A Project
Presented to the
Faculty of
California State University,
San Bernardino

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Francisco Ron
December 2015
ABSTRACT

The objective of this masters project is to develop a working application for Android devices. This is an application intended to be used by CSUSB. It has its own database, which has information about most of the facilities on campus. There are many GPS applications on the market, however I chose to design and implement WiN GPS, short for Walking GPS, because it will allow the possibility of a personalized GPS for the school and for users should they choose to do so.

In order to develop WiN GPS it was necessary to research the available tools, and to become familiar with the ones that were selected. These tools such as map application providers, i.e. Google-maps, integrated development environments, database managers, software development kits, and mobile device emulators were analyzed and compared.

Once the tools were selected, it was necessary to study, to become familiar with, and to learn how to use them. Finally an app is developed and its main functions/code will be explained. This masters project will allow potentially Android developers to evaluate possible barriers, such as price and limitations of map application providers, so they can make an informed decision.

Keywords: GoogleMaps, Android, app, SQLite3, Eclipse, Java, Database, GPS, mobile development
ACKNOWLEDGEMENTS

Upon reflecting on the journey traveled to put this masters project together, I have come to realize that without the assistance of the following persons there would not be such a project.

My committee members Dr. Turner, Dr. Voigt, and Dr. Gomez. I want to thank especially Dr. David Turner, my project advisor: he has been very helpful, not only through the development of this project, but also on every class I took with him. His knowledge in up-to-date technologies has been helpful. I also want to thank Dr. Schubert. It was an honor to be one of his students. Finally, it’s my pleasure to mention Mrs. Monica Latimer always committed to help students.
TABLE OF CONTENTS

Abstract ................................................................. iii

Acknowledgements .................................................. iv

List of Tables .......................................................... vii

List of Figures .......................................................... viii

1. Project Summary .................................................... 1
   1.1 Purpose ......................................................... 1
   1.2 Scope ........................................................... 2
   1.3 Development Tools ............................................ 2
   1.4 Definitions, Acronyms, and Abbreviations ................. 2

2. OVERVIEW OF DEVELOPMENT TOOLS .......................... 4
   2.1 Maps ............................................................ 4
   2.2 IDE ............................................................ 7
   2.3 Java ........................................................... 11
   2.4 SDK and Google Packages .................................. 12
   2.5 ADB ........................................................... 13
   2.6 Emulators ...................................................... 13
   2.7 Database ...................................................... 13
3. WiN-GPS MECHANICS ............................................ 15
   3.1 Summary ..................................................... 15
   3.2 screens ..................................................... 15
   3.3 Walking Navigation ................................. 16

4. SOFTWARE ARCHITECTURE AND DESIGN .................. 32
   4.1 Main Activity Class ................................. 32
   4.2 DBmanager Class ................................. 36
   4.3 DisplayTrackingMap Class ................................. 42
   4.4 4.4 New Location Class ................................. 49
   4.5 Edit Location Class ................................. 52
   4.6 Get Coordinates Class ................................. 54
   4.7 Show Location Info Class ................................. 57
   4.8 Changing Font for the Entire Application ................. 60

5. Conclusion and Future Direction ......................... 62
   5.1 Conclusion ..................................................... 62
   5.2 Future Direction ............................................. 62

References ............................................................. 64
LIST OF FIGURES

2.1  Project Defined in Google Console ........................................ 5
2.2  Credentials for the Project .................................................... 6
2.3  API Key in the Manifest ......................................................... 7
2.4  Eclipse Features I ................................................................. 8
2.5  Eclipse Features II ................................................................. 9
2.6  Values Definition ................................................................. 10
2.7  Android Application Life Cycle[8] .......................................... 12

3.1  Screen Flow Diagram ......................................................... 16
3.2  Initial Screen ................................................................. 17
3.3  Walking GPS Global Screen ................................................ 18
3.4  Walking GPS Initial Screen ................................................ 19
3.5  Walking GPS I ................................................................. 20
3.6  Walking GPS II ................................................................. 21
3.7  Info Screen ................................................................. 22
3.8  Walking GPS III ................................................................. 23
3.9  Initial screen ................................................................. 24
3.10  Add a New Location Screen ............................................... 25
3.11  Add a New Location Partially Filled Screen ................................ 26
3.12  Get Coordinates screen .................................................... 27
3.13  Select a location from screen ............................................. 28
3.14  Save a new location screen ............................................... 29
3.15 Initial screen ........................................ 30
3.16 Edit Location screen ............................... 31

4.1 UML package application .......................... 33
4.2 UML Diagram of DBmanager and MainActivity relationship .......... 34
4.3 Main Activity Code (database opening) ................. 34
4.4 populating the list ................................... 35
4.5 When a Location is Clicked ......................... 35
4.6 Calling the methods to ”Add” a new location .......... 36
4.7 On Long Click: Edit a Location ..................... 36
4.8 Table android meta-data: structure .................. 37
4.9 Table locations: structure .......................... 37
4.10 UML DBmanager class .............................. 38
4.11 UML DBmanager instantiations ..................... 39
4.12 Database definition ................................ 39
4.13 Edit Location, Code ............................... 40
4.14 Deleting a Location ............................... 41
4.15 Get Info for a Location ............................ 41
4.16 Creating a New Location ......................... 42
4.17 UML instantiation of fragment activity ............. 43
4.18 Fragment definition in map layout .................. 44
4.19 Map fragment instantiation in DisplayTrackingMap .......... 45
4.20 Initializing the map ................................ 46
4.21 Importing ConnectionCallbacks .................... 46
4.22 Request: Parameters for Callback ................... 47
4.23 OnConnected definition ........................... 47
4.24 Updating user’s movement ........................ 48
4.25 New Location graphic layout in Eclipse .......... 49
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.26 New Location layout code</td>
<td>50</td>
</tr>
<tr>
<td>4.27 New Location declaring layout</td>
<td>50</td>
</tr>
<tr>
<td>4.28 Saving a New Location on Locations Table</td>
<td>51</td>
</tr>
<tr>
<td>4.29 Looking for Coordinates</td>
<td>51</td>
</tr>
<tr>
<td>4.30 Look up for location information</td>
<td>52</td>
</tr>
<tr>
<td>4.31 Update a location information</td>
<td>53</td>
</tr>
<tr>
<td>4.32 Deleting a location</td>
<td>54</td>
</tr>
<tr>
<td>4.33 Defining Callback parameters</td>
<td>55</td>
</tr>
<tr>
<td>4.34 Connecting map layout to activity (class)</td>
<td>55</td>
</tr>
<tr>
<td>4.35 On Long Click select lat. and log</td>
<td>56</td>
</tr>
<tr>
<td>4.36 onConnected method requestLocationUpdate</td>
<td>56</td>
</tr>
<tr>
<td>4.37 return latitude and longitude</td>
<td>57</td>
</tr>
<tr>
<td>4.38 Location Info Layout</td>
<td>58</td>
</tr>
<tr>
<td>4.39 Displaying location information</td>
<td>59</td>
</tr>
<tr>
<td>4.40 Returning to caller</td>
<td>60</td>
</tr>
<tr>
<td>4.41 Font imported into Project assets</td>
<td>60</td>
</tr>
<tr>
<td>4.42 Font Defined to be Used in a Button</td>
<td>61</td>
</tr>
</tbody>
</table>
1. PROJECT SUMMARY

Mobile applications are quite popular nowadays. It’s possible to find apps pretty much for anything, and yet, there is still so much to be done. One particular area of mobile apps deals with maps and their management. Multiple applications have been designed using maps, such as GPS, transit information, bike routes, and so on[3]. This masters project’s purpose is to create a walking navigation GPS app for the CSUSB campus.

It must be noted also that mobile devices come in different shapes, brands, and operating systems. This particular project will focus on mobile devices using the Android Operating System.

1.1 Purpose

This is a masters degree project, therefore the main purpose is to deliver an application according to the masters program level of knowledge. The application might be used by CSUSB visitors unfamiliar with the facilities, or students that are familiar with them, but they want to point out their 'preferred' places on a map. Another purpose, not less important, is to provide a document that might be useful to students interested in Android applications.
1.2 Scope

The scope of this project is limited to maps on Android devices. Several tools are available toward this kind of development, therefore I’ll mention the most popular and I’ll explain the reasons I chose, or not, to use some of them.

Subjects such as publication, usage, distribution, or modifications of this app are out of scope of this report. Also out of the scope is image designing.

1.3 Development Tools

This project deals with several tools:

- Programming language: Java version 1 (1.7.0 51) for Android
- Programming tools: Java JDK, Java SDK, Android Debug Bridge
- Database: Sqlite3
- IDE: Eclipse
- Maps: Google maps, Google play services, Google API client
- Graphic design: Photoshop CS4

1.4 Definitions, Acronyms, and Abbreviations

- ANDROID: Operating System. Open-source operating system for mobile devices such as cell phones, and tablets. Android is Linux based.
- API: It stands for Application Programming Interface. This is a particular set of previously written classes and interfaces for the developer to use. Example: a calendar, Google Maps, etc.
- **WiN-GPS**: The CSUSB walking navigation application (deliverable from this project).

- **IDE**: It stands for Integrated Development Environment. It is a software application that provides the appropriate environment to develop a software application i.e. Eclipse for Android, JavaBeans for java, etc.

- **MOBILE APPLICATION**: It refers to any software or application that runs on current mobile phone and/or smart technology such as tablets.

- **SDK**: It stands for Software Development Kit. A library of previously written functions that aids a programmer to re-use code[3].

- **TAP/TAPPING/PRESSING/CLICK/DOUBLE-CLICK/LONG-CLICK**: Ways of input used on a mobile device through a touch screen interface. Tapping is quite equivalent to a click on a standard desktop computer.

- **ADB**: Android Debug Bridge is a tool that allows commands to be sent from a terminal to an Android device Zigurd, Laird (2010)[3].

- **GEO CODING**: worldwide location translator.

- **WALKING**: A path from point A to point B with everything in between.

- **POI**: points of interest.

- **PHOTOSHOP**: An image editing software developed by Adobe Corporation.
2. OVERVIEW OF DEVELOPMENT TOOLS

2.1 Maps

It’s long gone the time when static GPS devices were the only way to manage maps. Most map software based applications have been moved to smart devices. This project deals with Android devices, so I’ll mention some of the main map applications available for them.

Map Quest: It has been in the industry for a while now. They have the background expertise working with map software development. I personally used it on the web even before I knew about the existence of Google maps. Map Quest offers a developer account for free. The only prerequisite is to create an account with them. The basic services are offered for free for the first 15,000 transactions per month. Technical support is restricted to paid accounts, and developer forums are not quite common on the Internet. All in all, Map Quest is definitely a good choice to be considered when it comes to map development[7].

Yahoo Boss Geo Services: It offers location and search services worldwide. It is a paid service, which begins at six dollars for up to 10,000 queries per day. In the present case a paid service is a limitation[9].

Google Maps: This is nowadays the most known web mapping service for developers. Among other benefits, it offers unlimited free usage of maps through the Android APIs[6]. There’s plenty of technical information on the Internet, plus there are lots of independent developers sharing their experiences on websites such as Stack Overflow; where I happened to post most of my novice questions at the beginning of this
This development does not contemplate a budget for using a mapping service, therefore Yahoo maps in not an option. On the other hand, Map Quest has limited technical service, so in conclusion Google Maps is the adequate choice for this particular project.

In order to develop a google map based app, we must acquire a Google Maps API key. For that purpose it is necessary to have a google account. A shortened description on how to generate an API key is as follows. Sign in with your account and create a project in the google console developers. Figure 2.1 shows the website address as well as the project created for WiN GPS[6].

[Image of Google Developers Console]

Fig. 2.1: Project Defined in Google Console

Select the project; on the next screen select ”use Google API”[6]. See Figure 2.2:
Follow the instructions and an API key will be generated. This API key must be part of the AndroidManifest.xml file of the project or it won’t even start. This is part of the AndroidManifest.xml where the API key is present[4]. See Figure 2.3
More detailed instructions on how to generate an API key can be found at the following site: https://developers.google.com/maps/documentation/android-api/config

2.2 IDE

By the time this project started two IDE were available for Android development: Eclipse and Android Studio. Eclipse was released in the industry before Android Studio. I have used Eclipse for a couple of projects so I gained some expertise on it. That is the only reason I chose Eclipse over Android Studio on this particular task[4].

ECLIPSE the IDE of choice for this task may be used for different kinds of development i.e. Java, or Android Development. Using the figures 2.4 and 2.5, I will explain the components for an Android development[3].
Fig. 2.4: Eclipse Features I

As follows:

1: name of the project
2: Java code source
3: libraries imported for this project
4: database, database backups, and imported font
Following:

On 1: images used for the project. PNG format keeps the accuracy of images in the smaller size if they don’t have motion incorporated, which is the case for the present project. All the images for this project were manipulated with Photo-shop. For an Android project it is possible to use high, low, middle, extra high, and extra extra high density files. There is a tab for each of them so the operating system might automatically find them.

On 2: Layouts, a layout is nothing but a screen that a program will use to interact with the user. It is possible, sometimes it becomes mandatory though, to have more than one layout for the same program. When the user shifts a device from vertical to
horizontal the operating system will handle to shift the information from a vertical layout to an horizontal one. If there’s a XML file in the layout-land it will use it, if not it will just crop some items so they fit in the current screen position: either vertical or landscape. A layout that is used by the same program must have the same name either vertical or landscape.

On 3: Values. All the system values to be used.

On 4: the package’s definition file. This is the first file the application will look for before anything else.

Values, in light blue, might be defined globally, or they can be determined every time they are gonna be used in a program. For example, instead of typing an hexadecimal number every time we need an item to be black, we can simply define it as a value and use it by its name when necessary. The following figure 2.6 depicts this process:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<resources>
  <color name = "acid_green">#E0F1A</color>
  <color name = "bitter_lemon">#CAE000</color>
  <color name = "bitter_lime">#BFFFF0</color>
  <color name = "gainsboro">#D0DCCC</color>
  <color name = "black">#000000</color>
  <color name = "white">#FFFFFF</color>
  <color name = "olive">#949337</color>
  <color name = "grey">#272727</color>
  <color name = "cream">#E6E6E6</color>
  <color name = "light_blue">#28A981</color>
  <color name = "navy_blue">#163848</color>
</resources>
```

*Fig. 2.6: Values Definition*
2.3 Java

Java is a programming language, its virtues and capacities are beyond the study of this project. Eclipse, the IDE used for this project is Java based. The supporting libraries for eclipse are Java based as well. Naturally Java is the programming language of choice for this project. Java and Java for Android have quite similar capabilities, however there are some dissimilarities basically due to the devices that will host the applications[3]. A desktop, for instance, won’t have some of the problems a mobile device might have, such as: a discharged battery , an incoming call interrupting a process, etc. Besides, the hardware capabilities a desktop has are quite different from the ones of a mobile device. For instance: memory restrictions plays an important role in mobile development. Due to these differences a Java program for Android has a different life cycle than a Java source for a desktop has. The following graphic was presented in the course: “Programming Mobile Services for Android Handheld Systems”. It is a self explanatory Android application life cycle. [8]
2.4 SDK and Google Packages

The Software Development Kit has to be installed along with the IDE. However, there are some additional packages that must be installed, such as Google APIs, and Google Play Services. These packages can be obtained free of charge from the Google developers website: https://developers.google.com/maps/documentation/android-api/config

These are typical installations that won’t present a problem if the instructions are
followed properly[6].

2.5 ADB

Android Debug Bridge is the terminal for android. It becomes handy when a malfunction occurs with the project. It happens sometimes that a session ends abnormally for any reason. For instance, if things are not working properly on the next connection, the command ’adb kill-server’ from a terminal will fix the issue[3].

2.6 Emulators

An emulator is a piece of software that allows us to test an app if we don’t have an actual device or several devices for that matter. It is also useful to emulate the actual device behavior, such as a low battery[8]. Eclipse comes with its own emulator system, however it takes quite a while to load an app. This makes it cumbersome to test an application in the process of development. In order to overcome this problem, Blue Stacks is a solution.

Blue Stacks Is an Android emulator service. It’s fast and reliable. Once it’s installed it’s just a matter to locate the file applicationName.apk double click on it and the emulator will open it[5]. Blue stacks will open the application in a matter of seconds compared to minutes that the Android emulator takes.

Other than emulators, it is always safe to test an application over the actual device it’s supposed to work on. For this project I used Blue Stacks to test the app on a emulated tablet. I also tested the app on a phone LG Stylus with OS Android 5.1.1

2.7 Database

The native database for Android applications is SQLITE3. It is possible to work with other databases, however, according to its specifications a SQLITE3 database would
easily manage up to 10,000 records. This application is intended to manage no more than 500 locations at a time, and that’s quite optimistic, so SQLITE3 will handle with no problem WiN GPS database[8].
3. WIN-GPS MECHANICS

3.1 Summary

WiN-GPS is a personalized GPS Google maps based application. It comes with an installed database, in other words: it’s a Google map with an address book incorporated that will help you to get to that place you still don’t know the address to, or the location of it is not in a regular map.

3.2 screens

WiN-GPS uses touch screen inputs, and GUI buttons to navigate between the different app screens. This is shown in the next figure 3.1:
3.3 Walking Navigation

The initial screen lists all predetermined locations. The list consists of location code and description. From this screen the user might start a walking GPS, add a new location, or modify a location. See Figure 3.2.
By tapping on a location: the walking GPS will start. The following screen will show a map on a global scale with both user’s current location and target destination. On this screen the distance between user location and target destination is shown in meters. However, it is necessary to zoom on the map in order to get a close up of the area. Figure 3.3 shows the global view of the map.
Now, this figure shows a street level map where user will clearly see their starting location (‘you started here mark’). A blue dot will show the user’s current position. A red line will show their walking along the campus. Finally a red arrow indicates the selected destination. All the tools that are necessary to help the user to get to their destination are available on this screen. See Figure 3.4.
The following figures show a complete walking GPS, from the user’s starting point to their target destination. This print screens were generated while walking from point A to point B in a test. See Figures 3.5 and 3.6.
Fig. 3.5: Walking GPS I
While using the map, the user can at any point, press the "Info" button. This will display some information like the telephone of the selected destination. Figure 3.7 shows is the information screen:
Finally the user arrives to their destination. The user will visually see the complete path. Figure 3.8 shows an entire walking GPS from point A to point B.
Once the walking GPS has been reviewed, let’s remember that in the initial screen, the user can also create a new location by pressing the ”Add” button. See Figure 3.9.
The add button will display a form in a new screen. Here the user can create the information about a preferred location, or a favorite spot. See Figure 3.10.
Most of these attributes are pretty straightforward and the keyboard is part of the Android system. Most of the information to be entered here is pretty straightforward. See Figure 3.11.

Fig. 3.10: Add a New Location Screen
However, attributes Latitude and Longitude are values not easy to remember, neither they are handy without help. So, the application offers the possibility of selecting the location directly from a map. That’s what the "Get Coordinates" button is for. See Figure 3.12.

Fig. 3.11: Add a New Location Partially Filled Screen
After pressing the "Get Coordinates" button, a screen with a map will be shown. A blue marker indicates the current user’s location. In this street level map the user can select a destination by making a "Long Click" over the map. A red arrow will appear showing a location has been selected. By pressing "Save", latitude and longitude values are sent to the ”Add location” or ”Edit location” screens, depending on which one did the call. Figure 3.13 shows this:
Fig. 3.13: Select a location from screen

In this case the "New location" class called the "Get Coordinates". This is how the add a location screen will look like. Here the user may save a new location or discard the process. See Figure 3.14.
Let’s remember that in the initial screen the user can also make a ”long click” in order to modify an existing location. See Figure 3.15.
We have just reviewed the walking GPS functionality of the app, and the "Add" button. Now, let’s check the modification of a specific location. By making a long click over a specific location in the list the process will start. See Figure 3.16.
The application shows in a screen the current information of a location. The user may edit any attribute on this screen. The functioning of the button ”Get Coordinates” is just as it was described above. Something important to point out is the button ”Delete”. By pressing this button, that specific location will be deleted.
4. SOFTWARE ARCHITECTURE AND DESIGN

4.1 Main Activity Class

The entire app can be accessed from the main activity class. From this class the user can add, modify, or delete a location. The user also may use the walking GPS to navigate in the campus. The following UML package graphic explains it visually[1]. See Figure 4.1.
This UML graphic expresses the relationship between the classes DBmanager and MainActivity. See Figure 4.2.
MainActivity class instantiates the DBmanager class. The MainActivity class checks the existence of a database and opens it. This piece of code shows it. See Figure 4.3.

```java
final DBManager myDbManager = new DBManager(MainActivity.this); // Copy/create DB
try {
    myDbManager.createDataBase();
} catch (IOException ioe) {
    throw new Error("Unable to instantiate database");
}

try {
    myDbManager.openDataBase();
} catch (SQLException sqle){
    throw sqle;
}
```

A cursor is defined as "c". The results of a locations query are assigned to this cursor. The content of the cursor is assigned to a list. Finally the list is displayed[2]. Figure 4.4 depicts this process.
The application is listening to any input; When the user taps on a location, method OnItemClick activates. It will recover the string clicked and its position. The location code is recovered from the string. A query for that location is submitted in order to recover its info. With all that information a walking GPS is started. See Figure 4.5.

```
public void onItemClickListener(AdapterView<?> parent, View view, int pos_clicked, long row_clicked) {
    String val = (String) parent.getItemAtPosition(pos_clicked);  // get the whole text selected
    String s[] = val.split(“ “);  // getting locCode
    HashMap<String, String> locationMap = new HashMap<String, String>();
    locationMap = myDbManager.getLocationInfo(s[0]);
    if(locationMap != null) {
        Intent theIntent = new Intent(getApplication(), DisplayTrackingMap.class);
        double latitude = Double.parseDouble(locationMap.get("LocLat"));  // latitude
        double longitude = Double.parseDouble(locationMap.get("LocLong"));  // longitude
        theIntent.putExtra("_id", locationMap.get("_id"));  // auto generated Id
        theIntent.putExtra(“LocCode”, locationMap.get(“LocCode”));  // location code "AD"
        theIntent.putExtra("LocDesc", locationMap.get("LocDesc"));  // location description
        theIntent.putExtra("LocLat", latitude);  // destination latitude
        theIntent.putExtra("LocLong", longitude);  // destination longitude
        theIntent.putExtra("LocTelephone", locationMap.get("LocTelephone"));  // telephone
        theIntent.putExtra("LocComments", locationMap.get("LocComments"));  // comments
        startActivity(theIntent);  // display map with user location and destination for tracking
    }
}
```

Fig. 4.5: When a Location is Clicked

Should the user tap on the "Add" button, MainActivity will start the new location feature of the app. This method doesn’t need any parameters since we are creating a location from scratch. See Figure 4.6.
public class DBmanager extends SQLiteOpenHelper{
    String DB_PATH =null;

    private static String DB_NAME = "csusbDB";
    private SQLiteDatabase myDataBase;
    private final Context myContext;

    public DBmanager(Context context) {
        super(context, DB_NAME, null, 1);
        this.myContext = context;
        // retrieves DB path
        DB_PATH="/data/data/"+context.getPackageName()+"/"+"databases/";
    }
}

Fig. 4.6: Calling the methods to "Add" a new location

If the user "long clicked" on a location from the list, the class EditLocation, will be executed. The only parameter sent to the called activity is the location code. See Figure 4.7.

Fig. 4.7: On Long Click: Edit a Location

Those are the main functions on the MainActivity class.

4.2 DBmanager Class

WiN GPS has a database incorporated. It is handled with SQLITE3. The database name is csusbDB and it has two tables: android meta-data and locations. Table
android meta-data is a system required table, which holds information used by Android. It has one record indicating language and country i.e. en-US. The process to design this database was quite straightforward since there are no relationship among tables involved. There was no need to normalize the database. Figure 4.8 shows table meta-data structure:

![android_meta-data]

*Fig. 4.8: Table android meta-data: structure*

There’s just one main table used by the entire application, which is Location. This is its structure. See Figure 4.9.

![locations]

*Fig. 4.9: Table locations: structure*

Class DBmanager handles all the methods involving the locations information. Its tasks are to create a new location, to locate information related to them, to edit, or to delete a specific location. The following UML diagram of the DBmanager class reflects that[1]. See Figure 4.10.
Dbmanager instantiates MainActivity, NewLocation, and EditLocation classes. These classes control all the information related to locations. The following UML graphic depicts that. See Figure 4.11.
DBmanager inherits SQLiteOpenHelper, which is part of SQLite3. The database's name is defined and its path retrieved from the project assets as shown in Figure 4.12.

```
public class DBmanager extends SQLiteOpenHelper{
    String DB_PATH=null;
    
    private static String DB_NAME = "csusbDB";
    private SQLiteDatabase myDataBase;
    private final Context myContext;

    public DBmanager(Context context) {
        super(context, DB_NAME, null, 1);
        this.myContext = context;
        // retrieves DB path
        DB_PATH="/data/data/"+context.getPackageName()+"/databases/";
    }
```

Fig. 4.11: UML DBmanager instantiations

Fig. 4.12: Database definition
DBmanager has several methods that will be instantiated by MainActivity, EditLocation, and NewLocation: checkDataBase, openDataBase, Cursor, insertLocation, updateLocation, delete location, and getLocation among the most important.

For instance, the method update location will receive values for the attributes modified and it will update the locations table using a SQL sentence pretty straightforward as figure 4.3 depicts.

```java
/**************************************************************************
 * update a location
 */

public int updateLocation(HashMap<String, String> queryValues){
    SQLiteDatabase database = this.getWritableDatabase();
    ContentValues values = new ContentValues();

    values.put("locCode", queryValues.get("locCode"));
    values.put("locDesc", queryValues.get("locDesc"));
    values.put("locComments", queryValues.get("locComments"));
    values.put("locTelephone", queryValues.get("locTelephone"));
    values.put("locLat", queryValues.get("locLat"));
    values.put("locLong", queryValues.get("locLong"));

    return database.update("locations", values,
        "locCode" + " = ?", new String[] {queryValues.get("locCode") });
}
```

Fig. 4.13: Edit Location, Code

Method deleteLocation receives location code as a parameter. That location is deleted from table locations using the method deleteLocation from the DBmanager class. See Figure 4.14.
Another functionality from the DBmanager class is the getLocationInfoMethod. This method receives as a parameter the location code. It uses a hashmap that will be populated with a query for that specific location. See Figure 4.15.
The last method to mention from DBmanager class is the insertLocation method. It receives as a parameter a string with the values for the new location. These individual values are recovered from the string and then used in an insert query command. See Figure 4.16.

```java
public void insertLocation(HashMap<String, String> queryValues){
    SQLiteDatabase database = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("locCode", queryValues.get("locCode"));
    values.put("locDesc", queryValues.get("locDesc"));
    values.put("locTelephone", queryValues.get("locTelephone"));
    values.put("locComments", queryValues.get("locComments"));
    values.put("locLat", queryValues.get("locLat"));
    values.put("locLong", queryValues.get("locLong"));
    database.insert("locations", null, values);
    database.close();
}
```

Fig. 4.16: Creating a New Location.

### 4.3 DisplayTrackingMap Class

This class handles a street level map with the user’s location, destination, and walking GPS. It is not possible to instantiate a Google Map object directly into the app. The way to do it is by adding a MapFragment and then instantiating the FragmentActivity object, which is part of the Google Play Services library I mentioned before. Figure 4.17 shows the UML instantiation[1]:
There's more than one way to instantiate a map fragment in an Android app. It
could be done directly in the activity or in the activity’s layout[2]. The approach I
used is the second one. The following code shows the fragment definition in the Map
layout. See Figure 4.18.
This is instantiated in DisplayTrackingMap class. See Figure 4.19.
It also inherits methods:

- OnMarkerClickListener,
- ConnectionCallbacks,
- OnConnectionFailedListener,
- LocationListener,
- OnInfoWindowClickListener

By using a map fragment Google maps will display the map, it will connect the activity to the API services, and it will manage the basic controls of the map such as zooming.

Activity DisplayTrackingMap on its create method defines the connection between the activity and its layout. It gets target latitude and longitude from table locations. It will set markers on user and target locations. Besides, it will turn off the control zoom and it will set up the camera[2]. See Figure 4.20.

```java
public class DisplayTrackingMap extends FragmentActivity implements
  OnMarkerClickListener,
  ConnectionCallbacks,
  OnConnectionFailedListener,
  LocationListener,
  OnInfoWindowClickListener

{ // initial user's coordinates
  public double latitude;
  public double longitude;
  public Location lastloc = null;
  public boolean flagOrigin = false;
  public Location ORIGIN;
  public Location DESTINY;
  private Marker #Destiny;
  private Marker #Origin;
  private Marker #LastSelectedMarker; // keeps track of last selected marker
  public Location #LastLocation;
  public Location #CurrentLocation;
  public double targetlat; // target coordinates
  public double targetlon;
  ProgressDialog dialog; // "waiting for location" window

  private GoogleMap #map;
  private GoogleApiClient #googleApiClient;

  // Activity displaytrackingmap on its create method defines the connection
  // between the activity and its layout. It gets target latitude and longitude
  // from table locations. It will set markers on user and target locations.
  // Besides, it will turn off the control zoom and it will set up the camera[2].
```
Once the map is settled, it is necessary to track the user’s movements across the map. For this is necessary to import the library for the function callbacks. See Figure 4.21.

This library allows callbacks. A function to be called when the user connects or disconnects. In WiN GPS I implemented method onConnected to make requests about the user’s location changes. First I define the REQUEST parameters that will be used by the callback[3]. See Figure 4.22.
This request is used on method OnConnected. The requestLocationUpdates relates the callback with method onLocationChanged, which will be executed every time the user’s location has changed[8]. See Figure 4.23.

Method onLocationChanged will update continually the user’s location. It will also calculate the distance between the current location of user and the selected destination. Method showMyLocation draws the user’s path one line at a time. Figure 4.24 depicts that.
public void onLocationChanged(Location location) {
    showMyLocation();
    mCurrentLocation = location;
    updateCurrentLocation();
}

private void updateCurrentLocation() {
    double mLatitude = mCurrentLocation.getLatitude();
    double mLongitude = mCurrentLocation.getLongitude();

    if (mCurrentLocation != null) {
        // Calculate the distance between current location and target
        float[] results = new float[3];
        Location.distanceBetween(mLatitude, mLongitude, targetLat, targetLon, results);
        BigDecimal bd = new BigDecimal(results[0]); // results in meters
        BigDecimal rounded = bd.setScale(2, RoundingMode.HALF_UP);
        String msg = "Distance curr -> target = "
            + Math.round(rounded.doubleValue()) + " mts", Toast.LENGTH_LONG).show();
    } else {
        String msg1 = "last location null ";
        Toast.makeText(getApplicationContext(), msg1, Toast.LENGTH_LONG).show();
    }
}

public void showMyLocation() {
    if (mGoogleApiClient == null) {
        if (lastLoc != null) {
            mMap.addPolyline(new PolylineOptions()
                .add(new LatLng(lastLoc.getLatitude(), lastLoc.getLongitude()), new LatLng(mCurrentLocation.getLatitude(),
                .width(5)).
                color(Color.RED));
        }
        // this condition will be met only the first time
        if (lastLoc == null) {
            dialog.dismiss(); // dismiss "Waiting for location..." message
            String msg = "Connection acquired, \n\nPlease hit the zoom button ";
            Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_LONG).show();
            mCurrentLocation = LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
            latitude = mCurrentLocation.getLatitude();
            longitude = mCurrentLocation.getLongitude();
            ORIGIN = new LatLng(latitude, longitude);
            flagOrig = true;
            addMarkersToMap();
        }
        lastLoc = mCurrentLocation;
    }
}
This is a recursive process that will take while the user walks to the selected location.

4.4 4.4 New Location Class

NEW LOCATION class is to create a new location in table locations. It uses a layout to input data. Eclipse allows to manage layouts graphically. This is a print screen of the New Location layout. See Figure 4.25.

![New Location graphic layout in Eclipse](image)

Fig. 4.25: New Location graphic layout in Eclipse

Internally this layout is constructed by one row tables of text views and edit fields as follows. See Figure 4.26.
This is how a Java Android class relates to its layout, in this case NewLocation to layout new-location. See Figure 4.27.

The user inputs the location information and press the Save button, the method addNewLocation calls the corresponding method on the DBmanager class that will insert a new location in table. See Figure 4.28.
In case the user don’t remember latitude and longitude of a location, then class Get Coordinates will solve this. If everything is correct, latitude and longitude will be retrieved. The following code shows that portion of the program. See Figure 4.29.

```java
// ON CLICK for button "Save"
public void addNewLocation(View view) {
    HashMap<String, String> queryValuesMap = new HashMap<String, String>();
    queryValuesMap.put("locCode", locCode.getText().toString());
    queryValuesMap.put("locDesc", locDesc.getText().toString());
    queryValuesMap.put("locTelephone", locTelephone.getText().toString());
    queryValuesMap.put("locComments", locComments.getText().toString());
    queryValuesMap.put("locLat", locLat.getText().toString());
    queryValuesMap.put("locLong", locLong.getText().toString());
    final DBManager myDBManager = new DBManager(NewLocation.this);
    try {
        myDBManager.insertLocation(queryValuesMap);
        catch (SQLException sqle) {
            throw sqle;
        }
    } catch (ActivityNotFoundException e) {
        this.startActivity(new Intent(NewLocation.this, MapActivity.class));
    }
}
```

Fig. 4.28: Saving a New Location on Locations Table

That wraps up New Location class.
4.5 Edit Location Class

EditLocation class is to modify a specific location’s data. It receives location ID as a parameter and looks up its information into table locations; then it uses a hashmap as a parameter for class DBmanager to get the location info. See Figure 4.30.

```java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.edit_location);
    final DBManager myDBManager = new DBManager(EditLocation.this); // Copy/create DB
    locCode = (EditText) findViewById(R.id.locCode);
    locDesc = (EditText) findViewById(R.id.locDesc);
    locTelephone = (EditText) findViewById(R.id.locTelephone);
    locComments = (EditText) findViewById(R.id.locComments);
    locLat = (EditText) findViewById(R.id.locLat);
    locLong = (EditText) findViewById(R.id.locLong);

    Intent i = getIntent();
    String _id = i.getStringExtra("locCode");
    try {
        HashMap<String, String> locationInfo = myDBManager.getLocationInfo(_id);

        // Loading location info
        if(locationInfo.size() != 0){
            locCode.setText(locationInfo.get("locCode"));
            locDesc.setText(locationInfo.get("locDesc"));
            locLat.setText(locationInfo.get("locLat"));
            locLong.setText(locationInfo.get("locLong"));
            locComments.setText(locationInfo.get("locComments"));
            locTelephone.setText(locationInfo.get("locTelephone"));
        }
    }
    catch(SQLiteException ex) {
    }
```

Fig. 4.30: Look up for location information

Any modified information will be saved by pressing the button ”Save”. All location information is handled to DBmanager; it will update location info. See Figure 4.31.
public void editLocation(View view) {
    HashMap<String, String> queryValuesMap = new HashMap<String, String>();
    locCode = (EditText) findViewById(R.id.locCode);
    locDesc = (EditText) findViewById(R.id.locDesc);
    locTelephone = (EditText) findViewById(R.id.locTelephone);
    locComments = (EditText) findViewById(R.id.locComments);
    locLat = (EditText) findViewById(R.id.locLat);
    locLong = (EditText) findViewById(R.id.locLong);
    queryValuesMap.put("locCode", locCode.getText().toString());
    queryValuesMap.put("locDesc", locDesc.getText().toString());
    queryValuesMap.put("locLat", locLat.getText().toString());
    queryValuesMap.put("locLong", locLong.getText().toString());
    queryValuesMap.put("locTelephone", locTelephone.getText().toString());
    queryValuesMap.put("locComments", locComments.getText().toString());
    final DBOManager myDbManager = new DBOManager(this); // update location
    try {
        myDbManager.updateLocation(queryValuesMap);
    } catch (SQLException sqlc) {
        throw sqlc;
    }
    finish();
}

Fig. 4.31: Update a location information

It is also possible to eliminate a location. If the user decides to do this, button "Delete" should be pressed. The following code will delete selected location. See Figure 4.32.
That is in general the structure of the Edit Location class.

4.6 Get Coordinates Class

Get Coordinates class is used by the NewLocation and the EditLocation classes. Attributes Latitude and Longitude are decimal numbers with several digits, therefore it might be quite possible that user needs some help with them. Both, edit and insert location layouts, have a Get Coordinates button. By pressing this button a map will be displayed. From this map, the user can select a location by making a long click on the map. Coordinates latitude and longitude will be returned to the caller routine. This class has a similar structure than the DisplayTrackingMap class. It defines a fragment that will hold a map. A request for the callback is defined as well. The values for this request might be different though, since it’s not necessary to check a location status changed frequently. It’s just necessary to detect a long click and transform it into latitude and longitude values. These values will be returned them to the caller routine. See Figure 4.33.

```java
// ON CLICK "Delete" button
public void removeLocation(View view){
    Intent i = getIntent();
    String _id = i.getStringExtra("locCode");
    final DBManager myDbManager = new DBManager(EditLocation.this);
    try {
        myDbManager.deleteLocation(_id);
    } catch (SQLException sqle){
        throw sqle;
    }
    this.callMainActivity(view);
    finish();
}
```
Fig. 4.33: Defining Callback parameters

The parameters request for the callback function. Intervals to determine the frequency of time to check for any possible variation on the map, such as a long click over the map. A high priority request means that the most precise tool for locating a coordinate must be used. This accuracy is controlled by Google Maps[2]. See Figure 4.34.

```java
// Settings for the call back
private static final LocationRequest REQUEST = LocationRequest.create()
    .setInterval(2000) // user position updated every 2 seconds to view
    .setFastestInterval(15) // 16ms = 60fps
    .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY); // for an accurate location
```

Fig. 4.34: Connecting map layout to activity (class)

This is how the class relates to its layout in order to use a fragment that will hold the map, and it connects to Google-maps[2]. See Figure 4.35.
When a long click over the map is detected a red arrow will be drawn on the map to indicate that a location has been detected. A Google method: addMarker is used for that. See Figure 4.36.

Method requestLocationUpdate is to connect the callback function with the onLocationChanged method. See Figure 4.37.
4.7 Show Location Info Class

The locationInfo class is a window with an specific location information that can be reached from classes newLocation and editLocation. Figure 4.38 shows its graphic layout:

```java
public void saveLocation(View view){
    Intent intent = new Intent();
    intent.putExtra("locLat", locLat);
    intent.putExtra("locLong", locLong);
    setResult(RESULT_OK, intent);
    finish();
}
```
Class ShowLocationInfo define the layout used to display the info. See Figure 4.39.
When the user is ready to leave the location information window the go back button control will return control to the last active screen seen. See Figure 4.40.
4.8 Changing Font for the Entire Application

This should not be treated as a class itself, however I believe it is important to be mentioned because of the particular process to change the font for an Android application. This is the way to do it under Android development[2]. First, import the font into the assets folder of the project, Lato-Light.ttf in this case. See Figure 4.41.
Then the font must be defined in the activity you need to use it. See Figure 4.42.

```java
setContentView(R.layout.activity_main);

// Change the default font for button1
Button button1 = (Button)findViewById(R.id.button1);
button1.setTypeface(Typeface.createFromAsset(getAssets(), "data-light.ttf"));
```

Fig. 4.42: Font Defined to be Used in a Button

That wraps up the app software architecture.
5. CONCLUSION AND FUTURE DIRECTION

5.1 Conclusion

It has been very interesting to learn about Android development. I read the book "Programming Android" by Zigurd Mednieks, and Laird Dornin[3]. I also took the online class: "Programming Mobile Services for Android Handheld Systems" offered by The University of Maryland USA in order to get acquainted with the environment of Android, and specifically with Google maps[8]. In the end, one of the deliverables is an app, which is developed and it’s ready to be deployed. Another deliverable is this document, which explains the most relevant details of this specific development. The application is fully functional on Android mobile devices and it can be implemented if CSUSB chooses to. On a personal level I acquired some knowledge on a very marketable field as Android development is. It’s a good starting point before entering in the development of more challenging applications.

5.2 Future Direction

Android development and Google maps offer a lot of functionalities that could be additionally implemented on the application. The database can be manipulated as well according to new requirements if needed. Of course, it can also be used as it is right now.

Besides, it is also possible to standardize the application so it could be used in places other than CSUSB. I, for instance, have tested the app on the community I
live in.
REFERENCES


