

Event Manager for Large Scale Enterprise Events

by

Graeme Hosford

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

in the
Faculty of Engineering and Science
Department of Computer Science

May 2020

Declaration of Authorship

I, Graeme Hosford, declare that this thesis titled, ‘Event Manager for Large Scale Enterprise Events’ and the work presented in it are my own. I confirm that:

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

Signed:

Date:

CORK INSTITUTE OF TECHNOLOGY

Abstract

Faculty of Engineering and Science

Department of Computer Science

Bachelor of Science

by Graeme Hosford

The organisation of events on the scale of hundreds of people inside a company comes with some serious difficulties such as managing accommodation, keeping track of times and venues.

These problems are magnified further when there are multiple events which need to be handled and multiple different pieces of software are required to keep track of all the relevant data. There could be an app for managing accommodation bookings; a website for organising flights for remote workers; and so on. All of these software solutions could also be on different platforms and thus lead to further difficulties in keeping track of event information.

Upon completion this app will act as a single source of truth for managing these events and provide tangible benefits to companies such as tracking the stats for each worker attending an event; where they are staying; which events they are attending if there are multiple; and so on. This app will also allow for the direct management of event details by the event organiser and allow for both attendees and organisers to keep up to date with changing event information

Acknowledgements

Thanks to my research and implementation supervisor Mary Davin.

Contents

Declaration of Authorship	i
Abstract	ii
Acknowledgements	iii
List of Figures	viii
List of Tables	x
Abbreviations	xi
1 Introduction	1
1.1 Motivation	1
1.2 Contribution	2
1.3 Structure of This Document	2
2 Background	5
2.1 Thematic Area within Computer Science	5
2.1.1 User Roles	6
2.2 A Review of the Thematic Area	6
2.2.1 Eventbrite	6
2.2.1.1 Eventbrite Pros	7
2.2.1.2 Eventbrite Cons	8
2.2.1.3 How This Project Improves on Eventbrite	8
2.2.2 SocialTables	9
2.2.2.1 SocialTables Pros	9
2.2.2.2 SocialTables Cons	10
2.2.2.3 How This Project Improves on SocialTables	10
2.2.3 Eventzilla	11
2.2.3.1 Eventzilla Pros	11
2.2.3.2 Eventzilla Cons	12
2.2.3.3 How this Project Improves on Eventzilla	13
2.2.3.4 State of the Art Summary	14
2.2.4 Technical Review	15

2.2.4.1	Mobile Applications	16
2.2.4.2	Kotlin	19
2.2.4.3	Cloud Computing/Server-side Programming	20
3	Problem - Enterprise Event Manager	23
3.1	Problem Definition	23
3.2	Objectives	25
3.3	Functional Requirements	25
3.3.1	Mobile App Functional Requirements	25
3.3.1.1	Functional Requirement No. 1	26
3.3.1.2	Functional Requirement No. 2	26
3.3.1.3	Functional Requirement No. 3	26
3.3.1.4	Functional Requirement No. 4	27
3.3.1.5	Functional Requirement No. 5	27
3.3.1.6	Functional Requirement No. 6	27
3.3.1.7	Functional Requirement No. 7	28
3.3.1.8	Functional Requirement No. 8	28
3.3.1.9	Functional Requirement No. 9	28
3.3.1.10	Functional Requirement No. 10	29
3.3.1.11	Functional Requirement No. 11	29
3.3.1.12	Functional Requirement No. 12	29
3.3.1.13	Functional Requirement No. 13	30
3.3.1.14	Functional Requirement No. 14	30
3.3.1.15	Functional Requirement No. 15	30
3.3.2	Firebase Functional Requirements	31
3.3.2.1	Functional Requirement No. 16	31
3.3.2.2	Functional Requirement No. 17	31
3.3.3	Ruby Server Functional Requirements	32
3.3.3.1	Functional Requirement No. 18	32
3.3.3.2	Functional Requirement No. 19	32
3.4	Non-Functional Requirements	33
3.4.1	Mobile App Non-Functional Requirements	33
3.4.1.1	Non-Functional Requirement No. 1	33
3.4.1.2	Non-Functional Requirement No. 2	33
3.4.1.3	Non-Functional Requirements No. 3	33
3.4.1.4	Non-Functional Requirement No. 4	34
3.4.1.5	Non-Functional Requirement No. 5	34
3.4.1.6	Non-Functional Requirement No. 6	34
3.4.2	Firebase Non-Functional Requirements	35
3.4.3	Ruby Server Non-Functional Requirements	35
3.4.3.1	Non-Functional Requirement No. 7	35
4	Implementation Approach	37
4.1	Architecture	37
4.1.1	Mobile App Technologies Used	38
4.1.1.1	Language - Kotlin	38
4.1.1.2	Code Structure - Clean Architecture	38

4.1.1.3	Dependency Injection - Dagger2	40
4.1.1.4	AndroidX	41
	Android KTX	42
	Android Jetpack Navigation Component	42
4.1.1.5	RxJava/RxAndroid	42
4.1.1.6	Retrofit	43
4.1.1.7	Butter Knife	43
4.1.1.8	Glide	44
4.1.1.9	Moshi	44
4.1.2	Firebase Technologies Used	44
4.1.2.1	Firebase Authentication	45
4.1.2.2	Firebase Cloud Firestore	45
4.1.2.3	Firebase Cloud Messaging	46
4.1.3	Ruby Server Technologies Used	47
4.1.3.1	Ruby on Rails	47
4.1.4	How These Technologies Are Used	48
4.2	Risk Assessment	49
4.2.1	Risk No. 1	49
4.2.2	Risk No. 2	50
4.2.3	Risk No. 3	50
4.3	Methodology	51
4.3.1	Learning New Technologies	51
4.3.2	Project Management Approach	52
4.4	Implementation Plan Schedule	52
4.5	Evaluation	54
4.6	Prototype	55
4.6.1	Mobile App Prototype	55
4.6.2	Ruby Server Prototype	68
5	Implementation	69
5.1	Difficulties Encountered	69
5.1.1	Easy	69
5.1.1.1	Heroku Database Support	69
5.1.1.2	Android Navigation Component Support	69
5.1.1.3	Limited Screen Space	70
5.1.1.4	Android FCM Library	70
5.1.1.5	Android List State Handling	72
5.1.1.6	Firebase Cloud Functions Parameter Names	72
5.1.2	Medium	73
5.1.2.1	Ruby Development Environment Setup	73
5.1.2.2	Mobile App Architecture Setup	73
5.1.2.3	Ruby Notification Support	74
5.2	Actual Solution Approach	75
5.2.1	Solution Architecture	75
5.2.2	Use Cases	77
5.2.3	Risk Assessment	77
5.2.4	Development Methodology	77

5.2.5	Implementation Schedule	77
5.2.6	Evaluation	78
5.2.7	Prototype	78
6	Testing and Evaluation	81
6.1	Metrics	81
6.1.1	Functional Requirement Metrics	81
6.1.2	Non-functional Requirement Metrics	83
6.2	Results	85
7	Discussion and Conclusions	86
7.1	Solution Review	86
7.2	Project Review	87
7.2.1	Skills Developed	88
7.3	Conclusion	89
7.4	Future Work	90
	Bibliography	92

List of Figures

2.1	Eventbrite Logo	7
2.2	SocialTables Logo	9
2.3	Eventzilla Logo	11
2.4	Android Logo and Version Icons	16
2.5	Kotlin Logo	19
4.1	Overview of the Proposed System	37
4.2	Clean Architecture Diagram	38
4.3	Android Specific Clean Architecture Diagram	39
4.4	Android DI Framework Recommendations	40
4.5	Android App Size Approximations	41
4.6	Dagger2 Component Graph Setup	41
4.7	RxJava Basic Observable Example	43
4.8	Firebase Logo	45
4.9	Proposed Database Design	45
4.10	Ruby on Rails Logo	47
4.11	Mobile App Prototype Sign In/Register Screen	56
4.12	Mobile App Prototype Create/Join Company Screen	57
4.13	Mobile App Prototype Create and Join Company Screens	58
4.14	Mobile App Prototype Event List	59

4.15	Mobile App Prototype Event Details	60
4.16	Mobile App Prototype Attendees List	61
4.17	Mobile App Prototype Create Event Screen	62
4.18	Mobile App Prototype Add Person Event Details Screen	63
4.19	Mobile App Prototype Event Notification Example	64
4.20	Mobile App Prototype Company Details Screen	65
4.21	Mobile App Prototype Company Member Requests Screen	66
4.22	Mobile App Prototype User Profile Screen	67
4.23	Ruby Server Prototype Create Company Response	68
5.1	Mobile app screen for creating events	71
5.2	Final System Architecture Implementation	75
5.3	Mobile app package structure	76
5.4	Mobile App Prototype and Implementation versions of Event List Screen	79
5.5	Mobile App Event List Differences	80

List of Tables

2.1	User Roles	6
2.2	State of the Art Summary	14
2.3	Research Sources - Conferences, Journals and Academic Papers	15
2.4	Research Sources - Blog/Forums/Video Resources	16
2.5	Research Sources - Books	16
2.6	Android Version History	18
4.1	Sprint Schedule	54
6.1	Mobile app functional requirements metrics	82
6.2	Firebase functional requirements metrics	83
6.3	Ruby Server functional requirements metrics	83
6.4	Mobile Non-functional Requirement Metrics	84
6.5	Ruby Server Non-functional Requirement Metrics	85
7.1	Future Work User Stories	91

Abbreviations

App	A pplication
AS	A ndroid S tudio
POJO	P lain O ld J ava O bject
OS	O perating S ystem
Admin	A dministrator
Info	I nformation
IJTC	I nternational J ournal of T echnology and C omputing
HTML	H yper T ext T ransfer P rotocol
DI	D ependency I njection
SDK	S oftware D evelopment K it
JSON	J ava S cript O bject N otation
SQL	S tructured Q uery L anguage
REST	R epresentational S tate T ransfer
NoSQL	N ot O nly S QL
MVC	M odel V iew C ontroller
FCM	F irebase C loud M essaging
MVP	M inimum V iable P roduct
UI	U ser I nterface
CPD	C ontinuing P rofessional D evelopment
MVVM	M odel V iew V iew M odel

Chapter 1

Introduction

1.1 Motivation

It is important to do a project on this topic as the organisation of events is a time consuming and tedious process particularly when the details for a large scale event must be handled along with the needs of potentially hundreds of people. I decided to develop this app after witnessing these issues occur first hand in the SaaS company Teamwork.

Each Summer, around June or July, Teamwork organises a week long get together for the whole company, including flying in remote workers to the main offices, with the goal of promoting team building and cross team communication. This week culminates in an activity day followed by a party later that evening.

The organisation of this week took months of work from the HR team to organise flights, accommodation, venues and entertainment for all employees. The initial idea for this app came from the Mobile team lead after a member of the HR team suggested it to him that a single piece of software which would let them track all the required info for these events would have made their task of organising and tracking information much easier.

While this approach would still require they use different software to organise the individual elements such as a flight booking website, hotel website, etc, they would now be able to enter all this info into a single source of truth thus allowing the easy retrieval of information and preventing any potential confusion which may arise between people checking potentially out of sync information from different sources

1.2 Contribution

From the point of view of a company, for both admins and regular employees using this application the benefits are obvious.

For company admins all relevant info related to events are kept in one place which limits potential miscommunication when pulling info from different sources and this info can be tightly managed by said admins therefore company events can be created and managed in a more efficient way by reducing the potential for human error.

For the average employee within the company it makes getting the info about an upcoming event far easier, particularly if this employee is a remote worker who is travelling or the event itself is remote. Rather than having to contact whomever is organising an event to get the required info such as location, hotel booking and so on, or vice versa the organiser having to contact each employee to do the same thing, the needed info can be added to this app and employees can check it whenever is convenient and the need for a time consuming back and forth about arbitrary details is removed.

1.3 Structure of This Document

Chapter 2 - Background

- Section 2.1 **Thematic Area with Computer Science** - This section contains an overview of the core goals of the project and how they relate to computer science.
- Section 2.2 **Review of the Thematic Area** - This section contains an overview of some already existing products which are similar in scope to this project, including their perceived pros, cons and how this project can improve on it. This section also goes through a review of the thematic area outlined in section 2.1.

Chapter 3 - Problem - Event Enterprise Manager

- Section 3.1 **Problem Definition** - This section outlines the problem definition being tackled in this project.
- Section 3.2 **Objectives** - This section outlines the objectives of this project.
- Section 3.3 **Functional Requirements** - This section outlines the functional requirements for each facet of this project, mobile app, Firebase backend and Ruby backend.

- Section 3.4 **Non-Functional Requirements** - This section outlines the non-functional requirements of this project giving a broad outline of how it will work.

Chapter 4 - Implementation Approach

- Section 4.1 **Architecture** - This section outlines the proposed architecture of the project including the libraries and frameworks to be used, the different aspects of the system, and how all of these will work together.
- Section 4.2 **Risk Assessment** - This section outlines the risks which could be faced throughout the development of this project as well as steps which can be taken to mitigate them.
- Section 4.3 **Methodology** - This section outlines the plan to learn any unknown technology and also outlines the agile approach taken to development.
- Section 4.4 **Implementation Plan Schedule** - This section outlines the sprint plan for how this project will be developed over the course of semester 2.
- Section 4.5 **Evaluation** - This section discusses the steps which will be taken to evaluate how well this project achieves its aims.
- Section 4.6 **Prototype** - The prototype section shows a proposed prototype of the entire system.

Chapter 5 - Implementation

- Section 5.1 **Difficulties Encountered** - This section enumerates and discusses the difficulties which arose during the implementation phase of this project.
- Section 5.2 **Actual Solution Approach** - This section outlines the actual solution used in the implementation phase of this project, including any changes results from the difficulties encountered.

Chapter 6 - Evaluation

- Section 6.1 **Metrics** - This section details the metrics used to determine if this project can be considered a success or not.
- Section 6.2 **Results** - This section discusses the metrics from Section 6.1.

Chapter 7 - Conclusions and Future Work

-
- Section 7.1 **Solution Review** - The solution review section reviews how well the project tackles its goals based on the results from Chapter 6.
 - Section 7.2 **Project Review** - The project review section discusses how well addressed the implemented project is, as well as what could be done differently if given another chance to tackle this project.
 - Section 7.3 **Conclusion** - The conclusion section enumerates the main conclusions gained from this project.
 - Section 7.4 - **Future Work** - The section on future work details further work which could have been achieved with this project if not for time constraints and other limiting factors.

Chapter 2

Background

2.1 Thematic Area within Computer Science

The core goal of this project is to provide an easy to use mobile app which allows organisers to create and manage large scale events and to deal with the issues that arise, with a focus on company issues such as organising travel and accommodation.

This system will consist of an Android mobile app front end supported by a Firebase back end to handle most necessary server side features such as database management and sending notifications. This will be supplemented by a Ruby server to handle the few things Firebase cannot do such as scheduling when a notification should be sent and assigning an ID to a company.

The Android app will be the user facing side of the system, allowing for all the features outlined in section 3.3.

Firebase will act as the engine to the app by using Firebase Cloud Storage to save all new users, new companies and adding events and their related details, as well as keeping track of all edits done to each of these. Firebase will also take charge of sending notifications to a users device and authenticating new accounts which sign up to the app.

The Ruby server will play a relatively minor but important role in this system. It will be in charge of generating a unique ID for new companies to allow users to easily join them and will be in charge of scheduling when notifications will be sent to users.

2.1.1 User Roles

For the use of this app there will be two roles a user can play the role of an attendee and the role of an admin.

Role	Description
Attendee	An attendee is a user who will use this app purely for responding to events, they will not have permissions to create events
Admin	An admin will be able to do all the same things attendee can however they will also have permission to create events, invite potential attendees to these events and add info to an attendee regarding specific event(s)

TABLE 2.1: User Roles

The example functionality of each user role stated in table 2.1 is not extensive. A far more detailed overview can be found in section 3.3.

2.2 A Review of the Thematic Area

There are many Event Manager systems available for use. Here a few of the more popular will be listed along with their perceived merits and faults and how this project will improve upon those faults.

It should be noted here that while suggested improvements will be offered on the perceived faults of a particular product this does not mean this finished project will be an equal in every way to the pros listed for each product below. These products have had years on the market to grow and change while being backed by a team of developers, neither of which are options in this individually produced fixed term project.

Among this list will also be systems which may not be as popular but are similar in scope to what this project is envisioned to be and will therefore be included for comparison purposes.

2.2.1 Eventbrite

First up is Eventbrite, stating that "Our mission is to bring the world together through live experiences" [2]. Eventbrite is an Event Manager which allows for organising events



FIGURE 2.1: Eventbrite Logo[1]

quite simply through either their main website (<https://www.eventbrite.com>) or through a dedicated mobile solution on both iOS[3] and Android[4].

From investigation into Eventbrite I have gathered the following list of pros and cons. It should be kept in mind however, that this list is not 100% extensive and is merely some of the more obvious points which came to mind after an hour or so of using the product. There are certainly elements both positive and negative missing here.

2.2.1.1 Eventbrite Pros

- Easy to follow yet extensive event setup process allowing for the expected input such as Event Title, location, Description, etc. but also allowing for a great deal of control over details listed in later points.
- All events created using this product must have tickets for attendees whether it be a free or paid event, this allows for managing attendance based on confirmed numbers.
- Both public and private events are supported.
- Tickets have different sale options, they are either sold online through Eventbrite or "on the day" at the event itself.
- The ticket booking process can be heavily customised to suit the needs of the event organiser allowing for such changes as buying timeout when ordering tickets online, whether or not group bookings are allowed and if so whether information should be collected from every group member or just the member booking the tickets.
- Discount codes can be created for people to use when buying tickets.
- The screen for a successfully booked ticket(s) and confirmation email can both be customised to add extra information which attendees may need to be aware of.
- Eventbrite integrates with Facebook to allow people to buy event tickets through their Facebook account.

- Event organisers can choose to pay to have ads shown on both Facebook and Instagram.
- Affiliate programs can be utilised to encourage people to buy tickets to an event through the use of rewards.
- Event analytics can be viewed to get information on how many tickets have been sold and how much profit made.

2.2.1.2 Eventbrite Cons

- No integration with social media other than Facebook and Instagram, while these services certainly have many users and allow for a large awareness campaign other social media giants such as Twitter are completely left out.
- While an event having mandatory tickets is good for tracking attendance it is not well suited to free events open to the public as a ticket really isn't necessary in that scenario. In this case Eventbrite may be used simply as a way of advertising an event and the use of tickets may go unused this is basically ignoring a major feature of this product and also means attendance analytics are much harder to track if attendees have no need of a ticket and can just turn up without one.
- In the same scenario listed in the previous point the attendee analytics may also be skewed the other direction by online campaigns of reserving tickets with no intention of attending the event, reports of such false reservation campaigns have been well noted in recent years.
- The Eventbrite Mobile App, on both iOS and Android, does not provide an all-in-one solution for using Eventbrite as a service. The main Eventbrite app is used for seeing what events are on, buying tickets and so forth but an entirely separate app called Eventbrite Organiser must be used if one wants to create and manage an event. Furthermore, while the Android version of this app seems a well made piece of software, the iOS version on the other hand holds a rating of 3.0 stars on the AppStore[3] at the time of writing with common complaints relating to a lack of what users consider to be important features and unexpected behaviour.

2.2.1.3 How This Project Improves on Eventbrite

Eventbrite does not contain specific attendee detail management as is currently envisioned for this project. From a specifically mobile point of view its main features, event attendance and creation/management, are also split across multiple apps.

As outlined previously, the intention behind this project is to provide an all-in-one solution for Event Management including viewing events, responding to invites, and creating/managing events all in one app. Beyond this it is also envisioned that an admin managing an event will be able to input attendee specific details for an event such as accommodation, meals and so forth, a feature which Eventbrite is lacking on all of its platforms.

2.2.2 SocialTables



FIGURE 2.2: SocialTables Logo[5]

The next example to look at is Social Tables. On their About Us page SocialTables describes what they do as "We are connecting the hospitality industry through effortless group management solutions that create successful face-to-face events" [6]. SocialTables takes a much higher focus on attendee detail management in comparison to Eventbrite which makes it a good example to look at here for this project.

A list of perceived pros and cons are again listed below. Once again these lists are not 100% extensive and are only the perceived good and bad points from this author's point of view.

2.2.2.1 SocialTables Pros

- Guest management is far more comprehensive in comparison to Eventbrite. Details such as whether a guest will be bringing additional people with them, whether they require wheelchair access, as well as extra fields to indicate their choice of meal at an event. Input for additional custom notes on a guest is also available which could cover any other required information.
- The seating arrangements and associated meal choices can be set out for guests.
- Diagrams for venues can be created to show the size and shape of the venue and including the previously mentioned seating arrangement. This diagram creator also comes with some templates to allow for creating certain types of venue diagrams easier, such as classrooms, conferences halls and so forth. This feature also includes the type of seating that will be used at a venue - round tables, rows, benches, etc.

2.2.2.2 SocialTables Cons

- Based on experience using the service and their own description of themselves[6] SocialTables seems to cater exclusively to the management of formal sit down events. Many of the highlight features of this product, such as seating arrangements, wouldn't be useful for a casual or informal event which doesn't fit specific criteria.
- Diagram creation measurements can only be done in inches or feet. This comes off as an odd choice as surely a venue, for example a conference hall, would be measured in either feet or metres and not inches.
- Another issue raised by the previous point is that measurements cannot be supplied using the metric system. All input given by a user who uses metric would therefore require them to either convert from metres to feet/inches themselves or input the metre measurement and assume guests will take it to be metres but it would technically be incorrect.
- Having used the diagram creator a few different times on reliably fast connections it can be said with some confidence that it is a frustratingly slow feature to use. While it is relatively simple to create the diagrams themselves it frequently becomes unresponsive for a few seconds at a time and leaves the user in this case wondering if it has crashed and even when it does not freeze it typically gives sluggish performance at best.
- SocialTables mobile presence is quite poor with no Android app at the time of writing. To use this product one is constrained to either the website or iOS. In a world which is becoming increasingly mobile first and with Android being the most dominant OS on the market currently this comes off as a massive issue for potential and existing users alike.

2.2.2.3 How This Project Improves on SocialTables

The main area this project will improve upon in regard to SocialTables is their lack of an Android mobile app.

Though the areas this product covers in regard to guest management are certainly impressive as outlined in section 2.2.4.1 Android by far makes up the majority of worldwide smartphone usage and with people increasingly going mobile first in their way of accessing the internet, and its related services, the lack of an Android app comes off as a massive oversight on the part of SocialTables

2.2.3 Eventzilla



FIGURE 2.3: Eventzilla Logo[7]

Eventzilla defines what they do, based on their About Us page of their website as "Eventzilla helps people organize successful events" [8]. Eventzilla is an event management solution meant to cover both formal and informal event types and can be accessed through www.eventzilla.net or by use of dedicated mobile apps on both iOS[9] and Android[10].

Once again laid out below are the perceived pros, cons and potential improvements to be made on Eventzilla during the course of this project.

2.2.3.1 Eventzilla Pros

- Covers event types of both formal and informal settings. This is already something of an improvement over SocialTables which provided a solution for formal events only.
- Reserved seating plans can be made for paid events.
- Seating plans can cover many potentially very different venues from sectioned seating in a stadium to the non-sectioned seating of the type found in most casual event settings.
- Seating plans can be generated by uploading an existing image and tracing the plan over this.
- Seats can be reserved specifically for users of wheelchairs or other people requiring certain accessibility options be made for them.

- Overall the seating tool is far easier to use than the similar tool available from SocialTables. While they provide almost identical functionality there were no instances of unresponsiveness or slowness while testing Eventzilla.
- Eventzilla provides integration with social media and email marketing providers.

2.2.3.2 Eventzilla Cons

- Specific parts of the seating tool can become tedious to use. Laying out rows of seats provides a simple to use shortcut which allows for any number of rows and columns to be set out at once. However, when setting positions of tables in the layout they must be set one at a time. This quickly becomes frustrating, especially when one considers that a small movement before setting a position would result in an incorrect position being set and thus require further editing to fix.
- When attempting to save the created reserved seating plan there was no save button on the dialog being used. Once the close button was clicked a message was displayed saying that the seating plan had been saved. However when attempting to move onto the next step in event creation an error message was shown saying that the created seating plan must be saved before proceeding. After searching for the option to save to no avail the process could only be continued after reserved seating was disabled completely.
- The dialog which allows a user who is creating an event to create ticket types seems to have some bugs in how it handles errors. Firstly when adding a new ticket, even when all required information is filled in there is an error stating "Please add at least one standalone ticket type to add a merchandise ticket type". There is no provided definition of what either standalone ticket type or merchandise ticket type mean and neither is there any link which could take the user to the screen which adds a standalone ticket type.
- The social integration provided only extends as far as Facebook, while it is not stated this could potentially include Instagram as well. However this goes back to the same issue which Eventbrite had where major social platforms such as Twitter or LinkedIn are completely left out.
- Once again similarly to Eventbrite the efforts at a mobile app from Eventzilla are not an all-in-one solution, instead offering two separate apps, one for event creation and management, and the other for attendees going to an event. As noted by some of the reviews[10] on the attendee app this is not a choice many users are happy with and leads to the somewhat disappointing score of 3.6 at time of writing.

Although it should be noted that at the time of writing there is only 7 reviews made by users.

- From testing the Android mobile app some usability issues have been noticed. When attempting to scroll through a list of options rather than scroll, or do nothing if scrolling is not possible, whichever option is under the scrolling finger when it is lifted acts as if it has been clicked on and opens unwanted screens.
- Again, as noted in the Play Store reviews[10], there are issues with unresponsiveness. From this author's own experience clicked items can take an estimated 1 - 2 seconds to open. While this may not seem like a long wait compared to a website which may take that long to load a new page, this is relatively poor performance for mobile apps where opening a new screen should be more or less immediate. i.e. less than 1 second.

2.2.3.3 How this Project Improves on Eventzilla

Similarly to SocialTables the main area where this project can improve on Eventzilla is on their efforts at a mobile experience. As outlined in section 2.2.3.2 Eventzilla does not provide a mobile app which acts as an all-in-one solution, instead forcing users to download separate apps to get the same functionality as if they were using the website. These apps then have some usability problems which make them frustrating to use.

The issues outlined in regards to the reserved seating and adding tickets can also be addressed. While a feature of creating seat graphs and generating tickets will almost certainly not appear in this project as they would be far too large in scope to complete in the allotted time frame they do speak to a more general issue of usability.

Not including a save button on the seat layout and the error messages on adding tickets which give no usable hints as to how the problem should be fixed are both fairly major violations of Nielsen's Heuristics[11]. The error messages for the seating and tickets, both outlined in section 2.2.3.2, which do not point out a clear path on how to fix the problem is an obvious failure to take into account the ideas behind the following heuristics.

- **Visibility of system status** - "The system should always keep users informed about what is going on, through appropriate feedback within reasonable time"[11].

- **Error prevention** - "Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action"[11].
- **Help users recognize, diagnose, and recover from errors** - "Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution"[11].

These issues of poor mobile experience and usability concerns are both areas of potentially major improvement. While a slow website may be excusable in some contexts this is not the case for mobile apps which are so easily discarded from a users device if they deem it not up to standard.

2.2.3.4 State of the Art Summary

In table 2.2 is a summary of sections 2.2.1, 2.2.2 and 2.2.3 containing the main points outlined in those sections.

Product	Summary
Eventbrite	Eventbrite provides an easy to follow event creation process with almost every tool one might need to manage their event. The main downsides of Eventbrite are their mobile apps which do not provide an all-in-one solution for organisers and attendees and certain features which are not always useful depending on the type of event.
SocialTables	SocialTables is more suited to formal events with options to set seating arrangements and provide extra details for attendees - wheelchair access, food restrictions, etc. The main con of SocialTables is the poor attempt at a mobile experience which does not include an Android app at all.
Eventzilla	Eventzilla gives event management which covers both formal and informal events and like Eventbrite provides all the expected options an organiser could want in basic event setup. However, there are some major usability issues as outlined in section 2.2.3.

TABLE 2.2: State of the Art Summary

2.2.4 Technical Review

At a generic high level the scope of this project falls under the following areas:

1. **Mobile Applications** - The mobile app used in this project requires knowledge of Android, Kotlin, Java and communicating with a back end server.
2. **Cloud Computing** - Knowledge of Ruby and communication with server-side REST APIs is required for this project. The Firebase aspect of this project is mostly self-contained within the provided library which handles all network communications and caching while providing a simple to use API, therefore only a basic knowledge of Firebase should be more than enough for this project.

In relation to these core areas, the sources listed in tables 2.3, 2.4 and 2.5 have been chosen as points of research.

Aside from the sources listed in these tables others may also be referenced for minor points but are not listed for the sake of conciseness and readability.

Source	Description
International Journal of Technology and Computing	IJTC[12] is a monthly journal which publishes papers containing the latest in cutting edge research and innovation.
Evolution of Android Operating System: A Review	This paper presents a study of how the Android OS has improved over the years with each major release[13].
Characterizing the Transition to Kotlin of Android Apps	This paper covers a study on the transition from Android development being 100% Java to the gradual emergence of Kotlin[14].
Review Study of New Era of Android Kotlin	This paper studies the differences between Java and Kotlin and the potential benefits which can be gained from using Kotlin[15].
Architectural Styles and the Design of Network-based Software Architectures	This paper written by Roy Fielding as his Ph.D thesis outlines his concept of a RESTful web architecture which tackled the issues of scalability the web faced at the time[16].

TABLE 2.3: Research Sources - Conferences, Journals and Academic Papers

Source	Description
--------	-------------

Android Developers Blog	The official Android developers blog from Google[17]
Android Authority	Android Authority[18] is a blog following all things Android and is a major source of Android news
Kotlinlang.org	The official website of the Kotlin language containing invaluable resources such as documentation and examples[19].

TABLE 2.4: Research Sources - Blog/Forums/Video Resources

Source	Description
REST API Design Rulebook 1st Edition	REST API Design Rulebook by Mark Masse[20] is a book focusing on the design and implementation of web-based REST APIs.
RESTful API Design 1st Edition	RESTful API Design by Matthias Biehl[21] offers an insight into the best practices of API design for the modern day web.
Mastering Android Development with Kotlin: Deep dive into the world of Android to create robust applications with Kotlin	Mastering Android Development with Kotlin by Milos Vasic[22] gives a rundown on developing for Android using Kotlin and why Kotlin has become the language of choice for Android developers.

TABLE 2.5: Research Sources - Books

2.2.4.1 Mobile Applications

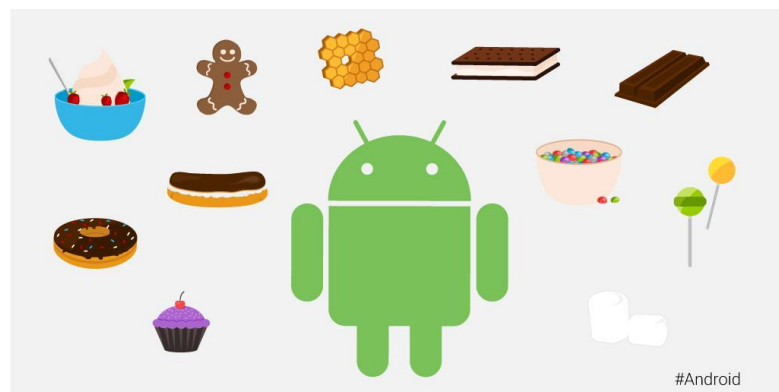


FIGURE 2.4: The Main Android Logo and Logos for Different Versions[23]

The history of mobile application development is far longer than most people would assume. Most people today think of iOS and Android when they hear mobile app but the field goes back far earlier to include development for many distributions of mobile system, Symbian OS being the most prolific of these. These days Symbian is a dead project, having been truly supplanted by both Android and iOS, with the latest release being October 2012 and with no newly manufactured devices supporting the system. An excellent description of why Symbian lost favour compared to modern phone operating systems is "... its story perhaps illustrates the brutal fate that [a]waits any technology that cannot keep pace with modern demands" [24]

The modern day idea of mobile app development really came to fruition with the release of iOS and Android.

From this point onward only Android development will be looked at as this project does not contain any iOS development.

Android is a mobile operating system initially developed by Android Inc. who were later bought out by Google [25]. Designed to be the engine behind modern smartphone devices, a quote from one of the founders of Android Inc outlines the goal of Android as developing "smarter mobile devices that are more aware of its owners location and preferences" [26].

Throughout its development Android has gone through a significant evolution, going from what would today be considered a bare-bones OS to being one which for many people easily acts as a replacement to a laptop or desktop computer. Below in table 2.6 is a summary of the major changes each iteration of Android has brought. This table will only include the changes from version 21 and upward as this version introduced major changes to Android and previous versions are generally no longer supported by most new projects.

OS Name	Version	Major Features
Lollipop	21	New Material Design for system and apps, 3D views, 64-bit MIPS, Support for a range of new sensors [27]
Marshmallow	23	New power saving mode, New device permission restrictions, Fingerprint detection, Added shortcuts to quickly access device camera from lock screen [27]

Nougat	25	Major improvements on power saving mode introduced in version 23, support for multi-language input, device boot time improvements[27][28]
Oreo	26	Faster boot speed, Limits on apps which are rarely used doing background work, Improved text suggestion, System wide form auto-fill, Picture-in-picture mode[29]
Pie	28	Adaptive battery, Predictive app actions, New gesture controls, App time limits[30]
Android 10	29	Dessert codenames dropped, System wide live caption, System wide dark theme, More fine grained location and privacy controls, new system wide gesture navigation[31]

TABLE 2.6: Android Version History

As of 2019 Android holds the majority in worldwide smartphone market share, while exact figures are hard to find, even the lowest estimates put its share at over 75%[32], and some sources as high as roughly 85%[33][34] of devices around the world.

Android applications are developed using either Java, Kotlin or a combination of the two allowed by Kotlin's Java compatibility features. Android development is done primarily through Android Studio, the officially developed and supported IDE from Google based on the IntelliJ IDE from JetBrains. There are also other options available such as IntelliJ with the Android plugin, Eclipse with the Android plugin or various tools such as Xamarin or Appcelerator which allow for code to be written in JavaScript and compiled to create cross platform apps for both Android and iOS.

Android Studio comes conveniently packaged with little additional setup overhead while including all a developer needs for Android development. From Android specific features such as a layout editor and device emulator, which is particularly useful for detecting compatibility issues across different versions of Android, as well as built-in support for running instrumentation tests and UI tests which all comes together to offer a robust easy to use IDE. More standard IDE features are also offered such as lint checks, unit tests and code highlighting.

When it comes to getting apps on an Android device the Google Play Store is the main option as it comes pre-installed on all Android devices, as well as being officially supported by Google, it also includes various protections against potential malware, in

the form of Play Protect, whereas third party app sources may not have a comparable protection, these factors make the Play Store the most convenient source of new apps.

Besides the Google Play Store the Amazon App Store provides basically the same functionality but even its nearly 500,000 app selection[35] pales in comparison to the roughly 2.7 million available from the Google Play Store[36]. Aside from these two major options there are also various other third party sources for mobile apps available which won't be discussed for the sake of conciseness.

2.2.4.2 Kotlin



FIGURE 2.5: Kotlin Logo[37]

Kotlin is a newer language and not currently as well known as many other languages such as Java. While it is not specifically tied to Android development it is heavily used by Android developers around the world with "More than 50% of professional Android developers now use the language to develop their apps"[38]. Besides just this, statistics on Google Play apps have found that "nearly 60% of the top 1,000 Android apps contain Kotlin code"[39]. Google themselves have even announced "If you're starting a new project, you should write it in Kotlin"[38][40].

Furthermore according to the conclusions found in "Characterizing the Transition to Kotlin of Android Apps: A Study on F-Droid, Play Store, and GitHub"[14], the uptake of Kotlin in Android projects is summarised by the words "the transition from Java to Kotlin was most of the times fast and unidirectional"[41]. In effect, from the 1200+ projects looked at in this study those which featured Kotlin generally had its use growing rather than diminishing and doing so at a relatively quick pace. This is hardly unexpected given the reduction in code that Kotlin allows with "estimates indicate approximately a 40% cut in the number of lines of code"[42] among its many other improvements over the Java language.

Aside from these benefits Kotlin also comes with Java compatibility which allows all Kotlin projects to include Java if the developers wish to do so but importantly, means that all existing libraries which work with Java will also work with Kotlin[43].

Additionally, Google has also announced that Android Jetpack, their suite of libraries designed to reduce boilerplate Android code and make working with difficult features easier[44], will also become increasingly Kotlin first[38][40].

2.2.4.3 Cloud Computing/Server-side Programming

These days one can rarely hear about anything tech related without the word "cloud" being involved somehow. indeed in recent years the tech industry has begun to move away from the business model of standalone applications which are downloaded and reside on a users computer and there is now a major focus on online services which provide the same functionality as any desktop application could.

While there is a long history behind how we got to this point, this project will not have a web based user interface and any cloud computing used will focus entirely on a server-side, backend which the user has no direct access to. Therefore this summary will focus on that side of things and leave out any discussion on Web applications as they are not of any relevance here.

Originally developed at CERN by scientist Tim Berners Lee[45] the idea of the World Wide Web began initially as a tool for scientists and other academics to rapidly share data with one another[46]. Aside from the initial concept and creation of the web Lee is also credited with the invention of HTML, the first web browser and the first web server amongst many others[45].

Note here that the World Wide Web and the internet are not the same thing despite being incredibly similar. The internet refers to a series of computers linked together worldwide while the World Wide Web refers to the information accessed on the internet. In essence "the Internet is infrastructure while the Web is service on top of that infrastructure" [47].

These days this idea of instantly accessible information has exploded in popularity, this is hardly surprising given the web as we know it today but even in its infancy there was a popularity to it that was rarely seen in many historical innovations. At one point the number of web users was doubling every two months[48].

The backbone of the web, and its associated popularity, is the server. A server exists to act as a link in the massive network that is the internet and typically hosts some piece of information.

In this project there will technically be two server backends. One of these will be Google Firebase which requires very little setup from the developer. Firebase provides services such as cloud database storage, sending notifications to devices and authenticating users

among other things, all with no need to create login forms or database tables. It simply works as an out-of-the-box solution once the minimal setup is completed. Firebase will be discussed in more detail in section 4.1.

The other server backend to be used in this project will be a Ruby language server. While this will require more setup than Firebase it will be used far less often and only for some small scenarios in which Firebase is unsuited.

These scenarios include generating an ID for each company which is created and scheduling a point in time when notification should be sent to a device. While Firebase does create unique IDs for each entry added to a database, these IDs are made up of hex strings typically 30+ characters in length which makes them unsuitable to use as human readable text as is intended for this project. Again more on this in section 4.1.

The Ruby server in this project will work as a REST API which sends data in the form of JSON. This will allow for sending data which is as descriptive as needed due to the hierarchical structure of JSON. Key-value pairs are used to make up the JSON data but the 'value' can be anything from lists to nested JSON data rather than just simple text or numbers.

More importantly than the data structure however is the use of a RESTful API to achieve this. By using an API developed on the REST principles there are great gains made in terms of scalability. As described in "RESTful API Design"[21] by Matthias Biehl, REST APIs "are used for building distributed software systems, whose components are loosely coupled"[49] and when correct standards are followed they are further described as "simple, clean, clear and approachable"[49].

This REST system was initially developed by Roy Fielding. In 1993 he set about tackling the issues cropping up in regard to scalability in the web[48][50]. In addressing this problem he came up with 6 key areas in which the web architecture of that time was not fit for purpose.

The proposals Fielding came up with are as follows:

- **Client-Server** - The web should work based on a client-server relationship in which "client and servers have distinct parts to play. They may be implemented and deployed independently, using any language or technology, so long as they conform to the Web's Uniform Interface"[51]. The enforcement of a client-server type relationship in web-based applications essentially amounts to separation of concerns[52]
- **Uniform Interface** - "The interactions between the webs components...depend on the uniformity of their interfaces. If any of the components stray from the

established standards, then the Web's communication system breaks down" [51]. Through a uniform interface between otherwise relatively unrelated components "overall system architecture is simplified" [52] and "implementations are decoupled from the services they provide, which encourages independent evolvability" [52]. The downside to this is that "information is transferred in a standardized form rather than one which is specific to an applications needs" [52] this in turn leads to a degradation in efficiency

- **Layered System** - "The layered system constraints enable network-based intermediaries...to be transparently deployed between a client and server...Network-based intermediaries are commonly used for enforcement of security, response caching and load balancing" [51]
- **Cache** - "Caching response data can help reduce client-perceived latency, increasing the overall availability and reliability of an application" [51]. In a RESTful system "the data within a response must be implicitly or explicitly labelled as cacheable or non-cacheable" [52]
- **Stateless** - "The stateless constraint dictates that a web server is not required to memorize the state of its client applications" [51]. In essence this means all communications "from client to server must contain all the information necessary to understand the request" [52]
- **Code-On-Demand** - The Code-On-Demand constraint "enables web servers to temporarily transfer executable programs, such as scripts or plug-ins to clients" [51]

This new web architecture with a focus on scalability was then detailed by Fielding in his Ph.D thesis "Architectural Styles and the Design of Network-based Software Architectures" [16] in 2000.

Chapter 3

Problem - Enterprise Event Manager

3.1 Problem Definition

The problem being tackled in the development of this project stems from the fact that to organise an event many details must be managed by whomever the event organiser is. In the case of large scale event planning, such as with a company, issues which may seem trivial and considered once off problems are magnified tenfold and can now become serious, time-consuming tasks.

To take a seemingly trivial example, if organising an event which includes a sit-down meal. the organiser must gather information on food allergies among the event attendees and pass this on to catering staff. In the worst case the organiser would have to ask all attendees individually if they have any allergies, which granted is an unlikely scenario. However, even in the far better case where each attendee with allergies contacts the organiser to let them know, the event organiser must still go through responses one by one which is a time consuming task.

The best scenario in this case is obviously that there is some online form made available for attendees to put in their allergies. This then acts as a single source of information on food allergies which the organiser can gather the information from. This approach still holds some drawbacks however.

Firstly a major potential issue is that the following series of events could take place:

1. An online form is provided for attendees of an event to add their allergy information

2. All attendees enter their allergy information correctly
3. This first event takes place with no problems
4. A second event is scheduled and allergy information is required again. The previous information cannot be reused as this event has some new attendees
5. It may be assumed that all attendees know how and where to enter their allergy information and so a reminder may not be provided. It could also be the case that an older version of this allergy information is used by mistake through human error
6. Believing that they have the correct information the event organiser passes this on to the catering service for the event
7. Attendees of this second event who did not attend the first event do not have their allergy information recorded correctly. This could pose a serious health risk to these people if they are given the wrong food

Secondly, unless the information from this form is saved and in some way associated with the person who entered it then for a future event it may be required that attendees enter this information all over again. For multiple events this would quickly become a tiresome, repetitive task for attendees to complete.

This is something that many would likely have considered a trivial detail to get sorted, but as outlined here the task of asking people if they have any food allergies quickly becomes tedious for both organisers and attendees and can potentially have serious consequences for people if the newest information is not used at all times.

This is obviously a serious issue in the process being used here. A potentially far better solution is one which allows the attendees to input their allergy information within their own profile. Once this is entered the attendee most likely will never have to think about it again thus removing their need to enter the same information over and over. Tied to this could be an option for the event organiser to simply add attendees to an event then with the click of a button be given a list of all attendees and their associated allergies. As long as the event organiser clicks this button to get allergy information on each event, out-of-date information will not be used and each attendee is saved from entering the same information repeatedly.

While this is still open to human error if the organiser forgets to click the button for the allergy list, when this is done correctly it solves both problems outlined above. Besides this, it would be a lot less likely that an event organiser forgets the allergies when they are provided in the same system as event organisation. It is certainly less likely they

will forget in this scenario when compared to them needing to get this information from a separate system.

This is just one detail of many involved in event organisation. Now imagine the many hours of work that must go into organising dozens of other seemingly trivial details for dozens or more people at an event. This time quickly adds up. This workload is not helped by having many sources of information which an organiser must sort through and put together in a clear way in order to make sure mistakes are not made.

3.2 Objectives

The objectives behind the development of this mobile-first event management system can be classified under the following points:

- Develop an Android based mobile app which allows users to manage their events and related details.
- Develop a backend Ruby based server which handles notification scheduling and creating IDs for a company.
- Perform the required setup steps for including Firebase in the project such that the Firebase instance successfully integrates with both the Ruby server and the mobile app to allow both to complete their requirements.

The requirements of each of these are outlined in section 3.3.

3.3 Functional Requirements

The requirements of this project are broken down here between the three different parts which make up this project, the mobile app, the Firebase backend and the Ruby server backend.

3.3.1 Mobile App Functional Requirements

Some features in the mobile app are available to admin users only, these are highlighted as such along with an explanation of why. All features with no such highlight will be available to all users.

3.3.1.1 Functional Requirement No. 1**Description**

A user can sign up for an account.

Reason

A user will need to sign up for an account in order to use the system.

Fit Criterion

A new user account is created with the email and password the user entered.

3.3.1.2 Functional Requirement No. 2**Description**

A user can sign into their account.

Reason

A user will need to be signed into their already existing account in order to use the system.

Fit Criterion

When successfully signed in a user will have access to the main features of the app (seeing events, creating events, etc).

3.3.1.3 Functional Requirement No. 3**Description**

A user can create a new company.

Reason

The idea of a company containing users will need to exist in this app so that the correct users are grouped together and receive the correct information.

Fit Criterion

Once the company is created the user can see and create events. The user who created the company will automatically be an admin user.

3.3.1.4 Functional Requirement No. 4

Description

A user can join a company.

Reason

A user will need to join a company in order to see that company's events.

Fit Criterion

Once a company has been joined the user can see the events being organised by that company.

3.3.1.5 Functional Requirement No. 5

Description

A user can see the list of events they are invited to.

Reason

A user will want to see what events they are invited to and be able to see the details of these events such as time and location. This will include events the user has declined an invitation to attend so that they can change their response later.

Fit Criterion

When a user opens the app and is signed into their account and has joined a company they can see a list of events they are invited to.

3.3.1.6 Functional Requirement No. 6

Description

Admin Only Feature - A user can create an event.

Reason

An admin will want to create an event for the company.

This is an admin only feature as events which will be considered an official company event should not be open to just anyone to create.

3.3.1.7 Functional Requirement No. 7**Description**

Admin Only Feature - A user can add attendees to an event.

Reason

An admin will want to add people to an event.

This is an admin only feature as events intended to have specific attendees should not allow new attendees to be added by unauthorised users.

Fit Criterion

Once added to the event the chosen people are shown in the list of invited attendees.

3.3.1.8 Functional Requirement No. 8**Description**

A user can respond to an event invite.

Reason

A user should be able to inform event organisers whether or not they can attend the event.

Fit Criterion

Once a response - either Going or Not Going - has been sent the list of invited attendees will be updated to reflect this response.

3.3.1.9 Functional Requirement No. 9**Description**

A user can see the list of people within their own company.

Reason

A user will want to see who is in their own company.

Fit Criterion

A user will be able to see a list of all other members of their company.

3.3.1.10 Functional Requirement No. 10**Description**

A user can see their own profile.

Reason

A user will want to review the details of their own profile in order to verify the information is correct and inform any decisions on editing their profile information.

Fit Criterion

When the user chooses to see their own profile the correct information as stored in the Firebase backend database is displayed to them.

3.3.1.11 Functional Requirement No. 11**Description**

A user can edit their own profile.

Reason

A user will want to edit their own profile and update any desired information such as name, contact details or picture.

3.3.1.12 Functional Requirement No. 12**Description**

A user can see the profile of other users

Reason

A user will want to see who else is in their company and also see the provided contact details of that user.

Fit Criterion

A user can click on another user from the list of company members and is shown that person's profile.

3.3.1.13 Functional Requirement No. 13**Description**

A user receives a notification when invited to an event or as an event reminder.

Reason

A user will want to be kept informed on events they are invited to and will also want appropriate reminders when getting near the time of an event they are attending.

Fit Criterion

When invited to an event a user receives a notification shortly after informing them of this invite. When approaching an upcoming event reminder notifications are pushed to the users device at a selected amount of time before the event starts.

3.3.1.14 Functional Requirement No. 14**Description**

Admin Only Feature - An admin can set event specific details for other users.

Reason

An admin will want to set extra information specific to a single event on certain users. For example, times and flight number of flights, location and check-in time at a hotel.

This is an admin only feature as event related expenses which are being paid for by the company should be inputted only by a chosen company representative. Any such expense which is being paid for by the attendee in question is their responsibility and therefore this person should not be allowed to enter those expenses alongside company expenses as this would cause confusion.

Fit Criterion

Once an admin has entered some information for an attendee of a specific event this information will become visible to the attendee in question.

3.3.1.15 Functional Requirement No. 15**Description**

Admin Only Feature - An admin can approve or deny a users request to join a company.

Reason

The system of joining a company will involve entering a simple human-readable series of digits which represent a company on the system. As these IDs will be generated as simple integer values they would be far too easy for a user to guess and join a company they should not.

This is an admin only feature as only admins should be able to approve new company members.

Fit Criterion

Once a user enters the ID in order to join a company they will not have access to company event information until they are approved by an admin.

3.3.2 Firebase Functional Requirements

3.3.2.1 Functional Requirement No. 16

Description

FirebaseAuth can create and verify a new user account when one is created through the sign-up process in the mobile app.

Reason

Once an account is created it will need to be verified as containing a valid email and password and also be persisted to allow for repeated use.

Fit Criterion

Once sign up is complete on the mobile app an account with the user specified email and password is created by FirebaseAuth. This account is persisted and a user will not have to create an account again.

3.3.2.2 Functional Requirement No. 17

Description

Event manager information will be persisted to Firebase Cloud Storage.

Reason

Information on user profile, events and company members will need to be persisted to allow for constant access and always with the most up-to-date information.

Fit Criterion

When any information in the mobile app is changed this will be updated on the Firebase Cloud Storage database to reflect these changes. Any other user viewing this information after the changes are applied will see the new information displayed.

3.3.3 Ruby Server Functional Requirements**3.3.3.1 Functional Requirement No. 18****Description**

A unique ID is generated for each company created.

Reason

A company will need a unique human-readable ID generated to allow for users to join this company as joining simply by the company name would not be secure or scalable.

Fit Criterion

Once a company has been created a unique ID is returned. When joining a company this ID should successfully allow the user to access the main features of the mobile app once entered.

3.3.3.2 Functional Requirement No. 19**Description**

The server will schedule the exact time a notification will be sent.

Reason

Notifications which are event reminders will need to be sent at specific times.

Fit Criterion

A notification which has been scheduled will arrive on a users device at the time specified.

3.4 Non-Functional Requirements

Same as with functional requirements the non-functional requirements are broken down between the various parts of this project.

3.4.1 Mobile App Non-Functional Requirements

3.4.1.1 Non-Functional Requirement No. 1

Description

Setting up a user account should be a quick and easy process.

Reason

Potential users are unlikely to use the app if the initial setup process is slow.

Fit Criterion

A user should be able to create their account within 2 minutes.

3.4.1.2 Non-Functional Requirement No. 2

Description

Creating or joining a new company should be a quick and easy process.

Reason

Similarly to creating a user account, creating or joining a company should be a quick process as a lengthy setup will leave users frustrated with the experience of using the app.

Fit Criterion

A user should be able to create or join a company within 2 minutes.

3.4.1.3 Non-Functional Requirements No. 3

Description

The app should be easy to navigate.

Reason

A confusing and non-intuitive navigation will leave users frustrated when using this app if they have trouble accessing basic features.

Fit Criterion

A user should know how to navigate to any given part of the app after 10 minutes of usage.

3.4.1.4 Non-Functional Requirement No. 4**Description**

The user should always see information relevant to the current screen being viewed even during network failure.

Reason

Network failures are unavoidable but showing an empty screen to a user is not at all helpful to them.

Fit Criterion

When new data can not be loaded, previously saved cache data should be displayed instead.

3.4.1.5 Non-Functional Requirement No. 5**Description**

Data should be loaded as quickly as possible.

Reason

Slow load times will not provide the user with an enjoyable experience in using the app.

Fit Criterion

It should take no longer than 5 seconds to load data when on a reliable internet connection.

3.4.1.6 Non-Functional Requirement No. 6**Description**

The app will work in the same way across all versions of Android from API version 21 and upward.

Reason

Android is an ever changing and evolving system. As part of its continued development there are occasionally non-backward-compatible or breaking changes introduced on new version releases, Google does not have the best track record on informing developers of changes made to existing classes which change how they work on certain versions of the OS.

It is therefore important to be careful when using certain classes provided by the Android SDK to ensure that all users have the same experience using the Android app.

Fit Criterion

There should be no difference in look or functionality when using the app on a device with Android version 21 compared to a device using Android version 28.

3.4.2 Firebase Non-Functional Requirements

Firebase is group of third-party libraries providing backend database and authentication support among other features. All non-functional requirements which may exist related to uptime and availability requirements as well as constraints on how quickly the system is updated with newly provided data, are handled by Google and are therefore not under the control of the developer of this project.

3.4.3 Ruby Server Non-Functional Requirements

Similarly to Firebase the server in this case will be one provided by a third-party host therefore constraints on uptime and availability are outside the control of the developer of this project.

3.4.3.1 Non-Functional Requirement No. 7**Description**

The ID generated for a new company will be human-readable and easy to remember.

Reason

The ID which is generated by the server for a new company allowing members to join should be a short and simple to remember sequence allowing for a user to quickly enter it on the Android app. A lengthy string of random characters would not be suitable for this but a numeric sequence of values would. e.g. 00001, 00002, 00003 and so forth.

Fit Criterion

A user should be able to see the ID associated with the company they wish to join and be able to enter it on the Android app without needing to double check that the ID is correct.

Chapter 4

Implementation Approach

4.1 Architecture

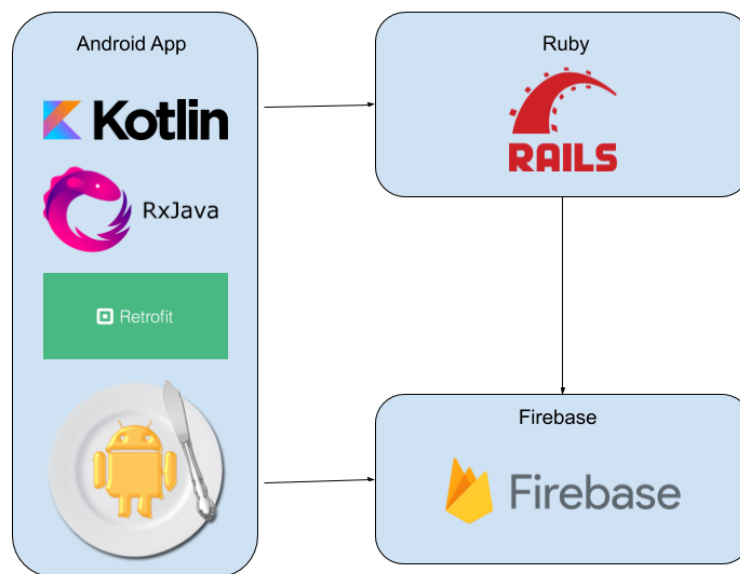


FIGURE 4.1: Overview of the Proposed System

As shown in Figure 4.1 the system consists of three parts, the Android mobile app, the Firebase backend and the Ruby backend.

The mobile app will communicate with both Firebase and the Ruby server as denoted by the arrows in the figure. The Ruby server will also have a communication to the Firebase backend in order to facilitate particular functionality such as notification scheduling as mentioned previously.

It should be noted that the logos in Figure 4.1 represent the libraries used in each part of the system but this is not extensive, as some libraries simply do not have logos. A

more complete list and their associated functionality is given in sections 4.1.1, 4.1.2 and 4.1.3.

4.1.1 Mobile App Technologies Used

4.1.1.1 Language - Kotlin

As outlined in section 2.2.4.1 Kotlin is now the language Google recommends for Android development. For this reason, and others outlined in section 2.2.4.1 Kotlin will be the language used for the Android app development aspect of this project rather than Java.

4.1.1.2 Code Structure - Clean Architecture

For the Android application aspect of this project the code will be using Clean Architecture, a code architecture developed by Robert Martin in his book "Clean Architecture: A Craftsman's Guide to Software Structure and Design" [53].

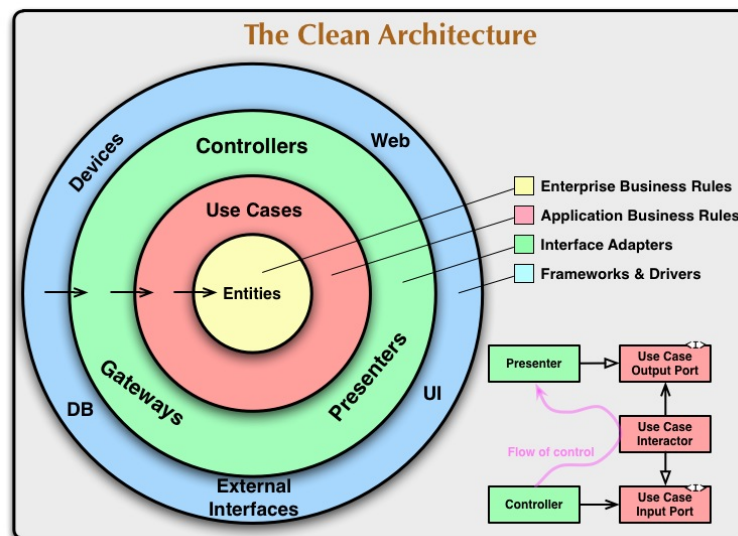


FIGURE 4.2: Clean Architecture Diagram[54]

Figure 4.2 shows the approach to clean architecture as devised by Robert Martin. In this diagram the outer layers know about their immediate inner layer but holds no reference to any layer inside that. A layer also knows nothing of its outer layers.

This project will be using a slightly modified version of clean architecture which has been devised by the Android development team at the company Teamwork in order to handle some Android specific issues which this base version of clean architecture does not address.

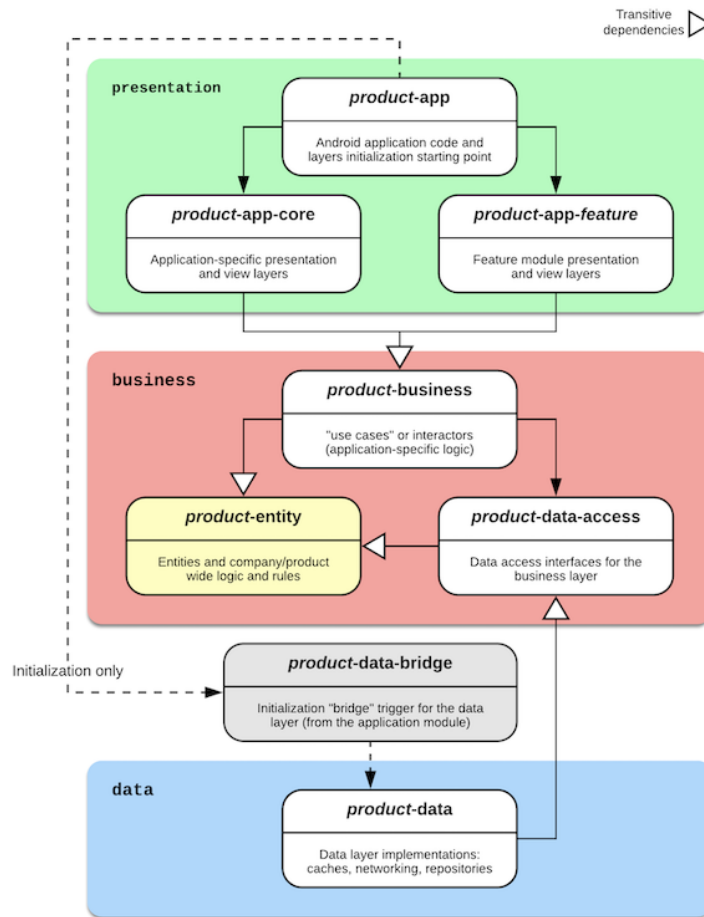


FIGURE 4.3: Android Specific Clean Architecture Diagram[55]

As seen in figure 4.3 there are three distinct parts to this clean architecture setup, Data, Business and Presentation. In this implementation there are extra modules included to deal with Android specific problems in relation to clean architecture.

The data-bridge modules exists only to initialise the DI graph for the data layer and contains no specific application logic.

The data layer covers operations related to network requests and database storage. In this project Firebase is used as the backend storage service and the Android libraries to access the Firebase API automatically handle caching. Therefore the data layer in this project will be used mainly for accessing Firebase but any extra work related to caching or threading will be unnecessary as the Firebase libraries handle these internally. The data layer will also be used to access the Ruby backend server but this will be in only limited circumstances given the relatively limited functionality of this server when compared to the Firebase backend.

The business layer is in charge of accessing the data layer functionality and applying any necessary application level business rules to the result obtained. This typically amounts to checking the validity of the data received and either just passing it on to the presentation layer when it is valid or passing some default value to the presentation layer when this data is not valid.

The presentation layer of this setup is in charge of calling the appropriate business layer methods when needed and obtaining a result. The presentation layer is also responsible for performing any transformations to the result obtained in order to make it suitable to show to the user.

While there are some clear cut differences between the presentation and business layers some functionality may seem like it could fall into either category. A common way of distinguishing responsibilities is to think of the presentation layer deciding the when and the business layer deciding the how. The presentation layer decides when something should be done, such as at the click of a button. The business layer decides how it should be done, for example whether it should happen asynchronously or not.

More in-depth detail on this exact clean architecture implementation can be found at <https://github.com/Teamwork/android-clean-architecture>. The only changes this project will make to this base setup is the conversion of any Java classes to Kotlin.

Clean Architecture will be employed extensively in this project as all classes in the mobile app will conform to the layering and restrictions set on this layers by the Clean Architecture code structure.

4.1.1.3 Dependency Injection - Dagger2

This mobile app will use Dagger2 as the dependency injection framework of choice. While there are many DI frameworks available Dagger2 is maintained by Google and as seen in figure 4.4 it is the only DI framework recommended for all sizes of Android app.

Project Size	Small	Medium	Large
Tool to Use	Manual DI Service Locator Dagger	Dagger	Dagger

FIGURE 4.4: Android DI Framework Recommendations[56]

The size of an Android app is based approximately on how many different screens it has. These approximate size definitions are outlined in figure 4.5.

Project Size	Small	Medium	Large
Number of screens	1-3	4-7	8+

FIGURE 4.5: Android App Size Approximations[56]

Dagger2 will be used extensively in this project as a means of keeping with the SOLID principles of Single Responsibility and Dependency Inversion. It is anticipated that almost all classes will use Dagger2 to inject dependencies.

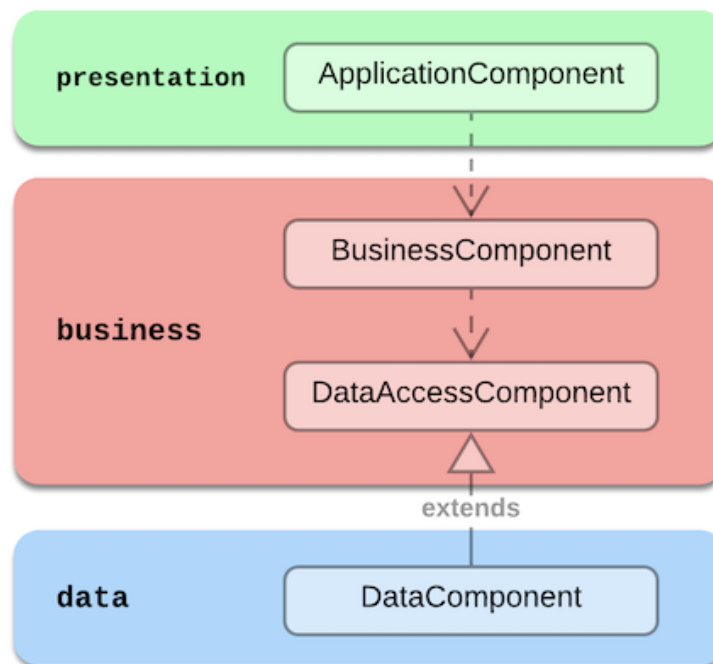


FIGURE 4.6: Dagger2 Component Graph Setup[55]

Shown in figure 4.6 is the setup for how Dagger2 will utilise multiple components in order to build a dependency graph across multiple modules

4.1.1.4 AndroidX

AndroidX refers to the Android support libraries which provide backwards compatibility to Android apps. It also encompasses Google's Android Jetpack suite of libraries.

AndroidX, despite officially being labelled as a group of support libraries, are a necessity in every Android project as many of the basic classes used in Android development come from it. There would therefore be far too much information to go into here if all AndroidX packages which are to be used were mentioned.

However, by leaving out the common uses of AndroidX which can be found in most, if not all, Android projects we can look at some of the more helpful features which come out of the Android Jetpack subset of libraries which will be used in this project.

Android KTX Android KTX acts as an add-on for the Kotlin language when used in Android. This library provides a set of extensions to Android classes which can replace verbose code with more concise versions which achieve the same functionality[57].

Android Jetpack Navigation Component The Android Jetpack navigation component is a library which handles the boilerplate code associated with navigating between screens in an Android app.

While it makes up a minor part of mobile apps navigation in Android can often require a lot of boilerplate code to handle states. This code can often grow massive and lead it to be buggy and difficult to maintain.

The navigation component takes care of these problems as well as adding support for features which would normally be considered quite difficult to implement, such as deep linking.

In short the use of this library allows for a massive reduction in boilerplate code, handles difficult use cases and thus frees the developers time allowing them to focus on user facing features[58].

There are many more helpful libraries available as part of the Android Jetpack suite. More information can be found on them at <https://developer.android.com/jetpack>.

The AndroidX libraries will be used heavily throughout this mobile app as they are now a standard in Android development and the exact uses are listed above.

4.1.1.5 RxJava/RxAndroid

RxJava is a library designed to allow for quick development of asynchronous and event-based programs using observable sequences[59].

By using RxJava and its RxAndroid extensions a lot of boilerplate code relating to threading can be avoided while also remaining conscious of the Android lifecycle with little effort on the part of the developer.

Figure 4.7 shows one of the most basic examples of how RxJava emits items in an Observable. The arrow line indicates time passing. First the circle is emitted, then the



FIGURE 4.7: RxJava Basic Observable Example[60]

pentagon, then the triangle and finally the thread terminates with an error. The thread can also continue on indefinitely or close without an error.

This is only one of the most basic examples with no other operations being performed on the items emitted. More detailed examples and diagrams can be found at <https://medium.com/@jshvarts/read-marble-diagrams-like-a-pro-3d72934d3ef5>.

The use of RxJava in this project will not be extensive as it is not anticipated for the use of separate threads to occur frequently. This is taking into account that Firebase, the main source of slow operations in this project, handles its own threads internally.

4.1.1.6 Retrofit

Retrofit is a library designed to turn a HTTP client into a Java interface. Retrofit essentially takes a remote REST API and by providing an interface which outlines the structure of the various request types the REST API uses, Retrofit handles generating the correct code for contacting the remote server with the query URL and all parameters formatted correctly[61].

Retrofit will be playing a relatively small role in this project as it will only be used in interactions with the Ruby server which plays a limited role in this project.

Further information on Retrofit is used can be found at <https://square.github.io/retrofit/>.

4.1.1.7 Butter Knife

Butter Knife is a view binding library designed to reduce the boilerplate code associated with typical Android view binding by generating the necessary UI code at compile time[62].

When a view exists but not in the same scope as the logic controller it is attached to, Android Studio will not recognise this mistake and allow a successful build despite the fact it won't work. By using Butter Knife an error is thrown at compile time which highlights this issue.

Butter Knife will be used quite extensively in this project as it will be in use in every screen presented to the user to bind all UI elements.

Further details on Butter Knife can be found at <https://jakewharton.github.io/butterknife/>.

4.1.1.8 Glide

Glide is an image loading and caching library for Android which handles all loading asynchronously. By using this library there is a massive reduction in boilerplate code and the developer can avoid working with difficult issues in regard to threading and background operations, as well as massively improving efficiency on the time-consuming and resource heavy task of image loading in Android.

Glide will not be used very much in this project as it is anticipated to be used mainly when loading images which are stored online, such as user profile pictures. All other images will be either packaged with the app or defined as vector images using XML.

Further details on the usage of Glide can be found at <https://github.com/bumptech/glide/blob/master/README.md>.

4.1.1.9 Moshi

Moshi is a library for converting JSON responses to POJO and vice versa.[63]. It holds some significant improvements over similar libraries such as GSON in regards to speed and Kotlin support[64].

Moshi will play a relatively small part in this project as it will be used only for interactions with the Ruby backend server and, as stated previously, this server will have a limited role in comparison to the Firebase backend.

4.1.2 Firebase Technologies Used

While Firebase is technically a single technology provider, it encompasses many different types of features so it is still worth outlining here which of these will be used in this project.



FIGURE 4.8: Firebase Logo[65]

4.1.2.1 Firebase Authentication

Firebase Authentication provides a drop-in UI solution for including a complete register and login flow in a mobile app. There are a lot of built in options such as allowing sign-in through Twitter, Google and other sites but for this project the default email and password sign-in will be used.

A detailed overview of Firebase Authentication can be found at <https://firebase.google.com/docs/auth>.

Firebase Authentication will form the one and only register and login procedure for users in this project. The Firebase UI for registering or logging in will be the first screen presented to the user on startup of the app. Once signed in, it is expected that this screen will rarely, if ever, be used again by the user. As with most services requiring authentication it is expected that once signed in a user will not sign out again for the sake of convenience.

4.1.2.2 Firebase Cloud Firestore

Firebase Cloud Firestore is a realtime NoSQL cloud based storage solution which allows for many clients to quickly access up-to-date information.

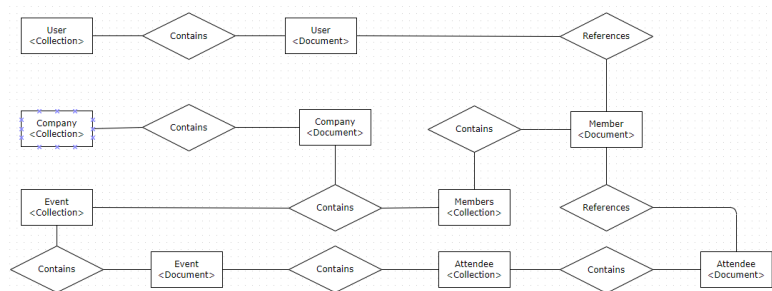


FIGURE 4.9: Proposed Database Design

In Figure 4.9 is the proposed design of the Firestore NoSQL database.

The structure of the Firestore database is a document-based NoSQL database, each document is made up of key-value pairs for the data and subcollections which contain further collections of documents.

In this diagram we can see collections contain documents and documents, in turn, may contain collections. Unfortunately, due to the lack of a strictly enforced data structure as with SQL databases, many modelling tools for NoSQL databases do not seem to contain the tools for detailing key-value pairs of data.

The two main collections making up this database will be 'User' and 'Company'. These then contain documents of an individual User or Company in turn. While the User collection remains simplistic, the Company collection is more complicated due to the data points associated with a company in this project.

Each Company document will contain the subcollections of 'Event' and 'Members'. Each Member document will reference an already existing User document. Each Event document will contain a further subcollection of 'Attendees' with each Attendee document referencing a company member.

Aside from any subcollections each document will also contain any other required information for its data. For example an Event document will contain event name, location and so forth on top of its Attendees subcollection.

The exact features of Firestore are far too extensive to list here but details can be viewed at <https://firebase.google.com/docs/database/rtdb-vs-firestore>.

Firestore will be used extensively in almost every facet of this project as it will need to be accessed by the mobile app for every display of information as well as by the Ruby server to allow for the required database access.

4.1.2.3 Firebase Cloud Messaging

FCM is a component of Firebase which allows for sending messages reliably to connected client devices. In the case of mobile devices, these are almost always then shown as notifications.

Further details on FCM can be found at <https://firebase.google.com/docs/cloud-messaging>.

FCM will be used for all instances of notification delivery in this project.

4.1.3 Ruby Server Technologies Used

4.1.3.1 Ruby on Rails



FIGURE 4.10: Ruby on Rails Logo[66]

Ruby on Rails is a library for the Ruby language which provides an easy to setup environment and quick development of applications[67].

As this project will be using the Ruby server in a limited number of cases and purely as an API, Ruby on Rails is chosen as it provides built-in tools for creating exactly this kind of setup quickly[68].

Rails applications also come with a default database using SQLite. In this project the data storage will be handled primarily by Firebase, however as the ruby server is required to generate a human readable ID for each new company created, this SQLite database will be used as a convenient way to auto-generate an ID for new companies. This will then be stored by Firebase with all other data[69].

Rails also gives the benefit of strictly enforcing an MVC code architecture which in turn tends to produce more robust code[70].

Further details on Ruby on Rails can be found at <https://rubyonrails.org/>. Details on specifically the API features of Rails can be found at https://guides.rubyonrails.org/api_app.html.

Rails will form the entirety of the Ruby server in this project.

4.1.4 How These Technologies Are Used

The proposed use of the outlined technologies and the interactions between different parts of the system are as follows:

The mobile app code structure will be using Clean Architecture as detailed in section 4.1.1.2. In an effort to make code as reusable as possible and to increase adherence to the SOLID principles of Single Responsibility and Dependency Inversion, Dagger2 will be used to provide instances of all possible classes throughout the project.

The mobile app will need to have means of communication with both Firebase and the Ruby server. Firebase includes all networking and threading operations as part of its libraries but connection with the Ruby server will need to be handled by the developer. In this scenario RxJava, Retrofit and Moshi are all used. Retrofit is the means of connecting to the server and receiving a response, Moshi will parse that response into usable objects in code and RxJava will handle the threading operations which come with networking functionality.

As listed in section 4.1.1 other libraries such as Glide and the AndroidX suite will be used. The AndroidX libraries will be used extensively for various pieces of functionality throughout the app. Glide will be used in far more limited cases. In order to avoid too much repetition here on what functionality these libraries offer details can be found above in section 4.1.1 giving an overview of their benefits.

The Firebase backend will provide most of the server side functionality for this project, most heavily used will be Firebase Cloud Firestore for storing user data.

The separate Firebase components listed in section 4.1.2 will not actually have any direct interaction with one another in this project.

Most interactions with the Firebase backend will come from the mobile app through the use of the various Firebase libraries. Firebase Cloud Storage will be in charge of providing the data on events, people and companies. Firebase authentication will provide the means of new users registering and logging into the mobile app and FCM will provide the mechanism for sending notifications to users at the appropriate times, such as for event invitations, reminders and so forth.

Finally the Ruby server will provide some limited functionality which Firebase does not support currently. Within the scope of this part of the system is creating a new company ID and scheduling when notifications are to be sent.

The Ruby server is in charge of creating a new company ID as it provides a convenient way of generating unique human readable IDs for each company whereas Firebase Cloud Storage does not. By default Ruby on Rails comes with an SQLite database. This database will not be used for data storage but by creating a new entry for each new company the auto-generated ID created by the database gives a simple unique value which can then be passed onto Firebase.

Firebase does not currently support sending a notification at a specific time but by including the Ruby server in the notification process a workaround can be found. When some action is taken which results in notifications being sent sometime in the future the Ruby server can schedule when to connect to Firebase and send the notification.

For these purposes there will also be a direct means of interaction between the Ruby server and the Firebase backend.

4.2 Risk Assessment

4.2.1 Risk No. 1

Description

Firebase backend goes down.

Consequences

Newly entered information from Firestore will not be made available to users.

Severity

Critical

Possibility of Occurring

Rare

Risk Mitigations

Unfortunately not much can be done in the event that Firestore somehow becomes unresponsive as this is the backbone of the app. However as Firebase is owned and backed by Google the odds of the service becoming unresponsive are incredibly rare.

Besides this the Firestore library for Android comes with caching functionality built in. In such an event users will still be able to view the latest cached information on their

device, also any newly entered data will be cached until such time as it can be persisted to the cloud based backend.

4.2.2 Risk No. 2

Description

Unable to complete sprints due to time constraints

Consequences

In such a case the app will be missing planned features and bugs may not be fixed.

Severity

Major

Probability of Occurring

Probable

Risk Mitigations

The chances of this occurring in the development of the Ruby backend or the Firebase setup is very unlikely given the limited functionality and ease of setup of these two parts of this project. This risk is therefore almost entirely a risk for the mobile app development.

The only mitigation in this scenario is to consistently follow the sprint outline devised in section 4.4 and, when possible, get to work on the next sprint early.

By doing this and strictly following the concept of the MVP of this app, the project development phase can still end with an app which provides a high degree of value despite missing some planned features/functionality.

4.2.3 Risk No. 3

Description

Functionality differences between Android OS versions

Consequences

Users on different Android OS versions, particularly older versions, may experience non-typical or unexpected behaviour when using the app.

Severity

Minor

Possibility of Occurring

Probable

Risk Mitigations

While this is an issue known to occur in Android apps it is incredibly rare that this will affect the actual logic of the app. What is far more common is that UI elements are affected by this issue in that styling may not be applied correctly and thus UI elements which employ certain known troublesome classes can appear differently depending on the OS version.

While this risk is probable in its likelihood to happen, the chances of it adversely affecting a user and preventing them from using the app is practically non-existent.

These issues most commonly occur on devices running Android OS version 22 or lower. Therefore to mitigate this issue, and to allow reduction in handling various edge cases which come with older Android versions, this project will set Android version 23 as the minimum OS version the app can be used on.

Furthermore there are also well known workarounds for these styling issues which should be employed consistently over the standard classes provided for styling which in turn will yield consistent results across devices.

4.3 Methodology

4.3.1 Learning New Technologies

In regard to learning new technologies I am already somewhat well experienced in working with Android including each of the libraries listed in section 4.1.1. I have also used each of the Firebase features listed in section 4.1.2 quite extensively in the past.

The main part of this project implementation requiring practice and research is the use of Ruby on Rails for the server backend. Due to the relatively limited scope of the Ruby server and the expected time constraints by the time the implementation phase of this project comes about, the plan for learning how to use this technology will focus only on learning what is absolutely needed rather than obtaining a broad knowledge.

I have already taken steps in reviewing how to setup a Ruby on Rails server which acts only as an API interface.

From this initial learning I believe the Ruby aspect of this project will be relatively short as Rails provides a quick setup tool for this exact use case. Further to this the Firebase documentation provides various examples of using Firebase in Ruby. Given these factors the Ruby part of this project is expected to be short and, relative to everything else, quite simple.

4.3.2 Project Management Approach

The development approach to be taken in this project will be Scrum with a sprint time of 2 weeks.

4.4 Implementation Plan Schedule

Starting Date	Working On
January 6th	<ul style="list-style-type: none">• Setup FirebaseAuth• Setup Firebase Cloud Storage• Setup FCM• Setup clean architecture mobile project structure• Setup Ruby on Rails for Ruby backend

January 20th	<ul style="list-style-type: none">• Add FirebaseAuth to mobile app to allow for sign up-/sign in• Create Ruby API for adding a new company• Add new company to Firebase Cloud Storage from Ruby server• Create mobile frontend for adding a new company• Create mobile frontend for users to join a new company• Create mobile frontend for users to view the events they are invited to
February 3rd	<ul style="list-style-type: none">• Create mobile frontend for an admin to create a new event• An admin can set event-specific user details• Allow users to view the list of members of their company
February 17th	<ul style="list-style-type: none">• Allow users to respond to an event invite• A user will receive a notification when invited to an event• A user can respond to event invites directly from the notification• A user will receive a notification for an upcoming event

March 2nd	<ul style="list-style-type: none"> • Update event details screen to show event location on a map • Implement app rating to allow users to give feedback on ease of use and reduction in difficulties experienced • Add mobile frontend to allow a user to view their own profile • Allow a user to view another user's profile
March 16th	<ul style="list-style-type: none"> • Create mobile frontend for admins to view membership requests • Update join company functionality to now make a request rather than immediately join • Allow a user can edit their own profile
April 6th	<ul style="list-style-type: none"> • An admin can set times for event reminders • An admin can make another user an admin

TABLE 4.1: Sprint Schedule

The remainder of the project implementation phase will be dedicated to working on any bug fixes or improvements which should be made to the system. The remaining time is also partially set aside for working on the implementation report.

4.5 Evaluation

As shown in section 4.4, in the sprint beginning February 17th, one of the goals for this sprint is to implement user feedback.

This will be the way in which users can give a rating on the app, the usual 1 - 5 stars, as well as include any comments they have to give. As this app is the only user facing

portion of the project any issues which occur due to problems in the Ruby or Firebase backends will also be reflected in this rating.

The top apps on the Google Play Store tend to have a rating of roughly 4 stars[71]. Quite often 4 stars tends to be the minimum rating which apps aim to achieve therefore the same idea will be applied here. by taking the in-app star rating as a percentage, a score of 80% or more will be taken to mean the project has been successful in achieving its goals.

Despite referencing Google Play Store ratings here the proposed user feedback will be separate. It will follow a very similar concept but by making this an in-app rating, customisations can be made on the type of feedback given. Feedback can be requested on very specific metrics, such as ease of event creation, reduction in organisation difficulties compared to previously used event management systems and so forth. This kind of feedback would prove far more useful than the generic good or bad review which is typically left on the Play Store. This feedback can then also be aggregated in Firebase to more easily allow for any desired data analysis.

4.6 Prototype

This section gives a general idea on the look and functionality of the finished project. A prototype for the Firebase implementation is not included in this section as Firebase is a third party backend implementation which requires a client to work. Therefore the results of data stored by Firebase or notifications sent are shown as part of the mobile app prototype.

All features in the mobile app prototype are shown but some of these will be unavailable if the user is not an admin. These will be highlighted in section 4.6.1 when appropriate.

4.6.1 Mobile App Prototype

Shown in Figure 4.11 is the outline of the Sign In/ Register screen for users to create an account for themselves or sign into an existing account. In the final project this screen will be handled by Firebase Authentication which provides a built-in sign in and register UI.

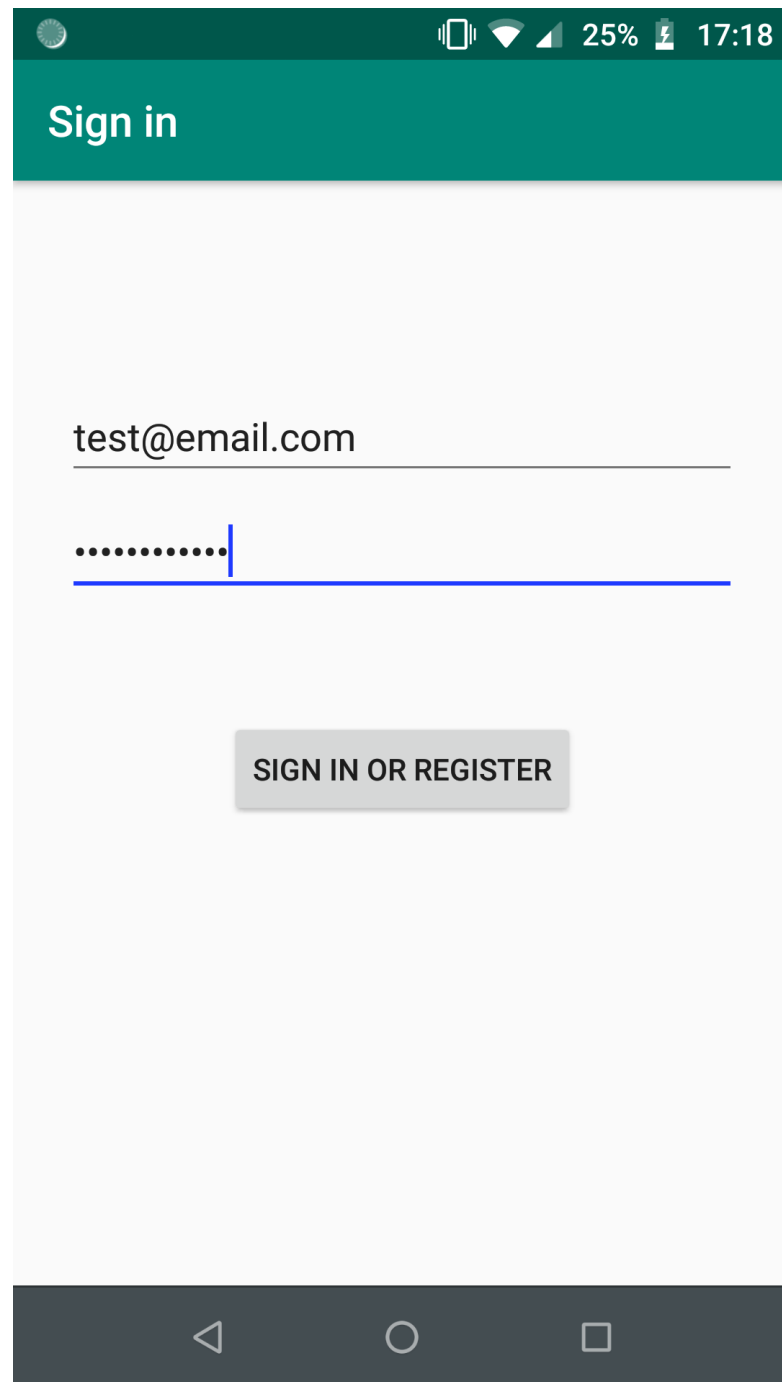


FIGURE 4.11: Mobile App Prototype Sign In/Register Screen

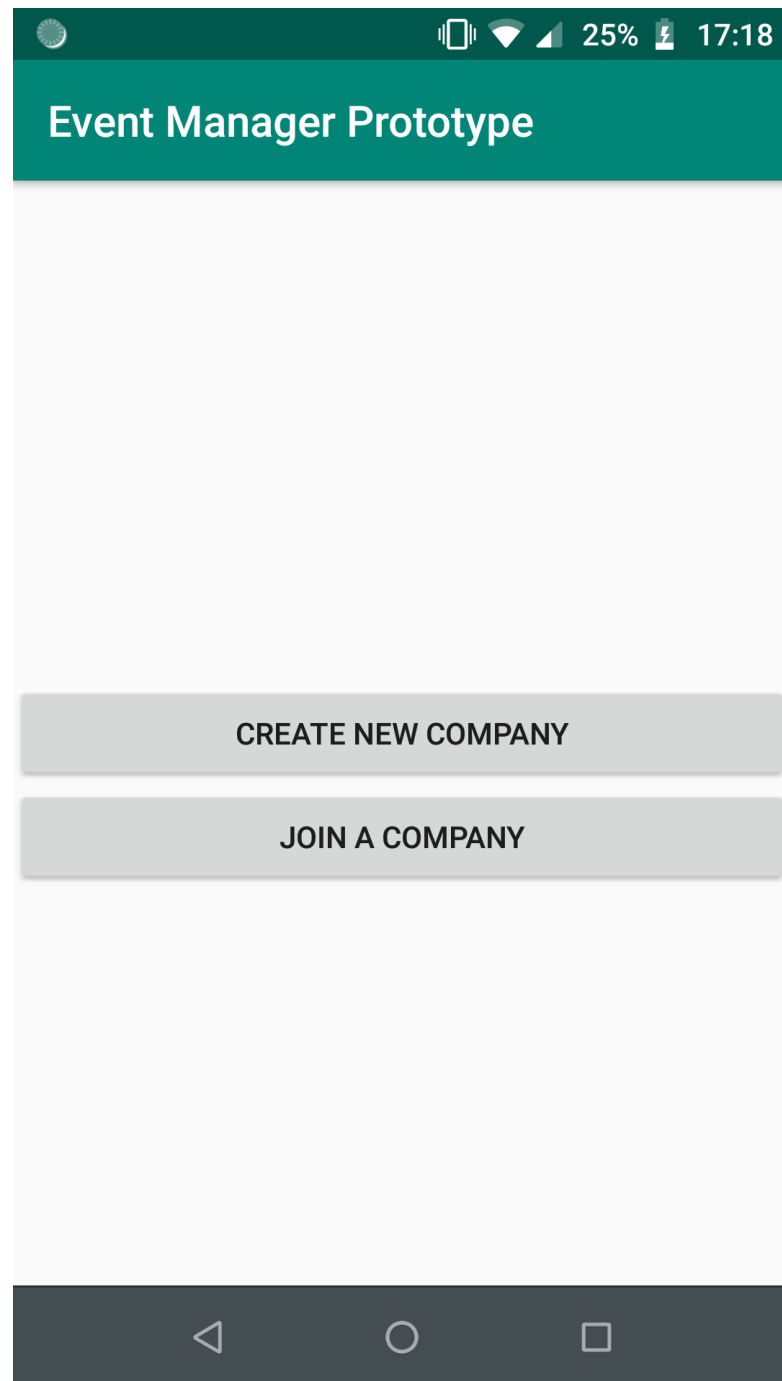


FIGURE 4.12: Mobile App Prototype Create/Join Company Screen

Once a user has created their account they will be presented with the screen shown in Figure 4.12. Here they will be presented with the option of either creating a new company or joining an already existing company.

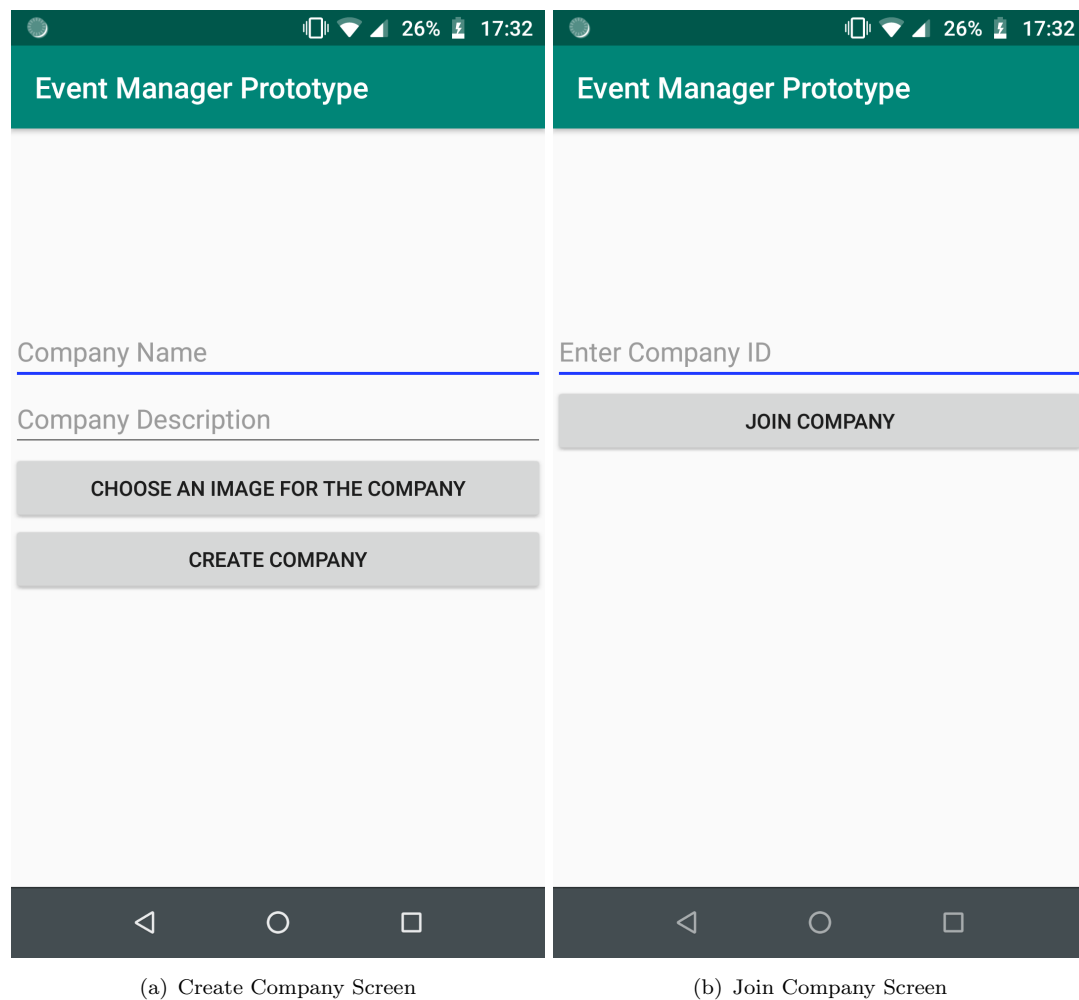


FIGURE 4.13: Mobile App Prototype Create and Join Company Screens

Following on from the options presented in Figure 4.12, in Figure 4.13 is shown the new options presented depending whether a user has chosen to create a new company or join an existing one.

For creating a company the user can enter the company name, a description of the company and choose a company image. When joining a company a user must simply enter the company ID to make a membership request.

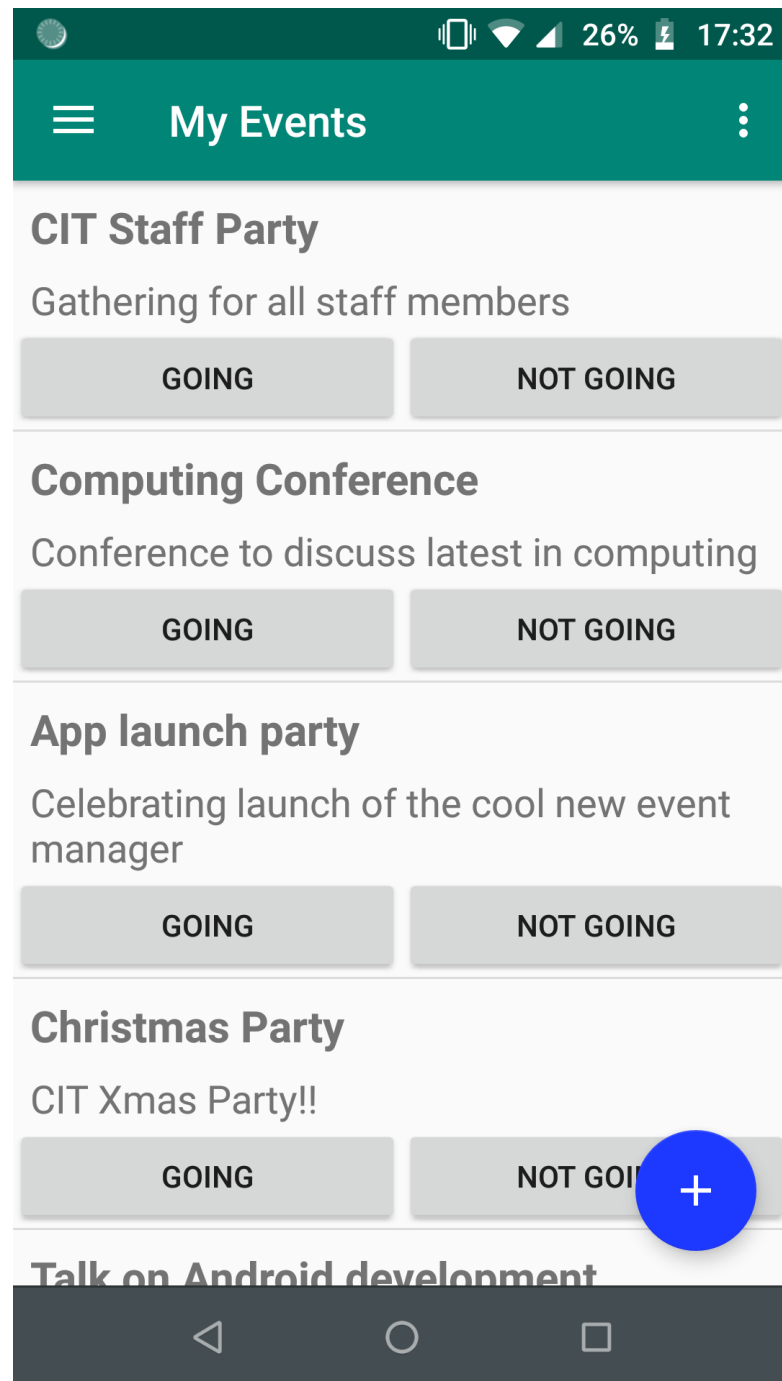


FIGURE 4.14: Mobile App Prototype Event List

After completing the sign in and company creation process a user is able to gain full access to the app's features. First to be shown is the list of events as shown in Figure 4.14. This screen shows the list of events the user is invited to and provides options for responding to those invites.

Also provided in this screen is a means of creating a new event through the blue '+' button in the lower right hand corner. This is a feature available to admins only. For users who are not admins this button will not be shown.

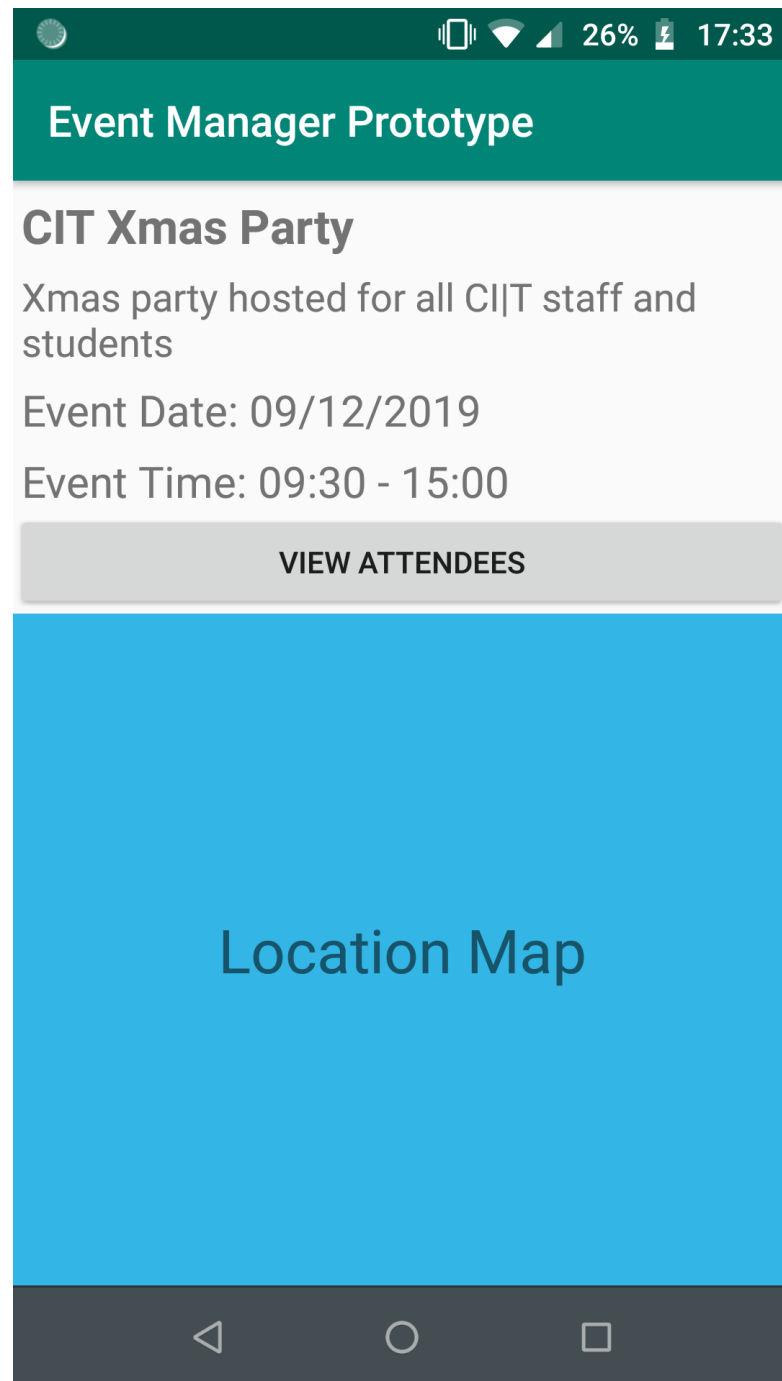


FIGURE 4.15: Mobile App Prototype Event Details

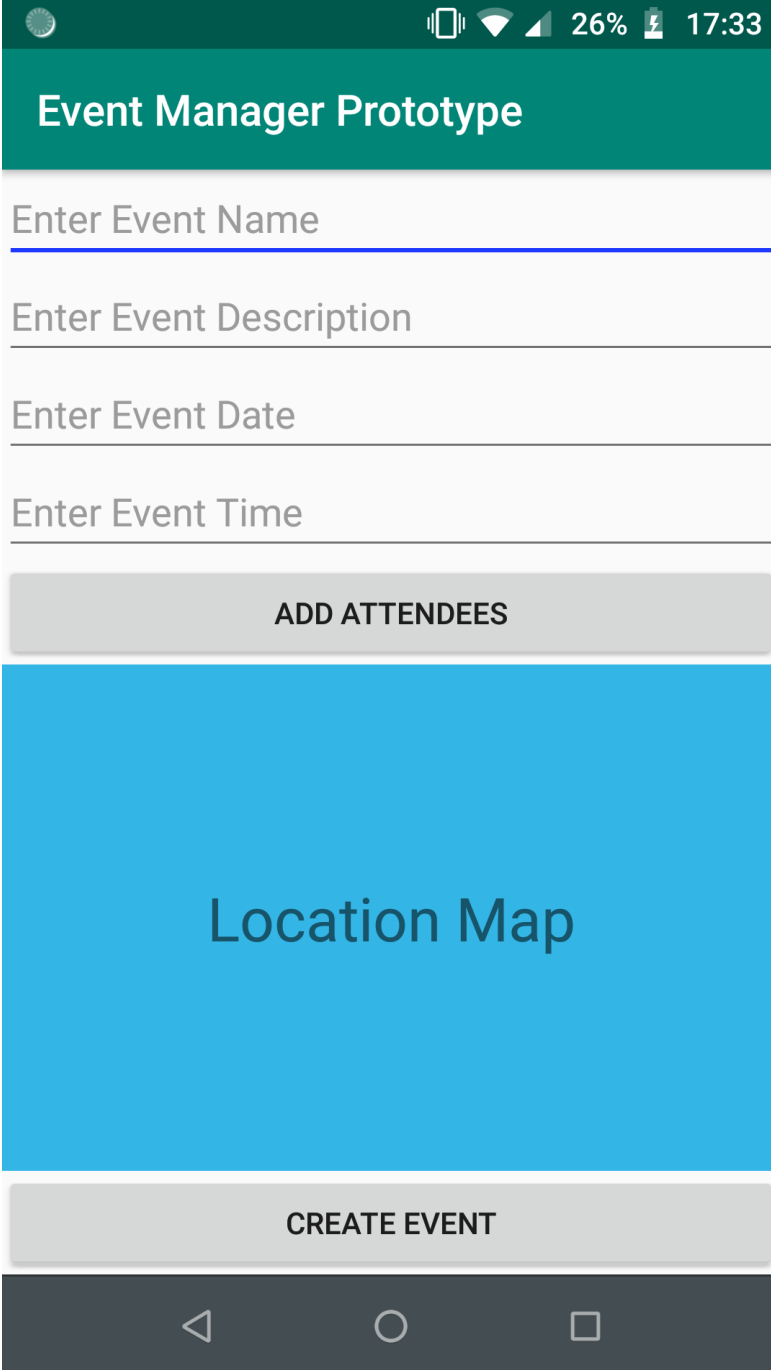
When the user clicks on an event from the events list they will then be provided with further details on that specific event as shown in Figure 4.15. This allows the user to learn about the event in further detail such as viewing other attendees and seeing the event location on a map.



FIGURE 4.16: Mobile App Prototype Attendees List

The list of attendees for an event can be viewed as shown in Figure 4.16.

Also provided on this attendees screen is an option to invite more users, this will be an admin-only option and for other users will not be shown.



The image shows a mobile app prototype for creating an event. At the top is a dark green header with the title "Event Manager Prototype" in white. Below the header are four text input fields with labels: "Enter Event Name", "Enter Event Description", "Enter Event Date", and "Enter Event Time". Each field has a thin blue underline. Below these fields is a grey button labeled "ADD ATTENDEES". Underneath the button is a large blue rectangular area labeled "Location Map". At the bottom of the form is another grey button labeled "CREATE EVENT". The entire screen is framed by a dark grey Android-style navigation bar at the bottom with back, home, and recent apps icons. The top status bar shows a clock icon, signal strength, Wi-Fi, 26% battery, and the time 17:33.

FIGURE 4.17: Mobile App Prototype Create Event Screen

When creating an event the screen shown in Figure 4.17 will be presented. This allows the admin to enter the event information which, in turn, will be shown to attendees as in Figure 4.15. The option to add attendees will lead to the same screen as in Figure 4.16.

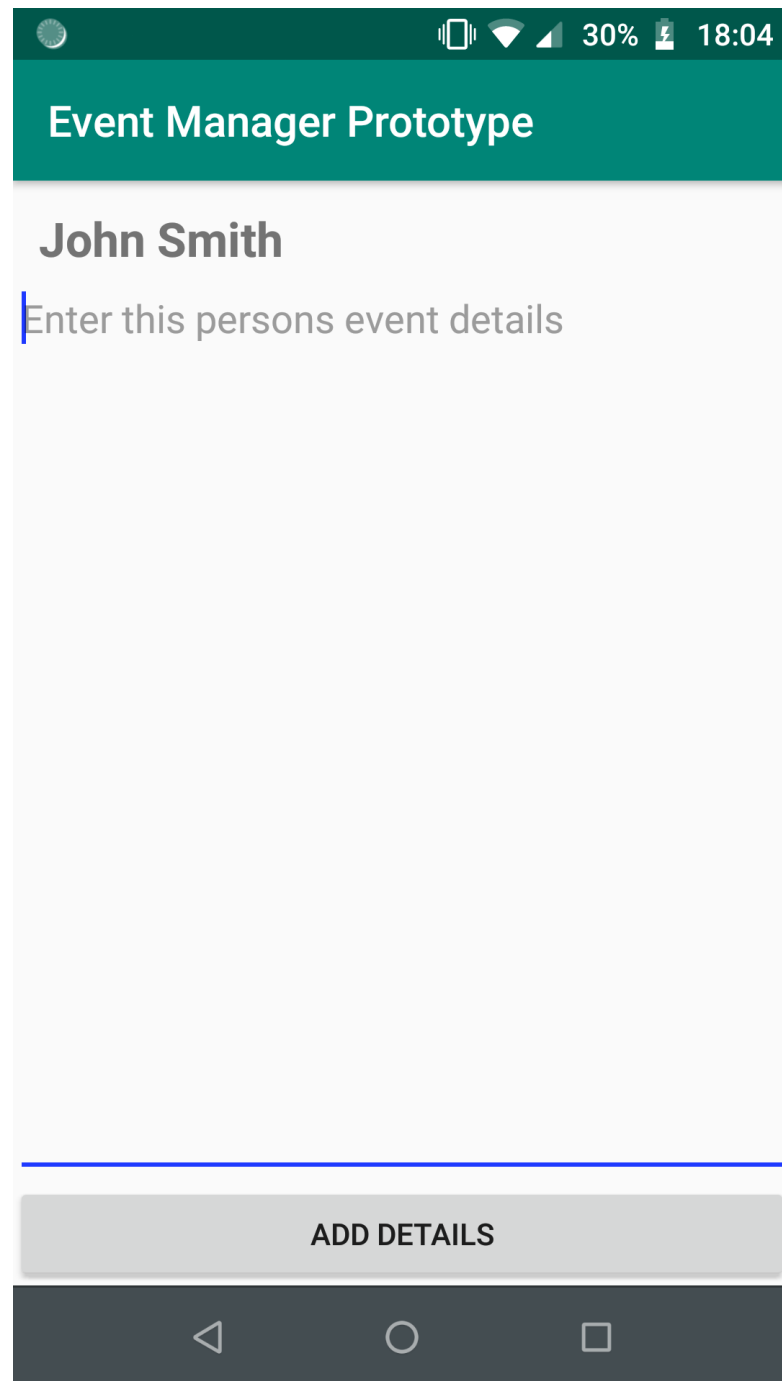


FIGURE 4.18: Mobile App Prototype Add Person Event Details Screen

When on the attendees list screen as shown in Figure 4.16 an admin may choose to add event specific details to that attendee as shown in Figure 4.18. As there are far too many variable options that an event organiser could add it is deemed easiest to allow a free input through a textbox as shown here rather than trying to create a UI which covers all of the many possibilities.

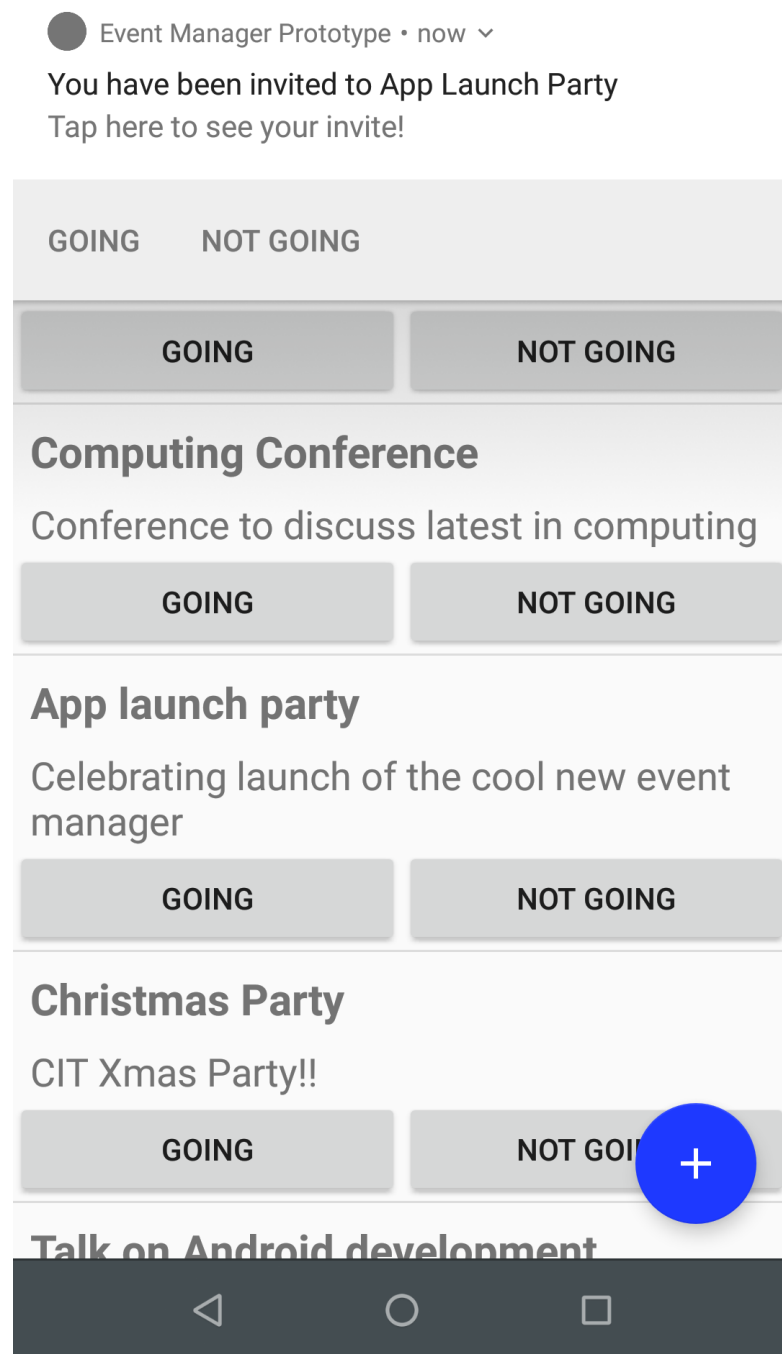


FIGURE 4.19: Mobile App Prototype Event Notification Example

An example of the notifications sent by Firebase when invited to an event is shown in Figure 4.19. A similar notification will be sent when an event the user has responded to as 'Going' is sent shortly before the event starts as a reminder.

Tapping on the main body of this notification will take the user to view the event details as outlined in Figure 4.15. Also included are shortcut actions to quickly respond to an invite.

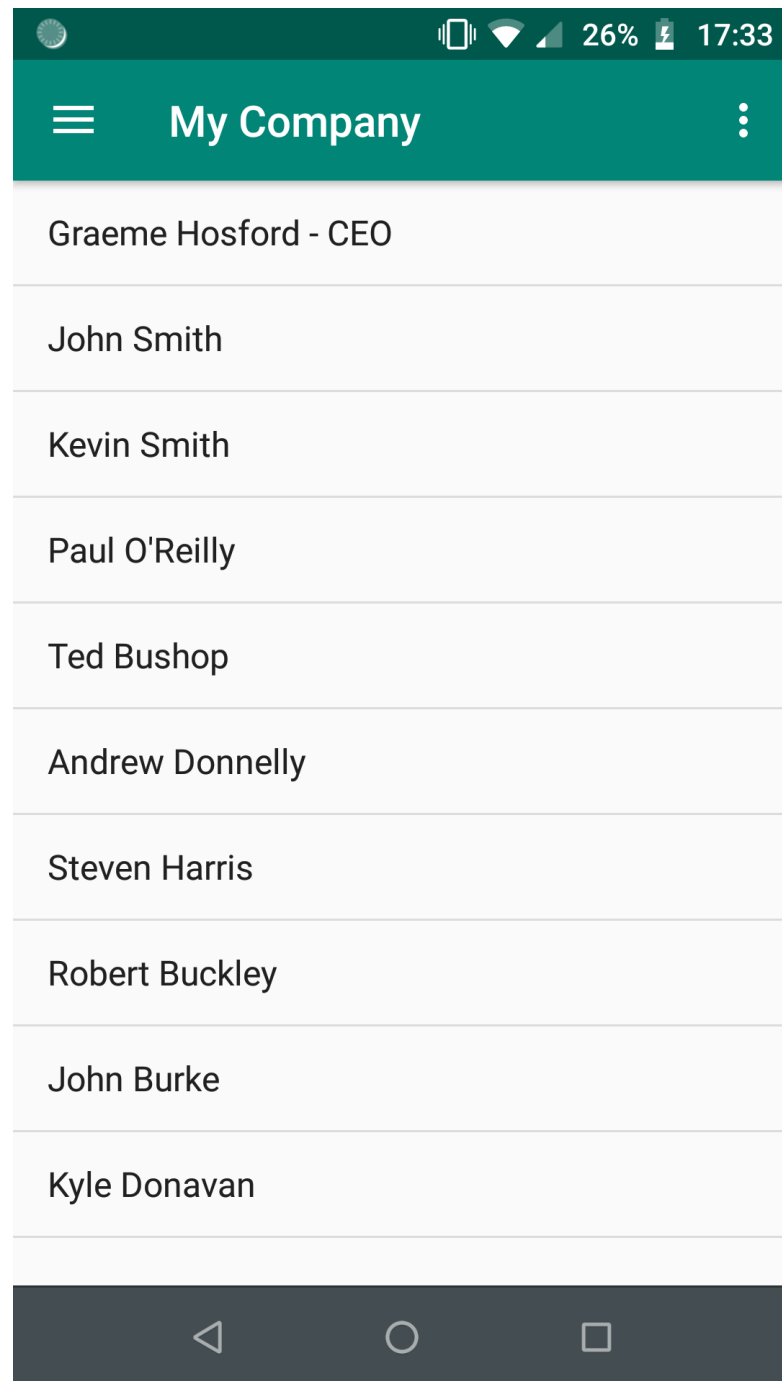


FIGURE 4.20: Mobile App Prototype Create Company Details Screen

As shown in Figure 4.20 the user can view the list of members of their own company.

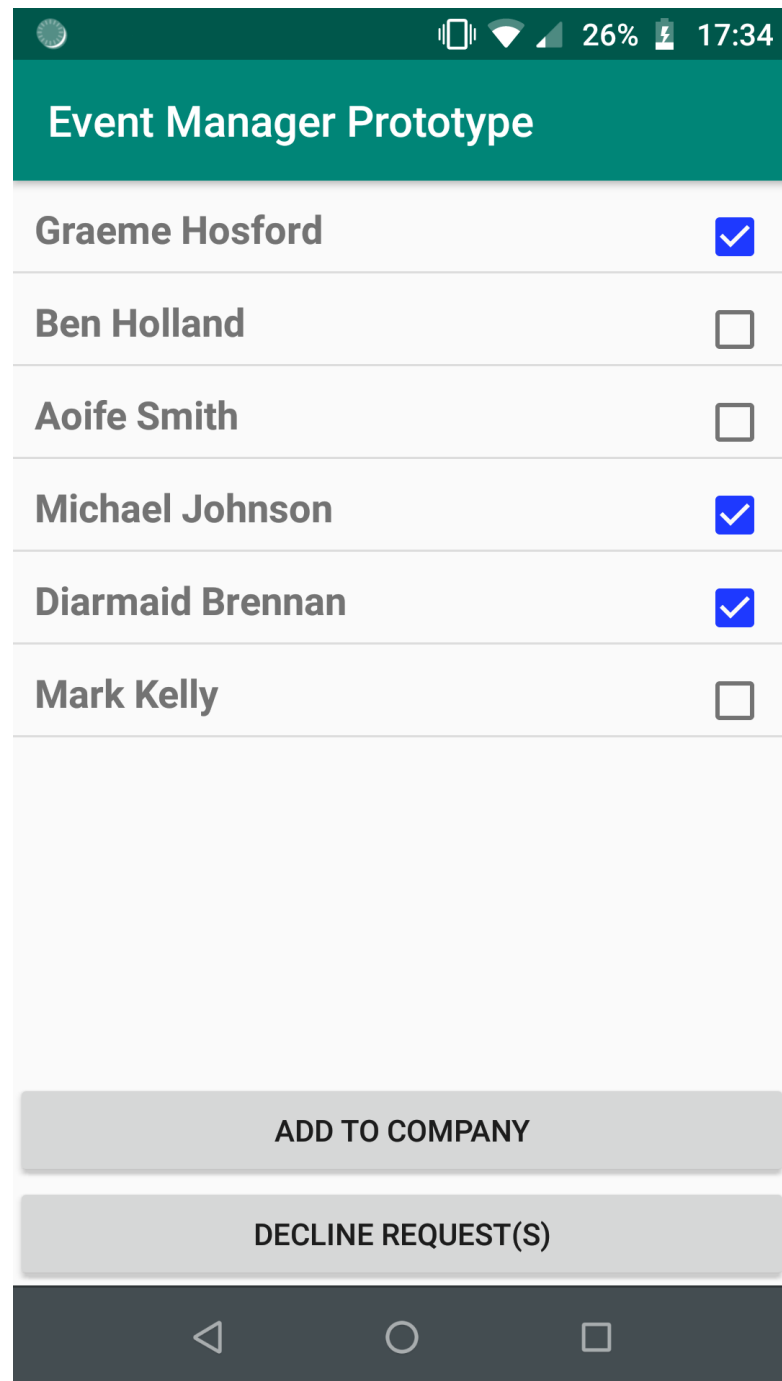


FIGURE 4.21: Mobile App Prototype Company Member Requests Screen

In Figure 4.21 is the admin-only feature of approving or denying member requests for the company. Any users added from this list is able to view the company's events and members, and any user whose request is declined simply has their request removed from this list.

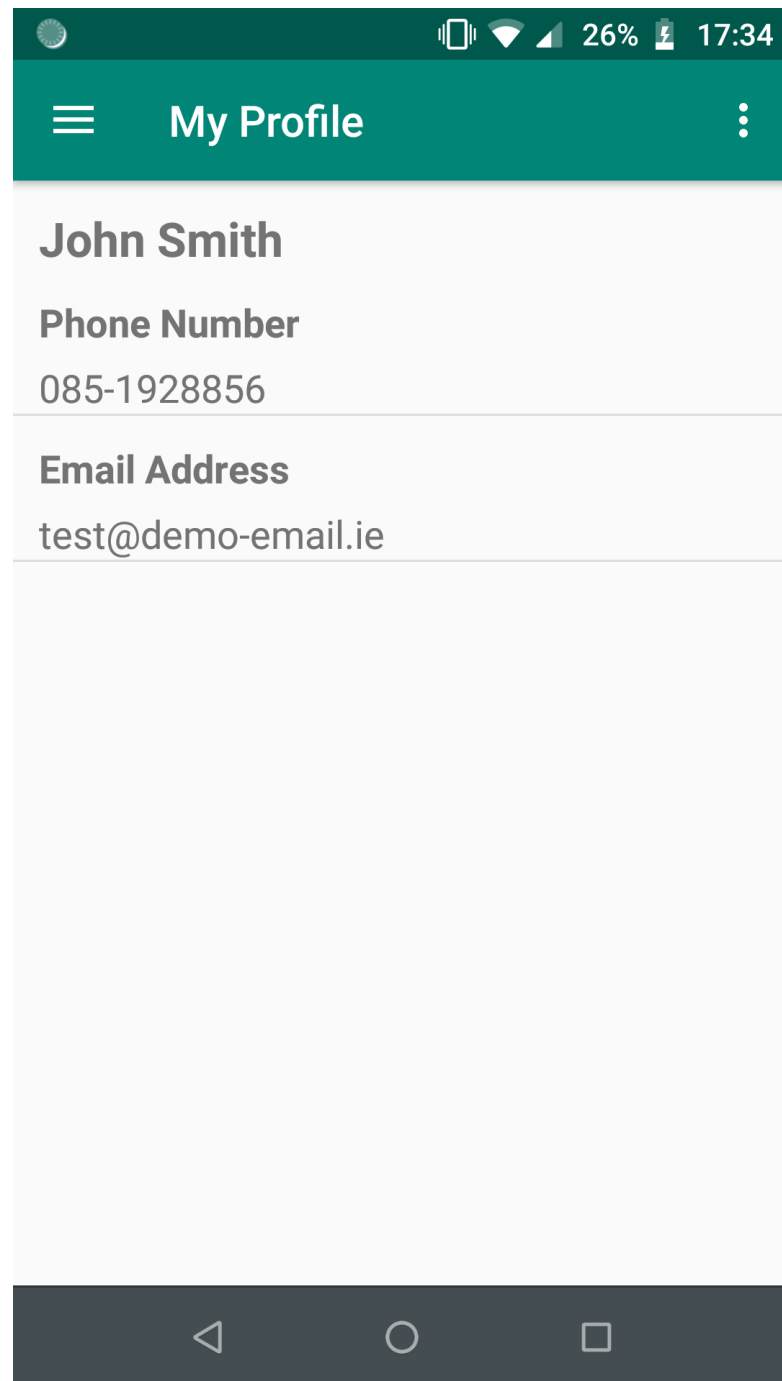


FIGURE 4.22: Mobile App Prototype User Profile Screen

A user profile is shown in Figure 4.22. A user can view their own profile through choosing the option from the main navigation menu or selecting their own name from the list of company members. The profiles of other users can also be accessed through the member list.

When a user is on their own profile they also have the option of editing it to change and update the information shown there.

4.6.2 Ruby Server Prototype

In this section is just the prototype JSON responses for when a new company is created. Unfortunately it was not possible to prototype the notification scheduling here, as again this is mainly Firebase related. Also it was not possible to demonstrate a delayed action. Some prototype output for this however can be seen in Figure 4.19.



FIGURE 4.23: Ruby Server Prototype Create Company Response

Shown in Figure 4.23 is the response generated when a new company has been created by the Ruby on Rails server. The key part of this response is the "id" field. As mentioned before the default implementation of Ruby on Rails will automatically generate a simple human readable ID through its database implementation. This ID is then forever associated with the created company and can be entered by users to request to join a company as outlined in Figure 4.12.

The other fields in this output are considered unnecessary. The company name is included as there must be some piece of data sent to the server to create a database entry and the 'created at' and 'updated at' fields are automatically generated by Rails. All required fields related to the company will instead be saved in Firestore.

Chapter 5

Implementation

5.1 Difficulties Encountered

Throughout this project there were many difficulties encountered. Fortunately each of them was solved and none lead to promised functional requirements in the project being abandoned.

5.1.1 Easy

5.1.1.1 Heroku Database Support

Heroku is used as the server hosting for the Ruby part of this project. As already mentioned, the database used by the Ruby on Rails app is a simple MySQL database. It is not truly used for data storage, merely to generate unique company IDs.

Heroku does not provide built-in support for a MySQL database but instead supports PostgreSQL as its database solution.

While it took a while to find the solution in the Heroku documentation, it was eventually found that Heroku provides a plugin for MySQL support. This allowed me to use the MySQL database I had already created and prevented any need to move to a PostgreSQL database.

5.1.1.2 Android Navigation Component Support

As mentioned in section 4.1.1.4, the AndroidX Navigation Component would be used heavily in this project's mobile app.

It was discovered during project development that this navigation component is missing a vital piece of support for receiving data from another screen.

In the older Android method of navigation there could be a scenario in which a user is prompted to open another screen for the sole purpose of doing a single interaction. The result of this interaction is then passed back to the previous screen. This is commonly seen when a user, using an app A, taps a button, the camera app B opens, the user takes a picture, and the camera app B closes and that picture is passed back to the app A which requested it. In this, the app A is effectively navigating into the totally separate camera app B with a request that a picture be taken. The camera app B then navigates back to app A with the picture data bundled with the navigation request.

The AndroidX Navigation Component does not provide support for this scenario. The library does not currently support navigating with a request that data be given back to the screen which requested the navigation.

To deal with this problem the only solution was to simply use the older method of navigation when such a scenario occurred. This allowed for the desired functionality to be enacted. Unfortunately this also lead to code which effectively uses two versions of navigation in some screens for the sake of achieving a result which is not supported by the newer method.

5.1.1.3 Limited Screen Space

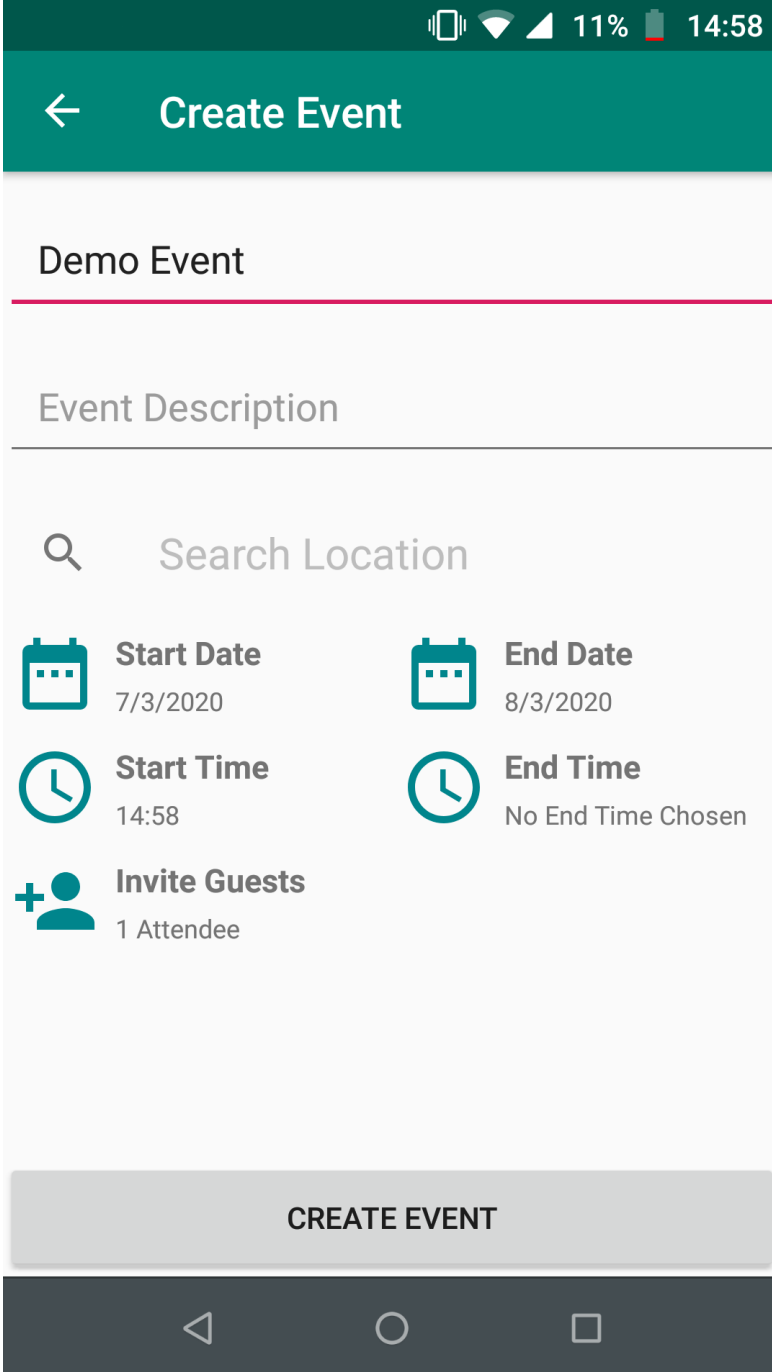
Limited screen space is a potentially huge problem for any mobile app. Thankfully dealing with limited space was only a problem on one specific screen and the solution was easy to implement.

The Figure 5.1 shows the screen in the app which is used to create events. Initially there was too little space to display the chosen event info including the options to change this info. The solution I came up with was to combine the info and the options to change it into one element each.

5.1.1.4 Android FCM Library

The FCM library treats incoming notifications differently depending on if the host app is in the foreground or background.

When the app is in the foreground the developer must take the data included in the incoming notification and define how this data is used to create a notification.



The image shows a mobile application interface for creating an event. At the top, there is a status bar with icons for signal, Wi-Fi, and battery (11%), along with the time 14:58. Below this is a teal header bar with a back arrow and the text "Create Event". The main content area is light gray and contains the following elements: a title "Demo Event" followed by a horizontal line; a section titled "Event Description" followed by another horizontal line; a search bar with a magnifying glass icon and the text "Search Location"; a grid of four event details: "Start Date" (7/3/2020) and "End Date" (8/3/2020) with calendar icons, "Start Time" (14:58) and "End Time" (No End Time Chosen) with clock icons, and "Invite Guests" (1 Attendee) with a plus and person icon; and a large gray button at the bottom labeled "CREATE EVENT". The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps icons.

FIGURE 5.1: Mobile app screen for creating events

When an app is in the background the FCM library parses the included data itself and displays a standard non-modifiable notification to the user.

The only scenario in which an incoming notification can always be handled by the user is if it is not technically a notification message. If the message is instead a data message the FCM library always defers to the user defined code for how to handle it.

The difference between these two types is based on the JSON structure of the message.

For a true notification message the JSON must contain an array named "notification" and this array must contain the two fields "title" and "body". The FCM SDK will then automatically parse this info when the app is in the background in order to display a generic notification.

A data message on the other hand has no such rigid structure to it. There must be an array named "data" but there are no fields which must be included by default. There is instead only user-defined key-value pairs. When a data message is received the FCM SDK always defers to the developer defined code on how to handle the message.

In effect, to get the same notification behaviour in all instances the Firebase Cloud Functions code had to be modified to send a data message instead of a notification message. The app then also needed minor code changes to retrieve its info from the messages data fields rather than its notification fields.

5.1.1.5 Android List State Handling

In Android development lists are implemented using a class called RecyclerView. As the name implies each row or item is recycled when it is no longer needed. This occurs when it is scrolled off screen. The RecyclerView class works by saving each items state and reapplying it when that item is scrolled back on screen.

This approach often leads to small UI issues if each item displayed is mutable in some way. For example if tapping a button were to change the colour of the background. Scrolling that item off screen then back on could show it as having an incorrect background colour as the change which occurred is no longer applied.

Luckily the solution to this is quite easy. Each instance of RecyclerView needs an adapter to determine how it displays its items. Inside this adapter a list of POJO classes can be kept which are used purely to store data for each item. Instead of making changes to an items data directly in the view, these changes should be done by applying methods from this POJO class. While the UI may only have its state saved by the RecyclerView class this list of POJO classes is kept in memory. it is then a simple matter of reapplying the data in these POJO classes to prevent unintended UI changes.

5.1.1.6 Firebase Cloud Functions Parameter Names

In the code used by Firebase Cloud Functions to send a notification to a user, a piece of data called eventId is included to allow the notification to display what event the user has been invited to. As part of the standard Cloud Functions library there is also a

built-in parameter called `eventId` which defines what kind of event lead to the function being called, whether it was a create, update, or delete event and so forth.

Unknown to me at the time the Cloud Functions built-in parameter was the one being used rather than the ID of a company event. This lead to unexpected behaviour from the app notifications not displaying the correct event name or allowing for a response.

The solution was a very simple parameter name change. Having a parameter name specific to your own use case matching the name of a built-in parameter in Cloud Functions is obviously quite a rare case. This made finding that answer slightly difficult. It took about 40 minutes of searching through documentation to find the problem.

5.1.2 Medium

5.1.2.1 Ruby Development Environment Setup

The setup for the Ruby development environment was the first, and probably most time consuming, difficulty encountered in this project. My development machine is a Windows laptop. Ruby provides an installer for setup similarly to any other software setup on Windows. Unfortunately this setup had a lot of problems which prevented it from being installed correctly. It was then, of course, unusable.

Approximately 6 hours was spent trying new solutions that were found online to no avail. Eventually it was only by talking to a fellow student who has more Ruby experience than me, that I was informed the Ruby installer for Windows is widely considered to be unreliable.

Based on this other student's advice I installed Ubuntu as a secondary OS on my laptop and performed the ruby setup on that. It worked flawlessly on the first attempt and then I could finally work in Ruby but it took an entire 8 hour day of work to get to this point.

5.1.2.2 Mobile App Architecture Setup

Further difficulty was encountered in setting up the architecture for the mobile app. As shown in Figure 4.3 the mobile architecture is split into layers of presentation, business and data. Each layer can only interact with the one directly below it. Presentation interacts with business and business interacts with data. No direct communication in the other direction is allowed. This is achieved using the Android specific module structure also outlined in Figure 4.3.

Unfortunately when creating the module structure outlined in Figure 4.3 and as given at <https://github.com/Teamwork/android-clean-architecture>, build errors occurred due to the business layer being unable to interact with the data layer. My attempt to fix this solved this issue but created a new one with the DI framework now not able to create its build graph to inject dependencies.

At this point quite a lot of time had been lost to dealing with module structure problems so I decided to go with a single module setup and create the various clean architecture layers in the package structure. In the end this gives effectively the same setup but allows for the possibility of layers to communicate which should not as there is now no module barrier separating them.

5.1.2.3 Ruby Notification Support

One of the biggest issues I encountered in the course of this project was setting up support for the Ruby on Rails app to contact the FCM service and have a notification sent to the user's device.

I already had Ruby code for this exact use case I had developed about a year ago which worked at the time. I used this code, updated the access keys it used for this new project and it simply would not work.

A few hours were spent looking into documentation and double and triple checking code and the access tokens for the project to no avail. I could find no solution online for why my code which worked a year previously no longer did.

I also attempted to use a third party Ruby library for FCM support (<https://github.com/spacialdb/fcm>) which also did not solve the issue. I found quite a few answers to my problem online stating the best solution was to use this library but it got me nowhere closer to a solution.

On one of the documentation pages which details how to use Ruby to send a notification through Firebase there is a warning that the method I was using, and which the library I tried to use also implements, will soon become deprecated. This warning did not include any date I could find for when support would be stopped but the warning did clearly state the method was deprecated but still supported. I still have no conclusive answer but my best guess is that support for this method was stopped at some point and the warning in the documentation was simply not updated. I cannot be 100% certain about this but it's the only answer I can think of.

In the end the solution to this problem ended up being simpler than the initial Ruby method. By using Firebase Cloud Functions I was able to create a small NodeJS script

which is applied as a listener to the Firebase Firestore database. When a new event is created this piece of code is called. The invited people each then have a notification sent to them as they should.

5.2 Actual Solution Approach

Throughout the implementation of this project new challenges arising lead to some changes to the original solution approach outlined in Chapter 4. These changes are split into the following sections:

1. Solution Architecture
2. Use Cases
3. Risk Assessment
4. Development Methodology
5. Implementation Schedule
6. Evaluation
7. Prototype

5.2.1 Solution Architecture

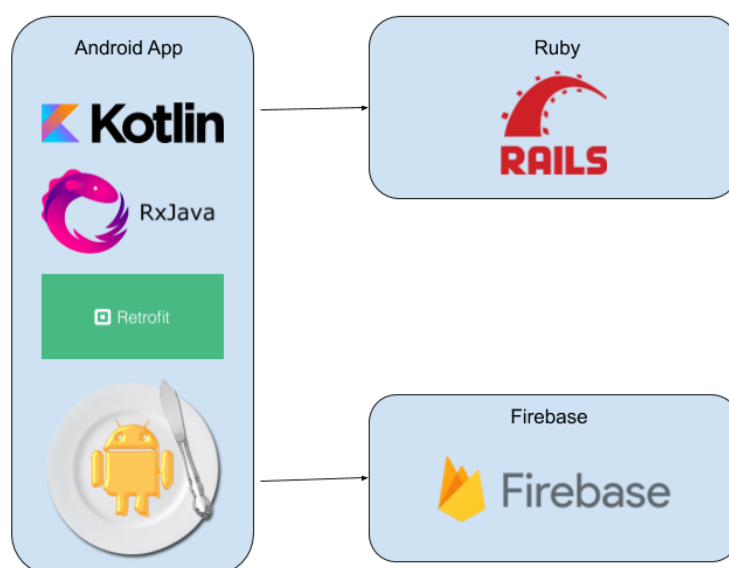


FIGURE 5.2: Final System Architecture Implementation

My overall system architecture remained almost entirely unchanged from the proposed solution outlined in Section 4.1. The only difference is the Ruby server no longer communicates directly with the Firebase backend. This occurred as the original plan to have the Ruby server support sending notifications became unfeasible as already outlined. Another Firebase library, Firebase Cloud Functions, instead filled in that functionality.

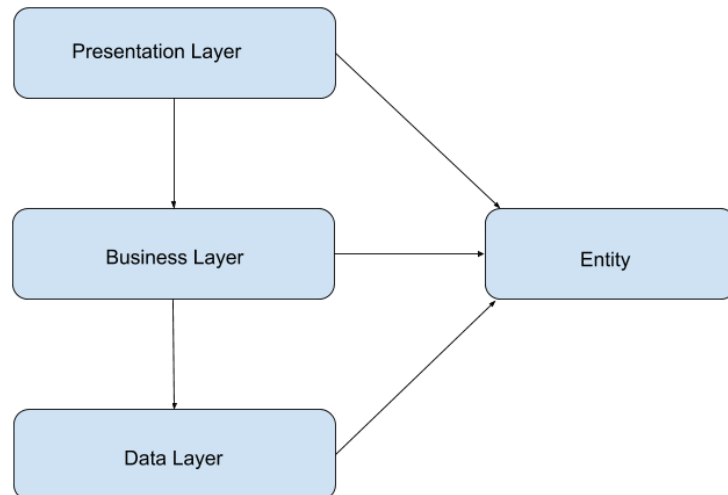


FIGURE 5.3: Mobile app package structure

As discussed in Section 4.1.1.2 the Android app in this project would be using clean architecture as its code structure. While I did achieve this, there were some serious difficulties with the module setup.

As discussed in Section 5.1.2.2 I encountered plenty of build errors in attempting to setup the module structure of the project. Perhaps these could have been solved with more time invested but these errors were beginning to eat into time I had planned for feature development. Instead of using the module setup laid out in Figure 4.3 I used a more basic approach as shown in Figure 5.3. The same result was achieved but some potential benefits were lost as mentioned in Section 5.1.2.2.

Aside from these two points the solution architecture remained almost exactly as initially envisioned. The Ruby server does not handle notifications as originally planned but this turned out to be a good thing in the end. Notifications are now handled through Firebase Cloud Functions which integrate into Firebase much nicer. So overall while this was a change from the original plan it ended up being a minor one which lead to a much better overall solution.

5.2.2 Use Cases

The existing use cases for this project stayed largely the same as when they were first envisioned. There was quite a few new use cases added as new functionality was thought of. However not all of these new use cases were implemented due to time constraints. The exact status of each use case is detailed in Sections 6.1.1 and 7.4.

5.2.3 Risk Assessment

The only update needed for the risk assessment is that the hosting service used for the Ruby server may experience downtime, which did occur during project development. This lead to being unable to test Ruby functionality for about 2 hours at one point.

5.2.4 Development Methodology

As mentioned previously the plan for development in this project was to use a Scrum approach with 2 week sprints. While I did keep with this for the entire semester, the amount of work accomplished in each sprint was almost immediately less than initially planned.

This setback occurred due to the potential time constraint mentioned in the risk assessment. Despite being an anticipated risk the problem was far worse than expected with other assignments needing to take precedence at certain times. This problem was then made worse near the end of the semester due to the COVID-19 health pandemic causing cancellation of college classes and final exams being replaced by new unplanned assignments.

In my initial implementation plan, each sprint contained 2 - 4 user stories I had hoped to complete. This plan had to be abandoned almost immediately due to other assignment work. I typically ended up only managing to complete 2 stories per 2 week sprint.

This new approach worked well enough but lead to planned features being implemented later than planned or not at all.

5.2.5 Implementation Schedule

As already mentioned in the Development Methodology section, the implementation schedule worked out in Section 4.4 had to be quickly abandoned in favour of doing lighter project work to prioritise other assignments as needed.

The addition of new features and user stories throughout the semester also helped to exacerbate this issue. These new user stories would have been great additions to the project but trying to find time for them in an already crowded semester was near impossible.

5.2.6 Evaluation

The evaluation plan for this project is the one aspect of it which changed entirely.

My initial evaluation plan was to take feedback within the app and an average score of over 80% would be considered a success. This plan has some serious holes in it, such as where this feedback would be coming from being the most obvious. Feedback from myself and friends of mine I got to use the app would obviously be totally biased and therefore lead to incorrect conclusions for what this project achieved.

Upon further reflection, and realising my major error here, I decided the best form of evaluation is instead a metric of how many of the functional and non-functional requirements I achieved.

5.2.7 Prototype

In Section 4.6 I showcased a prototype of the mobile app and Ruby server functionality I created as a means of showing a basic UI.

The Ruby server prototype stayed the same in the final implementation due to its very minor piece of functionality in the overall project. However, the UI and functionality for the mobile app changed dramatically compared to the prototype.

Outlined below are the comparisons between the prototype and final implementation version of the app. In order to avoid filling the document with unnecessary images only the main screens which contain major UI changes will be included:

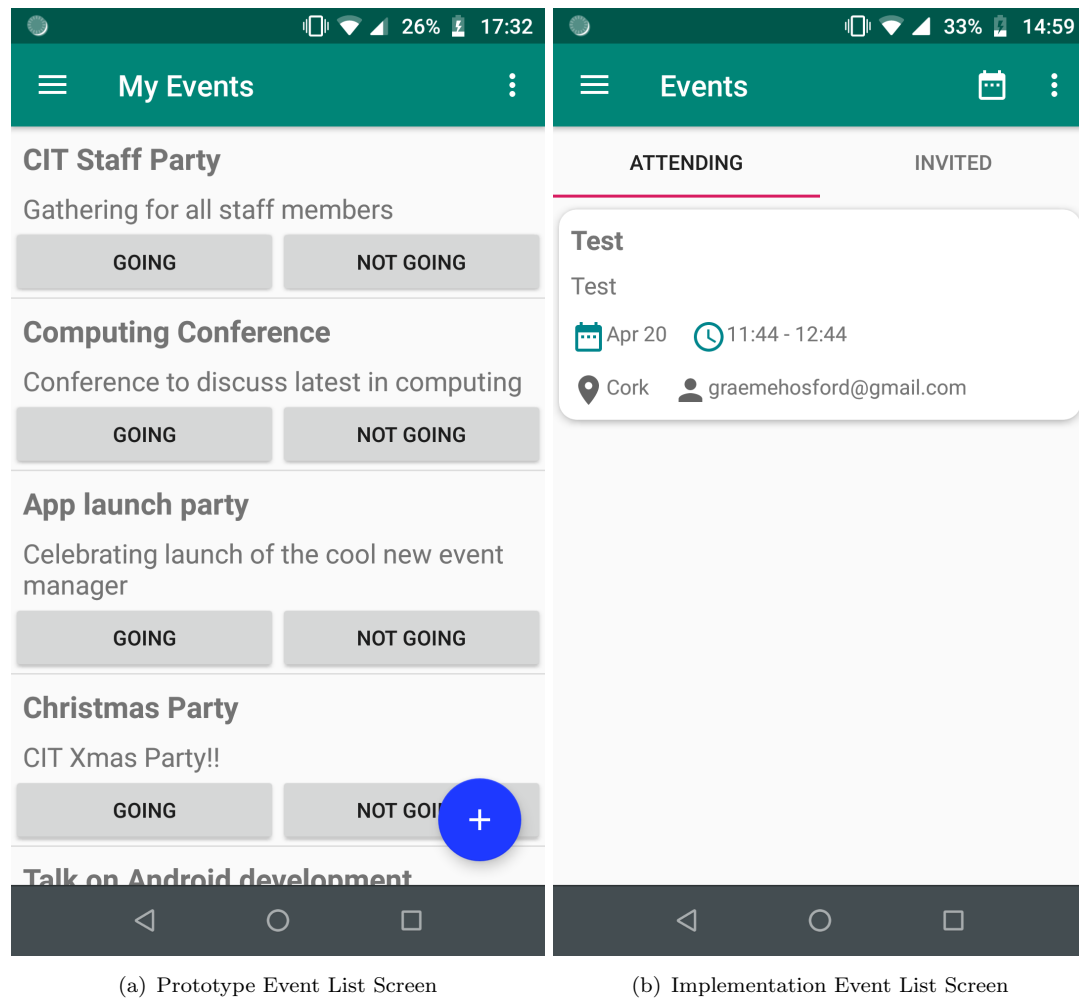


FIGURE 5.4: Mobile App Prototype and Implementation versions of Event List Screen

Aside from the obvious UI differences between each row of the event list in Figure 5.4, the implementation approach also split up the events based on whether the user is attending the event or invited but not yet responded. The user can also tap the calendar icon in the top right corner to view their events on a calendar view. They can also open the options menu by tapping the 3 dot menu in the top right corner and choose to view the events they have created.

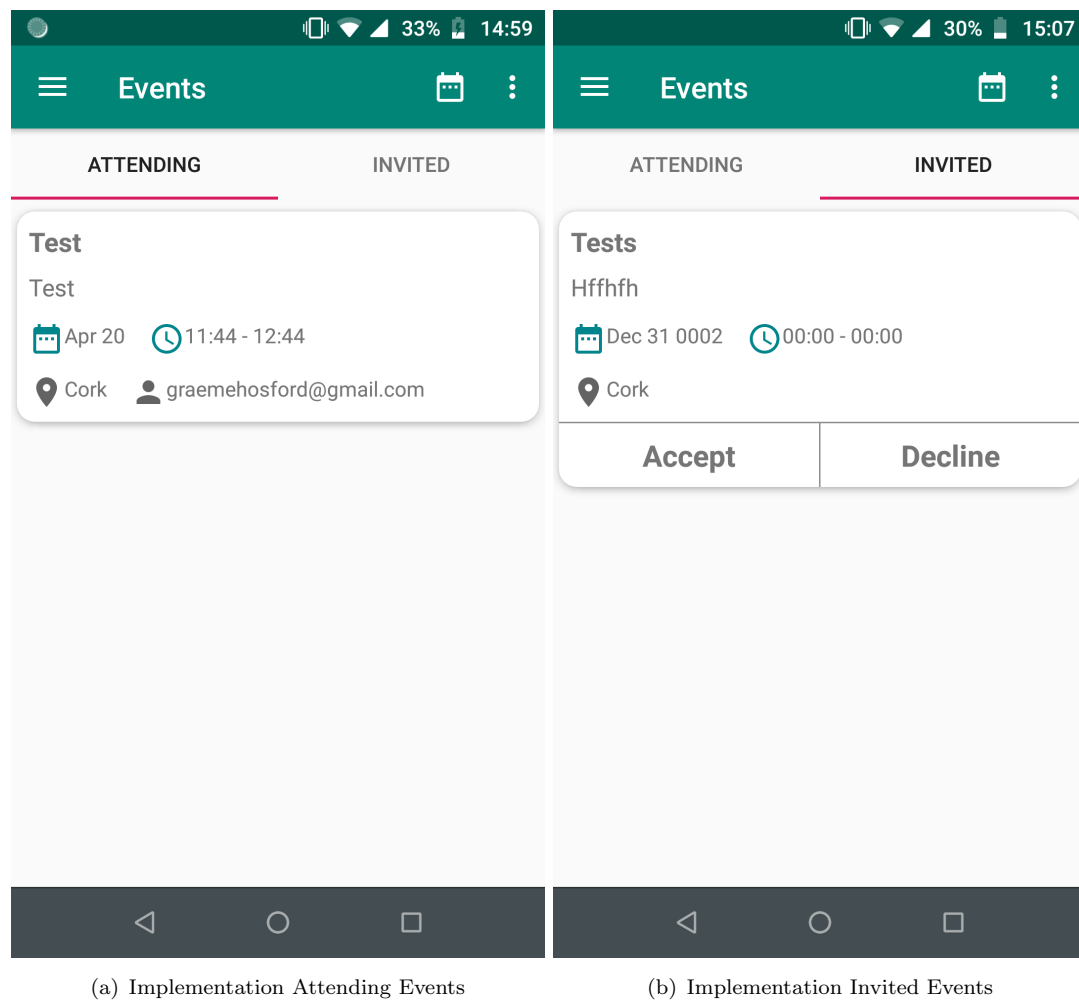


FIGURE 5.5: Mobile App Event List Differences

The Figure 5.5 shows the difference between the two tabs which show lists of events. The tab of events which the user is invited to just includes an extra few options for responding to the event. The lack of a button in the bottom right of the screens in Figure 5.5 is due to the account being used not being an admin account and is not related to any difference that comes with this screen. An admin viewing this screen would have the option to create events as normal.

Chapter 6

Testing and Evaluation

6.1 Metrics

As previously mentioned in Section 5.2.6, the initial evaluation strategy based on user feedback was abandoned in favour of a look at which functional and non-functional requirements were met in this project.

Below are the tables of functional and non-functional requirements as they stand at the end of project development. Also included is whether or not these requirements are implemented and passing or not. Like previous sections these tables are split between Android, Ruby and Firebase functionality. These requirements in these tables are also made up of those outlined in Chapter 3 as well as new requirements that were devised during the semester of development.

6.1.1 Functional Requirement Metrics

Mobile Functional Requirement	Implemented?
An unregistered user can sign up for an account	Yes
A registered user can sign into their account	Yes
A registered user can create a new company	Yes
A registered user can join a company	Yes
A company member can see the list of events they are invited to	Yes
An admin can create an event	Yes
An admin can add attendees to an event	Yes
An invitee can respond to an event invite	Yes

A company member can see the list of people within their company	Yes
A company member can see their own profile	Yes
A company member can edit their own profile	No
A company member can see the profile of other company members	Yes
An invitee can receive a notification when invited to an event	Yes
An event attendee can receive a reminder notification for an upcoming event	No
An event organiser can choose when reminder notifications are sent	No
An admin can set event specific details for other users	Yes
A registered user can make a request to join a company	No
An admin can approve or deny a users request to join their company	No
An invitee or attendee can see an events location on a map	Yes
An invitee can respond to an event invite directly from the invite notification	Yes
An event organiser can see the job titles of those they are inviting to an event	Yes
A company member can view their upcoming events on a calendar	Yes
A company member can provide feedback on the app	Yes
An event organiser can view the events they have created	Yes
An admin can make another user an admin	No
An admin can check attendance for an event	No
A company member can view their CPD points	No
An admin can view the CPD events attended by their employees	No
An admin can register a late invite to an event	No

TABLE 6.1: Mobile app functional requirements metrics

Based on the metrics from Table 6.1 there were 29 functional requirements associated with the Android app part of this project and 10 of these were not implemented. This gives a implementation rate of 65.5% in the Android app.

However, five of the unimplemented requirements were added as ideas after the research phase, including the sprint schedule outlined in Table 4.1, was completed.

Firestore Functional Requirements	Implemented?
FirestoreAuth can create and verify a new user account when one is created through the sign-up process in the mobile app	Yes
Event manager information will be persisted to Firestore Cloud Storage	Yes
Firestore Cloud Functions will send a notification to invitees when a new event is created	Yes

TABLE 6.2: Firestore functional requirements metrics

All three Firestore functional requirements were implemented giving an implementation rate of 100%.

Ruby Server Functional Requirements	Implemented?
A unique ID is generated for each company which is created	Yes

TABLE 6.3: Ruby Server functional requirements metrics

Originally there were two Ruby Server requirements with the other being to schedule the time a notification was sent. Due to the difficulty outlined in Section 5.1.2.3 this functionality was instead moved to being handled by Firestore Cloud Functions and therefore no longer applies as a requirement for the Ruby server.

6.1.2 Non-functional Requirement Metrics

Below are the non-functional requirements for each section of the project. Some of these which require further justification on why they pass or fail will have comments after the table.

Mobile Non-functional Requirements	Pass?
Setting up a user account should be easy to do within 2 minutes	Yes
A user should be able to create or join a company within 2 minutes	Yes
The app should be easy to navigate	Yes
The user should always see information relevant to the current screen being viewed even during network failure	Yes

Data should be loaded as quickly as possible	Yes
The app will not give unexpected behaviour across different versions of Android	Yes

TABLE 6.4: Mobile Non-functional Requirement Metrics

To justify some points here.

In terms of the app being easy to navigate there is a navigation drawer with acts as the main method of navigation within the app. This contain four items; Events, My Profile, My Company, and Give Feedback. Each of these I would say is very self-explanatory on what they contain. Each of these screens except for Events is very simple and needs no more explanation but the Events screen has more going on in it.

The Events screen contains two tabs for types of events; Attending and Invited. Again I would say these are quite self-explanatory and there shouldn't be any confusion to the user about what they are seeing. The Events screen also contains an options in the top-right corner to view their events on a calendar and to view the events they have created.

The calendar option is always available within a single tap and allows the user to view their upcoming events which they are attending on a calendar. This option uses a calendar icon and using it once should make it very obvious what it does.

The option to view the events you have created is always displayed as a text option labelled as My Events. Again I firmly believe this is very self-explanatory and tapping the option just once would make its purpose quite obvious once the user sees the resulting screen.

Given that the original non-functional requirement states that to pass the user should be familiar with navigation within 10 minutes I think it is safe to consider this a pass. The app uses a method of navigation which has been a standard in mobile apps for years now and each different section on the more complicated Events screen is clearly labelled.

In terms of the point that data should load quickly, there will obviously be outliers at different points for various reasons, whether as simple as poor internet connection or otherwise. But from my three months of using this app I can say confidently this point easily passes the original pass criteria of a wait no longer than five seconds for new data.

Finally in terms of no unexpected behaviour on different versions of Android. This one is harder to test than the others. Unfortunately I don't have access to Android devices running each OS version from 21 and up, but from my extensive testing on my own

device running version 28; and the minor checks I was able to carry out on the Android system emulator testing version 23 everything appears to be in order.

As previously mentioned in Section 3.4.2 there are no non-functional requirements relating to Firebase for this project.

Rub Server Non-functional Requirements	Pass?
The ID generated for a new company will be human-readable and easy to remember	Yes

TABLE 6.5: Ruby Server Non-functional Requirement Metrics

The ID generated by the Ruby server for each new company is merely an integer which identifies the company. This ID will, of course, get harder to remember as more companies are added to the system and this number increases but for the scale of this project this ID value will not get very high at all.

6.2 Results

In terms of final results based on the metrics my project had 65.5% of mobile functional requirements implemented and 100% implementation of both Firebase and Ruby functional requirements. Taking the mean between these results we get 88.5% as the implementation rate.

For the non-functional requirements by my measure in Section 6.1.2 I have achieved 100% pass rate for all parts of the project.

I think it is important to note here that this 100% non-functional pass rate could easily change in the face of further testing. For example, as I mentioned when justifying giving my requirement about unexpected behaviour across Android OS versions a pass. I do not have access to devices running every version of Android and therefore do not have a 100% extensive test. However I would also assert that it still deserves a pass based on the testing I was able to do and the verification of common behaviour I was able to get from that. By testing the app on a relatively new OS version (28) and a relatively old version (23) and getting the same behaviour it is incredibly unlikely to get different behaviour on any OS version in between.

Chapter 7

Discussion and Conclusions

7.1 Solution Review

Overall I think the solution provided in this project works as a good starting point in solving the problem outlined in Section 3.1. There could surely be plenty of improvements made and new features added; all of which will be discussed in Section 7.4. However, given the time allotted to complete this project I think it makes a good first step in tackling the outlined goals.

As already discussed in Section 4.5, there is an implementation rate of 88.5% in terms of functional requirements and 100% pass rate in regard to the non-functional requirements. I think this reflects quite well on the project overall but would be quick to note that the 88.5% implementation rate is the average of the mobile, Firebase and Ruby functional requirements. The Firebase and Ruby requirements are both 100% implemented but there are far fewer of these compared to the mobile functional requirements which were only 65.5% implemented.

In comparing these values I would say the project turned out well overall but there was still room for some heavy improvement on the mobile side of things.

Keeping with this line of thought, I would also mention the major reason the mobile requirements have a much lower implementation rate compared to Firebase and Ruby is that many more were added during the implementation phase. As already mentioned, the time simply wasn't there to implement these new user stories.

Having considered these points though I think it is still fair to say this project did quite well with its implementation rate of 88.5% and non-functional requirement rate of 100%. Given these values I think it goes to show this project did quite well in implementing

the necessary features as a means of achieving the project goal. Again though I would note the room for improvement in regard to the mobile app which will be discussed in Section 7.4.

7.2 Project Review

As previously mentioned the implementation schedule devised during the research phase was far too optimistic in terms of how much time I thought I would have for this project. Despite this I tried my best to keep up with it as much as I could and focus on what the most valuable functionality to add would be at all times.

Overall I think I addressed the project reasonably well. Over the course of roughly January 10th up to April 13th I managed to complete a reasonably large Android application using a code architecture which, I personally find makes it difficult to break things in code, but was quite time consuming to implement properly.

As mentioned in Section 5.1, there were a few issues which arose during development which caused a severe loss of time which could have been spent further developing features or implementing new user stories.

In reflection I am quite pleased with how I handled this project. Despite a heavy workload with other assignments and some time being eaten up dealing with the problems outlined in Section 5.1, I feel I still managed to complete quite a lot of the work outlined in my original sprint schedule in Table 4.1. Despite this there are a few things I would definitely do differently now that I have the benefit of hindsight.

Firstly I probably would not use the same Clean Architecture code structure for the Android app. While I still hold that this code architecture works very well in terms of separation of concerns, I would probably instead use the MVVM architecture devised by Google. In brief this code architecture works very similarly to Clean Architecture but heavily uses Observables to update data and UI elements. Using this approach would allow the removal of quite a few classes from my Android project. Thinking back over the semester now I think I would have gotten a good bit more completed if I had not needed to create and spend time on all these classes which were necessary for Clean Architecture.

Using an MVVM code architecture would have also given the benefit of allowing me to learn this Google-recommended architecture on top of simplifying the codebase.

Further details on MVVM code architecture can be found at <https://developer.android.com/jetpack/docs/guide#recommended-app-arch>

Another thing I wish I had done differently during development would be to spend more time streamlining the Android -> Firebase communications. For both the presentation and business layers in the Android project I created a number of abstract base classes for handling common functionality. However I did not do this for the data layer. I falsely assumed that I would rarely need to abstract away common functionality when using Firebase.

By the time i realised my mistake I was at a point in the semester where finding extra time was difficult, and so creating an abstract superclass for handling Firebase functionality was never feasible. This mistake caused a lot of repeated code in the data layer of the Android project.

If I had tackled this problem early on as I did with my other superclasses I likely would have saved quite a lot of time and been able to focus more on implementing new features or improving existing features.

Another thing I wish I could have done during the implementation would be to create a full test suite for the Android app. One of the benefits of using Clean Architecture is that classes are very loosely coupled and therefore quite easy to test. When beginning development I did create quite a few tests for my classes but once again the time constraints that came with other assignments meant that writing these tests had to be abandoned early on.

At the end of development there are roughly 70 unit tests for the app. In a better scenario I would have preferred to have the few hundred that a codebase of this size would require. Unfortunately that just couldn't be done. Having these almost definitely would have saved me quite a bit of time later in relation to finding and fixing bugs in the code.

7.2.1 Skills Developed

In terms of skills gained from this project I wouldn't say there was much gained in the area of new technology. I was already very familiar with developing for Android from years of hobbyist development and completing an 8 month internship as an Android developer. This all also lead me to using Firebase quite a lot over the years. I had the least experience with Ruby but that part of this project is so small and focused entirely on creating a basic REST API which I have done before.

In terms of technical development I think my biggest achievement in this project was my use of clean architecture as the means of structuring my code for the Android app. In all my years of hobbyist Android development I never put much thought into a good code

structure. I had also worked with clean architecture as part of the Android internship I completed but at the point i joined the team the whole structure was already setup and was incredibly stable, almost never needing any changes.

Considering all that, the chance to create the code structure from scratch by myself was definitely a new challenge. As mentioned in Section 5.1.2.2, I did have plenty of difficulties early on in this regard. But after completing this project I would consider myself far more capable now than I was 3 months ago of creating a good code architecture which leads to a far more stable codebase than if no code architecture was used.

As part of writing code for Firebase Cloud Functions I had to write the code using NodeJS which is a new language to me. However I wouldn't say I developed any real skill in NodeJS. The code I needed was only a few lines long and and the only update it ever needed was a one line change at one point.

The biggest skill I feel I got from this project came from the focus on the Android app. I have made plenty of Android apps before but none of them were large projects and for almost all of them I simply forgot about them and moved on when something else caught my interest. So I would say the biggest benefit to me was developing my ability to focus entirely on one project for months on end and stay dedicated to it while tuning out distractions.

Another skill I feel I developed came from writing the Android code 100% in Kotlin. All previous Android projects I have worked on contained a majority, if not 100%, Java code. By developing this project in Kotlin I got some good first-hand experience in how an Android app can be written to produce a codebase which achieves the same functionality as with Java but with far less code. I feel using 100% Kotlin development gave invaluable skills and experience when it comes to future Android development. Particularly as Android becomes an increasingly Kotlin-first system.

7.3 Conclusion

Looking back over the last 7 or so months from the beginning of the research phase up to the end of the implementation phase I feel the project went quite well.

The research phase gave me plenty of time to look into the domain of event management and to think on which technologies I should be leveraging in order to make this project work.

In relation to the problem definition I think this project makes a good start on tackling the problem outlined although I do recognise some areas which could be heavily improved. Looking over the Android app at the end of the implementation phase I believe that the features included are all good starting points and sum up to make a suitable MVP for a student project created entirely by one person. For a real world application I would probably want another 6 months or more of full-time development before doing any kind of public release.

My solution approach for this project worked very well overall. Despite the issues I encountered in Section 5.1 and my thoughts on what I would do differently in Section 7.2 my solution approach held up quite well. An implementation rate of 88.5% of functional requirements and a 100% pass rate of non-functional requirements was achieved. I'm pretty confident in calling that a success based on what I set out to achieve in this project.

To sum everything up, I feel this project adequately accomplished its goals given the time available. As much as I would love to improve this project further with another few months of development, as it stands right now I am very pleased with what was accomplished in just under 4 months of development.

Looking back I was definitely overly-ambitious when I planned out this project. But I would rather a project which is too ambitious and has some unimplemented features over a project which is easy to complete but not of a high quality for a final year project.

7.4 Future Work

There were a few use cases from my original sprint schedule outlined in Table 4.1 which did not make it into the project due to time constraints. On top of these, there is also user stories which were created during the implementation phase coming from new ideas from myself and my project supervisor. Had I more time I think each of these would make an excellent addition to the system.

User Story
As a User I want a reminder of my upcoming events so that I can properly schedule time enough for them
As an admin I want to choose when reminder notifications are sent to event attendees so that I can ensure there is suitable advance notice for the event
As a User I want to edit my own profile so that I can change any outdated or incorrect information

As an event organiser i want to save an event as a draft before inviting any attendees
As an Admin I want to make another User an Admin so that the organisation of events can be done by many people
As a User I want to request to join a company so that I can be included in that companies events
As an Admin I want to view membership requests for my company so that I can decide who should be allowed to join the company events and who should be declined
As an administrator I can check attendance for an event
As an administrator I can register an invite to the event
As an employee I can view my CPD points
As a manager I can view CPD events attended by my employees

TABLE 7.1: Future Work User Stories

Bibliography

- [1] Eventbrite Press Resources. [Online]. Available: <https://www.eventbrite.ie/press/>
- [2] Eventbrite - About Us. [Online]. Available: <https://www.eventbrite.com/about/>
- [3] Eventbrite Organiser iOS. [Online]. Available: <https://apps.apple.com/us/app/eventbrite-organizer/id368260521>
- [4] Eventbrite Organiser Android. [Online]. Available: https://play.google.com/store/apps/details?id=com.eventbrite.organizer&hl=en_IE
- [5] SocialTables Logo Resource. [Online]. Available: <https://www.businesswire.com/news/home/20181016005991/en/Cvent-Acquires-Social-Tables-Power-Tighter-Collaboration>
- [6] SocialTables - About Us. [Online]. Available: <https://www.socialtables.com/about/>
- [7] FitSmallBusiness - Eventzilla. [Online]. Available: <https://fitsmallbusiness.com/eventzilla-user-reviews-and-pricing/>
- [8] Eventzilla - About. [Online]. Available: <https://www.eventzilla.net/us/about>
- [9] Eventzilla iOS App. [Online]. Available: <https://apps.apple.com/us/app/eventzilla/id1437123810>
- [10] Eventzilla Android App. [Online]. Available: https://play.google.com/store/apps/details?id=com.eventzilla.attendee.app&hl=en_US
- [11] Nielsen's Heuristics. [Online]. Available: <https://www.nngroup.com/articles/ten-usability-heuristics/>
- [12] International Journal of Technology and Computing. [Online]. Available: <http://www.ijtc.org/>

- [13] F. H. K. Muhammad Haris, Basit Jadoon, “Evolution of Android Operating System: A Review,” in *International Conference on Advanced Research*, vol. 2, November 2017.
- [14] M. T. Riccardo Coppolla, Luca Ardito, “Characterizing the Transition to Kotlin of Android Apps: A Study on F-Droid, Play Store, and GitHub,” in *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, August 2019.
- [15] S. T. Jahanvee Narang, “Review Study on New Era of Android Kotlin,” *International Journal of Technology and Computing*, vol. 3, August 2017.
- [16] R. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, University of California, Irvine, 2000.
- [17] Android Developers Blog. [Online]. Available: <https://android-developers.googleblog.com/>
- [18] Android Authority. [Online]. Available: <https://www.androidauthority.com>
- [19] Kotlinlang.org. [Online]. Available: <https://kotlinlang.org/>
- [20] M. Masse, *REST API Design Rulebook*, 1st ed. O’Reilly Media, October 2011.
- [21] M. Biehl, *RESTful API Design*, 1st ed. CreateSpace Independent Publishing Platform, August 2016, vol. 3.
- [22] M. Vasic, *Mastering Android Development with Kotlin: Deep dive into the world of Android to create robust applications with Kotlin*, 1st ed. Packt Publishing, November 2017.
- [23] Android Logo with Versions. [Online]. Available: <https://9to5google.com/guides/android/>
- [24] T. Jowitt. (2018, January) Tales in Tech History: Symbian. [Online]. Available: <https://www.silicon.co.uk/mobility/smartphones/symbian-mobile-history-227097>
- [25] F. H. K. Muhammad Haris, Basit Jadoon, “Evolution of Android Operating System: A Review,” in *International Conference on Advanced Research*, vol. 2, November 2017, p. 1.
- [26] J. Callaham. (2019, August) History of Android. [Online]. Available: <https://www.androidauthority.com/history-android-os-name-789433/>
- [27] F. H. K. Muhammad Haris, Basit Jadoon, “Evolution of Android Operating System: A Review,” in *International Conference on Advanced Research*, vol. 2, November 2017, pp. 4 – 5.

- [28] Android - Nougat. [Online]. Available: <https://www.android.com/versions/nougat-7-0>
- [29] Android - 8.0 oreo. [Online]. Available: <https://www.android.com/versions/oreo-8-0/>
- [30] Android 9 Pie. [Online]. Available: <https://www.android.com/versions/pie-9-0/>
- [31] Android 10. [Online]. Available: <https://www.android.com/android-10/>
- [32] (2019, August) Mobile Operating System Market Share Worldwide. [Online]. Available: <https://gs.statcounter.com/os-market-share/mobile/worldwide>
- [33] (2019, June) Smartphone Market Share. [Online]. Available: <https://www.idc.com/promo/smartphone-market-share/os>
- [34] F. H. K. Muhammad Haris, Basit Jadoon, “Evolution of Android Operating System: A Review,” in *International Conference on Advanced Research*, vol. 2, November 2017, p. 1.
- [35] J. Clement. (2019, August) Amazon App Store Number of Apps by Q2 2019. [Online]. Available: <https://www.statista.com/statistics/307330/number-of-available-apps-in-the-amazon-appstore/>
- [36] ——. (2019, July) Google Play Store Number of Apps up to June 2019. [Online]. Available: <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>
- [37] Kotlin FAQ with Logo Resources. [Online]. Available: <https://kotlinlang.org/docs/reference/faq.html>
- [38] F. Lardinois. (2019, May) Kotlin is now Googles preferred language for Android app development. [Online]. Available: <https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/>
- [39] D. Winer. Android’s commitment to Kotlin. [Online]. Available: <https://android-developers.googleblog.com/2019/12/androids-commitment-to-kotlin.html>
- [40] M. Vasic, *Mastering Android Development with Kotlin: Deep dive into the world of Android to create robust applications with Kotlin*, 1st ed. Packt Publishing, November 2017, p. 6.
- [41] M. T. Riccardo Coppolla, Luca Ardito, “Characterizing the Transition to Kotlin of Android Apps: A Study on F-Droid, Play Store, and GitHub,” in *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, August 2019, p. 7.

- [42] FAQ - Kotlin Programming Language. [Online]. Available: <https://kotlinlang.org/docs/reference/faq.html>
- [43] M. Vasic, *Mastering Android Development with Kotlin: Deep dive into the world of Android to create robust applications with Kotlin*, 1st ed. Packt Publishing, November 2017, p. 7.
- [44] Android Jetpack. [Online]. Available: <https://developer.android.com/jetpack>
- [45] M. Masse, *REST API Design Rulebook*, 1st ed. O'Reilly Media, October 2011, p. 1.
- [46] The Birth of the Web - CERN. [Online]. Available: <https://home.cern/science/computing/birth-web>
- [47] What's the Difference Between the Web and the Internet. [Online]. Available: <https://www.geeksforgeeks.org/whats-difference-internet-web/>
- [48] M. Masse, *REST API Design Rulebook*, 1st ed. O'Reilly Media, October 2011, p. 2.
- [49] M. Biehl, *RESTful API Design*, 1st ed. CreateSpace Independent Publishing Platform, August 2016, vol. 3, p. 19.
- [50] R. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.
- [51] M. Masse, *REST API Design Rulebook*, 1st ed. O'Reilly Media, October 2011, pp. 3 – 4.
- [52] R. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.
- [53] R. Martin, *Clean Architecture: A Craftman's Guide to Software Structure and Design*. Prentice Hall, December 2016.
- [54] Clean Coder Blog: The Clean Architecture. [Online]. Available: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- [55] Clean Architecture on Android: The Teamwork.com Way! [Online]. Available: <https://github.com/Teamwork/android-clean-architecture/blob/master/README.md>
- [56] Dependency injection in Android. [Online]. Available: <https://developer.android.com/training/dependency-injection>

- [57] Android KTX. [Online]. Available: <https://developer.android.com/kotlin/ktx.html>
- [58] Android Jetpack Navigation. [Online]. Available: <https://developer.android.com/guide/navigation/>
- [59] RxJava. [Online]. Available: <https://github.com/ReactiveX/RxJava>
- [60] RxJava Understanding Marble Diagrams. [Online]. Available: <https://medium.com/@jshvarts/read-marble-diagrams-like-a-pro-3d72934d3ef5>
- [61] Retrofit. [Online]. Available: <https://square.github.io/retrofit/>
- [62] Butter Knife. [Online]. Available: <https://jakewharton.github.io/butterknife/>
- [63] Moshi. [Online]. Available: <https://github.com/square/moshi/blob/master/README.md>
- [64] Top 10 Android Libraries Every Android Developer Should Know. [Online]. Available: <https://infinum.com/the-capsized-eight/top-10-android-libraries-every-android-developer-should-know-about>
- [65] Firebase Logo. [Online]. Available: https://commons.wikimedia.org/wiki/File:Firebase_Logo.png
- [66] Ruby on Rails Logo. [Online]. Available: <https://www.trzcacak.rs/imgb/hmJJRxT/>
- [67] Ruby on Rails. [Online]. Available: <https://rubyonrails.org/>
- [68] Ruby on Rails API. [Online]. Available: https://guides.rubyonrails.org/api_app.html
- [69] Ruby on Rails: Getting Started. [Online]. Available: https://guides.rubyonrails.org/v2.3/getting_started.html#configuring-a-database
- [70] Ruby on Rails API. [Online]. Available: <https://api.rubyonrails.org/>
- [71] 10 Effective Ways to Improve User Ratings. [Online]. Available: <https://thetool.io/2017/improve-user-ratings>