Scouting Ireland Event Manager

Cormac White

C16406154

Submitted in partial fulfilment of the requirements for the degree of

BSc in Business Computing

Technological University Dublin

May 2020

Supervisor: Catherine Higgins

Declaration

This is original work. All references and assistance are acknowledged

Signed: _____

Date: 01/05/2020

Acknowledgements

I would like to take this opportunity to thank all of the people who have played a role in completing this project. First I would like to thank my family for all the support they have offered not only this year, but over the four years of my time in TUD. Their continued support helped me to get through some of the more stressful aspects of college life.

I would also like to thank my supervisor Catherine Higgins, who's guidance and suggestions helped to keep me on track throughout the year. It was through our weekly one-on-one meetings, I was able to receive a number of helpful tips and design ideas. I would also like to express my gratitude to Audrey Jennings, who also provided me with a number of helpful suggestions during the development of this app. I would also like to thank all other lecturers for their help during these last four years in TUD.

I would like to thank Evros for providing me with real world experience in the software development industry and providing me with the facilities to continue to work uninterrupted during the government lockdown. In particular, I would like to thank Trevor Dagg, Andrew Taylor, Eoin Sheils and Mick Quinlan.

I would also like to thank my friends, who provided me with emotional support throughout this year. In particular, I would like to thank Simon Duffy, for his input over the course of this year.

I would also like to take this opportunity to thank my classmates, who made the last four years quite enjoyable. I would like to thank them for all of the help over the last four years.

Contents

Abstract

This application provides both leaders and parents in Scout groups the ability to manage the various scouting events that go on during a year.

It allows leaders to create events over a number of days, store the event details and display them in a calendar view over that set number of days, which is visible to parents. The number of available spaces is set by the number of leaders attending the event and defined ratios set by Scouting Ireland.

Leaders can select the group that the event is for and then select leaders from that particular group to go on the event. Leaders assigned to that event will receive an SMS notification, to say that they have been assigned to the event. Each leader assigned to that particular event can indicate whether they are going or not through the app.

Parents can tap on a particular date and view all events for their child's group. If their child has not already indicated they are going, the parent can make a payment in the app. If the payment is successful, the child is added to the list of members going on the event

There are a number of other features in this application such as event reporting, group reporting and viewing profiles, which I will go into further detail in later sections.

1 Introduction

Project Title: Scouting Ireland Event Manager

1.1 Introduction to Project

The main goal for this project was to create an app that would allow group leaders and parents to be able to organize events and make payments for those events.

Due to the nature of the group this application is for, I knew I would have to make an application that would be very intuitive, while still being able to complete complex tasks around event organization within Scouting Ireland.

As a member of Scouting Ireland for the last sixteen years, I have gained a good insight into the technical knowledge of both leaders and parents within my group. This helped me to make design decisions that would suit users on both ends of the technical spectrum, while filling the functionality requirements of the application.

While developing this app, I thoroughly enjoyed the challenges I faced and learning about new technologies that I had not come across before, such as Firebase.

1.2 Objectives of Project

The main objective of this project was to create an application that facilitates event organization and payments within individual scout groups. While I did find applications that were based around similar ideas such as NorCamp[1] or Google Calendar[2], however I was not able to find an application that served the same features that this application is providing. While there is a World Scouting [3] application, the application only handles events on an international level and does not facilitate payments from parents to groups.

Another objective of this project is to create an application that provides a sense of transparency between leaders and parents. To be able to achieve these objectives, there are a number of pieces of functionality that the application should be able to do.

General requirements:

- Log in to the application
- View their own details
- View other leader's details from their own group
- View their group's members

Leaders should be able to:

- View the event calendar
- Create an event for a specific group by clicking on a date
- Assign leaders from that group to the event if they are free in the specified date range
- View event details
- Update event details
- Indicate whether they are going on the event or not if they have been assigned to it
- View individual member's details
- Update a member's details
- View Reporting on events and members
- Create member, parent and other leader accounts

[1] https://play.google.com/store/apps/details?id=de.schwarzstoff.norcamps
[2] https://www.google.com/calendar/about/
[3] https://play.google.com/store/apps/details?id=org.scout.android

Parents should be able to:

- View their group's events in the event calendar
- View event details
- Make a payment for an event if they have not already done so and there is available space.
- View events they have paid for.
- Update their own details
- View their child's details
- Update their child's details
- Create a review for an event if they have paid for it

1.3 Business Case for Scout Event Manager

As a member of Scouting Ireland for most of my life, I have had first-hand experience with both sides of event organization, both as a member and a leader. Throughout the years, I have noticed that not much has changed in relation to events in Scouting Ireland. While most groups have embraced some modern innovations such as social media and online banking, the way in which most scout groups organize events leaves a large amount of room for mistakes to occur. An improperly formatted bank transfer or a missed social media post can lead to confusion between parents and leaders, and amongst other leaders. With this application allows both parents and leaders to view events on a month-by-month basis and view the details of each event, ensuring parents and leaders know what events are happening and when.

From my own experience in Scouts, events are typically paid for in cash. This can also leave room for issues as when events are paid for in cash, a leader must be trusted with the cash to deposit into a bank account. While all leaders are Garda Vetted[4] to ensure they are trustworthy, there is still room for genuine mistakes to occur, such as cash being misplaced before it is deposited in a bank. This application removes that issue by facilitating the payments through the PayPal API.

Especially with the revelations [5] that have come out over the last few months, transparency has become a vital aspect of all groups within Scouting Ireland. While Scouting Ireland works to ensure every leader is vetted every three years[4], this application will provide parents with piece of mind that they know all of the details about an event, who is going and that they have easy access to each leader's details and contact information.

[4] https://www.scouts.ie/Scouter/Management-Resources/Policies-Procedures/All-Official-Policies/SID-39B-10-Garda-Vetting-Policy.pdf
[5] https://www.thetimes.co.uk/article/scouting-ireland-sued-over-historical-sex-abuse-claims-76px7qjgq

1.4 Business Rules

The business rules are a list of rules that the application must follow in order for it to function properly. I will go into more detail about these rules further on in section 2.2.1 Functional Requirements section.

Events:

- Only a leader can create an event
- When an event is created, the leader who created the event is automatically assigned to that event and marked as going
- Only the leader who created the event can edit the event details
- The end date must be before the start date
- A minimum of 2 leaders are required
- A leader can only be assigned to one event per day
- Once the end date of an event has passed, the details cannot be edited
- Each event can either be a camp or a hike

Parents:

- A parent can only view events for their group
- Payments can only be made if there are available spaces on the event
- Parents can only view details of leaders associated with their own group
- Each parent must be associated with a member
- Parents can update details for their child only

Leaders:

- Leaders can only view and edit member details of members in their own group.
- Leaders can view events for all groups in the calendar
- Only leaders are able to view reports
- Only leaders are able to create accounts

1.5 Technologies

1.5.1 Java:

The main language I decided to use for this application was Java, which was developed by Oracle[6].

I chose this language as it is the official language of Android app development[7]. This meant that Java would have the most resources for Android development, making the development of this application much simpler.

The fact that Java is also object-oriented, suited the nature of this application perfectly.

I used this language in tandem with XML to create the application's UI and Android Studio as my IDE.

1.5.2 Firebase:

Firebase is a suite of products, which is owned by Google[8].

For this application I used the Firebase Realtime Database as my database, which stores data in a JSON format. As this application also uses data that will be used by a number of different users on different devices, it is important that all users have the same information. As Firebase is a cloud based service, it was clear that it was the best option for this application's database. Another reason that led to me using the Realtime Database was the fact that it was the database of choice for an online course [9] I was using when I started learning to develop Android applications.

Firebase also has an Authentication feature, which allowed me to create accounts with passwords and link them to user elements in the Realtime Database through a generated ID. This allows a user's password to be stored more securely by separating the actual password from the user object in database.

Due to the nature of the data being used in this application, security is a vital requirement when using a database. By using the authenticator for my sign in, I am able to completely hide user's passwords and allow them to securely log into the application. As Firebase is a Google product, it has the robust security that other cloud based database services might not have. This helps to provide reassurance that any data being stored in this application will be secure.

---

[6] https://vertex-academy.com/tutorials/en/history-of-java/
[7] https://www.androidauthority.com/develop-android-apps-languages-learn-391008/
[8] https://howtofirebase.com/what-is-firebase-fcb8614ba442
[9] https://www.udemy.com/course/complete-android-developer-course/learn/lecture/7665756

1.5.3 Compact Calendar View:

This calendar view works similarly to the standard calendar view that comes with Android Studio, however, the problem with the default calendar view was that it does not have functionality to add icons for events.

Not only does the Compact Calendar View support this functionality, it allows for a variety of coloured markers to be assigned to a particular date, which I felt would help in terms of the application's usability

1.5.4 PayPal API:

I used the PayPal API in this application to allow parents to make payments for events.

When looking for API's to handle payments from parents to groups, I constantly had to bear in mind, the wide range on technical knowledge that the users of this application would have. It was for this reason that I decided to use the PayPal API. My reasoning for my selection was due to the fact that when speaking with other leaders in my group, PayPal was often the first company that came to mind. It was this reason, coupled with PayPal's easy-to-use interface, that led me to use their API for this application.

The PayPal API's premade payment screen, allows users to either pay with their own PayPal account or make a one-time payment as a guest, without needing to register an account with PayPal. I felt this would be a major benefit, as some parents may not feel comfortable having a PayPal account registered with their bank details. As well as this, the API also allows users to take a photo of their credit card, and reads the necessary details from the image, making it even easier for parents to make payments for events.

PayPal also provides a sandbox, which allows developers to make simulated payments. It also allows developers to generate fake credit cards, meaning that there is none of the risk involved when dealing with real credit cards and real accounts.

1.5.5 Google Maps:

For this application I am using Google Maps to handle all of the navigation based functionality.

While researching different navigation API's, it was clear from both my interactions with other Scout leaders and online articles [10] [11] that Google Maps was by far the most used, meaning that a larger number of users would already understand how to use Google Maps, making the usability of the application better. It was for this reason that I decided to use Google Maps to serve this functionality.

1.5.6 MP Android Chart:

I used this API to create the reports that are used in the application. Developed by Philipp Jahoda[12], the MP Android Chart API offers a suite of different graphs to display data such as pie charts, line charts and bar charts.

While looking at a number of different reporting API's, I quickly noticed that many of them were quite technical, with a large amount of information being displayed at once. My fear was that by using an overly-technical report, some users may be put off the feature all together and avoid using it.

The MP Android Chart API suited this application quite well, with its more simplistic design. I feel that the more simplified design of this API's charts works well to ensure that all the relevant information is being shown while ensuring that the user is not overwhelmed by the amount of information being displayed

1.5.7 GitHub:

I used GitHub as my source control manager. As I was familiar with Github through using it over the last four years. Through Android Studio, I was able to link my project with my GitHub repository, allowing me to commit and push changes directly from the IDE.

Github Repository:
https://github.com/cormac-white97/FinalYearProject.git

[10] https://www.carbibles.com/best-gps-apps/
[11] https://www.androidheadlines.com/2019/06/best-android-navigation-apps-2019.html
[12] https://github.com/PhilJay/MPAndroidChart#documentation

2 Requirements and Analysis

2.1 Business Actors

In this application there are two types of users, a leader and parent.

2.1.1 Leader:

In this application the leader has a number of extra permissions that are required for this application to function. Leaders are able to create Events and assign leaders to those events. The leaders that are available to be assigned to the event depend on the group that the event is being created for. Leaders can also go to the details page of an event they have been assigned to and indicate if they are available or not.

If they select that they are available, the field in the event element in the database will change from “Pending”, to “Approved” and if they indicate they are not going, the leader’s ID will be removed from that event all together. Leaders can also view all of the events that are in the database regardless of their assigned group. When a leader creates a new event, the number of available spaces is determined by the number of leaders that are indicated as going on the event, based on leader to member ratios that are set by Scouting Ireland.

2.1.2 Parent:

A parent is able to view all of the events that are assigned to their child’s group that have been marked as open. They can view the details for these open events, and if there are available spaces, they can make a payment and add their child to the list of members attending the event. Leaders can also view the details of leaders in their own group. This allows parents to have direct contact details for each leader in their child’s group. Parents can view their own details as well as the details of their child. Parents can also update the details of their own child, which allows for more direct updating of member’s information such as a change of contact number or medication.

2.2 Requirement Analysis

For my requirement analysis I tried to find applications as similar to what I was hoping to create as possible. I looking at generic camping applications [13]and school organization applications [14]in order to gain a better understanding of what features I wanted to use for this application.

Luckily, thanks to my membership in Scouting Ireland, I already had a good understanding of Scouting specific requirements such as the leader to member ratios. As well as my own experience, my parents had a good insight into what features a parent would like to have in this application, such as the ability to update their own child's details and having easy access to their group's leader's details. This insight also aided me in the design of this application by allowing me to see the design through both a leader's and a parent's eyes. Through speaking with other leaders in my group, I was able to create a list of important features that they saw as important for an application of this type. With this list in mind, I looked at a number of different school and camping related applications on the Google Play Store, where I was able to see how other, similar applications implemented similar features, which gave me inspiration for the design of this application.

[13] https://play.google.com/store/apps/details?id=de.schwarzstoff.norcamps
[14] https://play.google.com/store/apps/details?id=com.uniquepublishing.presentationcollegebray&hl=en_IE

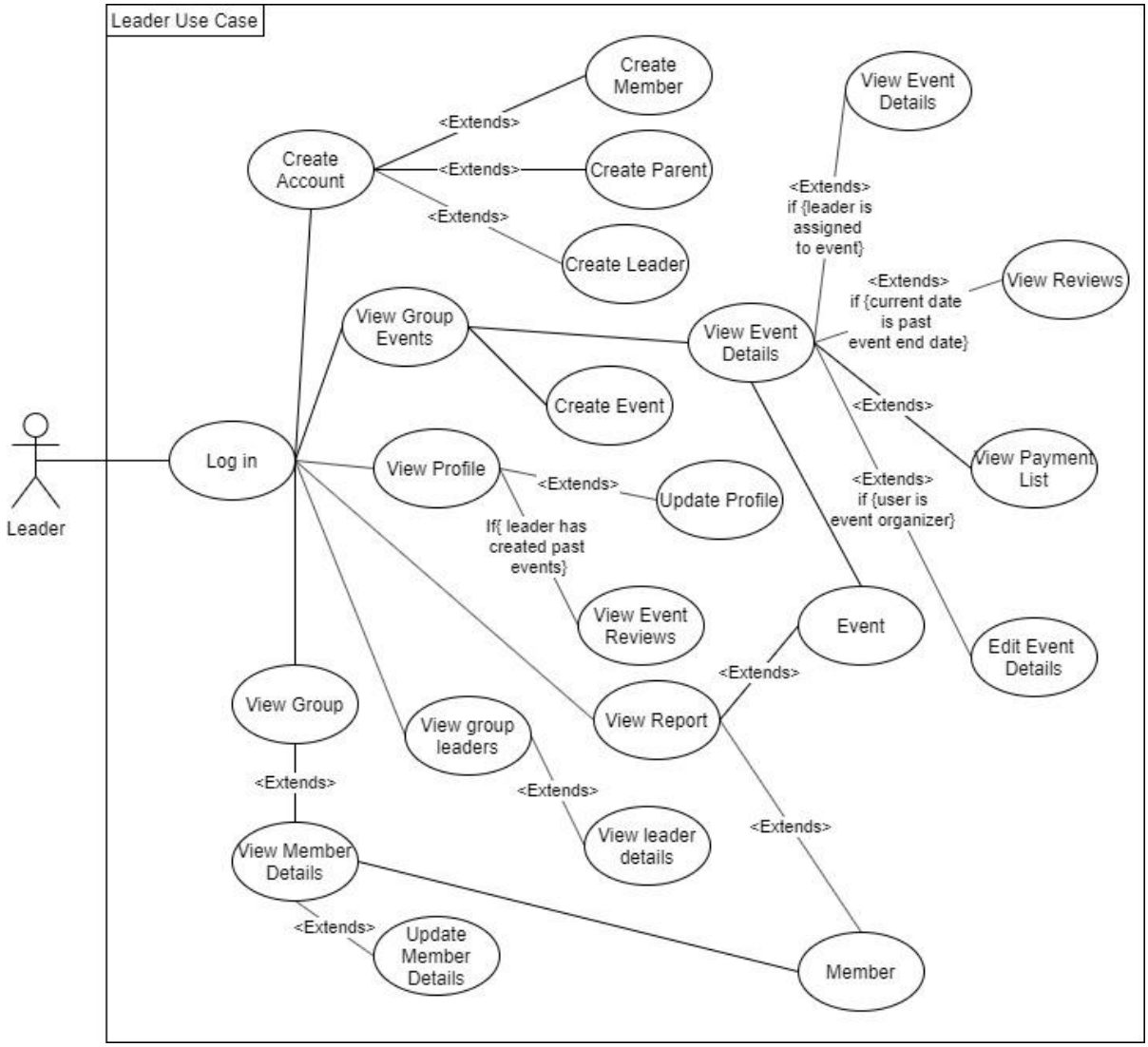
Leader Use Case
Leader
Log in
Create Account
Create Member
Create Parent
Create Leader
<Extends>
<Extends>
<Extends>
View Group Events
Create Event
View Profile
Update Profile
If{ leader has created past events}
View Event Reviews
View Group
View Member Details
Update Member Details
View group leaders
View leader details
View Report
View Event Details
<Extends> if {leader is assigned to event}
View Reviews
<Extends> if {current date is past event end date}
View Payment List
<Extends> if {user is event organizer}
Edit Event Details
Event
Member

Figure 1

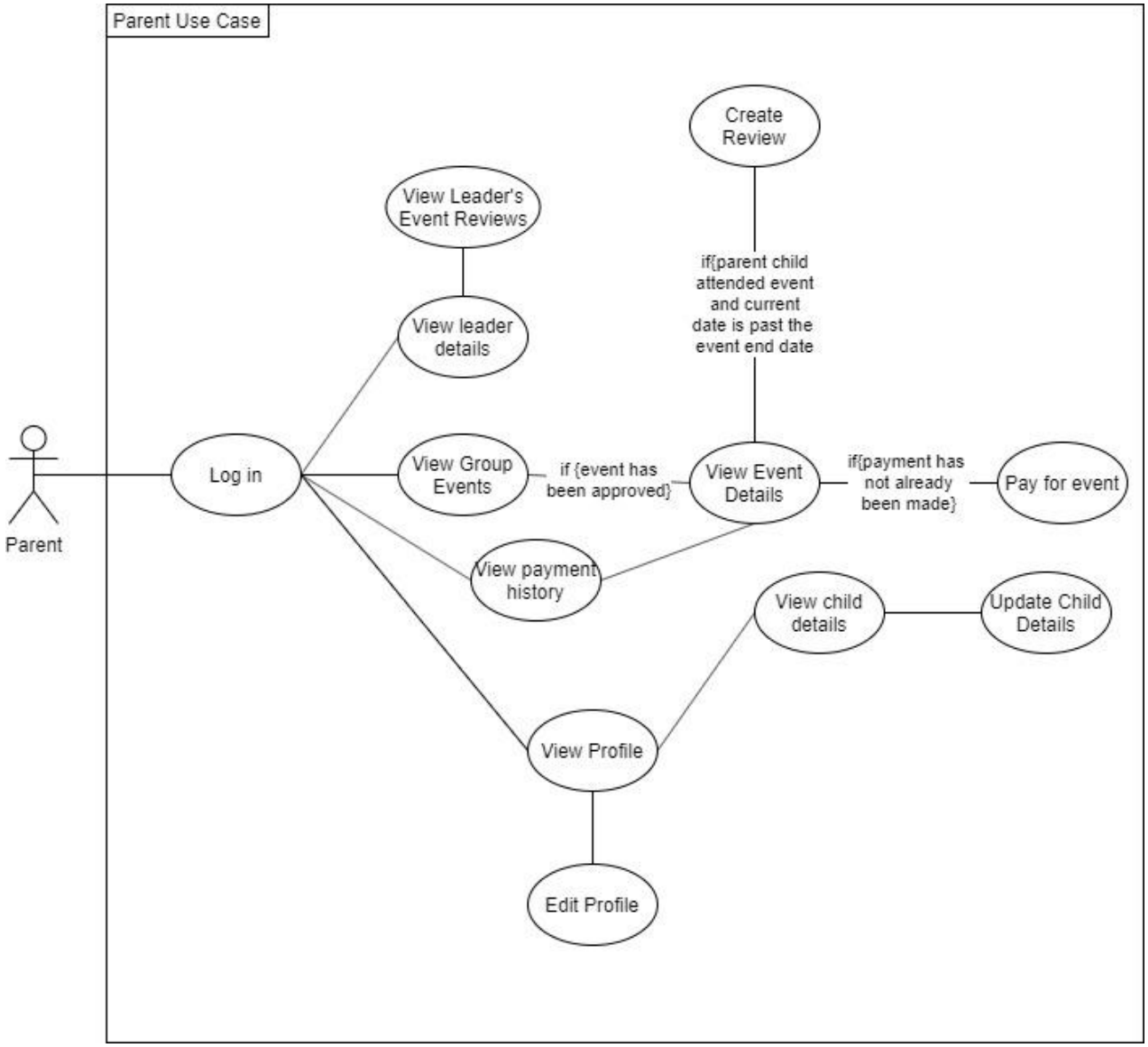
Parent Use Case
Create Review
View Leader's Event Reviews
if{parent child attended event and current date is past the event end date
View leader details
Parent
Log in
View Group Events
if {event has been approved}
View Event Details
if{payment has not already been made}
Pay for event
View payment history
View child details
Update Child Details
View Profile
Edit Profile

Figure 2

2.2.1 Functional Requirements

The application's functional requirements describe the basic functions that the application must be able to perform.

There are a number of functions for different types of users that are required in this application, which include:

2.2.1.1 General Requirements

Function	Requirement	Justification
Log in / Log out	The ability to log in and log out of the application.	A user needs to be able to log in to the application to use the application's functionality.
Reset password	The ability to reset a password	A password needs to be able to be in the event the original password is forgotten.
View Profile	The ability to view the user's own profile	The user needs to be able to view their own details to ensure the information about them is correct.
Edit profile	The ability to edit the user's profile	The user needs to be able to edit their profile to correct possible errors or update their details.
View leaders	The ability to view leader's profiles	The user needs to be able to view the profiles of the leaders in their group to get contact details, check their vetting date or reviews about events they have organised.

View Event Details	The ability to view event details	This is one of the core pieces of functionality in this application. The user needs to be able to view the details for an event.

2.2.1.2 Leader Requirements

Function	Requirement	Justification
View all events.	View all events in the calendar.	A leader needs to be able to view all events for all groups to know what is happening in each group in their troop
Add an event.	The ability to create a new event.	This is the core piece of functionality in this application. Leaders must be able to create new events for parents to make payments for.
Edit an Event	The ability to edit the details of an event	The leader must be able to update an event's details as certain details may change after the event is created or mistakes may be made.
Delete an Event	The ability to remove an event from the calendar.	If an event is cancelled, the leader must be able to remove the event.

Assign leaders to events	The ability to assign a group of leaders in a group to an event.	The user needs to be able to assign leaders to an event to inform the parents about who is going on the event and set the number of available spaces on the event.
Check leader availability	The ability to check a leader's availability for certain dates	This functionality is required as it ensures that no leader can be assigned to two different events that are on the same date.
Indicate availability	The ability to decide if the user is going on the camp.	The user needs to be able to decide if they are going on the event they are assigned to and indicate it on the application, so parents know exactly what leaders are going on the event.
View group	The ability to view members in the user's group	Leaders need to know exactly who is in their group
View member	The ability to view a member in the user's group	A leader needs to be able to view their group member's details to view parents' contact information or specific notes about the member
Edit member	The ability to edit a member's details	Leaders need to be able to update their member's information in the event that their details change

2.2.1.3 Parent Requirements

Function	Requirement	Justification
View events that are assigned to their group and have approval.	A parent should only be able to view events that are assigned to their group and approved	A parent has no need to see events that are not assigned to their group. By only being able to view approved events, the event details that the parent are set and cannot change, helping to eliminate possible confusion.
Create review	The ability to create a review if they have paid for an event and the end date has passed.	The parent should be able to create a review as it is part of Scouting Ireland's plan, do, review philosophy.
Edit child's details	The ability for the parent to be able to update their child's details.	The parent should be able to update their own child's details to help to ensure that their child's details are as up to date as possible and any updates can be made faster.
View leader reviews	The ability to view a leader's event's reviews for past events.	The parent should be able to view reviews for events organised by a leader on that leader's profile page so they can see what experience that leader has with organising events.
View payment history	The ability for the user to be able to view their past payments for events.	The parent should be able to view their previous payments so they know what events they have already paid for.

2.2.2 Non-Functional Requirements

2.2.2.1 Security:

As the data being used in this application is so sensitive, security is a vital aspect. I was mainly for this reason that I decided to use Firebase as my database. The Firebase Authenticator completely hides user passwords, improving the security of the application. As well as this, Firebase only allows reads and writes from authorised applications, helping to secure the application's data.

2.2.2.3 Usability:

Due to the wide variety of possible users, it is important that this application be as intuitive as possible. To achieve this, I decided to add a number to messages around the page to inform the users what they need to do in certain parts of the application that could be confusing for new or non-technical users.

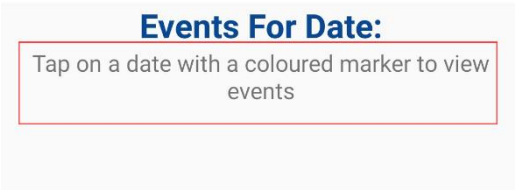

Figure 3

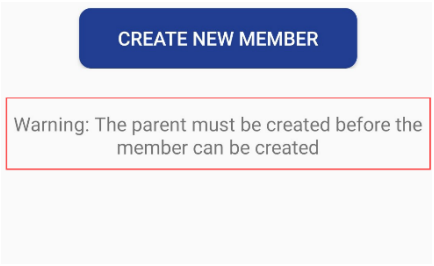

Figure 4

2.2.2.4 Class layout:

As I was working on this application, I noticed there was an increasing number of classes between classes to UI, view details, creation classes and adapters for the various RecyclerViews. To help make the navigation around the classes easier, I divided each class into different packages based on what function the class is serving.

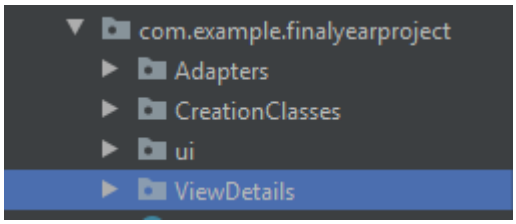

Figure 5

3 Design

3.1 System Architecture:

In this application, there is a view layer, which contains all of the UI elements that allow the user to interact with the application. Below this, each layout file has an associated class, which contains all of the business logic in the application and manipulate data in the database (Firebase Realtime Database). As this application also uses a number of RecyclerViews, I had to create a number of different adapters in order to display the data.

In order to display the locations of the various scout centres, I downloaded a KML file that is available on the Scouting Ireland website. This layer contained markers and information about each of the scout centres around the county. When the user creates a new event, this KML layer is added to the Google Map, allowing the user to select an event location.

To allow a user to make a payment for an event, I called on the PayPal API, which displays PayPal's own payment screen, allowing the user to pay with their PayPal account or enter their card details, either through their camera or enter the details manually. If the API returns a successful payment, the child is added to the event.

3.2 Data Model:

3.2.1 ERD Diagram

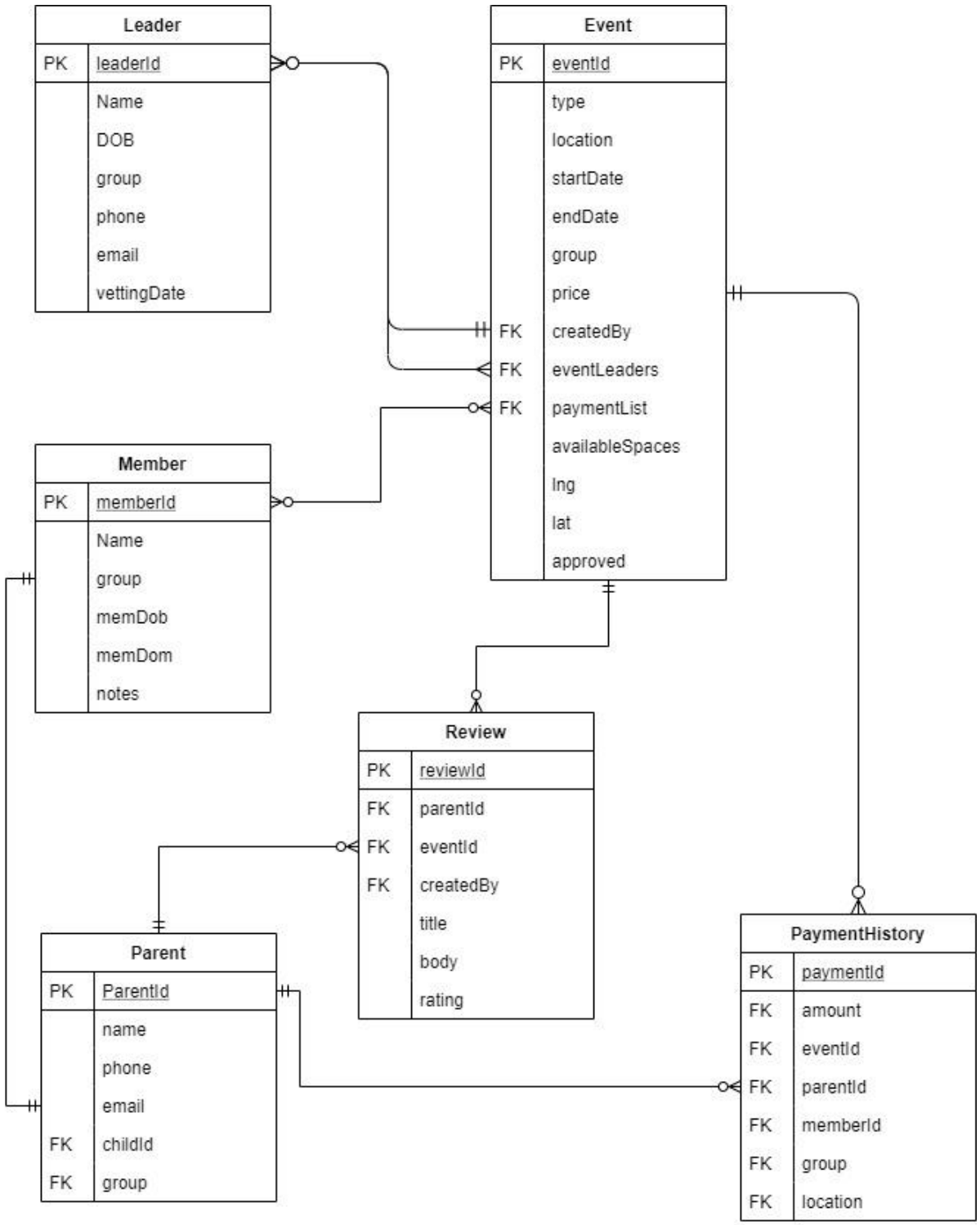

Figure 6

Entity Relationship Diagram:

As the Firebase Realtime database stores data in a JSON format, there are no foreign key constraints in the database. However, there are a number of relationships that exist in the database that are vital to allow the application to function properly. Although there are no foreign key constraints in the database, I was able to structure the database in a way that simulates foreign keys. For example, I have manually set the createdBy element in the Events table to be a leader's unique ID and the parent's ID as a foreign key in the payment history database.

In the relationship between the leader element and the event element, it is clear that there are two relationships that exist between them. I created this relationship between these two elements as each event has a leader who is the creator of that event, which only has one leader ID. However, many leaders can be assigned to that same event as attending leaders, which allows for many leader ID's to be stored in the eventLeaders element.

3.2.2 Database Object Summary

In this section, I will go into more detail about each element in the database. I will explain the purpose of some of the more abstract elements of each table in this database.

Leader	
PK	leaderId
	Name
	DOB
	group
	phone
	email
	vettingDate

This element stores information about a single leader.

DOB: This holds the leader's data of birth

Group: This is the group that the leader is a member of.

VettingDate: This describes the date that the leader was last vetted. Which as per Scouting Ireland guidelines, must be every three years.

Member	
PK	memberId
	Name
	group
	memDob
	memDom
	notes

This element stores information about a member

Group: stores the group that the member is a part of.

memDob: Stores the members date of birth.

memDom: Stores the date the member joined the group.

Notes: Stores possible extra information such as medications or possible allergies.

Parent	
PK	ParentId
	name
	phone
	email
FK	childId
FK	group

This element store information about the parent.

ChildId: This is a foreign key to the member element. It assigns a parent's account to their child's account, allowing for editing permissions.

Group: This is also a foreign key from the member element. This ensures that the group the parent is assigned to only their child's group.

Event	
PK	eventId
	type
	location
	startDate
	endDate
	group
	price
FK	createdBy
FK	eventLeaders
FK	paymentList
	availableSpaces
	lng
	lat
	approved

The event element stores information about upcoming events.

Type: This describes the type of event that is being held, which is either a camp or a hike.

Location: This holds the location that the event is being held.

StartDate: Holds the date value for when the event starts.

EndDate: Holds the date value for when the event ends. If the event is only going to be for one day, the endDate value can be the same as the startDate value, but it cannot be before the startDate.

Group: describes the group that the event was made for i.e. Beavers, Cubs, Scouts, etc.

Price: The price that the event will cost.

CreatedBy: This describes the leader that created the event. It is a foreign key related to a leader ID.

EventLeaders: Holds a HashMap of leader ID's that have been assigned to the event and a flag to say whether their response is approved or still pending.

PaymentList: Holds a list of member ID's, whose parents have paid for the event and are going on the event.

AvailableSpaces: This is the number of spaces that are available on an event. This is set by leader to member ratios that are set by Scouting Ireland, and decreases as payments are made.

Lng: This holds the longitude value of the location that the event is going to be held in.

Lat: Holds the latitude value of the location that the event is going to be held in. This coupled with the longitude allows the event's location to be displayed as a marker on a Google Map view.

Approved: This is a flag that is set by the event creator and determines whether the event is ready for parents to make payments for. This allows edits to be made without parents being able to view the event. This ensures that when parents view event details, they know that the

details are set, reducing possible confusion between parents and leaders.

PaymentHistory	
PK	paymentId
FK	amount
FK	eventId
FK	parentId
FK	memberId
FK	group
FK	location

This element holds information about payments that were made by parents.

Amount: This is a foreign key that is set by the event that the payment was made for.

ParentId: This holds the ID of the parent who made the payment.

MemberId: This holds the ID of the member that the payment was made for.

Group: This holds the group that the event was for.
Location: Holds the location that the event was held in.

Review	
PK	reviewId
FK	parentId
FK	eventId
FK	createdBy
	title
	body
	rating

This element holds information about reviews that were made by parents for each event.

ParentId: This holds the ID of the parent who made the review

EventId: This holds the ID of the event that the review was made for.

	CreatedBy: This holds the ID of the leader who made the event the review is being made for. **Title:** This is the title of the review. **Body:** This is the body of the review. **Rating:** This is a rating from 1 to 5 that a parent can add to a review.

3.3 Software Design

In this section, I will discuss some of the decisions I made while creating this application.

3.3.1 Log in/ log out functionality:

Due to the fact that the information in this application is highly sensitive, it is vital that users are able to log in and out of the application. To implement this feature, I used Firebase's authenticator to allow users to securely log in and log out of the application. I also used the Firebase User object as a session, to allow the system to determine who is logged in and what their permissions are.

3.3.2 Registration:

As the list of users is quite restrictive, I could not allow users to register themselves. For this reason, I decided to only allow a leader to create a new account

3.3.3 Nav Drawer:

As there are a number of pages that the parent should not be able to access. For this reason, I decided to hide the links to these restricted pages. To do this, I decided to read in the details of the logged in user and if the account type is a parent, hide the restricted links, and if the account type is a leader, show all of the necessary links to pages.

Figure 7 shows the links available to a leader account, while figure 8 shows the links available to the parent account.

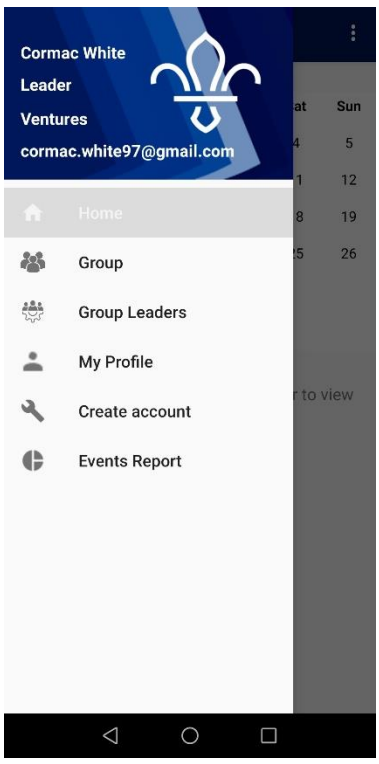

Figure 7

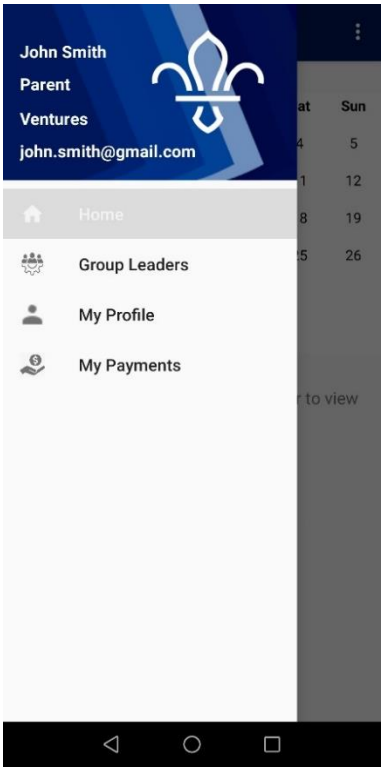

Figure 8

3.3.4 Home

This page functions as the main view in the application.

With the vast range of technical knowledge that the users of this application would have, I wanted to make the home page as intuitive as possible. For this reason, I decided to display only necessary information, depending on the type of account that is logged in. For example, I elected to show all of the events, regardless of group, to leaders, as from my own experience, leaders tend to prefer to know what groups are doing what in their troop. While this is useful for leaders, parents have no reason to need to see other groups events, which is why I decided to restrict the events that parents can see to only their own group's and events that have been made available.

As leaders are only able to create events, I had to be able to decide whether to show the add button or not when a date is tapped. Due to the nature of storing and having the parent and leader details in two different elements, this was made slightly more complicated. To find the account type of the user, I created a list of account types (leader and parent). I then iterated through the leader list and if the account was not found, I would then iterate through the parent list and hold the account type in a global variable. Then in the onDayClicked method, the system would check the account type and decide whether to display the add button or not.

3.3.5 Event Creation

Once the leader has selected a location for the event, there are a number of details that need to be added in order to complete the event creation process.

As the event type can only have two values (camp or hike), I decided it would be best to use a Spinner for the event type. This allowed me to limit the possible entries for the event type, while still keeping in intuitive.
As the event location is selected in the Google Map view in the previous screen, I elected to pass the name of the location, and the longitude and latitude of the location through an intent. It is for this reason, I decided to make the location field uneditable to ensure that the name of the location corresponded to its longitude and latitude.

As the start date of the event is selected in the calendar, I also decided to make the field uneditable. When adding an end date for the event, I have added checks to ensure that the end date is not before the start date, and that the end date is entered in a correct format (DD-MM-YYY). If either of these

checks fail, the write to the database is stopped and a toast message is displayed, informing the user what the problem is.

When saving the start and end dates of an event, I elected to store the values in the epoch date format. I chose this format, mainly as it is the format that the Calendar API uses. I also chose it as it kept the date values format-agnostic, which makes the code more scalable.

I decided to keep the price of the event as an open value, as there are a vast number of different variables that can affect the price of an event. For example, there could be two groups going to the same location, but the price for those two groups can vary depending on things such as the activities the group is going to do on the event, the menu the group is going to use or if the event has been subsided by the group in any way. I felt that there are too many possible conditions to be able to set the price automatically, which is why I left it to the event creator's discretion to set the event's price.

As each event must have other leaders assigned to it, I decided to use a dialog box with check boxes to allow the user to select leaders for the event. Once the group is selected, a TextView appears asking the user to select leaders. When the user taps on this TextView, the dialog box appears with leaders that are in the group the event is for. Once the leaders are selected, the TextView is set to a list of the selected leaders. When the user taps the "Create Event" button, all of the existing events are checked to see if any of the selected leaders are already going on an overlapping event. If this happens, a Toast message appears, indicating that one or more of the selected leaders are not available.

When the group is selected and the number of leaders is selected, the number of available spaces for the event is set based on ratios that are set by Scouting Ireland[15]. When the event is initially created, the number of available spaces is set as if one leader was going. Although this is not the case, there is only one leader who has accepted the invitation, which is the event creator themselves. I chose to only update the number of available spaces once the event leaders have accepted their own invitations.

Once the event is created, the user's unique ID is set as the event creator. As well as this, their ID is also added to the eventLeaders, and they are

[15] https://www.scouts.ie/Scouter/Management-Resources/Policies-Procedures/All-Official-Policies/SID-39-05-Code-of-Good-Practice.pdf

automatically indicated as going. As well as this, an SMS message is sent to each selected leader, informing them that they have been selected for this event, and that they can indicate their availability.

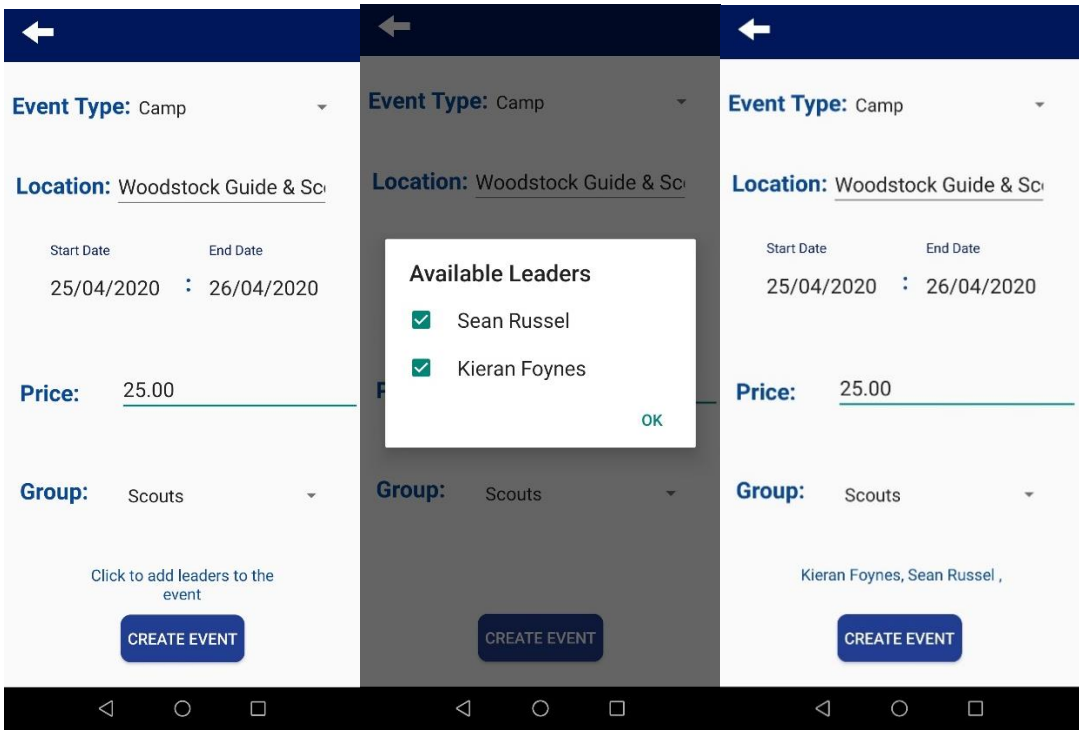

Figure 9

3.3.6 Viewing Events

3.3.6.1 Leader Event View

When viewing an event's details as a leader, there are a number of different permissions that I decided to add in order to make the application more useable.

When a leader is assigned to an event, their ID is added to a Map, with their value is set to "pending". This indicates that the leader has not yet made a decision as to whether they are going on the event or not. If the assigned leader accepts their invitation, their value in the map will be set to "approved" and their name will appear in the "leaders attending" TextField. The number of available spaces will also be updated based on the Scouting Ireland ratios. For example, if the event group is Scouts, there will be another 8 spaces made available on the event.

If the leader declines the invitation, their ID and value will be removed from the "eventLeaders" Map in the database. I chose to remove them from the list rather than set their value to "rejected", as it left an opportunity for that same leader to be readded to the event if the leader's circumstances change and they can go on the event after rejecting the invitation.

If a leader is the event creator, they can edit the leaders that are assigned to an event and the price of the event. I chose to make these two fields editable, as they are the two pieces of an event that are most subject to change. As leaders are able to decline invitations, it is important that an event creator can add other leaders in their group that were not already selected when the event was originally created. The price can also be subject to change in event such as under-budgeting or entering in the wrong value.

Once the event creator is happy with all of the details and the assigned leaders have accepted their invitations, they can now make the event available. This means that all of the event details are set and can no longer be edited. Parents in the same group will now be able to see the event in their calendar views and make payments for the event. I decided to only allow payments for approved events, as it ensures that parents know that the details they are seeing are set and will not change. Ensuring that they know exactly what they are paying for.

If the end date of the event has not passed, the event creator and attending leaders can view the payment list for the event. This gives those leaders and idea of their numbers and who will be attending the event, which is useful for things such as setting menus or deciding on activities.
If the event's end date has passed, the event creator and the attending leaders can view reviews from parents. I added this feature as it works in tandem with Scouting Ireland's One Program, which has "Plan, Do, Review"[16] as one of the key pillars of event organization.

[16] https://www.scouts.ie/Scouter/Programme-Scouter-Stuff/ONE-Programme/

3.3.6.2 Parent Event View

When viewing an event as a parent, there only two things that a parent can do; make a payment or create a review for the event.

When a parent views an event's details, it first checks that the event's end date has not passed. If it has not passed, it then checks if the parent has already made a payment for the event. If they already have, a TextView is shown explaining that they have already paid for the event. Otherwise, a button is shown which will make a call to the PayPal API, allowing them to make the payment. If the payment is successful, a Toast message is shown indicating that the payment was successful and the number of available spaces is decremented by one. The parent's childId is then added to the paymentList in the event object as well as a paymentHistory object is created and added to the database. This functions as a type of informal receipt, which shows the parent what events they have paid for.

If the event's end date has passed, the system checks to see if the parent has made a payment for the event. If they have, they will be given the option to create a review for the event. By checking that the parent has made a payment for the event, I can ensure that parents whose children did not attend the event cannot make a review for the event.

When the parent creates a review, they will add a title, body and a rating from 1 to 5. Once this review is created, it can be seen by the event leaders. As well as this, it is added to a RecyclerView at the bottom of a leader's profile. I chose to do this, as it gives parents a better idea how experienced each leader is at organising events, and how those events went. I felt this was important, as it gives parents piece of mind when sending their children on events with these leaders, which is a major aspect as this application.

3.3.7 Viewing Group Members

When viewing a group's members, I decided to only show the members who are in the group that the user is in, as this would help to reduce possible confusion and ensure that leaders know who is in their own group. I am also only allowing leaders to view members as parents have no need to be able to see member details for members who are not their child.
Leaders can view the details of a particular child by tapping on that child's name. This will then open a page that contains all of the details about that child.

3.4 User Interface Design

When making decisions on how I was going to design the UI for this application, I constantly had to focus on the intuitiveness of it, due to the vast range of technical knowledge its users would have. For this reason, there were a number of different decisions that I had to make while developing this application.

3.4.1 Navigation Drawer:

As the use of this application is not linear, the first UI decision I had to make was how I was going to allow the user to easily navigate around the application. To make is as useable as possible, I decided to implement a drawer to manage the application's navigation. By using this, I can allow the user to easily switch back and forth between pages.

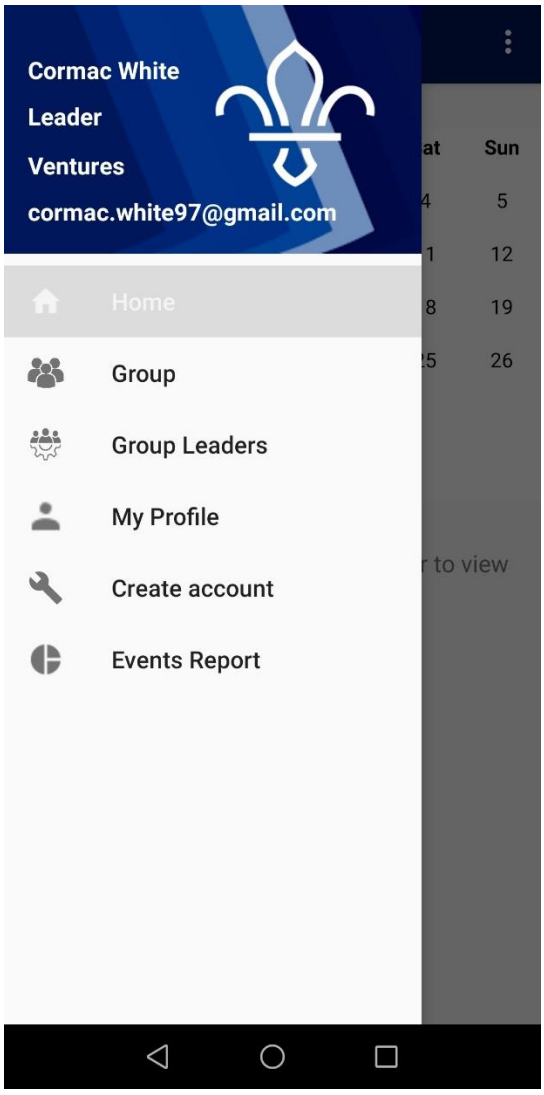

Figure 10

3.4.2 Event Calendar:

As I am allowing leaders to view all upcoming, regardless of their group, I felt it was important to distinguish each event based on their group. By implementing this, leaders are able to view other group's events, while easily being able to navigate to their own group's events.
To make the use of the calendar as clear as possible, I decided to add a TextView, giving a brief explanation on how to use the calendar, and how to view events.

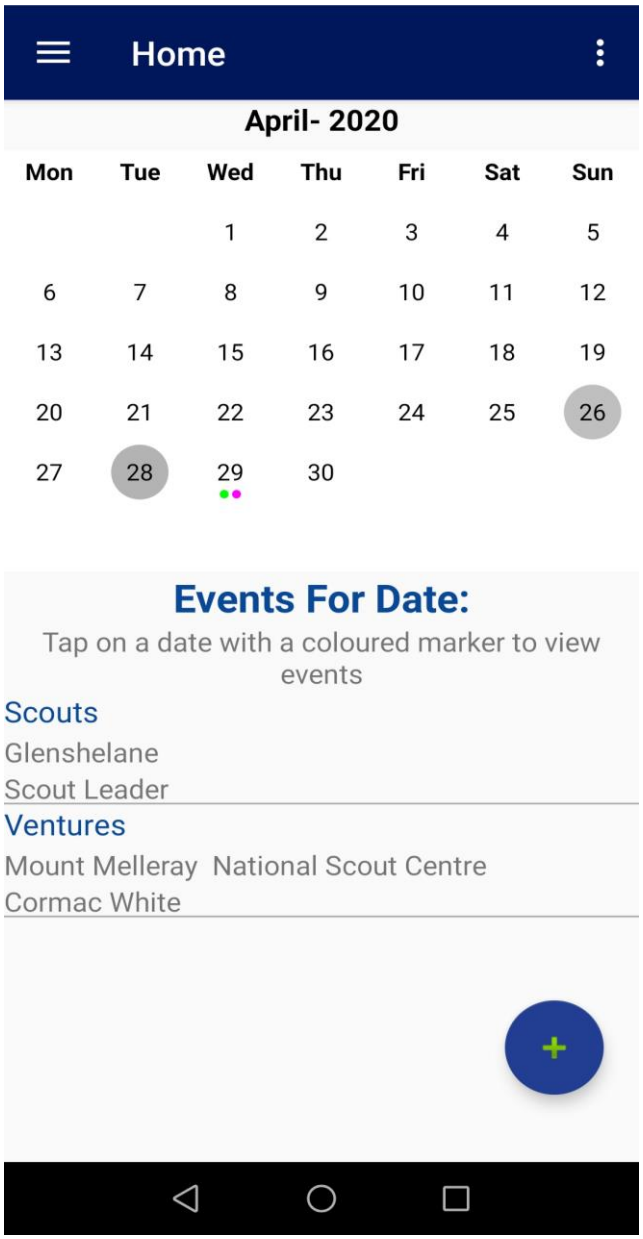

Figure 11

3.4.3 Map View:

When selecting a location for an event, there were a number of different ways in which I could have implemented this. My initial idea was to implement a dropdown list of all of scout centres around the country. To implement this, I would have had to create another table in the database, which would have stored a location ID, the name of the location and the longitude and latitude. I felt this would be overcomplicating things.

Instead, I decided to use Google Maps, paired with a KML file that is available on the Scouting Ireland website[17]. This KML file allows me to display each centre as a marker on the map. Each marker has an information page that is displayed when the marker is tapped, and when the information page is tapped, it opens the event creation page, passing in the longitude, latitude and the name of the location to that event creation page.

I am also using Google Maps to display the event location marker in the events details page, buy reading in the longitude and latitude of the event location from the database and adding a marker to that location. By adding this extra map view, I feel that parents would have a better understanding of where an event is being held if they can see it on a map, rather than just reading the name from a TextView.

The Google Maps API also allows users to get directions to a marker, by passing the marker to the Google Maps application, which I believe really helps to add to the usablity of the application.

Figure 12 below shows the Google MapView being used to show the user all of the Scout Centres around the country. *Figure 13* shows the MapView being used in the ViewEventDetails page.

[17] https://www.scouts.ie/Scout-Centres/Camp-Site-Locator/

Figure 12

Figure 13

3.4.4 Reviews:

When deciding on the UI for viewing reports, I felt it was important to display the ratings in a way that would jump out to the user. Even though I am storing the rating as an integer value in the database, I felt simply returning this value would not be as user friendly. To solve this problem, I implemented a RatingBar, which I am using to display a set of stars corresponding to the rating value. I felt that using this star rating would help to give the rating more meaning to users.

Figure 14

3.4.5 Reporting:

I felt that by adding an events report, I could allow leaders to view all of the events that each group is going on without needing to know the date of their events. This report allows me to display the number of events each group is going on and colour code them in a pie chart based on the group. The RecyclerView below the report allows users to quickly navigate to a specific event based the selected group.

I also added a bar chart to the page to view an event's reviews. I elected to do this as I felt it would make the review's much less abstract, and offer a clear visual representation of the event ratings.

Figure 15

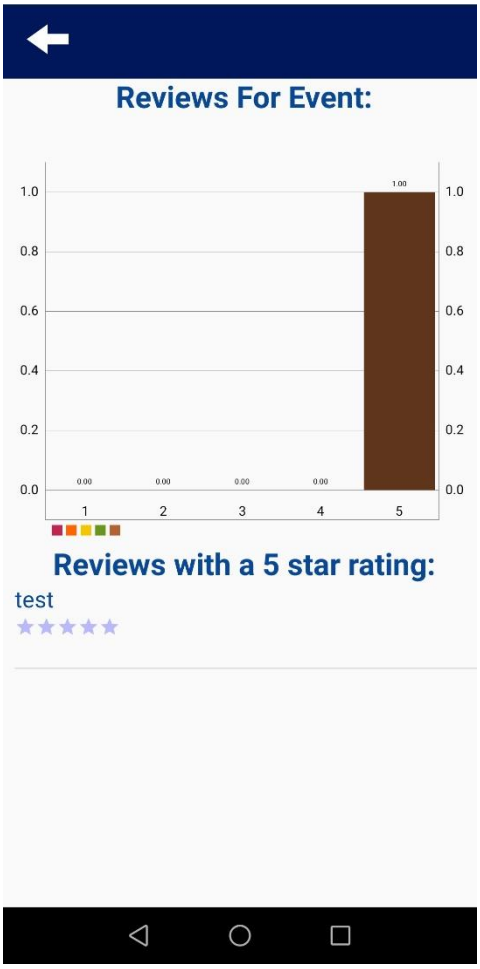

Figure 16

3.4.6 Input Fields:

As I was working on this application, it quickly became apparent that I would be very limited in relation to the amount of space I would have to implement the UI for this application. For this reason, I decided to use Google's TexInputLayout. This functions as a dynamic label for an input field, by setting the title of the field as the text until the user taps on the field. When it is tapped, the text becomes smaller and moves to the top left corner of the field. I chose to use this as I felt it was a good way to inform the user what the purpose of the field was without having sacrifice space to show a label.
The two figures below show the input field in its idle state and its state after it has been tapped.

Figure 17

Figure 18

4 Implementation

To create this application, there were a number of steps that I had to take. There were a number of features that relied on other pieces being implemented. For example, in order to create permissions for the different user types, I had to implement some form of a session to get the logged in account's type.

4.1 Overview of Implementation

4.1.1 Firebase:

At the time of me starting to create this application, I knew that the biggest knowledge gap that I would have was how to use the Firebase Realtime Database and the Firebase Authenticator. To bridge this gap, I used a Udemy Course[18], which covered all the basics of Firebase, from linking it to an application to making reads and writes, to and from the database.

4.1.2 User Login

The first piece of functionality that I implemented was the ability to log in. I started out by manually creating an account in the Firebase Authenticator, and then trying to enter the email and password into a sample log in page. I Chose to implement this first as this would be the first piece of functionality that the user would interact with when using the application.

Once I had this working, I started to work with the FirebaseUser feature, with functions as Firebase's own version of a session. Once I was able to write full objects to the Realtime Database, I linked the user ID's with the generated ID's in the Authenticator.

4.1.3 Event Calendar:

Once I was able to allow a user to log in, I began to implement the SundeepK Compact Calendar View. To learn how to use this API, I used the documentation on the its GitHub, as well as a YouTube tutorial[19].

Once I was able to get the API implemented in its most basic form, I then began to start adding colour coded markers to dates, in order to familiarize myself with the API.

I then started to read events from my database, and add them to the calendar based on the date of each event. When I had initially started developing this feature, I planned to only have each event last for one day. However, when I thought about this further, I quickly realised that this would not be realistic.

[18] https://www.udemy.com/course/complete-android-developer-course/learn/lecture/7665762
[19] https://www.youtube.com/watch?v=xs5406vApTo

After this decision, I began to implement events that would last over multiple dates, buy using the epoch date format that the API uses. I then decided to assign each group with a specific coloured marker, making the calendar much more useable.
Once I had the calendar working overall, I began to filter the event types that would be made available to the user based on their account type. I was able to do this by having two different addition statements, one for a leader account and one for a parent account, in which the event group must be the same as the parent's own assigned group and must be an approved event.

```
if (loggedInType.equals("Parent")) {
    if (group.equals("Beavers") && group.equals(loggedInGroup) && approval.equals("approved")) {
        ev2 = new Event(Color.BLUE, longDate, type);
    } else if (group.equals("Cubs") && group.equals(loggedInGroup) && approval.equals("approved")) {
        ev2 = new Event(Color.RED, longDate, type);

    } else if (group.equals("Scouts") && group.equals(loggedInGroup) && approval.equals("approved")) {
        ev2 = new Event(Color.GREEN, longDate, type);

    } else if (group.equals("Ventures") && group.equals(loggedInGroup) && approval.equals("approved")) {
        ev2 = new Event(Color.MAGENTA, longDate, type);

    } else if (group.equals("Rovers") && group.equals(loggedInGroup) && approval.equals("approved")) {
        ev2 = new Event(Color.BLACK, longDate, type);


    }
    if(ev2 != null){
        calendar.addEvent(ev2);
    }
}
else if(loggedInType.equals("Leader")){
    if (group.equals("Beavers")) {
        ev2 = new Event(Color.BLUE, longDate, type);
    } else if (group.equals("Cubs")) {
        ev2 = new Event(Color.RED, longDate, type);

    } else if (group.equals("Scouts")) {
        ev2 = new Event(Color.GREEN, longDate, type);

    } else if (group.equals("Ventures")) {
        ev2 = new Event(Color.MAGENTA, longDate, type);

    } else if (group.equals("Rovers")) {
        ev2 = new Event(Color.BLACK, longDate, type);


    }
    calendar.addEvent(ev2);
}
```

Figure 19

4.1.4 Google MapView:

When adding the MapView to this application, I initially started by adding the Google Maps API to an empty activity. Once I had this implemented, I Added an addition button to the home page, to allow users to access the MapView when a date is clicked on the calendar. To use the Google MapView, I had to register my Google account with the Google API library. Once the registration was complete, I was then able to get a generated API key, which I added to my AndroidManifest. I then added the dependency to me build.gradle, which allowed me to make calls to the Google Maps API.

I then began to add the KML layer that contains the locations of each scout centre around the country to the MapView. I then added an onClickListener to each information window that is attached to each marker. The purpose of the onClickListener is to take the user to the final page, in terms of event creation, with the necessary details of the location being passed over.

As well as this, I added a second MapView to the ViewEventDetails page, which animated and zoomed to the location of the marker.

4.1.5 Viewing Events:

This page would be one of the pages that would have the highest amount of business logic. Most of this logic comes from deciding whether to show certain buttons on the page based on the account.
To make this decision, as well as reading in the event details from the database, I also read in the logged in user's details. I did this by creating a list of strings, which would contain the two account types (Leader or Parent), and then iterated through this list, setting the reference statement to the account type.
I was able to find the logged user's details by comparing the object's ID that was being read from the database with the Firebase User's ID. I then set the visibility of each button to invisible in the onCreate method, and only showed them if the user's details met certain conditions. For example, if a user is logged in as a parent, the system will check whether or not their assigned child's ID is on the payment list or not. If a payment has not been made, a button prompting them to make a payment will appear, otherwise if the event's end date has not passed, a message will appear, informing the user that a payment has already been made.

4.1.6 PayPal API:

This was one of the most important features in this application. To implement it, I added a button to the ViewEvents page, which would appear if a parent has not already made an account. Once I set up my own PayPal account with the API, I received a client ID, which I added to my PayPalConfiguration object. I then added the PayPal dependency to my build.gradle This then allowed me to make an intent to the PayPal payment screen.
To get the response from the API, I overrode the onActivityResult method, where I was able to add the child ID to the payment list if the payment was successful and write a PaymentHistory object to the database,

4.1.6 Reporting:

This was one of the last features that I implemented in this application. I chose make this one of the last features to be implemented, as I wanted to follow the process that an event would follow, from its creation, to the payment, to creating and viewing reviews and viewing reports about events.
To implement this feature, I simply had to add the dependency to my build.gradle. As there are a number of different charts that come with this API, I used tutorials[20] [21].
As the labels for these reports would not change, I was able to hard-code them into the code. For example, in the events report, I hard-coded the group types for the labels, I then read every event from the database and added them to and ArrayList. I then iterated through this array list, and incremented a counter for each group, depending on what the group type was.
To display specific group's events when their section of the pie chart was clicked, I used the onValueSelected method. Using this method, I was able to get the label of the selected section, loop through the events ArrayList and add the event object to another ArrayList called myDataset, if the event group was equal to the selected label. Once I had all of the selected group's events, I then reset the RecyclerView and set the data to be the myDataset ArrayList.

[20] https://www.youtube.com/watch?v=iS7EgKnyDeY
[21] https://www.youtube.com/watch?v=naPRHNfzDk8&t=494s

4.2 Issues and Resolutions with Implementation

The main issue that I came across the most when working on this application, was my own inexperience with the technologies I was working with. Using Firebase would be the main area where this problem arose, in particular, the use of the Realtime database.

During my time in this course, the main database that I would have used would have been SQL. While I had some small experience with the JSON format, I knew I did not have enough to confidently work with the Realtime database. To solve this issue, I spend a few days working with tutorials [22] and Firebase's own documentation[23], practicing making reads and writes, and getting to grips with the format this databases uses.

As I was implementing the PayPal API, I found that I slightly underestimated the amount of time it would take to get it fully working. However, I feel that it was worth the time spent, as it was a popular service, that many parents and leaders were already familiar with.

When reflecting on the development of this application, an issue that occurred time and time again, was the issue of time management. The main issue that I found was trying to balance this project with other assignments and assessments for other modules in this course. While I did find this difficult at times, through me weekly meetings with Catherine, I was able to find a balance between the project and my other course work.

There were times during the development of this application, that I began to become unsure of what the best way of implementing features was. While I constantly kept the usability of this application in mind, there were times when I was unclear of what certain users would want from this application. Luckily, as I am a leader in a Scout group myself, I had a long list of contacts that I could get feedback on ideas from, giving me a real-world insight into how leaders would want to use this application. As well as this, as I was also a member of a Scout group over many years, my own parents had their own experience in being a Scout's parent. Their valuable insight also helped me to get into the mind-set of a member's parent. This meant that I could get constant feedback from the two use types that would use this application

[22] https://www.udemy.com/course/complete-android-developer-course/learn/lecture/7665762
[23] https://firebase.google.com/docs/android/setup

5 Test Plan

5.1 Tests:

Due to the nature of this application, there were a number of features that could not be automatically tested. For example, to be able to test the isLeaderAvailable method in the CreateNewEvent class, I would need to call on the onCreate method in the class to get the existing events list, but due to the fact that this is a protected method, I cannot call on it from my testing class. It was for this reason that I had to run majority of testing for this application manually.

5.1.1 General Tests:

#	Name	Description	Expected Result	Result
1	Login	The user should be able to log in with the correct details.	The home page is displayed	PASS
2	Check details	Ensure that the right details are appearing in the top of the navigation drawer.	The user's name, group, account type and email should appear.	PASS
3	Log out	Check that the user can log out of the application.	The user should be logged out and returned to the login page	PASS
4	View Own Details	A user should be able to view their own details.	A page with the user's details should appear	PASS
5	Update details	Check that user details can be updated and saved	The user's details should be changed in the database when they update them.	PASS
6	View list of group leaders	Check that a user can view a list of leaders in their own group.	A list of leader's names in the user's group should appear	PASS
7	View leader's details	Ensure that the user can view the details of a leader in their group	A page containing the leader's details should appear	PASS
8	View event details	A selected event's details can be viewed	The event's details should be displayed	PASS

5.1.2 Leader tests

#	Name	Description	Expected Result	Result
1	View Events	Check that a leader can view all events.	The calendar should display all of the events in the database	**PASS**
2	View Group	Check that the user can view a list of members in their group	The user's name, group, account type and email should appear.	**PASS**
3	View Members	The user should be able to view their group's members.	A list of members in the user's group should appear.	**PASS**
4	View Member Details	Check that a selected member's details can be viewed.	A page with the selected member's details should be shown	**PASS**
5	Update Member Details	Ensure the user can update the details of their members	The updated member's details should be updated in the database.	**PASS**
6	Create Account	Ensure that the user can create a new account for a leader, parent or member.	The newly created account should appear in the database.	**PASS**
7	Assign member to parent	Ensure that a member's ID is assigned to their parent	The member's ID is set as the childID in their parents account and the parent's group should be set to the child's group.	**PASS**
8	View Events Report	Check that the user can open the events report	A pie chart should be shown with a segment for each group, with a RecyclerView shown below	**PASS**
9	Event Creator	Ensure that the user has the appropriate permissions if the event is still pending and they are the event's creator.	The delete, edit and make available buttons should appear on the screen.	**PASS**
10	Approve Event	Check that the user can approve an event.	If all of the leaders are set as approved, the event is set to approved and available to parents	**PASS**
11	Assigned to an event	Tests that a leader who has been assigned to an event can indicate their availability.	Two buttons should be shown to the user when they view the event's details allowing them to indicate their availability.	**PASS**
12	Accept Invitation	The user is able to accept the invitation to attend the event.	The user's value in the eventLeaders list should be set to "Approved"	**PASS**

5.1.3 Event tests:

#	Name	Description	Expected Result	Result
1	Add an event	The user should be able to add an event to the calendar	The addition button should appear when a date is picked	**PASS**
2	Select a location	Check that the user can select an event location.	The user should be able to select an event location by tapping on a marker on the Google MapView	**PASS**
3	Check passed in details	Ensure that the details passed in from the previous pages are correct.	The start date that was selected in the home page and the event location that was selected in the Google MapView should be shown.	**PASS**
4	Create event with no details.	Check that a user cannot create an event with no details	A Toast message should appear informing the user that they need to fill in all of the details	**PASS**
6	Create an event with one leader	Create an event and only assign the event creator as the attending leader.	A Toast message should appear telling the user they need at least two leaders for an event.	**PASS**
5	Update event details	Check that the user can update the event leaders and the price.	The updated details should be set in the database.	**PASS**

5.1.4 Parent tests:

#	Name	Description	Expected Result	Result
1	View Events	Check that the user can only view events that are assigned to their own group	The calendar should only show the events for the user's own group	**PASS**
2	Payment	Check that the user can make a payment	The user should be able to make a payment and their child should be added to the attendee list.	**PASS**
3	Option to add review	Check if the user is eligible to leave a review if they had paid for the event and the end date has passed	The button to add a review to the event should be shown.	**PASS**
4	Add a review	Add a review for an event	A review should be added to the database and be visible.	**PASS**
5	View Reviews	Check that the user can see an event's review if they have left a review.	The button to see the event's reviews should be visible.	**PASS**
6	View Child Details	Ensure that the user can view their own child's details.	A page should appear with the user's child's details	**PASS**
7	Edit Child Details	Check that the user can edit their child's details.	The child's details should be updated in the database.	**PASS**
9	View leader's details	Ensure that the user can view the details of a leader in their group	A page containing the leader's details should appear	**PASS**
10	View event details	A selected event's details can be viewed	The event's details should be displayed	**PASS**

5.2 Usability Testing

As the intuitiveness of this application is of vital importance, my initial plan was to show the application to other leaders in my group and time how long it took them to perform the most common tasks in the application, such as creating a new event. Unfortunately, due to the recent government lockdown, I was unable to meet with any of my fellow leaders.

Fortunately, my mother currently works as a Cub leader, meaning that she would have the valuable insight that I was looking for from the leaders in my group. To test the usability of my application, I used myself as a benchmark and asked my members of my family, who have never had any experience with this application. I asked them to perform the list of tasks that I was originally going to ask the leaders in my group to do, and see how long it took to complete with no initial instruction.

The table below lists the tasks that were completed for the purposes of this test, along with the time in minutes that it took each person to complete it. I used myself as a benchmark, as I have the most experience with the application.

There are a number of different variable that can affect the speed in which these tasks are completed, such as typing speed or internet connection, but the wide variety of technical abilities that give a good indication of the intuitiveness of the application.

Before each of these tasks were completed, they were logged into the appropriate accounts.

	Benchmark	User 1	User 2	User 3
Create an event	0.33	0.52	1.33	0.58
Create an account	0.31	1.07	0.53	0.53
Indicate availability	0.06	0.09	0.09	0.04
Make a payment	0.32	1.26	1.33	1.04
View specific leader	0.02	0.06	0.15	0.04
Write a review	0.08	0.24	0.27	0.12
View reviews	0.03	0.07	0.8	0.04

As well as being able to gain a better understanding of how usable my application was, I was also able to get some good feedback on how to make to application more usable to someone who has never seen the application before.

6. Future Plans:

While I feel I was able to implement most of the features that were required for this application, there are a few features that I was not able to implement within the timeframe that I had, over this year, that I believe would have improved the usability and functionality of this application.

When I began the development of this application, I had originally intended to add a group equipment ordering system. While I still believe that this would be a useful feature, I somewhat underestimated the complexity and time consumption of the features that I was able to implement, which resulted in me not being able to implement this feature. If I were to continue working on this application, I would most-of-all like to see this feature implemented, as I spent time working as a quartermaster for my own Scout group and have had first-hand experience with the equipment ordering process. The group I am in currently use an ordering system on a Wordpress website. I feel that implementing a mobile based ordering system, would make equipment orders for both leaders and the group's quartermaster much more accessible.

One feature I feel could be improved is the way in which the application sends notifications. Currently, to send notifications I am using SMS notifications from the user's phone. I had originally intended to use the Firebase Cloud Messaging service, which uses Node.js to send push notification's based on certain triggers. While I was able to successfully implement this, I found that it was quite unreliable, often not sending the notification when it was called on. If I were to continue to work on this application, I would like to look at alternatives for sending notifications.

If I were to continue to work on this application, I would like to develop it to a point where it could actually be used by my group, and solve the real-world problem it was designed to resolve.

As usability can always be improved, I would do more research on ways in which I could make my application easier to use. While I feel that I have gotten it to a stage where event users who have never used the application can quickly figure out how to use it, there will always be ways to improve this application.

7 Conclusion

Through my own experience and my conversations with various leaders in my group, I have discovered that there is defiantly a gap in the market for an application. Although it is quite a niche market in relation to the global application market, I believe this application could provide an important service to the 13,000 adult volunteers and the parents to 40,000 youth members[24].

Overall, I am happy with the application I have produced. While there are some features that I would have liked to implement, I feel I was able to implement most of them to a high standard. I believe the features that I have implemented in this application solve a real-world problem, that I have had first-hand experience with.

As I was developing this application, its complexity quickly grew. While this application became more complex than I had originally intended, I thoroughly enjoyed the challenge it posed. Not only did it give me the opportunity to gain a better understanding of new technologies such as Firebase, it also gave me a better understanding of the importance of setting targets and deadlines. I found that by setting my own deadlines, I was able to create a healthy sense of urgency, which stopped me from getting too invested on one feature or another. As this was, by far the longest time I have spent on a project during my time in this course, I quickly had to learn to view this project as a marathon rather than a sprint.

I feel that this project has been a major benefit to me, by allowing me to develop a number of skills from this course, as well as through my own self-directed learning.

[24] https://www.rte.ie/news/ireland/2018/0625/973048-scouting-ireland/

8. References

- https://play.google.com/store/apps/details?id=de.schwarzstoff.norcamps
- https://www.google.com/calendar/about/
- https://play.google.com/store/apps/details?id=org.scout.android
- https://www.scouts.ie/Scouter/Management-Resources/Policies-Procedures/All-Official-Policies/SID-39B-10-Garda-Vetting-Policy.pdf
- https://www.thetimes.co.uk/article/scouting-ireland-sued-over-historical-sex-abuse-claims-76px7qjgq
- https://vertex-academy.com/tutorials/en/history-of-java/
- https://www.androidauthority.com/develop-android-apps-languages-learn-391008/
- https://howtofirebase.com/what-is-firebase-fcb8614ba442
- https://www.udemy.com/course/complete-android-developer-course/learn/lecture/7665756
- https://www.carbibles.com/best-gps-apps/
- https://www.androidheadlines.com/2019/06/best-android-navigation-apps-2019.html
- https://github.com/PhilJay/MPAndroidChart#documentation
- https://play.google.com/store/apps/details?id=com.uniquepublishing.presentationcollegebray&hl=en_IE
- https://www.scouts.ie/Scouter/Management-Resources/Policies-Procedures/All-Official-Policies/SID-39-05-Code-of-Good-Practice.pdf
- https://www.scouts.ie/Scouter/Programme-Scouter-Stuff/ONE-Programme/

- https://www.scouts.ie/Scout-Centres/Camp-Site-Locator/
- https://www.youtube.com/watch?v=xs5406vApTo
- https://www.youtube.com/watch?v=iS7EgKnyDeY
- https://www.youtube.com/watch?v=naPRHNfzDk8&t=494s
- https://www.udemy.com/course/complete-android-developer-course/learn/lecture/7665762
- https://firebase.google.com/docs/android/setup
- https://www.rte.ie/news/ireland/2018/0625/973048-scouting-ireland/