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](#_Toc8790699)

[2 User stories and requirements 4](#_Toc8790700)

[2.1 User stories 4](#_Toc8790701)

[2.2 Functional requirements 4](#_Toc8790702)

[2.3 Nonfunctional requirements 4](#_Toc8790703)

[3 Analysis 4](#_Toc8790704)

[3.1 Use case diagram 4](#_Toc8790705)

[3.2 Use case description 4](#_Toc8790706)

[3.3 Activity diagram 4](#_Toc8790707)

[3.4 Domain model diagram 4](#_Toc8790708)

[3.5 System sequence diagram 4](#_Toc8790709)

[4 Design 4](#_Toc8790710)

[4.1 IOT Design 4](#_Toc8790711)

[4.1.1 Conceptual diagram IOT 4](#_Toc8790712)

[4.1.2 Class diagram IOT 4](#_Toc8790713)

[4.1.3 Sequence diagram IOT 4](#_Toc8790714)

[4.2 Database design 4](#_Toc8790715)

[4.3 Android design 5](#_Toc8790716)

[4.3.1 Conceptual diagram Android 5](#_Toc8790717)

[4.3.2 Class diagram Android 6](#_Toc8790718)

[4.3.3 Sequence diagram Android 7](#_Toc8790719)

[5 Implementation 8](#_Toc8790720)

[5.1 IOT Implementation 8](#_Toc8790721)

[5.2 Database Implementation 8](#_Toc8790722)

[5.3 Android Implementation 8](#_Toc8790723)

[5.3.1 Firebase implementation 8](#_Toc8790724)

[5.3.2 UI implementation 11](#_Toc8790725)

[5.3.3 Android application architecture 12](#_Toc8790726)

[6 Testing 13](#_Toc8790727)

[6.1 IOT Testing 13](#_Toc8790728)

[6.2 Android Testing 13](#_Toc8790729)

[7 Conclusions 18](#_Toc8790730)

[8 References 18](#_Toc8790731)

[9 Appendices 18](#_Toc8790732)

# 1 Introduction

# 2 User stories and requirements

## 2.1 User stories

## 2.2 Functional requirements

## 2.3 Nonfunctional requirements

# 3 Analysis

## 3.1 Use case diagram

## 3.2 Use case description

## 3.3 Activity diagram

## 3.4 Domain model diagram

## 3.5 System sequence diagram

## 4 Design

## 4.1 IOT Design

### 4.1.1 Conceptual diagram IOT

### 4.1.2 Class diagram IOT

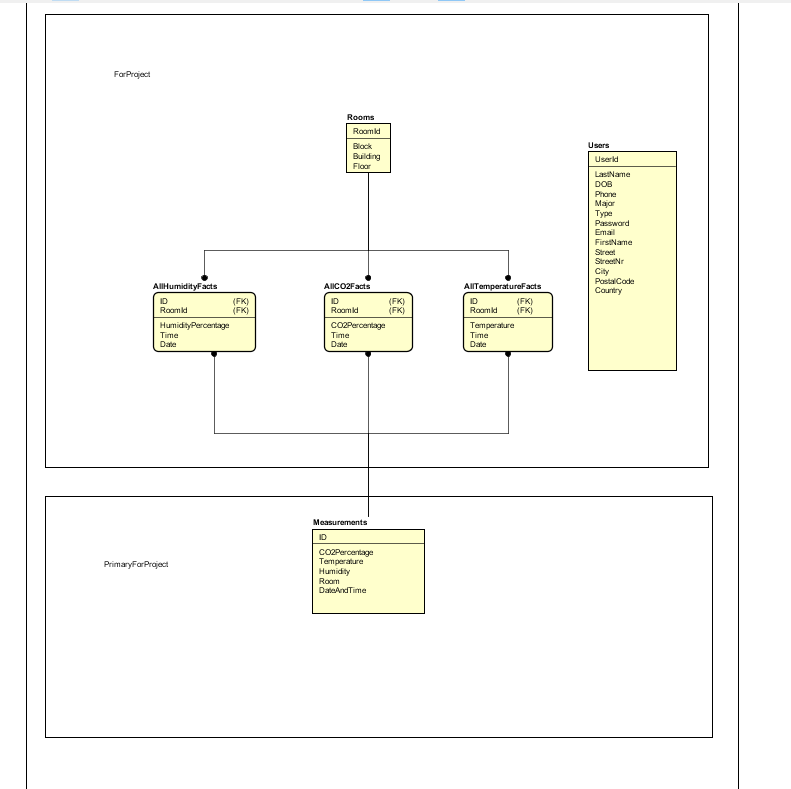
### 4.1.3 Sequence diagram IOT

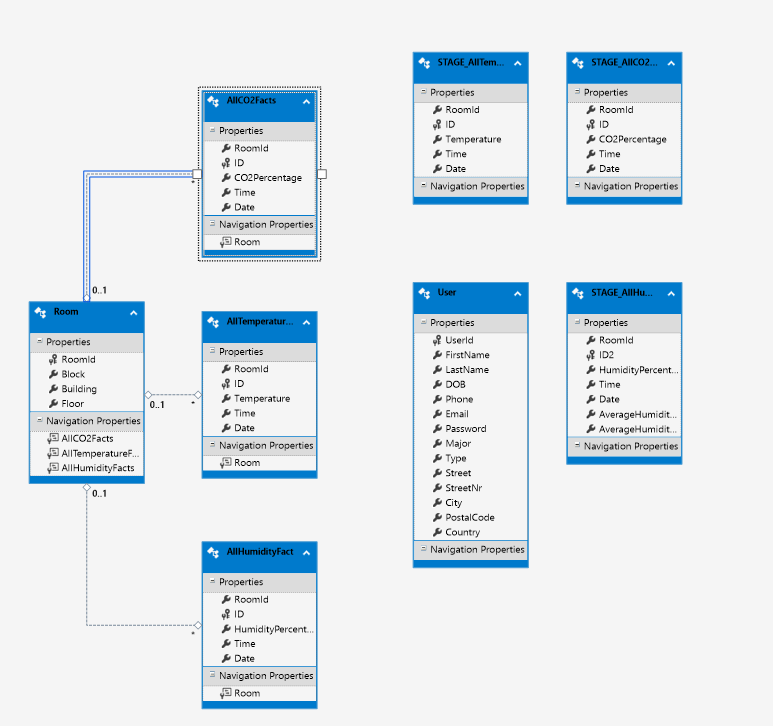
## 4.2 Database design

4.2.1 ER Diagram

Its purpose is to allow the user to create a design that will enable the said user to see a high level view of the database before actually implementing it. Therefore, it allows the use to capture all the needed information in different tables and makes changes big or small, before actually creating the database.

The ER Diagram helps the user to communicate a design to a client that has no technical knowledge in such a way that they can understand. By doing this, the user can easily understand where the communication went poorly or entirely misunderstood.





(Model.edmx API)

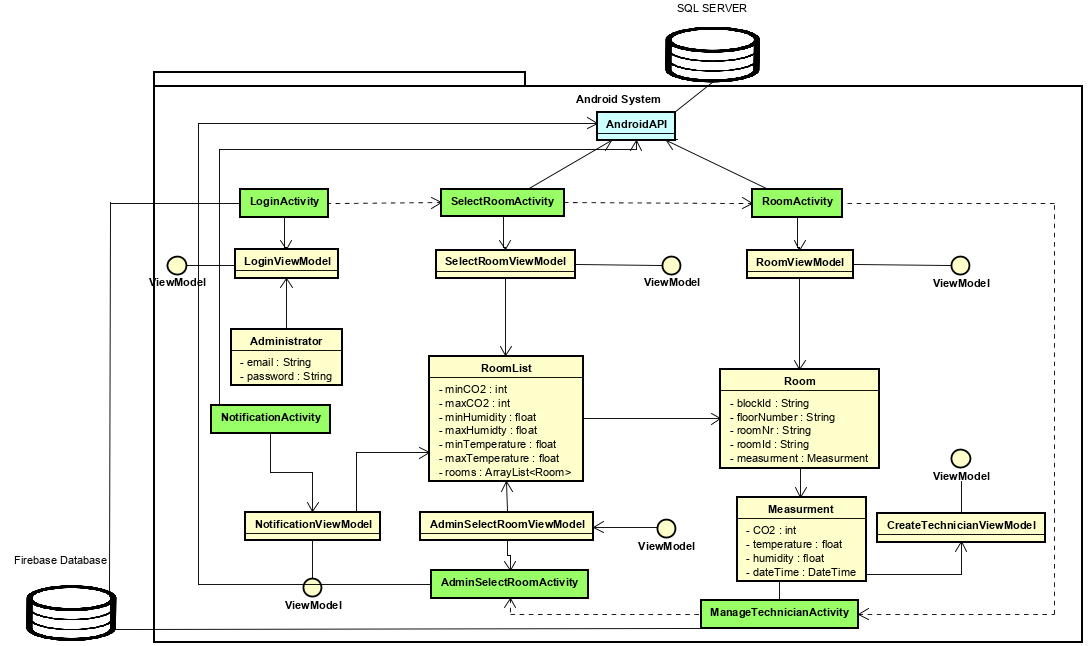
## 4.3 Android design (Alexandru Vieru and Rares Dumitru Bunea)

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

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***

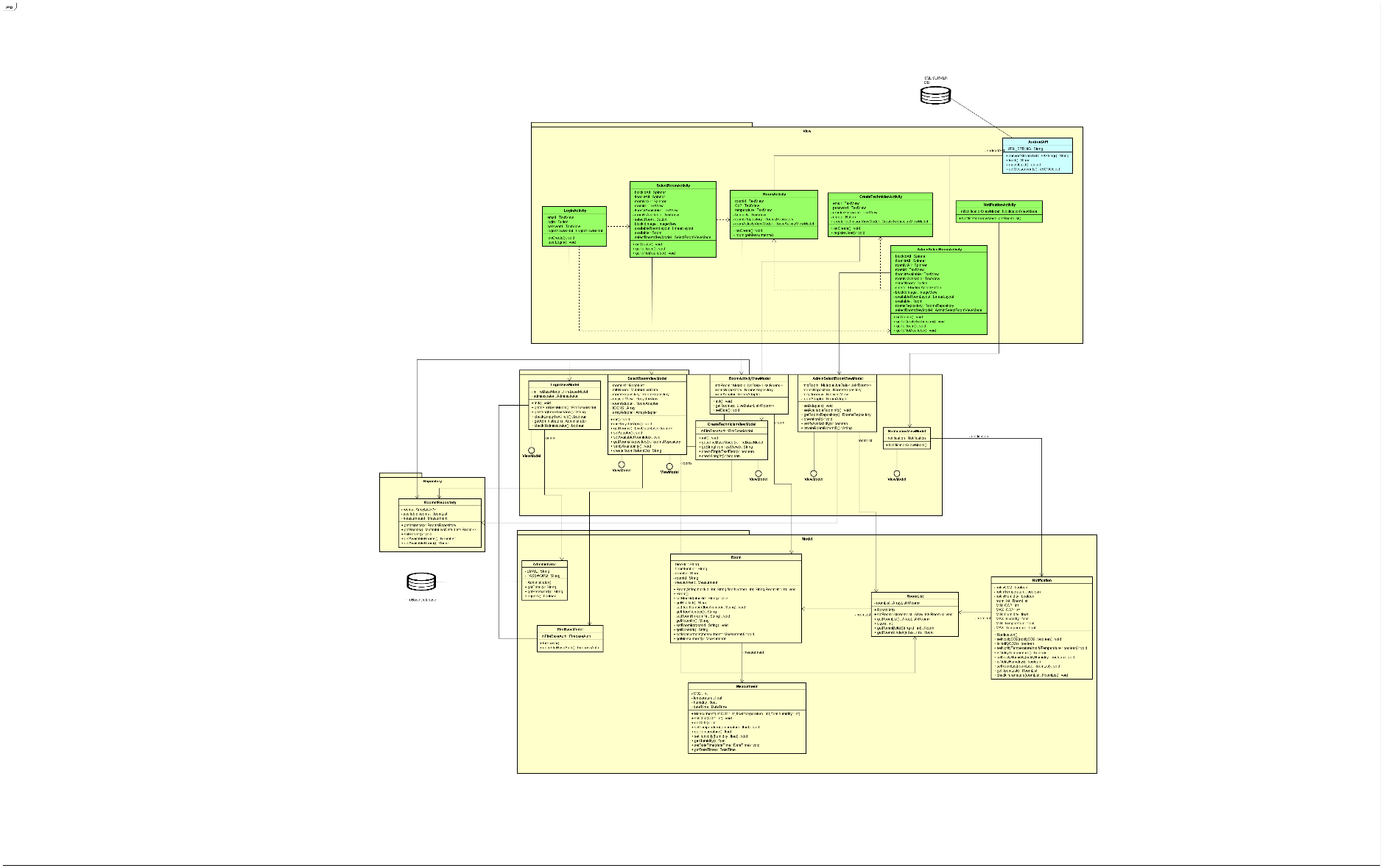


As seen in the above figure, a view model class such as “**RoomViewModel**” will take all the information from the model class “**Room**” and adapt it such that it will become fit to be displayed in the view class “**Room Activity**”. The same can be said with respect about the rest of the entities since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

### 4.3.2 Class diagram Android

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.

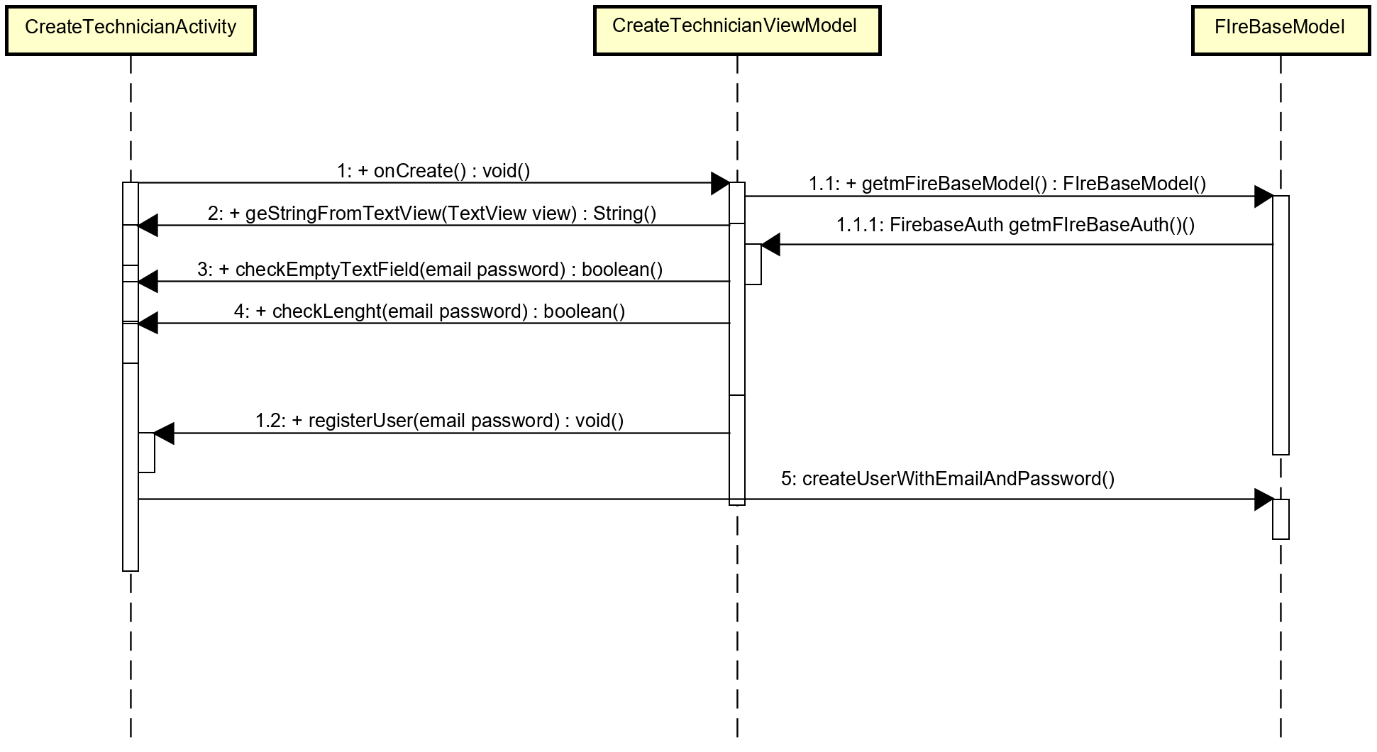
In the **ViewModel** package, the **CreateTechnicianViewModel** class is responsible for instantiating the Firebase class as an object since it is needed to authentificate when registering user accounts since it contains methods such as **checkEmptyTextField**().

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**.

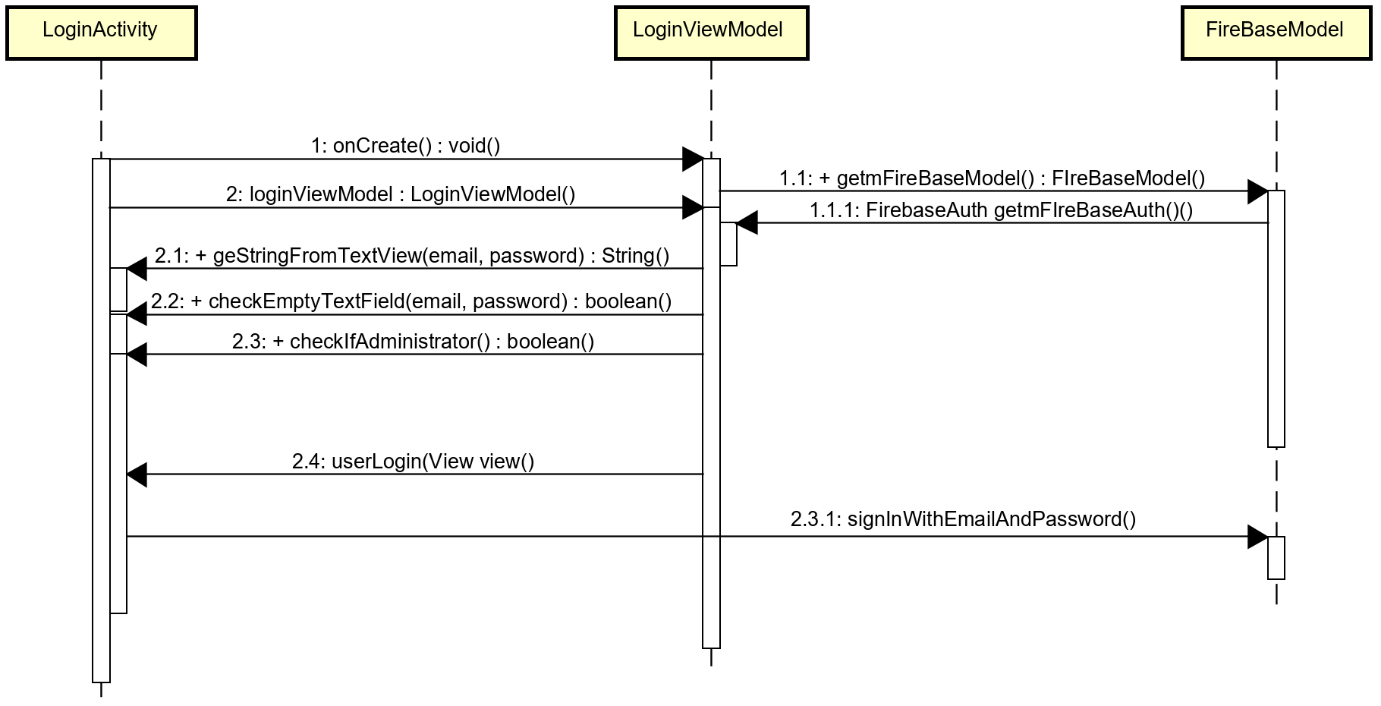
### 4.3.3 Sequence diagram Android

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

## 5.2 Database Implementation

In this chapter of the report the implementation for the Data will be explained and examples will be illustrated. This chapter will start with presenting the Transferring of data from MongoDB to Microsoft SQL Server.

5.2.1 Transfer to SQL

In order to transfer data from the NoSQL database (MongoDB) to the Microsoft SQL Server it was use a script written in C#.

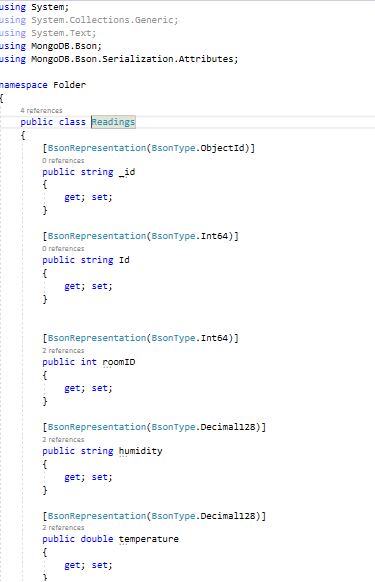
1. First part will retrieve the information from the MongoDB database and add objects into a list of “Read”



1. Second part was to make a connection with the Microsoft SQL Server and add data . (not active MongoDB Database in the example)



Model:

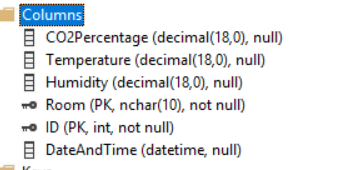


5.2.2 Data warehousing

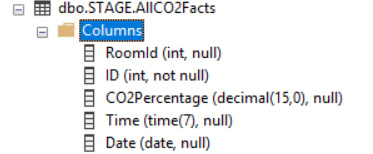
Data from the NoSQL database was added on “PrimaryForProject” Database(Microsoft SQL Server) . From here we made the stages on “ForProject” , splitting DateTime in Date and Time. The Final facts tables were created based on Stages and “Rooms” Table.

A local server was created in order to allow different members of the team to get access on data. There were also created users that were granted the rights to read and write in any tables.

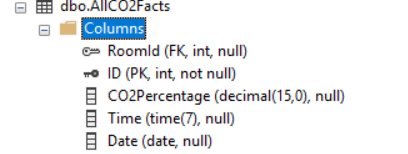
PrimaryForProject, Measurements Table:



Stage for CO2:

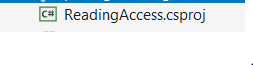


Final Fact table for CO2:



5.2.3 FinalWebAPI (Web-Service)

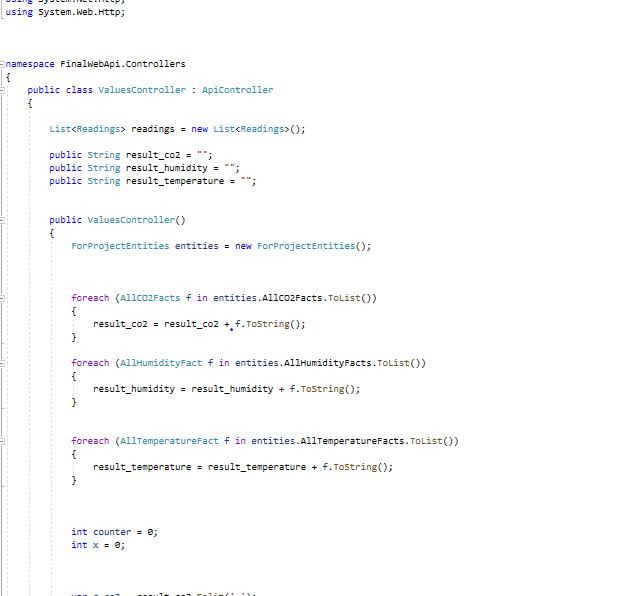
The web-service is made in C#. The FinalWebAPI it’s made out of an API and an adaptor between Microsoft SQL Server Management and the web service.



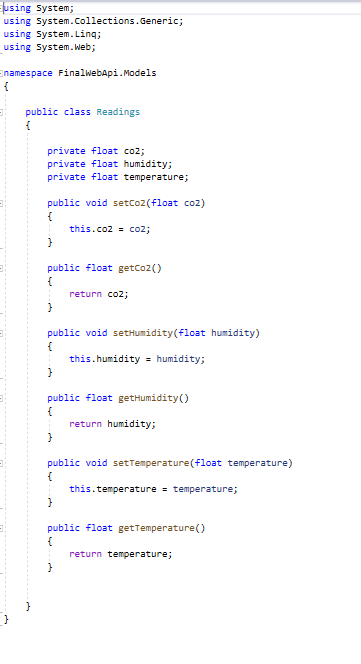


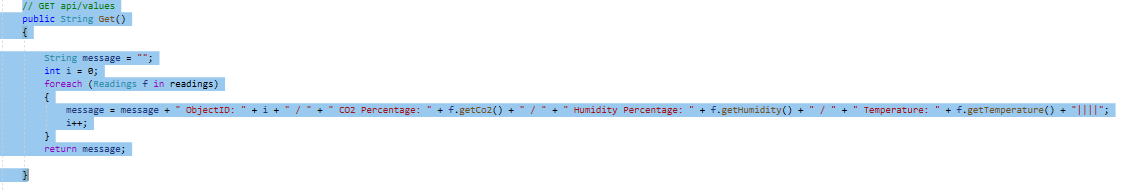
The API has a controller(ValuesController) and a model(Readings)

Part of the controller:



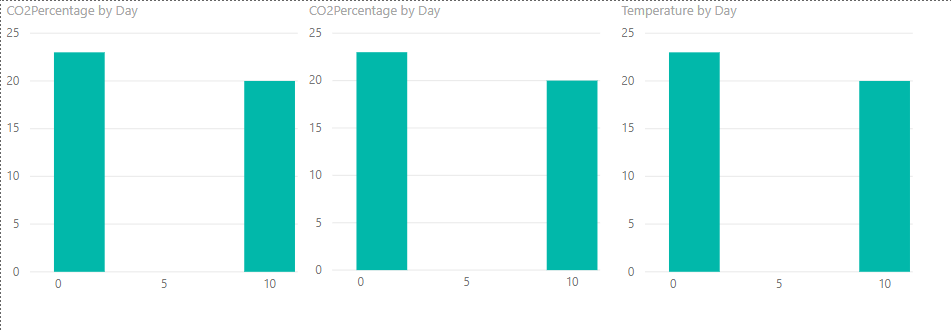
Part of Model:





Running the the project will display a list with all the data from Temperature,CO2 and Humidity.

5.2.4 Power BI (image)



## 5.3 Android Implementation

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

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***



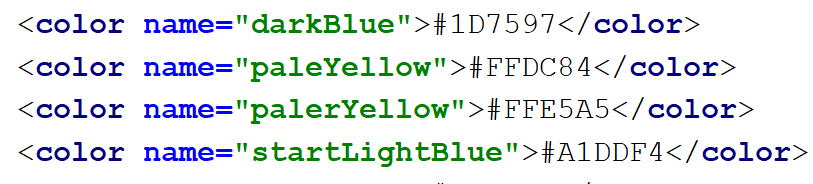
In the above picture, the logic behind the registration process is presented. With the connection to Firebase established, the set layout, the press of the “create” button sends the user credential are sent to Firebase and the specified activity starts. If one of the inputs does not respect the standards,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

### 5.3.2 UI implementation

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

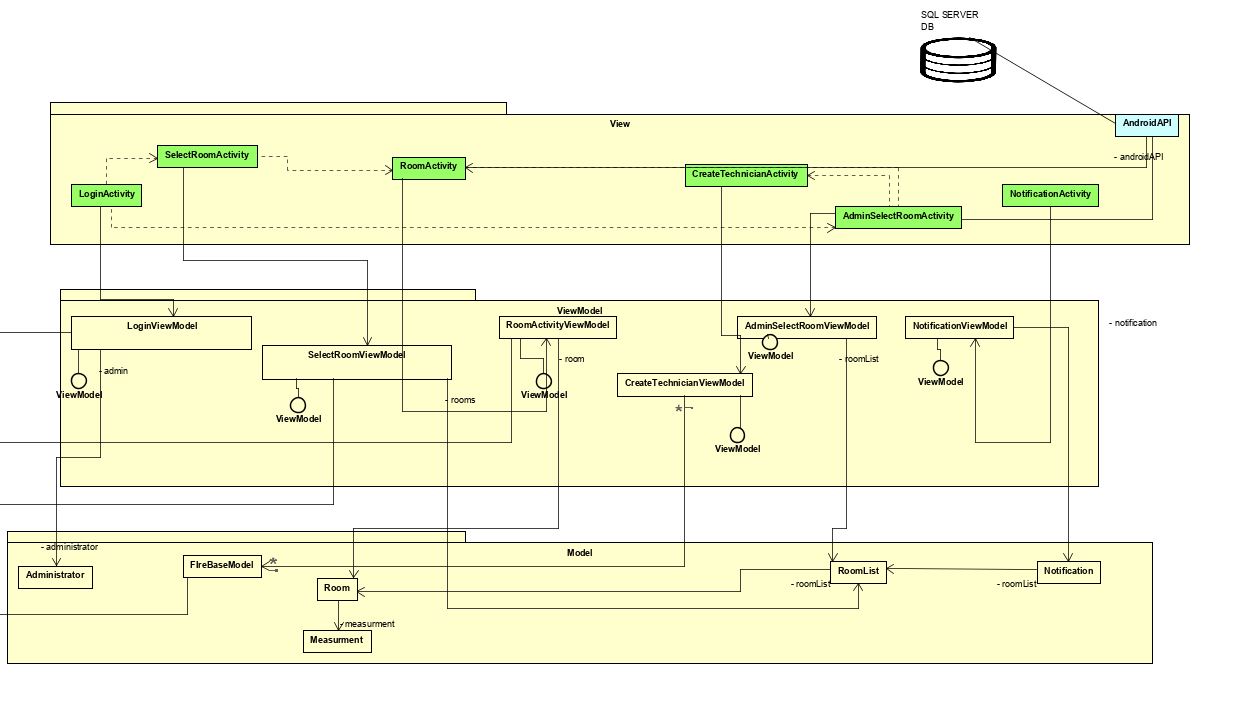
The main design pattern used to develop the Android app, was **MVVM**. This pattern allowed a better structure for the code that had to be built. This architecture splits the code into smaller parts each having their own part to play.

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 IOT Testing

## 6.2 Android Testing

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 6.2.1 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 6.2.2 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 6.2.3 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 6.2.4 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 6.2.5 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