Group no: 1
Date: 10.04.2018
Proj. mgr: Vladimir Djukanovic
Email: vdc16001@student.mdh.se

DESIGN DESCRIPTION

Project **Online Component Repository**

DISTRIBUTION

Steering group:

Frank Lüders

Project group:

Vladimir Djukanovic

Cristian Capozucco

Oskar Palmgren

Aleksandar Matovic

Bastien Delbouys

Mohamed Abdi

CONTENTS

1 Introduction

1.1 Background

This report describes the design of a software component repository that is a part of the course Component-based Technologies.

The system is divided into two applications. One of them is a web-based application that lets the users browse, search and download components. The other is a standalone windows application for administrators that enables adding, removing and editing (description editing) of the components.

The software architecture is a component-based model-view-controller architecture written in C#. Some components must however be written in C++ and Java to access information from COM component and JavaBeans components.

1.2 Definitions

Terms	**Definitions**

1.3 Related Documents

Document identity	**Document title**

2 Functional Description

2.1 Use Case Model

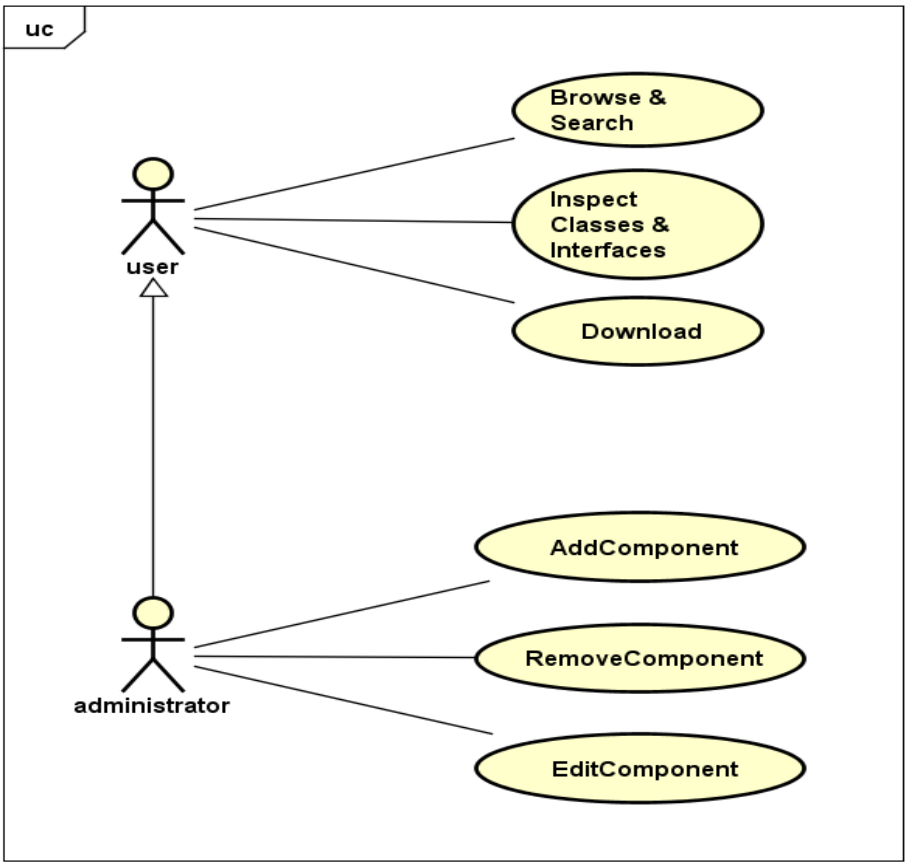

Figure 1 - Use Case Diagram

2.1.1 Actors

As it can be seen in Figure 1, actors in this system are Users and Administrators. Users (or more precisely visitors) can browse and search, inspect and download components. Administrators can perform all actions as users and some additional actions. They are able to perform actions as adding, editing and deleting components.

2.1.2 Use Cases

Each use case is described in a separate section in the remainder of this chapter.

2.2 Browse & Search

2.2.1 Participating Actors

User

2.2.2 Related Use Cases

2.2.3 Precondition

There is no precondition in order to be able to perform this use case.

2.2.4 Main Flow of Events

1. The user search for the name of a component
2. The system looks into the component database for any correspondences
3. The system displays the list of components that match the search

2.2.5 Alternative: No component found

- At step 2 : If no component match the search

The system displays a message to the user to inform him that no component matches the search.

- Return to step 1

2.3 Inspect Classes & Interfaces

2.3.1 Participating Actors

User

2.3.2 Related Use Cases

Browse & Search

2.3.3 Precondition

A precondition for this use case might be searching for the certain component (Browse & Search).

2.3.4 Main Flow of Events

1. The user selects one of the displayed components.
2. The user clicks the “INSPECT” button.
3. The system display component’s Classes and Interfaces.
4. The user has to clicks the “RETURN” button to end the case.

2.4 Download

2.4.1 Participating Actors

User

2.4.2 Related Use Cases

Browse & Search

2.4.3 Precondition

A precondition for this use case might be searching for the certain component (Browse & Search).

2.4.4 Main Flow of Events

1. The user selects one of the displayed components.
2. The user clicks the “DOWNLOAD” button.
3. The system checks the availability of the component.
4. The system sends the component to the user.

2.4.5 Alternative: The component is not available

- At step 3 : The component is not available for the download.

The system displays a message telling the user than the component in not available for the download.

- The system return to the previous user’s search.

2.5 AddComponent

This use case is responsible for adding components to the repository and displaying a component in the UI if the upload was successful.

2.5.1 Participating Actors

The initiating actor is administrator and only the user with admin privileges can perform this use case.

2.5.2 Related Use Cases

Based on the belonging subsystem the related use cases is/are:

- Remove Component and
- Edit Component

2.5.3 Precondition

There is no precondition in order to be able to perform this use case.

2.5.4 Main Flow of Events

1. The administrator initiates to add the component (by pressing a button etc.)
2. The system gives the user the form to upload component
3. The administrator fills the form and press the upload button

4. The system checks the validity of form
5. The system stores the component in the database
6. The system displays newly added component

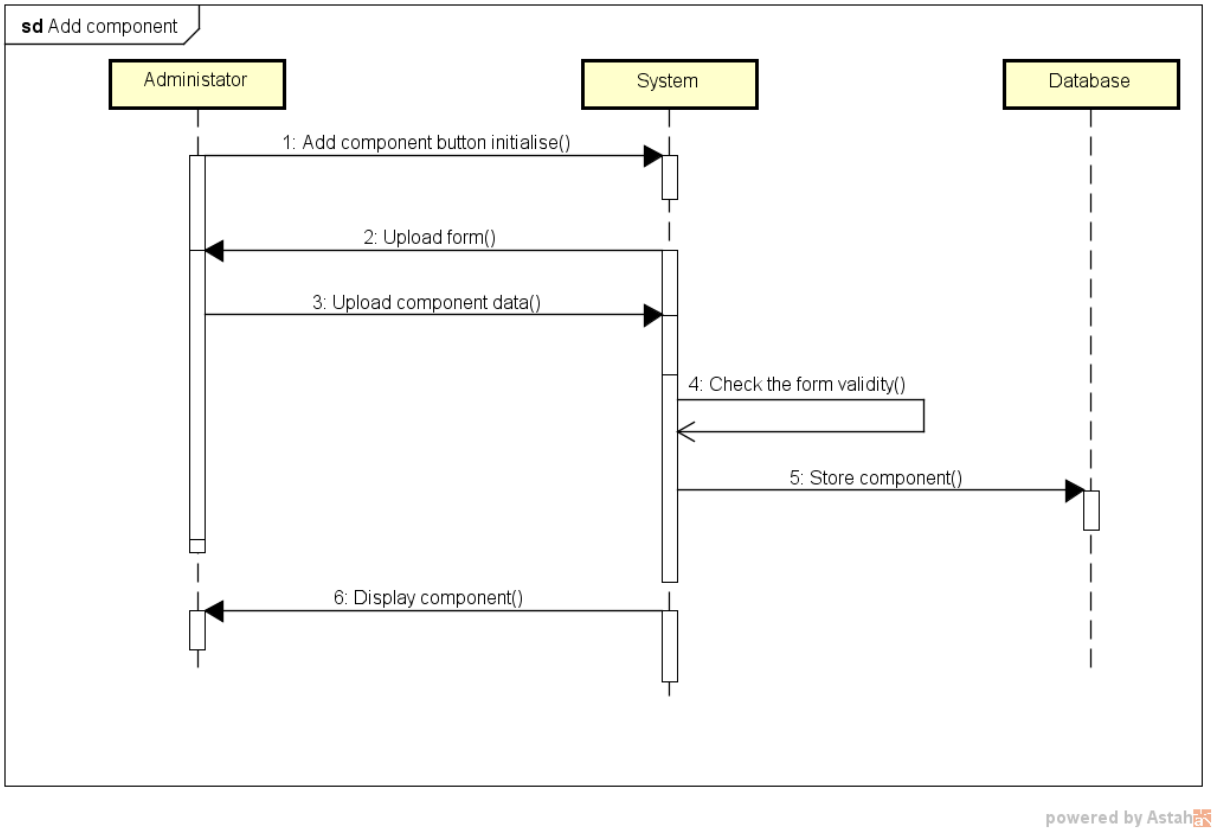

Figure 2: Add Component sequence diagram

2.5.5 Alternative: The upload component form is invalid

- At step 4: Form is not valid, all data not provided or invalid data.
  The system displays the error committed by the user in the form.
- Return to step 3

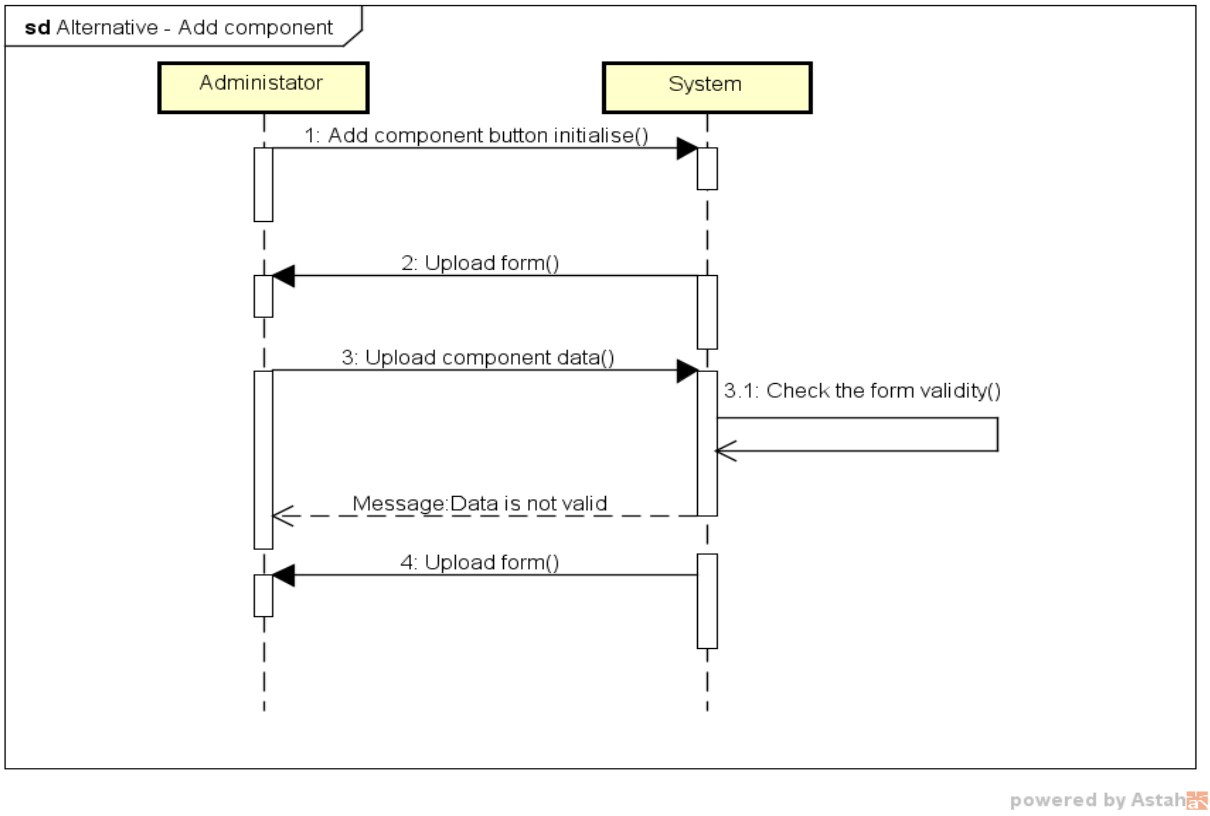

Figure 3: Alternative sequence diagram (form invalid)

2.6 RemoveComponent

This use case is responsible for removing components from the repository and updating UI if the removal was successful.

2.6.1 Participating Actors

The initiating actor is administrator and only the user with admin privileges can perform this use case.

2.6.2 Related Use Cases

Based on the belonging subsystem the related use case/s is/are:

- Add Component and
- Edit Component

2.6.3 Precondition

There is no precondition in order to be able to perform this use case.

2.6.4 Main Flow of Events

1. The administrator selects the component and clicks remove button

2. The system asks for confirmation to remove selected component
3. The administrator clicks to confirm removal
4. The system removes the component from database
5. The system updates UI

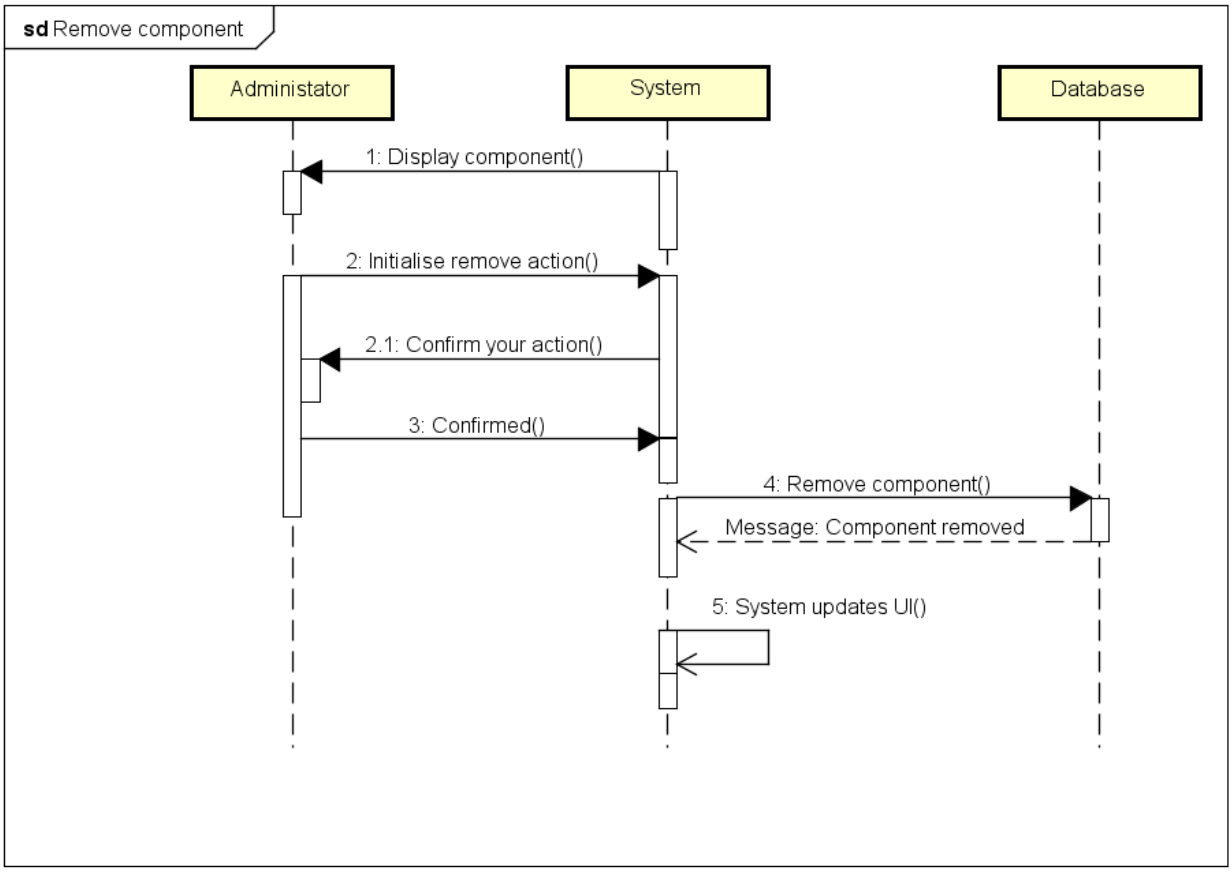

Figure 4: Remove Component sequence diagram

2.6.5 Alternative: Cancel confirmation panel for component removal

- At step 3: The administrator clicks to cancel component removal
  The system closes the confirmation panel.
- The system returns to home page.

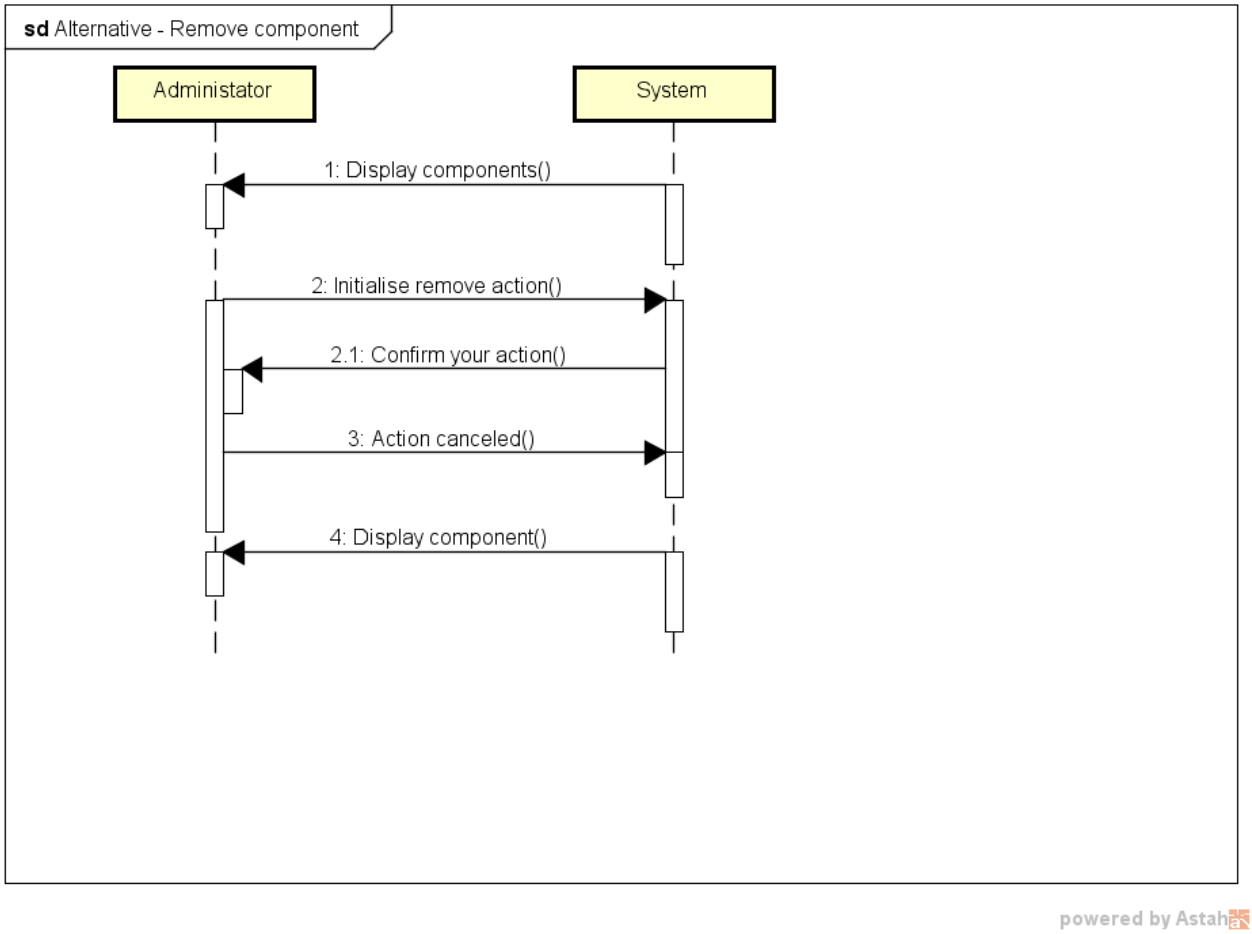

Figure 5: Alternative sequence diagram (cancel confirmation)

2.7 EditComponent

This use case is responsible for editing component information and therefore updating the repository and UI if the editing was successful.

2.7.1 Participating Actors

The initiating actor is administrator and only the user with admin privileges can perform this use case.

2.7.2 Related Use Cases

Based on the belonging subsystem the related use cases is/are:

- Remove Component and
- Remove Component

2.7.3 Precondition

There is no precondition in order to be able to perform this use case.

2.7.4 Main Flow of Events

1. The administrator clicks the edit button for specific component
2. The system gives administrator the form to edit component information
3. The administrator changes the form and clicks to update component information
4. The system checks the validity of the form
5. The system updates the component in the database and updates UI

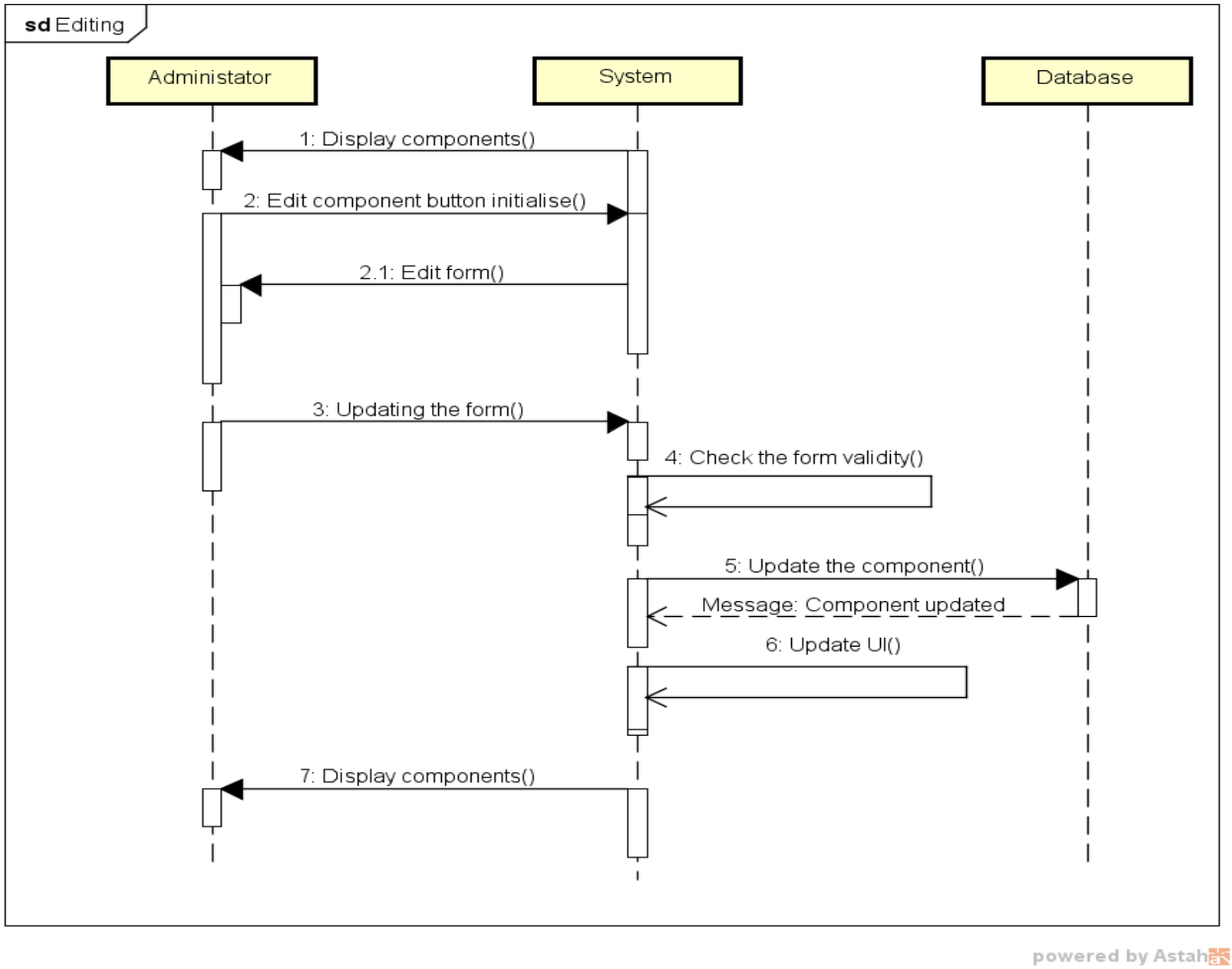

Figure 6: Edit Component sequence diagram

2.7.5 Alternative: The updated form is invalid

- At step 4: Form is not valid, all data not provided or invalid data.
  The system displays the error committed by the user in the form.

- Return to step 3

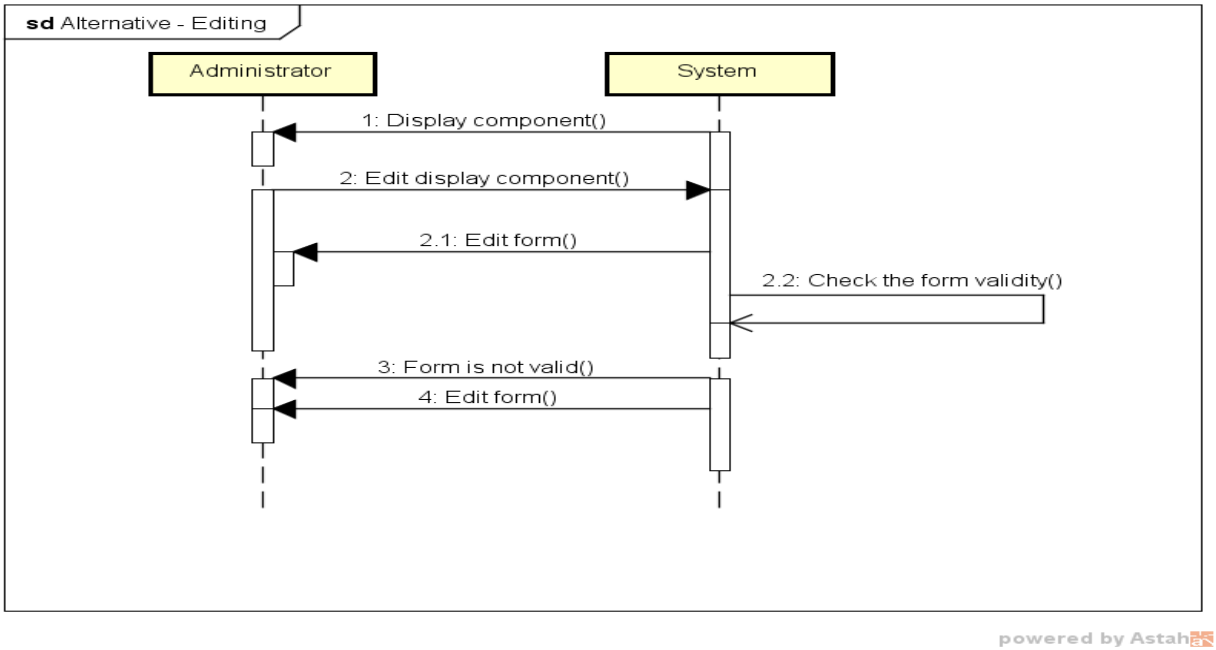

Figure 7: Alternative sequence diagram (form invalid)

3 External Interfaces.

3.1 Graphical User Interface

3.1.1 Administrator UI - Windows Forms App (.NET)

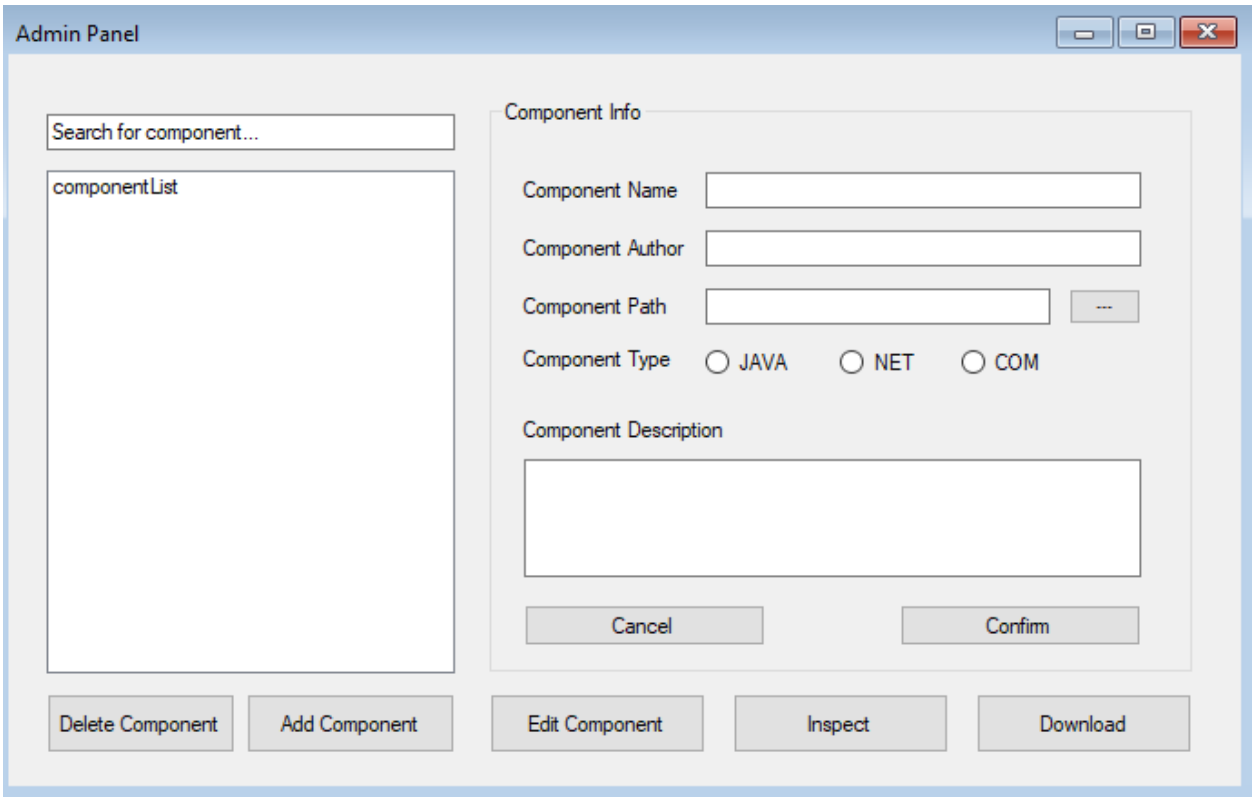

Figure 8 - Admin Panel

If a user is an administrator of the system, then he can access this panel. On the left, a *list box* shows every component registered in the system. If the admin selects one of them and clicks the *Delete component* button, then the component is deleted from the system. If a component is selected, then every information regarding it are showed in the textboxes on the right and the admin can edit them by clicking *Edit component button.* When component is selected admin can download component and inspect component and also delete component by clicking on corresponding buttons. Also, if the editing is not active the admin can add new components which will also deselect any selected component in the list.

3.1.2 VISITOR (USER) WEB UI - ASP (.NET)

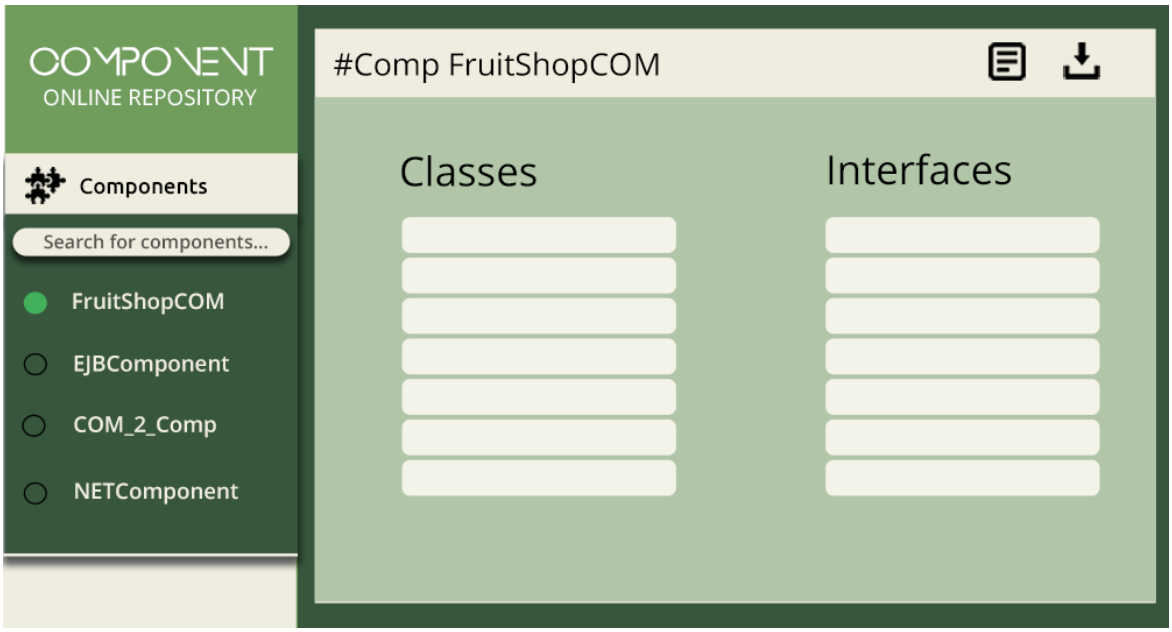

Figure 9 – Web UI

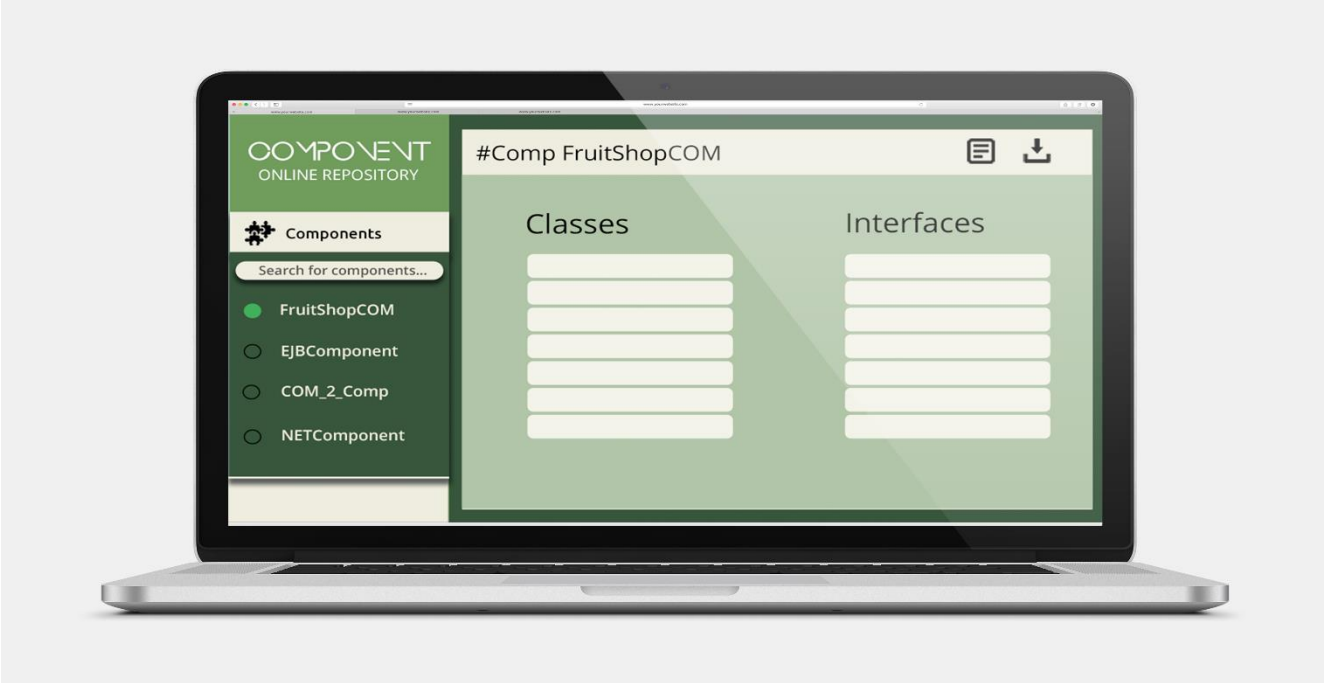

Figure 10 – Web UI Mock

On the Figure 5 and Figure 6 we can see the WEB UI application for visitors or all the users that interact with the components (with view permission). The list of all components can be viewed on the left side. Above the list of components, there is

a field for searching by name of the component. On the selection of the component, inspection is automatically done by updating all the classes and fields in the central component panel. Beside classes and interfaces, name of the component and description will also be visible in this panel area. Also, in the right top corner of the component panel there is a button for downloading the selected component. Beside download button there is also a button to view the component info (information about author, path and so on). This summarizes the available possibilities for user interaction with the components.

4 Software Architecture

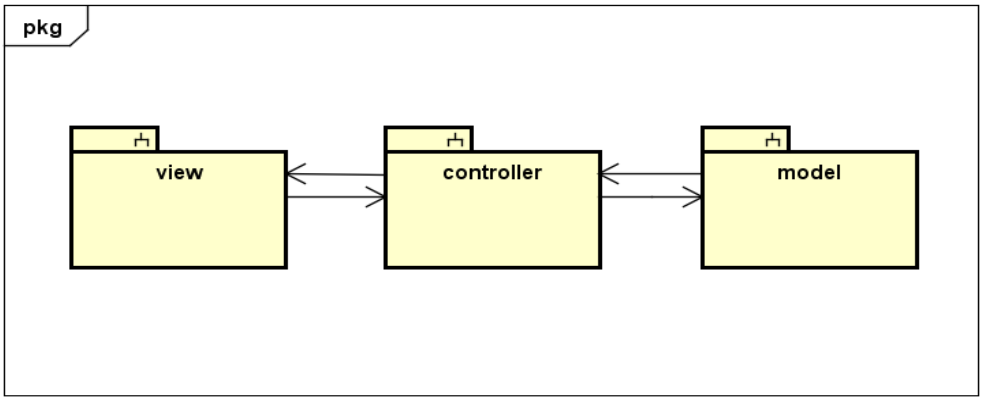

Figure 11 – Software Architecture (model-view-controller)

4.1 Overview and Rationale

- The system is designed with a model view controller architectural style.
- Model part presents a bridge with database by connecting database tables with model representations.
- Controlling Unit part provides the main system functionality and logic that handles the views and communication with database through model (database unit).
- Views are different GUIs that will be showed to the end user defined by different user interactions.
- To inspect JAVA Component with reflection, a component called ikvm will be used. Thanks to this library, it possible to convert the java component responsible for inspection from .jar to .dll. So, it can be directly used from .net components.
- To retrieve information from the components written in .net a combination of System.Reflection and System.Type libraries is used. The System.type library retrieves information about the methods and properties of the component and stores them in an array of type MethodInfo and PropertyInfo.

4.2 System Decomposition

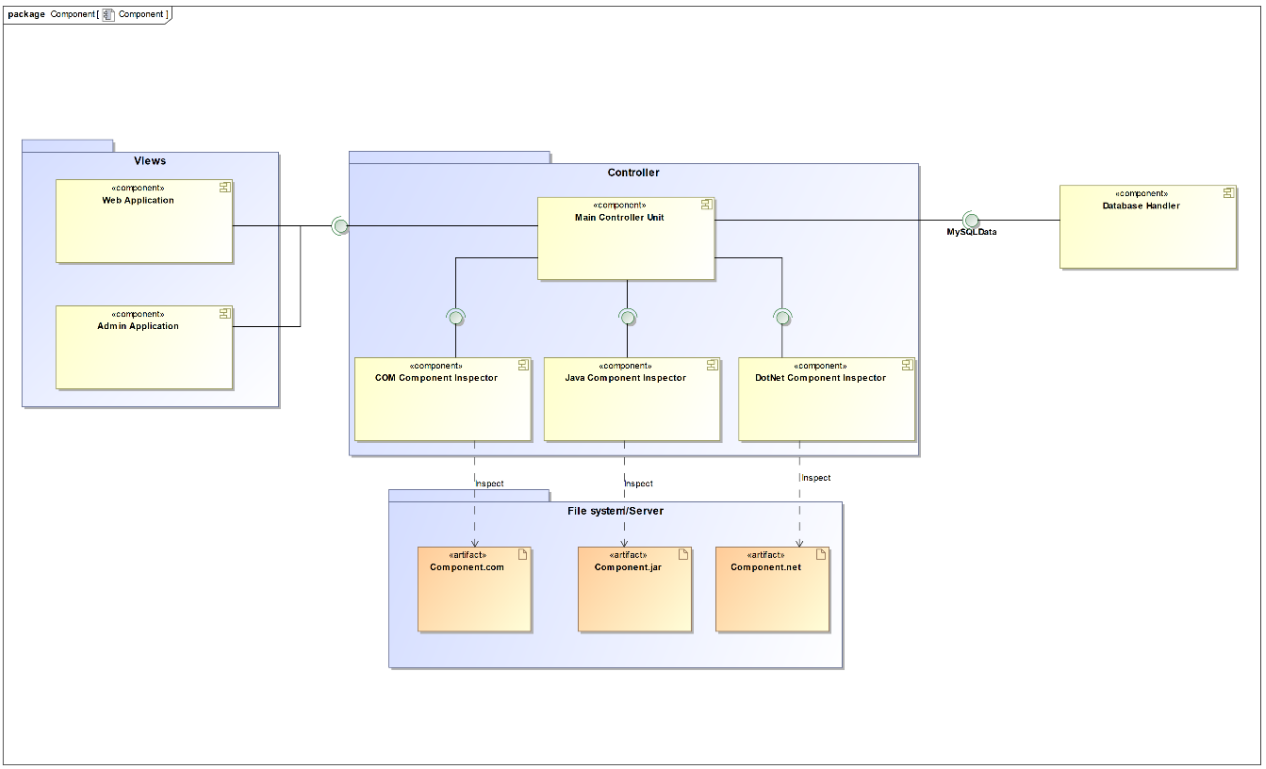

Figure 12 – Component Diagram

4.3 Hardware/Software Mapping

ASP Web Application system uses following external dependencies:

- Materialize Framework

JAVA Reflection is handled by using following dependencies:

- Java Reflection API. IKVM.NET

4.4 Persistent Data

Persistent data is data that does not change across time, systems and memory. In case of the online Component Repository components are the persistent data. Furthermore, components source code cannot be change in any way. When the administrator uploads the component, system handles its source code and redirect component from end to end (i.e. from the user computer to server where is eventually the component stored). Having that in mind, source code of component is not modified throughout this process. However, user has an option to modify the component, but we must emphasize that this option is only possible for modifying component description without changing the component itself.

4.5 Access Control

The system is accessed directly for both applications without login/register capability.

4.6 Synchronization and Timing

Since the admins can delete components from the database, the system must block this operation if a user is downloading a component and an admin wants to delete it. After the download finish/abort, the system allows the admin to remove that component from the repository.

Another thing to consider: a component can be browsed and downloaded if and only if its uploading is correctly finished. That is, component information cannot be displayed until it is in the repository.

4.7 Start-Up and Shut-Down

The application should be available on the web so it needs to run on a server that have a good uptime. This is because the repository should be available to users so they can browse and download components.

There can be a Shut-Down in two cases:

- The server is down for maintenance.
- The server is down because of hardware/software failures

4.8 Error Handling

The system is designed to recover from any unexpected errors caused by user false inputs or system malfunction. When the specific error occurs system will graphically display appropriate message with explanation for the user. System is designed to narrow down the possibility for users mistake with a proper explanation about particular function.

Four major categories of errors are predicted in the system:

- Logical errors
- Generated errors
- Compile-time errors
- Runtime errors

As errors could be fatal, error handling is one of the crucial areas of development in general. Hence, we pay close attention to this problem.

5 Detailed Software Design

5.1 Web application

5.1.1 Static Structure

- When the user accesses the web application, he is able to see system UI. On the left corner there is a system description (e.i. Component online repository)
- On the left side (bellow system description) user can see all available components. However, if there is none of the components in the moment. System will display appropriate information. i.e . “None of the components are available at the moment”
- Above the components tab user can see input field for purpose of searching the component
- In the main application window user can see the field with component description, also in the right corner download button is located.
- Also, a view component information button is located next to the download button.

5.1.2 Dynamic Behaviour

- User can interact with the components by selecting them. After this action, in the main window the system displays classes and interfaces of selected component. User has permission to view the components but he is unable to interact with them in any way.
- User has an option to search and find the required component.
- User can download the component by clicking the download button. After this user action, system send the command for download to the server side. If everything is done successfully user has downloaded the chosen component.
- User can view the component information (author, path, and so on) by clicking on the view button next to the download button.

5.2 Admin Application

5.2.1 Static Structure

- When the administrator accesses the administrator application, he is able to see system UI which represents admin panel.
- On the left side administrator can see all available components in the area that is marked in Figure 9 as *componentList*. If the list is empty than the initial information will state certain appropriate message to inform the user that there are no components available currently. Also, above the list there is a search field that enables searching for components.
- On the right side, we have component info section that contains all descriptive information (text fields, radio buttons, ..) such as component name, author type, path and component description. All these are initially empty. It includes initially disabled buttons, Confirm and Cancel (that becomes active when adding component or editing an existing component).
- Under these two sections, we have buttons for adding, editing, removing component and also inspect and download buttons. The buttons for adding and removing are initially disabled until selection action is performed for certain component.

5.2.2 Dynamic Behaviour

- User can interact with the components by selecting them in the component list field, as presented in the static behaviour section. After this action is performed, the User has permission to view the component information in the component info (section group on the right side of panel). This includes name, path, author, type and description. All controls that are part of the component info group will be disabled until the button Edit component or Add Component are clicked.
- When the button *Edit component* is clicked (that will become enabled after the component is selected), all fields will become editable in the component info section. By clicking on *Confirm* button, the changes will be updated. By clicking on *Cancel* button, the changes will be rejected, and editing will be canceled (fields in component info area will become disabled). Also, while editing is active, buttons for adding component and removing will be disabled.
- When the component is selected, admin can click to delete the component which will ask for confirmation before deleting is performed. This button will be disabled in case no component is selected.
- The admin can click *AddComponent* button to add new component but only if editing is not active. This will deselect any selected component in the list and therefore disable buttons for editing, deleting, inspecting and editing component.
- User has an option to search and find the required component. This controls is located above the *componentList* area.
- Beside that user can also view the list of classes and interfaces by clicking on the *Inspect* which will open a dialog with classes and interfaces.

- User can download the component by clicking the download button. After this user action, system send the command for download to the server side. If everything is done successfully user has downloaded the chosen component.

5.3 Controller and main functionality

5.3.1 Overview of the subsystem

There are four components in this subsystem:

1. Main Controller Unit
2. DotNet Component Inspector
3. Java Component Inspector
4. COM Component Inspector

The Main Controller Unit is actually a bridge between the View Subsystem and the Model subsystem. This logic part provides an interface to the view components (mainControllerInterface) and requires the interfaces from the components 2,3,4 of the above-mentioned list and from the DatabaseHandler component. Thanks to this component the system will be able to:

- Allow the Admin to perform CRUD operations on components by interacting with a simple GUI;
- Allow the Normal User to browse, search, inspect and download components through a simple GUI;

The main goal of the components 2,3,4 is the same: inspect the components stored in the database and retrieve information concerning the Classes and the Interfaces provided by them.
They will be realized using different technologies:

- C# for the Component 2
- Java for the Component 3
- C++ for the Component 4

Concerning the Java inspector, it will be converted in a .DLL file thanks to an External Component called IKVM: this component can convert a .JAR/.CLASS file into a .DLL file that can be used later by .NET technologies.
For the C++ inspector, the LoadTypeLibEx function will be used to get an ITypeLib interface pointer, from this pointer we could see the component properties and methods, then the component could be converted from COM to .NET using the TypeLibConverter.ConvertTypeLibToAssembly method.

Since they are related to each other, the Static and the Dynamic structure will be described together.

5.3.2 Static Structure

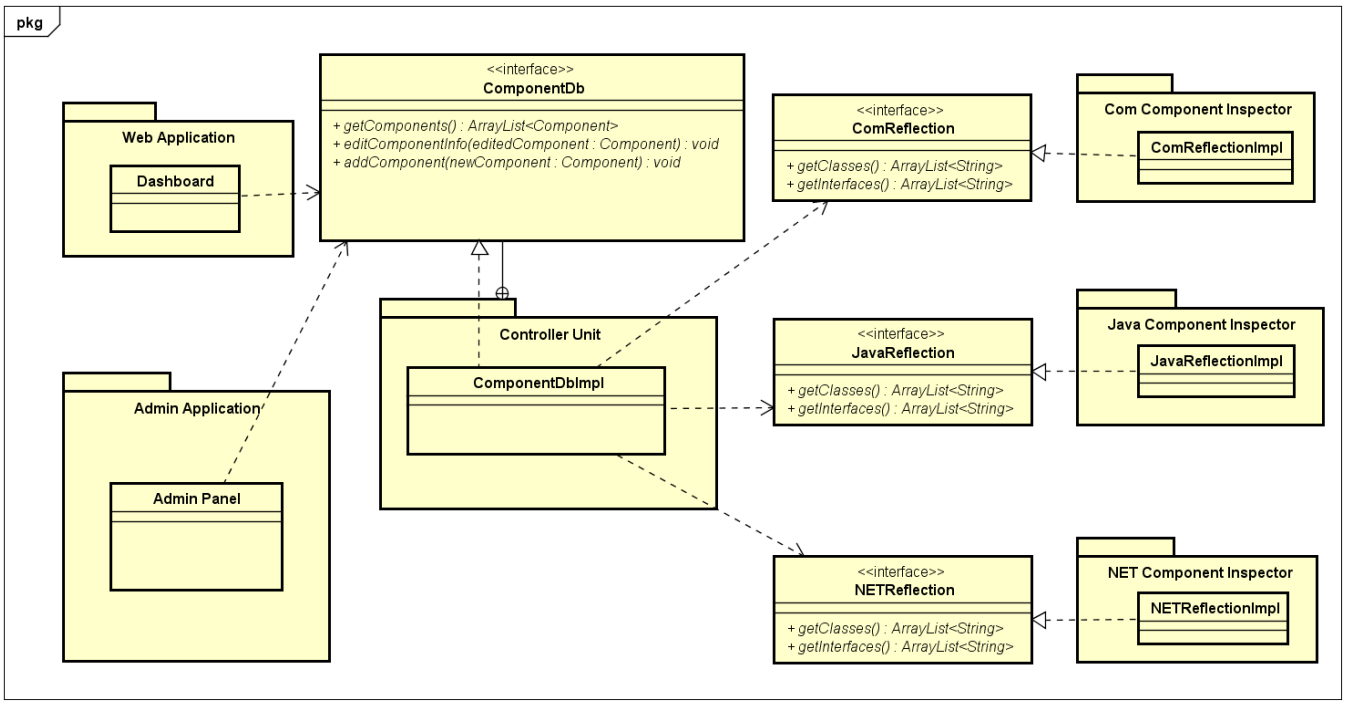

Figure 13 – Class diagram for main system functionality

We will briefly describe the included classes in Figure 13:

- On the left side, we have two packages: Web Application package and Admin Application. As part of the Web Application package we have the Dashboard class which represents a partial class that handles the UI for the web application. We see that this class uses the ComponentDb interface (main controller interface) that handles all the logic directly or indirectly. In other words, all communication with application views is accomplished through this controller interface. Same can be said for the Admin Panel class that is part of the Admin Application package and represents a partial class that handles the UI (Win Forms) for the administrators' application.
- In the middle section of the class diagram we have Controller Unit package that is responsible for handling the main logic of the application. It consists of ComponentDbImpl class and corresponding interface for this class that is a bridge that VIEWS are using for communication with the Controller Unit to get all the necessary data from.
- On the right side, we have reflection classes (ComReflectionImpl, JavaReflectionImpl, NETReflectionImpl) and corresponding interfaces that *ComponentDbImpl* is using for the communication with reflection classes in order to be able to handle appropriate reflections for different component types (components implemented in different programming languages).

5.3.3 Dynamic Structure

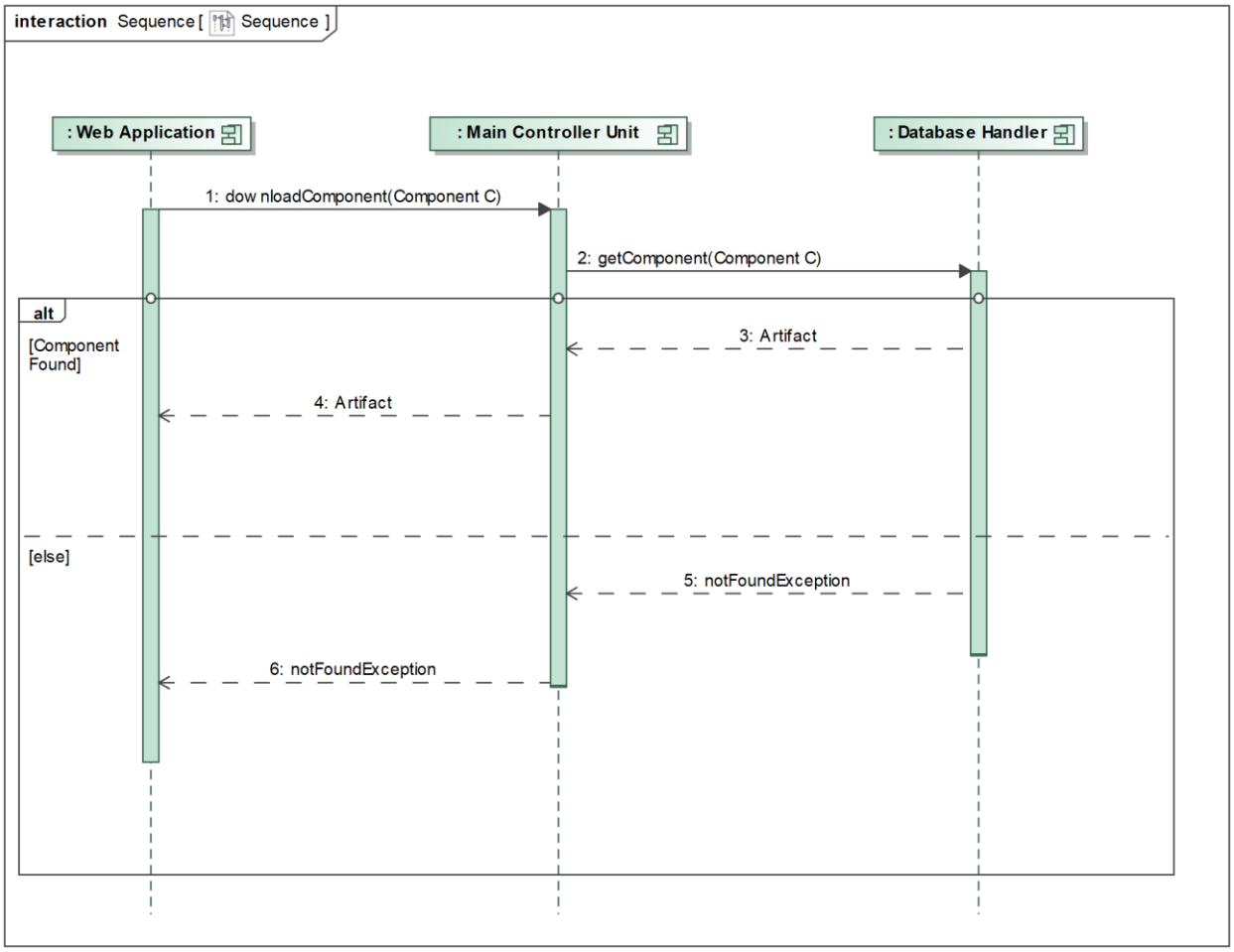

Figure 14 – Sequence diagram: download component scenario

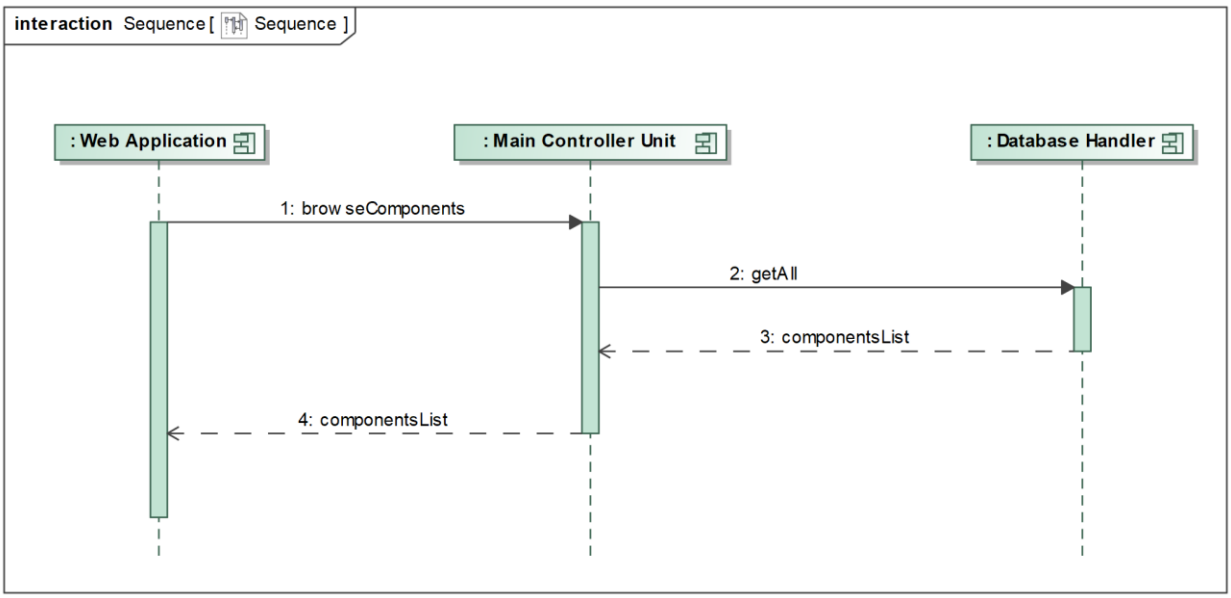

Figure 15 – Sequence diagram: browsecomponents scenario

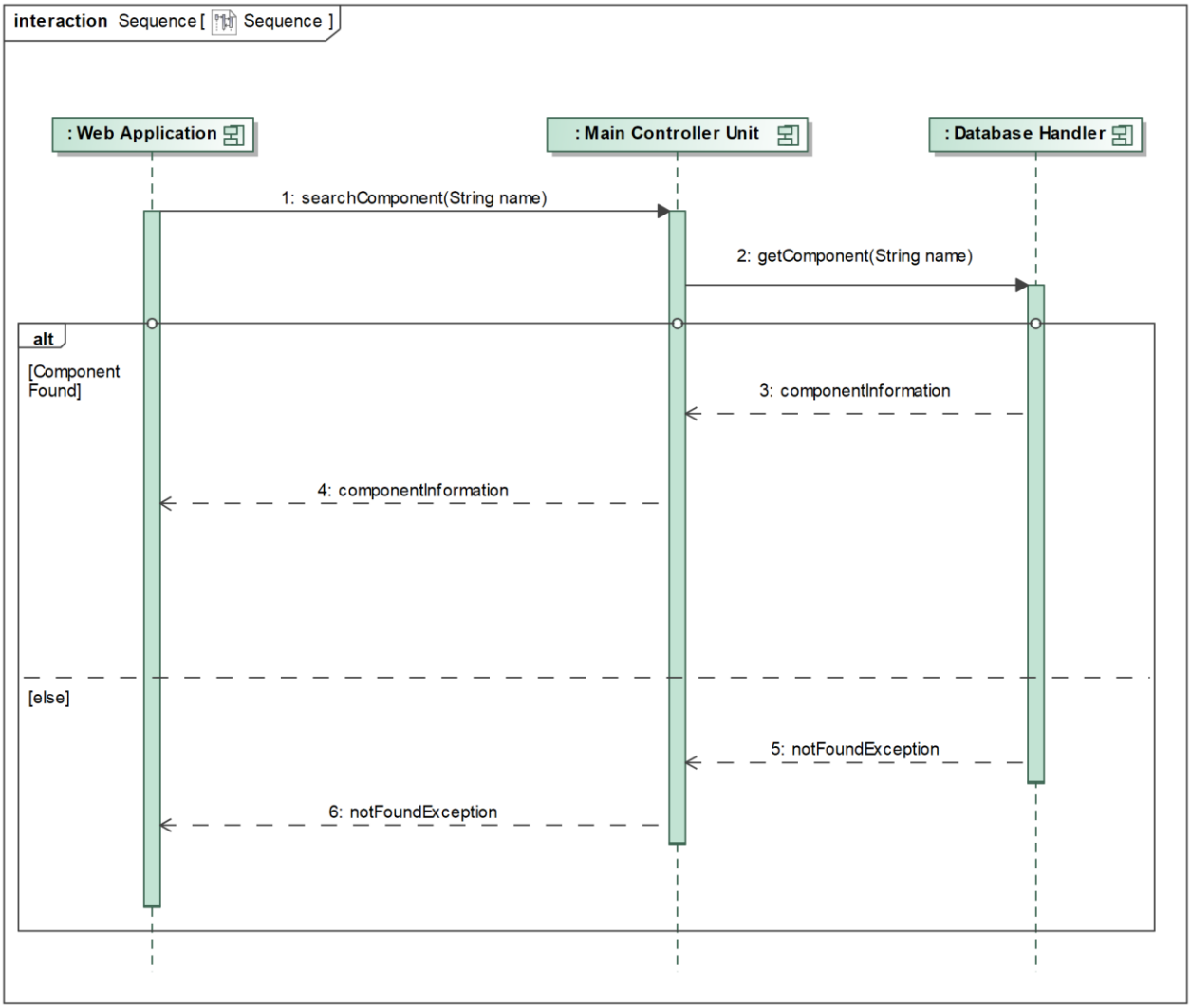

Figure 16 – Sequence diagram: search component scenario

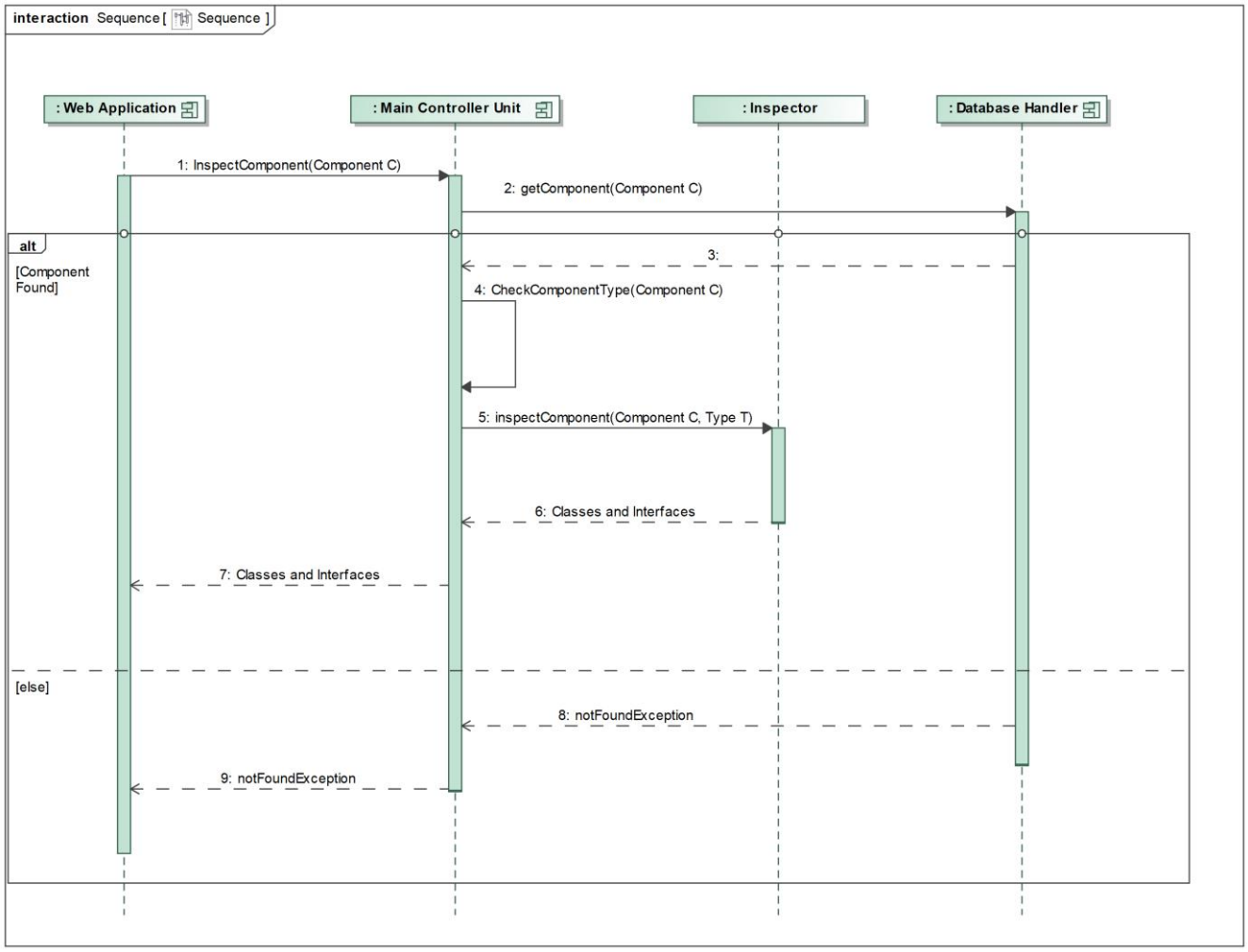

Figure 17 – Sequence diagram: inspect component scenario

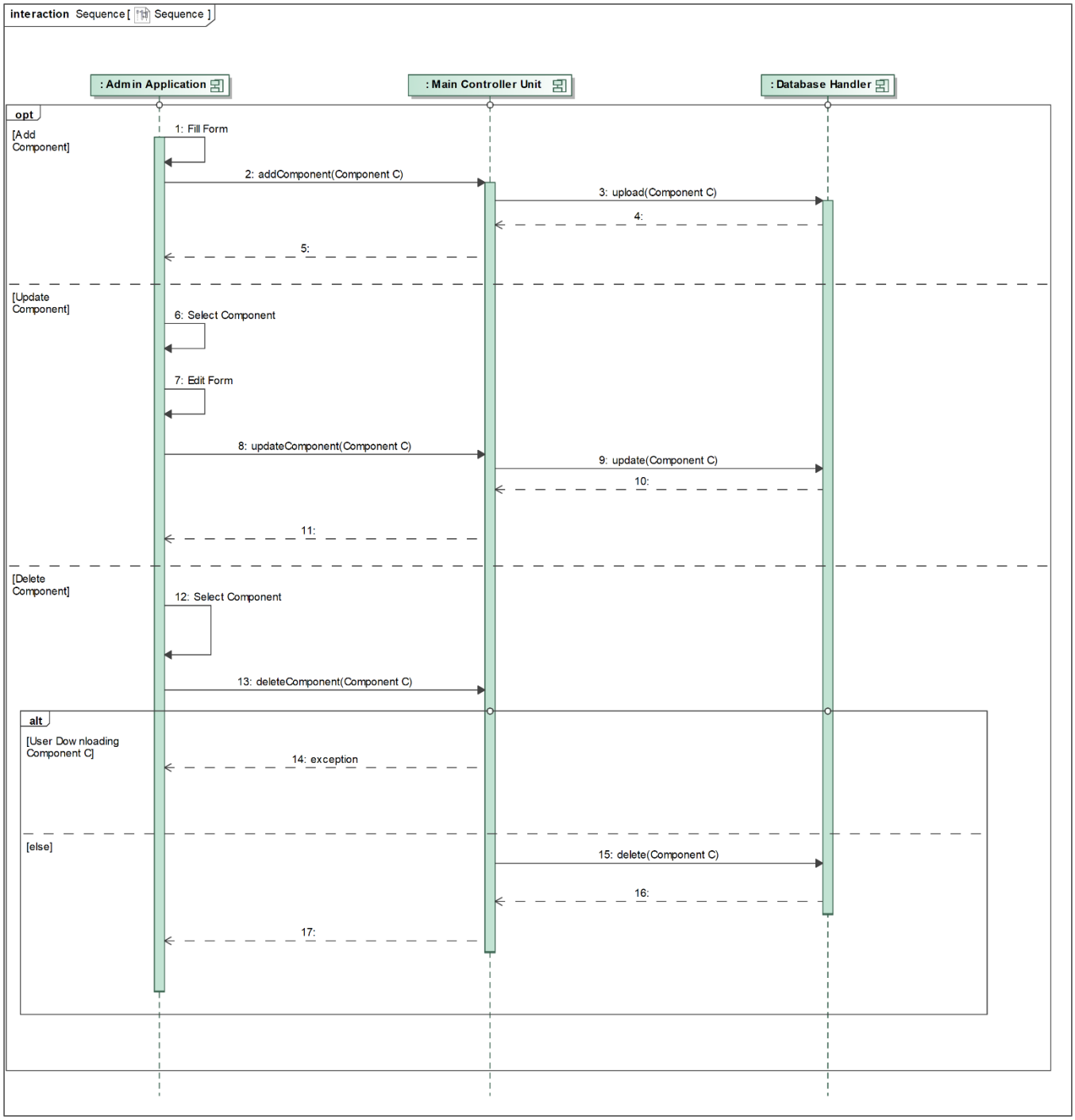

Figure 18 – Sequence diagram: admin operations

5.4 DatabaseHandler

Thanks to this component, it is possible to access the Database and to read/write data on it.

5.4.1 Static Structure

The only class representing the Static Structure is named “Component”. This class holds the information of the components. There are six attributes in this class, ID to identify the component, the name of the component, a

description of the component, the author of the component, the type of component (COM, Java component or .Net) and the path of the component on the Server.
The purpose of this component is to interact with the database. This includes both reading data from the database and updating the information. There are also functionalities to add a component to the database and deleting components.

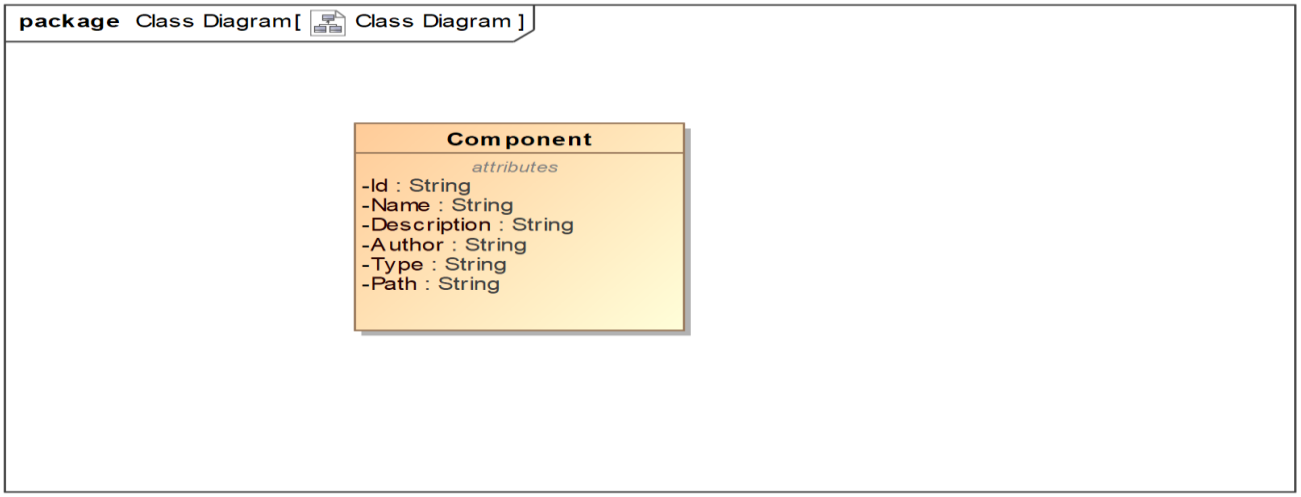

Figure 19 – Class diagram of the DatabaseHandler component

REVISION

Rev. Ind.	Page (P) Chapt.(C)	Description	Date Initials
001	2.1	Changed Use Case diagram (removed login and register for users and	20181610
002	4.2	administrators)	20181610
003	2.2	Updated component diagram for splitting the application into two (web-based application and windows application) Updating sections for individual use cases based on the new use case diagram	20181610