Framework for a User-Populated Map of Geotagged Images

BSc (Hons) Computer Science

Matthew Reza Tarabadi (14822525)

Project Supervisor: Lyn Pemberton

Abstract

This project aims to create a proof of concept prototype for the framework of an online map of viable locations for a person to skateboard, populated by users and delivered as an Android application.

Contents

Chapter 1 - Introduction

Few sports or hobbies are as versatile as skateboarding. A game of football is always played with two goals opposite each other, but a person can skate wherever there is fairly smooth and fairly hard ground, and despite a large number of skateparks all over the UK people are constantly finding new areas that are viable to skateboard in; known as spots. Additionally, a number of skateparks are not documented online, and the locations of spots are primarily word-of-mouth.

1.1 Project Aim

The aim of this project is to create the framework for an online map of skateparks and spots, populated through user interaction and accessible through a mobile application. Users have the option to upload a previously taken photo of a spot – provided the photo is geo-tagged – or take a photo through the application, provide a name and description for the spot, and upload the spot to a database. Users will be able to see nearby spots in the application, and upon selecting a spot will be met with more information about the spot and the option to get directions from their current location to the spot. The application will deal with local data, acting as a proof of concept.

1.2 Requirements

Requirements for the project were obtained via informal interviews with a random sampling of the target audience. Interviewees were given an outline of the purpose of the application – an application to share and find skateparks and spots – and if interested were asked what they deemed necessary or important for the application to have. After collating and analysing the results, five key objectives were identified:

1. **The application must be able to display spots effectively**. It's important for users of the application to be able to see information of all nearby skate spots, such as a photograph of the area and how far away it is. A photograph serves as both a reference to be able to locate the spot, and to see what obstacles the area may have or what condition the area is in
2. **The application must be able to use a previously taken photograph with a geotag or take a new photograph when creating a spot**. If a user comes across a new, undocumented skate spot, they should be able to take a photograph and upload the spot without having to exit the application
3. **Directions need to be able to be given from the current location to a selected spot**. Users must be able to obtain directions to a selected spot from their current location, through Google Maps or a similar service
4. **Data about spots must be able to be written to and read from the database efficiently**. Uploading and retrieving data from the database must be efficient, to ensure the application runs smoothly and doesn't suffer a slowdown when dealing with the database
5. **The application must be responsive and simple**. An overwhelming majority of people interviewed gave responsiveness of the app as a high priority, especially when compared to the design of the app, preferring the application to be as responsive and simple as possible

1.3 Deliverables

- **The Android application** as an APK (Android Package Kit), a file that can be installed on most Android smartphones. All required features will be implemented.
- **The project report** detailing the lifecycle of the project, from planning to implementation and evaluation

1.4 Relevance to Degree

This project requires aspects from several previously studied modules:

CI101 – Programming
The Programming module undertaken in first year explained the basic concepts and principles of programming, specifically with regards to Java.

CI102 – Introduction to Databases
The Introduction to Databases module studied in first year introduced relational databases, and how to manipulate them using SQL.

CI236 – Integrated Group Project in Computing
The Integrated Group Project module taken in second year not only taught how to work on software projects in a group, but the project carried out involved creating a Java application interacting with relational databases. This helped bring together what I had learnt in both CI101 and CI102, and proved useful when creating and interacting with the database in this project.

CI285 – Introduction to Functional Programming
Despite the module focusing on Haskell as opposed to Java, Introduction to Functional Programming introduced me to APIs, specifically the RESTful API. While REST services are not used in this project, I was able to implement third-party APIs.

CI360 – Mobile Application Development
Mobile Application Development introduced developing applications for Android devices, which plays a key role in this project. Undertaking this module ensured correct practices were followed with regards to developing applications for a mobile platform.

Chapter 2 - Background Research

Background research was carried out to learn about any previous implementations or usage of geotag scraping, the potential applications of geotag scraping, and the availability of online resources for skateboarding. In addition to this, research was carried out with regards to potential APIs and libraries the project may have to utilise.

2.1 Location Metadata

The majority of Android smartphones have an option to enable geotagging of photographs, usually accessed through the settings menu. When enabled, location information in the form of GPS coordinates is stored in the Exif (Exchangeable image file format) metadata of the image, effectively assigning a location to the image. As the metadata is attached to the image itself, many images on social media contain metadata that can be accessed and used.

World Seer[2], described as a realtime geo-tweet photo mapping system, is a program that collects images with geotag metadata from twitter and plots them both on a map and in street view, allowing users to see the latest geotagged images tweeted for given areas and keywords. However, as the program also downloads and uses photos exclusively tweeted in Japan for a photo clustering system, the system architecture utilises two databases; the first for all geotagged images tweeted and the second for images taken within Japan. The first of the databases simply stores the URL of an image and the metadata accompanying the image.

2.2 Skateboarding

Despite being a sport that's ever gaining popularity, so much so that it's been approved for the 2020 Olympic Games(15), skateboarding facilities are often poorly looked after – in some cases poorly built – resulting in damage occurring to the facilities. Furthermore, skateboarding facilities are few and far between, often being expensive to build and maintain. Coupling this with the fact that skateboarding came in to prominence in the 1970s as a way for surfers to mimic surfing on concrete streets (15), it's no surprise that skateboarders prefer to utilise obstacles – such as stairs, ledges, or rails – found within cities, towns, and other areas. However, online resources for these areas where skateboarding is viable are few and far between, and as such these areas can be difficult to locate.

2.3 APIs and Libraries

As Android provides no functionality for reading the metadata of an image, the *metadata-extractor* library[1] was used. The library has support for Exif metadata, so geolocation information can be read and the GPS coordinates stored.

With regards to commercial APIs I researched information about the Google Maps API, as my initial thoughts were to utilise the API within my application to be able to provide directions to a given spot. Multiple APIs are offered, dependent on the target platform and what needs to be achieved by the target product. However, for my intended use – to provide directions from one location to another – the API does not need to be implemented. Instead, the Google Maps application can be launched by my application (13), supplying the two locations for directions to be given from and to.

Chapter 3 - Design

Described in this chapter are the plans for development of the application and implementation of required features.

3.1 Methodology

Methodologies are adopted to shape the development cycle of a project and help ensure the development sees completion. Different methodologies can vary greatly in approach, but can be loosely grouped in to two schools of thought; sequential – such as the Waterfall approach – and iterative. By utilising an iterative approach towards this project the mobile application can be divided in to four main iterations, with each iteration building upon the previous. This allows testing and evaluation of each implementation after creation to ensure a satisfactory result.

3.2 Schedule

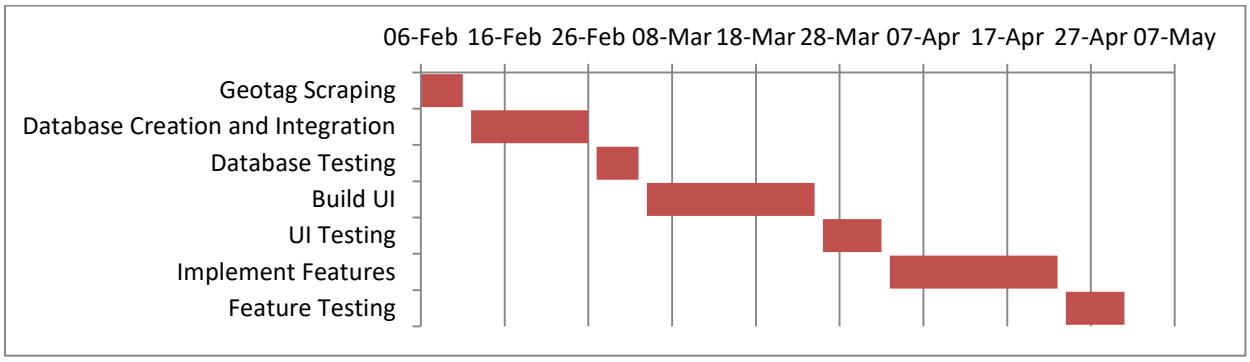

Figure **1**: Iterative Gantt Chart Detailing the Development Schedule

A Gantt chart detailing the development schedule is illustrated in Figure **1**.

The development cycle of this project consists of four key iterations. Implementations are tested upon their completion, ensuring the implantation in question is working as intended before the project progresses. This is key to the success of the project, as each iteration builds upon what is achieved by the previous iteration.

3.2.1 Geotag Scraping

As reading image metadata is an integral part of the project, it is the first aspect to be implemented. The mobile application will need to be created, and the process of selecting an image from the device's storage and reading the location metadata will need to be implemented to ensure the project can continue.

It is worth noting that the mobile application will have a very limited UI until the full UI is built, as all that is required it to select an image from the device's storage.

3.2.2 Database Creation and Integration

The database will need to be created and implemented within the application to store images and their location metadata. Geotag scraping should be operational, and will write to the database.

Adopting ideas from the World Seer program (see **2.1**) a reference to the image will be written to the database as opposed to the image itself. This reference will be converted to and stored as plain text for insertion, and parsed into a reference upon retrieval.

Testing on the database will be carried out by ensuring selected images and their location data write to the database, and all items in the database can be retrieved. Ensuring functional implementation of the database is crucial to the success of the project, as without a database no data can be stored or retrieved.

3.2.3 Building the UI

The Android framework provides functionality for the use of XML layout resources, XML files describing the layout of screens for an Android application. Multiple XML files will need to be created to build the UI, as multiple features are required. Following the creation of all required XML layouts, the UI logic will need to be created and implemented. This will utilise the geotag scraping features achieved in the first iteration of the development, and interact with the database created in the second. The UI will need to be able to display nearby skate spots, allow the creation of a new spot through either image selection of taking a photograph, and enable users to pick a certain skate spot to see more information about the spot, and obtain directions to get to the spot.

3.2.4 Implementing Additional Features

Following the creation of the UI and the underlying logic behind displaying the UI, a number of additional features are required to be added before full functionality of the application can be claimed. These features consist of enabling the user to take a photograph, and use the photograph to create a new skate spot, enabling the user to obtain directions to a specified skate spot through Google Maps or a similar service, and adding the ability for the application to create and display notifications, informing the user of any issues or communicating any necessary information.

Chapter 4 - Implementation

This chapter outlines the creation of the mobile application, utilising the designs in Chapter 3, and explaining how required functionality was achieved.

4.1 Geotag Scraping

The objective of the first iteration was to create a basic mobile application with the ability to read location metadata for a given image. The application would let the user chose an image from their device's storage, and parse the image's metadata for latitude and longitude values.

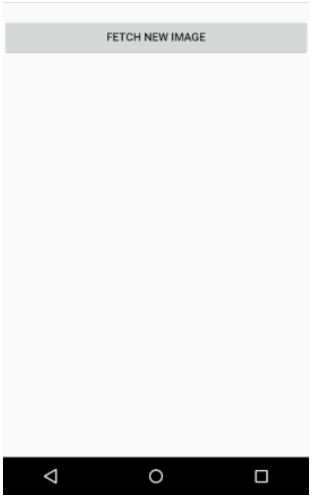

Figure **2**: Initial application UI

Given the simple nature of the task to be implemented the full user interface is not implemented, with a simple UI being implemented instead containg a single unstyled button, of which a screenshot can be seen in Figure **2**. When this button is selected an Intent[5] is created utilising the predetermined Android Intent **ACTION_PICK** and the specified MIME data type of image, and launched as an activity returning a result. This causes the device's gallery to open, allowing the user to select any image stored on the device or on external storage, such as an SD card.

Selection of an image results in the closing of the device's gallery, storing of the URI[6] of the image in the data of the image selection intent and returning to the main activity. The URI is obtained from the data of the selection intent, and is then used to open an input stream of the selected image, which is then in turn used to create a buffered input stream. Metadata can be parsed from the buffered input stream, allowing for the scraping of an image's geotag data.

4.2 Database Creation and Integration

The second iteration involved the creation and integration of an SQLite database with the mobile application. As Android online documentation[3] has in depth information on the subject, no issues arose during the creation and integration of the database. A custom database helper class, called SpotReaderDbHelper, was created, and is utilised for both the creation and for access to the database. This database would hold the location data of an image, and a reference to the image's path on the current device, saved as a String.

4.2.1 Testing the Database

Testing on the database was carried out by ensuring selected images are inserted in the database, along with their location metadata. Additionally, images and their metadata must be able to be retrieved from the database via the application activity, and as such retrieval was implemented for testing purposes.

4.3 Building the UI

At this point the database is operational with no faults and the geotag scraper works. To properly handle the data contained within the database, and to provide functionality to the mobile application, the third iteration focused on creating the UI and UI logic.

The Android framework handles UIs by utilising layouts, a way to define the structure of an interface. Implementation of layouts can be handled in two ways; creating an XML file containing declarations of UI elements or instantiating layout elements programmatically. Additionally, both of these methods can be utilised in conjunction to create and manage a user interface.

4.3.1 XML Layouts

Due to the nature of the project, several XML layouts were created. The layouts are of varying degrees of complexity and enable various different aspects of the UI.

4.3.1.1 The Toolbar

The first aspect of the UI to be created was the toolbar, as the feature to create a new spot would be contained within the application's toolbar. The toolbar must have the feature to create a new skate spot, via the gallery or the camera, a feature to enable users to manually refresh the list of data, and provide an option for users to open the settings screen. These aspects of the toolbar are achieved by utilising two XML layouts – one containing a toolbar widget and one detailing the menu layout.

A menu layout is described by utilising a menu tag containing item tags, each tag having an id and a title. Given the requirements of the menu, the layout is composed of four items; upload photo, take photo, refresh, and settings. The layout describing a toolbar widget is also created, with these two XML files being used together by an activity to programmatically create the toolbar, add the menu to the toolbar, and add functionality for selecting menu items.

In addition to the main application toolbar, a second toolbar was created. This second toolbar is to be displayed when on the settings screen, and enables the user to return to the previous screen through the navigation button. As a user will not be able to create a skate spot from the settings screen, this is all the functionality required for the second toolbar.

4.3.1.2 The Main Screen

The main screen is the interface the user is greeted with upon opening the application. This screen was required to display information obtained from the database, and provide the user with the option to create a spot either by utilising a previously taken photograph or enabling the user to take a new photograph. Given that the latter is contained within the application's toolbar the main screen's XML layout consists of two aspects; inclusion of the toolbar and a ListFragment to display obtained data.

This required another XML file to be created, detailing the layout of a row to be displayed in the ListFragment. Given that spots would have an image, location, title, and description attached to them, the layout contains an ImageView – to display the picture of the spot – and two TextViews; one for the title and one to display the distance between the user's current location and the location of the spot. Upon selecting the spot, a dialog is shown containing the description of the spot as well.

Much like the toolbar these XML files are used programmatically upon loading of the application to create and display a list of spots.

4.3.1.3 The Settings Screen

Unlike other aspects of the user interface, a settings or options screen is implemented by utilising the PreferenceScreen layout. Each option is described as a Preference component – such as CheckBoxPreference for an option with a check box, or EditTextPreference for an option opening a dialog, taking input, and saving the input as a String. Due to the simplicity of my settings screen, the XML file describes a single CheckBoxPreference, enabling a user to disable or enable a dialog reminding the user of the need for images to require location metadata when selecting an image from storage. The dialog is shown by default, and can be disabled either through the settings screen or by selecting the CheckBox on the dialog itself.

4.3.1.4 Dialogs

Dialogs are used as a method of interaction with the user, both alerting the user to issues and enabling user interaction with the data. Each dialog required has an XML file describing its layout, with exception for a dialog created programmatically, as different dialogs will be required to handle and display different forms of data. Creation of spots, selection of a spot from the list, and error messages are all handled via dialogs.

Spots require a title and description in addition to a geotagged image. Due to this a dialog is to be displayed upon the user selecting an image from the device's storage or taking a new photograph through the application, requiring the user to enter a title and short description of the spot. The dialog for creating and uploading a new spot is described with an ImageView and two EditText components; one for title and one for description. The ImageView enables the user to see the image to be uploaded, ensuring it is the correct image and of a good quality, and the EditText components enable the user to input a title and description of the spot before uploading.

In addition to a dialog being displayed when attempting to create a new spot, a dialog must be displayed when an existing spot is selected from the list. The dialog displaying information after selecting a spot is similar to the dialog for creating a spot, utilising TextView components as opposed to EditText components to display the title and description. Additionally, this dialog has a button, enabling users to obtain directions to the selected spot from their current location.

Two of the dialogs are simple error messages displayed in certain cases, and as such only contain a TextView displaying a previously specified message.

A further dialog reminding the user of the need for location metadata before launching the gallery is also utilised, however this dialog is intended to be seen at least once by users and has a checkbox to disable it from appearing again. As this is dependent on preference data and likely to be disabled, this dialog is only created programmatically if needed.

4.3.2 Activities

Activities are used to utilise an XML layout file or files and handle the logic required. Due to the design of the application – with regards to the intent to use dialogs to handle user interaction and

display additional information – only four activities are used in the application; the main activity, the settings activity, the image picker activity and the camera activity. Furthermore the latter two of these activities are handled by the Android platform; meaning only the first two activities were required to be created for the application.

4.3.2.1 The Main Activity

The main activity consists of a significant amount of the logic for the user interface, responsible for querying the database, scraping geotag data from selected images, and building and updating the UI. The first two of these tasks had already been implemented successfully, allowing focus to remain on the final task.

Upon launching the application, two main processes take place; the XML layout containing the toolbar widget added to the current activity, and a custom LocationGetter class attempts to retrieve location information from the device via the device's location services.

With a toolbar added to the activity, the method *onCreateOptionsMenu()* can be overridden and the XML file containing the menu layout loaded using the MenuInflater [(8)]. The method *onOptionsItemSelected()* is also required to be overridden, as this method is called when a menu item is selected and as such enables implementation of the course of action to be taken dependent on the item selected:

- **Upload Photo** – check if the user has disabled the default reminder dialog, and if not display it. If the dialog has been disabled, or when okay is selected on the dialog, open the gallery and let the user select an image in the device's storage for spot creation
- **Take Photo** – obtain current location and let the user take a photograph, using location information and image taken for spot creation (see **4.4.1**)
- **Refresh** – obtain current location using the LocationGetter and query the database for data, rebuilding the rows for the ListFragment if a location was obtained and displaying a dialog if the current location was unobtainable
- **Settings** – start the settings activity

If the LocationGetter class is unsuccessful in retrieving location information from the device when opening the application a dialog informing the user to enable location services and refresh the list, described in an XML resource, is displayed. If location information is obtained from the device, the current location is saved and an SQL statement retrieves all data from the database, with the data being parsed into an ArrayList of Spot objects. Each Spot object has a title, description, latitude, longitude, and image URI.

As information retrieved from the database is stored in an ArrayList ,and the layout for the main screen features a ListFragment, a custom ArrayAdapter [(7)] – named SpotAdapter – was implemented, allowing for the data in the ArrayList to be used in conjunction with the XML layout for the row to be used in the ListFragment. The SpotAdapter creates a row from the XML layout for each item in the ArrayList – setting the text in the TextView components to their relevant Strings, inserting the relevant image in the ImageView, and setting a custom onClickListener to enable the displaying of a dialog when a row has been selected – and is used to populate the ListFragment.

Five dialogs are required for the main activity; a dialog for creating a spot, a dialog for selecting a spot from the list, a dialog for alerting a user the selected image has no metadata, a dialog alerting the user current location information could not be obtained, and a dialog reminding users that to select an image from storage to use requires the image to have location metadata.

The latter of these dialogs is created programmatically after using an AlertDialog Builder [(9)], while the four other dialogs are described with XML layouts. Furthermore, the dialogs alerting the user are the same in layout – with the message displayed being the only difference between the two – and as such can be displayed by inflating a new View with the relative XML file, and using an AlertDialog Builder to both set the current view to the dialog and add a button to the bottom of the dialog enabling the dialog to be closed.

A method is implemented – taking a parameter of the URI of the selected image – to deal with the dialog for spot creation. The method inflates a new View with the XML layout for the spot creation dialog, loads the image specified into the ImageView, assigns the two EditText components to variables to enable retrieval of the Strings used as input, and uses an AlertDialog Builder to display the dialog. When the okay button is selected, checks ensure the title and description are neither empty nor too long, and if satisfying these conditions the spot is written to the database and the dialog is closed.

The final dialog implemented was the dialog displayed upon selecting a spot from the list of displayed spots. Much like the dialog for spot creation, a method inflating a new View with the dialog layout is created. However, the parameter for this method is an integer representing the position of the spot selected in the ListFragment. This integer is used to retrieve data about the spot from the local ArrayList, and the image, title, and description of the spot are displayed via the relative layout components. Additionally, a custom onClickListener is created for the button in this layout, to enable users to obtain directions to the specified spot (see **4.4.2**)

4.3.2.2 The Settings Activity

The settings activity consists of considerably less code than the main activity, due to having considerably less functionality. A FragmentManager [(10)] is used to ensure preferences are loaded in the *onCreate()* method, and the toolbar for the settings page is loaded in the *onPostCreate()* method, ensuring the toolbar is displayed. Attempting to load the toolbar in the *onCreate()* method results in the settings page being loaded over the toolbar and overlapping the toolbar, with text from the toolbar visible behind the items on the screen.

4.3.3 Problems Encountered

Due to the level of functionality required from this stage of development, a small number of problems were encountered and subsequently dealt with. Additionally, some aspects of the application were reworked to be more efficient.

Creation and implementation of the UI caused some changes to be made to the database created in the second iteration, to accommodate storing a title and description with an image and location.

4.3.3.1 Responsiveness Issues

A key requirement of the project is ensuring the application stays responsive to user action. Two aspects of the projects – when initially implemented with the user interface – caused responsiveness issues, and as such had to be reworked.

The first of the issues arose when multiple ImageView components were being displayed on the screen at one time. ImageViews initially had their data set using the *.setImageUri()* method, supplying the URI of the respective image to be loaded. However, this caused heavy memory usage when using multiple ImageViews – causing the application to crash due to running out of memory – and caused a delay of around 12 seconds to the response of touch in the application. By utilising the Glide [(11)] library to load images into ImageView componenets responsiveness was drastically improved.

Further responsiveness issues within the application – although far less impactful than the issues caused by the multiple ImageViews – were caused by reading and writing to the database, as these operations were carried out on the same thread as the UI. These issues were addressed when reconfiguration of the database was carried out (see **4.3.3.2**).

4.3.3.2 Database reconfiguration

The introduction of the user interface allowed a user of the application to give a spot a title and description. Due to the database being set up to store an image URI and location metadata only, alterations in the form of two further columns – a column for the title of the spot and a column for the description – were carried out on the database table. Additionally, as reading and writing to the database caused issues with application responsiveness, the mode of access with regards to the database was altered too.

Two classes were created – DbRead and DbWrite – to provide a way for an activity to interact with the database. Both classes extend AsyncTask [(12)], enabling the tasks to be carried out by the class to be done in a background thread, updating the UI upon finishing. DbWrite is called with the parameters of the current activity, the spot to be inserted in the database, and the database helper, adding the new spot to the database and refreshing the list displayed on the main screen. DbRead acts in a similar way, performing an SQL query on the database in the background and parsing the information into an ArrayList. After the retrieval of all relevant records, the ArrayList is returned to the main activity and the information is represented in the ListFragment.

4.4 Implementing Additional Features

With the UI successfully implemented, a small number of additional features were required to complete the functionality of the application.

4.4.1 Camera Implementation

Similarly to choosing an image from the device's storage, the Android framework has a predetermined Intent for taking a photograph using the device's camera – namely **ACTION_IMAGE_CAPTURE**. The URI of the taken photograph is stored in the data of the Intent, allowing for the URI to be read in the main activity and used to create a new dialog for creating a spot. As an image taken this way may not always have location metadata attached, the current location is used and written to the database as the location data for the spot.

4.4.2 Directions

Due to directions being between two different latitude and longitude values, Google Maps can be utilised to get directions via various modes of transport. The multiple ways in which to launch Google Maps from within an application with specified location data are documented within the Google Maps API documentation [13], however, all methods described require Google Maps to be installed on the device running the application.

By implementing the methods for creating the Intent described in the documentation, yet creating a URL for the Google Maps web application to find directions between the two points and using that as the URI, the directions can be opened in either Google Maps or a web browser. When the Intent for directions is started, it is initially restricted to opening in Google Maps only. If Google Maps is not installed on the device, directions to the selected spot from the current location will be opened in the device's web browser – giving the user the option to select a web browser if more than one is discovered.

4.4.3 Toasts

Toasts [14] are utilised by the Android framework to display a quick message for the user in a small popup. By implementing Toasts users can be notified of events occurring in the background, or warned of an error or issue. With regards to the application Toasts are used to remind the user to enter information about a spot before uploading it if a spot is attempted to be uploaded with missing information, to warn the user if the title or description of a spot is too long when attempting to upload it, or to notify the user when data for the list is being refreshed or has been obtained.

Chapter 5 - Evaluation

This chapter is intended to evaluate the mobile application, the degree to which the requirements are met, potential enhancements and the overall result of the project.

5.1 Meeting the Requirements

As outlined in section **1.3**, five key requirements for the application were identified:

1. **The application must be able to display spots effectively**. By utilising a ListFragment, spots are displayed with their image, title, and distance from current location in a list on the main screen. Selecting a spot will provide the user with a dialog also containing the description of the spot
2. **The application must be able to use a previously taken photograph with a geotag or take a new photograph when creating a spot**. Spots can be created with either an image selected from the device's storage – provided the image has geotagged data – or by taking a photograph through the application
3. **Directions need to be able to be given from the current location to a selected spot**. The dialog displayed after selecting a spot from the list allows the user to get directions to the spot via Google Maps, either through the Google Maps application or through the device's web browser
4. **Data about spots must be able to be written to and read from the database efficiently.** The application utilises multithreading for database queries, helping to ensure efficiency and that dealing with the database doesn't affect the UI thread
5. **The application must be responsive and simple**. Through utilisation of the Glide library and multithreading with regards to database queries, the application stays responsive. The application was designed with simplicity in mind, resulting in an uncluttered UI displaying the needed information

The mobile application created fulfils all requirements outlined to an acceptable degree, implementing the functionality required to achieve each objective.

5.2 Areas of Note

The implementation of multithreading was crucial to the success of the project. Both the Glide library used and the implementation of the database utilise AsyncTasks to make use of multithreading, allowing computations to occur in the background as opposed to on the UI thread. Without the use of multithreading, the use of ImageViews brought the responsiveness of the app almost to a complete standstill, and database queries resulted in a noticeable slowing of responsiveness of the user interface.

5.3 Areas for Further Enhancement

While the application successfully fulfils the identified requirements, there are many areas that could benefit from further enhancement. The database implemented is a local database stored on the device, and given the small size of sample data querying the database results in all rows in the table being returned.

Further enhancement could result in implementations allowing connection to an online database, enabling users to query results from and write to one large shared database. Given the way database queries are implemented in the project, extending the functionality to allow for connection to an online database would not require much alteration. Furthermore, image URIs can be replaced with image URLs, at which the image is hosted, allowing for all written functionality to remain intact. Additionally, if the database contained a significant amount of data, the SQL queries to retrieve rows can easily be altered to ensure only data for spots within a specified range – such as up to 5 kilometres away – is obtained by way of a simple WHERE clause.

In the application's current state, the database is queried and data is retrieved upon every opening of the app. Implementation of a cache would cut down on the number of queries required, as queried spot data can be written to memory and information only obtained from the database when the time since the last query has passed a certain limit – 24 hours, for example – or when a user manually refreshes their list. If the functionality for connections to an online database is implemented, a cache will be necessary, as constant querying of the database may result in significant data usage.

Scraping tools can be written for social media services, searching for geotagged images and videos related to skateboarding. By parsing the location metadata from these sources and collating the results potential skate spots can be discovered, dependent on the concentration of location data for particular areas.

Given that the application's database functions on a purely local level, uploaded spots are instantly written to the database. If connected to an online database, a system or systems must be put in place to check the images, titles, and descriptions of spots for offensive material before uploading. Implementation for checking the title and description of a spot for offensive material could be something simple such as ensuring the Strings representing each of these fields don't contain any key offensive terms, but an automated system to ensure all pictures uploaded are only of skate spots is practically impossible. As a system would need to be implemented, all potential spots could be uploaded to a secondary database, where they await inspection by a person. If any aspect of the uploaded spot is determined to be offensive the data is deleted, otherwise the spot data is sent to the main database, and is added to the list of spots.

While not providing any further functionality, the design of the app is also an area beneficial from further enhancement. Due to the focus on the functionality of the application, many aspects are using default Android styles and designs. The application would greatly benefit from a style being designed, with the creation of custom resources to be used as icons for example.

5.4 Assessment of Project as a Whole

If assessment of the project is to be considered by fulfilment of the identified requirements the project can be considered a success (see **5.1**). However, when considering the project aim (see **1.1**), the project can be considered only partially successful. In the project's current state an online framework could not be directly implemented, instead requiring alterations to be made to the application (see **5.3**). The application works with the same principles and in the same manner as an online variant would do, supporting the idea that an online map of skateparks and skate spots is feasible. It is important to note that the data with which this map would be built would be submitted

by users, and as such would require significant user interaction to generate an adequate amount of data. Despite all members of the target audience interviewed expressing interest and being in support of the project, a significant number of users populating the database cannot be guaranteed.

References

1. drewnoakes/metadata-extractor. *GitHub*. 2017. Available at: https://github.com/drewnoakes/metadata-extractor.

2. Yanai K. World Seer: A Realtime Geo-Tweet Photo Mapping System. *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval - ICMR '12*. 2012. doi:10.1145/2324796.2324870.

3. Saving Data in SQL Databases | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/training/basics/data-storage/databases.html.

4. ListFragment | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/reference/android/app/ListFragment.html.

5. Intent | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/reference/android/content/Intent.html.

6. Uri | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/reference/android/net/Uri.html.

7. ArrayAdapter | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/reference/android/widget/ArrayAdapter.html.

8. MenuInflater | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/reference/android/view/MenuInflater.html.

9. AlertDialog.Builder | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/reference/android/app/AlertDialog.Builder.html.

10. FragmentManager | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/reference/android/app/FragmentManager.html.

11. bumptech/glide. *GitHub*. Available at: https://github.com/bumptech/glide.

12. AsyncTask | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/reference/android/os/AsyncTask.html.

13. Google Maps Intents | Google Maps Android API | Google Developers. *Google Developers*. Available at: https://developers.google.com/maps/documentation/android-api/intents.

14. Toast | Android Developers. *Developerandroidcom*. Available at: https://developer.android.com/reference/android/widget/Toast.html.

15. IOC approves five new sports for Olympic Games Tokyo 2020. *International Olympic Committee*. 2016. Available at: https://www.olympic.org/news/ioc-approves-five-new-sports-for-olympic-games-tokyo

16. Brisick J. *Have Board, Will Travel: The Definitive History Of Surf, Skate, And Snow*. 1st ed. New York: HarperEntertainment; 2004.

Documentation

The application is included as a signed .apk on the attached USB drive. Please install this .apk on an Android phone and ensure location services are enabled before starting the application.

Upon loading the application, the user will be met with the main screen. A portion of this screen may be blank due to a lack of data in the database. To select an image from the device's storage to upload to the database as a spot, choose the "Select Photo" option in the toolbar (N.B. the selected image must have location data attached to it. If the image does not, you will be met with a dialog detailing this). To alternatively take a photo to upload to the database as a spot, select "Take Photo".

After an image has been taken, or an appropriate image has been selected, you will be met with a dialog showcasing the image selected and requesting a title and description be created for the skate spot. Entering information and selecting "Okay" will add the spot to the database, and you will now see it displayed in the list on the main screen.

Selecting any item on the list opens a dialog with the image, title, and description of the spot. A button is also displayed in this dialog with the text "Get Directions to Spot". Selecting this button will launch the Google Maps application (if installed on the device), or open Google Maps through the device's web browser, with directions from the current location to the location of the spot.