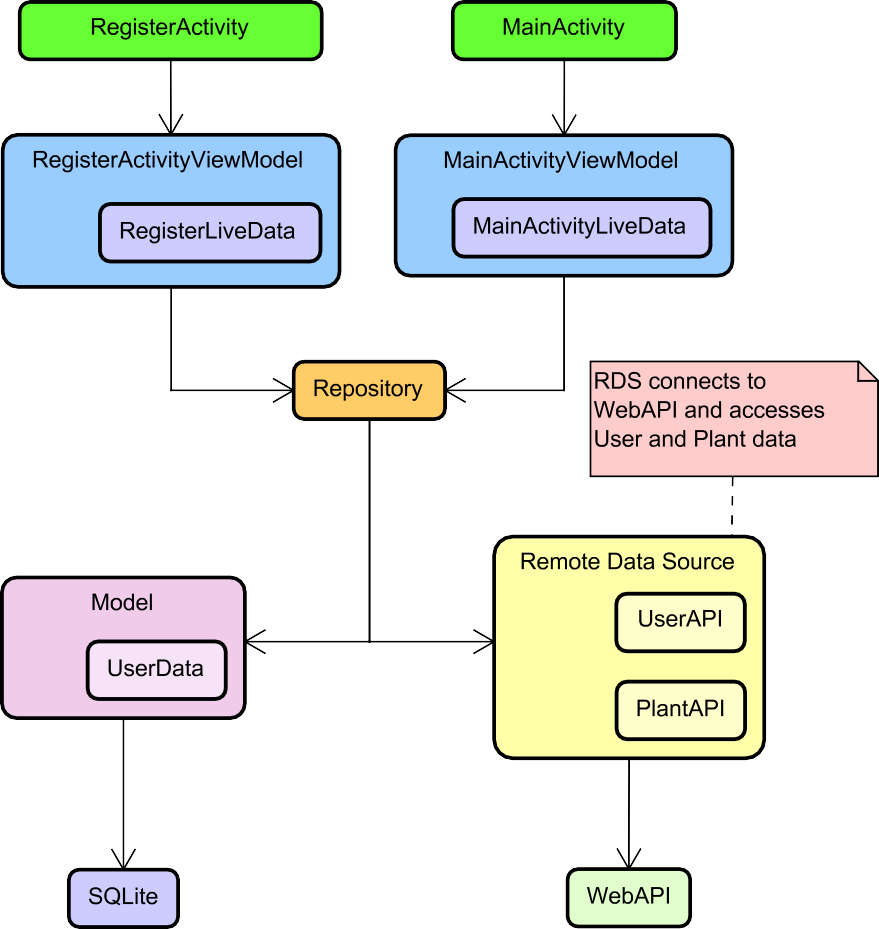
# System Design

**Section 1: Android application**

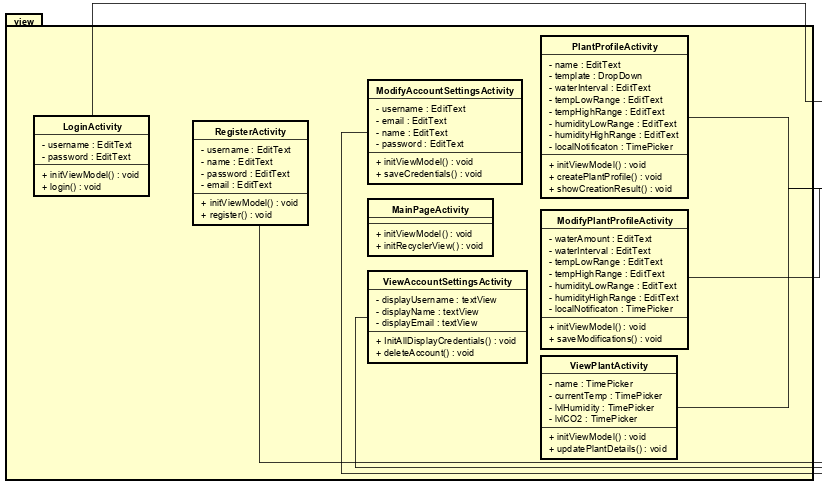
*Responsible: Angel Petrov, Erika Szasz, Ionel-Cristinel Putinica*

The responsibility of the Android application is to grant the end-user the possibility to control a plants’ environmental conditions. A user has the possibility to register and thus login afterwards. This gives them access to sections where they are able to see the list of their associated plants, they can register new plants and modify their account and profile settings.

The implementation of the application follows the MVVM – Model-View – ViewModel system architecture. The following diagram represents the use of MVVM:

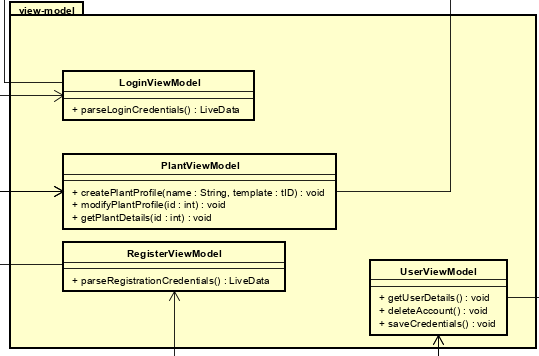


*Figure 1 in Android – MVVM Architecture Diagram (Simplified Architecture Class Diagram)*



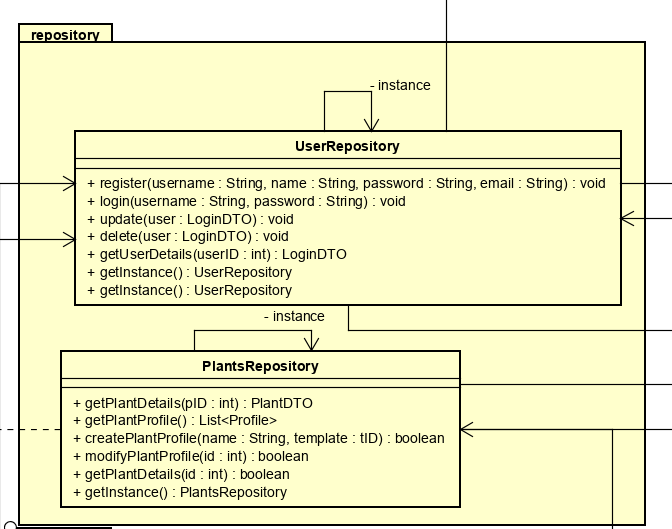
*Figure 2 in Android - Design of MVVM Architecture Diagram – View*

In the View, there are the view controllers that are responsible for handling the View elements and updating them. The view is using ViewModels that are in the view-model package to perform actions in the system. ViewAccountSettingsActivity and ModifyAccountSettingsActivity are responsible for presenting the users’ details and making changes to them. PlantProfileActivity is responsible for creating the plant profile. ViewPlantActivity has the purpose of displaying plant data, whilst ModifyPlantProfileActivity changes only the waterAmount, waterInterval attributes as well as the temperature and humidity low-high ranges.



*Figure 3 in Android - Design of MVVM Architecture Diagram – View-Model*

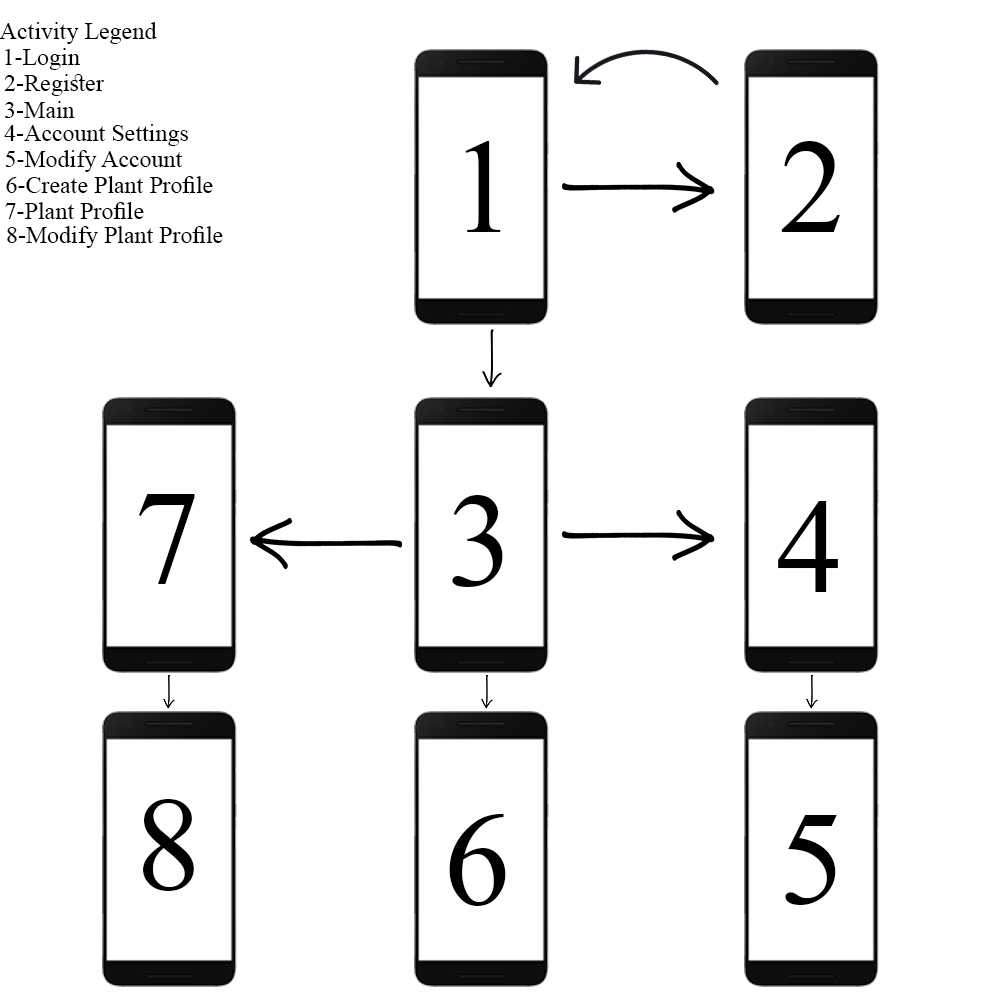
The view-model’s purpose is to handle actions from the view and to serve as a bridge between the view and the model. All view-models pass data to their respective associated repositories. UserViewModel handles operations for user-related activities, PlantViewModel from plant related activities and finally, Login/RegisterViewModel – user specific credentials.



*Figure 4 in Android - Design of MVVM Architecture Diagram – Repository*

The repositories in the design implementation serve as a facade to the database and WebAPI. The User and Plants repositories are handling CRUD operations on model entities. They implement Singleton design pattern to ensure that there is only one instance of the mentioned repositories in the system. The repositories are using DTOs from the model to exchange data with the WebAPI.

# Section 1.1: UI Design and Navigation

****

*Figure 4 in Android – Layout Navigation of Android Application*

There will be a total of 8 activities for the Android application. These views will include graphical objects such as drop downs, text fields for input, buttons and other visual elements to enhance the user experience. On launch, the application will first display the LoginActivity (number 1 in Figure 4). The MainActivity will have multiple possible redirects to other activities. All activity layout sketches have been added as an appendix to this document.



*Figure 5 in Android – Layout Design of Main Activity*

# Section 2: Data

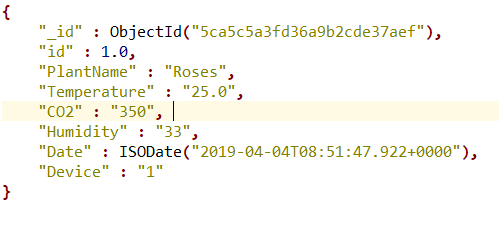
*Responsible: Josipa Babic, Mihai Tirtara, Eduard Costea*

## Cloud Based Data

### IoT system needs to exchange data with android application. To make it possible cloud-based database were necessary. MongoDB is chosen in this case and it fulfils all requirements.

MongoDB is NoSql database program, it uses JSON like documents.

The collections are going to be caped. Caped collections support fixed-size collections. This type of collection maintains insertion order and, once the specified size has been reached, behaves like a circular queue. Capped collections will help with limited available space in the cloud.

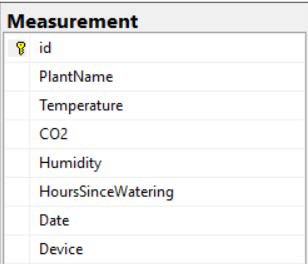


*Figure 1 in Data - Document in MongoDB*

## SQL Server

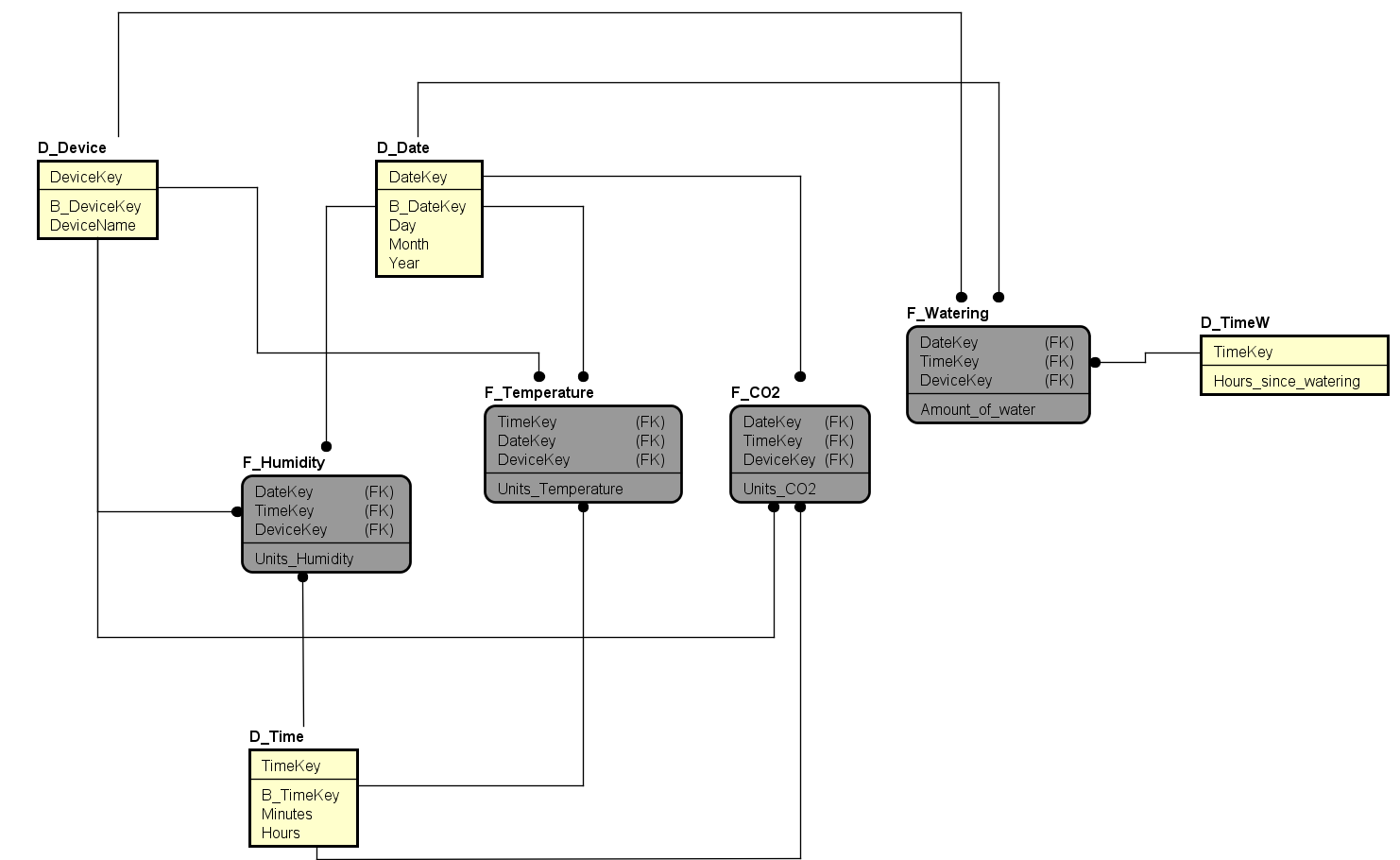
### A document will be presented in the following format. It will contain the values of all sensors: temperature, CO2, humidity and watering since, as well as id plant name, date and device.

Data from MongoDB should have backup in Microsoft SQL Server. The main purpose of Server is to store all data from the sensors. The data from the MongoDB should be stored in a transactional database that serves as the source database for the ETL process.



*Figure 2 in Data - Source Transactional Database*

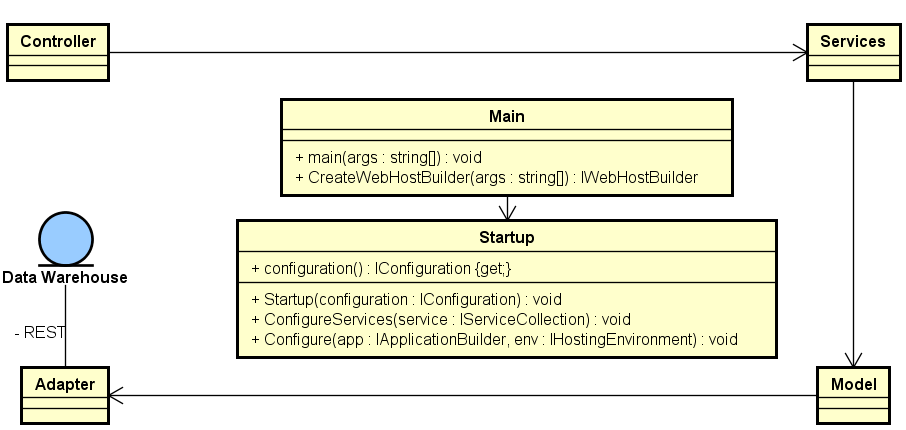
### The source database follows the structure from Figure 2. based on the structure the dimensional model is created. For the temperature the grain is units of temperature. For displaying temperature information’s about time, date and device are needed. For displaying watering, the information when was the last time of watering, date and device were needed. The grain is amount of water.



*Figure 3 in Data - Temperature Dimensional Model*

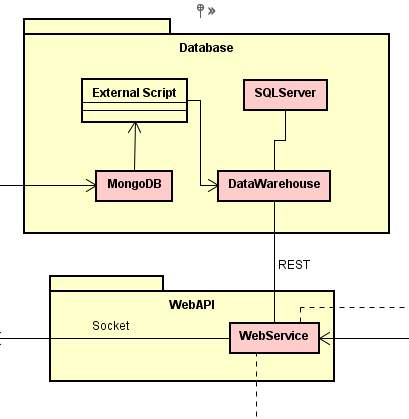
### There are four dimensions in our database. The date, device, time and time of watering. These dimensions allow us to look for more specific details. The date dimension allows looking for the date or more specifically for month or day. The time dimension is separated from date dimension, so users can look for specific information at specific hour or minute. The device dimension helps to identify from which sensor information is coming. The dimension watering clearly represents when was the last time that plant was watered.

***Web API***



*Figure 4 in Data - Diagram for API Design*

### 1. The API makes use of rest server node.js



*Figure 5 in Data - Data - Part of an IoT System*

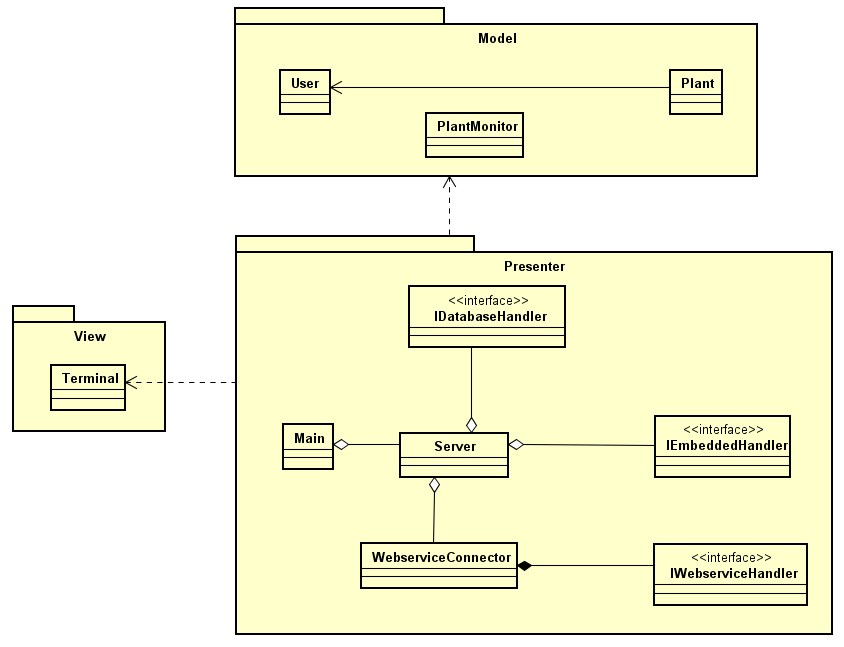
A result of the design is Figure 5 which is part of the bigger picture that consists the IoT System.

# Section 3: IoT Implementation Design

Responsible: Kenneth Petersen, Remedios Pastor Molines, Diyar Hussein, Christian Sørensen

Bridge Application

The bridge application consists of three handlers which will deal with the communication in each of the three parts of the system: the embedded system, the database and the webservice.



*Figure 1 in IoT – Bridge Application*

The communication with the database uses the MongoDB driver.

The communication with the embedded system is through a Socket protocol as well as the webservice.

The model classes are used to encapsulate data for users, plant profiles and plant sensors respectively.

The WebserviceConnector maintains the socket connection to the web service and uses the IWebServiceHandler to perform actions matching the protocol commands and writes a response to the socket.

The socket communication is based on json and the model objects are serialized to json and sent over the socket connection. The protocol for communication is as follows:

|  |  |
| --- | --- |
| **Message** | **Response** |
| getuser:id | json/null |
| adduser:json | id/null |
| modifyuser:json | true/false |
| removeuser:id | true/false |
| getplant:id | json/null |
| addplant:json | id/null |
| modifyplant:json | true/false |
| removeplant:id | true/false |
| getplantmonitor:id | json/null |

Zero byte is used for delimiting the individual messages and responses. All messages and responses are UTF-8 text.