CA400 Final Year Computer Applications Project

Functional Specification

ICE - a cross platform personal safety application.

Picture: our ICE application logo

Hannah O'Connor - 16382283
Catherine Mooney - 16416052

Submission Date: 21/11/2019

0 Table of contents

1. Introduction

1.1 Overview 2
1.2 Business Context 3
1.3 Glossary 3

2 General Description

2.1 Product / System Functions 4
2.2 User Characteristics and Objectives 4
2.3 Operational Scenarios 5
2.4 Constraints 6

3 Functional Requirements

3.1 Registration 7
3.2 Account Set Up 8
3.3 Follow me Session 9
3.4 SOS Mode 9
3.5 Network Alert 10
3.6 Emergency Services Message 11
3.7 Social Media Feed 11
3.8 Safe Zones 12
3.9 Fake Call 13

4 System Architecture

4.1 System Architecture Diagram 14

5 High-Level Design

5.1 Context Diagram 15
5.2 Physical Data Flow Diagram 16
5.3 Logical Data Flow Diagram 17
5.4 Class Diagram 18
5.5 Use Case Model Diagram 19
5.6 UI Mocks 20

6 Preliminary Schedule

6.1 Gantt Chart 21
6.2 Task List 24
6.3 Product Backlog (Trello Board) 25

7 Appendices 26

1 Introduction

1.1 Overview

Safety has always been a big concern and priority for adults and their children. From doing regular day to day errands, to making your way home, alone, especially at night. Parents are constantly worried about where their children are and if they made it to their destination ok. Young girls have the constant fear of getting into taxis unaccompanied or walking down unlit streets. Men and women of all ages are aware of the on-going dangers faced by society today. It's inevitable that people have to make trips alone, be it short or long journeys. But it wouldn't be near as daunting if you have reassurance that your guardians know where you are and that help is easily accessible. While we have many uses for our smartphone devices, one thing that it could have great purpose for is personal safety. This is where our application comes into use, providing that reassurance and extra comfort for those within today's society, through your phone.

ICE is a cross-platform iOS and Android personal safety mobile application built through the React Native framework. The name comes from the well known 'In Case of Emergency' and we took inspiration from the name ICE for the apps logo. The app will be designed for all types of user, with the idea being that only those within your trusted circle of contacts can monitor your real time location, communicate and be updated with notifications and alerts about your current status. Our mission is to establish a platform aiding those without protection and help to make a difference to when a situation could get serious. We plan to develop a web application alongside this, one that can be accessed via phone or desktop as another means of receiving updates from those within your inner circle. The central idea behind the application is that the app can provide a variety of personal safety features rather than just containing an SOS button and GPS tracking. Most personal safety app developed globally are delivered on a subscription basis and we believe that the price of safety should be free of cost.

1.2 Business Context

As the SafeZone app is already very popular among students of DCU, we can already see that there is a desire for apps like this. However, ICE is not only for students, it is for the general public. We did some research and found that there is nothing like this on the market in Ireland designed for men, women and children alike.
ICE could be potentially be made into new instances and be utilised within schools and universities, where they could have their own customized version. The SafeZone app is utilised specifically for institutes, Universities like DCU for example, and only cover the grounds of that institute, used to alert security of any emergencies. Our app, however, will cover the user no matter where they are, including trips to and from these institutes, where there is no security within your range.
Another possible direction for this application is to allow advertising upon uploading to Apple App or Google Play store and generate revenue to support the future growth of it.

1.3 Glossary

React-Native
A JavaScript framework, based upon React, which allows you to create cross-platform applications for iOS, Android & Web Applications. All from a single codebase.

Firebase
A Google cloud-based mobile application development platform. It acts as the server for the application, along with providing tools and SDKs for the development of our backend too.

Firebase Real-time Database
No SQL and cloud hosted database that syncs and stores data across our app's users in real time.

Firebase Cloud Storage
A Firebase SDK to allow for the storage of user-generated content, such as photos and videos.

Firebase Phone Authentication
A service that authenticates users using through their mobile phone number. When a new user signs up to the app, it sends a verification text to the provided number before storing to database. Once verified, that number will be assigned its only unique user id.

2 General Description

2.1 Product / System Functions

Displayed below are the main functions in which we plan to incorporate into our app, the functionality is subject to change over the course of development. However, for the time being, our proposed functions are listed below. These functions will require a user to first be registered successfully and verify their given phone number.

- Network of Trust
- Network of Trusts Invitations (add others or be apart of others)
- SOS activation
- Network alerts & Tracking
- Push notifications
- Follow me session
- Fake call activation
- Emergency text generator
- User profiles
- Medical profiles
- Safe zones
- Safe word
- Social media feed
- Fingerprint access to modify user information
- Messaging Service (among network)
- SMS functionality

The main functionality will be discussed in further detail within section 3 *'Functional Requirements'*.

2.2 User Characteristics and Objectives

We plan to design our app in a way that is open to users of all ages. Our app is not exclusive of any particular groups as we feel personal safety shouldnt be an exclusive for anyone. The only condition which our users must adhere to is that they must carry an iOS or Android enabled smartphone or device with a registered phone number.

Given that this application will be available to all, we want to establish a universal appeal for our user interface. Our apps logo is very welcoming so that from the outset, users feel relaxed in the knowledge that the app will be user friendly. From the moment the user enters the app, we plan to ensure

ease of use throughout. Our main goal for the UI of ICE is to allow for usability, readability and consistency. We will make use of fun and bright, but distinguishable colors and be considerate of font sizes and controls. We will make sure that transition from one section of the app to the other is very smooth and easily navigated. Whether acquiring rich technology skills or not, this app will be accommodating for all. We will base all UI decisions off our acquired knowledge from the previous year module CA357 (Human Computer Interaction) .

2.3 Operational Scenarios

User signs up

Following the user having installed the application onto their Android or iOS device, that user can then proceed to registration. If it is the first time on the application or there has been no prior registration, then the user will be presented with a sign up screen. The user will be required to enter their information to the mandatory field. When completed, a message will be sent to the inputted phone number where the user will be required to verify in order to proceed. Upon successful verification, the user must agree to the apps' terms and usages. The given phone number will be automatically registered with the 112 SMS service.

User edits their profile

Following sign up or through navigating to the user profile icon from within the bottom navigation bar, the user will be presented with their own profile. To modify it, the user must click on the 'edit' icon where they can alter the following stored data:

- Medical Profile
- Safe Zones
- Circle of trust
- Safe word

Upon completion the user must click 'save' at the bottom of the page.

User starts a 'Follow me' session

From navigating to the 'Follow me' from within the home menu, the must click on the function and wait for it to load. Here, the user will be asked to enter their destination, route and an approximated arrival time will be calculated. The user will have the option to modify this estimated time before clicking 'Start'. Following this activation, a notification will be sent to each member of the users' network along with their real time location, which will be retrieved and stored in database. Any given member of the user's network can query the location through clicking on the received notification from app or checking their inbox notifications (from bottom navigation bar).

User triggers a network alert.

Any given user of the app may want to alert their network of their real time location for some period of time without explicitly declaring any particular destination or arrival time. Here, the user will have clicked on the 'Network Alert' function and a notification will be sent. The location of the user will be monitored within the background and can be viewable by any in their network.

User adds/navigates to social media feed.
Given that the user is within the home menu, they will be presented with the 'Social Media' function. Here, the user must click on it and tap the '+' icon in the top right corner. A forum will be presented where can enter the text for their new post. The option to post to either to their network or to the public will be available. Once done, the user must click 'Share'.

User activates SOS mode.
A user can activate the SOS mode in one of three ways. These include:

- User says the safe word aloud whilst holding their device.
- User rigorously shakes their device.
- The button 'SOS' on main menu is touched by the user.

The activation of any of the above methods will create a 10 second countdown where the user can choose to cancel or wait. Notifications and alerts will be sent to their network. The mobiles' audio and camera footage will be recorded and sent to cloud storage where it can be then retrieved by their network.

User sends an emergency text message
User can send an emergency text through tapping on the function within the applications' main menu. Here the user will be presented will two modifiable attributes. These include:

- Which emergency service type is required (a scrollable list)
- Why its' required. (The default for this is: 'I need help and am unable to speak')

The user's location will be queried and reversed geocoded to a human reabled address. The option to forward medical info will be available too. The user must then press 'Send' one done.

User generates a fake call.
User can generate a fake call through navigating to the main menu and tapping on the 'Fake Call' button. Upon the loading of the function, the user can click 'Call me'. The default template will be shown and can be changed before it is activated.

2.4 Constraints

Time constraint
As we are quite restricted on our app development process due to the time, we will focus on implementing the basic functionality and ensure it all works accordingly with sufficient testing conducted. Over time we can grow the

application to incorporate more features and functionality to it. However, the management of time will be need to be a top priority to our team. Especially with the additional juggling of other lectures and coursework.

Firebase database constraint
With a limit set on our free account on Firebase, we're constrained on the amount that we can store, query and simultaneous connections.

Internet constraint
In terms of the user tracking, if there is a loss of Internet connection on the mobile device then the real time location won't be able to be retrieved & updated on private map to network of trust. A user must have internet access when attempting to open or operate the corresponding web app.

Hardware constraint
There is a hardware constraint when it comes to testing our devices on our machines as iOS devices can only be tested on OSX. However, given that each member has a separate OS type laptop, it means that one can take over the management and testing of iOS on a MacBook and the other takes care of the Android platform with their Windows machine.

3 Functional Requirements

3.1 Registration

Description

Registration is the first step and a requirement to using this application. The sign up form will require the new user to enter a valid mobile phone number. This step is crucial as it is how we can tie the user to that device. This avoids the possibility of a user being logged out and unable to activate safety functions .Other basic information will be required too, like name, email, etc.

Once verified, a unique user id will be shown to user, encouraging them to save it in a safe place. That id is what will allow the user to log into the corresponding web application.

Criticality

Without registration, potential users of the app wouldn't be able to access it's services. Through this function we want to enforce our rule to that only one mobile can be tied to a single account at any given time.

Technical issues

For the phone number validation process we will leverage Firebase's Authentication SDK.

Dependencies with other requirements

The user must have the application installed on their iOS/Android mobile device. Alongside that devices' phone number.

3.2 Account Set Up

Description

Following successful registration, the user can proceed to set up their account and details. Here the user will first be asked to establish their network of trust. They'll be required to add them via phone number and if that given person is not already registered with the application they'll be sent an invitation via SMS, otherwise, a request via the app where they can accept or decline. The Network of trust will then be established in their home screen.

Next, the user can choose to add their medical information, which can be later updated within their profile and optionally shown within their lock screen. Else, the user can skip this and edit later.

Lastly, the user can choose to define their safe zones as soon as they first launch the app. They user can also choose to ignore this step and edit later. *See section 3.8 for further details.*

Criticality

The criticality of this step is high as it enables the user to get the most out of the app's offered services.

Technical issues

Upon establishing the network of trust, there may be a delay in the use of the application upon waiting on each respective members' response to join.

The safe zones will be implemented through geofences, which can then be viewable by all members of that user's network.

Dependencies with other requirements

User is required to be successfully registered with the app.

3.3 Follow me Session

Description

This function allows for the user to be tracked by their network of trust. The sole purpose of this service is to allow the user to enter in their destination on the map and an optional estimated time. A route will be calculated and the corresponding network will be notified. That network will then be able to view that user in real time on a private map. The app will trigger a reminder if the user has not reached the destination on time and if no response from the user is received, then a network alert is triggered where further actions can be made from there.

Criticality

This is the essence of the app - it's what allows users to be tracked on their journey home safety, knowing that they're being watched over.

Technical issues

For this feature, we'll be utilizing the **Firebase real-time database** to query and update the user's coordinates every second on map. The map and user location will be made viewable through using the **React-native-map**s library and a route can be calculated using **React-native-maps-directions.** As of now, we want to trigger the timer using **react-native-countdown-component.**

Dependencies with other requirements

For this feature to work to its best ability, then the user's network must be established and must have accepted the permission in order for the application to access the devices' location.

3.4 SOS Mode

Description

This function will be put in place in order to accommodate more serious and life-threatening scenarios. We're attempting to develop three ways in which this mode could be triggered. Firstly, the user would simply tap the SOS button on the application. Secondly, the user could vigorously shake the device. Lastly, the user could clearly say aloud their defined safe word. This function would trigger a network alert and also contact emergency services on behalf of the user. The device's audio and camera footage will be retrieved & available to network, along with the device location. The record of this will then

be stored in the user's activity log to access footage from previous SOS activation.

Criticality

This function is highly critical in the case where there may be an emergency situation, the user can feel protected through the applications' functionality.

Technical issues

Once again, this function is another which will firstly need to have correct permissions in order to access devices' audio and camera. We plan to retrieve that footage using **react-native-camera** library. To detect the device shake, we'll use **react-native-shake.** To recognise the safe word, we'll use **react-native-voice.** To contact emergencies, we'll use **react-native-communications**.

Dependencies with other requirements

Firstly its' highly important that the user has registered successfully with the correct mobile number for the app to able to contact emergency services. The user's network must be established and must have accepted the permission in order for the application to access the devices' location too.

3.5 Network Alert

Description

This function is for a more subtle alternative when the user wants to alert their network of their location. Here an alert is sent to your network, notifying them of your location on the map and that it was activated. It enables real time tracking in the background without them having to specific any particular destination or route.

Criticality

Some users of our app may not want to be tracked continuously or they may just want to be cautious by sharing their location with their network on a given day. It enables tracking to the network whilst still being able to operate the other app services as normal.

Technical issues

This function could be implemented similarly to the *Follow me Session* through also using **react-native-map**s and **geolocation API**. To run this process as a background task, we're currently investigating which library would be the most

efficient. Our possible candidates for now include: **react-native-queue** and **react-native-background-task**.

Dependencies with other requirements

This is another feature that will also require specific permissions to be set so the device's geographical coordinates can be retrieved.

3.6 Emergency Services Message

Description

The purpose of this feature is to contact emergency services via text, through the app, where the exact users location could be sent and ensure the specified format is fulfilled. Here, we'll create a default emergency message template where the user can choose to be more specific, like choosing what emergency service they required from the drop down menu for example.

Criticality

We class this function as highly critical. It's important to have such a function in our app for in the case where a user is in an emergency and unable to speak. They may want to quickly send a preformatted text with their exact precise location - which could be unknown to the user at that time.

Many people are aware of the 112 text availability, however, there are some that are unaware that you need to have registered first. In the case of an emergency, if someone was to text 112 they would just receive instructions on how to register. Our app solves this by registering for the user.

Technical issues

Get coordinates with the **React native Geolocation API** and through reverse geocoding the fully formatted address could be retrieved. This function also needs to make use of **react-native-communications** to send SMS message.

Dependencies with other requirements

User must be registered successfully with the app so that so that they're registered to the 112 SMS service also. This feature also requires specific permissions to be set so the device's geographical coordinates can be retrieved.

3.7 Social Media Feed

Description

We're keen to implement this current feature as having an anonymous social media aspect to it could be beneficial to users safety. For instance, users could create posts about missing people, safety precautions, things to be wary of, etc. This could be based within a particular geographical region like a particular distance, which could be changed within settings. Also possibly, a feed of just updates of those within your network could be separated as its own component too.

Criticality

This feature itself isn't highly essential, but could serve as an extra means of safety through reminding people to be precautious. Not only that, but we believe that this could establish a strong sense of community where members within would look over for one another.

Technical issues

To implement this, we'll utilize NativeBase for it's UI features. Along with integrating React Navigation and its Stack Navigator and Tab Navigator to recreate a social media like feed.

Dependencies with other requirements

User must be registered successfully with the app.

3.8 Safe Zones

Description

The user will set places like their home, school, relatives, etc.as safe zones and then ICE will send a notification on behalf of the user to those within your circle by default when you enter one of your defined safe zones.

Criticality

The idea behind this is that rather than a user having to constantly send reminders to their network that they are safe, the app will do this for them once they have entered a zone they have set as safe.

Technical issues

We plan to send a user's network notifications upon them entering their defined safe zone, which will be created through using geofencing. While the user is within a specific range of the zone, the network will be notified that they are safe.

Dependencies with other requirements

User must be registered successfully with the app.

3.9 Fake Call

Description

This function allows the user to generate a fake call from inputted values. The user will select 'Fake Call' and then be presented with a template which they can then fill out. This will include from who and what time the call will start. If the user does not fill out these fields, the user will be called by default from an unknown number 30 seconds after activating the call.

Criticality

This function would be highly useful in situations where a person feels uneasy but not quite in danger. For example, if a stranger has stopped them to speak and they want an excuse to leave. This function would allow the user to remove themselves from uncomfortable situations.

Technical issues

For the moment, were planning on generating a call to the user's own phone number using the **react-native-phone-call** library. If this requires the user's network, we will try to find another alternative to just mock a phone call.

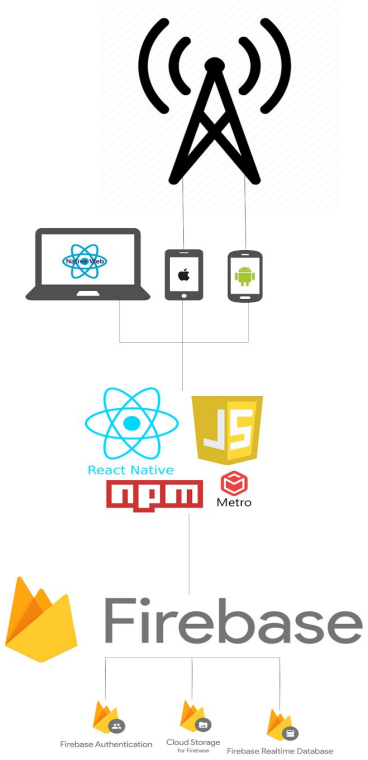
Dependencies with other requirements

User must be registered successfully with the app.

4 System Architecture

4.1 System Architecture Diagram

Diagram



Description

The above system architecture diagram shows the relationships between the different internal and external components of our system design.

At the top level of the diagram, it shows the hardware components. Those representing the iOS and Android device. Along with the Web application. The tower is a representation of our plan to utilize the mobile devices' network provider to send SMS messages from within the application.

Below that shows the main software components of this application and what will be used to create it. The pictorial logos show the following: React Native, JavaScript, npm and Metro (a JavaScript Bundler). Our codebase will live within GitHub and will be written within VSCode where we can take advantage of its extensions and leverage JS linting and testing tools. It will also integrate multiple React Native libraries and APIs. Such examples include: **react-native-shake**, **react-native-map**, **react-native-sms** and

react-native-phone-call. All of which will be required for the various functionalities that we plan to offer.

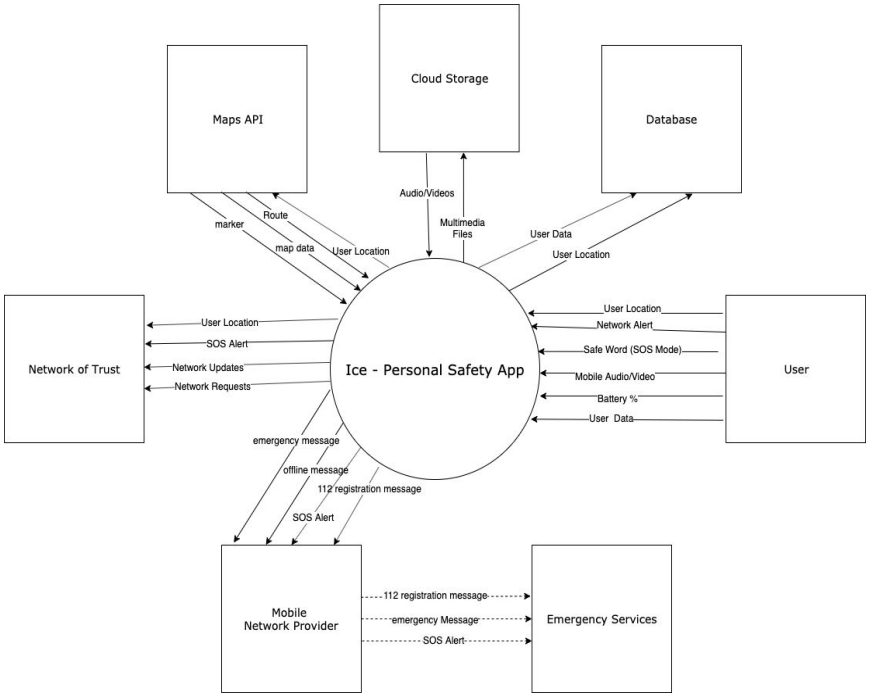
At the bottom level of this diagram is our applications' backend. Here the main component in this will be Google's Firebase. Firebase itself will be responsible for the hosting of ICE. We will also utilize some of it's available SDKs too. These include the user authentication system, the real-time cloud database and the cloud storage.

5 High-Level Design

Expanding on the recent section *System Architecture,* we've created multiple diagrams to demonstrate the high level functionality of our application. In this section we'll examine the system's components in greater detail through our use of UML diagrams, from defining how the user will interact with the application to the flow of data between said components.

5.1 Context Diagram

Diagram

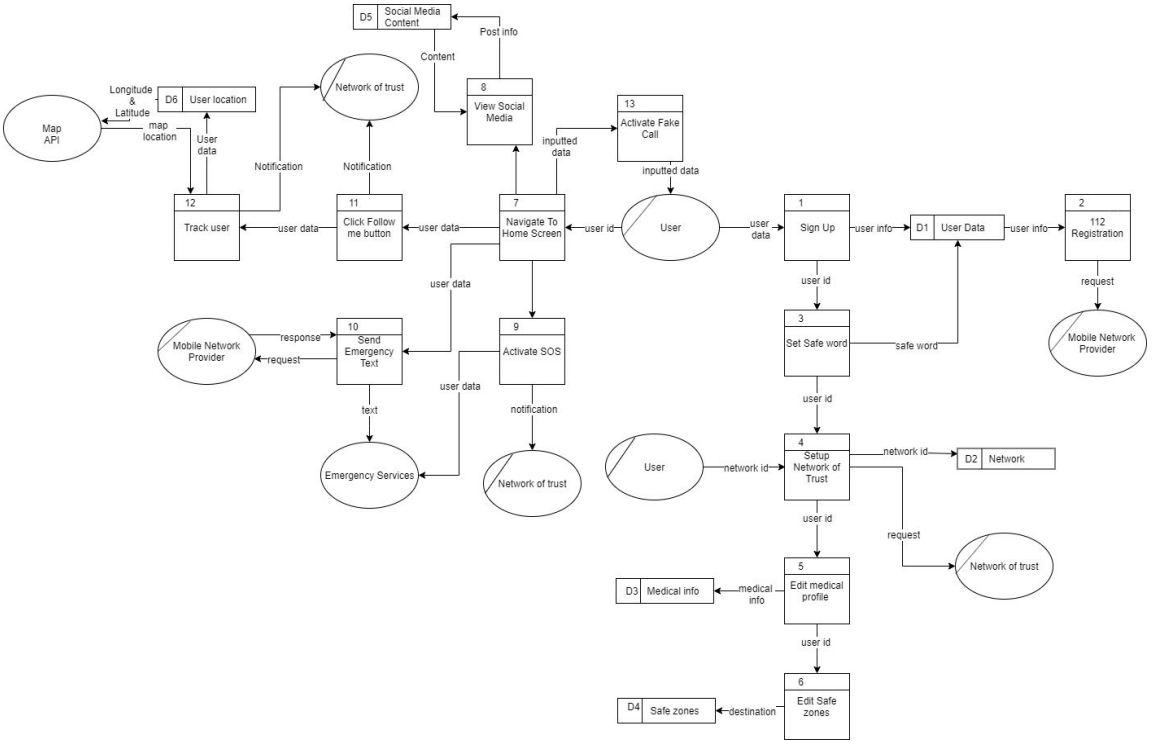


Description

In the above Context diagram, it shows the interactions and requirements of our various entities with the application. It showcases an overview of the data flow at the highest level. We'll examine the process of those data flows in more detail within the next diagram.

5.2 Physical Data Flow Diagram

Diagram

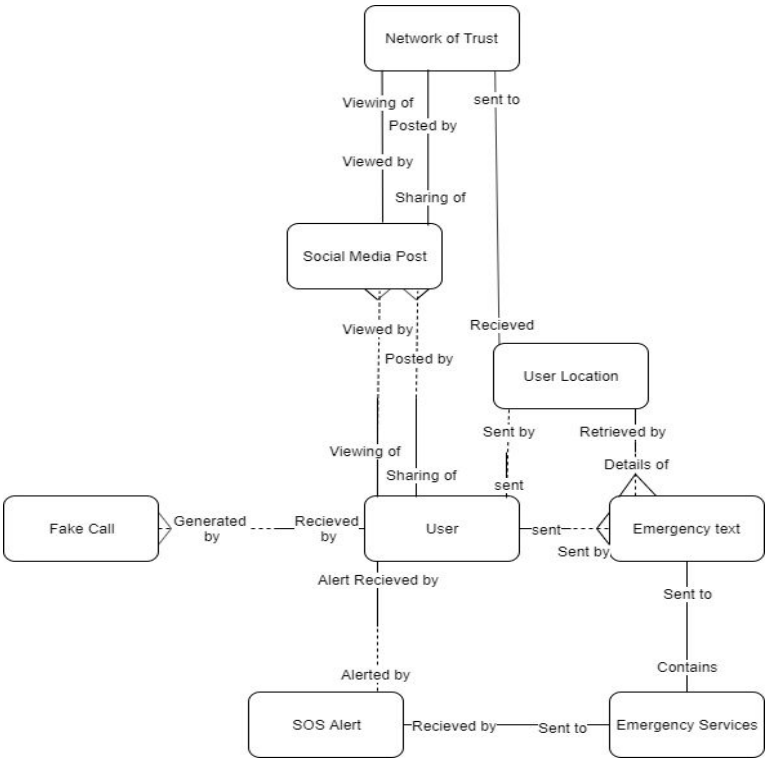


Description

This physical data flow diagram shows an overview of how ICE will be implemented. It shows the entities, processes and data stores that will be used in the making of the app. The processes are labeled in order, shown by the number, displaying the data that is passed between each one.

5.3 Logical Data Flow Diagram

Diagram

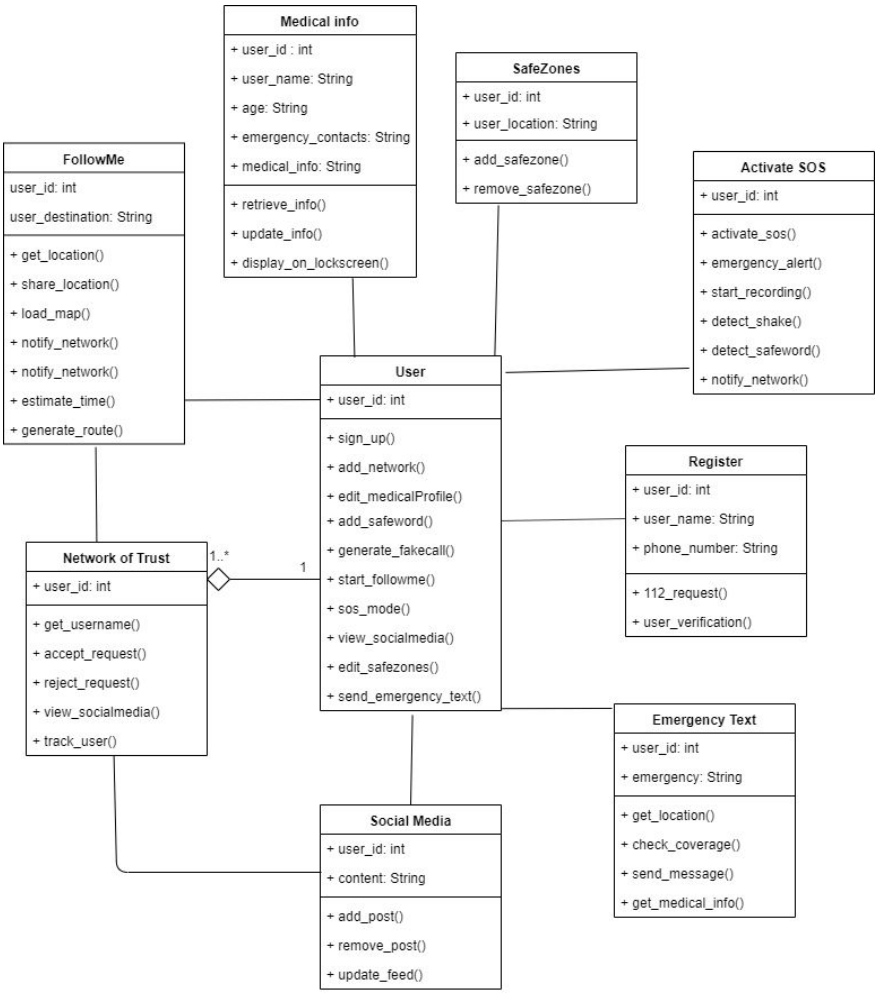


Description

This logical data flow diagram was made after examining the physical data flow diagram. It focuses on the different business activities, the round corner rectangles being the entities and the relationships between them. The ‘crows feet’ show that the many-to-many relationships and dotted lines depict the optional relationships.

5.4 Class Diagram

Diagram

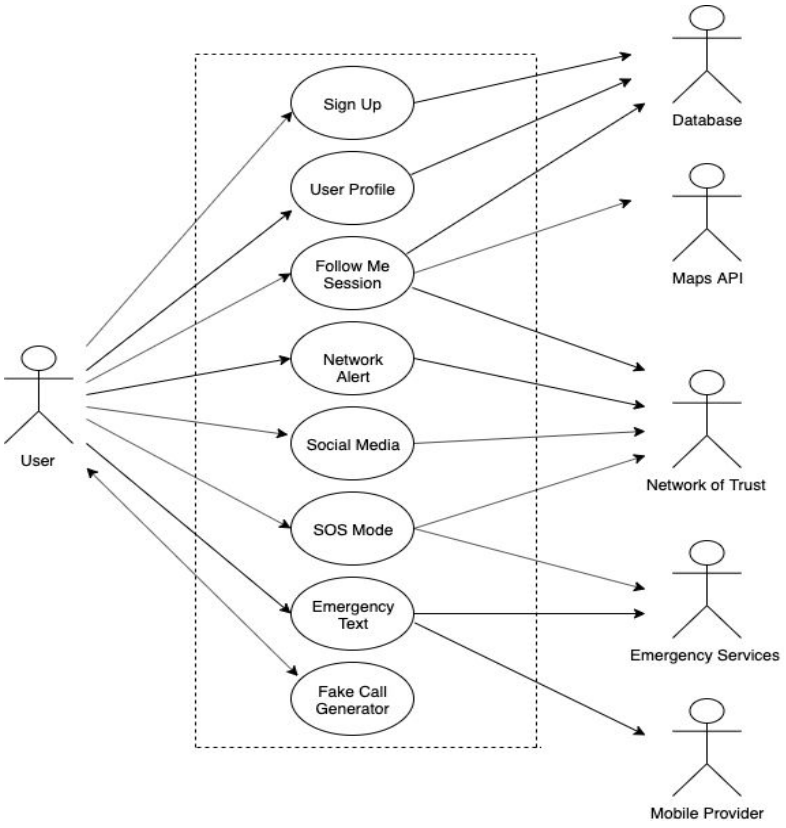


Description

This is a diagram of all the classes we will implement within our source code. Within each class, we listed the class variables and its' methods. We tried to include all the classes and functions that we think we will use, however, we may need to add more as our development progresses. But, for now, this is our basis to follow. The user and the network of trust is the only relationship which is one to many as each individual user has the option to have multiple users in their network of trust.

5.5 Use Case Model Diagram

Diagram



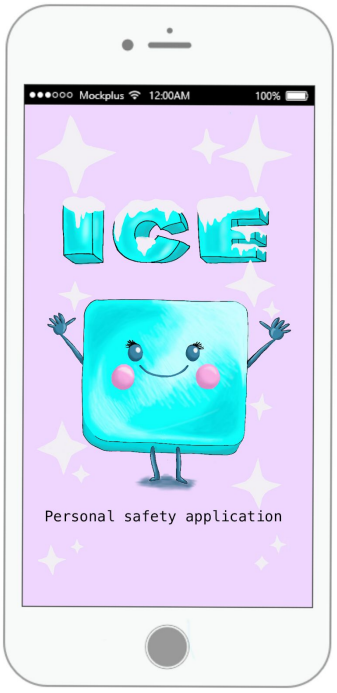
Description

The above Use Case model diagram shows the high level dynamic behaviour of our system with a user. The actor on the left shows the user of the application and the relationship between the various events of the system with the internal and external actors.

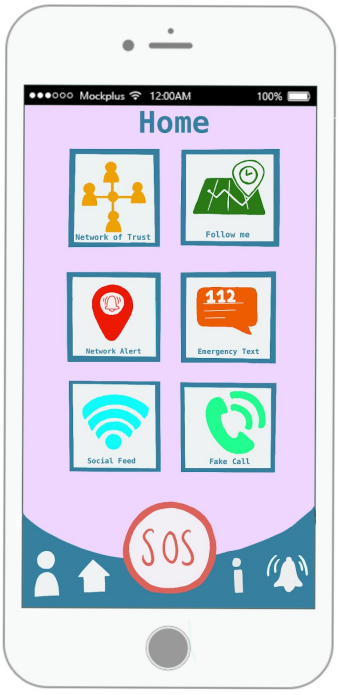
5.6 UI Mocks

Mocks

Splash screen:



Main Menu:



SOS Mode trigger:

Description

The above UI sketches shows the possible colours and design considerations that we would like to make when it comes to our interface. As you can see, we are opting towards adopting cool tones blues and purples for the overall theme, to achieve a calm feeling, as seen in our app's logo too.
As we are striving for a universal appeal, the interface itself is quite simple. This will ensure easy usability among all ages and technology backgrounds.

Our mocks and app logo were hand drawn an iPad using Procreate. We got the iPhone template from Mockplus.

6 Preliminary Schedule

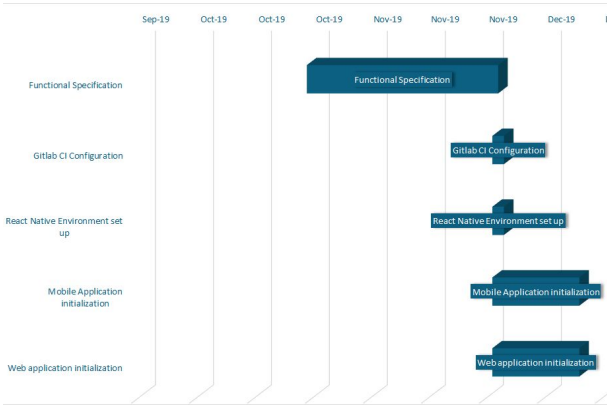
Below shows our current preliminary schedules for our final year project through the use of Gantt charts, a simple task list and trello boards.
We're aiming to be in contact on a regular basis with both one another and our project supervisor. We plan to work as a team through splitting up tasks, organising pair programming sessions, having online communication via slack and using software like Zoom when not on campus.

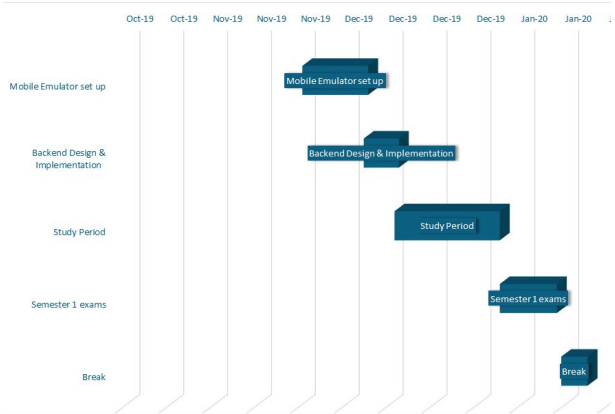
Given that we have included various types of testing within our schedules, we want to utilise Gitlab's C/I pipeline to automate those various test type executions. In that way we can run our defined tests against any new code changes, ensuring that nothing will break once pushed to our master branch.

6.1 Gantt Chart

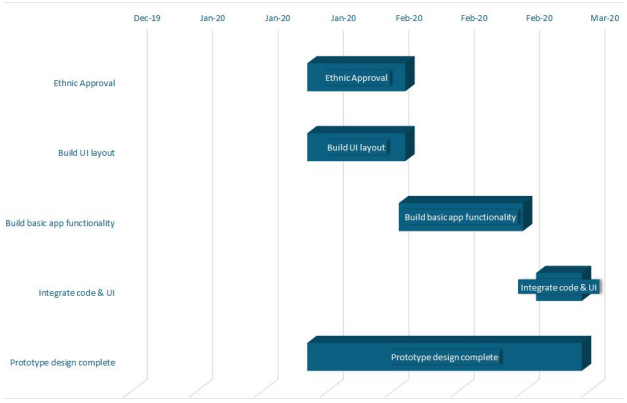
Charts

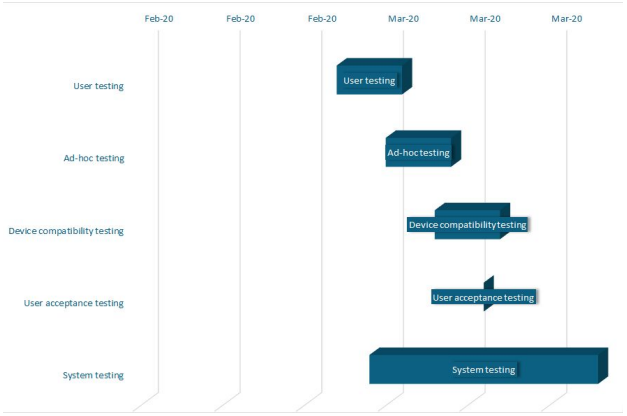
Semester one:

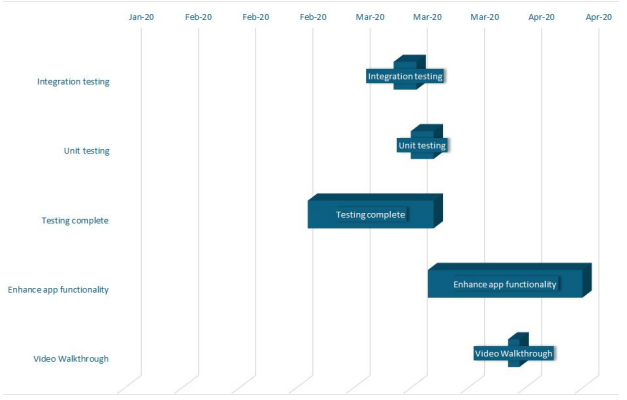


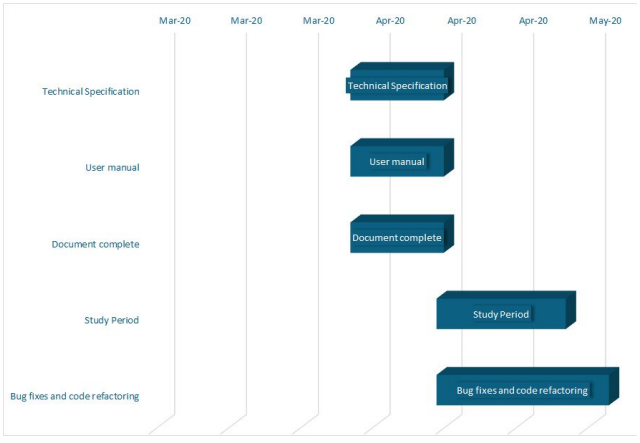


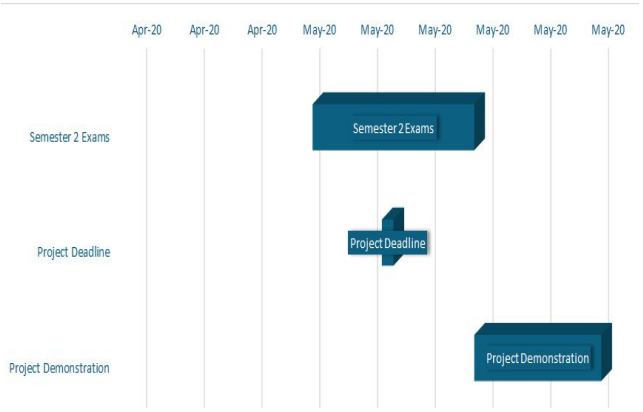
Semester two:











Description

The above gantt charts are sectioned according to the semester. They show our ideal schedule for the course of this project. However, this is subject to change. However, our goal for now is to stick tightly to the deadlines that we have set out.
Predicting the time frame for developing our source code can be tricky as it can be hard to determine the exact time, giving the unexpected bugs and errors that may later arise. We've attempted to incorporate some time for that nevertheless. Yet, speakingly realistically, it may require some extra time in our schedules.
We created the above charts through using our preexisting task list (shown in the next section) and Excel.

6.2 Task List

Chart

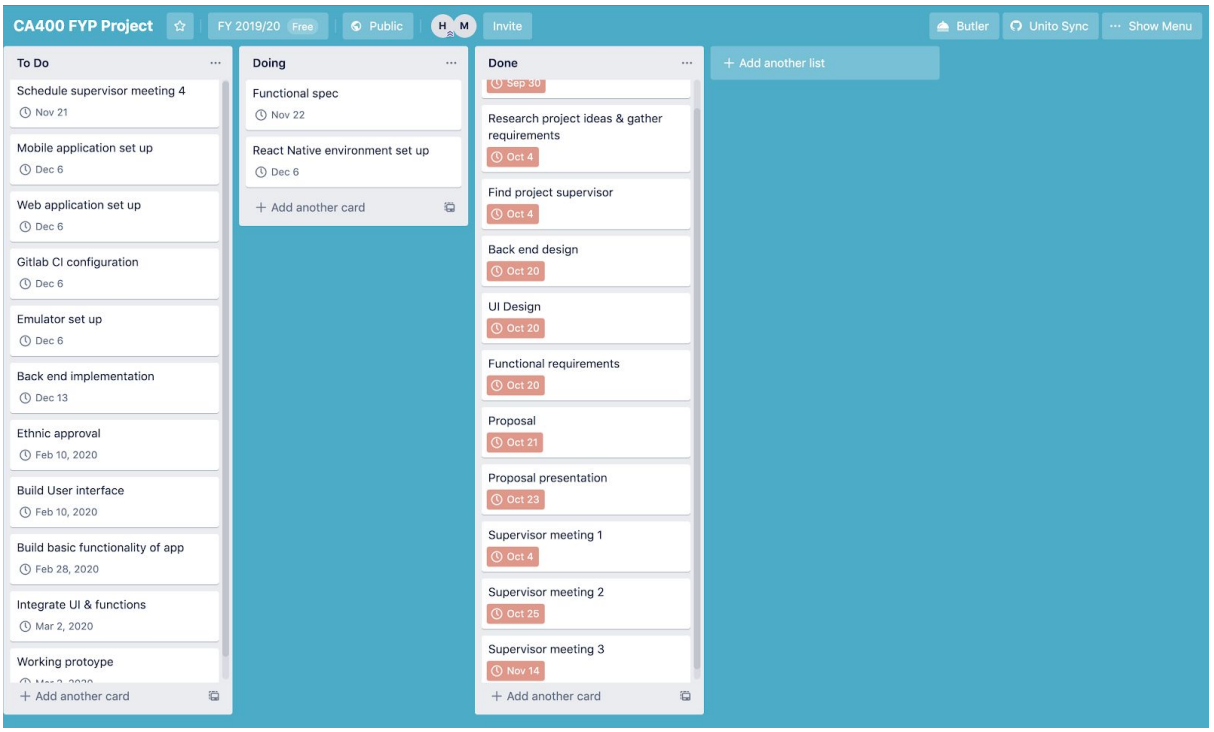
Task	Start Date	End Date	Duration
Research & Requirements Gathering	23/09/2019	4/10/19	14
UI Design	7/10/19	18/10/2019	11
Proposal	7/10/19	21/10/2019	14
Functional Requirements	21/10/2019	22/11/2019	30
Functional Specification	21/10/2019	22/11/2019	30
Gitlab CI Configuration	22/11/2019	23/11/2019	1
React Native Environment set up	22/11/2019	6/12/19	14
Mobile application intialization	22/11/2019	6/12/19	14
Web application intialization	22/11/2019	6/12/19	14
Mobile Emulator set up	22/11/2019	6/12/19	14
Backend Design & Implementation	6/12/19	13/12/2019	7
Study Period	13/12/2019	5/1/20	24
Semester 1 exams	6/1/20	18/1/2020	12
Break	20/1/2020	25/1/2020	5
Ethnic approval	27/1/2020	10/2/20	14
Build UI layout	27/1/2020	10/2/20	14
Build basic app functionality	10/2/20	28/02/2020	18
Integrate code & UI	2/3/20	8/3/20	6
Prototype design complete	27/1/2020	8/3/20	38
User testing	2/3/20	5/3/20	3
Ad-hoc testing	5/3/20	8/3/20	3
Device compabitliy testing	8/3/20	11/3/20	3
User acceptance testing	11/3/20	14/3/2020	3
System testing	14/3/2020	17/3/2020	3
Integration testing	17/3/2020	20/3/2020	3
Unit Testing	20/3/2020	23/3/2020	3
Testing complete	2/3/2020	23/3/2020	21
Enchance app functionality	23/3/2020	18/4/2020	26
Video Walkthrough	6/4/20	7/4/20	1
Technical Specification	6/4/20	18/4/2020	14
User Manual	6/4/20	18/4/2020	14
Documentation complete	6/4/20	18/4/2020	14
Study Period	18/4/2020	5/5/20	18
Semester 2 Exams	5/5/20	18/5/2020	14
Bug fixes and code refactoring	18/4/2020	11/5/20	17
Project Deadline	11/5/2020	11/5/2020	1
Project Expo	TBC	TBC	1
Project Demonstration	19/5/2020	29/5/2020	10

Description

The above task list shows the preliminary list of all our upcoming deliverables and milestones. We've attempted to groom our individual tasks as best possible. However, as previously mentioned, the timeline of the completion of each task is not set in stone.

6.3 Product Backlog (Trello Board)

Board



Description

Through both of our INTRA placements we used the scrum agile framework approach when it came to working with our teams. In an effort to follow and put into practice what we learnt, we plan to incorporate that methodology into our own project. Not only will this provide us with structure, but will help us to keep on track and be punctual. The goal is to have sprints every two weeks where both members would be assigned tasks from our product backlog. With that, we could update one another on our progress and help through any blockers.
To aid us with the above, we found the best online free tool to be Trello. With it, we can have an overall board to act as our Product Backlog with all tasks and stories and another board to keep track of those chosen tasks for our current sprint. Then, upon the completion of a task, we can request approvals on cards, to ensure that both members are satisfied with the new code change. On another note, through using such a board it will aid greatly in our bug & issue management also.

Link: https://trello.com/b/sDfu1uo7/ca400-fyp-project

7 Appendices

- ❏ React Native: https://react-native.org/
- ❏ Firebase for React Native: https://invertase.io/oss/react-native-firebase/
- ❏ Firebase Cloud Storage: https://invertase.io/oss/react-native-firebase/v6/storage/quick-start
- ❏ Firebase Real-time Database: https://invertase.io/oss/react-native-firebase/v6/database/quick-start
- ❏ Firebase Phone Authentication: https://invertase.io/oss/react-native-firebase/v6/auth/phone-auth
- ❏ 112 SMS functionality: https://www.112.ie/Terms___Conditions/146
- ❏ 112 how to: https://www.112.ie/Sending_a_text_to_112/144
- ❏ Learning React Native (O'REILLY book): https://www.oreilly.com/library/view/learning-react-native/9781491929049/
- ❏ Iphone mock screen template: https://www.mockplus.com/blog/post/iphone-wireframe-template
- ❏ React native shake library: https://www.npmjs.com/package/react-native-shake
- ❏ React native maps library: https://github.com/react-native-community/react-native-maps
- ❏ React native maps directions: https://www.npmjs.com/package/react-native-maps-directions
- ❏ React native countdown component: https://www.npmjs.com/package/react-native-countdown-component
- ❏ React native Geolocation API: https://facebook.github.io/react-native/docs/geolocation.html
- ❏ React native voice: https://github.com/react-native-community/react-native-voice
- ❏ React native communications: https://www.npmjs.com/package/react-native-communications
- ❏ React native camera: https://github.com/react-native-community/react-native-camera
- ❏ React native queue: https://github.com/billmalarky/react-native-queue
- ❏ React native background task: https://www.npmjs.com/package/react-native-background-task