Group Project (CM2305)

Final Report

Group 07: PAL (Peace and Love)

a personal security mobile application

Peace

and

Love

Team members:

Aaron Smith	**1739635**
Henrijs Princis	**1800857**
Euan Morgan	**1826905**
Benjamin Eddy[1]	**1827049**
Louis Davies-Cren	**1834661**
Marley Sudbury	**1838838**
Sara Abidi	**1880917**
Lizheng Huang	**1969507**

Client . . . . Kirill Sidorov

Supervisor . . . . Mohd Syafiq Bin Zolkeply

Published 15th May 2020

[1] Not an author of this report

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In this report, we will be documenting all aspects of development since the interim report, and provide you with a reflection of the entire development process. In the interim report, we began by interpreting the brief to recognise the requirements necessary to meet the client's expectations. From there, we designed several system diagrams and interface prototypes to demonstrate how we hoped to complete the project. As expected, there have been changes to several aspects of our initial designs. Because these were preliminary designs we primarily created to provide ourselves with some guidance.

This was our largest software development project yet and it was our first time using many of the technologies the project required. Despite this, we are pleased with the prototype we have created, especially given the current circumstances. After communicating with the client about concerns over certain features, there have been some changes since the initial requirements. We clarify what these changes are, and justify why they were made. For example, we expressed concerns over the 'grouping' features which would group users so that they dot walk alone. Our initial concerns were over whether this feature would contribute to improving the user's safety, but we were also unsure how we would go about implementing it. When we set out our requirements, this was not a top priority, as mentioned in the interim report.

We have performed several forms of testing on the prototype, details of which will be provided alongside the outcome and analysis of the results in chapter 4. Throughout this report we will also be analysing our performance as a team, as we believe the outcome of the team is just as important as the outcome of the product.

In summary, our prototype hosts a range of features to increase our users' safety; we provide users with 3 different route options to reach their destination (shortest, safest, and a combination of the two). The dead man's switch, deviation from path, or accelerometer will all initiate a countdown whilst recording audio and video, then prompt the user to call a trusted contact. There are also features we have not included, such as a heat-map displaying the crime data for users to inform themselves on the crime levels in particular area. We will also discuss why these features have not been included.

We have made a deliberate effort to only incorporate features we believe will increase the user's safety and, despite not including some of the features initially stated in the client's brief, we believe we have provided a comprehensive solution to the client's request. A video demo of the app can be seen via this link: https://youtu.be/qkQjLZABXX0.

Chapter 2

Final system architecture and design

2.1 Overall architecture of the application

Figure 2.1 and 2.2 shows the high-level data flow of the application.

On launching the application, the user is met with the 'Login' screen. Here, they are presented with the following:

- Screen title (Login)
- Username input field
- Password input field
- Login button
- Create an account button

Privacy was taken into consideration during the development phase of this screen. The team decided that a feature to hide the password characters

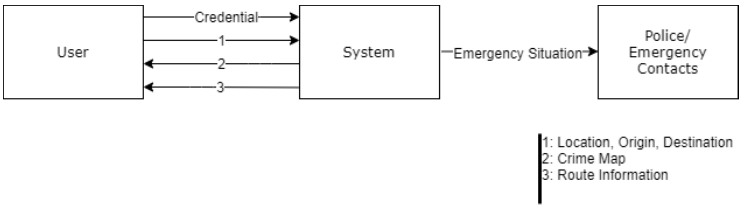


Figure 2.1: Level 0 of the data flow diagram

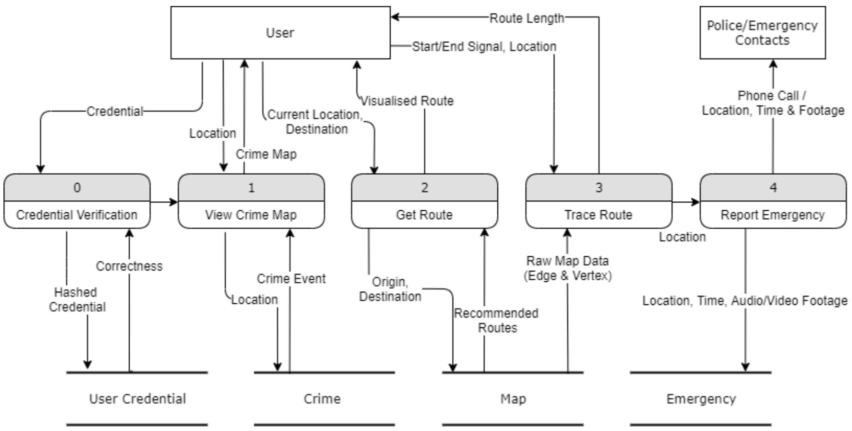


Figure 2.2: Level 1 of the data flow diagram

from showing up on screen was desirable, to help the user protect their input from any ‘screen snoopers’. This is automatically turned on, but can disabled to show the input text by toggling the eye icon in the password field.

If a user wishes to make a new account, they can do so by clicking the ‘Create Account’ button. This will take them to the ‘Create Account’ screen, where input prompts will require the user data necessary for account creation (see appendix A). Once the user creates their account, they are automatically logged in with their provided credentials.

However, should the user provide their username and password while at the ’Login’ screen, they will be logged in and taken to the app’s home (map) screen.

The home screen is a map centered on the user’s current location. This design was chosen by the development team for its efficiency in introducing the user to the application’s core functionality. This saves time for the large majority of users who have opened and logged into the application. The user can access the application’s ‘Main Menu’ from the home screen via the hamburger button. Using the menu, they can cycle through the additional functionalities, as well as launch straight into the core functionality of the system: begin a journey.

Once the user enters their chosen destination in the search bar featured in the map screen, and selects the address, their journey will begin and the application will begin tracking their progress. The user can cancel this process at anytime by selecting the ‘Cancel’ button; this will take them back to the home screen. After the journey has begun, the user will be presented with the ‘dead man’s switch’ to manually raise the alarm, should the need arise.

The items on the main menu are described in Appendix B.

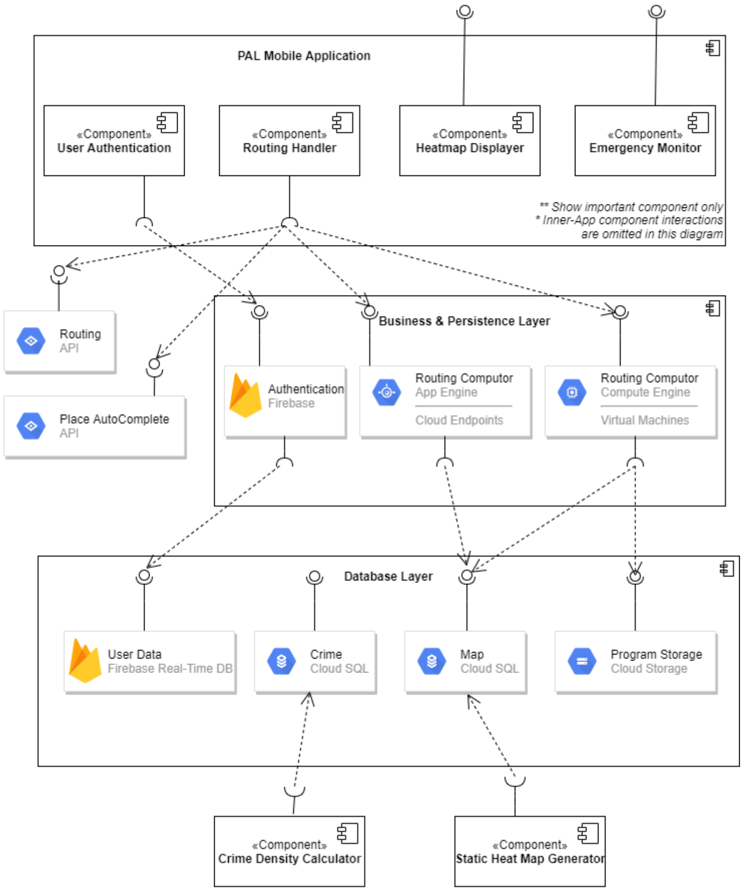

PAL Mobile Application
«Component» User Authentication
«Component» Routing Handler
«Component» Heatmap Displayer
«Component» Emergency Monitor
** Show important component only
* Inner-App component interactions are omitted in this diagram
Routing API
Place AutoComplete API
Business & Persistence Layer
Authentication Firebase
Routing Computor App Engine Cloud Endpoints
Routing Computor Compute Engine Virtual Machines
Database Layer
User Data Firebase Real-Time DB
Crime Cloud SQL
Map Cloud SQL
Program Storage Cloud Storage
«Component» Crime Density Calculator
«Component» Static Heat Map Generator

Figure 2.3: UML component diagram of the system

2.2 Overall architecture of the system

2.2.1 Presentation layer

The user interface is presented by React Native, a cross-platform open-source mobile application framework. Associated technical stacks are also used. For example, Redux is used to manage application state globally in a predictable way. By embracing React Native, the development team can learn once and write both Android and iOS compatible code.

2.2.2 Business layer

The business layer is fulfilled by both client-side React Native code and server-side Java code. This leads to a mixed service model, compromising the benefits and shortcomings of thin-client and fat-client. Business logic implemented in the client includes deviation detection, accelerometer-based jeopardy detection etc. These logics require little computation resources and depend more on local data. The major client-side business logic is path-finding. This requires a huge structured database storing road data. Some business logic is realised with a third-party solution provider, such as password encryption, which is handled by Firebase Authentication.

2.2.3 Persistence layer

A persistence layer facilitates the process of persisting its state (Richards 2015). Specifically, it stores data into and retrieves data from Google Cloud Database by executing SQL statements. It is realised with Spring JDBC in the program. There is some business logic that bypasses this layer or encapsulates the detail, making it invisible to the developer. One example is the Firebase Authentication module, whose REST API definition and database entries is the only visible component for the application developer.

2.2.4 Database layer

The database layer stores perpetual data in an organised way. The database is implemented with Google Cloud Database and Firebase Real-Time Database. Data includes authentication credentials, crime data, journey information and so on.

Chapter 3

Core implementation tasks

3.1 Signup and login

We decided that Firebase would be the best option for handling users, a user management system operated by Google. There were already packages available to integrate this service with React Native, but these didn't work with Expo which we used to package our app. Instead we used the Node packages made by Firebase themselves. Using a third-party tool for user management meant that we needn't worry about things like database security or encryption as this was already handled to a sufficient extent.

Access to Firebase is included in the Google Cloud Platform that we were already using for server-side code, so it was an obvious choice for user management. Although the service allows for many different types of user authentication (Google account, phone number), we decided to use the classic email-and-password combination.

If the user attempts to sign up with an email that is already in use, then Firebase states so in its response to the request, allowing the app to inform the user. This is similar to the process that occurs if they try to sign in with incorrect credentials. The login and create account procedures are described in pseudocode in appendices E and F respectively.

When a sign up or sign in request is successful, the client stores an object known as the 'current user'. This can be used to verify interactions between the client and the server, such as a user changing their name. A re-authentication token can be used to allow the user to remain logged in upon closing the app and reopening it, although this is not something that we have implemented. To see a sample of how authentication details are stored, see appendix C.

Originally we intended to use phone number as the primary identifiers of

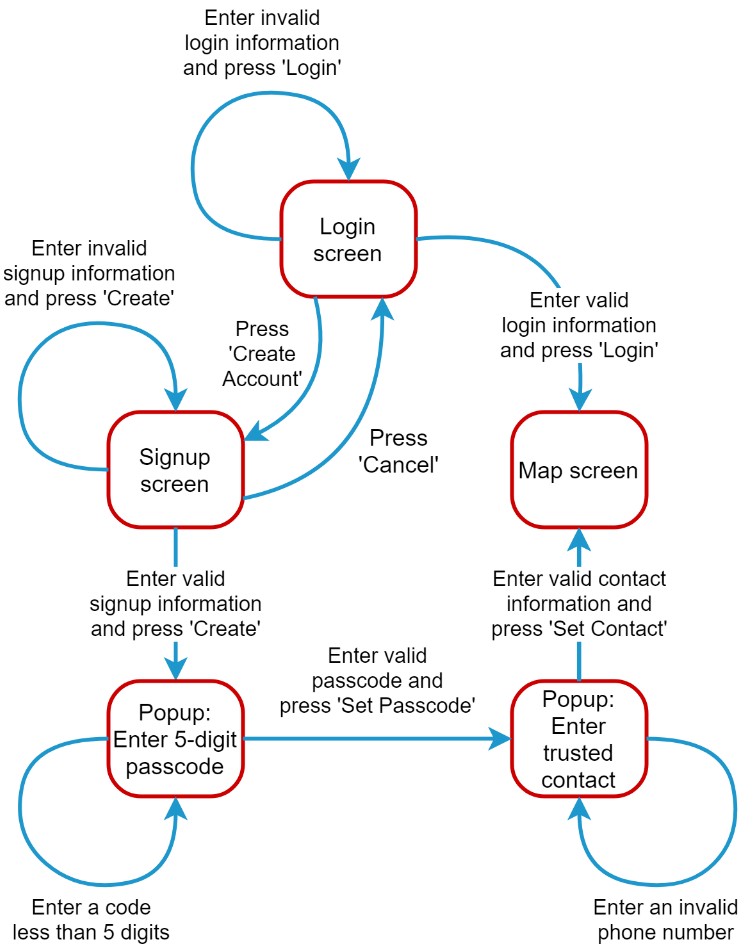

Enter invalid login information and press 'Login'
Login screen
Enter invalid signup information and press 'Create'
Press 'Create Account'
Enter valid login information and press 'Login'
Signup screen
Press 'Cancel'
Map screen
Enter valid signup information and press 'Create'
Enter valid contact information and press 'Set Contact'
Enter valid passcode and press 'Set Passcode'
Popup: Enter 5-digit passcode
Popup: Enter trusted contact
Enter a code less than 5 digits
Enter an invalid phone number

Figure 3.1: STN showing the process of logging in or signing up

accounts. This functionality is available in Firebase, and you can have up to 20k of these (10k in USA, Canada and India, 10k in the rest of the world) for free per billing period (month). However, for testing purposes it was much more practical to use email addresses, which we implemented. The amount of work required to switch this over in the future would be minimal.

Email address verification is ubiquitous and can be set up with Firebase. However, we have not done this at the moment as this app is just a prototype and is not ready for a wider release.

3.2 User data

When the user signs up certain data they have provided (name, address, username) are stored on the Firebase Realtime Database. The way this database works is explained well in the documentation by Google (2020a), but we will summarise here.

This data is stored on the server as branches of a JSON tree, rather than as records in a table. You can see an example of this in appendix D. This is a very easy concept to program with, but may have less efficiency depending on how the tree is organised.

These pieces of data should be stored in the local storage of the phone so that they can be displayed in the app without having to fetch them from the server every time. This would save the user time, and us money on the server as less requests are carried out. However, for this prototype it is sufficient for the app to fetch the data every time.

From the 'My Account' page of the app, the user can change their name, phone number and address. Ideally they would be able to change their email and password here as well. However we did not have time to implement the re-authentication process required for this by Firebase authentication.

In terms of security, the 'current user' of the app must have the unique user ID associated with the data to read or write it. Due to the way Firebase works, this should be secure enough that nobody can access data who shouldn't. However, in future we would change this to an even more secure system that uses expiring tokens (this is built it to Firebase, but slightly harder to implement in React Native).

For a visual description of how data is handled during sign-up and login, see Appendix G and Appendix H respectively.

3.3 Path generation

3.3.1 Navigation

We were able to implement the bi-objective path-finding algorithm described by Galbrun et al. (2014). To reduce the computational complexity of finding all non-dominant paths (i.e. paths that are either shorter or have lesser risk associated with them) we implemented early stopping. We felt it was very important not to overwhelm the user with numerous different paths, so we stuck to three main cases: safest, shortest, and a trade-off route.

3.3.2 General implementation details

These mostly have been described under the algorithm section in our previous report. The main points of interest are:

- All types of crime have equal weighting when calculating the risk of a path. In the previous report we acknowledged that some crimes are more severe than others; however, there are no guidelines on how to categorise and evaluate the change in risk.
- As described in the previous report, open street maps was used to work out the risks on all vertices and edges. This was later filtered to only roads that were 'paths' (walkways) to filter out unnecessary paths which improved the performance of the path-finding algorithm. Lastly, it is displayed to the user using the Google Maps API in React Native. This was done because Google Maps API is the default for Android and there were issues with the iOS maps.
- The data containing computed risks is stored locally due to the time consuming nature of calculating risks of all edges and nodes.

3.3.3 Google Maps API

We have implemented the display of map using Google Maps API. It is also used to auto complete the destination input and fetch a shortest path. Interestingly our shortest path often aligned perfectly with that provided by Google which gives us high confidence that our path finding works as intended.

3.3.4 Dijkstra bi-objective algorithm

As mentioned earlier, stopping was used to reduce the run-time of the algorithm described below. This has been adapted from the paper.

The shortest-route algorithm is changed to A* algorithm in recent iteration cycle. A* achieves better performance by using heuristics to guide the search. By using Euclidean distance as the heuristics function, the number of nodes traversed in one search reduces from over 20,000 to around few hundreds. As Euclidean distance can be approximated in constant time, the complexity of the algorithm reduces from $O(V \log V + E)$ to $O(E)$.

The possibility of using heuristics search for the trade-off path is also investigated. The difficulty is to determine whether the function is admissible this time or not, and the degree of negative impact in such case, which causes the route to be a sub optimal one.

Algorithm 3.1 Early stopping

```
Require: graph $G$ has edge weight $\ell()$ and $r()$
function SUM-REC($P_u$, $P_l$, $\mathscr{H}$)
    $\lambda \leftarrow (r(P_u) - r(P_l))/(\ell(P_u) - \ell(P_l))$
    $\forall e \in E : f(e) = r(e) - \lambda\ell(e)$
    $P_i \leftarrow$ DIJKSTRA($G$,$f()$)
    if $P_i \neq P_u$ and $P_i \neq P_l$ then
        $\mathscr{H} \leftarrow \mathscr{H} \cup \{P_i\}$
        SUM-REC($P_u$, $P_i$, $\mathscr{H}$)
        SUM-REC($P_i$, $P_l$, $\mathscr{H}$)
    end if
end function

function GET-PATHS()
    $P^*_\ell \leftarrow$ DIJKSTRA($G$,$\ell()$)
    $P^*_r \leftarrow$ DIJKSTRA($G$,$r()$)
    $\mathscr{H} \leftarrow \{P^*_\ell, P^*_r\}$
    SUM-REC($P^*_\ell$,$P^*_r$,$\mathscr{H}$)
    return $\mathscr{H}$
end function
```

3.3.5 Deployment

The algorithm is wrapped into a Spring Boot application, which can act as a REST API server. Subsequently, the server is deployed to Google Computer Engine and Google App Engine, i.e., Google's IaaS and PaaS platform. The server is running 24/7 and can be accessed with URL. An example of a GET request can be:

http://35.229.25.93:8080/route?o_lat=51.49&o_lon=-3.1694&d_lat=51.489&d_lon=-3.13543

The parameter o_lat represents latitude of the origin, d_lon represents longitude of the destination. The responded is organised in JSON, containing three routes that are corresponded to the choices mentioned previously.

3.4 Jeopardy detection

Once the user has selected their path, the jeopardy detection features of the application begin. These features are used to mitigate the impact of being attacked. This is split into two subsections:

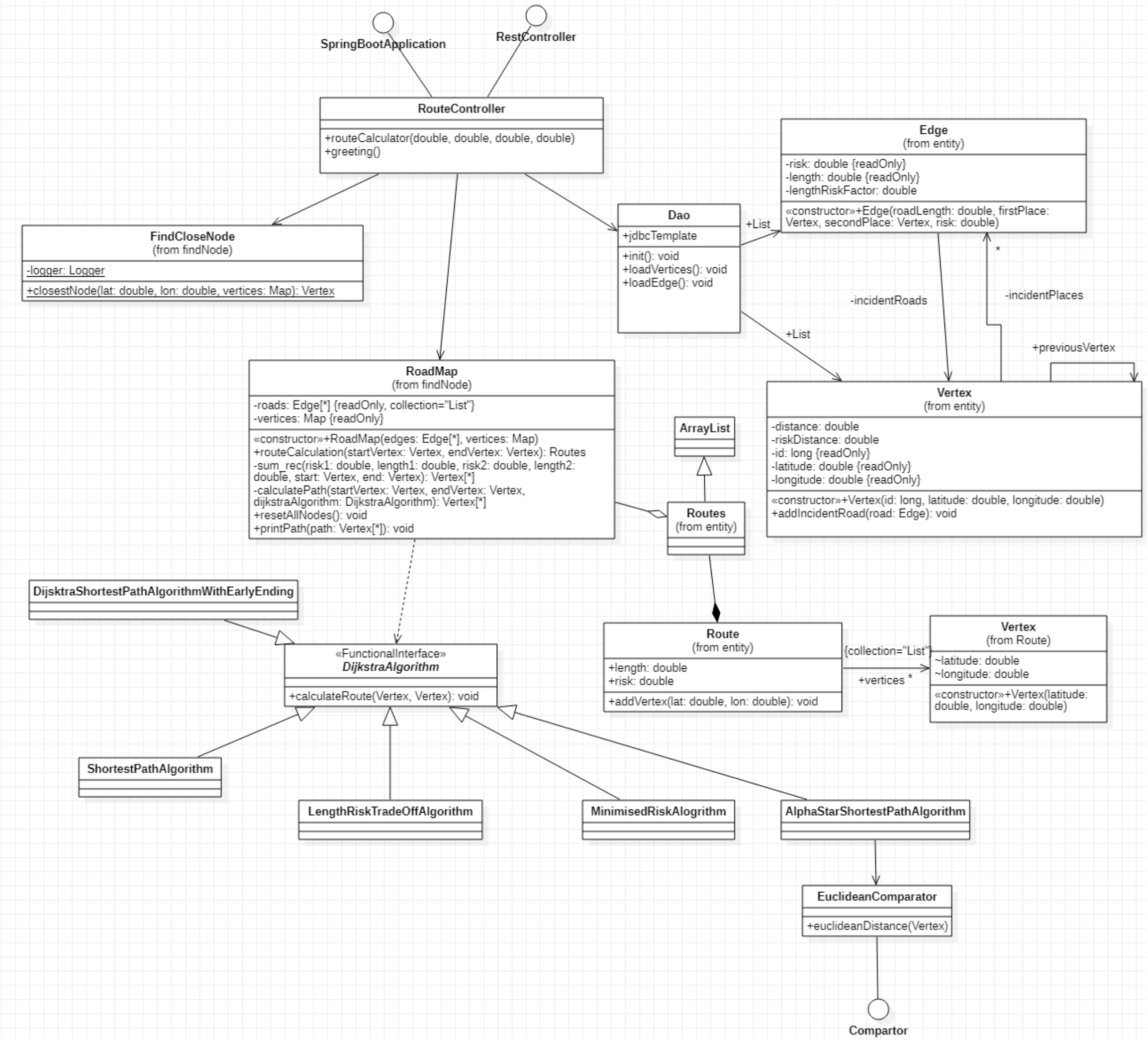

SpringBootApplication
RestController
RouteController
+routeCalculator(double, double, double, double)
+greeting()
FindCloseNode
(from findNode)
-logger: Logger
+closestNode(lat: double, lon: double, vertices: Map): Vertex
Dao
+jdbcTemplate
+init(): void
+loadVertices(): void
+loadEdge(): void
+List
+List
Edge
(from entity)
-risk: double {readOnly}
-length: double {readOnly}
-lengthRiskFactor: double
«constructor»+Edge(roadLength: double, firstPlace: Vertex, secondPlace: Vertex, risk: double)
-incidentRoads
-incidentPlaces
*
+previousVertex
RoadMap
(from findNode)
-roads: Edge[*] {readOnly, collection="List"}
-vertices: Map {readOnly}
«constructor»+RoadMap(edges: Edge[*], vertices: Map)
+routeCalculation(startVertex: Vertex, endVertex: Vertex): Routes
-sum_rec(risk1: double, length1: double, risk2: double, length2: double, start: Vertex, end: Vertex): Vertex[*]
-calculatePath(startVertex: Vertex, endVertex: Vertex, dijkstraAlgorithm: DijkstraAlgorithm): Vertex[*]
+resetAllNodes(): void
+printPath(path: Vertex[*]): void
ArrayList
Routes
(from entity)
Vertex
(from entity)
-distance: double
-riskDistance: double
-id: long {readOnly}
-latitude: double {readOnly}
-longitude: double {readOnly}
«constructor»+Vertex(id: long, latitude: double, longitude: double)
+addIncidentRoad(road: Edge): void
DijsktraShortestPathAlgorithmWithEarlyEnding
«FunctionalInterface»
DijkstraAlgorithm
+calculateRoute(Vertex, Vertex): void
Route
(from entity)
+length: double
+risk: double
+addVertex(lat: double, lon: double): void
{collection="List"}
+vertices *
Vertex
(from Route)
~latitude: double
~longitude: double
«constructor»+Vertex(latitude: double, longitude: double)
ShortestPathAlgorithm
LengthRiskTradeOffAlgorithm
MinimisedRiskAlogrithm
AlphaStarShortestPathAlgorithm
EuclideanComparator
+euclideanDistance(Vertex)
Compartor

Figure 3.2: Class Diagram of the Path-finding server

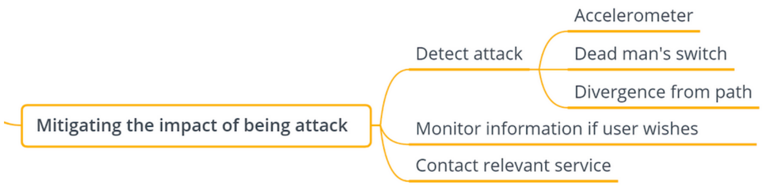


Figure 3.3: Part of approach diagram from Requirements presentation

- Using a variety of reliable methods to detect an attack
- Responding accordingly

Our three detection methods have remained consistent since our earliest design of the system. These can be seen in the diagram below which was part of our requirements presentation.

In order for the app to fulfil its purpose, the detection had to be reliable in order to minimise the number of false negatives. We also tweaked the detection methods to minimise false positives, though these aren't as vital. If the app thinks the user is being attacked when they are not, they can simply cancel the process. However, if the app fails to detect an attack there could be drastic consequences.

We also aimed to make the process require no user input, as otherwise it would be more efficient for the user to dial the phone number manually.

3.4.1 Accelerometer

Expo provides access to the device accelerometer sensors and allows us to listen for and respond to changes in acceleration. Every time the accelerometer detects a change, our function will be called and passed x,y,z keys. These keys represent the acceleration along the particular axis in Gs. Therefore, we can think of this as a vector describing the total acceleration of the device. Since we are not interested in the direction of the movement, we calculate the magnitude of the total acceleration, which is how quickly the speed changes. To calculate the magnitude of the acceleration using the x,y,z acceleration values, we use the following formula.

$$\sqrt{x^2 + y^2 + z^2}$$

If the magnitude of the acceleration is above a certain preset value, the app detects this as a potential attack and begins the response functions (these are described in subsection 3.4.4). The accelerometer detection process is described in algorithm 3.2 on page 14.

Algorithm 3.2 Accelerometer attack detection

```
Require: user is on a journey and their phone has an accelerometer
if there is a change in acceleration then
    magnitude ← CACLULATE-MAGNITUDE()
    if magnitude > threshold then
        begin countdown
    end if
end if
```

3.4.2 Dead man's switch

The simplest feature we have to detect a potential attack is a dead man's switch. This has been implemented in order to give the user more control over the detection process, if our two other functions are unable to assist them. For example, if they believe somebody is following them from a distance making them feel threatened, they may release the dead man's switch and get an instant response from the app. The implementation of the dead man's switch is described in algorithm 3.3.

Algorithm 3.3 Dead man's switch

```
Require: user is on a journey and they have pressed the DMS
while they haven't let go of the DMS do
    nothing
end while
begin countdown
```

Clearly, this is an extremely simple but effective feature that is vital to increase the safety of the user.

3.4.3 Deviation from the path

After calculating the user's desired path, we check how far away they are from it: done using Heron's formula. This calculates the area of a triangle. The three vertices used for the triangle are the two adjacent path vertices, and the user's current location. We then divide by the height of the triangle to get the deviation distance from the path. If the distance is large (more than 200m), then the user is far from the path and we alert their trusted contact.

To calculate the lengths of the edges of the triangle, we used a JavaScript library called GeographicLib. This allowed us to find the distance between two vertices given their latitude and longitude. To keep track of the user's location, every time user's current location is updated, the deviation detection algorithm is called.

Algorithm 3.4 Deviation detection

```
Require: user is on a journey and GPS coordinates are turned on.
deviation ← true
for every vertex i in current path do
    start point ← vertex(i)
    end point ← vertex(i+1)
    a ← DISTANCE(start point, end point)
    b ← DISTANCE(start point, user location)
    c ← DISTANCE(end point, user location)
    s ← $\frac{(a+b+c)}{2}$
    A ← $\sqrt{s \times (s-a) \times (s-b) \times (s-c)}$
    distance ← $\frac{2 \times A}{a}$
    if distance < 200 then
        deviation ← false
        break
    end if
end for
return deviation
```

3.4.4 SOS

Each of our three attack mitigation features responds by running our SOS countdown screen. This locks the user into a countdown which can range from 5 to 15 seconds depending on the user's application settings and can only be cancelled if the user inputs their passcode correctly. During the countdown, the device records audio and video silently in the background, which is then saved to the device. If the user does not enter their passcode correctly before the countdown ends, then they will be prompted to call their saved trusted contact. The original idea was to have this automatically call the police, however there were two problems with this. The first issue was that React Native doesn't allow you to automatically make calls, therefore we had to prompt the user instead. We changed it to a trusted contact as we wanted to eliminate the possibility of any false calls to the police during testing. The countdown process is described in algorithm 3.5 on page 16.

Algorithm 3.5 Countdown screen

```
Require: one of the three attack mitigation features have been activated
begin recording
while time remaining $> 0$ do
    ask user for passcode
    if user enters correct passcode then
        end countdown
    end if
    decrement time remaining by 1 second
end while
call trusted contact
```

Chapter 4

Testing

There are two elements of our system which required testing: the server and client. In our interim report, we outlined two main methods of testing that we would be using for our application: unit tests with the Jest framework, and test cases. We have carried out all of the test cases that were included in that report, and you can see the results in section 4.2.

We did not end up using the Jest framework, as we had originally planned, because there was not enough time for us to learn React Native, implement all the functions, and learn an additional framework just for testing purposes. This is unfortunate as it would have been beneficial for the app if these testing measures were applied. Were we to continue working on the app, it is definitely something that we would do.

In our previous report, we didn't adequately discuss user testing. Our team has since gained substantial experience with this, due to a separate coursework for *CM2101: Human Computer Interaction*; it is evident that this could lead to massive improvements to our application. Unfortunately, since user testing is laborious and time-intensive, we didn't have enough time to conduct it. Additionally, the rules under the pandemic made it that much more impossible.

4.1 Automatic testing

Automatic testing is deployed for the server and crime data allocator. The automatic testing tests whether the programming code can generate the correct output by comparing it with expected results.

For mimicking the web layer, Spring's `MockMvc` package is used, which handles the incoming HTTP request and process the traffic in the identical way as it is processing a real HTTP request but without the overhead of starting a server (Syer et al. 2020).

JUnit framework is used to support writing and running these repeatable tests. Its assertion library is heavily used to verify the correctness of the data. Another feature it provided is testing whether an expected exception is thrown, which is unable to test if using vanilla `System.out.println()` and if-else statement.

By using automatic testing, the costs spend on testing in each iteration cycle decreased. The ease of writing testing also promotes motivation for test-first programming, which ensure the implementation of the system meets the original intended use. Moreover, it prompts the development team to write less coupling code, facilitating the difficulty for writing a test case for each function as it becomes a relatively independent part.

The code coverage reaches 95.2% for the latest release. A test report can be found in the `code-coverage` folder. Although the figure seems promising, it is worth to pinpoint that most test cases do not consider decision coverage or decision-condition coverage, since the number of the test cases will grow exponentially when using stronger coverage method (Qian et al. 2016). Nevertheless, the tester endeavoured to follow the principle of boundary-value testing as close as possible by analysing equivalence class and valid scope.

4.2 Test cases

Test case	**Result**	**Explanation**
1	5/6	The test case failed at the last step as SMS verification was not implemented
2	2/3	The test case failed as logging in was implemented with email instead of phone number
3	3/4	The test case failed as the standard SatNav style display was not implemented
4	0/1	The test case failed as the user is not informed that they have completed the journey
5	2/3	The test case failed as the user must walk 200m to raise the alarm rather than 25m
6	2/3	The test case failed as the pause journey function does not ask the user for a password
7	0/2	The test case failed as the cancel journey function does not ask the user for a password

Table 4.1: Results of the test cases

The test case results (table 4.1) show that the application we have produced meets most of the functional requirements we set out to achieve, but that the design of our app is significantly different to what we had first imagined. Ideally, we would have written new test cases (or rewritten the ones already existing) to reflect the new design of our app, however we unfortunately did not have enough time to do this. The full results of the test cases can be seen in appendix I.

Chapter 5

Justification and evaluation

5.1 Framework

5.1.1 React Native and Expo

We chose React Native as our framework for the main application during the early stages of planning the project. Our very first idea was to simply develop a prototype desktop application using Python or Java. However, we made a u-turn on this idea after some discussion and decided that it would be more beneficial to learn mobile development using a framework used in the industry in order to get the most out of this project. After some research we discovered the React Native offered all of the features we thought we'd need for the project such as location and access to the camera. Though the main thing that attracted us to the framework was that we can write code once and have it work on Android or iOS straight away because it abstracts away all the necessary conversions to the native code for each platforms.

We used React Native with Expo which helped us develop the application much more efficiently. Expo is a third-party service which offers a 'Managed App Development' workflow and simplifies the React Native development process by acting as 'Syntactic Sugar'.

One of the many features it gives is is a live development environment which allows us to run the code on real devices instantly. In a bare React Native app this can be quite the chore to achieve and makes testing substantially more cumbersome. However, with Expo we just scan the QR code and the app loads on any mobile device. This also allowed us to instantly run the application on Android emulators which was convenient for testing on various screen sizes.

Expo was particularly helpful for our requirements as we were merely developing a prototype. It abstracts away all of the extra modifications that have to be made, such as configuring packages to run on Android or iOS and does most of the heavy lifting for us. Expo also has hot reloading feature which saves time when developing, however, this was fairly buggy and sometimes the entire local development server had to be restarted a few times to fix it.

The downside of using Expo is we are limited to the Expo ecosystem as it is essentially a wrapper around the app, and therefore any packages we used must also be compatible with expo. This added some unnecessary time to the development, particularly in the early stages, causing us to spend time implementing a package only to discover it wasn't compatible. Expo removes some of the complexity of the raw native code but in doing so also removes some of the fine-grain control we would get if we opted to use a bare workflow. However, for our purposes Expo was certainly the correct choice as we needed to use many native device features such as the camera, microphone, accelerometer and location. In a bare workflow we would have to do a substantial amount of extra configuring to get this to work which would take up valuable time. Time was our main issue with this project so we couldn't afford any extra overhead.

One final benefit of Expo is it allows us to 'Eject' from the managed workflow, and will automatically convert the entire project to a bare React Native app, with only some minor configuration required to get it working. Were we to continue working on this project in the future, a bare development environment would be a good choice as we could get more precise control over the application and would also be able to use some packages that are not compatible with Expo.

5.1.2 Spring

Spring is a framework for creating Java enterprise applications with features like Inversion of Control and Dependency Injection. As mentioned in the previous section, a REST API service is set up and run, with the support of the Spring framework. The development team decided to use Spring initially due to it is a de facto highly popular framework for building web server. During the programming experience with Spring, we concluded that there are several major benefits for adopting Spring:

1. **Integration with Apache Tomcat**
   When running the program, Spring Boot will detect the existence of a Spring MVC controller and start an embedded Apache Tomcat 7 instance, by default. (Long 2014). As the default configuration run impeccably for our program, this auto wrapping saves a considerable amount of effort.

2. **Integration with JDBC**
   We used `JdbcTemplate`, which is the central class in the JDBC core package. Spring open the connection, prepare and execute the statement, process and exception and close the connection. This avoids try-catch-with-resource boilerplate (Spring 2020).

3. **Powerful Annotation**
   The annotation provides the ability to define behaviours without writing XML configuration.

5.1.3 Maven

Maven is used as a software project management tool. Specifically speaking, the following tools are used in the Maven project lifecycle:

1. **Dependencies management**
   All the dependent libraries are defined in its project object model (POM). It eases the development by including transitive dependencies and allowing the inheritance from the parent POM (Apache Maven Foundation 2020), which is especially critical for using a complicated framework like Spring.

2. **Packing with automatic testing**
   Maven can build a JAR file by one-line command, after finding and running JUnit testing automatically. Moreover, it facilitates version management as the version number defined in the POM can be embedded into JAR without additional setting. The package JAR is convenient to be deployed into Google Compute Engine.

3. **Distribution to Google App Engine**
   After installing the App Engine Maven Plugin, the project can be deployed to the PaaS platform in a straight-forward way. Compared with other project management solution, namely Gradle, maven is favoured in our project as its grammar is more self-explained, requiring no extra learning cost. Although Gradle claims that it is faster in terms of compiling and building (Gradle Inc. 2020), the project is relatively uncomplicated which can be built by Maven in an acceptable time, typically less than 1 minutes with testing.

5.2 Cloud Service

We choose Google Cloud Service from public cloud service providers because we have used the API from google maps initially, thus the billing information have been set up. Google Cloud Service runs the server with no downtime, which contributes to the fulfilment of our system requirements. Although it is unpractical to evaluate it insightfully by comparing with other public cloud service provider due to the lack of experience, its generous free tier is a distinct advantage, providing 1 IaaS virtual machine and 28 hours of SaaS service for free per day from public cloud service providers (Google 2020[b]).

5.3 Path-finding algorithm

A key limitation of our algorithm stems from the paper on which it was based; it calculates the potential risk of a route based on the number of total crimes, as opposed to a ratio of total crimes to people using the route.

Essentially, this means that a less frequented route will always show up as a "safer" route. However, it may not necessarily be safer: for example, imagine ten people are using main street A, and one person is using side route B. If three people using street A get robbed, our algorithm will show that as a riskier route, even if the one person using route B gets mugged too. Despite this, it is evident that a random person would have a 30% chance of getting robbed on main route A, and a 100% chance of getting robbed on side route B.

Due to the simplistic nature of the above reasoning, it is possible that the app's suggestions might clash with the user's common sense. We conducted an informal survey of a group of 5-10 people falling in the app's target demographic (women, members of the LGBTQ+ community, minorities) all unanimously agreed that they would rather take the shorter, well-lit, main route A, than the dimly-lit and considerably longer side route B.

The consensus was that regardless of what the app said, they preferred a route with greater CCTV surveillance and greater overall footfall, to act as witness should the worst come to pass. They felt should someone start following them at any point, a long, dark, route composed of alleys would be akin to shooting oneself in the foot.

5.4 Evaluating design strategies

In retrospect, there are several ways our design strategy could be improved. If we were to repeat this project, we would have taken a user-centred design

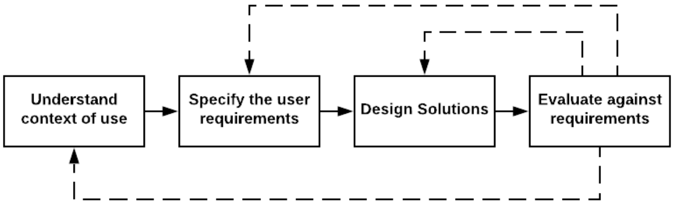

Understand context of use
Specify the user requirements
Design Solutions
Evaluate against requirements

Figure 5.1: The stages of a user-centred design approach, adapted from work by Interface Design Foundation (2020)

approach. This approach would focus more on building an understanding of what people would use the app, allowing us to identify the requirements from there.

The iterative process would assist us in the design, development and testing phases. Figure 5.1 on page 24 has been made to show how a user-centred approach would benefit each stage of the project.

The only reason we did not to use this design strategy was because we did not know how to use it properly, at the time of designing the project. We began our module *CM2101: Human Computer Interaction* at the beginning of the spring semester. In this module, we learnt an abundance of information about designing, testing and evaluating software - including the principles of user-centred design. Despite regretting not using this strategy, we recognise this as an opportunity to see the weaknesses in this method.

Instead, we designed the software as close as possible to the client's requests. We began by drafting requirements from our interpretation of the brief, then further discussed with our client about what they expect to see included and what may not be necessary. After, we confirmed our design and our development plan to the client in our interim report and requirements presentation. We tried to communicate consistently with the client - but we never thought to formally document how it would be designed around the user. A user-centred approach would begin by building personas for members of the target audience, from there every step of process will refer to the personas to see whether they benefit the personas.

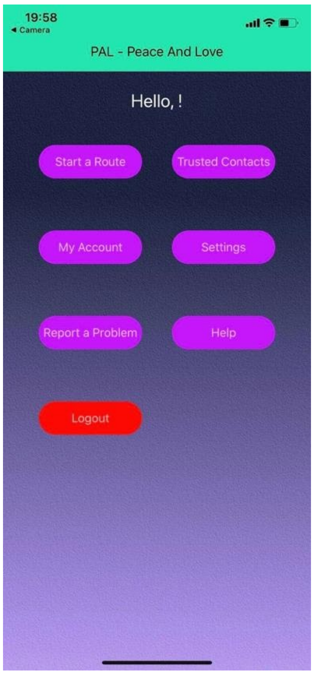


Figure 5.2: Prototype menu from the Interim Report

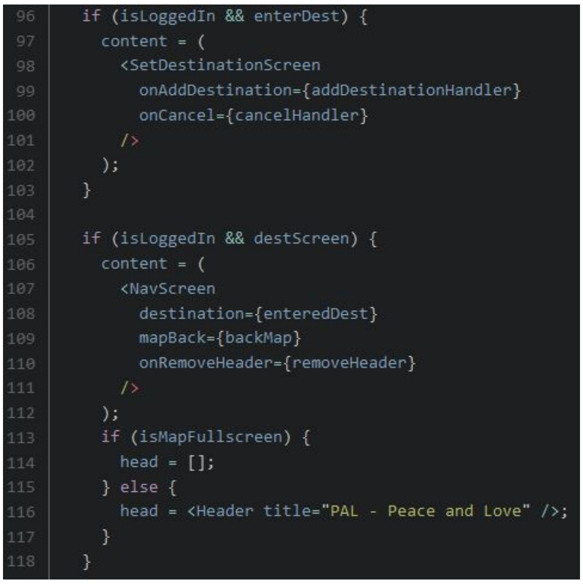
5.5 Changes to UI from prototype

5.5.1 Menu

The main menu went through the most immediately noticeable changes during development. The original design (see figure 5.2) had very little thought put into it and was mostly a result of insubstantial React Native knowledge.

The priority with this version was simply to have it working with little regards to appearance. Our inadequate knowledge of React Native at this point that we switched from screen to screen using a series of Booleans and complicated if statements since we hadn't yet been introduced to the React Navigation libraries (see figure 5.3).

After the interim report, we continued experimenting with different layouts, eventually adding icons for each button. However, we could never find a design which fully satisfied us, additionally it was rather difficult to make all the buttons fit on the screen and look good across different platforms and screen sizes. During a client meeting in February, one of the main points raised was that we should consider launching straight into the main function of the application and to limit the total number of clicks it takes to achieve any specific action. We followed this advice and scrapped the menu screen alto-



```
if (isLoggedIn && enterDest) {
  content = (
    <SetDestinationScreen
      onAddDestination={addDestinationHandler}
      onCancel={cancelHandler}
    />
  );
}

if (isLoggedIn && destScreen) {
  content = (
    <NavScreen
      destination={enteredDest}
      mapBack={backMap}
      onRemoveHeader={removeHeader}
    />
  );
  if (isMapFullscreen) {
    head = [];
  } else {
    head = <Header title="PAL - Peace and Love" />;
  }
}
```

Figure 5.3: Original navigation code snippet

gether, the application now launched into the map screen. Our first idea was to use a tab navigator system (see figure 5.4). This meant having a tab bar permanently affixed to the bottom of the screen with a separate icon for each screen. After some consideration, we decided against this approach mainly for one reason, it took up too much screen real estate. This was particularly apparent on the journey screen as the dead man's switch is also permanently affixed to the bottom of the screen. When testing on a Nexus One, which has a 480×800 screen the tab bar combined with the dead man's switch obscured the majority of the map. Our second reason for abandoning this design was that we wanted to minimise the number of false negatives. If the user meant to release the dead man's switch but instead click one of the tabs and navigated to a different screen by mistake. These two reasons made the tab navigator an unacceptable design choice for our application.

We finally settled with a side drawer navigation system (see figure 5.5). This was a much better implementation fella as when it wasn't in use it didn't take up any screen real estate and was also more expansive than the tab system. When developing we added many additional screens, which would have caused the tab bar to become overcrowded and confusing, this wasn't an issue with the side drawer. This was accessible via a 'hamburger' button on the map screens. Upon navigating to any subsequent screen, the 'hamburger' button was replaced with a back button to the map screen, we believed that navigating back to the root screen was a good design choice as it is the main

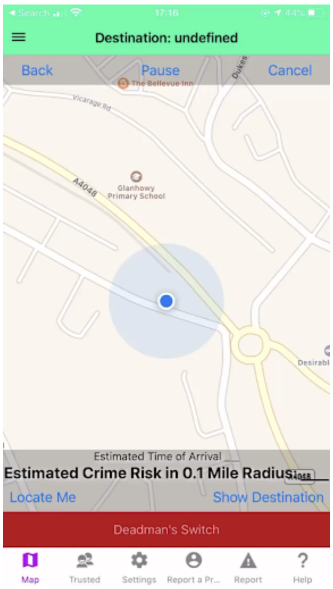


Figure 5.4: Tab menu

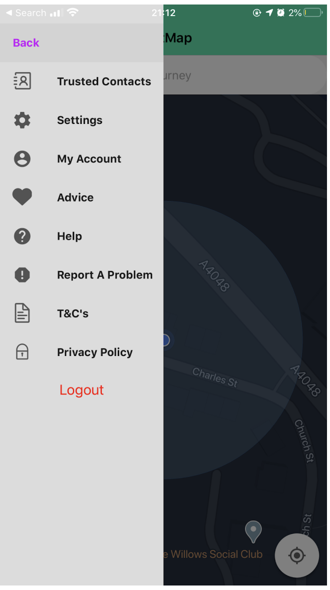


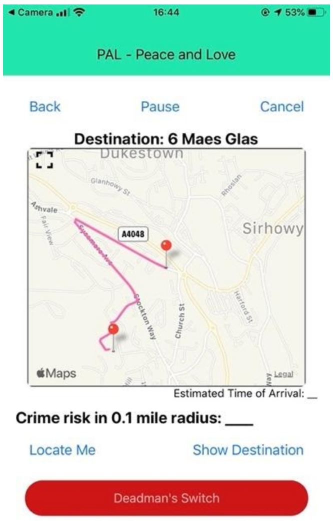
Figure 5.5: Sidedrawer menu

function of the application and is very likely to be the screen the user would want to go to next.

5.5.2 Journey screen

The Journey screen is the most important screen of the whole application as it is where all of the main functions are active simultaneously. Therefore, particular attention was payed to the user interface to make it is clear as possible. Once again, there were tremendous changes from the interim report prototype. Our original design was flawed in many areas as the only goal was to display a map on the screen (see figure 5.6). Our lack of React Native knowledge hindered us from achieving any form of presentable design but we were proud that we had a working map. It is clear where the flaw is with this design, the map itself is in a tiny box on the screen meaning a minute amount of visibility is offered. In a desperate attempt to remedy this, we added a fullscreen button which applied a different set of style rules to the map to try and make it full screen. This wasn't very successful either as it overwrote everything below it and also removed the screen header (see figure 5.7).

Once development on the project resumed in January, the map screen received a much needed overhaul. After spending a lot of time learning more about React Native we managed to fully implement a working map in full screen with buttons overlayed which looked good across all devices

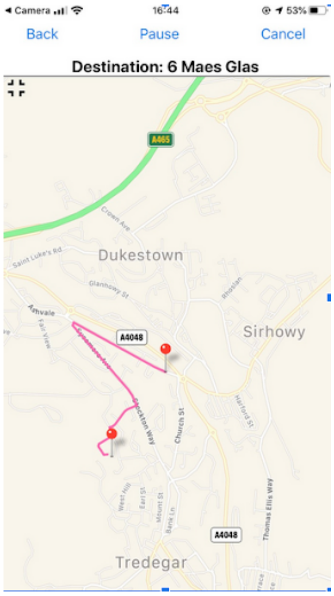


Destination: 6 Maes Glas

Estimated Time of Arrival: _

Crime risk in 0.1 mile radius: ___

Figure 5.6: Prototype map



Destination: 6 Maes Glas

Figure 5.7: Prototype map 'Fullscreen'

and screen sizes (see figure 5.8). We were happy with this as a final design as it looked clean and easy to use whilst given the user as much of the map to interact with as possible.

However, one final change to this UI was required when we discovered we could use a built in 'Locate Me' button with google maps. The use of this meant we could remove many lines of our code which did the same thing and replace it with one single line. However, we had no control over the design or location of the button so the rest of the interface had to be designed around it. On iOS the button was permanently on the bottom of the screen, just above the dead man's switch. Therefore all the buttons and labels had to be moved to the top of the screen to avoid obscuring it (see figure 5.9).

We were once again happy with this design until we discovered that on Android the button is at the top of the screen. We implemented a check for the device operating system within the style sheet and rendered different rules for android and iOS. For android all the content now had to be at the bottom of the screen (see figure 5.10). This wasn't an ideal scenario as we liked having the dead man's switch free from clutter but for a prototype design we were happy.

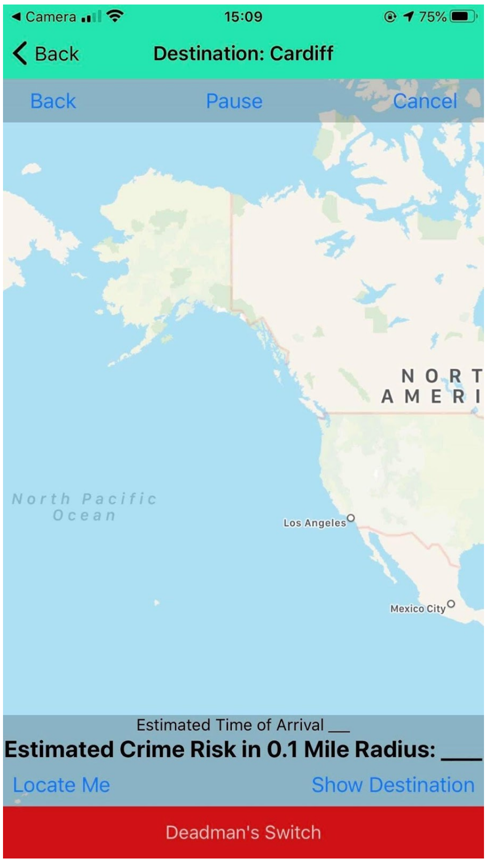


Figure 5.8: Post-prototype refinement of map screen

5.5.3 Create an account

The original Create an Account screen (see figure 5.11) was originally constructed by the UI team to get a functional form working to present to the user when they had opened the screen. The screen consisted of textual information guiding the user through the form, with a ‘Create’ and ‘Cancel’ button at the bottom of the form.

Once the core functionality of the application was developed, the team had started to think about the Terms and Conditions of the application and how the user should be informed of the applications Privacy Policy. The development team had come to an agreement that the user should have the opportunity to be informed of this before they create an account. This had led to the team changing the textual information presented to user by creating clickable text buttons within the information given to allow the user the view the Privacy Policy and Terms and Conditions if they wish (see figure 5.12).

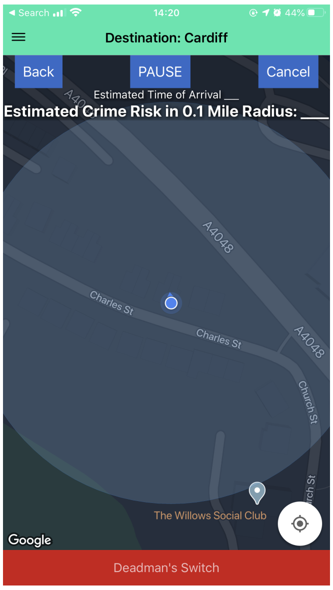


Figure 5.9: Final Map (iOS)

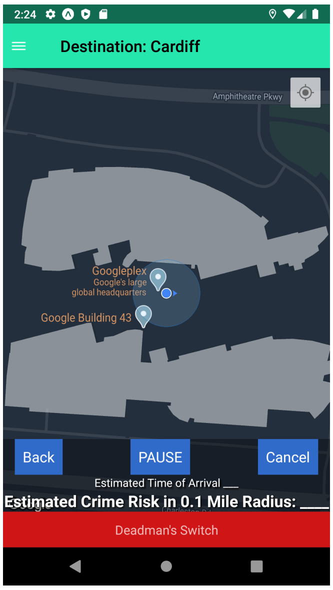

2:24
Destination: Cardiff
Amphitheatre Pkwy
Googleplex
Google's large global headquarters
Google Building 43
Back
PAUSE
Cancel
Estimated Time of Arrival __
Estimated Crime Risk in 0.1 Mile Radius: ___
Deadman's Switch

Figure 5.10: Final Map (Android)

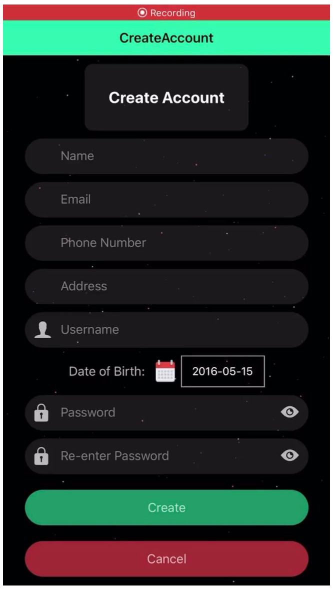


Figure 5.11: Old create account

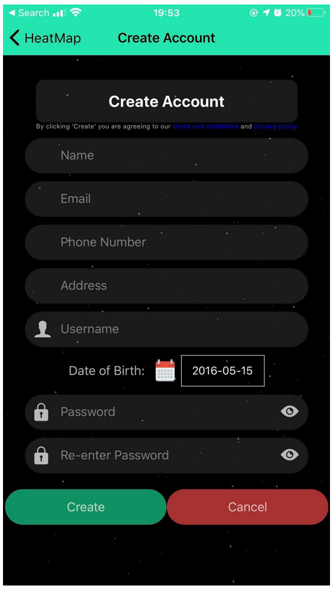
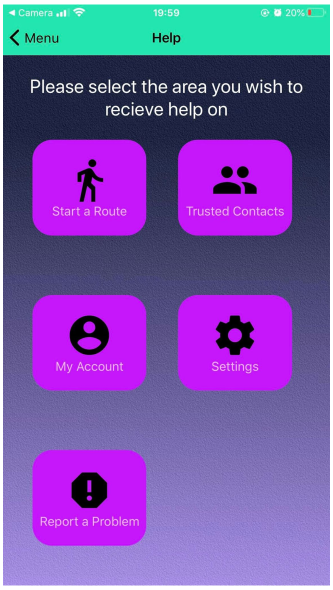
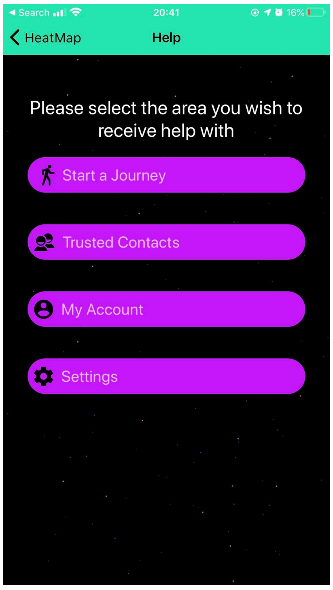


Figure 5.12: New create account



Please select the area you wish to recieve help on

Figure 5.13: Old help menu



Please select the area you wish to receive help with

Figure 5.14: New help menu

5.5.4 Help

The help screen had been designed in way in which it would present the user with several buttons that they could click to get help on a specific screen within the application. Once the user clicked the chosen button, the application would direct them to a screen which would provide textual advice, in a numbered list form to guide them through using that part of the applications functionality.

Originally the main help screen (that included the buttons to click on a specific section) was designed in a way which filled up the users phone screen with square icons that represent each section (see figure 5.13). However, after some internal reviews, this was changed to more a vertical list structure for compatibility with older devices which tend to feature smaller screens. (see figure 5.14)

5.5.5 Colour scheme

The original scheme had featured a shaded purple, with pink buttons. This was an unplanned colour scheme that was chosen to demonstrate clearly to the developers where different components was on the screen.

After some careful consideration, the original colour scheme followed the popular 'night shade', where the main background is black. PAL featured a mimic of stars within the background to represent the night time sky. The text fields were changed to a dark shade of grey so that they were distinguishable from the background, but also blended into the night theme. The decision to make this the default theme was a result of our target audience (usually) using the application at night time when walking alone.

5.6 Team work

As mentioned in the introduction, we decided it is appropriate to evaluate our team performance to develop an understanding of what may have worked effectively and what may have hindered our performance. We have broken this into several sections which have large contributing factors in our experience of collaborating.

5.6.1 Distribution of tasks and responsibilities

As a team, we decided the best approach to developing the prototype would be to split into different sub-teams. These teams were chosen democratically, based upon who we feel would achieve the best results. Although team members were assigned to work on a particular part of the prototype, people did not work exclusively on that task. This strategy ensured there was an expert within the team for each component of the project. Everyone was aware of who they should go to in order to get more information about something. This also led to consistency across the design, development and evaluation phases of each component. We believe this allowed us to have a detailed understanding of how each component works, which in hand, has allowed us to provide an exhaustive explanation of the prototype's functionality.

5.6.2 Agile

We continued to use Agile/Scrum throughout the remainder of the project, this was mainly because it had served us well during the first part of the project and encouraged us to communicate progress and problems consistently. It would be unfair for us to argue that we know for certain that Agile was the best method for us to use, even so, it seemed to work perfectly well. The main change we would encourage would be changing the duration of sprints more promptly according to the tasks that needed to be completed. Another thing pointed out was that we would greatly benefit from having a Scrum Master who was not part of a development sub-team. Overall, the team agreed that we would benefit from increasing the structure of the way we did agile. This was implied in the consistency of meetings and documentation. As you will see in the next section, we began experiencing difficulties when trying to use Agile and Scrum through remote collaboration.

In their guide to being an effective Scrum Master, Scrum.org (2020) suggests they have. From the experience of attempting to be a member of a development team and a scrum master simultaneously, it is very difficult to fulfil both of these roles.

5.6.3 Remote collaboration

Due to the COVID-19 pandemic, Cardiff University's School of Computer Science and Informatics phased out all face-to-face teaching on the 16th of March 2020. From here on, we were expected to continue to work together on the project remotely. We believe this is responsible for hindering the development phase of the project. We kept in touch consistently and strove to keep our levels of productivity high, as well as the spirits within the team. We agreed to continue to meet three times a week (Mondays, Wednesdays and Fridays) using Microsoft Teams, however due to some unavoidable circumstances, one of our team members was unable to attend due to an unreliable internet connection at home. Along with our regular meetings we tried to continue with our Agile/Scrum method, this proved to be more difficult than when our meetings were face-to-face, this could be partly because of how much easier it was to perform good time-management practices before lock down was imposed. This may also be further encouraged because at this point we were less focused on adding new features, but rather making changes and bug fixes in previous work.

5.6.4 Tuckman's *Stages of Group Development*

At the end of the project we conducted a meeting to see what the team-members thought of our performance throughout the project. All present team-members agreed that we did follow the trend of Tuckman's Stages of Group Development (Tuckman 1965). To prevent productivity rates flattening, we conducted a re-briefing at the beginning of Semester 2. The main objective of this was to ensure a smooth transition from the planning and design phases to the development phase of the project.

5.7 Legal, ethical, social and professional issues

In our interim report and requirements presentation, we outlined the range of issues that we should be aware of before developing the application. Since there have been minimal changes to the amount of user information our prototype uses and since there have only been features removed (rather than new ones put in place) from the design to development phase, we feel it is not necessary to repeat the aforementioned issues. Please refer to our interim report and requirements presentation to familiarise yourself with the them.

5.7.1 Accessible privacy policy

Under the General Data Protection Regulations (GDPR), a key transparency requirement states that individuals have the right to be informed about the collection and use of their personal data. However, companies often use a lengthy privacy policy to technically meet the requirement of informing their users; they know not many have the time or patience to work through the text and hence, users are susceptible to using apps without actually understanding what data the app collects or how it's used.

A privacy policy usually tends to be incomprehensibly long and chock-full of legal jargon, often baffling the everyday user. Thus, we decided to provide users with an accessible privacy policy as a good faith practice. For users to use our suggested paths, and consequently our app, it is essential to demonstrate that they can trust the app to work in their best interest. As such, we wrote an accessible privacy policy adhering strictly to the GDPR's guidance outlined below.

According to the GDPR, a privacy policy must be:

- In a concise, transparent, intelligible, and easily accessible form
- Written in clear and plain language
- Delivered in a timely manner
- Provided free of charge

The user is provided with a link to view this privacy policy at the time of creating an account. They can also view it once logged in, via the Menu hamburger button. It includes the data we collect, how the data is collected, used, and stored, and attempts to detail them in layman's terms. It also explains our cookies policy, and whether their data would be utilised for marketing purposes. Finally, it provides different channels to contact us via, should the need arise. At only a page-long, our privacy policy is succinct without sacrificing any information the user might need to know, in regards to their data.

5.7.2 Advice to users

While our app uses local crime data in order to recommend the safest routes, we have decided to provide the users with some general tips to improve their overall safety. These include carrying a pepper spray, a personal safety alarm, and checking in with someone frequently (further detailed in appendix J).

Our intent is to operate from a 'prevention is better than cure' mindset; we wish to empower the user to educate themselves and reduce the risk. There are also links at the bottom of the page, should the user wish to learn further.

We believe that improving the personal safety exercised by users could help supplement the app's function greatly, as well as imbue greater confidence in our users, and allow them to stroll the nights unencumbered.

5.8 Feasibility and business case

Within the team, there have been discussions about the possibilities this prototype presents to us. We have figured that we have several options:

1. Team member(s) choose to take on the project (or an element of the project) as a final year project.

2. Prepare to release the application and all documentation as an open source project.

3. We pass the project to other developers to further develop the prototype or build a new prototype based on the documentation we have provided.

The options above have been listed in the order of which we are prioritising, meaning, that we would only consider option 2 or 3 if there were no team members who wanted to take this project on for their final year. We have considered the pros and cons for each of the options and we have come to the conclusion that we should not choose an option unless all team members agree unanimously on a particular option. This is out of respect for all people's contributions to the project, as the entire process has been a team effort. For now, we have not finalised our decision but we are glad we have considered the options available to us, and as soon as we have a confirmation that no team member(s) want to continue the project in final year, we will begin preparing to make the project open source.

Our primary hesitations with making the project open source is that we believe there are some API calls that may need changing - we also experienced many issues installing/initialising Expo, React Native and the other libraries on our computers so we would also like to provide installation instructions among this. Quite frankly, we have decided that we will focus on this once the semester is finished because we do not need the added stresses of completing this at the same time as our final prototype, presentation, report as well as sitting our exams.

Chapter 6

Future work

6.1 Features that were not included

6.1.1 Grouping

We did consider implementing a grouping feature in the prototype, but decided decided to give it a low priority due to the following concerns, which were further compounded by the fact that it would be inconsiderate to implement a feature that contradicted the current guidelines involving concerning social distancing.

- It is difficult to evaluate how much safety is provided in a group. This feature would require us to weigh how much safety is provided by walking in a group with strangers and how much less safe they would be because someone might use our app maliciously. We were unable to find any studies regarding safety in groups.

- To minimise chances of our app being used maliciously, background checking or at least phone number checking would need to be added. While we did find a way to implement the use of a phone number as verification, we did not feel confident that this would be sufficient to prevent abuse.

- It is unclear how we would generate a combined path for the users. Ideally, we would take into account the desired safety options of both users, but we would also need to consider the size of the group and how much more time their route would be expected to take because of grouping.

We believe, however, that this feature would provide a lot of value if it was implemented correctly. Therefore, we did come up with a potential implementation for the prototype, but more evaluation would be needed on how viable it is.

Algorithm 6.1 Grouping algorithm

```
Require: user is logged in
ask the user for their destination and the time they wish to leave
calculate and store the optimal route
compare the route against those of other users
if the paths of three or more users intersect then
    ask all users if they would like to be grouped
    if at least three users accept then
        direct group members to join the journey at the correct time
    end if
end if
```

6.1.2 Heat-mapping

Unfortunately, we were unable to implement the heatmap. We had originally set out to build a fully-automated system, but due to time constraints this was not possible. After realising that automation would not be possible, we then decided to try implementing a static version instead. Example heatmaps can be seen in figures 6.1 and 6.2.

We find it very unfortunate that this feature was not able to be implemented as we know it would have been a useful feature for the users, and we had successfully built most of what had to be made. In the end, issues with including a dynamic/interactive heat-map within Expo and React Native applications was the main problem. We had developed a parser for the

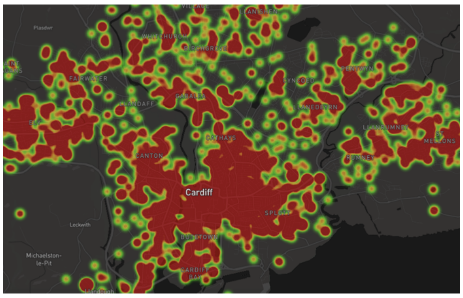


Figure 6.1: Heatmap zoomed-out

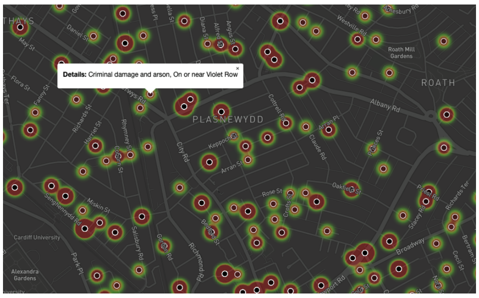


Figure 6.2: Heatmap zoomed-in

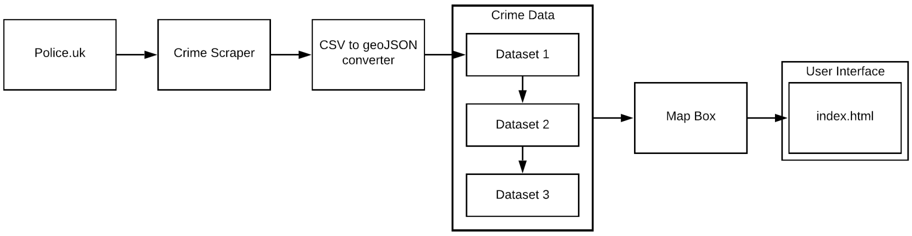


Figure 6.3: Proposed solution for an automated heatmap generator

Police.uk website, a scraper to retrieve the necessary files and a file converter to put these files into the relevant file types (from csv to geojson files).

After establishing a virtual server, 'index.html' displays the heatmap in browser, using the geojson files converted by 'csv2geojson.py'. In addition to exhibiting a heatmap as shown in Figure 6.1, the user can zoom in further to see the independent locations of where the crime occurred. Additionally, if the user clicks on any of these points, they are able to see information regarding the exact location of the crime, plus the type of crime (as illustrated in figure Figure 6.2).

The only real thing that held us back was integrating the HTML formatted heatmap, which was generated using MapBox API into the Expo application. The pseudocode in algorithm 6.2 on page 39 is a simple example of how we ensure a heatmap is generated in an appropriate way and a diagram of how this automation would be carried out has been provided below.

Algorithm 6.2 Generating heat-map based on crime data

```
for crime in crime data do
    age ← today's date − date of crime
    if age < 1 month then
        intensity ← 1
    else if age < 2 months then
        intensity ← 0.5
    else
        intensity ← 0.25
    end if
    ADD-TO-HEATMAP(crime location, intensity)
end for
```

In the future we would also like to implement this within the application to allow the user to visualise and explore the crime data near them to help them make their own decision on which route to take, so that they are less reliant on our route recommendations.

6.1.3 Path-finding algorithm

The current implementation loads full database at once. When the geographical domain scales to the entire UK in the future, the algorithm should dynamically load the corresponding neighbourhood, instead of reading the entire road map into memory.

6.2 Industrial level product

6.2.1 Logging

Loggers, namely slf4j framework are used to retrace bugs after the server is deployed to the production environment. Nevertheless, there is a deficiency of logging in the front-end, due to the time constraint and code style discrepancy. The system should send the log when the user utilises the contact us or report problems feature.

6.2.2 Security

The server is vulnerable to the Denial-of-Service attack as no firewall rules is configured. If the application is officially published, the traffic should be closely monitored and suspected traffic should be disallowed within a short time using Google Cloud Armor.

Another security issue is that some API key is hardcoded into the application itself. This may cause credential leakage if hackers decompile our code. A surrogated server is the solution to this problem.

6.2.3 Comparison with other similar applications

Ultimately, the crucial question is how our app compares to other similar apps on the market. There exist a surprising number of apps providing the same features in some capacity; their functionalities and how they compare to our prototype is discussed in appendix K. It is important to note that none of the 'top' apps seems to offer suggested safer routes, a grouping functionality, or a heatmap capability. There also exist simpler apps that compare to a single feature within our app: for example, 'Watch Over Me' activates an alarm, video recording, and alerts the emergency contacts simply by shaking the phone. This is similar to how our accelerometer feature functions. Finally, while Road Buddy does suggest alternative routes, it seems the app never materialized.

References

Apache Maven Foundation. 2020. *Introduction to the Dependency Mechanism*. Available at: https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism [Accessed: 12 May 2020].

Galbrun, E., Pelechrinis, K. and Terzi, E. 2014. Safe Navigation in Urban Environments. 3rd International Workshop on Urban Computing at KDD 2014, UrbComp'14. Available at: https://hal.archives-ouvertes.fr/hal-01399249.

Google. 2020. *Firebase Realtime Database*. Available at: https://firebase.google.com/docs/database [Accessed: 29 April 2020].

Google. *GCP Free Tier - Free extended trails and always free*. Available at: https://cloud.google.com/free [Accessed: 12 May 2020].

Gradle Inc. *Gradle vs Maven Comparison*. Available at: https://gradle.org/maven-vs-gradle/ [Accessed: 11 May 2020].

Interface Design Foundation. 2020. *User Centered Design*. Available at: https://www.interaction-design.org/literature/topics/user-centered-design [Accessed: 12 May 2020].

Long, J. 2014. *Deploying Spring Boot Applications*. Available at: https://spring.io/blog/2014/03/07/deploying-spring-boot-applications [Accessed: 13 May 2020].

Qian et al. 2016. *Ruan Jian Gong Cheng*. Beijing: Tsinghua University Press.

Richards, M. 2015. *Software Architecture Patterns*. (s.l.): O'Reilly Media.

Scrum.org. 2020. *What Is A Scrum Master?* Available at: https://www.scrum.org/resources/what-is-a-scrum-master [Accessed: 28 April 2020].

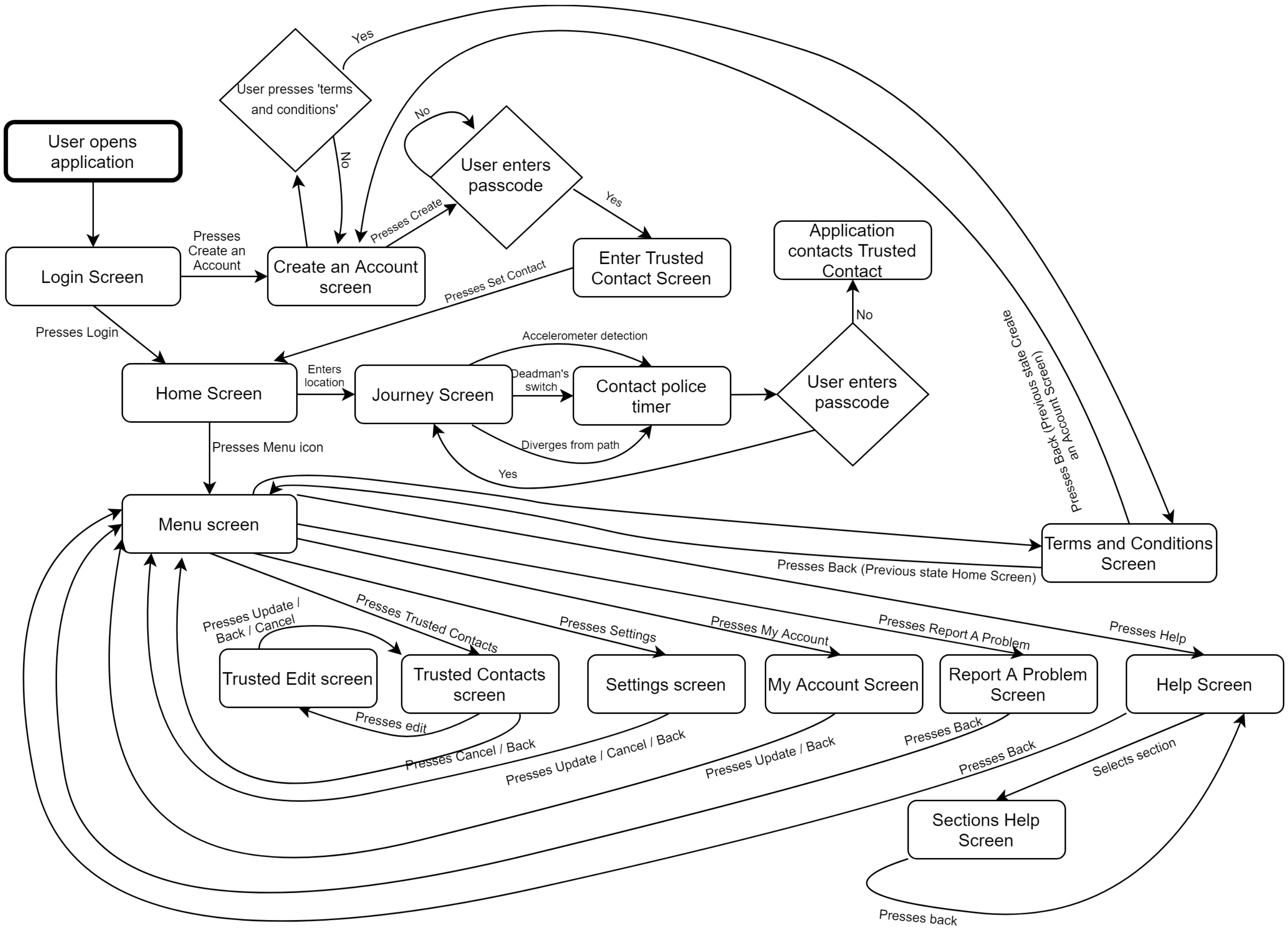
Spring. 2020. *Data Access with JDBC*. Available at: https://docs.spring.io/spring/docs/current/spring-framework-reference/data-access.htm [Accessed: 13 May 2020].

Syer et al. 2020. *Testing the Web Layer*. Available at: https://spring.io/guides/gs/testing-web/ [Accessed: 12 May 2020].

Tuckman, B. W. 1965. Development sequence in small groups. *Psychological Bulleting* 63(6), pp. 384–399. doi: 10.1037/h0022100.

Appendix A

Application STN


User opens application
Login Screen
Presses Create an Account
Create an Account screen
User presses 'terms and conditions'
Yes
No
Presses Create
User enters passcode
No
Yes
Enter Trusted Contact Screen
Presses Set Contact
Application contacts Trusted Contact
Presses Login
Home Screen
Enters location
Journey Screen
Accelerometer detection
Deadman's switch
Diverges from path
Contact police timer
User enters passcode
No
Yes
Presses Menu icon
Menu screen
Presses Back (Previous state Create an Account Screen)
Terms and Conditions Screen
Presses Back (Previous state Home Screen)
Presses Update / Back / Cancel
Presses Trusted Contacts
Presses Settings
Presses My Account
Presses Report A Problem
Presses Help
Trusted Edit screen
Trusted Contacts screen
Settings screen
My Account Screen
Report A Problem Screen
Help Screen
Presses edit
Presses Cancel / Back
Presses Update / Cancel / Back
Presses Update / Back
Presses Back
Presses Back
Selects section
Sections Help Screen
Presses back

Appendix B

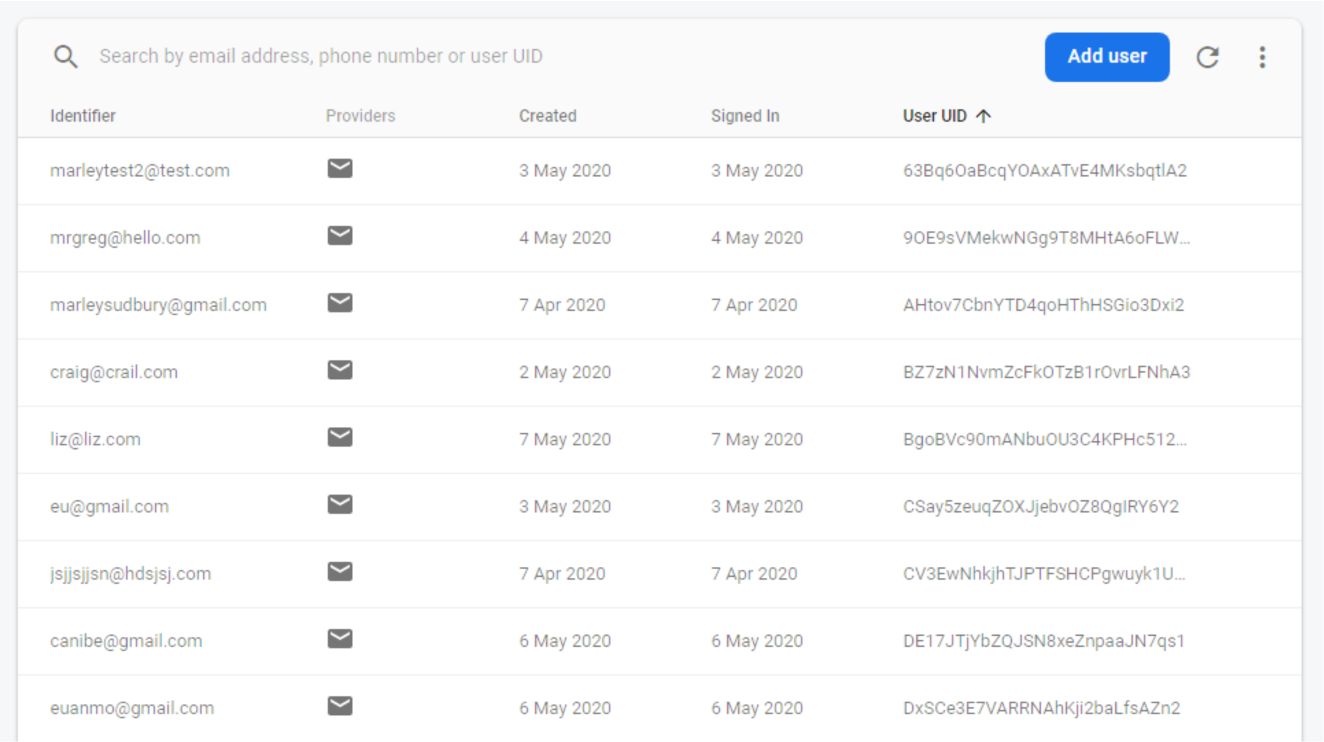
Menu pages

Trusted Contacts	The ‘Trusted Contacts’ screen is a section where the user can view the contact numbers they have set as their ‘Trusted Contacts’. They can click ‘Edit’, present besides each contact, to set a new name and number for the contact should they wish to replace the existing one(s). This is confirmed via the ‘Update’ button.
Settings	The ‘Settings’ section of the app contains functionality where users can adjust their personal preferences. This includes: police contact timer (adjustable via interactive slider), colour scheme (adjustable via drop down), and recording access (adjustable via on/off switch). The user can then confirm these changes via the ‘Update’ button.
My Account	‘My Account’ is where a user’s personal details associated with their account can be updated. This section provides entry fields for the user to adjust their name, email, phone number, and address. The user can confirm these changes via the ‘Update’ button.
Advice	The ‘Advice’ section of the app details steps the user could take to increase their overall safety on nights out. It summarises six simple steps shown to have the greatest impact in increasing overall safety. The user is encouraged to educate themselves further via links at the bottom of the page.
Help	The ‘Help’ section of the application provides textual assistance to users regarding how to use different features of the application. The main ‘Help’ screen presents 4 buttons for the users to select, each referring to a different feature of the application: ‘Start a Journey’, ‘Trusted Contacts’, ‘My Account’, and ‘Settings’. After the user selects the section that they require assistance with, they are given a step-by-step guide on how to use that feature of the application.
Report a Problem	The ‘Report a Problem’ screen is a simple part of the interface where the user is provided contact details of the team, should they experience issues when using the application. The user is provided with an email address plus a contact number, and given information regarding the maintenance team’s operation times during the work week.
Terms and Conditions	The ‘Terms and Conditions” section was generated via online tools and advice; in the future however, ideally we would have legally reviewed terms and conditions in this section, for the user to peruse at their leisure.
Privacy Policy	The ‘Privacy Policy’ section summarises what data the app uses, stores, and collect, plus how the data is used. It is written in an accessible format to avoid overwhelming the user, and empower them regarding choices involving their data.
Logout (button)	The logout feature is situated at the bottom of the main menu as a red button. Once the user selects this button they will be returned to the ‘Login’ screen.

Table B.1: Contents of the menu

Appendix C

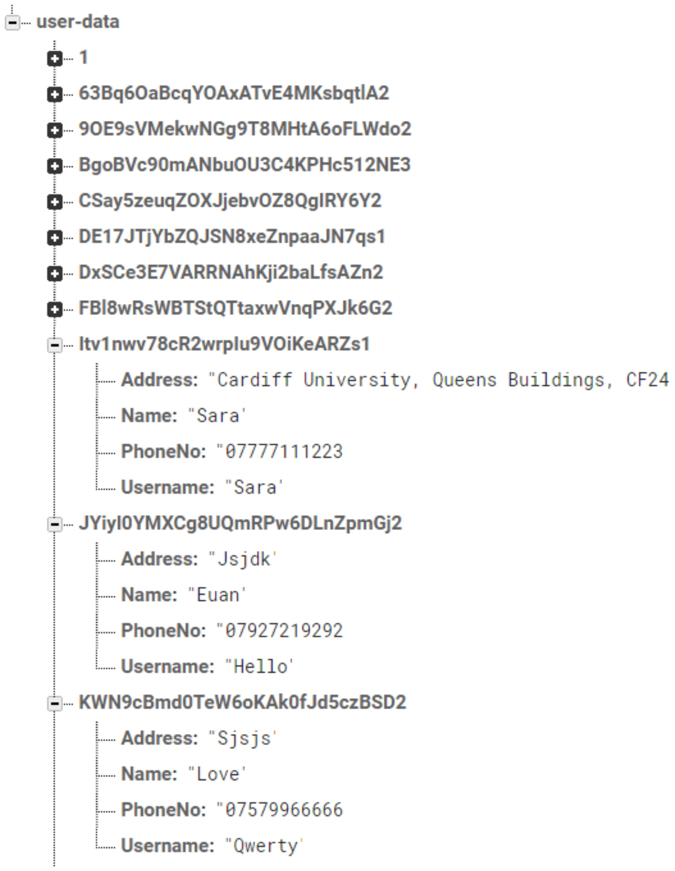
Excerpt from Firebase users



Identifier	Providers	Created	Signed In	User UID ↑
marleytest2@test.com	✉	3 May 2020	3 May 2020	63Bq6OaBcqYOAxATvE4MKsbqtlA2
mrgreg@hello.com	✉	4 May 2020	4 May 2020	9OE9sVMekwNGg9T8MHtA6oFLW...
marleysudbury@gmail.com	✉	7 Apr 2020	7 Apr 2020	AHtov7CbnYTD4qoHThHSGio3Dxi2
craig@crail.com	✉	2 May 2020	2 May 2020	BZ7zN1NvmZcFkOTzB1rOvrLFNhA3
liz@liz.com	✉	7 May 2020	7 May 2020	BgoBVc90mANbuOU3C4KPHc512...
eu@gmail.com	✉	3 May 2020	3 May 2020	CSay5zeuqZOXJjebvOZ8QgIRY6Y2
jsjjsjjsn@hdsjsj.com	✉	7 Apr 2020	7 Apr 2020	CV3EwNhkjhTJPTFSHCPgwuyk1U...
canibe@gmail.com	✉	6 May 2020	6 May 2020	DE17JTjYbZQJSN8xeZnpaaJN7qs1
euanmo@gmail.com	✉	6 May 2020	6 May 2020	DxSCe3E7VARRNAhKji2baLfsAZn2

Appendix D

Excerpt from Firebase RTDB



```
user-data
    1
    63Bq6OaBcqYOAxATvE4MKsbqtlA2
    9OE9sVMekwNGg9T8MHtA6oFLWdo2
    BgoBVc90mANbuOU3C4KPHc512NE3
    CSay5zeuqZOXJjebvOZ8QglRY6Y2
    DE17JTjYbZQJSN8xeZnpaaJN7qs1
    DxSCe3E7VARRNAhKji2baLfsAZn2
    FBl8wRsWBTStQTtaxwVnqPXJk6G2
    Itv1nwv78cR2wrplu9VOiKeARZs1
        Address: "Cardiff University, Queens Buildings, CF24
        Name: "Sara'
        PhoneNo: "07777111223
        Username: "Sara'
    JYiyl0YMXCg8UQmRPw6DLnZpmGj2
        Address: "Jsjdk'
        Name: "Euan'
        PhoneNo: "07927219292
        Username: "Hello'
    KWN9cBmd0TeW6oKAk0fJd5czBSD2
        Address: "Sjsjs'
        Name: "Love'
        PhoneNo: "07579966666
        Username: "Qwerty'
```

Appendix E

Login procedure

Algorithm E.1 Login procedure

```
Require: the user is not already logged in
function LOGIN(email, password)
    Firebase.auth login request with details provided
    if request failed then
        display error message                          ▷ a popup box
    else
        go to home screen
    end if
end function
```

Appendix F

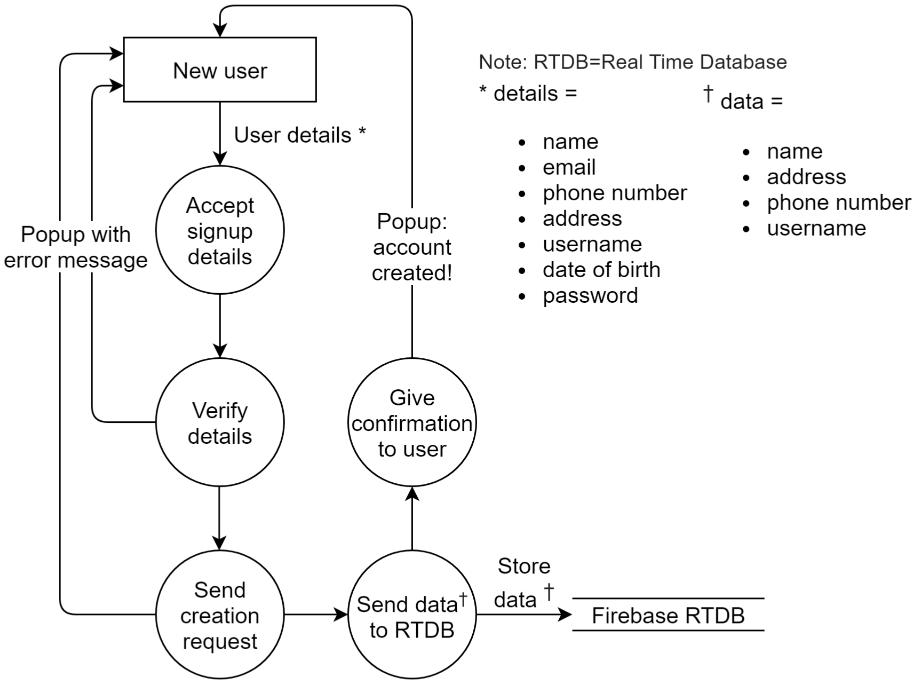
Create account procedure

Algorithm F.1 Create account procedure

```
Require: all fields are filled with valid information
function SIGNUP(email, password, name, address, phone, usern)
    Firebase.auth signup request with email and password
    if request failed then                          ▷ account already exists
        display error message                       ▷ a popup box
    else
        set Firebase.database “user-data/” + current user ID to
        {
            Name: name,
            Address: address,
            PhoneNo: phone,
            Username: usern
        }
        SET-PASSCODE()
        SET-CONTACT()
        go to home screen
    end if
end function
```

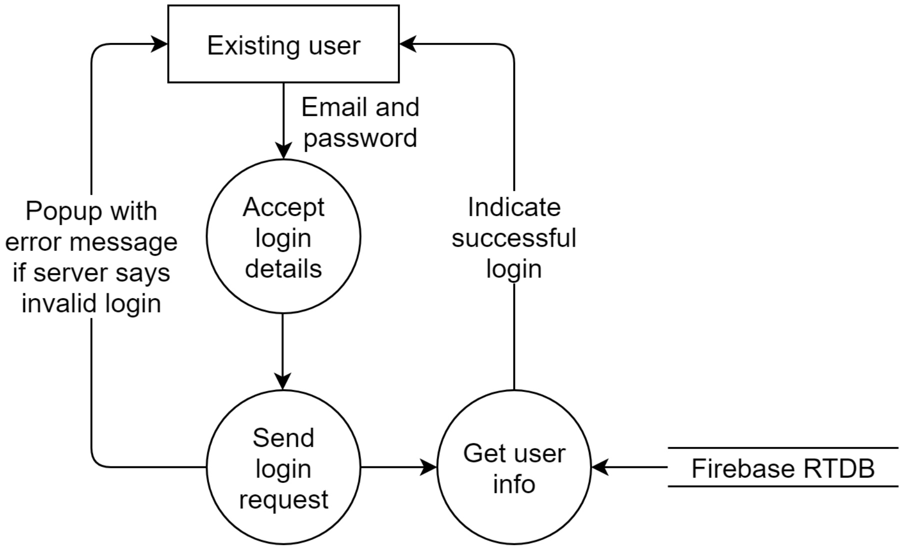
Appendix G

Data flow of sign-up



Appendix H

Data flow of login



Appendix I

Test cases

I.1 Test case 1

Test Case ID: 1.			
Test Purpose: The user can create an account.			
Environment: Expo client app (>=2.14.0) running on Android (>=10.0).			
Preconditions: The user has the app installed but is not logged in and has no existing account associated with their phone number.			
Test Case Steps:			
Step No.	**Procedure**	**Expected Results**	**P/F**
1	On the login screen, press the button which says ‘Create Account’.	A form will appear for the user to fill in.	P
2.1	Input the phone number of the test device in the Phone Number input.	A tick mark (e.g. ✓) will signify that the phone number is valid.	P
2.2	Input the password ‘TestingPassword52563’ into the Password and Verify Password inputs.	The passwords will appear as bold circles (e.g. •) and a tick mark (e.g. ✓) will signify that the first input meets requirements and that they are the same.	P
2.3	Enter ‘16/1/1990’ in the Birth Date input.	A tick mark (e.g. ✓) indicates that the birthdate is valid (i.e. the user is 18+)	P
2.4	Press the Create Account button.	A form appears asking for a number that the user will receive via SMS.	P
3	Enter the verification code once recieved via SMS.	The main menu appears.	F
Comments All of the test cases pass except the last one as it was decided to not use SMS as verification. This was done because it was much more practical for testing purposes.			
Author: Marley	**Date:** 06/12/19	**Checker:** Henrijs	**Date:** 30/04/20

Table I.1: Test case 1

The results of this test case show that the create accounts page works mostly as described in the interim report. The major change in the test case is that we decided not to implement the SMS verification code. This was done for testing purposes and would require minimal effort to switch over as described in section 3.1.

I.2 Test case 2

Test Case ID: 2.			
Test Purpose: The user can login with a previously existing account.			
Environment: Expo client app (>=2.14.0) running on Android (>=10.0).			
Preconditions: An account exists with the credentials ‘07700 900007’ and ‘TestingPassword52563’.			
Test Case Steps:			
Step No.	**Procedure**	**Expected Results**	**P/F**
1.1	On the login screen enter ‘07700 900007’ in the Phone Number input.	The phone number entered should be shown.	P
1.2	Enter ‘TestingPassword52563’ in the password input.	The password will appear as bold circles (e.g. •).	P
2	Press the Login button.	The main menu appears.	F
Comments This test case fails because now the email is required as opposed to phone number.			
Author: Marley	**Date:** 06/12/19	**Checker:** Euan	**Date:** 06/05/2020

Table I.2: Test case 2

Since the interim report, our implementation of the login function has changed slightly from our original plan. We now use and email address instead of a phone number which is why this test case fails. The login is fully functional and the user can login to a pre-existing account using the correct email and password.

I.3 Test case 3

Test Case ID: 3.			
Test Purpose: The user can successfully start a journey.			
Environment: Expo client app (>=2.14.0) running on Android (>=10.0).			
Preconditions: An account exists with the credentials '07700 900007' and 'TestingPassword52563'.			
Test Case Steps:			
Step No.	**Procedure**	**Expected Results**	**P/F**
1	From the main menu select 'Navigation'.	The navigation screen will appear.	P
2	Press 'Select Destination'.	A text field will appear for the user to input a destination.	P
3	Enter 'Cardiff University Students' Union, Park Place, Cardiff, CF10 3QN'.	A list of possible routes from the user's current location to the Student's Union will appear on the screen.	P
4	Select the first route in the list.	The journey will begin, with a standard SatNav style display guiding the user.	F
Comments All steps of this test passed except for the last one, this is because of changes that were made to how the journey would be shown in the 'SatNav' style.			
Author: Marley	**Date:** 06/12/19	**Checker:** Louis	**Date:** 04/05/20

Table I.3: Test case 3

I.4 Test case 4

Test Case ID: 4.			
Test Purpose: The user can finish a journey successfully.			
Environment: Expo client app (>=2.14.0) running on Android (>=10.0).			
Preconditions: The user has already started a journey (see section I.3).			
Test Case Steps:			
Step No.	**Procedure**	**Expected Results**	**P/F**
1	Follow the directions presented on the screen at walking speed until you reach the destination.	The map in the app moves with the user and then displays 'Journey Complete'.	F
Comments On reaching the end of journey, while the user is able to see that they have reached the end of their journey, they are not met with a message informing them of the same. As such, there is no way of the user realising their journey is complete despite reaching the end of their chosen suggested route.			
Author: Marley	**Date:** 06/12/19	**Checker:** Sara	**Date:** 06/05/20

Table I.4: Test case 4

I.5 Test case 5

Test Case ID: 5.			
Test Purpose: Alarm will be raised if the user significantly diverges from chosen route.			
Environment: Expo client app (>=2.14.0) running on Android (>=10.0).			
Preconditions: The user has already started a journey (see section I.3).			
Test Case Steps:			
Step No.	**Procedure**	**Expected Results**	**P/F**
1	Instead of following the directions provided by the app, walk 25 metres in the opposite direction.	Once the GPS registers that the device is more than 25 metres away from the selected route it will tell the user to return to the path.	F
2	Ignore the message and remain off of the path. Wait for the countdown to begin.	After a length of time determined by the users settings the app will start an alert countdown and ask the user to input their password to disable the alert.	P
3	Ignore the message and wait for the countdown to end.	After the countdown ends, a message is sent to the emergency contacts or police depending on the users settings.	P
Comments Contrary to our original plan, the app does not raise the alarm at a deviation of 25m; the revised limit is that of 200m. Following a deviation of 200m from the user's chosen route, the app proceeds to raise alarm as stated in the test case steps.			
Author: Marley	**Date:** 08/12/19	**Checker:** Sara	**Date:** 06/05/20

Table I.5: Test case 5

I.6 Test case 6

Test Case ID: 6.			
Test Purpose: The user can pause and resume a journey that they are on.			
Environment: Expo client app (>=2.14.0) running on Android (>=10.0).			
Preconditions: The user has already started a journey (see section I.3).			
Test Case Steps:			
Step No.	**Procedure**	**Expected Results**	**P/F**
1	Press the pause button at the top of the page.	A prompt will appear asking for the user's password.	F
2	Enter the password associated with the account.	Text at the top of the screen will indicate to the user that their journey has been paused. The map and route will still be on the screen, but will not respond to the user's GPS. The button at the top of the screen which previously said 'Pause' will now say 'Resume'.	P
3	Press the resume button.	The previous journey will be resumed if the user is still on the route, otherwise they will be prompted to select a new route to the destination.	P
Comments On pressing the 'Pause' button on top of the screen, the text changes to 'Resume', indicating that the current journey has been paused. The app however does not ask the user for a password to do so. Subsequently, on clicking 'Resume' the user's prior journey is duly resumed.			
Author: Marley	**Date:** 09/12/19	**Checker:** Sara	**Date:** 06/05/20

Table I.6: Test case 6

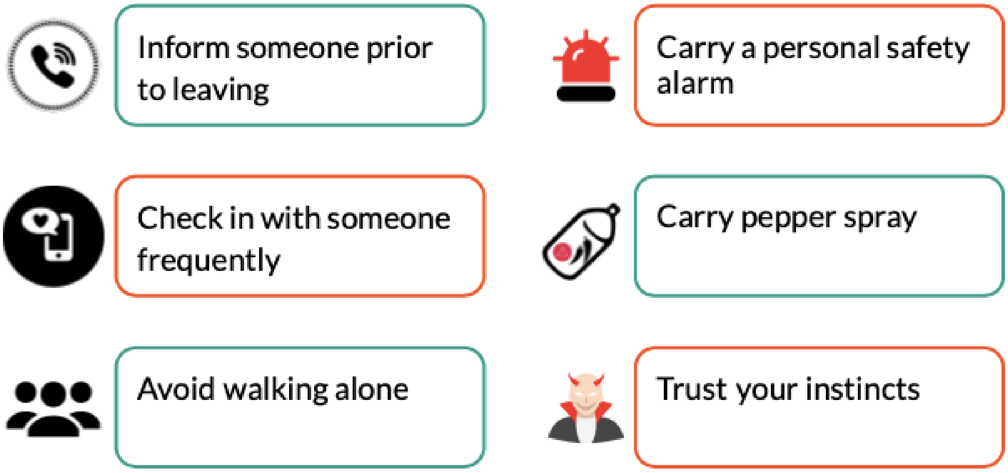
I.7 Test case 7

Test Case ID: 7.			
Test Purpose: The user can cancel a journey that they are on.			
Environment: Expo client app (>=2.14.0) running on Android (>=10.0).			
Preconditions: The user has already started a journey (see section I.3).			
Test Case Steps:			
Step No.	**Procedure**	**Expected Results**	**P/F**
1	Press the cancel button at the top of the screen.	A prompt will appear asking for the user's password.	F
2	Enter the password associated with the account.	A pop-up will tell the user they have successfully cancelled their journey.	F
Comments The user is able to press the 'Cancel' button on top of screen. However,the app does not prompt the user for a password; the user is not required to input password in order to cancel journey. The journey is successfully cancelled, but the user is not is not presented with a pop-up informing them of the same.			
Author: Marley	**Date:** 10/12/19	**Checker:** Sara	**Date:** 05/05/20

Table I.7: Test case 7

Appendix J

Advice given to users



Inform someone prior to leaving

Carry a personal safety alarm

Check in with someone frequently

Carry pepper spray

Avoid walking alone

Trust your instincts

Appendix K

Comparison table

App	**Description**	**Comparison with prototype**
bSafe	Set up a network of family/friends; they can follow the user home via a GPS trace. On hitting the emergency alarm feature, it will send the user's exact location to listed contacts, while recording audio and video for later. Free but the GPS could drain battery faster. Source: https://stylecaster.com/personal-safety-app/	Similar to our app in terms of basic features plus the recording of audio and video, plus informing trusted contacts of the GPS coordinates. However, doesn't possess the ability to suggest safer routes to user. Additionally, bSafe is largely subscription oriented, which makes safety an expensive affair.
Road Buddy	Suggests alternative routes to produce a safer journey. Emergency trigger lets user alert trusted contact in case of trouble. Source: https://www.telegraph.co.uk/technology/news/10047029/Road-Buddy-mobile-app-plots-safe-routes-to-walk-home.html	Similar in the core functionality but was initially developed in 2013 to work on Firefox OS, hasn't progressed since.
Circle of 6	Choose six contacts; contact them via car, phone, or chat icons within the app whenever required. Included is a danger button which when activated, contacts hotlines for victims of domestic and sexual abuse. Free app.	Underlying concept is similar: alert folks when faced with danger. However, our prototype is better not only due to the fact it suggests safer routes, but also since our dead man's Switch alerts either the trusted contacts, or the police, guaranteeing quicker action than contacting a hotline.
Red Panic Button	Sends location via Google Maps link to emergency contacts (entered when setting up the app) whenever the red panic button is pressed. Can also be set up to update Twitter/Facebook status via the button. Free app.	Their Red Panic functions much the same as our dead man's switch, barring the audio/video recording functionality present within ours. Once Again, lacks the suggested routes feature; however it's simplified design is instantly intuitive.
Hollie Guard	Is able to capture both GPS location plus audio/video recordings when required. Tracks user on their route and alerts trusted contacts, if the user fails to reach their destination within their preset time parameters. Source: https://hollieguard.com	Similar, but lacks the core of our app i.e. suggested routes.

Table K.1: Comparison of similar applications