G52GRP Final Report

Proximity Based Attendance Management via Bluetooth Low Energy

Supervisor: Dr. Radu Muschevici

Name	**Student ID**
Khor Yoong Joo	20208420
Rahul Sharma Yuddhishthir-Gopaul	20208621
Foo Yao Chong	20012508
Gilbert Valentino	20209463
Sharfa Dhamin	20213131
Syed Ahmad Azhad Bin Ahmad Rosehaizat	20220036

Contents

1 Abstract

The aim of this project is to develop an Android app which uses proximity range detection to register the attendance of students in classrooms. Imagine a student entering a classroom, as the student with a smartphone moves into the proximity Bluetooth range of another student's phone when they have already registered their attendance, their phone will automatically connect to the nearby student's phone. This will trigger the user's automatic attendance recording system. This cascade of registrations will initiate with the course lecturer opening the course for registration and a first round of students getting registered by coming into the range of the lecturer's phone. This first round of students will then be able to act as proxy for registering other students, so that not every single student has to enter the lecturer's Bluetooth transmission range (typically around 10m).

The app will use Bluetooth Low Energy for proximity detection of nearby smartphones, and for data exchange. This project attempts to leverage a technology that lets phones within a certain location collaborate to perform certain tasks. One current application of this technology is contact tracing of COVID-19 cases.

2 Introduction

2.1 Problem Description

University of Nottingham Malaysia has a mobile application called *UNM Instatt* to aid lecturers and students take attendance during academic sessions. Instead of signing on paper, students will press an icon within a limited time frame to indicate themselves as present after the lecturer has turned on the signing-in feature. Students will get to monitor their attendance rate and check their timetable schedule in the application and likewise for lecturers.

However, some students might forget to sign-in even though they attended the class. In that case, lecturers need to sign in manually for them. There is also no notification when attendance is opened to be signed and students might miss the signing time frame. The process of attendance taking is also disruptive to the academic process as students need to open the application every time to record attendance.

2.2 Proposed Solutions

This project aims to create an application called Beam that can automatically record attendance of students in class. This application will run in the background and use Bluetooth Low Energy (BLE) to detect the students' proximity to lecturer or students who already recorded attendance. Once the lecturer enabled sign-in of attendance on his device, that device will send a token through BLE to other BLE-enabled devices nearby. The application will communicate with a server once the token has been received and the server will record attendance for the students. Once the student's attendance is recorded, the student's device will act as a beacon which shares the lecturer's token to nearby devices so that students whose devices are out of the lecture's device Bluetooth range could receive the lecturer's token. There will be a separate website which handles the timetable scheduling and registration of lecturer and student details.

Furthermore, the application will notify students to turn on Bluetooth once attendance is open to sign in. When the device received a token from the lecturer or from other present students, the application will record timestamps to ensure attendance record matches the actual timetable.

2.3 Qualifications and Limitations

One of the limitations of this application is that it could not detect proxy devices. Students who will not attend class can just hand their devices over to their friends who will attend to sign their attendance. Unless the lecturer counts the number of physical students in the class and tally with the number of students in the attendance record, which is a time-consuming process, students can take attendance with the application on behalf of their friends. If there is a separate device, such as an infrared sensor, that can detect the number of physical students in class automatically, then proxies can be eliminated, albeit at a high monetary cost. Furthermore, students who have already taken attendance can bring their devices to their friends outside the classroom to record attendance for them. Some students may turn off Bluetooth after taking attendance to save on battery consumption. Without Bluetooth, the device cannot act as a beacon for other students' who have not taken attendance.

2.4 Development Constraints

Since this project is not commissioned by University of Nottingham Malaysia (UNM), the application will be linked with an artificial database instead of the real UNM student registry. Even though the data used in this project does not identify with any real user, the data processed will be encrypted as a good practice of data handling.

Testing cannot be performed in the university making it significantly harder to test for security requirements such as leakage of personal data to unauthorised parties amongst others.

The use of an artificial database and the fact that the app will not be used by multiple students means that the app will never be tested on many devices making it considerably harder to test for several performance requirements, for instance how smooth the app runs on different Android devices or different networks.

2.5 Target Audience

This project aims to improve the existing attendance management application of *UNM Instatt*. Hence, this application is created for UNM lectures and students.

3 Literature Review

3.1 Using Android Devices and BLE

There are some existing solutions that utilizes BLE to record attendance. One of them is to attach a sensor to each students' identity card which could interact with the application installed in the lecturer's Android device. This sensor contains a unique string that can be associated with the student card it is attached to. During class, the lecturer will open the application to scan the sensors to collect the students' data into the application. To avoid attendance taken by proxy (students carrying more than one identity card or students standing outside the classroom during class), an infrared sensor could be installed in each classroom at the correct location and angle that can count the number of students who are physically present in the class. If the number of present students recorded by the application does not match the data recorded by the infrared sensor, the lecturer will receive an error message from the application [1].

Another way is to install BLE beacon in each classroom that transmit a "magic number" to each nearby Android devices with the required application installed. Through a web-based attendance management system, the lecturer could set the ID and name of the class prior to the class. During class, the lecturer will turn on the BLE beacon to send a "magic number" to students' Android devices nearby. Furthermore, the lecturer also turns on registration for attendance on the web-based attendance management system, which offers some basic management features such as list of attendance rate and records and import, export, and manual alteration of attendance records. The application installed in students' device allows the students to scan their student identity cards through Near Field Communication (NFC) reader. It is also not necessary to scan their cards with their own device and any device with the application installed could scan their student card. Meanwhile, the Android devices will receive the "magic number" from the BLE beacon. To record students' attendance, the application will send the "magic number" and the scanned student card and name to the server [2].

Other than that, Bluetooth beacons can be installed to each room so that students' device can scan their presence and extract the beacon's UUID. The students will install an application which logs them into their respective user account and scan their surroundings in search for Bluetooth beacon's UUIDs. Each room will be represented by a beacon's UUID. The administrator server will then match the extracted UUID, retrieval timestamp, and user account, all which are sent from the application installed in the student's device, with the database to check if the student is present in the correct room during academic sessions [3].

3.2 Using BLE Indoor Positioning Technology

There is also an alternative method of attendance management which does not require any mobile application. The university could install four Bluetooth station modules in strategic locations in a room. The Bluetooth module is a sensor system programmed in Python over Raspberry Pi which is also equipped with Bluetooth USB dongles. They are installed in the walls of each room to connect with powerlines and the Local Area Network (LAN). In this system, the lecturer could set the classes and list of students on the web-based attendance management system. On the other hand, students will also register the media access control (MAC) address of their device on the web-based system.

During academic sessions, the web-based system will send the list of student MAC address based on the timetable registered by the lecturer to the Bluetooth modules. The Bluetooth modules will then scan for Bluetooth devices nearby and detect their RSSI and MAC address.

The RSSI data will be processed using fingerprint localization method based on Artificial Neural Network (ANN) to estimate the location of the student in the classroom and the MAC address can be associated with a student. The data collected will be sent to the server and matched with the records in the database to register the students' attendance [4].

3.3 Android Bluetooth Low Energy

Android introduced Bluetooth Low Energy (BLE) functionality in Android 4.3 (API level 18). BLE has a lower power consumption in comparison to Classic Bluetooth. The transfer of data (known as attributes) between two BLE capable devices is based around the Generic Attribute Profile (GATT) which is built on top of the Attribute Protocol (ATT) [5].

ATT defines a standard protocol for attribute transfer between BLE devices by defining how attributes are formatted for transfer. Each attribute is uniquely identified by a Universally Unique Identifier (UUID) which is a standardised 128-bit format for a string ID. Attributes that are transferred are formatted as either services, characteristics, or descriptors. A service is a collection of characteristics. A characteristic contains a single value and any number of descriptors which are attributes for describing the characteristic. These descriptors may specify the characteristic's use, minimum and maximum values, unit of measurement, etc. A GATT profile specifies what kind of attributes are transferred. Bluetooth SIG provides existing profiles (such as Alert Notification Profile and Heart Rate Profile) for common BLE devices, but custom GATT profiles may also be written by developers. Custom GATT profiles defines a new service and characteristics of the service.

Android documentation specifies roles taken by two BLE capable devices that are connected. Regarding making a connection between two BLE devices, there exists two roles, the central and peripheral roles. The device that acts as the central scans for devices by looking for advertisements of services. Once a device is found, the central device is responsible for initiating a connection. The peripheral device is responsible for sending out advertisements of its services defined by a GATT profile. After the connection is created, the devices take on two new roles, either GATT client or GATT server. Usually, the central device takes the client role while the peripheral takes the server role. The GATT client is responsible for making requests to read or write into the characteristics inside the server's service which are defined by a GATT profile. The GATT server is responsible for notifying the client of characteristic changes and responding to client requests.

3.4 Firebase

Firebase is a platform developed by Google for creating and maintaining mobile and web applications as their Flagship product for app development [6]. There are 18 products available split into 3 groups, namely Develop, Quality and Grow. For our application we are using Firebase to host our application as well as the Authentication and Realtime Database features available on the firebase platform.

We decided to use Firebase Hosting to host our app not only because it's free, but also because of the simplicity of using Firebase to understand our backend statistics such as the amount of data downloaded, the amount of storage used as well as the ease of using the built-in authentication features available. Since Firebase is cloud-hosted and free, it is both cost effective and the risk of losing data is minimal.

Firebase Authentication provides a way for developers to identify the users who access the application or website. There are many different types of authentication but project we have used the "email and password" based authentication type. We have decided to make use of a function that allows users to create accounts either on the firebase console itself or in their mobile apps. Developers or System Administrators update or register data on the Firebase

Realtime Database after successfully signing into the website. Every user account in the system has a corresponding User ID whether they are a lecturer or a student. Admin accounts also have User IDs but they are not useful in this project.

Firebase Realtime Database is a NoSQL cloud-hosted database whereby data as JavaScript Object Notation (JSON) tree and can be synchronised in realtime to every connected client. The database is structured as a JSON tree with parent and child nodes with their respective keys. For cross-platforms apps (e.g. Android and Web), every client shares the same data instance and simultaneously receive the same updates made in the database [7]. We keep the data denormalized in the database so that the data can be downloaded efficiently as separate queries instead of fetching all the data nested in a particular location. This is because when client fetches data, all the child nodes are retrieved. If too much data is nested in a location, the client might end up downloading data which is not needed. Granting users read and write access to a location also grants read and write access to all child nodes, so we try to keep the data structure as flat as possible [8].

Note that all the Firebase Services used within this project encrypt data in transit with HTTPS and in the Google Servers [9].

3.5 Vue.js

Vue is a framework for building user interface of a website and is especially useful in creating single-page application (SPA). Developers can write templates, which are based in HTML syntax, and bind them with the components instance data of a website. The component templates most useful in this project is a type called x-template, which allows the developer to write HTML code inside a script tag [10]. Instead of creating multiple html files and redirects, they can be written in a single JavaScript file or a single HTML file under script tags. They can also create a navigation bar and allow users to switch between different templates by using the Vue-router-library. Routes can be created as paths to templates so that the router can link to the items in the navigation bar to their respective templates [11]. The main purpose of using Vue in this project is to allow the user to access multiple templates without refreshing the webpage.

3.6 Selenium Browser Automation

Selenium WebDriver simulates how a real user would use a browser, on local or remote machines. It works with all major browsers, including Chrome, Edge, Firefox, etc. This WebDriver also refers both to the language binding and implementations of the browser controlling code. It is an objected-oriented API designed as a compact programming interface. In this project, we will use Python to implement the WebDriver and create a script to drive browser automation. The script will be used to populate the database and test the functionalities of the administration website.

4 Requirements Specification

4.1 Functional Requirements

a. Authentication

i. Application will redirect user to login screen if user is unauthenticated

ii. User authentication state is saved and remains outside of app lifecycle

iii. Users authenticated using student credentials will enter *student mode*

iv. Users authenticated using lecturer credentials will enter *lecturer mode*

b. Display and Navigation

i. Within the main screen, users can swipe left and right to navigate between 4 different screens

ii. Within the 1$^{\text{st}}$ screen, users can view a pulsing animation of the BEAM logo

iii. Within the 2$^{\text{nd}}$ screen, users can view the daily schedule of sessions they'll be teaching or attending

iv. Within the 3$^{\text{rd}}$ screen, users can view the weekly schedule of sessions they'll be teaching or attending

v. Within the 4$^{\text{th}}$ screen, users can view attendance statistics for the module they're teaching or enrolled in

vi. Users can press the settings icon on the top right to navigate to a settings screen

vii. Within the settings screen, the user can logout delete existing authentication state

c. Lecturer Mode

i. On arrival at main screen, app will schedule background services for opening attendance when each session begins (Updating database, sending out attendance tokens)

ii. On arrival at main screen, app will schedule background services for closing attendance when each session ends (Updating database)

iii. During background service for opening attendance, lecturer will be notified that the device is sending out attendance tokens to other devices

iv. By pressing a row containing a session, lecturer can view the attendance statistics of a particular session

v. By pressing a module in the 4$^{\text{th}}$ screen, lecturer can view average attendance percentage of students enrolled

d. Student Mode

i. On arrival at main screen, app will schedule background services for taking attendance when each session begins

ii. During background service for taking attendance, user will be notified that the device is scanning for other devices and receiving tokens

iii. On successful taking of attendance, user is notified that attendance has been taken

iv. On successful taking of attendance, device switches to sending out tokens to other devices and user is notified of this

v. By pressing a row containing a session or module, student can view their detailed attendance history of the module

e. Bluetooth Requirements

i. App can open a GATT server to advertise a custom GATT Service

ii. App can scan for BLE devices that advertise a custom GATT Service

iii. App can read characteristics of the advertised service from other BLE devices

iv. App runs all Bluetooth functionality as a background service (doesn't require app to be open)

f. Administration Website

i. This is a website built to initialise the Firebase database with lecturer details, attendance record of each academic sessions by module, academic sessions of each modules, module details, academic plan consists of sets of modules, student details, attendance record of each student by academic module and academic session, timetable, and account user details.

ii. This website aims to simulate a university administration site where the admins can access and update the database.

iii. The landing page of this website only has only feature: admin account login.

iv. Successful logins will redirect the user to a single-page application with four main features: Student Registration, Lecturer Registration, Add or Remove Module, Update Timetable.

v. All four main features will load on the same page without any page refresh and can be accessed via the navigation bar.

g. Firebase Realtime Database

i. The system should store student data which can queried using student authentication account's UID. The data shall consist of first name, last name, programme, and email.

ii. The system should store lecturer data which can queried using lecturer's authentication account's UID. The data shall consist of first name, last name, faculty, position, and email.

iii. The system should store module data which can queried using module id. The data shall consist of module name, lecture ids and student ids.

iv. The system should store academic plan data which can queried using programme. The data shall consist of the module ids of the modules of a programme.

v. The system should store timetable data which can queried using date (YYYYMMDD). Academic sessions are recorded by module id and have details such as session type, status, time begin, and time end.

vi. The system should store attendance record data which can queried using module id and session id.

vii. The system should group each academic session by module id.

viii. The system should store the academic sessions attended by a student, grouped by module id.

ix. The system stores each user account details including programme, user role, and modules.

4.2 Non-Functional Requirements

a. Development Environment

i. Java SE Development Kit 8 will be the main programming language used for implementing app functionality

ii. XML will be used for defining views and layouts for the app

iii. Gradle build tools will be used for managing dependencies such as the Android SDK and Firebase API

iv. Android Studio will be the IDE used for developing the app

b. Application Dependencies

i. Android Software Development Kit will be used for developing the app

ii. Firebase API will be used for user authentication and storing and retrieving records in a cloud database

c. Availability

i. App will support Android devices with Android 5.0 or above

ii. App will only run on devices connected to the Internet

iii. App will only run on devices with Bluetooth enabled

d. Security Requirements

i. Data transmitted between Firebase servers and the app should be encrypted with HTTPS

ii. Data stored in servers should be encrypted

iii. Password should be hidden on the interface

iv. Only attendance tokens are transferred between devices and received from other devices for the operation of the app

e. Performance Requirements

i. App should open in 2 seconds after the user clicks on the app icon.

ii. UI frames must not take longer than 700ms to render.

iii. App UI should respond to user input in 200ms.

5 Final Design

5.1 Scheduled Attendance Taking Services

The attendance taking functionality will be implemented as Android background services that are scheduled to run at the time the session will start. Four different services will be implemented for each of the following functions: opening attendance, closing attendance, advertising attendance tokens, and scanning for devices advertising attendance tokens. The first two services will only run on a device with lecturer credentials. On app startup, the services will be scheduled by an Android AlarmManager.

5.2 Custom GATT Profile

A custom GATT profile (dubbed Beam Profile) will be utilised for data transfer between two devices using the app. The Beam Profile contains one custom GATT service (Beam Service) which contains one characteristic, known as Attendance Token, that holds a string value generated as the hash of a lecture session ID. The value of this characteristic can be read by the GATT client, written by the GATT server, and the GATT client can be notified of changes in the characteristic.

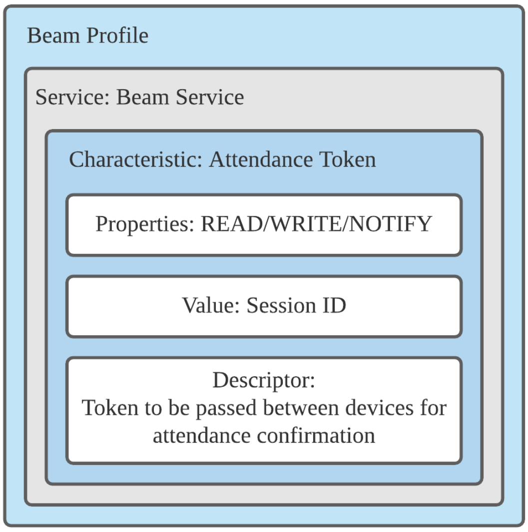


Figure 5.2.1: Custom GATT Profile

Regarding the peripheral role in the app, the lecturer's device acts as the first peripheral device. The session ID will be written into the service characteristic. The device will then advertise the Beam Service UUID and wait for connection request from a central device. Once a connection is established, the peripheral acts as the GATT server. Whenever the central device (GATT client) requests to read the value of the characteristic, the server will send a response containing the value. This is how the attendance token is passed between devices.

Regarding the central role in the app, the students' devices are central devices by default. Once the session has started (based on OS time), the central device will begin scanning for devices that advertise the Beam Service UUID. Once it detects one, it'll initiate a connection and begin acting as the GATT client. The device will immediately request to read the value of the characteristic within the service. Upon receiving the attendance token, the value is compared to that in the database. Once attendance has been taken, the device closes the connection and switches to a peripheral role.

5.3 Database and Web Hosting Choice

The database used by the app will be Firebase Realtime Database, which is a cloud hosted database whereby data is stored as a JSON tree. All clients share one Realtime Database instance. The data stored by the app is estimated to take up a few GBs of space and only basic querying is required by the app. Additionally, only one instance of the database is needed for all clients. Thus, Firebase recommended Realtime Database as a better choice compared to Firebase Cloud Firestore, a NoSQL relational database, which is better suited for apps that require multiple databases, advanced querying, and hundreds of GBs to TBs of space. Firebase was chosen over other web services (such as Amazon Web Services) because it's hosting services are free (to a certain degree) and the API is estimated to be easier to learn and work with.

The website is hosted using Firebase Hosting: `https://beam-5845a.web.app/`. Email for authentication is admin@nottingham.edu.my and the password is *password*.

5.4 Firebase Authentication Design

There are three types of user accounts: student, lecturer, and admin. Student and lecturer accounts grant permission to access the application, while admin accounts grant access to the administration website. Each student and lecturer accounts will have their own User ID stored in Firebase Authentication. The User ID will be used to query the database to search to student or lecturer details. The admin accounts are created on the console by the team and cannot be created in other way.

5.5 User Interface Navigation

a. Website

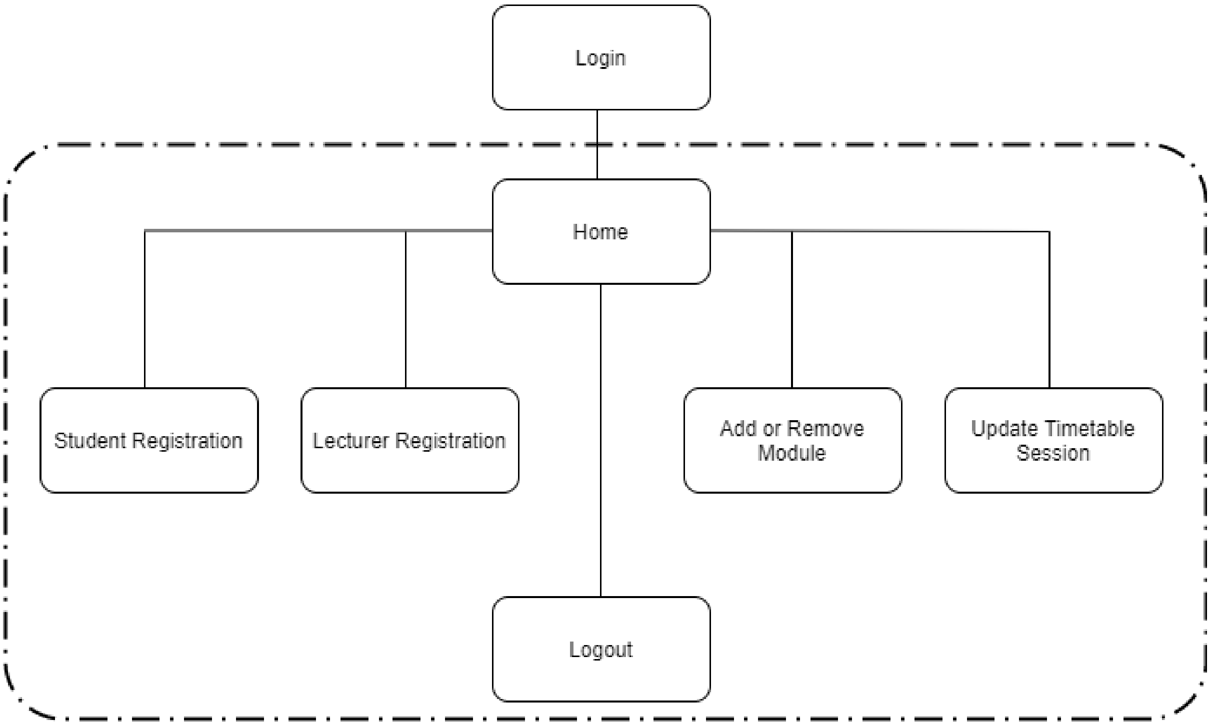
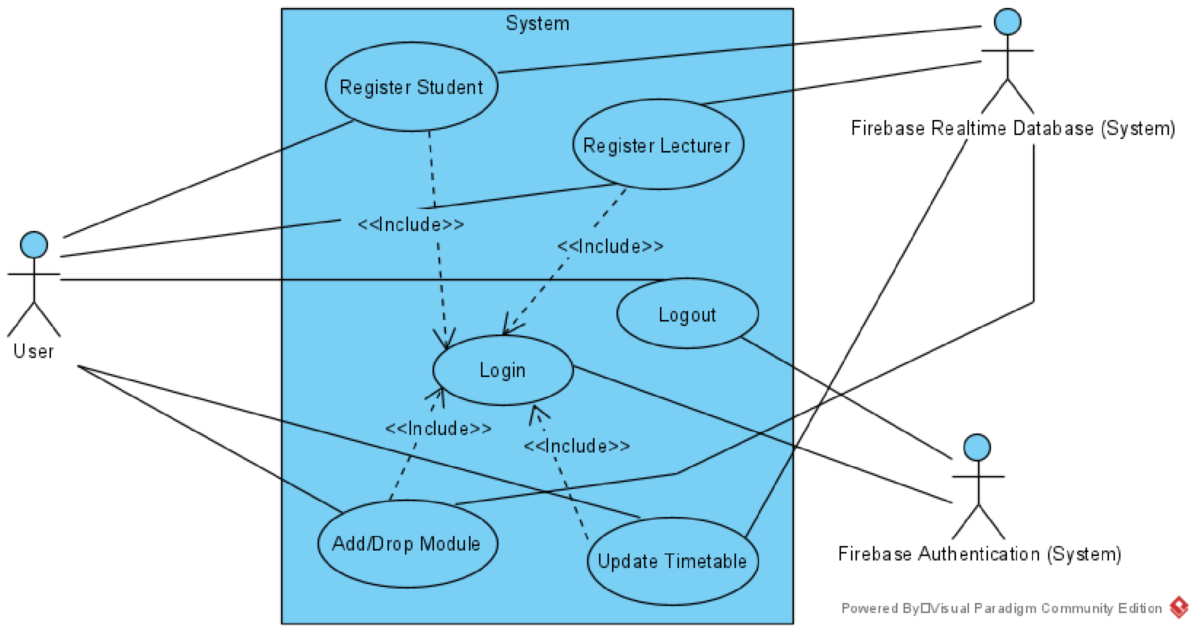


Figure 5.5.1: Sitemap of administration website. Components inside the dotted box belong to a single-page application

b. Application

5.6 Use Case Diagrams

a. Website



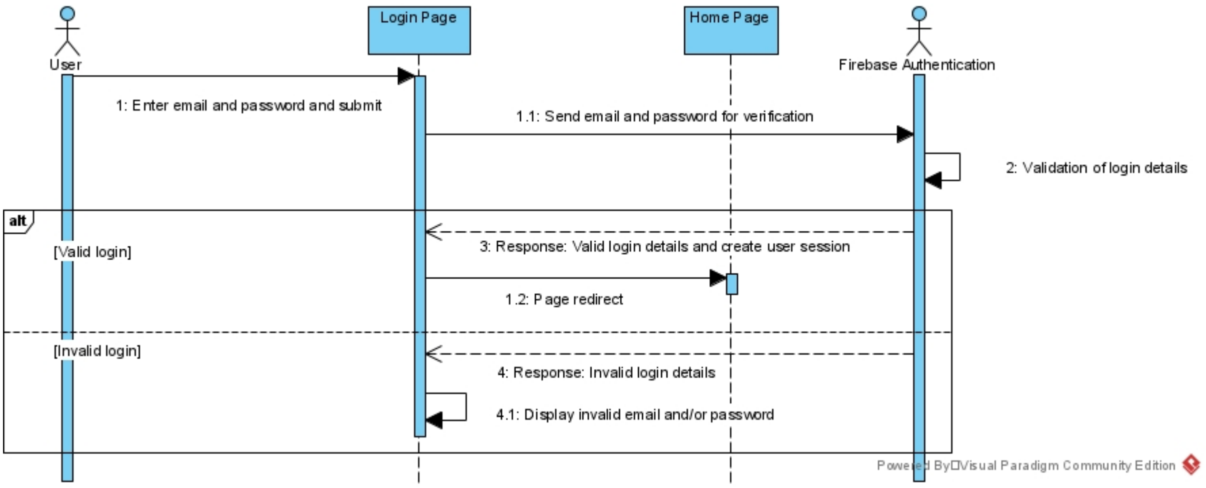
b. Application

TODO

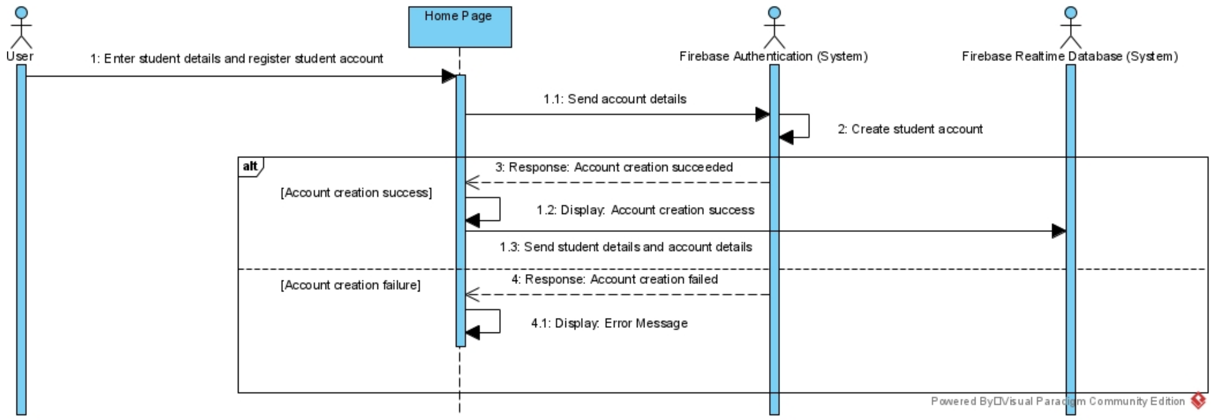
5.7 Sequence Diagrams

a. Website

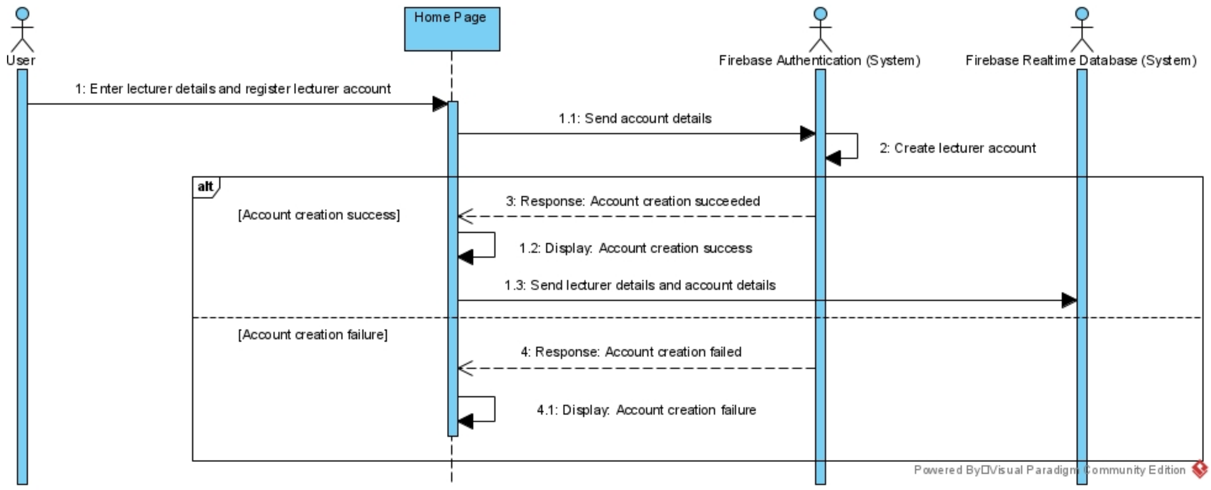
i. Login



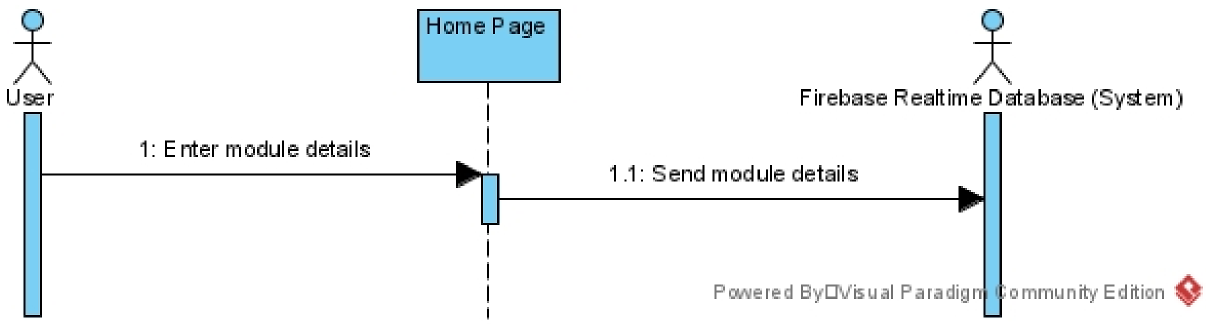
ii. Student Registration



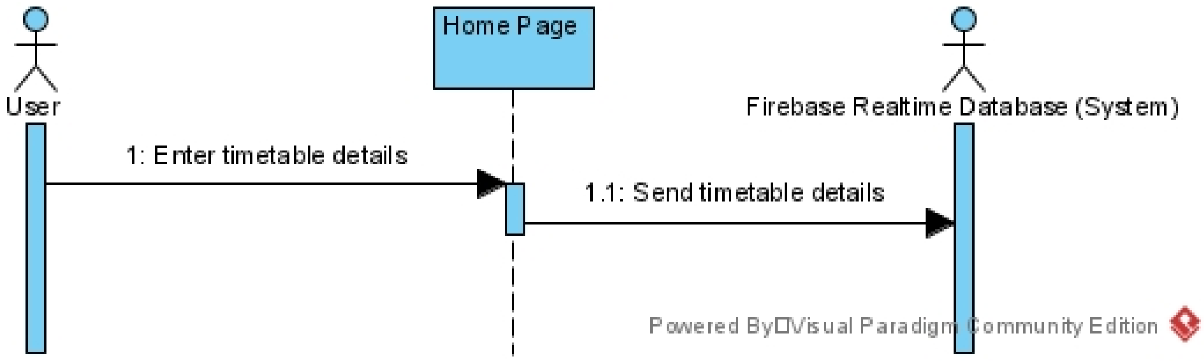
iii. Lecturer Registration



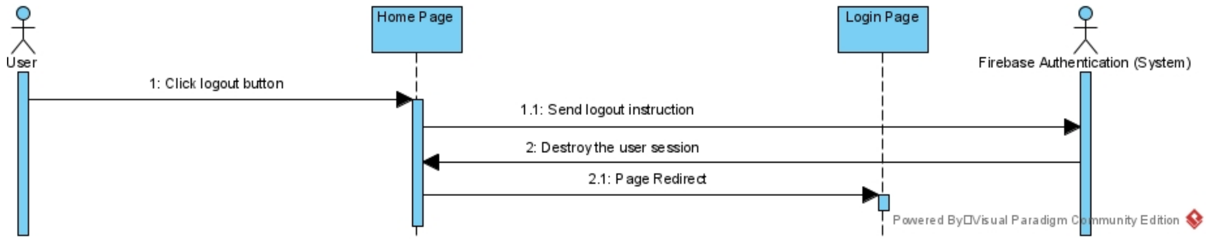
iv. Update Module Information



v. Update Timetable

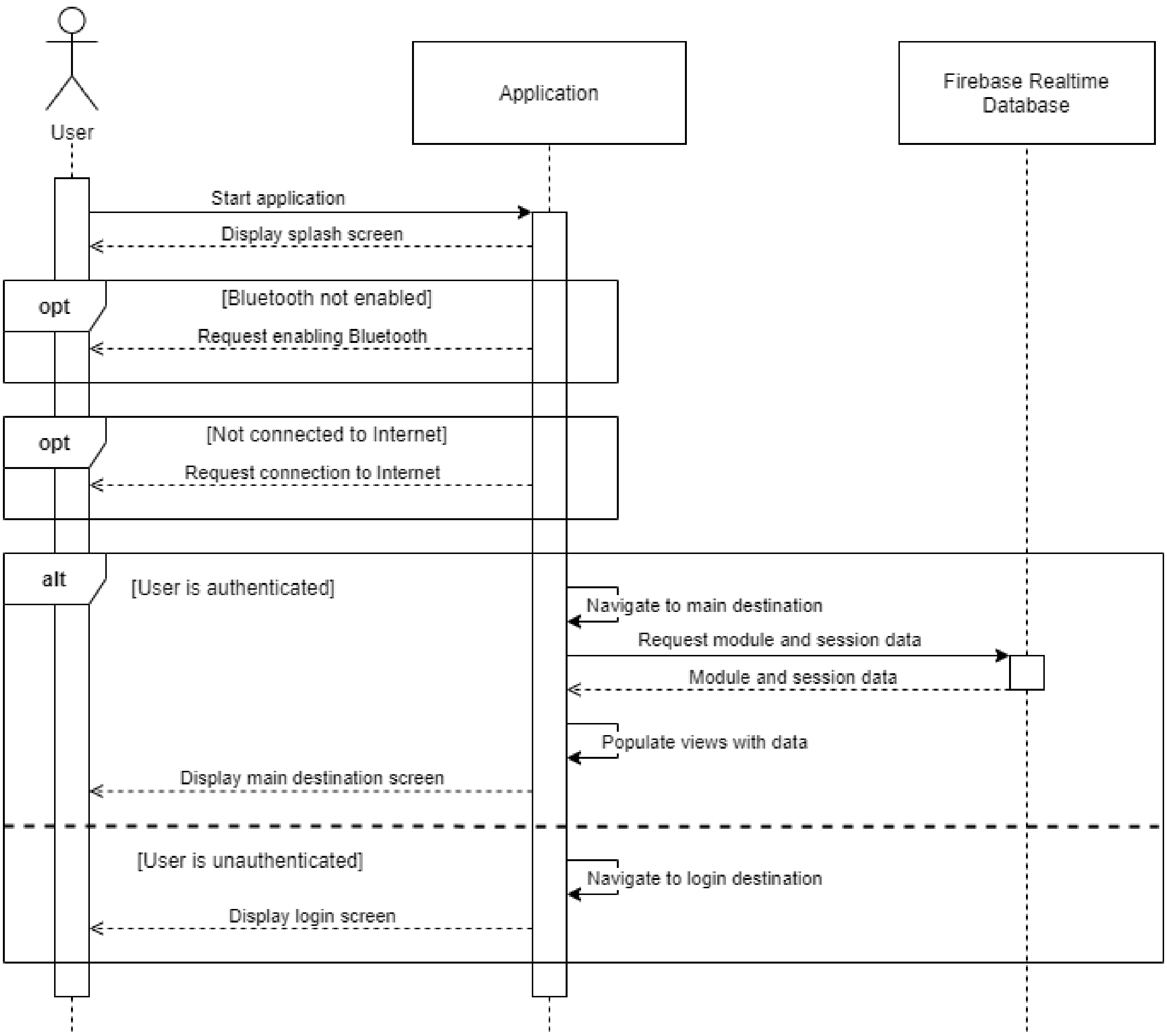


vi. Logout

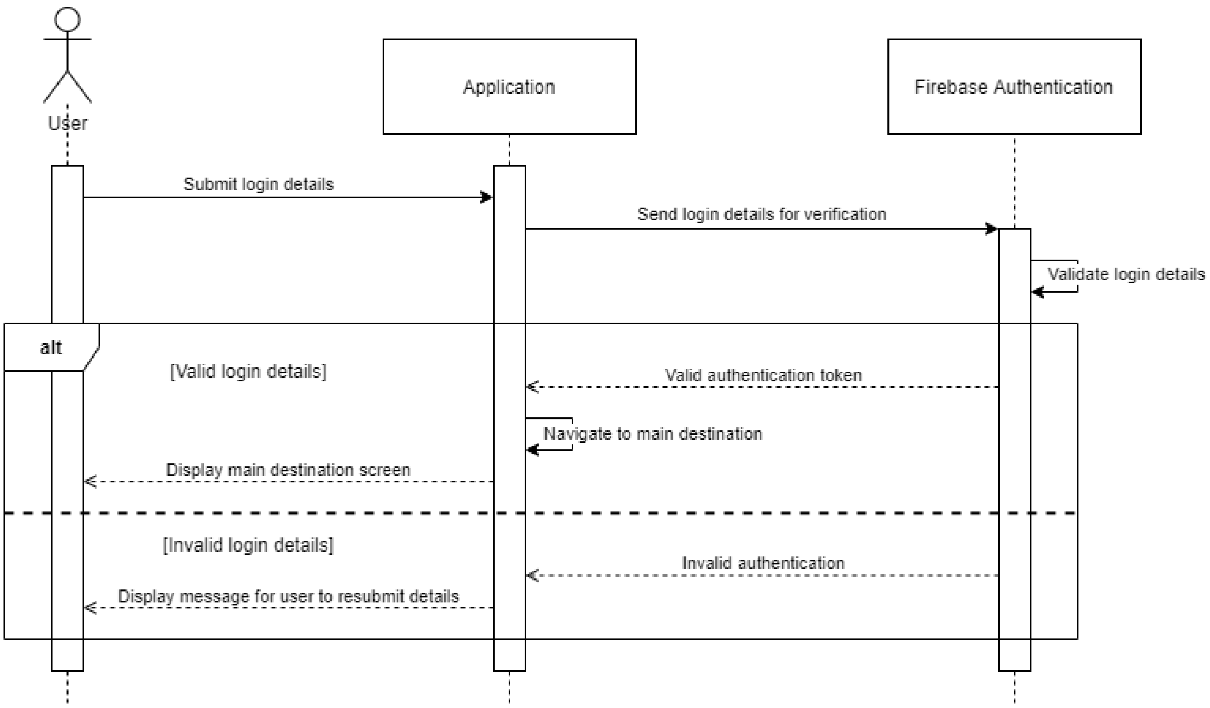


b. Application

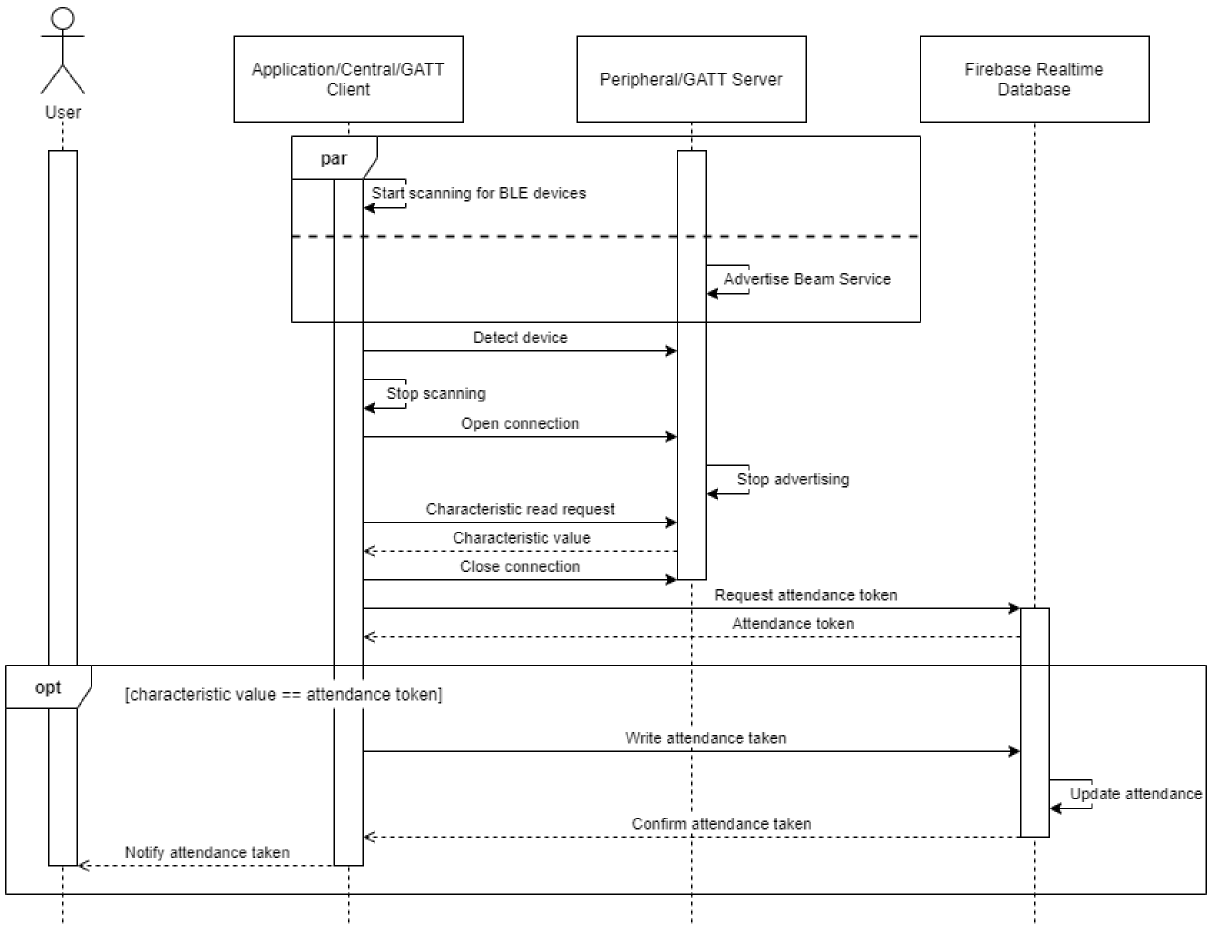
i. Application Start


User
Application
Firebase Realtime Database
Start application
Display splash screen
opt
[Bluetooth not enabled]
Request enabling Bluetooth
opt
[Not connected to Internet]
Request connection to Internet
alt
[User is authenticated]
Navigate to main destination
Request module and session data
Module and session data
Populate views with data
Display main destination screen
[User is unauthenticated]
Navigate to login destination
Display login screen

ii. Login



iii. Attendance Taking



5.8 Firebase Realtime Database Design

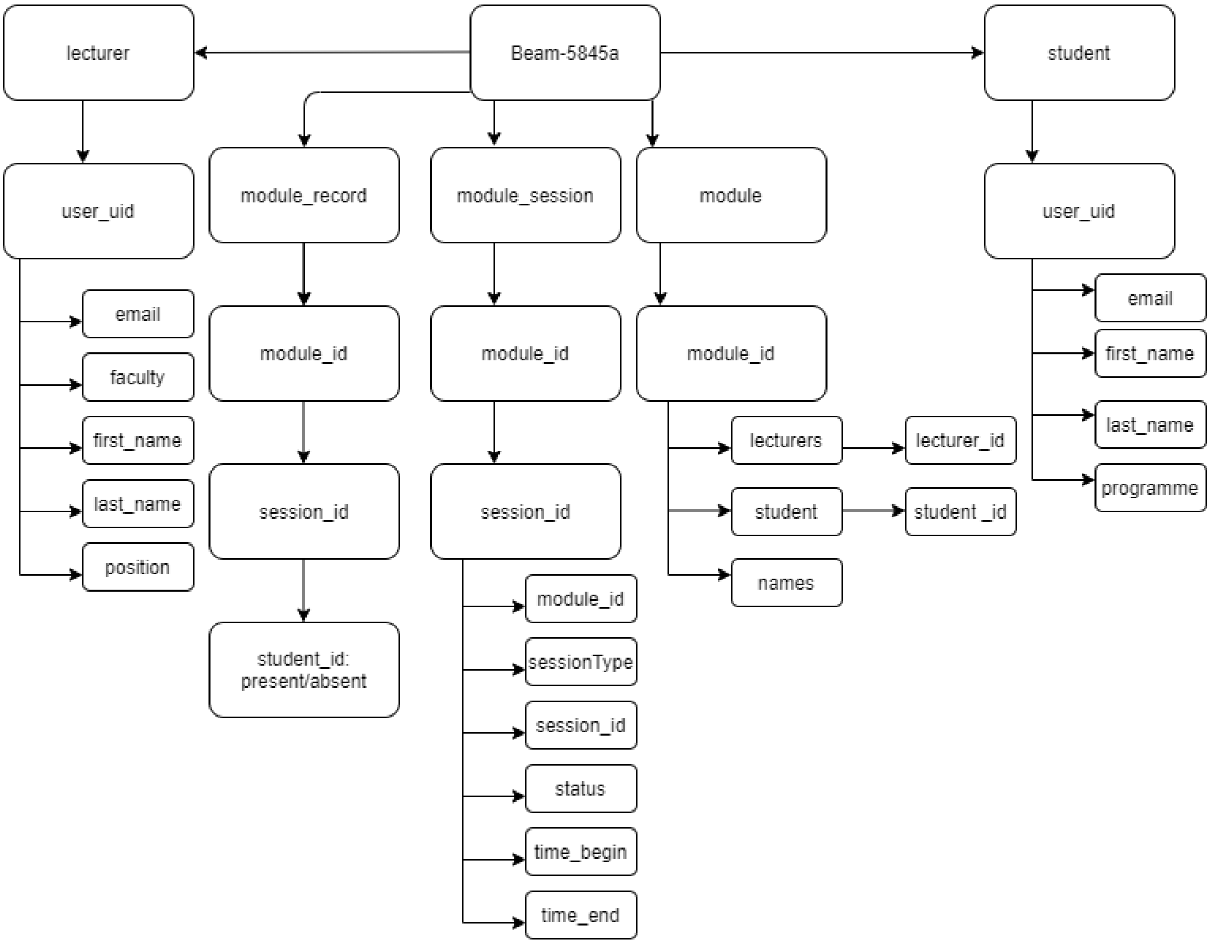


Figure 5.8.1: Database design part 1

The *lecturer* node groups the lecturer details by *user_uid*, which will be fetched to populate the profile in the app. The *module_record* node stores the students who attended an academic session by *module_id*. The app will calculate the percentage of students who attended the session. The *module_session* node groups each session by their module. The *module* node groups module details by *module_id*. The *student* node groups the student details by *user_uid*, which will be fetched to populate the profile in the app.

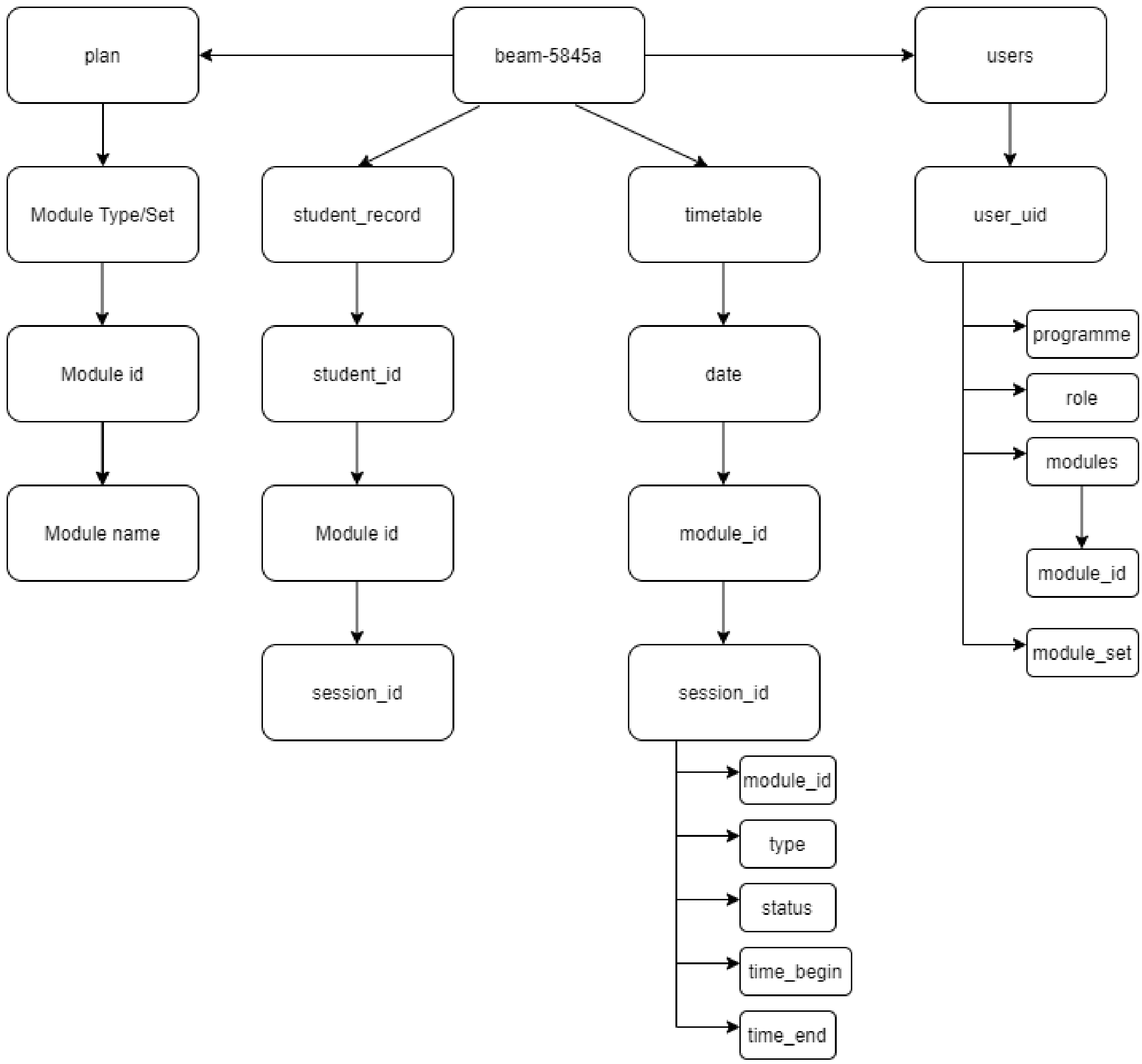


Figure 5.8.2: Database design part 2

The *plan* node stores the modules by programme. Modules in a programme are group by module type or set. Module type such as Core and Elective are applicable for students, while module set groups the modules taught by a lecturer. Grouping modules in this way allows the admin to perform only one data entry (by type or set) instead of entering each module one-by-one. The *student_record* node stores all attended academic sessions of a student, grouped by module id. The *timetable* node stores each academic session and its module id for every date when there is an academic session. The *user* node stores important identification details. The *programme* node groups all the modules a student is taking in an academic plan. The *role* node identifies whether a user is a student or lecturer. The *module* set groups all the modules a lecturer is teaching.

6 Testing Methodology

6.1 Website

Selenum WebDriver will be used to run test suites written in Python, which also populate the database of the attendance management system and automate the data entry process, in Microsoft Edge Browser.

```
def login():
    driver.find_element_by_id("email").send_keys("hcyyk1@nottingham.admin")
    driver.find_element_by_id("password").send_keys("123456")
    driver.find_element_by_id("login").click()
```

This function tests the login function on the landing page. The script enters email and password before clicking the login button to submit the user credentials.

```
def student():
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'Student')))
    driver.find_element_by_id("Student").click()

    for x in range(10):
        driver.find_element_by_id("fName").send_keys(get_first_name())
        driver.find_element_by_id("lName").send_keys(get_last_name())

        programme = ['Computer Science', 'Business', 'Engineering']
        driver.find_element_by_id("programme").send_keys(random.choice(programme))

        driver.find_element_by_id("email").send_keys(random_char(7) + "@nottingham.edu.my")
        driver.find_element_by_id("password").send_keys("123456")

        driver.find_element_by_id("submit").click()
```

This function registers 10 student accounts. It waits for the student tab to become clickable before clicking it. The script will enter the first name, last name, choose the student's programme randomly, enter email and password, and submit all details, for 10 times.

```
def lecturer():
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'Lecturer')))
    driver.find_element_by_id("Lecturer").click()

    for x in range(10):
        driver.find_element_by_id("LfName").send_keys(get_first_name())
        driver.find_element_by_id("LlName").send_keys(get_last_name())

        select_pos = Select(driver.find_element_by_id('position'))
        position = ['Assistant Professor', 'Associate Professor', 'Professor']
        select_pos.select_by_value(random.choice(position))

        select_fac = Select(driver.find_element_by_id('faculty'))
        faculty = ['Computer Science', 'Business', 'Engineering']
        select_fac.select_by_value(random.choice(faculty))

        module_set = ['Set1', 'Set2']
        driver.find_element_by_id("set").send_keys(random.choice(module_set))

```

```python
        driver.find_element_by_id("Lemail").send_keys(random_char(7) + "@nottingham.edu.my")
        driver.find_element_by_id("Lpassword").send_keys("123456")

        driver.find_element_by_id("Lsubmit").click()
```

This function registers 10 lecturer accounts. It waits for the lecturer tab to become clickable before clicking it. The script will enter the first name, last name, choose the lecturer's position randomly choose the lecturer's faculty randomly, choose the lecturer's taught module set randomly, enter email and password, and submit all details, for 10 times.

```python
def module():
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'Module')))
    driver.find_element_by_id("Module").click()
    module_codes1 = ["COMP1001", "COMP1002", "COMP1003", "COMP1004"]
    module_codes2 = ["BSC1001", "BSC1002", "BSC1003", "BSC1004"]
    module_codes3 = ["ENG1001", "ENG1002", "ENG1003", "ENG1004"]
    module_names1 = ["Computer Fundamentals", "Software Engineering", "Discrete Maths", "Software Maintenance"]
    module_names2 = ["Business Communication", "Leadership", "Business Principles", "Business Writing"]
    module_names3 = ["Thermodynamics", "Natural Mechanics", "Electricity", "Computer Skills"]
    module_selector("Computer Science", module_names1, module_codes1)
    module_selector("Business", module_names2, module_codes2)
    module_selector("Engineering", module_names3, module_codes3)
```

This function adds modules to the database. It waits for the module tab to become clickable before clicking it. The code snippet above contains all the module codes and names that will be entered on the website. The *module_selector* function will send the module details to the database, as shown below.

```python
def module_selector(plan, modules, module_code):
    for n in range(4):
        driver.find_element_by_id("mID").send_keys(module_code[n])

        driver.find_element_by_id("mName").send_keys(modules[n])

        driver.find_element_by_id("mPlan").send_keys(plan)

        select_type = Select(driver.find_element_by_id('moduleType'))
        module_type = ['Core', 'Elective']
        select_type.select_by_value(module_type[0])

        module_set = ['Set1', 'Set2']
        driver.find_element_by_id("mSet").send_keys(random.choice(module_set))

        driver.find_element_by_id("Add").click()
```

The function above sends the module id, module name, the programme of the module (plan), chooses the module type for the students (in this case all modules are 'Core', choose s the module set for the lecturers, and submit all the details.

```python
def timetable():
    date = 20210401
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'Timetable')))
    driver.find_element_by_id("Timetable").click()
    module_codes1 = ["COMP1001", "COMP1002", "COMP1003", "COMP1004"]
    module_codes2 = ["BSC1001", "BSC1002", "BSC1003", "BSC1004"]
    module_codes3 = ["ENG1001", "ENG1002", "ENG1003", "ENG1004"]
    for x in range(5):
        timetable_selector(module_codes1, date)
```

```
        timetable_selector(module_codes2, date)
        timetable_selector(module_codes3, date)
        date += 1
```

This function updates the timetable on the database. It waits for the timetable tab to become clickable before clicking it. The code snippet above contains all the module codes that represents. The *module_selector* function will send the module details to the database, as shown below, for 5 days: April 1^{st} 2021 to April 5^{th} 2021.

```
def timetable_selector(codes, date):
    for x in range(4):
        date_entry = str(date)
        driver.find_element_by_id("date").send_keys(date_entry)

        driver.find_element_by_id("module_id").send_keys(codes[x])

        session_type = ["Lecture", "Tutorial", "Lab"]
        driver.find_element_by_id("sessionType").send_keys(random.choice(
    session_type))

        driver.find_element_by_id("status").send_keys("Closed")

        if x < 2:
            time = ["0900", "1100", "1300"]
            driver.find_element_by_id("time_begin").send_keys(time[x])
            driver.find_element_by_id("time_end").send_keys(time[x + 1])
        else:
            time = ["1400", "1600", "1800"]
            driver.find_element_by_id("time_begin").send_keys(time[x - 2])
            driver.find_element_by_id("time_end").send_keys(time[x + 1 - 2])

        driver.find_element_by_id("Tsubmit").click()
        x += 1
```

This function enters the date, module id, chooses session type randomly, session status, time of each sessions, 4 in a day (0900-1100, 1100-1300, 1400-1600, 1600-1800) and submit the details.

```
def logout():
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, 'logout'
    )))
    driver.find_element_by_id("logout").click()
```

This function logs out the user from the website, after clicking the ‘Logout’ buton.

6.2 Application

Application testing was conducted in two different ways, testing via emulators and manual device testing using real devices. For functionalities such as the UI functionality and fetching from the Firebase Database, testing was done using emulators provided by Android Studio. For the UI, a test suite was created where each possible user action and its corresponding application response was tested. For database functionality, manual testing was conducted whereby data was inputted into the database (via the website), and the applications ability to fetch data and populate the UI was tested.

Regarding the testing of the attendance taking functionality, real life testing was conducted using a minimum of 2 Android devices. Attendance taking relies on Android background services and Bluetooth Low Energy connections between devices. Real life testing was conducted because Bluetooth is not available on emulators so physical devices must be used.

Regarding Android background services, testing was done by monitoring notifications posted by the app to the user. If the specific notification for the service was posted, the app's background services were functioning normally. A test suite was created for the scheduled background services where we monitored notifications to determine whether tests passed or failed. We tested whether the services started within the correct time frame and whether subsequent services were started.

A test suite was also created for app's Bluetooth functionality. There were some issues that occurred during testing. To monitor connections using Bluetooth Low Energy, Toast messages were used to monitor the connection state, devices scanned, and Bluetooth services discovered. However, Toast messages were an unreliable method for monitoring tests as not all Toast messages were displayed despite the app functioning properly. To solve this issue, tests were instead monitored using changes to the database. All Toast messages used for testing were converted to updates/writes to the database. By monitoring the database, we could see whether the Bluetooth functionality was working as expected.

7 Implementation Results

7.1 Website

a. Login Screen: index.html

b. Home Screen: home.html

7.2 Application

8 Conclusion

9 Reflective Comments

10 Appendix

11 References

[1] R. Apoorv and P. Mathur. "Smart attendance management using Bluetooth Low Energy and Android". In: *2016 IEEE Region 10 Conference (TENCON)*. 2016, pp. 1048–1052. DOI: 10.1109/TENCON.2016.7848166.

[2] S. Noguchi et al. "Student Attendance Management System with Bluetooth Low Energy Beacon and Android Devices". In: *2015 18th International Conference on Network-Based Information Systems*. 2015, pp. 710–713. DOI: 10.1109/NBiS.2015.109.

[3] Kok Loong Pang and Yuan Yew Choo. "METHOD FOR RECORDING ATTENDANCE USING BLUETOOTH ENABLED MOBILE DEVICES". Pat. 20200029173. 2020. URL: https://www.freepatentsonline.com/y2020/0029173.html.

[4] Apiruk Puckdeevongs et al. "Classroom Attendance Systems Based on Bluetooth Low Energy Indoor Positioning Technology for Smart Campus". In: *Information* 11.6 (2020). ISSN: 2078-2489. DOI: 10.3390/info11060329. URL: https://www.mdpi.com/2078-2489/11/6/329.

[5] Android Developers. *Bluetooth low energy*. Mar. 8, 2021. URL: https://developer.android.com/guide/topics/connectivity/bluetooth-le (visited on 04/19/2021).

[6] Crunchbase. *Firebase*. N.d. URL: https://www.crunchbase.com/organization/firebase (visited on 11/15/2020).

[7] Firebase. *Firebase Realtime Database*. Nov. 17, 2020. URL: https://firebase.google.com/docs/database (visited on 04/19/2021).

[8] Firebase. *Structure Your Database*. Apr. 15, 2021. URL: https://firebase.google.com/docs/database/web/structure-data (visited on 04/19/2021).

[9] Firebase. *Privacy and Security in Firebase*. Mar. 18, 2021. URL: https://firebase.google.com/support/privacy (visited on 04/19/2021).

[10] Gore Anthony. *7 Ways to Define a Component Template in Vue.js*. Feb. 3, 2020. URL: https://vuejsdevelopers.com/2017/03/24/vue-js-component-templates/ (visited on 04/19/2021).

[11] Vue.js. *Routing*. Apr. 1, 2021. URL: https://v3.vuejs.org/guide/routing.html#simple-routing-from-scratch (visited on 04/19/2021).