Architecture Final Report

*Team CAtS (Yihong Zou, Huairui Qi, Siqi Lu, Yun-Hsuan Chang)*

[**Context**](#_2cytppjtchil) **2**

[Introduction](#_o26szwesxbuc) 2

[Context Diagram](#_w3qapzk5lmg3) 3

[Stakeholders and Concerns](#_9d5h3j5rlyiv) 4

[Business Goals](#_53vsmn8hbpk6) 4

[**Architecture Drivers**](#_aou7i4gerrsw) **5**

[Functional Requirements](#_5jozvn2a4bn) 5

[Quality Attributes](#_zbwkhedm9ozw) 6

[Performance](#_1pbo8h4ipzxc) 6

[Modifiability](#_jim2homxpo1s) 7

[Constraints](#_5jozvn2a4bn) 9

[Technical Constraints](#_5jozvn2a4bn) 9

[Business Constraints](#_5jozvn2a4bn) 10

[**Dynamic Views**](#_vkmeua5iln7t) **10**

[Top Level Dynamic View](#_8718ftv3fpd0) 10

[Element Catalog](#_wvsj2ykhoen4) 11

[Rationale & Analysis](#_8ezksk8sncln) 12

[2nd Level Decompose Dynamic View](#_8ezksk8sncln) 13

[Element Catalog](#_9b76ufuswsq6) 13

[Rationale & Analysis](#_moyowi84ywlg) 14

[Sequence Diagrams](#_8ezksk8sncln) 15

[**Static Views**](#_w58gtfueyopr) **19**

[Top Level Static Module View](#_1sn5u8z3fpte) 19

[Element Catalog](#_lm2seu9ds6hd) 19

[Second Level Decompose Static Module View](#_fwhypjuq5xih) 20

[**Dependency Matrix**](#_qmw8lchvptul) **21**

[**Appendix**](#_x0texrnfu44) **21**

# Context

## Introduction

Continuous security assessments are becoming incredibly important during every phase of the **Software Development Lifecycle** (SDLC) to ensure application security. The basic idea is that whenever new code is committed and merged into the current repository, the system will automatically conduct security tests, either locally or globally, to ensure the security of the software application.

Since there does not exist any comprehensively and completely automated security evaluation service on either commercial or open source platforms, we are going to create a web-based **Continuous Authorization Service** (CAS) that automates the whole security testing process. The CAS system will implement the feature that users are able to customize their security testing preference, extract the user-focused security issues from multiple automated static testing tools, and automatically view the reports on their issue tracking system such as **Git**.

In addition to the CAS web service, a plugin which enables the integration of CAS and continuous integration (CI) tools will also be a part of the final deliverables. During the development stage, the primary CI tool to integrate will be **Jenkins**. In the future, CAS wishes to support more CI tools, such as **TeamCity, Travis CI, Bamboo**, etc.

## Context Diagram

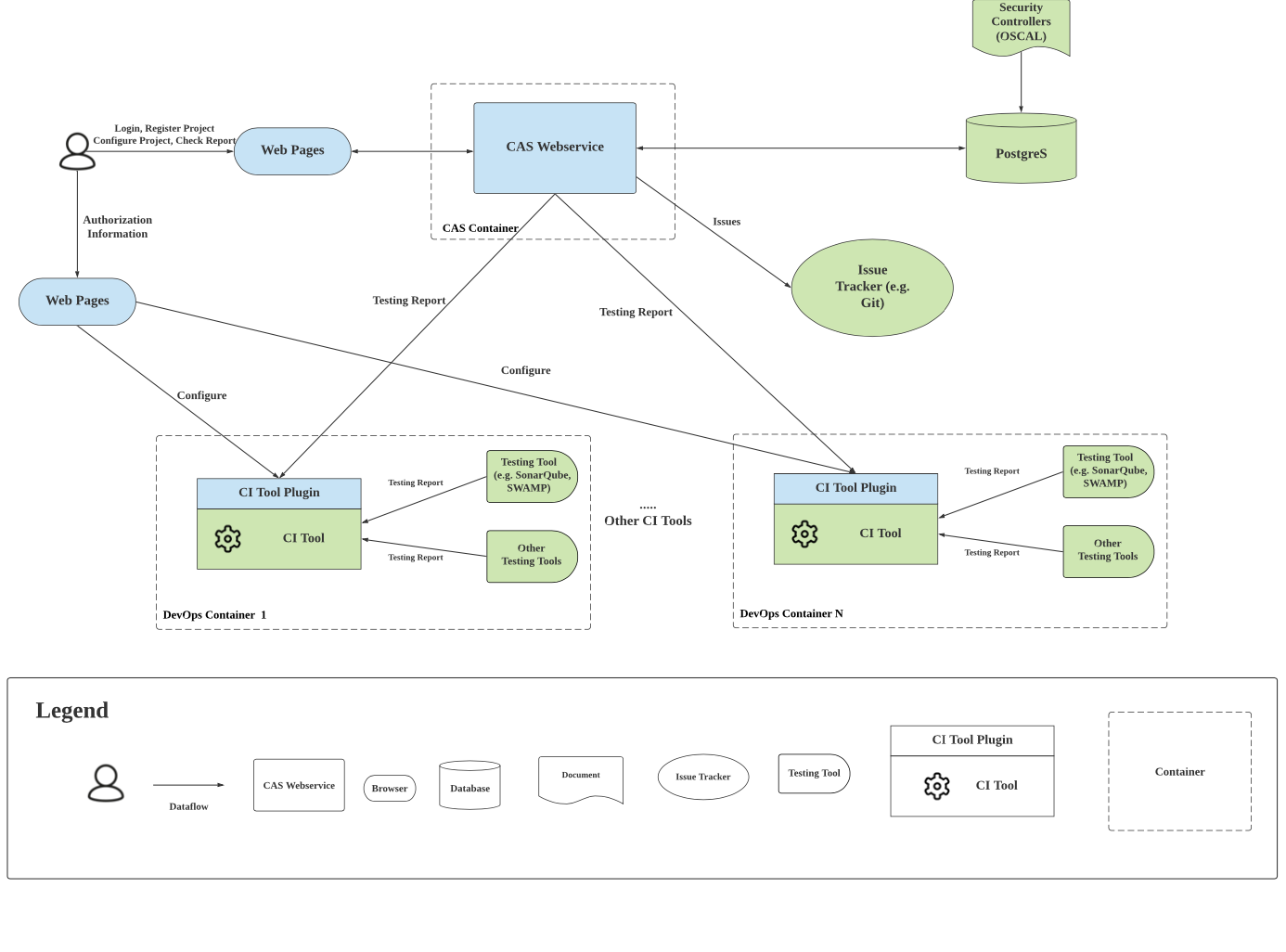


Figure 1. Context Diagram

Figure-1 shows the context diagram of CAS. Most security aspects can be checked by security controllers[[1]](#footnote-0) described by the Risk Management Framework (RMF). By specifying the types of controllers, users can specify which security aspects they want CAS to track from CI.

At first, users will have to register a project and configure their desired controller list for that project on CAS web service. Then users should configure CAS which has been installed as a plugin by the administrator in the user selected CI tools. That means the CI tools will establish connections with CAS through the form of a CI plugin, which will be referred to as CIPlugin (CIP). To interact with different CI tools, the CAS system should have different CIPs. Whenever the CI tool finishes building the project, the CIP will retrieve the testing report from CI that contains the test results of that build. CIP will then send the testing reports to CAS web service for further processing.

CAS web service will parse the testing reports with respect to user-specified controllers to collect the test results. CAS will maintain a table in the database to record these results of the selected controllers. Once the report parsing has been done, the database will be updated, so that the users will be able to see the latest status of the security assessment of their project and they can also choose to create issues for these security assessments with their desired Issue Tracker.

## Stakeholders and Concerns

We have 4 different stakeholders in total, and each of them have different requirements concerns.

The first stakeholder is **SEI CERT** as the project client. They want to create the first platform (both in context of open-source or commercial) that can provide continuous authorization service along the SDLC. As a nonprofit, public-private partnership that conducts research, SEI wants to better the infrastructure for the software development community to maintain their leadership in this field.

Our second stakeholder is the **software development community**. They care about the efficiency and reliability of CAS service. Using CAS, they are expecting to easily monitor the security aspects of their project through SDLC.

The third stakeholder is us as a **developing team**. Our primary concern will be our learning objectives. We want to practice software development management methodology, as well as our programming skills. We also want to use this project as our opportunity to join the open source community.

The last stakeholder is the **following developers** for this project. Their primary concern will be the maintainability and sustainability of the project.

## Business Goals

During the 2018/2019 Academic year, the SEI was fortunate to work with a group of MSIT Students to produce the “CATS” (Continuous Authorization as a Service) system, as well as create the corresponding deployment pipeline, infrastructure as code, and a custom plugin for the Jenkins Continuous Integration service.

The SEI would now like to expand and enhance the original v1.0 of the “CATS” system. This system is entirely open-source, and will be a great contribution and utilization of MITS skill sets to give back to the community.

# Architecture Drivers

## Functional Requirements

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Priority |
| FR-01 | The system shall be able to handle users credentials properly. | Critical |
| FR-01-1 | The system shall allow users to create an account through the website interface, setting their usernames and passwords. | Critical |
| FR-02 | The system shall allow users to manage their project repositories. | Critical |
| FR-02-1 | The system shall allow users to register or delete a project. | Critical |
| FR-02-2 | The system shall allow users to configure the controller with keywords. | Minor |
| FR-02-3 | The system shall allow users to share their projects with other users. | Critical |
| FR-03 | The system shall allow users to manage the continuous authorization process of their projects properly. | Critical |
| FR-03-1 | The system shall allow the users to view and download the generated testing report from dashboard. | Critical |
| FR-03-2 | The system shall be able to track the issues automatically for a certain project along with a user provided issue tracking system. | Critical |
| FR-03-3 | The system shall be able to parse reports in both csv and xml format. | major |
| FR-03-4 | The system shall be able to parse reports for each control of a project, with respect to the control’s keywords entered by users. | major |
| FR-04 | The system shall allow users to manage their user account settings | Major |
| FR-04-1 | The system shall allow users to change their profile settings. | Minor |
| FR-04-2 | The system shall allow users to change their credential settings. | Critical |
| FR-05 | The system shall allow an admin to control and manage the users, projects and controllers. | Critical |
| FR-06 | The system shall be able to install and configure CATS on Jenkins as a plugin. | Critical |

## 

## Quality Attributes

#### *Performance*

|  |  |
| --- | --- |
| ID | QA-P01 |
| Category | Performance |
| Priority | Medium |
| Source | CAS Web Service |
| Stimulus | Receives a new HTTP request from client |
| Artifact | CAS web service dashboard |
| Environment | CAS web service running with the normal condition (less than 50 concurrent users) |
| Response | The browser shows corresponding contents on the web page. |
| Response Measure | * Average response time should be less than 2 seconds. * The response time of 95% request should be less than 5 seconds. |

|  |  |
| --- | --- |
| ID | QA-P02 |
| Category | Performance |
| Priority | Medium |
| Source | Jenkins |
| Stimulus | A new testing result report generated by Jenkins. |
| Artifact | CAS web service dashboard |
| Environment | CAS web service running user’s jobs |
| Response | The dashboard shows the latest report of the result generated by Jenkins. |
| Response Measure | The report is updated on the dashboard within 2 minutes. |

#### *Modifiability*

|  |  |
| --- | --- |
| ID | QA-M1 |
| Category | Modifiability |
| Priority | High |
| Source | Developers in the open source community. |
| Stimulus | A directive to add new plugins for letting CAS compatible with other CI services (e.g. Travis). |
| Artifact | New plugin components |
| Environment | Other developers trying to modify CAS with new CI services. |
| Response | The new plugin component should be compatible with CAS and tested. |
| Response Measure | * The response from other developers in the open source community shows that the development time for additional CI plugins is acceptable (Within a week). * The only part shall be affected for adding additional CI plugin is the CI plugin itself. |

|  |  |
| --- | --- |
| ID | QA-A01 |
| Category | Availability |
| Priority | High |
| Source | CAS web service failure |
| Stimulus | The CAS web service is down |
| Artifact | Continuous Authorization Service (CAS) |
| Environment | CAS web service running user’s jobs |
| Response | The CAS service should restart itself. |
| Response Measure | * The availability percentage of CAS service should be more than 90% |

### 

|  |  |
| --- | --- |
| ID | QA-A02 |
| Category | Availability |
| Priority | High |
| Source | The CAS web service is down |
| Stimulus | CA service is down or is closed by the administrator. |
| Artifact | Jenkins plugin |
| Environment | Run time |
| Response | The testing reports should not be dropped. The reports should be saved in Jenkins\_home and be sent to CA service after it is back to work. |
| Response Measure | 1. All the reports during the CA service shutdown time should be saved. 2. The reports should be sent to CA in 1 minute after CA recovering. |

*Maintainability*

|  |  |
| --- | --- |
| ID | QA-M01 |
| Category | Maintainability |
| Priority | Medium |
| Source | Administration users |
| Stimulus | Administration users maintain the Jenkins server, plugins, and configurations. |
| Artifact | CAS Web Service, Jenkins |
| Environment | Design time |
| Response | The admin user should be able to localize and correct the errors as well as preserve the modification and extension of functionality. |
| Response Measure | 1. The debug process should be able to complete in one-person day. 2. Admin users should be able to extend or modify the functions of CAS without affecting the existing configurations |

*Usability*

|  |  |
| --- | --- |
| ID | QA - U01 |
| Category | Usability |
| Priority | High |
| Source | End users |
| Stimulus | Try to learn and configure the CAS Web service. |
| Artifact | Runtime |
| Environment | CAS Web service, Jenkins |
| Response | Users can use the CAS web service productively. |
| Response Measure | Finishing configuration task with 20 minutes. |

## Constraints

### Technical Constraints

|  |  |
| --- | --- |
| ID | Technical Constraint |
| TC01 | The system shall be compatible with Continuous Integration platforms, such as Jenkins. |
| TC02 | The CAS service shall be able to be deployed using Docker. |
| TC03 | The CAS service shall consume less than 8G memory. |
| TC04 | The system should be tested using Selenium and SonarQube. |
| TC05 | The CAS service should be written in Python. |
| TC06 | The CAS service should use the Django framework. |
| TC07 | The Jenkins plugin should be implemented in Java. |
| TC08 | The development process should be agile and use the Microcosm DevOps Pipeline. |
| TC09 | The version control of the system should be managed by GitHub. |
| TC10 | The system will use PostgreS as our database. |
| TC11 | The system should be deployed in an automated fashion to Azure. |

### Business Constraints

|  |  |
| --- | --- |
| ID | Business Constraint |
| BC01 | The second phase of the project has to be done within 8 months. |
| BC02 | The budget of the project is less than $1,000 and less than $200 for Azure. |
| BC03 | The project will be released to the open source community by Aug 2020. |
| BC04 | The project will support various platforms (e.g. Azure / AWS). |

# 

# Dynamic Views

## Top Level Dynamic View

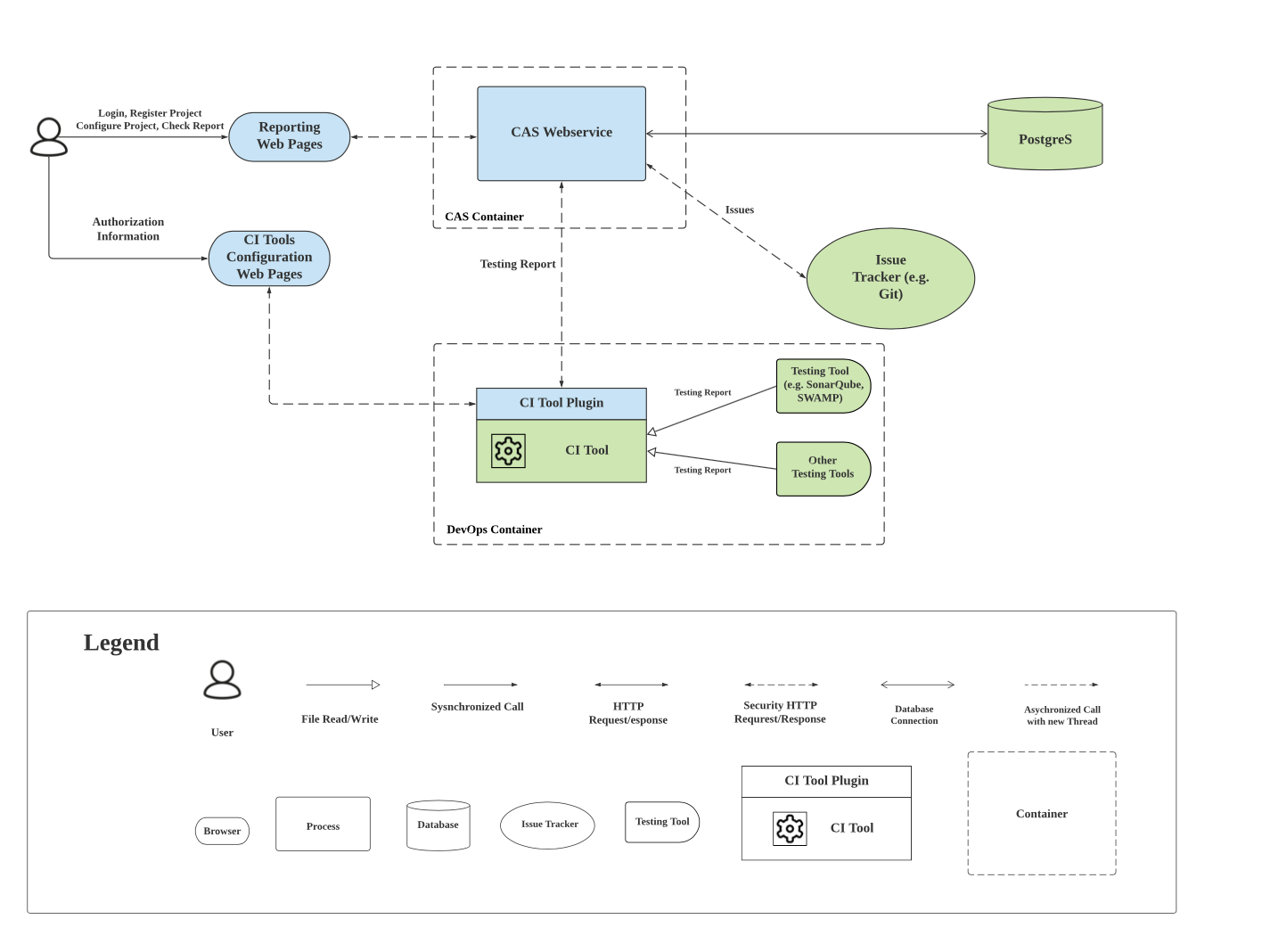


Figure 2. Top Level Dynamic View

### Element Catalog

**Reporting WebPage -** This component represents the user interface of CAS that runs on the web browser.

**CI Tools Configuration WebPage -** This component represents the user interface of a CI tool that runs on the web browser. The responsibility of CI Tools Configuration WebPage is allowing the user to configure CAS authentication information on it.

**CAS Web Service -** This component is the service/controller part of CAS, which uses Django framework.

**CAS Container -** This represents the Docker Container that CAS Web Service runs in.

**PostgreS -** This PostgreS database stores the users, projects, controllers that registered in CAS, and the data of parsed reporting results.

**Testing Tool -** This component represents several testing tools that are integrated in Jenkins, such as SonarQube and SWAMP. The testing tools are supposed to generate testing reports.

**Testing Report -** The testing reports are the testing result generated by the testing tools.

**CI Tool -** This component represents Continuous Integration Tools, such as Jenkins.

**Plugin -** This is the CAS plugin of a CI tool. The responsibility of it is to transfer reports that are generated in the CI tool (e.g. Jenkins) to CAS web service.

**DevOps Container -** This component represents the Docker Container(s) that run the CI tool(s), CI plugin and testing tool(s).

**Issue Tracker -** This component represents the Issue Tracking Tools, such as Git.

**Issues -** The issues are the testing results generated by the testing tools which are formatted with “Source file”, “Start line”, “End line”, etc.

## Rationale & Analysis

* We decided to make CAS a web service. The reason is that our system is supposed to be deployed distributedly, and it will be running continuously. In this way, CAS can serve multiple users and integrate multiple CI tools at the same time. RESTful api can separate the functionalities of CIP and CAS. When CIP is modified or added new plugins for other CI tools, CAS will not be affected.
* We decided to use containers because it is easy to deploy. Using containers also makes our project scalable during the run time. Operators can add more containers to support more concurrent users.
* We decided to build a CI plugin (CIP). Since CAS is able to retrieve reports from various CI tools, including Jenkins, Travis, etc. according to QA-M1. CIP is intended to integrate with different CI tools and send a standard form of static analysis report to the CAS system.
* We decide to use PostgreS as our database because it is NoSQL. In addition, our major data form will be in document, which is also supported by PostgreS. Last but not least, PostgreS is one of our technical constraints brought up by our customers.
* We decided to use https because it is easy to use and it provides desired security features for our connections.

## 2nd Level Decompose Dynamic View

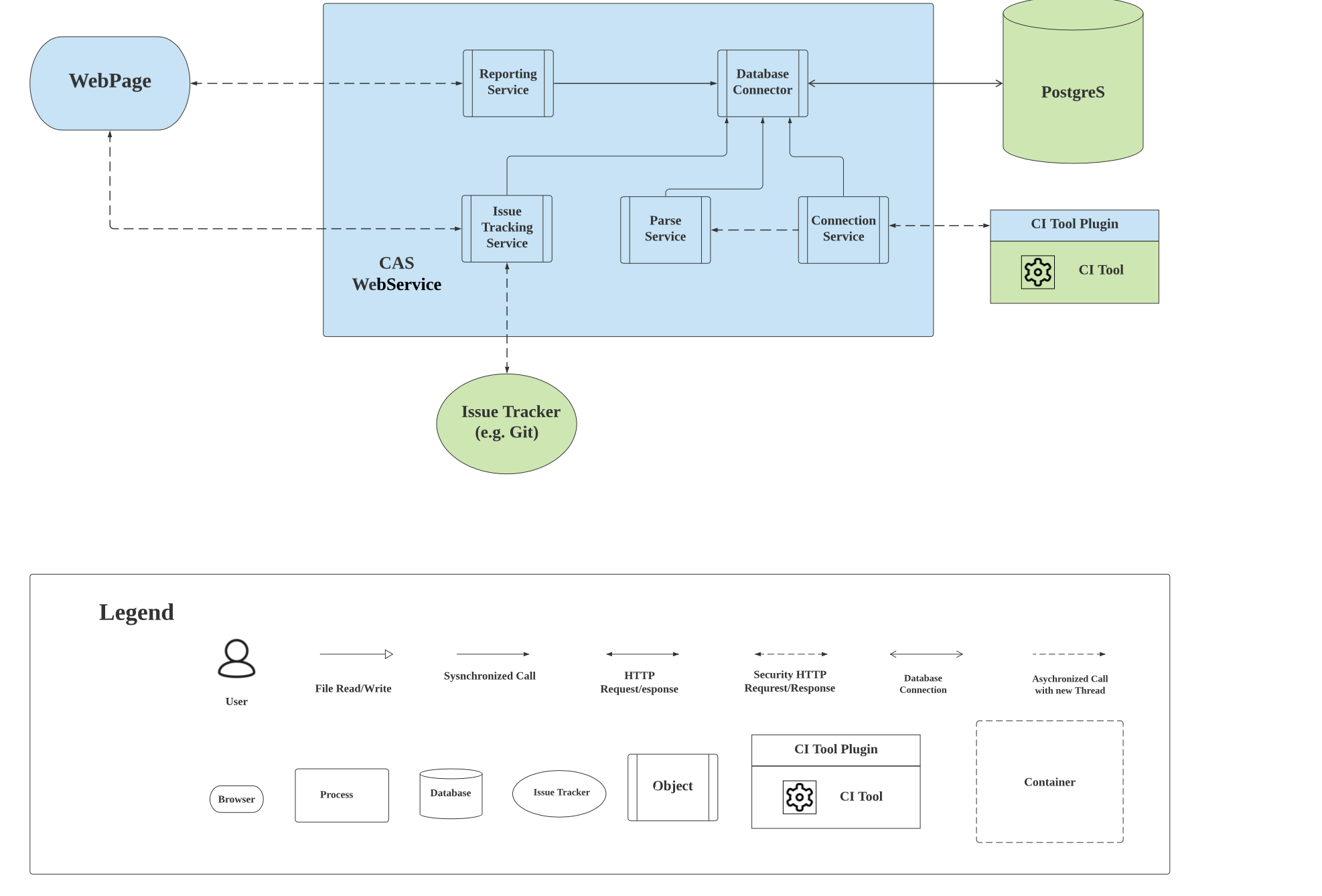


Figure 3. 2nd Level Decompose Dynamic View

### Element Catalog

**Reporting Service -** This unit will maintain the content that will be shown on the dashboard. It will fetch data from Database Connector, and then choose the content of the user selected controller for the WebPage.

**Connection Service -** Connection service will interact with CIP service, retrieving data from CIP and forward the report to the Database Connector and the Parse Service for further processing.

**Database Connector -** This component is responsible to connect with the database and fetch data and write data to database.

**Parse Service-** Parse Service will process raw reports from Connection Service, and then parse it according to user’s requirements.

**Issue Tracking Service -** The Issue Tracking Service will retrieve issue tracker information from the WebPage and fetch issue data from Database Connector, and then it will send https requests to create issues in the desired issue tracker.

### Rationale & Analysis

* Connection Service and Parser will run in separate threads with asynchronized calls. The reason for this design is that the parsing process may take a long time, and in the meantime another project build will be finished and new testing reports will be generated.
* We decided to put the static analysis testing report parser on the CA server instead of the Jenkins plugin since it can decrease the complexity of the Jenkins Plugin and also decrease the time and effort of the future developers spend on the plugins for other CI tools. This design also follows our quality attribute of modifiability (QA-M1).
* The strategy for storing the credential of users is that each time when a user wants to create a new issue on their issue tracking systems. They have to re-enter the username and password. The CAtS system will not store any credentials for user privacy purposes.

## Sequence Diagrams

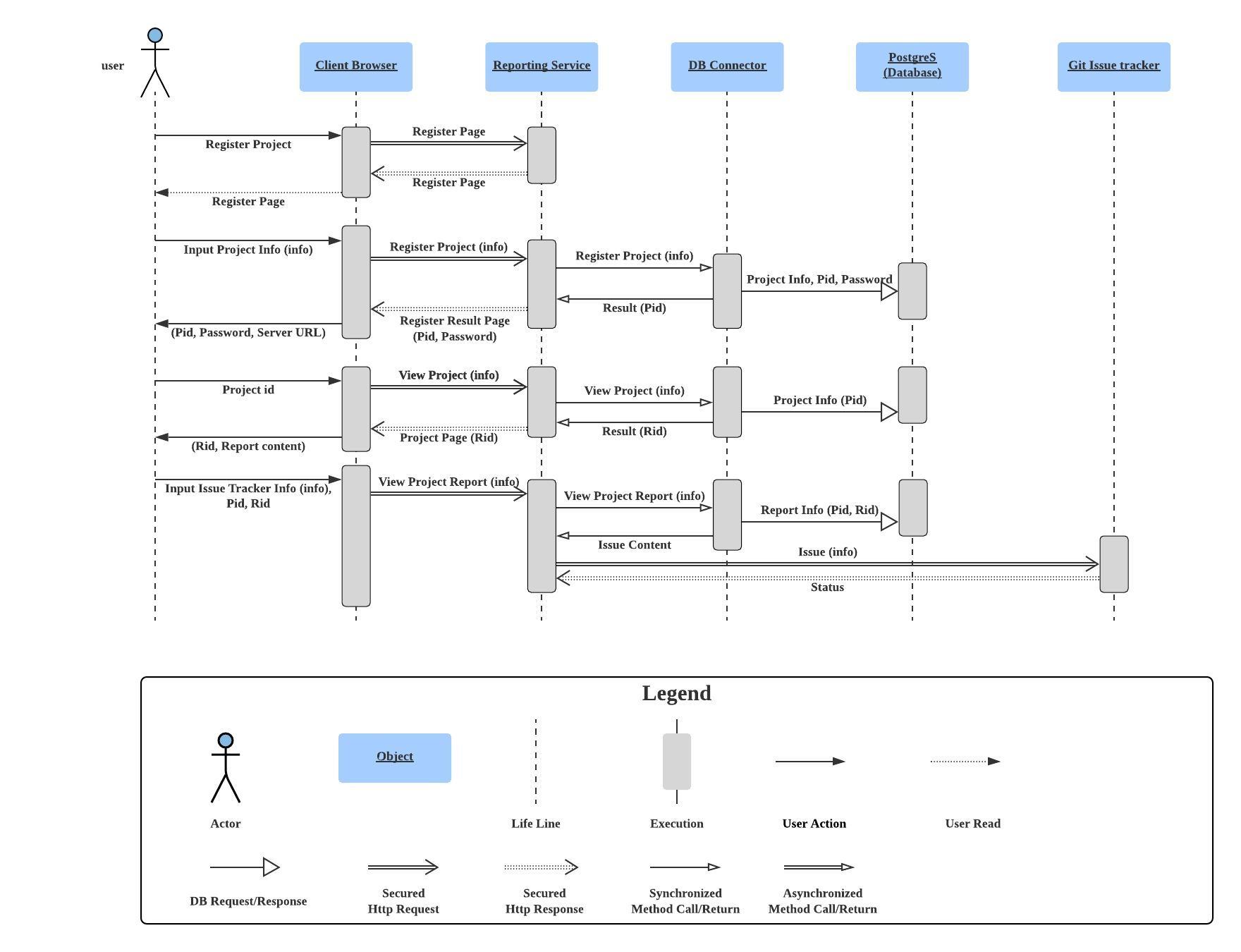


Figure 4. System Sequence

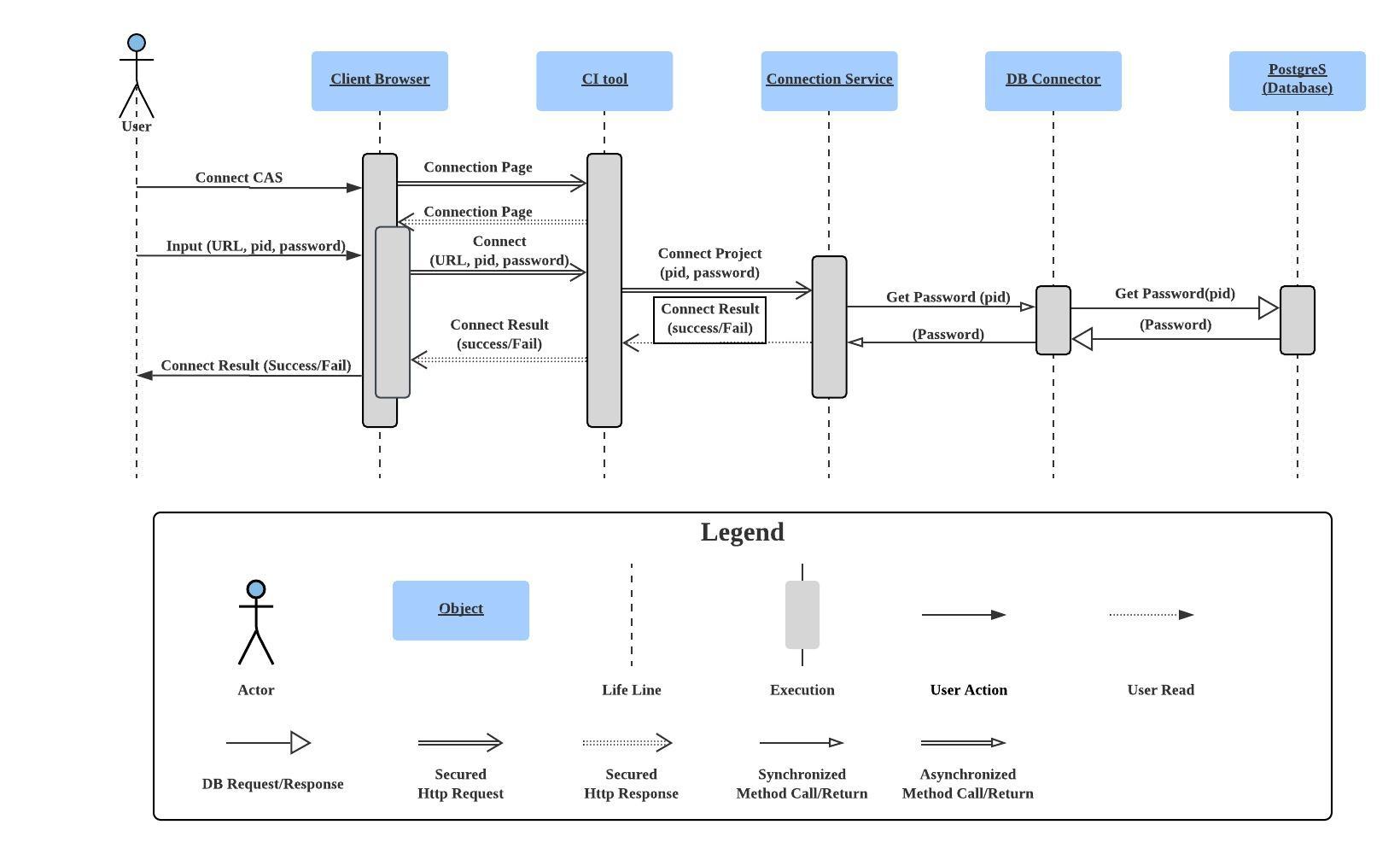


Figure 5. Configure CAS Plugin

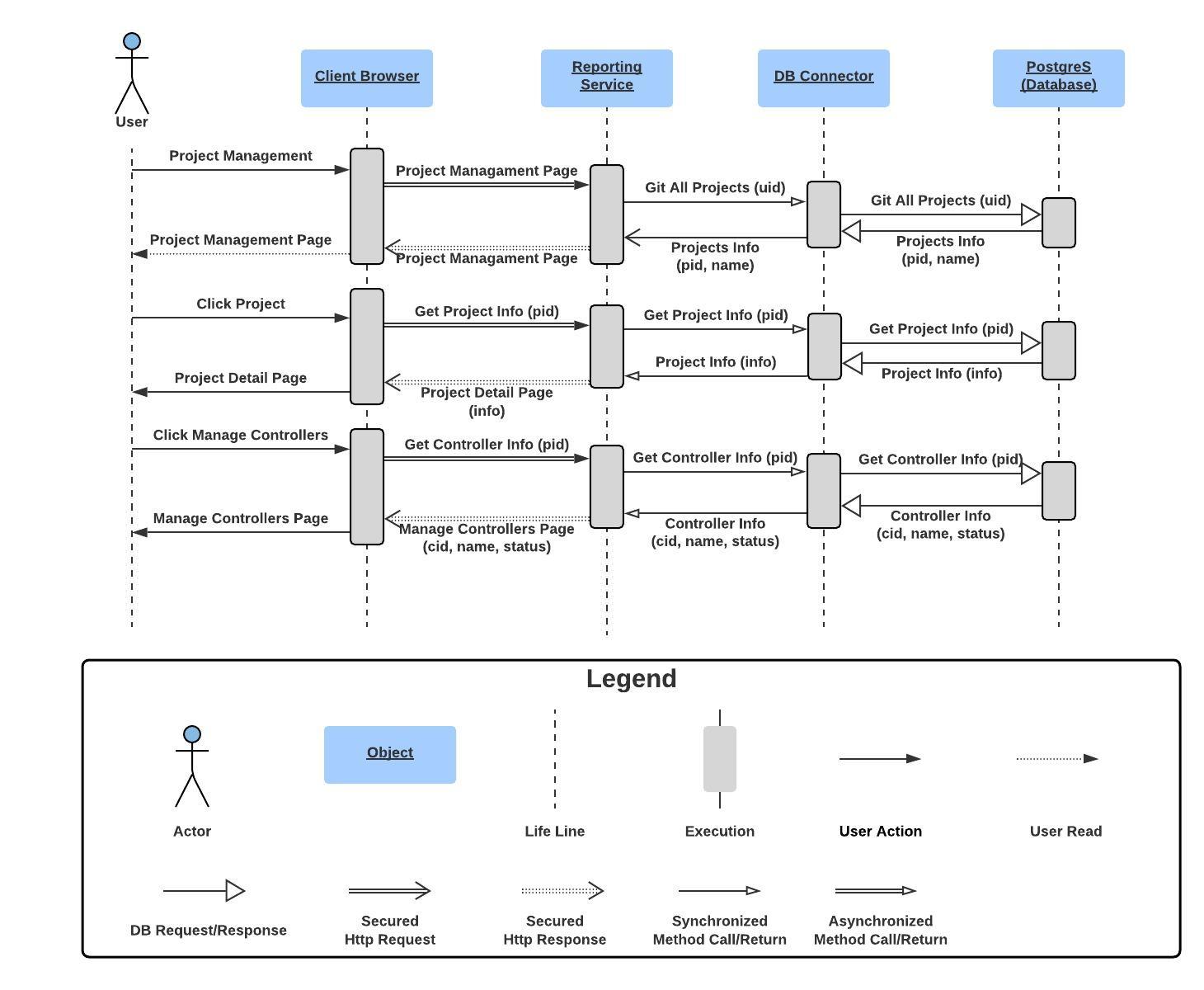


Figure 6. Configure Controllers

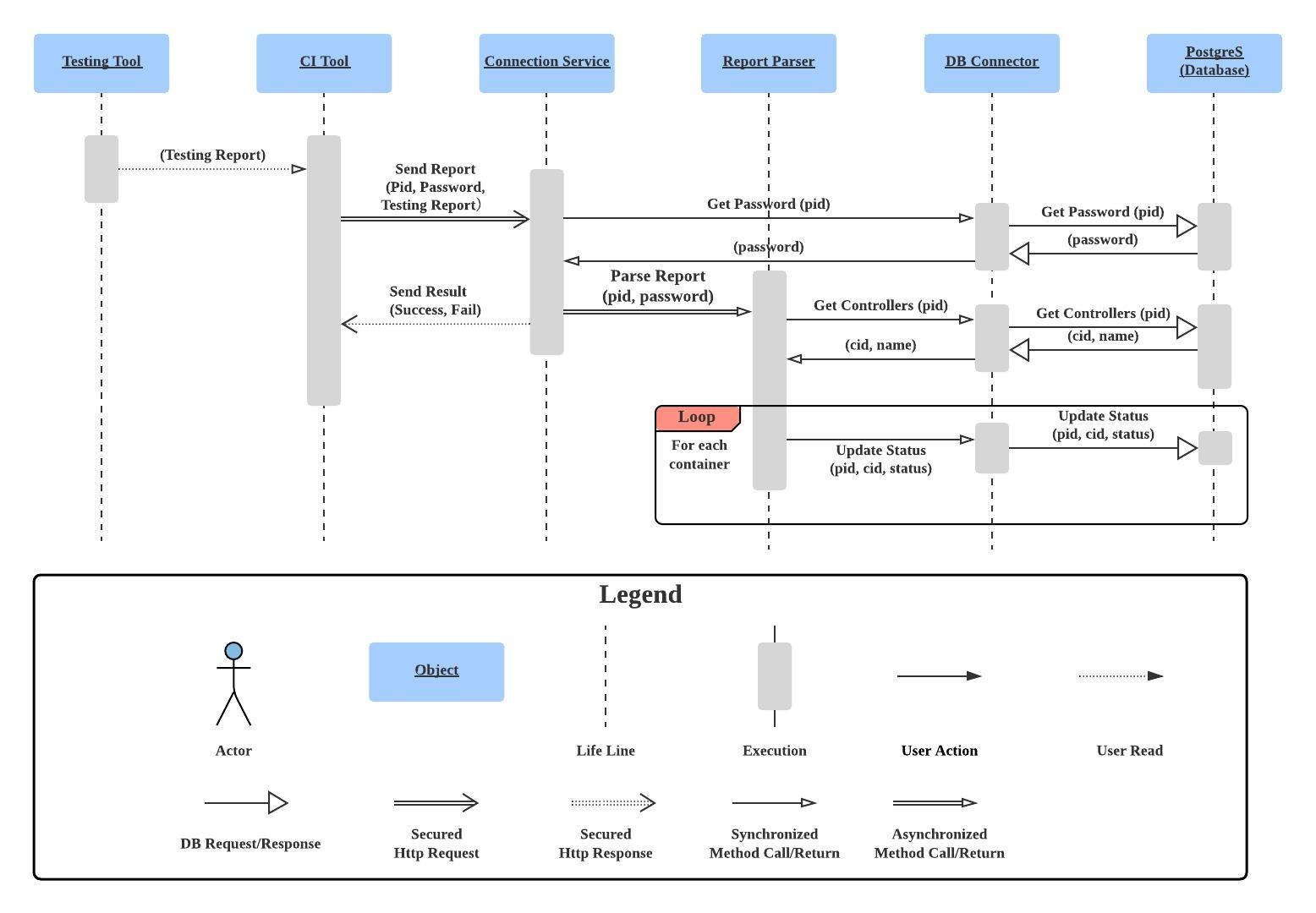


Figure 7. Parse testing report

# Static Views

## Top Level Static Module View

### 

Figure 4. Top Level Static Module View

### Element Catalog

**CAS WebService -** This component is the web based service of CAS, which uses Django framework. The responsibility of this part is to parse the testing report, update data in the database, visualize the parsed result mapped to the user-specified controller, create issues to the Issue Tracker and so on.

**CI Plugin -** This unit will be responsible for interacting with different CI tools. It will fetch reports from CI tools, and then turn the data into a standard form required by CAS service (e.g. JSON file).

**Database -**  This unit is responsible for storing any data that will be used by the CAS WebService.

**Git -** This is the issue tracker where the users will use out CAS WebService to create the issues found in the test.

**OSCAL -** This is the source of the Risk Management Framework (RMF).

## Second Level Decompose Static Module View

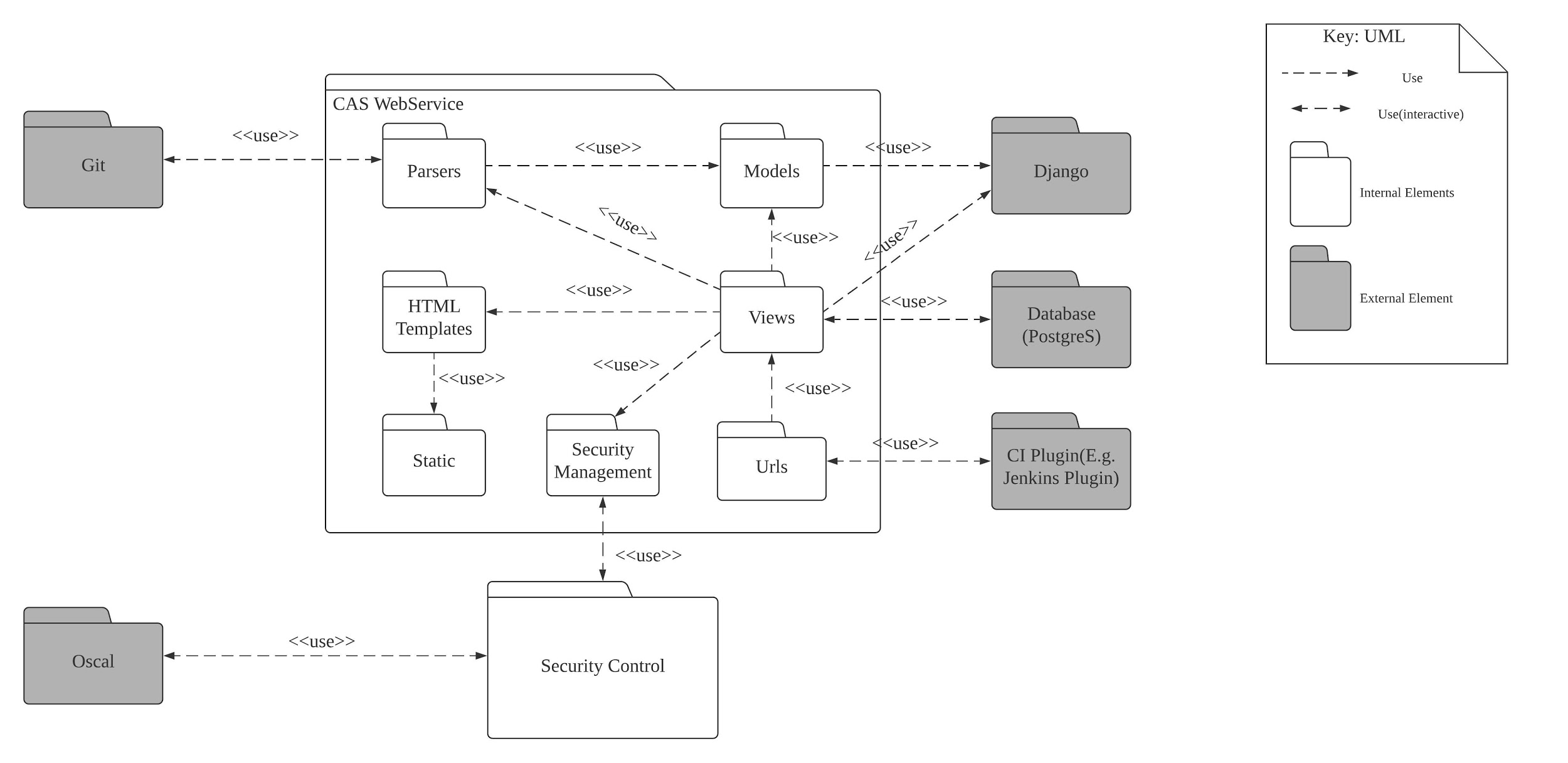


Figure 5. Second Level Decompose Static Module View

Element Catalog

**HTML Templates -** This package contains the HTML files of the user interface of CAS web service.

**Static -** Contains all the javascript codes and static images of the website.

**Models -** This contains the Python classes that specify the fields in the database tables.

**Parser -** This class is responsible to parse the testing reports and map the results to the controllers that specified by the user.

**Views -** This Python class contains the logical code of controller/ service part of CAS web service.

**Security Management -** This module is responsible for checking the Oscal provided control list and updating the corresponding controllers for CAS WebService.

**Urls -** This module is pure Python code that maps between URL path expressions to Python functions.

Rationale & Analysis

We choose Django as the framework of CAS web service.Django is a popular Python based web framework that is fast, fully loaded, scalable and secure. It provides a streamlined approach to developing. In addition, our clients also ask us to use Django because Django is popular and is easy to use (TC-06).

# Dependency Matrix

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| static | 1 |  | x |  |  |  |  |  |
| HTML templates | 2 |  |  |  |  |  | x |  |
| models | 3 |  |  |  | x |  | x |  |
| parser | 4 |  |  |  |  |  | x |  |
| Security Management | 5 |  |  |  |  |  | x |  |
| views | 6 |  |  |  |  |  |  | x |
| urls | 7 |  |  |  |  |  |  |  |

As the Matrix shows, the system has no circle dependencies.

# Appendix

#### View CA report and download

|  |  |
| --- | --- |
| Use case name: | View CA report and download |
| Unique case ID: | ViewCAReportAndDownload |
| Primary Actor(s) : | User |
| Secondary Actor(s): | None |
| Brief description: | Users can view the history of authentication testing of a project by using cas. |
| Preconditions: | 1. Users have a CAS account. 2. Users have already set up a project on their cas. 3. Users have internet access. |
| Flow of events: | 1. Users go to the CAS website, log in to their account. 2. On the project page, click the project name you want to check to go to the dashboard of the chosen project. 3. On the project dashboard page, click the test history you want to check. 4. Click “View Details” to view the detailed report. 5. Click “Download as PDF” to download the report to the local machine. |
| Postconditions: | 1. Users can see the test history of the authentication test for an existing project. 2. Users can see the test report of the authentication test for an existing project. 3. Users can see the report on their local computer. |
| Priority | High |
| Alternative flow or exception: | None |
| Non-behavioral requirements | None |
| Assumptions: | 1. The connection between CAS service and database works fine. |
| Source: | None |

#### 

#### Create new issue on Git

|  |  |
| --- | --- |
| Use case name: | Create new issue on Git |
| Unique case ID: | CreateNewIssueOnGit |
| Primary Actor(s) : | User |
| Secondary Actor(s): | None |
| Brief description: | Users can view the history of authentication testing of a project by using cas. |
| Preconditions: | 1. Users have a CAS account. 2. Users have already set up a project on their cas. 3. Users have internet access. 4. Users have an issue tracker account (Git). |
| Flow of events: | 1. Users go to the CAS website, log in to their account. 2. On the project page, click the project name you want to check to go to the dashboard of the chosen project. 3. On the project dashboard page, click the test history you want to create issues based on. 4. Click the “Create Issues” button. 5. Enter the issue tacker information (Git Username, Git Password and Git Repo Name). 6. Click the “Update” button. 7. Login to the issue tracker account to check the issues. |
| Postconditions: | 1. Users can view the newly founded issues on their issue tracking account correlated to the new commit action. |
| Priority | Medium |
| Alternative flow or exception: | None |
| Non-behavioral requirements | None |
| Assumptions: | 1. The connection between CAS service and database works fine. |
| Source: | None |

#### 

#### User change profile

|  |  |
| --- | --- |
| Use case name: | User change profile |
| Unique case ID: | UserChangeProfile |
| Primary Actor(s) : | User |
| Secondary Actor(s): | None |
| Brief description: | Users can change the profile of their accounts. |
| Preconditions: | 1. Users have access to CAS service. 2. Users have registered for a CAS account. |
| Flow of events: | 1. Users go to the CAS website, log in to their account. 2. On the main page, click the user name on the top-left corner, which will then show the “Profile” button and “Setting” button. 3. Click the “Profile” button. 4. On the Profile Page, click “Update Info” and modify the content 5. Click the “Update” to confirm the update. |
| Postconditions: | 1. New user profile is updated to the database. 2. Users can view the new profile on their profile page. |
| Priority | Low |
| Alternative flow or exception: | None |
| Assumptions: | 1. The connection between CAS service and database works fine. |
| Source: | None |

#### 

#### User change setting

|  |  |
| --- | --- |
| Use case name: | User change setting |
| Unique case ID: | UserChangeSetting |
| Primary Actor(s) : | User |
| Secondary Actor(s): | None |
| Brief description: | Users can change the setting of their accounts. |
| Preconditions: | 1. Users have access to CAS service. 2. Users have registered for a CAS account. |
| Flow of events: | 1. Users go to the CAS website, log in to their account. 2. On the main page, click the user name on the top-left corner, which will then show the “Profile” button and “Setting” button. 3. Click the “Setting” button. 4. On the Profile Page, click “Change Password” and enter the new password. 5. Click the “Confirm” to confirm the update. |
| Postconditions: | 1. New user settings are updated to the database. 2. Users can login with new passwords. |
| Priority | High |
| Alternative flow or exception: | None |
| Assumptions: | 1. The connection between CAS service and database works fine. |
| Source: | None |

#### User manage project accessibility

|  |  |
| --- | --- |
| Use case name: | User manage project accessibility |
| Unique case ID: | UserManageProjectAccess |
| Primary Actor(s) : | User |
| Secondary Actor(s): | None |
| Brief description: | Users can share or stop sharing their project with other users. |
| Preconditions: | 1. Users have access to CAS service. 2. Users have registered for a CAS account. 3. Users have registered a project on CAS. 4. Users have to know the username of the user they want to share with. |
| Flow of events: | 1. Users go to the CAS website, log into their account. 2. On the main page, click “Own Projects”. 3. Select an already uploaded project. 4. Click the “Project Setting” button. 5. Enter the username of whom the owner wants to share with. 6. Click the “Share” button to confirm. |
| Postconditions: | 1. The user which this project shared is able to view the project on their “Shared Projects” page. 2. The owner of the project will view all the shared members of the projects. |
| Priority | High |
| Alternative flow or exception: | 1. In step 5, users can enter the username whom the owner wants to stop sharing with and in step 6, click “Confirm” to stop sharing. |
| Assumptions: | 1. The connection between CAS service and database works fine. |
| Source: | None |

#### Admin login

|  |  |
| --- | --- |
| Use case name: | Admin login |
| Unique case ID: | AdminLogin |
| Primary Actor(s) : | Admin |
| Secondary Actor(s): | None |
| Brief description: | Login as admin and manage the CAS service. |
| Preconditions: | 1. Have the admin credential. |
| Flow of events: | 1. On the Login page of CAS website, click “Login as admin”. 2. On the admin login page, enter the username and password for the admin user. 3. Click “Log in” and type the admin’s credential. |
| Postconditions: | 1. Login as administrator of the CAS and manage the system. |
| Priority | High |
| Alternative flow or exception: | 1. After step 3, admin can maintain the User, Group, Projects and Controls. 2. Admin can click “VIEW SITE” on the right corner of the page to view the normal CAS web page. |
| Assumptions: | 1. The connection between CAS and database is working fine. |
| Source: | None |

#### 

#### Register project on CAS

|  |  |
| --- | --- |
| Use case name: | Register project on CAS |
| Unique case ID: | RegisterProjectOnCAS |
| Primary Actor(s) : | User |
| Secondary Actor(s): | None |
| Brief description: | Users can register a new project on CAS. |
| Preconditions: | 1. Users have access to CAS service. 2. Users have registered for a CAS account. |
| Flow of events: | 1. Users go to the CAS website, log in to their account. 2. On the main page, users click on “New Project” to go to the registering page. 3. On the registering page, users input the basic information of the project.    1. Users type the “Project Name”, “Project Description” into different input boxes respectively.    2. Users click on the register button to finish the registration. The page will automatically go to the project information page. 4. The project information page would show all the information including “Project Name”, “Project Description”. The page will also show the automatically generated random “Project ID” used for connection to the CI platform (E.g. Jenkins). |
| Postconditions: | 1. A new project is registered. 2. The project information is stored. 3. Project IDs are automatically generated. |
| Priority | High |
| Alternative flow or exception: | 1. In Step 3.2, if the user left any input boxes blank, stay on the same page and show “please fill in the blank” notices on the blank boxes. |
| Non-behavioral requirements | None |
| Assumptions: | 1. The project information includes “Project Name”, “Project Description”. 2. Users need “Project ID” to build an encrypted connection to the CI platform. |
| Source: | None |

#### 

#### Configure the controller list for project

|  |  |
| --- | --- |
| Use case name: | Configure the controller list for project |
| Unique case ID: | ConfigureProjectControllers |
| Primary Actor(s) : | User |
| Secondary Actor(s): | None |
| Brief description: | Users configure the controllers that they are concerned with for a certain project. |
| Preconditions: | 1. Users have access to CAS service. 2. Users have registered for a CAS account. 3. The project has been registered on CAS. |
| Flow of events: | 1. Users go to the CAS website, log in to their account. 2. Go to the page of “Own Projects”. 3. Choose the name of the project that users want to configure. 4. Click “Config Controls” to go to the page which shows all provided controllers in tree structure. 5. Select some classes of controllers to get all the controllers of these classes selected. 6. Click a controller class to make the controllers of this class expand below it, then select some of them. 7. Input the keywords of a certain controller in the search box then find the controller the users want, then select it. 8. After selecting all the controllers to be tested, click the “Save” button. |
| Postconditions: | 1. The controllers to be tested are configured for this project. 2. If some new tests are completed, the dashboard will show the result of these selected controllers. |
| Priority | High |
| Alternative flow or exception: | None |
| Non-behavioral requirements | 1. The list of provided controllers to be selected should be shown in a clear view so that the users can browse and find them easily. |
| Assumptions: | 1. Users are familiar with the controllers. |
| Source: | None |

#### Admins install CAS plugin on jenkins

|  |  |
| --- | --- |
| Use case name: | Admins install CAS plugin on jenkins |
| Unique case ID: | JenkinsInstallCASPlugin |
| Primary Actor(s) : | Admin |
| Secondary Actor(s): | None |
| Brief description: | Admins install the CAS plugin on their Jenkins service. |
| Preconditions: | 1. Admins have set up a Jenkins server. |
| Flow of events: | 1. Admins log on the the Jenkins GUI 2. Select “Manage Jenkins” 3. Select “Manage Plugins” 4. Click the “Advanced” tab 5. Go to “Upload Plugin”. 6. Select “cats-plugin”. 7. Click “Upload”. 8. Wait until the installation progress page displays “Success” |
| Postconditions: | 1. Cats plugin has been installed on Jenkins |
| Priority | High |
| Alternative flow or exception: | In event flow step 8, admins can also do:   1. Check the box “Restart Jenkins when installation is complete and no jobs are running”. 2. Wait until the download progress page displays “Please wait while Jenkins is restarting ...” 3. Login to Jenkins system again.   Then the postconditions can be reached with this alternative flow. |
| Non-behavioral requirements | None |
| Assumptions: | Admins are familiar with Jenkins configuration processes. |
| Source: | Own experience using Jenkins and Jenkins plugins. |

#### Admin configure CAS plugin on Jenkins

|  |  |
| --- | --- |
| Use case name: | Admin configure CAS plugin on Jenkins |
| Unique case ID: | JenkinsConfigCASPlugin |
| Primary Actor(s) : | Admin |
| Secondary Actor(s): | None |
| Brief description: | Admins have installed CAS plugin on Jenkins |
| Preconditions: | 1. Admins have set up a Jenkins server. 2. Admins have installed CAS plugin on Jenkins. 3. Admins have logged on to Jenkins GUI. 4. Admins have set up a CAS server. 5. Admins have registered for a CAS account. |
| Flow of events: | 1. Go to Jenkins system configuration 2. Under “CAS servers”, click “Add CAS” and provide: 3. CAS server name 4. CAS server URL 5. CAS server authentication token 6. Create a Jenkins job 7. Go to the job configuration 8. Under “Build”, click “Add build step”, and select “Execute CAS check” 9. Provide the following information to Jenkins: 10. Project ID generated by CAS 11. Project authentication token 12. Save build configuration. |
| Postconditions: | 1. Jenkins automatically invokes CAS check after every build of a project. |
| Priority | High |
| Alternative flow or exception: | None |
| Non-behavioral requirements | None |
| Assumptions: | Users are familiar with Jenkins build and configuration processes. |
| Source: | Own experience using Jenkins and Jenkins plugins. |

#### View CAS result on dashboard

|  |  |
| --- | --- |
| Use case name: | View CAS result on dashboard |
| Unique case ID: | ViewCASDashboard |
| Primary Actor(s) : | User |
| Secondary Actor(s): | None |
| Brief description: | Users can view the testing results of their existing project by visiting the dashboard. |
| Preconditions: | 1. Users have a CAS account. 2. Users have already set up a project on their cas. 3. Users have internet access. |
| Flow of events: | 1. Users go to the CAS website, log in to their account. 2. On the project page, click the project name you want to check to go to the dashboard of the chosen project.. 3. View the dashboard on the existing project page. |
| Postconditions: | 1. Users can see the test results of an existing project by visiting the dashboard page for that project. |
| Priority | High |
| Alternative flow or exception: | 1. On the dashboard page, users can click on each testing case to view the detailed information of that case. Possible information includes:    1. Case name.    2. Controller associated with this case.    3. Start time for that case.    4. End time for that case.    5. Output for that test case.    6. Error message for that test case if it fails. |
| Non-behavioral requirements | 1. The dashboard should show a list of test cases associated with their names, start time, end time and test results. Users should be able to click on each test case to view detailed information of that test case. |
| Assumptions: | 1. The connection between CAS and Jenkins works fine. |
| Source: | None |

1. Controllers E.g. Federal Risk and Authorization Management Program (FedRAMP) Profile Examples, <https://github.com/usnistgov/OSCAL/tree/master/content/fedramp.gov> [↑](#footnote-ref-0)