**MINI EXPLORER PROJECT**

**END TERM REPORT**

**Duc Nguyen, Jordan Pulido**

**TABLE OF CONTENTS**

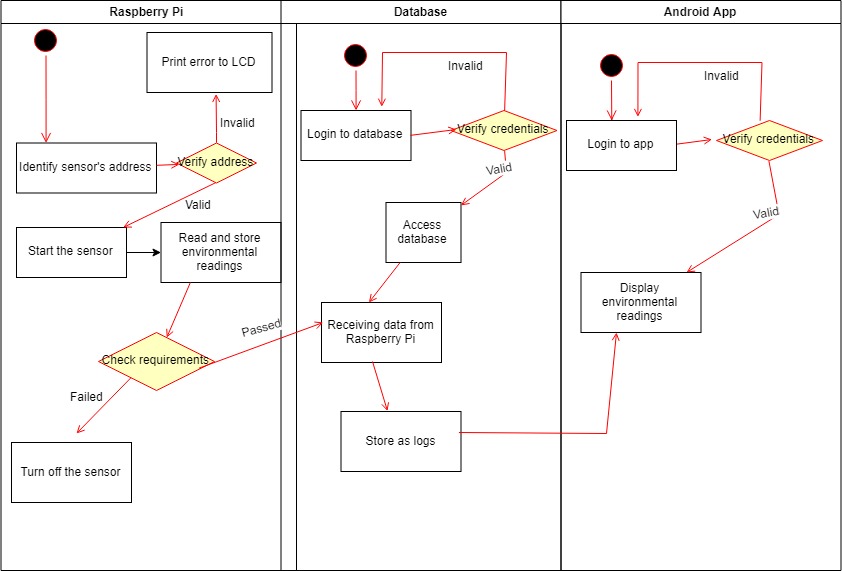
1. Introduction
2. Software Design and Specifications
3. Data Structure
4. Database Design
5. Hardware Design to Database Structure Implementation Plan
6. Test cases
7. Android Components
8. Contribution
9. Conclusion
10. References

**Abstract**

Taking advantage of technology to perform tasks that seem harmful to humans is what drives our creativity and innovation forward. Such one task is to evaluate environment conditions in certain surroundings. Some places are unreachable and thus, require small devices with enough functionalities and specifications to reach such locations. Mini Explorer Project is born to perform such activities: Approach small places (with slippery surfaces) and obtain environment readings from them. This project includes extensive research on certain hardware, software and database integration from multiple different sources. For the purpose of this report, we will only focus mainly on the software aspect.

**I. Introduction**

Mini Explorer consists of three main parts: Hardware, software, and database. The hardware backbone is the Raspberry Pi 3B+ Model, which acts as a central processing unit for every function. Along with it are three sensors: HDC1008 (Temperature and Humidity Sensor), PCA9685 (16-channel 12-bit PWM/Servo driver) and VL53L0X (Time of Flight Distance Sensor). The software is entirely written on Android Project, as the Mini Explorer is expected to be entirely controlled through the phone via Bluetooth. The database is stored on Firebase - a cloud service by Google. This is where all users’ information, environmental readings, device settings and logs are stored. All three main parts are interconnected and closely related. For visual purposes, this UML diagram will give a more thorough understanding of the whole system.



*Figure 1 - UML Diagram of Mini Explorer*

**II. Software Design and Specifications**

This is the main focus of the report, as it describes thoroughly the software aspect of the project.

II.1 Software Design

The software is to be developed by two people with no prior experience in mobile programming / development within nearly three months. With such resources, the software design is fairly simple. It consists of two main elements: Java Front-end / Back-end development and database development. While they are not entirely dependent on each other, they have the same priority level, as both of them are crucial to the application and can be developed at the same time.

The Java Front-end determines the appearance of all elements contained within the app, and the user experience the app is expected to bring. It includes, but not limited to, creating and manipulating icons, layouts, text, buttons. As of current, the app contains 8 different screens, each with its own way of design so as to serve its functions. The app also uses 13 different, specifically-customized icons ; 41 text strings, available in both English and French.

The Java Back-end determines how the app is going to be executed by the users. It consists of, but not limited to, authentication, intents, events and is used in all screens. The screens and their functionalities are:

1. Splash screen: Acts as a launcher, will go to the login screen after 0.5 seconds of showing the app’s icon.

2. Login screen: Prompts the user to enter their email and password, and then check with the database. If the typed information is not correct, then the user cannot log in. Also has a intent to display the register screen if the user wished to.   
 3. Registration screen: Allows users to register with email and password. Users must enter a legit email format, and confirm their password to register successfully. Data is then sent to database.

4. Main menu screen: Shows all other main screens that users can go to. Clicking on the live support will not take you to a new screen, but instead, redirect you to the call activity of your phone.

5. Reading screen: Displays the temperature and humidity retrieved from database

6. Explorer screen: Shows the car’s control and power state. Users can control the car’s movement and power in this screen.

7. Setting screen: Displays the language option, and certain settings that can be applied to the hardware (low-battery alert, scan intervals, and auto-shutdown timer).

8. About us screen: Displays the developers’ names.

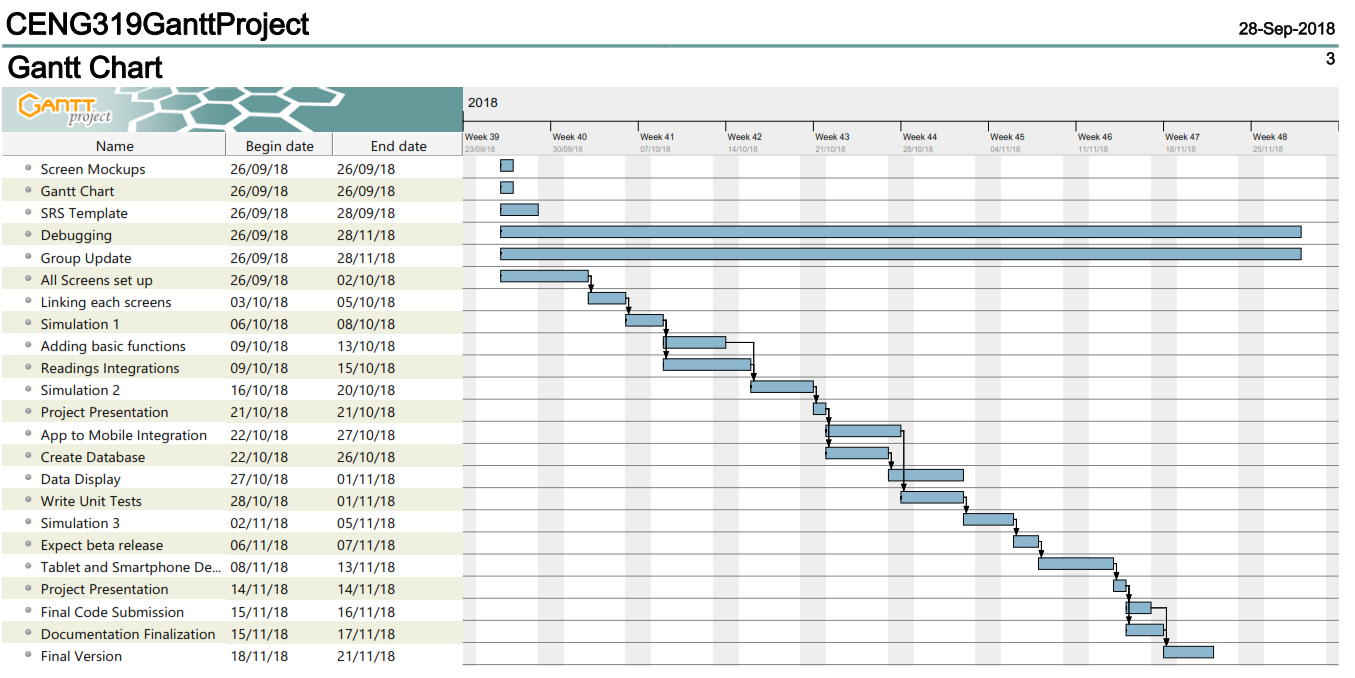
Listing and purposes of functions/methods being used will be discussed in part VIII.

The database development is started on the second-half of the project timeline. The database is stored on Firebase - a free Google cloud service - which is integrated in Android Studio as well. The database design will also be discussed on part V.

The requirements for the software are rather straightforward. The software should not crash or contain any bugs at all times, is expected to deliver its full functionalities, which include:

* Execute all tasks properly and flawlessly without bugs / crashes.
* Being able to register and login / logout users.
* Having access to all screens.
* Being able to retrieve environment readings from database.
* Being able to control the Mini Explorer (its power state, its movement and directions) from the maximum distance of 70 meters.
* Being able to adjust certain settings to hardware (Low battery alert, scan intervals, auto shut-down)
* Being able to call live support.

Overall, the whole software design and development can be demonstrated through this Gantt Chart. Some tasks are done much quicker, while some require more time and effort, such as working with the database.



*Figure 2 - Project Schedule Timeline*

II.2 Software Specifications

The app is intended to run on most Android devices, ranging from API 21 to API 28, which covers more than half of the Android devices today. Therefore, the app implements the Android appcompat, design and layout with 28.0.0 version. It also implements Firebase libraries with version 16.0.0. The app supports different screen sizes, so the only element to consider is the mobile’s API version.

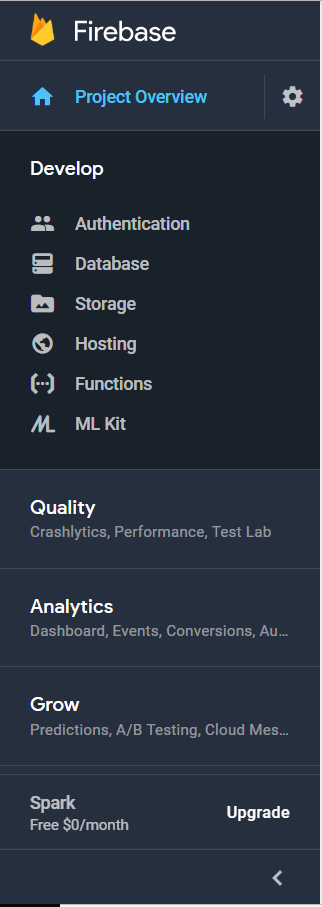
**III. Data Structure**

The project will use two main data structures: Environment readings and device status. Currently, the environment readings data structure consists of only temperature and humidity. It is a distinguish Java class in the project. In the future, it is expected to contain the time logs of each reading, and update logs of the device’s status as well. The second data structure should hold the update log of device’s status, and other settings (low battery alert, scan intervals, auto shutdown timer). These data are held in the database, but is not yet to be constructed as a data structure.

**IV. Database Design**

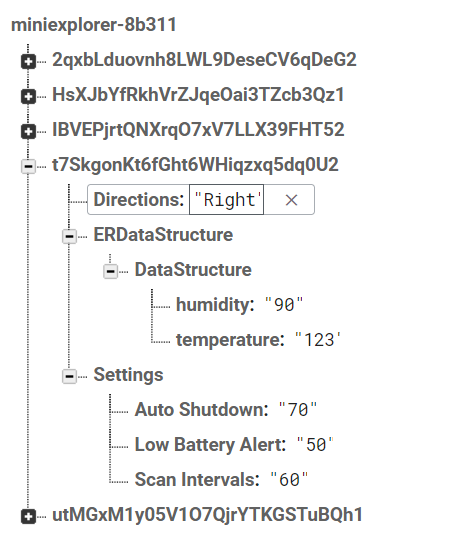
The database is designed and developed in Firebase platform. In order to use this technology, a Google Mail is required, so we decided to create an unique email solely for this project. Firebase has many utilities, but as of now, only Authentication and Database are used.

Authentication is where all the users’ email and passwords are stored. As soon as a registration is made successfully, they will be sent to Firebase and assigned with their unique user IDs. This is very beneficial to monitor individual’s activities and readings. Whenever a login attempt is made from the app, it will trigger the methods in Login Screen to scan for that specific email and password in here. Authentication also has the email address verification feature, which will be used in the future.



*Figure 3 - Snapshot of Firebase capabilities*

The Database section starts with the name of the project. Each next entry is assigned an unique User ID. Each User ID contains the necessary data. The first entry is “Directions” , which is to control the car’s movement. The second entry is a data structure called “ERDataStructure”, which contains the environment readings (temperature and humidity). The last entry is “Settings”, which consists of “Low battery alert”, “Scan intervals” and “Auto Shutdown”.



*Figure 4 - A snapshot of the database*

**V. Hardware Design to Database Implementation Plan**

The hardware is closely related to the database, and is expected to react spontaneously when there’s a change in the database and vice versa.

Firstly, whenever the sensor receives new environment readings, the corresponding entries in the database must be updated immediately. This should be of utmost importance, and can be implemented by adding Firebase functions to the HDC1008 python source code. More research on Firebase integration to python in Raspberry Pi is required.

Secondly, the Raspberry Pi should be able to read the new settings written in the Settings section, and then execute it. Different settings require alterations to different part of the hardware. For example, “Auto Shutdown” and “Low Battery Alert” will order the Pi to constantly check for its power value, and to turn off the Pi after a specific amount of time. This needs research and understanding how the Raspberry Pi functions. The “Scan Intervals”, however, requires minor tweaks only to the HDC1008 sensor.

**VI. Test Cases of Project**

Test cases are conducted on main two different parts: Hardware and Software + Database.

VI.1 Hardware Test Cases

The test cases on Hardware are rather simple and straightforward. Our goal is to ensure that the hardware is properly turned on and no physical damages are done.

VI.1.a Raspberry Pi

* Ensure that the Raspberry Pi can be turned on and read the SD card.
* Ensure that the Raspberry Pi can detect the necessary sensors by using the command “sudo i2cdetect -y 1”
* The Raspberry Pi should not have any scratches, broken or damaged components that may potentially harm the whole product.
* The Raspberry Pi can use the external battery.

VI.1.b Sensors

* The sensors are put in proper Raspberry Pi pinouts.
* The Raspberry Pi is able to execute the sensors’ code and programs without any errors.
* The sensors are able to function properly based on the code’s intentions.

VI.2 Software + Database Test Cases

The software and database tests cases are conducted on each screen, as they require different test methods.

VI.2.a Registration Screen

To test the process of the registration screen, adding database was implemented for authentication functionality. Its role for the registration screen is to add the typed email and password to Firebase, if the following conditions are met:

* Checks if email is in valid format.
* Password verification (notifies if password is not the same)

VI.2.b Login Screen

Logging into the database requires a registered account through Firebase. If the email is not recognized, the app will notify the user that the email does not exist, then will recommend to register a new account. If multiple accounts are labelled as the same username, unique IDs will be specified for each individual to determine one’s identity.

VI.2.c Main Menu Screen

Once successfully logged in, the main menu will be brought upon the user. In the main menu, different sections will be accessible by touching or clicking the corresponding icons. Provided in the main menu are:

* Environmental readings
* Settings
* About us
* Explorer Status
* Live support
* Logout

Intents, onClick, and findViewByID methods were implemented in the main menu to access the different activities within the app.

Live support function can be tested by pressing on the icon and checked if the phone number of the support team will show up.

VI.2.d Environmental Readings

Real-time readings would receive temperature and humidity integers from the sensors through the database. The sensors would gather these values from the temperature surrounding the Mini Explorer. From the database, the given values will be outputted in the mobile application.

VI.2.e Settings

The settings activity opens up to 3 options available for the user to input. “Low Battery Alert” asks what percentage the user wants to notify the phone when the Mini Explorer should be recharged. “Scan Intervals” asks how many seconds would the Mini Explorer detect temperature surrounding the vehicle. The “Auto-Shutdown” setting asks how many seconds would the RC car shut down after being idle for a specific amount of time.

VI.2.f Explorer Status

Explorer Status acts as the remote control for the Mini Explorer. Given up, down, left, and right arrows, determines the direction the RC car’s location. In the activity, there is also an option for the power state of the Mini Explorer. This feature determines when the power of the remote control car will be turned off or on. When an arrow key is pressed, inputs will be sent into the database, and from the database the information will gather to the remote control car, causing it to move. The remote-control feature will have Bluetooth implemented between the mobile device and Mini Explorer.

**VII. Androids Components**

Many Android components and libraries are used in this project for different purposes.

Java / Android libraries used:

* android.content.Context;
* android.support.annotation.NonNull;
* android.support.v7.app.ActionBar;
* android.support.v7.app.AppCompatActivity;
* android.os.Bundle;
* android.util.Log;
* android.util.Patterns;
* android.view.Menu;
* android.view.MenuItem;
* android.content.Intent;
* android.view.View;
* android.widget.EditText;
* android.widget.Toast;
* android.gms.tasks.OnCompleteListener;
* android.gms.tasks.Task;

Major methods used:

* findViewById (int resourceId): This is used to locate and interact with every element in the project. This includes, but not limited to, buttons, texts, images.
* toastMessage(applicationContext, text, duration): Is used to announce the result of certain scenarios.
* Intent (this, viewContacts.class): Is used to switch from one screen to another.

Firebase libraries used:

* firebase.auth.AuthResult;
* firebase.auth.FirebaseAuth;
* firebase.auth.FirebaseAuthUserCollisionException;
* firebase.auth.FirebaseAuth;
* firebase.auth.FirebaseUser;
* firebase.database.DataSnapshot;
* firebase.database.DatabaseError;
* firebase.database.DatabaseReference;
* firebase.database.FirebaseDatabase;
* firebase.database.ValueEventListener;

Major methods used:

* signInWithEmailAndPassword: Is used to give the user the access to use the app after using the correct, registered email and password.
* getCurrentUser(): To retrieve and manipulate the data of the current user being accessed.

**VIII. Contribution**

Duc Nguyen - Main Idea, Gantt Chart, Main Menu Layout, Icon Designs, Database Writing and Reading.

Jordan Pulido – SRS Documentation, Registration Screen, Login Screen, Authentication, Live Support, Language.

**IX. Conclusion**

Ultimately, the Mini Explorer achieved its goal and followed through the schedule from the Gantt chart provided. Over the course of 2 months, the planning of the project, the documentation, and the programming procedure of the application sometimes did not meet up with each deadlines, and had delays throughout the process. The database task which involved reading and writing from the application to the database itself took longer than expected, resulting a postponement towards other tasks. The authentication and reading process of the database encountered errors, however the errors were within task to solve. The writing procedure from the application to the database encountered problems where inputting values from the app would not appear on the database. Although the project brought about its purpose, the Bluetooth feature between the RC car and the mobile device has not yet been implemented. For the additional feature of Bluetooth, the remote control car would need to be built itself. In the future, more translations in languages would need to be added for the user’s convenience. Suggested languages that would be put into the application are German and Spanish. Due to language changes, the sizes of certain words would change which may affect the layouts of the activities. Therefore, the layouts of the screens would need to be adjusted so that all orientations fit to the mobile device’s screen. To further enhance the user’s security, email verification would be implemented upon registering an account.

**X. References**

[1]: Android Developers. (n.d.). Retrieved from

<https://developer.android.com/>

[2]: AustinCENG - Overview. (n.d.). Retrieved from

<https://github.com/AustinCENG>

[3]: Mitchtabian. (2017, February 05). Mitchabian/Firebase-Database-REAL-TIME-. Retrieved from

<https://github.com/mitchtabian/Firebase-Database-REAL-TIME->

[4]: SLICE YOUR WIREFRAME WORK IN HALF. (n.d.). Retrieved from

<https://ninjamock.com/>

[5]: Read and Write Data on the Web  | Firebase Realtime Database |  Firebase. (n.d.). Retrieved from <https://firebase.google.com/docs/database/web/read-and-write>

[6]: Manage Users in Firebase  |  Firebase. (n.d.). Retrieved from <https://firebase.google.com/docs/auth/web/manage-users>