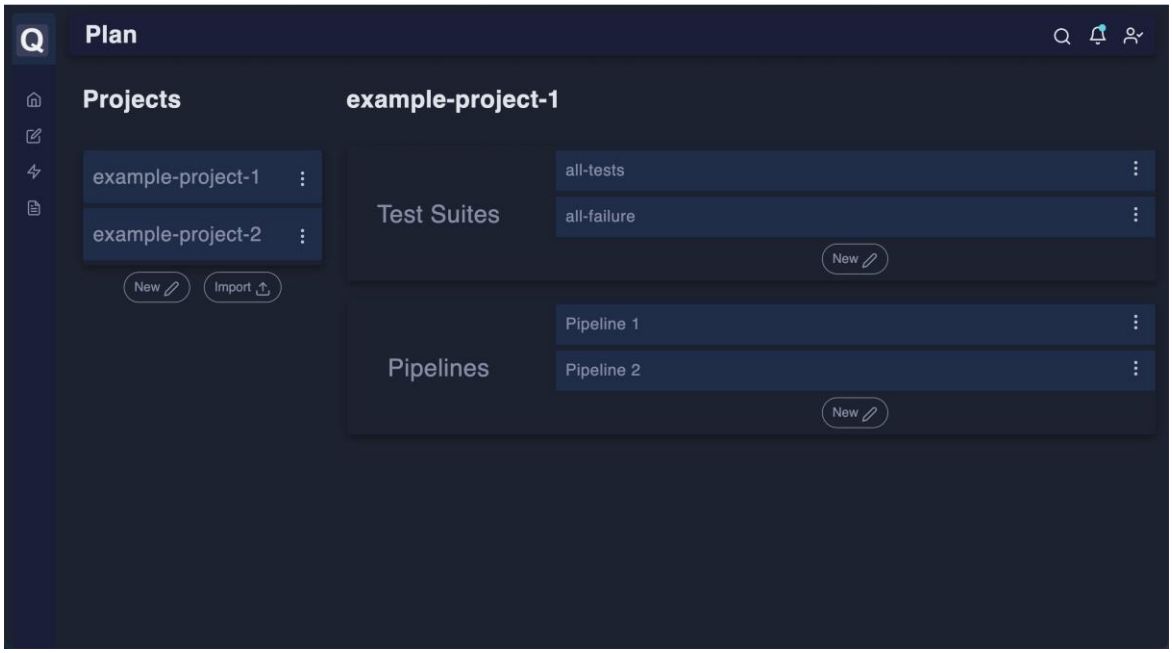


Figure 31. Plan view

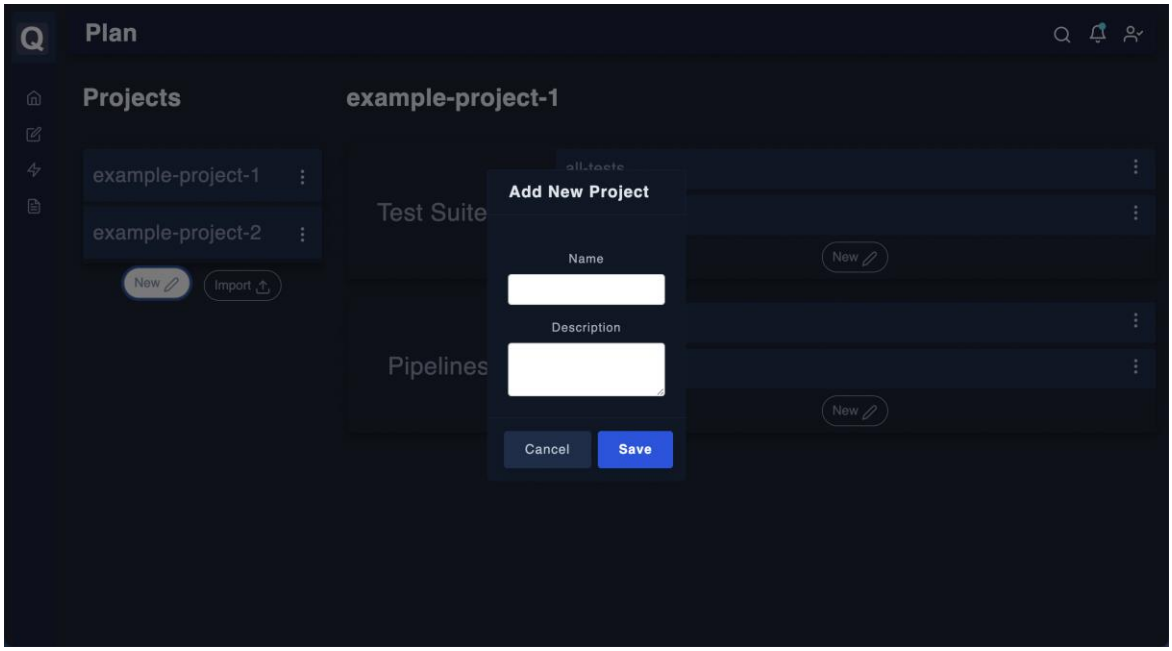


Figure 32. Add new project modal window

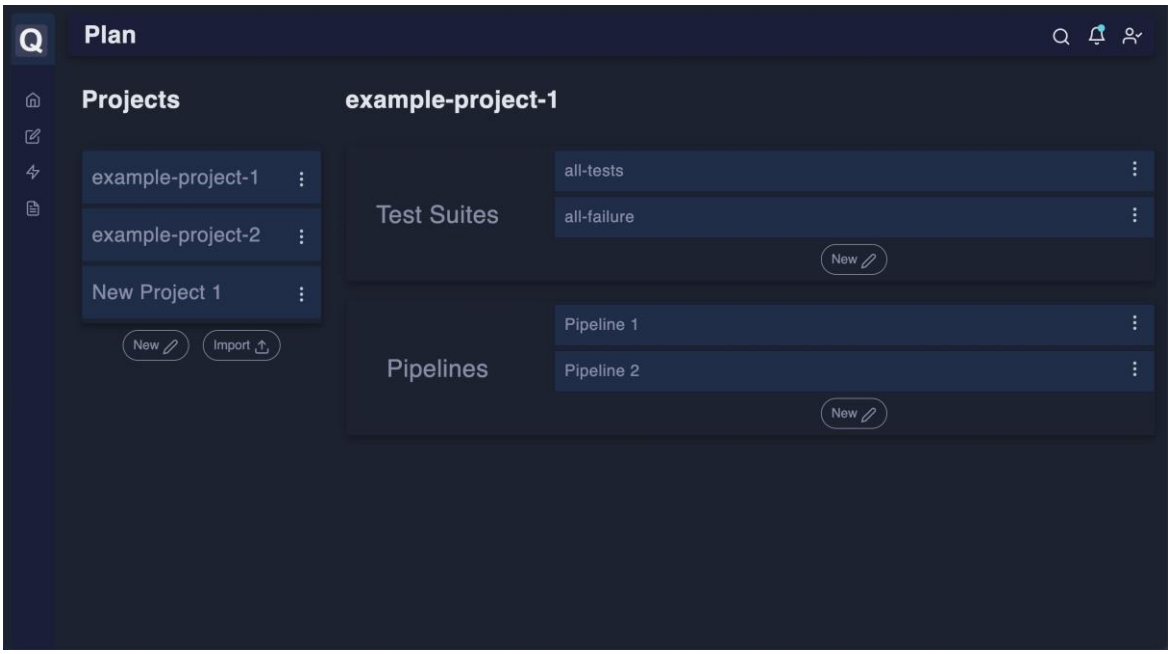


Figure 33. Plan view where a project named ‘New Project 1’ is shown

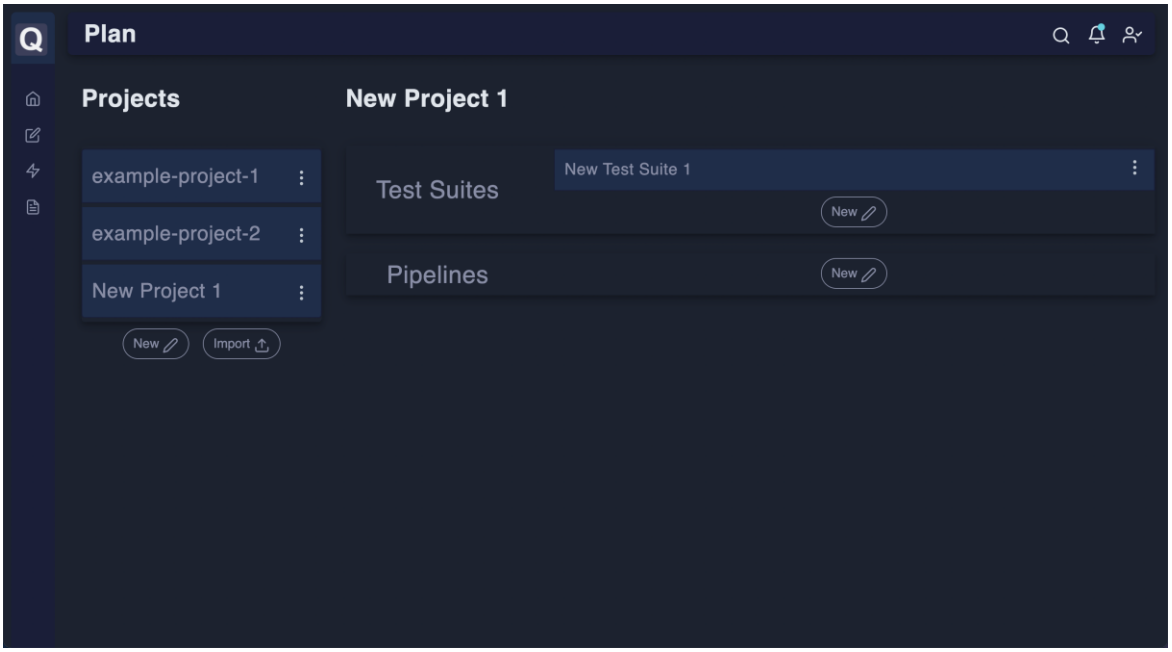


Figure 34. Plan view where a test suite named ‘New Test Suite 1’ is shown

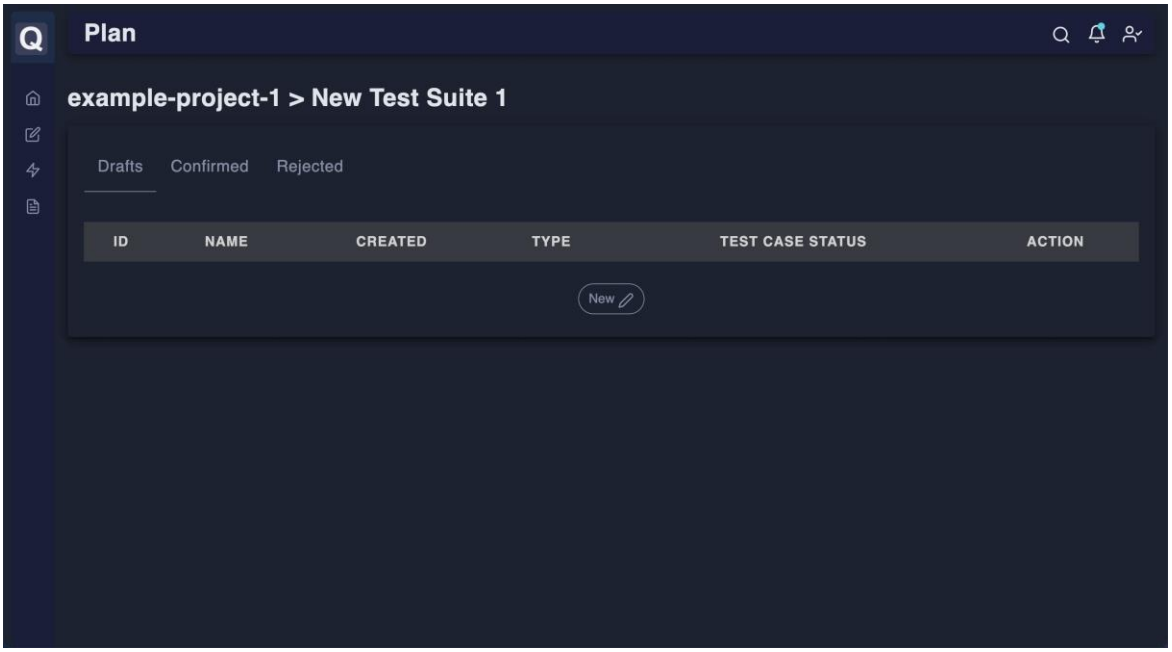


Figure 35. Test case window for a test suite named ‘New Test Suite 1’

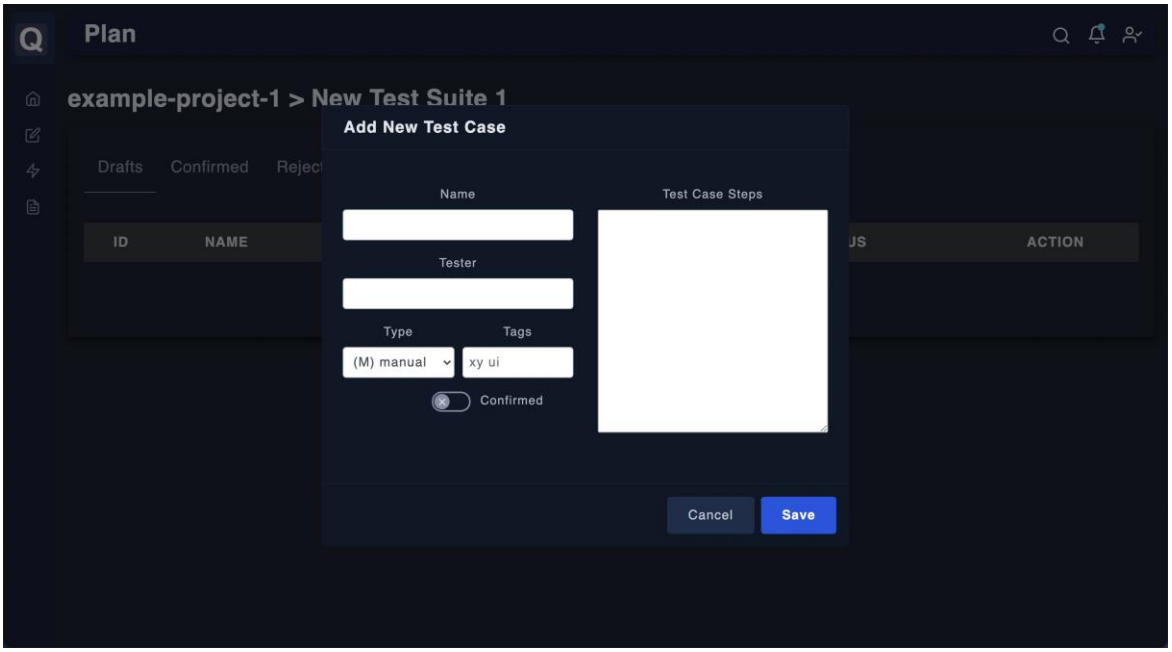


Figure 36. Add New Test Case Modal window

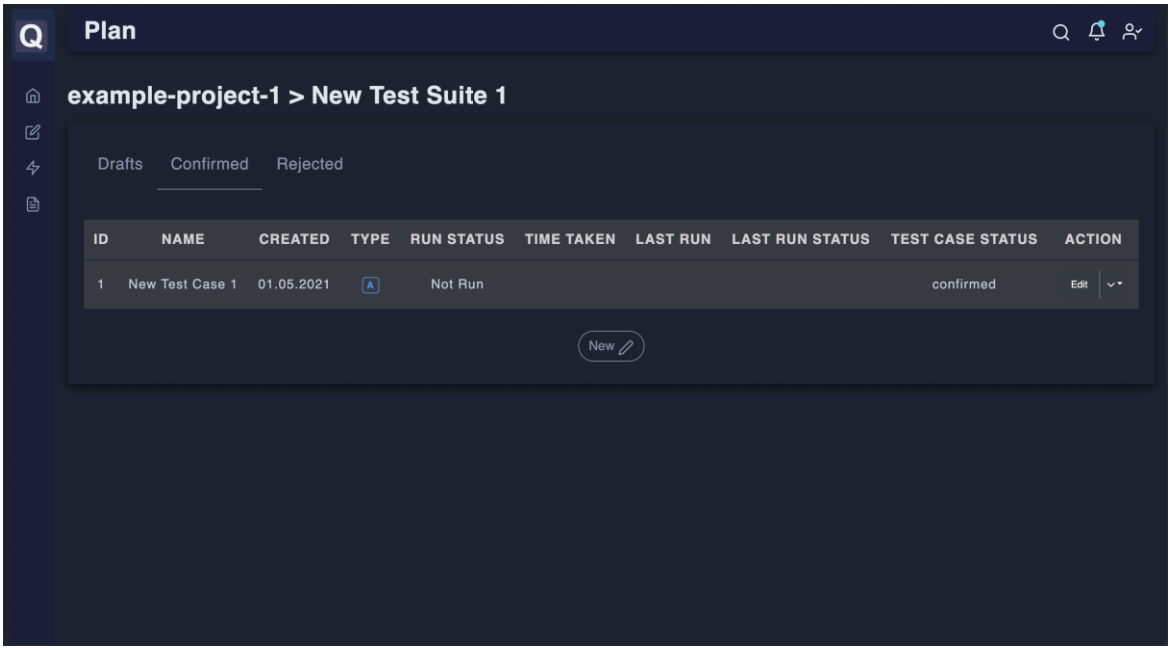


Figure 37. Test Cases within the test suite named ‘New Test Suite 1’

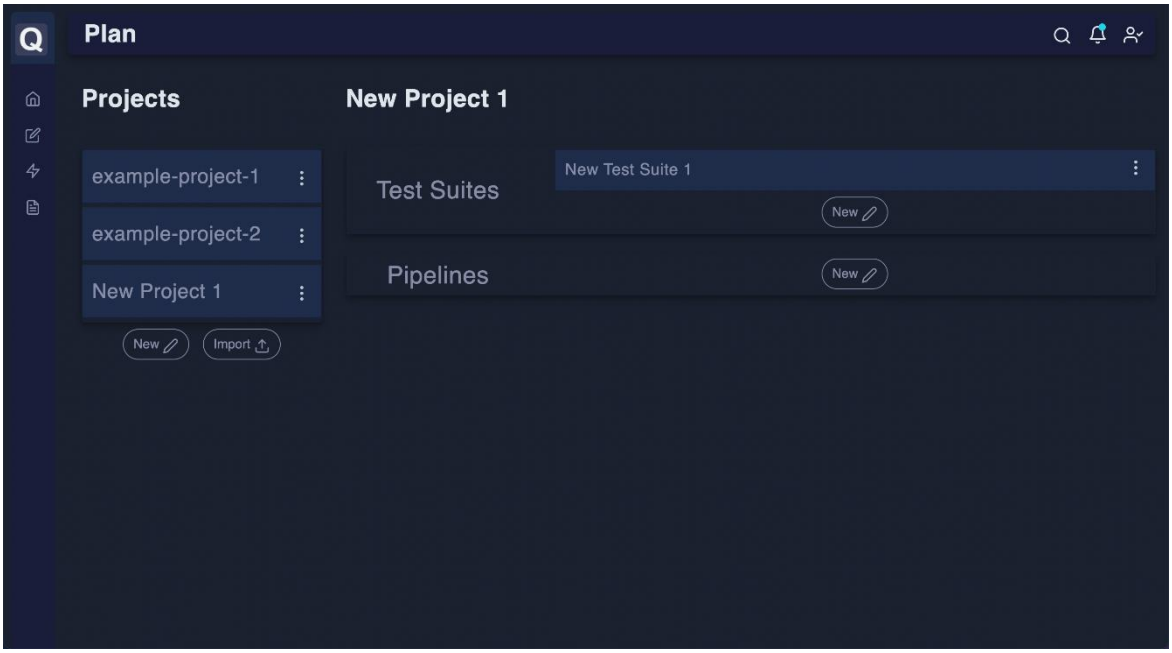


Figure 38. Plan view elements of collection for a project named ‘New Project 1’ are shown

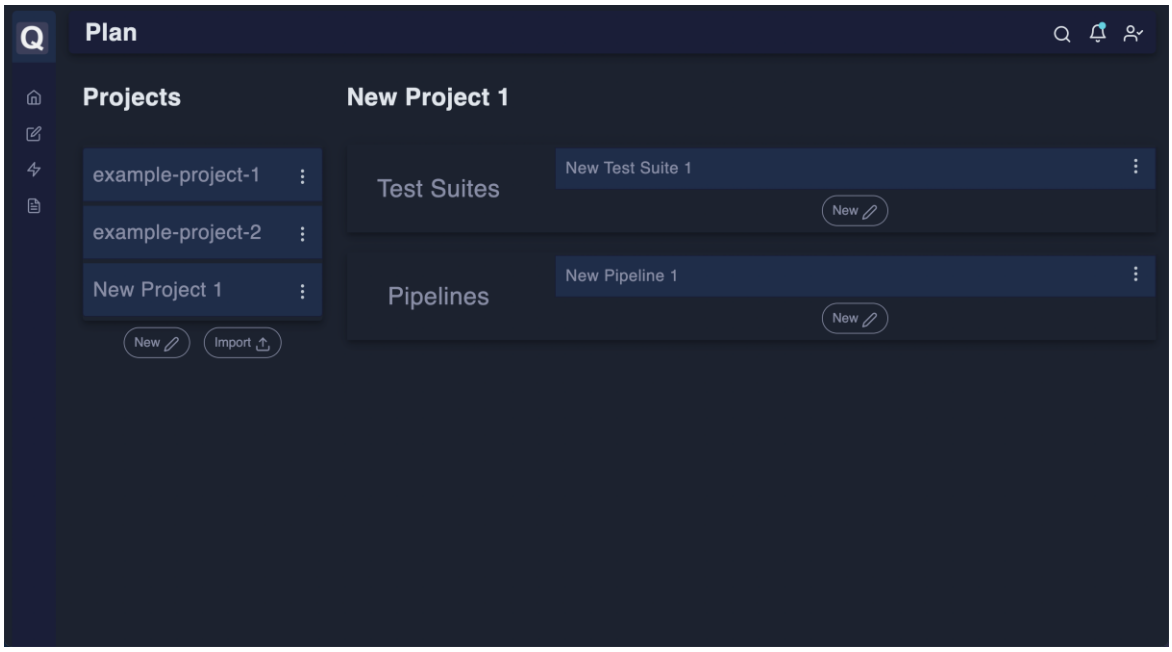


Figure 39. New pipeline named ‘New Pipeline 1’ is shown

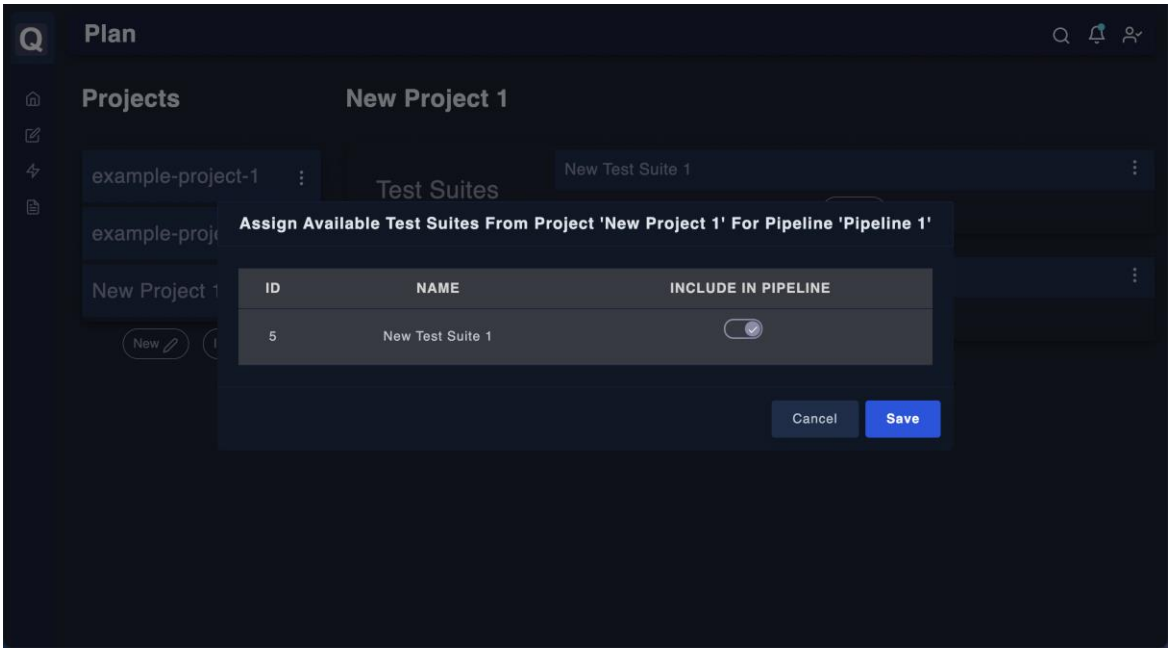


Figure 40. Assign Available Test Suites From Project ‘New Project 1’ For Pipeline ‘Pipeline 1’ are shown

| | Test Steps | Expected Results | Actual Results |
|---|---|---|---|
| 1. | Log in to the tool | Dashboard opens | Passed (Figure 41) |
| 2. | Go to planning view using the navigational panel on the left | Plan view opens | Passed (Figure 42) |
| 3. | Press the button “Import Project” | The project is imported and shown in the projects list | Passed (Figure 43) |
| 4. | Make a new pipeline and assign one of the test suites to it and click the button “Save” | A test suite is assigned to a pipeline | Passed (Figure 44) |
| 6. | Go to execute view using the navigational panel on the left and select the pipeline made in step 5 | Schedule a pipeline modal appears | Passed (Figure 45) |
| 7. | Schedule a pipeline to execute immediately | Modal closes and the pipeline appears in the scheduled tab | Passed |
| 8. | Wait for the pipeline to finish running | The pipeline appears in the finished tab | Passed (Figure 46) |

Table 3. Test case 2 test steps and results. Test execution and result acquisition.


Figure 41. Dashboard view

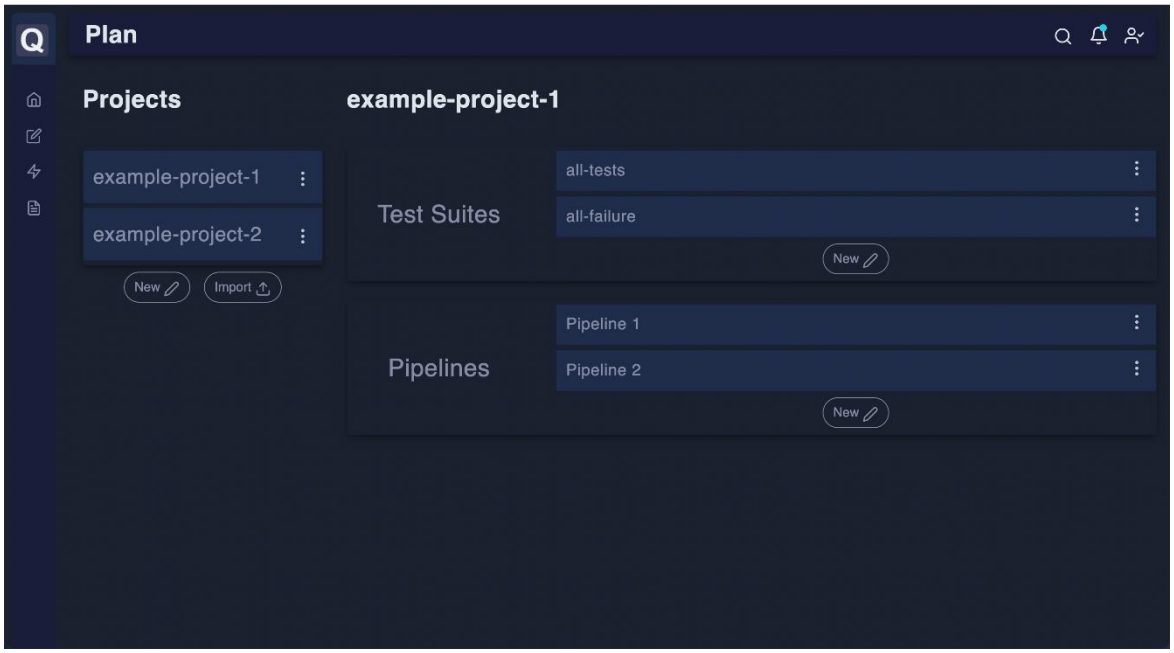


Figure 42. Plan view

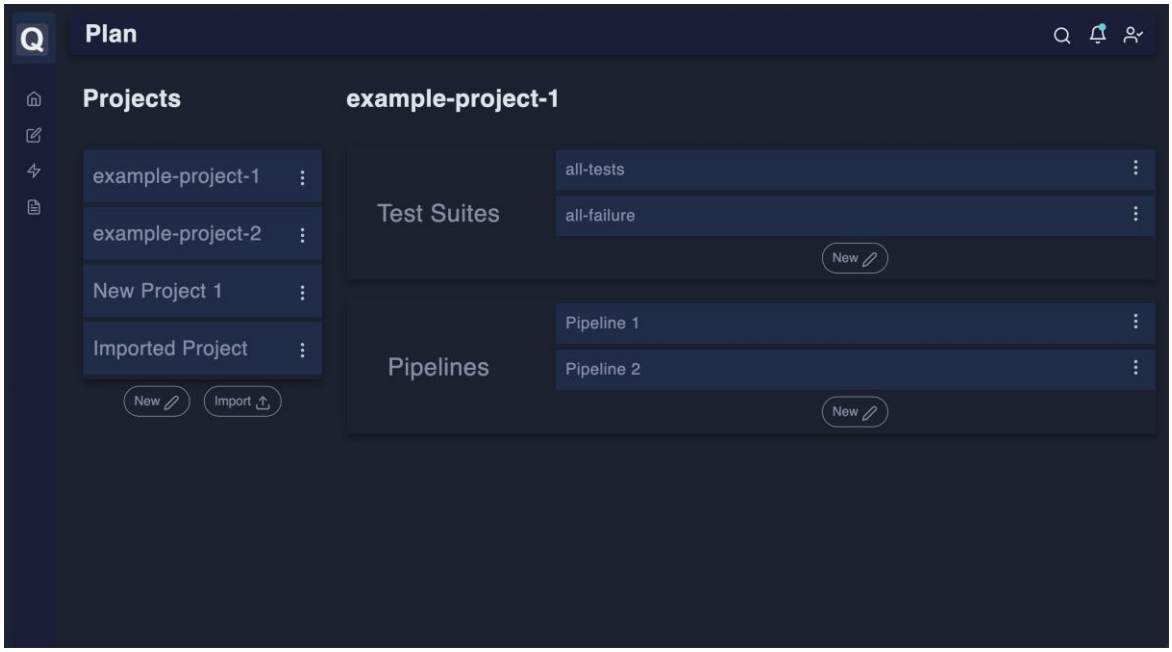


Figure 43. Plan view where new project ‘Imported project’ is added

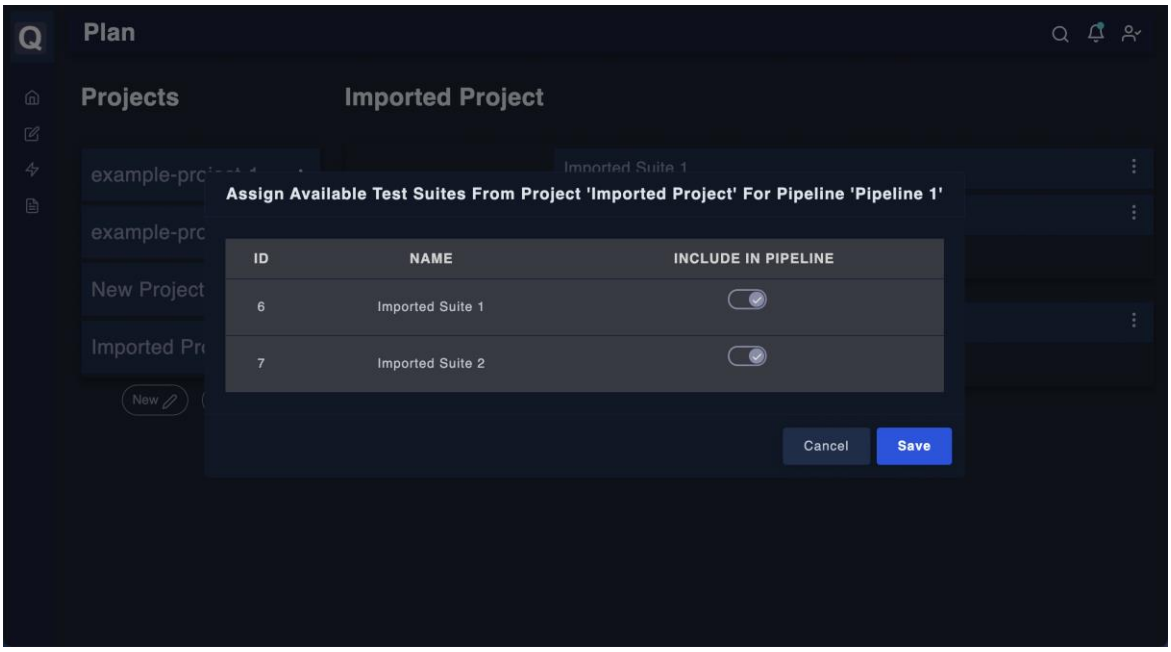


Figure 44. Assign Available Test Suites From Project ‘Imported Project’ For Pipeline ‘Pipeline 1’ modal window is shown

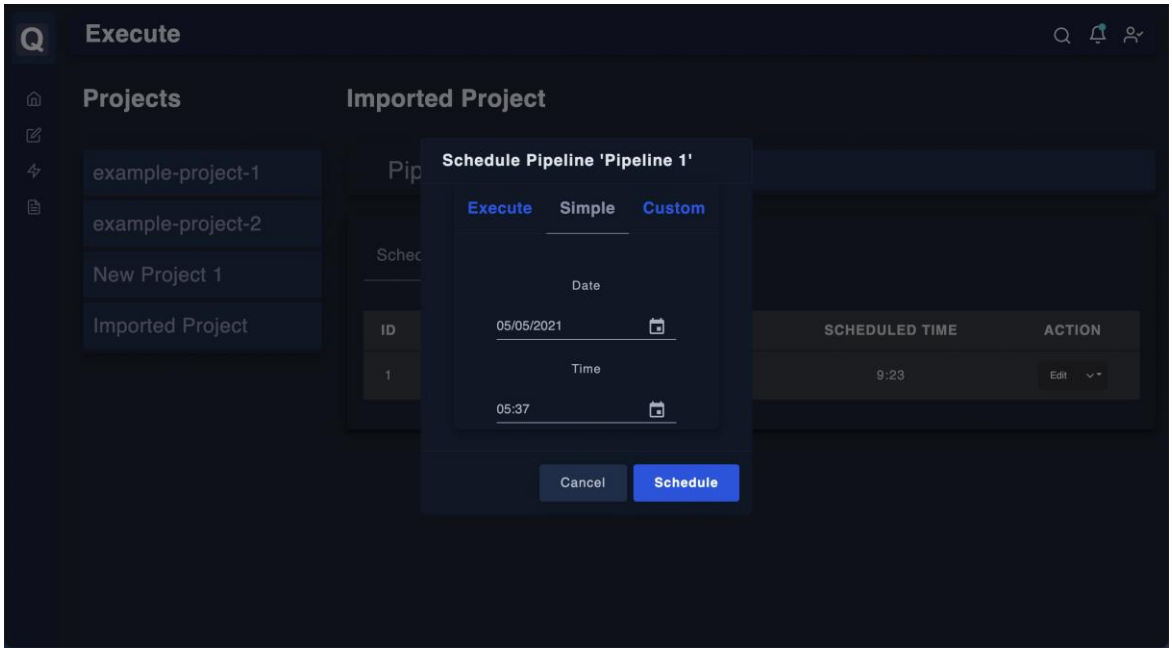


Figure 45. Schedule Pipeline ‘Pipeline 1’ modal window

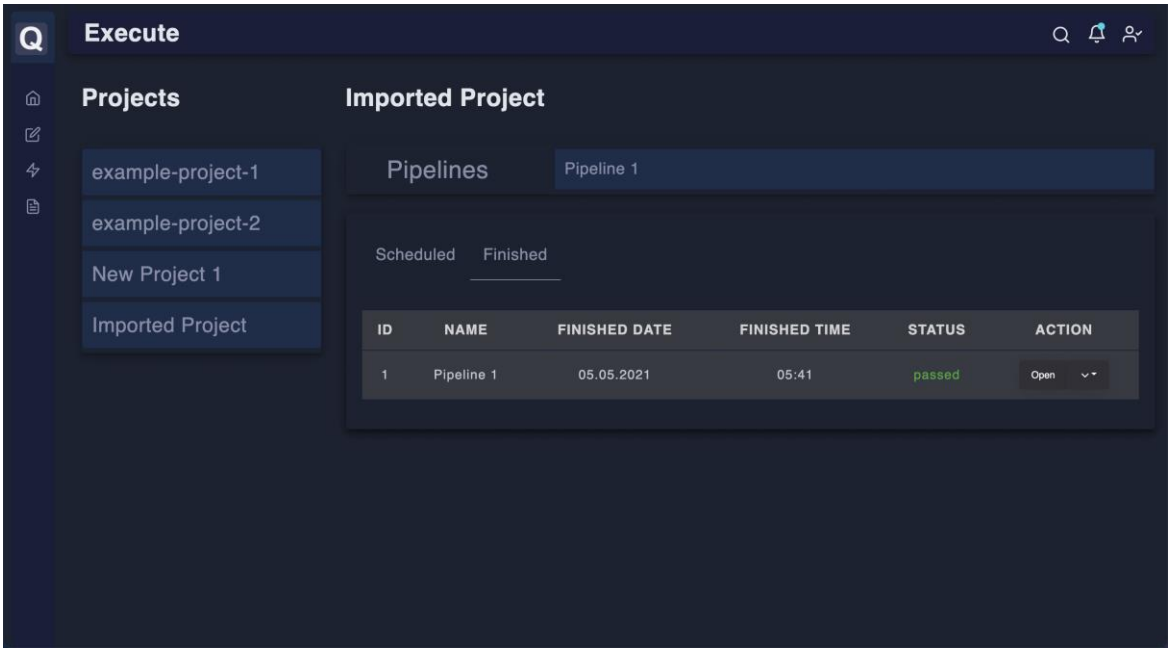


Figure 46. Finished pipeline ‘Pipeline 1’ shown in finished table

## III. License

## Non-exclusive licence to reproduce thesis and make thesis public

I, Laima Anna Dalbina,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, Design and Development of an Automated Website Test Management Tool at Rocketlab OÜ, supervised by Prof. Dietmar Alfred Paul Kurt Pfahl and Mr. Armin Haller.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Laima Anna Dalbina

07/05/2021