Mälardalen University
School of Innovation, design and engineering
Västerås, Sweden

Component Technologies - 7.5 hp - CDT401

COMPONENT REPOSITORY

Design Description

Anton Roslund
ard15003@student.mdh.se

Cécile Cayèré
cce18001@student.mdh.se

Milos Ojdanic
moc16001@student.mdh.se

Stanislas Pedebearn
spn18013@student.mdh.se

Vincenzo Stoico
vso18003@student.mdh.se

October 29, 2018

Contents

1 Introduction

The purpose of this project is to develop a component repository intended for software developers. There will be two ways to access the repository. The first one is a web page, intended for end users. The web page allows users to browse and download components. The repository is also accessible through a desktop application. The desktop application is intended as a way to curate the repository. The desktop application provides the ability to add, modify and delete components.

1.1 Definitions

Terms	Definitions
Component	A component is an element of a software that can be deployed independently and can be reuse easily. It have interfaces...
Interface	An interface delimits two components exchanging informations among them.
Repository	A repository allows to group various items.
Use Case Model	List of actions or events defining the interactions between a role and a system to achieve a goal.

1.2 Related Documents

Document identity	Document title
Requirements.pdf	Requirements Description

2 Functional Description

2.1 Use Case Model

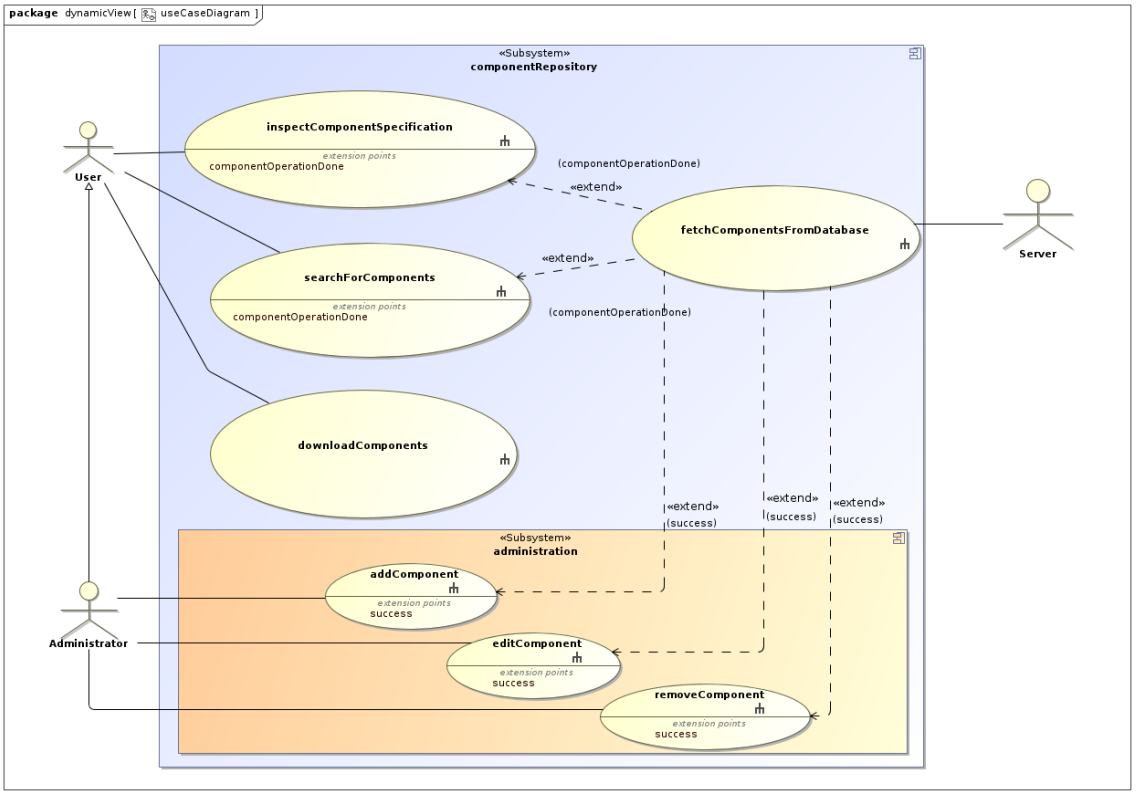

Figure 1: Use Case Diagram

2.1.1 Actors

- User: This actor represents the general concept of user embracing all the entities that can perform an action on the system (e.g. visitor, admin, script)
- Administrator: This actor represent the entity that manages the system. An admin does significant changes to the repository (e.g. deletion, addition).

2.1.2 Use Cases

The *inspectComponentSpecification* functionality is needed to show through graphical user interface the metadata of a selected component. The *searchForComponent* allows the user to search a for a specific component. They are connected with the *fectComponentFromDatabase* use case through an *extend* relationship since all of three are treated as independent functionality. The fectComponentFromDatabase is triggered all the times that the database is modified so the component information and the search results, shown on the GUI, can be updated. For this reason, the *operationOnDatabase* use case is connected to the fectComponentFromDatabase with an *include* relationship. Finally, the repository provides the *downloadComponent* feature the enables the downloading of a specific component in different formats (e.g. `.jar`, `.dll`).

2.2 Fetch Component From Database

Initiator : OperationOnDatabase Use Case
Goal : Get all the information about components from the databases

Main Flow of Events :

1. The repository queries the database asking for the available information about all the components
2. The database response is a data structure containing the information about components

Extensions : This use case is described as an extension of the *search* and *inspection* functionality to indicate that the outcome of the latter must be updated according to the content of the database, retrieved with the *fetching* functionality.
Exceptions : If the database is empty an empty list is returned.
Comments : The feature is useful to shown content list in the GUI (e.g. the list of the component). The fetching is used to keep the GUI always updated.

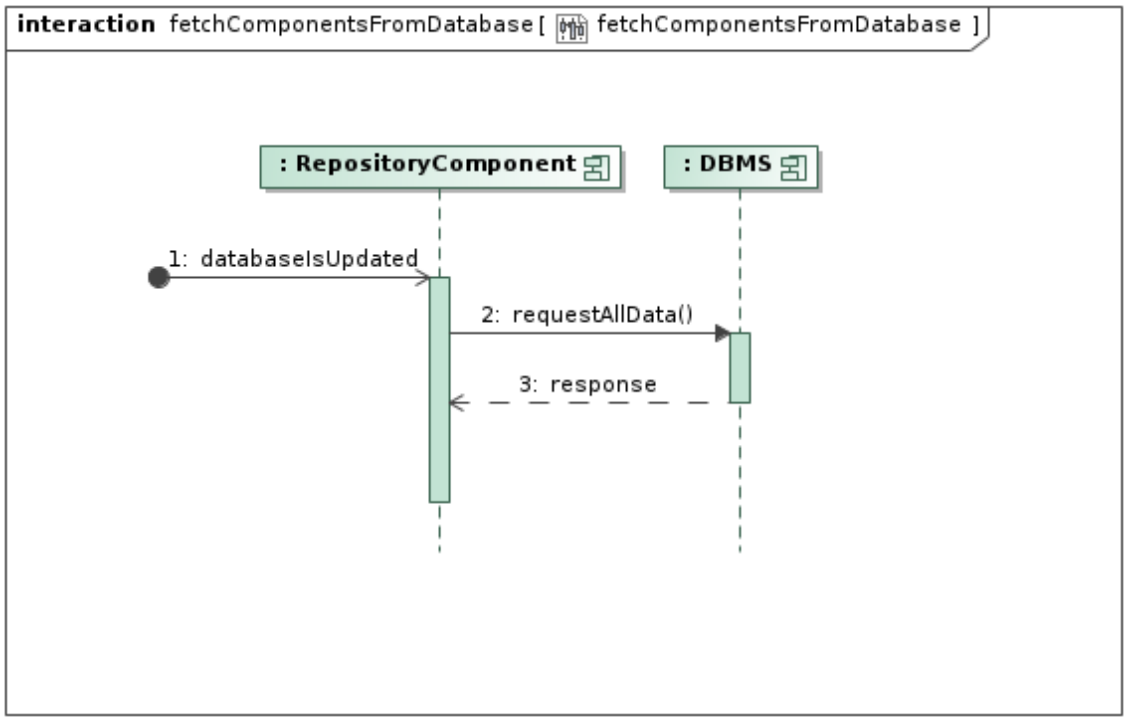

Figure 2: Main flow fetch component

2.3 Inspect Component Specification

Initiator : User
Goal : The system requests to inspect the component specification

Main Flow of Events :

1. User requests a component specification. The Web component provides the ID to the Repository component.
2. The Repository Component forward the request to the DBMS that returns the metadata file.
3. The presentation is managed by the Repository Component that associates the metadata file to the graphical view.

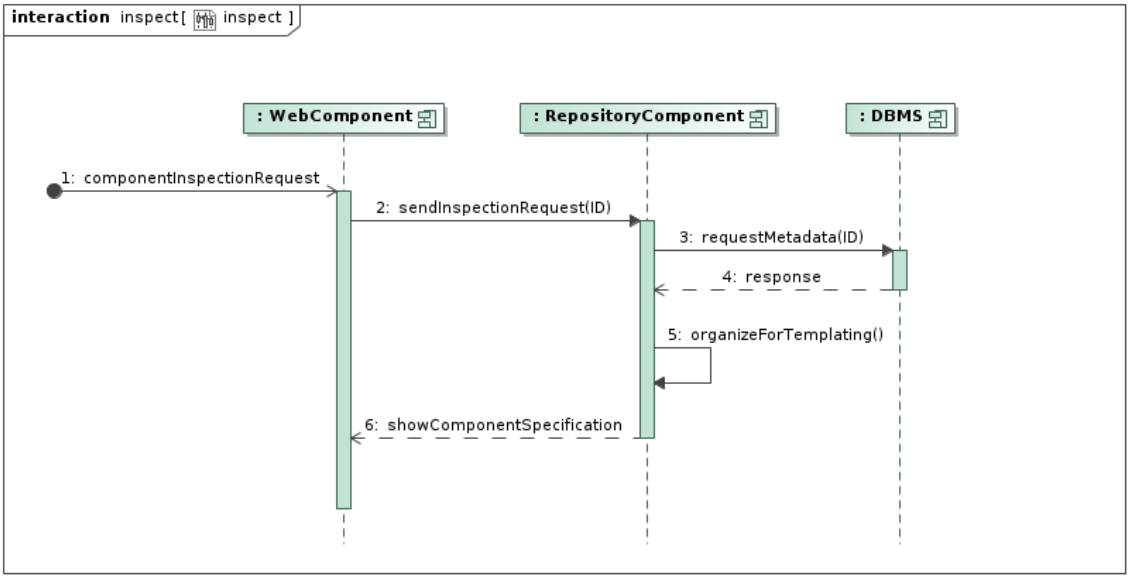

Figure 3: Main flow download event diagram

2.4 Search Component

Initiator : User
Goal : Find a particular component in the repository

Main Flow of Events :

1. The User is looking of a particular component, so inserts the component ID in the GUI. The Web Component provides the ID to the Repository Component.
2. A control about the existence of the component is done. In the negative case an error message is shown.

Exception :

1. The component does not exist. The system displays error message.

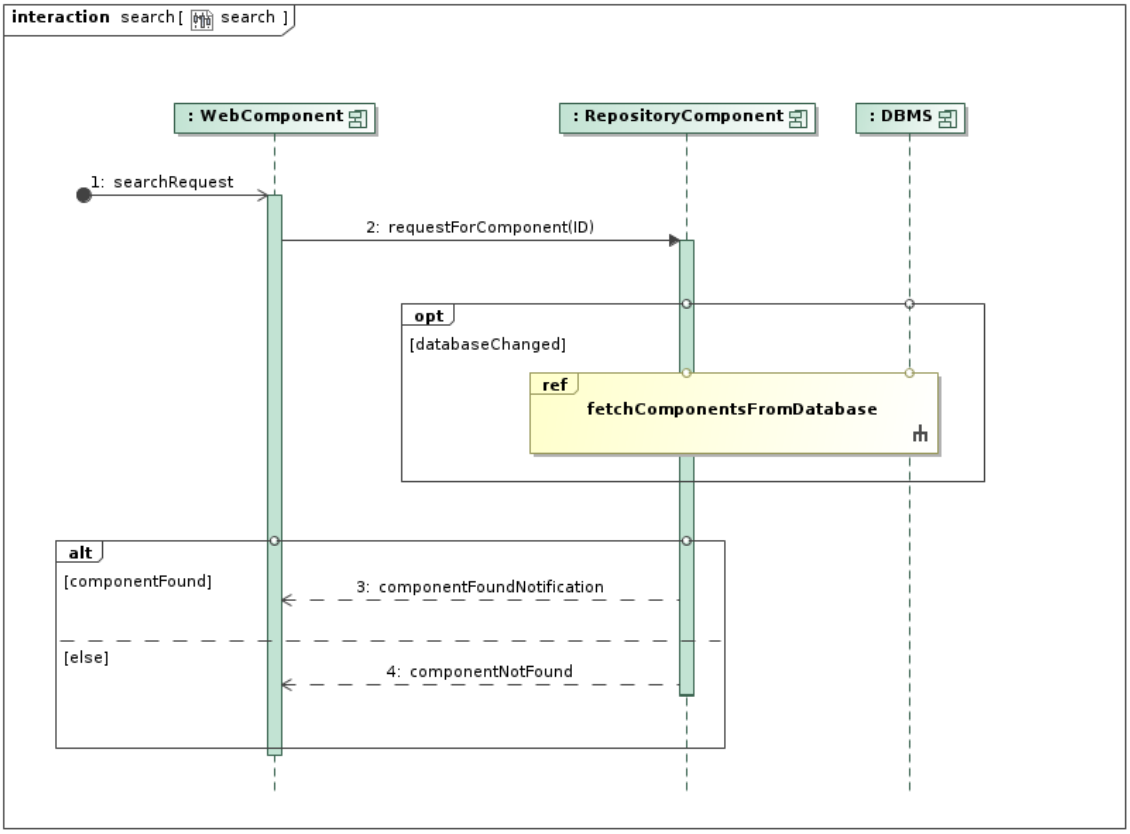

Figure 4: Main flow download event diagram

2.5 Download Component

Initiator : User.
Goal : Provide a component file to the user.

Main Flow of Events :

1. User requests to download a component.
2. The file request is forwarded to the DBMS that answers with the selected file.

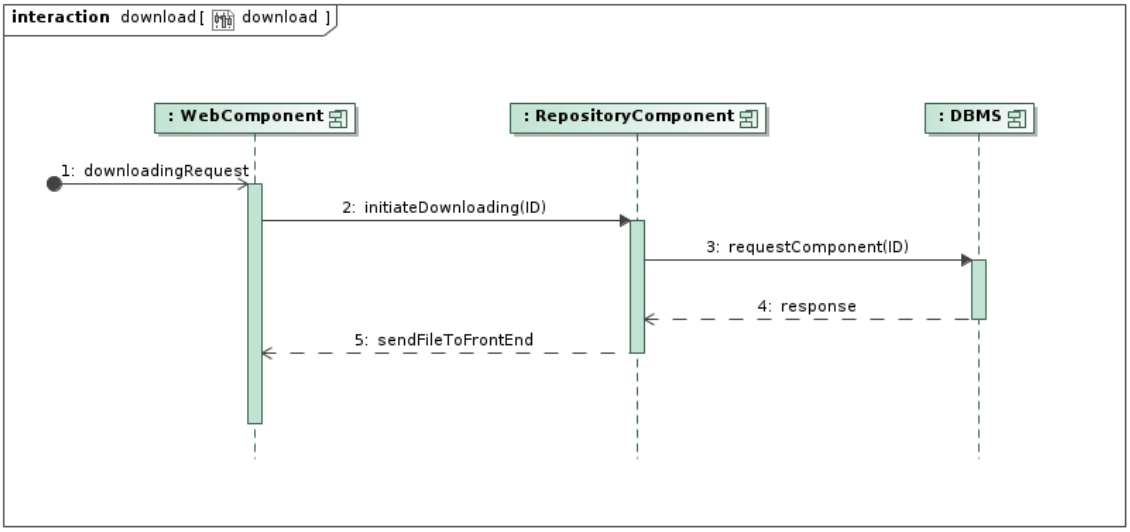

Figure 5: Main flow download event diagram

2.6 Add Operation

Initiator : Administrator
Goal : The admin add a component too the repository database

Main Flow of Events :

1. Admin requests an add operation on the repository.
2. The system sends a notification either in positive or negative case

Exceptions : The metadata structure is not correct. The system displays error message.

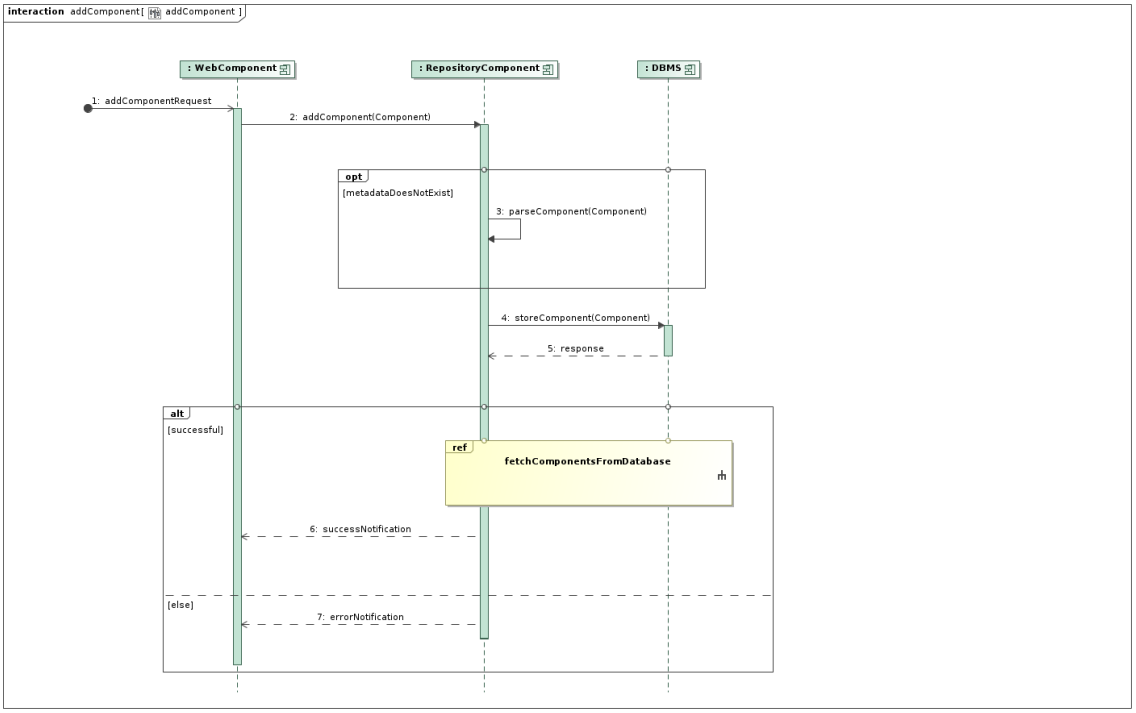

Figure 6: Main flow events component add diagram

2.7 Edit Operation

Initiator : Administrator
Goal : The admin edit a component of the repository database

Main Flow of Events :

1. Admin requests an edit operation on the repository.
2. The system sends a notification either in positive or negative case

Exceptions : Inconsistent request or the metadata structure is not correct or . The system displays error message.

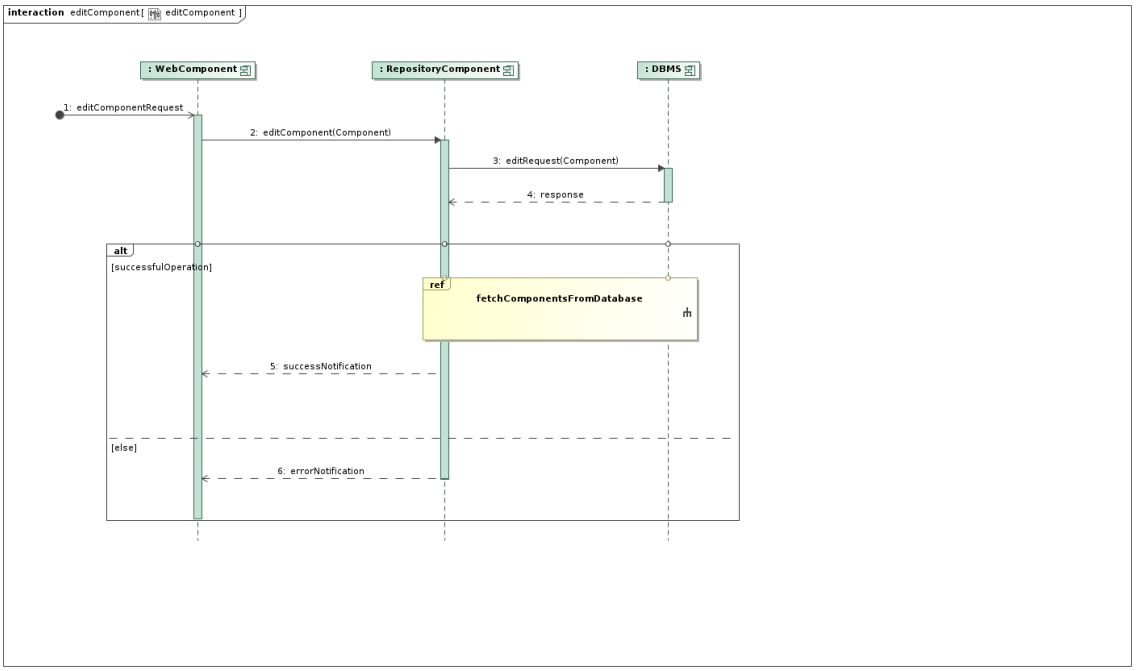

Figure 7: Main flow events component edit diagram

2.8 Remove Operation

Initiator : Administrator
Goal : The admin remove a component of the repository database

Main Flow of Events :

1. Admin requests a remove operation on the repository.
2. The system sends a notification either in positive or negative case

Exceptions : Inconsistent request. The system displays error message.

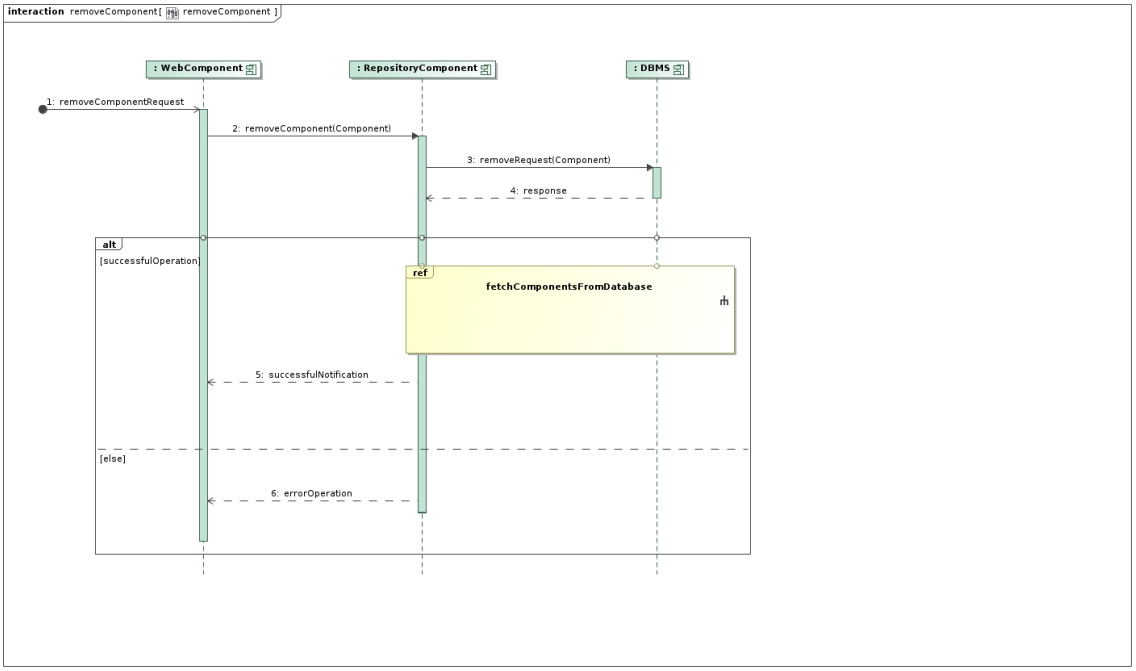

Figure 8: Main flow events component remove diagram

3 External Interfaces

3.1 Web User Interface

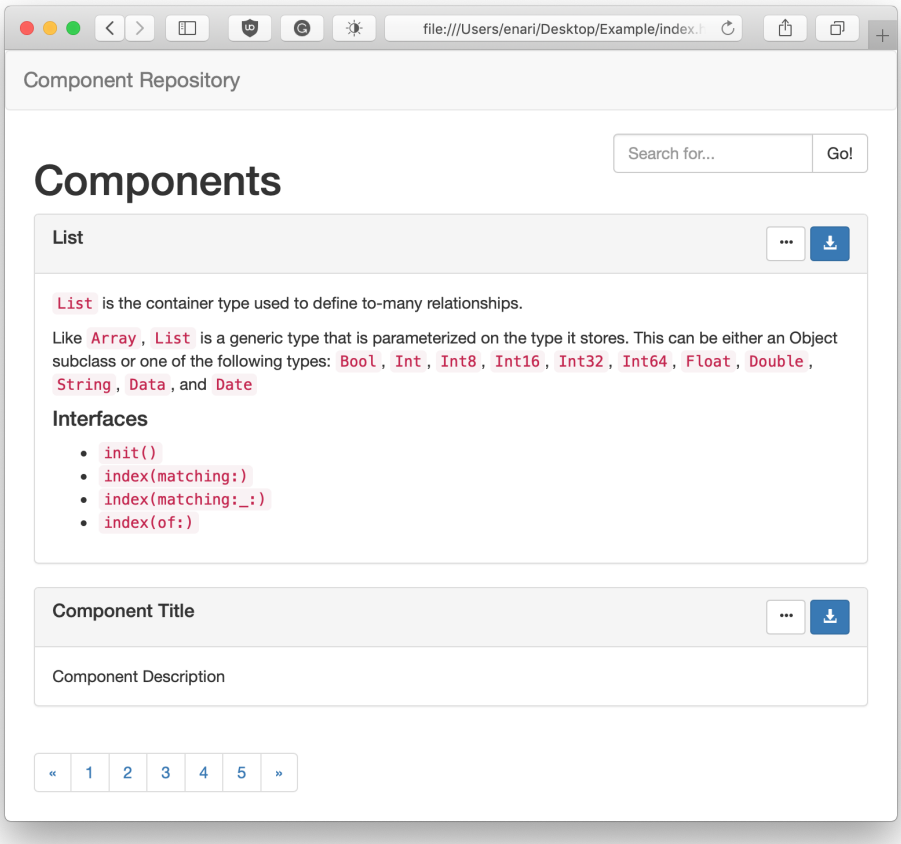

Figure 9: Web Interface

A design prototype of the web interface can be seen in 9. At the top right of the page, there is a search field where users can search for components. When a search is performed the user is presented with the components that match the search criteria. The search takes only the title of the component in consideration, the description is not taken into account.

In the list of components, only the title and a short description is displayed. The user can download the component by clicking on the DOWNLOAD button to the right on the component. In order to inspect a component, to see more information about it, the user can click on the title on the component or on the MORE button. When a component is inspected, a long description and the list of the various classes and interfaces of the component appears. Optionally the dependencies of the component is also shown.

3.2 App Admin Interface

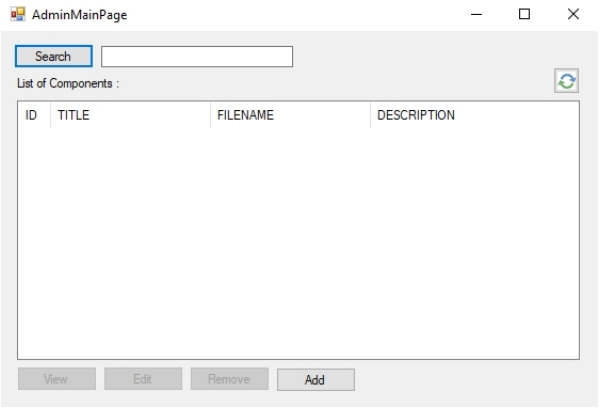

Figure 10: Admin List Window

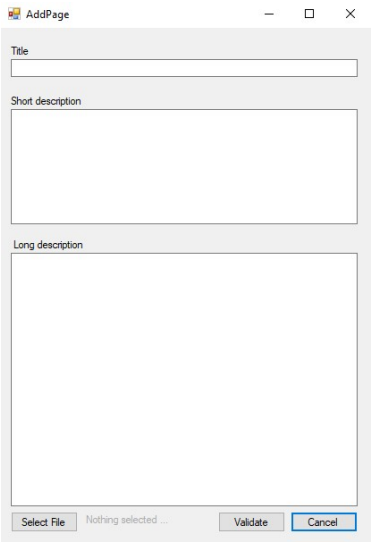

Figure 11: Admin Add/Edit Window

In order to access the repository, the admin have to execute the local application on his/her computer. The application allows the administrator to see the list of the existing components on the database, modify them (Their title, short description, long description or the component itself.), add a new component or remove an existing component.

When the application is executed, the main windows displays the entire list of components. The administrator. It has the possibility to search through the elements of this list with a search bar, in the same way that a standard user can do it on the web interface.

An administrator can add a new component. The window have four specific fields. There are a field for the title, a field for the short description of the component, a field for the long description of the component and a button that allows to search the component on the computer. When all the fields are correctly filled, the administrator can validate is action by clicking on the VALIDATE button.

An administrator can also modify a component. This window that is similar to the previous described window. The only difference is that the field are already filled with the previous values. Thus, the administrator can modify the component. When all the changes are done, the administrator can click on the VALIDATE BUTTON button.
After modification, the list is automatically updated.

A administrator can also make a search. When a search is performed, the list of component is updated to display only components who match the search criteria.

On the right top corner, you can see a refresh button. He is there for update the list view when the database is edited externally, either manually or by another admin session.
You can also use the refresh button too cancel a search and display the list of all components.

4 Software Architecture

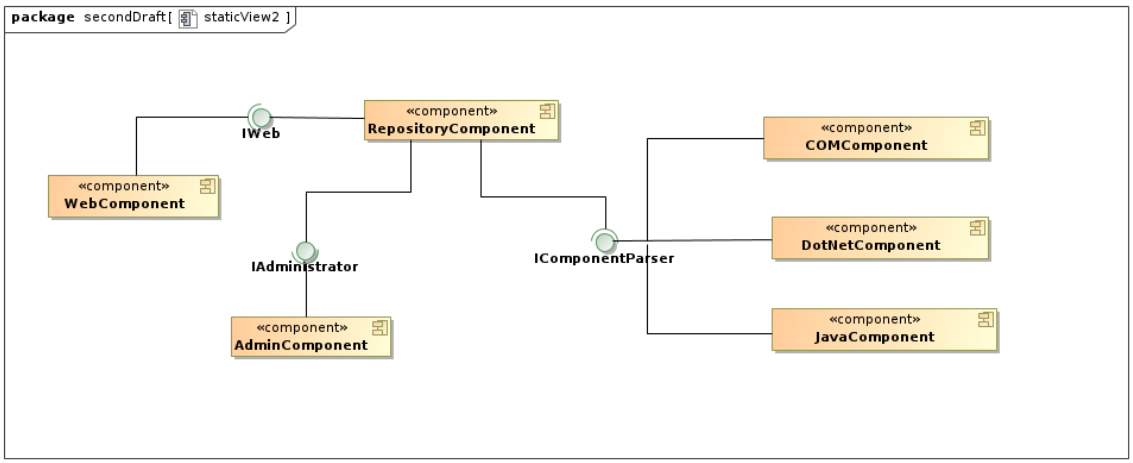

Figure 12: Static View

4.1 Overview and Rationale

We have chosen to divided the application in different components to increase reusability, flexibility, and organization. Visual users interfaces and applications functionalities are provided by different components. Moreover, to parse and analyze components we need to use 3 components (One to parse Com components, one to parse java components and the last one to parse dot net components).

4.2 System Decomposition

The system is divided in 6 components :

- **Web Component:** The goal of this component is to provide access to user's functionality and manage the web graphical interface.
- **Repository Component:** This component is the main component. He provides users' and admin's interfaces and manages all functionalities of the application. He also manages transactions carried out on the database.
- **Admin Component:** The goal of this component is to provide access to administration's functionality and manage the desktop graphical interface.
- **Com Parser:** This component implements a com parser to extract information about com components and writes this informations into a JSON FILE.
- **DotNet Parser:** Like the COM parser but for .NET components.
- **Java Parser:** Like the Com Parser ans the Java Parser but for Java components.

4.3 Persistent Data

Our system requires persistence to store uploaded component files. These uploaded files are archive files from different technologies. We are accepting EJB, COM and .NET files to be saved. The files extension from the enumerated technologies are .jar and .dll. According to this requirement, our model contains one table (Figure 11)

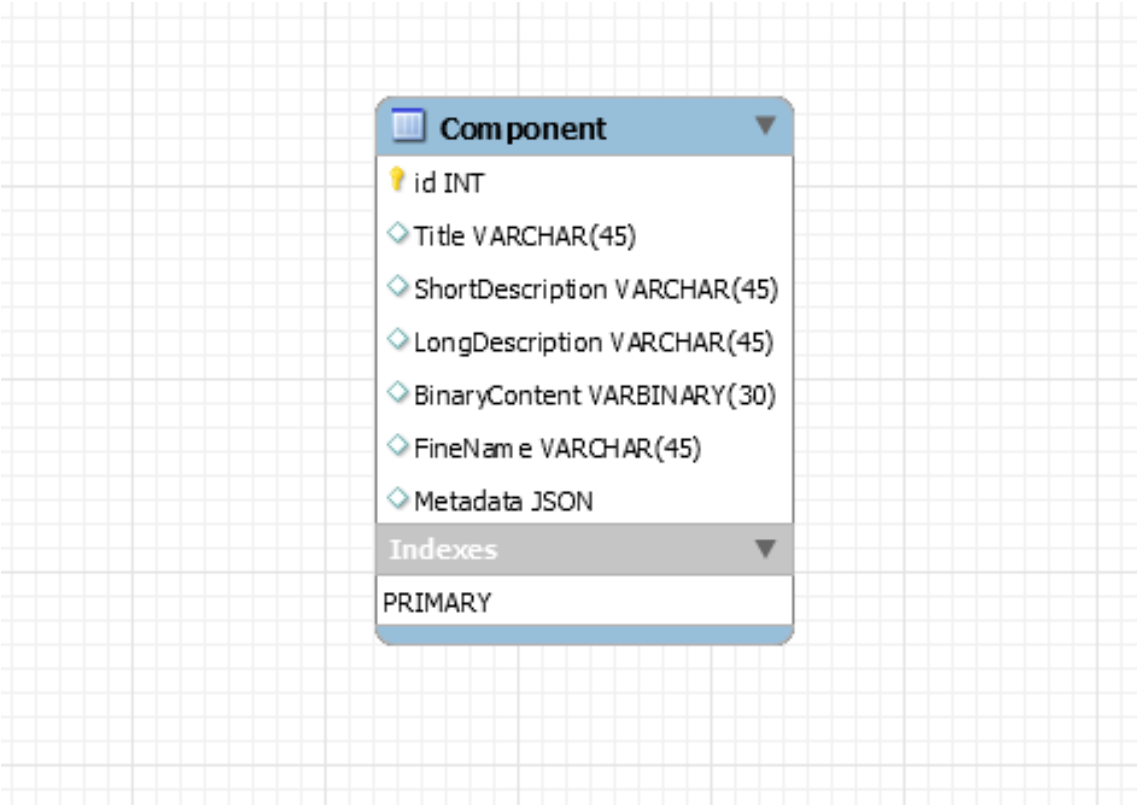

Figure 13: Database Model

The table represents a component. The components has unique identifier. In our case, we took Id field of type INT for primary key. Alongside with the field Id, our component has a *Title* of type VARCHAR,a *ShortDecrption* of type VARCHAR, a *LongDescription* of type VARCHAR. Main field of the Component model is a *BinaryContent* of type VARBINARY. In this field we are persisting the file as a whole. Then, in the field *FileName* of type VARCHAR we store the filename of the component. In order to fulfill our functional requirements and show the interfaces and classes of parsed *BinaryContent* to a web application user, we are persisting parsed metadata in field *Metadata* of type JSON. The JSON is as a set of nested dictionaries and arrays. An example of how the JSON could look is shown in listing 1

```
{
    "classes": {},
    "interfaces": {
        "Collection": [
            "subscript(position: Self.Index) -> Self.Element { get }",
            "mutating func popFirst() -> Self.Element?",
            "mutating func shuffle()"
        ],
        "MutableCollection": [
            "mutating func reverse()",
            "mutating func sort()"
        ]
    }
}
```

Listing 1: Example of how the metadata JSON could look

5 Detailed Software Design

5.1 Static Structure

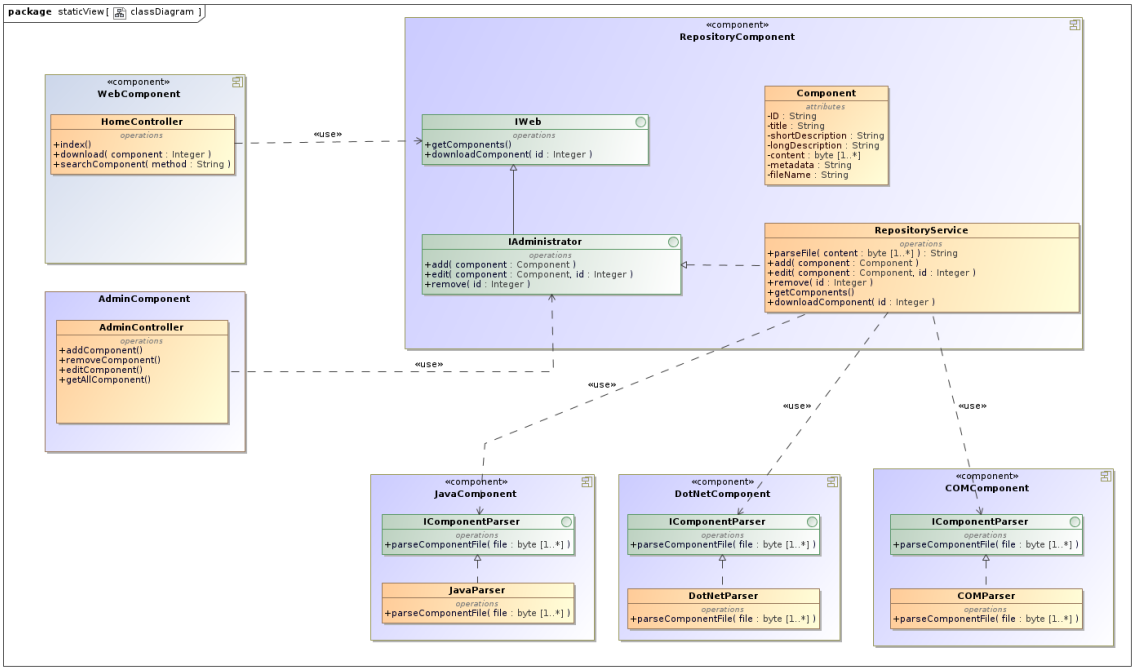

Figure 14: Class Diagram

5.2 Components

5.2.1 WebComponent

Classes :

- HomeController : IWeb
  - `index()` : Get the view.
  - `download(component: int)` : Downloads a component by passing its ID in parameter.
  - `searchComponent(method : string)` : Seek a component by passing the string of the text field.

5.2.2 AdminComponent

Classes :

- AdminController : IAdministrator
  - Properties : None.
  - Methods :
    - `addComponent()` : Add a component.
    - `removeComponent()` : Remove a component.
    - `editComponent()` : Edit a component
    - `getAllComponent()` : Get all components.

5.2.3 RepositoryComponent

Classes :

- Component
  - Properties :
    - `title : string` : The title of the component.
    - `shortDescription : string` : A short description that describe the component.
    - `longDescription : string` : A long description of the component
    - `content : byte[]` : The code of the component.
    - `ID : string` : An unique identifier that identify the component.
    - `metadata : string` : JSON string that parse the code of the component.
  - Methods : None.
- RepositoryService
  - Properties : None.
  - Methods :
    - `parseFile(content : byte[])` : Parse a the code of a component into a JSON file.
    - `addComponent(fileObject : Component, componentID : int)` : Add the component passed in parameter to the database.
    - `editComponent(fileObject : Component)` : Edit in the database the component that match the ID of the component in parameter.
    - `removeComponent(fileObject : Component)` : Remove from the database the component that match the ID of the component in parameter.

5.2.4 JavaComponent

Classes :

- JavaParser : IComponentParser
  - Properties : None.
  - Methods :
    - `parseComponentFile(file : byte[])` : Parse a the code of a Java component into a JSON file.

5.2.5 DotNetComponent

Classes :

- DotNerParser : IComponentParser
  - Properties : None.
  - Methods :
    - `parseComponentFile(file : byte[])` : Parse a the code of a .NET component into a JSON file.

5.2.6 COMComponent

Classes :

- COMParser : IComponentParser
  - Properties : None.
  - Methods :
    - `parseComponentFile(file : byte[])` : Parse a the code of a COM component into a JSON file.

5.3 Interfaces

5.3.1 IWeb

Methods :

- `getComponents()` : Get all the components of the database.
- `downloadComponent(componentID : int)` : Download the component with the ID in parameter.

5.3.2 IAdministrator

Methods :

- `addComponent(fileObject : Component)` : Add an the component in parameter to the database.
- `removeComponent(ID : int)` : Remove the component identify by the ID in parameter from the database.
- `editComponent(fileObject : Component)` : Edit the component in parameter in the database.

5.3.3 IComponentParser

Methods :

- `parseComponentFile(file : byte[])` : Parse a the code of a component into a JSON file.