CrewManagerApp

by

Muhammad Khan

CANDIDATE No: 33539295

SUPERVISOR: Dr.Golnaz Badkobeh

A report Submitted for the degree of Bsc. Computer Science

Date 12 June 2020

ABSTRACT

Shogun is an event crew company based in London that offers event crew services around the UK. The main ethos of the company is to ensure that their clients meet their project objectives with ‘efficiency and excellence’. As consumer behaviour continues to evolve, companies such as Shogun must adapt and therefore require a mobile application that can help facilitate and enhance communication between their managers and crew members. In this project, an IOS application named 'CrewManagerApp' was developed to fulfil this requirement. Which can not only be used by Shogun but also other companies that offers similar services. CrewManagerApp is an IOS Application which simplifies job distributions and communications between manager and crew members. It is built-in programming language swift and uses Cocoa touch IOS frameworks. The views are built-in storyboard and the backend utilises firebase.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my project supervisor Dr. Golnaz Badkobeh for all the support and genuinely helpful advice she has given to me throughout this project, her insightful encouragement and continuous push has enabled me to challenge myself and learn new things.

I would also like to thank my parents who have always supported me throughout my academic studies and have always encouraged me to do my best.

Table of Contents

STRUCTURE OF THE REPORT

Chapter 1: This chapter provides an introduction to the application; it presents the current problems and a motivation on how the current problems can be resolved.

Chapter 2: This chapter includes a background and research which includes the existing system along with core aim and objective of the project.

Chapter 3: This chapter includes the System design decisions made in the development of the project which consists of the application user guide and documentation.

Chapter 4: This chapter includes the implementation of core features of applications.

Chapter 5: This chapter includes a detailed testing undertaken, user testing, white box testing and black box testing.

Chapter 6: This chapter includes self-evaluation,

Chapter 7: This chapter includes a detailed conclusion of the project and future developments.

Chapter 1

1. INTRODUCTION

The UK event industry is worth £42.3 billion, 35% of the UK visitor economy is accounted for by events. There are over 1.3 million events held in London each year.[1] For each of these events, companies require event crews who can assist with the rigging and derigging of lighting, sound equipment and scenic elements of productions and events in performance spaces. Essentially, event staff support the logistical management of the corporate and public events. There are over 100 events crew companies that operate in London.[2]

Shogun is crew company based in London and offers event crew services across the UK. The company currently has over 60 crew members. Other than shogun, there are a number of other event crew companies such as Silverback, Crewsaddars and Affinity Crew of which offer similar services. The main service these companies provide is manpower, who help in rigging and derigging during events.

Most of the event crew companies operate in a similar manner. To book a crew for an event, the client is required to email the event crew company. Upon receiving the email, the company crew manager either accepts or rejects the job request. A description and instructions from accepted jobs will then be distributed to crew members via phone calls, text messages and WhatsApp. Crew members either accept or reject the job sent by the manager. Individuals who accept the job often manually make a note of the job's date, time, location and number of working hours. These individual's data also entered into a Microsoft excel spreadsheet by the company's crew manager.

Currently there is no custom-built mobile phone application in place which offers functionalities that can help event crew members manage their day-to-day tasks through the use of a single application. The competition between the event crew companies is on the rise, therefore there is considerable commercial interest in an application that allows such companies to perform their daily tasks using one application. CrewManagerApp will enable crew companies to move away from current, third-party communication methods (WhatsApp, text messages and phone calls, excel) for managing their crews ultimately making their jobs more convenient.

CrewManagerApp is an IOS Application which simplifies job distributions and communications. Mangers will have the ability to can send a job requests, add/remove crew members, check staff availability and create time sheets. Crew members will have the option to either accept or reject jobs request sent by their manager, view their accepted jobs, set their availability preferences and ultimately receive real-time updates on one centralized platform.

1.1 Problems Outline

Managing crews during events is a challenging operation. The process of distributing information related to jobs, recruiting crew members, allocating an array of individual tasks and manually transferring data to a spreadsheet can be demanding. Consequently, attempting to achieve the targets set out above using multiple applications can be cumbersome and inefficient, especially in larger organisations or during busy periods.

Currently, a crew manager is required to manually send a text or call each and every crew member in order to ask for their availability. The details of crew members who either accept or reject a job are then then manually entered into an excel spreadsheet; a process that is inefficient and potentially unreliable given the risk of human error. Additionally, managing crew members via text message can also result in a double-booking to occur whereby one individual could get booked on two different jobs at the same time – causing complications between the client and event crew company.

Having worked in the crew industry myself, I have found that current system very unorganized and time-consuming. I have had to install several different applications to my phone in order to communicate with the crew manager; keep track of my work shifts and display my availability to my manager. Crew manager's may experience difficulties while creating timesheets for their crew members. The timesheets are currently created manually using excel for each crew member, which could potentially increase the chance of human error while entering data crew worked hour into the excel spreadsheet. In some cases, this can result in crew member being paid the incorrect amount.

1.2 Aims and Motivation

My motivation for this project was based on my cousin who owns a crew company – Shogun – and also from my personal experience while working as a crew member. My cousin often experiences the problems outlined above, which made it clear to me that both managers crew members currently experience problems. I conducted a number of interviews with other crewing companies' managers and they were also in the same situation as my cousin. Consequently, this encouraged me to 'fill the gap in the market' by developing an application that meets the needs of crew members and crew managers.

The aim and objective of this project is to create a useful tool for event crew companies that will help facilitate and enhance communication, productively and efficiency between managers, staff members. This will be achieved by developing an all-in-one application – CrewManagerApp – that allows crew managers and members to create and schedule timesheets, send direct messages one another, set their availability status and ultimately users to perform their day-to-day tasks more efficiently and easily.

Chapter 2

2. BACKGROUND RESEARCH

This chapter provides a literature review and evaluation of existing smartphone application's currently utilised within the event crew industry, followed by a description of how CrewManagerApp seeks to address the limitations of current applications.

2.1 Motivation and Justification

Smartphones have fundamentally changed people's lives; they are not only used to communicate with each other using phone call, text messages and other mediums. Today, the endless functionalities in smart phones are integrated into one, such as taking pictures, making notes. Smart phones are a revolutionary technology that continues to evolve. According to Statista, more than half of the world's population currently owns a smartphone. Smart phones have become part of people lives; a survey conducted by Bank of America shows 96% of millennials aged 18 to 24-year-old said that smart phone is very important for them[3]. This project looks to use the features that smartphones offer and create an application that uses and offers more functionality to make it even more useful for the crewing industries.

2.2 Existing Technology

Building an understanding of the current market is essential to help identify deficiencies in application's system and design. Furthermore, this also supports the process of producing a superior application with greater enhanced functionalities. Several applications are frequently used within the event crew industry that delivers a similar system workflow to CrewManagerApp. There are some bespoke web-based applications used by well-established crewing companies such as Silverback and Gallowglass to manage their crews. Understanding the current marketplace helps to find gaps in the functionality of the current system features, which will enable me to implement enhance features in my application which the existing application does not have. Currently, in the events crew industry, some of the companies that use web-based applications can only be used on computers. They do not offer users the ability to download the application to their phone – a feature that encourages flexibility.

2.2.1 Gallowglass Client App

Gallowglass Client App[4] is a bespoke web application used by Gallowglass event crew company. The application shows crew member's details, photographs, and contact numbers, along with job timing, meeting places, and venue information for upcoming events. The unique feature of this app is the custom build application for the crew company, it has most of the features needed by the crewing companies to manage their crew members and clients efficiently. It allows the users to register using their details such as their email address and password. Once the user logs in using their credentials, the app navigates them to a dashboard where the user can see upcoming events, and it shows a list of crew members along with job timing and the date. It allows the crew manager to update and edit the crew member's details. This application makes it easier for Gallows crew company managers to manage their crew members due to all the essential features being built into one application.

However, this application lacks two important features: messaging and creating timesheets. To contact the crew, the crew manager must use another application to

communicate with the crew. Additionally, to create timesheet the manager is required to take all the data from the app and transfer it to a spreadsheet; which is time-consuming and could increase the chance of human errors to occur.

The colour scheme chosen for the application is very dull and the application can only be used on the web. Adding additional features such as sending messages to the user from the same app and creating a timesheet for each employee can make the application more useful. The application could be improved by allowing the users to use it on multiple platforms instead of only being able to use it on the web. CrewManagerApp aims to resolve these issues.

2.2.2 Event Staff App

Event Staff App[5] is a web application that offers similar functionality to CrewManagerApp. The unique feature of this is that it allows the users to send and receive messages within the application. The user no longer needs to make phone calls and emails to their event staff. To use the application the user is required to register as a manager or an employee and log in using their credentials. Once the user logins as a manager, they are directed to the dashboard shown in **Figure 2.1**. It offers a variety of features such as deals with schedule conflicts, calendar, creates timesheets, and allows the users to set their availability. Using Event Staff App will make everyday tasks very easier for the crew managers because it automates all the task managers are currently doing manually.

However, there are several issues with the app. Currently, the application is only available in the US and Canada showed in **Figure 2.2.** This creates room for the UK market. Finally, this application is only available on the web. It cannot be used on smartphone devices. It could be made into a multi-platform application.

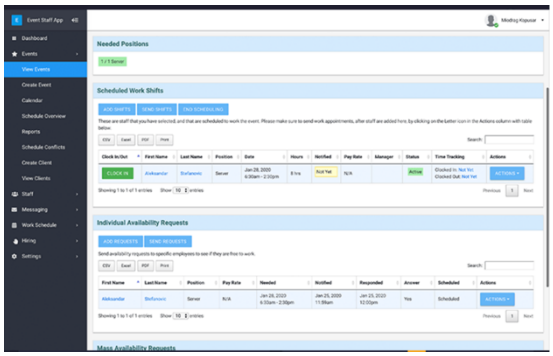


Figure 2.1. Shows event staff Dashboard

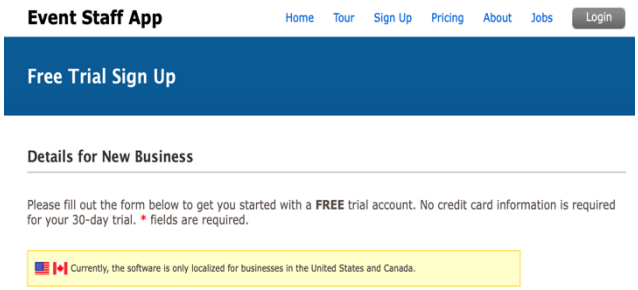


Figure 2.2. Shows event staff can only be used in US and Canada

2.2.3 Google Calendar

Google calendar[6] is considered to be one of the most popular calendars available for users. It's a free web and mobile calendar that lets the user keep track of their events and allows the user to share calendars with others, it's an excellent tool for managing personal and professional schedules. It also provides other great functionalities; Events such as hotels, holidays or any other reservation, are automatically added to users' calendars. It also allows the users to set reminders and create a 'to-do list' alongside their planned events.

During my research, I discovered most of the event crew members use Google Calendar to keep track of the events and their jobs. Using Google Calendar helps to prevent the crew member from double-booking – an issue that frequently occurs during the busy season.

Google calendar's interface is user friendly and very easy to use. It is simple with Google's characteristic pastel blues and yellows. CrewManagerApp will use Google calendar's primary feature as inspiration. Instead, jobs accepted by crew members will automatically be added to their calendar. They will also receive notification about upcoming jobs.

2.3 My Contribution

CrewManagerApp will seek to combat the limitations of current applications available on the marketplace by providing several exclusive tools that will facilitate event crew managers and crew members. This will be accomplished by allowing crew managers to directly send job requests to crew members and enabling crew members to stay up to date with their work schedule through the provision of a timesheet and work calendar.

CrewManagerApp will deliver several additional benefits compared to its competitors. Unlike 'Event Staff', CrewManagerApp will be available in the UK. CrewManagerApp also allows its users to send instant, direct messages to manager and create timesheets – a feature that is currently absent in 'Gallowglass Client'. Moreover, contrary to web-applications, CrewManagerApp will be available on smartphones – encouraging user flexibility. Finally, CrewManagerApp will seek to incorporate a user-friendly, smooth UI which will enable its users to learn the system quickly and use it efficiently.

Ultimately, CrewManagerApp will enhance communication, productively and efficiency between managers, staff members, and clients by offering unique or improve functionalities in applications currently available on the marketplace.

2.4 Questionnaire Results

To gain a better understanding of what features users would find most useful in CrewManagerApp, two separate questionnaires were conducted between two focus-groups. The two focus-groups include crew members (1) and crew managers (2). Results from the questionnaire will help identify any flaws or additional practical features that could be added to CrewManagerApp. The questionnaires can be found in **Appendix A and B.**

2.4.1 Event manager

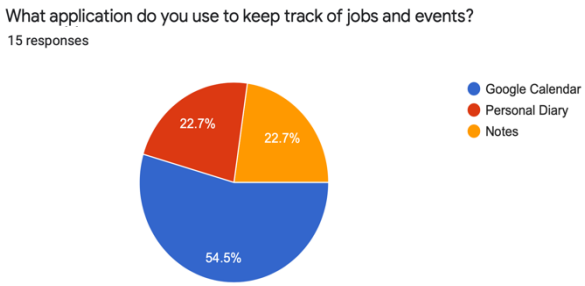


Figure 2.3. Applications currently used to keep the track of jobs and events.

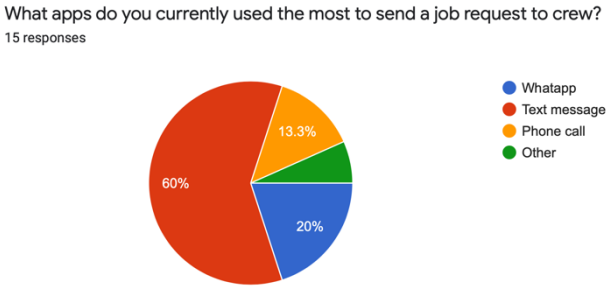


Figure 2.4. Applications currently used to send job requests.

Figure 2.3 shows that the majority of the users (54.5%) use Google Calendar to keep track of jobs and events. **Figure 2.4** shows that 60% of the jobs are distributed over a text message. These results suggest that potential users are likely to benefit from an application that incorporates both a calendar and messaging feature into my application.

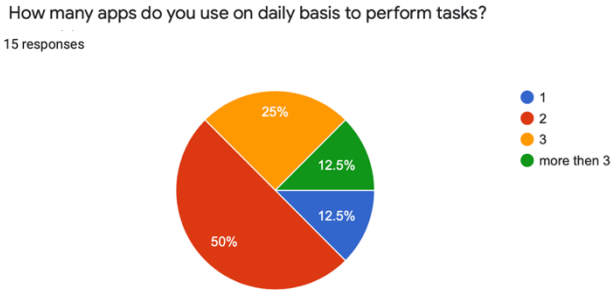


Figure 2.5. Applications used on a daily basis to perform tasks.

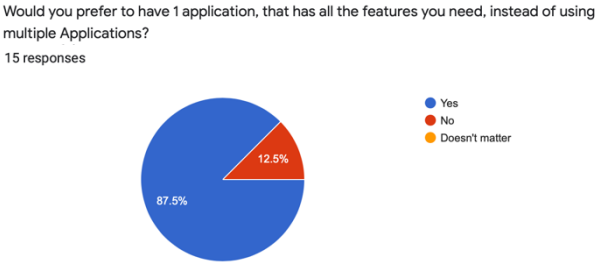


Figure 2.6. Users preference on how many Application they prefer to use.

Figure 2.6 Shows that the majority of users (87.5%) use at least 2 applications to manage their tasks and events. Based on the above data, potential users are likely to benefit from an application that allows users to manage their tasks through a single application.

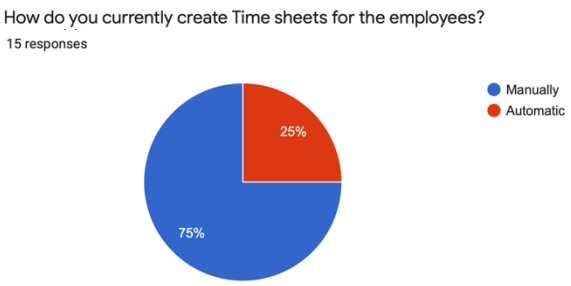


Figure 2.7. Current methods used to create Timesheets.

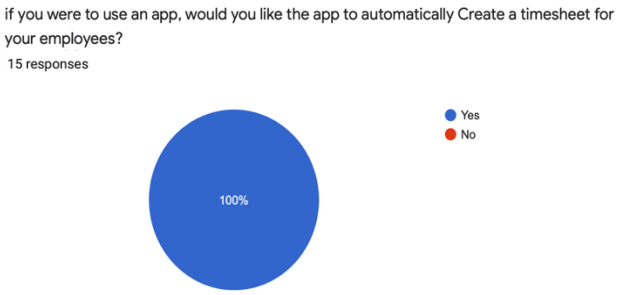


Figure 2.8. User preference on timesheet being created automatically.

Figure 2.7 highlights that the majority of the users (75%) currently create timesheets manually and in **Figure 2.8** all the users would prefer an application that can create a timesheet for each crew automatically. By looking at the data I will aim to add a feature to the app which will make it easier for managers to create timesheets.

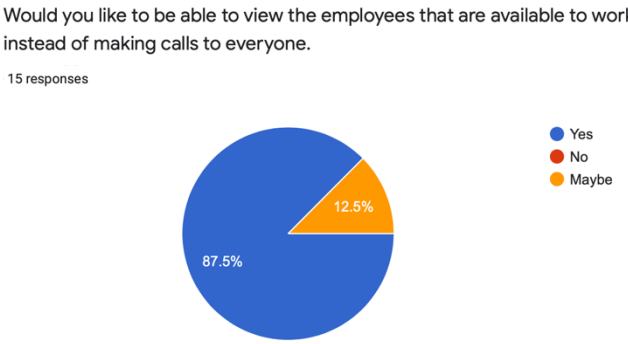


Figure 2.9. Shows whether the crew manager would like to view all the available crews.

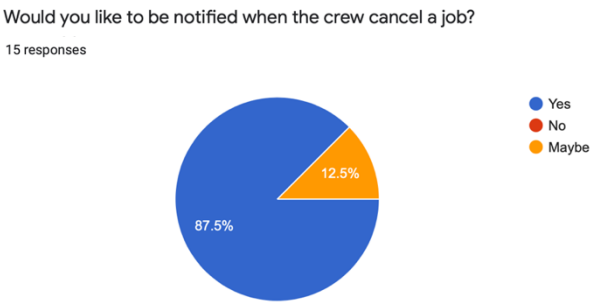


Figure 2.10. Shows whether the manager prefers to be notified when the crew member cancels a job.

Figure 2.9 highlights that crew managers would prefer to have a feature that enables their employees to set their work availability statues. Besides, the results show that the majority of managers would also like to be notified when a crew member cancels a job.

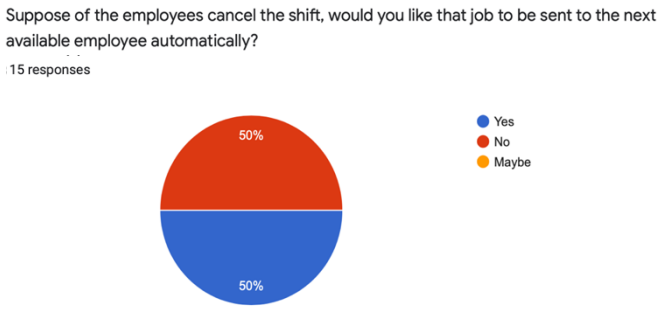


Figure 2.11. Shows whether the user would like cancel shift to another employee automatically.

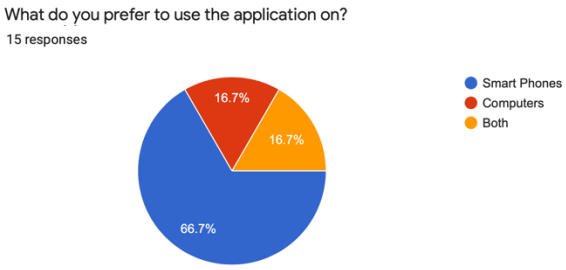


Figure 12.12 Shows user preference for using the application.

Figure 12.11 shows that 50% of crew managers would like to have an option whereby cancelled job requests are automatically transferred to available crew members. The Question in **Figure 12.12** was asked to find out what platform crew members would prefer to use the

application. The pie chart displays that the majority of the users (66.7%) would prefer to use the application on a smartphone.

2.4.2 Event Crew Members

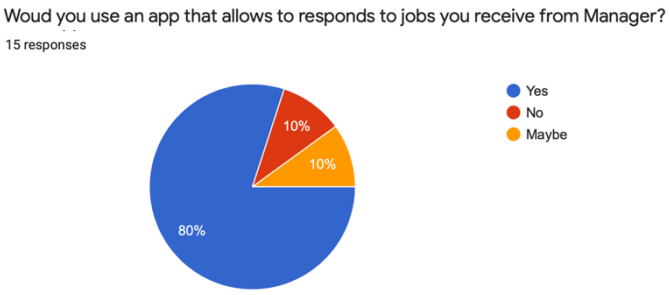


Figure 2.13. Shows whether the users would like to use an app that allows them respond to the job.

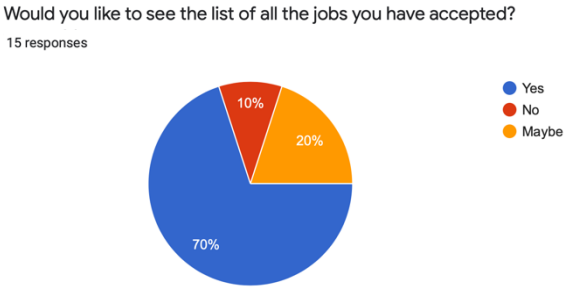


Figure 2.14. Shows whether the users would like to see the list of all the jobs accepted.

The questions in **Figure 2.13 & 2.14** were asked to get a better understanding of what additional features could be added to the application to make it more useful for crew members. The majority of the respondents (70%) would like to be able to view the jobs they have accepted and responds to the job.

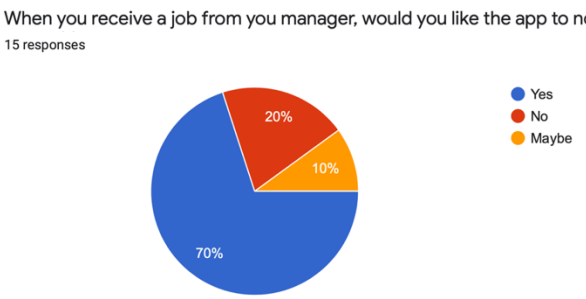


Figure 2.15. Shows whether the user would like to be notified if they receive a message.

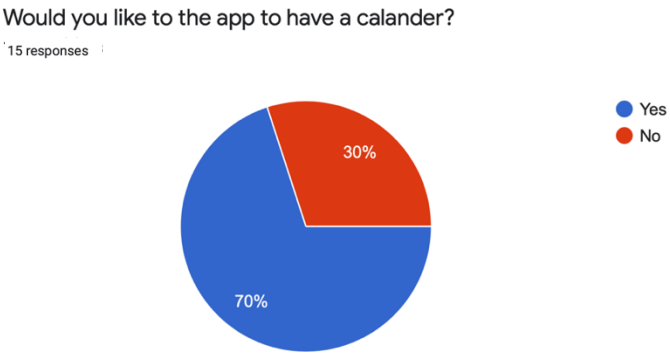


Figure 2.16: Shows whether the users would like to have a calendar in the app.

Questions in **Figure 2.15** was asked to find out if the crew members would like to be notified when the manager sends a job. As can be seen, (70%) of the respondents would like the app to have this feature. Furthermore, crew members were also asked to find if they would like the application to have a calendar, **Figure 2.16** highlights that the majority of users (70%) agreed to implement a calendar into the application.

2.5 Technology Research

2.5.1 IOS Development Technology

iOS stands for the iPhone operating system. It is an exclusive mobile operating system of Apple devices. To develop an iOS application there is a diverse set of tools and programming languages such as swift and objective C which can be used to develop an application to run on iPhone and iPad devices.

2.5.2 Programming Language

To develop an iOS application, there are two languages that the developer can choose from Objective-C and Swift. Objective-C was the first language for iOS application development, and it was successfully used to implement iOS apps until Swift was introduced.

Swift

Swift is a new programming language launched by Apple in 2014, which is used to implement iOS applications[7]. It was developed for watchOS, iOS, macOS, tvOS, and Linux operating systems, it gives developers an alternative to Objective-C. Swift is compatible with all of the existing iOS development tools such as Xcode, Objective-C, Cocoa, and Cocoa Touch frameworks.

2.5.3 Xcode

Xcode is an integrated development environment (IDE) used to implement iOS applications[8]. It provides users with the required tool they need to implement and manage source files, iOS projects, assembles user interface, and building codes into an executable file and run. It allows users to debug the code in iOS simulator or on a device, the developer can set breakpoints. Xcode incorporates many features to make the implementation of an iOS application easier, which include the following:

- **Interface Builder:** Interface builder is fully integrated within the XCode IDE, it allows users to implement and test user interfaces without writing any code, then graphically connect the interface to the source code within the Xcode editor, with the assistant editor it allows you to work on the graphical design side-by-side with development source code.

- **Fix-it:** In Xcode when the user makes any coding mistakes, they are notified and are provided with an auto-correct feature.

- **Simulator:** Using iOS SDK, Xcode can install, build, run, and debug the Cocoa touch app in a Mac-bases simulator for efficient development workflow.

- **SwiftUI:** this interface builder is a new interactive and revolutionary swift framework and has additional design tools and APIs to build a user interface. Xcode consistently runs the app interface live, allowing developers to watch how their application behaves directly in the design canvas. The library of controls and modifiers makes it easier for the developers to design and implement complex interfaces.

- **Story Board:** The interface builder storyboard is another feature built-in Xcode which makes it simpler for designers to create and design new views and link them together to build a complete user interface. The developer can easily drag and-drop the view and use navigation controllers onto a canvas and implement a design view for each one. Xcode generates all the code necessary to build the defined behaviour in the completed application.

2.5.4 Firebase

Firebase is a web and mobile application development platform that provides developers with a plethora of tools and services to help them implement application[9]. This includes features such as analytics, databases, Realtime database, authentication, file storage, and configuration. All of these services' backend components are fully maintained and functioned by Google. Client SDKs provided by Firebase interact with these backend services directly, without establishing any middleware between the app and services. This enables the developer to focus on improving the user experience and saves time.

- In traditional application development, the developer is required to write both front end and backend software. The frontend code calls on API endpoints exposed by the backend, and the backend performs all the tasks. Whereas, firebase bypasses the traditional backend.

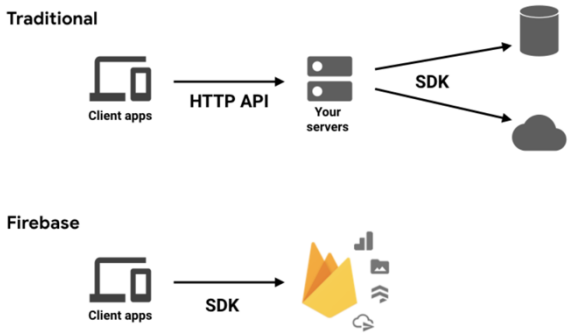


Figure 2.17. Traditional app development vs firebase.[9]

- **Firebase Realtime Database**
  Firebase offers Realtime cloud-hosted NoSQL database10, that allows users to store and sync between users in Realtime. The data in the Realtime database is stored as a JSON and synchronized in real-time to every connected client. **Figure 2.18** shows an example of a Realtime database.

Figure 2.18. An example of Realtime database

- **Firebase Authentication**
  Firebase Authentication[11] provides backend services easy to use SDKs and ready-made UI libraries to authenticate users. The user can be authenticated using the email address and password or federated identity providers such as Facebook, Twitter, and Google. Firebase authentication makes it easier to implement a secure system. When the user is signing into the app, they use their credentials. When the user enters their credentials the firebase backend will verify user credentials and return a response to the user.

2.5.6 Coco pods

CocoaPods is a dependency management tool for iOS projects[12]. It makes it easy to add, remove, update, and manage the third-party dependencies in the app. One main benefit of using coco pods it that it makes installing and removing dependencies very easy by simply running a command in the terminal. The manager can read the file; check for the dependencies in the listed libraries and if there are any dependencies it installs the version that satisfies all of them.

2.6 Project Aims and Objectives

For this project, I aimed to implement an application that offers all the functionality the crew companies will need to be more organized and to run more efficiently. To make sure I did not miss out on any important features, I tried to arrange an interview with other crew company managers and crew members, so they can review the features I am trying to implement however, this was not possible due to Covid-19. However, I was able to arrange an interview with the owner of the Shogun crew company and I requested him to give me constructive feedback on the features that I plan on implementing into CrewManagerApp. Notes from the interview can be seen in Appendix C.

He suggested adding new features to the app such as a job cancellation request; if a crew member accepts a job for a specific date and later wants to cancel it, he should be able to send a cancellation request for that specific job. The manager should then be notified, and he can decide whether or not to cancel the job for the crew member.

He also suggested expanding the application so that it incorporates clients. Clients should have the ability to directly request crews' members. The request notifies the manager, allowing them to check which crew members are available and directly send the job request to them through the application. The manager will then be notified about crew members who accepted to the job and the client will automatically receive the list of all the scheduled crew members. After each completed event, the client should have the ability to give an individual rating to each crewmember based on their job performance. Ultimately, this feature will improve communication and efficiency between events crew managers and their clients by accelerating the booking process very easy for the client and the manager. The manager will also be able to monitor each performance based on the stars they receive from the client.

Another feature he suggested was to create an algorithm. Which shows the highest-rated, best-performing crew members featured at the top of the list when sending a job request. Additionally, the list should only show crew members who have set their availability status to 'available'. This algorithm specification will help prevent any disproportionate selection of crew members who receive job requests. Subsequently, crew members will receive more hours based on their performance; a method that should motivate crew members to perform

better to receive more hours. To avoid crew double-booking, the owner of Shogun suggested implementing a feature that only displays a crew member who has no scheduled jobs within the same timeframe. Finally, crew members should also have the ability to control their time-off work by selecting days or times they are unavailable to work.

To summarise the results from the questionnaire and interview, the features I will try to implement within the application are presented below

Crew managers will have the ability to…

- Send job requests to crew members.
- Add and remove crew members.
- Create a Timesheets.
- View all of the crew members who are on a specific job.
- Send direct messages to crew members and clients.
- See the list of all available crew members.

Crew members will have the ability to…

- Display their availability.
- Accept, reject, cancel job requests.
- Send direct messages to crew managers.
- View a list of all their accepted jobs.
- Have accepted automatically added to the calendar.
- Make notes on the calendar related to specific jobs.
- View and alter personal information.

Chapter 3

3. SYSTEM DESIGN

In this chapter, I will explain system requirements, and documentation, and user guide. Good design is crucial when implementing any application and defining its functionality. If an application is not designed correctly, it will fail to meet its purpose thus would not attract the target audience.

3.1 System Requirements

There are many aspects to consider when implementing an App, therefore, it was essential to conduct a detailed level of requirements analysis. The system requirements were broken down into functional and non-functional requirements.

All the requirements were collected by understanding the existing system and the problems faced by the system. The results gathered from the questionnaires and interviews were used to identify the features the application must-have.

3.1.1 Functional Requirements

Req ID	Description
FR01	The application should allow the users to sign up and their details must be stored securely in the firebase database.
FR02	The application should allow the managers to send job requests/messages and should be able to view the job requests/messages sent.
FR03	The application should allow users to sign-in securely and be able to sign-out.
FR04	The application should allow the users to view, alter/ update their profile data
FR05	The application should have a notification panel, the user gets notified whenever a job request/message is received.
FR06	The application should allow the crew members to set their availability and unavailability.
FR07	The application should allow the manager to view list of all available crew members.
FR08	The application should allow the crew members to respond to the job request/message.
FR09	The manager should be able to create/edit and send timesheets.
FR10	The application should allow the crew member to cancel Jobs.
FR11	The application should allow the mangers to view/add/remove/edit crew member data.
FR12	The application should allow the crew to view their time sheets.
FR13	The application should allow the crew members to view all the jobs, they have accepted.
FR14	The job accepted by crew members should be automatically added to the calendar.

Table 3.1. Functional Requirements

3.1.2 Non-Functional Requirements

Req ID	Description
NFR01	The application should work on all iOS handheld devices.
NFR02	The application should be easy to extend. The code should be written in a way that favours the implementation of new functions in the future.
NFR03	The system should be available for 24 hours. Users can access it at any time.
NFR04	The device should be connected to the internet for the system to communicate with the database.
NFR05	The system should have a fast response time.

Table 3.2: Non-functional Requirement.

3.2 Documentation and User guide

In this section, I will include a detailed explanation of the documentation and User Guide for the application alongside an illustration of what happens in each screen.

3.2.1 Welcome Screen

When the app is lunched welcome screen is displayed as shown in **Figure 3.3**. From the welcome screen, the user can choose to either sign in if they already have an account or sign up to create a new account. From the sign-in screen, the user can log in via email, Facebook, and Google.

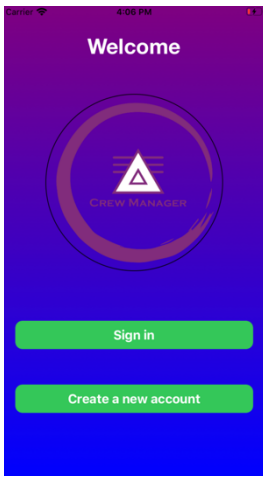


Figure 3.3. Welcome Screen.

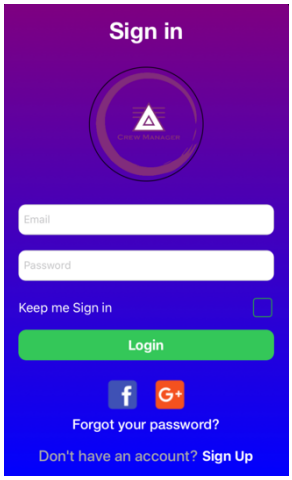
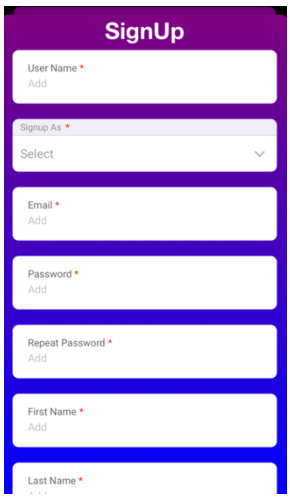


Figure 3.4. Sign-in Screen.



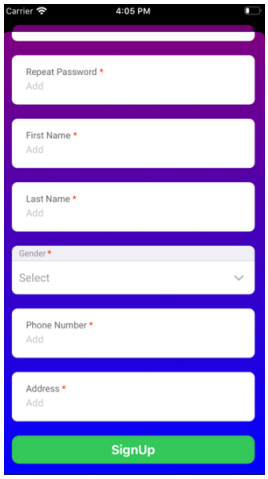


Figure 3.5. Sign-up Screen.

When the user selects to create a new account sign up screen appear as shown in **Figure 3.5.** On the signup screen, the user has to fill in all the required fields and select who they want to sign up as – manager or crew. Depending on their user type they will be directed to the relevant screen. **Figure 3.6** shows the crew dashboard and **Figure 3.7** shows the manager dashboard.

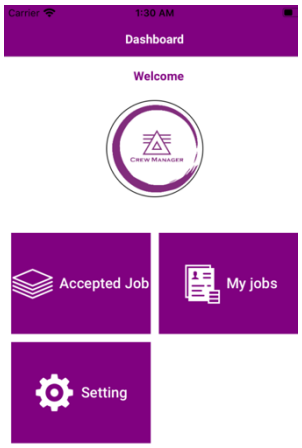


Figure 3.6. Crew Dashboard.

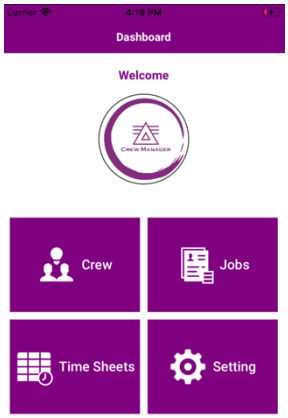


Figure 3.7. Manager dashboard.

3.2.2 User Logging in as a Manager

When the user successfully logins to the app as a manager, the manager dashboard will appear as shown in **Figure 3.7.** The manager dashboard contains 4 options the user can choose from:

- Crew
- Inbox
- Timesheet
- Setting

Crew Button

When the crew is selected it shows the list of all the crew members as seen in **Figure 3.8**. When the manager clicks on any crew member, it will display their contact details as seen in **Figure 3.9**. The manager can then contact the individual via email, phone, or direct message through the application.

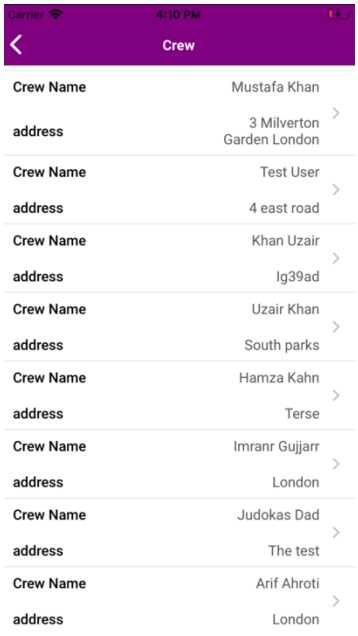


Figure 3.8. List of all the crew members.

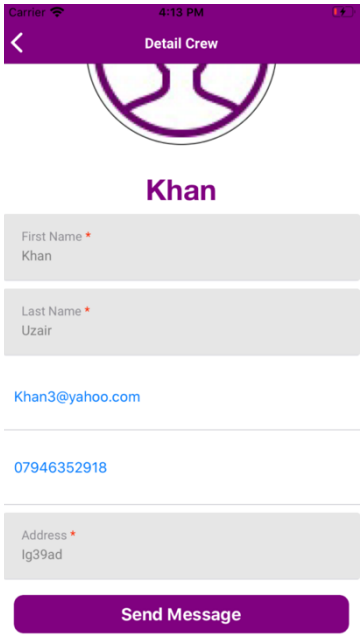


Figure 3.9. Crew members details

Inbox Button

When the 'messages inbox' tab is selected, it displays two options: 'inbox' and 'sent' as shown in **Figure 3.10**. The inbox tab displays the lists of all the messages sent by crews and the sent tab displays the messages sent by the manager. From the sent tab, the manager can send job requests to crew members by clicking on the plus button. A job request form will appear as shown in **Figure 3.11.** From the job request screen, the manager clicks on 'select' and it will show the list of all the crew members as shown in **Figure 3.12**. Once the form has been filled and the send button is selected, the job details are immediately sent to the chosen crew member.

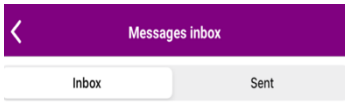


Figure 3.10. When the inbox button is selected.

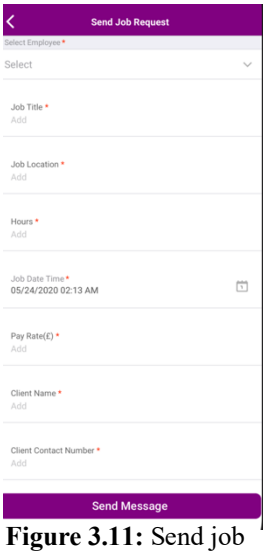


Figure 3.11: Send job request form.

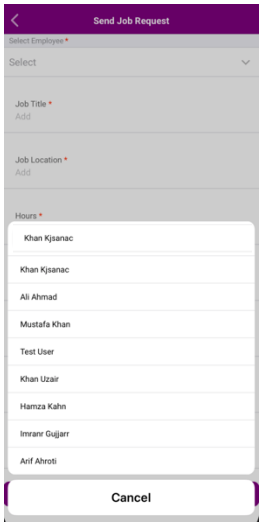


Figure 3.12: Shows the list of Crews whom the job can be sent to.

Timesheet Button

When the timesheet is selected, it displays the list of all the crew members who have accepted the jobs as shown in **Figure 3.13.** Clicking on the crew member's name will display all the previous jobs they have accepted as shown in **Figure 3.14.**

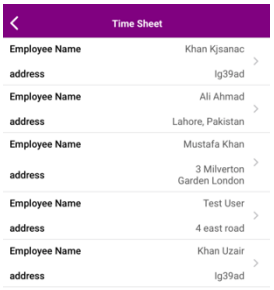


Figure 3.13: List of the crew members who has accepted the jobs

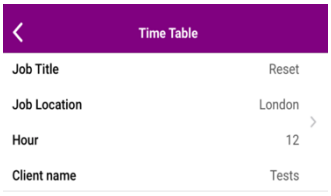


Figure 3.14. The jobs Crew members have accepted.

Setting Button
When the 'setting' option has been selected the screen in **Figure 3.15** is shown; the user can logout and view their profile. When the user clicks on logout a pop-up menu appears as shown in **Figure 3.16** to confirm if they want to logout. If the user clicks on the profile, it will show their profile information as shown in figure 35. The user can also edit and update their profile information.

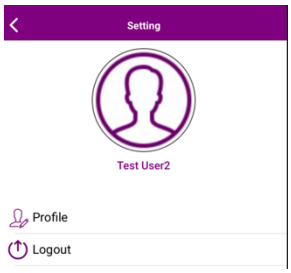


Figure 3.15: Setting Screen

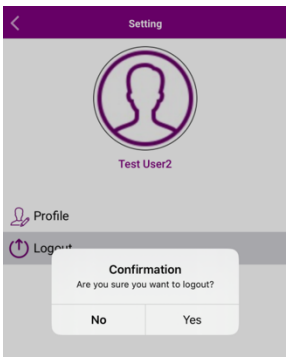


Figure 3.16. Shows pop-up for confirmation

3.2.3 User Logging in as Crew login

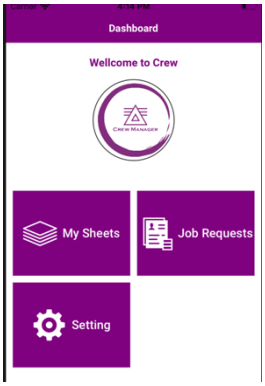


Figure 3.17. Manager dashboard

When the user successfully logs into the app as a crew member, the 'crew' dashboard will appear as shown in **Figure 3.17.** The crew dashboard has 3 options the user can choose from:

- My sheets
- Job Requests
- Settings

My Sheets Button

When my sheet is selected, the user is navigated to 'MySheets' screen which will show the list of all the jobs the user has accepted along with job details as shown in **Figure 3.18.**

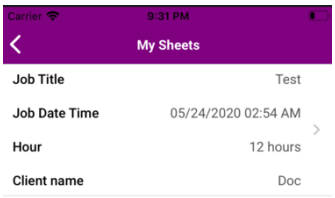


Figure 3.18. Shows jobs the crew member has previously accepted.

Job Request Button

When the job request is selected, the user is directed to a job request screen as shown in **Figure 3.19.**

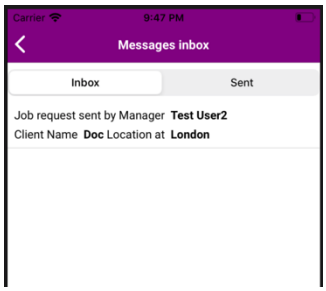


Figure 3.19. Shows job request screen

Inbox Button

The jobs request consists of two tabs – inbox and Sent. When Inbox is selected it shows the list of the jobs and messages sent by the manager. When the user clicks on the job sent by the manager, it shows the job details, and three buttons are shown which the user can select to Accept, reject and send a message as shown in **Figure 3.20.** When the user selects accept, the job is added to their shifts and if the user rejects the job, it is not added to their shifts.

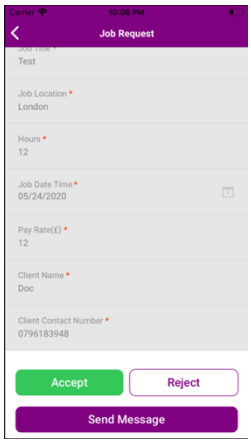


Figure 3.20. Shows the job accept or reject screen.

Setting Button

On Setting, the user can logout as shown **in Figure 3.21** and can view and update their profile details such as their email, phone, and password **Figure 3.22**. If the user logs out, a confirmation message will pop-up as shown **in Figure 3.23**.

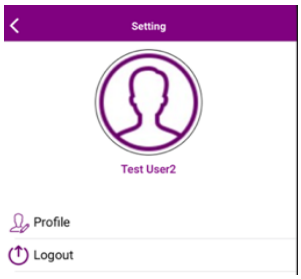


Figure 3.21: Shows setting screen.

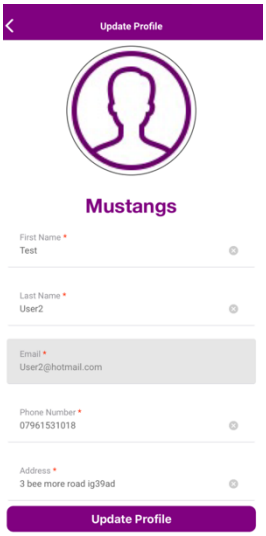


Figure 3.22: User profile Screen.

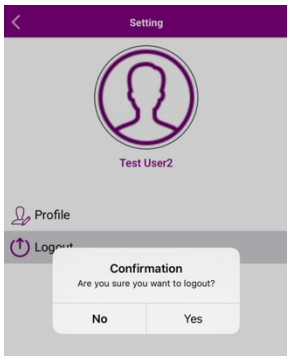


Figure 3.23. Shows pop-up for confirmation

CHAPTER 4

4. IMPLEMENTATION

This chapter describes the implementation of the core functionality of the application. CrewManagerApp uses three different technologies including Swift, firebase database, and cocoa touch framework. The chapter will be divided into 3 sections; Frontend, Backend**,** and Mid layer(also known as the business layer).

4.1 Front End - Swift

The interface builder editor within Xcode makes it very easy to design a full user interface by simply dragging and dropping objects from the object's library. The user interface for CrewManagerApp has been designed in the storyboard. The users will be using the application on different screen sizes. To make sure the app is responsive; auto layout constraints have been used to make the app responsive to any iPhone size that currently exists.

4.1.1 SignUpviewController.swift

Signup:
When the user wishes to use the app, they are required to provide their personal information to register. Once the user has entered their data, the isViewDatavalid() function on line 314 in Code 4.1, is used to ensure the user fills in all the required fields and all the fields with data are valid to the requirements. In the code below, you can see a few examples of the form validation.

```
314     func isViewDataValid() -> Bool {
315
316         if self.textUserName.text.count <= 0  {
317             SwiftMessages.showToast("UserName is missing. Please provide and try again.")
318             return false
319         }
320         if self.txtEmail.text.count <= 0 {
321             SwiftMessages.showToast("Email is missing. Please provide and try again.")
322             return false
323         }
324         if !self.txtEmail.text.isValidEmail() {
325             SwiftMessages.showToast("Please enter valid email address and try again")
326             return false
```

Code 4.1. Function used to validate the text fields.

On line 316 in **Code 4.1**, the validation requirements have been set for the length of the username. In this case, I have set the validation to ensure that the username is at least 1 character long. If no characters are inserted, an error message will appear inline 317 in **Code 4.1**.

From line 324 to 326 in **Code 4.1** outlines the validation for the email address field. First, I am validating if the field is empty or not. Next, on line 324 I am validating the email to ensure that the entered field is an actual email. If an invalid email address is inserted, then the corresponding error on line 325 is shown.

Once the validation is complete, the **createUser()** function on line 214 in **Code 4.2** is used to create the user with an email and password, the creation of the user happens in line 219 where it takes in the input email and password as a parameter.

```
214 //  This function is used to create
215     private func createUser() {
216         //Shows loading activty
217         self.showLoadingActivity()
218         //methods is used to create user with email address and passowrd
219         ServicesManager.createUser(email: self.txtEmail.text, password: self.txtPassword.text, success: { (auth: AuthDataResult) in
220 
221             self.hideLoadingActivity()
222             if let _ = ServicesManager.currentUser { // check user created .
223 
224                     // user profles picture is uploaded to firebase server created user
225                 self.uploadProfileImage(image: self.imageProfile.image!)
226             } else {
227                     // the user profile gets saved
228                 self.saveProfile()
229             }
230 
231         }) { (error:Error) in // Error handler
232             // show errror
233             SwiftMessages.showToast(error.localizedDescription, type: .error)
234             self.hideLoadingActivity() //hide loading activity
```

Code 4.2. Function used to create user.

While the user is being created in the firebase loadingActivity is shown. Once the user is created, if the user has uploaded a profile image, then the **saveProfile()** function on line 228 in **Code 4.2.** is used to make a custom object of the user for easy storing in the database. If all is successful, a success message is shown, and it hides the loading activity and takes the user to the dashboard.

The save profile function in **Code 4.3** below is used to make a custom object of the user type and save all the user data to firebase.

```
246     // Function makes a custom object of user and save all data to firebase.
247     private func saveProfile () {
248 
249         if !self.loginBySocialMedia { // Checks if user login from facebook and google
250             self.showLoadingActivity() // Shows loading activity
251 
252             // Makes user object in which the user type is saved
253             let userData:[String:String] = ["user_type": self.pickerServices.selectedId]
254 
255             // Casting user type from string to object.
256             if let type = UserTypes(rawValue: self.pickerServices.selectedId) {
257 
258                 CurrentUser.sharedInstance.userType = type // save user type
259                 ServicesManager.userType = type // Save user type to ServiceManager
260 
261             }else {
262                 // on failure it is call
263                 CurrentUser.sharedInstance.userType =  .employee
264                 ServicesManager.userType = .employee
265             }
266 
267             // Saves all user data related to user profile
268             ServicesManager.updateUserData(userObject: userData, success: {
                    (reference:DatabaseReference) in
269 
270                 // Makes firebase call in user save user profile data with success and failure
                        calls
271                 ServicesManager.saveProfile(userObject: self.userObject!.toDictionary(), success:
                        { (ref: DatabaseReference) in
272                     self.pushDashboard() // push view controoler
273                     self.hideLoadingActivity() // hide loading activity
```

Code 4.3. Function used to create a custom object of the user and saves the data to database.

First, it checks whether the user is signing into the app using social media. Line 253 in **Code 4.3** makes a user object in which the user type is saved. If it is a success, the user type is saved onto the database using the ServiceManager; failed inputs will not store onto the

database and will display an error. The updateUserData() function is used in the service manager, on line 320, to save all the user-related data in the database.

4.1.2 LoginViewController.swift

Login:
For the user to log in, they are required to enter their email and password. To integrate the sign-in methods in the app, the FirebaseAuth[13] library has been used for user authentication. The users can also be authenticated with Firebase using their Facebook and Google accounts. To allow the user to login using their Facebook and Google account, I watched a YouTube tutorial on channel "lets built that app"[14]. When the data has been inserted by the user, line 144 in **Code 4.4** checks if the data is valid, if the data is not valid an error is shown. Valid data makes an async call and shows the loading activity to block the user interface until the user authentication is complete.

An async call is a type of function that waits for something to be returned until it continues. For the loading activity a third-party source code library "SVProgressHUD"**15** is used to stop the user from having any input whilst the application gets a response from the firebase server. Line 153 in **Code 4.4** uses the sign-in method from FirebaseAuth library to try to login with the provided user credentials and returns a response, whether it is successful or not. Once the response is received as successful, the show activity is hidden, and the dashboard screen is shown. Otherwise, if the response gives an error, a relevant error message is shown on line 160 to the user and it goes back to the login form.

```
141     func login(email: String, password: String) {
142
143         // Validates the fields
144         if self.isViewDataValid() {
145
146             // Makes async call to block user interface until user can succcessfully login
147             DispatchQueue.main.async {
148
149                 //Shows loading Activity
150                 self.showLoadingActivity()
151
152                 // login from firease by email address and password
153                 Auth.auth().signIn(withEmail: self.txtEmail.text!, password: self.txtPassword.text!) { user, error in
154
155                     self.hideLoadingActivity()
156
157                     // Error handler
158                     if let error = error
159                     {
160                         SwiftMessages.showToast(error.localizedDescription, type: .error)
161                     }
162
163                     // check if user information get success from server
164                     guard user != nil else { return }
165                                         // push the dashboard view controller
166                     self.pushDashboard()
167                 }
```

Code 4.4. User logging in with email and password.

4.1.3 addJobMessageViewController.swift

Sending Job Request:
To send a job request, the manager has to fill in the fields in the send job form. This form consists of job details such as date, pay rate, location, and most importantly which employee they want to send the job to. To display all of the employees, first I get all of the employees from the database, which is achieved through the function getAllEmployeeList() on line 307 in AppNetworkManager.swift. This function gets all the users' data stored in the employee collection. After getting the data, all the employees get displayed using the JMPicker16. This is a third-party extension that I used to make the creation of the IOS data picker easier.

Once the manager has selected which employee to send the job to, they can send the job by selecting the send button. When the send button is pressed **Code 4.5** is run on line 168 in addJobMessageViewController.swift.

```
@IBAction func sendJobRequestBtnClicked(_ sender: UIButton) {

    // Validates the fields
    if self.isViewDataValid() {

        //shows loading activty
        self.showLoadingActivity()

        //sent message to employee with success and fallure call back Methods
        ServicesManager.sentMessageToEmployee(senderId:ServicesManager.uid!
            ,reciverID:self.pickerEmployee.selectedId,userObject: self.jobReques!.toDictionary() , success: {
            (reference:DatabaseReference) in
            // Shows sucess message
            SwiftMessages.showToast("Job request sent successfully.", type: .success)
            //hide loading activity
            self.hideLoadingActivity()
            self.pop()// pop the view
        }) { (error:Error) in // error handler
            // Shows Error Message
            SwiftMessages.showToast(error.localizedDescription, type: .error)
            // Hides loading activity
            self.hideLoadingActivity()
```

Code 4.5. Send job request button is clicked.

Firstly, the form is validated using the isViewDataValid() function similar to the login system. It checks if all required fields have been inputted. Once the form is successfully validated the loading activity is shown. Next, storing to the database happens on line 177 in **Code 4.5**. It uses the sendMessageToEmployee() function in the AppNetworkManager.swift class on line 269 to store the job in the database.

The sendMessageToEmployee function stores two copies of the job in the database. A receiver copy in the employee's inbox collection, and a sender copy in the managers sent collection. To achieve this, it takes in the uid for the sender, the uid for the receiver, and the message itself. It then uses the uid to store the message in the correct users' and managers' collections. Two copies are stored to ensure that the manager can see the job in their sent screen and employee can see the job request in their inbox screen. When the message gets stored successfully the form screen closes and a success message is shown on the screen.

4.1.4 ViewJobRequestViewController.swift

Viewing Job Request:
Once an employee has received a job request, they can either accept or reject the job. When accepting the job, it simply runs the acceptRequestBtnClicked() function on line 137 in **code 4.6**. This function sets the job status to accepted in the corresponding job record in the database using its unique id.

When the employee rejects the job request, the jobs status gets set to rejected in the database, similar to how the accept button works.

Accepting job Request:
When accepting a job request, the **acceptRequestBtnClicked**() function on line 137 in ViewJobRequestViewController.swift gets run.

First, it gets the current date inline 148 in **Code 4.6**, so it can be stored in the accepted date field in the database, informing the manager when the job was accepted.

```
@IBAction func acceptRequestBtnClicked(_ sender: UIButton) {

    if let job = self.jobDetail { // Gets the job request details.

        // user infoview is instance of JobRequestAcceptPopup which is initlized with current job detail
        let userInfoView = JobRequestAcceptPopup(showInView: self.view, jobDetail: job)

        // call back on success which. Returns dictionay
        userInfoView.handleSuccessBlock = {(dic) in


            let statusDate = DateHelper.stringFromDate(Date())! // set date for acccepted. Datehelper is used to convert date object in string
            // gets request id and manager id from job detail object
            if let requestID = self.jobDetail.jobUID ,let managerUID = self.jobDetail.managerUID {

                // shows loading activity
                self.showLoadingActivity()

                // Makes user object which have all information needed to save with job request. employee uid,  manager uid, employee name and job accept status.
                let userObject : [String : Any] = ["manager_uid": self.jobDetail.managerUID! ,"employee_uid":self.jobDetail.employeeUID! , K.AcceptedJobRequestDataKey:statusDate , "employee_name":self.jobDetail.employeeName!]

                // Merge job request object with user object to save job details related to employee and manager.
                var newDic = dic.merging(userObject) { (current, _) in current }.merging(self.jobDetail.toDictionary()) { (current, _) in current }

                // Job satus is saved as accepted.
                newDic["status"] = "accepted"

                // Updates the job request status, with two call back methods is return as a success and failure.
                ServicesManager.acceptJobRequestStatus(managerUid:managerUID, employeeUid: self.jobDetail.employeeUID!, userObject:newDic, jobRequestUID: requestID, statusDate: statusDate, success: { (referece:DatabaseReference) in

                    self.pop() // pop the view controller on success
```

Code 4.6. Accept Job button is clicked.

Next, the user data is put in a map to be able to store in the database, this can be seen inline 156 in **Code 4.6.** The change in status happens inline 162 in **Code 4.6**, where the status gets set to "accepted". Once this is done the data can be updated in the database which occurs on line 165, using the acceptJobRequestStatus() function on line 182 in the AppNetworkManager.swift class. This ensures that both copies of the job (in the manager collection and employee collection) get set to accepted. If the job has been accepted successfully, the current screen gets closed and a success message is shown to the user.

Rejected Job request:

```
115         if let requestID = self.jobDetail.jobUID { // get job id for rejction
116
117             self.showLoadingActivity() // show loading activity
118             let statusDate = DateHelper.stringFromDate(Date())! // set date for rejection
119             //rejectJobRequest is method with success and failure call back
120             ServicesManager.rejectJobRequest(requestId: requestID , senderId: self.jobDetail.managerUID, reciverID:
                    self.jobDetail.employeeUID, statusDate: statusDate, success: { (referece:DatabaseReference) in
121
122                 self.pop() //pop view controller
123                 //sucess message on job request is reject
124                 SwiftMessages.showToast("Request is rejected.", type: .success)
125                 self.hideLoadingActivity() // hide loading activity
126             }, failure: { (error:Error) in // failure call back
127                 // show error messag
128                 SwiftMessages.showToast(error.localizedDescription, type: .error)
129                 self.hideLoadingActivity() // hide loading activity
130             })
```

Code 4.7. Job reject button is clicked

The process of rejecting a job is similar to the 'accepted job' procedure. However, instead of the job status getting set to accepted, it gets set to 'rejected' for both copies in the database and a corresponding success or error message is shown.

4.1.5 DetailEmployeeViewController.swift

View Employees:

The manager dashboard has an option to view all the employees that have signed up to the app. This can be used to view employee details and contact any employee via message, email, or phone call.

```
59      fileprivate func getEmployeeList() {
60
61          self.showLoadingActivity() // shows loading activities
62          self.tblEmployee.reloadData() // load employee data
63
64          //get all employee list from firebase server with DataSnapshot
65          ServicesManager.getAllEmployeeList(success: { (snapshot:DataSnapshot) in
66
67              self.employeeProfiles.removeAll() //removes previous employee list
68              self.tblEmployee.reloadData() // reload table view
69
70              for child in snapshot.children.allObjects { // for loop to iterates over list of children
71
72                  if let snap = child as? DataSnapshot { // type casting of object
73
74                      if let values = snap.value  as? [String:Any] { // check type casting
75
76                          if let profileDic = values["profile"] as? [String:Any] { // get profile object
77
78                              let profile = UserProfile(fromDictionary: profileDic) // initlize the user profile
79                              profile.uid = snap.key // set user id in profile object
80                              self.employeeProfiles.append(profile) // append the user profile object on employeeProfiles
```

Code 4.8. Function used to get list of all the employees.

How this works can be seen in **code 4.8**. This function gets all the employees from the database on line 65, which is the same function I used to get the employee list in sending a job request implementation. Then inline 70, it iterates over the list, so I can create an object with just the profile data which I need to show to the manager in the table view.

Then the profile data of each employee gets appended to the employeeProfiles array of objects on line 80 in **code 4.8**, I then use this list to be able to create a table of all the employees on the screen.

```
113     func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
            UITableViewCell { // compose tableview cell
114
115         let profile = self.employeeProfiles[indexPath.row] //get profile object
116         let cell = tableView.dequeueReusableCell(withIdentifier:
                R.CellIdentifiers.EmployeeCell,
117                                                  for: indexPath) as! EmployeeTableViewCell
118         cell.lblTitle1.text = "Crew Name" // set employee name label
119         cell.lblDetail1.text = profile.fname + " " + profile.lname // set user name
120
121         cell.stackView2.isHidden = true // hide extra views
122
123         cell.lblTitle3.text = "address" // set address label
124         cell.lblDetail3.text = profile.address // set address
125
126         cell.removeSeparatorInsets() // remove insects
127         return cell
128     }
```

Code 4.9. Table view to show the list of all the Employees

The above code shows how the table view is created. First, the profile data is taken from the employeeProfiles list on line 115 in **code 4.9**. Then the cell is created from lines 116 to 127 **code 4.9**. What this does is has two labels in the cell, the first is for the crew name which uses the employees first name and surname to show it on the cell on line 119. The second label in the cell is for the address of the employee on line 124.

Messages

In this feature the manager can send a message to an employee by selecting the message button. When this option is selected, a form gets displayed on the screen, which has a subject and message body field. This message is similarly sent like the job request, however, the difference being that the message type is set to response message.

The following code gets run when sending the message;

```
    @IBAction func sendMessageBtnClicked(_ sender: UIButton) {
81      // check all fields are valid
82      if self.isViewDataValid() {
83
84          // make required object to save data in fire base data base tree
85          let userObject = ["message":self.txtMessage.text , "subject":self.txtSubject.text , "type":K.messageTypeResponse ,
                "employee_uid":self.jobDetail.employeeUID!, "employee_name":self.jobDetail.employeeName!
                ,"manager_uid":self.jobDetail.managerUID!].merging(jobDetail.toDictionary()) { (current, _) in current }
86
87          if self.isFromManagerViewEmployee { // check is it employee or manager
88
89              self.showLoadingActivity() // show loading activity
90
91              //sentMessageToEmployee is a method used with sender information and job detail
92              ServicesManager.sentMessageToEmployee(senderId: self.jobDetail.managerUID, reciverID: self.jobDetail.employeeUID, userObject: userObject , success:
                    { (reference :DatabaseReference) in
93                  // on success pop the view
94                  self.pop()
95                  // hide loading activity
96                  self.hideLoadingActivity()
97
98              }, failure: { (error:Error) in // error call back
99                  // error message show
100                 SwiftMessages.showToast(error.localizedDescription, type: .error)
101                 self.hideLoadingActivity() // hide loading activity
```

Code 4.9. Send Message Button is clicked

Code 4.9. shows that when the message is sent to the employee it first creates the object that needs to be stored in the database on line 84. The object has fields of all the necessary data such as message, subject, manager_uid, employee_uid, etc. Then the object gets stored in the database on line 92 using the 'sentMessageToEmployee' function in the

'AppNetworkManager.swift' class. This function takes in the managers and employees uid so it can store the data in a similar function to how sending a job request works, it creates two copies to get stored in the managers sent collection and in the employee's inbox collection.

Once this is completed the screen closes on line 94, the loadingActivity gets hidden.

Phone call:
If a manager decides to contact an employee via phone, they can do this by directly calling the phone number in the employee's details. During signup the employee provides the phone number which gets stored in the user profile in the database, I can use this number to be able to call the user.

```
// make call url for call
if let url = URL(string: "tel://\(String(describing: self.profileData!.phoneNumber))"),
UIApplication.shared.canOpenURL(url) { // can device open the url
   if #available(iOS 10, *) {//check IOS version
     UIApplication.shared.open(url, options: [:], completionHandler:nil)//open the URL
    } else {
        UIApplication.shared.openURL(url) // open the otherdevice
    }
} else {
        // add error message here
}
```

Code 4.10. Code used direct user to phone dial app. This code is taken from[17]

When the phone number in the employee section is selected, it takes the user to their phones dial app and has the number already input in there. To achieve this, I looked at a tutori al[17], which shows how to do this. The code above was used from[17].

Email :
A manager can also send an email to the employee if they need to. The code for that was also used from an online tutorial[18]. The code below was used when the employees' email is pressed.

```
if MFMailComposeViewController.canSendMail() {
   let mail = MFMailComposeViewController()
    mail.setToRecipients([self.profileData!.email]) // set email
   mail.setSubject("") // set email subject
   mail.setMessageBody("", isHTML: true) //set email message body
   mail.mailComposeDelegate = self
   present(mail, animated: true) // email sending animation
}
else { //show message of not sent email
   print("Email cannot be sent")
}
```

Code 4.11. Function used to get list of all the employees. This code is taken from[18]

Code 4.11 gets the employee's email from the database and sets the recipient of the email as the employee's email, and the email can be composed by the manager, without having to manually enter the email.

4.2 Mid Layer – Swift

4.2.1 AppNetworkManager.Swift

The mid-layer manages the communication between the user interface and the database. The business layer in my app is entirely implemented in the AppNetworkManager.Swift class. This class contains all the functions needed to retrieve, update, and store data to the firebase database. This class contains all the reference variables of all the necessary collections such as sent collection or profile collection in the database, these reference variables are used to easily reference a collection in the database.

To specifically target a unique record, the corresponding unique id is used to look for that specific record in a collection.

```
110     // this is funtion gets all the employees jobs accepted
111     public  func getCurrentEmployJobs(uid:String ,success: @escaping (DataSnapshot)->Void, failure: @escaping (Error) ->
            Void) {
112
113         //make referece to get all jobs
114         var reference: DatabaseReference? {
115
116             return self.databaseReference.child("\(K.employeeTree)/\(uid)/\(K.inboxTree)")
117         }
```

Code 4.12. Function used to get current employee Jobs.

For example, the function in code **4.12** is used to get the current jobs an employee has. To accomplish this, it has a parameter "uid" which is visible in source code line 111, this is the unique id of the employee I want to get the jobs for. This uid is then used in line 116. This line is looking at the employee tree, then in the employee tree looking for the record with the specified uid then finally looking into the inboxTree. Once this is complete, it returns all the message records in the inbox tree.

4.3 Backend – Google Firebase

4.3.1 Firebase Real Time Database

For CrewManagerApp I have utilised the firebase Realtime database. The structure of a firebase database is different from a SQL database because it does not use a primary key and secondary relationship. Instead, it stores data in a JSON formatted tree. In a firebase database to pull data, we look at the node and the nodes are identified through references using unique ids. When data is added to the JSON tree, it becomes the node in the existing JSON structure with an associated unique id.

The main functionality of my CrewManagerApp is sending messages and the messages should be sent instantly. In the firebase database, the data is synced across all the clients in real-time and instantly. Because of this property of Firebase, I can ensure messages get synced and sent to a user instantly. In the database, we have an employee tree and a manager tree;

The manager collection has the records of all the users who sign up as a manager. Each manager is unique and has their own properties. **Figure 4.1** is a screenshot of how a manager record looks like:

Figure 4.1. Shows the manager tree in the database.

As seen in **Figure 4.1** each manager user has three branches, an inbox branch, profile branch and sent branch. The inbox branch contains all the messages someone has sent the manager. The profile branch contains the profile information such as name, email, phone number etc. The sent branch contains all the sent messages and job requests the manager has sent. The above screenshot shows a sent job request and all the properties it contains.

When a user signs up as an employee their data is stored under the employee collection, each employee record has a profile, inbox and sent collection. **Figure 4.2** is a screenshot showing what a typical employee record looks like.

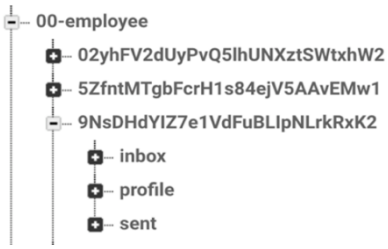


Figure 4.2. Shows employee tree in the database.

It is almost identical to the manager collection. It has an inbox collection for all the messages or jobs the employee has received. A sent collection to store all the sent messages and a profile collection to store the employee's personal details such as. Name, address, email etc.

4.4 Disruptions and Solutions

Initially, when I started the implementation, I followed firebase documentation to set up firebase into my app, I completed the process but realised that it would not permit me to write data to the firebase database. I read firebase documentation, but I was unable to find a solution. Following this, I explored several forums on stack overflow and found a user's post that was related to the same issue I was experiencing. The issue was solved by adding rules to the database, which gave me permission to read and write into a firebase database.

Another issue I experienced was related to Xcode once I completed the signup and login functionality. I attempted to test this function by registering as a new user, the data would successfully get stored into the firebase database. When the signup option was selected, it would save the user details to the firebase database, however, the app running on a simulator would freeze. Xcode debugger did not display any error message. I searched on stack overflow to find a solution; however, I was unable to find anything useful. One of my friends had a similar issue, he suggested to remove and reinstall Xcode. I decided to reinstall Xcode which rectified the issue.

The signup form consists of an array of UI text fields. The issue I experienced was with the iPhone's keyboard slides that covered up some of the text fields which prevented the text fields from automatically scrolling down. Once the user clicked on any of the fields, the iPhone keyboard would be revealed, and it would remain there even if it was not in use. Once again, I searched online for a solution and I found a YouTube tutorial[19], which suggested the use of a third-party library 'IQKeyboardManagerSwift'[20] to solve the problem. By using this library. the text field automatically scrolls. Subsequently, the keyboard only now shows when the user clicks on the text fields and is hidden automatically when not in use.

4.5 Third Party Libraries

These are the resources and third-party libraries that have been used in this project.

Firebase/Storage[21]: library is used to store user profile pictures on to the firebase,
FirebaseUI[22] **:** library is used to connect common UI elements to the firebase database for data storage and to get access to user authentication methods.

- **Firebase/Auth**[23] is used to authenticate users to the app.
- **Firebase/Database**[24] is used to store data in firebase.
- **GoogleSignIn**[25] is used to integrate google sign into the app.
- **Fbsdkloginkit**[26] is used to integrate Facebook sign into the app.

SwiftMessages[27] is used to show toast messages for errors and success.
IQKeyboardManagerSwift[28]: is used to prevent the keyboard from covering text fields.
SVProgressHUD[29]: is used to display the progress of an ongoing task.
Alerts-and-pickers[30]**:** is used to show alerts and for pickers.
Materials[31]**:** is used for font styling.

Chapter 5

5. TESTING

Testing was an on-going process throughout the development of the project. This is conducted to ensure that the system is working as intended and I evaluated the functionality of the application to check if the system meets the requirements. CrewManagerApp consisted of black-box testing, white-box testing, and user testing.

5.1 Blackbox Testing

Blackbox Testing was carried out to test the functionality of the application. The black box involves testing whereby the design and the structure of the codes are unknown to the tester. It is done without looking at the internal paths of the software. The central aim of black-box testing is on the functionality of the system as a whole.

The black box testing carried out by me; this was due to the current situation (COVID 19). In a normal situation, I would have made the test cases and get users with different levels of IT skills to test the application. I would have also given the users a list of specific tasks they need to perform such as "signup as a manager and send a job request" while they were performing the task, I would observe the user and evaluate how long it took to complete the task, and the issues or difficulty they faced while performing the task.

Table 5.1 shows the main functionality of the app being tested.

Description	Expected Result	Test Case	Actual Result	Status
Test the validation checks on the required fields when signing up a new user.	An error message should be shown informing the user to fill in the required fields.	On the register screen leave the required fields blank and click the signup button.	An error message was shown informing the user to "fill in the required fields"	Pass
Test if the User can register with the email address that already exists.	An error message should be shown "email address already exists use another one"	On the sign-up screen register using mkhan@yahoo.com and click signup.	An error message was shown informing the user, the email address already exists use another email.	Pass
Test if the user can enter a 5-character password.	An error message should be shown "Password must be 6 characters or more"	On the sign-up screen in the password field enter a 4-character long password and submit the form.	An error message was shown informing the password needs to be at least 6 characters.	Pass
Test the confirm password fields on the signup page.	If the password doesn't match an error message should be shown "Password does not match"	On the signup screen in the password field type in a mismatched password.	An error message was shown stating "Password does not match".	Pass
Test if the crew dashboard is shown when the user signup as a user type crew.	When the user signs up as a user type crew, the crew dashboard should be shown.	On the signup, screen complete all the fields and select user type Crew.	The crew dashboard was shown.	Pass
Test if an unregistered user can login using	When the user tries to sign in using unregistered email and password an error message should be	On the login screen in the email and password field type akhns@yahoo.com Password: 12345678	The user was not able to sign in an error message was shown "user with this Id doesn't exist"	Pass

email address and password.	shown "user with this Id doesn't exist".			
Test if the user can log in using their credentials.	It should allow the user to login and the dashboard should be shown according to their user type.	On the login screen, the user enters the correct username and password.	The user was able to log in successfully it showed the dashboard according to the user type.	Pass
Test if the list of the crews is listed, on the manager view controller under the crew button.	Should show the list of all the crews, who have signed up to the app.	On the manager dashboard, the user clicks on the crew button.	Displayed the list of all the crew signup to the app.	Pass
Test if the list of the crews that have accepted the job appears in the timesheets.	Should display the list of all the crews who have accepted a job.	On the manager dashboard, the user clicks on timesheets.	Displayed the list of all the employees who have accepted a job.	Pass
Test if the list of all the job crews has accepted appears when a specific crew name is selected.	Should display the list of all the jobs that a specific crew member has accepted.	On the manager dashboard, click on the time sheet and any crewmember name.	Displayed the list of all the jobs accepted by that crewmember.	Pass
Test if a crew manager can send a message to the crewmember.	The message sent by the manager should appear in the inbox of the crewmember whom it has been sent to.	On the manager dashboard, the user selects the crew, and selects any crewmember and sends a message.	The message appeared in the inbox of the crew member whom the message was sent to.	Pass
Test if the manager's inbox shows the list of all the messages sent by crew members.	All the messages sent to the manager should appear in the manager's inbox.	On the manager dashboard, the user clicks on the inbox.	It showed the list of all the messages sent to the manager.	Pass
Test if the list of all the crew members is shown in the dropdown menu when sending a job message.	A list of all the crewmembers should be shown in the drop-down menu.	On the manager dashboard, the user clicks on the sent button and selects the Crewmember.	It displayed the list of all the Crew members.	Pass
Test if the user profile information is updated.	When the user edits their profile information and updates it, their information should be updated.	On the manager dashboard, the user clicks on the setting, profile, and updates their information.	The information was updated successfully.	Pass
Test if the job request displays the list of all the jobs sent by the manager.	A list of all the job requests sent by the manager should be shown in the job request.	On the Crew dashboard, the user clicks on the job requests.	It displayed the list of all the job request send by the manager.	Pass

5.2 Whitebox Testing

White box testing is a method that evaluates the code, internal structure and design of an application. In Whitebox testing, the code is visible to the tester, therefore the tester knows the internal structure of the code and can access its viability.

Table 5.2 shows the main functions of the app being tested.

Function	Input	Process	Output	Actual Outcome	Status
signInWithEmailBtnClicked() in loginviewcontroller.swift	In the login view type valid email and password	Shows the loading activity while loading activity then input validations then checks if the user exists in the data base, if it exists it loading activity stops and take user to screen.	If login is successful take user to dashboard	Takes the user to dashboard	Pass
forgotPasswordBtnClicked() in loginviewcontroller.swift	User Presses Button after entering account email	Shows the loading activity while activity is happening. Validates the data and sends password reset email to the input email via firebase auth.	The user received rest password link	The user should receive rest password link	Pass
isViewDataValid() loginviewcontroller.swift	User leaves the login fields blank	Checks if all required fields are filled and shows an error message if any fields are left blank.	Error message is shown	Error message should be shown	Pass
sendJobRequestBtnClicked in AddJobMessageViewController.swift	Click on the button after valid input.	Validates input, then makes 2 database writes 1 for receiver and 1 for sender uses firebase plugin.	Should show sent message for both the receiver and sender.	Successfully send the message to the receiver and shows messages to both sender and receiver.	Pass
rejectMessageBtnClicked() in viewjobrequestcontroler.swift	User Clicks on reject button	Shows loading activity, uses the job id to mark the job as rejected in the data base and if successful pop the view controller and success message is shown.	Should rejects the job sent by the manager and toast message is shown on the screen “job is. Rejected.”	Successfully rejects the jobs and “job rejected message is shown”	Pass
acceptRequestBtnClicked() in viewjobrequestcontroller.swift	User Clicks on accept button.	Get the job id, marks job status as accepted in the manager and employee.	Should show accept the job, pop the view and toast message is shown “job is accepted successfully”	Successfully accepts the jobs and success message was shown.	Pass

updateProfileBtnClicked() in profileupdateviewcontroller.swift	User inputs data and click on update profile.	Shows the loading activity, validates the input and sends a request to firebase data base to update user details, and also updates it locally.	Should update the user information and show a success message if field are left empty an error will be shown.	Successfully updates information and toast message is shown " profile saved Successfully	Pass
btnMakeCallOnCellPhoneClicked() in detailsviewcontroller.swift	User Clicks on the phone number.	Use the phone number stored for that user in the database, check the iOS version of the device to make sure the correct method is being called.	Take the user to device phone application with the correct phone already entered.	Successfully takes the user to phone application with phone number entered in it.	Pass

5.3 User Testing

User testing is used to evaluate the usability of the application and decide whether the app is ready to be launched into the market. The feedback given by users will be used to make any necessary changes before the application is released for the public.

Initially, I planned on testing the app using well-established crewing companies' crew members and crew managers so I could receive constructive feedback from intended users. However, due to the current situation (COVID-19) and lockdown, this was not possible. However, I was able to get my cousin (owner of Shogun) and some of my friends who work in the crewing industry to undertake user testing. The test result is not too reliable this is due to a small data sample. The testing was carried out by giving the users a questionnaire with a list of questions. while using the app they just answer the question according to their experience. The full questionnaire can be viewed in appendix **D & E.**

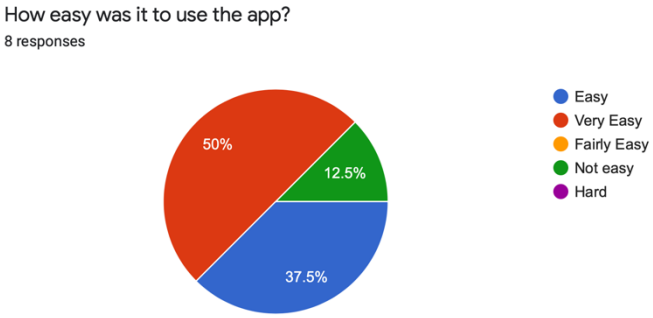


Figure 5.1. Shows how easy it is to use the app.

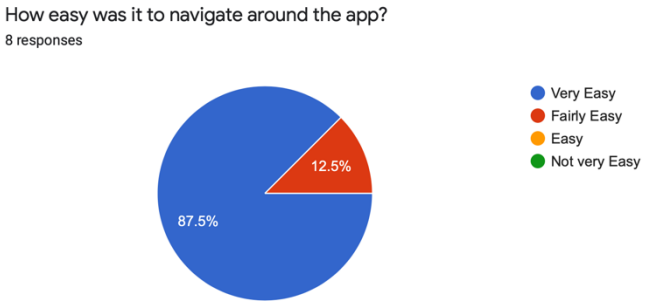


Figure 5.2. Shows user experience navigation around the app

To find out the usability of the application and how the user feels about the application. I asked the users how easy it is to use the Application as shown in **Figure 5.1.** Majority of the users found the app very easy to use, 87% of the users found it very easy to navigate around the app, Summarising the results received on these questions shows the Application is very straightforward to use and all the buttons are labelled clearly which makes navigating around the app easy.

Furthermore, two specific questions were asked about the user interface and the color scheme of the app, to find out what the user thinks about the color scheme and the design of the application because identifying an appropriate color scheme is a very important element of a high-quality mobile application. The result for the first question is shown in **Figure 5.3** which shows 87.5% of the users likes the user interface and 87.5% of the user likes the color scheme as shown in **Figure 5.4**, from looking at the responses the color scheme of the app won't need many improvements in the future.

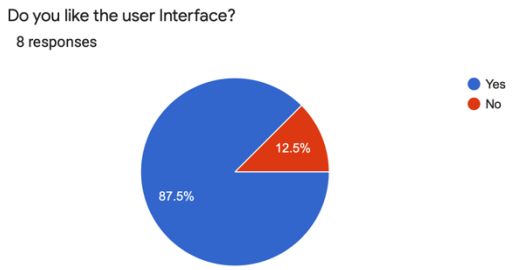


Figure 5.3. Shows the whether the user likes user interface.

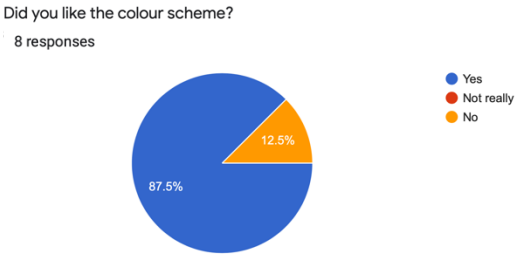


Figure 5.4. Shows whether the user likes the colour scheme.

Furthermore, questions related to the core functionality of the application were asked and looking at the responses in **Figure 5.5** (87.8%) of the users agreed the application makes their communication with the manager easier, which is a success because one of the aims of the project was to make the communication between the crew easy. **Figure 5.6** shows that 75% of the users found the features very useful, the 12.5% of users who said they didn't find it useful, I investigated it further to find out the reason, they suggested some changes which will be discussed in conclusion.

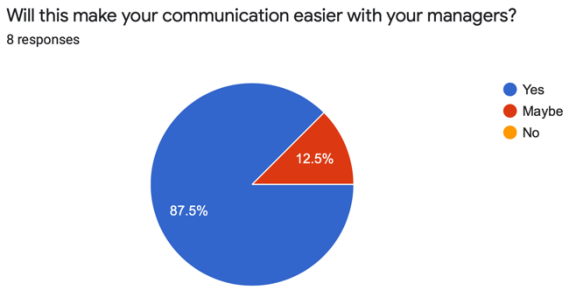


Figure 5.5: Shows user response on whether the app makes their communication easier with managers.

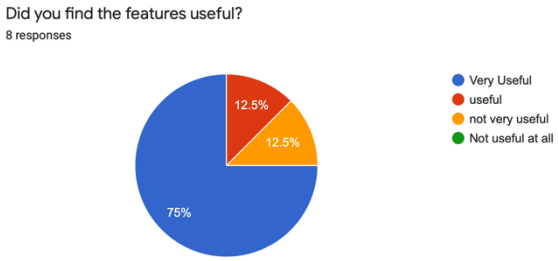


Figure 5.6. Shows if the user finds the features useful.

To target the crew manager experience, questions were asked related to the usability of the application, **Figure 5.7.** shows around 80% of the users found it fairly easy to view the list of the job request, however, the 20% of the users who did not find it easy were investigated from looking into it, I found out the reason why they did not find it easy was because currently when the crew accepts the job their name appears in the timesheet but it does not show when they have accepted the job. **Figure 5.8.** shows the majority of the users found it easy to send a job request.

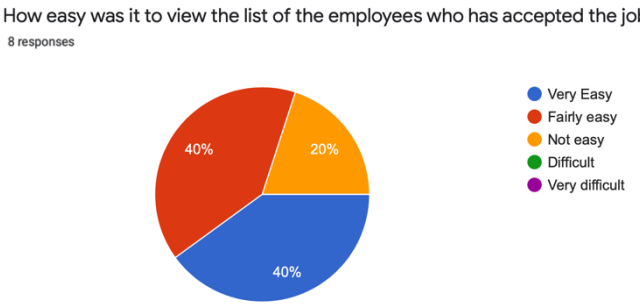


Figure 5.7. Shows user response on how easy it was to view the list of crews who has accepted the job.

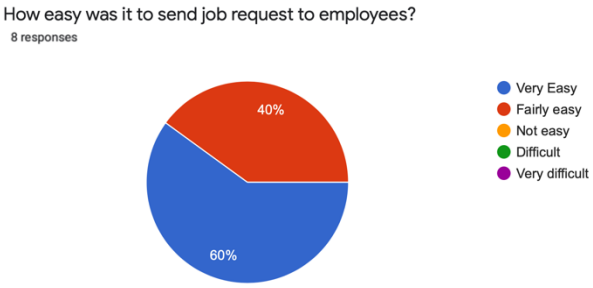


Figure 5.8. Shows the user response on the usability of the app.

Furthermore, it was important to know if the managers would want to add more functionality to the application. A specific question was asked to know if they would like to app to have more features and looking at the results, all of the managers who did the test suggested they want the app to have more features. I will be discussing this more in the system failure section.

5.4. Functional requirement Evaluation

Table 5.3 Shows the list functional requirements that has been met and not met.

Req ID	Description	Status
FR01	The application should allow the users to sign up and their details must be stored securely in the firebase database.	Met
FR02	The application should allow the managers to send job requests/messages and should be able to view the job requests/messages sent.	Met
FR03	The application should allow users to sign-in securely and be able to sign-out.	Met
FR04	The application should allow the users to view, alter/ update their profile data	Met
FR05	The application should have a notification panel, the user gets notified whenever a job request/message is received.	Not Met
FR06	The application should allow the crew members to set their availability and unavailability.	Not Met
FR07	The application should allow the manager to view the list of all available crew members.	Not Met
FR08	The application should allow the crew members to respond to the job request/message.	Met
FR09	The manager should be able to create/edit and send timesheets.	Not Met
FR10	The application should allow the crew members to cancel Jobs.	Not Met
FR11	The application should allow the mangers to view/add/remove/edit crew member data.	Not Met
FR12	The application should allow the crew to view their time sheets.	Not Met
FR13	The application should allow the crew members to view all the job, they have accepted.	Met
FR14	The job accepted by crew should be automatically added to the calendar.	Not Met

Chapter 6

6. SELF-EVALUATION

In this project, I was able to develop a fully functional, sophisticated but user-friendly smartphone application that would primarily serve the event crew industry. The CrewManager application targets two key users; event crew managers and event crew staff members.

The learning curve of the project was challenging at all stages, however, creating CrewManagerApp proved to be a very enlightening experience that has resulted in a great deal of learning and self-improvement. Identifying problems and finding solutions related to the application has not only enhanced my analytical and problem-solving skills but has also enlightened me about the practical application of Computer Science – using information, coding, and programming language theory to solve technically challenging real-world issues. Moreover, this experience has reinforced and strengthened my understanding of IOS programming skills.

There were a number of complications I encountered throughout the project that I could have managed better. For instance, I underestimated the complexity of some of the features I planned on adding such as notification, timesheet, and calendar due to time constraints. However, I recognized that too many features would make it more difficult to focus on the core functionality of the application. In the future, I would complete the features I originally intended to implement first and conduct the necessary tests to ensure they were free of errors or bugs. Once they were fully functional, I would have then considered adding other less crucial features.

My methodology could have been improved including more participants in the questionnaire which would have not only provided me with more valuable feedback related to improving CrewManagerApp but would have also verified what features are essential according to the users. This primary data would have been helpful in the beginning stages of the project, as I originally planned on creating a web application. However, upon further research, I later realized that users would prefer a mobile application instead. Subsequently, I had to restart the project -– wasting time that could have been used to implement essential application features.

Chapter 7

7. CONCLUSION

The development of this project was a challenging yet rewarding process. The theory of how the development should be carried out and the thing that could go wrong in the process provided invaluable experience in developing the project over many months.

The core functionality of the app includes sending job requests and the crew members accepting or rejecting the job requests that were successfully implemented in the Application. Other functionalities such as displaying the list of all the crew members, secure login system, forgot password, displaying the list of jobs accepted by the crews, logging in as manager, and as a crew member and updating profile information were implemented and tested to be working correctly. However, due to the pandemic (COVID 19), this had affected me a lot, the scope of the features had to be scaled down to finish the project on time. However, despite having to make these changes, I was still able to meet the MVP and these additional features can always be added later for the potential development of the application.

It was unfortunate that all the essential features needed were not implemented as planned such as showing crew availabilities, creating a timesheet for the employees, sending the jobs to multiple users, adding notifications, calendar, job cancelation, the main reason for this was time shortage.

7.1 Future developments

The CrewManagerApp app offers the core functionality which can be used by event crew managers and crew members to communicate with each other. However, to be efficient and stand out in the market it still needs to provide more practical options to be used as the sole management app by crew companies. These features and solutions are presented below:

7.1.1 Ranking feature

One of the features I would add in the future would be an algorithm that can calculate a crew member's performance rating based on multiple reviews; the crew member with the highest rating would appear at the top of the managers and clients list. This feature can be added by allowing the client to sign up to the app and instead of emailing a crewing company for crews, they can use the app to request for the crew. On the client app, it will show the list of all the crews, and once the job is over, they can give a rating to each crew based on their performance.

7.1.2 Calendar

Another feature I add in the future will be calendar which automatically displays crew member’s accepted job request alongside and relevant information about the job. This feature will also help prevent the risk of crew managers double-booking their employees.

7.1.3 Time sheet

Another feature I could add in the future is the timesheet. At the start of each month, the app creates an empty timesheet for each crew member. Additionally, when the crew accepts any job request, their working hours are automatically added to a timesheet. This will help the crew manager to verify each payment.

7.1.4 Employees availability / job cancelation

Allowing Crew members to set their availability and being able to cancel jobs already accepted is another practical feature I would include in future development. The users will be able to set their availability for specific days and time based on their availability when the crew manager is sending job requests it will only display the list of available Crew members.

7.1.5 Notification

During testing, the majority of the participants complained about notification panel –when a job request/message is received there no alert or notification, the user has to keep checking the app for messages and job requests. I would ensure that this feature is rectified within the future development.

7.1.6 Multi-platform

Currently, the app can only be used on IOS. I would ensure that the App will be expanded into a web-based application and available on android devices so that the app can be used on different platforms.

Bibliography

[1] Eventbrite UK Blog. (2012). An Introduction to The UK Event Industry in Numbers - Eventbrite. [online] Available at: https://www.eventbrite.co.uk/blog/academy/uk-event-industry-in-numbers-ds00/

[2] Lusk, G. (n.d.). UK Event Industry statistics factsheet. [online] blog.liveforce.co. Available at: https://blog.liveforce.co/uk-event-industry-statistics-factsheet.

[3] Statista. (2014). Number of smartphone users worldwide 2014-2020 | Statista. [online] Available at: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[**4]** https://www.gallowglass.com/gallowglass-client-app/. (2020). Gallowglass. [online] Available at: https://www.gallowglass.com/gallowglass-client-app/

[5] www.capterra.com. (n.d.). Event Staff App Reviews and Pricing - 2020. [online] Available at: https://www.capterra.com/p/132149/Event-Staff-App/

[6] accounts.google.com. (n.d.). Google Calendar. [online] Available at: https://calendar.google.com/calendar/r?pli=1

[7] Apple.com. (2019). Swift - Apple Developer. [online] Available at: https://developer.apple.com/swift/.

[8] Mac App Store. (n.d.). Xcode. [online] Available at: https://apps.apple.com/gb/app/xcode/id497799835?mt=12

[9] Firebase. (2018). Firebase. [online] Available at: https://firebase.google.com/.

[10] Firebase. (2019). Firebase Realtime Database | Firebase Realtime Database | Firebase. [online] Available at: https://firebase.google.com/docs/database.

[11] Firebase. (2019). Firebase Authentication | Firebase. [online] Available at: https://firebase.google.com/docs/auth.

[12] CocoaPods Dev Team (2019). CocoaPods.org. [online] Cocoapods.org. Available at: https://cocoapods.org/.

[13] Firebase. (2019). Firebase Authentication | Firebase. [online] Available at: https://firebase.google.com/docs/auth.

[14] Swift 3: Firebase Social Login - Facebook Authentication and CocoaPods. (2016). YouTube. Available at: https://www.youtube.com/watch?v=iSszeW1aH6I

[15] GitHub. (2020). SVProgressHUD/SVProgressHUD. [online] Available at: https://github.com/SVProgressHUD/SVProgressHUD

[16] GitLab. (n.d.). Files · master · Wouter / alerts-and-pickers. [online] Available at: http://git.addrenaline.nl/wouter/alerts-and-pickers/tree/master

[17] www.tutorialspoint.com. (n.d.). How to make phone call in iOS 10 using Swift? [online] Available at: https://www.tutorialspoint.com/how-to-make-phone-call-in-ios-10-using-swift

[18] https://www.hackingwithswift.com/example-code/uikit/how-to-send-an-email

[19] YouTube. (n.d.). Keyboard Magic. [online] Available at: https://www.youtube.com/playlist?list=PL_csAAO9PQ8aTL87XnueOXi3RpWE2m_8v

[20] GitHub. (n.d.). hackiftekhar/IQKeyboardManager. [online] Available at: https://github.com/hackiftekhar/IQKeyboardManager/tree/master/IQKeyboardManagerSwift

[21] Firebase. (2019). Cloud Storage | Firebase. [online] Available at: https://firebase.google.com/docs/storage.

[22] Haskins, A. (n.d.). firebaseopensource. [online] firebaseopensource. Available at: https://firebaseopensource.com/projects/firebase/firebaseui-ios/

[23] Firebase. (2019). Firebase Authentication | Firebase. [online] Available at: https://firebase.google.com/docs/auth.

[24] Firebase. (2019). Firebase Realtime Database | Firebase Realtime Database | Firebase. [online] Available at: https://firebase.google.com/docs/database.

[25] Firebase. (n.d.). Authenticate Using Google Sign-In on Android. [online] Available at: https://firebase.google.com/docs/auth/android/google-signin

[26] Firebase. (n.d.). Authenticate Using Facebook Login on iOS. [online] Available at: https://firebase.google.com/docs/auth/ios/facebook-login

[27] GitHub. (2020). SwiftKickMobile/SwiftMessages. [online] Available at: https://github.com/SwiftKickMobile/SwiftMessages

[28] GitHub. (n.d.). hackiftekhar/IQKeyboardManager. [online] Available at: https://github.com/hackiftekhar/IQKeyboardManager/tree/master/IQKeyboardManagerSwift.

[29] GitHub. (2020). SVProgressHUD/SVProgressHUD. [online] Available at: https://github.com/SVProgressHUD/SVProgressHUD.

[30] York, M. (2020). Matt/AlertsandPickers. [online] GitHub. Available at: http://git.addrenaline.nl/wouter/alerts-and-pickers

[31] Material Design. (n.d.). *Material Design*. [online] Available at:

https://material.io/resources [Accessed 9 Jun. 2020]

Appendix

Git Repository link: https://github.com/mkhan012/Mkhan012Project

Folders Contents:

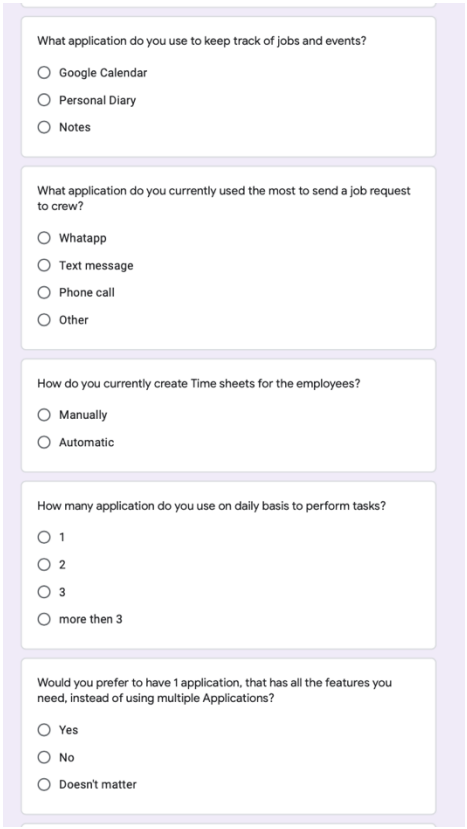
Please Note: All my code can be found in the following directory:
App/CrewManager/**ViewController**
App/CrewManager/**UtilitiesFolder**

The rest of the folders contains System files and third-party libraries.

LibrariesFolder: Contains all the third-party libraries used.

Appendix A

A Manager Questionnaire



What application do you use to keep track of jobs and events?

- Google Calendar
- Personal Diary
- Notes

What application do you currently used the most to send a job request to crew?

- Whatapp
- Text message
- Phone call
- Other

How do you currently create Time sheets for the employees?

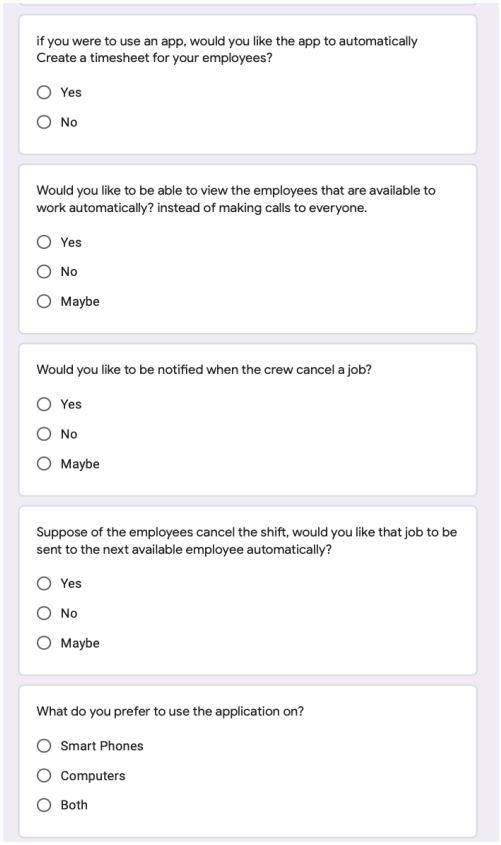
- Manually
- Automatic

How many application do you use on daily basis to perform tasks?

- 1
- 2
- 3
- more then 3

Would you prefer to have 1 application, that has all the features you need, instead of using multiple Applications?

- Yes
- No
- Doesn't matter



if you were to use an app, would you like the app to automatically Create a timesheet for your employees?

- Yes
- No

Would you like to be able to view the employees that are available to work automatically? instead of making calls to everyone.

- Yes
- No
- Maybe

Would you like to be notified when the crew cancel a job?

- Yes
- No
- Maybe

Suppose of the employees cancel the shift, would you like that job to be sent to the next available employee automatically?

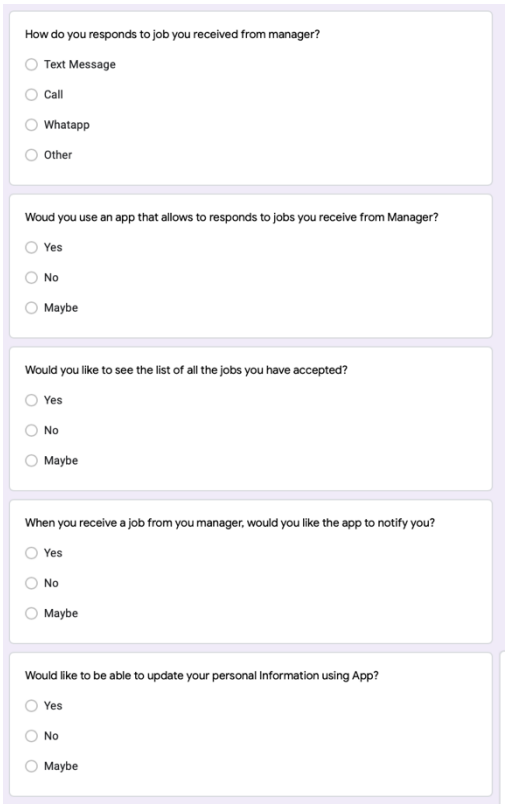
- Yes
- No
- Maybe

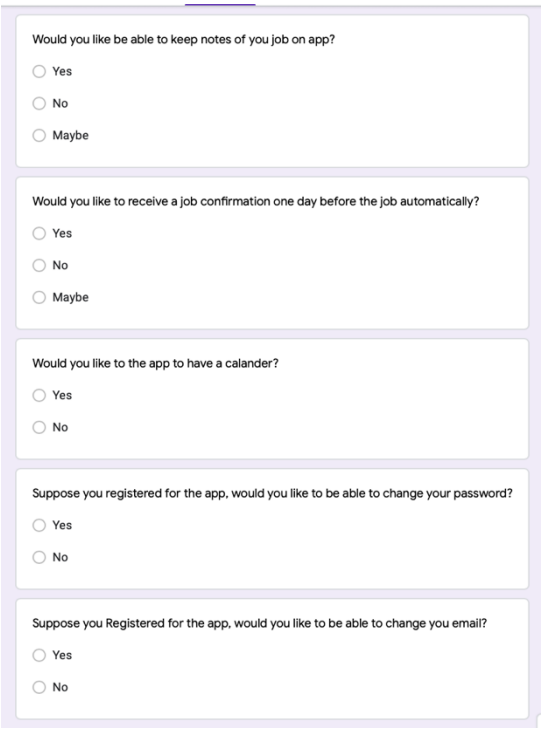
What do you prefer to use the application on?

- Smart Phones
- Computers
- Both

Appendix B

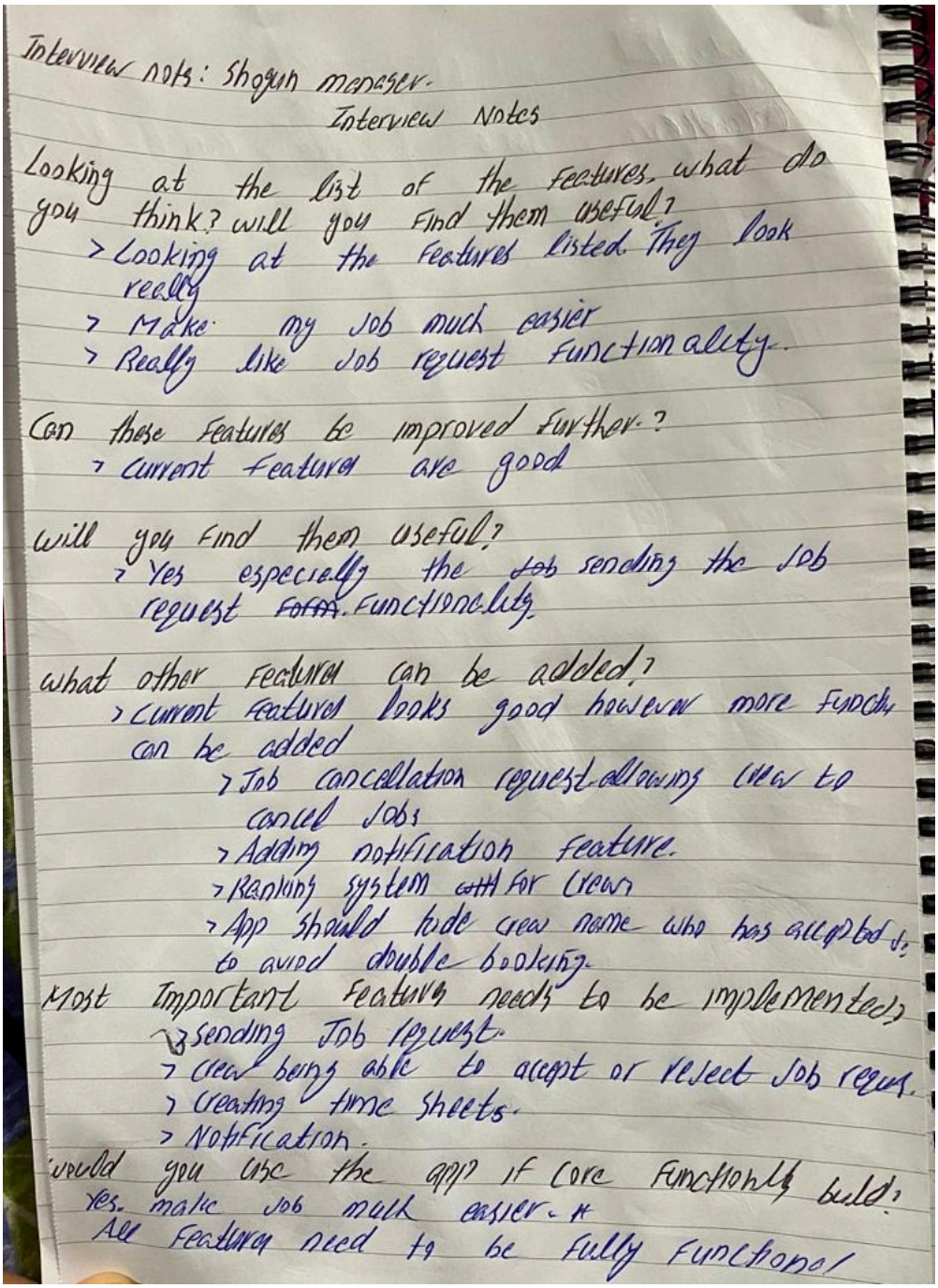
Crew Members Questionnaire





Appendix C

A Manager Interview Notes



Interview nots: Shogun manager.

Interview Notes

Looking at the list of the features, what do you think? will you find them useful?
> Looking at the features listed. They look really
> Make my job much easier
> Really like job request Functionality.

Can these features be improved further.?
> current features are good

Will you find them useful?
> Yes especially the ~~job~~ sending the job request ~~form~~. Functionality.

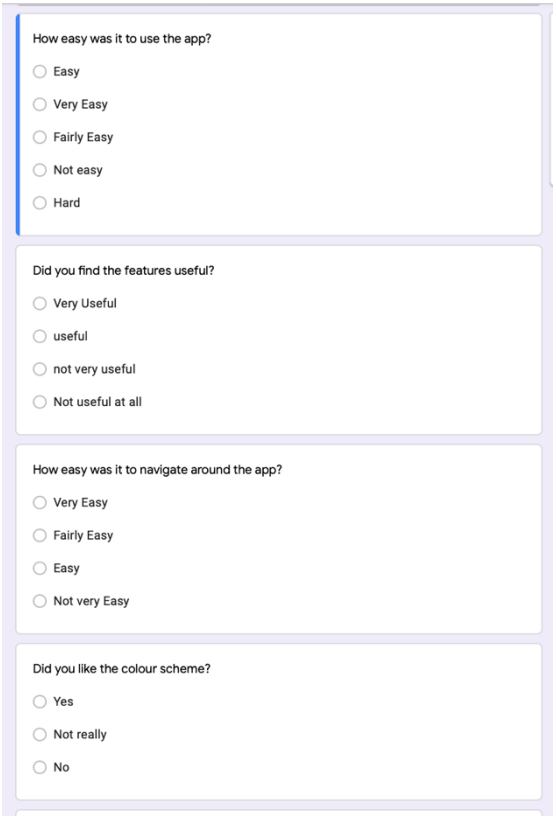
what other features can be added?
> current features looks good however more functions can be added
> Job cancellation request allowing crew to cancel jobs
> Adding notification feature.
> Ranking system ~~with~~ for crews
> App should hide crew name who has accepted job to avoid double booking.

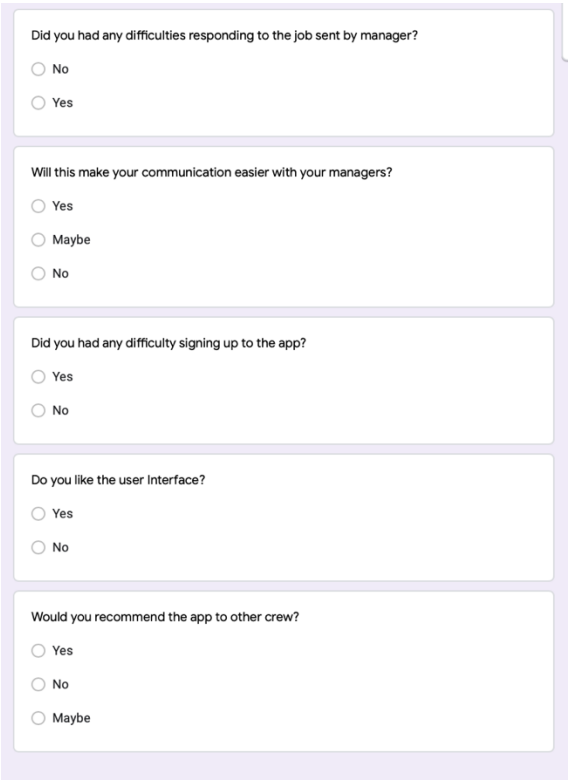
Most Important features needs to be implemented?
> sending Job request.
> crew being able to accept or reject job requ.
> creating time sheets.
> Notification.

would you use the app? if core Functionlity build?
yes. make job much easier. ✱
All features need to be fully functional

Appendix D

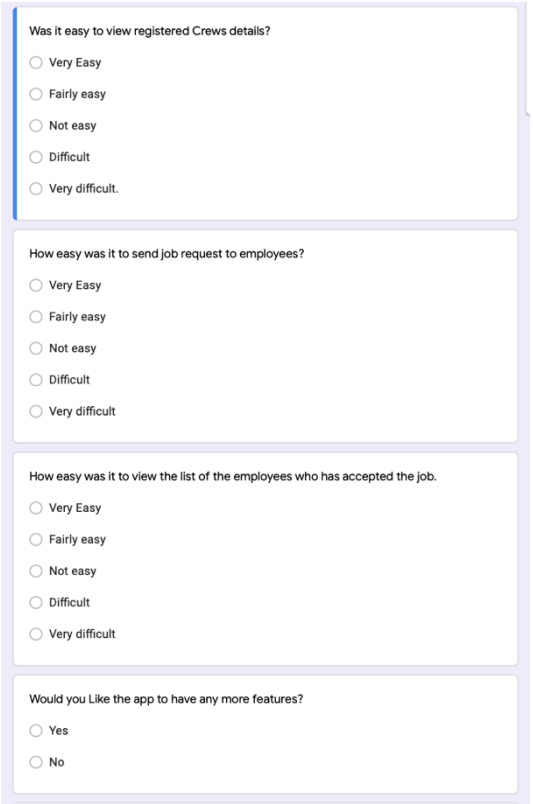
A User Testing Questionnaire





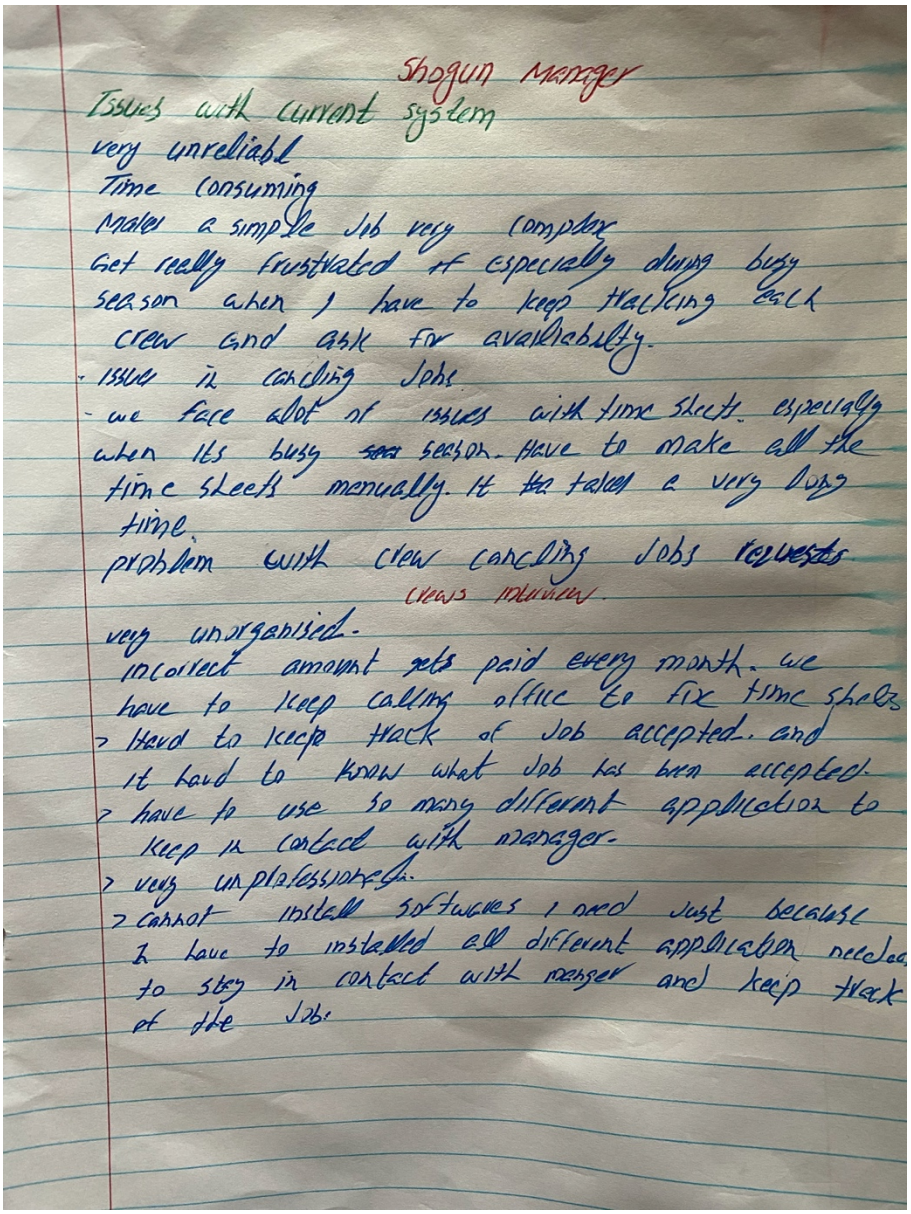
Appendix E

A User Testing Questionnaire



Appendix F

Interview Notes



Shogun Manager

Issues with current system

very unreliable

Time consuming

Make a simple Job very complex

Get really frustrated at especially during busy season when I have to keep tracking each crew and ask for availability.

- issues in canceling Jobs
- we face alot of issues with time sheets especially when its busy season. Have to make all the time sheets manually. It takes a very long time.

problem with crew canceling Jobs requests.

Crews interview.

very unorganised.

incorrect amount gets paid every month. we have to keep calling office to fix time sheets

> Hard to keep track of Job accepted. and it hard to know what Job has been accepted.

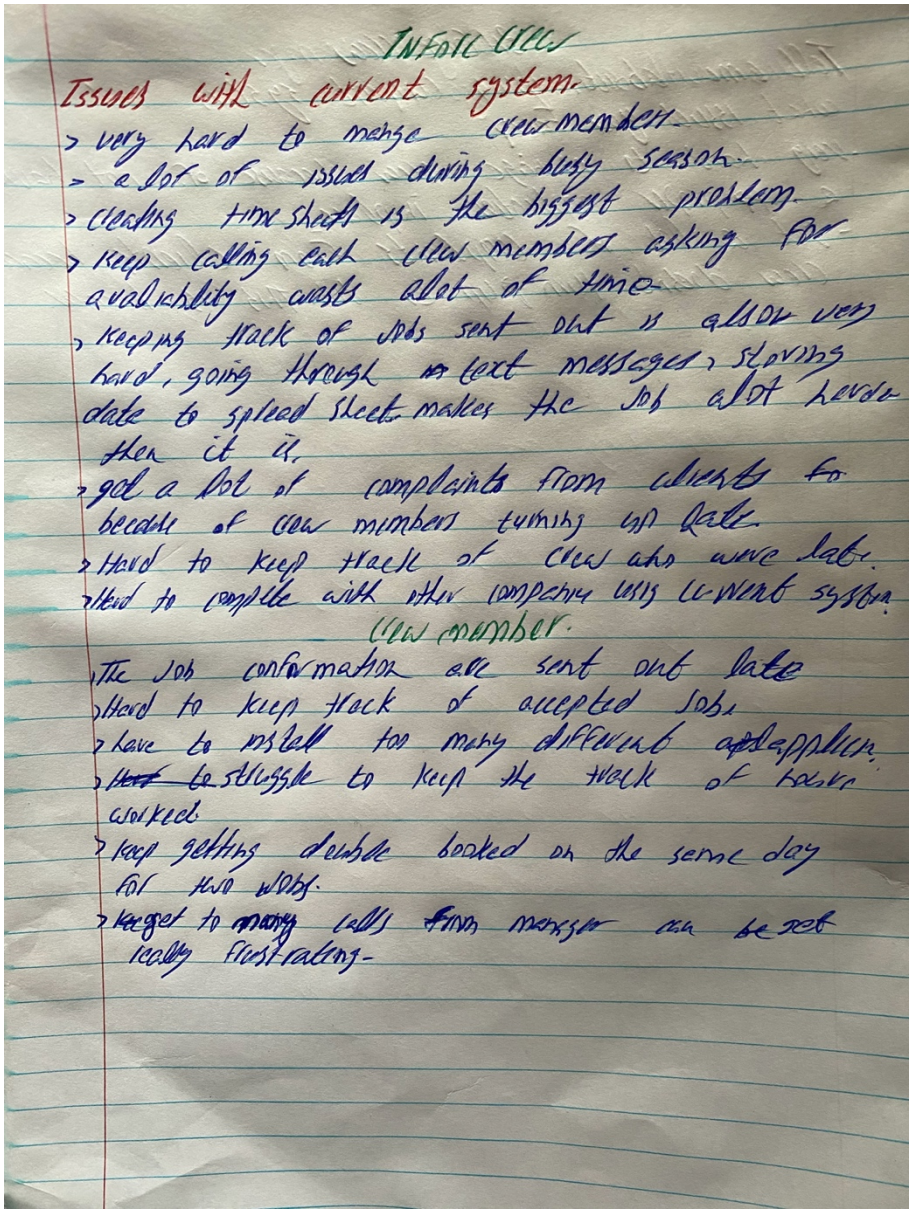
> have to use so many different application to keep in contact with manager.

> very unprofessional.

> cannot install softwares I need just because I have to installed all different application needed to stay in contact with manger and keep track of the Jobs.

Appendix G

Interview Notes



Info for crew

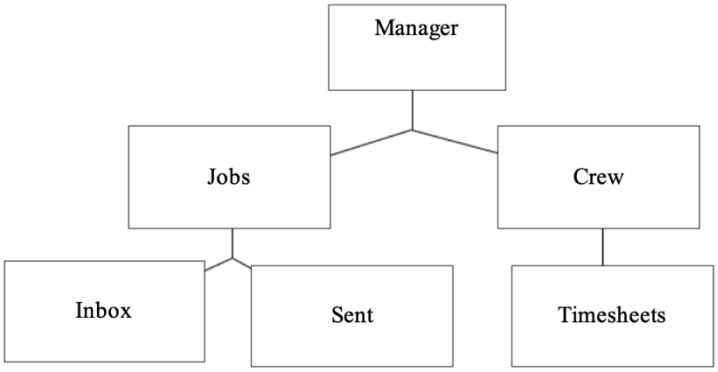
Issues with current system.

- very hard to manage crew members.
- a lot of issues during busy season.
- creating time sheets is the biggest problem.
- keep calling each crew members asking for availability wastes alot of time.
- keeping track of jobs sent out is also very hard, going through text messages, storing data to spread sheets makes the job alot harder then it is.
- get a lot of complaints from clients to because of crew members turning up late.
- Hard to keep track of crew who were late.
- Hard to compete with other companies using co-workr system.

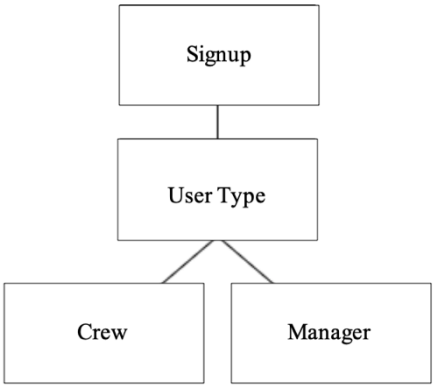
Crew member.

- The job conformation are sent out late
- Hard to keep track of accepted jobs
- Have to install too many different applications.
- ~~Have~~ to struggle to keep the track of hours worked.
- Keep getting double booked on the same day for two jobs.
- Forget to many calls from manager can be get really frustrating.

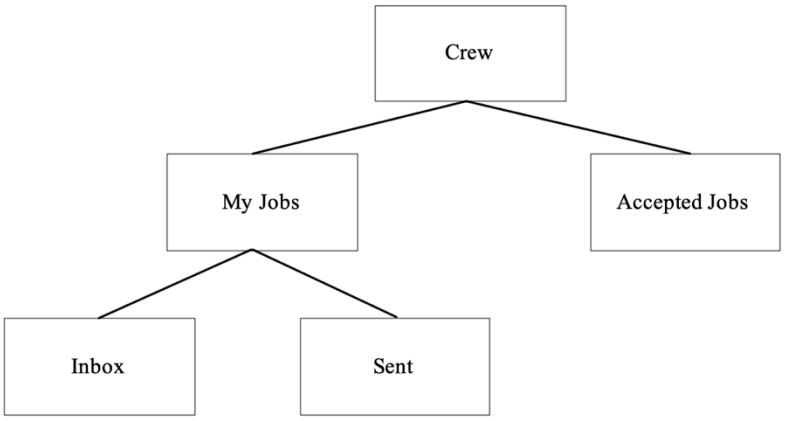
Appendix H



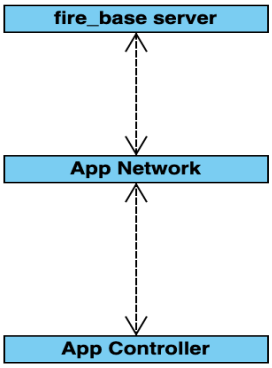
Object chart for the manager



Object diagram for system.



Object chart for the Crew member



Shows diagram for data transfer