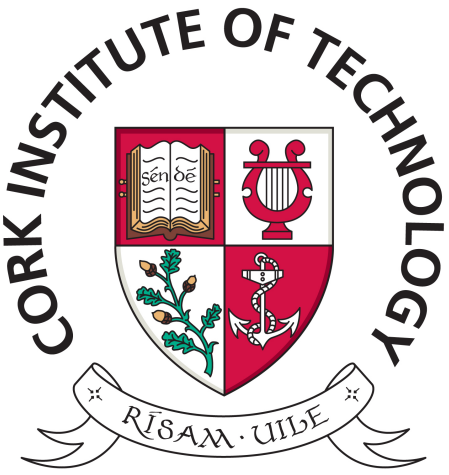
"Thesis" style, based on the ECS Thesis style by Steve Gunn



School attendance management system

by

Pavel Vasilev

This thesis has been submitted in partial fulfillment for the
degree of Bachelor of Science in Software Development

in the
Faculty of Engineering and Science
Department of Computer Science

May 2017

Declaration of Authorship

I, Pavel Vasilev, declare that this thesis titled, 'School attendance management system' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for an undergraduate degree at Cork Institute of Technology.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institiute of Technology or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

CORK INSTITUTE OF TECHNOLOGY

Abstract

Faculty of Engineering and Science
Department of Computer Science

Bachelor of Science

by Pavel Vasilev

Implementation project report based on fully automated school attendance system focused on the use of beacons.

Acknowledgements

I would like to express my special thanks of gratitude to my project supervisor Sean Mc Sweeney and Karl Grabe who gave me the golden opportunity to do this wonderful project, which also helped me in doing a lot of Research and I came to know about so many new things I am really thankful to him.

Contents

List of Figures

List of Tables

Abbreviations

SAMS **S**chool **A**ttendance **M**Management **S**ystem

Chapter 1

Introduction

This report follows on from the research report and presentation that outlined a particular strategy for implementing an automatic attendance management system(AAMS).

This report will give details on the initial design and modifications to that design, illustrate the issues that were foreseen and some that weren't and how it was possible to overcome them.

The report documents the implementation in the following headings :

1. Design Implementation
2. Web Application
3. Android Application
4. Cloud Database
5. Difficulties
6. Testing and Future Enhancements

1.1 Executive Summary

The project was very interesting, challenging and educational to complete. The project goal was to provide an automatic attendance management system.

The project was more like a proof of concept. The idea of the project is very unique and not implemented anywhere. There are similar projects, but none of them are using Beacons as a identification device.

The AAMS consist of 3 main components - Web Application, Android Application and a Cloud Database. In this report I will go through each of these components and outline how and why it was implemented that way.

The difficulties of the project were never foreseen during the research phase of the project. While the Web application of the project was fairly understood, the Android application was quiet underestimated.

Even the Web application proved to be quiet challenging, especially the framework selection. Due to the multiple frameworks available, deciding which one is most comfortable for your project and whether it is compatible with the rest of the components, required extraordinary time and testing.

While implementing the cloud database. This was my first involvement in this area of technologies. I have never before used real-time database, but it helped me acquire knowledge for the future.

Chapter 2

System Deployment

2.1 How to deploy/run ?

In this section I will outline in details how to build and deploy the system. I will elaborate for the 3 components and required information for deployment.

2.1.1 Web Application

Features

1. Polymer 1.0
2. Firebase
3. Polymerfire
4. Gulp build

Steps to follow :

1. Clone the project
   git clone https://github.com/pvasilev94/School-Attendance-System
2. cd website
3. Download and install nodejs.
   Required version is at or above 0.12.x. Nodejs is available - https://nodejs.org/en/

4. Install gulp and bower globally.
   Commands - 'npm install -g gulp bower

5. Install dependencies.
   Commands - 'npm install bower install

6. Build and vulcanize sources.
   Commands - 'gulp'

7. Run development web-server
   Commands - 'gulp-serve'

8. Run production web-server
   Commands - 'gulp-serve:dist'

2.1.2 Android Application

Features

1. Firebase
2. Jackson annontiations
3. Sign in
4. Sign up - added for testing purposes

Steps to follow :

1. Git clone the project
   Commands - 'git clone https://github.com/pvasilev94/School-Attendance-System
2. cd app
3. Run with Android Studio or IntelliJ
4. Download BlueVisionSDK Download the file from - http://developer.bluvision.com/developer/bee beacons-sdk/ and put it in app/libs, any other place would not be detected by the project.
5. Build the project and run on device.
6. APK file is available in the github repo

2.1.3 Cloud Database - Firebase

Features

1. NOSQL
2. Asynchronous
3. Accomodates any kind of scenarios
4. Allows scalability
5. Supports multiple platforms

Project is using my own account in the firebase console. In the case that you want to create your own version of the system.

Steps to follow :

1. Create a firebase account
2. Create a new project
3. Add application to the project
   In this section you have 3 options - Android, iOs, Web.
4. On selecting Android and Web save the config file provided by firebase.
5. Download the config - google-services.json and added in the place of the current ones in both web app and android app.
6. Make sure to enable authentication in the firebase console

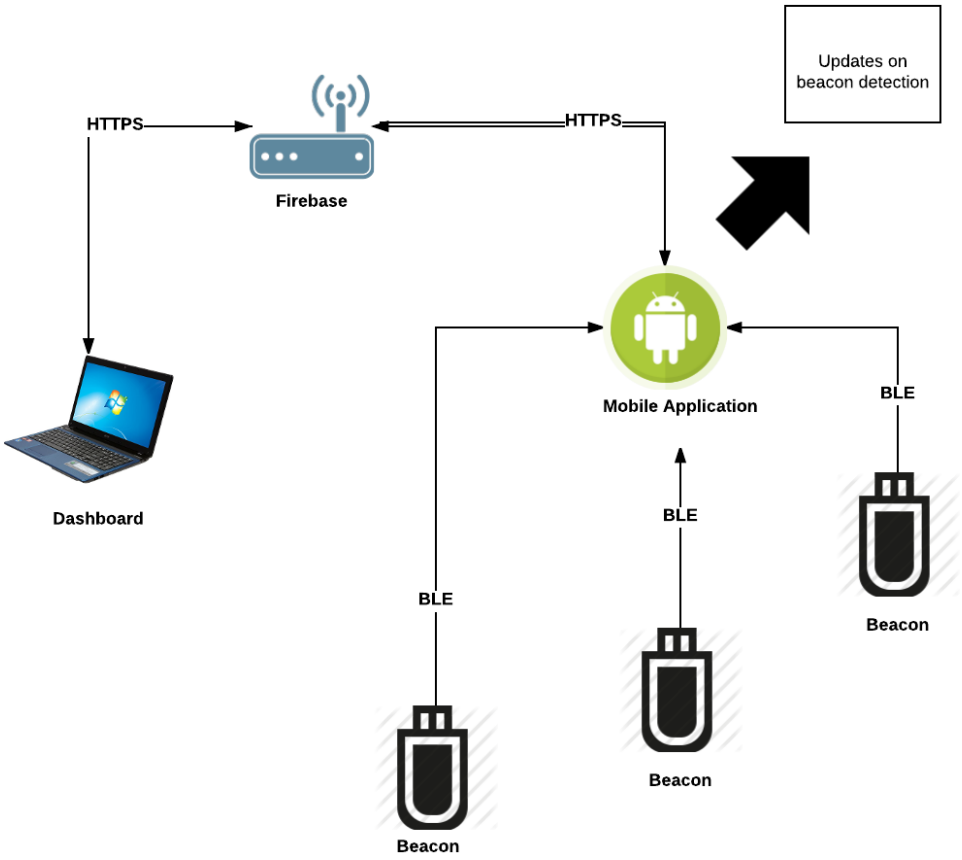
Chapter 3

Design Implementation

3.1 Overview

During the implementation phase, the original design architecture haven't changed. Most of the changes has been around the technologies part of each component, either more suitable technology was found or easier to work with.

FIGURE 3.1: Architecture Design



This chapter we will go through each component and discuss the implementation part. Each has it's own unique features.

3.2 Web Application

The web application component serves as a dashboard to display data and add authorization to the rest of the users.

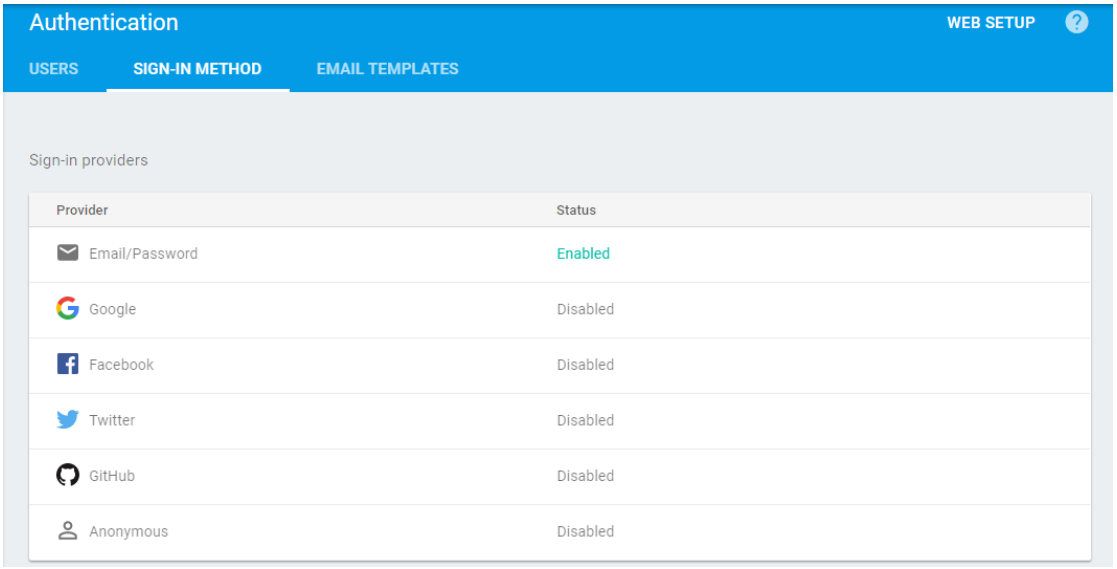
3.2.1 Configuration

Requirements

1. Registration to Firebase
2. Enabling authentication in firebase console

Authentication should only be enabled for Email/Password, otherwise the web app can be accessed from any other enabled provider. This way it's using only emails and passwords provided in the Users section of the platform.

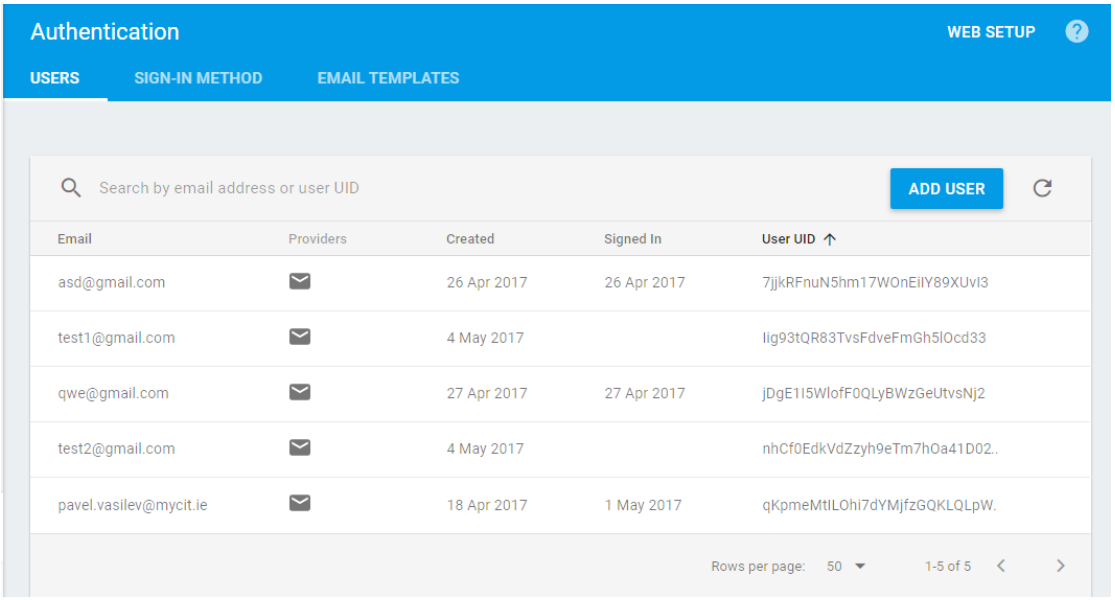
FIGURE 3.2: Firebase - Sign in Section



3. Add users

   While users can be added from the dashboard, at the deployment of different configuration file for the firebase, the first users can only be added manually. Off course any additional can be added as well.

FIGURE 3.3: Firebase - User section



Disadvantages

1. Paid service.
   Currently the platform is used as a developer and it doesn't require payment and has limited usage. However a paid option has to be taken in order to be deployed to the world.

2. Deep knowledge in the technology to be able to implement difficult features.
   The platform offers very unique features and the ability to be change in the most suitable way, however it requires immense amount of time dedicated especially for testing. Due to the information going to and from the database constantly.

3. Managing the authentication can prove rather difficult in terms of coding.

3.2.2 Technologies

The initial design of the web application started using Spring framework, however since the framework didn't support firebase, another technologies had to be chosen, 2 other frameworks under consideration were ReactJS,React Router and Polymer.

ReactJS and React Router is a very impressive framework, it allows implementing interesting features and the graphic configuration is quite amazing as well. Unfortunately it requires extensive knowledge on the subject to be able to properly manage by yourself.

Another reason for not choosing React over Polymer was that the framework doesn't directly support firebase. A workaround was needed to configure the platform to work with the framework. I decided that the amount of time I have for this task is not sufficient and went with Polymer.

Polymer is a framework that let's you use Web components. It creates reusable HTML and use them to build maintainable apps.

Google provides a library called Polymerfire - available at - "https://github.com/firebase/polymerfire", that acts as additional web component to the polymer framework and provides API to the firebase platform, which was the main reason for choosing this framework.

3.2.3 Implementation

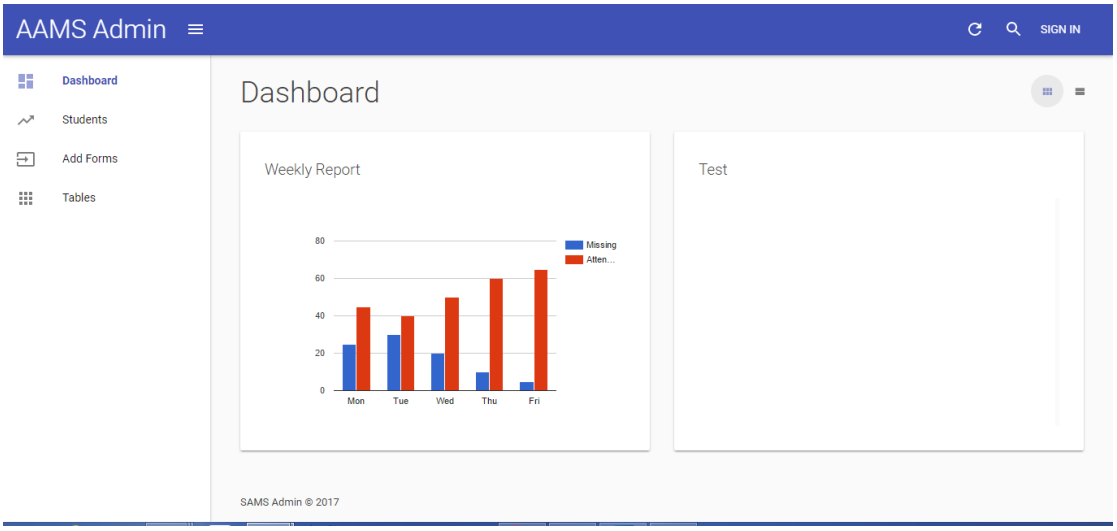
The implementation of the Web application proved rather difficult. Almost none previous experience was applicable in this area so starting to learn it from the beginning turned out to take effect on the limited time provided.

The application consist of a few simple reusable HTML pages and it was build using the gulp build. The task have been specified in the gulpfile.js in the web app. The purpose of gulp is to automate the repeatable task in the project.

1. Dashboard

   The dashboard is used to show graphical status of the state of missing,attending people. The current implementation for that is just a dummy data since there wasn't enough data collected to insert into the chart.

FIGURE 3.4: Dashboard



2. Add Forms

   The Add form elements purpose is to add users/students.

   (a) Students - people that wear beacons and being detected

   (b) User - people interacting with the Web application

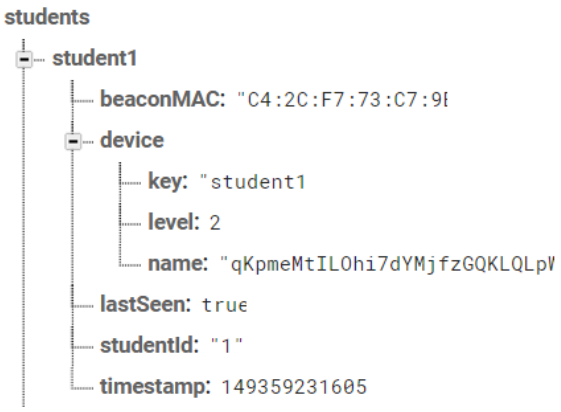
   Adding students - interacts with the firebase platform and adds a child to the it.

   An example of the type of child added to the system.

   (a) **Beacon MAC** - MAC address retrieved from the beacon through many available application online. I'm using BlueVision's app - BEEKS available on the Apple Store.

   (b) **Device** - is additional child to store information about the monitoring device.
   Key - it's the main child name used for simplification and is being used to get id of the current child
   Level - Level settings on the last monitoring device.
   Name - Uid name of the last monitoring device that has seen the beacon.

FIGURE 3.5: StudentDatabase



Monitoring device has to be logged in using an registered account in the firebase platform, this way enables us with a way to monitor the flow of data in the system.

(c) **LastSeen** - status of the beacon - true/false

(d) **Timestamp** - a timestamp of when the device was last seen.

Web application is mainly used for graphical interface of the database, so further explanation of these variables will be discussed in the Android Application section of the report. This is just a preview of each child and the purpose it's doing to the system.

Picture below shows the Add Forms page

FIGURE 3.6: Add User

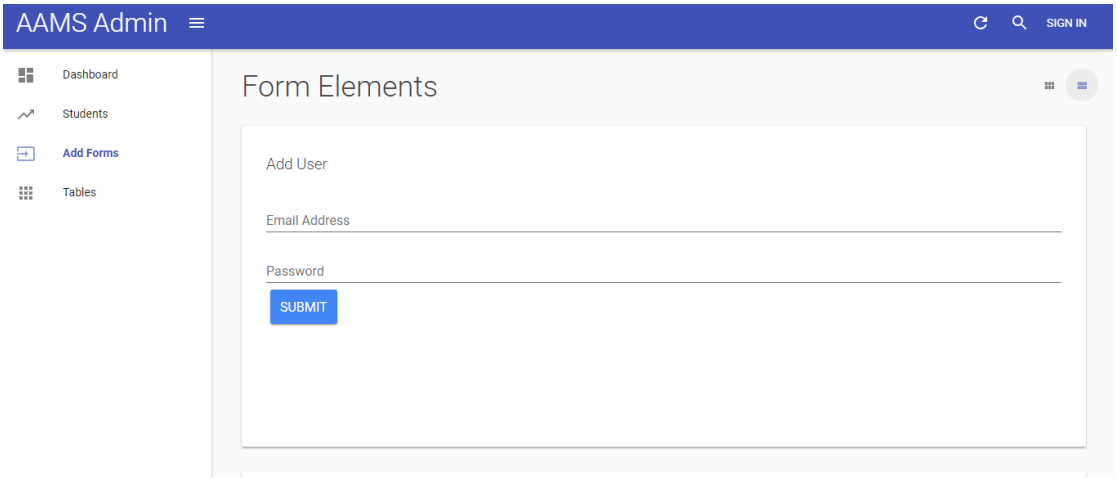
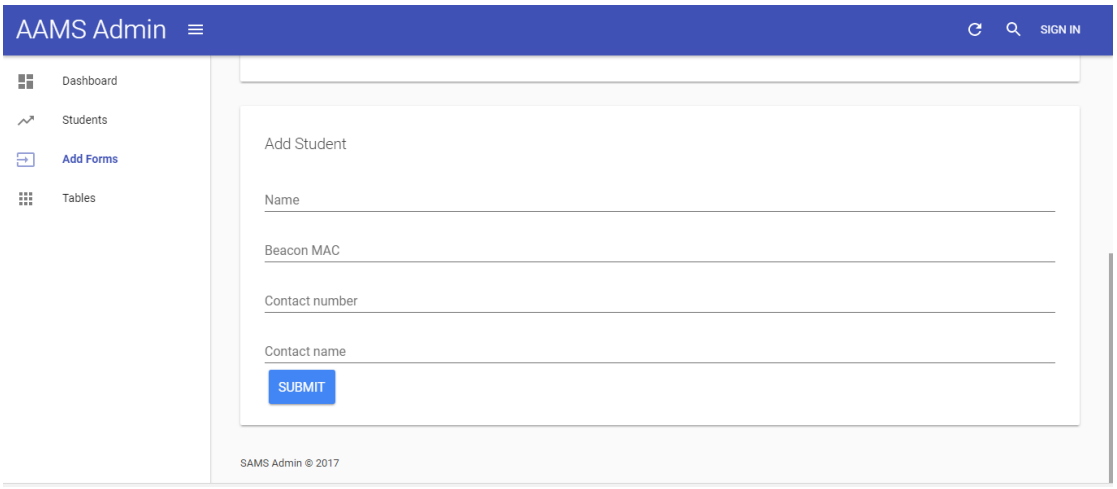


FIGURE 3.7: Add Student



Add User adds authorizes another user to use the dashboard. The data of the user is then saved in the database authentication section and to the real-time database.

Example of the users database is shown below :

FIGURE 3.8: User database

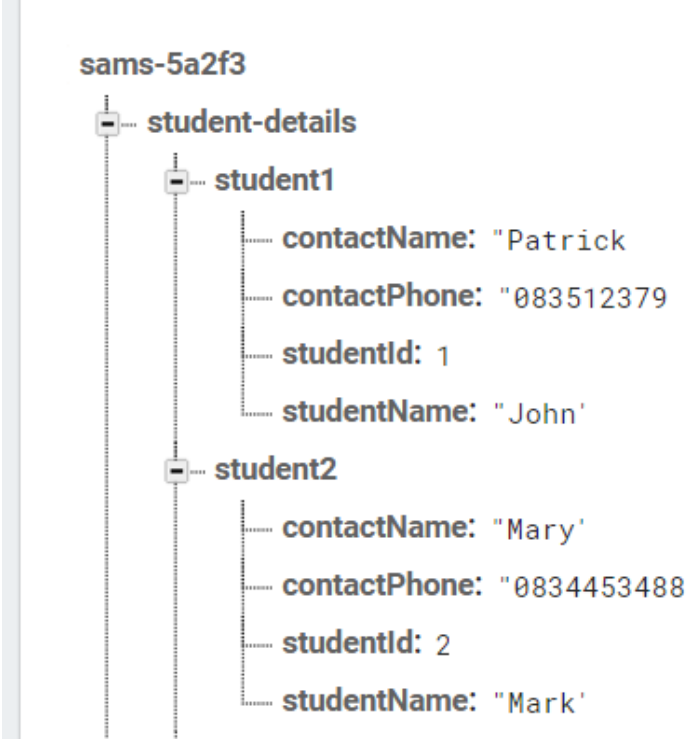
3. Tables

   This is the last section of the Web app and it contains all the child nodes in the students and students-details database to show a graphical image of the current status.

   It uses listeners connected to the database and signals us with data about the modified child, the data is then compared to the current child saved in the web application and updated.

   Below is an example of the student-details database :

FIGURE 3.9: Student Details



Features

I have implemented a few extra features on the web application, which are not necessarily need, but a good thing to have.

1. Minimize left side of the screen.
   The feature minimizes left side of the screen and expands the other components.
   Below are a few examples :

FIGURE 3.10: Minimized

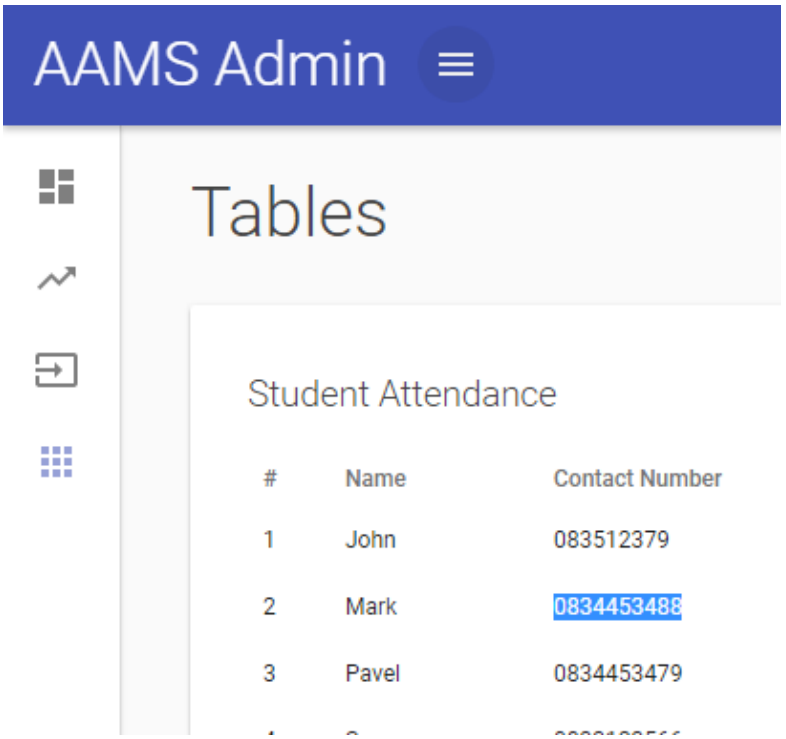
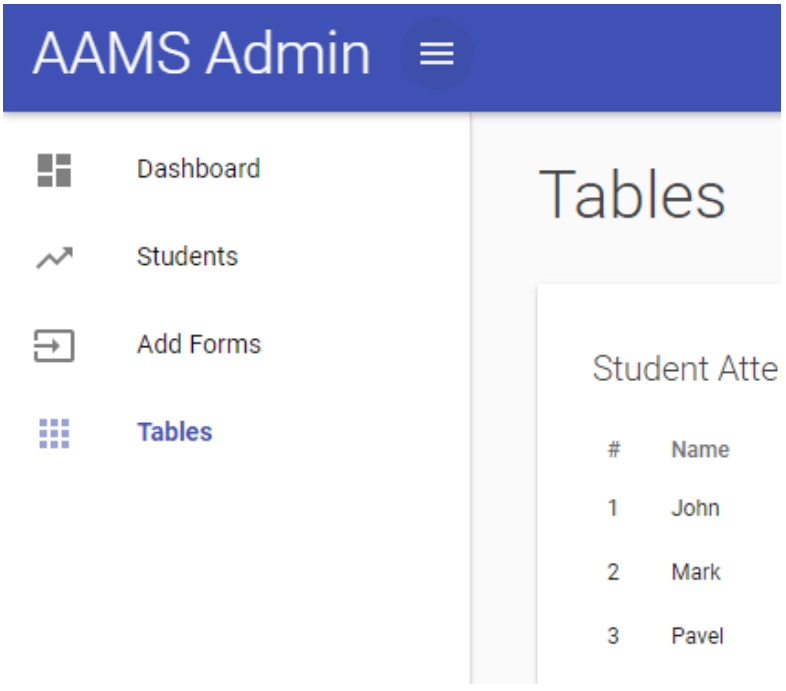


FIGURE 3.11: Not Minimized

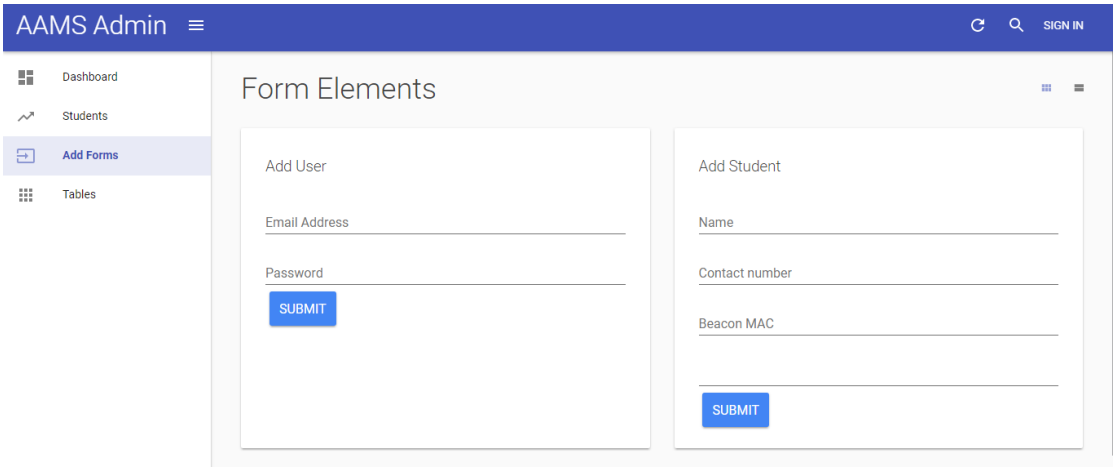


2. Search button

   At the header there is a search button that is connected to Google.com, if you type in your search criteria it will redirect you to a separate tab with the given results from google.com.

3. Stacking multiple components horizontally.
   As you have seen the previous figures 3.6 and 3.7. The Add user and Add student figures, with compressing the forms, we are able to show both of them in 1 screen
   Example below :

FIGURE 3.12: Stack components example

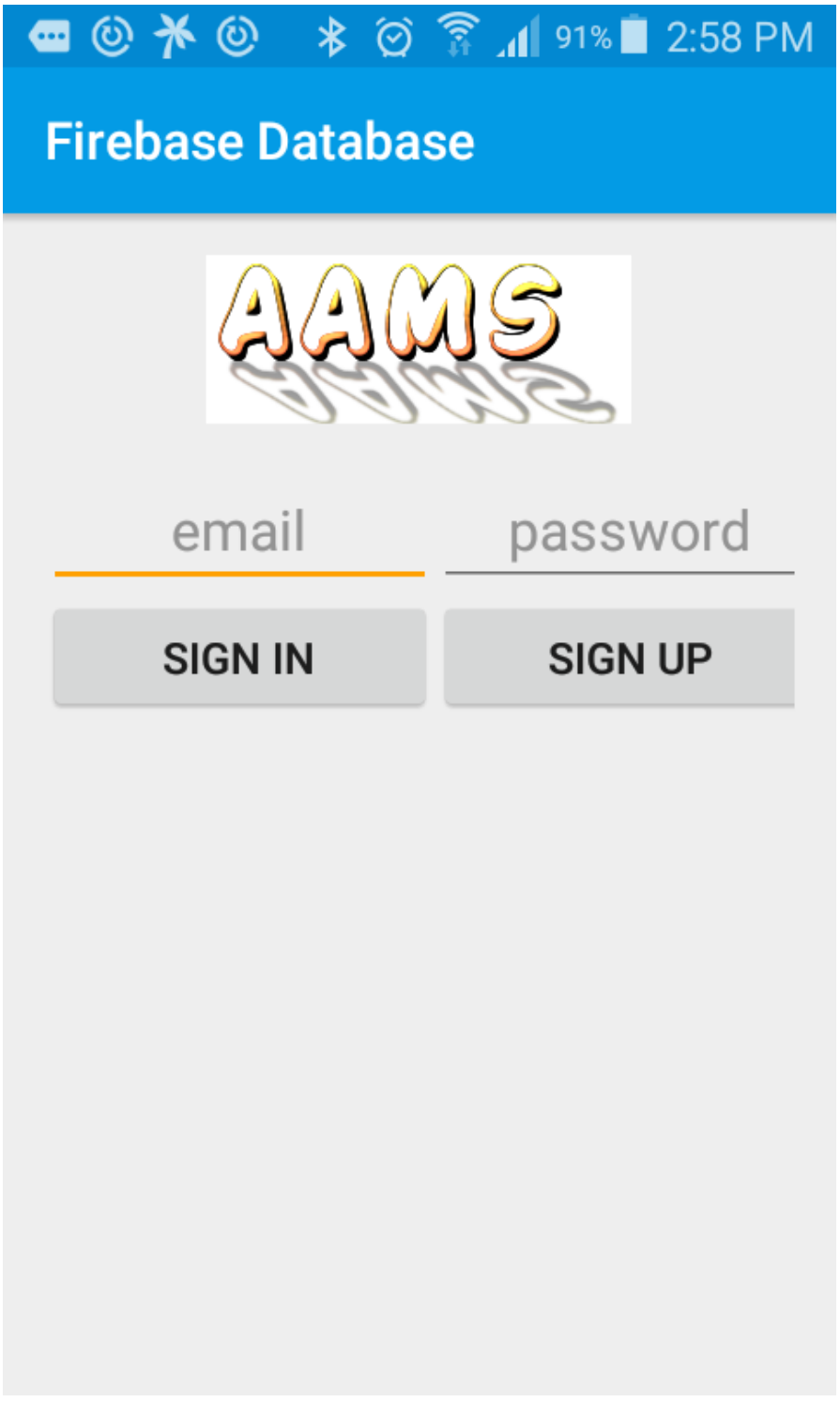


3.3 Android Application

In this section we will discuss the Android Application, which acts as a monitoring device to the system. It's purpose is to detect beacon's MAC address and update the real-time database.

The application requires to be run in the background, removing the need for a graphical interface. However a screen is added to enable sign in. Sign up button has been added as well to enable easier testing.

FIGURE 3.13: Android Sign In Interface



3.3.1 Configuration

This section we will discuss the requirements and their purpose in the android project.

Requirements

1. **Software**
   Minimum API support required is 18
   INTERNET$_{P}ERMISSION$
   $ACCESS_{C}OARSE_{L}OCATION_{P}ERMISSION$
   $ACCESS_{F}INE_{L}OCATION_{P}ERMISSION$
   Network
   $Theappwastestedonbothmobilenetworkandwirelessnetwork.Whilethenetworkremainedstablem$

2. **Physical**
   The Android application requires an Android phone with minimum requirements specified in the Software section. Testing was done with Samsung Galaxy 4.

3.3.2 Technologies

Technologies involved in the building of the Android project :

1. Java
2. Gradle - build application
3. FasterXML Jackson library - manage annotations
4. JUnit 4.12
5. Espresso testing
6. Runner testing
7. Firebase Auth 10.2.1
8. Firebase Database 10.2.1
9. BluevisionSDK
10. Google Play Services

3.3.3 Implementation

In this section we will discuss the different components within the android project itself. We will separate it in 3 different components :

1. Data Model
2. Activities

3. Service

Data Model

The purpose of the data model is to map data from/to database. Direct mapping can be achieved through the DataSnapshot.getValue, if give parameter User.class data will be mapped directly to the Object. "@IgnoreExtraProperties" is used to remove any non-required data from the database.

```
@IgnoreExtraProperties
public class User {

    public String username;
    public String email;

    public User() {
        // Default constructor required for calls to DataSnapshot.getValue(User.class)
    }

    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }

}
```

Listing 3.1: User.java

Second data model is the Student.java, however the application doesn't directly interact with it. The initial implementation was meant to consist all the variables of the "Student" in one database. Due to the research conducted, that proved to be a wrong approach and could cause a massive amount of data going to/from the database.

```
@IgnoreExtraProperties
public class Student {
    private String studentId;
    private String studentName;
    private String parentName;
    private String parentPhone;

    public Student() {
        // Default constructor required for calls to DataSnapshot.getValue(Student.class)
    }

    public Student(String studentId, String studentName, String parentName, String parentPhone)
        this.studentId = studentId;
        this.studentName = studentName;
        this.parentName = parentName;
        this.parentPhone = parentPhone;
```

```
    }

    public String getStudentId() {
        return studentId;
    }

    public void setStudentId(String studentId) {
        this.studentId = studentId;
    }

    public String getStudentName() {
        return studentName;
    }

    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }

    public String getParentName() {
        return parentName;
    }

    public void setParentName(String parentName) {
        this.parentName = parentName;
    }

    public String getParentPhone() {
        return parentPhone;
    }

    public void setParentPhone(String parentPhone) {
        this.parentPhone = parentPhone;
    }
}
```

Listing 3.2: Student.java

The next 2 data models are StudentBeacon.java and MonitoringDevice.java and the application interacts with them on a constant basis. Monitoring device is a required object to create a StudentBeacon object.

```
@IgnoreExtraProperties
public class StudentBeacon {
    private String studentId;
    private String beaconMAC;
    @JsonProperty
    private Object TIMESTAMP;
    boolean lastSeen;
    private MonitoringDevice deviceData;

    public StudentBeacon() {
        // Default constructor required for calls to DataSnapshot.getValue(StudentBeacon.class)
    }
```

```
public StudentBeacon(String studentId, String beaconMAC, boolean isAvailable, MonitoringDevi
    this.studentId = studentId;
    this.beaconMAC = beaconMAC;
    this.TIMESTAMP = ServerValue.TIMESTAMP;
    this.lastSeen = isAvailable;
    this.deviceData = deviceData;
}

public String getStudentId() {
    return studentId;
}

public void setStudentId(String studentId) {
    this.studentId = studentId;
}

public String getBeaconMAC() {
    return beaconMAC;
}

public void setBeaconMAC(String beaconMAC) {
    this.beaconMAC = beaconMAC;
}

public boolean isLastSeen() {
    return lastSeen;
}

public void setLastSeen(boolean lastSeen) {
    this.lastSeen = lastSeen;
}

public Object getTIMESTAMP() {
    return TIMESTAMP;
}

public void setTIMESTAMP(Object TIMESTAMP) {
    this.TIMESTAMP = TIMESTAMP;
}

public MonitoringDevice getDeviceData() {
    return deviceData;
}

public void setDeviceData(MonitoringDevice deviceData) {
    this.deviceData = deviceData;
}

@Exclude
public Map<String, Object> toMap() {
    HashMap<String, Object> result = new HashMap<>();
    result.put("beaconMAC", beaconMAC);
    result.put("studentId", studentId);
    result.put("lastSeen", lastSeen);
```

```
        result.put("timestamp", TIMESTAMP);
        result.put("device", deviceData.toMap());

        return result;
    }
    @JsonIgnore
    public Long getCreatedTimestamp() {
        if (TIMESTAMP instanceof Long) {
            return (Long) TIMESTAMP;
        }
        else {
            return null;
        }
    }

    @Override
    public String toString() {
        return "StudentBeacon{" +
                "studentId='" + studentId + '\'' +
                ", beaconMAC='" + beaconMAC + '\'' +
                ", TIMESTAMP=" + TIMESTAMP +
                ", isAvailable=" + lastSeen +
                ", deviceData=" + deviceData.getName() + deviceData.getLevel() +

                '}';
    }

}
```

Listing 3.3: StudentBeacon.java

StudentBeacon is POJO to correspond directly with the database using "DataSnapshot". Below is an example of the JSON received through DataSnapshot from firebase database.

```
{
  "beaconMAC" : "C4:2C:F7:73:C7:9B",
  "device" : {
    "key" : "student1",
    "level" : 2,
    "name" : "qKpmeMtILOhi7dYMjfzGQKLQLpW2"
  },
  "lastSeen" : true,
  "studentId" : "1",
  "timestamp" : 1493592316058
}
```

Listing 3.4: MonitoringDevice.java

MonitoringDevice.java is responsible for creating an object to be uploaded to the database.

1. **name** - returns auth uid of the monitoring device, the one used to login.

2. **level** - level of the device

3. **key** - child node

```
public class MonitoringDevice {

    private String name;
    private int level;
    private String key;

    public MonitoringDevice() {

    }

    public MonitoringDevice(String name, int deviceLevel, String key) {
        this.name = name;
        this.level = deviceLevel;
        this.key = key;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getLevel() {
        return level;
    }

    public void setLevel(int level) {
        this.level = level;
    }

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

    @Exclude
    public Map<String, Object> toMap() {
        HashMap<String, Object> result = new HashMap<>();
        result.put("name", name);
        result.put("level", level);
        result.put("key", key);

        return result;
    }
}
```

Listing 3.5: MonitoringDevice.java

Activities

The android application consist of 2 activities - SignInActivity and MainActivity. The app launches with the SignInActivity. Initial idea didn't included authorization on the monitoring device, an extensive research on the subject was decided that authentication would be required to use as identification on the device that detected the beacon. Authentication implemented is firebase-auth. The activity also allows signing up to the database, which was added for testing purposes.

```
public class SignInActivity extends BaseActivity implements View.OnClickListener {

    private static final String TAG = "SignInActivity";

    private DatabaseReference mDatabase;
    private FirebaseAuth mAuth;

    private EditText mEmailField;
    private EditText mPasswordField;
    private Button mSignInButton;
    private Button mSignUpButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sign_in);

        mDatabase = FirebaseDatabase.getInstance().getReference();
        mAuth = FirebaseAuth.getInstance();

        // Views
        mEmailField = (EditText) findViewById(R.id.field_email);
        mPasswordField = (EditText) findViewById(R.id.field_password);
        mSignInButton = (Button) findViewById(R.id.button_sign_in);
        mSignUpButton = (Button) findViewById(R.id.button_sign_up);

        // Click listeners
        mSignInButton.setOnClickListener(this);
        mSignUpButton.setOnClickListener(this);
    }

    @Override
    public void onStart() {
        super.onStart();

        // Check auth on Activity start
        if (mAuth.getCurrentUser() != null) {
            onAuthSuccess(mAuth.getCurrentUser());
```

```
        }
    }

    private void signIn() {
        Log.d(TAG, "signIn");
        if (!validateForm()) {
            return;
        }

        showProgressDialog();
        String email = mEmailField.getText().toString();
        String password = mPasswordField.getText().toString();

        mAuth.signInWithEmailAndPassword(email, password)
                .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        Log.d(TAG, "signIn:onComplete:" + task.isSuccessful());
                        hideProgressDialog();

                        if (task.isSuccessful()) {
                            onAuthSuccess(task.getResult().getUser());
                        } else {
                            Toast.makeText(SignInActivity.this, "Sign In Failed",
                                    Toast.LENGTH_SHORT).show();
                        }
                    }
                });
    }

    private void signUp() {
        Log.d(TAG, "signUp");
        if (!validateForm()) {
            return;
        }

        showProgressDialog();
        String email = mEmailField.getText().toString();
        String password = mPasswordField.getText().toString();

        mAuth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        Log.d(TAG, "createUser:onComplete:" + task.isSuccessful());
                        hideProgressDialog();

                        if (task.isSuccessful()) {
                            onAuthSuccess(task.getResult().getUser());
                        } else {
                            Toast.makeText(SignInActivity.this, "Sign Up Failed",
                                    Toast.LENGTH_SHORT).show();
                        }
                    }
                });
```

```
}

private void onAuthSuccess(FirebaseUser user) {
    String username = usernameFromEmail(user.getEmail());

    // Write new user
    writeNewUser(user.getUid(), username, user.getEmail());

    // Go to MainActivity
    startActivity(new Intent(SignInActivity.this, MainActivity.class));
    finish();
}

private String usernameFromEmail(String email) {
    if (email.contains("@")) {
        return email.split("@")[0];
    } else {
        return email;
    }
}

private boolean validateForm() {
    boolean result = true;
    if (TextUtils.isEmpty(mEmailField.getText().toString())) {
        mEmailField.setError("Required");
        result = false;
    } else {
        mEmailField.setError(null);
    }

    if (TextUtils.isEmpty(mPasswordField.getText().toString())) {
        mPasswordField.setError("Required");
        result = false;
    } else {
        mPasswordField.setError(null);
    }

    return result;
}

// [START basic_write]
private void writeNewUser(String userId, String name, String email) {
    User user = new User(name, email);

    mDatabase.child("users").child(userId).setValue(user);
}
// [END basic_write]

@Override
public void onClick(View v) {
    int i = v.getId();
    if (i == R.id.button_sign_in) {
        signIn();
    } else if (i == R.id.button_sign_up) {
        signUp();
```

```
        }
    }
}
```

Listing 3.6: SignInActivity.java

The second activity is the MainActivity, it's functionality is to check if the build version is 23 or above. FYI current project is build on sdk version 25. In case the phone doesn't allow location access, it shows an alert dialog for the user to click and enable it. The other functionality is to start the service using the Application context, because it has longer lifecycle than Activity.

Below is the code snippet for the MainActivity :

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        startService(new Intent(getApplication().getApplicationContext(), MyService.class));
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
            // Android M Permission check
            if (checkSelfPermission(android.Manifest.permission.ACCESS_COARSE_LOCATION)
                    != PackageManager.PERMISSION_GRANTED) {

                final AlertDialog.Builder builder    = new AlertDialog.Builder(MainActivity.this
                builder.setTitle("This app needs location access");
                builder.setMessage("Please grant location access so this app can detect beacons"
                builder.setPositiveButton(android.R.string.ok, null);
                builder.setOnDismissListener(new DialogInterface.OnDismissListener() {
                    @Override
                    public void onDismiss(DialogInterface dialog) {

                        requestPermissions(
                                new String[]{android.Manifest.permission.ACCESS_COARSE_LOCATION}
                                PERMISSION_REQUEST_COARSE_LOCATION);
                    }

                });
                builder.show();
            }
        }
```

Listing 3.7: MainActivity.java

Service

MyService.java is the core of the project, it is initialized through the application context, meaning that the app would never crash due to UI messing up.

The OnCreate() method requires application context in order to initialize the BeaconManager and the service has added itself as a listener to it. The beacons used for the project are BlueVision and their format is added to the scan type.
Scanning of beacons doesn't begin immediately. Through testing it was determined that firebase database needs 1-3 seconds based on wifi broadband to initialize itself. A handler was created to wait 10 seconds before starting to scan for beacons.

On initialization on the service a ChildListener is added and we have to implement each method. OnChildAdded() method is the one that initializes all the current data of the firebase project. Each call to the onChildAdded consist of only one child of the corresponding database.

My Service implements BeaconListener interface allowing us to implement the following methods - OnBeaconFound() and BluetoothIsNotEnabled(). OnBeaconFound returns a beacon and the only information required from it is the MAC address. Each beacon has it's own unique address and can be used as a identification. After we get the beacon MAC address it is compared to the current list of beacons locally stored. If the MAC address matches any of the beacons. The information about him is either set to true/-false depending on other variables as level of the device, status of the beacon. The new information is then updated to the database, however the local list of beacons is not updated locally, but after the database has called onChildChanged() with the updated data. Thus enabling us to keep clean state of the data.

The monitoring device is level based system, meaning that higher level's override lower level. For example if device with level - 2 detects a beacon who was last seen by a level 1 device. It automatically assigns that the beacon is in the facility. If this same beacon is again detected by device with level 1, the status of the beacon is marked as false, reporting that the beacon has left the facilities.

There are a few different uses cases in the level based device:

Device A = Level 1, Device B = Level 2 - deeper in the facility

If Device A detects beacon, which variable lastSeen is false, it marks the beacon as true.
If Device B detects that same beacon or a beacon that might have been missed by Device A, it is marked as true.
If Device A detects beacon marked as true with level 2. This signals that the beacon is leaving the facility.

If Device A detects beacon marked as true with level 1. This signals that the beacon is leaving the facility.

```
public class MyService extends Service implements BeaconListener {

    public static final String TAG = "MyService";
    public static final String STUDENT = "students";
    public static final String MONITORING_DEVICE = "device";
    public static final String USER = "users";
    public static final boolean TRUE = true;
    public static final boolean FALSE = false;
    private DatabaseReference databaseReference;
    private DatabaseReference userReference;
    private DatabaseReference updateReference;
    private User user;
    private List<StudentBeacon> currentStudentBeacons;
    private List<StudentBeacon> updateStudentBeacon;
    private String DEVICE_ID;
    private App app;
    private boolean isAuth;

    private BeaconManager beaconManager;
    private OnBeaconChangeListener listener;


    //App context can be received only from onCreate method
    @Override
    public void onCreate() {
        super.onCreate();
        app = (App) getApplication();
        beaconManager = new BeaconManager(app.getApplicationContext());
        beaconManager.addBeaconListener(this);
        beaconManager.addRuleToIncludeScanByType(BeaconType.S_BEACON);
    }

    public MyService() {
        currentStudentBeacons = new ArrayList<>();
        updateStudentBeacon = new ArrayList<>();
        updateReference = FirebaseDatabase.getInstance().getReference();
        userReference = FirebaseDatabase.getInstance().getReference().child(USER).getRef();
        databaseReference = FirebaseDatabase.getInstance().getReference().child(STUDENT).getRef(
        isAuth = false;
        if (isAuthenticated()) {
            startMonitoring();
        }
        final Handler handler = new Handler();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                beaconManager.startScan();
            }
        }, 10000);
    }
```

```
@Override
public IBinder onBind(Intent intent) {
    return null;
}

/**
 * Initialize monitoring and populate data through firebase
 */
public void startMonitoring() {
    Log.i(TAG, "startMonitoring() has started");
    if (currentStudentBeacons == null) {
        userReference.child(DEVICE_ID).addListenerForSingleValueEvent(new ValueEventListener
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                Log.d(TAG, "IS USER DATA CALLED");
                user = dataSnapshot.getValue(User.class);
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                Log.w(TAG, "getUser:onCancelled", databaseError.toException());
            }
        });
    }
    ChildEventListener childEventListener = new ChildEventListener() {
        @Override
        public void onChildAdded(DataSnapshot dataSnapshot, String s) {
            StudentBeacon beacon = dataSnapshot.getValue(StudentBeacon.class);
            MonitoringDevice device = dataSnapshot.child(MONITORING_DEVICE).getValue(Monitor
            beacon.setDeviceData(device);
            addStudent(beacon);
            Log.d(TAG, "onChildAdded with data" + dataSnapshot.toString());
        }

        @Override
        public void onChildChanged(DataSnapshot dataSnapshot, String s) {
            StudentBeacon beacon = dataSnapshot.getValue(StudentBeacon.class);
            MonitoringDevice device = dataSnapshot.child(MONITORING_DEVICE).getValue(Monitor
            beacon.setDeviceData(device);
            updateStudent(beacon);

            Log.d(TAG, "CHILD HAS CHANGED" + dataSnapshot.toString());
        }

        @Override
        public void onChildRemoved(DataSnapshot dataSnapshot) {
            StudentBeacon beacon = dataSnapshot.getValue(StudentBeacon.class);
            MonitoringDevice device = dataSnapshot.child(MONITORING_DEVICE).getValue(Monitor
            beacon.setDeviceData(device);
            removeStudent(beacon);
        }

        @Override
        public void onChildMoved(DataSnapshot dataSnapshot, String s) {
            //should never be called if it does log it as error
```

```
                Log.e(TAG, "onChildMoved error : KEY --- " + s);
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
                //should never be called if it does log it as error
                Log.e(TAG, databaseError.toString());
            }
        };
        databaseReference.addChildEventListener(childEventListener);
//        listener = new OnBeaconChangeListener() {
//            @Override
//            public void onRssiChanged(Beacon beacon, int i) {
//                Log.d(TAG, "OnRssi changed : " + i);
//            }
//
//            @Override
//            public void onRangeChanged(Beacon beacon, Range range) {
//                Log.d(TAG, "On Range Changed :" + range.toString());
//            }
//
//            @Override
//            public void onBeaconExit(Beacon beacon) {
//                Log.d(TAG, "onBeaconExit");
//            }
//
//            @Override
//            public void onBeaconEnter(Beacon beacon) {
//                Log.d(TAG, "On Becon Enter");
//            }
//        };
    }

    /**
     * Check to see if user has authenticated
     * @return
     */
    public boolean isAuthenticated() {
        if (getUid() != null) {
            DEVICE_ID = getUid();
            isAuth = true;
        }
        return isAuth;
    }

    /**
     * Returns Uid of authenticated user
     * @return
     */
    public String getUid() {
        return FirebaseAuth.getInstance().getCurrentUser().getUid();
    }

    /**
     * Return device LEVEL SETTINGS
```

```
     * @return
     */
    public int getLevel() {return 2;}
    /**
     * Writes to firebase cloud database with
     * @param beacon
     * @param key
     */
    public void writeToDatabase(StudentBeacon beacon, String key) {
        StudentBeacon updateBeacon = beacon;
        beacon.getDeviceData().setName(getUid());
        Map<String, Object> studentValues = null;
        studentValues = updateBeacon.toMap();
        Map<String, Object> childUpdates = new HashMap<>();
        childUpdates.put("/students/" + key, studentValues);
        updateReference.updateChildren(childUpdates);
    }

    /**
     * When child changes replaces the beacon and that place with the current one from database
     * @param beacon
     */
    public void updateStudent(StudentBeacon beacon) {
        int index;
        if (currentStudentBeacons != null) {
            for (StudentBeacon b1 : currentStudentBeacons) {
                if (b1.getBeaconMAC() == beacon.getBeaconMAC());
                index = currentStudentBeacons.indexOf(b1);
                if (index != -1) {
                    currentStudentBeacons.set(index, b1);
                }
            }
        }
    }

    /**
     * Called through onChildRemoved
     * @param student
     */
    public void removeStudent(StudentBeacon student) {
        currentStudentBeacons.remove(student);
    }

    /**
     * Called through onChildAdded - creates a local arraylist with all the students
     * @param student
     */
    public void addStudent(StudentBeacon student) {
        if (currentStudentBeacons != null) {
            Log.d(TAG, "ADD STUDENT CALLED : " + currentStudentBeacons.size());
        }
        currentStudentBeacons.add(student);
    }

    /**
```

```
     * Update firebase database(complicated algorithm)
     * @param beacon
     */
    public void updateBeaconDatabase(StudentBeacon beacon) {
        beacon.getDeviceData().setName(getUid());

        if(beacon.getDeviceData().getLevel() == 1) {
            if (getLevel() == 2) {
                beacon.setLastSeen(TRUE);
                beacon.getDeviceData().setLevel(getLevel());
            writeToDatabase(beacon, beacon.getDeviceData().getKey());
        }
        if (beacon.isLastSeen() && getLevel() == 1) {
            beacon.setLastSeen(FALSE);
            beacon.getDeviceData().setLevel(getLevel());
            writeToDatabase(beacon, beacon.getDeviceData().getKey());
        }

        }
        if (beacon.getDeviceData().getLevel() == 2) {
            if (getLevel() == 1) {
                beacon.setLastSeen(FALSE);
                beacon.getDeviceData().setLevel(getLevel());
                writeToDatabase(beacon, beacon.getDeviceData().getKey());
            }
        }
        if(beacon.getDeviceData().getName() == null) {
            beacon.setLastSeen(TRUE);
            beacon.getDeviceData().setLevel(getLevel());
            writeToDatabase(beacon, beacon.getDeviceData().getKey());
        }
    }
    /**
     * Compares MAC address when detected against the currentStudentBeacon list
     *
     * @param MAC
     * @param currentStudentBeacons
     */
    public void compareMAC(String MAC, List<StudentBeacon> currentStudentBeacons) {
        if (currentStudentBeacons != null) {
            for (StudentBeacon beacon : currentStudentBeacons) {
                if (beacon.getBeaconMAC().equals(MAC)) {
                    Log.d(TAG, "IS TRUE");
                    updateBeaconDatabase(beacon);
                }
            }
        }
    }

    /**
     * Start of BeaconListener methods
     * This listener is used to listen when a beacon is discovered
     */
    @Override
    public void onBeaconFound(Beacon beacon) {
```

```
        if (beacon != null) {
            if (beacon.getDevice().getAddress() != null) {
                String MAC = beacon.getDevice().getAddress();
                compareMAC(MAC, currentStudentBeacons);
            }
        }
    }

    @Override
    public void bluetoothIsNotEnabled() {
        Log.e(TAG, "ENABLE BLUETOOTH");
    }

}
```

Listing 3.8: MyService.java

3.4 Beacon

In this section I would discuss the software required to set up your own beacons and acquire their MAC address.

3.4.1 Configuration

For this project the beacons that were under development are from the BlueVision manufacturer available at - https://shop.bluvision.com/collections/all.

BlueVision has it's own configuration app called BEEKS available at the Apple Store.

Beacons Configuration

1. Broadcast rate - 1Hz - once a second
2. Transmit power - 0dBm(¡50 meters)
3. Energy Saving - 0dBm(¡50meters)
4. Energy Saving - 1Hz - once a second
5. Type - sBeacon

How to get MAC Address ?

MAC address can be received after you implement the bluevision sdk and ask for that address as shown in the code snippet above, each beacon has it's own unique MAC address.

Chapter 4

Difficulties

Web application This project was very challenging and interesting and at the same time very nerve wrecking, the range of issues faced from a single semicolon to changing frameworks and platforms.

In terms of the Web application too much consideration was given for the UI, which proved rather difficult task and time consuming.

Spring framework is java based framework that is used to create web applications. It's a provides very easy to work environment and ability to top up extra features, without damaging the others. Initial design started with this framework, but after testing I was unable to connect it to the database, so it was changed for reactJS.

ReactJS is a component based JavaScript library that uses javascripts pages and renders them to the single html page in the app. ReactJS proved very complicated, especially with the addition of firebase, since the framework is from different company their comparability is not perfect with each other, which led me to my next choice - Polymer.

Polymer uses web components as a reusable HTML pages. In terms of features and way of work, it's very similar to ReactJS, never the less it has it's own component build for the framework build by the firebase company(Google) and requires less work for the same functionality - Polymerfire

Android application

This is the first I've been exposed to beacon technology, it turned out very complicated and competitive in terms of which technology is more appropriate.

There are multiple manufacturers and multiple platforms with their own special functionality and libraries.

Initial library was AltBeacon, it's very popular library in the beacon environment and applicable by both iOS and Android. However after 2 weeks of testing and trying different implementations the application kept on missing a few beacons, or they were never discovered in the first place.

The results forced me to keep looking for more suitable library. I looked for the manufacturers website and there I found the BlueVisionSDK, it's not very wildly used library, however it proved with high compatibility with their own beacons. The most important requirement was to almost/never miss a beacon.

Difference to actual approach

Actual approach was supposed to be a student attendance management system, due to the broad nature of the project it was changed to automatic attendance management system. It provides more generic view of the system. The implementation still remains the same. However instead of adding a few extra database components in terms of classes, teachers and connecting them I decided to focus on actual automation of the system and render it as usable as possible, which in fact was the main purpose of this project.

Another feature that will be missing due to resources is the messaging use case, which was that at certain time, people marked who have added their contactNumber in the student-details database would receive notification when the carrier of the beacon is not in the facility.

Chapter 5

Testing and Future Enhancements

5.1 Testing

Testing of the complete project is broken in 3 categories

1. Test Web application against database.
2. Test Android application against database.
3. Test the whole system together.

Test Web Application against database

Web application was the first component build, the testing was focused on retrieving and uploading data to the firebase database. Main method of testing was regression testing. Similar task were tasted over and over after each code change.

Adding logs in any insecure places in the code resulted in very helpful information and reducing a few problems.

Test Android application

The android app was first build to detect beacons without any other task. Breakpoints in the code showed the flow of information, which reduced quite a bit of problems as well.

Due to the nature of the project regression testing was again the most successful testing approach.

During the complete implementation all the use cases of the app were tested, to test for a proper behaviour.

Test the whole system together

Even if both of the components worked perfectly separately during the integration of all components together, a few errors arise in terms of in proper management of the state of the information. Some of the beacons were changed in the firebase database while they remained unchanged locally and during the next encounter with a beacon they didn't behave properly.

Regression testing was the main method of testing as well. A set of use cases was used to analyze the state of the system.

5.2 Future Enhancements

Two categories of future enhancements are :

1. Implementation enhancements
2. Resource enhancements

Implementation enhancements

Localization monitoring can be included in the system to provide extra features as permanent location monitoring. Information would give us details of exact position of the user in the facility.

Resource enhancements

Beacon's that are being used for this project are a bit outdated, new beacon models get released regularly. Estimote beacons are gaining quite a bit popularity and the sticker beacons can become very suitable for this particular project.

Chapter 6

Overall Conclusions

The purpose of this project was to devise and implement an automatic attendance management system. When the research documentation was completed, no firm detail on implementation was decided upon. In hindsight this was the biggest miscalculation.

This resulted in additional research conducted to try and find a suitable solution. This limited the available human resources that ended up not being utilised to it's fullest.

6.1 Project Review

The project was unique in it's own and very challenging. It is great achievement to complete this kind of project by myself. Time management is one of the things that I would do differently. The overall project is a success, however a lot of personal time was spend working as well. In terms of task management in the project I started working on the components I considered "easy", which limited my time on the other parts of the project. The implementation of the web application was initially thought to be a simple process. This was not the case.

If I get to chance to build the project again there are a few things I would do differently :

1. Involve more people in the project
   If more people were involved would've been much easier to deal with issues when I get stuck on a problem or have an important decision to make
2. Unknown technologies
   All of the components of the project are unknown and previous experience in coding was barely applicable.

3. Begin basic implementation alongside research
   It would've reduced most of the problems and allowed for more time to be spend on the project

6.2 Key Skills

Extensive knowledge was gained in the different platforms/technologies in the project. This would be a great asset when starting a new project. The research made on the allows me to be confident in my further decisions on a similar topic.

6.3 Conclusion

Overall the project is considered a success and the automated attendance management(AAMS) is working the way it's supposed to be. It was a pleasant experience, even the unforeseen helped gain additional knowledge on the subject.

Bibliography

[1] Firebase, 2017. [Online]. Available: https://firebase.google.com/

[2] Github, 2017. [Online]. Available: http://github.com

[3] Polymer-Fire, 2017. [Online]. Available: https://github.com/firebase/polymerfire

[4] Polymer, 2017. [Online]. Available: https://www.polymer-project.org/

[5] BlueVision, 2017. [Online]. Available: http://bluvision.com/

[6] NodeJS, 2017. [Online]. Available: https://nodejs.org/en/

[7] Gulp, 2017. [Online]. Available: http://gulpjs.com/

[8] StackOverFlow, 2017. [Online]. Available: http://stackoverflow.com