Room Monitoring System

**Students**

Alexandru Dima Mircea: 266006

Alexandru Vieru: 267013

Rares Dumitru Bunea: 266983

Liviu Lesan: 241737

Ionut Boitan: 266869

Alexandru Ciornea: 266875

Alexandru Mihai Serb: 266913

Raul Pologea: 266240

**Supervisors**

Ib Havn

Lars Bech Sørensen

Erland Ketil Larsen

Knud Erik Rasmussen

Kasper Knop Rasmussen

**ICT Engineering, VIA University College, Horsens**

**4th Semester**

**Abstract**

***The purpose of this project is to create a room monitoring system which will monitor and retrieve sensor data of CO2 emissions, humidity and temperature from an IOT device and send them to a data storing unit. The measurements will be retrieved using an Android application and will make the data available for a given user.***

***This system will increase efficiency in managing monitoring tasks for the given location and will allow users to retrieve data in a fast and reliable manner.***

***The benefit of using this system is that the information will be stored on a database which provides scalability and a more fast and efficient manner.***

***Using this system technical staff will be able to view different data from different units at the same time and will be displayed to them in a fast way by just a click of a button.***

Table of Contents

[1 Introduction 4](#_Toc8801445)

[2 User stories and requirements 5](#_Toc8801446)

[2.1 User stories 5](#_Toc8801447)

[2.2 Functional requirements 5](#_Toc8801448)

[2.3 Nonfunctional requirements 6](#_Toc8801449)

[3 Analysis 7](#_Toc8801450)

[3.1 Use case diagram 7](#_Toc8801451)

[3.3 Activity diagram 9](#_Toc8801452)

[3.4 Domain model diagram 14](#_Toc8801453)

[3.5 System sequence diagram 15](#_Toc8801454)

[4 Design 17](#_Toc8801455)

[4.1 IOT Design 17](#_Toc8801456)

[4.1.2 Class diagram IOT 18](#_Toc8801457)

[4.1.3 Sequence diagram IOT 19](#_Toc8801458)

[4.2 Database design 19](#_Toc8801459)

[4.3 Android design (Alexandru Vieru and Rares Dumitru Bunea) 20](#_Toc8801460)

[4.3.1 Conceptual diagram Android 20](#_Toc8801461)

[4.3.2 Class diagram Android 20](#_Toc8801462)

[4.3.3 Sequence diagram Android 21](#_Toc8801463)

[5 Implementation 23](#_Toc8801464)

[5.1 IOT Implementation 23](#_Toc8801465)

[5.2 Database Implementation 24](#_Toc8801466)

[5.3 Android Implementation 24](#_Toc8801467)

[5.3.1 Firebase implementation 24](#_Toc8801468)

[5.3.2 UI implementation 27](#_Toc8801469)

[5.3.3 Android application architecture 28](#_Toc8801470)

[6 Testing 29](#_Toc8801471)

[6.1 Android Testing 29](#_Toc8801472)

[7 Conclusions 35](#_Toc8801473)

[8 References 35](#_Toc8801474)

[9 Appendices 35](#_Toc8801475)

# 1 Introduction

One of the greatest focus of the modern society is to educate people that will be able to improve it. Studying or teaching are difficult activities that require a good ambiance, so that the evolution of an individual will be noticeable.

Many students work hard to acquire good study skills, but not many realize that having the right place to study is just as important. The study environment can be a big factor in how successfully some people learn and retain information and be able to apply it in their assessments and on the job. For a short time, you may be able to stay focused in hot or humid places, but after a while, these circumstances can become unbearable. Similarly, if you’re too cold, that quickly becomes all you can think about, and studying suffers.

A human being that will pursue his duties in a balanced environment will show higher productivity.

Living in a time when technology is at its peak, most people have access to a mobile android device.

Having a system that will help monitor a certain room or group of rooms on their CO2, temperature and humidity parameters and eventually alert the users about surrounding conditions out of normal, is a good start in helping the community progress.

# 2 User stories and requirements

## 2.1 User stories

1. As a technician I want to be able to collect information from all the sensors (air temperature, CO2 and humidity) so that I can oversee the environment inside a room using a phone.
2. As a technician I want to see the minimum and maximum standard values for optimal living/working/studying conditions so that I could be able to take certain actions before the conditions break the optimal ranges.
3. As a technician I want to get push notifications assessed before and when the sensor’s readings reach unfavorable parameters so that I can prevent or fix different problems.
4. As a technician I want to be able to have the data stored in a database so that I can access both current and past information about the environment state.
5. As a technician I want to be able to control the motor that controls the window so that I can open or close a window as much as it is required using a phone.
6. As an administrator I want to be able to add, edit and remove users.

## 2.2 Functional requirements

Cross Media Requirements

1. The application must retrieve and parse relevant data from a webservice
2. The application must be able to send data to a webservice to interact with actuators
3. The application must have a responsive user interface
4. The application should utilize authentication to sign in
5. The application should have a settings menu
6. The application should persist some data locally on the device
7. The application must be under version control for the entire development process
8. The application must be developed using the official Android framework
9. The application must be developed with Java
10. The application should follow the Google Material Design guidelines
11. The source code should be structured using an architectural pattern

Embedded Requirements

1. The system should measure the level of CO2.
2. The system should measure the ambient temperature.
3. The system should measure the humidity in the air.
4. The system should send gathered data to MongoDB.

Data Engineering Requirements

1. Users must be able to collect information from all the sensors (air temperature, CO2 and humidity)
2. Users could be able to see the maximum and minimum standard parameters
3. Users should be able to get push notifications
4. Data must be stored in a database
5. Users should be able to control the actuator
6. Administrators must be able to add, edit and remove users

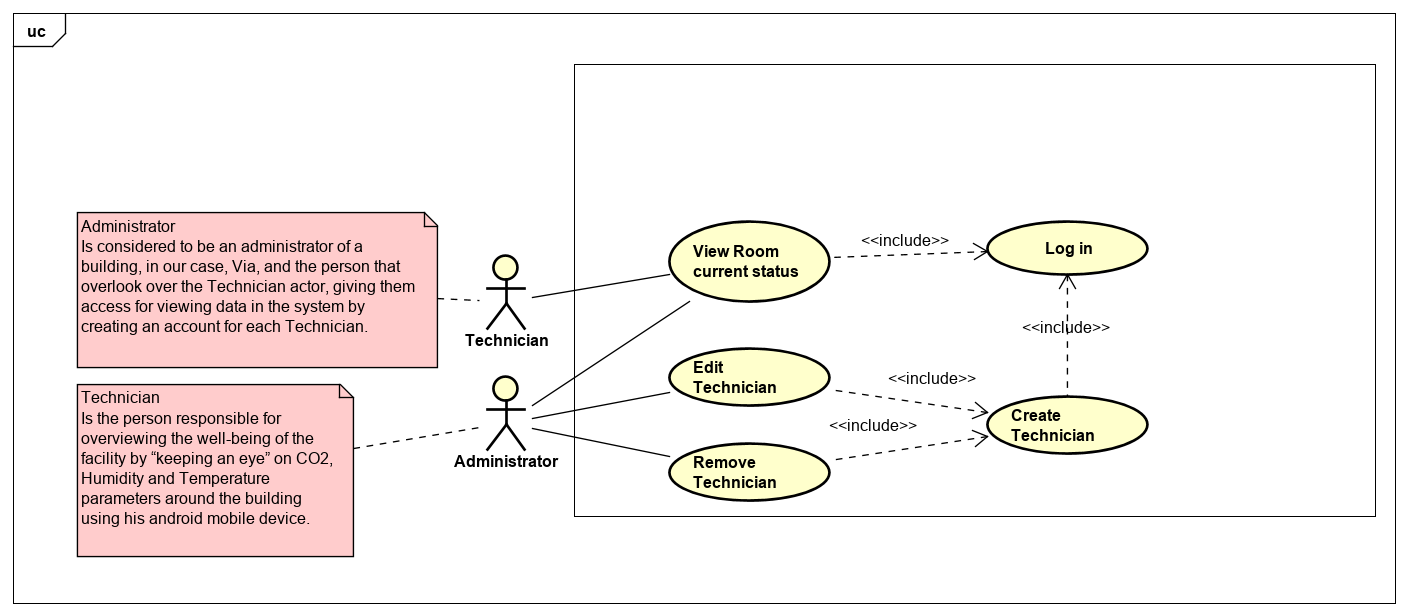
## 2.3 Nonfunctional requirements

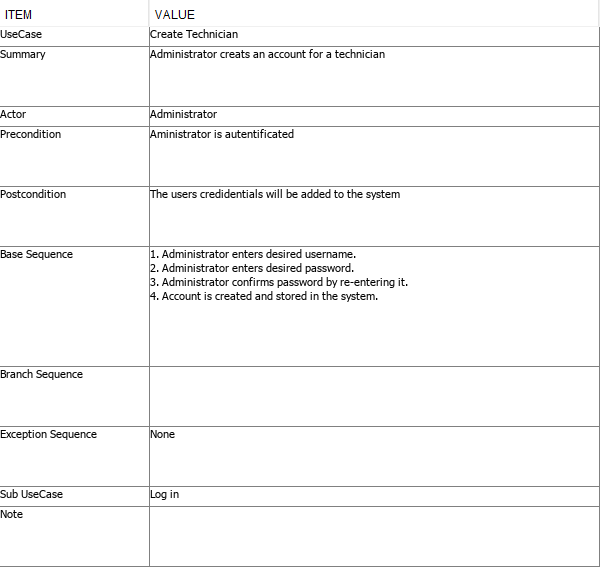
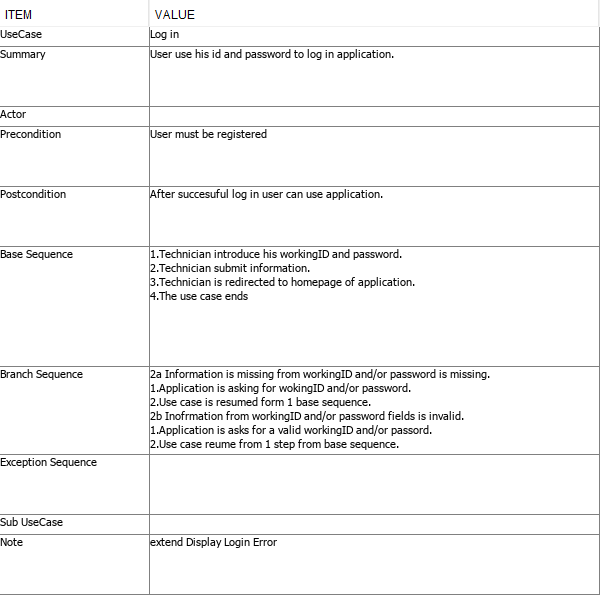
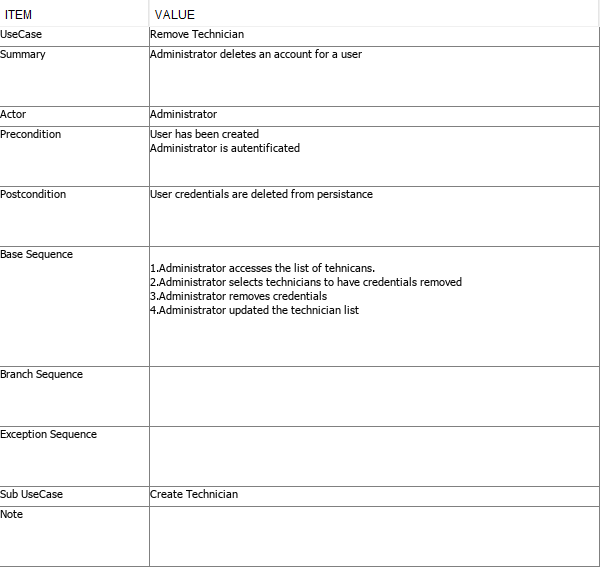
1. The system must use at least five tasks.
2. Some data must be used by more than one task.
3. The system must use semaphores, mutex and queues.
4. Part of the system must be tested by using unit test.
5. The system should use LoRaWAN and Bridge application to transfer the data to database from device.
6. SCRUM & AUP must be use for the development process
7. The project and process report must include authors for each section
8. Links must be handed in to the source code on GitHub
9. A link to a video demonstration on YouTube must be presented

# 3 Analysis

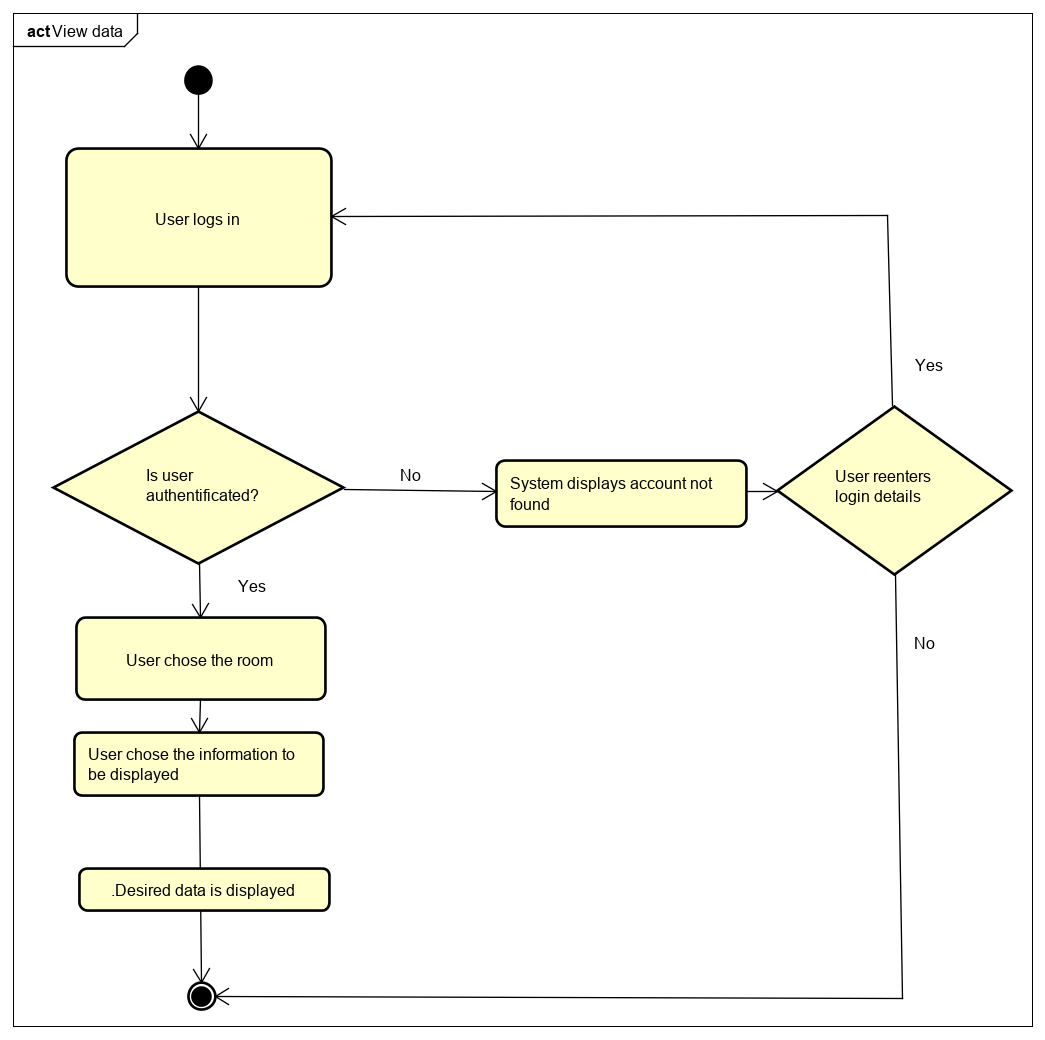
## 3.1 Use case diagram

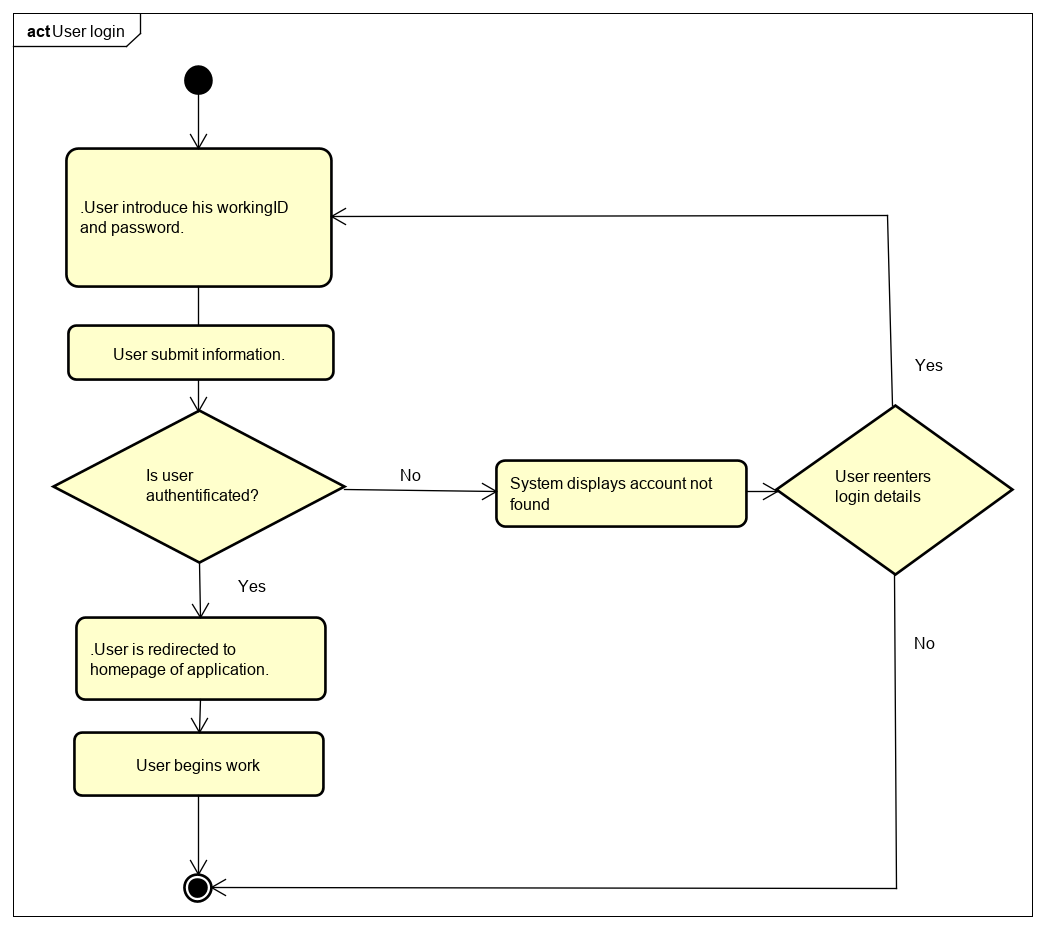
The system has two types of actors. One is at a basic usage level-the technician- and the other the Administrator having an extra capabilities such as user administration, as it is shown in the diagram bellow.

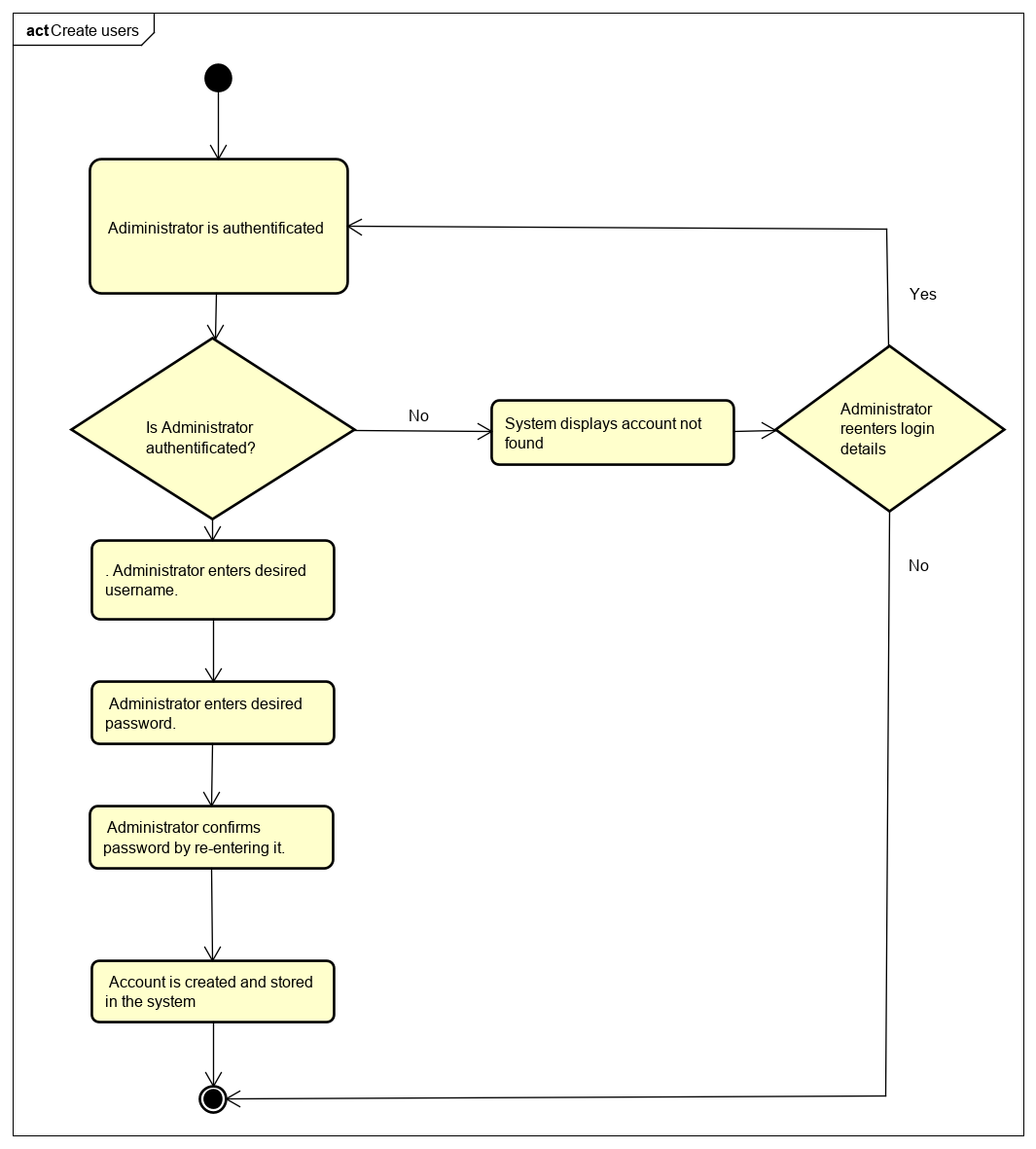
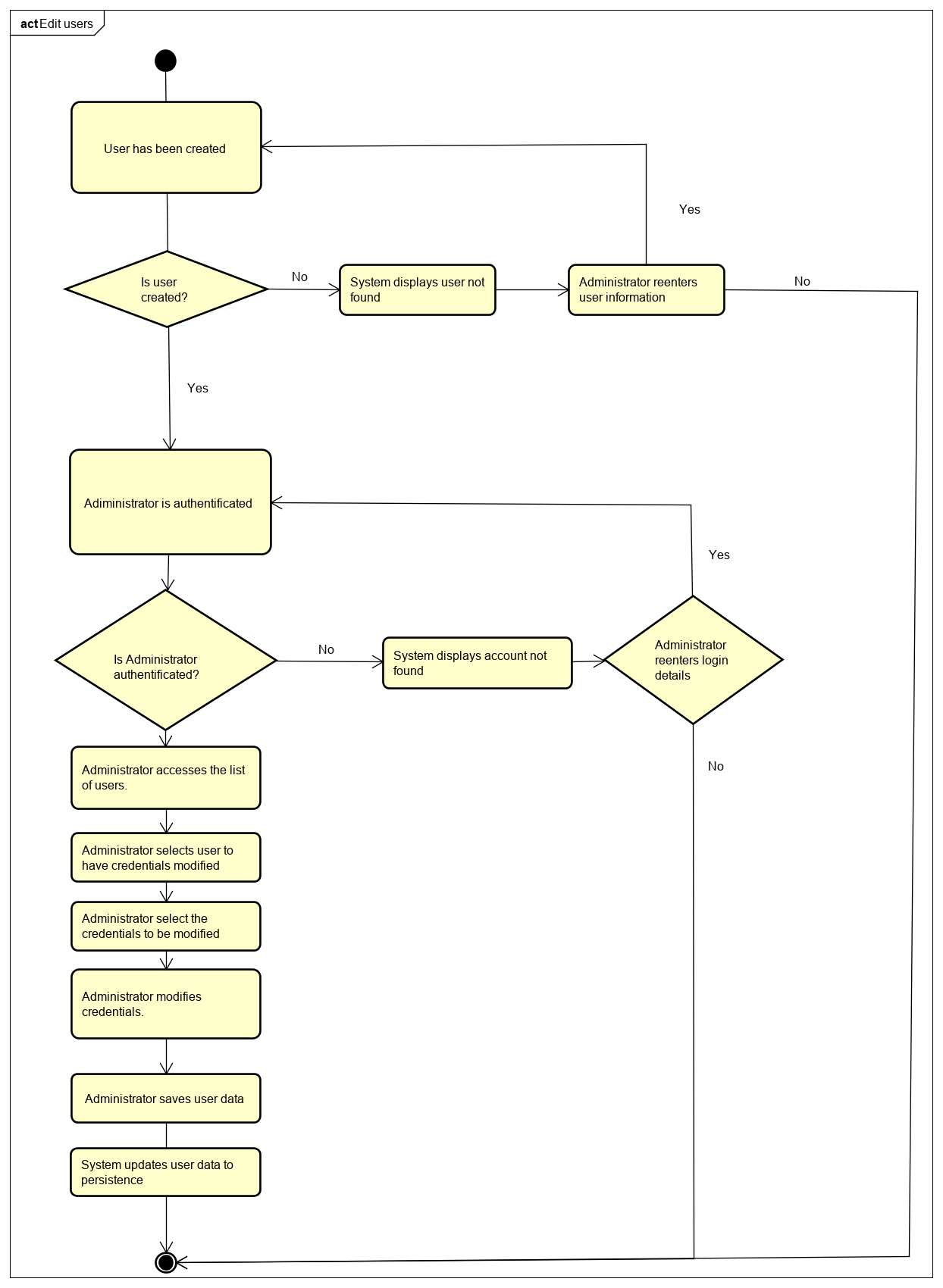
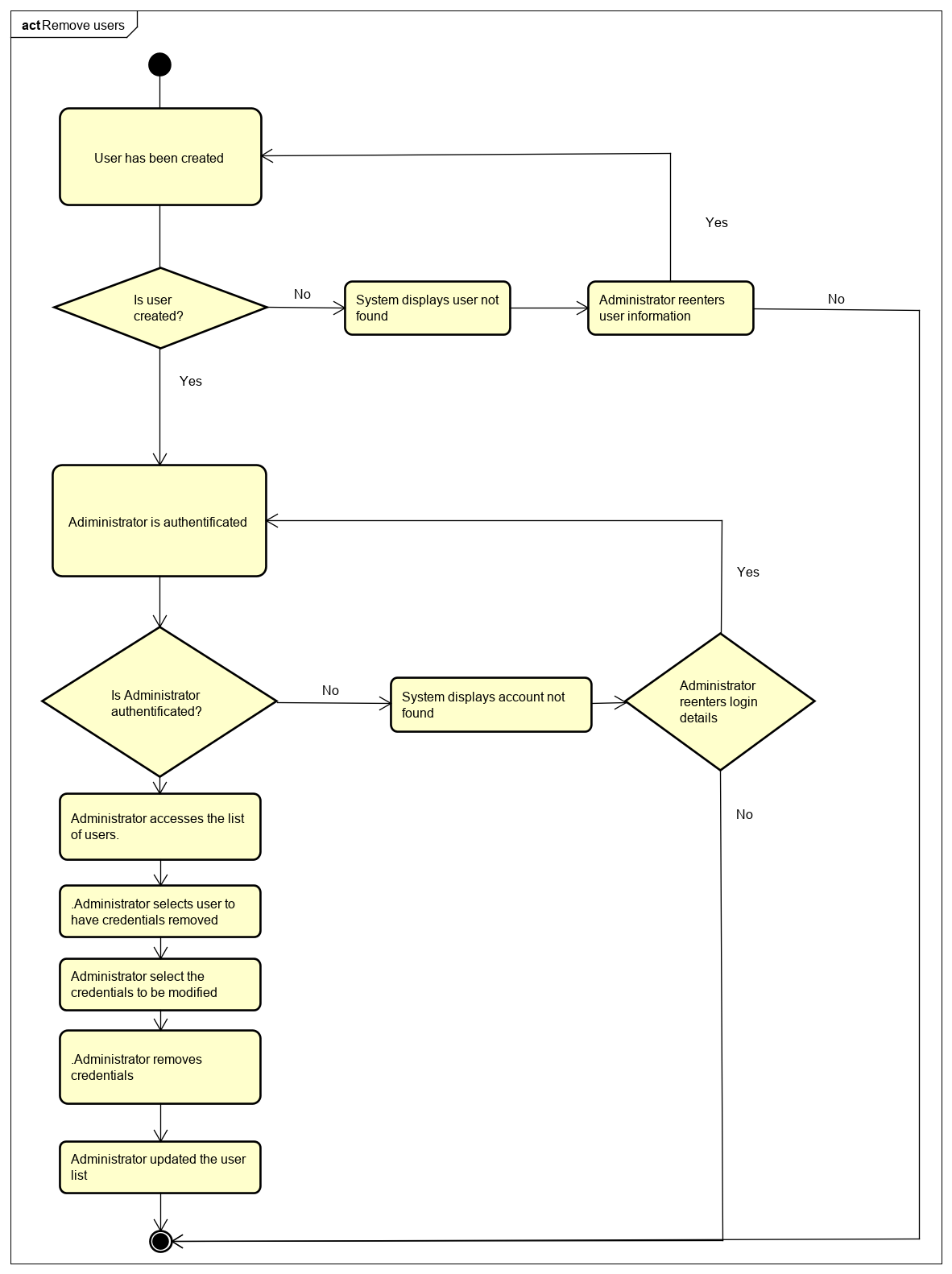


3.2 Use case description

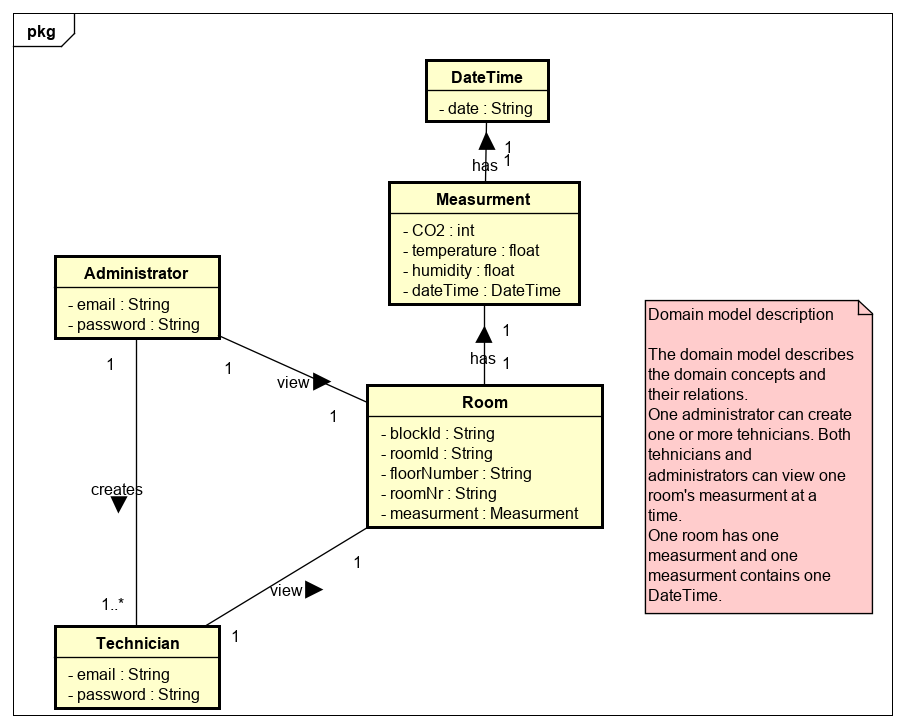
## 3.3 Activity diagram



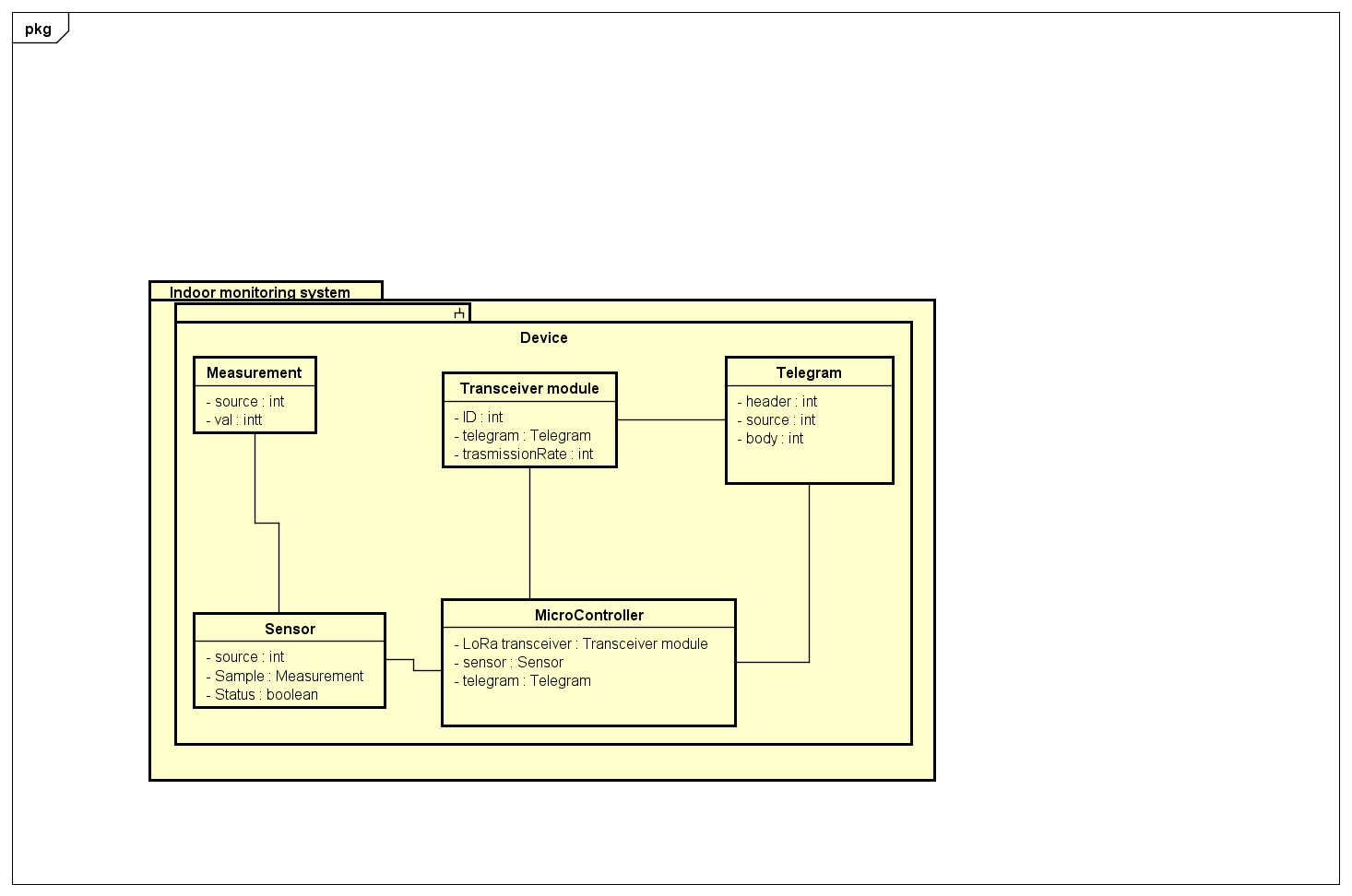




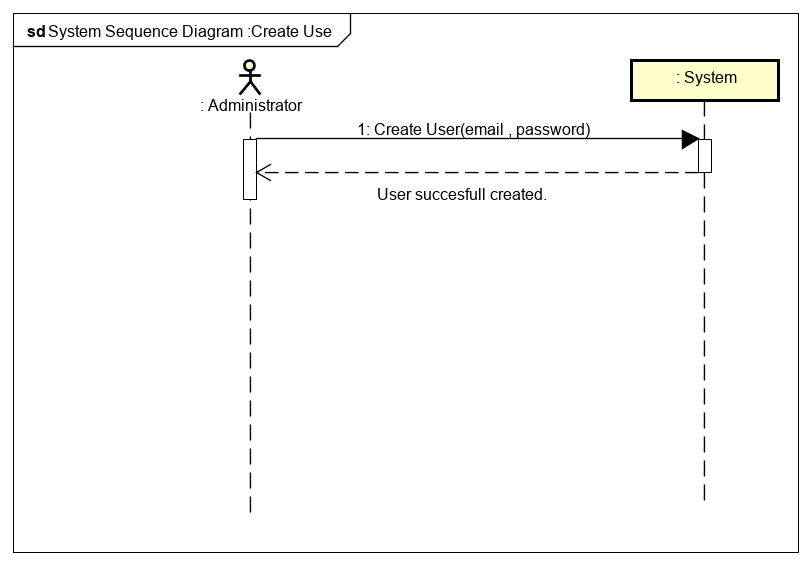
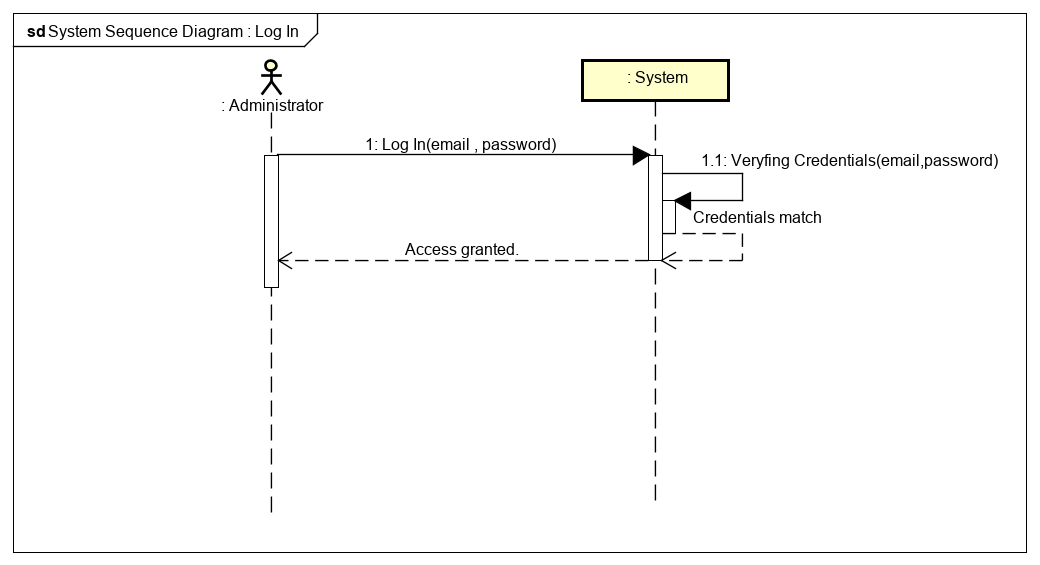
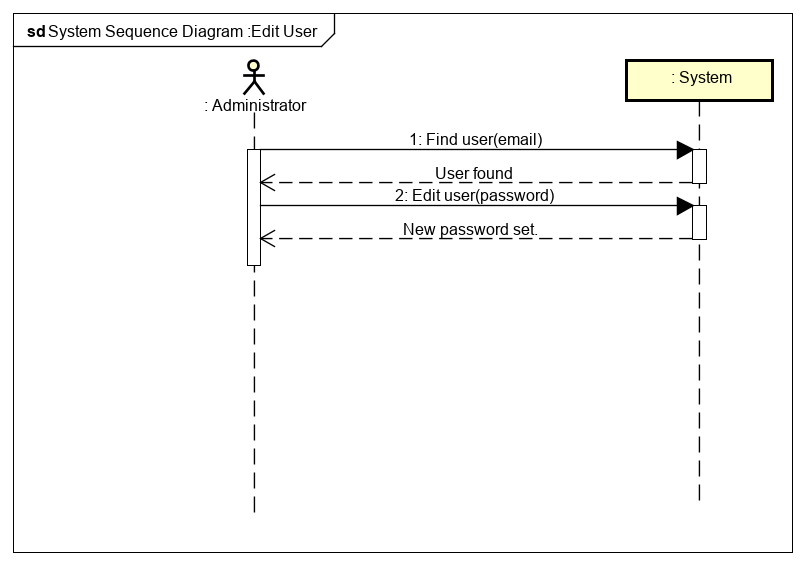
## 3.4 Domain model diagram

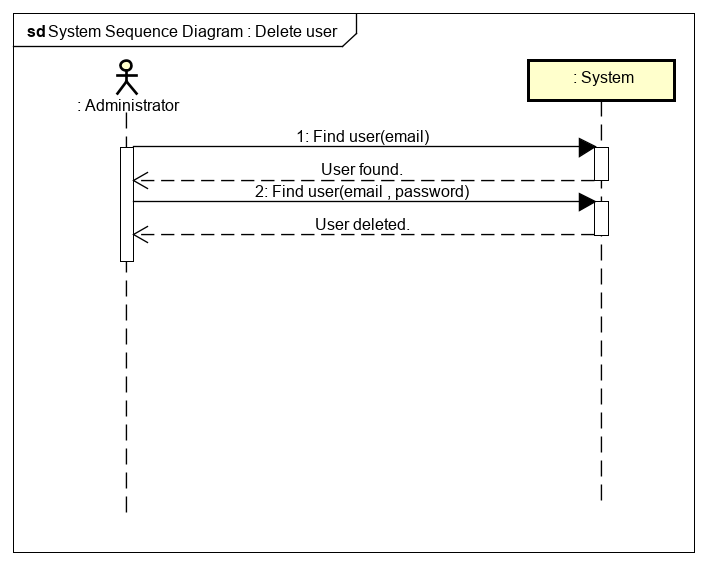
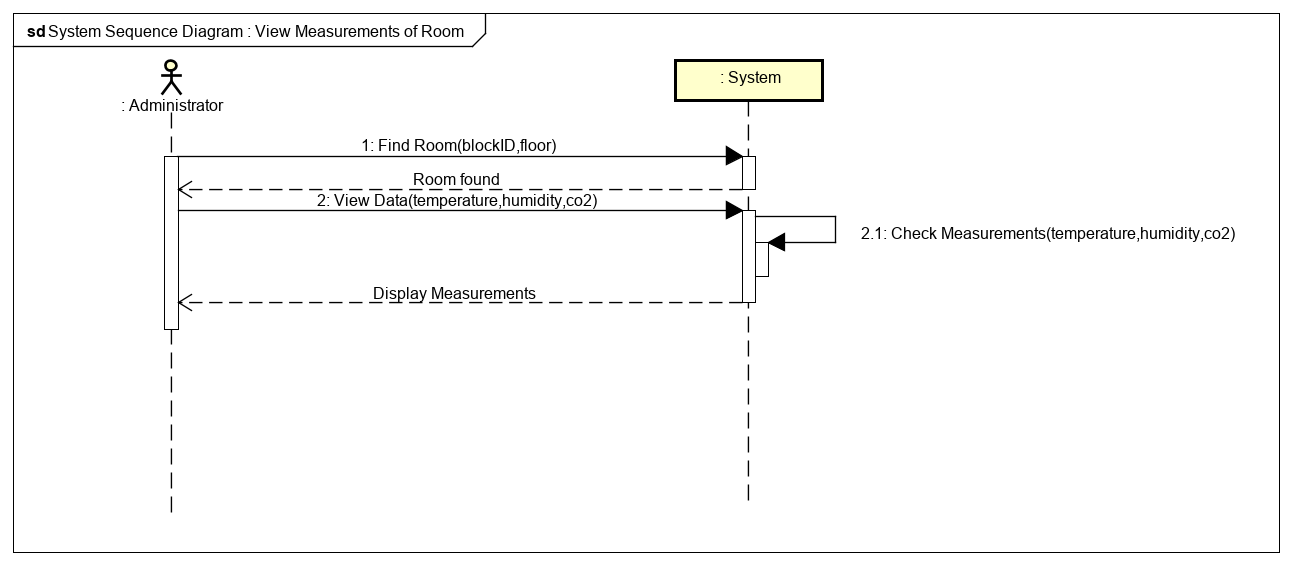


From the viewpoint of embedded engineering, the system can be represented as in the diagram bellow. The center entity in this block is the MCU which is connected to sensors from which measurements are retrieved, processed and stored in a telegram to be delivered by the transceiver module on a network. From further analysis, is concluded that another subsystem is required in order to facilitate the data transfer to a permanent storage so that it can reach a final user.



## 3.5 System sequence diagram



## 4 Design

## IOT Design

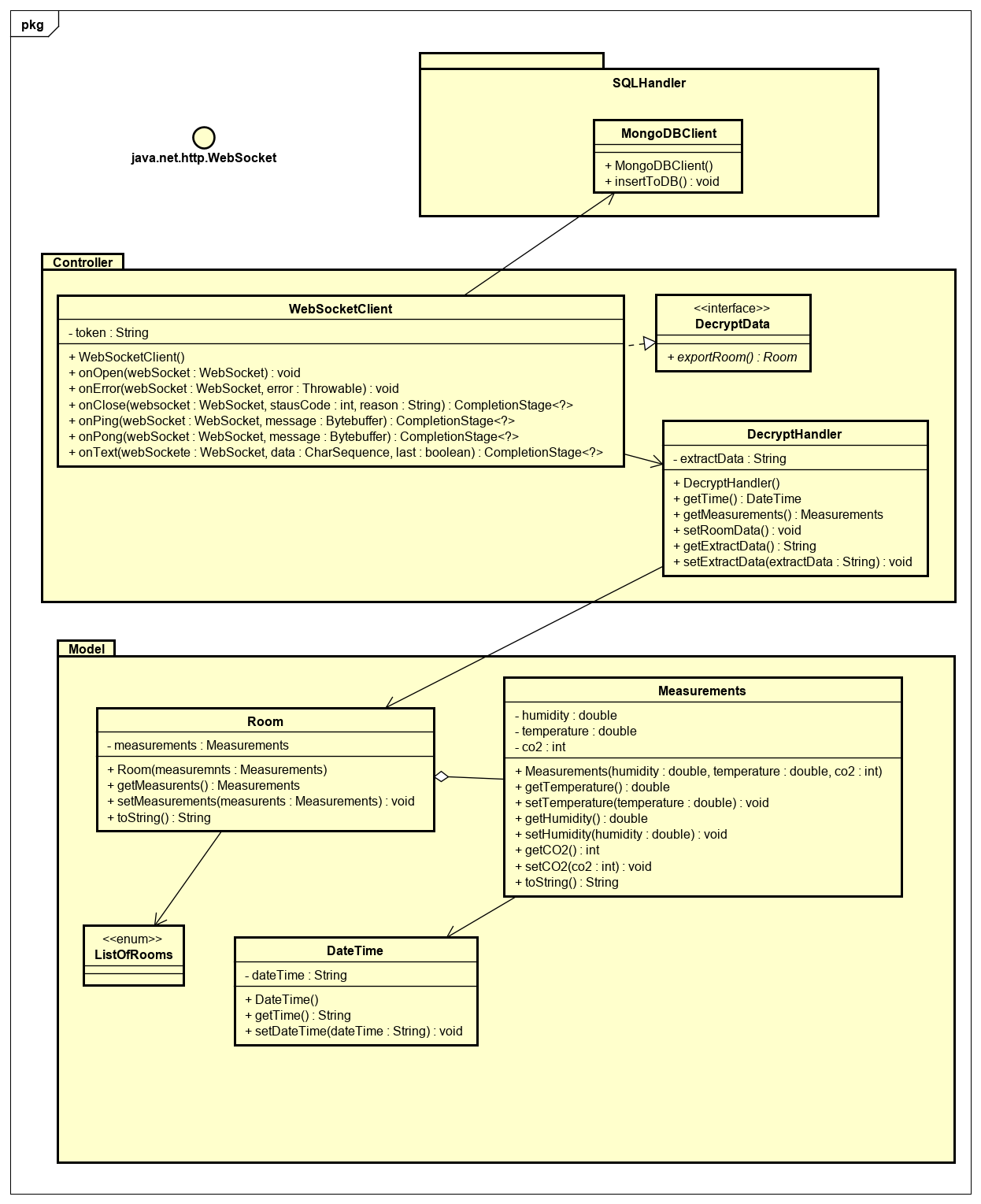
This subsystem is divided in two modules responsible for the transfer of data to and from the LoRaWAN network. First module, represented in the above diagram by the IMS1 package, is responsible to acquire data from the three sensors mounted on an Arduino shield and transmit it to the LoRaWAN network making use of an LoRa module mounted on the same shield. The second module, represented in the above diagram by the Java Bridge Application package, has the responsibility to retrieve the data from Loriot.io server, decoded in a readable format and send it to a MongoDB sever.

The module responsible for data retrieval and sending to LoRa network is designed so that the samples from the sensors are stored in a data structure that is protected from corruption while writing the values in a queue. A telegram to be sent is constructed after all sensors have been queried for data and is ready to be delivered on the server. Each measurement is handled by different task. The tasks are running based on a cycle controlled by a timer so that the frequency of this operations is matching the requirements of the LoRa network. Each operation specific to a data source is contained in its own submodule so that encapsulation is achieved. The communication between the submodules is realized using interfaces.

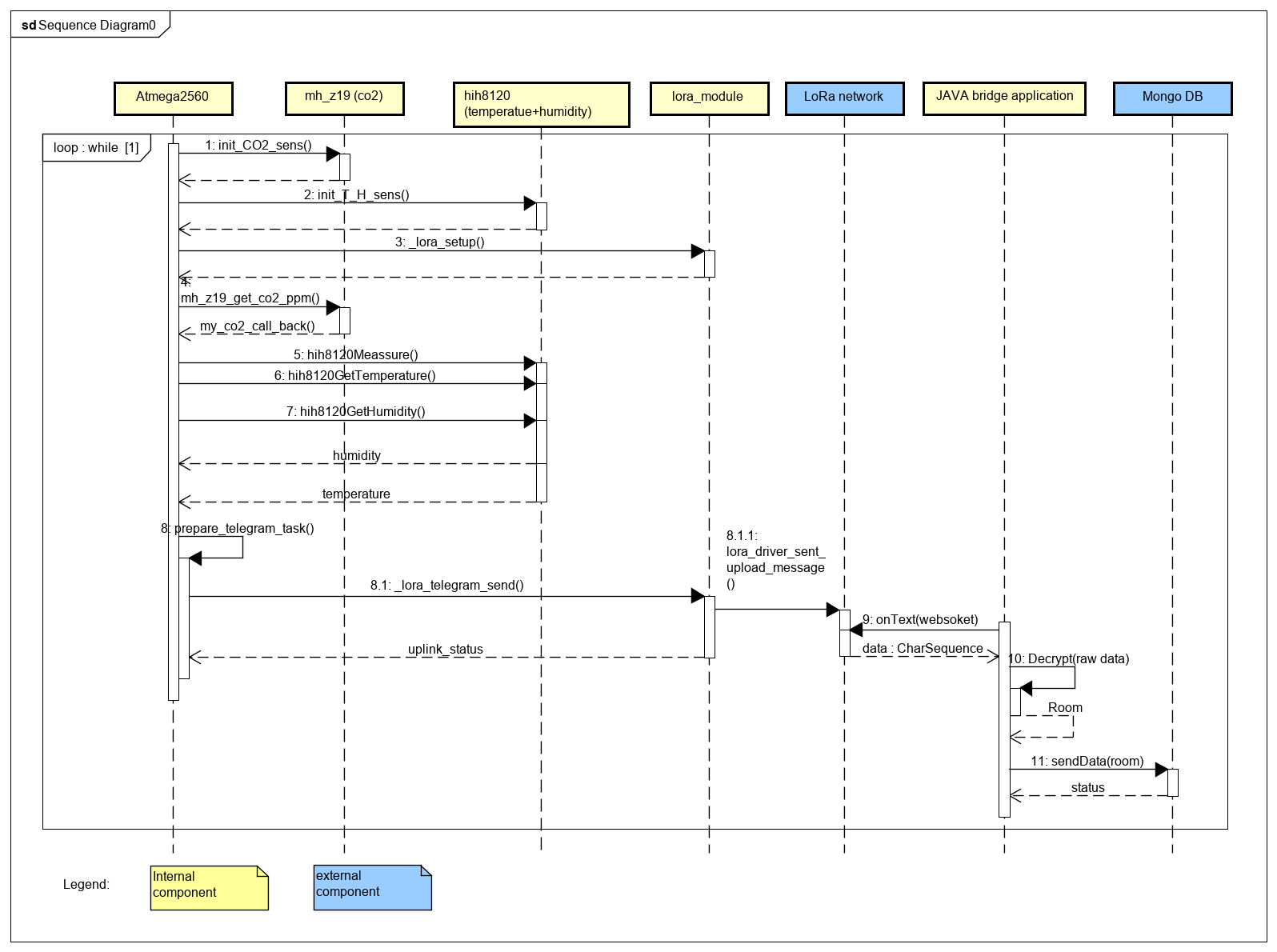
Java Bridge Application should be thought as a significant part of the project in order to visualize data obtained from sensors on Atmega2560, reflecting on how you get desired data and serialization of it in a suitable object following requirements, afterwards to be sent furthermore, ended up with this concept. WebsocketClient class which has responsibility as a listener should connect to LoRaWan socket for retrieving data. DecryptionHandler takes role of decrypting data and make a new model object of Room following Room object requirements (Room has measurements where it has date when it was retrieved). Responsibility of MongoDBClient class isto make connection with mongoDB and to upload retrieved data formatted in suitable Object in order be sent toward database.

### 4.1.1 Class diagram IOT

Java Bridge Application

Conceptual design had a starting point in generating the actual design. Following basic principles of designing, every class should have one responsibility. WebSocketClient implements interface (ExtractData) as responsibility has to retrieve and return it as a suitable object from Loriot where it uses DecryptionHandler tools of transforming obtained data in a Room model, DecrtypionHandler has a role of a middleman in this case. MongoDBClient responsibility is to use implemented method of WebSocketClient(extractData) to parse it further to database.

### 4.1.2 Sequence diagram IOT



## Database design

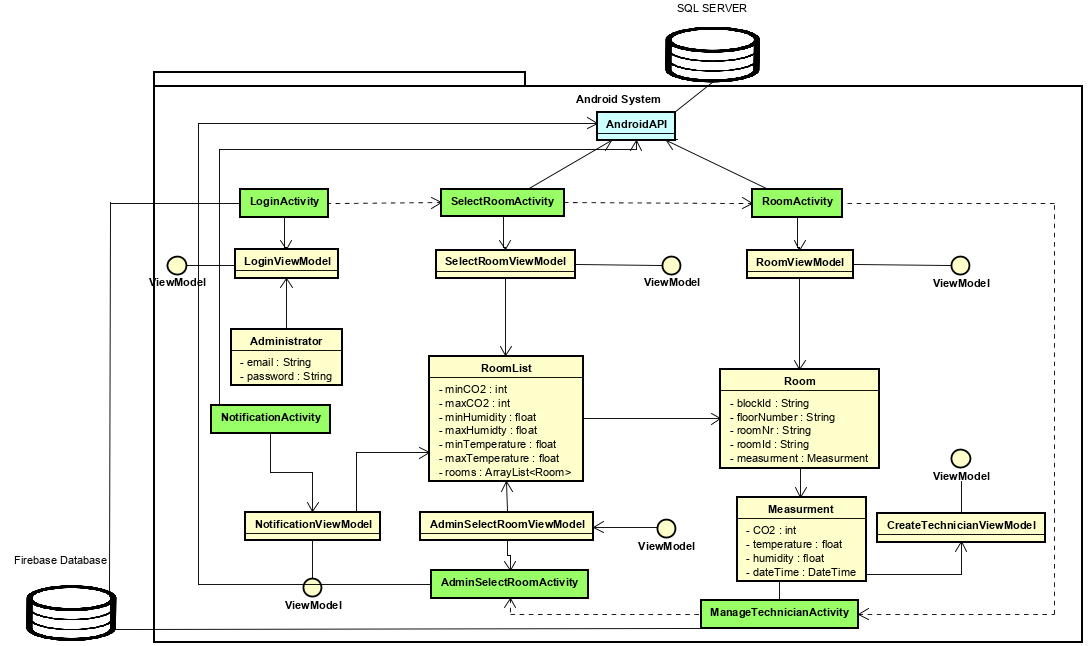
## 4.3 Android design (A. V. and R. D. B.)

In this chapter the android design part of this report will be explained. This stage of the report is crucial since it shows how the actual Android system was conceived. In this part of the report design patterns will be discussed, a conceptual diagram will be illustrated and explained, same with the class diagram and finally will be ended with the presentation and explanation of a sequence diagram.

### 4.3.1 Conceptual diagram Android (A. V. and R. D. B.)

This subchapter will present the conceptual diagram for the Android app. This part is essential since it is the starting point for the developed code. It shows the way the application has been structured for it to function more efficiently and to be able to be further developed in the future.

***Figure 4.3.1 Conceptual diagram***

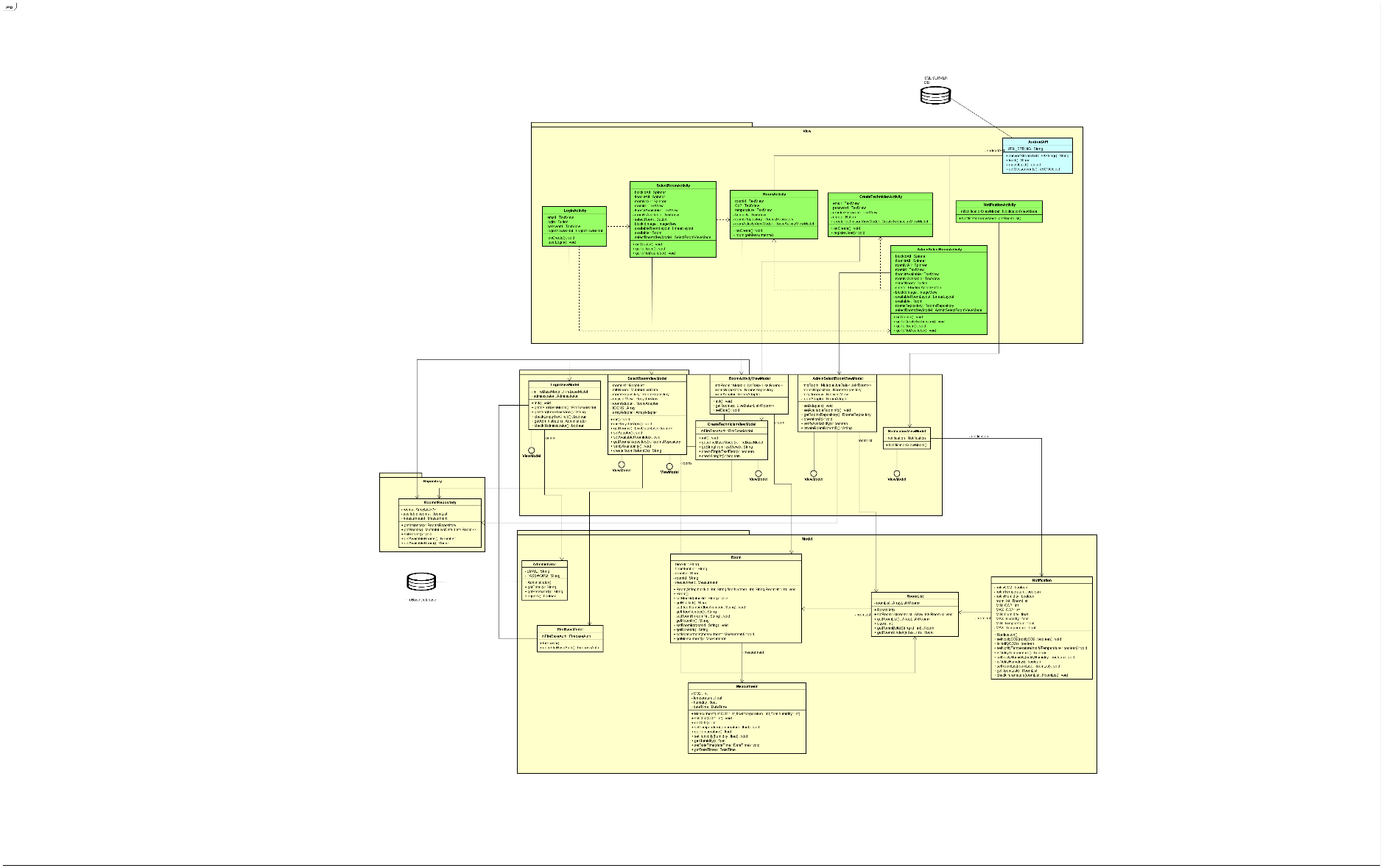


As seen in the above figure, a view model class such as “**RoomViewModel**” will take all the information from the model class “**Room**” and adapt it such that it will become fit to be displayed in the view class “**Room Activity**”. The same can be said with respect about the rest of the entities since the all follow a similar pattern.

### 4.3.2 Class diagram Android (A. V. and R. D. B.)

This subchapter will focus primarily on illustrating and explain the class diagram that was made for the Android app. The diagram is the main blueprint for the code that has been developed. It showcases class interactions and implementation of methods which create functionality for the written code.

***Figure 4.3.2 Class diagram Android***



The figure above shows the class diagram created for this assignment. The design consists in separating the code into 4 packages. The model package which contains all the class models such as the Administrator class, Firebase class, Room object and so on.

One of the most important classes in this package is the Administrator class which is responsible for creating user object that will serve as user accounts.

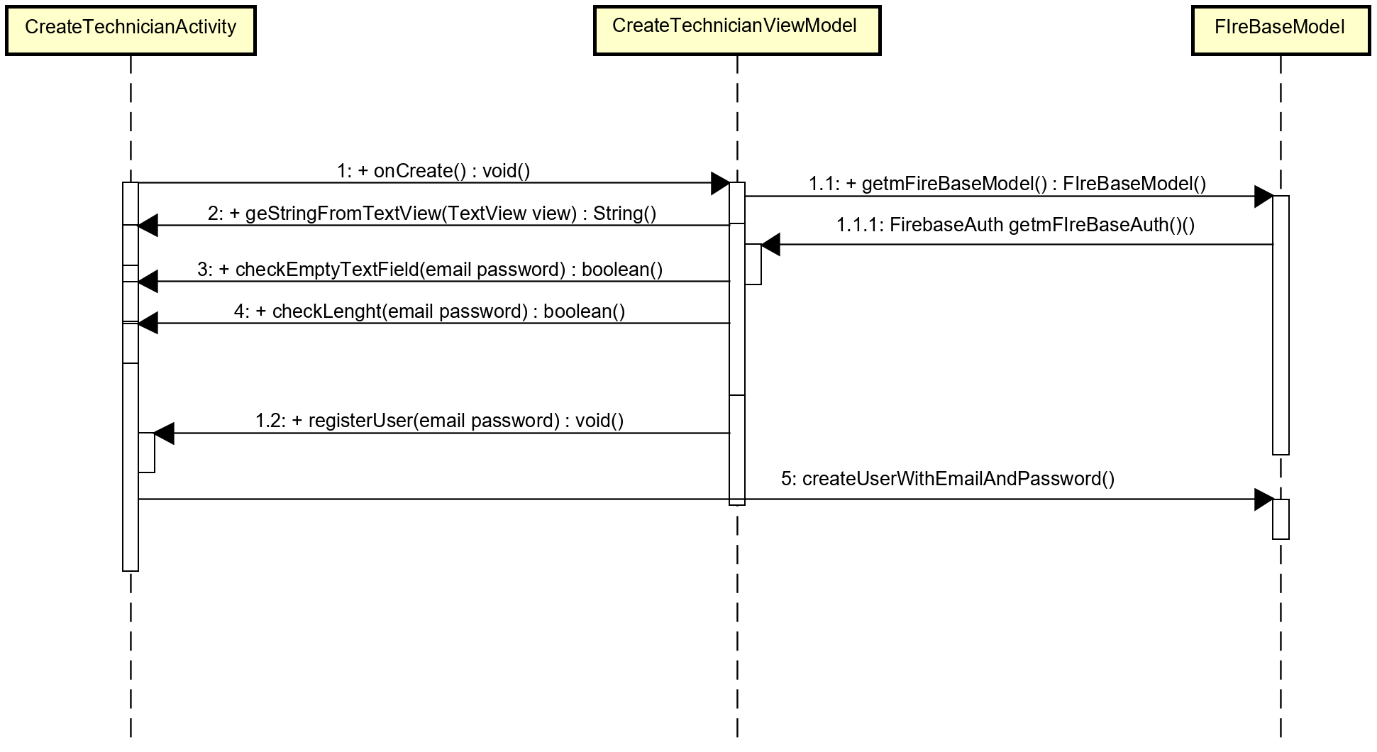
In the **ViewModel** package, the **CreateTechnicianViewModel** class is responsible for instantiating the Firebase class as an object since it is needed to authentificate when registering user accounts since it contains methods such as **checkEmptyTextField**().

Finally, the last package it needs to access is the **View** package where the **CreateTechnicianActivity** will instantiated the **createTechnicianViewModel** as an object. By doing so, the methods it contains can be used by simply calling the **“object.method”.** In here an **onCreate()** methodwill load all the layouts, items and initialize the needed attributes. And finally, the **registerUser()** will be set as an **onclickListner** for the register button and if all conditions are met it will register a user account in the **FireBase**.

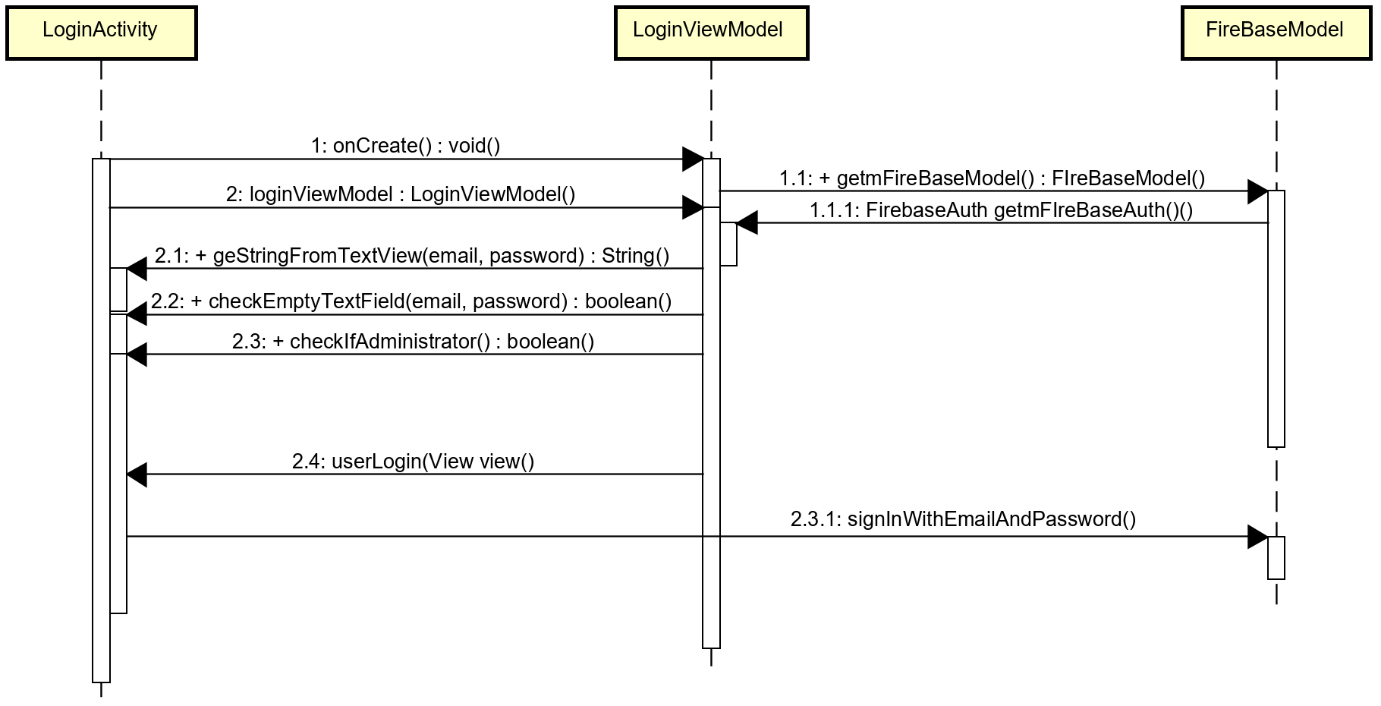
### 4.3.3 Sequence diagram Android (A. V. and R. D. B.)

The sequence diagrams for the given app will be showed and explained. They show the interaction between classes and give a clear view what Life Cycles exist in the code and what methods are invoked to perform a given task successfully. Only one diagram will be explained and that will be the login sequence diagram.

***Figure 4.3.3.1 Create technician account sequence diagram***



***Figure 4.3.3.2 Login user sequence diagram***

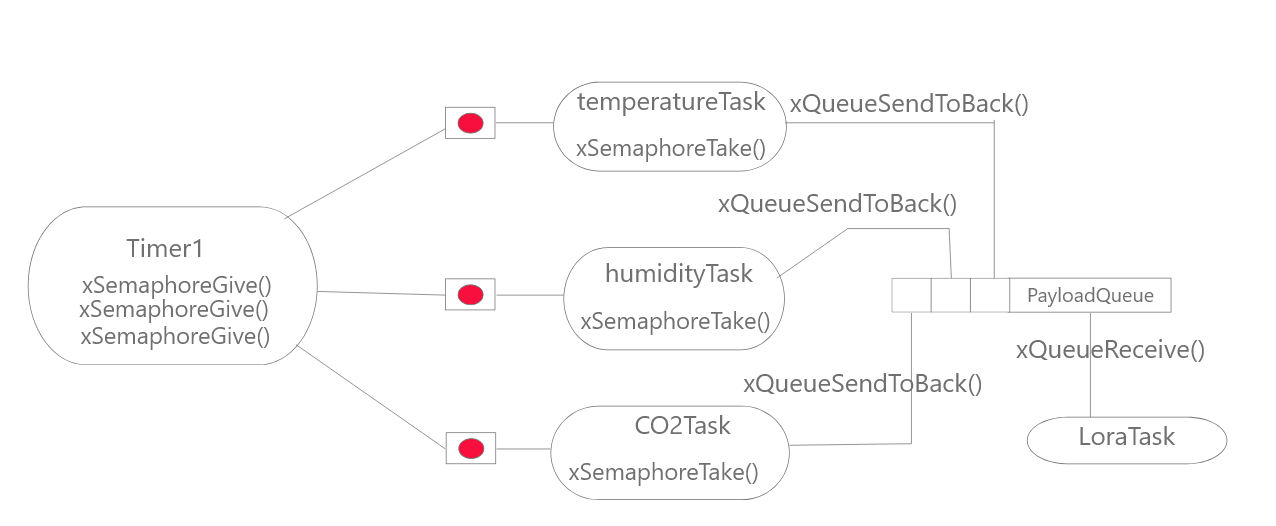


The cycle starts from the LoginActivity which resides in the View. This is when the user interaction starts. Once the user enters the necessary input, this will send an onCreate request to the ViewModel which will in term request to get a Firebase object from the FireBase model class. This class contains a FireBaseAuth method which is responsible for authentication. The model will send this object back to the ViewModel. This will invoke methods that will get the imputed values and check to see if they fit the established conditions. It will execute a method called signInWithEmailAndPassword which will perform the user request of login.

# 5 Implementation

## 5.1 IOT Implementation

Considering the design approach described above, the following diagram describes the process flow in the subsystem following well known Producer-Consumer design pattern.



The subsystem is hosted on an ATmega2560 processor, that runs FreeRTOS operating system. Taking advantage of software timers already existing in the libraries, the operating cycle designed in the previous stage is achieved.

First step is initialization of the drivers controlling the sensors and the Lora module.

The timer is responsible to release three semaphores, one for each sampling task and after expiration is auto reloading with a period enough for data sampling and telegram construction and forwarding while making sure that the network is not overloaded.

The tasks responsible for data acquisition are implemented with the same priority(3) and blocking time, while delivering data to a shared data structure queue as it can be seen in the code snipped bellow. Taking advantage of the queue capabilities, easy synchronization is established.

xTaskCreate**(**temp\_tsk**,**"Task1"**,**configMINIMAL\_STACK\_SIZE**,NULL,**3**,NULL);**

xTaskCreate**(**co2\_tsk**,**"Task2"**,**configMINIMAL\_STACK\_SIZE**,NULL,**3**,NULL);**

xTaskCreate**(**hum\_tsk**,**"Task3"**,**configMINIMAL\_STACK\_SIZE**,NULL,**3**,NULL);**

The measurements are stored in a data structure incapsulating the value as well as the source it comes from, allowing the telegram construction based not only on the sensor’s value, but the source also, as it can be seen in the following snippet.

values**[**recValue**.**s\_src**]=**recValue**.**s\_value**;**

test\_payload**.**bytes**[**2**\***recValue**.**s\_src**]** **=** values**[**recValue**.**s\_src**]** **>>** 8**;**

test\_payload**.**bytes**[**2**\***recValue**.**s\_src**+**1**]** **=** values**[**recValue**.**s\_src**]** **&** 0xFF**;**

A receiving task is implemented, also having same priority as the sampling tasks, so that each time a measurement is stored in the queue, this event notifies the task allowing it to perform. This task has the responsibility of constructing the telegram and forwarding it to the Loriot server after establishing a successful connection.

Running the tasks in parallel results in energy saving.

## 5.2 Database Implementation

## 5.3 Android Implementation (A. V. and R. D. B.)

In this chapter of the report the implementation for the Android app will be explained and examples will be illustrated. This chapter will start with presenting the Firebase database since this was a very useful tool for creating and authenticating users that will use the given software.

### 5.3.1 Firebase implementation (A. V. and R. D. B.)

When talking about implementing a login, creating your own authentication system can sometimes lack security and also require a unnecessary amount of time.

Firebase provides a bunch of built-in services that help and improves the development of web and mobile applications. Among the service offered by Firebase we have Authentication. Firebase Authentication offers an easy and straightforward way to implement a system that offers real-time updates and secure encryption for user data, that separates user’s sensitive data from application data. From the multiple ways of implementing the login, such as getting accounts from certain social media platforms, the classical combination of an email and a password is maybe the standard way to-go and the choice to use in this case. The setup is done by adding an application to an existing google account.

The main activity opens the app in the login screen where existing users can log.

The logic behind the login activity and its connection to Firebase can be observed in the figure below.

***Figure 5.3.1.1 Firebase example LoginActivity***



The first step is to set the layout for this activity followed by the setting of the button and the input fields. The method for setting the button is created and attached to the specific button through the

XML file. When the login button is pressed, the data submitted in the specific from the input fields is compared to the one in the Firebase Authentication. If the input matches with an account from the database, depending on the account type the user is directed to the appropriate activity. Otherwise an error message will be displayed. Furthermore, Firebase offers functionality for registering new account, which in this case refers to to higher ranked user creating a new account.

***Figure 5.3.1.2 Firebase example CreateUserActivity***



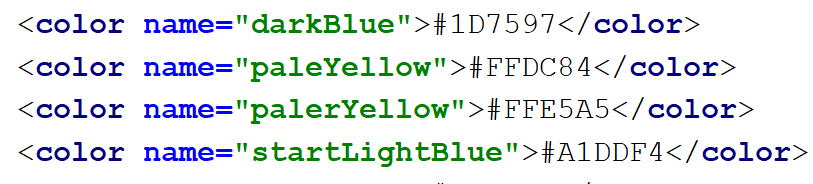
In the above picture, the logic behind the registration process is presented. With the connection to Firebase established, the set layout, the press of the “create” button sends the user credential are sent to Firebase and the specified activity starts. If one of the inputs does not respect the standards, an error message will be displayed guiding to the mistake.

### 5.3.2 UI implementation (A. V. and R. D. B.)

The user interface was implemented using XML, keeping in mind that that the content of the layouts is placed in the exact desired manner. Consistency was a target for the user interface design. This was made by using the same layouts for a big majority of the items such as: buttons, text views etc.

For the purpose of cleaning and optimizing code, the string and files color are used, in order to organize text all around the application.





### 5.3.3 Android application architecture (A. V. and R. D. B.)

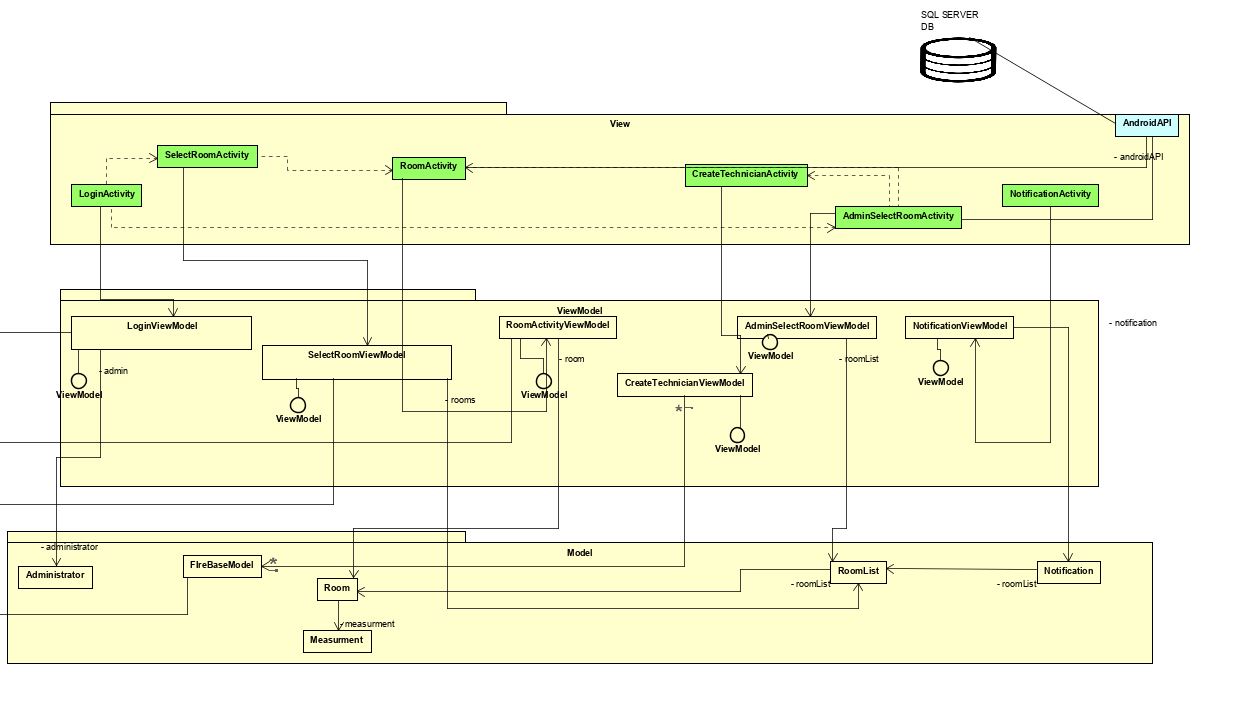
The main design pattern used to develop the Android app, was **MVVM**. This pattern allowed a better structure for the code that had to be built. This architecture splits the code into smaller parts each having their own part to play.

In the **Model** package resides the actual objects that will contain the basic attributes for a class and the needed behaviors.

The **ViewModel** package is the link between the model and the **view.** Its sole purpose is to transform model information into values that can be transformed in the view package.

Lastly, the **View** package will be used for user interaction since it displays visual elements and the controls in form of button handlers.

***Figure 5.3.1 MVVM design pattern***



In **Figure 5.1** the **MVVM** model has been applied. Classes have been created in the **Model** make an interaction with the **ViewModel** package that contain most of the logic which will be called by classes in the **View** package.

Basically, classes with corresponding attributes have been created in the model which where instantiated as object in the view model where the information is transformed so that it can be displayed on the view which contains all the user interaction tools.

# 6 Testing

## 6.1 Android Testing (A. V. and R. D. B.)

Regarding testing for the Android app, the chosen method was **Black Box** testing. This method primarily focuses on test cases for the user interaction. Different scenarios where created to test if the set rules are functioning properly for the given app.

This subchapter will continue in illustrating the test cases that where implemented for this part of the report.

***Figure 6.1.1 User Login Test Case***

***Pre-condition:*** *Make sure that the user account exists*

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Action** | **Reaction** | **Result** |
| 1 | Verify if a user can login entering a valid username and password | System checks to see if the information entered is of a valid type | System displays a confirmation message.  Access is granted |
| 2 | Verify if a user can login entering a valid username and invalid password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 3 | Verify if a user can login entering an invalid username and valid password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 4 | Verify if a user can login entering an invalid username and invalid password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 5 | Verify if a user can login entering a password not matching the required length | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 6 | Verify if a user can login entering a username and not entering a password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 7 | Verify if a user can login not entering a username but enters a password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 8 | Verify if a user can login by entering no information in the two fields | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 9 | User has internet access | System verifies if information is sent by the user | System displays a confirmation message.  Access is granted |
| 10 | User has no internet access | System verifies if information is sent by the user | System displays an error message.  Access denied. |

**Figure 6.1.2 Create Users Test Case**

***Pre-condition:*** *Make sure that the administrator account can create user accounts*

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Action** | **Reaction** | **Result** |
| 1 | Verify if a user can be created by entering a valid username and password | System checks to see if the information entered is of a valid type | System displays a confirmation message.  Account is created |
| 2 | Verify if a user can be created by entering a valid username and an invalid password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 3 | Verify if a user can be created by entering an invalid username and a valid password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 4 | Verify if a user can be created by entering an invalid username and an invalid password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 5 | Verify if a user can be created by entering a password not matching the required length | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 6 | Verify if a user can be created by entering a username and not entering a password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 7 | Verify if a user can be created by not entering a username but enters a password | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 8 | Verify if a user can be created by entering no information in the username field and password field | System checks to see if the information entered is of a valid type | System displays an error message.  Access denied. |
| 9 | Administrator has internet access | System verifies if information is sent by the administrator | System displays a confirmation message.  Access is granted |
| 10 | User has no internet access | System verifies if information is sent by the administrator | System displays an error message.  Access denied. |

***Figure 6.1.3 View Data Test Case***

**Pre-condition:** *Data must be stored an accessible for the give users*

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Action** | **Reaction** | **Result** |
| 1 | User must be logged in | System checks to see if the information entered is of a valid type | System displays a confirmation message.  Access is granted. |
| 2 | User must be connected to the internet | System checks user credentials | System validates user credentials.  Access is granted. |
| 3 | Establish a secured connection between the data source and the mobile app | System establishes a connection | System displays a confirmation message.  Access is granted. |
| 4 | Connection between data source and the mobile app is not established | System fails to establish a connection | System displays an error message.  Access is denied. |
| 5 | User selects which type of data he/she wants to view | System finds the data and returns it to the app | Data is displayed for the user. |
| 6 | User selects which type of data he/she wants to view | System cannot find the requested data | System displays an error message.  Data is not found |
| 7 | User selects which type of data he/she wants to view | System finds the data but cannot display it | System displays an error message. |
| 8 | User is be able to click and choose the selected data | System checks to see if the function is implemented | System displays the data |
| 9 | User is not able to click and choose the selected data |  |  |

**Figure 6.1.4 Remove User Test Case**

**Pre-Condition:***User account must be created and should be accessible .*

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Action** | **Reaction** | **Result** |
| 1 | User account is available | System checks to see if the account is available | System displays a confirmation message.  Access is granted. |
| 2 | User account is not available | System checks to see if the account is available | System displays an error message.  Access is denied. |
| 3 | Administrator can log in | System checks credentials | System displays a confirmation message.  Access is granted. |
| 4 | Administrator cannot log in | System fails to establish a connection | System displays an error message.  Access is denied. |
| 5 | Administrator can select the user account | System executes the specified task and removes account | System displays a confirmation message. |
| 6 | Administrator cannot select the user account |  |  |
| 7 | Administrator can select the user account | System fails to remote user account |  |
| 8 | Administrator cannot find the user account |  |  |

**Figure 6.1.5 Edit User Test Case**

**Pre-Condition:***User account must be created and should be accessible .*

|  |  |  |  |
| --- | --- | --- | --- |
| **Step** | **Action** | **Reaction** | **Result** |
| 1 | User account is available | System checks to see if the account is available | System displays a confirmation message.  Access is granted. |
| 2 | User account is not available | System checks to see if the account is available | System displays an error message.  Access is denied. |
| 3 | Administrator can log in | System checks credentials | System displays a confirmation message.  Access is granted. |
| 4 | Administrator cannot log in | System fails to establish a connection | System displays an error message.  Access is denied. |
| 5 | Administrator can select the user account | System executes the specified task and edits account | System displays a confirmation message. |
| 6 | Administrator cannot select the user account |  |  |
| 7 | Administrator can select the user account | System fails to save changes for the user account | System displays an error message. |
| 8 | Administrator cannot find the user account |  |  |

# 7 Conclusions

# 8 References

# 9 Appendices

## 9.1 Appendix 1 Project Description