OULU UNIVERSITY OF APPLIED SCIENCES

Ashrafuz Zaman Noor

GATEWAY FOR BLUETOOTH COMMUNICATION IN ANDROID

GATEWAY FOR BLUETOOTH COMMUNICATION IN ANDROID

Ashrafuz Zaman Noor
Bachelor's Thesis
Spring 2015
Degree programme in information technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree programme: Information Technology

Author: Ashrafuz Zaman Noor
Title: Gateway for Bluetooth Communication in Android
Supervisor: Kari Laitinen
Term and year of completion: Spring 2015 Number of pages: 53 + 4

This Bachelor's thesis provides a detailed overview of the Gateway software which acts as an intermediary between Ceruus' manufactured IoT devices called Catchers and the Ceruus' cloud system named IoLiving. The idea of the Gateway software is to collect data from the Catchers and upload the data into the cloud using a smartphone as a medium.

The thesis describes the main theory behind each major functionality of the Gateway and how the design was done based on the theory. Important issues such as Internet of Things, Bluetooth Low Energy and Android platform are also discussed in the theoretical part of the thesis.

The implementation part of the thesis covers how each of the major use cases of the Gateway were handled programmatically. The testing phase was a repetitive process and the results were used to fine tune the Gateway even further.

Tools that were used for developing the Gateway software includes Star UML, Eclipse with the ADT plugin and Android SDK.

The outcome of this thesis is a fully functional Gateway software for Android that can be downloaded from the Google Play Store under the name 'IoLiving' app. The application can successfully retrieve data from the Catcher and upload it to the IoLiving cloud.

In conclusion, this thesis states that Gateway software, with its continuous background process is unique in nature. Gateway can connect up to 20 Catchers consistently and disconnect from them automatically. It is not a typical case that so many devices are handled automatically by background services. Furthermore, it is possible to add new features by tweaking the design a little.

Keywords: Android application, Internet of Things, Bluetooth Low Energy, Gateway

CONTENTS

ABBREVIATIONS

Terms	**Definitions**
BLE	Bluetooth Low Energy
IoT	Internet of Things
IoLiving	Internet of Living
EEPROM	Electrically Erasable Programmable Read-Only Memory
GATT	Generic Attribute Profile
SDK	Software Development Kit
HTTP	Hypertext Transfer Protocol
UI	User Interface
API	Application Program Interface
Sync	Synchronisation
CPBP	Ceruus Proprietary Broadcast Protocol

1 INTRODUCTION

The time has come that we get quantifiable data about our environment instead of solely trusting our senses. The feeling of ‘warm’ or ‘cold’ should rather be replaced by what the temperature is in centigrade scale; the rather vague feeling of moist or dry can be replaced with the exact relative humidity of the day. At this age of technology, it is about time to get more out of our gadgets, and make wiser choices based on solid and more reliable information than our senses.

The Gateway software for the Bluetooth communication in Android lets us do exactly that. It is the intermediate between sensors called ‘Catchers’, which measure and transmit environmental data. The Gateway software collects these data in our cell phone and uploads it to a cloud network for a storage and further processing. The software is designed for a Finnish startup named Ceruus Oy.

Ceruus Oy is a company which develops reliable, new mobile Internet services for facilitating everyday household tasks in an affordable cost. The service named 'IoLiving’ and the Gateway application enables web-based temperature monitoring and energy efficiency optimisation. Catcher, a small device invented by Ceruus, creates a secure web connection through a smartphone or a tablet while the Catcher is in vicinity.

The service offers a solution to a major challenge in the temperature monitoring and use of electricity. The Room-specific temperature can now be monitored so that energy savings can reach 15-25 % in a detached house with electric heating. Should customers have a spot agreement with their energy company, the savings can be substantially larger due to the service’s capability to alert the user while the temperature reaches a certain level. A usable interface allows a complete temperature monitoring. (Ceruus internal documents, date of retrieval 6.4.2015).

IoLiving service can also be used in a restaurant for monitoring the food temperature. Starting from the storage of the food products and until they are served to the customers, the service can provide a consistent temperature logging and thus it can provide a great solution for the quality control of the food.

IoLiving service also provides a solution for motion monitoring. This offers an affordable choice for a tracking activity compared to an expensive smartwatch or an activity wristbands. Using this service, users can keep track of their daily workout target, facilitating a better health. For seniors living alone, family members or care home staff can monitor the movement and can check on them, if the module has not moved for a certain period.

Ceruus is also introducing a handy Internet-based carbon dioxide and moisture module for Catcher which can be placed to a number of places unobtrusively, and which allows monitoring the air carbon dioxide and moisture in the room.

‘There are typically several smartphones or tablets in a household. IoLiving system does not require separate wireless base stations, because the connection to the Catcher is maintained directly from the mobile device. Therefore, the price of the service and related hardware is substantially lower in comparison to traditional wireless solutions.’ (Ceruus internal documents, date of retrieval 6.4.2015).

There are several modules of Catchers and each module is designed to collect a specific type of data. For example, the temperature module of Catcher collects temperature data whereas the movement tracking module collects only movement data. Gateway acts as a common entry point for the data from all modules of Catchers.

The Gateway is unique in nature as, once initiated, it continues to run for an indefinite amount of time in the background of the Android phone and can synchronise data from up to 20 Catchers consecutively. It is not typical that so many devices are actively connected to one phone. Thus, it presents a new set of challenges to handle the connection management.

An even bigger challenge is to handle different Bluetooth stacks of Android. Based on different Android versions and manufacturers, the Bluetooth stack that handles the functioning of the Bluetooth varies, which in turn needs a special consideration so that Gateway works in a similar manner in most of the smartphones.

In order to better understand how the Gateway is designed, we need to understand how the whole system works. Figure 1 shows an overview of the system.

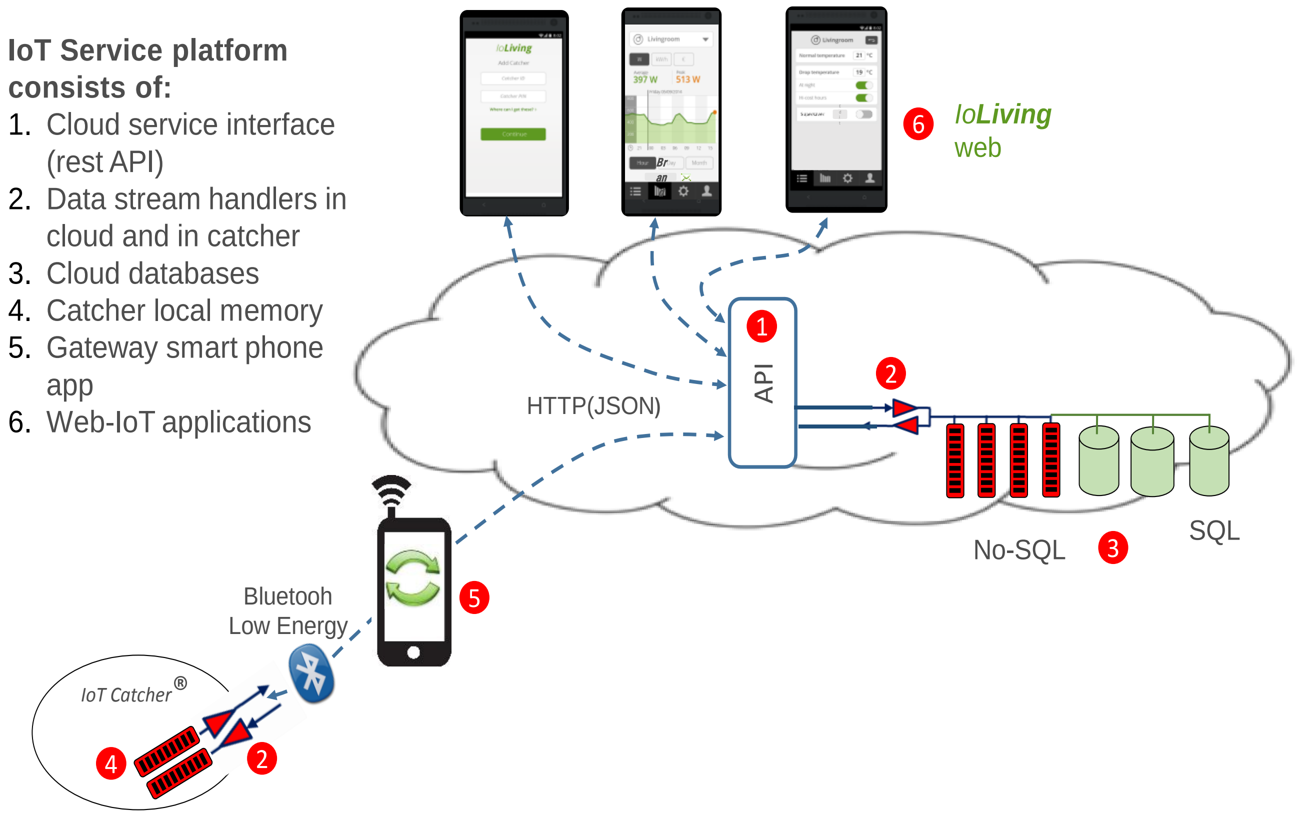

IoT Service platform consists of:
1. Cloud service interface (rest API)
2. Data stream handlers in cloud and in catcher
3. Cloud databases
4. Catcher local memory
5. Gateway smart phone app
6. Web-IoT applications
IoLiving web
6
1
API
2
HTTP(JSON)
No-SQL
3
SQL
5
Bluetooh Low Energy
IoT Catcher®
4
2

FIGURE 1. IoLiving System Overview (Ceruus internal documents, date of retrieval 6.4.2015)

The whole system can be divided into three parts:

i. Information collection: Small devices called Catchers handle the collection of data.

ii. Information transfer: Gateway installed in Android smartphone handles the transfer of data from Catcher to the IoLiving cloud service. Gateway retrieves data from Catcher either by scanning for the Catcher when it is in the advertising mode or by establishing a connection and requesting the stored data from the EEPROM of the Catcher in the form of notifications.

The Gateway also enables the Android phone to establish a connection with the IoLiving cloud server and then to upload the collected data for a storage and further processing. An active internet connection is required for server communication to take place.

iii. Information storage and handling: When Gateway uploads information to the cloud server, it is stored in the database in a bulk format. This stored information is then further processed to make it readable and presentable to the end user.

The main objective of this thesis is to provide a comprehensive overview of the current state of the Gateway software. However, this document does not cover detailed solutions to the presented Android Bluetooth stack problem or connection management system but rather this document presents the user with a chance to understand how the Gateway software is designed to work in harmony with the whole IoLiving system in typical situations.

2 USED TECHNOLOGIES

This chapter provides information about some of the technologies used in the project and the core concept of the Gateway software.

2.1 Internet of Things

The idea of data exchange over a network between uniquely identifiable physical objects, embedded with electronics, software, sensors and connectivity with the minimal human interaction is believed to be the next form of revolution in technology after the Internet. Internet of Things is the term that defines this idea and has evolved from the convergence of wireless-technologies, micro-electro-mechanical systems and the Internet. (Rouse, Date of retrieval 6.4.2015 and Wikipedia, date of retrieval 6.4.2015).

2.2 Android

This section provides a general overview of the used features of Android to develop the Gateway software.

2.2.1 Configuration

In order to start an app component in Android, the Android system must know that the component exists. The Android platform provides a configuration file named AndroidManifest.xml to configure the application components for an optimal functioning.

In addition to declaring app components, the manifest file identifies any user permission the app requires, declares API Level required by the app, declares hardware and software features used or required by the app and other configuration related tasks. (Android Developers, date of retrieval 8.4.2015).

2.2.2 Application components

In Android, application components can be considered as building blocks of an app and can be categorised into four main types:

Activities:

In Android, interaction with the user mostly occurs through activities. Therefore, an activity is a class that presents a user a screen with which the user can interact and perform a certain task. The Android platform assigns a frame by means of a layout to each activity in order to build the user interface on top of the frame.

An activity lifecycle is based on a series of callback methods and these callback methods are responsible for starting, stopping and running an activity.

Services:

As long running operations tend to affect the responsiveness of the UI thread, the Android platform provides components that run in a separate thread. It can perform long running operations in the background without sacrificing the efficiency of the UI thread. These components, referred to as services, can essentially take a form of a bound service or a started service.

A started service can be started by an application component by calling startService() and it can run indefinitely in the background whereas a bound service can be bound when an application component is bound to it by calling bindService().It will run as long as the application component bound to it runs.

Content providers:

The app data in Android is managed by the content provider which allows a data storage in the file system, SQLite database, on the web or in any other persistent storage location the app has access to.

Broadcast receivers

Broadcast receivers respond to system-wide broadcast announcements and thus act as a communication module between applications or within the system itself. For example, if a broadcast announces that the battery is low, it can in turn be used as a means to reduce the power consumption by turning off Bluetooth.

Android provides a rich framework that allows developers to build innovative apps for mobile devices in a Java language environment. Android is one of the most popular platforms for a smartphone and it provides a rich documentation and a technical support. It is up to date and equipped with the latest features. Moreover, from API level 18 and onwards, it provides a full functional support for BLE communication which makes this platform particularly suitable for this project. (Android Developers, date of retrieval 8.4.2015).

2.3 Bluetooth Low Energy

Bluetooth low energy (BLE), as the name suggests, is a technology for wireless communication that uses a considerably less amount of energy compared to Bluetooth classic while providing a very similar communication range. (Gibbele, date of retrieval 9.4.2015).

However, it is crucial that a Bluetooth low energy module remains in a low-power mode for most of its lifetime and makes any connection for a short amount of time to achieve the low energy consumption from the module.

In context of the IoLiving BLE system, the communication process consists of one central and one peripheral role. The peripheral role is played by a Ceruus manufactured device named Catcher, while in the central role is any Android device where the Gateway application is installed.

The peripheral, Catcher acts as a server and advertises ‘data’ to the air interface periodically which is detected by the central. This periodic advertisement helps to keep the power consumption at minimum. The central, Gateway acts as a client and scans the air interface for an available Catcher periodically. Figure 2 shows how this process works:

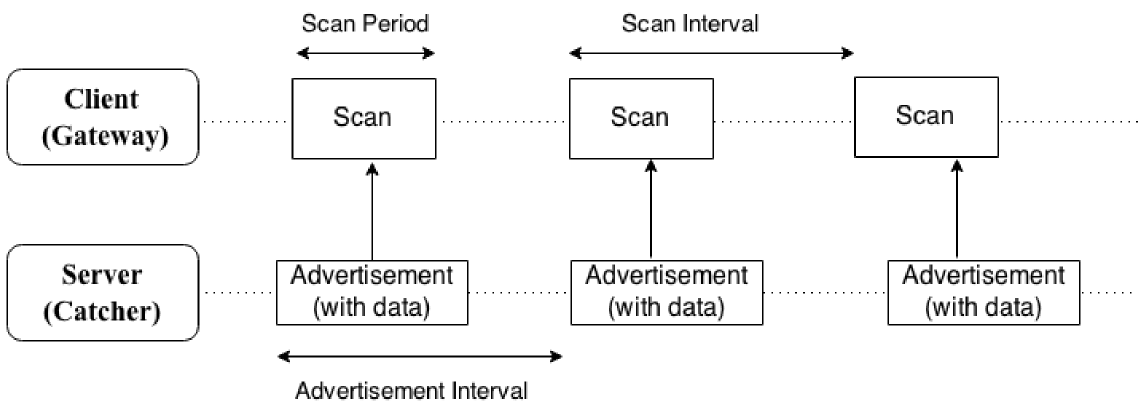


FIGURE 2. Advertising data transfer (Connect Blue, date of retrieval 13.04.2015)

Measurement data is saved to EEPROM of the Catcher and can be retrieved by establishing a connection. The broadcast state indicates that the peripheral is ready for a connection. And when the central intends to make a connection, it initiates a connection request. Figure 3 illustrates the connection process:

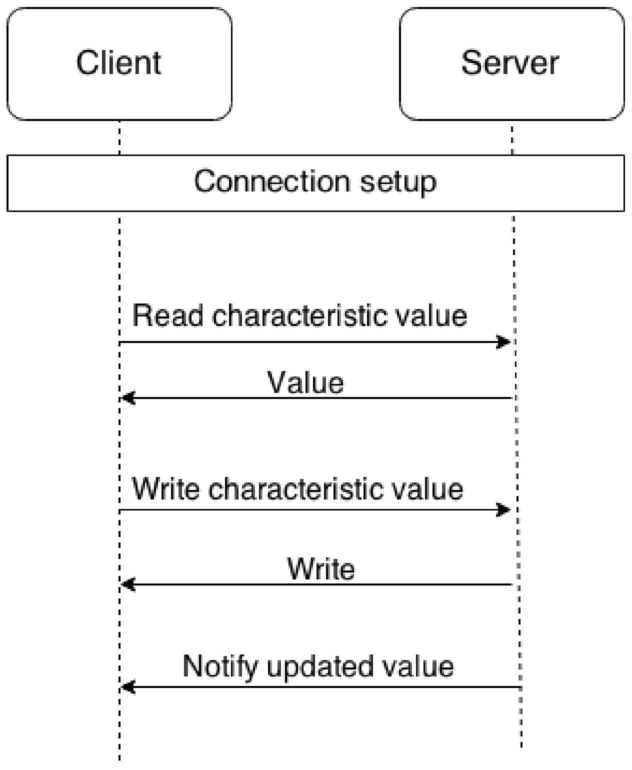


FIGURE 3. Measurement data transfer (Connect Blue, date of retrieval 13.04.2015)

‘The GATT profile of the Catcher specifies the structure in which data is exchanged. The top level in a hierarchy is a profile which contains several services including a proprietary service for the Catcher. Each service includes one or more characteristics. A characteristic has a value and may contain optional information about the value.’ (Bluetooth Developer portal, date of retrieval 9.4.2015 and Connect Blue, date of retrieval 13.04.2015).

2.4 Catcher

Catcher is Ceruus’ invention of Bluetooth low-energy (BLE) technology equipped with various sensors to create a different way of providing environmental information such as temperature, humidity and motion based data to smartphones and web.

There are several module of Catchers. For example, ‘IoLiving T’ is a Catcher module that measures temperature, ‘IoLiving M’ measures motion. Based on the module, the size may vary but the colour remains white for every type of Catchers. A waterproof casing enables a Catcher to be installed outdoor and underwater. Figure 4 shows how Catchers look like.

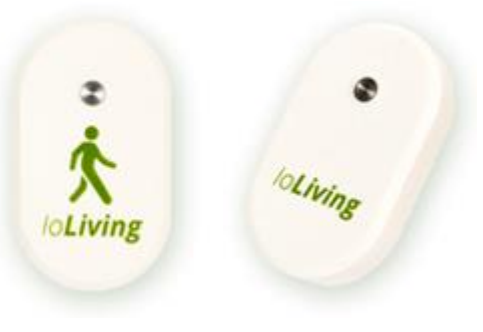


FIGURE 4. Catchers (IoLiving, Date of retrieval 13.04.2015)

Catchers collect data from the environment and periodically advertise it in the form of packets. Ceruus Proprietary Broadcast Protocol (CPBP) uses the manufacture specific fields of the advertising packets of BLE through the advertising channels. It is aimed at the fast data transfer from Catcher to Gateway. The broadcast data frame is 27 octet long. Broadcast frame is a system parameter and it can be set in a range of 20 milliseconds to 10 seconds.

Catcher also stores the measurement data from sensors to an external EEPROM memory chip. This data can be collected by establishing a connection between the Catcher and the Gateway. (Ceruus internal documents, date of retrieval 2.03.2015).

2.5 Project management

The scope of this project extends from just an Android app to a fully-functional system where each part of the system works in harmony with the others. Management of such a large scale project development needs to fulfil certain criteria such as collaboration between self-organising and cross-functional teams. Hence Scrum, an iterative and incremental agile software development methodology, was the most suitable development method. Also, this method promotes adaptive planning and continuous improvement, and encourages a rapid and flexible response to change. (Wikipedia, date of retrieval 28.3.2015).

However, this document only focuses on the Android app part of the system and other associated parts are only described in order to facilitate the understanding of the Android app. Also, the Bluetooth connection mechanism in Android comes as a supporting material to better understand where the actual data comes from.

3 GATEWAY SOFTWARE DESIGN

This chapter describes the detailed design and architecture of the Android gateway software for the multi device BLE communication.

3.1 Gateway Architecture

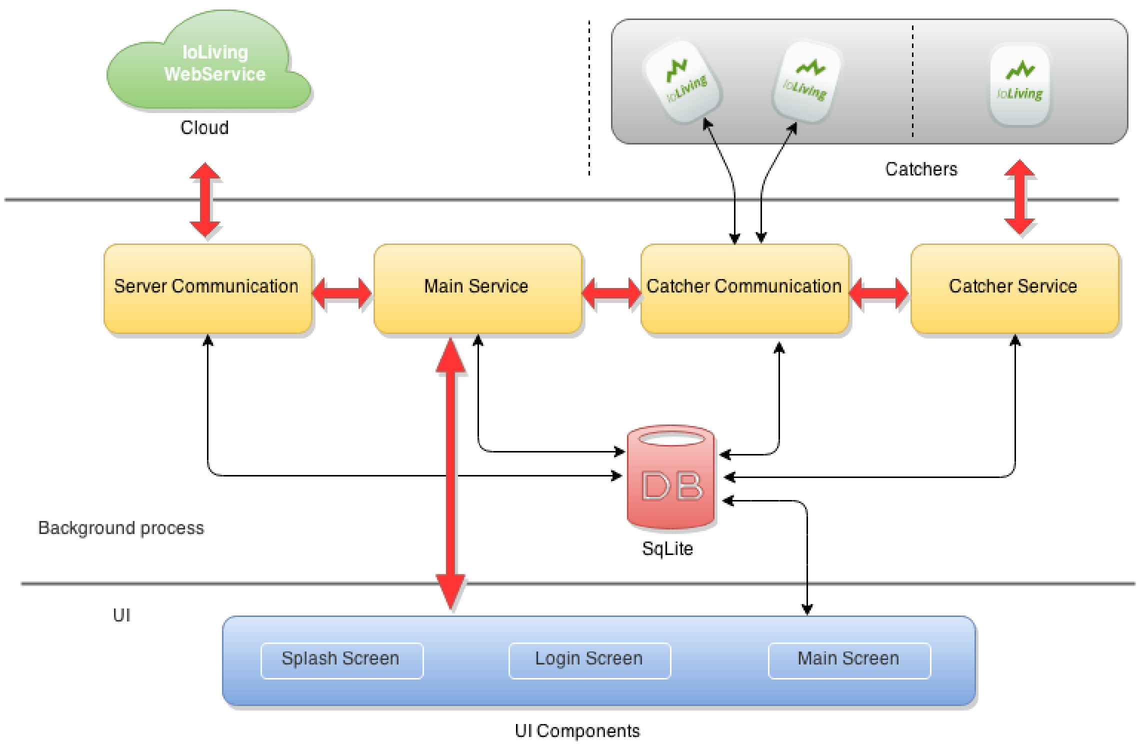


FIGURE 5. Architecture of Gateway (Noor, A.2015)

The Gateway software can be compartmentalised into two parts. The UI where interaction with the user occurs and the background processes. Figure 5 shows the main architecture of the Gateway software.

The Gateway application consists of three user interfaces namely Splash Screen, Login Screen and Main Screen. These activities are loosely bound to each other.

The Gateway starts with Splash Screen which decides whether to open the Login Screen or the Main screen based on the presence of the authentication token in the SQLite database of the application in the phone memory. This is illustrated in Figure 6. This feature facilitates the usability by minimising the necessity to login each time the user opens the Gateway and allows the user to maintain a steady session and an uninterrupted use of the IoLiving services.

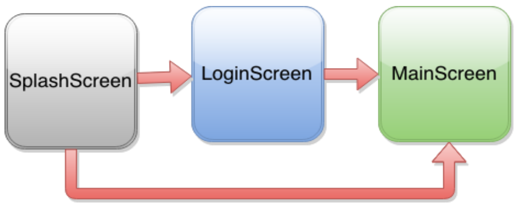


FIGURE 6. UI flow of Gateway (Noor, A.2015)

For the very first time, when the user opens Gateway, there is no authentication token present. So it opens the Login Screen which holds the login form. Figure 7 shows the Login Screen.

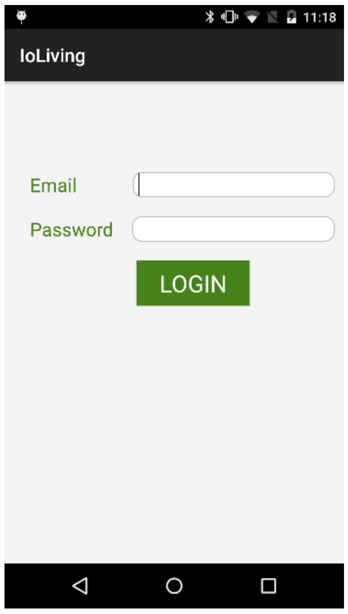


FIGURE 7. Login Screen (Noor, A.2015)

Login form has two fields, namely, email and password and a login button. The user must fill in the two fields with correct information in order to login. When the user presses the login button, the Main Service of the Gateway sends an authentication request to the IoLiving web service via the server communication module. The server then sends 200 in response along with the authentication token and the number of Catchers the user has access to. The server communication module then saves the authentication token and the list of Catchers to the database and sends a message to the Login Screen that the login is successful. The Login Screen then sets the Main Service to run in a loop until it is explicitly stopped and starts the main screen.

The Main Service is responsible for the Gateway to work in the background.

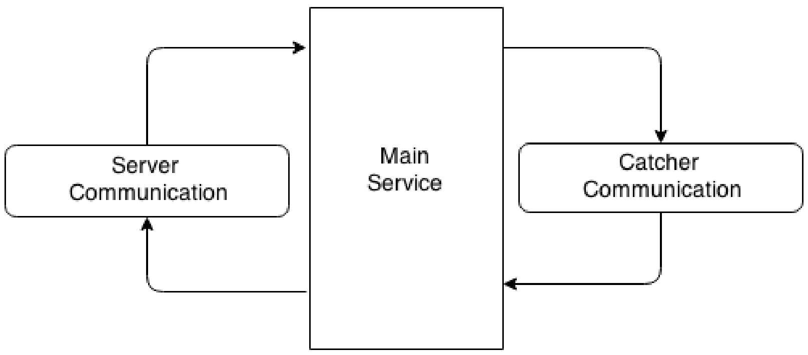


FIGURE 8. Main Service Loop (Noor, A.2015)

The Main Service runs in a loop and performs the tasks in an alternate fashion which is shown in Figure 8. The main tasks are: server communication and Catcher communication. When the Main Service loop is running and the task is set to Catcher communication, it starts the Catcher communication module.

The Catcher communication starts by initialising the Bluetooth adapter of the phone. After that, Gateway searches for devices by scanning for Catchers. When Gateway finds Catchers in proximity, it collects the advertising data from the respective Catchers and saves them to the database. While searching, Gateway cross checks each Catcher found, whether the Catcher has enough priority to go for a connection process.

When Gateway finds a suitable Catcher for a connection, Gateway starts Catcher Service, the remaining part of the Catcher communication module.

Catcher Service then sends a connection request to the selected Catcher and collects the information stored in the EEPROM of the Catcher. Catcher Service then saves the collected information to the database. The Catcher communication module unbinds the Catcher Service and sets the next task for the Main Service loop to be the server communication.

The server communication module then retrieves the saved information from the database and if the phone is connected to the Internet, it sends the data to the IoLiving cloud. On a successful data transfer, the server communication module removes the information from the database and sets the next task for the Main Service loop to be the Catcher communication.

Because IoLiving is a paid service, the user will be able to view most of the information collected from the Catcher only from the web service with a paid subscription. Nevertheless, the user can view some information from the Main Screen of the Gateway.

The Main Screen, as shown in Figure 9, contains the list of all Catchers the user has access to. For each Catcher, Main Screen shows Catcher communication and the server communication status as well as the latest temperature if the Catcher contains a temperature sensor. Each Catcher also shows the signal strength based on the scan result. The Main Screen retrieves this information from the database. Other features of the Main Screen include a ‘Logout’ button and a ‘Settings’ button in the action bar, a button to open the web service and an error message and instructions to the user depending on the state of the Gateway (e.g. If there is no Internet connection, the error message appears on top of the web service button).

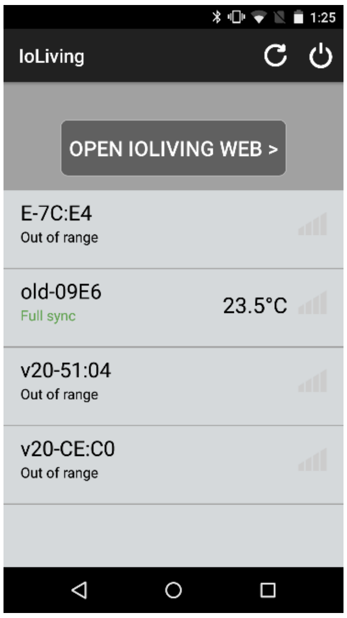


FIGURE 9. Main Screen (Noor, A.2015)

Based on the general design of the Gateway, the functionality can be divided into five major parts:

i. Login functionality
ii. Background processes
iii. Server Communication
iv. Catcher communication
v. Main Screen functionality

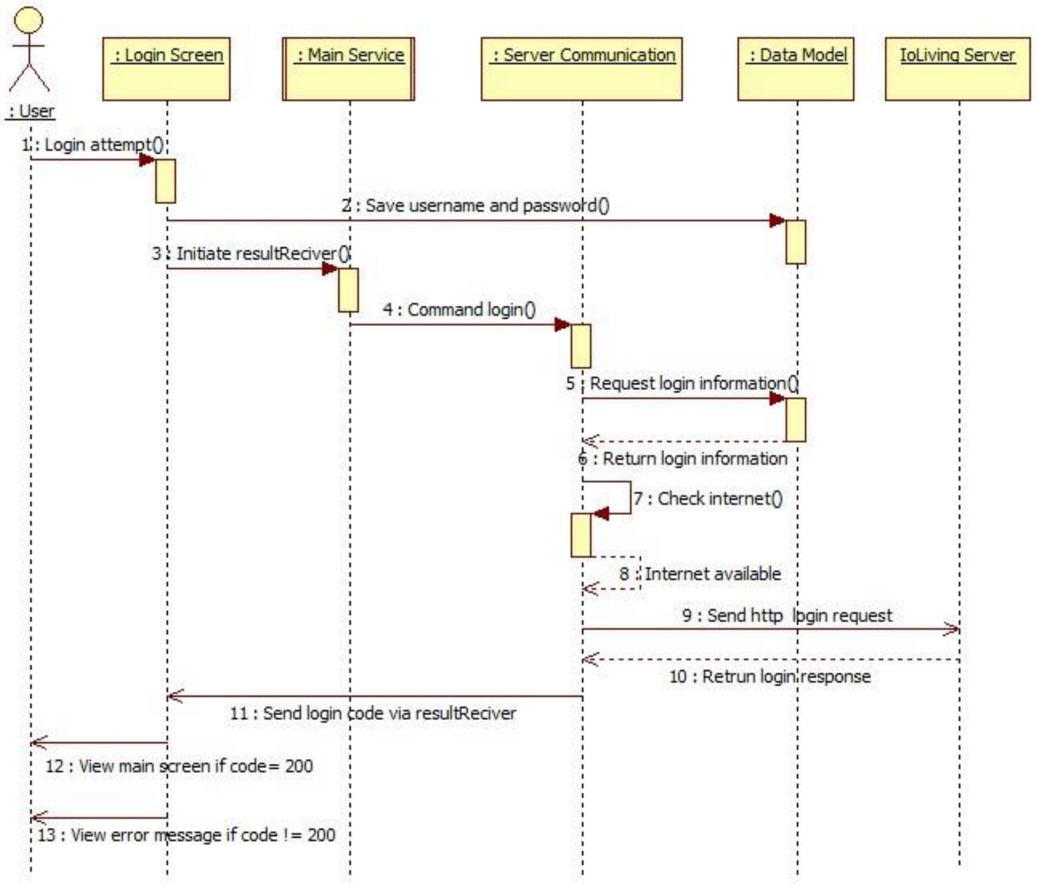
3.2 Login functionality

Gateway presumes that the user is already a registered member of the IoLiving web service and possesses a proper email and password in order to log in to Gateway. Figure 10 shows the sequence of events that takes place when the user attempts to log into Gateway.

When the user fills in the login form and presses the login button, Gateway makes a call to loginAttempt() which is a method in the Login Screen object. This method checks whether the user filled the login form with the proper email and password. If any of the fields in the form remains empty, the user will get an error message instructing the user to fill in the form correctly.

However, if the information is in a correct format, Gateway proceeds to the next step, i.e. saving the information to the database and initialising the ResultReceiver, a generic interface for receiving a callback result from the some other part of the application, a server communication module in case of Gateway. At the same time, Gateway calls onStartcommand() of the Main Service object which then commands the server communication module to perform an authentication sync.

The authentication sync is actually the process of sending the HTTP request to the IoLiving Server with login information retrieved from the database and receiving the response from the server. Before performing the sync, Gateway always makes a check for the Internet availability and only proceeds with the sync if the Internet is available.



: User
: Login Screen
: Main Service
: Server Communication
: Data Model
IoLiving Server
1 : Login attempt()
2 : Save username and password()
3 : Initiate resultReciver()
4 : Command login()
5 : Request login information()
6 : Return login information
7 : Check internet()
8 : Internet available
9 : Send http login request
10 : Retrun login response
11 : Send login code via resultReciver
12 : View main screen if code= 200
13 : View error message if code != 200

FIGURE 10. UML diagram of login functionality (Noor, A.2015)

The server communication module then saves the data to the database and sends the response code via the ResultReceiver. The onReceiveResult() method of the ResultReceiver in the Login Screen receives the response code and decides what to do next. Usually, the server sends 200 as a response code if the login information is correct. If the code is 200, then the Login Screen opens the Main Screen and closes itself. If the code is anything other than 200, the Login Screen shows an error message to the user, e.g. ‘a wrong username or password’.

3.3 Background processes

One of the features of Gateway is that once the user starts it, Gateway is able to run in the background for an indefinite amount of time unless the user explicitly stops it or the phone is switched off. This feature of running in the background is achieved by using a service to perform the background operations and a repeating alarm to continuously call the service once it completes its work. Figure 11 provides the sequence of events that take place in the background.

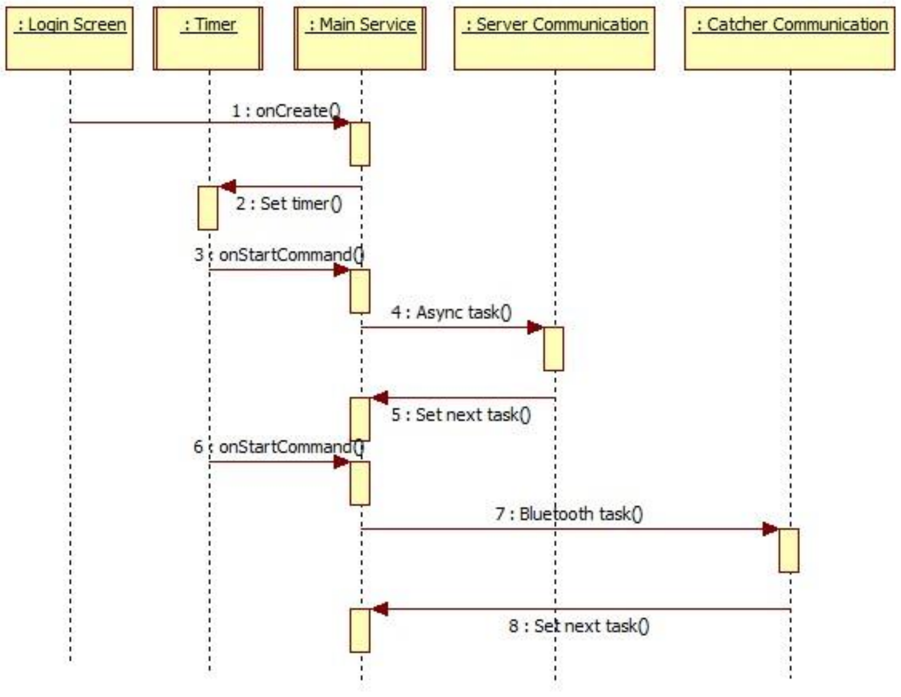


FIGURE 11. UML diagram for background processes (Noor, A.2015)

A successful login starts the Main Service of Gateway with a call to the onCreate() method from the Login Screen. The next call is to the setTimer() method of the Timer object which then sets a repeating alarm based on the AlarmManager class. AlarmManager enables Gateway to perform time-based operations outside the lifetime of the Main Service. The repeating alarm calls the Timer class at a certain time interval regularly. The timer class extends the BroadcastReceiver, a base class which responds to broadcast messages from the AlarmManager. The onReceive() method of the BroadcastReceiver then starts the Main Service with a call to the onStartCommand(). The onStartCommand() then commands a server communication module to carry out a measurement sync.

The ServerTask, before completion, sets the next task in the Main Service as the BluetoothTask. When the Timer class calls the onStartCommand() of the Main Service next time, it performs the BluetoothTask in the Catcher communication module. Before completion, the Catcher communication module also sets the next task, but this time it sets the next task to be the ServerTask. The next time the Timer calls the onStartCommand() method of the Main Service, it performs the ServerTask. This process runs in a loop for an indefinite amount of time in a regular interval between each task.

3.4 Catcher communication

The Catcher communication plays a crucial role in the IoLiving system architecture as this is the entry point of all the information collected by the Catchers and the sequence of events are shown in Figure 12.

When the Main Service calls the doBluetoothTask() method of the Catcher communication module, it checks whether the Bluetooth is tuned on. The process of Catcher communication only proceeds if the Bluetooth is turned on.

The next step in Catcher communication module is a BLE scan. The BLE scan allows a smartphone to look for advertising devices in proximity, Catchers in this case. To find Catchers, the Catcher communication module uses the startLeScan() method which has a callback as a parameter and which thus can retrieve the result of the scan. The scanning process continues for a certain amount of time and the Catcher communication module saves the data to the database.

Based on the scan result, the Catcher communication module checks if there is any Catcher with enough priority to make a connection. When there is no device with enough priority, the Catcher communication module sets the next task to be the server communication in the Main Service and then it exits.

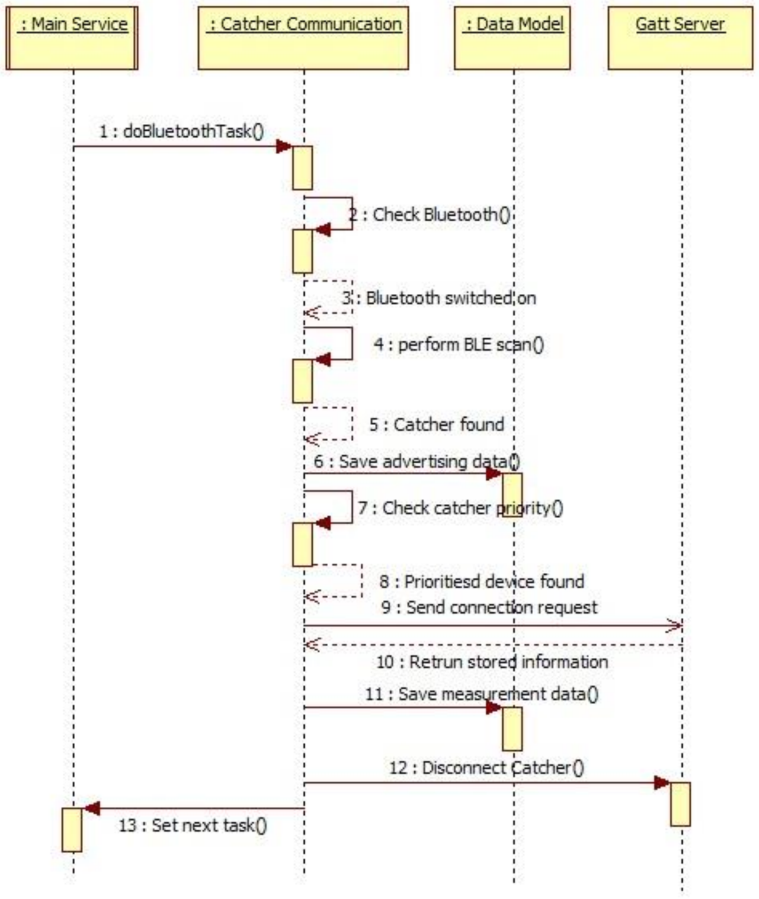

: Main Service
: Catcher Communication
: Data Model
Gatt Server
1 : doBluetoothTask()
2 : Check Bluetooth()
3 : Bluetooth switched on
4 : perform BLE scan()
5 : Catcher found
6 : Save advertising data()
7 : Check catcher priority()
8 : Prioritiesd device found
9 : Send connection request
10 : Retrun stored information
11 : Save measurement data()
12 : Disconnect Catcher()
13 : Set next task()

FIGURE 12. UML diagram for the Catcher communication (Noor, A.2015)

However, when there is a Catcher with priority, the Catcher communication module starts a bound service, namely a Catcher Service, to take care of the connection process.

The Catcher Service sends the connection request to the GATT server of the Catcher. The request includes a callback parameter which delivers results to Gateway such as the various state of the connection phase as well as any further Gateway operations, e.g. reading or writing characteristics.

Once the Catcher Service makes a successful connection to the GATT server of the Catcher and discovers the required services, the Catcher Service can read and write an attribute to the service which supports it.

The Catcher Service then sends a request for notifications, which returns the packets of information stored inside the EEPROM of the Catcher in the form of notifications. When the Catcher Service finishes receiving notifications and saving the information to the database, it calls the close() method to release the resources appropriately.

The Catcher communication module then unbinds the Catcher Service. Right before exit, the Catcher communication module sets the next task in the Main Service to be the server communication.

3.5 Server communication

Once the user passes the login phase, there comes the server communication. The server communication includes mostly a measurement sync, which is very similar to the authentication sync of the login functionality. The target is to send the information collected from Catchers to the IoLiving web service. The sync process requires an Internet connection, so the first task in the server communication module is to check the availability of the Internet. If the smartphone is not actively connected to the Internet, the process cannot continue and it always skips to the next task: a Bluetooth communication task.

However, if the smartphone has an active Internet connection, the server communication module checks whether enough time has passed since the last server connection. If the time elapsed since the last server connection is not enough for the next server connection to take place, the server communication module sets the next task to the Bluetooth communication task and exits. Conversely, if enough time has elapsed since the last server connection, the server communication module proceeds with the usual request-response process. Figure 13 shows the sequence diagram for the server communication module.

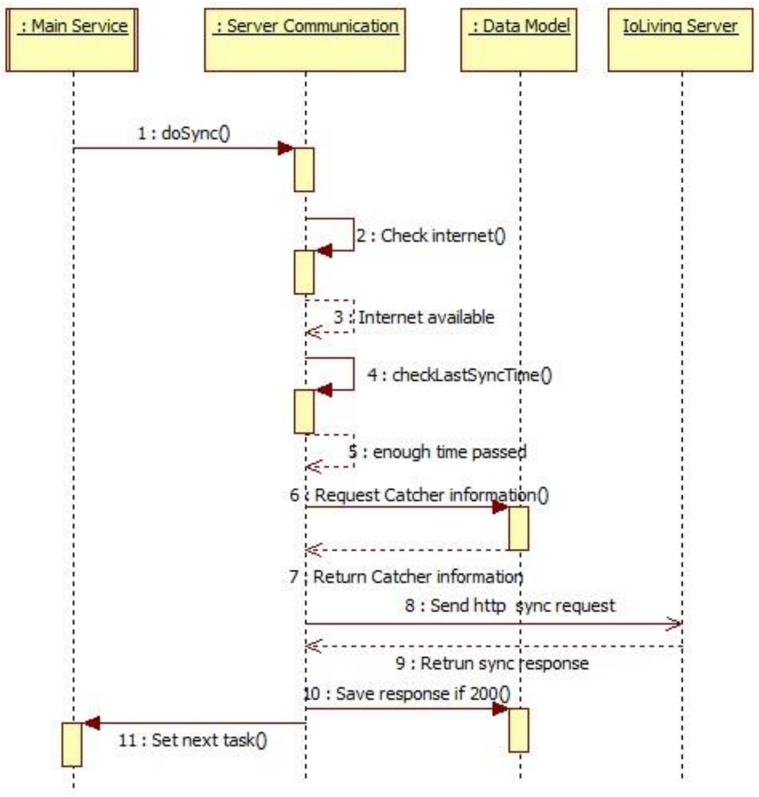


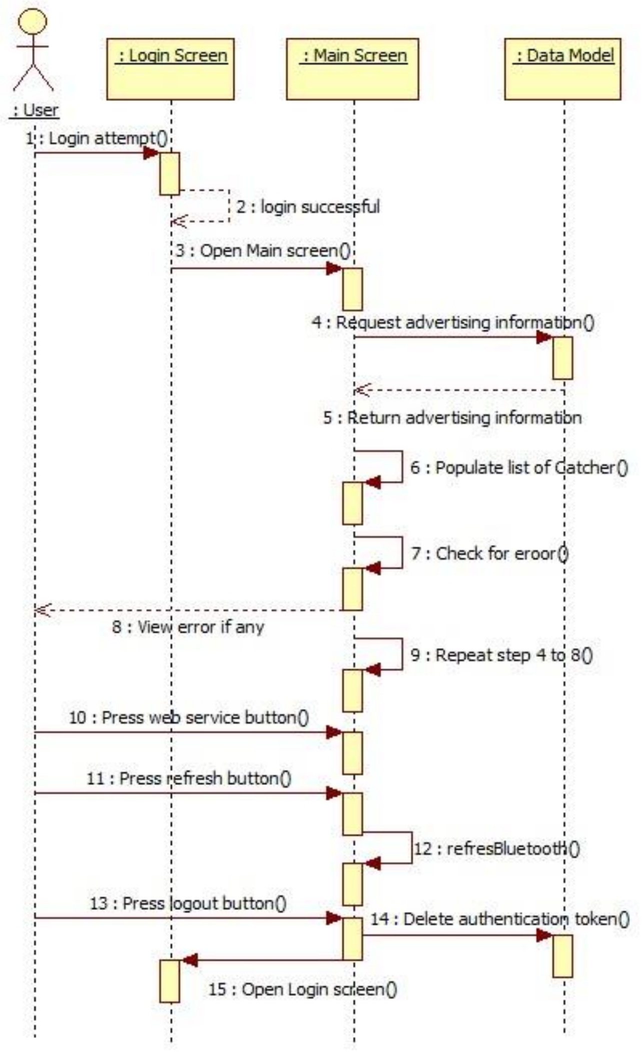
FIGURE 13. UML diagram for the server communication module (Noor, A.2015)

The request-response process starts by retrieving the saved information from the database and converting this information to a proper format so that the IoLiving web service can recognise the information.

In the next step, the server communication makes the actual request to the IoLiving web service, i.e. an HTTP request containing the necessary information, using the IoLiving API for the measurement sync. The server then sends a response, based on the information the server received. Usually, in a successful case, the response contains a list of Catchers to which the user has an access and the header contains the response code which is 200. The server communication module then saves the response to the database by converting it again to a proper format. In case of failure, the server communication module typically throws an exception which is displayed on the Main Screen. The server communication module then defines the next task to be the Bluetooth communication task and exits.

3.6 Main screen functionality

The Main Screen of Gateway provides the user with information of the intermediary state of what is going on in the background service of Gateway. The Main Screen also shows the user the current temperature of the Catcher if the Catcher is equipped with a temperature sensor. It provides ways to refresh Bluetooth and Logout, and to open the web service. Figure 14 sums up the sequence of events that take place on the Main Screen.



: User
: Login Screen
: Main Screen
: Data Model
1 : Login attempt()
2 : login successful
3 : Open Main screen()
4 : Request advertising information()
5 : Return advertising information
6 : Populate list of Catcher()
7 : Check for eroor()
8 : View error if any
9 : Repeat step 4 to 8()
10 : Press web service button()
11 : Press refresh button()
12 : refresBluetooth()
13 : Press logout button()
14 : Delete authentication token()
15 : Open Login screen()

FIGURE 14. UML diagram for the Main Screen functionality (Noor, A.2015)

After a successful login attempt, Gateway opens the Main Screen. The Main Screen has a loop of its own that keeps the main screen up to date with the data Gateway collects. The loop starts by retrieving advertising information, an error message (if any) and the Catcher communication and the server communication status from the database.

The Main Screen functionality is a foreground process and only takes place when the user opens the Gateway UI. Closing the application results in exit from the loop.

As stated before, the Main Screen includes some additional features like a web service button and a menu. The menu item includes a ‘Settings’ button and a ‘Logout’ button. The Settings button opens a dialog that contains various setting options to facilitate the user experience. The Logout button deletes the authentication token from the database and redirects the user to the Login Screen. The web service button allows the user to directly open the IoLiving web service page from the Main Screen.

3.7 Database

A database can be considered as one of the most crucial parts of Gateway as it saves and manages all the information based on which the Gateway runs. The database acts as a bridge between the background services and the foreground activities. Gateway makes use of three tables to manage the communication modules and the in-app functionalities:

- The settings table stores the general settings of Gateway, e.g. a username, a password, an authentication token.
- The catcher table stores Catcher specific data like a catcher name, advertising data and various timestamps.
- The measurement table is dedicated to handle only the measurement data, which is the information collected as notifications from the Catcher.

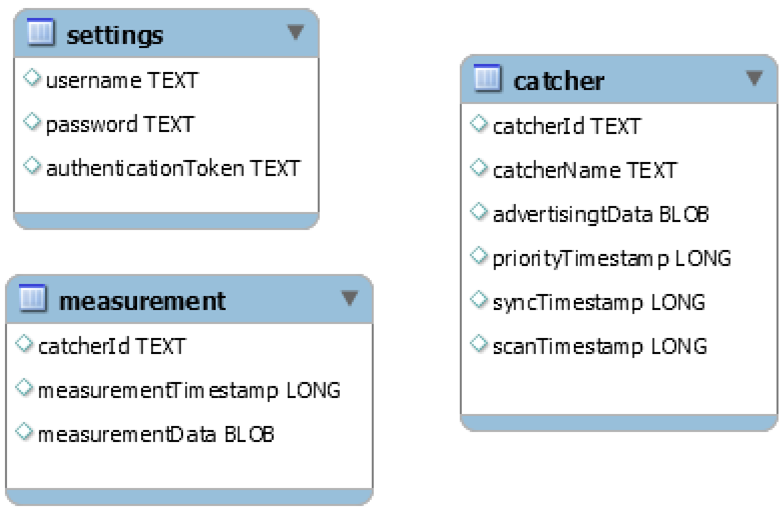


FIGURE 25. Database Model (Noor, A.2015)

In the database class, the actual data management is handled through methods that contain a SQLite statement to take care of data retrieval, data removal, data insert and update to the respective tables. Figure 15 illustrates the database model.

4 GATEWAY SOFTWARE IMPLEMENTATION

This chapter describes the implementation of the design and the theory upon which Gateway is based. Taking into account of all the functionalities of Gateway, the implementation part can be split into five parts: user interfaces, background services, the Bluetooth low energy communication, the server communication and the data model.

4.1 User interface

Gateway defines the user interface by declaring UI elements in a file with '.xml' extension and also, in some part, by managing the user interface by instantiating layout elements at runtime.

4.1.1 Splash Screen

As the task of the Splash Screen is just to check whether Gateway has a valid authentication token saved in the database, the Splash Screen activity does not have any dedicated user interface functionality.

The UI of the Splash Screen just shows the IoLiving logo in an ImageView when the Gateway starts for the first time. The following codes from the activity_splash_screen.xml file uses an ImageView inside a LinearLayout in order to view the logo:

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent">
   <ImageView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:src="@drawable/splash_screen" />
</LinearLayout>
```

The Splash Screen uses an onCreate() callback from the activity lifecycle to load the activity_splash_screen.xml file:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      //set up the UI elements
      setContentView(R.layout.activity_splash_screen);
}
```

In order to check the availability of the authentication token, the Splash Screen uses the following Java code:

```
// create instance of Data Model
DataHandler dHandler = DataHandler.getInstance(this);
// declare variable and retrieve authentication token
// from data model instance
String authToken =dHandler.authToken();
if(authToken!=null){
 //open MainScreen
 startActivity(new Intent(this,
      com.ceruus.ioliving.ui.MainScreen.class));
}
else{
 //open LoginScreen
 startActivity(new Intent(this,
      com.ceruus.ioliving.ui.LoginScreen.class));
}
//close SplashScreen
this.finish();
```

4.1.2 Login Screen

In Gateway, the Login Screen is the entry point for the user who has a valid email and password. The onCreate() callback of the Login screen loads the xml file which uses a similar layout as the Splash Screen layout but has more UI components including e.g. TextView, EditText and a Button.

The button has an onClickListener, which listens to the click event of the button. The following Java statement creates the listener:

```
View.OnClickListener loginButtonListener = new View.OnClickListener() {
             public void onClick(View v) {
                   EditText nameText = (EditText) findViewById(R.id.LoginNameEdit) ;
                   EditText passwdText = (EditText) findViewById(
                                                       R.id.LoginPasswordEdit) ;
                   String strUserName = nameText.getText().toString();
                   String strPassword = passwdText.getText().toString();
      //save email and password to database
            mDataHandler.updateLoginCredentials(namText.getText().toString(),pass-
            wdText.getText().toString()) ;
      //Now we have possible username + password in db,
      //ask for MainService to try them out:
Intent intent = new Intent(LoginScreen.this, MainService.class);
                          intent.setAction(Intent.ACTION_CALL) ;
                          intent.putExtra("receiver", resultReceiver);
                          startService(intent);
      }
};
```

When the button is clicked by the user after inserting an email address and a password in their respective fields, the onClickListener saves them to the data model and starts the Main Service in order to attempt login. An instance of the ResultReceiver is passed to the service. The ResultReceiver practically works as a callback from the Main Service and tells Gateway what to do in a success or an error case of the login attempt. The implementation uses the onReceiv-eResult callback method like the following:

```
class LoginScreenResultsReceiver extends ResultReceiver {

      public LoginScreenResultsReceiver(Handler handler) {
                   super(handler);
      }

      @Override
      protected void onReceiveResult(int resultCode, Bundle resultData) {
            super.onReceiveResult(resultCode, resultData);
            if(resultCode==0){
                   // start main service and main screen. Close this screen
       }
            else{
                   //view error message to the UI
            }
      }
}
```

4.1.3 Main Screen

As the name suggests, the Main Screen serves the purpose of the main view as most of the interactions with the user take place in the Main Screen. The xml layout includes an action bar, a TextView, a button and a customised ListView inside a LinearLayout. The ListView uses a RelativeLayout in a different xml file to define each row of the ListView and includes the TextView and ImageView.

The ListView occupies the most part of the Main Screen. The ListView shows information by retrieving it from the database and the following lines demonstrate the implementation:

```
ArrayList<HashMap<String, Object>> catcherData = mDataHandler.getAllCatcherData();
String[] catcherName = new String[catcherData.size()];
for (int i = 0; i < catcherData.size(); i++) {
catcherName[i] = (String) catcherData.get(i).get("catcher_name");
}
```

After a successful data retrieval from the database using a database, the Main Screen puts the data in a map and with the help of a ListView adapter, shows them in the ListView. The code below shows how this is done:

```
String[] from = new String[] { "catcherName" };
int[] to = new int[] { R.id.catcher_name };
ListView listView = (ListView) findViewById(R.id.list);
List<HashMap<String, String>> ListViewMap = new ArrayList<HashMap<String,
String>>();
HashMap<String, String> map = new HashMap<String, String>();
map.put("catcherName", catcherName[i] );
ListViewMap.add(map);
ListAdapter adapter = new SimpleAdapter(this, ListViewMap,R.layout.list_row,
from, to);
listView.setAdapter(adapter);
```

The Main Screen also shows error messages, if there are any, while trying to make a server connection. The error messages come from the server communication side of Gateway and they are then shown in the Main Screen. The code below shows an example of how the error messages are shown:

```
TextView errorView = (TextView) findViewById(R.id.tv_error_handler);
errorView.setText(“no internet!”);
```

In order to view the latest information to the user, the Main Screen needs to update its contents repeatedly after a certain time interval. This update method runs in a loop by means of a handler:

```
private Runnable runnable = new Runnable() {
            @Override
            public void run() {
                  if(mDataHandler.authToken()!=null){
                        //update Main Screen here
                        //repeat the process after certain delay.
                        mHandler.postDelayed(this, 10 * 1000);
                  }
                  else{
                        close();
                  }
            }
};
```

The Main Screen includes a web service button, which has an onClickListener that redirects the user to the IoLiving web service by means of the following code:

```
Intent webIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(
      "https://ioliving.com/login.php"));
startActivity(webIntent);
```

The action bar of the Main Screen has two menu items: Settings and Logout. The click function of these items use the following lines to make selections:

```
case R.id.logout :
// logout function
break;
case R.id.settings:
// show a dialog listing settings
break;
```

4.2 Background processes

The functionalities of Gateway requires it to perform certain operations in the background because running these operations in the foreground, i.e. in the UI thread, will affect the responsiveness of the user interface. The Android framework offers several ways to carry out these operations in a separate thread, thus minimising the load from the UI thread. Gateway makes use of mainly different types of services. It also uses AsyncTask to take care of the background operations and a Timer class to repeat these operations at a regular interval.

4.2.1 Main Service

The Main Service of Gateway is a started service and it is started by the following code:

```
Intent intentForService = new Intent(LoginScreen.this,MainService.class);
intentForService.setAction(Intent.ACTION_SYNC) ;
startService(intentForService);
```

Once started, the onStartCommand() method of the Main Service decides the next operation based on the type of action set to the intent which starts the service. The following code shows an overview of how the Main Service decides the next operation using the action type of intent as a filter:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
      if (actionOfIntent.equals(Intent.ACTION_SYNC)) {
      // perform authentication sync
      authSync();
      }
      else if (actionOfIntent.equals(Intent.ACTION_CALL)) {
      // perform measurement sync or Bluetooth communication
            if (mNextTask == taskType.ServerTask) {
                  measurementSync();
            }
            else if (mNextTask == taskType.BluetoothTask) {
                  bluetoothCommsTask();
            }
      }
}
```

The Main Service makes use of an enum to enlist the type of operations the service needs to perform. These operations always fall into two categories, namely, a server communication and a Bluetooth communication and the code below shows the use of enum in the Main Service to handle these tasks:

```
private enum taskType {
            BluetoothTask, ServerTask
      }

private taskType mNextTask = taskType.ServerTask;
public void onTaskTypeChange(String nextTask) {
            switch (nextTask) {
            case "BluetoothTask ":
                  mNextTask = taskType.BluetoothTask;
                  break;
            case "ServerTask":
                  mNextTask = taskType.ServerTask;
                  break;
            }
}
```

4.2.2 Timer

The Timer class basically enables Gateway to run background operations outside the lifetime of Gateway and provides a certain amount of interval between operations for the smooth functioning of Gateway. The Timer class accomplishes this with the aid of repeating alarm which is set when the Main Service is first initiated and after that, the repeating alarm keeps the background processes of Gateway running as long as the phone is turned on. The code to set the repeating alarm is as follows:

```
public void setInterval(Context context, int seconds) {
AlarmManager alarmManager =
(AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
      Intent intent = new Intent(context, Timer.class);
PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0, intent,
PendingIntent.FLAG_UPDATE_CURRENT);
alarmManager.setRepeating(AlarmManager.RTC_WAKEUP,
(1000 * 10), 1000 * seconds, pendingIntent);
      }
```

The Timer class extends the BroadcastReceiver. Thus, each time the repeating alarm wakes up, the onReceive() method of the timer class retrieves the broadcast performed by the PendingIntent. The onReceive() method, presented below, then starts the Main Service to perform the other background operations:

```
@Override
public void onReceive(Context context, Intent intent) {
       try {
Intent intentForService = new Intent(context, MainService.class);
              intentForService.setAction(Intent.ACTION_SYNC);
              context.startService(intentForService);
              }
       catch (Exception e) {
              Log.v(TAG, "timer exception: " + e.toString());
              }
       }
```

4.2.3 Catcher Service

The Catcher Service is solely responsible for handling the Bluetooth smart connection between the phone and the Catcher. The Catcher Service is a bound service which does not run indefinitely in the background. This feature makes the Catcher Service very useful to perform connection related tasks between the Catcher and the phone.

The following code is used to bind the service to the Bluetooth communication task:

```
Intent serviceIntent = new Intent(context, CatcherService.class);
isBound = context.bindService(serviceIntent, mServiceConnection,
Context.BIND_AUTO_CREATE);
```

To end the lifetime of the Catcher Service, it is necessary to unbind the service which is done by means of the following code:

```
if (isBound) {
      mOwningService.getBaseContext().unbindService(mServiceConnection);
}
```

In order to manage the lifecycle of the Catcher Service, a callback function is implemented in the Bluetooth communication module of Gateway. The code for the callback function is the following:

```
public ServiceConnection mServiceConnection = new ServiceConnection() {
       @Override
public void onServiceConnected(ComponentName componentName, IBinder service) {
              // commands for establishing Connection with the Catcher
       }

      @Override
      public void onServiceDisconnected(ComponentName componentName) {
              // commands for terminating Connection with the Catcher
      }
};
```

4.2.4 AsyncTask

The server communication module in Gateway is mostly an implementation of the AsyncTask, as this module does not require a direct UI thread and the operations in this module are generally short lived. Gateway uses the following code to execute an AsyncTask to perform either an authentication sync or a measurement sync:

```
// retrieve login url from config file
URL urlOfSync = new URL(Config.loginUrl);
// initialise the serverCommunication module object with data model instance and
// application context
ServerCommunication serverSync = new ServerAuthController(mDataModel, this);
// excute the AsyncTask
serverSync.execute(urlOfSync);
```

4.3 Bluetooth communication

The operations in the Bluetooth communication module, categorised as either an advertising data collection or a Measurement data collection, always start by a Bluetooth initialisation. The sequence of events, which take place when the Main Service commands Gateway to perform the Bluetooth communication, includes the Bluetooth initialisation, the BLE scan and the Bluetooth connection. The code for initialising Bluetooth is the following:

```
private BluetoothAdapter mBluetoothAdapter;
private void initializeBluetooth(Context context) {
// Initializes Bluetooth adapter.
final BluetoothManager bluetoothManager =
        (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();
if (mBluetoothAdapter == null || !mBluetoothAdapter.isEnabled()) {
  Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
  startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
 }
}
```

4.3.1 Advertising data

After confirming that the phone has an active Bluetooth adapter, the next step in the Catcher communication module is scanning for Bluetooth low energy devices, i.e. Catcher to be more specific. The scan is initiated using the following code:

```
 scanLeDevice(true);
```

The scanLeDevice() method then performs the actual scan for Catcher for a certain period of time after which, this method commands the BLE scan to stop and start selecting Catcher from the priority list to go for a Low Energy connection. The code is illustrated below:

```
private void scanLeDevice(final boolean enable) {
      if (enable) {
           mHandler.postDelayed(new Runnable() {
                 @Override
                 public void run() {
                       // stops scanning
                       mBluetoothAdapter.stopLeScan(mLeScanCallback);
      }
           }
      },
      //runs scan for certain period
      Config.scanPeriod);
      mBluetoothAdapter.startLeScan(mLeScanCallback);

     } else {
           //stops scanning
           mBluetoothAdapter.stopLeScan(mLeScanCallback);

     }
}
```

The callback function is used to get broadcast data when there are Catchers in proximity is as follows:

```
private BluetoothAdapter.LeScanCallback mLeScanCallback = new
BluetoothAdapter.LeScanCallback() {
     @Override
public void onLeScan(final BluetoothDevice device,
 final int rssi, final byte[] scanRecord) {
            // save broadcast data to database
            //add found Catcher to the database
            }
};
```

4.3.2 Measurement data

The next usual step for the Bluetooth communication module is to select a suitable Catcher to go for a BLE connection. Gateway prioritise Catcher on the basis of timestamp when the latest connection was made to that Catcher. Usually, the earliest Catcher that went through the process of connection, or a newly introduced Catcher which is yet to make a connection, is given the priority by Gateway. However, it is crucial that the Catcher is in proximity of the phone in order to proceed with the connection attempt.

The command for the connection attempt is implemented in the Catcher Service and the following is the statement to execute it:

```
mBluetoothGatt = device.connectGatt(this, false, mGattCallback);
```

The following callback is used to retrieve and save the notifications:

```
private final BluetoothGattCallback
btleGattCallback = new BluetoothGattCallback() {
      @Override
public void onCharacteristicChanged(BluetoothGatt gatt,
 final BluetoothGattCharacteristic characteristic) {
        // this method gets called when Gateway
         // performs a read or write characteristic operation
         }

      @Override
      public void onConnectionStateChange(final BluetoothGatt gatt,
   final int status, final int newState) {
           // this will get called when a device
            // connects or disconnects
         }

      @Override
      public void onServicesDiscovered(final BluetoothGatt gatt,
   final int status) {
             // this will get called after the client initiates a
              // BluetoothGatt.discoverServices() call
         }
}
```

When the notifications from Catchers are saved to the database, the Catcher communication module destroys any existing connection. The following code is used to close the GATT:

```
mBluetoothGatt.close();
mBluetoothGatt.disconnect();
```

4.4 Server communication

In order to perform network operations, Gateway must include the following permissions in its manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

However, having these permissions does not always confirm that a communication with the IoLiving web service can be established since an active network connection is needed for that. So, before making any connection attempt, Gateway always checks whether a network connection is available using the following statements:

```
ConnectivityManager connMgr = (ConnectivityManager)
                    getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
if (networkInfo != null && networkInfo.isConnected()) {
      // proceed with communication process
} else {
      // report error
}
```

Gateway makes use of the Apache HTTPClient to send and receive data from the IoLiving web service. AsyncTask provides a separate simple thread to carry out network operations using the HTTPClient which is demonstrated by the following code:

```
// retrieve data from database
JSONObject dataToSend = mDataHandler.authCommsEntity();

// prepare proper request format for IoLiving API
URL realUrl = urls[0];
HttpParams httpParameters = new BasicHttpParams();
int timeoutConnection = 10 * 1000;
HttpConnectionParams.setConnectionTimeout(httpParameters, timeoutConnection);
int timeoutSocket = 30 * 1000;
HttpConnectionParams.setSoTimeout(httpParameters, timeoutSocket);

DefaultHttpClient httpclient = new DefaultHttpClient(httpParameters);
HttpPost httppost = new HttpPost(realUrl.toString());
httppost.setEntity(new ByteArrayEntity(dataToSend.toString().getBytes("UTF-
8")));

// Execute HTTP Post Request
HttpResponse response = httpclient.execute(httppost);
```

The Measurement sync and authentication sync use a very similar mechanism to send and receive data to and from the server, apart from using a different API. Another significant difference is that while making the authentication sync with the server, the response from the server contains messages that need to be interpreted by the Login Screen in order to decide what to do next. The Login Screen has a ResultReceiver and the following lines of code is used to send the response from the server communication module to the Login Screen:

```
Bundle resultData = new Bundle();
resultData.putString(aOptionalErrorMessage, aOptionalErrorMessage);
mResultReceiver.send(aSuccess == true ? 0 : 1, resultData);
// set the resultreciver value to null after sending
mResultReceiver = null;
```

After the communication with the server is complete, the following code is used to set the next operation for the Main Service:

```
// set the next task to Bluetooth communication
((MainService) mContext).onTaskTypeChange("BluetoothCommunication");
```

4.5 Data Model

In Gateway, the data storage and management is done with the help of three database tables and methods to insert, select, and update data. The creation and data management processes for all three tables are the same except that they contain different numbers and types of fields. The code below illustrates how a ’settings’ table is created:

```
String CREATE_SETTINGS_TABLE = "CREATE TABLE settings ("
                         + "user_account text,"
                         + "user_password text,"
                         + "auth_token text)";
db.execSQL(CREATE_SETTINGS_TABLE);
```

In a data model, methods are implemented to perform certain tasks. Each method has a specific task assigned to it. The following method shows how to insert data to the database:

```
public boolean insertAuthToken(String auth_token) {
ContentValues valuesToInsertToDb = new ContentValues();
valuesToInsertToDb.put("auth_token", auth_token);
db.insert("settings", null, valuesToInsertToDb);
}
```

In some cases, Gateway needs to override the previous data in the table. The code below shows how this update process is done:

```
ContentValues cv = new ContentValues();
cv.put("auth_token", auth_token);
db.update("settings", cv, null, null);
```

Data is retrieved from the database table in the following way:

```
String authToken = null;
SQLiteDatabase db = getReadableDatabase();
Cursor cursor = db.rawQuery("SELECT auth_token FROM settings", null);
if (cursor.moveToNext()) {
	authToken = cursor.getInt(0);
}
cursor.close();
return authToken;
```

5 TESTING

Testing the functionalities is one of the major phases in software development. Testing provides new information about how the software works in different situations. It will help in further modifications and improvements of the software. This section provides a description of the test cases for Gateway. They can be divided into three categories: an application start-up, background processes and Main Screen functions. The tests are carried out using different smartphones to ensure that Gateway works similarly in every smartphone.

5.1 Application start-up

After the installation of the Gateway app in a smartphone, launching the app for the first time should take the user to the Login Screen. The user should then fill in the login form. On pressing the login button, wrong or inadequate information should result in an error message in the UI, whereas correct information should open the Main Screen. The next time the user launches Gateway, it should automatically open the Main Screen.

Table 1 provides a list of tests that are carried out to check how Gateway works during the start-up. From the table it can be seen that the desired outcome matches the results of all tests. These ensure that, at start-up, the final version of Gateway works as specified in the design.

TABLE 1. The test cases used in functionality test of the application start-up process of the Gateway (Noor, A.2015)

	Initial State	Step(s)	Desired Outcome	Match Desired Outcome
1.	Application is installed in the system	Launch application Application checks if there is authentication token in local database	No authentication to-ken in local database. Application opens Login Screen.	Yes
2.	Application is launched	Login attempt with wrong username or password	Error message telling that username or pass-word is wrong	Yes
3.	Application is launched	Login attempt with empty username and password	Login button disabled. Error message show-ing empty field	Yes
4.	Application is launched	Login attempt with proper username and password	Application opens Main Screen	Yes
5.	Application is launched	Application checks if there is authentication token in local database	Application opens Main Screen	Yes

5.2 Background processes

After the initial start-up process is done, the next test cases for Gateway are the automated background processes. These test cases presume that the user al-ready has a registered Catcher available near the phone and Gateway is run-ning in the background.

The background processes can be tested simply by looking at the UI. The ListView that displays the list of Catchers should also show the state of the data collected from the Catchers. The UI should update this state when there is a change.

Table 2 enlists the test cases that are used to confirm the proper functioning of the background processes. The final version of Gateway, as seen from the table, functions as expected.

TABLE 2. The test cases used in functionality test of the background processes of the Gateway (Noor, A.2015)

	Steps	Desired Outcome	Match Desired Outcome
1.	Open Main Screen and wait for some time	‘Scanning’ or ‘Data waiting’ text should appear beside the Catchers which are near	Yes
2.	Close all network connections from phone	Data from nearby Catchers owned by user will be read to phone but not sent to server	Yes
3.	Allow network connection from phone and wait for some time	‘Full sync’ or ‘Partial sync’ text should appear beside the Catchers which are near	Yes

5.3 Main screen function

The Main Screen in Gateway should act as a window and provide the user with information on what is going on in the background. Also, the Main Screen has buttons which upon click, should redirect to the IoLiving web service and option menu in the action bar. This should enable users to logout or set various preference options.

Table 3 presents a list of test cases that are used to test the Main Screen of the final version of Gateway. The results for all the Main Screen functionalities match the desired outcomes, as seen from the table.

TABLE 3. The test cases used in functionality test of the Main Screen (Noor, A.2015)

	Steps	Desired Outcome	Match Desired Outcome
1.	Open Main Screen and wait for some time	Nearby Catcher should show temperature value in real time	Yes
2.	Press web service button	IoLiving web service page should appear	Yes
3.	Press Settings button	Provides option to reserve Bluetooth for IoLiving	Yes
4.	Press Logout button	Main screen should open Login Screen	Yes

6 CONCLUSIONS

6.1 Final status of the work

Gateway can already be downloaded from the Google Play Store and be used commercially by Ceruus' clients which is a clear indication of the success of the whole project. The commercial name for the Gateway software is the 'IoLiving' app.

In its current state, Gateway can consistently connect to and disconnect from up to 20 Catchers without any problem. However, a key point to note is that the Android Bluetooth stack is not designed to handle such frequent communication and over time, it can become unresponsive to certain connection commands. Therefore, it is crucial that all the parameters in the sync logic are correctly set. The optimal values for the parameters in the sync logic are selected by testing with different phone models.

The data collection interval from Catcher and the data upload interval to a server might also vary from phone to phone due to a different manufacturer and a different Bluetooth stack. At some point of the application lifetime, the data collection can even be a failure. As the background process for Gateway is continuous and, once started, runs for an indefinite amount of time, the failure cases are often ignored and in critical cases Gateway automatically resets the Bluetooth of the phone.

6.2 Personal comment

Being able to work in collaboration with the other team members is the key thing that helped me to do my part in this project. The deadline was always a factor we could not avoid and it was necessary to be able to adapt to the situations on the basis of requirements. These, apart from gaining more skills in programming and software design, version control and work load management issues helped me to enrich the experience that I am sure will help me a lot in the near future to develop my career. I am very pleased with my colleagues and employer for presenting me with such a wonderful opportunity to work in a completely professional environment. Also, I am grateful to my teacher and School for making it as a part of study and all the support they provided. I believe that this type of professional project for a thesis can be considered as one of the most fundamental key to learn. It will also give a head start to an early career plan.

REFERENCES

1. Android Developers. 2015. Getting Started. Date of retrieval 8.4.2015. http://developer.android.com/training/index.html

2. Android Developers. 2015. Developer Support Resources. Date of retrieval 8.4.2015. http://developer.android.com/support.html

3. Android Developers. 2015. Package Index. Date of retrieval 8.4.2015. http://developer.android.com/reference/packages.html

4. Rouse, M. 2014. Internet of Things(IoT). Date of retrieval 6.4.2015. http://whatis.techtarget.com/definition/Internet-of-Things

5. Wikipedia. 2015. Internet of Things. Date of retrieval 6.4.2015. http://en.wikipedia.org/wiki/Internet_of_Things

6. Gibbele, M. 2014. Android Bluetooth Low Energy Tutorial. Date of retrieval 9.4.2015. http://toastdroid.com/2014/09/22/android-bluetooth-low-energy-tutorial/

7. Wikipedia. 2015. Bluetooth low energy. Date of retrieval 9.4.2015. http://en.wikipedia.org/wiki/Bluetooth_low_energy

8. Bluetooth Developer portal. 2015. Generic Attribute Profile (GATT). Date of retrieval 9.4.2015. https://developer.bluetooth.org/TechnologyOverview/Pages/GATT.aspx

9. Wikipedia. 2015. Agile software development. Date of retrieval 28.3.2015. http://en.wikipedia.org/wiki/Agile_software_development

10. Wikipedia. 2015. Scrum (software development). Date of retrieval 28.3.2015. http://en.wikipedia.org/wiki/Scrum_(software_development)

11. Dmazzoni. 2010. How to send a JSON object over Request with Android? Date of retrieval 13.04.2015. http://stackoverflow.com/questions/3027066/how-to-send-a-json-object-over-request-with-android

12. Connect blue. 2013. Bluetooth Low Energy Serial Port Adapter - Getting Started. Date of retrieval 13.04.2015. http://support.connectblue.com/display/PRODBTSPA/Bluetooth+Low+Energy+Serial+Port+Adapter+-+Getting+Started

13. Ceruus Internal Documents.2014. Connection Establishment of Bluetooth Low Energy Devices. Date of retrieval 2.03.2015.

14. Ceruus Internal Documents.2014. Press release. Date of retrieval 6.04.2015.

APPENDIX

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ceruus.ioliving"
    android:versionCode="3"
    android:versionName="1.0.3" >
    <uses-sdk
        android:minSdkVersion="18"
        android:targetSdkVersion="19" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-feature
        android:name="android.hardware.bluetooth_le"
        android:required="true" />
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
   <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <service android:name="com.ceruus.ioliving.MainService" />
        <service
            android:name="com.ceruus.ioliving.catcherComms.CatcherService"
            android:enabled="true"
            android:exported="true" >
        </service>
       <receiver
            android:name="com.ceruus.ioliving.TimerHandler"
            android:enabled="true"
            android:process=":remote" >
        </receiver>
       <activity
            android:name="com.ceruus.ioliving.ui.SplashScreen"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.ceruus.ioliving.ui.LoginScreen"
            android:launchMode="singleTask" />
        <activity
            android:name="com.ceruus.ioliving.ui.MainScreen"
            android:launchMode="singleTask" />
    </application>
</manifest>
```

XML Layout Contents

TextView

```
<TextView
android:id="@+id/errorMessageText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginLeft="40dp"
android:layout_marginTop="20dp"
android:layout_weight="0.21"
android:textColor="#000000"
/>
```

EditText

```
<EditText
android:id="@+id/loginNameEdit"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_gravity="center"
android:background="@drawable/layout_bg"
android:inputType="textEmailAddress"
android:layout_marginStart="39dp"
android:textColor="@color/ioLivingTextColor"
android:textCursorDrawable="@drawable/color_cursor" >
```

Button

```
<Button
android:id="@+id/loginButton"
android:layout_width="120dp"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_marginLeft="130dp"
android:layout_marginRight="100dp"
android:text="@string/buttontext_login"
android:textSize="25sp" />
```

ListView

```
<ListView
android:id="@+id/list"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_weight="1"
android:divider="#a1a1a1"
android:dividerHeight="4px"
android:listSelector="@drawable/list_row_selector" >
</ListView>
```

ImageView

```
<ImageView
android:id="@+id/status_image"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentRight="true"
android:layout_marginRight="10dp" />
```

Menu

Main.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:yourapp="http://schemas.android.com/apk/res-auto" >

      <item android:id="@+id/menu_new_form"
          android:title=""
           android:icon="@drawable/detail"
          android:showAsAction="ifRoom|withText">
        <menu>
        <item
        android:id="@+id/settings"
        android:orderInCategory="101"
        android:showAsAction="withText|always"
        android:icon="@drawable/settings"
        android:title="@string/menu_settings"/>

       <item
        android:id="@+id/sync"
        android:orderInCategory="101"
        android:showAsAction="withText|always"
        android:icon="@drawable/search"
        android:title="@string/menu_sync_now"/>

        <item
        android:id="@+id/logout"
        android:orderInCategory="101"
        android:showAsAction="ifRoom|withText"
        android:icon="@drawable/logout"
        android:title="@string/menu_logout"/>
       </menu>
    </item>
</menu>
```