UNIVERSITY OF LEICESTER

Department of Informatics, University of Leicester

CO7501 Individual Project

Android Application - City Tour Planner

Author Name: Muhammad Sohaib Furqan

University email: msf7@student.le.ac.uk

University ID: msf7

Final Dissertation (Word Count: 15,587)

Supervisor: Dr. Rayna Dimitrova

Second Marker: Prof. Reiko Heckel

Submission Date: 02 September, 2019

Abstract

We live in an era where most of our daily lives are consumed by never-ending professional commitments, to the extent that people are left with next to no time to attend to their hobbies, and travelling is no different. Most people are so occupied with work that they see little value in undertaking long trips that require setting aside fixed periods of time. City Tour Planner for Android is an attempt to address this problem, the idea being to allow users to schedule single-day, single-city trips at times of their choice. This will provide value to the users in terms of flexibility – enabling them to schedule trips around their work routine as they see fit. Users will be able to retrieve the types of places they specify. The application will also keep track of Scheduled, past and deleted trips.

Declaration

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people s work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s). Any part of my own written work, or software coding, which is substantially based upon other people s work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Muhammad Sohaib Furqan

Signed:

Date: 02 September, 2019

To my family, who stood by me through thick and thin and brought the best out of me. Thank you everyone for being a great support through this journey which I once thought would be impossible. You equally share my success. Much Love!

Acknowledgements

I would like to start by expressing my thankfulness and gratitude to my worthy supervisor, Dr. Rayna Dimitrova, for her sincere guidance and friendly advice throughout the course of the project. If it hadn't been for her, my journey through the last part of my MSc degree probably won't have been as smooth. Thank you Dr. Rayna for being an ideal supervisor and showing me the way to go if I ever get the chance to be at your position.

It would also be unfair of me to not mention my second marker, Dr. Reiko Heckel, whose valuable insight and comments on my progress throughout the project were a constant source of self-reflection and improvement.

All the credit for this dissertation is also equally shared by my family. My parents, wife and sisters for pushing me on and not letting me give up. I dedicate this work to my parents, who showed me how to lead to lead an exemplary life. Their never-ending sacrifices, relentless hard work and unceasing encouragement and dedication towards my growth is the only reason for me to be in a position to write this dissertation today. This is an ideal opportunity for a special mention of my personal superhero, my dad. I could have only dreamed of achieving an MSc degree from an institute as prestigious as the University of Leicester, had it not been for his blind faith in my abilities. Thank you mum and dad for everything. I hope I made you proud (finally) and also that there's a lot more of that to come. This dissertation is my promise to you that I will always strive to make sure you can hold your head high. I can never repay you for the role you have played in bringing me to the point where I am today, but you will not regret the sleepless nights and tense days you had to endure because of me!

Table of Contents

Chapter 1: Introduction

Technology is becoming increasingly pervasive in today's world. It has changed the way people perceive things and go about their daily routine. Most things that were considered tedious and time-consuming in the past can now be done with a few keystrokes. To add to that, the tech industry itself is evolving at a lightning pace. Each day, there are new developments and milestones that were seemingly unimaginable a decade ago are achieved. The reliance on technology is now ingrained into our lives to the extent that the present order of existence would be in total disarray without it. The advent of smartphones has been the cause of yet another paradigm shift in everyday utility computing. Tasks that required being hooked up to a computer screen to achieve can now be done on the go from the comfort and portability of a personal handheld device.

It goes without saying that the boom of technology offers enormous benefits that have changed life in almost all dimensions, but this ease instilled into our lives comes at a cost – work consumes most of our daily routine and most people seldom have the time to attend to their hobbies. Over time, maintaining a balance between professional commitments and leisure time has become exceedingly difficult. It would be nice if the reliance on technology could be used to address this problem in a way that minimizes this tradeoff.

City Tour Planner attempts to utilize the value offered by smartphones to provide users with a single-day trip planning application of a city. Users are able to register with the application and login to maintain their personalized records of scheduled, past and deleted trips. The application allows users to retrieve places at their desired location which can then be added to a trip. Also, there is an option to rate trips and places. User feedback can also be collected through the application which could prove handy for future improvements. Before selecting a place to add to a trip, users also have the option to view its details. Settings have also been provided to customize user experience by allowing to specify the category of places to be retrieved. The choice can be made from a wide variety of places such as banks, hotels, airports, museums, parks and points-of-interest among others.

Aims and Objectives

Although it builds upon other similar trip-planner apps, the purpose of the project is twofold. From a value-offering perspective, a successful implementation will relieve users of having to set aside dedicated time periods for trips, but instead will be able to organize short daily trips around their work routines. With respect to demonstration of technical ability, the project will serve as a means to access software development abilities along with the capacity to execute the complete software development life-cycle within a fixed period of time.

- The following are the objectives of the project:
- Allow users to set up a profile
- Allow users to customize trip experience
- Retrieve and present relevant information from third-party sources in an appropriate manner
- Customize retrieved information according to user interests
- Provide an estimate of the cost incurred by a specific trip
- Include pictures of venues along with user reviews of the most popular spots
- Incorporate user feedback and suggestions for improving the user experience of the application

Challenges

- Get up to speed with the latest technologies that are being used to program android applications
- Research the best suited database technology for the project given the nature and requirements of the project
- Identify and explore appropriate APIs for retrieving data for the application
- Slow loading of images from the Google Places Web Service
- Maintaining user sessions and showing only relevant data for each logged in user
- Adopting appropriate best practices and techniques while working with the database for consistency

Risks and Mitigation Strategies

Risk 1: The author has some experience in Java programming but none in building native Android applications.

Mitigation strategy: To enable the design of a robust product with good user experience, it is required that knowledge of the Android SDK be gained, refreshed and updated throughout the course of the project.

Risk 2: Possibility of backward compatibility issues.

Mitigation strategy: The constantly evolving nature of the Android platform often gives rise to backward compatibility problems that can be a source of frustration and inefficient utilization of resources. It is therefore necessary to be aware of API level specific changes when developing the features of the application.

Risk 3: Constraints enforced by the APIs or other software artifacts being used.

Mitigation strategy: One of the drawbacks of using a third-party API or service for development purposes comes in the form of the limitations they bring to the table in terms of functionality being offered and the security risks associated etc. It is needed that the developer continuously researches approaches to deal with, and work around, these limitations or threats.

Risk 4: IDE-generated files a likely source of merge conflicts in the project repository.

Mitigation strategy: Various configuration files are generated and modified by an IDE during the course of the development of the project. Most of these files are not controlled directly be the developer but rather are changed by the IDE as needed. These files can induce frustrating merge conflicts which become evident only when code is cloned on a different machine. If these conflicts are too many, as is the case in most projects, steps must be taken to resolve them in a manner that the project is not disturbed. If risky, it is wiser to work on a single machine. The developer has kept multiple copies of the current state of the project in an attempt to guard against this risk.

Risk 5: Possibility of missing deliverable deadlines or compromising project functionality

Mitigation strategy: To ensure timely submission of project deliverables and accurate design of project functionality, the author has maintained close contact with the project supervisor and the

second marker at all times. This has enabled constant feedback and timely rectification where needed.

Literature Review

Similar android applications including TripIt, RoadTrippers and Sygic Travels were studied as part of the literature review for this project. The following is a brief description of the main features offered by each of these applications. The reading list section indicates resources used to learn android application development. This section concludes with a look at the ways in which project design and implementation has been influenced by this background research.

TripIt

TripIt is a travel planning application which allows for creating an itinerary for each trip. Among other features, this application allows users to forward trip confirmation emails to a dedicated email address (plans@tripit.com). It then creates an itinerary out of the forwarded emails, separately for each trip. Users also have access to a master itinerary where they can access every trip. These plans can then be added to a calendar or shared with friends. Further details can be found at [2]

RoadTrippers

This is a web and mobile based application intended to help users in planning road trips. Focusing specifically on areas in the US and Canada, users of this application can choose from over 5 million locations and can use category filters to customize preferences. The application also provides users with turn-by-turn navigation of their desired location. Further details can be found at [3].

Sygic Travels

A GPS navigation software that works along the lines of Google Maps but was the first to offer offline maps. Though Google has also started offering this feature now, there is still a huge difference. Sygic maps are far more memory efficient compared to Google and also allow for storing larger maps for offline usage. Further details can be found at [4]

Reading List

- Google Codelabs – android fundamentals

- Android developer guides
- Android material design guidelines

Outcomes of the background research

A detailed study of the aforementioned applications provided the knowledge and the background necessary for designing a robust, state-of-the-art application. The best practices pertinent in industry were also easily evident. Although details of the coding practices and design patterns could not be retrieved from the high-level study of the applications, sufficient insight was gained to identify technologies being used, or in some cases, their alternatives. Specifically, the research led to getting familiar with the use of material design guidelines, which are extensively used in all modern applications and enable the design of rich UI/UX artifacts. Since the project involves fetching data from various APIs, the study also prompted a comparison of the various options that were available in this regard and helped in making a more informed decision.

Dissertation Structure

This chapter contains seven chapters:

Chapter 1 of the dissertation will look at the motivation behind the project, its objectives and scope. It also looks at the literature survey conducted as part of the background study for the project

Chapter 2 will be directed at the foundations laid down for starting the project, including a discussion of the general approach throughout the course of the project, the software architecture, the milestones and the plans for risk management.

Chapter 3 will be devoted to detailing the requirements specifications for the project, covering high-level and detailed requirements. Non-Functional Requirements will also be included.

Chapter 4 covers the various details of the project at the technical level.

Chapter 5 provides a walkthrough of the various modules (classes, methods etc.) that constitute the project. The purpose served by each module in the broader context of the overall application will be elaborated

Chapter 6 discusses the approaches used for testing the application at length and presents the main test cases

Chapter 7 closes the dissertation with a glance at some code snippets from the application, ideas for future enhancements to improve the application.

Summary

This chapter lays down the foundation for the rest of the dissertation. It addresses the motivation behind the project and attempts to explain to the reader why it is different and useful, the aims and objectives, the challenges faced, the potential risks that could hamper the progress of the project and the appropriate mitigation strategies adopted. It also looks at the review of literature conducted prior to the start of the development phase and discusses critically how this study of existing work enabled the developer to make better design and implementation decisions. A breakdown of the structure of this dissertation is also provided.

Chapter 2: System Specification

Requirements are at the very core of any successful product, and owing to the pervasive nature of software systems and their influence on every aspect of life as we know it, it is critical that the importance of successfully documenting and handling ever-changing requirements is not overlooked. Requirements gathering is one of the first steps in the Software Development Lifecycle, and if not done properly, things tend to go wrong very quickly and can prove very hard to rectify in the later stages of development. This has obvious damaging effects on the long-term quality of a product and the value it can provide to its users. It is therefore of utmost importance that client requirements for a product be gathered with great caution.

From a software engineering standpoint, requirements are divided into four major sub-categories: Business Requirements, User Requirements, Functional Requirements and Non-Functional Requirements. The remainder of this chapter discusses each of these requirements with reference to the project. It also looks at some other design aspects for the City Tour Planner application.

Business Requirements

Software in the modern scenario inevitably finds itself linked to a business scenario in one way or the other. In the context of software engineering, business requirements document what a system should be able to do from the perspective of the end user. Collecting business requirements is quite often the first step in the software development lifecycle and involves critical assessment of the business model and domain on the part of a business analyst.

The business requirements for City Tour Planner are as defined in the following paragraph:

An android application is needed which should allow users to plan trips. For simplicity, the application should focus on generating single city trips only. The application should allow users to organize trips that fit around their daily work routine, thus enhancing flexibility and providing a rich user experience. Users should be able to choose from a list of available place types around which to build each customized trip.

The above text is an illustration of the way a business analyst might set out business requirements for the application being developed. Due to the vast scope of such requirements, there is no single correct way of stating these, and business requirements for one application may be stated in a number of different ways.

User Requirements

Where business requirement documents the needs of a business as a whole, user requirements deal with what individual stakeholders of a business want the system to be able to do. These are usually derived from business requirements.

User requirements specific to the project are as stated below:

1. User signup and login
2. Select city to show places
3. Retrieve place details and create trip
4. Display trip information in an appropriate way
5. Edit trip headers
6. Delete trips
7. Restore trips
8. Reschedule trips
9. Permanently delete trips
10. Manage trip itinerary
11. Show pictures of selected place(s)
12. Allow user to manage profile
13. Allow user to change display name
14. Allow user to change email
15. Allow user to change password
16. Allow user to permanently delete account
17. Allow user to customize trip generation criteria
18. Allow user to select type of places to retrieve
19. Show only relevant data to each user
20. Show information about application

Functional Requirements

This category of requirements is specific to software engineering and is often derived from user requirements. Functional requirements define the desired functionality from the point of view of the system. A given requirement of this category may address all or part of the system at hand. A single user requirement may correspond to multiple functional requirements.

Following are the functional requirements for the application being developed as part of this project.

FR01: User signup

FR01-01	System shall allow users to create an account
FR01-02	System shall get email from user
FR01-03	System shall get first and last name from user
FR01-04	System shall get password from user
FR01-05	System shall prompt user to confirm password
FR01-06	System shall save account details to database when user submits credentials

FR02: User login

FR02-01	System shall allow users to login using preferred authentication provider
FR02-02	System shall get email from user
FR02-03	System shall get password from user
FR02-04	System shall authenticate user when the credentials are submitted
FR02-05	System shall navigate to home screen if correct credentials are entered

FR03: Select city to show places

FR03-01	System shall allow user to enter the name of desired city
FR03-02	System shall provide autocomplete suggestions to user
FR03-03	System shall allow user to select from autocomplete suggestions or type in the city name manually

FR04: Retrieve place details and create trip

FR04-01	System shall search details of selected city
FR04-02	System shall show points of interest in selected city
FR04-03	System shall allow users to choose out of the list of retrieved places by clicking adjacent checkbox
FR04-04	System shall allow user to save place selections as a trip by clicking the save icon on toolbar
FR04-05	System shall prompt user to enter a trip name and date
FR04-06	System shall allow user to select desired date using the date-picker widget
FR04-07	System shall save trip details to database in an appropriate manner when the trip name and date are submitted by user
FR04-08	System shall assign trip status as “scheduled” if date of trip is after the current system date
FR04-09	System shall assign trip status as “past” if date of trip is before the current system date
FR04-10	System shall notify user that the new trip has been created
FR04-11	System shall update database state to reflect changes

FR05: Display trip information in an appropriate way

FR05-01	System shall check trip status of each child node from database on loading of main activity
FR05-02	System shall show trip with “scheduled” status in the “scheduled” tab on the home screen
FR05-03	System shall show trip with “past” status in the “past” tab on the home screen
FR05-04	System shall show trip with “deleted” status in the “trash” tab on the home screen
FR05-05	System shall show each newly created trip in the appropriate tab on the home screen by reading its status from the database in real time
FR05-06	System shall provide an option to perform further operations on a selected trip showing in any tab in the home screen

FR06: Edit trip headers

FR06-01	System shall allow to select additional operations for a trip by clicking the “more options” button next to the trip
FR06-02	System shall allow user to edit trip name and date for each trip showing in the “scheduled” tab by clicking the “Edit trip Details” option
FR06-03	System shall navigate to the “update trip headers” activity
FR06-04	System shall popular edittexts with the current name and date of selected trip
FR06-05	System shall allow user to input new name and date
FR06-05	System shall update the name and date of the selected trip when user clicks the update button in the database
FR06-06	System shall decide the new category of the trip according to updated date and show the trip in the appropriate tab with the new details

FR07: Delete trips

FR07-01	System shall allow to select additional operations for a trip by clicking the "more options" button next to the trip
FR07-02	System shall allow user to delete each trip showing in the "scheduled" or "past" tab by clicking the "Delete Trip" option
FR07-03	System shall prompt user for delete action confirmation
FR07-04	System shall assign a trip status of "Deleted" when the user clicks the "OK" button in the confirmation dialog
FR07-05	System shall update the application to remove deleted trip from the associated tab and display it in the "Trash" tab

FR08: Restore trips

FR08-01	System shall allow to select additional operations for a trip by clicking the "more options" button next to the trip
FR08-02	System shall allow user to restore each trip showing in the "trash" tab by clicking the "Restore trip" option
FR08-03	System shall check the date of the trip to decide the new category for the trip. If the date of the trip is before the current date, the system shall assign a trip status of "past". If the date of the trip is after the current system date, the system shall assign a trip status of "scheduled"
FR08-04	System shall display the restored trip in the appropriate tab according to the trip status assigned
FR08-05	System shall update the application to remove deleted trip from the associated tab

FR09: Reschedule trips

FR09-01	System shall allow to select additional operations for a trip by clicking the "more options" button next to the trip
FR09-02	System shall allow user to reschedule each trip showing in the "past" and "trash" tab by clicking the "Reschedule trip" option
FR09-03	System shall check the date of the trip to decide the category for the trip. If the date of the trip is before the current date, the system shall assign a trip status of "past". If the date of the trip is after the current system date, the system shall assign a trip status of "scheduled"
FR09-04	System shall display the restored trip in the appropriate tab according to the trip status assigned
FR09-05	System shall update the application to remove restored trip from the "past" or "trash" tab and show it in the "scheduled" tab

FR10: Permanently delete trips

FR10-01	System shall allow to select additional operations for a trip by clicking the "more options" button next to the trip
FR10-02	System shall allow user to permanently remove each trip showing in the "trash" tab by clicking the "Permanently Delete Trip" option
FR10-03	System shall prompt user to confirm delete operation
FR10-04	System shall remove all details of selected trip from the database when the user confirms the operation
FR10-05	System shall update the application to remove deleted trips from the tabs of the home screen

FR11: Manage trip itinerary

FR11-01	System shall open trip itinerary when user clicks on a trip in any tab on the home screen
FR11-02	System shall show details of places added to selected trip
FR11-03	System shall allow the user to specify start and end times for each place in the itinerary within a single day
FR11-04	System shall display user-specified start and end times next to each place in the trip
FR11-05	System shall provide the user with options to manage place entries for each trip

FR12: Show pictures of selected place(s)

FR12-01	System shall allow user to view pictures of places while selecting places to add to a trip
FR12-02	System shall present the user with a list of popular places
FR12-03	System shall allow user to view pictures of selected place when the user clicks on that place
FR12-04	System shall allow user to swipe to view more images
FR12-05	System shall present the user with further details regarding the selected place on the same screen

FR13: Allow user to manage profile

FR13-01	System shall allow user to manage app settings by clicking on the “settings” icon in the toolbar of the main screen or by choosing the “settings” option in the navigation drawer

FR13-02	System shall allow user to change application display name
FR13-03	System shall allow user to change email
FR13-04	System shall allow user to change password
FR13-05	System shall allow user to permanently delete account

FR14: Allow user to change display name

FR14-01	System shall allow user to manage app settings by clicking on the “settings” icon in the toolbar of the main screen or by choosing the “settings” option in the navigation drawer and choose to manage profile settings
FR14-02	System shall allow user to select the desired setting
FR14-03	System shall open a dialog box populated with the current user’s name when the user selects to edit display name
FR14-04	System shall allow user to enter a new name
FR14-05	System shall update the display name when the user clicks the OK button

FR15: Allow user to change email

FR15-01	System shall allow user to manage app settings by clicking on the “settings” icon in the toolbar of the main screen or by choosing the “settings” option in the navigation drawer and choose to manage profile settings
FR15-02	System shall allow user to select the desired setting
FR15-03	System shall open a dialog box populated with the current user’s email when the user selects to edit email address
FR15-04	System shall allow user to enter a new email
FR15-05	System shall update the email for the user when the user clicks the OK button

FR15-06	System shall send an email change confirmation email to the old email address of the user

FR16: Allow user to change password

FR17-01	System shall allow user to manage app settings by clicking on the “settings” icon in the toolbar of the main screen or by choosing the “settings” option in the navigation drawer and choose to manage profile settings
FR17-02	System shall allow user to select the desired setting
FR17-03	System shall navigate to the password change activity when the user selects to change password
FR17-04	System shall allow user to enter current password
FR17-05	System shall allow user to enter new password
FR17-06	System shall allow user to reenter new password after confirming
FR17-07	System shall check if the password entered in the “current password” field is a valid one
FR17-08	System shall check if the passwords entered in the “new password” and the “confirm new password” fields match
FR17-09	System shall update password if password is valid and the new passwords match when the user clicks the update icon on the toolbar

FR17: Allow user to permanently delete account

FR17-01	System shall allow user to manage app settings by clicking on the “settings” icon in the toolbar of the main screen or by choosing the “settings” option in the navigation drawer and choose to manage profile settings
FR17-02	System shall allow user to select the desired setting

FR17-03	System shall prompt user to enter current password when the user chooses to permanently delete account
FR17-04	System shall check if the supplied password is correct when the user clicks the OK button on the dialog box
FR17-05	System shall prompt the user for confirmation to delete account
FR17-06	System shall delete user details permanently from the database when the user confirms the action

FR18: Allow user to customize trip generation criteria

FR18-01	System shall allow user to manage app settings by clicking on the “settings” icon in the toolbar of the main screen or by choosing the “settings” option in the navigation drawer
FR18-02	System shall allow user to specify whether or not they want to retrieve custom places by using the designated toggle switch
FR18-03	System shall allow user to specify a place category if the user selects to retrieve custom places by using the listpreference
FR18-04	System shall retrieve only places of the category specified in the listpreference if custom place retrieval is enabled

FR19: Allow user to select type of places to retrieve

FR19-01	System shall allow user to manage app settings by clicking on the “settings” icon in the toolbar of the main screen or by choosing the “settings” option in the navigation drawer and choose to manage profile settings
FR19-02	System shall allow user to change trip preferences
FR19-03	System shall allow user to enable/disable custom place retrieval

FR19-04	System shall allow user to select the category of places to retrieve using a listpreference widget if custom place retrieval is enabled

FR20: Show only relevant data to each user

FR20-01	System shall display only those trips which are created by the currently logged in user
FR20-02	System shall add only those newly created trips which are added by the currently logged in user

FR21: Show information about application

FR21-01	System shall allow user to view information about the application when the user selects the “About” item in the application navigation bar
FR21-02	System shall display a dialog containing general information about the application, the developer and the project supervisor and second marker

Non-Functional Requirements

NFRs deal with the specifics of how the system being developed should operate as opposed to functional requirements, which document what the system should do. Additionally, while functional requirements are directly linked to user needs associated with the system, non-functional requirements are often not specified by the user and are automatically understood by the development team.

Following are the non-functional requirements for the application being developed as part of the project:

NFR01: Performance

NFR01-01	The application must have a startup time of less than 5 seconds after the user taps the launcher icon
NFR01-02	User actions must be responded to by the application in the least possible time, preferably less than 10 seconds
NFR01-03	The application must be able to run smoothly on Android devices with API level 27 and above

NFR02: Security

NFR02-01	The application requires users to register and login with their credentials before they can use the application. Only properly validated users must be used to access the application
NFR02-02	The application must ensure that any user does not have any level of access to other users' data. This includes not being able to view, create or modify trips created by other users
NFR02-03	Once logged in, the application retains credentials of the logged in user until they log out explicitly, in which case the system shall perform the necessary steps to ensure complete removal of the previous user session
NFR02-04	Once digitally signed and made available on the play store, the application must only be available to the user through designated app download platforms. The developer must make a conscious effort to ensure that unofficial or pirated APKs are not available over the internet on illegal sites

NFR03: Availability

NFR03-01	The application shall be available to the user from any part of the world once released on the play store. When not available, the developer may be contacted
NFR03-02	The application shall be available for use all the time unless the underlying API services are down or the developer has made the application unavailable for maintenance or upgrade

NFR04: Disaster Recovery

NFR04-01	Appropriate disaster recovery mechanisms must be available in case an unforeseen exception arises when using the application.
NFR04-02	The firebase real-time database must synchronize data across the effected client devices to ensure consistency in the event of a sudden crash

NFR05: Documentation

NFR05-01	The application must be accompanied by detailed documentation that describes how the entire software development lifecycle was executed during its production
NFR05-02	Prompt help must be available to the user. This application uses startup tutorial slides for this purpose

NFR06: Usability

NFR06-01	The application should be built with a central focus on providing the best user experience possible

NFR06-02	The application should be easy to understand and use for a new user

Assumptions and Constraints

Development Languages and Tools

It is assumed that the end user of the application has no direct interest in the programming language that was used to build it and the approach that was taken to do so. As in most applications, users are only concerned mainly with the value it provides to them.

Operating System

The application will be built for the Android platform and will run conveniently on devices with API level 27 or higher. Where possible, backward compatibility will be ensured.

Browser Support

This is a native android application, and does not require support from web browsers external to the application to run.

Summary

This chapter pens down the various constructs used in software engineering in order to specify the behavior of a system. The requirements for the application being developed as part of the project are highlighted. These include business requirements, user requirements, functional requirements and non-functional requirements. The assumptions made during the development of the application and the constraints applicable (if any) are also documented.

Chapter 3: Planning and Design

This chapter looks at the planning done to ensure successful completion of the project. It also discusses the approaches of implementation and design adopted throughout the project lifecycle. The chapter concludes with a thorough walkthrough of the architecture of the system and the ideas formulated to work with the underlying database technology.

Planning

Methodology

Work on this project is based on agile approaches in software development. This ensures that the end product is of a satisfactory quality and also that any arrangements can be accommodated along the way if necessary. In addition, best practices prevalent in industry are reinforced through application of these organization and planning techniques to the project.

The scrum framework is adopted for the most part of this project. The methodology consists of three roles each of which play dedicated roles within the SDLC. The roles namely are the Product Owner, Scrum Master and the Scrum Team. The Product Owner is responsible for the management of the Product Backlog, which is a prioritized set of requirements. The Scrum Master is in charge of the sprint backlog and is responsible for chairing and regularizing meetings. The scrum team implements requirements set out in each sprint.

In the context of the project, the supervisor acts as both the product owner and the scrum master because they set out the requirements for the project and also manage meetings and provide progress feedback. The scrum team comprises of myself as the developer and I am responsible for implementing requirements as discussed in each biweekly meeting. The time duration from one meeting to the next is considered a sprint for the project. At the end of one sprint, the developer discusses the progress made with the supervisor and notes down the feedback received. Any corrections or improvements are carried over to the next sprint where they are re-evaluated. Additionally, new requirements are set at each sprint (biweekly meeting), which collectively constitute the work plan for the next sprint.

Approach

The commencement of the project involved researching the technologies required for successful implementation of the project. Identification of the most appropriate technology keeping in view the nature of the project was also done during this period. This involved carefully considering the advantages and limitations of each available technology and critically evaluating which of the available choices would be best suited to the project.

After the initial assessment, a tentative project schedule was developed that listed tasks along with the estimated duration for each. This schedule is meant to act as an indication of the estimated completion dates and is not set in stone.

Development of the project started by implementing login and signup through Firebase Auth UI and setting up the main layout of the application. A trip can fall in one of the following three categories: scheduled, past and trash. The main screen consists of three tabs – one for each of the previously mentioned categories. This screen also provides a handy navigation bar that allows the user to navigate to most activities (screens) within the application.

Next, functionality was added to retrieve a list of places in a city of the user's choice from the Google places web service. The type of place to be retrieved was initially set to points of interest but later customization feature was provided while developing application settings. Once the list of places were successfully retrieved, they needed to be added to a trip. This involved deciding on the appropriate database structure and storing the necessary details for each trip in a consistent format.

A trip "itinerary" is also added to the application which allowed for setting start and end times for each place in a trip.

Settings for the application are divided into two distinct categories – profile settings allow for managing details related to user credentials. Trip settings allow for making a selection of the place type to be retrieved.

Design

Software Architecture

The application retrieves place details through communication with the Google Places web service. These places are shown in the application and the user can make selections from the available places to add to a trip. Upon saving, the user has to provide a name and a date, following which the trip is saved to the backend database. The application then retrieves saved trip and details and handles them in the appropriate manner to display the information to the user in conformance with the requirements of the application. The complete architecture is shown figure 1 below. It can be seen from the figure that communication between the application and the database and the application and the Google Places Web Service is two-way and performing using JavaScript Object Notation (JSON).

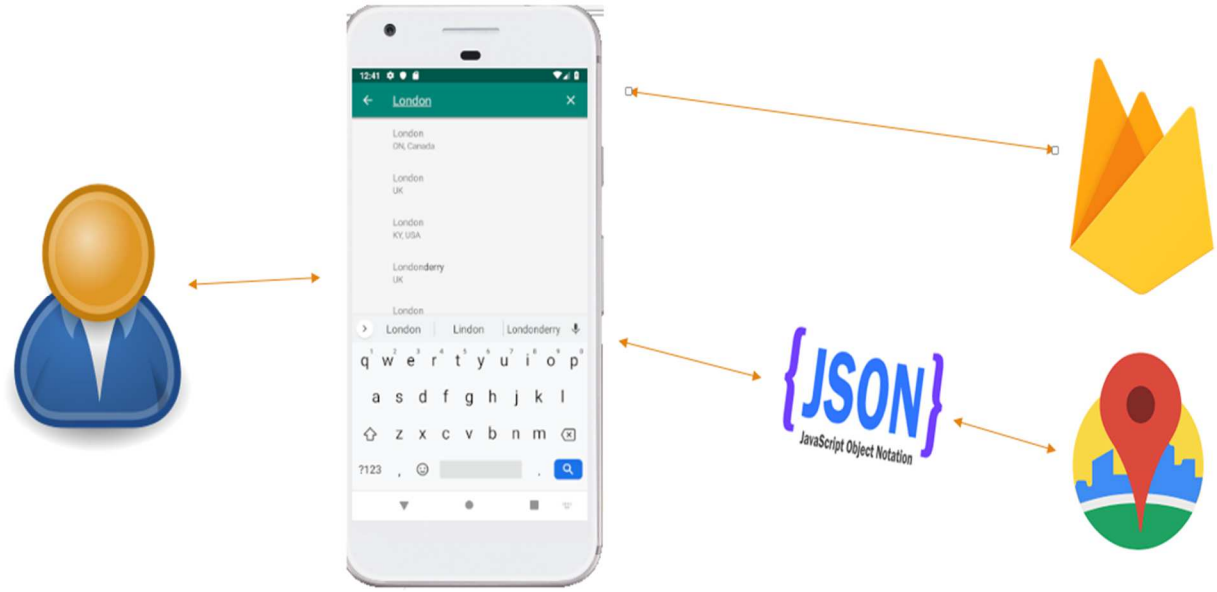

Figure 1 - Software Architecture

The application requires internet and location permissions, without which the main features will not work.

Database Design

The application makes use of the Firebase real-time database for persistent data storage. It is a NoSQL database and hence needs to be structured differently than relational databases. A typical database structure for City Tour Planner is shown in figure 2 below:

Figure 2 - Database Structure

Firebase stores data in the form of a JSON tree, where the entire database is a single JSON object. As shown in the above figure, the root node, “Trips” has two child nodes, namely “GeneralDetails” and “SelectedPlaces”. The “GeneralDetails” node stores general information regarding the trip such as the latitude, longitude and name of the city for which the trip is being created, the name given to the trip and the scheduled date. It also has a field “tripStatus” that indicates whether a given trip is scheduled, past or deleted. The “SelectedPlaces” node is on the same level as the

“GeneralDetails” node and stores information regarding the individual places added to a trip by the user, including the place name, a brief description a unique place ID.

It is notable that each child node of “GeneralDetails” as well as “SelectedPlaces” is a random sequence of characters (consider -Li9_96Yg6okLZUERAN1 in the example above). This random sequence of characters is a Push ID (a unique ID generated automatically by the Firebase real-time database whenever new data is inserted). This helps in making sure that no two entries in the database are the same. It is also worth noticing that entries with the same Push ID can be found under both “GeneralDetails” and “SelectedPlaces”. It is through the same Push IDs that the application identifies details belonging to a single trip. In the above example, the “GeneralDetails” for -Li9_96Yg6okLZUERAN1 correspond to “SelectedPlaces” entries for the same Push ID.

It would seem logical (at least superficially) if all the details of a particular trip had been stored under a single child node at the same level, but since Firebase guidelines suggest that excessive nesting of data be avoided, different nodes were created for the classification. Apart from being able to avoid nested levels of data, this approach is advantageous in that only the relevant data is downloaded .i.e. if the general details of a trip are to be shown, the data for selected places will not be downloaded. Similarly, if only place details for a particular trip are required, data about the city latitude and longitude itself will not be downloaded. Of course, data can be downloaded from both nodes if needed by the application logic.

Summary

This chapter elaborates on the planning and design strategies and approaches employed over the course of the project. It provides a biweekly task schedule, discusses the overall software architecture and the rationale behind designing the database.

Chapter 4: Technical Details

Thinking about and identifying the best-suited technologies for a project can act as a stimulus in starting development effort with the right direction. In the following text, I discuss the various technical decisions taken during the course of the project and the rationale behind them.

Architecture Overview

The application is intended for use on the Android OS and will run on almost all android-powered devices. The target audience is any user interested in automated trip-planning. The aim of the application is for users to be able to quickly and efficiently find and customize trip experiences to their liking. Users will be able to set custom search criteria, and retrieve related information on the basis of these settings from multiple sources. Users will be asked to set preferences and these preferences will be taken into account during the search. The information will then be presented in a user-friendly way.

The user is the primary point of interaction for the application. The user may perform operations on the application (such as specifying a city name), to which a response will be returned (such as a list of places of interest within that city). The user could then use these results to create trips. Moreover, the application will also provide appropriate options for trip management (CRUD operations) and keeping history of past trips.

The application will use third-party APIs for retrieving place information. At the time of writing, the Google Places API web service and the Google Places SDK for Android is being used in conjunction to provide the necessary user experience, however, this is likely to be enhanced in future.

The project uses the Firebase database for persistent data storage, the merits of which have been laid out in significant detail in the following discussions. It is worth noting that, due to the nature of the project, changes and revisions are likely to be made throughout the course of project completion.

Technologies

Although, the downsides of a particular technology depend largely on the nature of the project being developed, making the correct decision is crucial to timely completion of the project. An informed choice of the technologies to use can also greatly reduce the development effort involved.

Platform and programming languages

This application targets the android platform and therefore, a significant part of it will be developed in Java using the Java Platform Standard Edition (Java SE). The database technology for the purposes of this application was initially decided to be SQLite and Object-Relational Mapping (ORM) systems such as ROOM might but as implementation progressed it become clear that the firebase real-time database will be a much more viable option. Hence the switch to that was made.

Database selection

There are a few obvious advantages that Firebase offered over SQLite in the context of this project. While SQLite happens to be the traditional, and still the most popular way of persistent data storage within android applications, it comes with a fair share of downsides. One striking limitation of SQLite which prompted use of firebase for this project is that synching SQLite data across devices can prove to be difficult. A SQLite database operates on the physical device running the application. As a consequence, data that is stored is internal to each device and cannot be easily transferred from one device to the other. This can give rise to user dissatisfaction as changes made on one device, will most likely not reflect when the same user logs in to the same application from a different device. Firebase uses cloud storage, and hence makes the process automatic with instantaneous synchronization across devices.

API selection

The application makes use of APIs to fetch the necessary data regarding places. This includes (but is not limited to) Google Maps, Trip Advisor and Foursquare. These technologies will be used as deemed appropriate at the time of development so as to enhance the user experience of the application. At the time of writing, the Google Places API is being used to fetch points of interest in a city selected by the user.

There were some choices available with regards to available APIs. One of the apps studied during the literature review, Sygic Travels, used the Sygic Travel API in their app, but for the purposes

of this project, Google's offering was preferred because it has a richer places database and is generally more popular.

The following is a comparison of the various APIs as noted during the literature review. Further details can be found at [5]

Place API Comparison

Feature	Google	Twitter	Yahoo	Foursquare	Factual	Facebook
Service Name	Places	Places	GeoPlanet	Venues	Global Places	GraphLocation
First Released	Nov. 2010	Jun. 2010	May 2009	Nov. 2009		Aug. 2010
Identifier	Place ID	GEO ID	WOE ID	Venue ID	Factual ID	Numeric ID
Data Size	95 millions data from 70 countries		6 millions from 150 countries		65 millions data from 50 countries	
Data Structure	JSON, XML	JSON	JSON, XML	JSON, JSONP	JSON	JSON
Checkin	No	No	No	Yes		No
Create Place	No	No	No	Yes		No
Geo Coding	Yes	Yes	Yes	No		No
Reverse Geo	Yes				Yes	
Nearby Data	Yes (Mandatory)	Yes	Yes	Yes (Mandatory)		Yes (Mandatory)
Text Search	Yes	No	Yes	Yes		Yes
Popular Place	No	Yes	No	Yes	No	No
Autocomplete	Yes	No	No	Yes	No	Yes
More About API	https://developers.google.com/places	https://dev.twitter.com/docs/platform-objects/places	http://developer.yahoo.com/geo/geoplanet/	https://developer.foursquare.com/	http://www.factual.com/products#location-data	https://developers.facebook.com/docs/reference/android/3.0/interface/GraphLocation/

Figure 3 - Places API comparison

Libraries and packages

The application makes use of various libraries to perform otherwise tedious tasks. These are detailed below:

Android dependencies

1. Android support library: implementation 'com.android.support:support-v4:28.0.0'
2. Android support compatibility library: implementation 'com.android.support:appcompat-v7:28.0.0'
3. Constraint Layout: implementation 'com.android.support.constraint:constraint-layout:1.1.3'
4. Recyclerview: implementation 'com.android.support:recyclerview-v7:28.0.0'

Firebase dependencies

1. Firebase core: implementation 'com.google.firebase:firebase-core:16.0.8'

2. Firebase database: implementation 'com.google.firebase:firebase-database:16.1.0'
3. Firebase auth-UI: implementation 'com.firebaseui:firebase-ui-auth:4.3.1'

Third-party libraries and dependencies

1. Ted permissions: This library offers easy permissions handling for Android Marshmallow and above versions. It is available at the Ted Permissions Github page [6]
2. Volley: An HTTP library that simplifies networking tasks for android applications and makes it faster. For this project, the library is being used to parse JSON data from APIs so that it can be displayed locally within the application. It is available at the official Github page for Volley library [7]
3. Picasso: It is a common task for android applications to load images from online sources. Picasso is an image loader and caching library for android that greatly simplifies this task. It is available at the official Github page for Picasso library [8]

Other specifications:

The application will run on any device running Android 4.0.3 (IceCreamSandwich) or higher.

Summary

The "Technical Details" chapter of the dissertation details the application architecture, the technologies involved, the database(s) selected and the third-party APIs and libraries utilized. Where applicable, pros and cons of technologies have also been discussed in the context of the project and care has been taken to bring to the attention of the reader the thought-process behind making certain decision at the expense of others.

Chapter 5: Implementation

The application code follows an Object Oriented paradigm and is based on the use programming constructs such as classes and interfaces among others. Additionally, many constructs specific to Android programming are also used. In this chapter, we look at the classes, interfaces, layout files etc. designed in the app. The methods contained therein are explained to aid clarity. An effort has also been made to provide a comprehensive walkthrough of the development of the application.

Classes

AboutDialog

Produces a dialog box that displays information about the application when the user clicks on the "About" menu item in the navigation bar on the home screen.

EditTripHeaders

This class is responsible for the feature that allows users to change trip name and date. This is accomplished by using a TextWatcher object that collects text in variables everytime it is changed by the user. The UpdateTripHeaderDetails method uses the text collected by the TextWatcher object, stores them in a HashMap object and updates the database to reflect the modified values.

FeedbackDialog

Opens an InputDialog allowing users to send their feedback about the application by entering name, email and feedback message. This feedback is saved to the database.

GeneratedTrip

A Plain Old Java Object (POJO) class used to store and retrieve trip objects to and from the firebase real-time database.

LoginActivity

Implements firebase signup and login functionality using the pre-built Auth UI.

MainActivity

Checks application permissions, implements methods related to the navigation drawer.

PastFragment

Handles fragment display for showing past trips in the TabLayout on the home screen.

PastTripRecyclerViewAdapter

Contains code for appropriately processing trips that are shown in the “past” tab, including options to restore and delete trips, and checks on when to add a trip to the “past” tab.

PlaceDetailActivity

Handles the task of viewing more details of a selected place.

SavePlaceDialog

Opens an InputDialog that allows user to save place selections as a trip to the database after prompting the user to enter a name and date for the newly generated trip.

ScheduledFragment

Handles fragment display for showing scheduled trips in the TabLayout on the home screen.

ScheduledTripRecyclerViewAdapter

Contains code for appropriately processing trips that are shown in the “scheduled” tab, including options to edit trip header details and delete them, and checks on which trip qualifies as a scheduled one.

SearchedPlacesAdapter

Responsible for displaying information regarding places retrieved via the Google Places web service.

PlaceSearchActivity

Retrieves place details using Google Places Web service. This class contains methods to format nearby place search request URLs which can then be used to retrieve place data. The nearby place search request returns a JSON response which is handled in the application using Volley.

SearchedPlacesItem

A Plain Old Java Object (POJO) class used to define how a place searched by the user will look like in the application.

SettingsActivity

Checks whether the settings fragment being opened is a valid fragment or not. The Android platform requires application settings to be implemented in this way.

SettingsFragment

Contains code to handle the different settings offered by the application.

TrashFragment

Handles fragment display for showing deleted trips in the TabLayout on the home screen.

TrashTripRecyclerViewAdapter

Contains code for appropriately processing trips that are shown in the "trash" tab, including options to restore and permanently delete trips, and checks on when to add a trip to the "trash" tab.

UpdatePasswordActivity

Re-authenticates and updates the password for the current user.

Interfaces

FeedbackDialogListener

An interface declared in "FeedbackDialog" which passes values between the dialog and the calling class. The "MainActivity" class implements this interface.

SaveTripDialogListener

An interface declared in "SavePlaceDialog" which passes values between the dialog and the calling class. The "PlaceSearchActivity" class implements this interface.

OnItemClickListener

An interface declared in "SearchedPlacesAdapter" which defines methods for handling item clicks. The "PlaceSearchActivity" class implements this interface.

Layout files

Each Java class mentioned above is has an associated layout file (.xml) that defines the user interface for that activity.

Application Development

User authentication – signup and login

City Tour Planner is a trip planning application and therefore individual user accounts for all users play an important part in providing a personalized user experience. Authentication for City Tour Planner is implemented using FirebaseUI, a library provided by Firebase that provides ready to use authentication functionality which just needs to be plugged in to the application. This approach enabled to save significant developer time and effort since it eliminated the need to reinvent the wheel. Functionalities that are an integral part of most modern applications which would take significant work if implemented manually are taken care of by default. Examples of such features include:

- The ability to incorporate multiple authentication providers (City Tour Planner allows users to love in by Email and by Google)
- Automatic and simplified handling of user account management tasks including account creation, password reset, email notifications etc.
- FirebaseUI can be customized to match the application being developed

A noticeable drawback of using FirebaseUI to implement authentication features is that the developer has little control over the appearance of the authentication screens. The FirebaseUI has a pre-defined user-interface to go along with the functionality, which enforces constraints over the extent to which its look-and-feel can be customized.

If a user opts to sign-in by email, they are prompted to enter the email address. FirebaseUI then automatically checks to see if the email that has been input has an existing account associated with it or not. If an account already exists, the authentication library saves developer time and effort by identifying that the corresponding user is not a new one and requires a login password. If an account using the supplied email address is not found, the library detects that a new user is trying to sign-up, and asks the user to provide values for the first and the last names and an account password. In such a scenario, when the user enters the same email, they will automatically be taken to the login screen.

If the user opts to login via Google, the application opens Google's login screen where the user can authenticate themselves using those credentials. In either case, once successfully logged in,

the credentials of the user are automatically remembered by the authentication library so that the user doesn't have to repetitively login every time they use the application. Login details are stored as long as the user doesn't explicitly sign-out.

Generating a new trip

Once logged in, the user is taken to the home screen which makes use of a TabLayout with three tabs, namely "Scheduled", "Past" and "Trash". The user can create a new trip by clicking on the floating action button at the bottom right end of this screen. Doing so takes the user to the autocomplete screen.

The application uses the Places SDK for Android and the Google Places web service together. The Autocomplete feature is implemented using the SDK for Android, which presents the user with an EditText where they can search for places. As the user types, the application presents the user with suggestions of the places that match the text typed by the user. Keeping in view the requirements of the application, place suggestions shown to the user while generating a new trip are limited to cities only. The user can then click one of the suggestions to select a city for which places need to be retrieved.

Once a city has been selected by the user, the application makes use of the Nearby Place search capability provided by the Google Places web service API to display a list of places within the selected city. From an implementation perspective, this is achieved by generating a request URL through code in the appropriate format. The URL hence generated is used to request the Google Places API for the place information. The API returns a JSON response containing a list of places within the designated radius. This information is then parsed by the Android application through Volley, and the output is shown in a recyclerview.

The recyclerview has a checkbox next to each retrieved place which the user can mark to select the places that they want to add to a trip. After making the selections, the user clicks the save button. The application opens an input dialog, prompting the user to input a name and date for the trip being created. Upon confirming, the details of the newly created trip are saved to the firebase database. Some of these details such as the name and date of the place, and the place selections are made defined explicitly by the user when creating a new trip, while other details such as the north-east and south-west latitude longitude coordinates are computed and saved by the system

automatically. These implicitly acquired details are essential for various other features of the system to function, and will be discussed in the following text where relevant.

After the creation of the trip has successfully completed and the details have been saved to the database, the system compares the current date to the date of each trip to decide whether the trip should be classified as “scheduled” or “past”. If the date assigned to a trip is greater than the current date .i.e. the date assigned to the trip is in the future, it is classified as having a “scheduled” status. Likewise, if the date of a trip is lesser than the current date, i.e. the date assigned to the trip is in the past, it is classified as having a “past” status. The application then makes use of this status to decide the tab in which a particular trip needs to be shown. Scheduled trips are shown in the “Scheduled” tab on the main screen, whereas past trips are shown in the “Past” tab of the main screen.

Trip options and in-app navigation

Context-menu options

Each of the three trip categories of trips have different options which can be used to perform additional actions for the corresponding trip.

Trips in the “Scheduled” category have access to a context menu with three options which are as follows:

Edit trip details**:** This option allows for changing the name and the date that was initially set for a trip at the time of creation. The application navigates to another screen where a new name and date for the trip can be set by overwriting existing values. Once the update button on the toolbar is clicked, the name and date for that trip is changed in the database and reflected in the application.

Delete trip: Changes the status of the trip from “Scheduled” to “Deleted” in the database. The trip is then removed from the “scheduled” tab and is visible in the “Trash” tab on the main screen instead. Modifying the status in such a way causes the trip to be classed as “temporarily deleted” from the end-user perspective.

Assign start time: Presents a time-picker dialog which allows the user to assign a starting time to the corresponding trip. When the user confirms a selection, the starting time is stored in the database. As explained later, the assigned starting time is used to calculate the total trip duration and the ending time for the trip.

Trips in the “past” category have access to a context menu with the following option:

Delete trip: Changes the status of the trip from “Past” to “Deleted” in the database. The trip is then removed from the “past” tab and is visible in the “Trash” tab on the main screen instead. Modifying the status in such a way causes the trip to be classed as “temporarily deleted” from the end-user perspective.

Trips in the “trash” category have access to a context menu with the following option:

Permanently delete trip: Removes all details of the corresponding trip from the firebase database permanently.

Navigation

The main screen has a navigation drawer that provides easy access to major features of the application. The header of the navigation drawer displays the name and email of the currently logged-in user. The navigation drawer items include the following:

Trips: Navigates back to (or re-launches) the main screen of the application

About City Tour Planner: Opens a dialog showing information about why and by whom the application was developed

Settings: Takes the user to the settings of the application where particular aspects of the application can be personalized. This is explained in detail under the corresponding section later in this chapter

Submit Feedback: Opens a dialog prompting the user to give feedback regarding their experience with the application. The user is asked to enter their name and email along with the comments that they have. Each feedback is stored under a separate “feedback” node in the Firebase database

Share: Allows to inform others about the application by sending an email. The application uses an intent to launch the Gmail application with the sender and body of the email pre-defined in the application code

Trip places

Any trip in any of the three categories can be clicked to see the places associated with that trip. The push ID of the trip that is clicked is passed to the trip places activity through an intent. The trip id is used to get a reference to the particular trip’s places in the firebase database. A child event

listener then returns a data snapshot containing each individual place. Similarly to how it was accomplished in place search during the creation of a new trip, places selected for a trip are displayed to the user in a recyclerview.

The trip places screen provides a similar user interface to other parts of the application. Each place has an accompanying context menu which allows for permanent deletion of that place from the trip. This action is not reversible and in the event that the user wants to add the same place to the trip again, it must be done explicitly.

The toolbar of the trip places screen has three icons, each of which is associated with a distinct but related action.

Assign duration to places

The icon on the left causes the application to navigate to yet another screen where the user can set the duration for each of the places assigned to a trip. The screen has a dropdown list (spinner) which displays all of the places for that trip. The user can assign values for the number of hours and minutes they want to spend at that location. Keeping in view that City Tour Planner is geared towards allowing the user to create one day trips instead of those spanning multiple days, constraints have been applied accordingly on the allowed number of hours and minutes that the user can assign to a place. In an attempt to keep things simple, it was decided at the time of implementation that the maximum number of hours that can be spent at a place will be no more than three. From an input validation perspective, the application will report an error if the value input for hours falls outside the range of 1-3. Similarly, an error will be reported if the value that is input for minutes is outside of the range of 0-59.

If values are already assigned to a place, they will be shown in the respective edit-texts when the corresponding place is chosen from the spinner. If the duration values are to be updated, the user will only have to over-write the displayed values and confirm the action on the AlertDialog after clicking the save icon on the toolbar.

Where a duration is already assigned to a place, a delete icon will also be visible next to the save icon. This can be used to delete existing duration values for the corresponding place. If a duration has not been assigned to a place, it follows that there is nothing to delete. Therefore, in such a

scenario, the delete icon is hidden from the user so that incorrect actions can be avoided and tedious errors don't arise.

Add place to trip

The middle icon on the trip places screen allows the user to add a new place to an existing trip. The user is presented with an autocomplete widget similar to the one used for searching places in a city, but this time, instead of suggesting cities, it suggest places in that city as the user types. Simply put, if the user is adding a place to a trip generated for Leicester, they will be shown places in that city only. When the user selects a suggested place and confirms that they want to add it to the trip, the new place selection will be saved to the database and the recyclerview will be updated to reflect the addition of the new place.

Ensuring that the autocomplete widget displays only those places that belong to a given city involved passing a RectangularBounds object to the SetLocationRestriction method provided by the Places SDK for Android. This object takes two LatLng objects as arguments. The first LatLng object passed to the RectangularBounds object should contain the latitude and longitude coordinates for the south-west of the bounded area (which in our case is the corresponding city). The second LatLng object must contain latitude and longitude coordinates for the north-east of the bounded area. To aid clarity, it will be helpful to think of RectangularBounds in its literal meaning – a rectangular boundary enclosing the area the autocomplete wizard needs to be restricted to. The SDK uses the south-west and north-east latitude longitude coordinates to confine autocomplete suggestions within a specific area only.

It was mentioned earlier that the application stores some values automatically. The south-west and north-east latitude longitude coordinates is an example. These fields are retrieved through the autocomplete widget at the time of creation of the trip and stored in the database. When a new place needs to be added to an existing trip, these values are read from the database and used as appropriate to restrict autocomplete suggestions.

The rest of the working of the "add place" feature is largely similar to that of place search within a city, expect that instead of searching for all places within a city, the application requests a JSON response for only the place selected by the user. Similar to the in-city place search described earlier, a place fetch request URL is formulated in the application code. The implementation is also similar

in that the JSON response is parsed in the application using volley and used as needed after successful parsing.

Calculate trip summary

The note icon causes the trip summary dialog to be displayed. This has useful information about the trip, some of which is obvious and some of which is calculated. The dialog displays the trip name, the trip date, the assigned start time, the duration of the trip and the ending time. The duration of the total trip is calculated by adding up the durations for the individual places whereas the end time is calculated based on the assigned start time and the total duration of the trip.

To calculate the total duration for a trip, the duration values for each of the places are retrieved. The hours are added together to give an overall number of hours and the minutes are added together to give the total number of minutes. These values for total hours and total minutes are then added to the start time to give the ending time of the trip. This is accomplished by utilizing the new DateTime API in Java 8. In Android, this new API is supported only on API level 23 and above. Therefore, the facility to calculate ending time is supported only on devices that run Android Oreo and above. The user is notified of this limitation if the application is running on a device with a lower API level.

It makes sense that the trip summary dialog should only be usable when duration values have been assigned to all places in a trip (because only then will the application be able to accurately predict the total duration and the ending time). Therefore, if the user tries to invoke this dialog without making the assignment, a toast message is presented to the user prompting them to assign durations to all places and the trip summary button is made invisible.

Place details

A place item showing in any part of the application can be tapped to reveal further details about it. Doing so causes the application to navigate to the place details screen where the website URL, physical address, phone number and ratings etc. of the selected place are listed. Only the fields that are available in the JSON response sent by the Google Places API are shown. Where a field is unavailable, it is stated in the application. The place details screen also makes use of viewpager widget to implement a swipe Imageview. Images of the selected place are shown in the viewpager which can be swipped back and forth to reveal more images.

As with the other parts of the application that use the Google Places API, displaying additional details about a place is achieved by generating a details request URL in the format specified by the API. A place details request uses the place ID (which is a unique identifier assigned to each place by the API) to identify the place for which details are to be retrieved. This place ID is retrieved from the database and passed to the details activity when a particular place item displayed in the application is selected.

The JSON response yielded from the details request is parsed in the application through the Volley library and used to display details to the user as appropriate.

As mentioned before, a viewpager widget to show the images of the selected place is included. This viewpager is restricted to show five photos for any given place, which are also acquired through a place photos request to the Google Places API, which uses a "photo reference" (a unique string identifier used to identify a photo) provided when generating the URL in the required format. The application saves this customized URL to an array. When this array is passed to the viewpager adapter, it parses the images found at the URLs on each of the array's index and displays them in the viewpager widget.

Place photos for the lesser popular places might not be available in the places API to begin with, owing to which the viewpager will be left empty in such a scenario.

Application settings

Most modern mobile applications allow for certain aspects of the user experience to be personalized through settings. City Tour Planner provides application settings of two different types:

User profile settings give users the ability to manage aspects of the user accounts with which they have registered on the application

Trip preference settings give the ability to customize some aspects of the trips that are generated using the application

These are explained in greater detail as follows

User profile settings

This category of settings offer the following options to the user

- Change display name: The user can update the display name that shows on the navigation bar of the home screen when logged in
- Change email: Provides the ability to change the registered email address for the application. This will require the user to verify identity by authenticating themselves with the current password. Once the email update operation is successful, the user will have to login using the newly set email address the next time
- Change password: Allows the user to update the password they use in order to log in to the application. In order to update the password, the user must verify their identity by entering the current password. If the password change operation is successful, the user will be required to sign-in with the new credentials the next time
- Delete account permanently: Deletes a user account that is registered with the application. All data associated with the corresponding user account is also deleted. This is a permanent action and cannot be rolled back.

Trip preferences

This category of settings offer the following options to the user

- Retrieve custom places: A toggle switch allows the user to select whether they want to retrieve customized categories of places or not. If disabled, the second option in this category is disabled too
- Select a place category: This option is enabled if the toggle switch is enabled. It presents a list of place categories from which the user can select which type of places to retrieve while planning future trips. The application allows the user to select from five different types of places which include points of interest, museums, parks, hotels and art galleries. Once the user picks a place type, place searches for future searches retrieve only places of the selected type
- From an implementation perspective, this is achieved by using shared preferences API in Android. The currently selected value of the place type is stored as a shared preference. When the user searches for places in a city the next time, the type value is read from the shared preferences file and that value is used in the URL that is used to query the Google places API.

Graphical User Interface

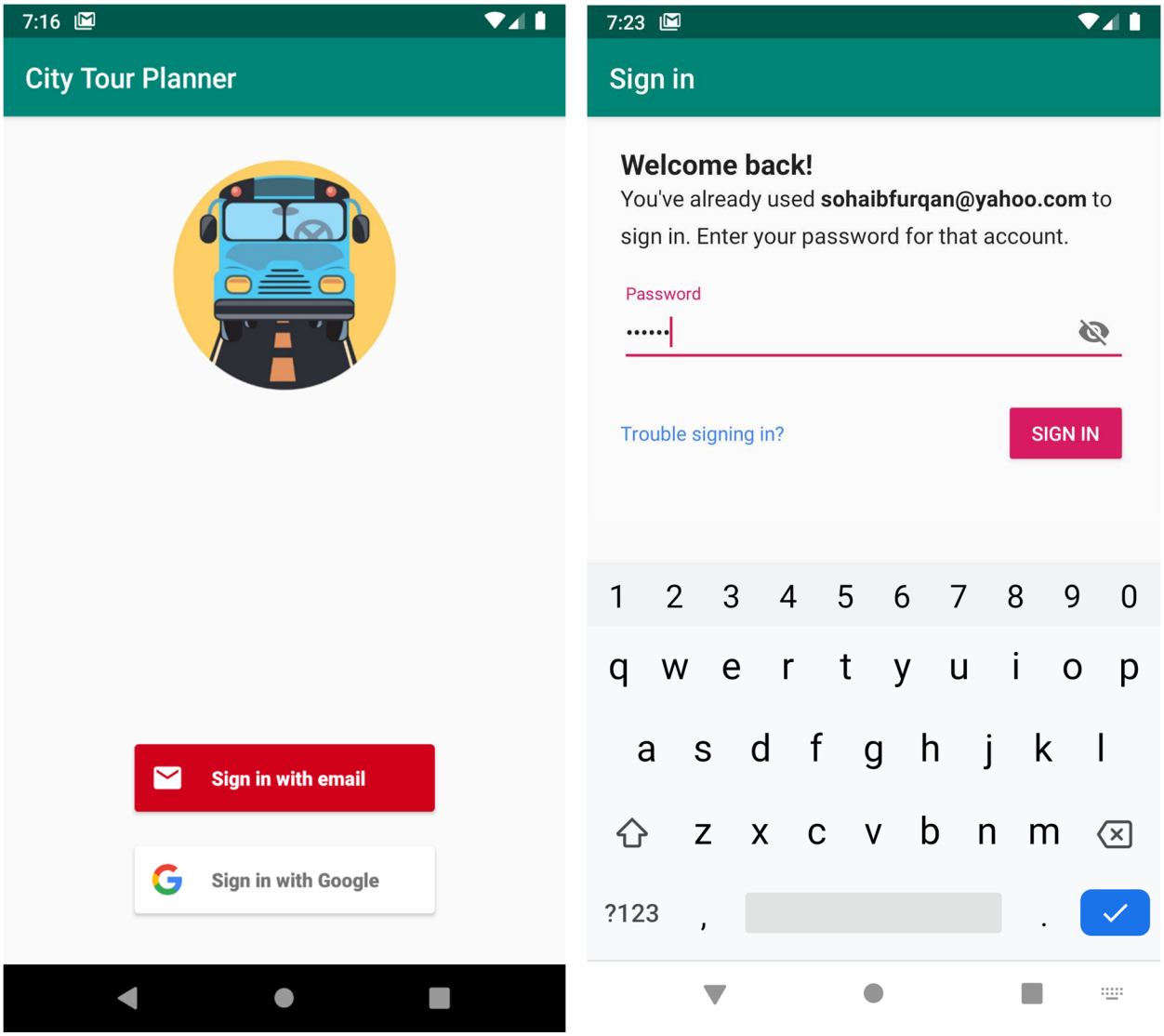

Figure 4 - Sign In flow for the email provider

The above figure shows the login screen and the sign-in flow of the application. It can be seen that the application automatically detects that the user has signed-in before, and therefore asks for the password instead of getting user account details as it would do in case of a new user account being created.

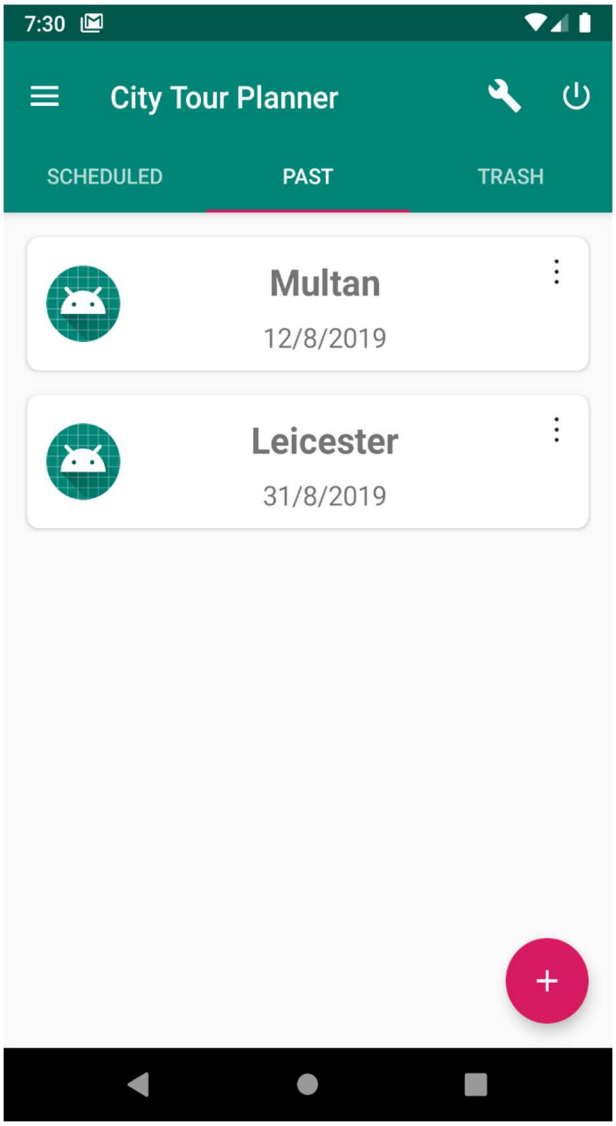

Figure 5 - main screen visible after signing in

The above figure shows the home screen as visible after a particular user has logged in. The figure shows two past trips displayed.

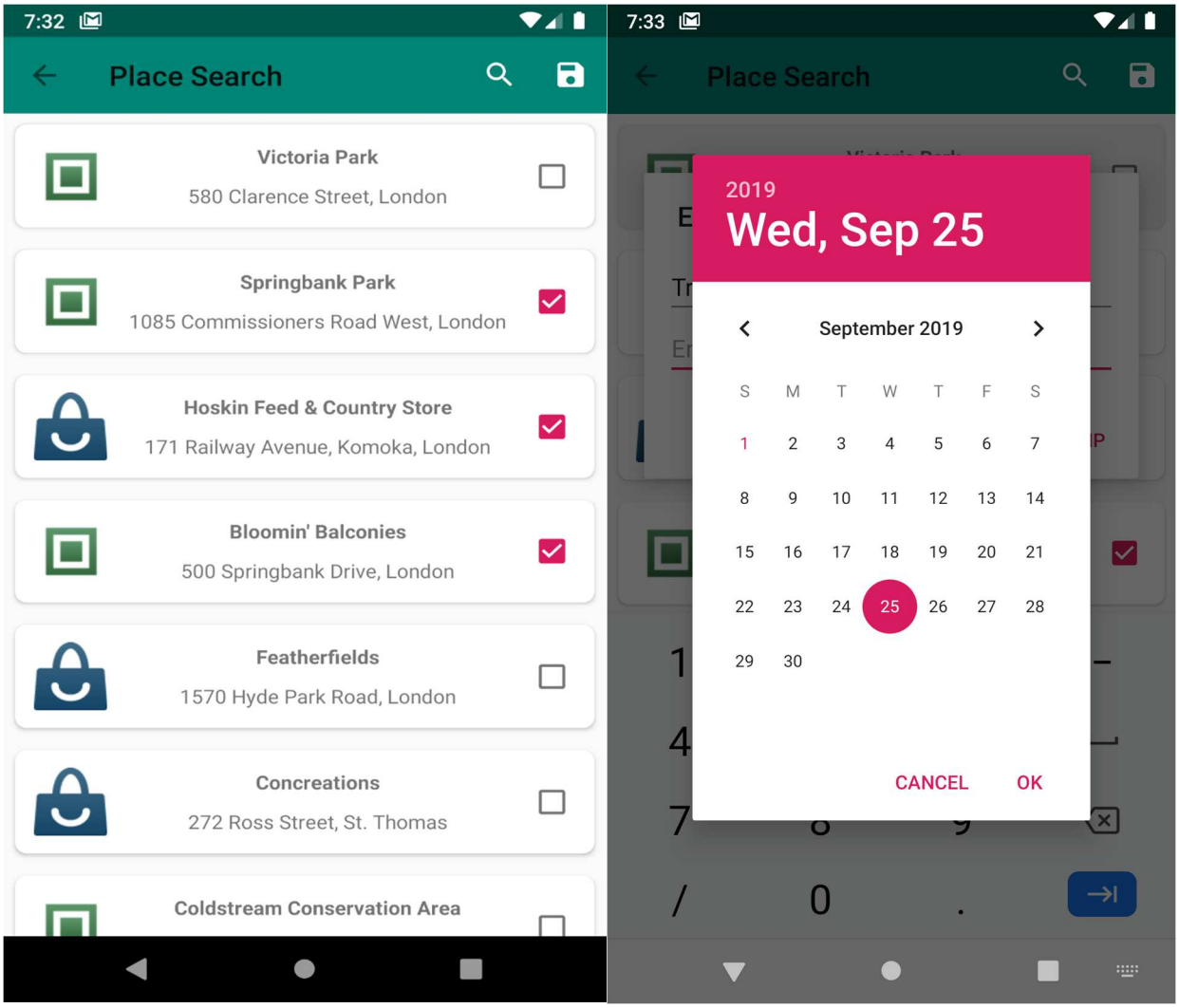

Figure 6 - Place Search and save trip

The figure above shows the list of places in London, Canada on the left and a date being selected while saving the trip on the right. Three places are being selected and a date picker widget is being used to assign the date. The date picker is launched automatically whenever the Edit-text for date gets the focus.

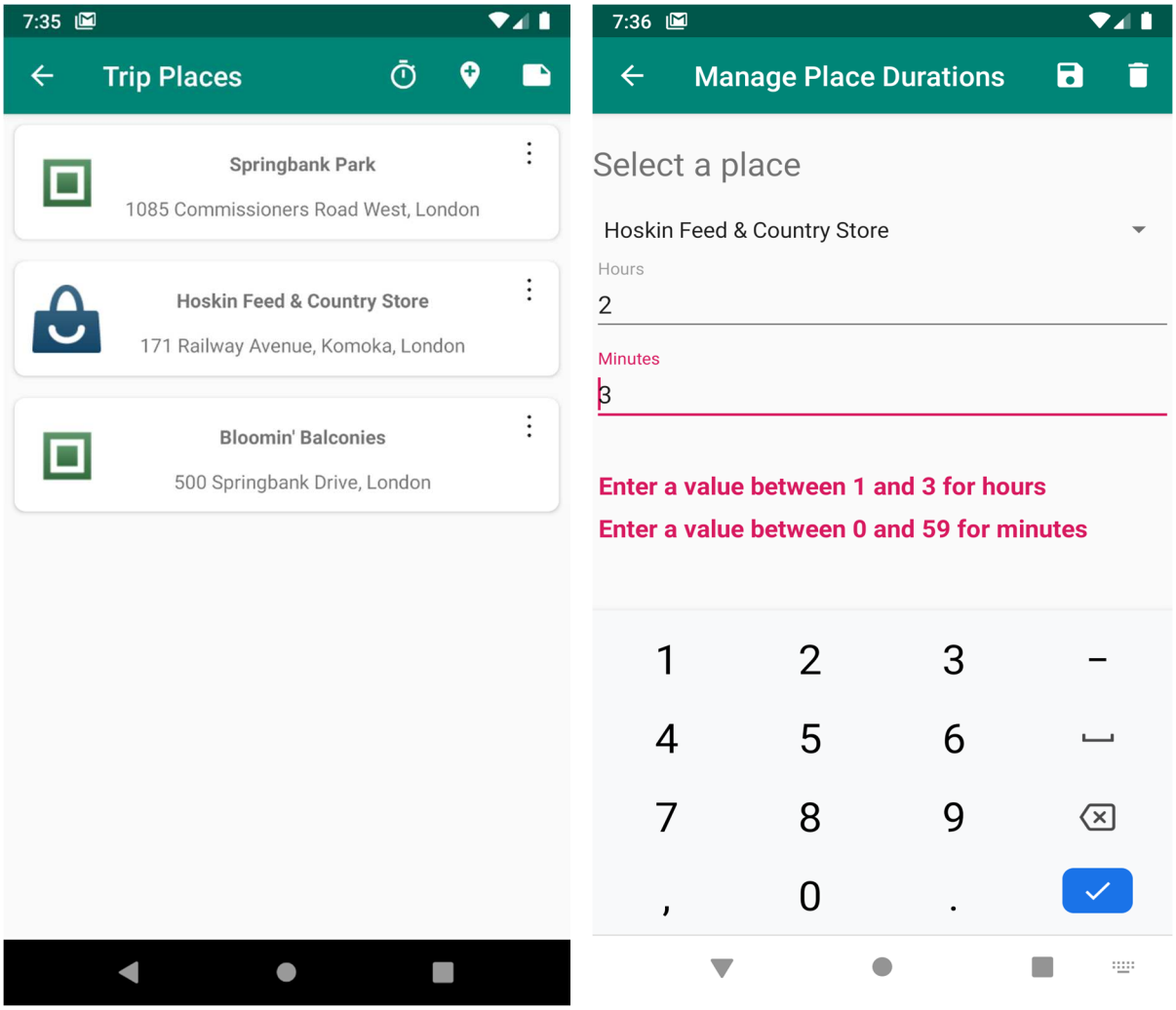

Figure 7 - Trip Places and Assign Duration

Figure 9 shows the places associated with a trip being displayed on the left. There are three icons on the toolbar each of which present the user with a different action related to places. On the right, it can be seen that duration values are assigned to one of these places. Other places may be assigned duration values using the spinner at the top. Clear instructions are provided to the user regarding the permitted range of duration values that can be assigned to a place.

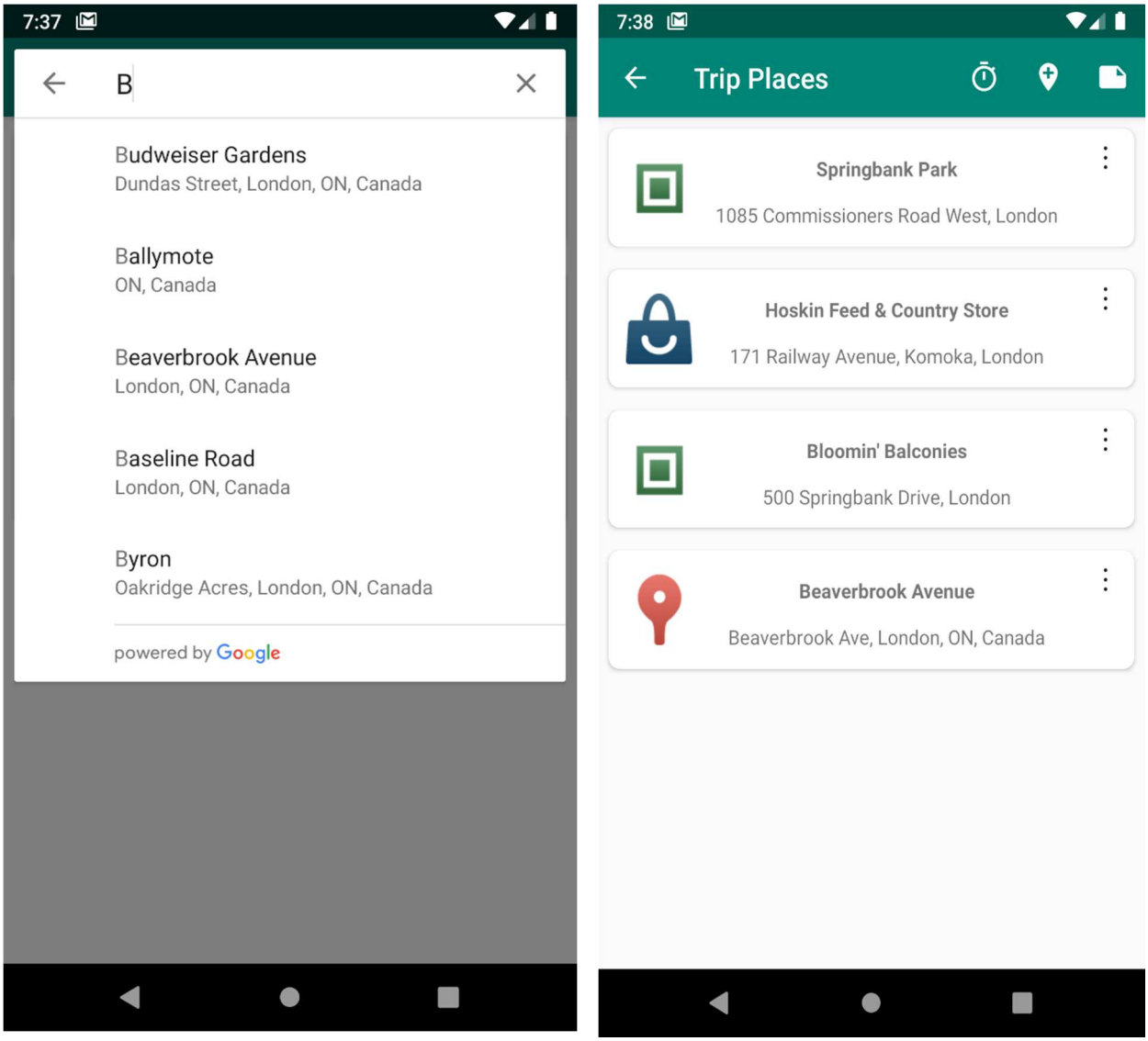

Figure 8 - Add Place to existing trip

In the above figure on the left, some place suggestions are visible through an autocomplete widget similar to the one that was used to suggest places in a city. The trip in which a new place is being added is for London, ON and it can be seen that the autocomplete widget generates suggestions accordingly. On the right, Beaverbrook Avenue has been added to the trip after the user clicked on a suggested place and confirmed the action.

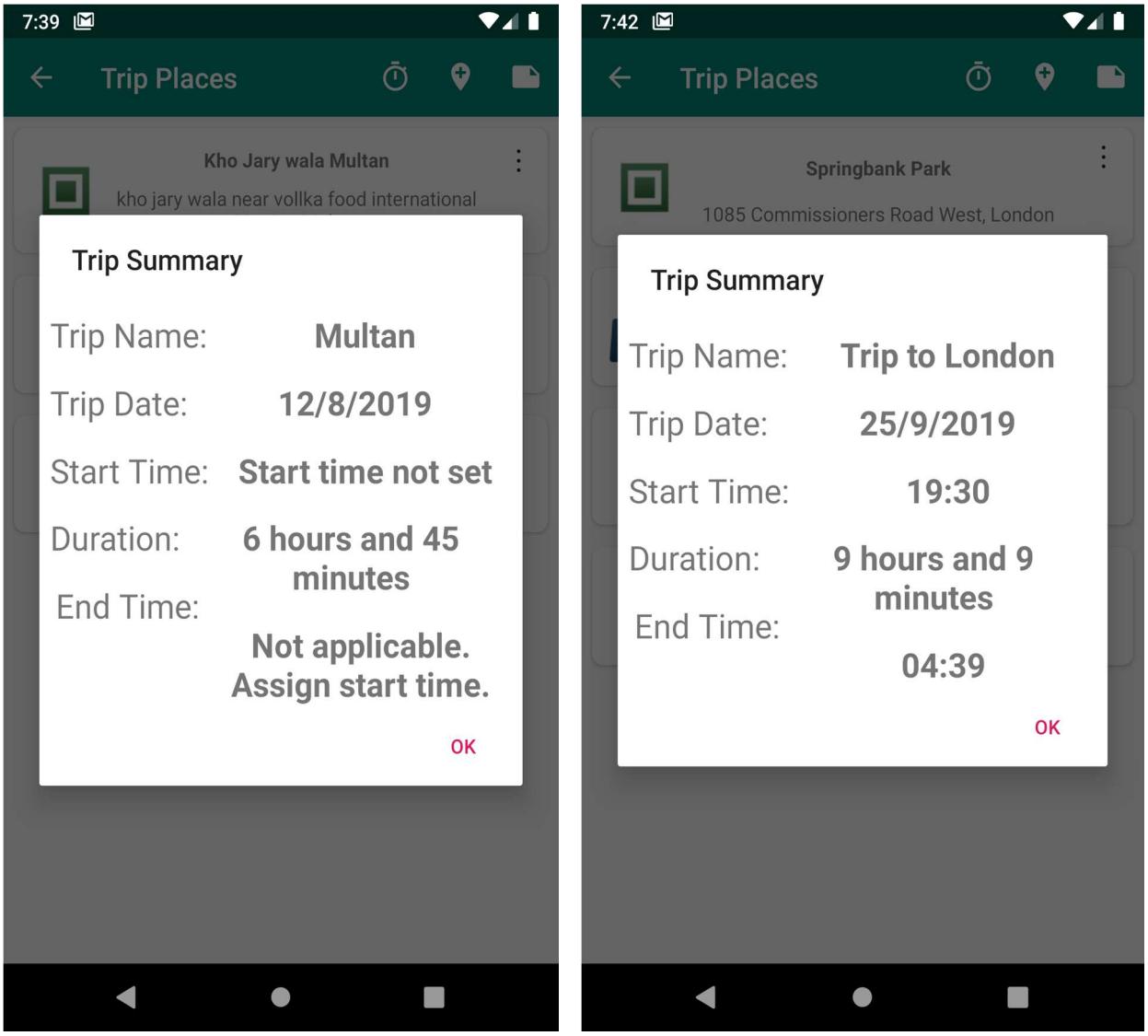

Figure 9- Calculate Trip Summary

Figure 11 depicts yet another feature associated with trip places. On the left, the summary dialog has calculated a duration for the trip but the user has been notified that the ending time of the trip cannot be calculated since the start time has not been set. On the right, the start time has been assigned and hence the end time for the trip is also calculated.

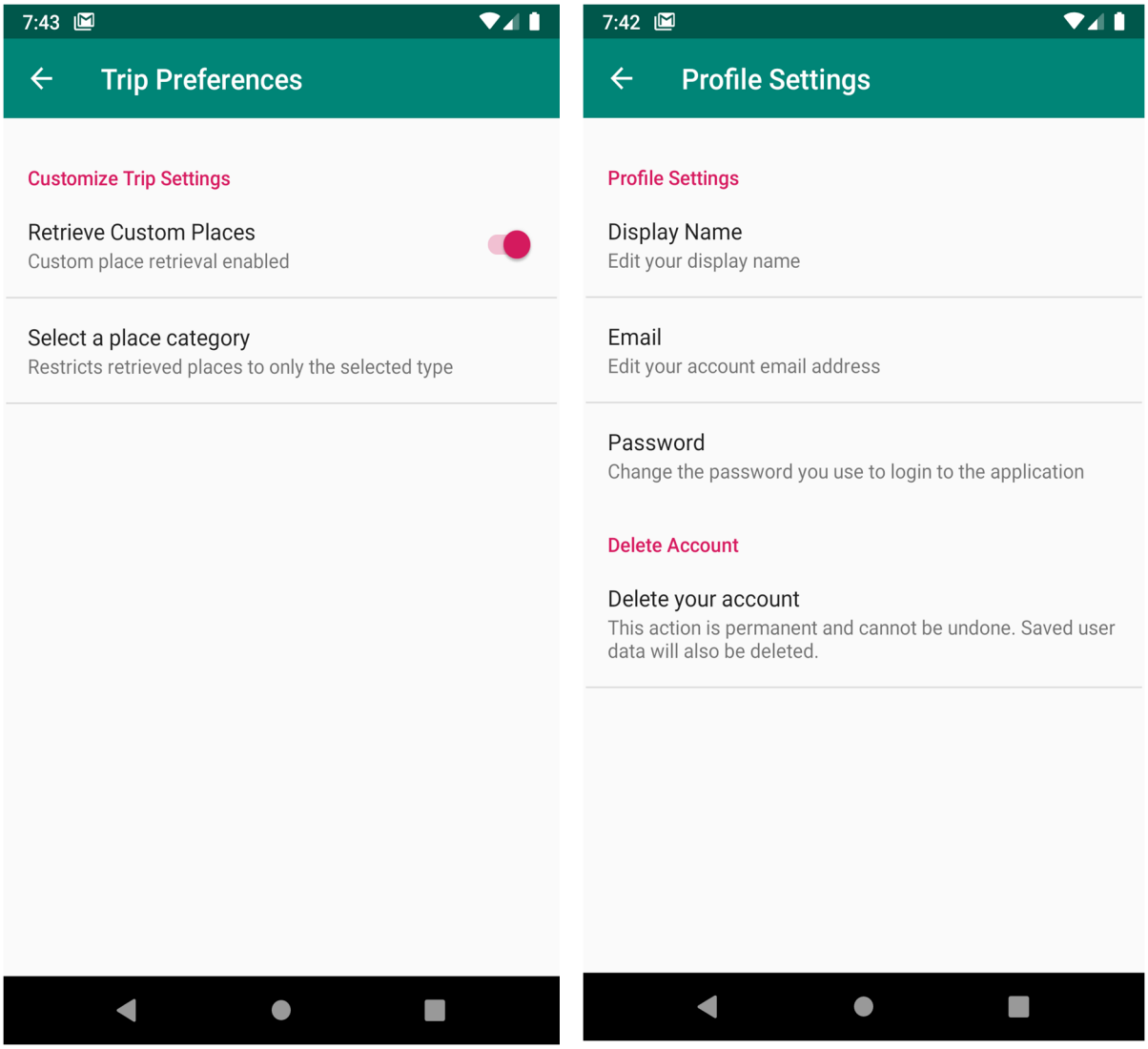

Figure 10 - Application Settings

The above figure shows the settings screens for the application. The settings shown on the left allow users to customize their trip planning experience by selecting the type of places to retrieve for planning future trips. On the right, settings related to basic user account management are shown.

Code Snippets

This section includes code snippets for the main features of the application along with brief explanations of each.

Place Search

```
private String generateNearbyRequestURL(double latitude, double longitude) {
    //requestURL = "https://maps.googleapis.com/maps/api/place/nearbysearch/
    // json?location=-
33.8670522,151.1957362&radius=1500&type=restaurant&keyword=cruise&key=AIzaSyBrqqD2yEMK
BRABm-3hFUzZy3xzZ-hI-to";

    double radius = 50000;
    String type =
PreferenceManager.getDefaultSharedPreferences(this).getString("specify_places_type_pre
f","point_of_interest");
    String outputFormat = "json";
    String formattedNearbyRequestURL;

    formattedNearbyRequestURL =
"https://maps.googleapis.com/maps/api/place/nearbysearch/"
            + outputFormat + "?location=" + latitude + "," + longitude + "&radius=" +
radius + "&type=" + type + "&key=" + getString(R.string.googlePlacesAPIKey);

    return formattedNearbyRequestURL;
}
```

The above method takes a latitude and longitude value and generates a request URL to query the Google Places API for places in any given city. The URL produced as a result of execution of this method is then passed to the following method which handles the JSON response.

The method for parsing the JSON response is given below. It accepts a request URL as a parameter and then uses Volley to parse through the JSON as needed. In the code snippet, values are being assigned to java variables, which will allow the fetched values to be natively used in the application.

```
private void ParseNearbySearchJSON(String requestURL) {
    mSearchedPlacesList.clear();

    JsonObjectRequest request = new JsonObjectRequest(Request.Method.GET, requestURL,
null, new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
            try {
                 //other code

                   String mPlaceID = result.getString("place_id");
                    String placeName = result.getString("name");
                    Log.d("placeName", "onResponse: placeName " + placeName);
                    String placeSummary = result.getString("vicinity");
                    String iconURL = result.getString("icon");

                 //continued

                    …

            } catch (JSONException e) {
                Log.d("Parse error", e.toString());
            }

        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.e("Error fetching results", error.toString());
            Toast.makeText(PlaceSearchActivity.this,"Server Error! If the problem
persists contact administrator.",Toast.LENGTH_LONG).show();
        }
    });
    mRequestQueue.add(request);
}
```

Add Place

The following method takes the place information to save as input parameters and uses Firebase real-time database's single value event listener to save the values from the input parameters to the database.

Place information is retrieved from a fetch place request. The values retrieved from that response after parsing is supplied to the addPlace method as input to save to the database.

```
private void addPlace(final String placeID, final String placeName, final String
placeSummary, final String iconURL) {
    DatabaseReference addPlaceRef =
FirebaseDatabase.getInstance().getReference("Trips/SelectedPlaces/"+mSelectedTripID);
    addPlaceRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            int childrencount= (int) dataSnapshot.getChildrenCount();
            Map<String,Object> placeValues = new HashMap<>();
            placeValues.put("iconURL",iconURL);
            placeValues.put("placeDesc",placeSummary);
            placeValues.put("placeID",placeID);
            placeValues.put("placeName",placeName);

mRef.child("Place"+(childrencount+1)).setValue(placeValues).addOnCompleteListener(new
OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if(task.isSuccessful()){
                        Toast.makeText(getBaseContext(),"Place added to
trip",Toast.LENGTH_LONG).show();
                    }
                    else{
                        Toast.makeText(getBaseContext(),"Couldn't add
place",Toast.LENGTH_LONG).show();
                    }
                }
            });
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
            Log.e("AddPlace", "onCancelled: "+databaseError );
        }
    });
```

Show scheduled trip

The code snippet given below shows how a child event listener is used to show only “scheduled” trips under the scheduled fragment in the main activity. This is done by using the assigned date and the trip status as stored in the firebase database and implemented through the application logic. Using a similar approach, trips are showed in the appropriate category when applicable.

```
mChildEventListener = new ChildEventListener() {
    @Override
    public void onChildAdded(@NonNull DataSnapshot dataSnapshot, @Nullable String s) {
        GeneratedTrip generatedTrip  =dataSnapshot.getValue(GeneratedTrip.class);
            if(generatedTrip.getTripStatus().equals("Scheduled")){
                generatedTrip.setTripID(dataSnapshot.getKey());
                mTripsList.add(generatedTrip);
                mScheduledTripAdapter.notifyDataSetChanged();
                if(dataSnapshot.hasChildren()){
                    HandleEmptyRecyclerView();
                }
        }
    }
```

Delete User Account

Below is an extract of the code that deletes an existing user account. This is done using the Firebase SDK's delete() method, calling it on the currently logged in user and re-authenticating them.

```
AuthCredential credential = EmailAuthProvider.getCredential(useremail,passwordInput);
mCurrentUser.reauthenticate(credential).addOnCompleteListener(new
OnCompleteListener<Void>() {
    @Override
    public void onComplete(@NonNull Task<Void> task) {
        if(task.isSuccessful()){
            final String nameOfUser = mCurrentUser.getDisplayName();
            mCurrentUser.delete()
                    .addOnCompleteListener(new OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull Task<Void> task) {
                            if(task.isSuccessful()){
                                accountDeletionStatus=true;
                                Toast.makeText(getActivity(),"User Account
Deleted!",Toast.LENGTH_LONG).show();
                                DeleteUserData(nameOfUser);
                                startActivity(new
Intent(getActivity(),LoginActivity.class));

                            }
```

Essential implementation notes

This section aims to pen down the major constraints that were faced during implementation and discuss how these influenced decisions at the time of application development.

- One of the limitations of the Google Places API that I faced during developing the application is that the nearby place search was that only one value could be passed for the "type" parameter at a single instance. It follows that if multiple types of places are to be retrieved, it will have to be explicitly done through multiple API calls which could prove

to be expensive in terms of computing resources depending on how many types of places are being requested

- The Google Places API can return responses to requests in either XML or JSON formats. However, since JSON is the de-facto standard for data exchange these days, I decided to use that for the purposes of my application. Moreover, the firebase real-time database also operates in the same format, so deciding on a single format for all data-related tasks helped in not letting things get too overly complex
- The viewpager widget on the place details activity that is used to show place photos is useful only when photos for that place exist in the APIs response. This means that the widget doesn't always display photos, but significant slowness has been observed in the parsing process even when images are available.
- The application settings allow users to set a place category to retrieve when generating trips, however, sometimes, it might appear as though the wrong categories of places are being retrieved. This happens due to the fact that the Google Places API has more than one categories assigned to some of the places. Therefore, a place that is classed as both a "park" and a "hotel" will show up in both categories.

Summary

This chapter lists the various classes, interfaces and layout files designed for the application. Details of the implementation phase and, where applicable, the rationale behind the decisions taken is also discussed. Snippets from the actual application code are included to give an idea of how different features of the application were implemented along with images of the working application. The chapter concludes by a mention of the limitations observed and their impact on the development of the project.

Chapter 6: Testing

Thorough testing of any application is critical to ensure that the software system does what is required of it and carries a certain degree of reliability. Not too long ago, testing was considered an activity separate from the rest of the software development life-cycle. However, with advancements in the field over time, testing is now seen as a core activity that is central to producing a high-quality software. Not only that, a multitude of software testing techniques come into use and are constantly being employed in the industry today. This chapter documents the procedures adopted for testing the application.

Unit Testing

This software testing technique is concerned with testing the individual components that constitute a larger software with the idea that if the smaller parts are validated for correctness, it would imply overall correctness for the software. Unit testing was performed throughout the implementation phase of the project, where each new feature was tested for correctness after being developed. This helped to rectify hard to spot bugs in the code and verify the behavior of the features. For this, the Logcat feature of Android Studio in conjunction with the debugger proved to be particularly useful. I used the Logcat window to observe values of variables after specific operations and the Android debugger to home-in on the data flow during the execution of methods.

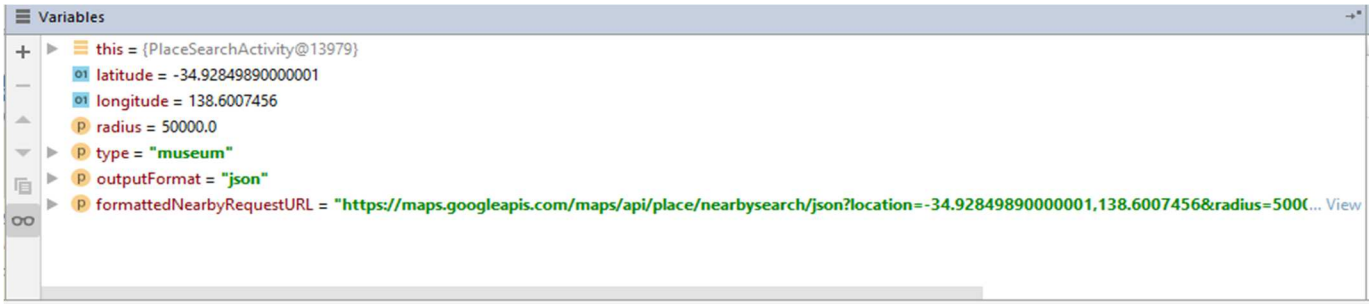

Figure 11 - Debugging the application

In the above figure, values that collectively make up the request URL can be observed. The request URL itself is also shown. The URL shown retrieves a list of all museums in Adelaide, Australia.

This is just one example of the methodology adopted for unit testing various features of the application. Similar steps were taken for every new feature to ensure that the newly written code didn't cause the application to break.

Functional Testing

Functional testing involves testing a system against a set of requirements to ensure conformance. In the context of the project, the responses of the Google Places API were observed to reaffirm that they provided the data necessary for the application. To do so, the different request URLs produced during the execution of the application were observed in the browser. This aided in confirming that the responses were as expected.

The following figure shows the result of a fetch place request in the browser. It can be seen that the JSON response contains fields for name, address, and photos etc. all of which are used at various points within the application.


```
{
   "html_attributions" : [],
   "result" : {
      "adr_address" : "\u003cspan class=\"street-address\"\u003e300 King St\u003c/span\u003e, \u003cspan class=\"locality\"\u003eLondon\u003c/span\u003e, \u003cspan class=\"region\"\u003eON\u003c/span\u003e \u003cspan
class=\"postal-code\"\u003eN6B 1S2\u003c/span\u003e, \u003cspan class=\"country-name\"\u003eCanada\u003c/span\u003e",
      "international_phone_number" : "+1 519-439-1661",
      "name" : "DoubleTree by Hilton Hotel London Ontario",
      "opening_hours" : {
         "open_now" : true,
         "periods" : [
            {
               "open" : {
                  "day" : 0,
                  "time" : "0000"
               }
            }
         ],
         "weekday_text" : [
            "Monday: Open 24 hours",
            "Tuesday: Open 24 hours",
            "Wednesday: Open 24 hours",
            "Thursday: Open 24 hours",
            "Friday: Open 24 hours",
            "Saturday: Open 24 hours",
            "Sunday: Open 24 hours"
         ]
      },
      "photos" : [
         {
            "height" : 3144,
            "html_attributions" : [
               "\u003ca href=\"https://maps.google.com/maps/contrib/111750390914533529170/photos\"\u003eDoubleTree by Hilton Hotel London Ontario\u003c/a\u003e"
            ],
            "photo_reference" : "CmRaAAAAvkQ-GkIvaRzeWR6F3vsENYpQKaTwaCEXVOYW_RqDLrdVqYTAsiJGWu5WKANkE2ILGq1eVXzmuQ021RPluL-L9kFsY32hpE8QXCT7MfTiekpxKbwQjTt2rzWXLH65Zn7hEhAzq-eWpzR4t0WDSkJ6tXs6GhS9Ztyup-Rs6jEWaoasPEVSSoH13w",
            "width" : 2088
         },
         {
            "height" : 2088,
            "html_attributions" : [
               "\u003ca href=\"https://maps.google.com/maps/contrib/111750390914533529170/photos\"\u003eDoubleTree by Hilton Hotel London Ontario\u003c/a\u003e"
            ],
```


Figure 12 - Verifying the JSON response

User Interface Testing

The UI of the application was tested manually throughout the implementation and where necessary, rectifications were made. To cross-check, the testing was performed both on an emulator and on an actual Android device. During the testing process, it was sometimes noted that the user interface had inconsistencies, in which case a conscious effort was made to provide a uniform look-and-feel to enhance user experience.

Test Cases

This section lists the major test cases that were used to verify application behavior and conformance.

Login and displaying relevant data

Objective: This test case is directed at testing one of the features of the application. Any user should be able to login to the application upon entering the correct credentials and only the relevant trip data should be visible to each user

Testing Strategy: Multiple mock user accounts were created on the application and sign-in was attempted for each. Once logged in new trips were created from each of the accounts involving different cities

Result: It was observed that each user was able to successfully log in to the application and only the relevant data was shown to each user. As part of the test, it was also noted that the application adequately responded to situations where incorrect credentials were input while logging in or when the user forgot the login details.

Create a new trip

Objective: To test if new trips are accurately created and displayed to the user. Trips should be created within the constraints chosen by the user and also should be displayed in the appropriate category based on the assigned date

Testing Strategy: Trips were created on different dates and from different user accounts. Some of the trips were assigned a date in the past while some of them were assigned a future date. In addition, a different category of places was selected for each trip

Result: Trips that were assigned a date in the past were displayed under the “past” tab while those that were assigned a future date showed up under the “scheduled” tab in the main screen. It was also noted that the place search retrieved only the places of the type that was set by the user in application settings.

Display places associated with a trip

Objective: To reaffirm whether the places associated with a trip show up as expected

Testing Strategy: Places associated with a trip were changed many times to see the outcome

Result: The places associated with any trip were displayed as expected. It was noted that the order in which the places were displayed was the same as their order of insertion in the firebase real-time database.

Set duration values for each place

Objective: To test if duration values are properly assigned to each place associated with a trip. The application must allow insertion of only those values that fall within the permissible range for hours and minutes

Testing strategy: Correct and incorrect values for hours and minutes were deliberately assigned to each place and the behavior of the application was monitored.

Result: The application handled constraints on input values as expected. Where values for duration were already assigned, they showed up in the corresponding edit-texts. Where not, error messages were appropriate displayed. Places could be easily selected using the spinner widget and values could be over-written and saved to the database or could just be deleted altogether

Add a new place

Objective: To verify that a new place gets added as expected to an already existing trip

Testing Strategy: New places were added to an already existing trip using the location icon on the toolbar

Result: The autocomplete widget for the add place operation showed only place suggestions limited to the city that the trip was based. Once a place was selected, the associated details were saved to the database and the newly added place was shown in the recyclerview. Since the new

place was the last one to be added to the database, it also showed up after the previously existing places in the recyclerview.

Calculate trip summary

Objective: To verify if the total trip duration and the end time was calculated appropriately. The application must calculate end times only if a start time was assigned. Otherwise, the user should be informed

Testing Strategy: Start time values were set for different trips and the output was observed once the trip summary icon was clicked.

Result: It was observed that calculations for the total trip duration and the end times was accurate. If a start time was not allocated for a trip, the user was notified through the dialog. Additionally, end times were calculated only on devices running Android Oreo and above. As previously mentioned, this functionality is not supported on lower versions of the Android operating system and the user is notified in such a scenario.

Trip settings

Objective: To confirm that application settings behave in the intended way and cause the intended changes and to see if sensitive operations require users to re-authenticate before the changes are committed to the database.

Testing Strategy: Each setting was applied and the result was noted

Result: All settings work as expected and sensitive operations required the user to verify identity before continuing

Summary

This chapter is dedicated to application testing. It discusses how various types of testing techniques were applied in the context of the project and presents test cases for the prominent features of the application.

Chapter 7: Conclusion

This chapter looks at the future work that can be done to build upon the existing features of the application. It also briefly discusses the limitations of the current implementation.

Outcomes of the project

The project allowed me the opportunity to undertake a full-fledge system and develop it from scratch, using agile practices. I was able to learn how to communicate with APIs to achieve the required tasks and also take responsibility for my work. The work enabled me to hone my time-management skills as well, which will prove to be useful in future endeavors of life.

Future Work

This section provides some ideas regarding what can be improved in this application and how the current work can be built upon.

- It can be a good idea to allow users to pick places to add to a trip off a map instead of using autocomplete widgets. Both approaches serve the same purpose and this is being proposed as a suitable alternative
- Improvement is needed for the viewpager widget that is used to show place photos in the current version of the application. The current implementation has a drawback of photos loading quite slowly where available.
- Push notifications can be implemented informing the user when it is a trip day
- A Broadcast receiver can be used to automatically monitor when a date changes and make the appropriate adjustments to the application
- The ability to specify multiple place categories to retrieve during trip creations is another useful update that can be added to future versions of the application

Limitations:

- The UI can be improved to enhance user experience. An idea in this regard is to incorporate material design techniques in the future versions of the application.
- The current implementation of the project is a native Android application. It will be a good idea if the same application could be made available on ios as well. There are two

approaches to do this: A separate application could be developed exclusively for the ios or a framework such as ionic could be used to develop a hybrid application that works on both platforms.

- Some features such as place re-ordering have only be implemented partially due to shortage of time. The place re-ordering feature, for example, does not change the order of the places in the database. These features can be completed to enhance user experience.

Summary

This chapter provides ideas of future work along with limitations of the current implementation that can be rectified.

Bibliography

1. Android Developer Guides, available at http://developer.android.com/guides (Accessed: 10 February, 2019)
2. TripIt, highest rated trip planner and flight tracker, available at http://tripit.com/web (Accessed: 15 February,2019)
3. Roadtrippers, plan your journey and find amazing places, available at http://www.roadtrippers.com (Accessed: 15 February, 2019)
4. Sygic Travel, planner and itinerary maker, available at http://travel.sygic.com/en (Accessed: 15 February, 2019)
5. Material Design Guidelines, available at https://material.io/design/guidelines-overview/ (Accessed: 23 February, 2019)
6. Google Codelabs – Android Fundamentals, available at https://codelabs.developers.google.com/android-training (Accessed: 10 February, 2019)
7. The Java Tutorials by Oracle, available at https://docs.oracle.com/javase/tutorial/ (Accessed 10 February, 2019)
8. Places API Comparison, available at https://docs.google.com/document/d/1f_3r8Ebx94ojzeXpOb9-P6ehnSHV5Pt3WlgA0ryzYE/edit (Accessed 9 May, 2019)
9. Ted Permissions library, available at https://github.com/ParkSangGwon/TedPermission (Accessed 12 June, 2019)
10. Volley library, available at https://github.com/google/volley (Accessed 16 June, 2019)
11. Picasso library, available at https://github.com/square/picasso (Accessed 16 June, 2019)
12. University of Leicester, referencing styles by department, available at https://www2.le.ac.uk/library/help/referencing/styles-by-department (Accessed 01 September, 2019)
13. University of Leicester's guide to the Author-Date referencing style, available at https://www2.le.ac.uk/library/help/referencing/author-date (Accessed 01 September, 2019)