DESIGN DOCUMENT

INDUSTRIAL ROBOT MONITORING FOR OPIFLEX AB

GROUP 5

2017-12-07

VIKTING GUSTAV LINDBERG, WILLIAM ACHRENIUS, NOÉ BAYLAC JACQUÉ, NAIDA DEMIROVIC, ANDREAS HEMLÉN, ERIK MARTIN CAMPAÑA, ELAHEH AILINEJAD, PONTUS WEDÉN

Table of Contents

1 Background

Our client OpiFlex has asked us to develop an app and back-end server solution for them regarding monitoring of their robots. The company can only see information on their robots locally through a device which is physically put on the robots. Therefore, they have asked us to develop a more mobile solution for them, so that their operators aren't limited to being on site if they want to monitor their robots. They've specified that the mobile solution to be able to monitor the robots, presenting information from the server. The system should also be able to handle error logs and possibly send these to the company for quick support.

2 High-level Description

After having a meeting with OpiFlex we immediately started on our designs for the mobile solution which consists of one app and one server.

The server's purpose is to relay data from the robots to the app. If a user log into the app then the server sends the information that the account is permitted to view. The server keeps sending the data asynchronously as it gets new data from the robots continuously. The app will receive a push notification from the server if a robot has stopped production or if another event of critical importance happens.

The app is meant for robot monitoring as mentioned above, hence the apps focus is to present this data to the user, through detailed lists and push notifications. The app presents this data in three views, one overall view of all the robots, a more detailed view of a specific robot in that list, and a "Log" page for easy viewing of all the messages the server has sent to the app, including error logs. The users will be able to edit these logs or send them to OpiFlex.

Another desired functionality that was requested was that there should be an “Administrator” account where the administrator can select which data accounts of their company can see on the app. For example, Administrator of company X could select to not show what products are produced when a worker uses the app and look at a specific robot’s details. This is an “implement if there’s time” feature, although, the functionality for an operator to choose what data he/she personally wants to see in the app will be implemented.

3 System Overview

The current industrial robot has a screen attached to it that displays messages and other important data. It has sensors that can identify in what cell the robot is currently located. For safety reasons, it can also identify close objects, for instance if any workers are nearby, so that it can slow down to avoid accidents. Currently it does not upload the data to any database which means that everything is stuck inside the robot. A human worker must therefore always be nearby to detect if a robot has stopped and why.

4 Software Architecture

The overall system can be divided in two units, the firebase server provided by Google, and the android application developed by ourselves. For demonstration purposes, we will also develop a simple program to fill the server with data, since we will not have access to the robot meant to do that. The server has a few requirements that need to be fulfilled, it should be able to:

- Receive JSON objects.
- Store the JSON data correctly.
- Send notification alerts to the android device when important data is received.
- Be scalable.
- Distinguish between different companies and only allow access to each company's own data.

Based on these requirements we decided to use a firebase server. Firebase is a server built upon handling JSON data and it stores all its data in JSON format, it is also easy to integrate into apps. Android applications handle all their database connections through JSON objects and it makes it fit perfectly with firebase as a backend server. Firebase makes it possible for us to skip the database building part. We still need to configure the database and plan how to structure our data. Firebase allows the use of cloud functions that can be run on the database. These functions will allow us to structure data however we want in the database and send notifications from the database to the connected users.

The application is where most requirements are listed, the application should:

- Be an android application.
- Request data from the server.
- Use login functionality to access the server.
- Display a list of all robots owned by the currently logged in user.
- List the most relevant data for a selected robot.
- Have a setting page where the user can choose what data to display.
- Have scalable code.
- Support “as early android version as possible”.
- Have a page for listing all errors for a robot.
- Allow the user to write and edit logs for each error in the error list.
- Display notification alerts when received from the server.

- Notified errors should be “acknowledged” before the notification is removed.

One of the requirements was to make an android application, so we had no choice there. Making login functionality and creating users is easy through the firebase server. It has its own API that can be used for login purposes and for creating new accounts. One important part of the android app was that it should support as early android version as possible. We decided to use the android version KitKat. KitKat was released year 2013 and is the android version between 4.4 and 4.4.4. It is supported on most current android devices [1] and should also have the necessary support required to integrate it with the firebase server and implement the functions needed. After the implementation is done we will have to refactor the code to make sure it’s scalable. This means making sure there are no arbitrary limitations in the application or server, no fixed limit structures or too small integer sizes.

4.1 Robot Simulation Script

The client provided us with a simple script for demonstration and testing purposes and the script is meant to simulate how the actual robot might send the data. The data is in JSON format and is shown in the figure 1 below.

```
{
        "robotid": |string|,
        "robotcellid": |string|,
        "productname": |string|,
        "starttime": |int|,
        "currenttime": |int|,
        "running": |bool|,
        "produced": |int|,
        "target": |int|,
        "cycletime": |int|,
        "station": |string|,
        "error": |bool|,
        "eventcode": |int|,
        "inputpallet": |int|,
        "outputpallet": |int|,
        "speed": |int|,
        "log": |string|
}
```

Figure 1. Data sent from a robot.

This script was sending this data to the localhost of the device running it. To make it compatible with our database we had to complement it with a simple python program that receives the data from localhost and then uploads it to firebase.

5 Detailed Software Design

5.1 Android Application

The application is supported by 12 classes whose functionality is based on the requirements from OpiFlex.

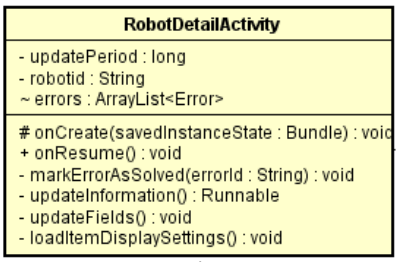

Figure 2. RobotDetailActivity class.

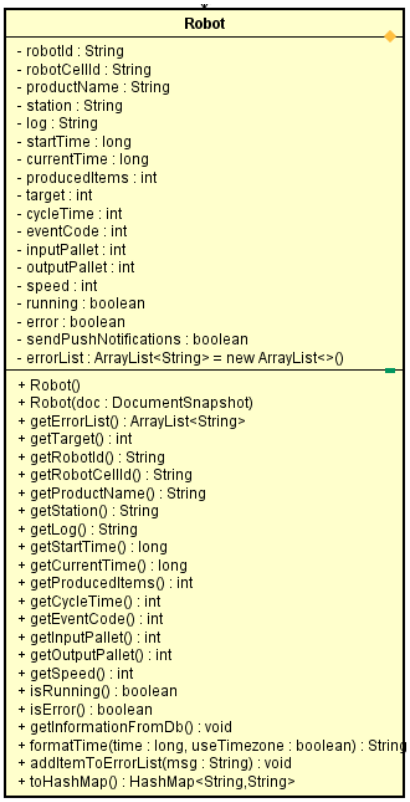

Figure 3. Robot class.

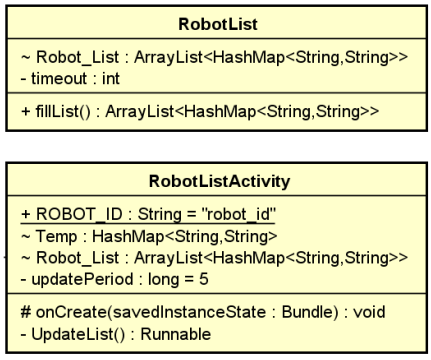

Figure 4. RobotList and RobotListActivity classes.

The robot detail activity class, shown in figure 2, displays detailed information about a robot to the user. The Robot class, shown in figure 3, has attributes to store real-world robot data from OpiFlex (which are sent from the server), and RobotListActivity, shown in figure 4, is used to display a list of Robot objects to the user. The list used by the robot list activity is both fetched and filled from a RobotList object (figure 4). In the application, the user will be presented with a list of robots, which is an instance of the RobotListActivity class. If they want to see more detailed information about a robot, they must select it, which means displaying RobotDetailActivity.

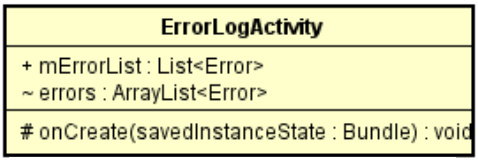

Figure 5. ErrorLogActivity class.

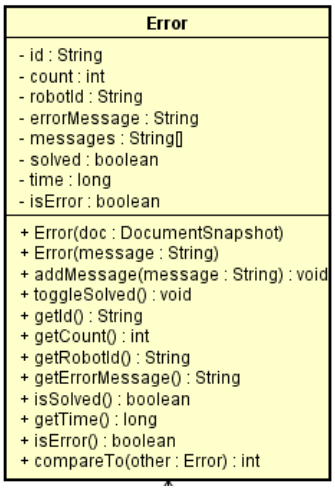

Figure 6. Error class.

The error log activity (figure 5) displays the error log made of objects of type Error (figure 6) to the user. The user can access the error log to view, edit and delete entries for a robot.

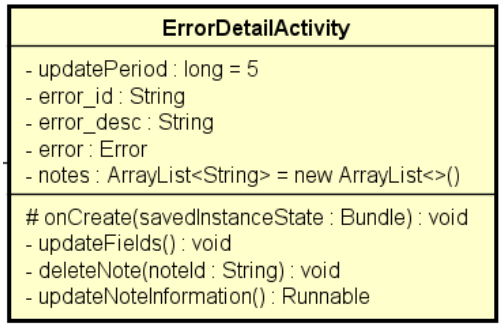

Figure 7. ErrorDetailActivity class.

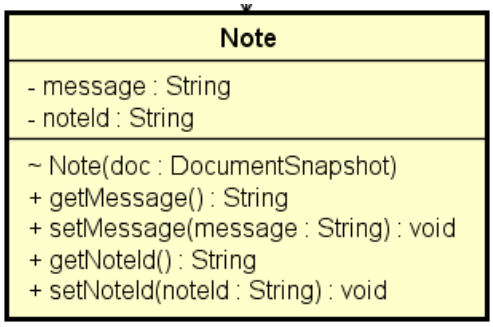

Figure 8. Note class.

For each error in the error log, the user can see detailed information about it. This information is displayed via the error detail activity class (figure 7). This class displays a list of Note (figure 8) objects to the user. Notes can be added, deleted and edited by the user.

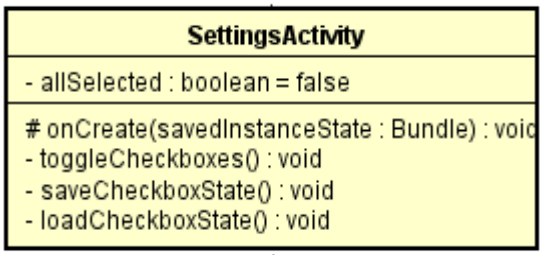

Figure 9. SettingsActivity class.

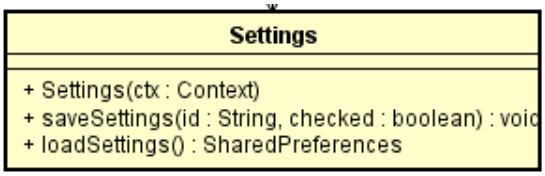

Figure 10. Settings class.

The user will be able to change what information is displayed when viewing detailed information for a robot. The settings activity (figure 9) displays options to the user, which are then saved in a Settings object (figure 10).

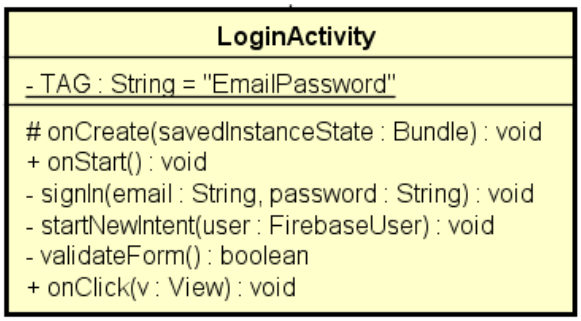

Figure 11. LoginActivity class.

The login activity class (figure 11) displays a login page to the user. The class has a connection to the server which validates the user credentials. Upon successful login, the user will be redirected via the "startNewIntent" function to the list of robots.

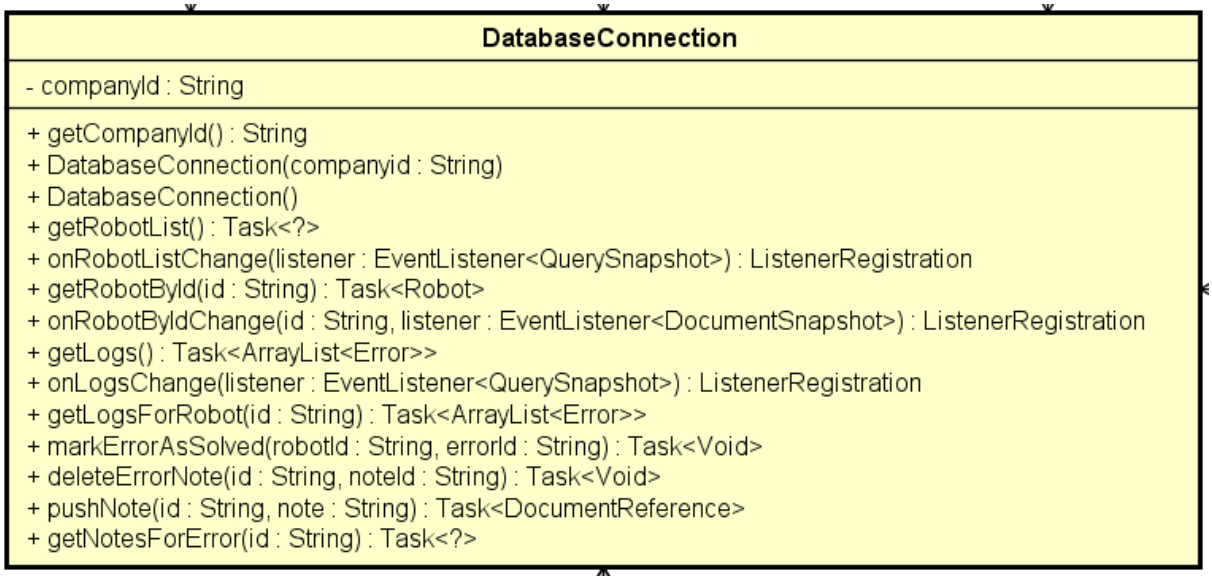

Figure 12. DatabaseConnection class.

All communication with the server is performed via the DatabaseConnection class (figure 12). This means that each object that communicates with the server will have an instance of this object. Most of the functions represent actions that can be performed on the server, which is getting and setting data.

The classes described and their relations are presented in the complete class diagram (figure 13).

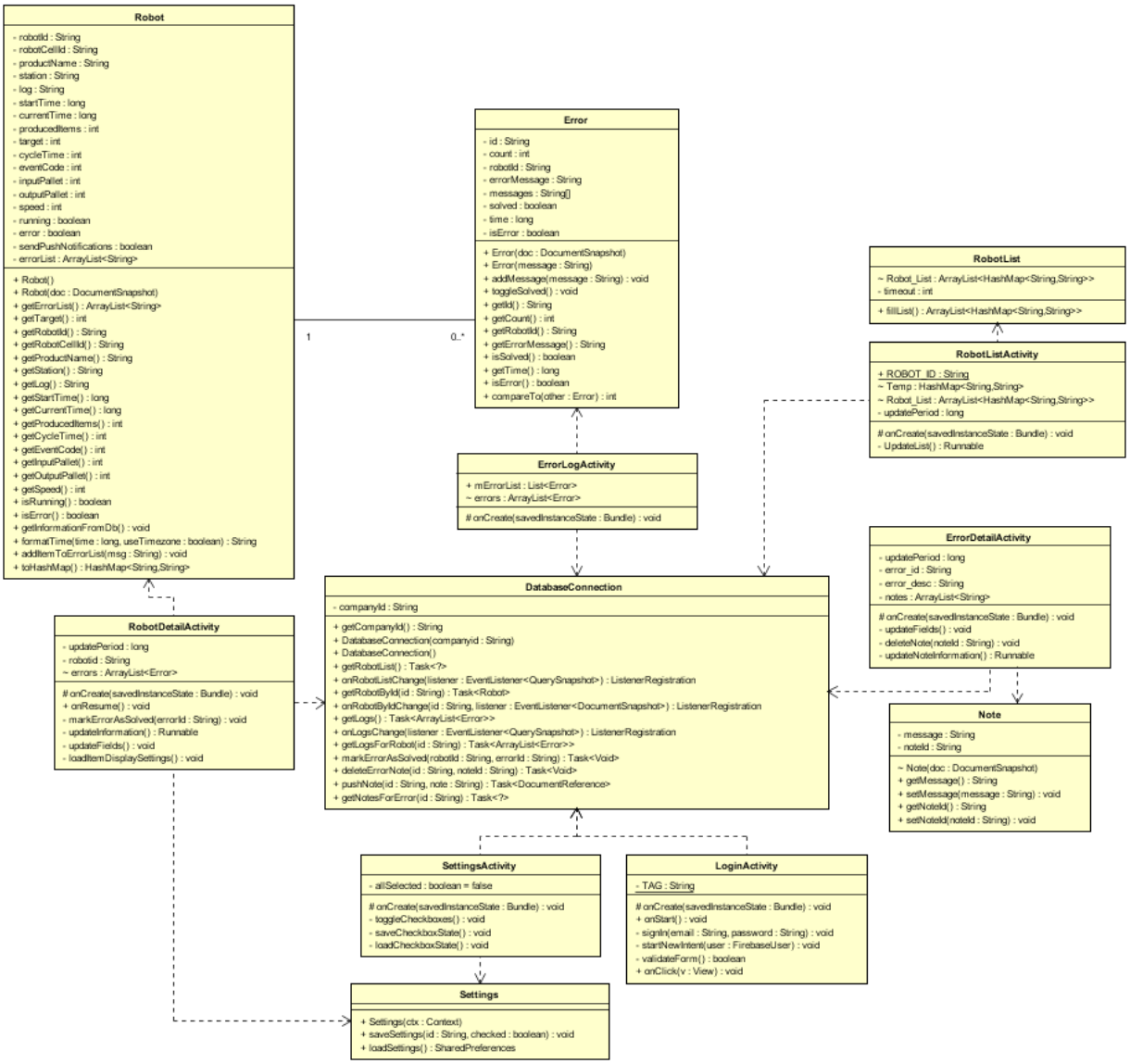

Figure 13. Complete class diagram of the system.

5.1.1 Sequence Diagrams

This section features a selection of the different interactions between the objects, covering some of the most important functionalities of the system. These functionalities are:

- Displaying the robot list (figure 14).
- Possibility for user to add errors to the error notes (figure 15).
- Displaying detailed robot information (figure 16).
- Displaying the error log (figure 17).

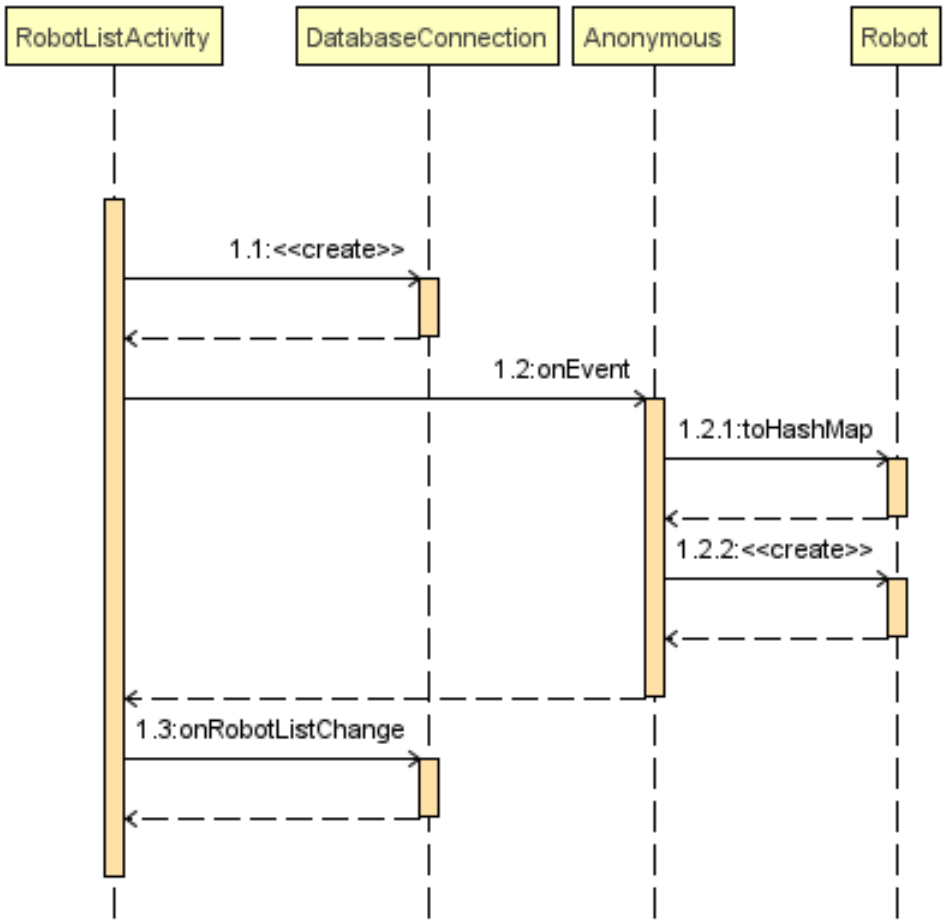

Figure 14.

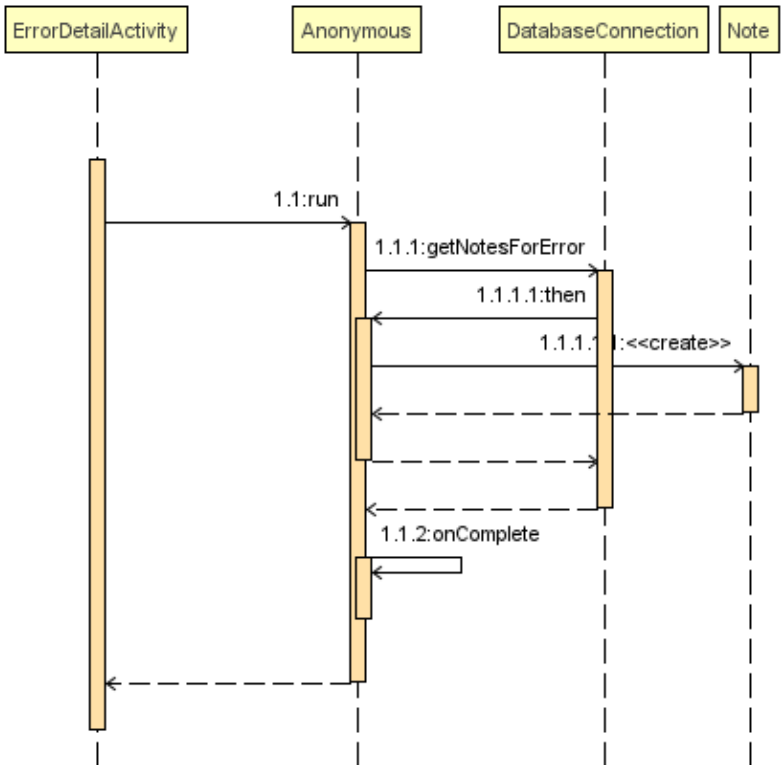

Figure 15.

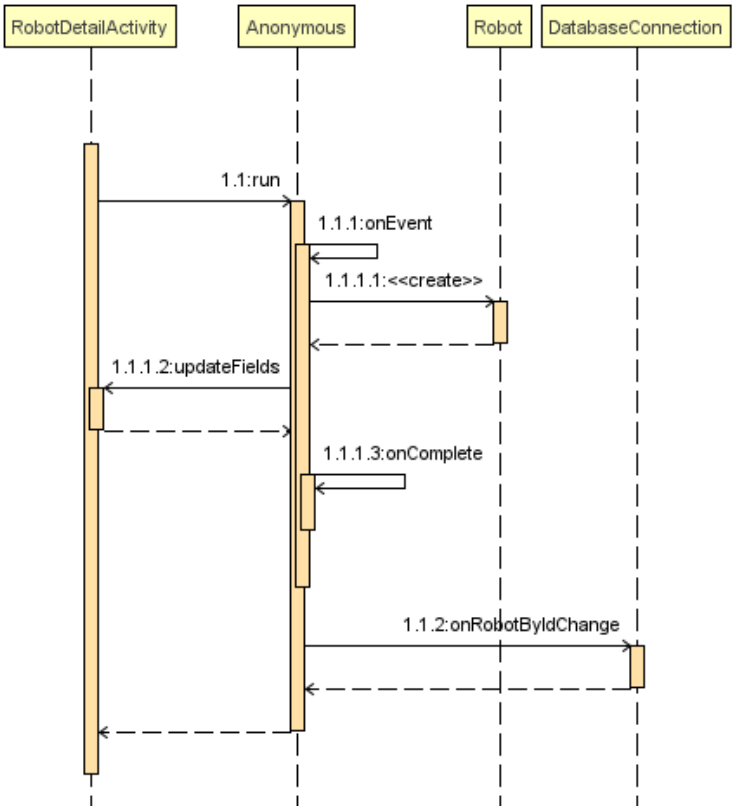

Figure 16.

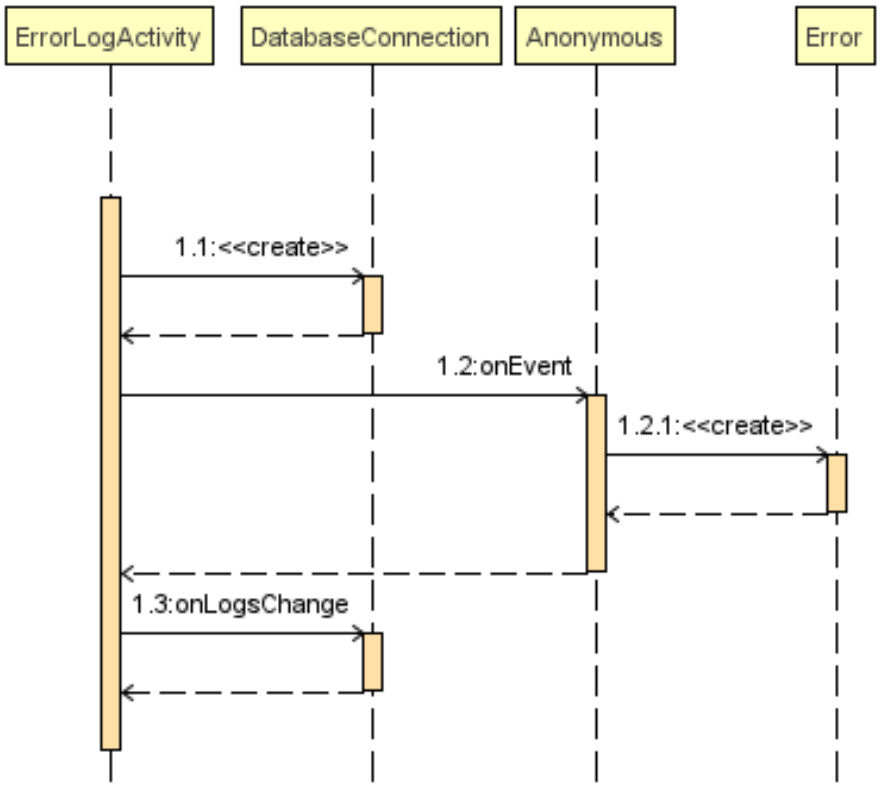

Figure 17.

5.2 Server

The server structure is based on firebase. Firebase have tools for managing different aspects of a server. The tools used in this project is authentication and Firestore. Authentication is the way firebase manages users for the application. They are multiple types of users available, this project uses the email/password type, phone, google and Facebook user types could be implemented in the future if needed. The Firestore is a NoSQL database, this means that no tables are used, instead the data is structured in a way of Documents and Collections in a tree structure.

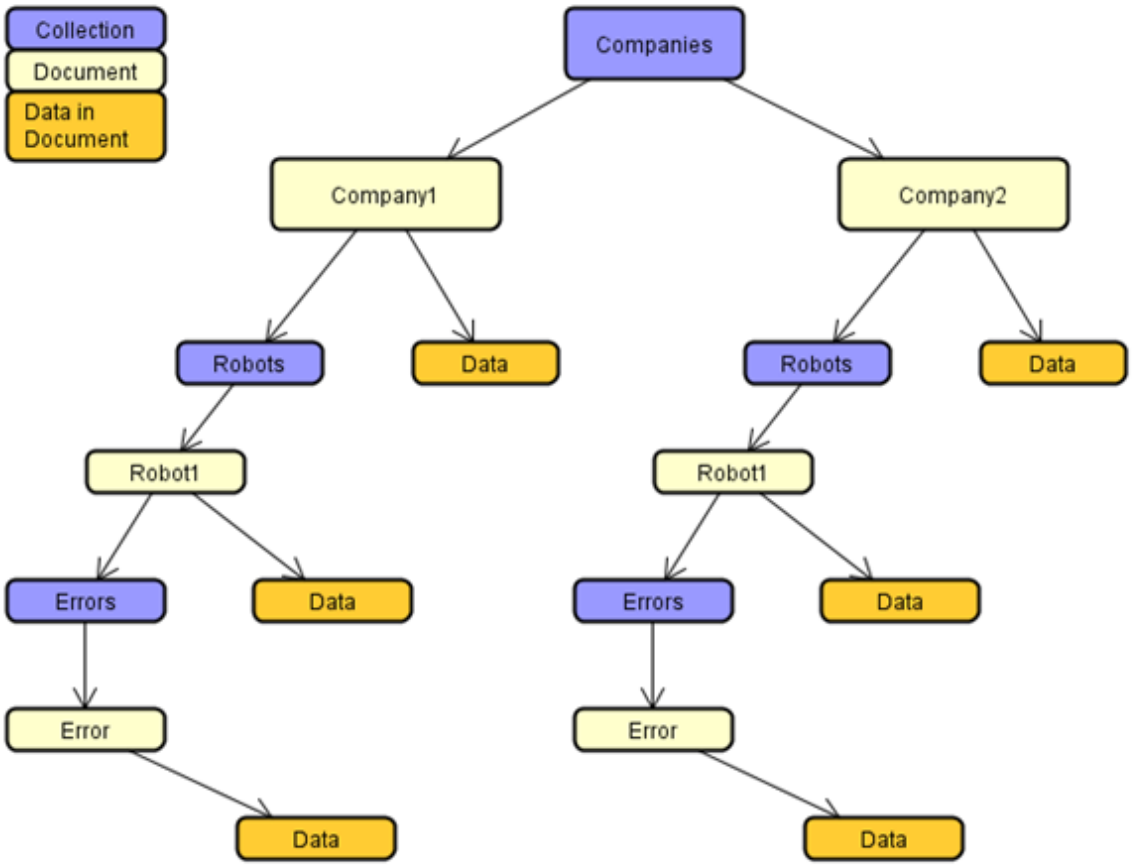

Figure 18. User structure.

Collection is storing different Document which have the data in them, shown in figure 18. Document can't have another document in them, but they can link to another document. The key difference from a regular NoSQL database is that you can build a deep tree structure without the need of loading all the child nodes of the targeted node. Users from the Authentication is linked to their assigned company, so they can only access the company's data. This is performed through the rules section in Firestore. With this setup, the data is stored in an efficient and secure way.

6 Graphical User Interface

The application is composed of four pages:

- Login
- Robot list

    - The list of the robots owned by the company of the user.
- Robot information
    - Detailed information about a robot from the robot list.
- Logs
    - The list of the errors/messages from the robots owned by the company of the user.

6.1 Login Page

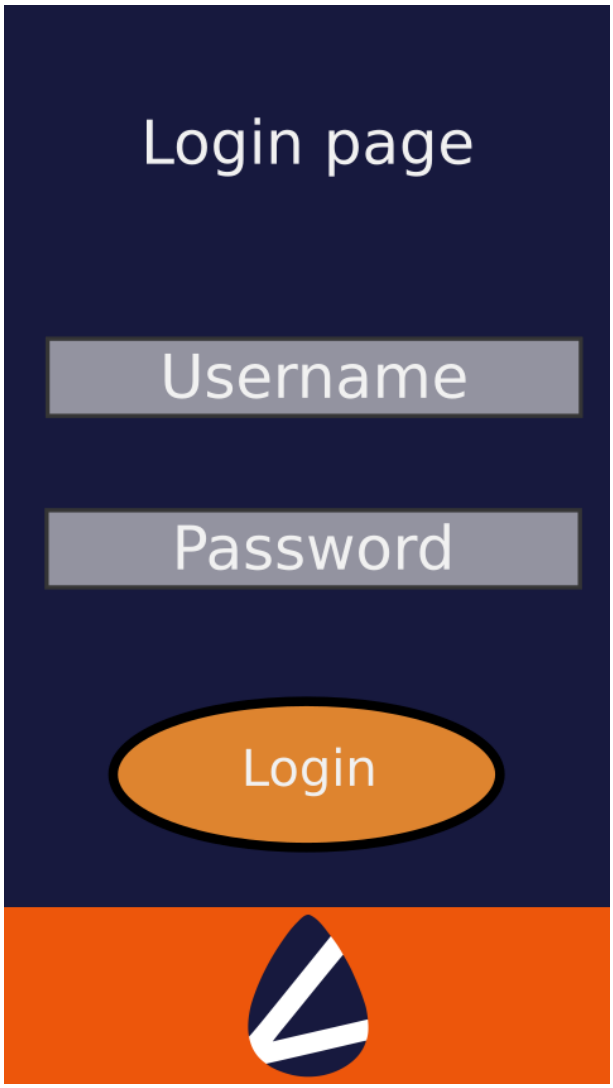

Figure 19. Login page.

The login page (figure 19) will be seen the first time the user launches the application. There are two text fields to write the username and the password. Once the user has entered a correct username and password, he/she will enter the robot list page.

6.2 Robot list

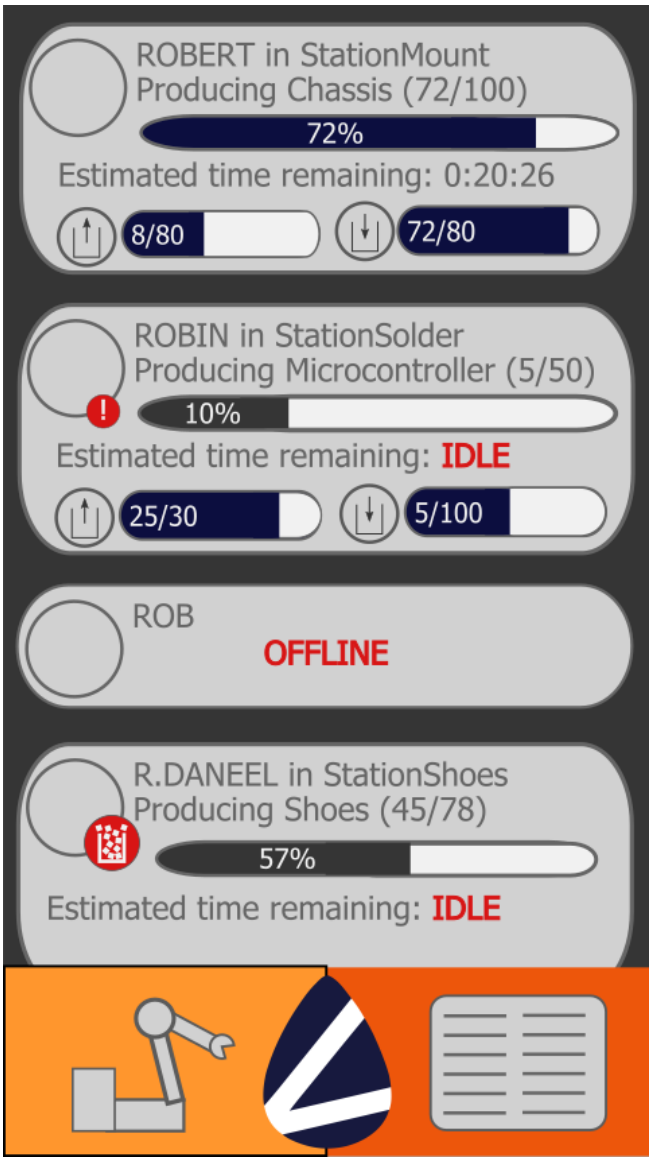

Figure 20. Robot list page.

The robot list page (figure 20) lists, as its name says, the different robots of the company. This is the page any users already logged in will first see when they start the application again.

This page displays the robots owned by the company, and some essential information like the progress of the task of the robot, status of the input and output pallet and, the estimated remaining time of the task. If there are any errors, they will be displayed as a bubble next to the picture of the robot as the example above. The bubble can be a simple exclamation mark or a more descriptive icon for the most common errors (in the example above, R.DANEEL have a special icon depicting the output pallet being full).

In the bottom, there are three buttons: the robot button which takes us to the robot list page (the actual page), the log button which takes us to the log page and the logo of the company, which opens a small menu that allows the user to logout or to access the settings.

If the user taps on any robot of the list, it will open the detail page of this robot.

6.3 Robot Details

Figure 21. Detailed robot information page.

The detail page (figure 21**)** of the robot displays the same information as in the robot list but adds more detailed data. The detailed data include the average cycle time, the current speed, the estimation of the ending of the task and additional data. It will also display some statistics about the robot, like the average numbers of product produced per day.

6.4 Log page

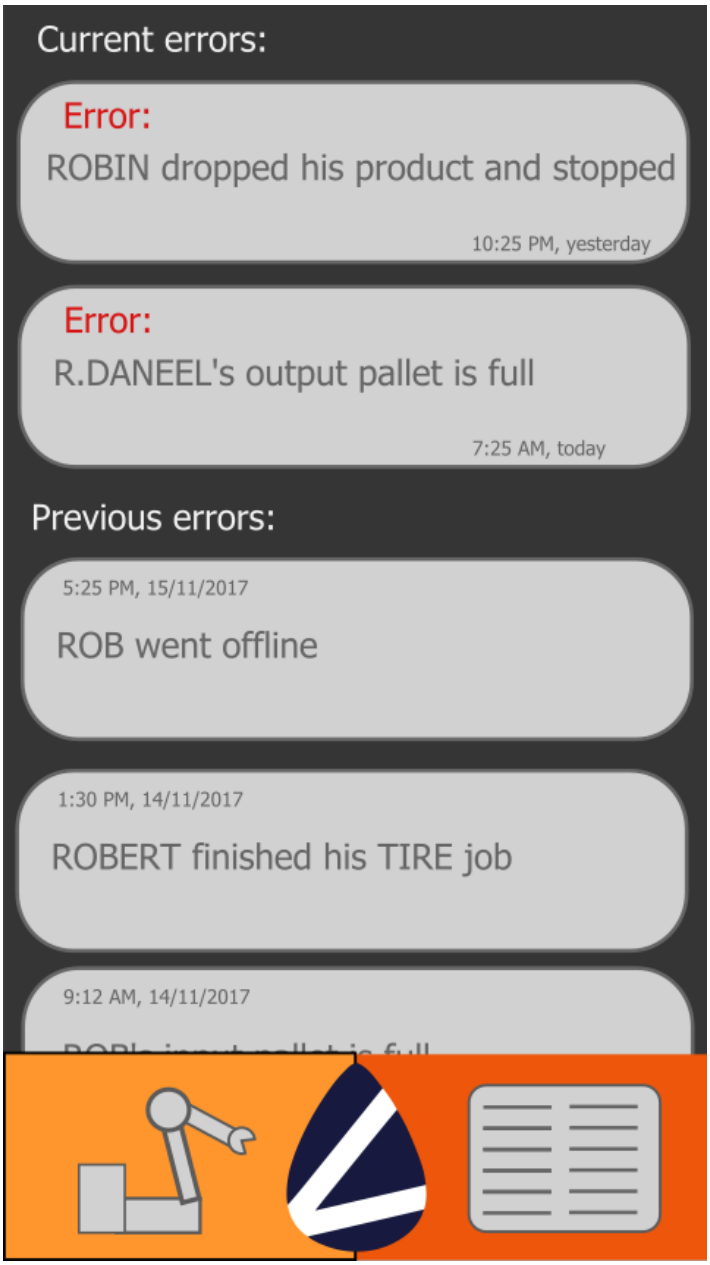

Figure 22. Robot Log page.

The log page displays the different events the robots sends (figure 22). This is the quickest to see the important events and when they happened. The events include warnings like when the input or output pallet is full but also include errors like robot failures or environment hazards (someone standing next to the robot forcing it to stop).

7 References

[1] Google LLC, "Platform Versions," [Online]. Available: https://developer.android.com/about/dashboards/index.html#Platform. [Accessed 28 November 2017].