VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Isa Asadi Vakil Kandi

Augmented Reality Client-Server Application for Schneider Electrics' Protection Relay

Information Technology
2016

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Program in Information Technology

ABSTRACT

Author	Isa Asadi Vakil Kandi
Title	Augmented Reality Client-Server Application for Vamp Protection Relay
Year	2016
Language	English
Pages	54
Name of Supervisor	Ghodrat Moghadampour

In marker based Augmented Reality (AR) applications any known target is recognized within the received data from the camera and the application adds the predefined computer made data (which can be text, 3D object, video, image, animation, etc.) into the live video according to the position and orientation of the target and displays on the head-mounted screens or mobile apps.

The aim of this thesis was to develop a Client-Server application for Schneider Electrics' protection relay's AR application, which makes it possible to maintain the augmented data through a web application and save the data into a remote database. The AR application can then always receive updated data from the web application for the device which has a unique serial number and is being videoed live by an AR camera. The web application also provides the possibility for the client application to update the configuration of a relay. The significant improvements of the current application over the earlier implementation is that data is not saved into runtime variables but permanently in the database and the size of the application is much smaller.

The web application was implemented by using Java (Servlets and JSP pages), HTML, CSS and jQuery, but the client application was implemented by using C# in Unity 3D engine. The client and the server applications exchange data in JSON format using HTTPS protocol. The data is stored in MySQL database, but the client application also has a local SQLite database for offline use.

The whole application consists of many modules. In this project work the web application, communication between the client and the web application, JSON data handling and synchronization of the MySQL and SQLite databases were implemented.

Key words Schneider Electrics' Relay configuration, Java web application jQuery, Augmented Reality, JSON, Unity 3D, MySQL, SQLite

ACKNOWLEDGEMENTS

I would like to take the opportunity to thank my thesis supervisor, Dr. Ghodrat Moghadampour, Principle lecturer in Vaasan Ammattikorkeakoulu, University of Applied Sciences (VAMK), for his support during this project. Thank you very much for your support and encouragement that you have provided during last for years.

I would also like to thank my colleague and project manager at 3D Studio Blomberg LTD. Company, Mr. Tony Nystrand for his valuable guidance and support during this project.

I also appreciate all of my teachers at VAMK for everything that I learned from them.

CONTENTS

ABSTRACT

Table of Contents

LIST OF FIGURES AND TABLES

LIST OF CODE SNIPPETS

LIST OF ABBREVIATIONS

AR	Augmented Reality
VR	Virtual Reality
ODG	Osterhout Design Group
WebGL	Web Graphics Library
AREA™	Augmented Reality for Enterprise Alliance
Ltd	Limited company
JSP	Java Server Page
JSON	Java Script Object Notation
XML	Extensible Markup Language.
PDF	Portable Document Format
PNG	Portable Network Graphics

1 INTRODUCTION

“Schneider Electric develops connected technologies and solutions to manage energy and process in ways that are safe, reliable, efficient and sustainable.” /1/.

VAMP is a part of Schneider Electrics now. Schneider Electrics’ Vamp protection relays have arc sensors which detects the fault in the power system and cuts off the current when a fault occurs. As the consequence, the damage caused by the arc is minimized /2/.

1.1 Backgrounds of the topic

The previous AR (Augmented Reality) application for Schneider Electrics’ protection relay was displaying the static data which were kept in the variables and the content was the same for all the devices of a specific model. In addition, there was another application for configuring the relays and all the configuration rules were saved in an XML file. The application was fetching the information from the XML file and helping/forcing the user to make an allowed configuration according to the rules, then it was assembling the 3D model of the parts together and displaying the 3D model of the configured relay.

1.2 Motivations

As it mentioned in section 1.1, there were two separate applications for Schneider Electrics’ protection relays. The content displayed in the AR interface was kept in the variables inside of the code and the 3D models of the parts were kept locally. Furthermore, the application had to be modified in order to make updates in the content. In addition, the Configurator application was not that flexible in adding a new version of the relay into the application and its size was big because of keeping all of the information and 3D models in the application locally. The aim of solving these problems was a good motivation for the developers.

1.3 Objectives

The objectives of this thesis are to develop a web application for being able to update the information of a relay with a unique serial number in the database by

authorized users, to combine the existing AR and Configurator applications, to display data and assemble 3D models dynamically depending on the information saved in the database, to keep the history of a device configurations during its life cycle, to reduce the size of the applications which are installed on the iPad by keeping the 3D models on a server and import them during runtime, to make it possible that the client application can work offline by saving temporarily the necessary data in the SQLite database and 3D models in a local directory.

Figure 1 shows the structure of the application from the initial idea.

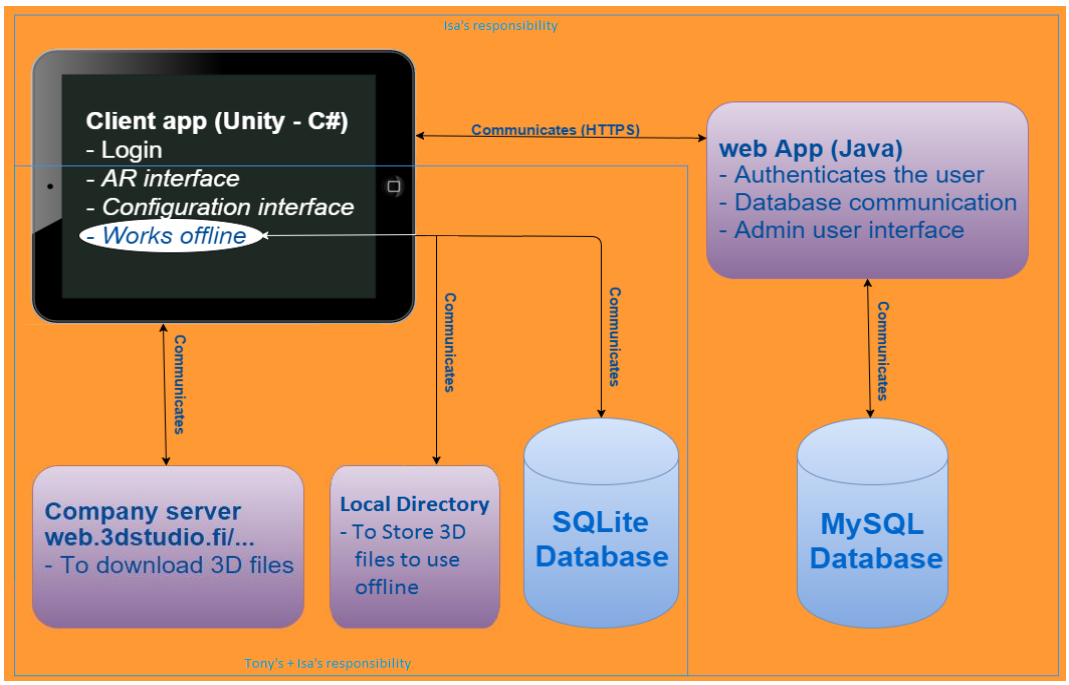

Figure 1- The structure of the application from the initial idea

1.4 Client Organization (3D Studio Blomberg Ltd)

This application was a project developed by 3D Studio Blomberg Ltd for Schneider Electrics. 3D Studio Blomberg is a Finnish company started in 2001 and located mainly in Vaasa. It also has three branches in Kokkola, Korsnäs and Jakobstad towns in Finland. The main focus of the company is to provide innovative visual tools and solution for the industries to promote their business, but they also develop software programs for customers. They use some of the latest technologies like AR,

VR (Virtual Reality), ODG (Osterhout Design Group) glasses and WebGL (Web Graphics Library) in their visual solutions.

The company is working as a supplier for some of the well-known companies like Wärtsilä, ABB, Schneider Electrics and KWH Mirka Ltd. It is also a development partner company with ODG Company and a member of AREA™ (Augmented Reality for Enterprise Alliance). Currently the company has eight employees with different skills.

The vision of 3D Studio Blomberg Ltd is “One Global Visual Language” and the company is trying to achieve with visual tools and solution. ‘GlobAR’ and ‘Thermo Polishing’ are examples of the company’s products.

2 RELEVANT TECHNOLOGIES

In this part the main technologies used in this project are introduced. The main technologies were Augmented Reality, Unity 3D Engine, JSON (Java Script Object Notation), jQuery and SQLite database.

2.1 Augmented Reality

Augmented Reality is an integration of computer made objects and live video that AR camera gets from the environment around the users in real time /3/. There are some AR technology providers that AR application developers can use their SDK or plugins for development purpose. Vuforia™ and Wikitude are two of the most known AR technology providers.

In order to see the AR content, the users must install the application on their smart phones or use any device which supports the AR technology. The application gets the real time video from the AR camera and processes it, then computer made objects will be integrated into the live video if the application recognizes any known target. A target is an image or a 3D object which has its own specific features and these features are known by the application.

The client application of this thesis has an AR interface that uses Vuforia's Unity plugin /4/.

Figure 2 - AR application

2.2 Unity 3D engine

Unity 3D engine is a cross-platform game engine which gives possibility to develop 2D or 3D games using C# or Java Script languages and it has powerful tools to make a user-friendly view for the game. The developer can make the application once and build it for different platforms. For example, you make a game for smart phones and test it inside of the Unity 3D editor, then build it for different platforms like Android, iOS, PC /5/.

It has a very user-friendly environment for developers that they can control the physic, position, rotation, animation and scale of an object in 3D world, write programs to give the desired behavior for it, add animation and many nice features for game applications. It is also possible to import 3D models exported from other 3D programs like 3DS Max /5/.

AR provider companies have plugins for the Unity 3D engine, so the developers can make the visual environment that they would like to integrate to the reality. One of the important reasons for developing an AR application with the Unity 3D engine is that developers have high ability of making visual content with desired

animations, physics, position and scale, then augment the live video received from the AR camera to be displayed on the user's screen and that is a good feature.

Both of the interfaces of the client application are implemented in Unity 3D engine, because of the ease of making/importing 3D objects and the possibility of integrating them with the live video from camera in real time.

2.3 JSON

JSON is a standard text format based on a non-strict subset of the JavaScript scripting language for exchanging information, very easy to understand and generate by human. It is completely independent from any programming language and many programming languages like Java, C#, Java Script, PHP, Python can decode it. The structure of a JSON content is a collection of keys and values where a value also can be a pair of keys and values or an array of the same keys for many times /8/.

In this application, the data are transmitted between the server side web application and the client application in JSON format using HTTPS protocol.

```
{
   ″first name″ : ″Isa″,
   ″last name″ : ″Asadi″,
   ″Phone″ : ″123456789″,
   ″email″ : ″isa@isa.com″,
   ″address″ : {
      ″country″ : ″Finland″,
      ″city″ : ″Vaasa″,
      ″postal code″ : ″65200″,
      ″Street″ : ″Wolffintie 30″
   }
}
```

Code snippet 1 – One sample of JSON content

2.4 jQuery

jQuery is one of the most popular cross-platform Java Script library that can be downloaded freely. It is developed by the jQuery team in 2006 to ease writing codes

in Java Script. It is being used for event handling, generating some animations or do many other interesting things in the HTML content in the browser /9/.

```
$(document).ready(function(){
  $('#searchCategoryCat').change('input',on_searchCategoryCat_change);
});

function on_searchCategoryCat_change (){
  var searchCategoryCat = $('#searchCategoryCat').val();
  if (searchCategoryCat.length > 0) {
    $('#getCategoryInfoButtonEdit').removeAttr('disabled');
    $('#getCategoryInfoButtonEdit').css('background-color', '#009530');
  } else {
    $('#getCategoryInfoButtonEdit').attr('disabled', 'disabled');
    $('#getCategoryInfoButtonEdit').css('background-color', '#adadad');
  }
}
```

Code snippet 2 – One sample of jQuery event handling function

2.5 SQLite database

SQLite is a serverless relational database engine written in C language that can be embedded to applications. The syntax of queries for SQLite and MySQL databases are the same and data are saved in tables. A SQLite database is a small cross-platform disk file with size of less than 200 KB. Many programming languages like Java, C#, Python, C++ have support to use SQLite /10/.

3 APPLICATION DESCRIPTION

Augmented Reality Client-Server application for Schneider Electrics' protection relay is a combination of two separate applications with additional features. The additional features are mentioned in section 1.3.

One of the applications is an AR application which shows only predefined information about a version of the relay. For instance, all of the devices for version V300 looks the same, the AR application was recognizing the version of the device from the style and features of its body, then displaying some general information about version V300 as augmented data as shown in figure 2.

The second application is an interactive application which is designed only for three versions of the Schneider electrics' VAMP relays: V300 (Feeder and Motor modes), V321 and V57. The rules of the possible configurations are saved in an XML file and the application was providing the possibility to the user to configure a relay, then assembling the 3D models of the parts together depending on the user's configuration and displaying the 3D model of the device.

Each part of a relay has its own configuration diagram and the application was getting the configuration details and assemble connection diagrams of all the parts used in the configuration together. Finally the configuration information and connection diagram of the device were saved in a PDF file. Figure 4 shows the configuration view of the application.

Each version of the relay has its own default configuration word and configuration pattern. Configuration word is a special word consisting of the representative letters of the parts structured based on the configuration pattern. One example of the configuration word is shown on top of figure 4 and it looks like F CBGIA AABAA B1.

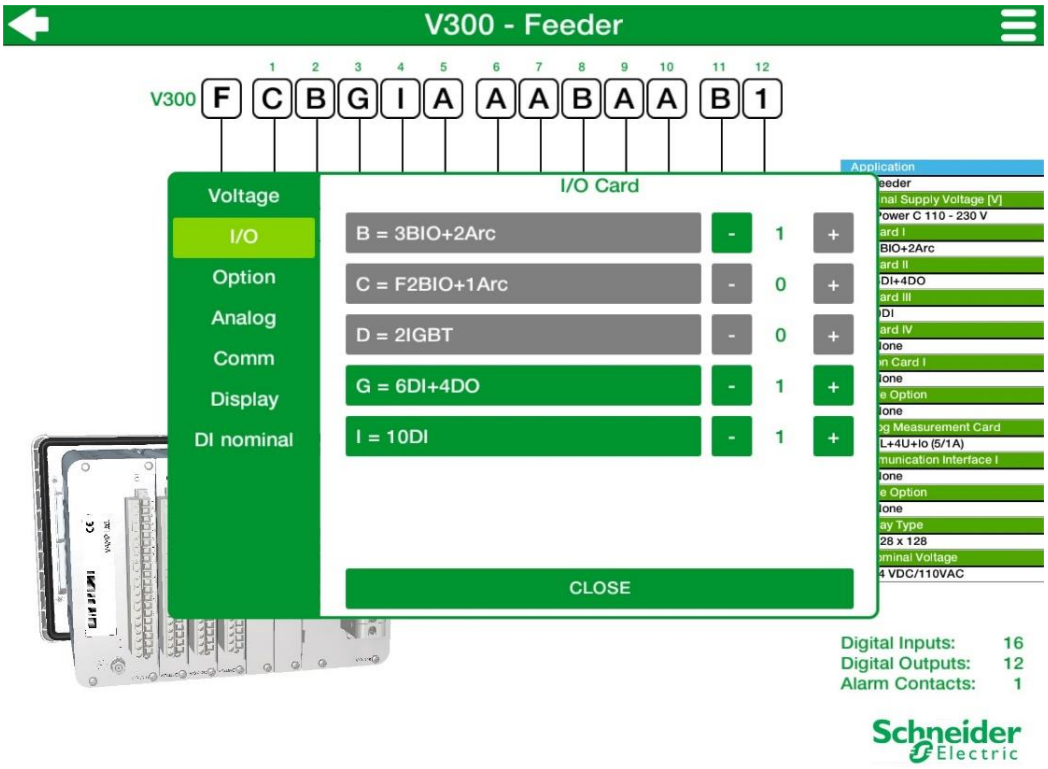

Figure 3 - Screen shot of Configurator interactive application

3.1 Details about protection relays

Some Information about the Schneider Electrics' protection relays that had to be taken into the consideration while designing this application are listed below.

a. There are many versions of Schneider Electrics' protection relays.
b. Each version has mostly its own 3D design, but in some cases it is common between some versions
c. Each mode of a version is presented with a letter (Feeder/F, Motor/M, Differential/D….)
d. Each mode has a default configuration
e. A version might have one or more usage modes
f. Each version has its own configuration word pattern with a special meaning for each character. For instance, X-XXXXX-XXXXX-XX is VAMP 300's pattern
g. One device can be used only in one mode at a time
h. It is possible to change the mode and configuration of a relay

i. Each version has its own manual containing connection diagrams, configuration word pattern, default configuration for each mode, etc.
j. There are thousands of relays for each version with a unique serial number
k. The configuration of a device can be changed
l. Each version has a maximum number of slots in the back side of the device
m. Each slot can accept parts from one or more categories of parts
n. There are many parts for different categories
o. A part can be used in different modes of its own version
p. Some parts can occupy or disable one or more slots
q. Each slot can accept only and only one part
r. Each part has a connection diagram
s. Different parts can have the same connection diagram
t. Each part has its own 3D design

3.2 Application's parts

All of the information listed in section 3.1 had to be handled in the application and saved into a database. In addition, the application must be able to insert/edit/delete/search correct data from the database for a device with a unique serial number which can be any available version of the relays.

3.2.1 Server side application

The server side of Augmented Reality Client-Server application for Schneider Electrics' protection relay is a web application and implemented with Java servlets, JSP (Java Server Pages), HTML, CSS and jQuery running on Tomcat 8.0 server. It is structured in MVC architectural pattern and consist of forty two JSP pages, two jQuery files, one CSS file, eleven servlets and ten Java classes to provide all the possible functions to the administrator and the user. The administrator and the user have to login to the application in order to use the application and they will be logged out automatically in the case they stay inactive for more than ten minutes.

The administrator interface of the web application gives full rights to the administrator of the system to make any changes, for example, adding, editing,

deleting and viewing all the information. The administrator also has the right to add, edit and delete an administrator or a user and change his/her own password.

The user interface of the web application only gives the possibility of updating the profile and changing the password. The user will mostly use the client side of the application to make configurations.

The web application is also responsible for generating the JSON content and providing information to the client application. The client application sends a post request to the web application, then the web application collects the necessary data from the database, generates suitable JSON content and prints it to the browser, then the client application reads the JSON content from the browser.

The Tomcat server is configured to support the HTTPS protocol for communication between web and client applications. In addition, in order to keep the password of the database in a safe place, one database connection pool is made in the Tomcat server that the web application uses to connect to database.

3.2.2 Client application

The client application is an iOS application which is implemented in C# using the Unity 3D engine for iPad and it communicates with the web application. It also has augmented reality and configuration interfaces.

The users have to login in order to be able to use the client application, because the web application requires a username and password to provide data. Both of the interfaces receive/send the data from/to web application in JSON format.

The AR interface reads the serial number of a device, generates a suitable URL and asks the data from the web application to augment the received video from the camera in real time. However, through the configuration interface, the administrator or user can make a valid configuration according to the configuration rules and press the save button, then the application sends it to the web application to be saved as a new configuration. The client application sends the configuration data to the web application and receives the response and informs the configurator person.

3.3 Application requirements specifications

The requirements of the application are prioritized in three different categories. The main focus during the development was on the normal (must have) and expected (should have) requirements, but all of the requirements of all three categories were fulfilled during the project. The requirements of the project and their priority are listed in table 1.

Table 1: The requirements of the application and their priority

Priority	Requirements
Normal requirements (must have)	- Two existing applications should be integrated. - The application should not need modification if any change in data is required and all the data must be updated dynamically - All the configurations of the device during the life cycle should be saved as history - Flexibility of adding new versions, usage modes, categories, parts, slots and users - Data should be saved in MySQL database - The client application must have the ability to work offline - The 3D models should be added or removed during runtime - The colors used in the web app's user interface are defined by the customer - Only the authenticated users should be able to use the application - The client application must be implemented for iOS - The client application must not allow invalid configuration for a device - Each version of the relay must have a default configuration saved in the database.

	- It should be possible to select a connection diagram from a list of diagrams saved in the database - An easy interface to use and as few clicks as possible
Expected requirements (should have)	- The administrator and user must be able to change their password and update their information - The application must not allow adding wrong data - The web application should be logged out if the user does not interact with the application for more than 10 minutes - The data should be transmitted safely - All the required data must be entered into the form, then user will have the possibility to submit. - Passwords should match to the pattern and be encrypted when inserting into the database - The data should not be added repeated in the database
Exciting requirements (nice to have)	- Responsive interface - Showing the name of the user and its role on top of the page.

3.4 Main functions of the application

There are seventy three different methods in the databaseHandler.java file which are responsible for inserting, updating, deleting and getting the data to/from the database. These methods are being called from different pages, controller or classes to handle all the necessary functionalities in the web application or provide all the requested data to the client application.

The main functions of the application are shown in figure 4. Since it is difficult and unclear to demonstrate all the seventy three functions in the diagram, all the relevant function are presented under a general name. For example adding, editing, deleting and searching for configurations is presented as manage configurations.

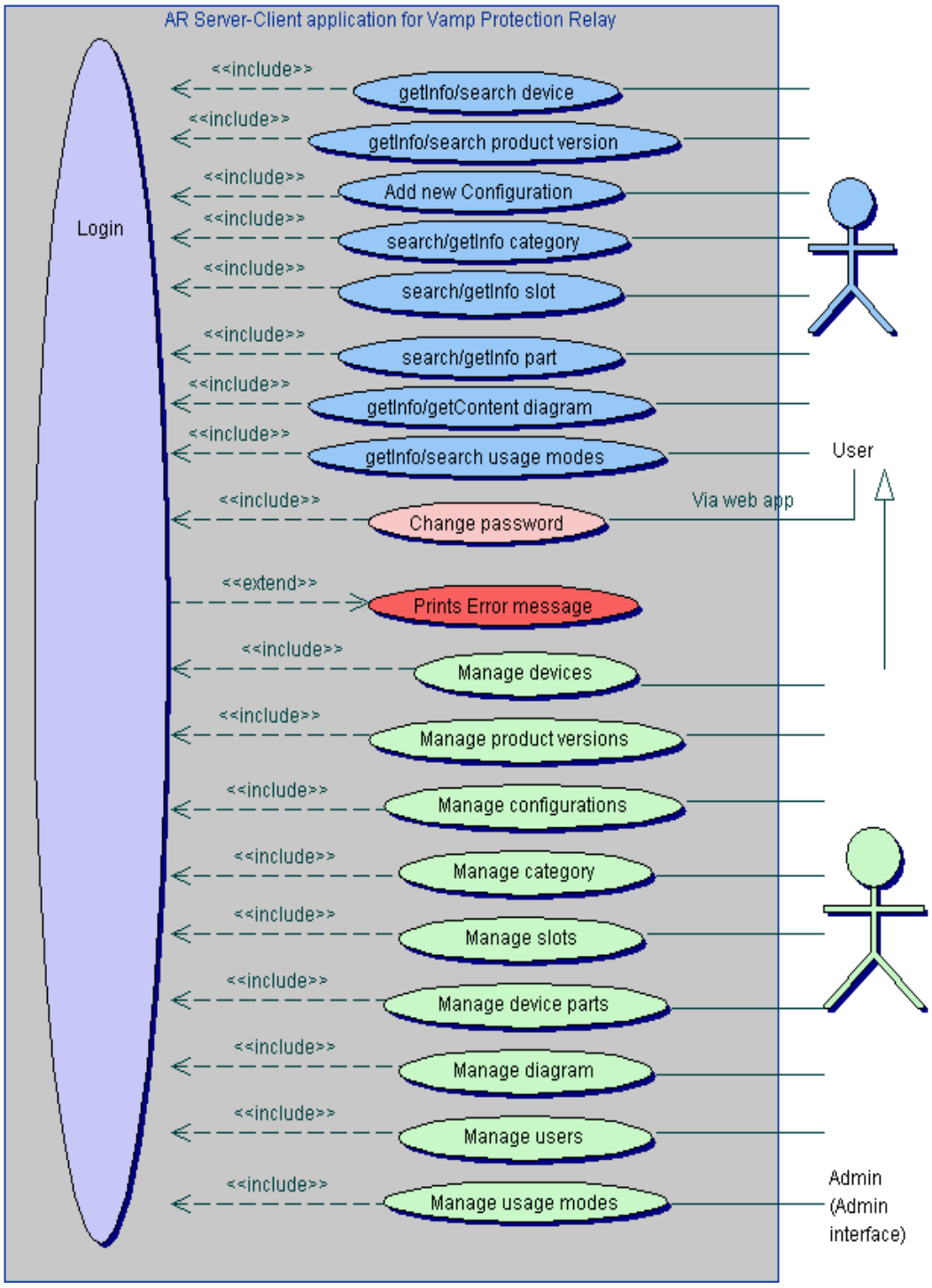

Figure 4 - Main functions of the application

3.5 Main modules of the web application

The relation of the packages is shown in figure 6. The model packages contain the Java classes, the Controller package contains the Java servlets and the View package contains the JSP, jQuery and CSS file.

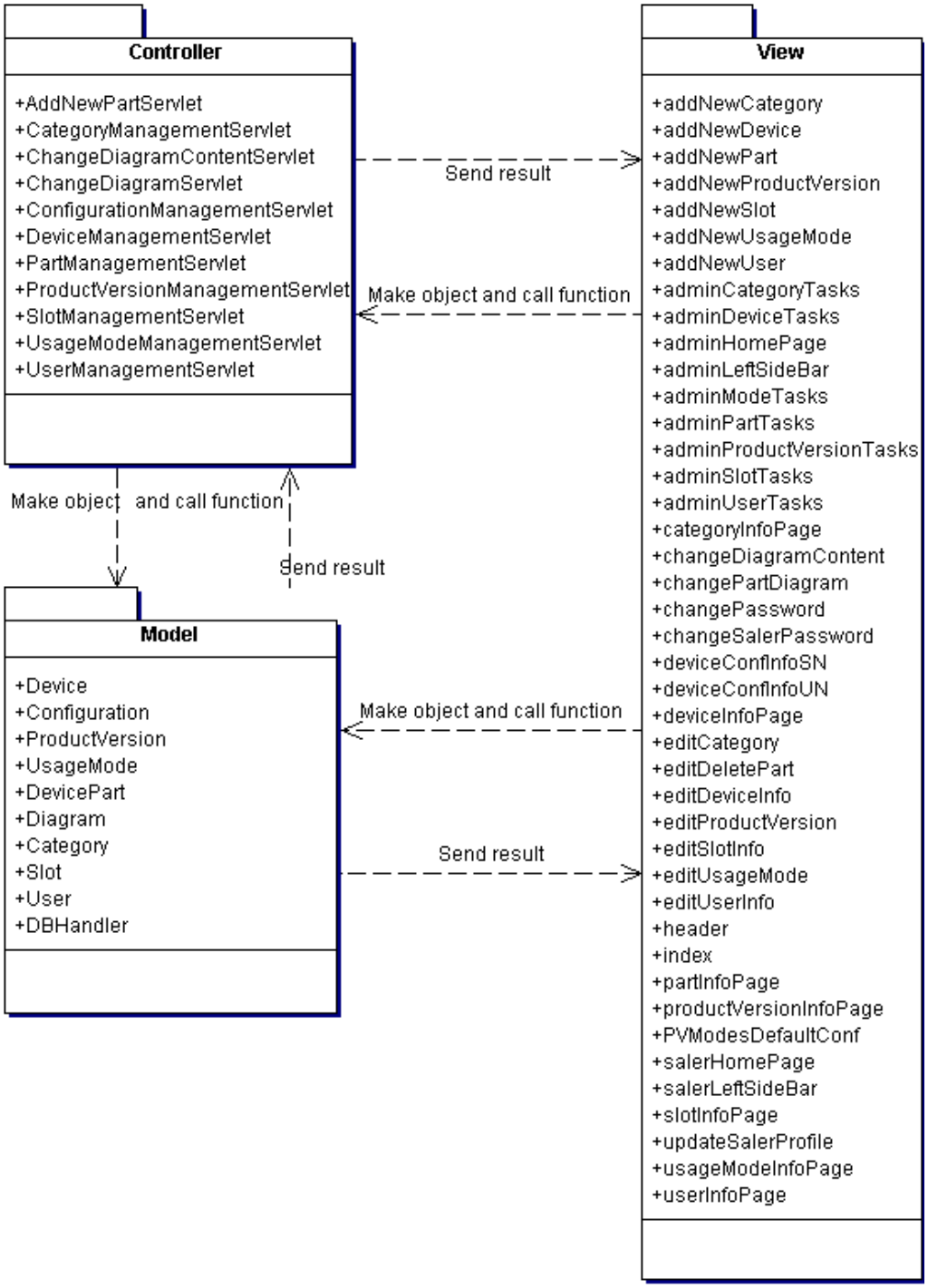

Figure 5 – Main modules diagram

3.6 Server side services

Since some methods are working similarly to each other in different classes, for not repeating the same diagram, a general explanation with a diagram is presented for similar methods. For example, there are adding method for all classes, but only one diagram is presented.

When a user opens a desired page, the application checks if it is necessary to provide any data in the form, then the application calls an appropriate function to receive data and provide to the user in the form if it requires data to display, otherwise an empty form will be displayed.

For adding, editing, deleting and searching any data into/in/from the database, there are suitable pages with different forms which are calling a relevant controller. The user can fill the required fields in the form and submit, then the application will do the task and inform the result to the user.

3.6.1 Login method

The login page for administrator and user is the same, but the application checks the role of the user and redirects the user to the appropriate home page if logging in has succeeded.

The login page is the first page of the web application that will show when any user browses the website. The administrator or user has to provide username and password and click the ‘Login’ button. Then the application will check if the combination of the given username and password exist in the database. Finally, the application will check the role of the user in the system and redirect to his/her homepage, but the application will redirect back to the login page if the logging was not successful.

The client application should also provide username and password when requesting data from the web application. First, the web application checks if the received username and password from the client application are valid, then the data will be provided with the validation code of 85 but if only the username and password were

correct. Otherwise a message will be replied to the client application with validation code of 75.

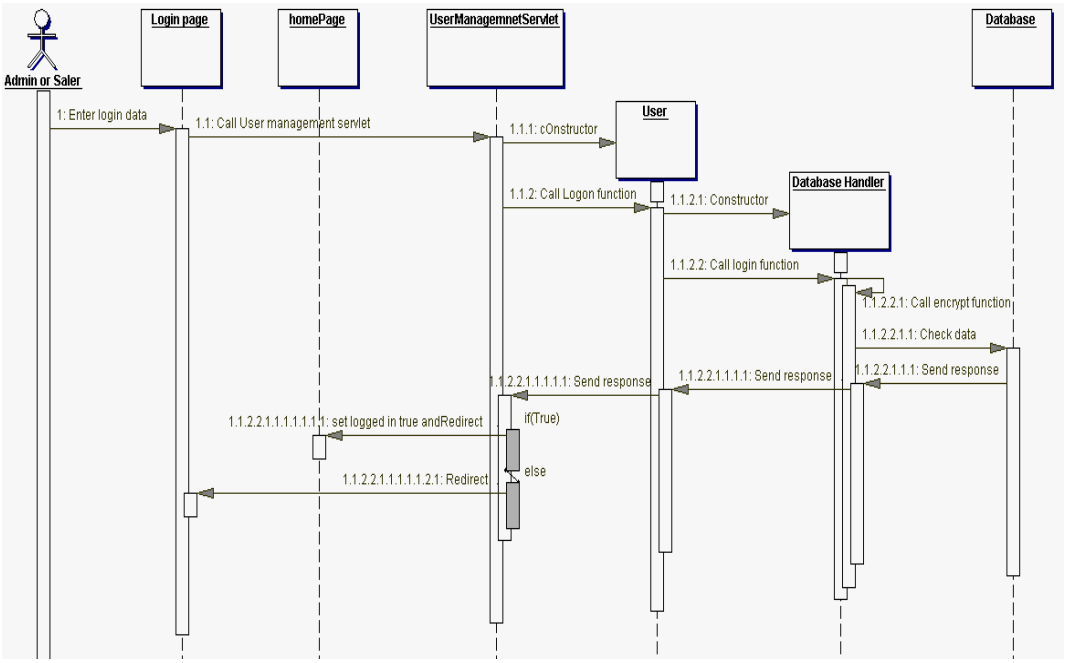

Figure 6 – Details of Login method

3.6.2 Adding new data

There are eight pages for adding data into the database by the administrator and all of them work in the same way. When the administrator browses a page for adding some data into the database, first, the application checks if any data needs to be displayed in the form, then requests the data from an appropriate method and displays it if necessary. Otherwise an empty form will be displayed.

The user can fill the form with suitable and acceptable data, at the same time the application checks if the given data are in the correct format and activates the submission button only if the given data are valid. Then, the user can press the ‘Add’ button.

When the administrator submits the form, it calls an appropriate servlet. The servlet reads the data from the form and passes it to a suitable method to be inserted into the database. Finally, the result of the addition will be printed on the same page as a message.

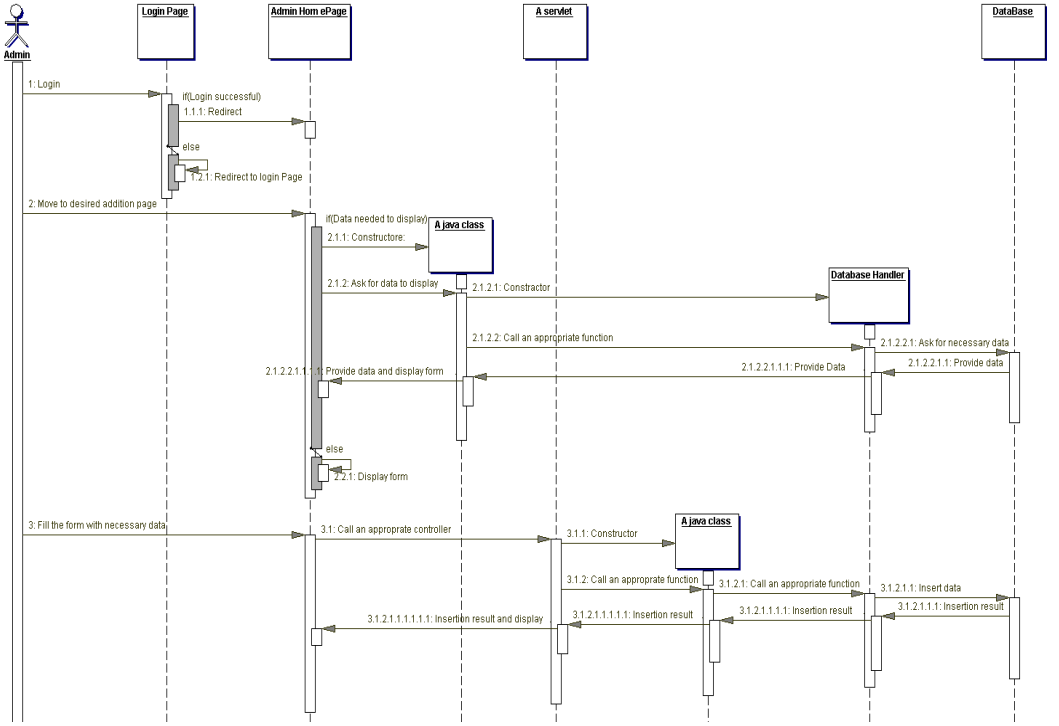

Figure 7 – Details of adding new data

3.6.3 Editing data

There are ten pages for editing data in the database by the administrator and all of them work in the same way. When the administrator browses a page for editing the data in the database, first, the application checks if any data needs to be displayed in the form, then requests the data from an appropriate method and displays it if necessary. Otherwise an empty form will be displayed.

The user should fill the form with suitable and acceptable data and press the ‘Get Info’ button, then the application gets the information that the administrator wants to edit from the database and displays it in the editing form.

The user can edit the data in the form, at the same time the application checks if the given data are in the correct format and activates the submission button only if the given data are valid. Then, the user can press the ‘Edit’ button.

When the administrator submits the form, it calls an appropriate servlet. The servlet reads the data from the form and passes it to a suitable method to be updated in the

database. Finally, the result of the edition will be printed on the same page as a message.

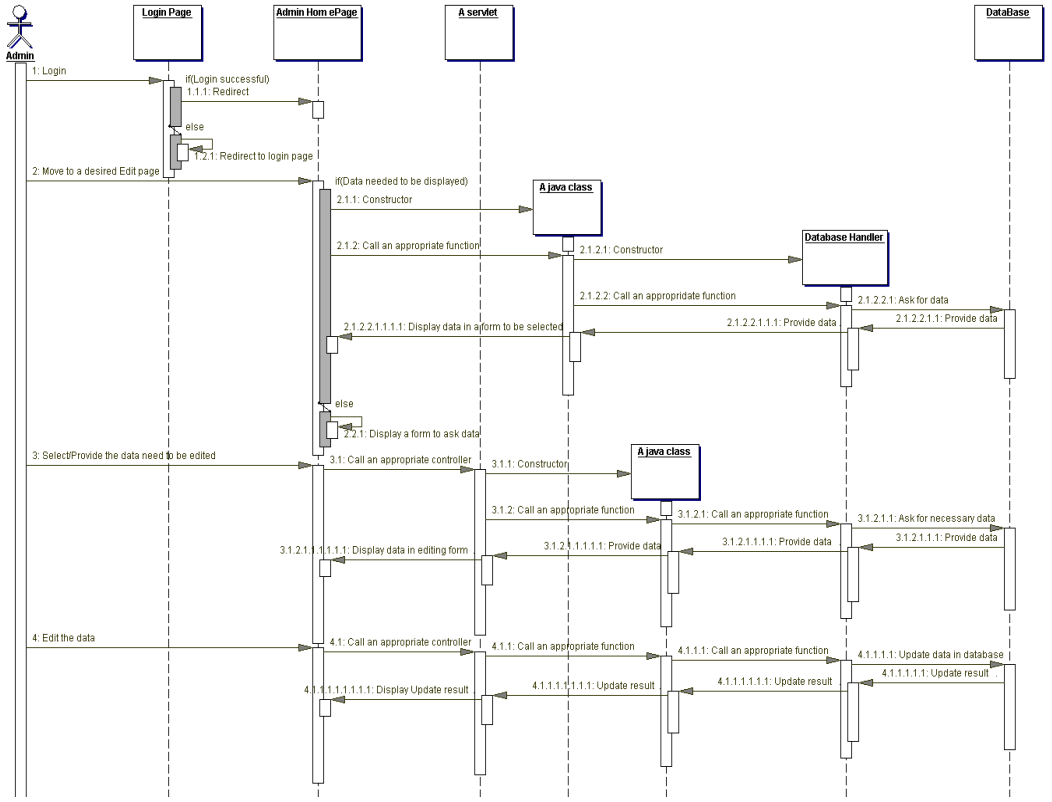

Figure 8 – Details of editing data

3.6.4 Deleting data

There are two pages for deleting data from the database by the administrator and all of them work in the same way. When the administrator browses a page for deleting some data from the database, first, the application checks if any data needs to be displayed in the form, then requests the data from an appropriate method and displays it if necessary. Otherwise an empty form will be displayed.

The user should fill the form with suitable and acceptable data and press the ‘Get Info’ button, then the application gets the information that the administrator wants to delete from the database and displays in the deleting form.

The user can check if he/she is deleting the correct data and agree to the deletion by checking a checkbox, then press the ‘Delete’ button

When the administrator submits the form, it calls an appropriate servlet. The servlet reads the data from the form and passes it to a suitable method to be deleted from the database. Finally, the result of the deletion will be printed on the same page as a message.

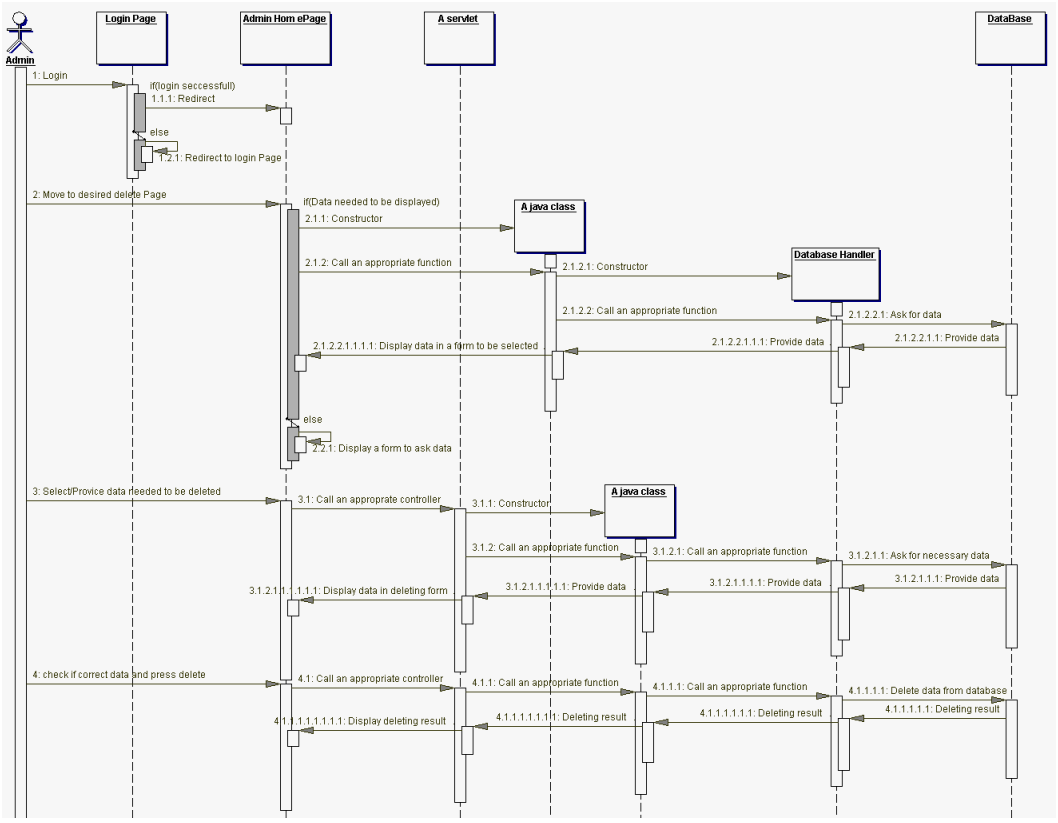

Figure 9 - Details of deleting data

3.6.5 Searching data

There are thirteen pages for searching data from the database by the administrator and all of them work in the same way. When the administrator browses a page for searching some data from the database, first, the application checks if any data needs to be displayed in the form, then requests the data from an appropriate method and displays it if necessary. Otherwise an empty form will be displayed.

The user should fill the form with suitable and acceptable data and press the ‘Get Info’ button, then the application gets the requested information from the database and print on the same page for the administrator.

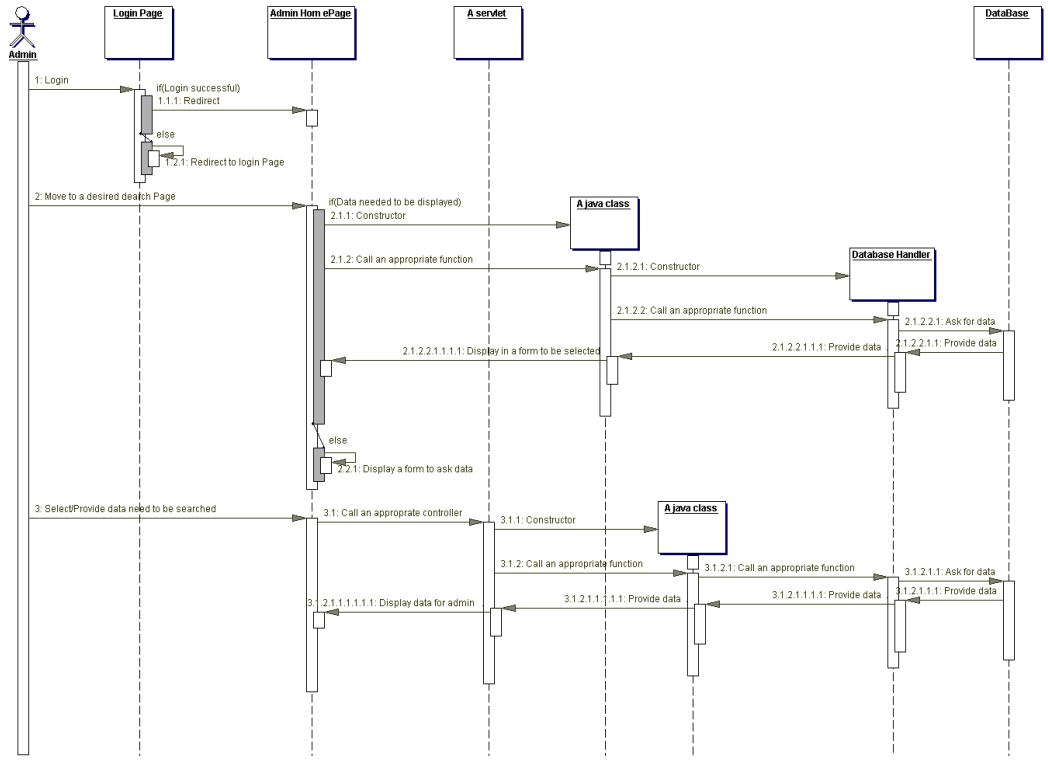

Figure 10 – Details of searching data

3.6.6 Changing password

The user must be logged in and browse the ‘Change password’ page in order to change the password. Since the application knows who is logged in, it will print the name of the user in the form and the user must provide the current password to make sure that the right user is updating the password, then the new password must be entered and repeated.

The application checks if the user has entered valid passwords and they match each other. The submit button will be activated only if the given passwords match the pattern and each other, then the user can press the ‘Change password’ button. Finally, the application updates the password in the database if all the statements were correct and prints the result of changing the password on the same page to the user.

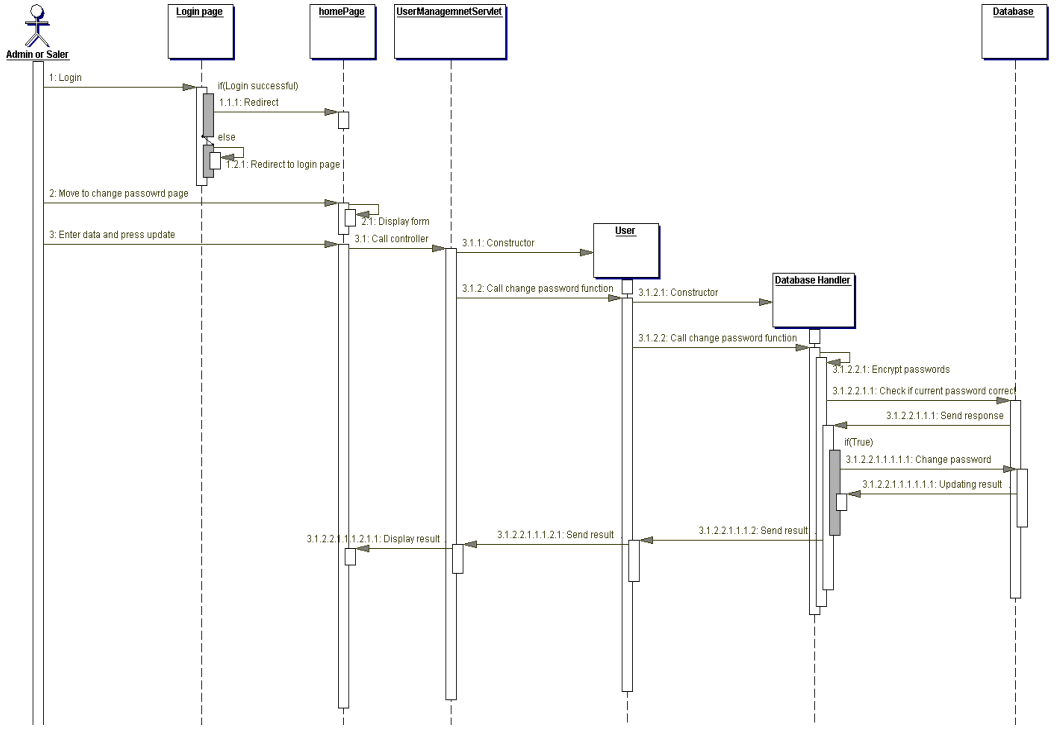

Figure 11 - Details of changing Password

3.6.7 Handling the client requests

When the user works with client application, some data must be requested from the web application. In order to receive the necessary data, the client application must send a post request to the web application. The URL contains the full path of the web application and the name of a suitable controller. In addition the client application passes the username, the password, the name of the method that should be called and the necessary arguments to the web application in a form.

When the client application sends a post request, the web application reads the arguments, calls the method and passes the arguments to it. Then the method gets the data from the database, makes the suitable JSON content and sends it to the controller to be printed in the browser. Finally, the client application reads the data in JSON format from the page, decodes and uses them.

A JSON content with the correct data has validation code of 85 and the client application does not use the data if the validation code is different than 85. The validation code for errors and messages for the failures is 75. For example, if the

given username and password were not correct, the JSON content will have validation code of 75 and an error message.

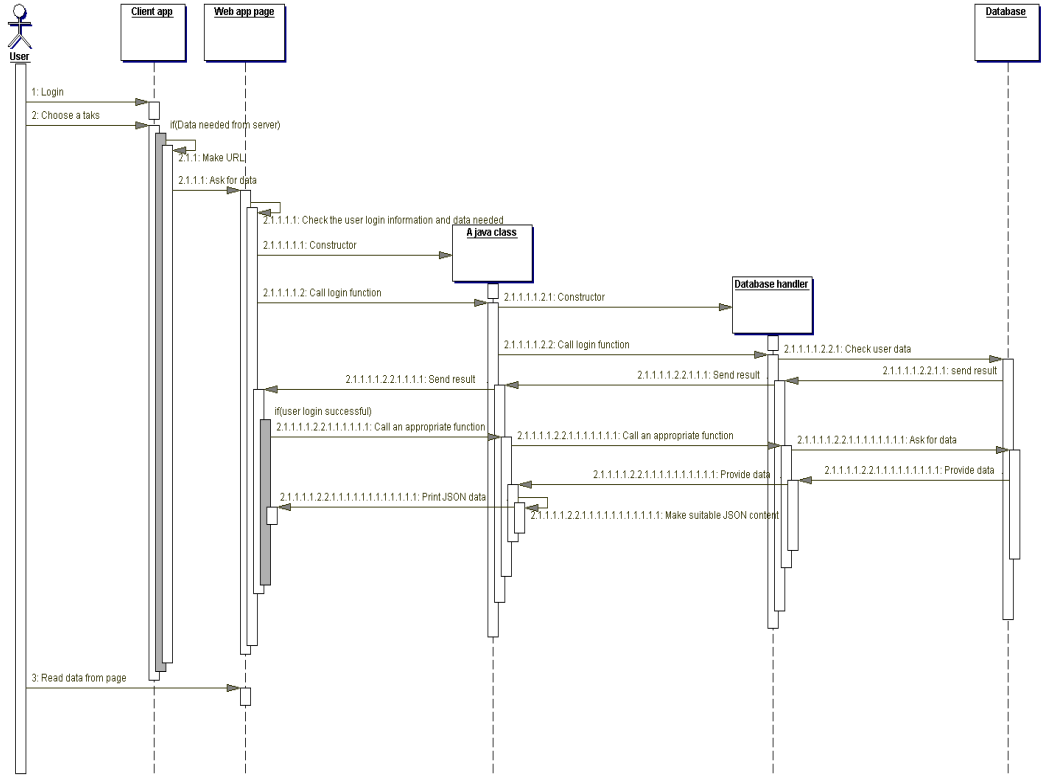

Figure 12 – Details of handling the client application's request

3.7 Web application's architecture

The Server side of the application which is a web application is structured in MVC architectural pattern. The Java classes are located in the 'model' package, the servlets in the 'controller' package and the JSP pages in the 'WebContent package which is representing the 'View' package.

The JSP pages, jQuery files and CSS files are responsible for providing a user-friendly interface. The files located in the 'controller' package are Java servlets which are responsible for controlling input and output data, checking the result of calling functions and preparing a suitable message to the user, the user rights and providing the necessary data to the client application. The files located in the 'model' package are Java classes. The DBHandler.java class is responsible for the database communication and executing queries.

Figure 13 demonstrates the structure of the web application.

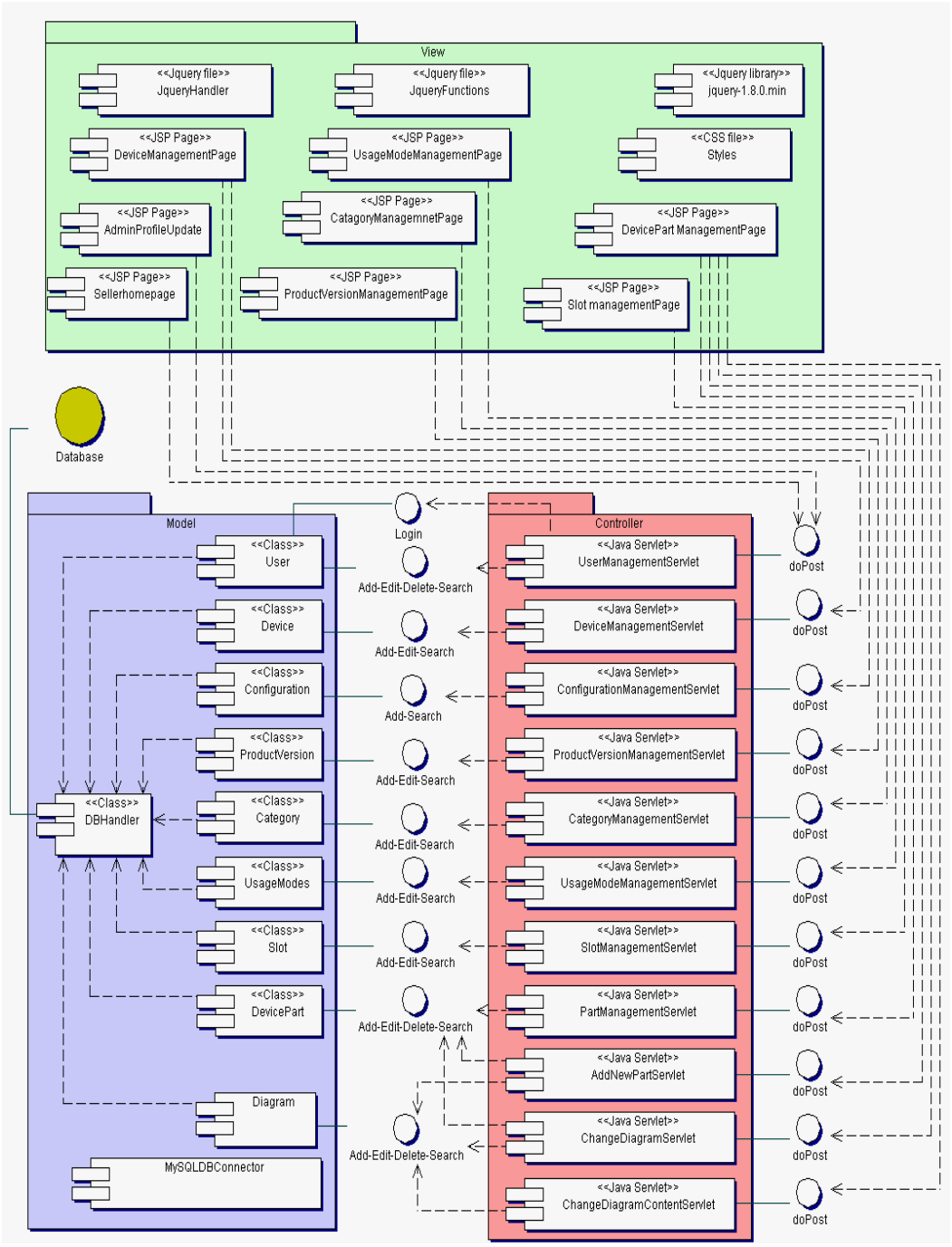

Figure 13 – Web application's architecture

3.8 The system's required infrastructure

The architectural diagram shows how the parts of the system are located physically. The 3D models of the relays are saved in the 3D Studio Blomberg's server, the

database server and Tomcat 1.8 server are set up by the customer. A SQLite database was embedded in the client application and a local folder was made to be able to work offline.

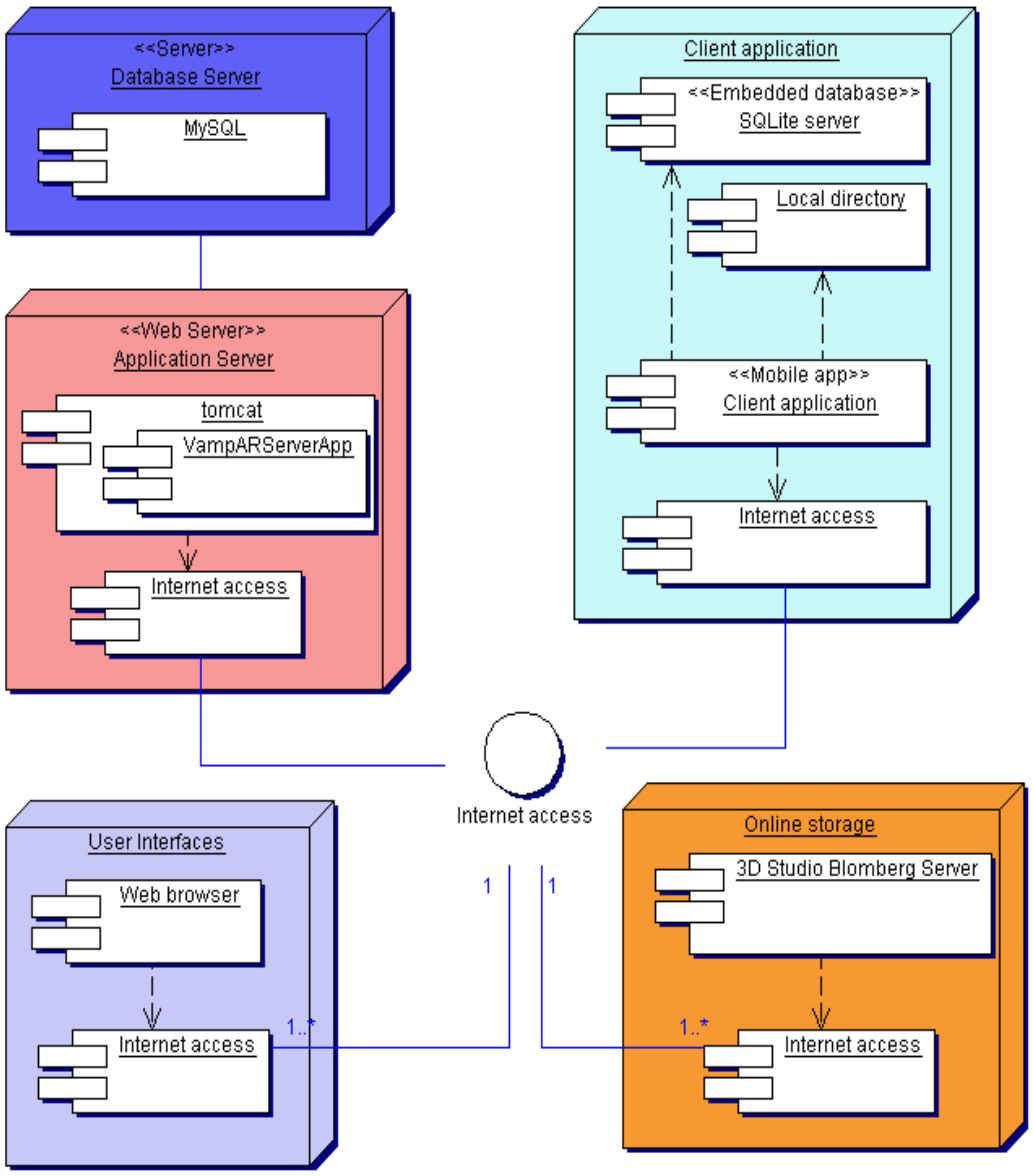

Figure 14 – The required infrastructure of the system

3.9 Web application's view design

It was decided that the user interface of the web application should look similar to the interface of the Schneider Electrics webpage for VAMP products and the same color should be used /6/.

The user interface of the Schneider Electrics webpage has a header bar with Schneider Electrics and VAMP logos and a left sidebar with many buttons. The user interface of the web application also has a header bar with the same logos and a left sidebar with different buttons which are categories of different relevant functions. When the user clicks a button in the left sidebar, all of the buttons in that category appear in another bar in front of the left sidebar, then the user can choose a task by clicking a button.

Figure 15 shows the view of the web application's administrator interface.

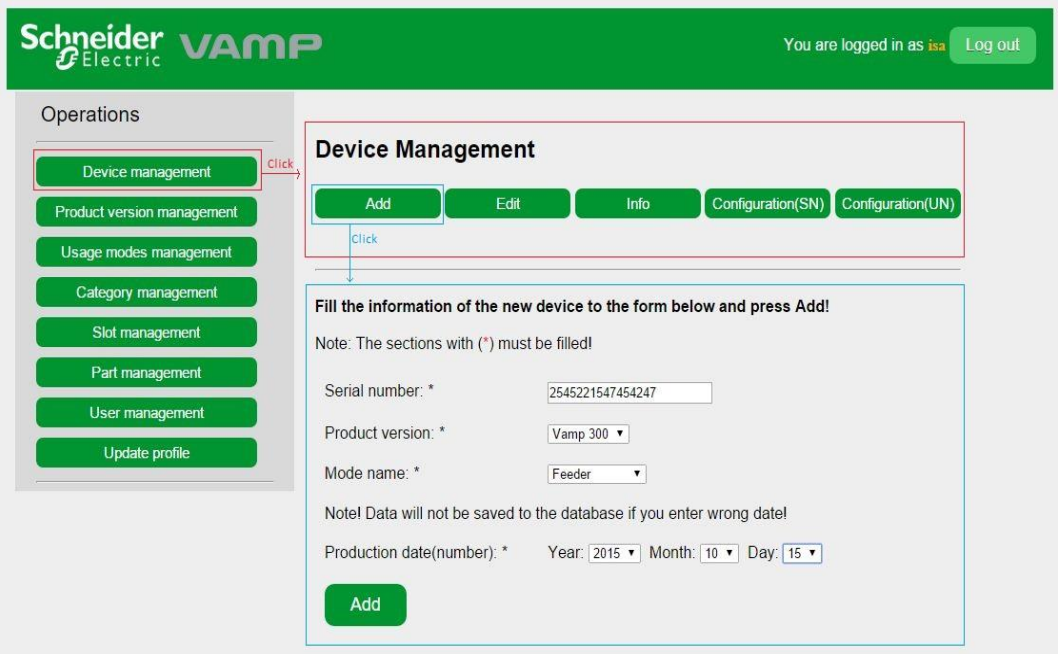

Figure 15 – The view of the web application's administrator interface

4 DATABASE DESIGN

In this application, MySQL database is used in order to keep all the data structured and secured. The database is designed in a way that all the data and the details mentioned in section 3.2 can be saved according to the configuration rules of a relay. It has twelve tables which are devices, configuration, usage_modes, product_version, default_conf, category, slot_map, part_info, part_usage_modes, connection_diagrams, users and user_roles. The database is located in the Schneider Electrics' server.

The primary keys of the database tables are:

- devices: serial_number of a relay which is unique
- configuration: configuration_id
- usage_modes: mode_id but the mode_name is also unique
- product_version: product_version which is the name of the relay version
- default_conf: product_version and mode_id
- category: category_id but the category_name is also unique
- slot_map: product_version and slot_number
- part_info: part_id but part_letter, category_id and product_version are also unique
- part_usage_modes: part_usage_mode_id but the usage_mode_id and part_id are also unique
- connection_diagrams: diagram_id but the diagram_name is also unique
- users: user_name but the user_name together with the password are also unique
- user_roles: role_id but the role_name is also unique

The SQLite database also has the same structure as the MySQL database.

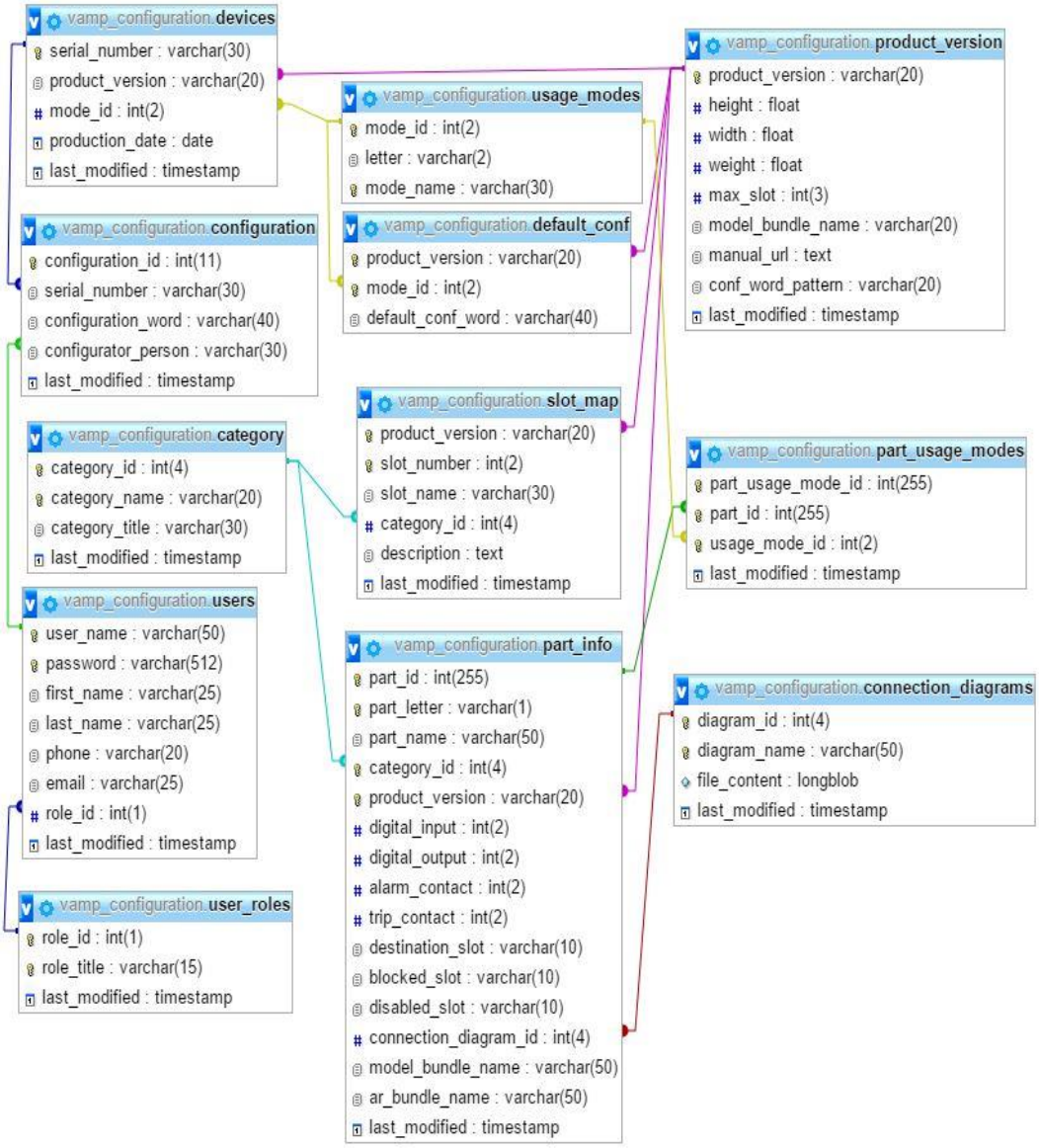

Figure 16 - Database Structure

5 IMPLEMENTATION

Since the application is structured in an MVC architectural pattern, each part of the pattern is explained and the code of the most important parts is provided.

5.1 Model implementation

The ‘model’ package contains the Java classes. Each class is the structure of an entity of the application. The DBHandler.java class is the responsible for database communication via a connection pool that is configured in the Tomcat server and executing the queries.

5.1.1 Creating JSON content

Each class has its own special method which is responsible for providing the information about the objects of the class in JSON format. The number 85 is defined to be the validation code for the correct data, but the validation code for the errors and failures is defined as 75. The implementation of this method for one of the classes is provided in code snippet 3.

```
// This function return the JSON string representation
//of the productVersion object
public String toJson() {
   return
      "{\"validation_code\":\"85\","
      + " \"product_version\":\"" + this.productVersion
      + "\", \"height\":\"" + this.height
      + "\", \"width\":\"" + this.width
      + "\", \"weight\":\"" + this.weight
      + "\", \"max-slot\":\"" + this.maxSlot
      +"\",\"model_bundle_name\":\""+ this.modelBundleName
      + "\", \"manual_url\":\"" + this.manualUrl
      + "\", \"conf_word_pattern\":\"" + this.confWordPattern
      + "\", \"last_update\":\"" + this.lastUpdate
   + "\"}";
}
```

Code snippet 3 – Implementation of a method which creates JSON content

One example of a JSON content created by the web application which is the information of a product version is provided in code snippet 4, it is a valid JSON content and it can be decoded in most programming languages.

```
{
   "validation_code":"85",
   "product_version ":" V300 ",
   "height ":"10.0 ",
   "width ":"10.0",
   "weight":"10.0",
   "max_slot":"12",
   "model_bundle_name": "V300.unity3d",
   "manual_url":"https://m.vamp.fi/dmsdocument/13",
   "conf_word_pattern ":"1 - 5 - 5 - 2 ",
   "last_update":"2016 - 02 - 08 15: 45: 11"
}
```

Code snippet 4 - One example of a JSON content created by the web application

Sometimes the web application has to provide a list of the similar data to the client application. Therefore, some classes have another method for converting a collection of objects to JSON content. In order to add the similar data into the JSON content, the information of the object will be an array of the dataset.

The implementation of one of these methods and a sample result is provided below.

```
// This method gets the product version modes from database as
// a list and converts it to the JSON content and returns
public String getProductVersionModesJson(String productVersion) {
            ProductVersion pv = new ProductVersion();

            // Here it gets the list of the mode ids which from the
            //product_version table
            Vector<Integer> modeIdsVector =
            dbHandler.getProductVersionModeIds(productVersion);

            // Here it adds the validation code to the content
            String jsonString = "{\"validation_code\":\"85\","
            +" \"modes\":[";

            // Here it goes through all of the mode ids, gets the
            // information of the mode and add to the JSON content
            for (int i = 0; i < modeIdsVector.size(); i++) {
            String mode =
            dbHandler.getModeNameByModeId(modeIdsVector.elementAt(i));
            jsonString += "{\"product_version\":\""
               + productVersion + "\", \"mode\":\"" + mode
               + "\"," + " \"default_conf\":\""
               + pv.getProductVersionDefaultConf(productVersion, mode)+"\"}";

            // Here it checks the index of the current mode in the list
```

```
            // and add  ‘,’ if it is not the last mode in the list
            if (i != modeIdsVector.size() - 1)
            jsonString += ",";
            }

            // Here it ends the list and JSON Content
            jsonString += "]}";

            // Here it returns the result
            return jsonString;
}
```

Code snippet 5 – Method for presenting a list of similar data in JSON content

The result of a method which provide the content of a vector in JSON format:

```
{
   "validation_code": "85",
   "modes": [
      {
            "product_version": "V300",
            "mode": "Motor",
            "default_conf": "CAAAAAACAAB3"
      }, {
            "product_version": "V300",
            "mode": "Feeder",
            "default_conf": "CAAAAAABAAB3"
      }, {
            "product_version": "V300",
            "mode": "Transformer differential",
            "default_conf": "CAATAAACAAB3"
      }, {
            "product_version": "V300",
            "mode": "Generator",
            "default_conf": "CAAAAAACAAB3"
      }
   ]
}
```

Code snippet 6 - An example of the JSON content to provide a list of data

5.1.2 Password encryption

The application encrypts the password before saving to the database and it uses AES (Advanced Encryption Standard /11/.) encryption algorithm for encrypting and decrypting the password. The implementation of the encrypt() and decrypt() methods are demonstrated in code snippet 7.

```
private static final String ALGO = "AES";
private static final byte[] keyValue = new byte[] {'*', '*', '*', '*', '*', '*',
'*', '*', '*', '*', '*', '*', '*', '*', '*', '*'};

// This method creates and returns the encryption key
private static Key generateKey() throws Exception {
   Key key = new SecretKeySpec(keyValue, ALGO);
   return key;
}

// This method encrypts the string using the generated key and
// Returns the encrypted string
public static String encrypt(String Data) throws Exception {
   Key key = generateKey();
   Cipher c = Cipher.getInstance(ALGO);
   c.init(Cipher.ENCRYPT_MODE, key);
   byte[] encVal = c.doFinal(Data.getBytes());
   String encryptedValue = new BASE64Encoder().encode(encVal);
   return encryptedValue;
}

// This method decrypts the encrypted data using the generated key
// and the plain text
public static String decrypt(String encryptedData) throws Exception {
   Key key = generateKey();
   Cipher c = Cipher.getInstance(ALGO);
   c.init(Cipher.DECRYPT_MODE, key);
   byte[] decordedValue =
            new BASE64Decoder().decodeBuffer(encryptedData);
   byte[] decValue = c.doFinal(decordedValue);
   String decryptedValue = new String(decValue);
   return decryptedValue;
}
```

Code snippet 7 - Implementation of the password encryption and decryption

5.2 Controller implementation

The ‘controllers’ package consists of eleven Java servlets and each servlet has Get and Post methods. The Get method of the servlet redirects to the login page of the web application.

The Post method is responsible for handing the requests from the forms of the web application and from the client application. It checks if the request is from a page or from the client application, then reads the required arguments depending on the

requested data. The application replies with an error message if any argument was missing in the request.

When the Post method of a servlet is being called by a form, it reads the name of the submit button, then reads necessary data from the form fields according to the name of the button which is representing the name of the method that should be called. Finally, the servlet sends an appropriate message to be printed on the JSP page that its form has called the Post method of the servlet. One example of the Get and Post methods are provided in code snippet 8.

```
// This is the Get method which redirects to the login page
protected void doGet(HttpServletRequest request,
 HttpServletResponse response)throws ServletException,
  IOException {
    // Here It redirect the get method to login page
    response.sendRedirect("index.jsp");
}

// This is Post method which is responsible for handling requests
// from the pages and from the client application
protected void doPost(HttpServletRequest request,
  HttpServletResponse response)      throws ServletException,
  IOException {

    // here it checks if  'submit' parameter exists in the request,
    // if yes, therefore request is from a page
    if (request.getParameter("submit") != null) {
      HttpSession session = request.getSession(false);

      // it checks if the user is logged in
      if (session.getAttribute("adminId") != null) {

        // Here it gets the value of the submit button which is the
        // name of the method too
        String action = request.getParameter("submit");
        if (action.equalsIgnoreCase("Add")) {

          ProductVersionManagementServlet pvms =
            new ProductVersionManagementServlet();

          // Here it reads the parameters from the form
          String productVersionAdd =
            request.getParameter("productVersionAdd");

          float heightAdd =
            pvms.convertToFloat (request.getParameter("heightAdd"));
```

```
    float widthAdd =
     pvms.convertToFloat(request.getParameter("widthAdd"));

    float weightAdd =
     pvms.convertToFloat(request.getParameter("weightAdd"));

    int maxSlotAdd =
     pvms.convertToInt(request.getParameter("maxSlotAdd"));

    String modelBundleNameAdd =
     request.getParameter("modelBundleNameAdd");

    String manualUrlAdd =
     request.getParameter("manualUrlAdd");

    String confWordPatternAdd =
     request.getParameter("confWordPatternAdd");

    // Here it generates an object of product version using
    // the values from the form
    ProductVersion productVersion =
     new ProductVersion(productVersionAdd, heightAdd,
      widthAdd, weightAdd, maxSlotAdd, modelBundleNameAdd,
       manualUrlAdd, confWordPatternAdd);

    // Here it calls the method to add the product versio
    // the database
    int result = productVersion.addNewProductVersion();
    if (result == 1) {
     String message =
      "<font size=4 color=green>Product Version "
      + "is added sucessfully!</font>";

     // Here it sets a message depending on the result and
     // redirect to the same page
     session.setAttribute("task", message);
     response.sendRedirect("addNewProductVersion.jsp");
    } else {
     String message =
      "<font size=4 color=red>Device was "
      +"NOT added sucessfully!</font>";

     // Here it sets a message depending on the result and
     // redirect to the same page
     session.setAttribute("task", message);
     response.sendRedirect("addNewProductVersion.jsp");
    }
   }
  } else {
   response.sendRedirect("index.jsp");
```

```
 }
// Here it checks if the ‘method’ parameter exists in the request,
// if yes, therefore the request is from the client application
} else if (request.getParameter("method") != null) {
 PrintWriter out = response.getWriter();

 // here it reads the parameters
 String userName = request.getParameter("username");
 String password = request.getParameter("password");

 if (userName != null && !userName.isEmpty()
     && password != null && !password.isEmpty()) {

  // Here it authorizes the user
  User user = new User();
  Boolean isLoggedIn = user.login(userName, password);

  if (isLoggedIn) {

   // Here it reads the method parameter which is the name of
   // the method that should be called
   String method = request.getParameter("method");
   if (method != null && !method.isEmpty()) {
    if (method.equalsIgnoreCase("getProductVersionInfo")) {

     // Here it reads the value of the parameters
     String pv = request.getParameter("productVersion");
     if (pv != null && !pv.isEmpty()) {
      ProductVersion pv2 = new ProductVersion();

      // Here it call the method to get the product version info
      ProductVersion productVersion =
       pv2.getProductVersionInfo(pv);
      if (productVersion != null)

       // If result received, converts to JSON format and prints
       out.print(productVersion.toJson());
      else // Otherwise it prints error messages with code 75
       out.print("{\"validation_code\":\"75\", \"error\""
       +":\"Product version can not be found!\"}");
     } else
      out.print("{\"validation_code\":\"75\", \"error\":"
      +"\"Missing argument!\"}");
    } else
     out.print("{\"validation_code\":\"75\", "
     +"\"error\":\"Unknown method!\"}");
   } else
    out.print("{\"validation_code\":\"75\", "
    +"\"error\":\"Missing argument!\"}");
  } else
   out.print("{\"validation_code\":\"75\", "
```

```
        +"\"error\":\"Login was not successful!\"}");
      } else
       out.print("{\"validation_code\":\"75\", "
       +"\"error\":\"Missing argument!\"}");
    } else {
     response.sendRedirect("index.jsp");
  }
}
```

Code snippet 8 – An example of Get and Post methods

The Post method of the ‘Part Management Servlet’ servlet is also responsible for displaying the connection diagram of a device part as an image in the browser when the servlet is called. Then the client application gets the connection diagram from the browser. Code snippet 8 shows how the post method of ‘Part Management Servlet’ displays a PNG file in the browser.

Since it is not allowed to make BufferedOutputStream and PrintWriter response types for the same request, therefore it is not possible to define them as global variable.

```
protected void doPost(HttpServletRequest request,
 HttpServletResponse response) throws ServletException,
 IOException {

 // Here it checks if method parameter exists in the request,
 // if yes, therefore the request is from client
 if (request.getParameter("method") != null) {

  // here it reads the parameters
  String userName = request.getParameter("username");
  String password = request.getParameter("password");

  if (userName != null && !userName.isEmpty() &&
       password != null && !password.isEmpty()) {
   User user = new User();
   // here it authorizes the user
   Boolean isLoggedIn = user.login(userName, password);

   if (isLoggedIn) {

    // Here it reads the method parameter which is the name of
    // the method that should be called
    String method = request.getParameter("method");
    if (method != null && !method.isEmpty()) {
```

```
if (method.equalsIgnoreCase("getConnectionDiagram")) {

 // Here it reads the value of the parameters
 String partLetter =
  request.getParameter("partLetter");

 String categoryName =
  request.getParameter("categoryName");

 String productVersion =
  request.getParameter("productVersion");

 DevicePart devicePart = new DevicePart();

 // Here it ask for the PNG file and buffers it
 BufferedInputStream bis =
  devicePart.getDiagramContent(partLetter,
   categoryName, productVersion);

 BufferedOutputStream output = null;
 if (bis != null) {
  try {
   output = new BufferedOutputStream
                            (response.getOutputStream());

   byte[] buffer = new byte[8192];

   // Here it goes through the buffered content and
   // write it to the browser
   for (int length = 0;(length = bis.read(buffer)) > 0;
            length++) {
    output.write(buffer, 0, length);
   }
  } catch (Exception) {
  } finally {
   if (output != null)
    try {
     // Closes the BufferedOutputStream
     output.close();
    } catch (Exception) {
    }

   if (bis != null)
    try {
     // Closes the BufferedInputStream
     bis.close();
    } catch (Exception) {
   }
  }

 // prints error message with code 75 if not succeed
```

```
            } else {
              PrintWriter out = response.getWriter();
              out.print("{\"validation_code\":\"75\", "
              +"\"error\":\"The content can not be displayed!\"}");
              out.close();
            }
          } else {
            PrintWriter out = response.getWriter();
            out.print("{\"validation_code\":\"75\", "
            +"\"error\":\"Unknown method!\"}");
            out.close();
          }
        } else {
          PrintWriter out = response.getWriter();
          out.print("{\"validation_code\":\"75\", "
          +"\"error\":\"Missing argument!\"}");
          out.close();
        }
      } else {
        PrintWriter out = response.getWriter();
        out.print("{\"validation_code\":\"75\", "
        +"\"error\":\"Login was not successful!\"}");
        out.close();
      }
    } else {
      PrintWriter out = response.getWriter();
      out.print("{\"validation_code\":\"75\", "
      +"\"error\":\"Missing argument!\"}");
      out.close();
    }
  } else {
    response.sendRedirect("index.jsp");
  }
}
```

Code snippet 9 – Post method for displaying a PNG file in the browser

One example of a connection diagram which the ‘Part Management Servlet’ provides to the client application is shown in figure 17.

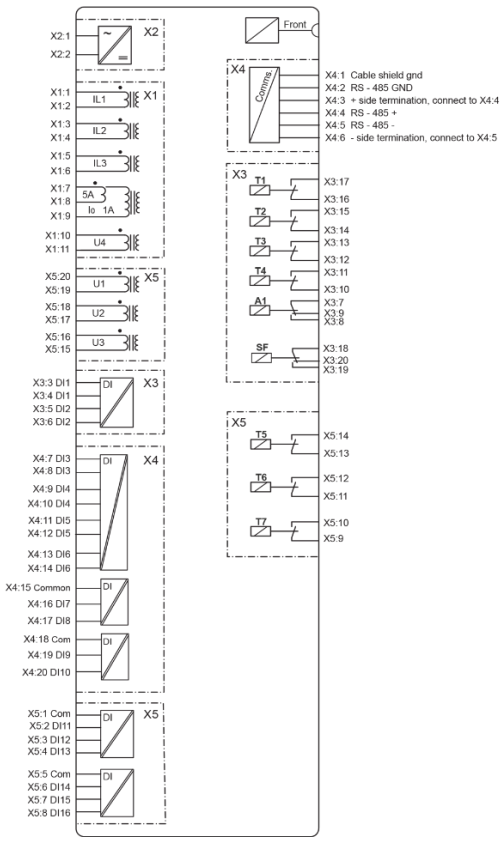

Figure 17 - One example of a connection diagram

5.3 View implementation

The ‘WebContent’ package which plays the role of the ‘View’ package in this application, contains JSP, jQuery, CSS, XML files and the necessary images. All of the files in this package cooperate with each other to provide a user interface that the administrator can insert/update the data to/in the database. The first page of the user interface, in other words, the index.jsp page of the application is the login page.

Both the administrator and the user log in from the same login page. The role of the user is saved in the database and the application checks the role of the user and redirects to his/her home page depending on his/her role. All of the possible and allowed tasks are categorized and listed as links in the left side bar of the web page. The individual tasks of each category appear on top of the main container when the user clicks a category link.

An appropriate form is displayed when the user clicks the task link from the list of tasks which are on top of the main container. Then the user can fill/edit the data in the form and submit or request and receive data from the database.

5.3.1 JSP implementation

When creating a JSP file, there were two jQuery files, a jQuery library, a CSS file and a favicon.ico image that had to be linked to it. The administrators and users have their own leftSideBar.JSP file that is also included in their homepage. In addition, some classes from the model package also had to be imported if necessary.

In this application, HTTP Sessions are used to check if the user has logged in and who is logged in. Code snippet 10 shows the structure of a normal JSP file in this application.

```
<!-- Here it imports the necessary packages -->
<%@page import="model.*"%>
<%@page import="java.util.*"%>
<%@ page
   language="java"
   contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1" %>
<html>
 <head>
   <title>Schneider Vamp</title>
   <link rel="shortcut icon" href="helpers/favicon.ico">

   <script type="text/javascript" src="jquery-1.8.0.min.js">
   </script>

   <script type="text/javascript" src="helpers/functions.js">
   </script>

   <script type="text/javascript" src="helpers/JqueryHandler.js">
   </script>

   <link rel="stylesheet" type="text/css"
         href="helpers/styles.css" />
 </head>
 <body>

   <!-- here it includes the header file on top of the body -->
   <jsp:include page="header.jsp" />
   <div class="bodyContainer">
```

```
    <!-- here it includes the admin left sidebar -->
    <jsp:include page="adminLeftSideBar.jsp" />

    <!-- this is the main container that forms and data are
         being displayes -->
    <div class="mainContainer">

      <!-- Here it includes the task link in front of the left
           side bar -->
      <jsp:include page="adminCategoryTasks.jsp" />
      <%
      // here it authorizes the user
      if (session.getAttribute("adminId") != null) {

        // here it checks if the task attribute is set, if yes,
        // then shows the message and removes the attributes
        if(session.getAttribute("task") != null){
          out.println("<hr/><br/>"
          + session.getAttribute("task") + "<br/>");

          session.removeAttribute("task");
        }
      %>
      <!--
        The content is inserted here and it
        can be a form, information to display...
      -->
      <%
      } else {
        // It redirects to the login page if adminId attribute
        // was not set
        response.sendRedirect("index.jsp");
      }
        %>
    </div>
   </div>
  </body>
</html>
```

Code snippet 10 – Structure of a JSP file

5.3.2 jQuery implementation

In this application, jQuery framework is used to handle some events in the user interfaces. The jQuery files are responsible for handling some events like enabling or/and disabling the submit button of a form depending on the user inputs and doing some changes on the data during the interaction with the user.

There are two jQuery files linked to all of the JSP files. One of the jQuery files keeps the track of the events in the user interface and call a function if any event happens, where the other one contains only jQuery functions.

Code snippet 11 shows the structure of the JQueryHandle.js file and a function of the functions.js file that is being called. The on_addNewCategory_change() function enables or disables the submit button depending on user inputs in a form.

```
// This is from JQueryHandler.js file
$(document).ready(function() {
 // here it disables the submit button in the from
 $('#addNewCategoryButton').attr('disabled', 'disabled');

 // It calls an appropriate function when the inputs in form change
 $(' #categoryName, #categoryTitle').on('input',
                                        on_addNewCategory_change);
});

// This is from functions.js file
function on_addNewCategory_change() {

 // Here it gets the values from the input boxes
 var searchCategory = $('#categoryName').val();
 var searchProductVersion = $('#categoryTitle').val();

 // Here it checks if the inputs are given and enables/disables
 // the submit button in the form, the color of the button
 // also being changed
 if (searchCategory.length > 0
       && searchProductVersion.length > 0) {
  $('#addNewCategoryButton').removeAttr('disabled');
  $('#addNewCategoryButton').css('background-color', '#009530');
 } else {
     $('#addNewCategoryButton').attr('disabled', 'disabled');
     $('#addNewCategoryButton').css('background-color','#adadad');
 }

 // Here it deletes the variables before leaving the function
 delete searchCategory;
 delete searchProductVersion;
 return false;
}
```

Code snippet 11 – Event handling by jQuery

One of the events that jQuery takes care of is checking if the selected year is a leap year or non-leap, then updating the drop down list of the days of February.

When the form is being loaded for the first time, the drop down list for the days of the month will be empty until the user selects the year and month, then the application checks if it is a leap or non-leap year and add the days of the selected month in the drop down list. Code snippet 12 shows how the jQuery function updates the drop down list of the days for each month.

```
function findFebruaryDaysByYear() {

 // Here it gets the year and month from the form
 var yearAddDevice = $('#yearAddDevice').find(":selected").val();
 var monthAddDevice =
          $('#monthAddDevice').find(":selected").val();

 if (yearAddDevice.length > 0 && monthAddDevice > 0) {

  // Here it converts the year and month to integer values
  var year = parseInt(yearAddDevice);
  var month = parseInt(monthAddDevice);
  var days = 0;

  // Here it find the number of the days in the month and
  // initializes the number of the days,
  if (month == 1 || month == 3 || month == 5 || month == 7
                 || month == 8 || month == 10 || month == 12)
   days = 31;

  // Here it check if it is a Leap year or not and initializes
  // the correct number of the days for February
  else if (month == 2 && !(year % 400 == 0 || (year % 100 != 0
                 && year % 4 == 0)))
   days = 28;
  else if (month == 2 && (year % 400 == 0 || (year % 100 != 0
                 && year % 4 == 0)))
   days = 29;
  else if (month == 4 || month == 6 || month == 9 || month == 11)
   days = 30;

  // Here it add the days into the drop down list in the form
  for (var i = 1; i <= days; i++) {
   $('#dayAddDevice')
      .append($("<option></option>").attr("value", i).text(i));
  }
 } else {
  // Disables the submit button if year and month is not inserted
```

```
    $('#getModeInfoButtonEdit').attr('disabled', 'disabled');
    $('#getModeInfoButtonEdit').css('background-color', '#adadad');
  }

  // Here it deletes the variables before leaving the function
  delete yearAddDevice;
  delete monthAddDevice;
  return false;
}
```

Code snippet 12 – jQuery function to update the days in the drop down list

In some cases jQuery is also used to make changes in the forms according to user's need. For example, when the administrator wants to add a new device part into the database, he/she must upload a connection diagram or choose from the list of the existing connection diagrams in the database.

The form is designed in a way that the input for uploading files and drop down list of the existing diagrams can be toggled by the user. Code snippet 13 shows how the jQuery function handles toggle between to input type.

```
$('#toggleDiagramSelection').toggle(
  function() {

    // Here it changes the text of the titles in the form
    $('#toggleDiagramSelection').text(
                                        'Click here to upload a diagram');
    $('#diagramSelectionTitle').text(
                                        'Select diagram: *');
    $('#diagramHiddenInfo').val('select').trigger('change');

    // Here it switches between the upload and selection inputs
    $('#diagramSelection').show();
    $('#diagramUpload').hide();
    $('#existingConnectionDiagram').val("");
    $('#connectionDiagram').val("");

    // Here it disables the submit button
    $('#addNewPartButton').attr('disabled','disabled');
    $('#addNewPartButton').css('background-color', '#adadad');
  }, function() {
    // Here it changes the text of the titles in the form
    $('#toggleDiagramSelection')
                  .text('Click here to select an existing diagram');
    $('#diagramSelectionTitle').text('Upload diagram: *');
    $('#diagramHiddenInfo').val('upload').trigger('change');
```

```
    // Here it switches between the upload and selection inputs
    $('#diagramSelection').hide();
    $('#diagramUpload').show();
    $('#existingConnectionDiagram').val("");
    $('#connectionDiagram').val("");

    // Here it disables the submit button
    $('#addNewPartButton').attr('disabled','disabled');
    $('#addNewPartButton').css('background-color', '#adadad');
});
```

Code snippet 13 - Implementation of the toggle function

5.4 Client communication implementation

Since the implementation of the whole client application is not a part of this thesis work, only those functions will be explained that belong to this thesis work.

5.4.1 Communication between web and client applications

In order to request data from the web application, the client application must send a post request with the required arguments. For example, for asking information about a version of the relays, the client application has to send the username, password, the name of the method and the name of the product version to the web application. Then the web application can authorize the user, it knows what method to call and the arguments that should be passed to the method.

The requested information will be printed in the browser and the client application will read it. Since the client application is implanted in Unity 3D engine, special methods are required to be used, because the application updates the scenes every frame and the rate is almost 50-60 frame per second and the application should handle the delay.

Code snippet 14 shows the implementation of the WebApp.cs class with some sample methods which make the suitable URL, request the category data from the web application, decodes the JSON content, reads the data and call a method from the SqliteHandler.cs class to insert into the SQLite database.

```
public class WebApp : MonoBehaviour{
```

```
string domainURL ="http://web.3dstudio.fi:9090/VampConfARApp_Server/";

// The definition of the delecate
public delegate void OnCategoryInfoSucceed(
           string categoryInfoResult, string type);

// The definition of the event
public event OnCategoryInfoSucceed onCategoryInfoSucceed;
SqliteHandler sqliteHandler = new SqliteHandler();

void Start(){
 onCategoryInfoSucceed += WebApp_onCategoryInfoSucceed;
}

// This methos id responsible for sending a post request to the
// web application by using WWW and WWWForm classes
public IEnumerator getCategoryInfoByCategoryNameIE(string categoryName){
 string categoryInfoResult = "";

 // Here it adds the name of the servlet that should be called to
 // the URL
 url = domainURL + "CMS";

 // Here it generates the form and adds the parameters
 WWWForm form = new WWWForm();
 form.AddField("username", "isa");
 form.AddField("password", "asadiIsa1985");
 form.AddField("method", "getCategoryInfoByCategoryName");
 form.AddField("categoryName", categoryName);

 // Here it posts the from to the url of the servlet
 WWW getCategotyInfo = new WWW(url, form);

 // Here it waits for the response, it will continue when
 // all the content of the response is received
 yield return getCategotyInfo;

 // Here it checks if any error occurs and calls the error event
 if (!string.IsNullOrEmpty(getCategotyInfo.error))
 {
  onEventError(getCategotyInfo.error);
 } else {

  // Here it reads the received content character by character to
  // the end
  foreach (char c in getCategotyInfo.text)
  {
   categoryInfoResult += c;
  }

  // Here it calls the event when JSON content is received fully
```

```
    onCategoryInfoSucceed(categoryInfoResult, "simpleType");
   }
  }

  // This method will be called when as the event and it is
  // responsible for receiving the JSON content and the type
  // of the JSON, calls the decoding and passing the data to
  // the SQLite handler class to be saved into the SQLite database
  private void WebApp_onCategoryInfoSucceed(
        string categoryInfoResult, string type){
   string validationCode = "", categoryName = "",
           categoryTitle = "";

   int categoryId = 0;

   // Making the object of JSONObject class
   JSONObject jsonObject = new JSONObject(categoryInfoResult);

   // Here it checks the type of the JSON content and calls the
   // decode function to decode the JSON
   if (type.Equals("simpleType")){
    Dictionary<string, string> resultDictionary =
                                            decodeJson(jsonObject);

    if (resultDictionary.Count > 0) {
     // Here it checks the validation code of the JSON and reads
     // the data only if it has 85 as the code
     if (resultDictionary["validation_code"] != null) {
      validationCode = resultDictionary["validation_code"];
      if (validationCode.Equals("85")){

       // Here it reads the values and makes the category object
       categoryId = Convert.ToInt16(resultDictionary["category_id"]);
       categoryName = resultDictionary["category_name"];
       categoryTitle = resultDictionary["category_title"];

       Category category = new Category(categoryId, categoryName
                                            , categoryTitle);

       // Here it calls a function of SQLite handler to save the data
       sqliteHandler.saveCategoryToSqlite(category);
      }
     }
    }
   }
  }
}
```

Code snippet 14 - Implementation of the WebApp.cs class with sample methods

Code snippet 15 shows the implementation of the SqliteHandler.cs class which is responsible for handling the data in the SQLite database.

```
class SqliteHandler{
 private SQLiteDB db = null;
 SQLiteQuery qr;

 // Constructor
 public SqliteHandler() { }

 // The name of the SQLite database
 public string filename{get { return "my_demo_sqliteHandler.sqlite"; }}
 // This method is responsible for making connection to the SQLite database,
 // calls the generate method to generate the table if does not exists,
 // delete the data if already exists in the database and inserts the new data
 public void saveCategoryToSqlite(Category category){

  // opening the database
  db = new SQLiteDB();
  db.Open(filename);

  // Calling generate methos to make the table if not existing
  SqliteHandler sqh = new SqliteHandler();
  sqh.createCategoryTable();

  // Deleting the old data if exists
  string queryInsert = "DELETE FROM category WHERE category_name=?;";
  qr = new SQLiteQuery(db, queryInsert);
  qr.Bind(category.CategoryName);
  qr.Step();
  qr.Release();

  // Inserting the new data into the database
  queryInsert = "INSERT INTO category (category_name, "
              +"category_title, last_sync) VALUES(?,?,?);";
  qr = new SQLiteQuery(db, queryInsert);
  qr.Bind(category.CategoryName);
  qr.Bind(category.CategoryTitle);
  qr.Bind(DateTime.Now);
  qr.Step();
  qr.Release();
  // Closing the connection
  db.Close();
 }
}
```

Code snippet 15 - Implementation of SqliteHandler.cs with a sample method

6 TESTING

At the end of the implementation of each part of the application, a testing was performed to make sure that the data are handled correctly and the application provides the correct information in a valid format. All the JSON content that was created by a method was validated and checked at the end of the implementation of the method. Furthermore, all of the forms also were tested individually.

A final testing was done after the implementation of all the parts of the web application to check if all of the parts work together as a system.

6.1 Test cases

There were some error noticed while testing the application and all of them were solved successfully. The errors that were found while testing are explained below.

6.1.1 Page including errors

The view of the web application was designed in a way that all pages was being replaced in a main container of the index page, but the browser was crashing after some clicks. The reason was that the jQuery functions could not handle the events of the included content into the main container of the index page when jQuery files were linked only to the homepage. In addition, the pages could not be loaded when the jQuery file was linked to all JSP pages. Therefore, the error was solved by separating the pages and redirecting to each page with a link. In order to avoid the repetition of the same codes in different pages, the common codes were written in a separate JSP file and included in the pages.

Code snippet 16 shows the structure of a JSP file and how the common contents were included.

```
<body>
 <!-- here it includes the header file on top of the body -->
 <jsp:include page="header.jsp" />
 <div class="bodyContainer">

  <!-- here it includes the admin left sidebar -->
  <jsp:include page="adminLeftSideBar.jsp" />
```

```
    <!-- this is the main container that forms and data are
           being displayes -->
    <div class="mainContainer">

      <!—The cone of each page was inserted here -->

    </div>
  </div>
</body>
```

Code snippet 16 - Structure of a JSP file

6.1.2 Password matching error

When a user wants to change his/her password or administrator wants to add a new user, the password should be repeated and match a pattern. A jQuery function is responsible for checking if the passwords match each other and the pattern.

However, when the user entered a password in the first input, the function checked if the first password matched the pattern. But when the user repeated the password and it matched the first given password, the function printed a message to the user that it matches the previous password without checking if it matches the specified pattern. The error was noticed and solved successfully. Figure 32 shows a screenshot of the error.

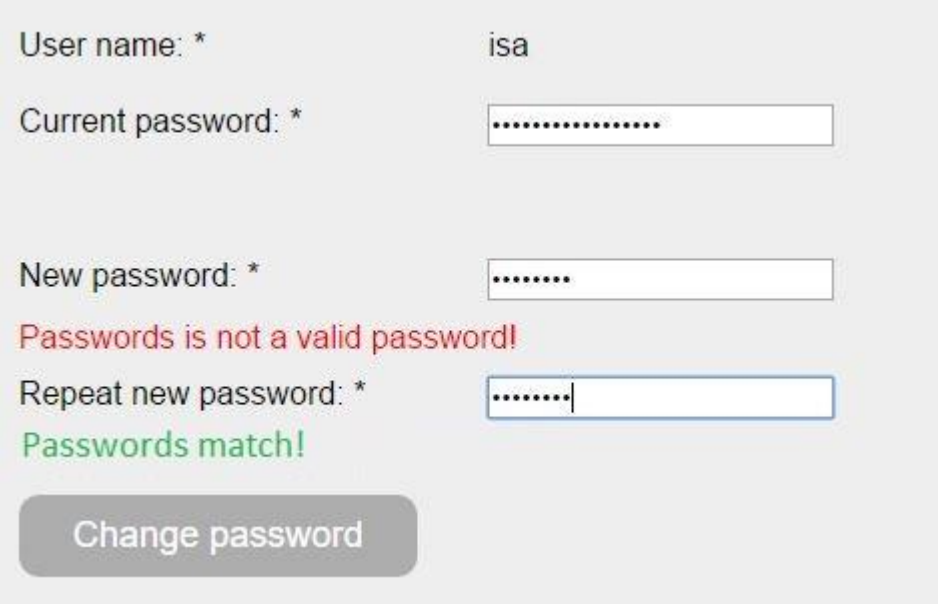

Figure 18 - Password matching error

6.1.3 Data handling error

When an administrator was inserting the data into the database, it was noticed that an important detail about the relays was not taken to account. A device part can be used in different modes of a specific version of relays. For example, a part may be used in the feeder and motor modes, but another one may be used only in the motor mode and this rule was not taken into account while designing the database and application.

Data insertion was stopped and continued after making the modification of the application and the database. Figure 33 shows the form for adding a new device part.

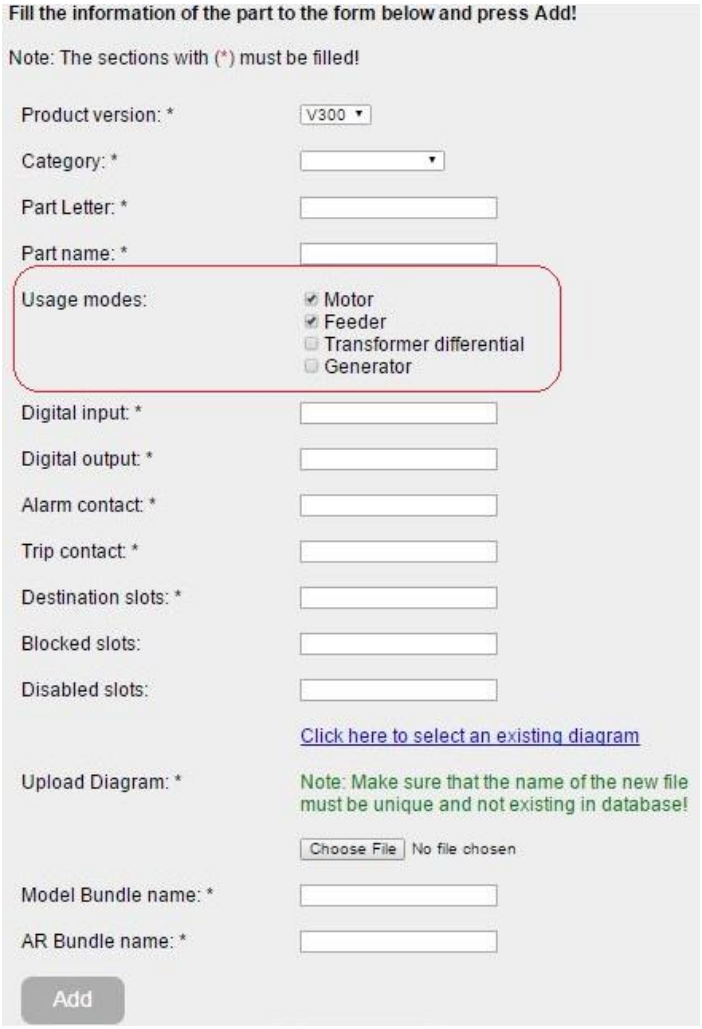

Figure 19 - The form for adding device part

7 SUMMARY

This application is a combination of two existing AR and configuration application with some additional features. Previously all the data were kept in the applications locally, therefore, the size of the applications was large, data were the same for all devices of a specific version and the application had to be modified if any changes were required.

The idea was to minimize the size of the application installed on iPad by keeping the data, 3D models, connection diagrams on a server and importing them during runtime. Another objective was to make data dynamic for devices so that their updated data could be fetched based on their unique serial numbers using the AR interface of the client application.

The application should also function offline and be flexible for future versions and products. In order to achieve all of these goals, a Client-Server application was designed, the data was stored in a MySQL database and 3D models of the relays and parts were stored in a remote storage. A web application was also developed to handle the data transmission between the database and the client application safely, and an administrator user interface for maintaining data in the database was created. In addition, a SQLite database was embedded and a local directory was created to/in the client application for offline usage.

The server side Web application was implemented using Java servlets, JSP pages, jQuery, HTML and CSS, running on Tomcat 8.0. The client side of the application was implemented as an iOS application using C# in Unity 3D engine.

8 CONCLUSIONS

All of the parts and methods of the application are implemented successfully and the goals and objectives are achieved. The web application is implemented fully and tested, then data are inserted into the database. The communication between the web and client applications is made and the client application requests and receives any data from the web application. The web application creates valid JSON content with the correct data and the client application can fetch the data from the JSON content and use or save to SQLite databasse.

The web application provides a user-friendly interface where the administrator of the system can enter the necessary data and save in the database. Both administrator and user can update their profiles and change password.

The jQuery handles the events in the user interface and forces the administrator user to enter correct data.

Previously all of the application made by the company used static information saved in the application itself, but the result of thesis opened a way for the company to develop their application more efficiently and update the data dynamically.

The main challenges during this thesis work were to learn how to develop applications and write code in Unity 3D engine, to get familiar with AR applications, to learn how to develop an AR application and to learn how to send a post request to a web application which is implemented with Java servlets from a client application which is implemented in C# in Unity 3D engine.

9 REFERENCES

1. Schneider Electrics' website – Last access 06.04.2016
   http://www.schneider-electric.com/en/about-us/company-profile.jsp
2. Schneider Electrics' Vamp arc protection – Last access 06.04.2016
   http://www.schneider-electric.com/en/product-range/62049-vamp-arc-protection/
3. What is Augmented Reality – Last access 06.04.2016
   http://whatis.techtarget.com/definition/augmented-reality-AR
4. Vuforia™ Developer Portal
   https://developer.vuforia.com/downloads/sdk
5. Unity 3D
   http://unity3d.com/unity
6. Products of the Schneider Electrics/VAMP - Last access 27.03.2016
   https://m.vamp.fi/products/vampset/
7. Apache Tomcat server - Last access 20.03.2016
   http://tomcat.apache.org/
8. Introducing JSON - Last access 20.03.2016
   http://www.json.org/
9. jQuery Library and documentation - Last access 15.03.2016
   https://jquery.com/
10. SQLite database: - Last access 14.03.2016
    https://www.sqlite.org/
11. AES Encryption - Last access 26.03.2016
    http://aesencryption.net/
12. Vuforia™ Developer Portal - Last access 30.03.2016
    https://developer.vuforia.com/
13. Learning Unity 3D - Last access 18.03.2016
    http://unity3d.com/learn
14. WWW class in Unity - Last access 26.03.2016
    http://docs.unity3d.com/ScriptReference/WWW.html
15. WWWForm class in unity – Last access 06.04.2016
    http://docs.unity3d.com/ScriptReference/WWWForm.html

16. MySQL Documentation - Last access 14.03.2016
https://dev.mysql.com/doc/
17. JSON content validator - Last access 27.03.2016
http://jsonlint.com/
18. JSON content viewer - Last access 27.03.2016
http://jsonviewer.stack.hu/