Project : Online Component Repository
Course CDT401, Component Technologies
Group 3

Steering group:
Frank Lüders
Gabriel Campeanu

Project group:
Valerio Lucantonio
Manvisha Kodali
Filip Markovic
Alexandre Le Borgne
Tania Matamoros Santamaría
Marc Terrasa Bonet

November 2, 2014

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

For the Component technologies course we need to implement an online components repository. The aim of this paper is to focus on the design of the project, technologies that will be used and what decisions has been made. In order to develop the application we have several stackeholders to consider:

- Frank Lüders and Gabriel Campeanu: as the customers of the projects.
- End User: the user will have access to the repository through a dedicated
- Admin: the administrator of the system. It will be able to add, remove and modify components in the repository. It will have access to the application through a dedicated desktop interface.

For this project, we are using Client-Server architecture. The application is built using ASP.Net on Visual Studio 2010 and Winforms. The application also uses Java and for accessing database, using Microsoft Access Fatabase. It consists of several parts: 1. Client web application. 2. Server application. 3. Desktop application. 4. Database.

1.2 Definitions

Terms	**Definitions**
Component Repository	A folder where all components reside

1.3 Related Document

Document Identity	**Document title**
ProjectDescription-revised.pdf	ProjectDescription

2 Functional Description

2.1 Use Case Model

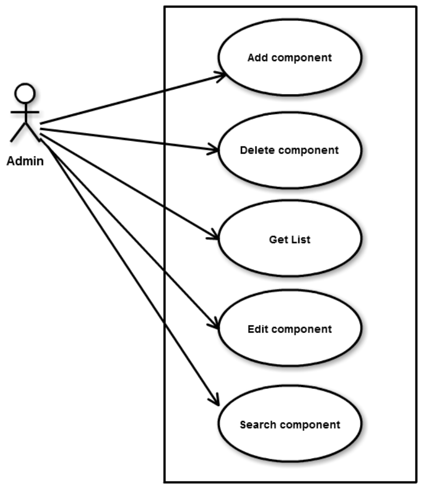


Figure 1: Use-Case for Admin

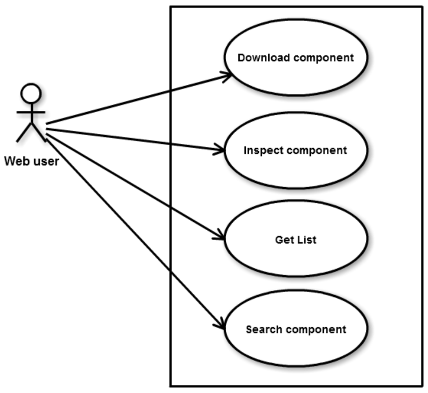


Figure 2: Use-Case for web user

2.1.1 Actors

Web user:	is the basic user of our application. the web user can browse, inspect and download components in the repository.
Admin:	basically he manages resources in the repository by adding, modifying or removing the components of the database.

2.1.2 Use Cases

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

a web user
Download component, described in section 2.2;
Inspect component, described in section 2.3;
Get list, described in section 2.4;
Search component, described in section 2.5;

b Admin
Get list, described in section 2.4;
Search component, described in section 2.5;
Add component, described in section 2.6;
Remove component, described in section 2.7;
Get all components in section 2.8;
Modify component in section 2.9;

2.2 Download Component

2.2.1 Participating Actors

Client

2.2.2 Precondition

You need to be in the interface of the Client.

2.2.3 Main Flow of Events

1. The Client searches component (usecase 2.5) or asks for a list of them.
2. The system gives the searched component or required list.
3. The Client selects and downloads the component into his computer.
4. The system retrieves the component data from the server.
5. The Client saves the component into his/her local computer.

2.2.4 Alternative

- The system is unable to retrieve the component data from the server.

  4 Client gets an error message.
  5 Resume step 1.

- The Client is not able to save the component into his/her local computer.

  5 Client gets an error message.
  6 Resume step 1.

2.3 INSPECT Component

2.3.1 Participating Actors

Client

2.3.2 Precondition

You need to be in the interface of the Client.

2.3.3 Main Flow of Events

1. The Client searches component(usecase 2.5) or get the list(usecase 2.4) and chooses the component in order to get its information.
2. The system retrieves the appropriate component data from the server.
3. The Client views a detailed information of the components for their public classes and interfaces.

2.3.4 Alternative

- The system fails to get the component's information.

  2 The Client receives an error message.

  3 Resume step 1.

2.4 GET LIST

2.4.1 Participating Actors

Client

2.4.2 Precondition

You need to be in the interface of the Client.

2.4.3 Main Flow of Events

1. The Client asks for the list of components from the database.
2. The system retrieves with the corresponding list.

2.4.4 Alternative

- The system doesn't retrieve information from the database

  2 An error message is shown to the interfaces.

  3 The Client reloads the page.

2.5 SEARCH Component

2.5.1 Participating Actors

Client Admin

2.5.2 Precondition

You need to be in the interface of Client and Admin .

2.5.3 Main Flow of Events

1. The Client and Admin types the name of the component that he wants to find the component.
2. The System retrieves the information from server.
3. The component is displayed on the Client and Admin's system.

2.5.4 Alternative

- The system doesn't retrieve information from the database

  3 An error message is shown to the interfaces.

  4 The Client and Admin reloads the page.

  5 Resume step 1.

2.6 ADD component

2.6.1 Participating Actors

Admin

2.6.2 Precondition

You need to be in the interface of the admin.

2.6.3 Main Flow of Events

1. The Admin access to the adding window, fills all mandatory documentation and indicates where is the component that has to be added.
2. The component is added to the database.
3. The Admin gets confirmation that it is added and returns to the home page.

2.6.4 Alternative

- All the mandatory data is not filled.

  2 The Admin gets a warning.

  3 Resume at step 1.

- The component can't be located.

  2 The Admin gets a warning.

  3 Resume at step 1.

- Tthere is a failure with the database in the attempt to adding.

  1. An error message is shown to the interfaces.
  2. The Admin reloads the page.

2.7 REMOVE component

2.7.1 Participating Actors

Admin

2.7.2 Precondition

You need to be in the interface of the admin.

2.7.3 Main Flow of Events

1. The Admin searches component (usecase 2.5) or gets the list (usecase 2.4).
2. The system retrieves the component or list.
3. The Admin chooses a component and deletes it.
4. The system asks for a confirmation.
5. The Admin confirms.
6. The component is deleted from the database.
7. The Admin is redirected to the main page.

2.7.4 Alternative

- The Admin doesn't confirm.

  5 The process is exited without any changes.

- The component can't be deleted from the database.

  6 The Admin gets an error window.

2.8 GET ALL COMPONENTS

2.8.1 Participating Actors

Admin

2.8.2 Precondition

You need to be in the interface of the Admin.

2.8.3 Main Flow of Events

1. The Admin asks the list of components from the database.
2. The system retrieves the complete list of components.

2.8.4 Alternative : Information or component not validated

- The system doesn't retrieve information from the database

  2 An error message is shown to the interfaces.
  3 The Admin reloads the page.

2.9 MODIFY Component

2.9.1 Participating Actors

Admin

2.9.2 Precondition

You need to be in the interface of the admin.

2.9.3 Main Flow of Events

1. Admin searches for the specific component which he wants to modify.
2. The system retrieves him the component.
3. Admin changes the file or the information related to the component.
4. Changes are committed in the database and a confirmation message is shown to the admin.

2.9.4 Alternative: Information or component not validated

- The system doesn't retrieve information from the database

  4 An error message is shown to the admin.

  5 Admin reloads the page and the component won't be modified.

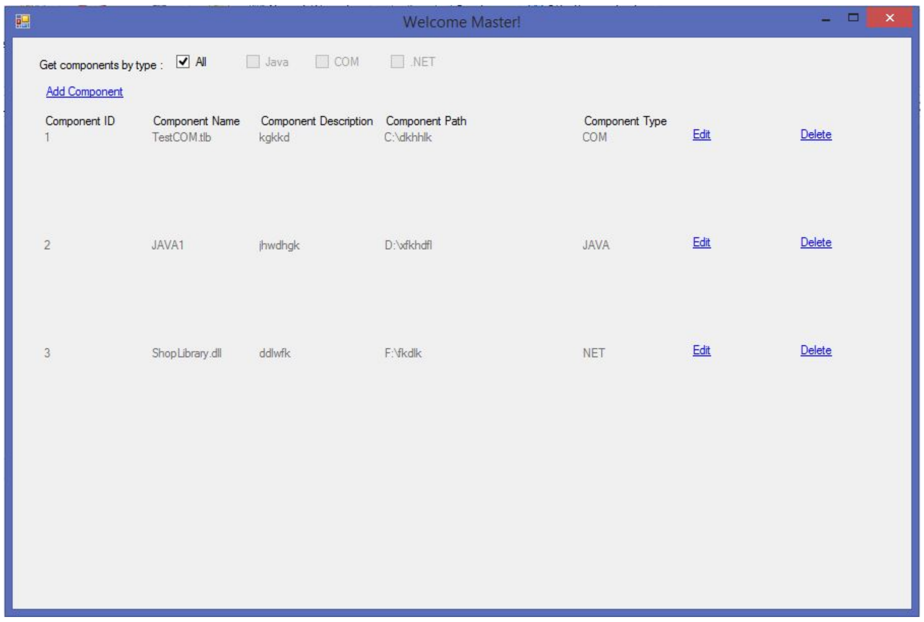
3 External Interfaces

Client's user Interface: The Client will use a web-browser to access the data on the database.ASP. Net will provide the web user with two views.

1. Component list view: Lists the type of components requested by the web user and are available in the component repository. This view also provides the web user with functionalities to:
   (a) Download a selected component.
   (b) Inspect a component.
   (c) Search for a component through a filter.
2. Component details view: Displays information about (public) classes, interfaces, methods in the component selected by the web user.

3.1 Graphical User Interface

3.1.1 Admin


Welcome Master!

Get components by type : [✓ All] Java COM .NET

Add Component

Component ID	Component Name	Component Description	Component Path	Component Type		
1	TestCOM.tlb	kgkkd	C:\dkhhlk	COM	Edit	Delete
2	JAVA1	jhwdhgk	D:\xfkhdfl	JAVA	Edit	Delete
3	ShopLibrary.dll	ddlwfk	F:\fkdlk	NET	Edit	Delete

Figure 3: Admin Interface

This picture corresponds with the admin interface. He has different options, as we can see:

- Add component: allows the admin to add a new component to the DB

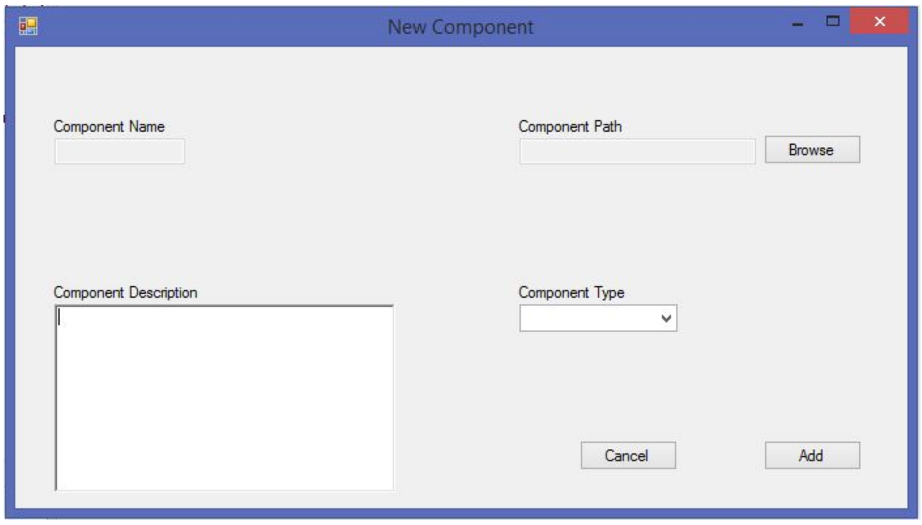
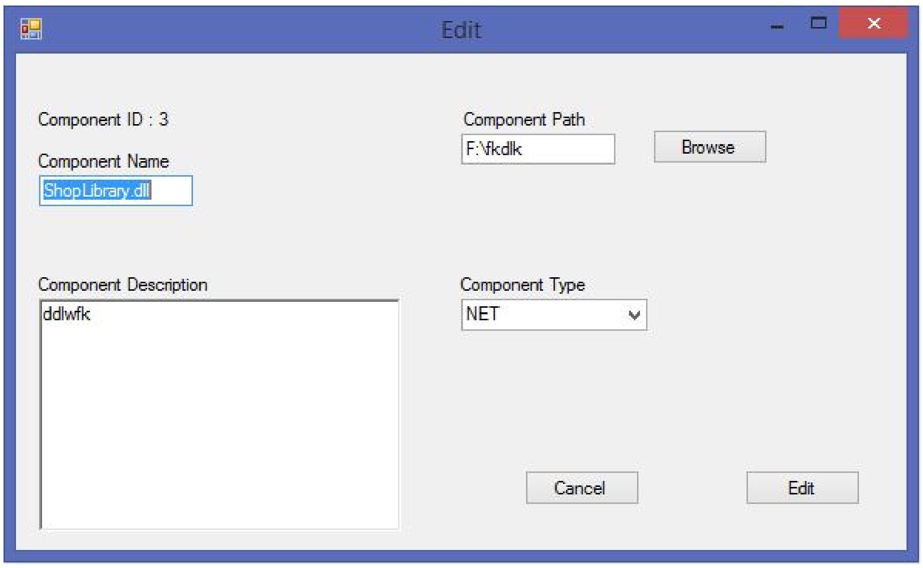


Figure 4: Add new component interface

- Edit Component: the admin can edit files or information about the component.



Component ID : 3

Component Name: ShopLibrary.dll

Component Description: ddlwfk

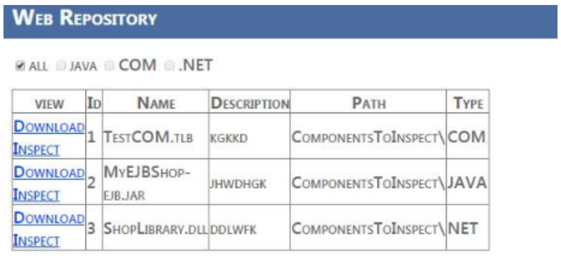
Component Path: F:\fkdlk

Component Type: NET

Figure 5: Edit component interface

- Delete component: the selected component is removed from the DB

3.2 Web user



Web Repository

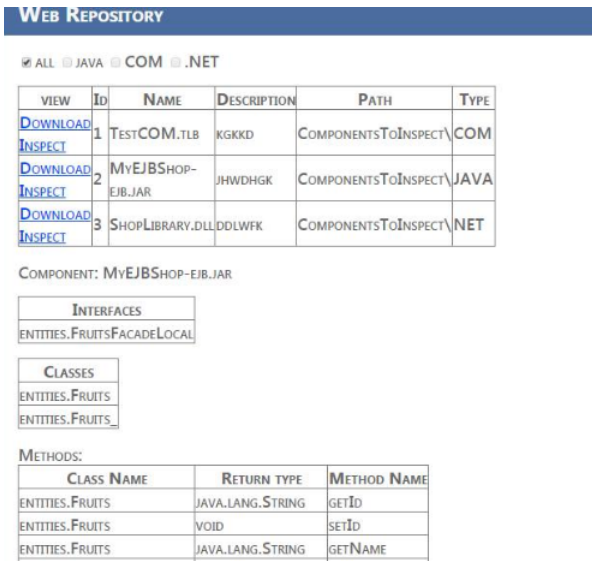
☑ All ☐ Java ☐ COM ☐ .NET

View	Id	Name	Description	Path	Type
Download Inspect	1	TestCOM.tlb	kgkkd	ComponentsToInspect\	COM
Download Inspect	2	MyEJBShop-ejb.jar	jhwdhgk	ComponentsToInspect\	JAVA
Download Inspect	3	ShopLibrary.dll	ddlwfk	ComponentsToInspect\	NET

Figure 6: Web user interface

The figure 6 corresponds with the web user interface. He has different options, as we can see:

- Download: download the component selected.
- Inspect: inspect one component selected.



Web Repository

☑ All ☐ Java ☐ COM ☐ .NET

View	Id	Name	Description	Path	Type
Download Inspect	1	TestCOM.tlb	kgkkd	ComponentsToInspect\	COM
Download Inspect	2	MyEJBShop-ejb.jar	jhwdhgk	ComponentsToInspect\	JAVA
Download Inspect	3	ShopLibrary.dll	ddlwfk	ComponentsToInspect\	NET

Component: MyEJBShop-ejb.jar

Interfaces
entities.FruitsFacadeLocal

Classes
entities.Fruits
entities.Fruits_

Methods:

Class Name	Return type	Method Name
entities.Fruits	java.lang.String	getId
entities.Fruits	void	setId
entities.Fruits	java.lang.String	getName

Figure 7: Inspect component interface

4 Software Architecture

4.1 Overview Rationale

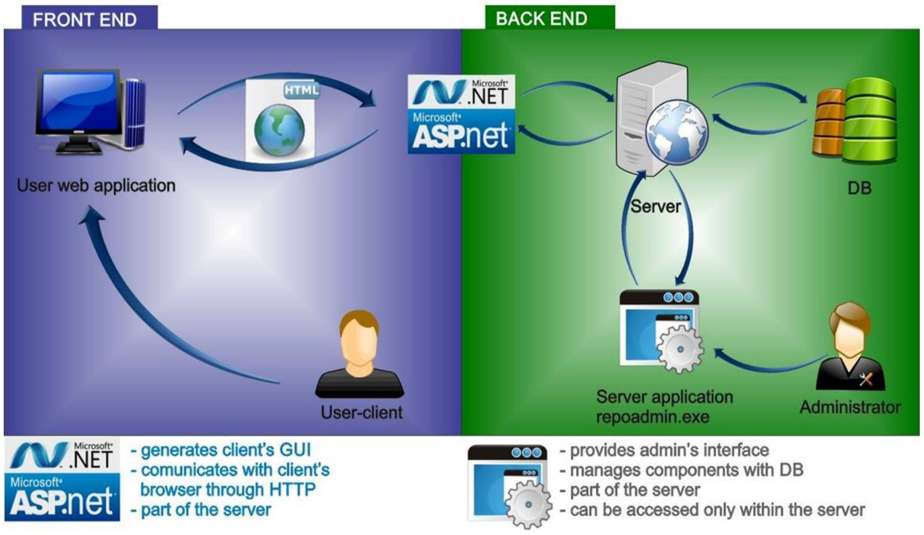


Figure 8: Software Architecture of project

The architecture of the project is divided in two main areas. First one is a client-server oriented component repository and the second is a repoadmin desktop application which is positioned on the main server with a component repository managing purpose. Together, they create an organized MVC (Model-View-Control) structure. In the front end of the application there is an user web application which represents the view. On the server side, there is an ASP.net and repoadmin.exe which represents the control part, and as for the model there is a component repository with a DB.

4.2 System Decomposition

- User web application: is the GUI that user "Client" uses to access our application.
- DB: In the database we can find all the components stored, with their specific descriptions and characteristics.
- Server application repoadmin: is the GUI that user "admin" uses to access our application. This GUI has different utilities than Client GUI.
- Server: is the connection between all the other components of our system (user applications and DB): it receives the request from both users, and send the suitable response, through interaction with the database.
- Java reflection: is the component that helps us to inspect Java components.

- NET reflection: is the component that helps us to inspect .NET components.
- COM reflection: is the component that helps us to inspect COM components.

4.3 Hardware/Software Mapping

Hardware requirement is achieved by only one server which will act as web server and database server. Component repository will be a part of its file system, as well as the database. Server application will be on that server as well. The only elements which will be on the web user's side are the generated HTML pages provided by ASP.net code, which is also a part of the server.

4.4 Persistent Data

Components will be stored on the server's file system, but as to provide valuable mechanisms, its representation in the database will have next persistent data:

- component_id - unique identifier in the database
- component_name - name of component provided by admin
- component_description - description of the component provided by admin
- component_file_path - location in the file system where the component is stored
- date_added - time when the component is added to the repository by the administrator
- downloads_counter - number of downloads of the component
- component_type - contains the information of the component type for the reflection purposes

4.5 Access Control

The only access control will be provided for the repoadmin.exe application which will manage the repository.

4.6 Start-Up and Shut-Down

Both web user and admin can access to the system whenever they want to. Also if they are executing an operation and it takes too long, or if they simply want to stop it, they can leave the app.

4.7 Error Handling

We will have error handlers in both sides of the communication: server and web user. If an error occurs in the web user side, he would received errors messages describing the problem, and recommendations like "try later" or any other specific behavior. If an error occurs in the server side, the system will check automatically if all the components are available.

5 Detailed Software Design

5.1 Component Diagram

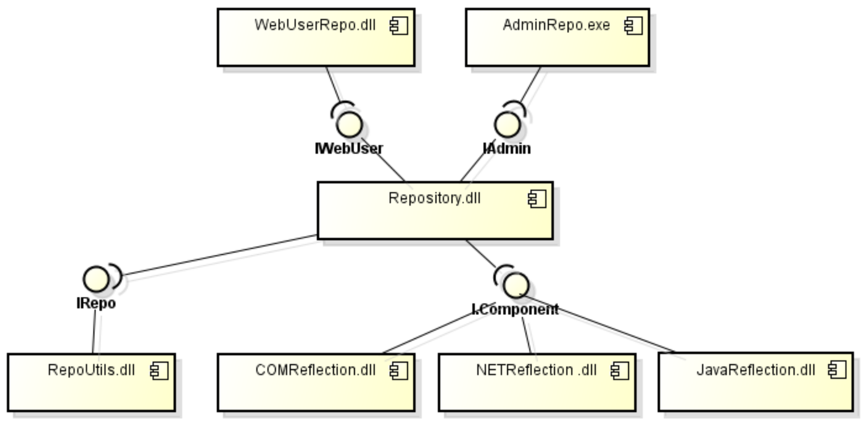


Figure 9: Component Diagram

AdminRepo (View) : this component will manage the admin graphical interface and will communicate with the repository through the offered interface. This component is an exe windows form application, and allows all the function described in the Admin use case diagram. WebUserRepo (View): this component represents one of the two views of the application in particular the Client application interface. It offers to the web users of the application all the use cases described in the WebUser usecase diagram. Repository (controller) : This component will manage all the server side operations and communications. It is the responsible of the system . It provides 2 interfaces both to the web user and the admin. It requires 2 interfaces: one from the RepoUtils component, and the other one from the 3 components providing reflections. Components (models) : NET.dll, COM.dll and JAVA.dll are the 3 component which will inspect their respective type of component. They will be called through their common interface. BackEnd_Libraries: it's the component containing the libraries that the controller uses to get access to the database and the components properties. It provides an interface for communicating with the Repository.

5.2 Static structure

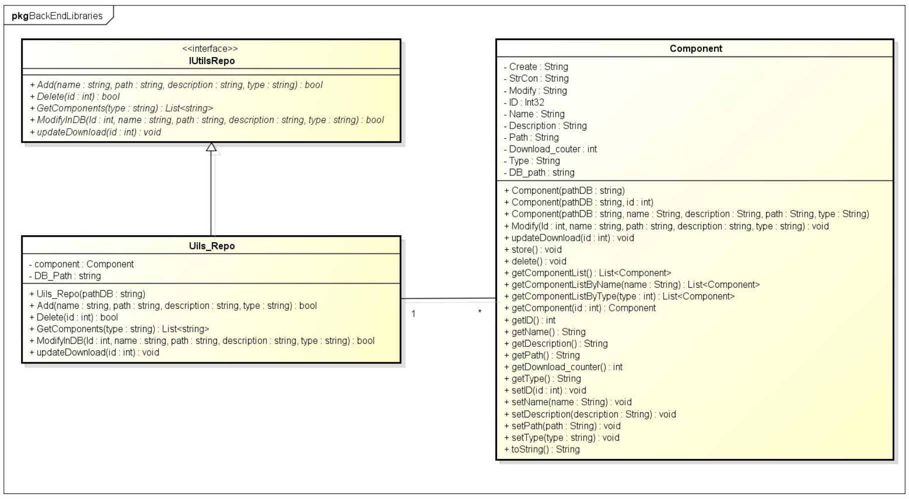

pkgBackEndLibraries
<<interface>>
IUtilsRepo
+ Add(name : string, path : string, description : string, type : string) : bool
+ Delete(id : int) : bool
+ GetComponents(type : string) : List<string>
+ ModifyInDB(Id : int, name : string, path : string, description : string, type : string) : bool
+ updateDownload(id : int) : void
Uils_Repo
- component : Component
- DB_Path : string
+ Uils_Repo(pathDB : string)
+ Add(name : string, path : string, description : string, type : string) : bool
+ Delete(id : int) : bool
+ GetComponents(type : string) : List<string>
+ ModifyInDB(Id : int, name : string, path : string, description : string, type : string) : bool
+ updateDownload(id : int) : void
1
*
Component
- Create : String
- StrCon : String
- Modify : String
- ID : Int32
- Name : String
- Description : String
- Path : String
- Download_couter : int
- Type : String
- DB_path : string
+ Component(pathDB : string)
+ Component(pathDB : string, id : int)
+ Component(pathDB : string, name : String, description : String, path : String, type : String)
+ Modify(Id : int, name : string, path : string, description : string, type : string) : void
+ updateDownload(id : int) : void
+ store() : void
+ delete() : void
+ getComponentList() : List<Component>
+ getComponentListByName(name : String) : List<Component>
+ getComponentListByType(type : int) : List<Component>
+ getComponent(id : int) : Component
+ getID() : int
+ getName() : String
+ getDescription() : String
+ getPath() : String
+ getDownload_counter() : int
+ getType() : String
+ setID(id : int) : void
+ setName(name : String) : void
+ setDescription(description : String) : void
+ setPath(path : String) : void
+ setType(type : string) : void
+ toString() : String

Figure 10: Class diagram : BackEndLibraries Component

The figure 10 represents the BackEndLibraries component. This component is accessed through its interface IUtilsRepo. The class Uils_Repo implements the methods of this interface. The class Component handle the connection with the database and represents a component. This component allows adding, deleting, searshing, and modifying functionalities on the database.

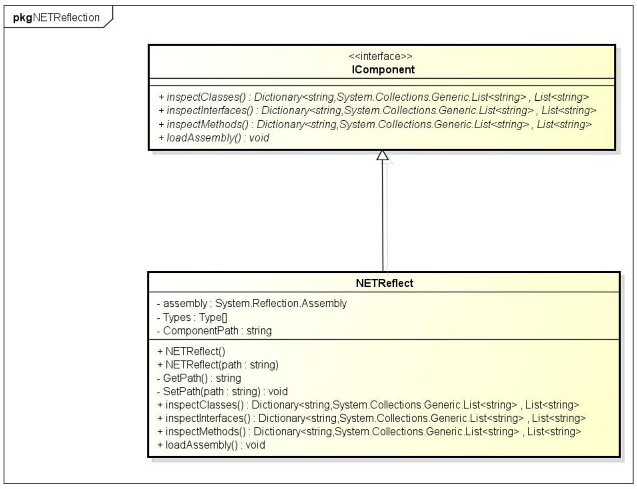

pkgNETReflection
<<interface>>
IComponent
+ inspectClasses() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectInterfaces() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectMethods() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ loadAssembly() : void
NETReflect
- assembly : System.Reflection.Assembly
- Types : Type[]
- ComponentPath : string
+ NETReflect()
+ NETReflect(path : string)
- GetPath() : string
- SetPath(path : string) : void
+ inspectClasses() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectInterfaces() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectMethods() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ loadAssembly() : void

Figure 11: Class diagram : NETReflection Component

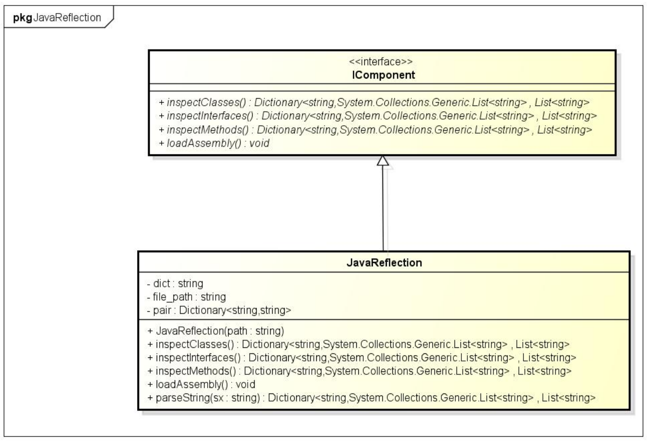

pkgJavaReflection
<<interface>>
IComponent
+ inspectClasses() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectInterfaces() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectMethods() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ loadAssembly() : void
JavaReflection
- dict : string
- file_path : string
- pair : Dictionary<string,string>
+ JavaReflection(path : string)
+ inspectClasses() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectInterfaces() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectMethods() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ loadAssembly() : void
+ parseString(sx : string) : Dictionary<string,System.Collections.Generic.List<string> , List<string>

Figure 12: Class diagram : JavaReflection Component

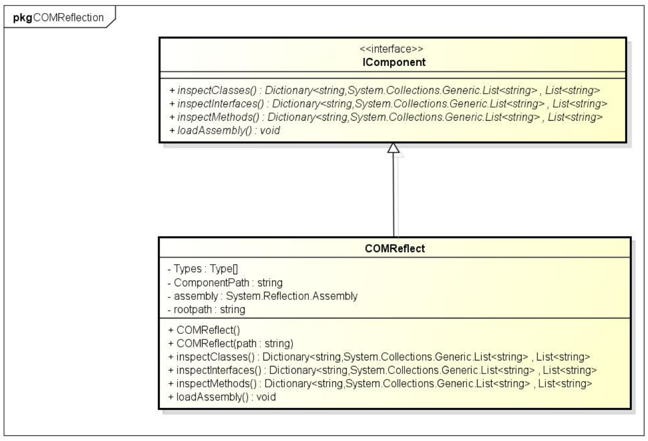

pkgCOMReflection
<<interface>>
IComponent
+ inspectClasses() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectInterfaces() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectMethods() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ loadAssembly() : void
COMReflect
- Types : Type[]
- ComponentPath : string
- assembly : System.Reflection.Assembly
- rootpath : string
+ COMReflect()
+ COMReflect(path : string)
+ inspectClasses() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectInterfaces() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectMethods() : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ loadAssembly() : void

Figure 13: Class diagram : COMReflection Component

The figures 11, 12 and 13 represent the class diagrams of the NET, Java and COM reflection components. They share the same interface (IComponent) which provides 4 methods : InspectClasses to get the Class names from the component, InspectInterfaces to get the interfaces names from the component, InspectMethods to get the method names from the component and load assembly to load the new component into the reflection component. By sharing the same interface, we can use polymorphism to call the reflection, whatever the type of the component.

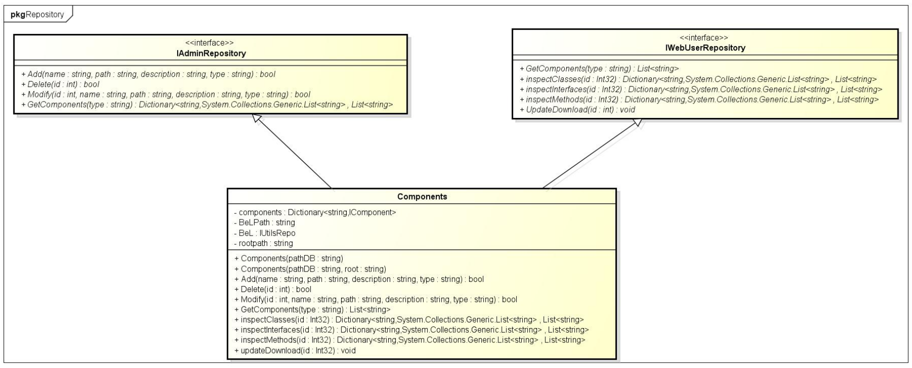

pkgRepository
<<interface>>
IAdminRepository
+ Add(name : string, path : string, description : string, type : string) : bool
+ Delete(id : int) : bool
+ Modify(id : int, name : string, path : string, description : string, type : string) : bool
+ GetComponents(type : string) : Dictionary<string,System.Collections.Generic.List<string> , List<string>
<<interface>>
IWebUserRepository
+ GetComponents(type : string) : List<string>
+ inspectClasses(id : Int32) : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectInterfaces(id : Int32) : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectMethods(id : Int32) : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ UpdateDownload(id : int) : void
Components
- components : Dictionary<string,IComponent>
- BeLPath : string
- BeL : IUtilsRepo
- rootpath : string
+ Components(pathDB : string)
+ Components(pathDB : string, root : string)
+ Add(name : string, path : string, description : string, type : string) : bool
+ Delete(id : int) : bool
+ Modify(id : int, name : string, path : string, description : string, type : string) : bool
+ GetComponents(type : string) : List<string>
+ inspectClasses(id : Int32) : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectInterfaces(id : Int32) : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ inspectMethods(id : Int32) : Dictionary<string,System.Collections.Generic.List<string> , List<string>
+ updateDownload(id : Int32) : void

Figure 14: Class diagram : Repository Component

The figure 14 shows the class diagram of the Repository component. The class Components implements two different interfaces which are IAdminRepository and IWebUser to be able to grant different rights on the component repository to the admin and web user.So Components class implements add, delete modify and GetComponents from the IAdminInterface. Obviously it will allow only the admin app to add, delete, modify and get component by type. All the web user functionalities are implemented from the IWebUserInterface to get and inspect components and to update the download counter when the user downloads a component. The Components class has a member named components which is type of Dictionary<string, IComponent> to reference components from the list, the string key is the id of the component and the value is the IComponent Interface(from the reflection components) this dictionary will allow us to inspect easily components using polymorphism. The Components class contains an instance of IUtilsRepo to communicate with the database.

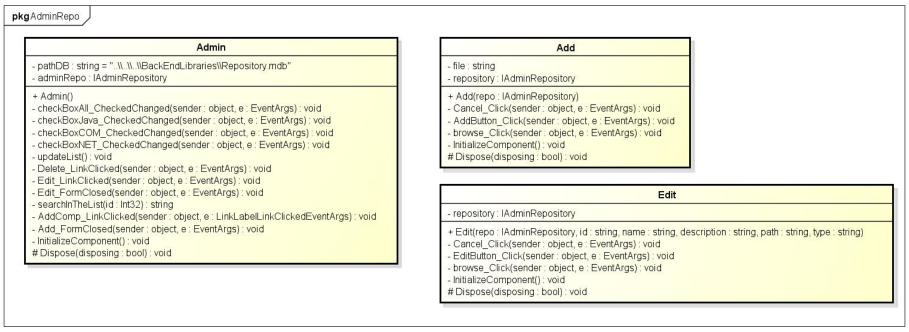

pkgAdminRepo
Admin
- pathDB : string = "..\\..\\..\\BackEndLibraries\\Repository.mdb"
- adminRepo : IAdminRepository
+ Admin()
- checkBoxAll_CheckedChanged(sender : object, e : EventArgs) : void
- checkBoxJava_CheckedChanged(sender : object, e : EventArgs) : void
- checkBoxCOM_CheckedChanged(sender : object, e : EventArgs) : void
- checkBoxNET_CheckedChanged(sender : object, e : EventArgs) : void
- updateList() : void
- Delete_LinkClicked(sender : object, e : EventArgs) : void
- Edit_LinkClicked(sender : object, e : EventArgs) : void
- Edit_FormClosed(sender : object, e : EventArgs) : void
- searchInTheList(id : Int32) : string
- AddComp_LinkClicked(sender : object, e : LinkLabelLinkClickedEventArgs) : void
- Add_FormClosed(sender : object, e : EventArgs) : void
- InitializeComponent() : void
# Dispose(disposing : bool) : void
Add
- file : string
- repository : IAdminRepository
+ Add(repo : IAdminRepository)
- Cancel_Click(sender : object, e : EventArgs) : void
- AddButton_Click(sender : object, e : EventArgs) : void
- browse_Click(sender : object, e : EventArgs) : void
- InitializeComponent() : void
# Dispose(disposing : bool) : void
Edit
- repository : IAdminRepository
+ Edit(repo : IAdminRepository, id : string, name : string, description : string, path : string, type : string)
- Cancel_Click(sender : object, e : EventArgs) : void
- EditButton_Click(sender : object, e : EventArgs) : void
- browse_Click(sender : object, e : EventArgs) : void
- InitializeComponent() : void
# Dispose(disposing : bool) : void

Figure 15: Class diagram : Admin application

The admin application (figure 15) contains 3 classes which are Admin, Add and Edit. Admin will perform the display of the list, the deletion of components, and the calls to Add and Edit which are designed to add and/or edit a component. All these classes communicate with the IAdminRepository interface of the repository component. (all of these classes contains an instance of IAdminRepository)

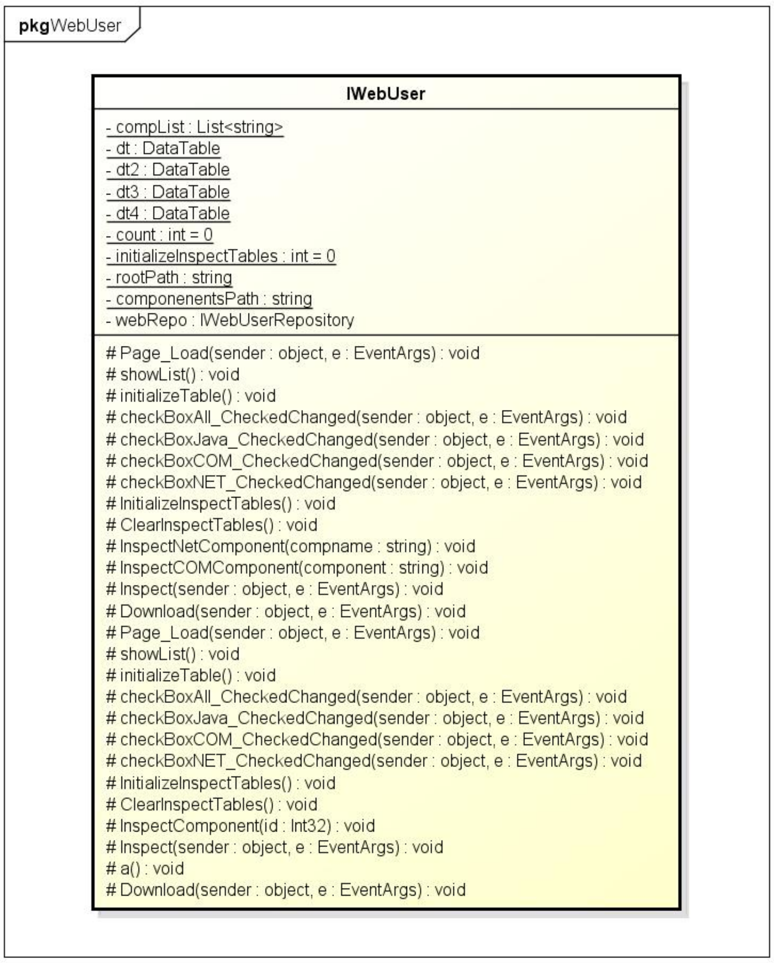

pkgWebUser
IWebUser
- compList : List<string>
- dt : DataTable
- dt2 : DataTable
- dt3 : DataTable
- dt4 : DataTable
- count : int = 0
- initializeInspectTables : int = 0
- rootPath : string
- componenentsPath : string
- webRepo : IWebUserRepository
# Page_Load(sender : object, e : EventArgs) : void
# showList() : void
# initializeTable() : void
# checkBoxAll_CheckedChanged(sender : object, e : EventArgs) : void
# checkBoxJava_CheckedChanged(sender : object, e : EventArgs) : void
# checkBoxCOM_CheckedChanged(sender : object, e : EventArgs) : void
# checkBoxNET_CheckedChanged(sender : object, e : EventArgs) : void
# InitializeInspectTables() : void
# ClearInspectTables() : void
# InspectNetComponent(compname : string) : void
# InspectCOMComponent(component : string) : void
# Inspect(sender : object, e : EventArgs) : void
# Download(sender : object, e : EventArgs) : void
# Page_Load(sender : object, e : EventArgs) : void
# showList() : void
# initializeTable() : void
# checkBoxAll_CheckedChanged(sender : object, e : EventArgs) : void
# checkBoxJava_CheckedChanged(sender : object, e : EventArgs) : void
# checkBoxCOM_CheckedChanged(sender : object, e : EventArgs) : void
# checkBoxNET_CheckedChanged(sender : object, e : EventArgs) : void
# InitializeInspectTables() : void
# ClearInspectTables() : void
# InspectComponent(id : Int32) : void
# Inspect(sender : object, e : EventArgs) : void
# a() : void
# Download(sender : object, e : EventArgs) : void

Figure 16: Class diagram : Web user application

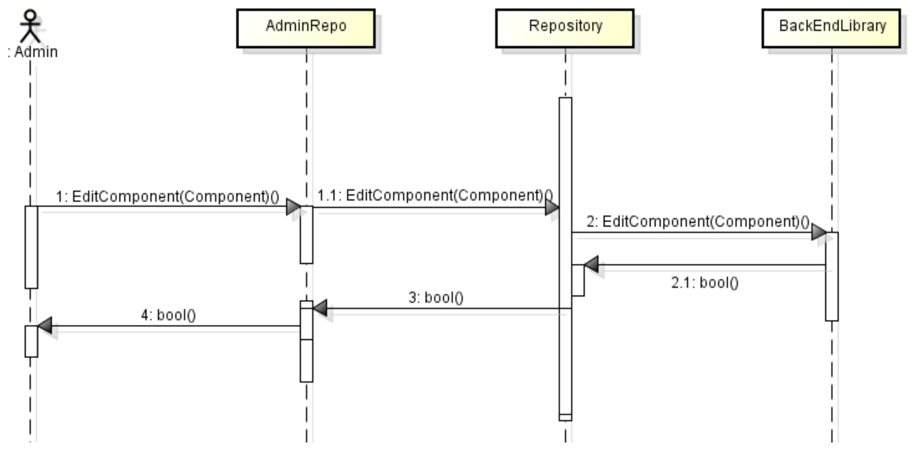
In the figure 16 we can see that the Repository component is instanciated via its IWebRepository interface. The web user aplication manages the browsing, downloading and inspecting in the same class. InspectNETComponent and InspectCOMComponent are deprecated since we use polymorphism and so only one method : InspectComponent.

5.3 Dynamic behaviour

For Admin : The actions will be changed dynamically depending upon the current state we are in. All the initiated events from the Windows Forms will invoke the logical components by using interfaces. This logical component wil communicate with database.

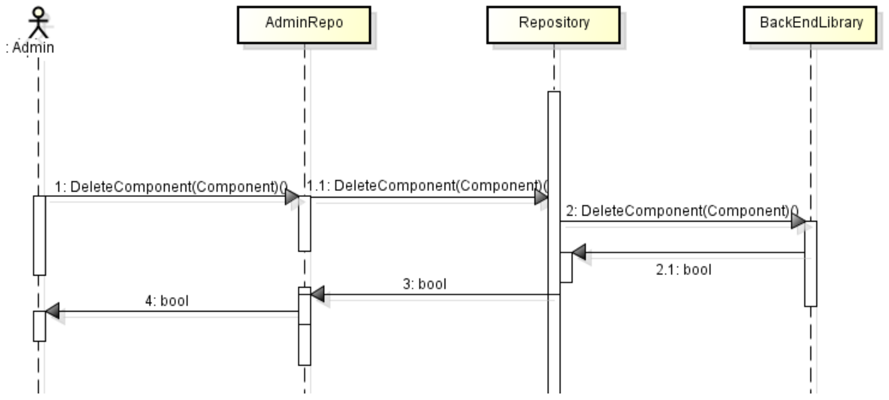
For Cient : The actions will be changed dynamically depending upon the current state we are in. All the initiated events from the ASP.NET will invoke the logical components by using interfaces. This logical component wil communicate with database.

5.3.1 Admin



: Admin
AdminRepo
Repository
BackEndLibrary
1: EditComponent(Component)()
1.1: EditComponent(Component)()
2: EditComponent(Component)()
2.1: bool()
3: bool()
4: bool()

Figure 17: Sequence Diagram:Edit Component



: Admin
AdminRepo
Repository
BackEndLibrary
1: DeleteComponent(Component)()
1.1: DeleteComponent(Component)()
2: DeleteComponent(Component)()
2.1: bool
3: bool
4: bool

Figure 18: Sequence Diagram:Delete Component

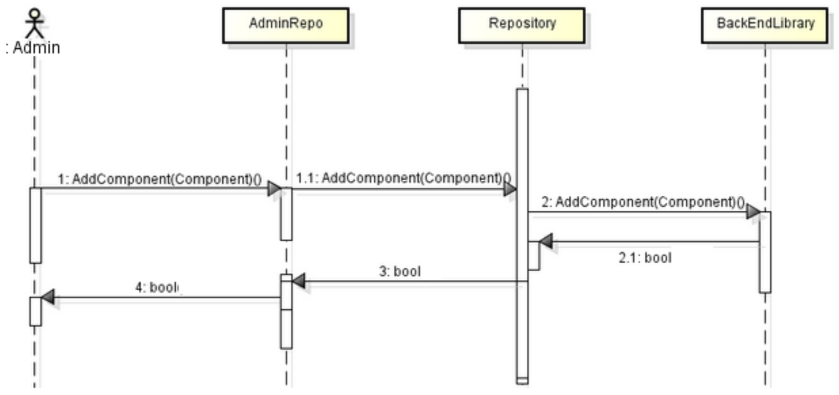

: Admin
AdminRepo
Repository
BackEndLibrary
1: AddComponent(Component)()
1.1: AddComponent(Component)()
2: AddComponent(Component)()
2.1: bool
3: bool
4: bool

Figure 19: Sequence Diagram: Add Component

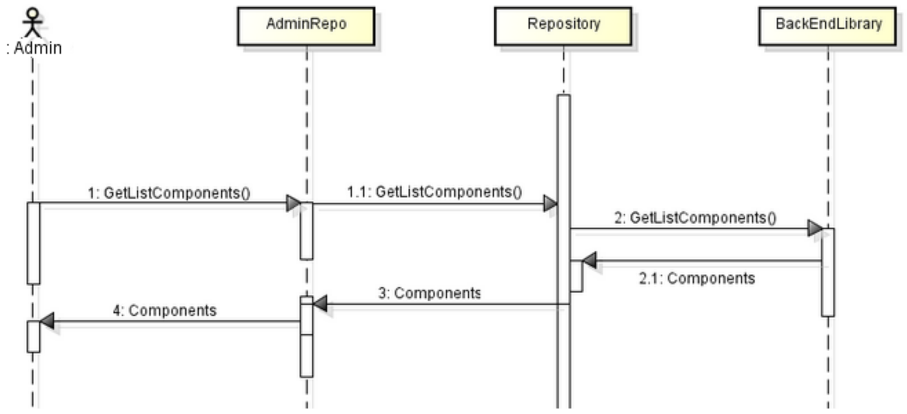


Figure 20: Sequence Diagram: Get List Components

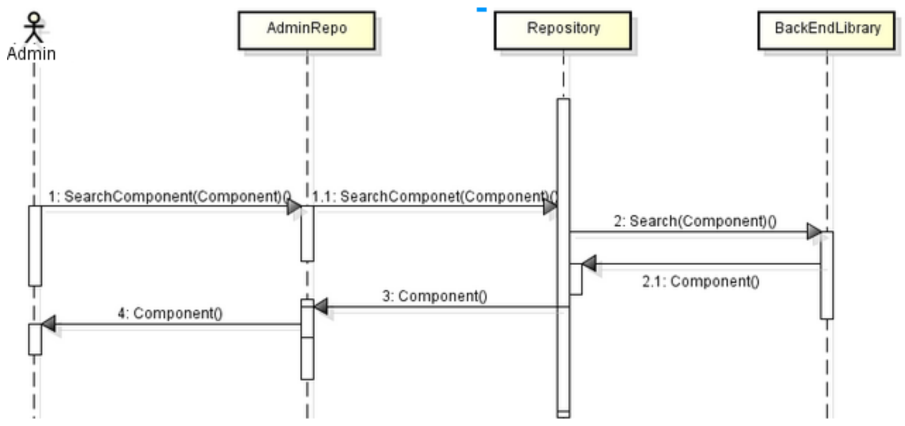


Figure 21: Admin Sequence Diagram: Search Component

5.3.2 Web user

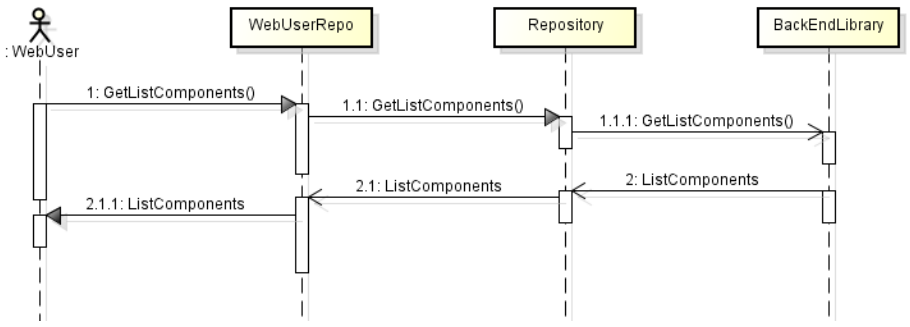


Figure 22: Web User Sequence Diagram: Get Components List

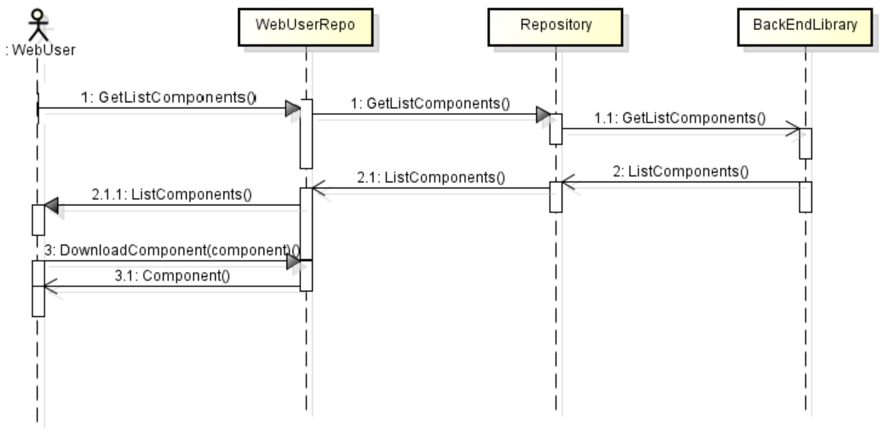


Figure 23: WebUser Sequence Diagram: DownloadComponent

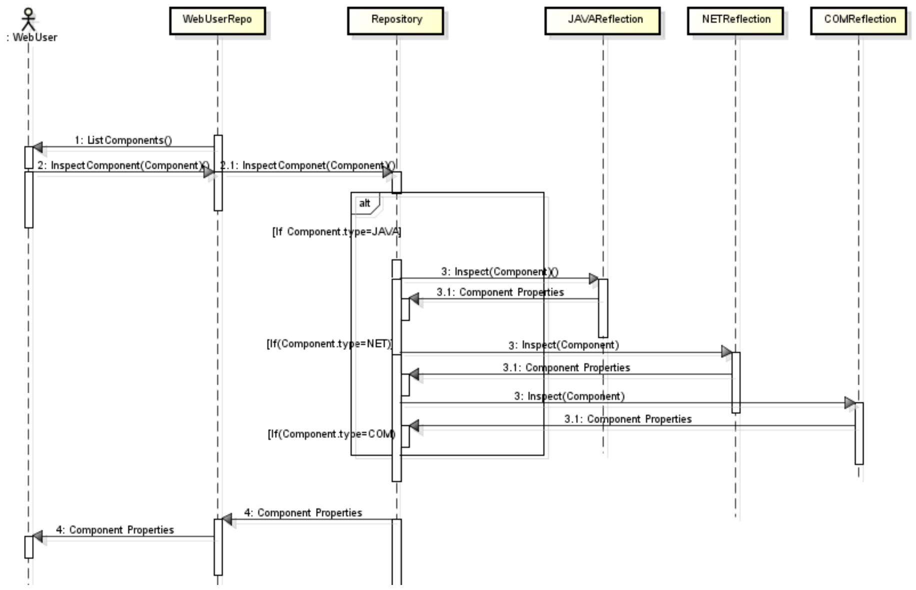


Figure 24: WebUser Sequence Diagram; Inspect Components

Table 1: REVISIONS

Rev. Ind.	Page (P) Chapt.(C)	Description	Date Initials
001	P5,6,7,8,9. C2	Filled functional description section, with all the possible use cases and the related actors and links.	2014-10-05
002	P4 C1	Filled first section, writing about the needed background, definitions and related documents.	2014-10-05
003	P12	Decomposition of the system into subsystems and described in the document	2014-10-05
004	P13 C4	Made decision about persistent data and written in the document	2014-10-05
005	P12,13 C4	Finished 4 section about software architecture	2014-10-06
006	P16,17. C5	First sequence diagrams done and added to the document	2014-10-06
007	P16 C5	Class diagram of the entire system added to the document	2014-10-06
008	P17 C5	Created last sequence diagrams and added to the document	2014-10-07
009	P12 C4	Updated overview of the system	2014-10-07
010	P14 C5	Added information to section 5	2014-10-07
011	P5 C2	Modify use cases model: added new use case	2014-10-15
012	P5 C2	Added description of the new use case	2014-10-15
013	P9,10 C2	Updated use cases model: one diagram for every actor	2014-10-16
014	P13 C4	Updated persistent data: added one new variable	2014-10-16
015	P16 C5	Updated class diagram	2014-10-16
016	P6,7,8,9,10,11 C2	Updated use cases description	2014-10-29
017	P11 C3	Added graphic interface figure	2014-10-31
018	P17, 18, 19, 20 C5	Updated class diagram	2014-10-31
019	P20, 21, 22, 23, 24 C5	Updated dynamic behavior and static structure	2014-10-31
020	P16 C5	Added Component Diagram	2014-10-31
021		Removed unnecessary chapters	2014-10-31