Requirement Specification:

# Introduction:

This chapter represents an analytical phase in which we fully describe our system’s behavior. We first describe the system’s main actors, then we iterate over the different functional and non-functional requirements. Afterwards we analyze those requirements through use case diagrams and system sequence diagrams in which we describe every possible interaction between the actors and the system to be.

# Actors:

In our system we have three actors:

* **User**:

The simple user can only register their Jenkins job in order to be recognized by the system, and consult the final dashboard.

* **Jenkins**:

Jenkins is the main actor of our system. It will automatically trigger our system if certain conditions are met. We will go through the details of this interaction in the following sections.

* **System administrator**:

A system administrator is the super user that has the ability to outweigh Jenkins and change certain job related information and the way our system treats them.

# Requirement Analysis:

## Functional Requirements:

Functional requirements are basically the contract that ties the system developers with the clients. That is why our system has to provide the following features:

* **User**:
  + Configure a Jenkins job to automatically save data (through our system) after every build (This is realized by the addition of a PowerShell script as a post-build action).
  + Consult a test per build velocity dashboard.
* **Jenkins**:
  + Save a job’s related build data from with or without external references.
  + Save a job’s related Nunit test results with or without external references.
  + Load job’s related data through accessing Jenkins REST API.
  + Load job’s related Git repository information through Stash REST API.
* **System administrator**:

The system administrator inherits all of the user’s functionalities in addition to the following one:

* + Manually register a Jenkins job in order to be recognized by the system.
  + Manually trigger a global search through Jenkins history links to find unsaved builds and save them.
  + Manage a Jenkins job related data (project, squad, team, test type) through CRUD operations (Create – Remove – Update – Delete).
  + Authenticate to the system through the company’s Active Directory (if user belongs to a certain group).

## Non-functional requirements:

Our main objective is to build a very performing solution. However, a system that only satisfies the functional requirements rarely satisfies the client. For this reason we have to also focus on non-functional requirements when developing the solution:

* **Ergonomics**:

The system has to provide a clear, intuitive and legible front-end.

* **Consistency**:

The system has to always provide coherent data and matching the data provided by other systems.

* **Performance**:

The system has to minimize the response time while saving and manipulating the massive data provided by the external actors.

* **Flexibility**:

The system has to provide several usage methods so as to suit the user’s needs and requirements.

* **Extensibility**:

Integration of new modules or features can be achieved without affecting already existent modules.

* **Maintenance**:

The system has to be easily maintainable. A good method to ensure this requirement is the clean coding patterns (e.g. variable names, function size), and good documentation.

* **Portability**:

The system has to be portable in way that it works under any working environment.

# Requirement Specification:

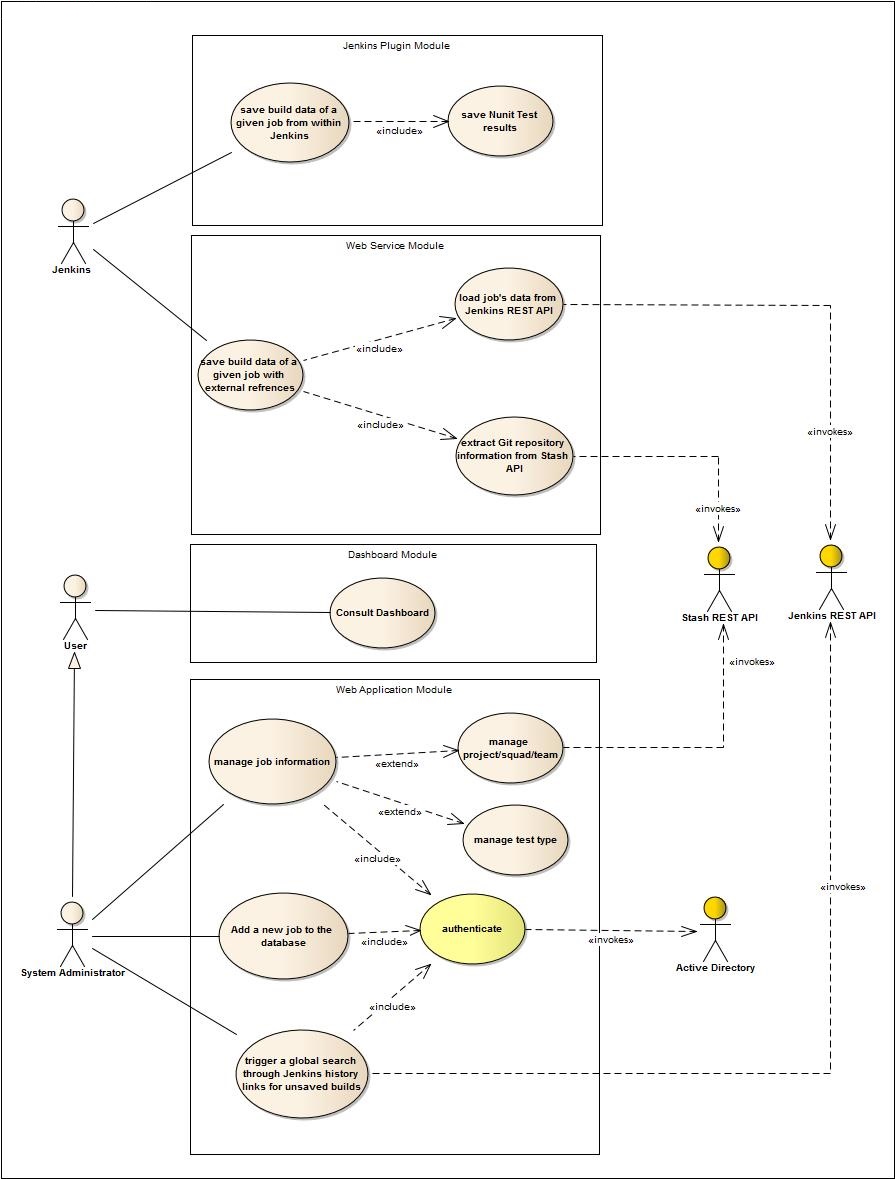
After the requirement analysis, specification proves to be crucial so as to get a better understanding of the required functionalities. This analysis will be based use case and system sequence diagrams.

## Use case diagrams:

Use case diagrams allow us to better describe the interactions between the different actors and our system.

### Global use case:

In this diagram we regroup all of the use cases, divided into sub-modules to reduce complexity and have a better global view.

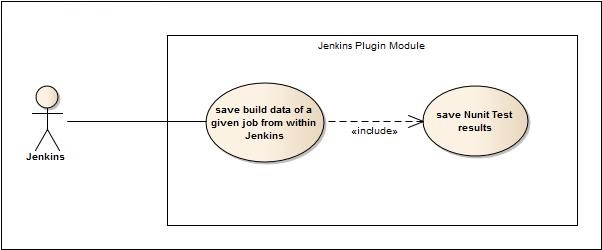


### Detailed description:

In the following section we detail each module’s use cases individually.

#### Jenkins plugin module:

##### Use Case Diagram:



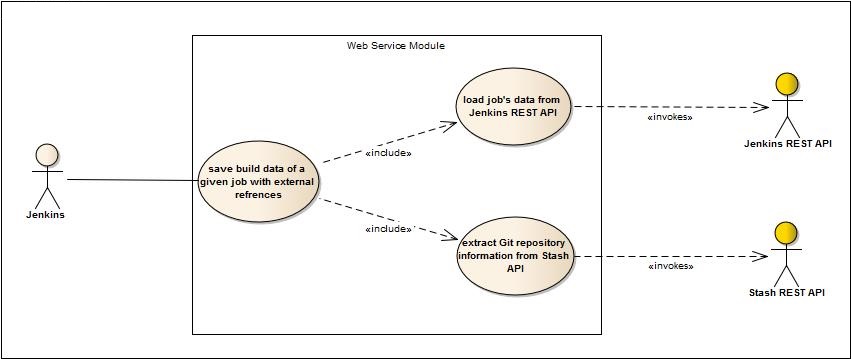
##### Case: Save build data of a given job from within Jenkins:

In this case, Jenkins uses the plugin to save a given job’s build data and then save the Nunit test results.

|  |  |
| --- | --- |
| **Use Case** | Save build data of a given job from within Jenkins |
| **Primary Actor** | Jenkins |
| **Brief** | The system saves a job’s build data through a Jenkins plugin. |
| **Preconditions** | The Jenkins job has to be successfully built by Jenkins and has generated Nunit test results XML files. |
| **Post-conditions** | The build data and the test results (if available) are both saved in the system. |
| **Triggers** | This case is triggered whenever a Jenkins job finishes building. |
| **Basic flow** | 1. The system receives build data through the Jenkins plugin. 2. If the test results files are available, the system receives the XML files through the plugin. 3. The system extracts the information from the XML files and transforms them into structured data. 4. The system saves the provided data in the database. |

#### Web service module:

##### Use case diagram:



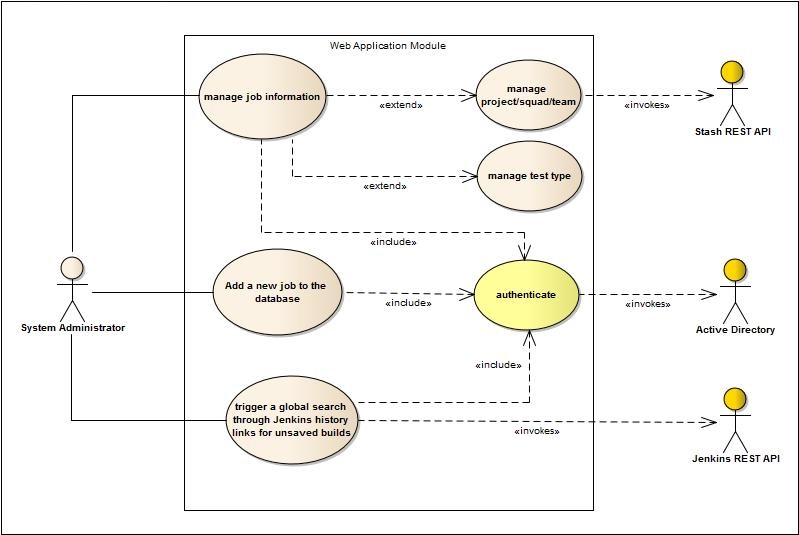
##### Case: Save build data of a given job with external references:

In this case, Jenkins uses a web service exposed by our system to save a given job’s build data and then save the Nunit test results.

|  |  |
| --- | --- |
| **Use Case** | Save build data of a given job with external references. |
| **Primary Actor** | Jenkins |
| **Brief** | The system saves a job’s build data through a web service call. |
| **Preconditions** | The Jenkins job has to be successfully built by Jenkins and has generated Nunit test results XML files. |
| **Post-conditions** | The build data and the test results (if available) are both saved in the system. |
| **Triggers** | This case is triggered whenever a Jenkins job finishes building. |
| **Basic flow** | 1. Jenkins provides the web service with the current job’s URL. 2. The system uses that URL to access Jenkins REST API and retrieve job’s information, build data and Nunit test result files. 3. If the job is new to the system, the system uses the provided Git information to access Stash REST API and retrieve the job’s related data (project – squad – team). 4. The system extracts the information from the XML files and transforms them into structured data. 5. The system saves the provided data in the database. |

#### Web application module:

##### Use case diagrams:



##### Case: add a new job to the database

##### Case: Manage job information

##### Case: Trigger a global search through Jenkins history links for unsaved builds

##### Case: Authenticate