



SchoolNav – Navigation System to Improve and Simplify Life at School

OCR A-Level Computer Science NEA

Isaac Gunananth

Table of Contents

Analysis	4
Problem Definition	4
Stakeholders	5
Computational Thinking	9
Thinking Abstractly	9
Thinking Ahead	9
Thinking Procedurally	10
Thinking Logically	10
Thinking Concurrently	10
Research the problem	11
Example application 1: School Locator	11
Example application 2: Google Maps	13
Example application 3: Waze	18
Interview with a Stakeholder	22
Stakeholder Questionnaire - Quantitative data	25
Specifying the Proposed Computational Solution	32
Essential Features	32
Desirable Features	32
Limitations	34
Hardware Requirements	34
Software Requirements	35
Success Criteria	35
Design	43
Decompose the Problem:	43
User Interface Design	44
Login-Page	45
Logo	46
Background Image	46
Main Menu	47
Settings Page	48
Weather Information Pop-Up	49
Search Locations Pop-Up	50
Settings Page: School Information Drop-Down:	51
Settings Page: Personal Information Drop-Down:	52
Settings Page: Additional Settings Drop-Down:	53
Settings Page - Iteration 2	54
Flowcharts and Pseudocode	55
Key Variables and Functions	76
Validation	81

Test Plan for Iterative Development	83
Login/Sign-up Page	83
Main-Menu Page	89
Settings Page	97
Robustness/Functionality Test Plan for Post-development	101
Usability Test Plan for Post-development	103
Development of Coded Solution	106
Initialisation	106
Setting Up Coding Environment	106
Libraries, Packages and Extensions	106
Coding Conventions and Testing Methods	108
Iterative Development of Coded Solution + Testing	110
Default Code and Initial Run Test	110
Version 1 - Initialise Login Screen	112
Version 2 - Validation for Log-In Screen + Other Key Components	142
Version 3 - Initialising Register Screen	163
Version 4 - Validation of Register Screen and Functioning Login System	176
Version 5 - Initialising Main Menu	187
Version 6 - Implementing Map of School	206
Version 7 - Routing Users to Selected Locations + Corridors	244
Version 8 - Initialising Settings Page	285
Version 9 - Add Commenting/Report Feature for Users and Adding ETA System	305
Post Development Testing	316
Review of Iterative Development Tests	316
Robustness/Functionality Testing	331
Evidence of Robustness/Functionality Testing Working:	334
Usability Testing	339
Evidence of Usability Testing:	342
Evaluation	350
Demonstration	351
Final Stakeholder Testing	352
Success Criteria Evaluation	358
Essential and Desirable Features	370
Further Development	375
Functionality/Meeting the Success Criteria	375
Robustness	376
Usability	377
Limitations	377
Maintenance	378
Final Code	378
Sources	433

Analysis

Problem Definition

Students, parents and even teachers can find it difficult to navigate through schools, specifically to initial days but not restrictively. Several apps exist to assist with path-finding to a particular building or place but fail to assist with in-building navigation. Students on open days, parents on parents evenings and a plethora of more niche examples can be listed as to why this is necessary, students need to be confident entering their new school, giving them a good experience overall and parents should be confident in navigating their children's school. This application is to improve the experience of those visiting a school for any reason. By improving this aspect they would be in better mind for their purpose of visit and reduce stress. Stress is a known culprit to headaches and difficulty concentrating, hence not priming students for the right learning environment, and visitors not being ready for their day. I have narrowed this precise navigation to schools alone but my model can be adapted to fit other buildings.

The application SchoolNav also aims to shorten the time spent commuting from rooms. Events of maintenance, construction and exam periods can mean certain paths are unavailable and could cause stress and a waste of time figuring out another route. Not only does my application aim to tackle the students finding a viable route, but also aids those undergoing exams as people walking past can be a distraction to students and hinder the progress of students on their papers. SchoolNav provides clear and short routes which can mean every stakeholder is happy and satisfied. Further, students finding rooms they are less familiar with can be smoother as well, also teachers assisting students with directions would be less common through the application.

Overcoming mazeophobia is also another purpose of SchoolNav. The fear of being lost can be a natural deterrent to most people coming to these events. It is safe to say that my proposed application can improve attendance rates by reducing the fear of being lost. By improving the attendance rates on induction days, parent evenings and other similar events, the school benefits from more people knowing about and experiencing them. More parents coming to parents' evenings without fear means more parents are given the feedback necessary to help their children grow further academically.

Rooms around the school are not numbered logically, for example, Room 30 can be in the Maths department and Room 29 can be in the English Department.

According to the room name, it is difficult to decipher where or which department that room is from. Hence even those in the school or building for a long time may need help finding certain rooms. This is where the school navigation application can help with these use cases.

Stakeholders

This table represents all the main stakeholders involved with SchoolNav and how they would use the application. All the stakeholders form part of the target market for the application, the app aims to help better the quality of life for these stakeholders. The use case column shows the frequency of uses for each stakeholder.

Stakeholder	Description	Use Cases
General Students (studying at school)	<p>General Students consist of members of the school or guests who need to move around the school to meet their needs for each day. Students already a part of their school would want the most optimised route to get to their lessons. Time can be minimised by also considering inactive corridors and pathways, which otherwise can disturb students. Not only does this mean students will be able to spend less time outside of classrooms but also promotes punctuality to lessons, allowing prompt arrival to learning.</p> <p>Transition days can also be an important day to enhance the first official day of a student to their new school, hence can be an important application of SchoolNav. New Year 7s and new 6th Formers can make functional use of the application since they are unaccustomed to the school and its rooms. Hence an important stakeholder of the project.</p>	<p>Multiple Times in a day; Every time they would have a lesson in a different room.</p> <p>Transition Days</p>
Disabled Students/Adults	<p>Accessibility routes can be specifically directed to those who need it. Disabled people will most definitely need this application since stairs and steps are inaccessible to them the application will first direct them to the lift which takes them to the floor the room/place they need to go</p>	<p>Same as General Students - Multiple Times a day; Every time they would have a lesson in a different room.</p>

	<p>to is in and then route them to the room. Those who require the use of a lift will need to traverse through a different route than those without. This is significantly important because disabled members may find it even more difficult to traverse a school/building since their route is so less common and less known about. These people who require the accessible routes will be on time for their lesson/other and not have to stress about the route being accessible to them.</p>	
Visiting Students (Guests)	<p>Visiting students (guests), especially on open days, need to be welcomed into the school. Most often these open days consist of departmental talks, requiring students to manoeuvre around the school. Considering that these students are usually new to the school and haven't navigated around the building/s before; they would need the help actually to go to their chosen talks at the appropriate time. My application also aims to take into account the capacity of students at each departmental talk, hence if the room is full the student would be directed to a different talk until that room is free again. This not only streamlines the process of choosing your departmental talk but also gives information about which ones are available to attend at that time. These guests would want their experience to be smooth and enriching, making the most use of their time attending the correct sessions and commuting in the shortest routes.</p>	<p>One-time use for that occasion/s Open Days</p>
Students Undergoing Exams	<p>Students undergoing exams would want their conditions to be as optimal as possible, meaning no distractions and also quiet conditions to allow precise focus in examinations. Hence why our program can provide alternate routes by closing off the corridors linked with the rooms/halls exams are held in. This can improve the quality of focus for students in their exams and also reduce their overall stress, perhaps even allowing them to get the results they want and can achieve. An extension of this could be to students in lessons since those walking in corridors (and especially talking) can be a distracting force for students. But since most</p>	<p>Multiple times a term/Occasionally During Exam Seasons – Daily</p>

	corridors are vital to certain lessons and unavoidable this is unfeasible at this point. Since exams hold a greater significance to students than lessons, I have chosen students undergoing exams over students in lessons.	
Parents	Parents would want to have the same uses as Visiting Students (Guests) but parents, in particular, are going to parent's evenings or meetings in which they are called into school. Going to the school of your children should have a positive impact on parents , they should be inspired and full of hope of how amazing their experience at the school was, amplified by their traversing time through the school. Not only that but calling parents in for these events is already a hassle for most parents, leading to busy work schedules , hence my proposed solution would meet their needs to get through the meetings for that day quickly . Time is minimised but also stress is minimised, having directions at hand can simplify the day for them. Also, they can look at the timetable and directions ahead of time. Not knowing where you are going in an event is a great factor in the number of people coming to that event. It not only creates an impression of mystery but fails to create a solid, safe environment for everyone. The fear of being lost will also be diminished through this, which can be stressful most of the time.	Rarely Parents' Evenings and Meetings
Teachers	New teachers applying for a position in the school would want their travel to the school, and the particular room to be as smooth as possible. Teachers also want students arriving to lessons ready to learn, in a calm state of mind . Teachers who have allocated appointments with parents would want them to arrive at the correct time , which can be affected by being lost on the day, and also to arrive calmly, ready for the meeting with their children's teacher. Teachers wouldn't have to guide students to rooms they are not so familiar with or would be better equipped to assist certain students.	Rarely Since teachers often have a single room.

Other Adults	<p>Certain adults visit schools to give lectures and perhaps inspections. There are even adults who install certain equipment who also come in. There are many more niche examples I haven't mentioned but my application can be used to help those adults to get to where they need to in the shortest possible time. Some people hired by the school may work by the hour, hence by spending more time traversing around the school, it may cost the school more.</p> <p>Overall my application can perhaps reduce certain costs.</p>	<p>Adults giving lectures Adults completing a job for the school.</p>
Developer/Maintenance Team	<p>Any new construction or closing of corridors should be logged and updated in the application. Exam times should also be inputted and any bugs should be dealt with as the application runs along. This is if as a developer, I had a budget of £3000 a year and time throughout the year to deal with all of this. This also involves communication with staff to update and log the times and events that affect SchoolNav.</p>	<p>Back-end Adjustments Testing through user access.</p>

Computational Thinking

<u>Thinking Abstractly</u>	<p>The up-to-date mapped network of the school and its associated rooms is most relevant. If the user requires disability access matters to the application. The current position of the user would be inputted rather than found using GPS. Audio-based directions and pictures to assist with each turn and room are necessary. Exam seasons, construction, maintenance and blocked corridors are also vital information to the app. The design of the application and its looks aren't too important however it could be argued a warm and welcoming theme can improve the user's experience with traversing the school and using the app, hence design is also important. The user's reason for traversing through the school is irrelevant, and so is their name and details.</p>
<u>Thinking Ahead</u>	<p>Inputs and outputs of the application are most important since they determine the route users follow. Inputs can be if you require disability access, where you are in the school(closest room) and what room you want to go into. Outputs consist of directions to the room(audio-based + pictures), also ETA(Estimated time of arrival).</p> <p>Caching can be used to suggest the start location as the previously assigned destination for that day, simplifying the user experience.</p> <p>Another possible use of caching is for each timetabled day for students, the routes can be reused as each week goes by. However this can be complicated to implement and also requires the correct data for it to work, e.g. if certain corridors are blocked, the cached route would not be viable.</p> <p>Libraries can be used such as React Native Maps which allows certain functions to be easily accessed, with no need to code them. It saves time to make use of libraries and also</p>

	<p>provides functions you wouldn't otherwise think of. Perhaps it also makes it easier to reuse program functions, hence making the project more achievable.</p>
<u>Thinking Procedurally</u>	<p>Decomposition of the application and project simplifies the task that needs to be completed. The sub-problems consist of mapping the school into a network, associating each corridor/edge with a distance/weight, associating each room and turn with a picture, taking user input to find the shortest routes from room to room, outputting audio and visual directions in that order of procedure.</p>
<u>Thinking Logically</u>	<p>Several points in my application are where a decision needs to be made. These conditions can be if the user needs a disabled access route, which room/place they are in now, and where they want to go. If the user requires disabled access, they may need to use a lift or a different route using a ramp. The room they are in now and where they want to go are used to decide what shortest route they can take to get there.</p> <p>Other conditions can be if a corridor is blocked and the response would be to provide an alternate route which doesn't make use of that blocked corridor.</p>
<u>Thinking Concurrently</u>	<p>Audial directions should be in sync with the visual directions and the pictures shown on the screen. As the user presses a checkpoint button, an animation centres the user's display and presents the next direction/picture. The application should be able to handle multiple requests at once, since students often go to the app at the same time, perhaps a server should be utilised.</p> <p>Concurrently, the map could be designed alongside the settings page, or the audio could be done alongside the pictures. However a critical activity is the map, before the map the pictures wouldn't fit in nor would the audio, hence critical and should be the first task.</p>

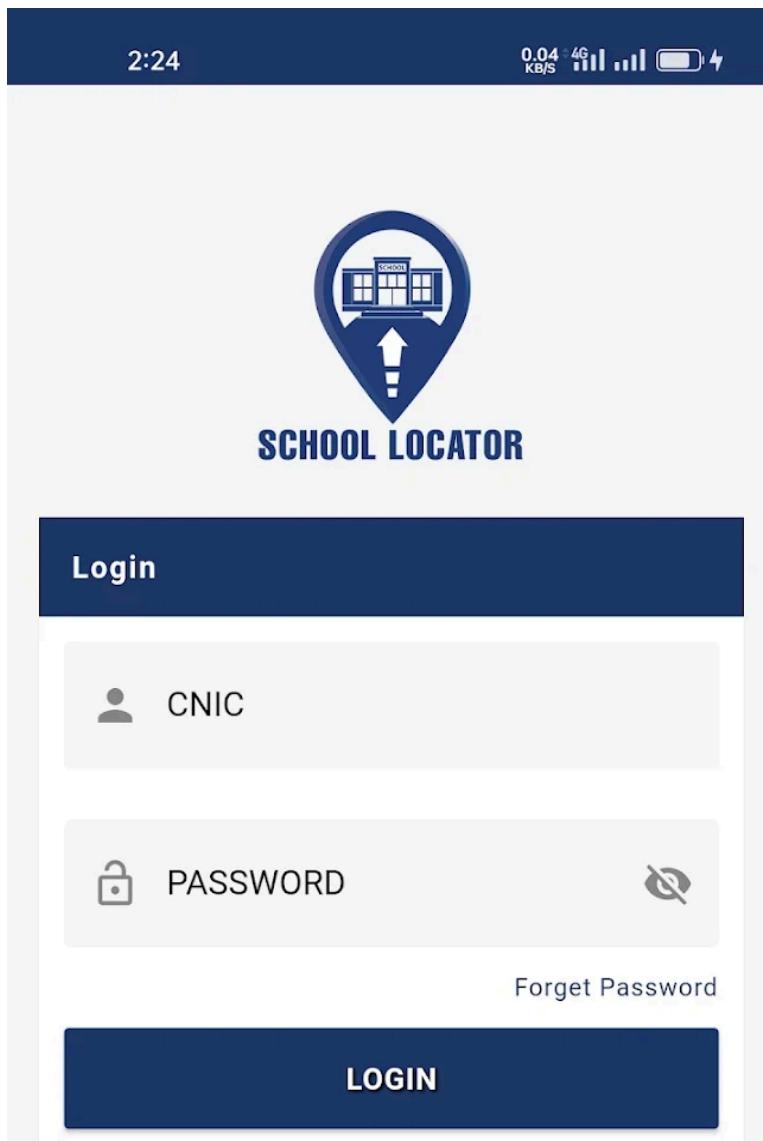
Research the problem

Example application 1: School Locator

This app -

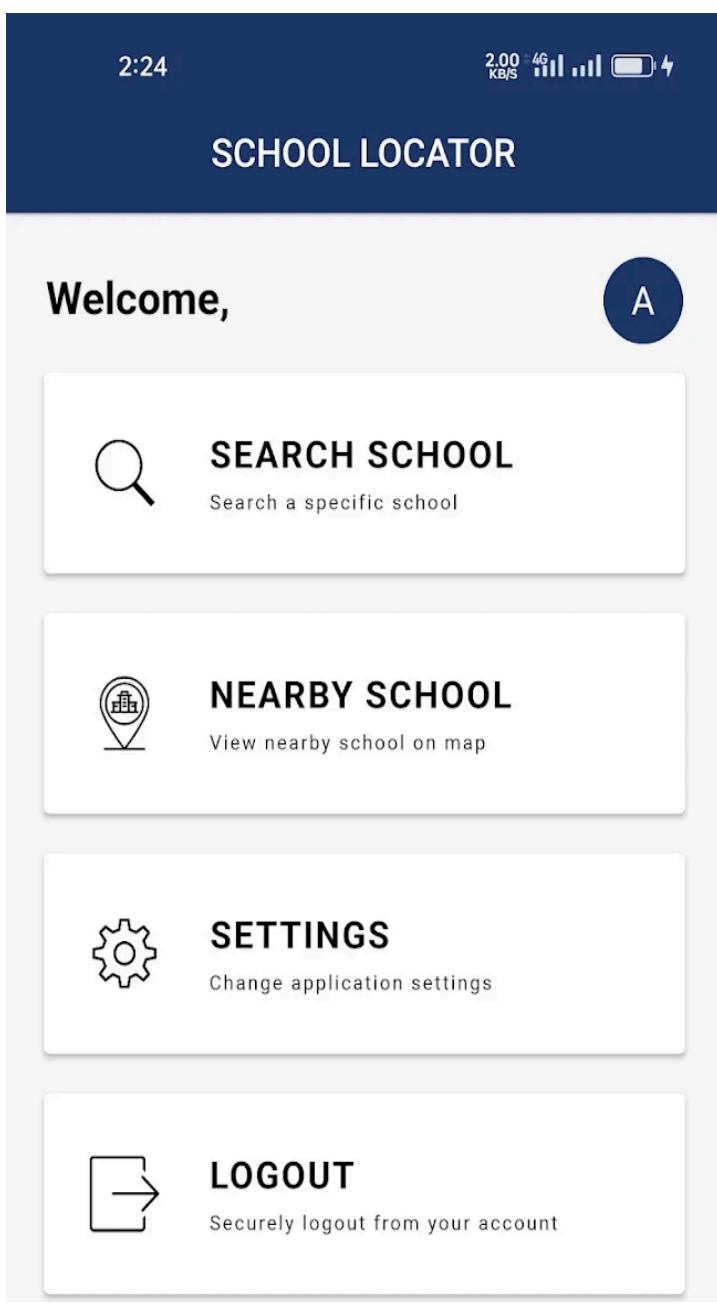
https://play.google.com/store/apps/details?id=sed.pmiu.sla&hl=en_GB&pli=1, School Locator locates nearby schools and presents the navigation to them. It also allows users to monitor the school and have access to certain vital information.

First of all this application is specifically mobile and I am going to use react.native to code and implement my design. Hence, I am testing using mobile software.



School Locator is a relatively small application with 100+ users, but the features in the application would assist and aid me in choosing what features I want to put in the application and what I don't want to put in.

The log-in page gives a sense of security and it prevents people without the right access to access the application. However, I would argue this creates friction with users as remembering and creating a login is an extra step which is not necessary. By simplifying the user's experience not only would more people be inclined to access the application but also people outside the school can enjoy going around the school and getting used to the buildings. This can also provoke a desire to join the school or to visit the school by not adding a login access page.



add a login page.

Features to take from School Locator

- Navigate School Button
- Nearby Schools
- Settings Page

This welcome page holds the essential features of the app. Although the app locates schools nearby and doesn't focus on a single school and mapping routes in it. Rather than the search school button I would add a

Navigate School Button - This would be the current school they are attending.

Nearby Schools could show the schools in a 1-mile radius and their mapped routes, for other schools, users should be able to search the application and download the mapped routes since having them all at once will cause the app to be slow and memory-dense.

In the **Settings**, users can specify if they need disabled access routes or what classes they are in so they're routes can be precalculated. The settings could also hold the option to choose audio-based directions or visual directions with pictures.

I would not add a logout section, since the users do not have to log in. Overall, after looking at the "School Locator" app, I would add to my project a "Navigate school" button, an "Other schools" section, and a "Settings" section and choose not to

Example application 2: Google Maps

This is the website for Google Maps:

<https://www.google.com/maps/@51.5675644,0.0823413,14z?entry=ttu>

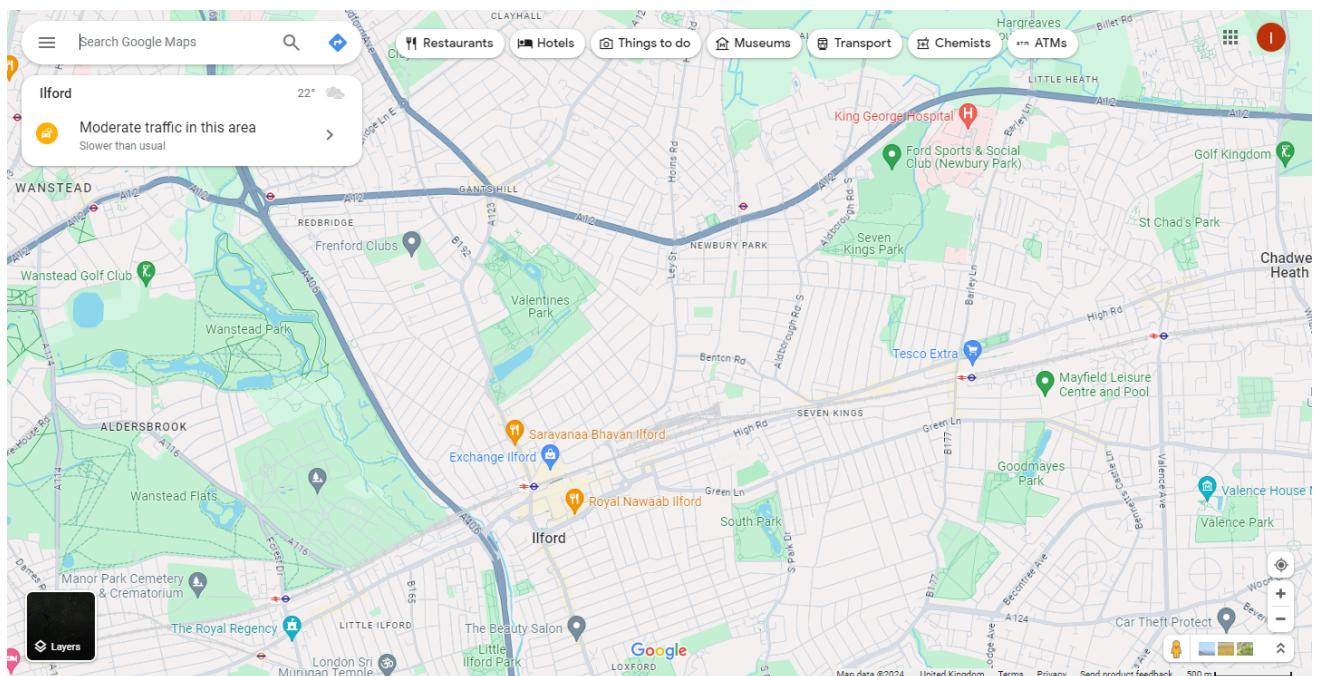
I used the mobile version of the app for testing since users would mainly use SchoolNav on mobile devices.

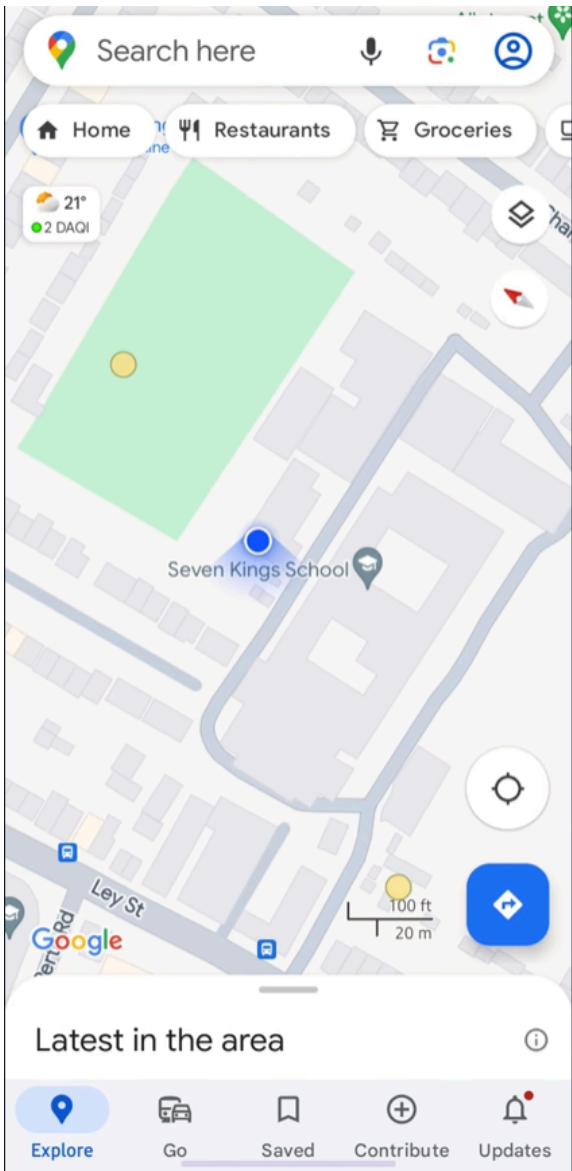
This video is the testing of Google Maps:

<https://www.youtube.com/watch?v=yFp26SkVYz0bn>

I am going to assess and discuss the pros and cons of the features in Google Maps.

General Overview



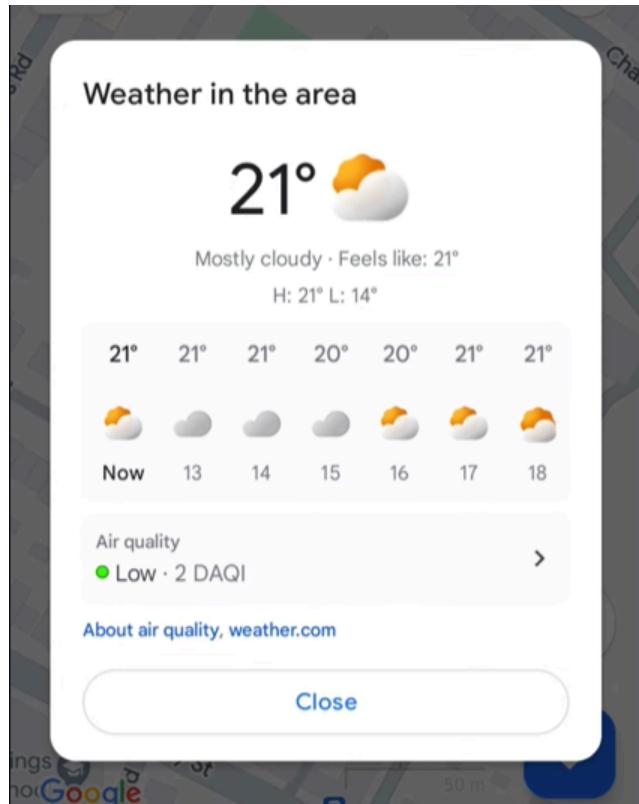


There are several features which make Google Maps so successful.

The **compass** on the right-hand side allows users to orient themselves. There is a **blue dot** representing where the user is relative to the map. There is a **voice function** to search, but since my application is for rooms in a school and certain places in the buildings, this wouldn't be a worthwhile feature. It also needs access to the user's microphone, which may put some people off.

Common places are listed in buttons on top below the search bar. This can be used in my application for places like their form room, main hall, canteen and toilets. The settings page(referenced previously) could be used to signify their gender, hence accordingly, gendered toilets can be shown in button format. Their form room can also be assigned on the settings page.

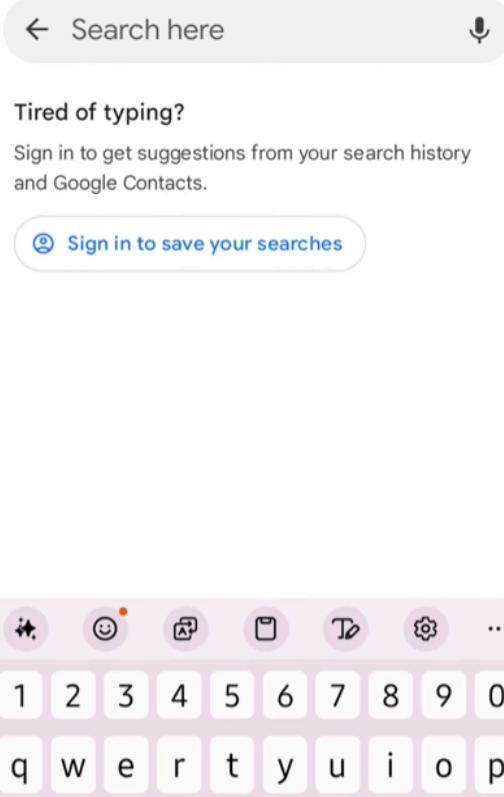
The **reorient button**, allows users to return to the position they are in.



Google Maps presents the option to view the weather in the area for the day, and also once clicked, the week.

This feature is vital for my application since if students know the weather in the area of their school, they can come prepared with an umbrella or coat. This will improve their experience with school and reduce stress and wet clothes and books.

It represents the temperature in Celsius and a visual representation of the weather with a sun and cloud in front. This is a nice feature which simplifies the user's experience in understanding the weather.



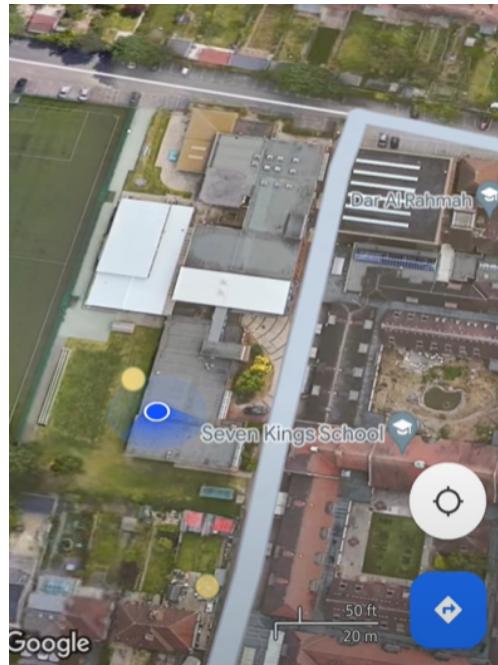
Google Maps also provides **saved places** you have searched recently. I think this is an amazing feature as it reduces the amount of work the user needs to do and it also makes it faster for users to get to where they want to go.

However, previously I have mentioned I don't need a sign-in for my users. Hence saving user searches is not viable without the sign-in feature.

Since it is a trade-off, having the sign-in vs saved searches, I would choose my users to have their searches saved since signing in is a one-time process, whereas searching is a daily/repetitive process. For that reason I will now include a sign-in page as shown in the School Locator application, to allow users to have their searches saved.

Not using a **sign-in** feature means users will have to store their searches, locally meaning

this uses their system resources and storage, which may put off users. Hence I would use a sign-in feature to link that user with their specific searches.



Making use of **satellite imagery** can allow students to understand and visualise where they are going. But it isn't viable or necessary for a small building in which the rooms are hidden, hence impractical to use.



When you zoom in enough with Google Maps in **3D mode**, with each house a number is represented which is the house's door number.

Perhaps the school map on default can show the **room numbers** for each room.

It can be argued that **3D** should be used instead of **2D**, but **2D** is better and easier to map. Since in **3D** all the rooms would have the same dimensions, it might be futile to map and offer a **3D model** of the school map. Hence I would include a **2D model** and not a **3D one**.



To present the **traffic** on the roads around is a very useful feature. This can be through users all using the app currently.

However, if I would want to implement this in my application, I would need a significant number of students. Also, factoring students in corridors and routing around traffic can be an amazing, sought-after feature.

I would add this traffic feature if I had more time and a greater financial budget.

Features to take from Google Maps

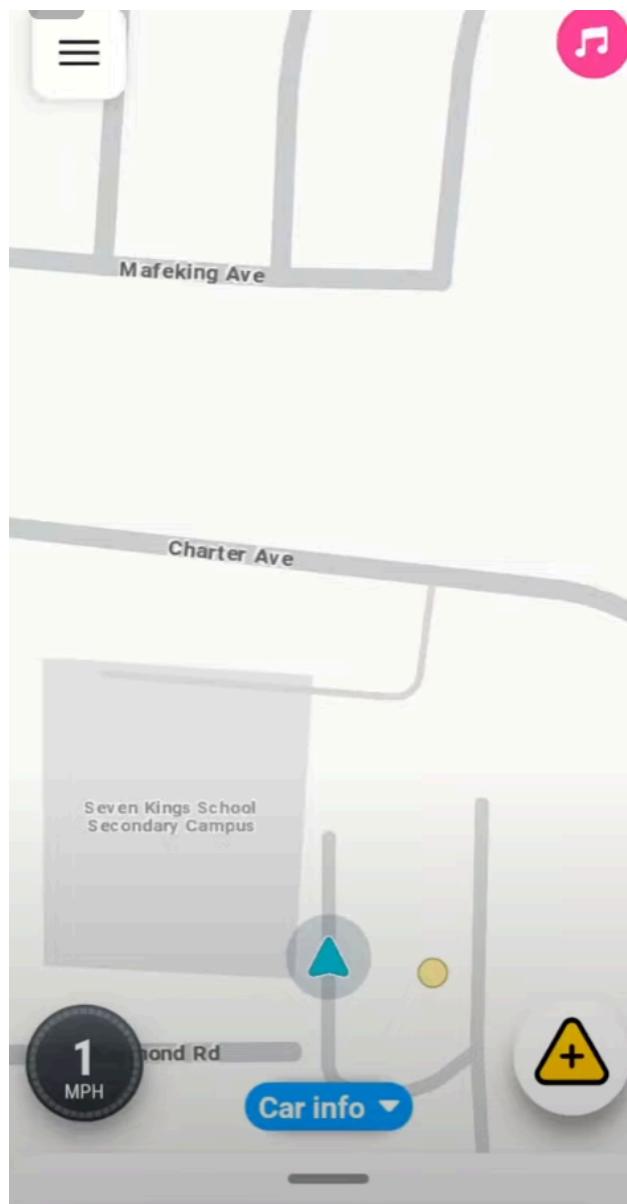
- A sign-in page
- Compass
- Pointer to show where the user is
- Common Places
- Reorient Button
- Weather for area
- 2D Mode with room numbers

Example application 3: Waze

Waze's target audience majorly uses the mobile application of Waze, but here is the website for reference: <https://www.waze.com/en-GB/live-map/>

I mainly chose Waze, since it matches the aim and vision of my application, also it uses a genius feature which uses user feedback to update its routes and maps, e.g. closure on the road.

The video of me testing through the mobile app is linked here:
<https://www.youtube.com/watch?v=wUOXDIAeM1Y>



This application uses an **arrowhead** rather than a blue dot to represent the user's position. I will discuss further in the design section what I will choose.

Waze is specifically designed for road users who drive a motor vehicle, hence a **speedometer** is unnecessary and would be taxing on the system's CPU usage to implement, by reducing the hardware requirement I am opening my software to a larger group of users.

What do you see?

X



Traffic



Police



Accident



Hazard



Closure



Blocked lane



Map Issue



Bad weather



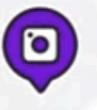
Fuel prices



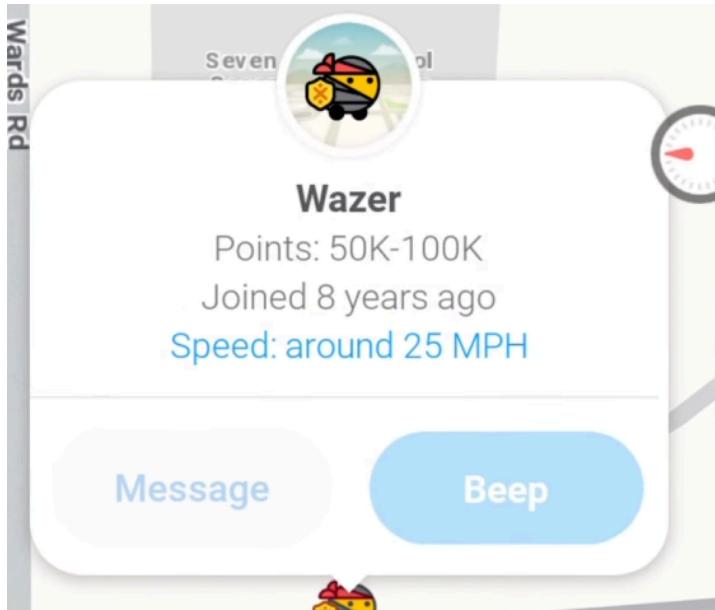
Roadside help



Map chat



Place



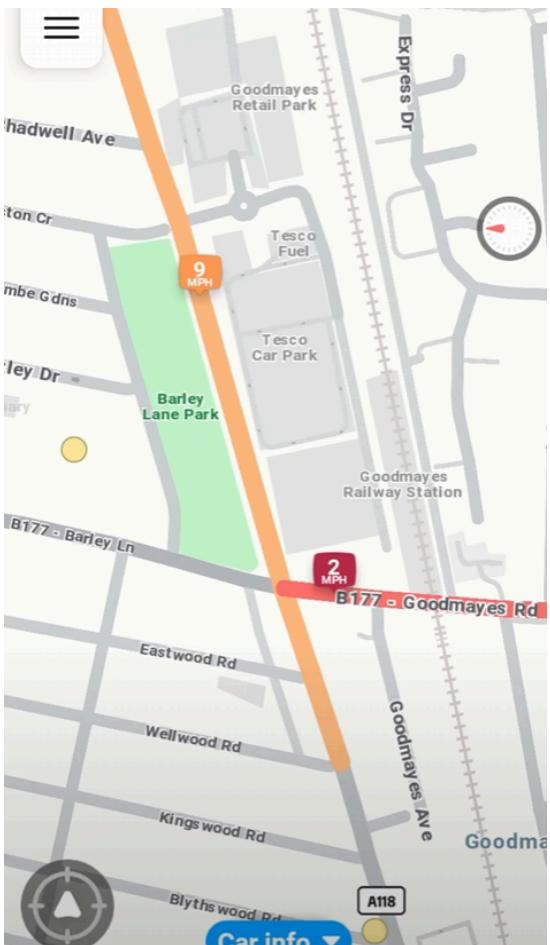
The app has the option to report in the left-hand corner. This allows the user to report several factors shown on the left-hand side. I could adapt this to show events of high traffic (lots of students traversing through a corridor) if an exam is going on etc.

This means there would have to be less work done on the developer's side to input exam periods and closed corridors, but this information could come from users themselves.

A cooldown is in place for this feature, preventing users from misusing this feature. Also, it takes in the likes and input of other passing users to confirm the remarks.

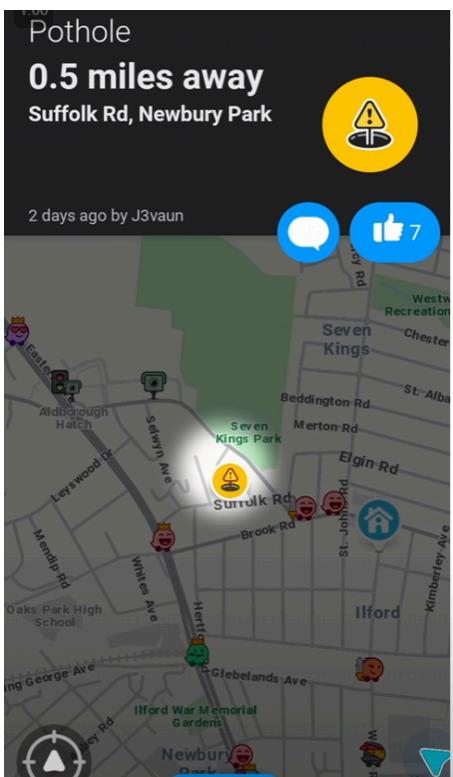
Waze also has a loyalty feature like a points scheme, encouraging users to point out valid events/issues. This would also require the user to sign in.

Showing the years that have passed since you joined the application creates a sense of community and loyalty. Also, it improves the reliability of the application presenting users having stayed with it for so long.



Waze also implements a **traffic feature**, presenting the impact of the traffic with a colour system, with lighter colours from green to darker colours to green proportionally presenting the intensity of traffic.

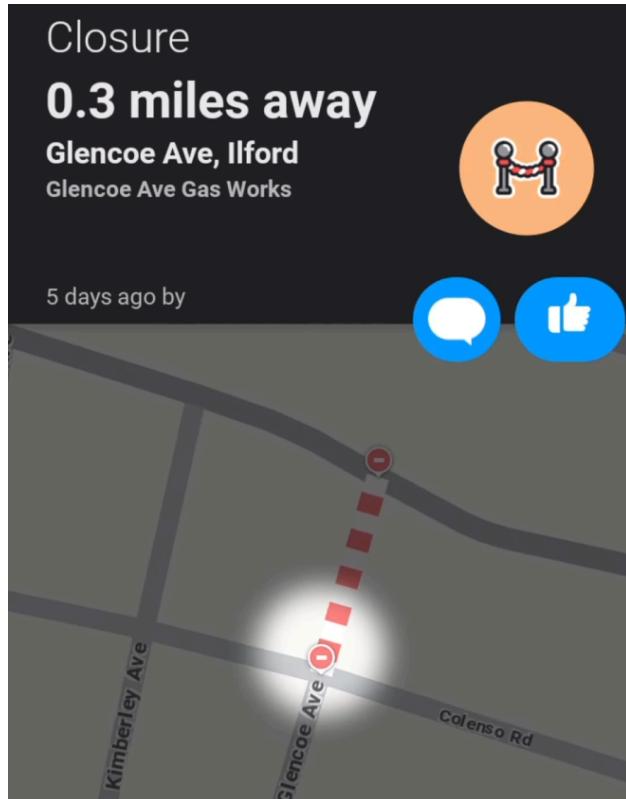
This educates users on which routes to take and the traffic to expect on each of them, improving their time with the app and route.



This screenshot is an example of the feature where users validate and comment on issues/factors that affect their route.

The **blue like button** conjoined with a **comment section** allows users to confirm statements other users provide.

For my application, I could use this with corridors closed for exam periods, construction or maintenance.



This is an example of a **road closure** noted by a user.

Waze uses a **red and white chequered line** to show that the road is inactive from the points it connects to the other roads.

This will make my application effective having a feature like this to inform students which corridors are closed and inactive.

Features to take from Waze

- Reporting feature
- Traffic colour system
- Like and comment function on reports
- XP loyalty points system
- Years since joined
- A red and white chequered line which shows a road is inactive

However, it could be argued that students/users can misuse this feature.

Interview with a Stakeholder

An interview with a stakeholder is an effective way to gauge your product needs and your target audience.

I chose to interview a teacher in the school I go to since they know the school and have lots of experiences which include manoeuvring around the school, hence I can identify and gauge the necessity of my product. I can also ask the interviewee direct questions about the software of my product and how it's going to work.

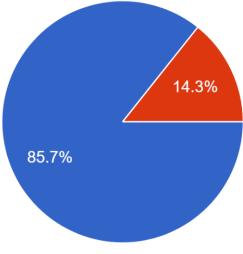
Question:	Justification:	Answer + Analysis:
For how long have you been working for this school?	This will give me an idea of the amount of experience/understanding they have of how the school operates, especially in school navigation.	"I've been working in the school 7 years" This highlights the experience and practise the interviewee has had at the school site.
In school, how often do you find yourself lost or unknown where you are?	This is to identify if there is a greater demand for my solution, in which a greater frequency of uses will be elicited. By exploring the frequency of use cases, I can then cater for the requirements of my product, to handle a greater number of users.	"There is a prominent issue with the lack of correlation between room numbers" The interviewee states that they dislike the lack of common sense within the room numbers in certain locations. They used particular examples within the school location to describe this. Nevertheless, the interviewee did not have a great issue with navigation. Perhaps due to the great experience they hold within the school.
What is the current system to navigate around the school and can you identify the pros and cons of it?	The second question acts to uncover what the current system does well and does not. This way, I can embed what they do well in my map-based	"It's just some kind of experience, isn't it? Sometimes it feels that you might cover lessons, you learn more, in particular classes,

	<p>application and reverse what they don't.</p>	<p>because you've had to find it in the past, there's still plenty of school rooms."</p> <p>The interviewee explains further how experience is the key factor to the current system of navigation, this clearly would not apply to members of the school who lack this experience. They do reiterate logically numbering the rooms.</p>
<p>Do you believe lateness (especially in the early years) is achieved through a lack of navigation support? If so, how does this affect your lessons?</p>	<p>This question aims to see if the current navigation system and lack of support are the reason for lateness. The follow-up question acts to identify if lateness is a major issue for teachers (and students) in lessons.</p>	<p>"Yes, definitely."</p> <p>The interviewee proclaims the great issue, at the start of the year where new students struggle to navigate around the school. Most often even disturb other classes as a result of not knowing where to go. Furthermore, the interviewee expresses the impact of coming in late has on the lesson as disruptive and inconvenient to their teaching.</p>
<p>When moving in and out of school do you think the weather is a factor which influences your decisions?</p>	<p>This is to gauge whether the weather display is necessary, in the final product.</p>	<p>"You want to avoid things like the rain... it's the weather, sunny and warm or walks around the outside, you know it's a nice walk, you know quite peaceful so if it's raining"</p> <p>The interviewee states how rain is a deterrent to them and is a factor which influences the</p>

		route they take. Not only that but sunny, clear weather promotes walking on the outskirts of the school, rather than throughout the building. Hence, presenting the need to add weather information to the application.
What features do you like in navigation apps, if you have used them?	To identify what features I can implement in my solution. This allows my solution to be well-rounded and adapted to meet my user's needs and wants.	The interviewee speaks about the estimated time of arrival most other navigation apps have. Also, the ability to "pan out" and see other places on the map. These are valid suggestions I could implement in SchoolNav.
What features do you not like in a navigation app?	To find out what features I should definitely not include and think about heavily if I do choose to include it.	<p>Interviewee struggled with this question, presenting the sheer quality and helpful nature of the current navigation apps. Hence proving it necessary to research other applications in this field. The interviewee does mention Google Maps, and I have researched this thoroughly and looked at several features, which I could include in my application.</p> <p>However, the interviewee does mention the lack of precision within the GPS tracker where it is "less accurate in a relatively small location". This is an issue I should look at in my application.</p>

Stakeholder Questionnaire - Quantitative data

Using a combination of qualitative and quantitative questions I have collated data from 14 stakeholders of which 12 were fit to provide useful feedback. I primarily got data from students and a few teachers. I have made best use of sections in this questionnaire which make sure the relevant precise questions are going to the right group of people. This data will allow me to make decisions on my application from what kind of directions I want to utilise to whether the navigation of schools is even a problem. My analysis assumes the entire questionnaire and its respondents completed the survey truthfully.

Question:	Justification and analysis:						
<p>Are you a member of a school or organisation which requires moving through a building? 14 responses</p>  <p>A pie chart titled 'Are you a member of a school or organisation which requires moving through a building?'. It shows 14 responses. The chart has two segments: a large blue segment labeled '85.7%' and a smaller red segment labeled '14.3%'. A legend indicates 'Yes' is blue and 'No' is red.</p> <table border="1"><thead><tr><th>Response</th><th>Percentage</th></tr></thead><tbody><tr><td>Yes</td><td>85.7%</td></tr><tr><td>No</td><td>14.3%</td></tr></tbody></table>	Response	Percentage	Yes	85.7%	No	14.3%	<p>Justification: This question filters out the people who can provide useful feedback, (stakeholders) in this application. Only if a person is a part of a school or organisation will they be affected by the problems SchoolNav aims to target.</p> <p>Analysis: Out of 14 responses 12 can provide useful feedback, 2 can not. Showing that overall a great proportion of people are stakeholders in my application.</p>
Response	Percentage						
Yes	85.7%						
No	14.3%						

Describe your feelings when you are new to a school location in a few words?

12 responses

Stressful

Confusing, Lost, Nervous, Anxious

can be overwhelming, scared to get lost/be late to lesson

confused

panicked, stressful, nervous

Anxious, Secluded, Confused

Confused

Navigating through new campus can be terrifying, feeling lost since it's a new environment

I feel lost

Justification:

I included this question to assess my target market and see if they could make use of and benefit from SchoolNav. This question also highlights the uncomfortable nature of being “new to a school location”.

Analysis:

The most common responses were **stressful, confused and nervous**.

This exemplifies the need for my application as it improves and impacts students, teachers and other members of an organisation who arrive and are new to a school location.

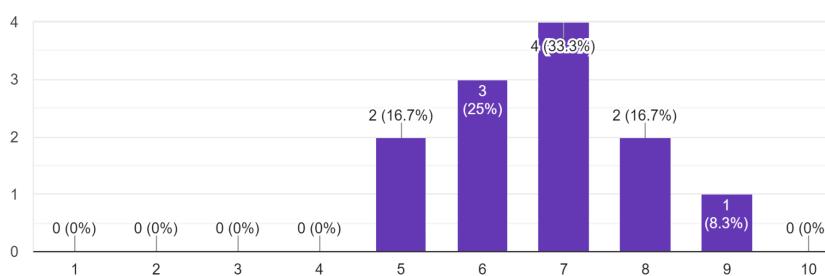
Words such as “*terrifying*” and “*overwhelming*” really emphasise the lack of navigation and support for new arrivals to schools and buildings, hence emphasising the need for my app.

Uncomfortable, distressed

Anxious, Curious, Overwhelmed, Lonely

How often do you find that other people travelling across a corridor are distracting to your lessons?

12 responses



Justification:

To assess and find out if routing people outside of classroom corridors or exam periods will help or benefit the students and/or teachers.

Analysis:

The modal class is at 7 out of a possible 10, showing that this is a major issue people are finding. The lowest value received is a 5, really presenting the distracting force of students/visitors traversing through corridors. It could be argued however that since no one put 10(max), it isn't

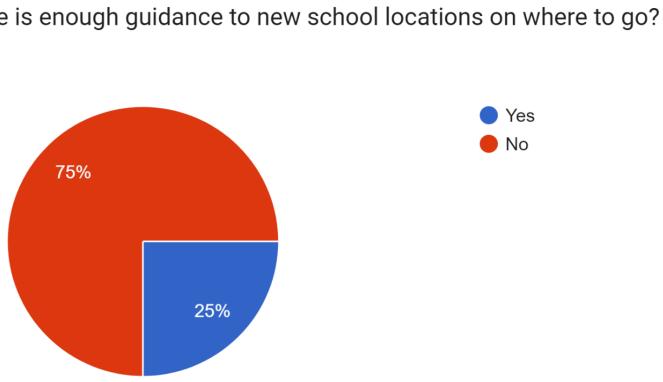
an issue too impacting or large. However since my application can minimise this, I will include this feature of rerouting in my app.

Justification:

This question is a branching question leading to separate sections. The main purpose of this question is to assess if my application is necessary, specifically to provide the right guidance to students visiting new student locations. If generally no, then there is a larger scope for my application, otherwise, less scope.

Analysis:

9 out of 12 individuals decided that there is **not** enough guidance in navigating schools. This presents the drastic gap in the market where students, teachers and others feel lost, anxious and several other words detailed in the second question. Hence, proving my application is necessary to fill in this gap and improve the livelihoods of these school users.



Since you have said there is not enough guidance to navigate through new school locations, to improve the navigation around the school more thoroughly, what can be a feature(s) in an app assisting with this?

9 responses

Route finding - for people with and without disabilities. An app that can say whether certain areas are off limits (exams etc)

a map of the school, people i could ask for help

gps

being able to enter what room you need for the app to tell your where to go

show current location and an Augmented reality on where to go

Quick Search

Interactive map that shows if any exams are taking place

g

Tell me where I am in the school and where my destination is

Justification:

Stemming from those who said “No” to the previous question there isn’t enough guidance to new school locations, majorly, this question aims to find out what example features I can add to my application that my stakeholders deem necessary/assisting.

Analysis:

One particular response is the mention of “*disabilities*”, this is a reference to the accessibility routes that are specific to those disabled. This is a great idea and particularly useful to disabled people. Overall, apart from a single anomaly, users who answered the questionnaire would like “*an interactive map*”, “*current location*” and “*information about where and if any exams are taking place*”, all of these mentioned features I will include in my application.

There is a mention of the use of augmented reality, however since the budget and time of this project does not suffice the time and cost of mapping and creating a 3D model of a school hence, I would neglect this idea.

Since you have said, there is enough guidance to navigate through new school locations, what do you think the school/building does well to assist with navigating?

3 responses

maps throughout the school, people can be asked

Clear signs on where certain buildings are located.

maps placed around the School

Justification:

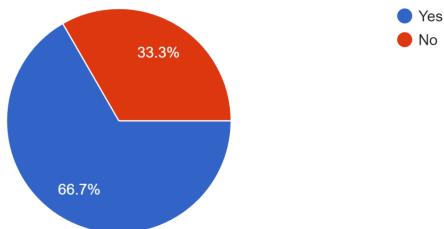
Branching off “Yes” that there is enough guidance. This question was primarily asked to highlight what schools and organisations do well with the matter of navigation and see how I as a developer can enhance and amplify this.

Analysis:

Although only 3 people chose

“No” and hence were given this question, it isn’t viable to make decisions based on this. However, there is an emphasis on the use of maps and clear signs which shows I need to look at implementing a clear and distinct map, which uses relevant signs and room numbers etc.

Do you think the distance/route to a building is often not ideal due to random factors e.g maintenance, construction or exam periods?
12 responses



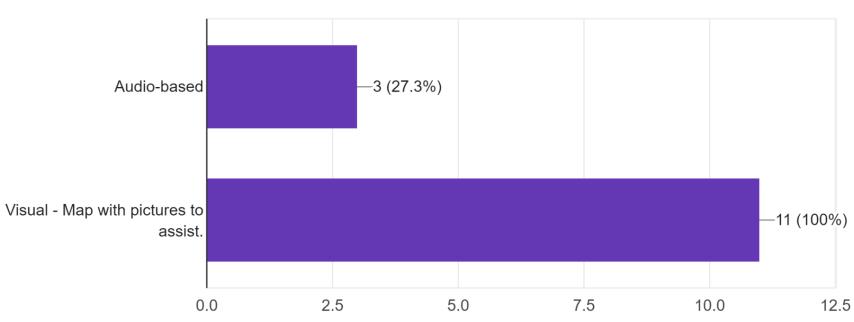
Justification:

This question is used to identify if the need to reroute students walking across exam halls/rooms is a necessary feature. Also, to find if those walking on a route having maintenance or construction would find it beneficial to have a rerouted path.

Analysis:

2/3 of the respondents have said the routes with these random factors are not at all ideal. Since a relatively large number of stakeholders (12) answered this question, it is reliable to base decisions, hence including a feature which factors these random issues and reroutes accordingly is necessary.

What kind of directions would you like when navigating through a school/building?
11 responses



Justification:

I used the checkbox feature, since users may like using both features. This question helps me find out which type of directions I should prioritise, hence being an essential feature and the other a desirable feature.

Analysis:

All of the respondents wanted Visual-map-based directions, with pictures to assist,

alongside, 3 of them wanted audio-based directions alongside this. Since the predominant choice is visual over audial, I will choose the visual map to be an essential feature and the audial directions to be a desirable feature.

What features from other mapping apps do you like the most and why?
11 responses

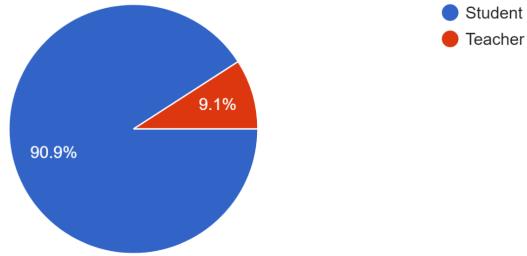
Live routing with a coloured line to follow and the line changes colour to highlight congestion.
live update of my location, street-view
time to travel
i often use google maps to see how long it will take me to get to my destination
Simple to understand interface
Apple maps: Augmented reality
Estimated time of arrival and pre schedule
I can see the different routes to my destination and compare times
u

It tells you the ETA (estimated time of arrival)
Lets you see whats around you

Justification:
This question helps me collect qualitative data which presents what particular features from other similar mapping apps I could utilise in my application.

Analysis:
One particular response is using a “*coloured line*” to show the route, this is a great idea which can prevent any confusion by guiding the user on where to go. A repeated mention of “*ETA*” shows the need and want of it. Again, a mention of “*augmented reality*”, which can be implemented, provided a larger budget and time.

As a current member of a school, which one of the following are you?
11 responses



Justification:
This question is a branching decision question which focuses on the user to answer particular questions relevant to them since a student's experience is not always equal to the teacher's experience. Also, to identify who my application will target majorly.

Analysis:
Primarily my application is directed towards students more than teachers, with a 10:1 ratio.

If you are undergoing any exams, describe how stressed you would feel if people make noise outside the hall/centre?

11 responses

Fairly annoyed

quite stressed as i tend to lose focus

very

extremely - it is distracting

Not much tbh, depends on the importance of the exam

Angry ,enraged ,disgusted , spartan rage

Very stressed and distracted

I would not be a happy boy. The stress would be a bit toooooo much for me because I wouldn't concentrate.

h

Justification:

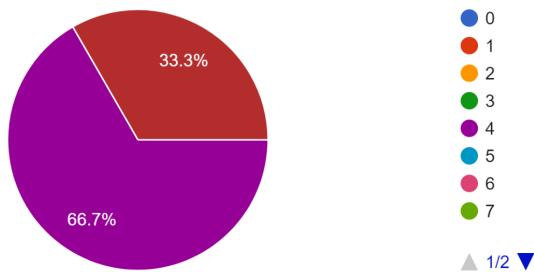
Being a question only targeted towards students, is to assess if it's necessary to reroute students walking past or through the corridors of exam halls/rooms. To also see if my application can overall reduce the stress on the students using my application.

Analysis:

Overall the results are negative. The students who answered this question find it very distressing and uncomfortable when students pass across an exam hall/room. With a few anomalous results, this question results are reliable, hence I would include a feature which reroutes students passing these halls at exam times.

How long into a lesson do you have to wait before actually starting the lesson?

3 responses



9

10

More than 10 minutes

Justification:

This particular question was only asked to teachers, branched from the previous question on status as a member of a school. I used a drop-down type question to assess the effects of students arriving late to lessons and even visitors disturbing students by passing by in the corridors.

Analysis:

The most common response is 4 mins but $\frac{1}{2}$ is saying 8 minutes worth of time is spent getting ready. That is over 10% of a 1-hour lesson wasted on students arriving late. The need for my application through this data is apparent. But, this is only based on 3 teachers' worth of data, hence perhaps not reliable to base decisions.

Specifying the Proposed Computational Solution

<u>Essential Features</u>	<u>Desirable Features</u>
<input type="checkbox"/> Map of school: this should be up-to-date and a clear representation of the school. The appropriate room numbers should also be enlisted.	<input type="checkbox"/> Pictures associated with each room: It needs to have pictures associated with each room, which can be accessed when tapped on but also when a route points to that room.
<input type="checkbox"/> Login Page: this should be the opening page to the application, ensuring saved information in settings so appropriate school maps can also be accessed. Further, saved searches of particular rooms or places.	<input type="checkbox"/> Audio-based directions: making use of audial directions e.g. “ <i>Take the next left</i> ” can be useful and has been deemed necessary in the questionnaire conducted above, hence desirable.
<input type="checkbox"/> Home Page: this should show the option to “ <i>Navigate school</i> ” and to also access the “ <i>Settings</i> ” page.	<input type="checkbox"/> Compass: users can use the visual compass to see which direction they are going.

<p><input type="checkbox"/> Settings Page: this allows users to specify their school, if they require the use of accessibility routes, their particular subjects and appropriate room numbers, form room and timetable information. Users can also choose to log out here.</p>	<p><input type="checkbox"/> Common places tab: this shows the common places users go, allowing quick, single-tap access. This is desirable.</p>
<p><input type="checkbox"/> Pointer to show where the user is: this is vital since it has been mentioned several times by the stakeholders in this application and is an important component in similar software around the market.</p>	<p><input type="checkbox"/> Weather for the area: this shows the weather for the school area for that day, and other days once pressed. A visual representation of the weather, alongside the degrees for the temperature, can be utilised.</p>
<p><input type="checkbox"/> Reporting feature: primarily the information for the application is going to be derived from users reporting incidents and activity around the school, hence vital.</p>	<p><input type="checkbox"/> 3D Mode: presents the 3D scale-up of the school map. Shows the height of rooms and buildings, which can be significant with buildings with more than one floor.</p>
	<p><input type="checkbox"/> Traffic colour system: this can show the traffic density of certain corridors and represent them visually by use of colours.</p>
	<p><input type="checkbox"/> XP points system: points can be generated by users who report factors which receive an amount of likes and/or comments. This can improve the validity of the reports.</p>
	<p><input type="checkbox"/> Red and white chequered line: this is to show a corridor in particular is blocked/inactive.</p>

Limitations

The most significant limitation is that this application needs to be on mobile since users have to be able to move around with the device and application, hence a desktop app isn't a viable option used primarily by users.

Another limitation is that users would need an internet connection and a GPS signal to track their whereabouts and map the routes appropriately, per the reported issues e.g. blocked corridors. This can be hard to get from users since most want you to know where you are going. In buildings, internet connection can be awful, hence a limitation to my application.

Another limitation is GPS, most users' GPS signal is not too precise to work in buildings and around them. Hence, I would use user input and the weak GPS signal to verify where they are going and checkpoints(user clicks on when reached) if the signal is too weak.

This application is designed for new users, although new students do come every year. It is not as significant as in Year 7, where the amount of lost students increases rapidly. Most often these students do not own these devices which can run the application, hence inapplicable to these users. Hence, a limitation being, those without a functional phone wouldn't be able to use my application.

Hardware Requirements

Hardware Requirements	Justification:
Minimum primary memory at least 4GB RAM	This is recommended for Javascript and react.native will run properly alongside the other functions on the device, so the routes will be easily reaccessed after opening other apps on the mobile device or desktop.
Hard disc: up to 10 GB of free space may be required	To allow Javascript and react.native to run optimally and to download its files. Further, this is because the app would need to download the maps and their associated details and needs this much space for it. So the related tools of react.native can run smoothly.
Processor: x86 or x64, at least single-core and at least 1 GHz	This is to allow the Javascript to run optimally. To run the functions of mapping the routes and logic.
Mobile device	This is to present the maps and directions and also allow user input to manipulate the

	routes and options.
GPS Signal	A GPS Satellite Connection is needed to verify where the user is heading.
Magnetometer	This is commonly found in mobile devices to allow a compass-like direction to show which direction the phone is pointing and its orientation.

Software Requirements

Software Requirements	Justification:
OS: iOS 13.4 and Android 6.0 (API 23) or newer	This level of operating system is needed to run the react.native program on the mobile device.
Javascript	This is the programming language the application will be implemented in.
Network	The location of nearby Wi-Fi networks and mobile towers helps the device know where you are.

Success Criteria

No.	<u>Requirements' Description</u>	<u>Justification</u>	<u>How do we measure success?</u>
SC.01	The application should have a smooth and responsive user interface (UI) system.	Only if the UI is responsive can users fully use the application and find the directions to where they want to go, adjust the settings and several other features all depend on receiving user input and making use of it, hence needs to be responsive. This allows users to smoothly access and route the paths they need to along as fast and convenient as possible. This is a major factor since the application aims to reduce time traversing, but by not having a responsive UI	<ul style="list-style-type: none"> - Manipulated and traversed - Easy to go to different pages of the application - Different processes of the application for e.g changing Maps, able to be school name, should be fast and easy to perform

		system the application wastes further time doing its job. Hence should be fast, smooth and responsive to user input.	
SC.02	The map should be accurate, easy to understand and reflective of the building	<p>The map is the most fundamental aspect of my application, with most of my primary research showing how a clear and distinct map is vital to creating a successful navigation app. Even though researching similar applications like Google Maps and Waze, they have an apparent and easy-to-understand map, with little to no learning curve.</p> <p>Users should be able to know where they are in the building by looking at the map and seeing which direction they are going along.</p>	<ul style="list-style-type: none"> - Room numbers are shown with each room - Common places highlighted and named on the map - Inactive corridors marked with red + white chequered line - Pictures should be shown at each junction or significant room to help direct users.
SC.03	Settings should allow updating and meet all different user needs.	This is because in settings, there will be an option to choose if accessible routes are necessary for the user or not. In addition, the particular subjects/timetable of the user, alongside the form room. These are common places which by using the settings I can input the locations and where the user should be at each certain time. Allowing faster route finding, which also acts as a reminder for the student of where to go.	<ul style="list-style-type: none"> - Settings are allowed to be changed and stored with the user's account - Settings should be acted upon by the routes provided and the maps for the users. E.g Ramps and lifts should be highlighted on the map for users.
SC.04	Estimated Time of Arrival should	This will be shown in conjunction with each route mapping. This is to give a rough idea to the user when	<ul style="list-style-type: none"> - ETA should be tested and timed, accurately giving the time you would arrive at your

	be accurate and realistic	they would arrive at their location.	destination.
SC.05	The application must be responsive to certain requirements/ details the user may have.	<p>Majorly disabled users of this application would need routes to involve a lift or ramp system. Steps have to be avoided on routes and pointed out on the maps of these users.</p> <p>The particular school of the user should be specified in the settings page and the map of the specific school should be accordingly presented.</p>	<ul style="list-style-type: none"> - After specifying the Disabled Accessibility routes, users should be presented with routes with lifts/ramps and not any steps. - If the school they have inputted as the current school is not mapped/in the application it should show an error message.
SC.06	The Login/Sign-up page should be the first screen for the users of the application, but the user should have the ability to use the app as a guest and should have appropriate validation.	<p>This allows new users to log into their account and access their saved settings like what school they go to, their form room and if they need disabled access. Also, to allow saved searches in the search bar.</p> <p>Users should have the option to sign in as a guest since it would faster access to the app and users can then sign in later on to save their details. As a guest, users can also view and enjoy traversing the map of schools allowing them to explore their options in future schools.</p>	<ul style="list-style-type: none"> - Should successfully create a new account. - Should set certain settings, log out and then log back in to see if the settings are saved. - After logging in search for some specific places logout and log back in to see if those searches are still present. - The guest mode should be tested to see if I can easily traverse and route paths without the features of saved searches or settings.
SC.07	Reporting feature - to	This is to allow users to report certain factors which	<ul style="list-style-type: none"> - Test if the report function is responsive

	highlight issues and factors that exist on the map and present them to other users	minimises the work that needs to be done by maintenance and easily provides information to be passed on to other users.	<ul style="list-style-type: none"> - and allows an array of factors to be reported - If reported issues by another user are visible to other users
SC.08	The weather shown in the app should be accurate and have the appropriate image to represent the climate for that day and the following days.	This is particularly handy for students since they would have to traverse an entire day in that climate and especially if they live further away, it will be harder to present the weather at the school. Hence appropriately students/users can wear suitable clothing and bring an umbrella if necessary.	<ul style="list-style-type: none"> - Screenshot weather shown by app, and weather shown through other reliable weather apps. - Screenshot the weather shown for the following days, and see if it matches.
SC.09	Common places tabs where it presents the ability to access the common places of the map.	This is a desirable feature for my application since it simplifies user's having to search for these common places such as toilets. The search can be mainly used for niche examples like rooms. This simplifies the user experience and makes the app more comfortable to use.	<ul style="list-style-type: none"> - After clicking on a place on the common places tab, the user is routed to that place from the start point. - There should be a wide range of places which meet the needs of all students. E.g. toilets, main hall or reception.
SC.10	Compass feature/Recent er feature and map adapting to user orienting the phone.	This is to orient users. Reduce the confusion and promote clarity on where users are going. Compasses have niche case uses in camping or other treks, hence can be utilised then, perhaps on a school trip. But majorly this feature is to meet all the needs of my stakeholders and to improve	<ul style="list-style-type: none"> - As I move the phone around the visual point representing myself, turn accordingly and face the appropriate buildings. - If I press the compass button, I should get the bearing and also

		the clarity of the visual representation of the user.	the primary direction I am facing.
SC.11	Traffic colour system for corridors	This is to inform users of the state of certain corridors, it can be slow and cramped to traverse in packed corridors, hence this feature can help users inform decisions on where to go and my algorithm can also use this information to reroute certain paths with high traffic.	<ul style="list-style-type: none"> - Corridors with a larger number of users on it should be presented with a red colour rather than a corridor with no traffic as green or uncoloured. Mild traffic is offered with amber. - The number of users for each threshold is reasonable.
SC.12	Response to if user is heading in wrong direction	If there is clear indication, that the user is heading in the wrong way. The user can be led to go more in the incorrect direction. This leads to more confusion and more anxiety which contradicts the purpose of this solution entirely.	<ul style="list-style-type: none"> - Red alert or vibration is produced if the user heads in the opposing direction on a set route.
SC.13	Product can handle multiple responses at the same time.	This is because, in schools, users/students utilise and make use of the app around the same time e.g at the start of the day or after break. Hence, my application needs to be able to handle different user requests simultaneously.	<ul style="list-style-type: none"> - Testing with different devices using the system at the same time and seeing if the application works in sync with other users.
SC.14	The maps should update according to the user commenting feature.	Factoring important changes to the school map is a vital component. Since the route may include corridors which are inactive. Most often corridors or paths which exams take place, shouldn't be traversed along, hence should those particular areas should be updated for the application to input and base routes off of..	<ul style="list-style-type: none"> - Comment as a user, corridor is inactive or that an exam session is active in that vicinity. This will result in a change within the map, a visual clue to the inactivity or certain event, also routes will be changed and altered depending on these comments.

			<ul style="list-style-type: none"> - Ability to report and see your report affect routes
SC.15	Logo Display should be prominent and visible to users, indicating the app and its purpose.	It allows the user to associate a brand with the app. Allowing easy access to the application, even in the midst of several other apps.	<ul style="list-style-type: none"> - A clear logo at the top of screen and when the user logs on.
SC.16	Aesthetically pleasing and functional design to the user which promotes the user experience.	For SchoolNav, an aesthetically pleasing and functional design is essential to enhance the user experience. A visually appealing interface keeps users engaged while ensuring that features are easy to use and accessible, promoting a smooth and enjoyable navigation process.	<ul style="list-style-type: none"> - Colourful and appropriate theme, which compliments the application. - The information should be legible and easy to understand.
SC.17	Register screen should function and take in username and password and have appropriate validation.	<p>This is to ensure that the application's account details and login verification is appropriate and the inputs are valid. This is to ensure the program works as intended.</p> <p>Ensures that only valid usernames and passwords are accepted, preventing errors and security risks. Proper validation improves data integrity and enhances user security.</p>	<ul style="list-style-type: none"> - Users can successfully register with a valid username and password. - Invalid inputs (e.g., empty fields, weak passwords) trigger appropriate error messages. - Duplicate usernames are not allowed, and users are prompted to choose a different one. - Passwords meet complexity requirements (e.g., minimum length,

			special characters).
SC.18	The username and password fields should work from the saved register screen.	Users should be able to log in seamlessly using their registered credentials. A functional authentication system ensures secure access and prevents unauthorized use.	<ul style="list-style-type: none"> - Users can successfully log in with their registered credentials without errors or delays. - Incorrect credentials trigger appropriate error messages (e.g., "Invalid username or password"). - Multiple users can log in and out without issues.
SC.19	The different pages should be easy to navigate through.	Users should be able to move between different sections of the app effortlessly. A well-structured navigation system ensures a smooth user experience, allowing quick access to key features.	<ul style="list-style-type: none"> - Users can switch between pages without confusion or unnecessary steps. - Navigation buttons and menus function correctly on different devices. - Page transitions are smooth and responsive, with minimal load time. - User feedback (e.g., usability testing) confirms that navigation is intuitive.
SC.20	The route should be clear and presented on the Map.	Users need to visually see their selected route to follow it correctly. A well-displayed route prevents confusion and enhances the app's usability for efficient navigation.	<ul style="list-style-type: none"> - The route is displayed accurately on the map with a clear path, markers, and visual indicators. - Users can distinguish different routes using colors or labels.

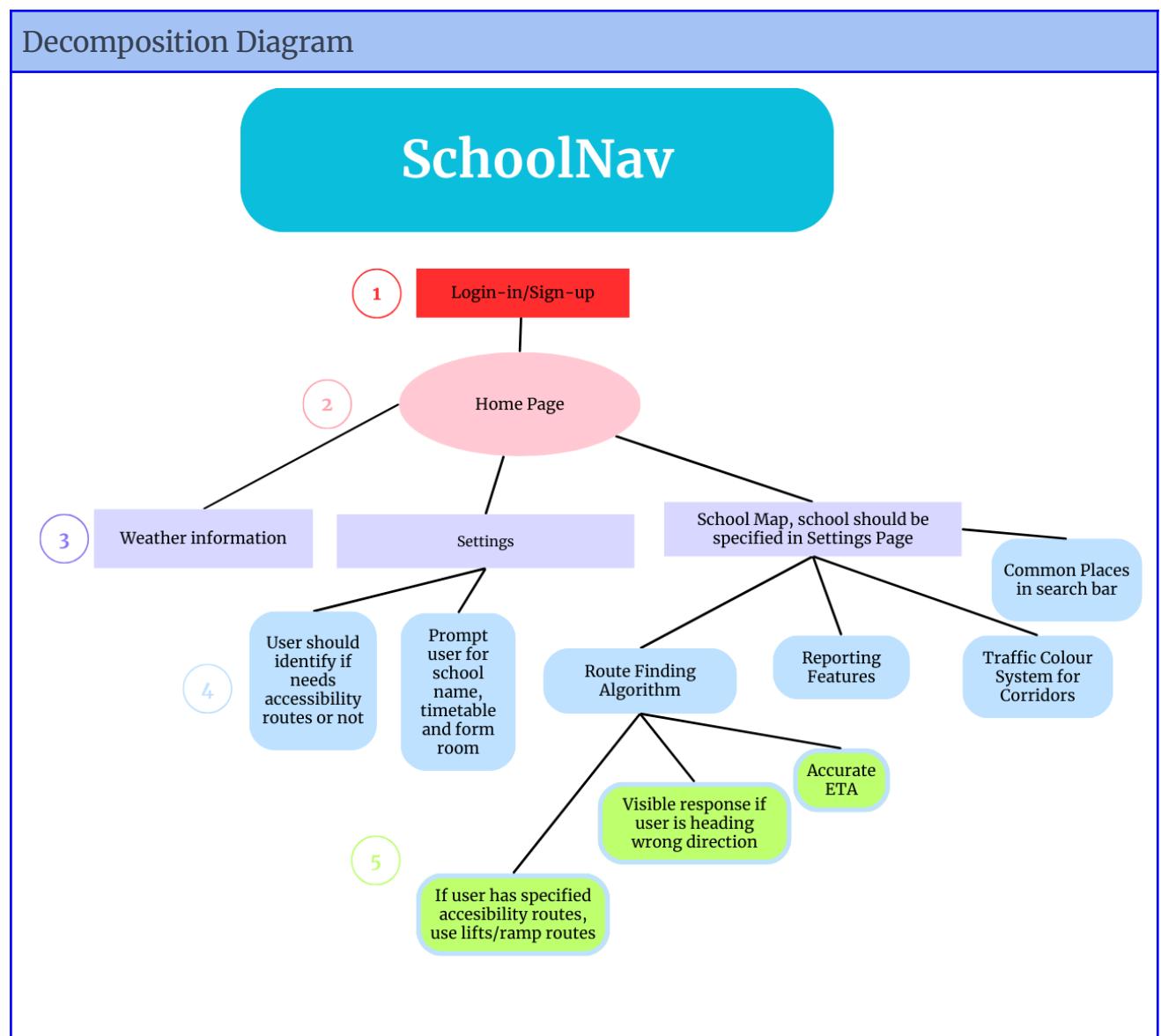
			<ul style="list-style-type: none"> - Zooming and panning do not distort or hide route information. - Users can successfully follow the route without ambiguity.
SC.21	The route should take the shortest distance and be accurate and precise.	<p>The app should calculate the most efficient path to save time for users. Accurate routing ensures reliability and effectiveness in navigation.</p> <p>For SchoolNav, routes must be accurate, precise, and take the shortest distance to ensure efficient navigation. Incorrect or inefficient paths could confuse users and waste time, especially in a busy school setting. Optimizing routing algorithms enhances reliability, helping students reach destinations quickly and easily.</p>	<ul style="list-style-type: none"> - The shortest path is consistently chosen, and users reach their destinations efficiently without unnecessary detours. - The calculated route matches real-world paths and avoids obstacles. - Time estimates align closely with actual travel times. - User testing confirms the route is logical and efficient.

Design

Decompose the Problem:

By decomposing the problem into smaller, manageable parts, I aim to simplify my work on SchoolNav, helping to reduce the project timeline. With a modular-based design, I can focus on individual tasks to improve workflow, eventually combining these modules to form the core components of the project.

The decomposition step makes the project easier to track and manage, also testing smaller chunks more regularly helps limit errors within the code.



Abstraction was central to designing my decomposition diagram, allowing me to simplify the tasks that will ultimately build my final project. I used layers with colour coding to distinguish the project timeline and emphasise each task's significance.

The first task is creating a Login/Sign-up page, enabling users to save searches and access specific schools, which they can input on the Settings page. SchoolNav's Home page will feature three main components: Weather information, the Settings page, and the School Map. Weather information will appear first, helping users plan what to wear and which route to take. The settings page will offer customization options, prompting users to enter their school name (for map accuracy), timetable (to suggest routes and locations), and Form Room (to highlight the location students visit daily after lunch). It will also ask if accessibility routes are needed.

The School Map has several key features: a route-finding algorithm, reporting tools, a traffic colour system, and a search bar for common places. The route-finding algorithm will offer accurate ETAs, notify users if they're heading in the wrong direction, and provide accessible routes, using lifts and ramps where specified. Reporting features will allow users to make reports visible on separate accounts. The traffic colour system will indicate congestion in specific corridors, and the search bar will help users find locations and access common places.

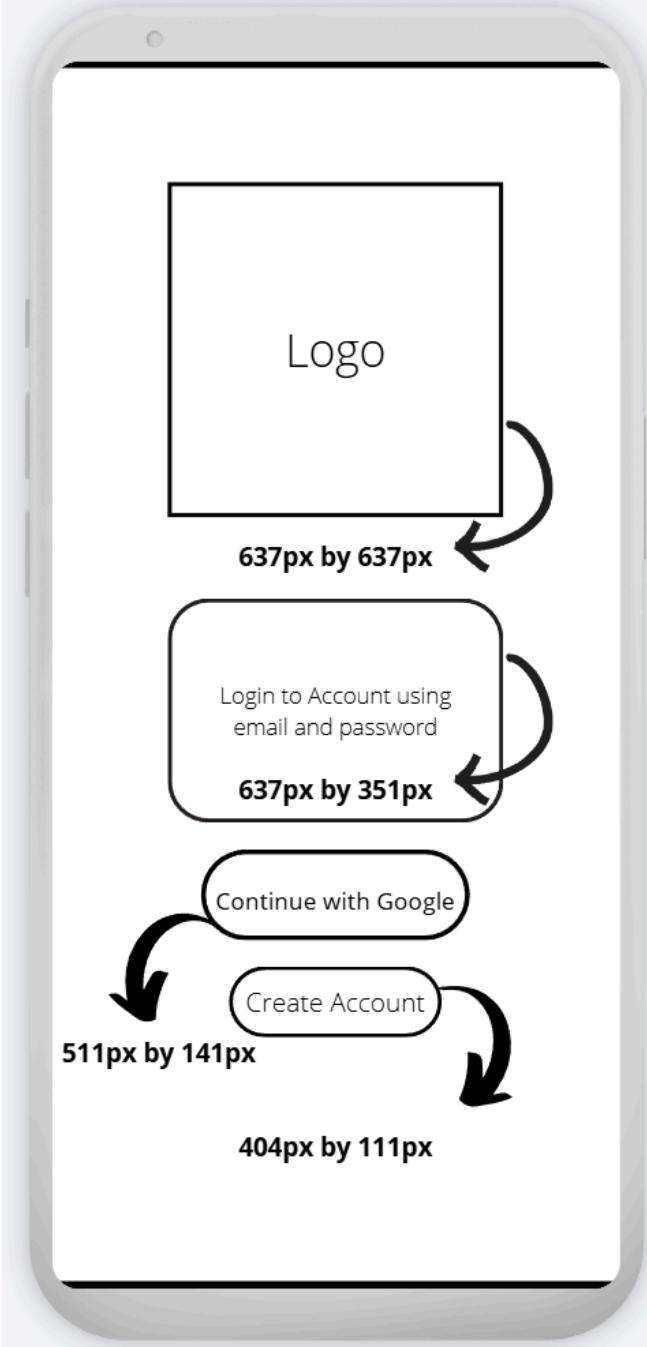
User Interface Design

The GUI (Graphical User Interface) is a significant part of any project. In particular, for SchoolNav, my GUI should be welcoming and easy to use, with a short learning curve. This is to account for how students/other users will use the app on the move, hence should be straightforward to use, to prevent unnecessary confusion and hassle.

Users can go to the School Map, Weather Information or Settings page and view and input more specific details. The colour scheme should be sky blue to convey a calm and controlled environment to mimic the desired effect of the application.

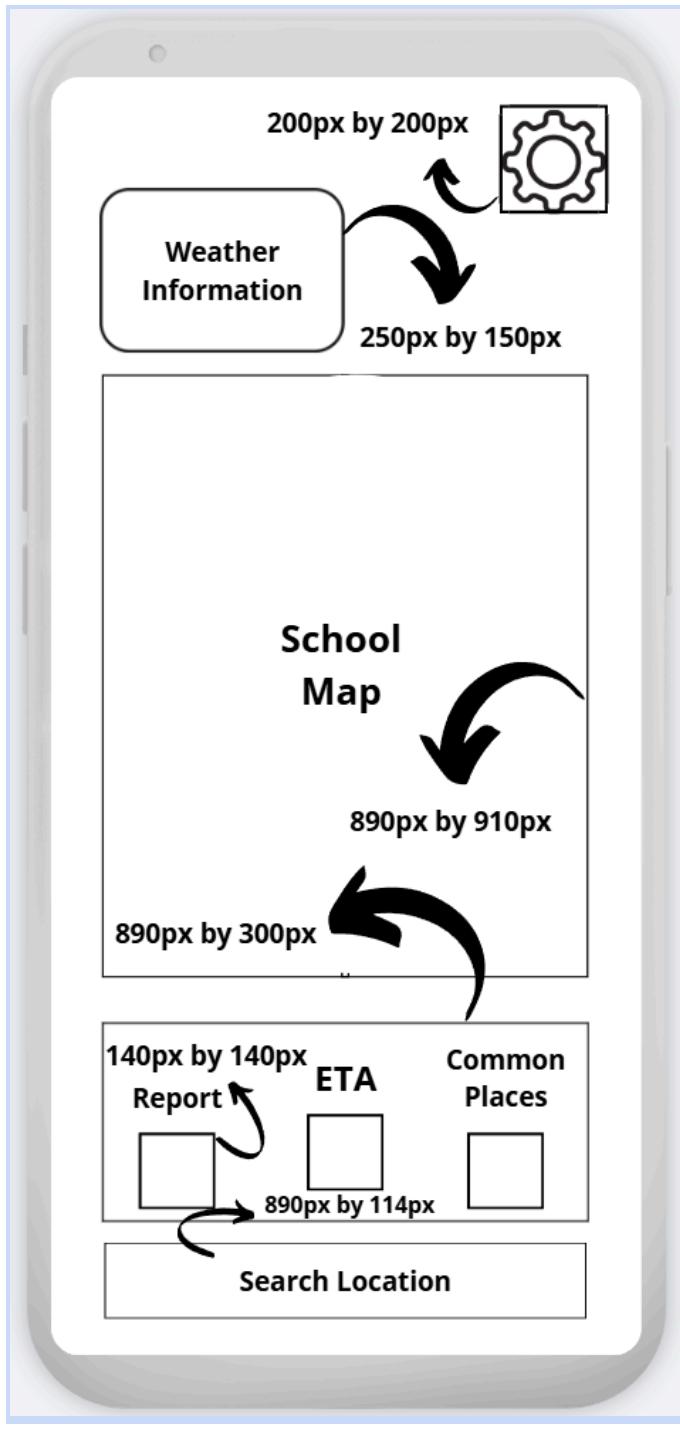
Using my analysis research, I am going to use Google Maps software and School Locator as a basis for my application.

Using wireframes to design my GUI helps to show key features and layouts clearly. I will then number aspects and explain them further in detail.

<u>Pages</u>	<u>Justification and description:</u>
<p data-bbox="446 384 637 428"><u>Login-Page</u></p>  <p>The image shows a mobile phone screen with a light gray gradient background. At the top is a large white rectangular area labeled "Logo". Below it is a rounded rectangular button labeled "Login to Account using email and password". At the bottom left is a smaller rounded rectangular button labeled "Continue with Google", and at the bottom right is another smaller rounded rectangular button labeled "Create Account". Arrows point from each button to its dimensions: the logo is 637px by 637px, the login button is 637px by 351px, the Google button is 511px by 141px, and the create account button is 404px by 111px.</p>	<p>1080 × 1920 Pixel Display for phones with an Aspect ratio of 16:9</p> <p>All text is in Merriweather Font (Blue).</p> <p>The background image should be a blue gradient colour.</p> <p>The logo on the login page should be prominent, helping users associate the app with the brand while maintaining a professional look that feels welcoming. Adding side padding around the logo creates a sense of simplicity and softness, enhancing user comfort. The ideal dimensions are 244x244.</p> <p>I used soft-rounded corners for the rectangular buttons for logging in this is to boost comfortability and encourage users to login. It gives a soft touch, perhaps giving the impression of security and hospitality as well.</p> <p>The Login/Sign-up page is the first screen, fulfilling SC.06</p> <p>User's should be able to use Google to sign-in to allow easy log-ins and to prevent having issues if forgotten passwords etc.</p> <p>I used a smaller-sized button to highlight the importance. Logging in via email and password is safest, hence the largest button and creating an account is not a frequent button pressed, hence smaller with the pixel dimensions of 155px by 52px.</p>

<p><u>Logo</u></p> 	<p>Logo size - 637px by 637px</p> <p>The SchoolNav app logo features a recognizable location icon, directly symbolising the app's core purpose of providing navigation assistance within schools. The calming blue colour scheme not only reinforces the logo's inviting tone but also promotes a sense of ease and accessibility for users, enhancing their overall experience with the app and fulfilling SC.15.</p>
<p><u>Background Image</u></p> 	<p>Background Image Size: 1080px by 1920px</p> <p>A simple, blue background image designed to provide a clean, uncluttered aesthetic for SchoolNav, fulfilling SC.16. This approach supports the app's focus on ease of navigation, with soft gradients that create a professional, calm atmosphere, keeping attention on the app's logo and content.</p> <p>The design is visually appealing with its gradients and will attract users in schools. It is a professional user interface with a calm, blue gradient.</p>

Main Menu



The settings icon should be small and less of an obstruction to the main content. Hence use a 200px by 200px icon in the top right hand corner.

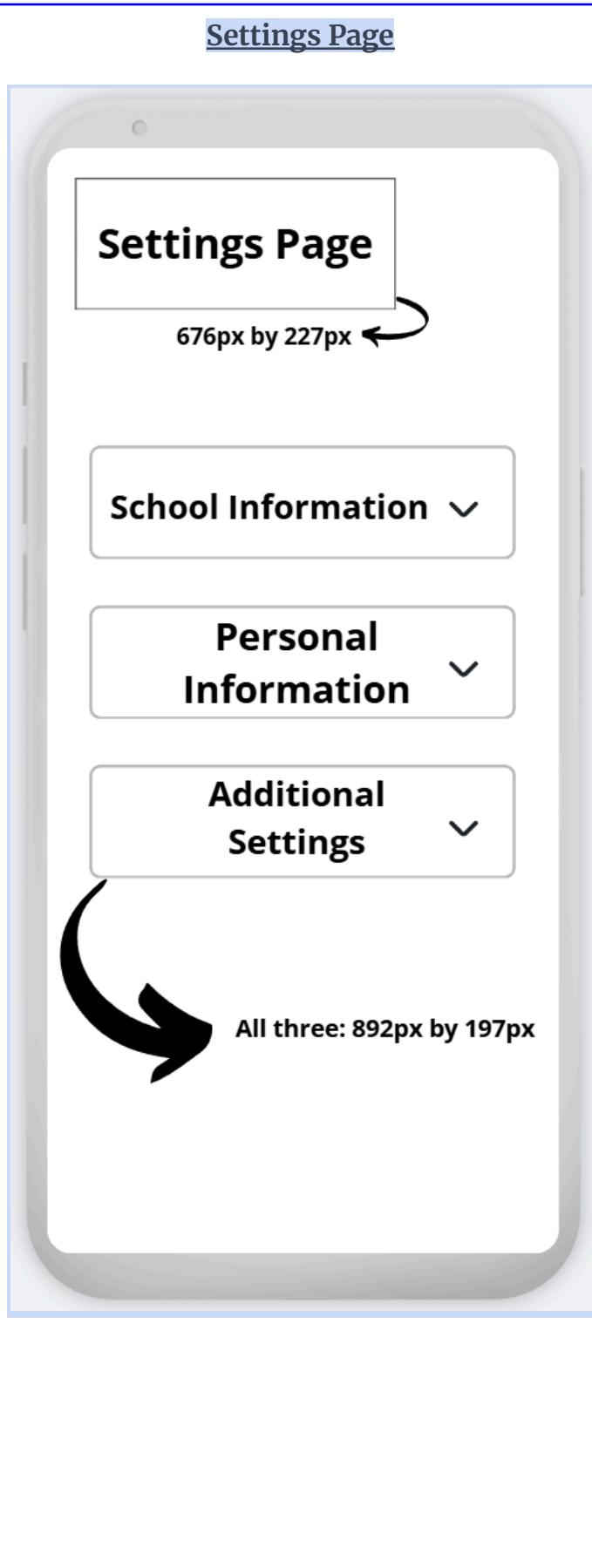
The weather information should be a block, with a visual icon according to temperature and details for the day. If clicked, it should pop-up a detailed weather information guide.

The School Map should be the main content and stand out in the Main Menu. That is why it has the largest space on the display with a 890 by 910 aspect ratio. There is padding on the sides to create space for the user.

At the bottom, we see 4 key buttons, the Report button, ETA display, Common Places button and the Search Location feature button. The first three buttons have equal size of 140px by 140px. It is relatively small since it is a feature less frequently used and by reducing the size, it would add to the simplicity of the menu.

Fulfilling SC.07, SC.08 and SC.09.

The search Location feature should bring up a different pop-up, which accesses a directory of all the locations, rooms, libraries etc. Fitted with an auto-spelling algorithm to help assess exactly what the user wants.

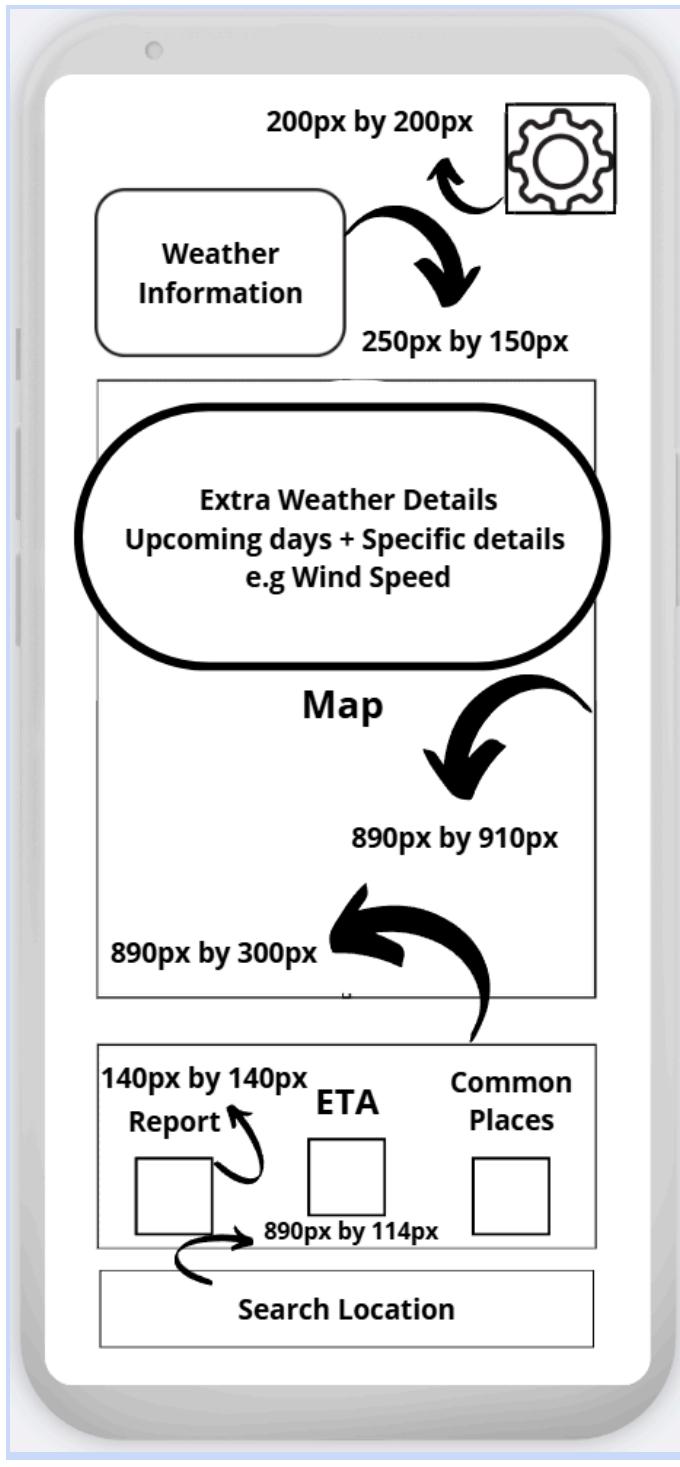


The settings button is to be white on a light blue background, with font colour Dark Blue.

SchoolNav's settings page is designed to be intuitive and accessible, fulfilling SC.03, with the settings icon positioned at the top of the screen to clearly indicate the current page, making navigation seamless. The page's layout emphasises simplicity, with three primary dropdown buttons: School Information, Personal Information, and Additional Settings, each sized at 892px by 197px. These buttons are spaced for easy differentiation and access, reducing clutter and enhancing readability.

Each button reveals targeted options, School Information may include details like school hours, location maps, and emergency contacts; Personal Information might offer customization options for the user's profile, notification preferences, and privacy settings; Additional Settings can provide access to system preferences, app themes, and language choices. This structured layout ensures that users can swiftly locate and adjust their settings, making the app both functional and user-friendly.

Weather Information Pop-Up

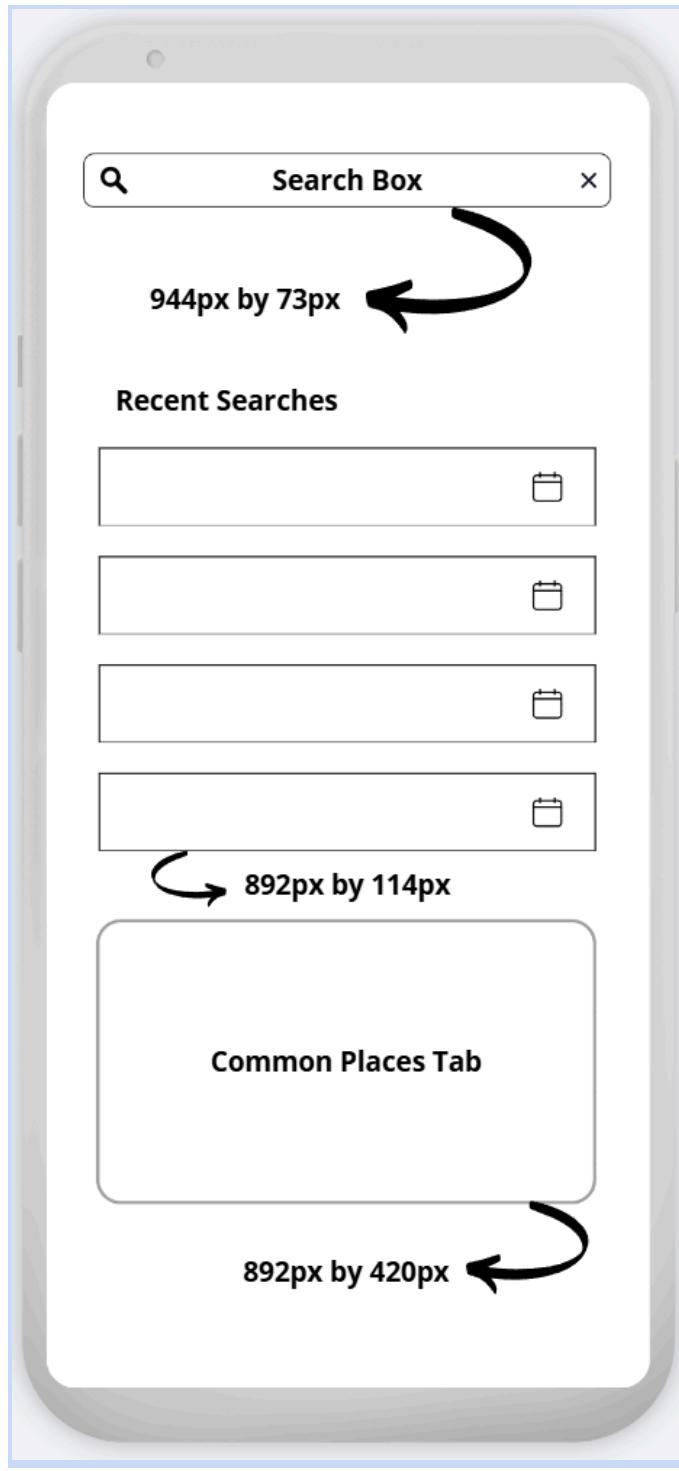


The pop-up weather detail display is of size: 957px by 392px

The pop-up should be presented if the compact weather information button is pressed. The details should consist of additional weather details and forecasts for the next days, further fulfilling SC.08.

SchoolNav could enhance user experience by displaying essential weather details like current temperature, conditions, and a brief forecast for the day and week. Key additions like severe weather alerts, air quality index, and UV levels can help users stay prepared, especially for health and safety. Wind speed, direction, and humidity also inform users on comfort levels, while a simple interface for these insights would keep the focus on navigation.

Search Locations Pop-Up



The box colours will be white on a light blue background, with a font colour of Dark Blue.

All four boxes of the recent searches are of the same size of 892px by 114px. 4 is the maximum number of recent searches to allow simplicity within the app and reduce clutter and overloaded information.

The common places tab is located at the bottom, it will have the common locations stored and saved by the user.

There is a separate interface for the common places tab, which can be accessed by a specific button on the main menu.

Settings Page: School Information

Drop-Down:

Settings Page

676px by 227px ↙

School Information ↘

All three: 704px by 154px ↗

School Name ↘

Form Room ↘

Time-table
Information ↘

892px by 782px ↗

Personal Information ↘

The drop down options for adding your school information details are in a list format. You can see items as you scroll down the list. But since I am only using three button options it can be avoided as it wouldn't be much use.

Fulfils components of **SC.03**.

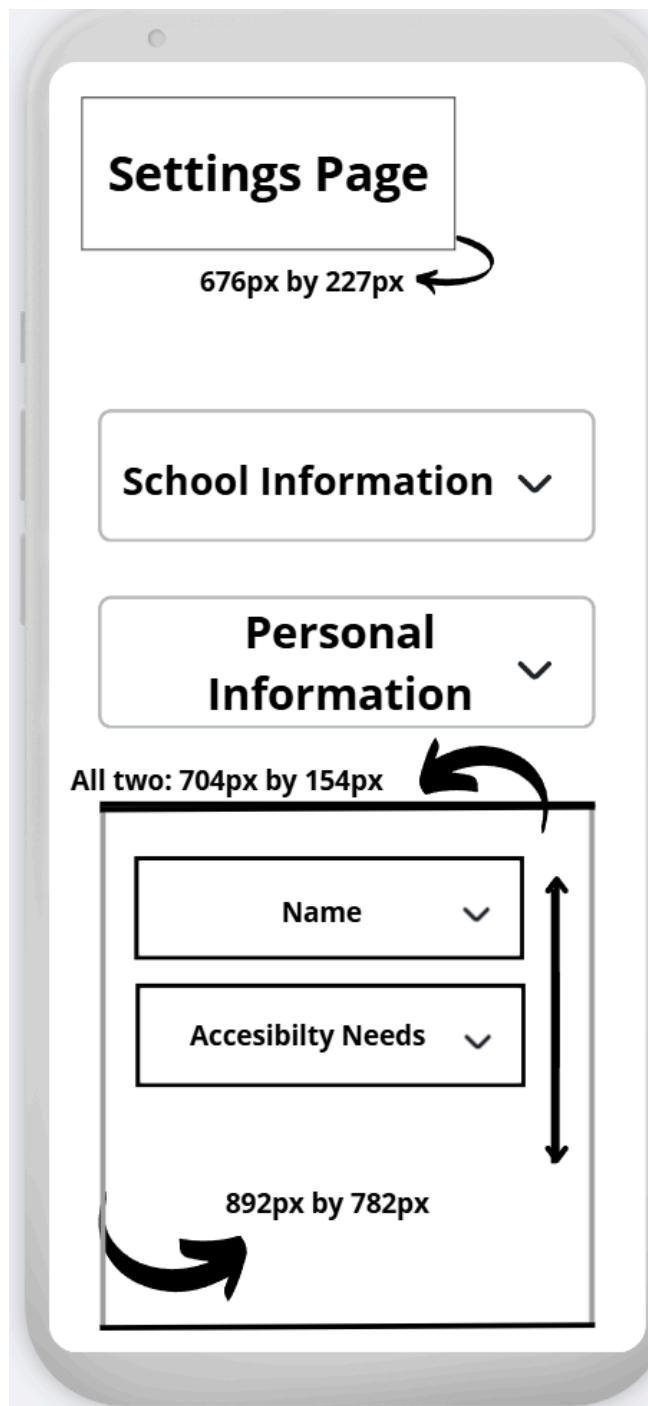
Each button will allow users to enter the details in a drop down menu.

Each timetable has an associated code which can be used to upload timetable information.

The buttons are of size 892px by 782px and are of colour white with a font colour of dark blue, maintaining this theme.

Settings Page: Personal Information

Drop-Down:



SchoolNav's settings page enhances personalization through the **Personal Information** dropdown, which contains two distinct buttons: **Name** and **Accessibility Needs**. Both buttons are styled with a white background and dark blue text for optimal readability and a cohesive design.

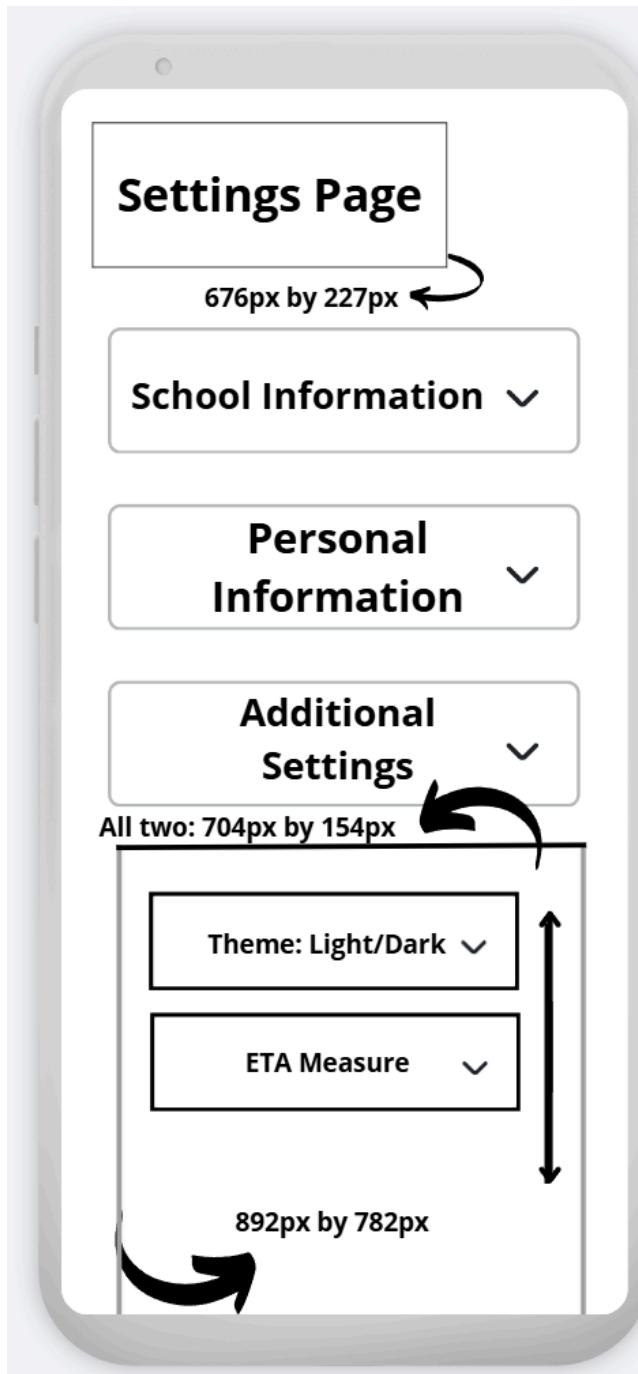
The **Name** button opens a dropdown field, allowing users to input their name, which personalises certain app functions, creating a more customised experience.

Fulfils components of SC.03.

The **Accessibility Needs** button functions as a toggle, enabling users to specify whether they require an accessibility route within the app, ensuring SchoolNav caters to diverse needs and provides an inclusive experience for all users.

Settings Page: Additional Settings

Drop-Down:



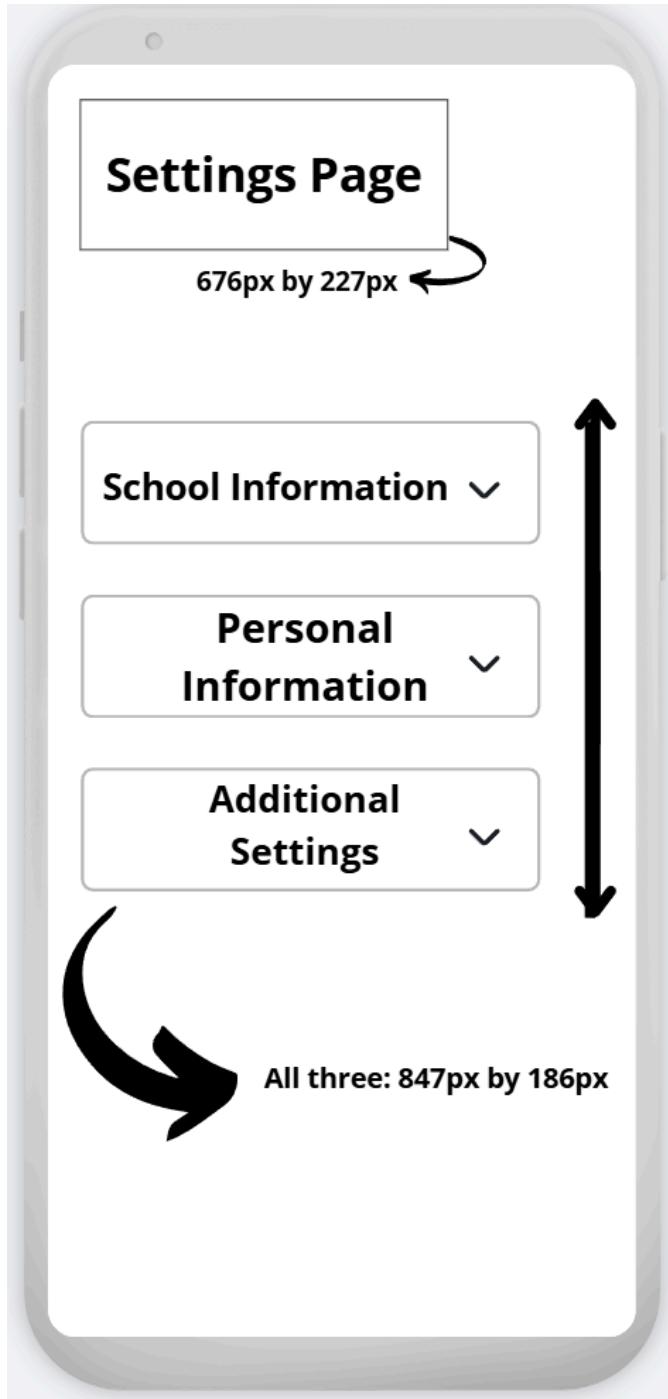
The Additional Settings dropdown in SchoolNav offers two streamlined options: a toggle to switch between light and dark modes, and an ETA measurement adjuster.

The light/dark mode option enables users to select their preferred visual theme, improving usability in various lighting conditions.

Fulfils components of SC.03.

The ETA measure setting allows adjustments to the estimated time of arrival calculations for added customization. To keep the project on schedule, only these essential settings have been included, balancing functionality and simplicity.

Settings Page – Iteration 2



The updated settings page in SchoolNav now features a dark blue scroll slider, making it convenient for users to navigate through options even when all dropdowns are expanded.

The button sizes have been adjusted from 892px by 197px to 847px by 186px to allow space for the slider on the right, preserving the page's clean, accessible design.

This change enhances user experience by providing seamless access to settings, while the dark blue slider aligns with the app's colour scheme, maintaining a cohesive look.

Iteration aims to meet and fulfil SC.16.

Flowcharts and Pseudocode

Sign-In Flowchart

Flowchart

```
graph TD; Start([Sign-In Page]) --> Decision{Login In with details ?}; Decision -- NO --> Guest([Guest]); Guest --> MainMenu[Main Menu<br>(Name is Guest by Default)]; Decision -- YES --> Input[/INPUT for Email and Password/]; Input --> Login{Login Correct?}; Login -- NO --> Register([Link to Register Page]); Login -- YES --> MainMenu;
```

The flowchart starts at the "Sign-In Page". It asks "Login In with details ?". If "NO", it leads to the "Guest" state, which then leads to the "Main Menu (Name is Guest by Default)". If "YES", it leads to an input field for "Email and Password", then to a decision point "Login Correct?". If "NO", it leads to a "Link to Register Page". If "YES", it leads to the "Main Menu (Name is Guest by Default)".

Pseudocode

```
FUNCTION LoginScreen(NAVIGATION, ROUTE)
    INITIALIZE EMAIL and PASSWORD as empty strings
    INITIALIZE ERRORS as an empty object
    INITIALIZE PASSWORD_VISIBLE as FALSE
```

```
FUNCTION VALIDATE_LOGIN_FORM:
    CHECK if EMAIL and PASSWORD are filled
    IF either is missing, add error message to ERRORS
```

RETURN TRUE if no errors, FALSE otherwise

FUNCTION HANDLE_LOGIN:

IF VALIDATE_LOGIN_FORM is successful:

 GET stored user data

 IF stored user data matches entered EMAIL and PASSWORD:

 NAVIGATE to MAIN_MENU screen, passing user's name

 ELSE:

 ADD "Invalid credentials" error to ERRORS

DISPLAY login screen:

 DISPLAY logo and title

 DISPLAY EMAIL input field

 DISPLAY PASSWORD input field (with visibility toggle)

 DISPLAY LOGIN button

 DISPLAY GUEST ACCESS button

 DISPLAY SIGNUP link

 DISPLAY error messages from ERRORS

Description

The Sign-In Screen is a user-friendly interface designed for secure authentication, allowing users to log in using their email and password. It features input fields for email and password, with the password field including a toggle to show or hide the entered text for added convenience. The screen validates user inputs, ensuring the email is in the correct format and the password meets the required criteria. Upon successful validation, it checks the entered credentials against stored user data and, if matched, navigates the user to the main menu.

Additionally, it provides options for guest access and a link to the registration screen for new users. A "Forgot Password?" link allows users to reset their credentials securely via email verification. The design is clean and intuitive, with clear error messages, accessible form controls, and a responsive layout that adapts to different screen sizes for an optimal user experience.

The function for Validate Login is to check if the email and password are filled in or empty and then supply an error message if it's not, similarly no error messages when there is a value in those fields.

Handle Login function is purely to check against the user data if the email and password match like the flowchart shows a decision is made here and if it matches the Handle Login function will navigate the user to the Main Menu, else present an error message, saying login is not working.

Register Page Flowchart

Flowchart

Register Page

Remember Details?

Register

Return to Login Page

INPUT for Name,
Email, Password
and Confirm
Password

Main Menu

Login Correct?

YES

NO

YES

NO

Pseudocode

```
FUNCTION RegisterScreen
    INITIALIZE state variables: name, email, password, confirmPassword, errors

    FUNCTION validateForm
        CREATE empty errors object

        IF name is empty
            ADD error: "Name is required"
        ENDIF

        IF email is empty
            ADD error: "Email is required"
        ELIF email is not valid (using a regular expression check)
            ADD error: "Please enter a valid email address"
        ENDIF

        IF password is empty
            ADD error: "Password is required"
        ENDIF

        IF password does not match confirmPassword
            ADD error: "Passwords do not match"
        ENDIF

        SET errors state
        IF errors object is empty
            RETURN true
        ELSE
            RETURN false
        ENDIF
    ENDFUNCTION

    FUNCTION handleSubmit
        IF validateForm returns true
            TRY
                SAVE user data (name, email, password) to AsyncStorage with key 'userDetails'

                SET name state to empty string
                SET email state to empty string
                SET password state to empty string
                SET confirmPassword state to empty string
                SET errors state to an empty object

                NAVIGATE to Login page, passing the name as a parameter
            CATCH error
```

```

PRINT "Error saving data:" error to the console
ENDTRY
ENDIF
ENDFUNCTION

RETURN Register screen UI
DISPLAY title: "Register your account"
DISPLAY subtitle: "Create an account to access SchoolNav features."

DISPLAY name input field
IF there is an error for name
    DISPLAY the error message below the input field
ENDIF

DISPLAY email input field
IF there is an error for email
    DISPLAY the error message below the input field
ENDIF

DISPLAY password input field
IF there is an error for password
    DISPLAY the error message below the input field
ENDIF

DISPLAY confirm password input field
IF there is an error for confirmPassword
    DISPLAY the error message below the input field
ENDIF

DISPLAY register button
WHEN register button is pressed, CALL handleSubmit function

DISPLAY "Already have an account? Sign in" link
WHEN "Sign in" link is pressed, NAVIGATE to Login page
ENDFUNCTION

```

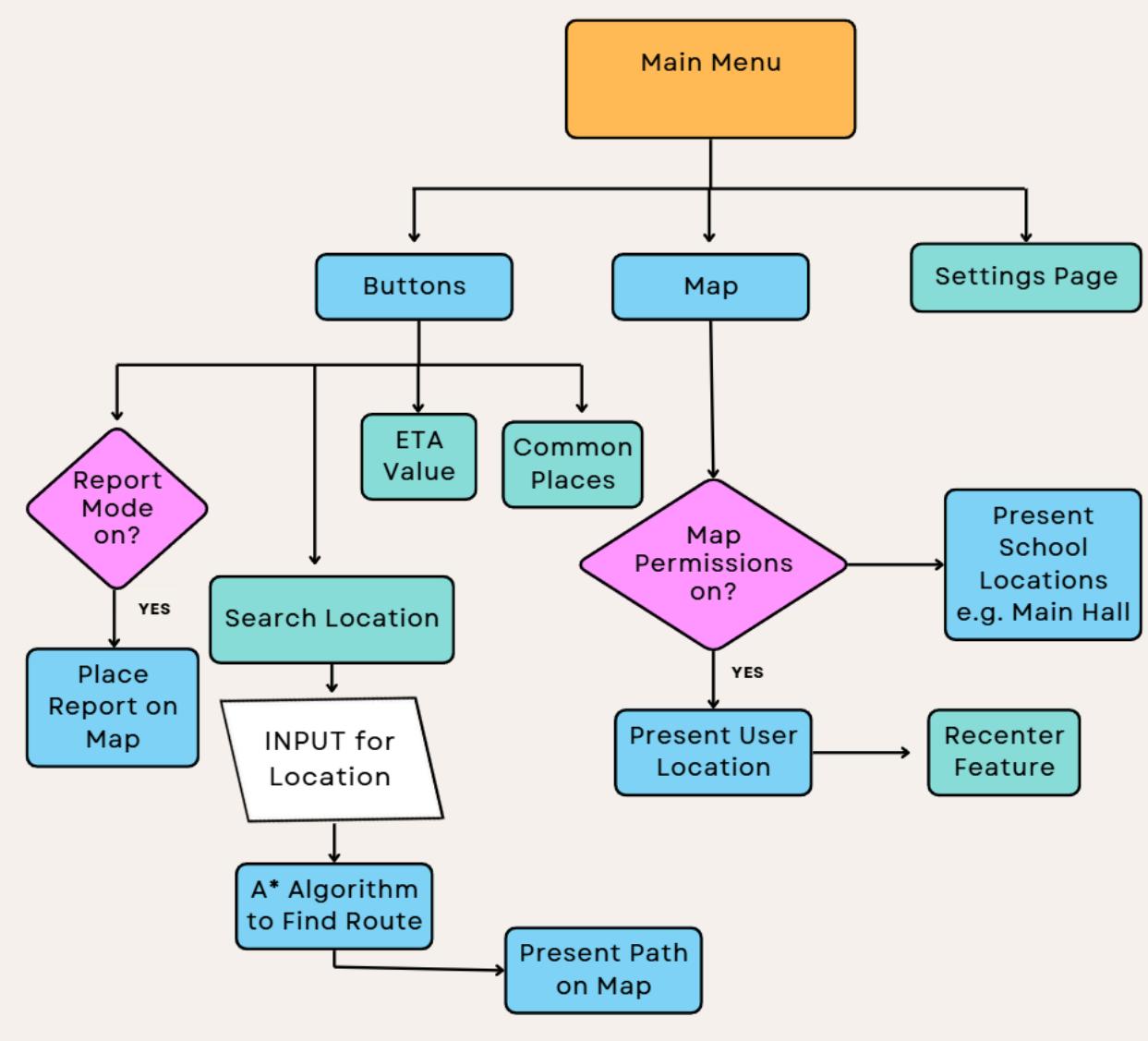
Description

The code creates a registration screen where users can sign up for an account. It checks if they've entered all their information correctly, like their name, email, and a matching password. If everything looks good, it stores their info securely and takes them to the login screen. If they made a mistake, like forgetting a field or having mismatched passwords, it shows them helpful error messages so they can fix it. There's also a link to the login screen for people who already have an account.

Very similar to the Sign-In page, the main difference is the added condition of

Confirm Password. The function validateForm checks and outputs an error, using a set of conditions e.g. is Name of correct form (string etc.). A set of IF and ELIFs show these conditions take place in the validate form function. The handleSubmit sets the appropriate values of the fields to the respective variables so it can be stored in a database or. This will then navigate users to the Login Page to then Sign-IN with the according details they just submitted.

Main Menu Flowchart



Pseudocode

```
FUNCTION MapScreen
    INITIALIZE state variables: location, destination, routeCoordinates, modalVisible,
    selectedPlace, searchInput
    INITIALIZE references: mapRef

    ON SCREEN LOAD:
        REQUEST location permissions
        IF permissions granted:
            GET current device location
            SET location state to current location
        ELSE:
            DISPLAY alert: "Location permissions denied"

    FUNCTION calculateRoute(destination):
        IF location and destination are valid:
            CALCULATE route between location and destination using map data (nodes)
        IF route found:
            SET routeCoordinates state to calculated route
        ELSE:
            DISPLAY alert: "Route not found"

    FUNCTION handlePlaceSelection(place):
        SET selectedPlace state to place
        SET destination state to place coordinates
        CALL calculateRoute(destination)
        SET modalVisible state to false

    FUNCTION handleSearch(text):
        SET searchInput state to text

    FUNCTION filterPlaces(text):
        RETURN places array filtered by place name containing text

    FUNCTION openDirections(destination):
        OPEN map application with directions to destination

    FUNCTION renderMap:
        DISPLAY map component centered on location
        DISPLAY markers for all places
        DISPLAY marker for location
        IF destination is set:
            DISPLAY marker for destination
        IF routeCoordinates is not empty:
            DISPLAY polyline representing routeCoordinates
```

```

FUNCTION renderModal:
  DISPLAY modal component
  DISPLAY search input field
  DISPLAY list of filtered places
  ON place selection:
    CALL handlePlaceSelection(selected place)

  RETURN screen UI:
    DISPLAY map component (CALL renderMap)
    DISPLAY button to open directions (CALL openDirections)
    DISPLAY button to open modal (SET modalVisible to true)
    IF modalVisible is true:
      DISPLAY modal (CALL renderModal)

CREATE array EDGES containing connection information between NODES with
WEIGHT (distance)
CREATE array NODES containing location data (latitude, longitude) for each node

FUNCTION findPath(START_ID, END_ID)
  CREATE OPEN_SET array containing START_ID
  CREATE CAME_FROM map to store the best path
  CREATE G_SCORE map to store cost from START_ID to a node
  CREATE F_SCORE map to store total cost (G_SCORE + heuristic)

  FOR EACH NODE in NODES:
    G_SCORE[NODE.id] = INFINITY
    F_SCORE[NODE.id] = INFINITY
  ENDFOR

  G_SCORE[START_ID] = 0
  F_SCORE[START_ID] = heuristic(START_ID, END_ID)

  WHILE length of OPEN_SET > 0:
    CURRENT = node in OPEN_SET with the lowest F_SCORE
    IF CURRENT is equal to END_ID:
      BREAK (path found)
    ENDIF

    REMOVE CURRENT from OPEN_SET

    CREATE PATH array
    CURRENT = END_ID
    WHILE CURRENT exists:
      ADD CURRENT to the beginning of PATH
      CURRENT = CAME_FROM[CURRENT]
    ENDWHILE

```

```

RETURN PATH
ENDFUNCTION

FUNCTION heuristic(NODE_ID, END_ID)
    NODE = NODE from NODES where NODE.id is equal to NODE_ID
    END_NODE = NODE from NODES where NODE.id is equal to END_ID
    RETURN calculateDistance(NODE, END_NODE)
ENDFUNCTION

FUNCTION handleOpenCommonPlacesModal()
    UPDATED_COMMON_PLACES = updateCommonPlaces(FORM_ROOM, PLACES,
    COMMON_PLACES)
    SET COMMON_PLACES to UPDATED_COMMON_PLACES
    SET IS_COMMON_PLACES_MODAL_VISIBLE to TRUE
ENDFUNCTION

FUNCTION MainMenu(ROUTE, NAVIGATION)
    GET NAME and ETA_UNIT from ROUTE parameters
    INITIALIZE state variables: USER_LOCATION, LOCATION_PERMISSION,
    SELECTED_PLACE, SEARCH_QUERY, SEARCH_RESULTS, IS_MODAL_VISIBLE,
    SHOW_MARKERS, PATH, PATH_COORDINATES,
    IS_COMMON_PLACES_MODAL_VISIBLE, IS_SEARCH_LOCATION_MODAL_VISIBLE,
    FORM_ROOM, REPORT_MODE, MARKER, DESCRIPTION,
    REPORT_MODAL_VISIBLE, ETA_VALUE, COMMON_PLACES

    ON SCREEN LOAD:
        REQUEST location permissions
        SET LOCATION_PERMISSION to TRUE if permissions granted, FALSE otherwise
        IF permissions granted:
            GET current device location with highest accuracy
            SET USER_LOCATION to current location coordinates
        ENDIF

    ON FORM_ROOM parameter change:
        IF FORM_ROOM parameter exists:
            SET FORM_ROOM to parameter value
            UPDATE COMMON_PLACES using UPDATE_COMMON_PLACES function

    FUNCTION handleOpenCommonPlacesModal:
        SET IS_COMMON_PLACES_MODAL_VISIBLE to TRUE

    FUNCTION handleSearch(QUERY):
        SET SEARCH_QUERY to QUERY
        IF QUERY is empty:
            CLEAR SEARCH_RESULTS

```

```

RETURN
ENDIF
FILTER PLACES by name containing QUERY
SET SEARCH_RESULTS to filtered results

FUNCTION goToPlace(PLACE):
IF MAP_REF exists:
    ANIMATE map to PLACE coordinates
ENDIF
SET SELECTED_PLACE to PLACE
SET IS_MODAL_VISIBLE to FALSE
CLEAR SEARCH_QUERY and SEARCH_RESULTS

FUNCTION recenterMap:
IF MAP_REF and USER_LOCATION exist:
    ANIMATE map to USER_LOCATION

FUNCTION handleCalculateRoute(ID):
FIND closest node ID to USER_LOCATION using FIND_CLOSEST_NODE_ID
CALL goToPlace for the closest node
FIND path from closest node ID to ID using FIND_PATH
MAP path to coordinates using NODES
SET PATH_COORDINATES to coordinates
CALCULATE total weight of path using EDGES
CALCULATE ETA in seconds
SET ETA_VALUE to calculated ETA

FUNCTION findClosestNodeId(USER_LOCATION, NODES):
IF USER_LOCATION or NODES are invalid:
    THROW error: "Invalid userLocation or nodes array."
ENDIF
INITIALIZE CLOSEST_NODE_ID to NULL, SHORTEST_DISTANCE to INFINITY
FOR EACH NODE in NODES:
    CALCULATE distance between USER_LOCATION and NODE using
CALCULATE_DISTANCE
    IF distance < SHORTEST_DISTANCE:
        SET SHORTEST_DISTANCE to distance
        SET CLOSEST_NODE_ID to NODE.id
    ENDIF
ENDFOR
RETURN CLOSEST_NODE_ID

FUNCTION MainMenu(ROUTE, NAVIGATION)
GET NAME from ROUTE parameters
INITIALIZE state variables: USER_LOCATION, LOCATION_PERMISSION,
SELECTED_PLACE, SEARCH_QUERY, SEARCH_RESULTS, IS_MODAL_VISIBLE,

```

```

SHOW_MARKERS, PATH, PATH_COORDINATES,
IS_COMMON_PLACES_MODAL_VISIBLE, IS_SEARCH_LOCATION_MODAL_VISIBLE,
FORM_ROOM, REPORT_MODE, MARKER, DESCRIPTION,
REPORT_MODAL_VISIBLE, ETA_VALUE, COMMON_PLACES
INITIALIZE mapRef

ON SCREEN LOAD:
REQUEST location permissions
SET LOCATION_PERMISSION based on permission status
IF permissions granted:
    GET current location with highest accuracy
    SET USER_LOCATION to current location coordinates

FUNCTION resetRoute:
CLEAR PATH and PATH_COORDINATES

FUNCTION calculateBearing(START, END):
CALCULATE bearing between START and END coordinates
RETURN normalized bearing

FUNCTION rotateMapToNode(CURRENT_NODE, TARGET_NODE):
IF mapRef, CURRENT_NODE, and TARGET_NODE are valid:
    CALCULATE bearing between CURRENT_NODE and TARGET_NODE

FUNCTION rotateMapToNode2(CURRENT_NODE, TARGET_NODE):
IF mapRef, CURRENT_NODE, and TARGET_NODE are valid:
    CALCULATE bearing between CURRENT_NODE and TARGET_NODE
    ANIMATE map camera to CURRENT_NODE

FUNCTION handleMapPress(MARKER_EVENT):
IF REPORT_MODE is TRUE:
    SET MARKER to pressed coordinate
    SET REPORT_MODAL_VISIBLE to TRUE
    SET REPORT_MODE to FALSE

FUNCTION submitReport:
SET REPORT_MODAL_VISIBLE to FALSE

RETURN screen UI:
DISPLAY status bar
DISPLAY weather and settings buttons
ON settings button press:
    NAVIGATE to "Settings Page"
DISPLAY greeting with NAME
DISPLAY map component
IF USER_LOCATION exists:
    DISPLAY USER_LOCATION marker

```

```

IF SHOW_MARKERS is TRUE:
    DISPLAY markers for all PLACES
IF PATH_COORDINATES is not empty:
    DISPLAY polyline for PATH_COORDINATES
IF MARKER exists:
    DISPLAY MARKER

DISPLAY report modal
IF REPORT_MODAL_VISIBLE is TRUE:
    DISPLAY report input
    ON submit button press:
        CALL submitReport

DISPLAY "Clear Route" button
ON button press:
    CALL resetRoute
DISPLAY "Recenter" button
ON button press:
    CALL recenterMap
DISPLAY "Show/Hide Markers" button
ON button press:
    TOGGLE SHOW_MARKERS

DISPLAY features section
DISPLAY "Report" button
ON button press:
    SET REPORT_MODE to TRUE
DISPLAY ETA section
    DISPLAY ETA_VALUE
DISPLAY "Common Places" button
ON button press:
    SET IS_COMMON_PLACES_MODAL_VISIBLE to TRUE
    DISPLAY common places modal
    FOR EACH place in COMMON_PLACES:
        DISPLAY place button
        ON place button press:
            SET IS_COMMON_PLACES_MODAL_VISIBLE to FALSE
            CALL goToPlace for place node
            DELAY and CALL handleCalculateRoute for place id
DISPLAY "Close" button
ON button press:
    SET IS_COMMON_PLACES_MODAL_VISIBLE to FALSE

DISPLAY "Search Location" section
ON button press:
    SET IS_SEARCH_LOCATION_MODAL_VISIBLE to TRUE
    DISPLAY search location modal

```

```

DISPLAY search input
ON search input change:
CALL handleSearch
DISPLAY search results
FOR EACH item in SEARCH_RESULTS:
    DISPLAY result item button
    ON result item button press:
        SET IS_SEARCH_LOCATION_MODAL_VISIBLE to FALSE
        CALL goToPlace for item
        DELAY and CALL handleCalculateRoute for item id
DISPLAY "Close" button
ON button press:
    SET IS_SEARCH_LOCATION_MODAL_VISIBLE to FALSE

```

Description

The location-based navigation feature integrates with Google Maps to display the user's current position and predefined places (e.g., rooms, halls, and facilities) as markers on a map. It uses GPS to track the user's location and provides features like searching for specific locations, calculating the shortest path between two points using the A* algorithm, and displaying the route as a highlighted polyline with an estimated travel time (ETA). Users can toggle markers, recenter the map, and access a list of common places for quick navigation.

Firstly, since the entire screen needs the location permissions, the program will start by requesting the user for these requests, shown in the REQUEST location permissions section, if this is not provided, an alert will be displayed stating location permissions denied.

The function calculateRoute shows how the A* Algorithm will be presented in an abstracted form, I explain how I would implement the A* Algorithm later down on this description. The calculateRoute function would find the route and if the route exists, present the route, else alerts the user and states Route not found. From here, the handlePlaceSelection function calls the calculateRoute function.

The function calculateDistance calculates the distance between two nodes using the Haversine formula, accounting for the Earth's curvature. This is used to determine the shortest path between locations.

The function findPath implements the A* Algorithm in an abstracted form. It initializes the required sets, assigns infinite scores to all nodes except the start node, and iterates through neighbors using a heuristic to determine the best path. If a route is found, it reconstructs the path and returns it; otherwise, no path is returned.

The handleCalculateRoute function calls findClosestNodeId to identify the nearest node to the user's location and then calls findPath to compute the route. If a path

exists, it maps node IDs to coordinates and calculates the estimated time of arrival.

The renderMap function visually represents the system by displaying markers for locations, including the user's current location, the selected destination, and the computed route as a polyline.

The modal system, handled by renderModal, enables users to search for locations and select a place, triggering handlePlaceSelection, which ultimately calls handleCalculateRoute.

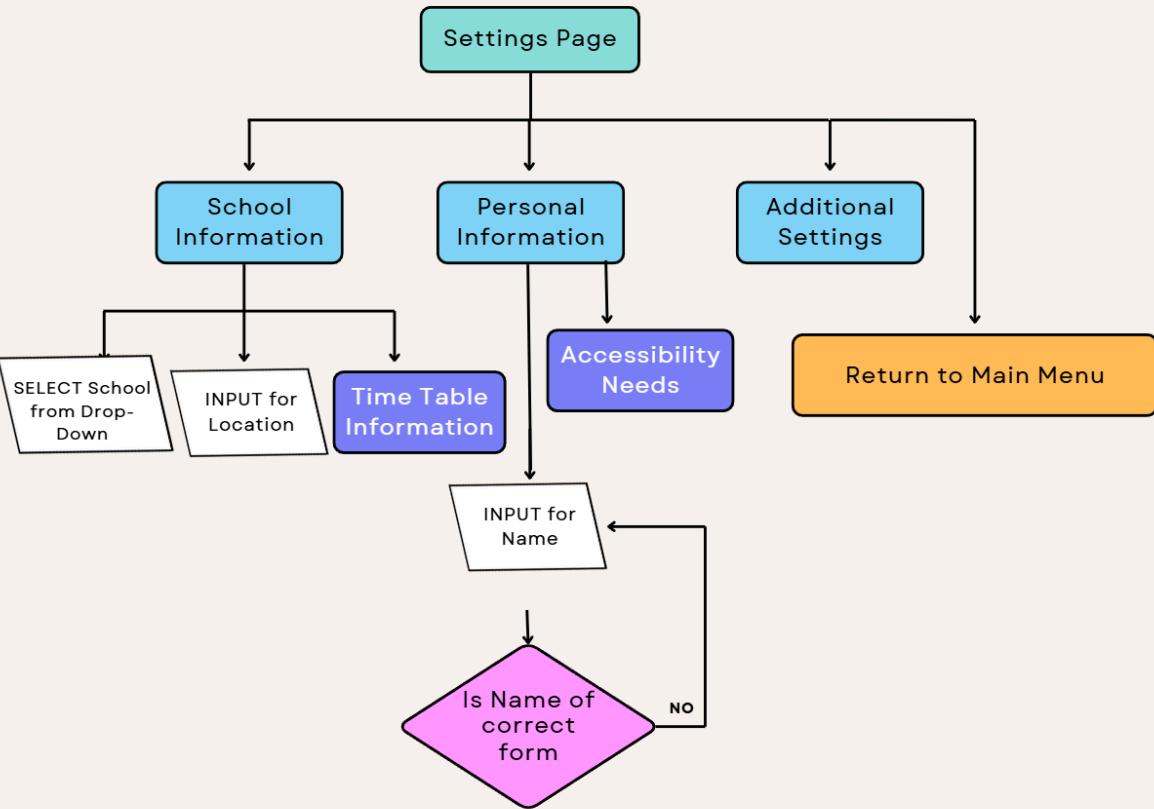
Additional functions such as openExternalLink ensure URLs are valid before opening them, while resetRoute clears the displayed route and path coordinates. The program also includes logic for updating commonly visited places, filtering places by room number, and handling search functionality within the UI.

Additionally, the app includes a reporting feature, allowing users to mark and describe issues on the map, such as blocked paths or maintenance concerns. These reports can be reviewed by administrators for timely resolution. The interface is designed with modals for search, common places, and reporting, ensuring a clean and interactive user experience.

The A* algorithm in SchoolNav efficiently calculates the shortest path between two locations by combining actual travel cost and an estimated distance to the goal. It represents the school as a graph, where rooms, halls, and facilities are nodes connected by corridors as weighted edges. The algorithm starts at the user's location and explores paths using a priority queue, selecting the most promising route based on the sum of the actual cost ($gScore$) and an estimated remaining distance ($h(n)$). As it progresses, it updates paths dynamically, recalculating if a shorter route is found. Once the goal is reached, it backtracks to reconstruct the optimal path, which is then displayed as a blue polyline on the map, complete with an estimated travel time (ETA). SchoolNav optimizes this system with real-time adjustments, weighted paths for accessibility, and precomputed common routes, ensuring smooth and efficient navigation.

To enhance usability, the navigation system dynamically updates the user's position in real time, ensuring accurate routing adjustments as they move. Custom icons differentiate various types of locations, improving visual clarity. The map also supports offline mode, enabling users to access predefined routes and locations even without an active internet connection.

Settings Page Flowchart



Pseudocode

```

FUNCTION SettingsPage(NAVIGATION, ROUTE)
  GET NAME from ROUTE parameters, or set to "Guest" if not provided
  INITIALIZE state variables: SCHOOL_INFO_EXPANDED,
  PERSONAL_INFO_EXPANDED, ADDITIONAL_SETTINGS_EXPANDED, FORM_ROOM,
  ERROR, ACCESSIBILITY_NEEDS, THEME, IS_SEARCH SCHOOL_MODAL_VISIBLE,
  SEARCH_QUERY, SEARCH_RESULTS, IS_DARK_MODE, MODAL_VISIBLE, ETA_UNIT,
  ETA_DROPDOWN_VISIBLE

  CREATE array SCHOOLS with school data
  
```

```

FUNCTION handleSearch(QUERY):
  SET SEARCH_QUERY to QUERY
  IF QUERY is empty:
    SET SEARCH_RESULTS to SCHOOLS
  ELSE:
    FILTER SCHOOLS by name containing QUERY
    SET SEARCH_RESULTS to filtered results
  
```

```

FUNCTION selectSchool(SCHOOL):
    SET IS_SEARCH_SCHOOL_MODAL_VISIBLE to FALSE
    PRINT "Selected School ID:", SCHOOL.name

FUNCTION validateName(TEXT):
    CREATE regular expression to validate full name
    IF TEXT does not match regular expression:
        SET ERROR to "Enter a valid full name..."
    ELSE:
        SET ERROR to ""
        SET NAME to TEXT

RETURN screen UI:
DISPLAY scrollable view
DISPLAY title "Settings"

DISPLAY "School Information" button
ON button press:
    TOGGLE SCHOOL_INFO_EXPANDED
    IF SCHOOL_INFO_EXPANDED is TRUE:
        DISPLAY "Select School" button
        ON button press:
            SET IS_SEARCH_SCHOOL_MODAL_VISIBLE to TRUE
        DISPLAY form room input
        DISPLAY "Time-Table Information" button

DISPLAY school search modal
IF IS_SEARCH_SCHOOL_MODAL_VISIBLE is TRUE:
    DISPLAY search input
    ON search input change:
        CALL handleSearch
    DISPLAY search results
    FOR EACH item in SEARCH_RESULTS:
        DISPLAY result item button
        ON result item button press:
            CALL selectSchool(item)
    DISPLAY "Close" button
    ON button press:
        SET IS_SEARCH_SCHOOL_MODAL_VISIBLE to FALSE

DISPLAY "Personal Information" button
ON button press:
    TOGGLE PERSONAL_INFO_EXPANDED
    IF PERSONAL_INFO_EXPANDED is TRUE:
        DISPLAY name input
        ON name input change:
            CALL validateName

```

```

DISPLAY error message if ERROR exists
DISPLAY accessibility needs toggle

DISPLAY "Additional Settings" button
ON button press:
    TOGGLE ADDITIONAL_SETTINGS_EXPANDED
IF ADDITIONAL_SETTINGS_EXPANDED is TRUE:
    DISPLAY theme toggle button
        ON button press:
            TOGGLE THEME
    DISPLAY ETA measure button
        ON button press:
            SET ETA_DROPDOWN_VISIBLE to TRUE

DISPLAY ETA unit selection modal
IF ETA_DROPDOWN_VISIBLE is TRUE:
    DISPLAY picker for ETA units
        ON picker value change:
            SET ETA_UNIT to selected value
    DISPLAY "Confirm" button
        ON button press:
            SET ETA_DROPDOWN_VISIBLE to FALSE

DISPLAY "Return to Main Menu" button
ON button press:
    NAVIGATE to "Main Menu" with NAME and ETA_UNIT
    SET button to disabled if ERROR exists

```

Description

This pseudocode outlines the steps for a settings screen. It initializes the user's name and settings states, ensuring the data is preloaded correctly. It provides functions to search and select schools, validate the user's name, and toggle settings sections to enhance user control over the interface. The screen displays buttons to expand or collapse different settings sections, inputs for the user to modify their room and name, and toggles to switch between accessibility options and theme preferences (light or dark mode). A modal is included for users to select their preferred ETA unit (minutes or hours). It also features a button to return to the main menu, passing the updated user's name and ETA unit as parameters to ensure a personalized experience. Additionally, the settings screen includes real-time validation to ensure all inputs are correct, and feedback is provided through clear error messages, maintaining an intuitive and responsive layout.

Firstly, since user settings require customization, the program initializes state variables to manage sections like school information, personal details, and additional settings. The NAME parameter is retrieved from ROUTE, defaulting to

"Guest" if not provided.

The handleSearch function filters school options based on user input, updating SEARCH_RESULTS dynamically. When a user selects a school, selectSchool is triggered, updating the chosen school and closing the modal.

For personal information, validateName ensures proper formatting using a regular expression. If invalid, an error message is displayed, preventing submission. Accessibility settings are also available in this section.

Additional settings allow users to toggle between themes and select an estimated time of arrival (ETA) measurement unit. If the user expands this section, they can open a modal to choose their preferred ETA unit.

The UI consists of collapsible sections controlled by toggle buttons. If SCHOOL_INFO_EXPANDED, PERSONAL_INFO_EXPANDED, or ADDITIONAL_SETTINGS_EXPANDED are TRUE, corresponding input fields and options are displayed.

A "Return to Main Menu" button navigates back with the user's selected NAME and ETA_UNIT. However, this button is disabled if an error exists, ensuring that users correct any invalid inputs before proceeding.

Key Variables and Functions

Function Name	Variable Name	Data Type	Purpose/Justification
handleLogin	name	String	To pass the name linked to the account to the Main Menu to greet the user and for other features.
	email	String	To validate the login and check against the password value input. This is the unique field which links each account.
	password	String	To validate the login and check against the email value input. This is not always unique.
validateForm	errors	String	A flag created to inform the program if any errors are present in the screen e.g. Password field is not defined. Also, to show what error to present in the function later on.
setIsPasswordVisible	isPasswordVisible	Boolean	This is used to change the nature of the password input from visible to hidden.
handleSubmit	confirmPassword	String	This is to check against the initial password in the Register Screen. Only if they match will the

			handleSubmit function run.
calculateDistance	Radius	Integer	A variable created to denote the radius of the Earth to be used in calculation with the distance between two locations.
	dLatitude	Integer	A variable created to denote the difference of Latitude between two locations to be used in calculation with the distance between two locations.
	dLongitude	Integer	A variable created to denote the difference of Longitude between two locations to be used in calculation with the distance between two locations.
findPath	openSet	String	A variable created to present the nodes in the path from A* Algorithms.
	cameFrom	Integer	This variable shows the path and where each node came from, this assists with the cost generation and the visualisation of the path later on.
	gScore	Real	This variable is a fundamental piece to work out the cost from the start to a node

	fScore	Real	This variable is a fundamental piece to work out the cost from the start to a node, by adding the gScore with the Heuristic.
	nodes	Integer	Variable which defines the nodes of the map for the algorithm and it contains key information about each of the points, e.g. Longitude, Latitude and Name.
	edges	Integer	Shows the edges which connect the nodes, to form the graph structure.
	path	Integer	Variable was created to show which nodes to traverse in order to present the path of the route.
useEffect	setUserLocation	Float	This is to present the user location, so that it can be animated and presented on the map.
	setLocationPermission	Boolean	Variable is used to request users for location permissions, so that the location can be tracked.
	setFormRoom	Integer	This variable was created to initialise and set the FormRoom of the user, so it can be presented in the Common Places list.

	setCommonPlaces	List	This variable was created to show the common places users can go to and the user's specific form room.
handleCalculateRoute	cNodeID	String	This is to find the ID of the closest node to the user! So the route can start from this node.
	coordinates	Float	This variable was created to show the longitude and latitude of the path and each node that it uses.
	totalWeight	Real	This value is what is used to find the ETA of the route. This is the weight of the path.
	etaValue	Integer	This variable was created to represent the ETA Value of the path. This is an integer because I round the value to the nearest integer, since this is being presented on the screen.
findClosestNodeId	shortestDistance	Real	This is created to allow comparison of the distances so we can find the shortest distance.
	closestNodeId	String	This is to show what the current closestNodeId is as the entire function finds this Id.

resetRoute	setPath	List	This sets the path to something more specific. The variable was created to allow the resetRoute function to reset the path and set it to null.
	setPathCoordinates	List	This sets the pathCoordinates to something more specific. The variable was created to allow the resetRoute function to reset the path and set it to null.
handleMapPress	setMarker	Float	This variable was created to present the location of the marker and the description, for the report feature.
	setReportModalVisible	Boolean	This variable was created to allow the report feature to work effectively. Setting it to visible means users can then place a report.
	setReportMode	Boolean	This was used to toggle between normal mode and report mode, so users can switch at will.
submitReport	setReportModalVisible	Boolean	This variable was created to allow the report feature to work effectively. Setting it to visible means users can then place a report.

Validation

Validation is to test the software using test data. The four types of test data to be used are Normal, Boundary, Invalid and Erroneous.

For the three main sections of my application, Login/Sign-up Page, Main-Menu, Settings Page, I need to test these four test data to ensure validation of my project, which means my software is running properly and doesn't have any errors or gaps/fail-safes.

Normal test data should be accepted without causing errors.

Boundary test data checks the maximum and minimum values that may be entered.

Invalid test data is data of correct type, but outside validation rules.

Erroneous test data should not be accepted, due to wrong data type.

Test Data	Login/Sign-Up Page Inputs:	Main-Menu Page Inputs:	Settings Page Inputs:
Normal	A standard string address with an @ in it for email. Password - any string.	Report function should have any strings. The search function should take any string.	School Information should take any string for the school name. Personal Name - Alphabetical string Form Room - Numerical String between 1-2 inclusive.
Boundary	N/A	Text for the reporting function should be capped at 200 characters. Hence the boundary is 201 character string.	School Name should have a limit of 30 characters and Minimum of 0 characters. 0 and 30 character strings should provoke error messages.

			Form Room should provide an error message for 3 digits input.
Invalid	<p>Any string without the character @ is invalid for the email address.</p> <p>Any string with more than one @ is invalid.</p> <p>Any string with no domain name after @ character is invalid.</p> <p>Any strings with any symbols other than these: ();;<>@[\].</p>	N/A	<p>Strings with character length of 0 or >29 is invalid for School Name.</p> <p>Form Room should be between 1-2 digits, inclusive, hence anything else is invalid.</p>
Erroneous	N/A	N/A	Form Room details anything that is not numerical is erroneous.

Test Plan for Iterative Development

These test plans focus on the Login/Sign-up Page, the Main Menu and the Settings Page. They are designed to meet the success criteria we have established in the analysis section. The iterative testing process in the development stage will use these test plans to judge success in each iteration and stage. From there, the post development testing will be after all development has finished and will be looking at Robustness and Functionality features and then stakeholder testing for the Usability features. The test plans allow me to find where I am not meeting the test plan and use that feedback and direction to then adjust my program to then meet these Test IDs, then the Success Criteria most importantly.

Test ID	Test Description	Test Method/Data	Expected Outcome/Justification
<u>Login/Sign-up Page</u>			
1.1	Application should run	Start the application and see if anything is visually seen as expected.	Expected text, images and boxes should be output.
1.2	Should display background image/colour	Log-in/Sign-Up Page , visual check for background image.	A blue background image presented in the User Interface Section.
1.3	The Log-in page should be the first page for new users who haven't yet logged-in/created account as stated in SC.06	Log out if logged and relaunch application.	Visual check that the screen present is Log-In/Sign-up Page.
1.4	Should display Logo clearly on top of background as a png as stated in SC.15	Check if the SchoolNav logo is displayed.	Logo referenced in User Interface should be displayed on top of Log-In Page. Use User Interface: Log-in/Sign-Up Page for reference.
1.5	Beneath the Logo, should appear Title for Sign-In	Check if the title to Sign-In is presented.	Sign-In text should appear below Logo Icon. Use User Interface:

			Log-in/Sign-Up Page for reference.
1.6	Add a subtitle below the title to greet the users and inform them of the application	Visual check for according subtitle	Subtitles should appear below the title and be descriptive of the app's purpose.
1.7	The input for the email address should be clear and obvious to the user	Visual check for email address box and title.	Present should be a text box for an email address with a placeholder and appropriate title.
1.8	The input for the password should be clear and obvious to the user	Visual checks for the password input and title.	Present should be a textbox for password with a placeholder and appropriate title.
1.9	Text should be able to be input into email address and password	Attempt typing into each of the fields and the values should be saved as email/password	The text input should open the keyboard to allow typing and the text should appear for each box.
1.10	Text should be encrypted when writing in the password field	Type into the password field and visual check for encryption symbols to prevent others from seeing the password.	The text typed into the field should be replaced with encryption symbols to prevent others from seeing the password.
1.11	Screen should have matching tone and clear theme	Visual check for consistent blue and white theme for Login Screen.	Consistent blue and white theme for Login Screen.
1.12	Sign-In button should be prevalent and clear to the user	Visual check for the Sign-In button	The Sign-In button should be present below the fields for input.
1.13	The Sign-In button should be intractable with user	Press the Sign-In Button	The Sign-In button should be able to be pressed.

1.14	Option to Sign-Up should be present and button should be pressable	Visual check for Sign-Up option and title to clearly define that option.	Present at the footer of the screen should be a Sign-Up option with a description.
1.15 – VALIDATION	If the Sign-In button is pressed without any input for either the email or password fields then the screen should output appropriate error messages to highlight the issue.	Sign-In without any input for both fields and visual check for any error messages. Input for email and leave out password and then repeat and vice versa to test each case.	When attempting to Sign-In without any text input for both fields you should find two error messages accordingly. If you Sign-In with one input then you should get one error message and if you Sign-In with both inputs, no error messages should be outputted.
1.16	Users should have the option to Sign-In as a guest since not all users would want to Sign-In/Sign-Up. This should be prevalent and clear to the user.	Visual check for Guest option.	There should be an obvious place for the Guest option located in a clear, distinct area on the screen.
1.17	Users should be able to toggle between visible password fields and encrypted fields	Visual check for toggle and run test cases by inputting text to see if it is encrypted and visible accordingly to toggles.	Text should be default encrypted and the toggle when pressed should allow the user to view it and vice versa.
1.18	Users should be able to identify Register Screen/Sign-Up Page through a clear Title.	Visual check for according title to signify Register Screen.	Titles should be prevalent and large to stand out and inform users of the page they are on.
1.19	Field for name should be implemented to provide customisation options.	Visual check for Name field and test through writing Text.	The text should save and not produce any error messages and the name field should be present.

Test ID	Test Description	Test Method/Data	Expected Outcome/Justification
1.20	Fields for Email Address and Password should be present and their purpose should be clear using relevant titles.	Visual check for the email address field and the password field with according titles and placeholders. Attempt inputting text.	The fields should be present and clearly visible and their purpose should be identifiable. Text should save and not produce any error messages.
1.21	Field for Confirm Password should function and be present	Visual check for the Confirm Password Field and input text into this field.	The fields should be present and clearly visible and their purpose should be identifiable. Text should save and not produce any error messages.
1.22	The register button should be present and pressable.	Visual check for the register button and attempt pressing it.	Register Button should be clear and prevalent and pressable.
1.23	Option to return back to Sign-In/Guest page	Visual check for options and the button should be pressable.	This option should be visible to users and pressable.
1.24 - VALIDATION	If the Register Button is pressed without any input for either the name, email or password fields then the screen should output appropriate error messages to highlight the issue.	Register without any input for all three fields and visual check for any error messages. Input for name and leave out password and then repeat and vice versa to test each case.	When attempting to Register without any text input for all fields you should find three error messages accordingly. If you Register with one input then you should get two error messages and if you Sign-In with three inputs, no error messages should be outputted, these error messages are in relation to their needing text in the fields.

1.25 - VALIDATION	If the Register Button was pressed without the Password field and the Confirm Password field not matching, appropriate error messages should be outputted.	Register with the passwords matching and register with them not matching visual check for clear and prevalent error messages.	According to both scenarios where they match or don't match, accurate and relevant error messages should not show and show.
1.26	The Register Button should clear all values of the fields to formulate a clean environment to compliment any error messages.	Press the Register button with inputs and without inputs and visual check if the inputs are no longer in the fields.	With each press of the Register button the fields on the register screen should empty out and be set to empty/null.
1.27	Navigation between different screens. “Sign-Up” button should navigate to the Register Screen etc.	Press each button that is meant to navigate to different screens and see if it navigates you to each screen respectively.	The buttons once pressed should navigate users to appropriate screens, once checked if no errors are present.
1.28	The Guest Button should function and be pressable. It should take users directly to the Main Menu and set the name to “Guest”.	Press the Guest button and check if it takes the user to the Main Menu and the name is set to “Guest”.	The Main Menu should be presented to the user by pressing the Guest button and the name is set to “Guest”.
1.29	Users should be able to navigate back to the Login Page from the Register Screen.	Press the Register Screen to navigate to the relevant screen and then visual check for the option to return to the Login Page and press it.	The user should be navigated to the Login Page and the button should be visible and clear to users.
1.30	User authentication with stored credentials	Enter a registered email and password for login. Simulate invalid login attempts with incorrect	Successful login for correct credentials redirects the user to Main Menu. Invalid credentials display an error message.

		credentials.	
1.31	Store and retrieve user data securely	Simulate saving user credentials during registration and retrieving them during login using persistent storage.	Stored user data matches inputted registration data and can be retrieved during login for authentication.
1.32-VALIDATION	Error handling for unregistered users	Enter an unregistered email during login.	The system displays an error message like "Invalid email or password."
1.33	Navigation to Main Menu after successful login	Simulate a login process with valid email and password.	The app should navigate to the Main Menu page and display the user's name.
1.34-VALIDATION	Error message display for failed login	Enter incorrect email or password during login attempts.	An appropriate error message is displayed below the email or password field.
1.35	Clear feedback for successful login	Complete login with valid credentials.	A message (or navigation) confirms login success and displays user-specific content on the next screen.
1.36	Retrieve name from Register Screen and display on Main Menu	Simulate navigating from Register Screen to Main Menu with a user-defined name inputted during registration.	The name should be displayed on the Main Menu after a successful login.
1.37 - VALIDATION	Incorrect email formats should throw error messages - Validation	<p>Any string without the character @ is invalid for the email address.</p> <p>Any string with more than one @ is invalid.</p>	Error message should be shown, incorrect format for email input.

		<p>Any string with no domain name after @ character is invalid.</p> <p>Any strings with any symbols other than these: () ; < > @ [\].</p>	
Test ID	Test Description	Test Method/Data	Expected Outcome/Justification
<u>Main-Menu Page</u>			
2.1	Main Menu Page should be initialised and set up to take in name from database/other screens.	Menu Menu page should open when pressed. Visual check for name output.	The Main Menu should be running and it should be initialised.
2.2	The Weather and Settings sections should appear side by side at the top of the screen.	Open the screen and visually check if the two sections are positioned horizontally next to each other.	The two sections should be aligned properly, without any overlap or gaps between them.
2.3	The Weather section should be clear in purpose and allow user interaction.	Examine the design of the Weather section to ensure it reflects its purpose. Test interaction by attempting to access weather information.	The section should clearly indicate it is for weather updates and should respond appropriately to user interaction.
2.4	The Settings section should be clear in purpose and allow user interaction.	Review the appearance of the Settings section to confirm its purpose. Test it by attempting to access settings options.	The section should clearly represent its function and respond appropriately to user interaction.

2.5	The title for the "School Map" section should be visible and clearly indicate its purpose.	Open the screen and check if the title for the map section is displayed prominently and is easy to read.	The title should be clearly visible and help users understand the purpose of the section immediately.
2.6	The layout should maintain a professional and user-friendly appearance.	Review the design and placement of elements on the screen to check for consistency and usability.	The screen should look organized and visually appealing, with all elements positioned intuitively.
2.7	Navigation back to the Login Page should work correctly.	Test the functionality by attempting to navigate from the Main Menu to the Login Page.	The app should seamlessly return to the Login Page without any disruptions or delays. This is for users who use Guest mode, but decide to return back to the Login In page.
2.8	The Features for Report and Common Places should be present and pressable.	Visual check for the Report and Common Places button and press it to see if it is interactable.	There should be two buttons reflecting the purpose of Reporting and for Common Places respectively.
2.9	The ETA should be a prominent section which is not pressable but is to present values.	Visual check for an ETA box with sufficient space to present information.	There should be a prominent box on the Main Menu screen for the ETA.
2.10	The Search Location button should be visual and prominent on screen.	Visual check for Search Location button and press to test interactability.	There should be a box for Search Location and it should be interactable.
2.11	Retrieve name from Register Screen and display on Main Menu	Simulate navigating from Register Screen to Main Menu with a user-defined name inputted during registration.	The name should be displayed on the Main Menu after a successful login.

2.12	The personalized welcome message at the top of the screen should display correctly.	Open the screen and check if the message is shown with the correct name for different users.	The welcome message should greet the user by their name or display a default value like "Guest" if no name is provided.
2.13	Verify that the map displays the user's current location correctly.	Check if the map is initialized to show the user's location as the center of the map. This can be tested by allowing the app to request location access and then confirming that the user's location is shown on the map with a visual marker or blue dot.	The map should center on the user's current location, with a marker or blue dot clearly indicating the position.
2.14	Verify that the recenter button works as intended.	After manually navigating away from the user's location, press the recenter button and observe if the map smoothly animates back to the user's current position.	The recenter button should recenter the map on the user's location, with a smooth animation back to the user's position.
2.15	Verify the map's zoom level is appropriate for the current location.	Check if the map is zoomed in enough to show detailed information for the user's location. This can be tested by visually inspecting the map after it loads to ensure it is neither too zoomed out nor too zoomed in.	The map should be zoomed in sufficiently to show a clear and detailed view of the user's location, without being too zoomed out or too zoomed in.
2.16	Verify that the map type is set correctly.	Check if the map is rendered with the correct type, such as	The map should display in the selected map type (e.g., hybrid, which

		hybrid or satellite, depending on the requirements.	combines standard map and satellite view).
2.17	Verify the correct handling of location permissions.	Test if the application asks for location permissions when needed and if the map displays the user's location only after permission is granted. If permissions are denied, the app should not attempt to access the user's location.	The app should request location permissions when necessary. If permission is granted, the map should display the user's location; if permission is denied, the app should handle it gracefully without attempting to fetch location data.
2.18	Verify that the map is responsive and interactive.	Test whether the user can interact with the map, including zooming in/out, panning, and moving around the map without issues.	The map should respond to user interactions such as zooming and panning. The map should be interactive and smooth when the user zooms or drags the map.
2.19	Verify that the app adjusts the map display based on the device's screen size.	Test the map on various devices with different screen sizes to ensure the map scales properly to fit the screen without distortion.	The map should adjust its size and scaling dynamically to fit different screen sizes, ensuring it looks properly scaled on all devices.
2.20	Verify the user's location updates dynamically as they move.	Move the device to a different location and check if the map updates the user's position in real-time.	The map should update dynamically as the user moves, with the location marker following the user in real-time.
2.21	Reporting Feature Button	Visual check for reporting feature button below School Map.	The "Report" button should be displayed below the Map.

2.22	The ETA section should be displayed	Visually check for section labelled ETA	The ETA Section should be in the middle, below the School Map.
2.23	Users can view the map interface with markers for all mapped classroom locations.	Open the app and visually check the map. Ensure all predefined locations have a marker displayed.	Markers for all classrooms are visible on the map in their correct positions.
2.24	The Search Location button is visible and functional.	Launch the app, check for the search button at the bottom of the interface. Input a search query.	The search bar is visible. Entering text displays a dropdown list of matching locations based on the input.
Test ID	Test Description	Test Method/Data	Expected Outcome/Justification
2.25	Modal displays a list of locations matching the search query.	Input part of a classroom name in the search bar (e.g., "Toilet"). Verify if a list of matching locations appears.	A filtered list of classroom names matching the input query is displayed within the modal.
2.26	Selecting a location from the search results zooms in on the selected location and after a short period of time, returns to the current location.	Tap on a location name in the search result list. Check if the map centers and zooms into the corresponding marker for that location.	Map smoothly transitions to the selected location and centers on it, for 2 seconds, then back to the current location of the user.
2.27	Users can dismiss the search modal if no selection is needed.	Open the search modal and press the close button or outside the modal area.	The search modal closes, and the user returns to the map interface.
2.28	The mapping of the school should be accurate and relevant.	Visual check and comparison with an actual school map and check if map plots well with	When the user selects a marker or searches for a location, the application should calculate and display a

		satellite imagery.	route from the closest node to the destination using the A* Algorithms.
2.29	Ensuring the route line is vivid and visible	Trigger a route calculation by selecting a destination or marker.	The route line should be visible, vivid (e.g., yellow or other contrasting color), and thick enough to be easily seen by users.
2.30	Testing route update based on user movement	Move the map view or change location after the initial route is drawn (e.g., by panning or zooming on the map).	The route line should update dynamically to reflect the new path based on the user's movement, maintaining accuracy.
2.31	Testing automatic panning aesthetically and smoothly to the destination and closest node	After selecting a location, ensure the map pans smoothly to the destination and the closest node within a set time (e.g., 3 seconds).	The map should smoothly pan to the destination and then to the closest node, following a clear route after a specified time.
2.32	Testing the calculation of the shortest path	Select different destination locations and check if the algorithm calculates and displays the shortest path.	The A* algorithm should calculate and render the shortest possible path from the current position or closest node to the destination.
2.33	Ensuring user feedback on path calculation status	After clicking the "Search Location" button or selecting a marker, verify the app shows a loading indicator or similar status while calculating the route.	The app should indicate the status of the route calculation process (e.g., loading spinner, message).
2.34	Button to Clear Route should be visible and prevalent and	Attempt pressing Clear Route with no route on screen, to	The Clear Route button should be obvious and present to the user and

	functional	detect errors. Search for a couple of routes and see if the route is being set onPress of the Clear Route button.	it should be on the map clearly. onPress it should remove any route lines.
2.35	Route resets when the reset option is selected	Visual check for a reset button. Click the button after selecting a route.	The route should disappear from the screen, and no polyline should be visible.
2.36	Resetting the route does not affect user location or school marker	Click the reset button and check that the user's location and school marker remain on the map.	Only the route should disappear, but user and school markers should remain unchanged.
2.37	If the user's distance from school exceeds a threshold, prompt Google Maps redirection	Simulate a location far from the school (e.g., 5+ km away). Then initiate the route finding algorithm.	A prompt should appear, offering an option to open Google Maps for navigation.
2.38	Google Maps opens with correct school destination	Click the Google Maps redirection button when prompted.	Google Maps should launch with directions pre-set to the school's location.
2.39	The Common Places button is pressable and triggers the modal.	Attempt to press the "Common Places" button.	Pressing the button should open a modal displaying preset locations.
2.40	Common Places modal has a list of preset locations.	Visual check inside the modal for multiple location buttons.	The modal should display a scrollable list of predefined locations.
2.41	Selecting a place closes the modal.	Press a place button inside the modal and observe if it closes.	The modal should close immediately after selection.
2.42	The new route is visually displayed on the screen.	Select a common place and check for a visual path update.	The screen should display a highlighted path to the selected

			destination.
2.43	Close button in the modal is visible and pressable.	Visual check and attempt to press the "Close" button in the modal.	The "Close" button should be visible and functional.
2.44	Routes' ETA proportional to Distance	Select a route and note down ETA time, then choose a route which is shorter and note ETA Time, and choose a route which is longer and note the ETA Time	The shorter route ETA Time should be lower than the initial route and the longer route ETA Time should be greater than the initial route.
2.45	ETA value should be a reasonable time estimate for the route.	Find a route and check if the ETA Value adjusts accordingly and is realistic, and time is with an actual test.	The ETA Value should be accurate and it should be reasonable, it is relative and proportional to the distance.
2.46	The ETA Value should be clear and visible and the value/unit should be clear and appropriate.	Visual check for the ETA Value and unit. User's should understand and infer the unit and meaning.	The ETA Value should be bold/clear in any way to the user. The unit should be defined either, minutes, seconds or hours.
2.47	Search Box should function properly within Search Location Pop-Up	Press Search Box button	Keyboard should be presented to type in location.
2.48	Search Box should function properly within Search Location Pop-Up v2 - VALIDATION	Press Search Box and then type in a valid location	Direct to School Map with a route and ETA.
2.49	Search Box should function properly within Search Location Pop-Up v3 - VALIDATION	Press search Box and then type in an Invalid Location	Error/message to say Location not found, perhaps "input" is not correctly typed.

Test ID	Test Description	Test Method/Data	Expected Outcome/Justification
2.50	Report/Comment button, when pressed should allow the user to click on a point on a map to place a marker and type in its description.	Press and attempt placing markers on various different locations and then, try to type in different inputs and see if they get saved and remain on map.	The report button should work and when pressed allows me to place a marker on the map and type in the label, it saves and I can see it as I traverse through the map.
2.51	The Report Marker is clear and visible to users.	Visual check for the Report Marker once placed. Zoom out and in to see if it's constant. Visually check for the description.	The Report should be clear and prevalent to users. It should pop out of the page and the description readable to all.
2.52	Users can Type and enter their suggestions and reports, and it will be presented and readable by users, on the marker.	Place Report and type in description/suggestion and see if it is presented on the report.	The Report has the appropriate description when initialised and is present and visually detectable.
2.53	The Weather Information Button should open up Weather Information Pop-Up	Press the weather information button.	The weather information screen should pop-up on top of the Main-Menu Screen
2.54	Weather Information should be accurate.	Look at the details provided in the weather information page, cross reference with other online resources e.g. Google Weather.	The weather should match and correspond to the online stating weather.
Test ID	Test Description	Test Method/Data	Expected Outcome/Justification
<u>Settings Page</u>			

3.01	Settings Page Label should be displayed	Visual check for the Settings Page label at the top	The label should be displayed at the top of the page and clearly visible to the user
3.02	Background image should be displayed	Visual check for background image consistency	The same background image used in the app should be present on the Settings Page
3.03	School Information Button should be displayed	Visual check for the "School Information" button	The button labeled "School Information" should be visible and displayed as the first option
3.04	Personal Information Button should be displayed	Visual check for the "Personal Information" button	The button labeled "Personal Information" should be visible and displayed as the second option
3.05	Additional Settings Button should be displayed	Visual check for the "Additional Settings" button	The button labeled "Additional Settings" should be visible and displayed as the third/final option
3.06	School Information Button should function to open a drop-down menu	Press the "School Information" button	The button should expand to reveal the "Select School," "Form Room," and "Time-Table Information" options
3.07	Personal Information Button should function to open a drop-down menu	Press the "Personal Information" button	The button should expand to reveal "Name" and "Accessibility Needs" sections
3.08	Additional Settings Button should function to open a drop-down menu	Press the "Additional Settings" button	The button should expand to reveal "Theme: Light/Dark" and "ETA Measure" sections
3.09	Settings Page should have a functioning	Attempt to scroll up and down the page	The scroll function should work smoothly,

	scroll feature		allowing the user to navigate all settings
3.10	School Name Button should function properly	Press the "Select School" button	A list of schools should be presented for selection from the SchoolNav Map database
3.11	Form Room Input should function properly	Press the "Form Room" input field	A field should be displayed with a numeric keyboard for input
3.12	Name Input should function properly	Press the "Name" input field, and return to Main Menu for a visual check to see if Name is updated.	A text input field and keyboard should appear, allowing users to enter their name
3.13 – VALIDATION	The name field should have proper validation to prevent any invalid inputs that are not of form name.	Enter a variety of names and check if an error message is produced, e.g. "123", "***", " " should not be accepted.	Any invalid inputs should output error messages to inform the user of the incorrect format.
3.14	Accessibility Needs Toggle should function properly	Press the "Accessibility Needs" toggle	A toggle should appear that allows users to enable or disable accessibility needs
3.15	The ETA Measure Button should function properly	Press the ETA Measure button	The ETA Measure button should be responsive and interactable.
3.16	Form Room Button should function accordingly v2	Press the Form Room Button and input a Form Room Number	Automatically add route to Form Room in Common Places Tab
3.17	The Theme selector should be functional and working as expected, clear and relevant toggle to switch between	Visual check for the Theme Selector and press it.	The Theme Selector should alternate as you toggle between it, from Light and Dark and this should be obvious and clear to the user.

	themes.		
3.18	The button to return to the Main Menu should function and be visible to users.	Visual check for option to return to Main Menu from Settings Page, Press the button.	The option should be clear and present to the user and it should take the user to the Main Menu when pressed.
3.19	Accessibility Needs Button should function accordingly v2	Press the Accessibility Needs button and toggle Yes Accessibility is necessary.	The routes which require use staircases/other unusable means, should be rerouted to nearest ramp/lifts.
3.20	Accessibility Needs Button should function accordingly v3	Press the Accessibility Needs button and toggle No Accessibility is necessary.	The routes which require use of staircases/other unusable means, should not be rerouted.
3.21	ETA Measure Button should function accordingly v2	Press the ETA Measure button and choose Hours, select a route to specific location.	The route should be calculated and the ETA should be in Hours (to the nearest 10 minutes).
3.22	ETA Measure Button should function accordingly v3	Press the ETA Measure button and choose Minute, select a route to specific location.	The route should be calculated and the ETA should be in Minutes (to the Nearest Minute).
3.23	ETA Measure Button should function accordingly v4	Press the ETA Measure button and choose Seconds, select a route to a specific location.	The route should be calculated and the ETA should be in Seconds, (to the nearest Minute)

Robustness/Functionality Test Plan for Post-development

Post-development testing (black-box testing) which takes place once the system is complete. Here we are testing the Robustness of the program and the main functionality of its features. The robustness looks at how durable and logical the program is to any input and output, e.g. Does your Login details save when the application is reset? This section is purposed to answer many of these questions which will help discover any gaps and holes in my SchoolNav for feedback and also further development.

Test ID	Test Description	Test Method/Data	Purpose/Justification
R1	<p>At the initial start of the application, the user should be prompted to Login/Sign-up.</p> <p>This function should work effectively and settings/searches within the application should be saved into account details.</p>	<p>Login/Sign-up Page should be the first page on initial start-up.</p> <p>I should be able to Log-in to the application and save my account settings to that account, which I can access if I log-out and log-back in.</p>	<p>The Login/Sign-up Page being the first screen allows Users to create an account and go into the Main-menu consequently.</p> <p>This is to ensure settings can be transferred along devices and saved.</p> <p>This prevents users from having to input details frequently.</p>
R2	After logged-in, subsequent program launches should open in Main-Menu	<p>When a user logs in, they should be directed to the Main Menu Page.</p> <p>After relaunching the application, users should be directed to Main-Menu straight away.</p>	<p>This is to ensure the user can manoeuvre around the application and perform its main functionality instantly.</p> <p>The main-menu holds all the main features and buttons for users.</p>
R3	The account user is on should not reset at any point.	For every launch of the application after the initial launch (where the user logs in/creates an account) the	This is to prevent unnecessary log-in attempts which can irritate the user and cause a hassle.

		application should not log out and continue on that account.	
R4	The settings of user should not change/reset once application is relaunched	The settings should be linked to the user account and since the account should be constant, unless logged out manually, the settings should be constant and not reset.	This is to improve usability and prevent unnecessary hassle within the application for the user, as assigning the settings again is time-consuming and tedious.
R5	Reported features within the School Map should be constant, until time-limit expires	The reported features, reported by users should appear and be constant, even if application resets, it should be updated to the school map online and be reset after 24 hours.	This is to ensure other users can see the reported features and that there won't be an overload of reports due to non-expiring ones, which are constant and linked to the School Map.
R6	Routes should reset/stop if application is reset	If the user escapes the application and closes it whilst a route was running/calculated, the application should cancel the application and flush any routes.	This is to ensure there are no bugs and processing power used in the background to calculate routes when the user doesn't require them etc.

Usability Test Plan for Post-development

Similar to the Robustness/Functionality Test Plans we can test the program against the Usability Test Plans after the completion and development of the program. The Usability Tests are going to be tested with a stakeholder of SchoolNav. These include Teachers, Students, Parents and Guests. Looking at how they use and work within the application will aid in my further development later along the line of the application.

The Test Method/Data acts as a way to check for the success of that Usability Test ID and using the result the stakeholder gets back determines the success and completion of that Test ID.

Test ID	Test Description	Test Method/Data	Purpose/Justification
U1	The application has functional buttons and responsive controls.	Each button should take you to the appropriate page or perform a specific function.	This to ensure the program is usable and accessible to the user.
U2	Settings should be easily accessible and visible, when in Main Menu Screen	The settings icon should function to present user Settings's Page and it should be apparent to where the setting's can be found, it should be in the top-right corner.	The setting's page is a vital component of my application as it allows users to select School and Personal Information which are key to the applications usability. These settings hold the input sections where users can input whether they require accessibility routes or not, which is an important factor to School vs usability.
U3	Easy to understand and self-explanatory sections and buttons	Every button should be clearly labelled, with accurate and descriptive labels which reflect the functionality of the button.	The purpose of each button should be obvious to the user to ensure usability.

		Each page should be labelled and described to ensure usability within the application.	
U4	The school map should reflect accurately and descriptively the location of user and route to take	<p>Clear arrows and routes with highlighted lines across the map corridors.</p> <p>Description and labelling of each location/room user crosses/will cross.</p>	This is to ensure the user knows where they are going and can reflect/follow the directions in real-life.
U5	Each screen should be easy to get to and indicative of its purpose.	<p>Check the buttons which lead to each page to ensure they are easily identifiable and functional.</p> <p>Test the responsiveness of the buttons, ensuring they work across different devices and screen sizes.</p> <p>Test the flow of navigation between pages to ensure it is logical and seamless, with minimal effort required to move from one section to another.</p>	<p>This is important since each page is necessary and vital in its own way for the user, so getting to those pages and knowing the individual purpose is an important feature that should be tested.</p> <p>It's crucial to recognize that each page in an application serves a unique and essential function for the user, contributing to their overall experience and ability to navigate the system. For example, the login page is the gateway to access the app, and it needs to be intuitive to ensure users can easily log in and proceed. If this page is confusing or hard to access, users could become frustrated and leave the app. Similarly, the signup page is necessary for new users, and its purpose needs to be clear from the start, with labels like "Register"</p>

Development of Coded Solution

Initialisation

I have taken up the decision to use JavaScript-based React Native software (a consequence of React). It is a platform to build native apps using JavaScript. This is because React offers compatibility with multiple devices using the same code, reducing project timeline and code to be written. React also offers features to customise and offer a great user experience, both which are vital to my applications usability and target market usage.

I used VSCode to code my application. I will use my phone and a virtual phone emulator, with IOS and Android to test and run my program code to see if test plans have been met. This IDE allows me to easily edit and save my code and refresh the application in my physical phone emulator.

To get started with React Native, I installed packages like Node.js using NPM (Node Package Manager) in Command Prompt.

Setting Up Coding Environment

Libraries, Packages and Extensions

Initially the libraries/functions I started of was:

```
import React from 'react';  6.9k (gzipped: 2.7k)
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';
```

```
npm install -g expo-cli
expo init SchoolNav
npm install react react-native
```



Description:	<p>The libraries shown in the top image are the default imports that a React Native file includes by default. The React component function was used to access the native components and language features that I will leverage in my project.</p> <p>Expo is external software I will mention later on, it allows me to emulate a virtual device. The {StatusBar} object is used to customise and acknowledge the constant status bar which screen real estate is allocated to at the top of the screen.</p> <p>From the react-native library, the components StyleSheet, Text and View were imported. This allows my code to be styled and viewed accordingly.</p> <p>A software package is a collection of related computer programs that can be licensed, downloaded, or subscribed to as a single bundle.</p> <p>npm (Node Package Manager) is a tool that helps manage and install packages (libraries or tools) for JavaScript projects. It allows you to easily download, update, and manage dependencies in your project. With npm, you can install packages from the npm registry, track them in a `package.json` file, and use them in your code. It also helps with version control, ensuring that the right versions of libraries are installed for your project. In short, npm simplifies the process of integrating and maintaining third-party packages in your Node.js or JavaScript applications.</p> <p>To start coding in React.Native I need to install the relevant libraries and I installed extensions in VSCode to help organise and benefit my coding.</p>
--------------	---

Coding Conventions and Testing Methods

Comments will look like this:

```
// Comments through out the development will look like this
```

```
{/* A JSX Comment */}
```



```
console.log("App executed");
```

Description:

The purpose of the comments is to sign-post the meanings and functions of certain codes.

Variable names will be named using CamelCase to assist with readability and flow of code. And I will use all Uppercase to signify constant variables since in Javascript I can make use of Constant variables as well as standard assignable variables.

I will be including screenshots of testing on IOS emulator devices after version, to develop and directly test code often, I will be using an Android Virtual Emulator using the [Android Studio Software](#).

I am using a Pixel 9 XL to replicate an Android Testing Device, it is being emulated on my PC.

Using the console.log() function for testing purposes in the terminal window of VS Code.

Using the npm manager I launched the testing emulator for an android device. This was the code of successful download and the bridgeless mode enables me to replicate and test through an android device without having to physically connect my device.

Code for connecting to Android Emulator using Expo and what to expect:

```
PS C:\Users\iguana\SchoolNav> npm start
```

```
> schoolnav@1.0.0 start  
> expo start
```

```
Starting project at C:\Users\iguana\SchoolNav
```

```
Starting Metro Bundler
```

```
> Metro waiting on exp://192.168.1.170:8081  
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)
```

```
> Using Expo Go  
> Press s | switch to development build
```

```
> Press a | open Android  
> Press w | open web
```

```
> Press j | open debugger  
> Press r | reload app  
> Press m | toggle menu  
> shift+m | more tools  
> Press o | open project code in your editor
```

```
> Press ? | show all commands
```

Logs for your project will appear below. Press Ctrl+C to exit.

```

> Opening on Android...
> Opening exp://192.168.1.170:8081 on Pixel_4_API_35
Downloading the Expo Go app
[=====]a\SchoolNav\index.js (656 modules)
(NOBRIDGE) LOG Bridgeless mode is enabled
INFO
💡 JavaScript logs will be removed from Metro in React Native 0.77!
Please use React Native DevTools as your default tool. Tip: Type j in
the terminal to open (requires Google Chrome or Microsoft Edge).

```

This terminal response downloads Expo Go automatically onto the target device, Expo Go is an app which allows the React.native Code to be run directly and wirelessly without a USB device being connected.

The fast refresh feature on React.Native allows my changes to appear as soon as I press save, this allows easy testing and manipulation of code.

Iterative Development of Coded Solution + Testing

Default Code and Initial Run Test

Default code:

```

import React from 'react';
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}

```

```
}
```

```
const styles = StyleSheet.create({
```

```
  container: {
```

```
    flex: 1,
```

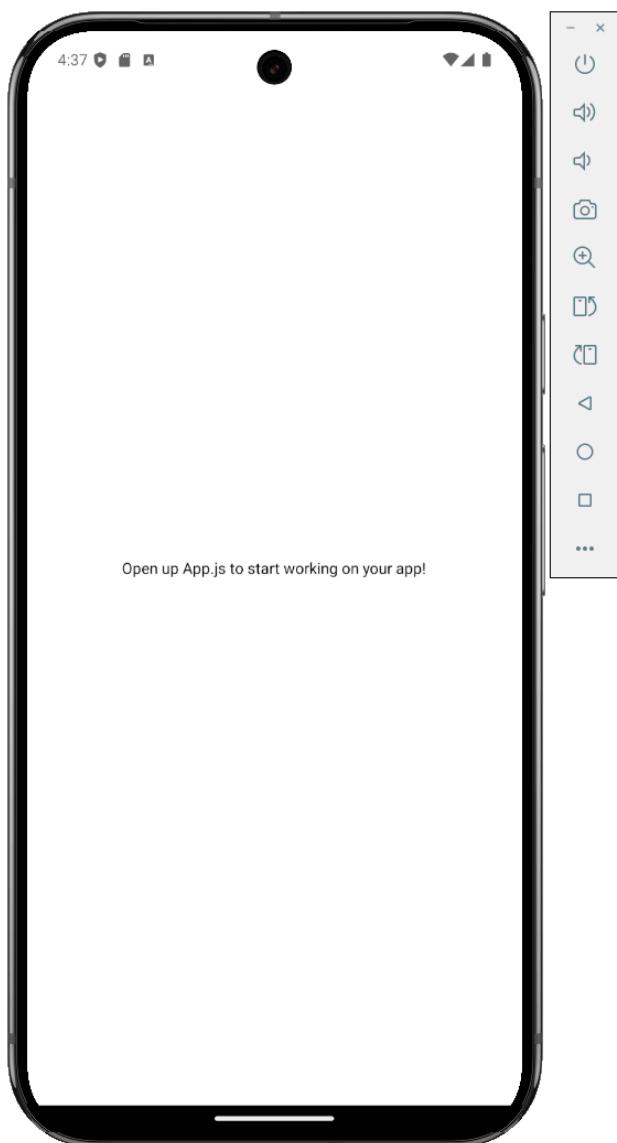
```
    backgroundColor: '#fff',
```

```
    alignItems: 'center',
```

```
    justifyContent: 'center',
```

```
  },
```

```
});
```

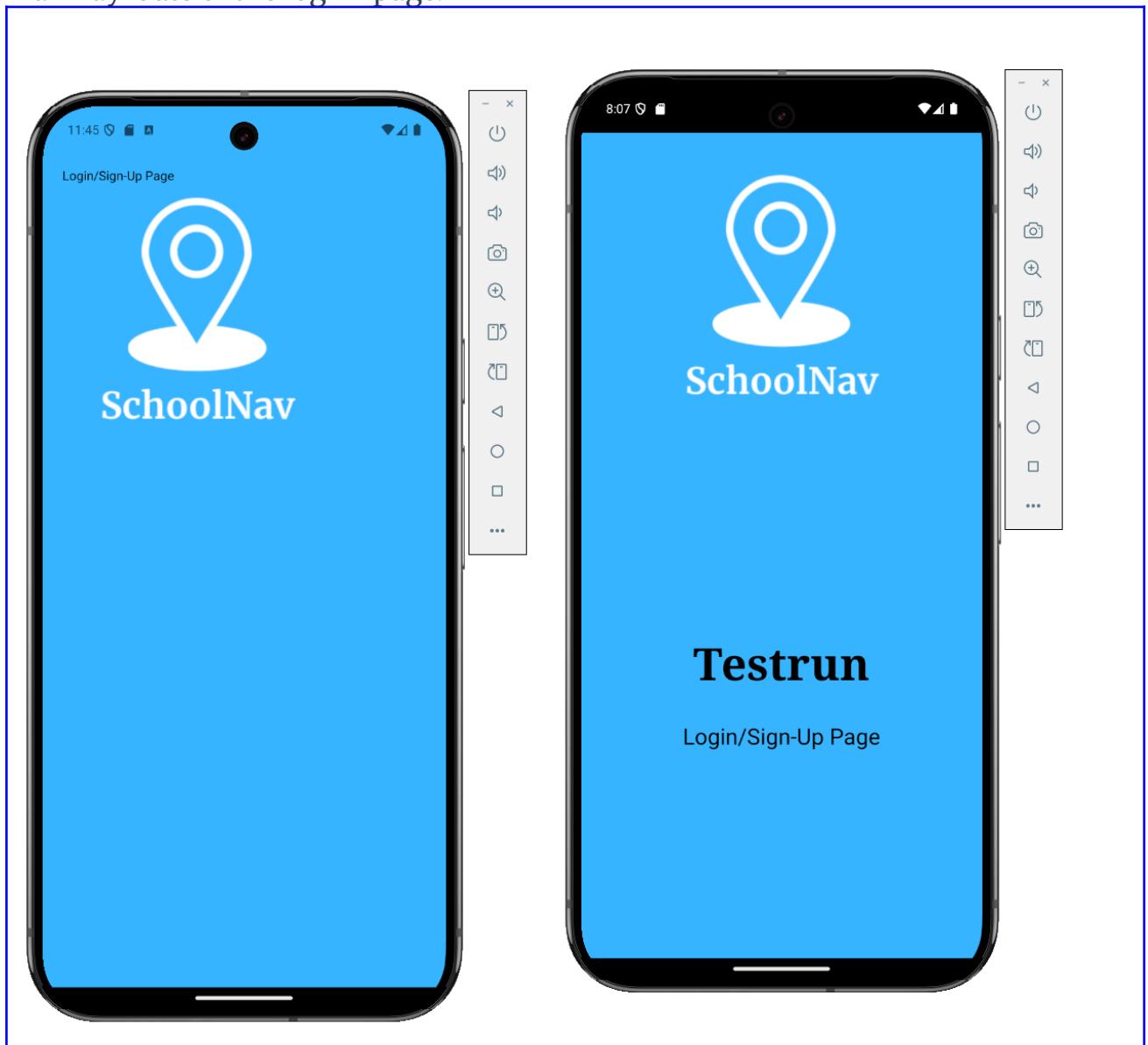


Description:	<p>This blank canvas defaultly was coded into the SchoolNav file I initialised.</p> <p>This test run presents the functionality and testing capabilities of Expo and Android Studios' Virtual Device Emulator.</p>
--------------	--

Version 1 - Initialise Login Screen

Introduction:

This iteration aims to initialise and launch the main working grounds for the log-in screen adding text input boxes for the email and passwords and structuring the main lay-outs of the log-in page.



Relevant Source Code:

```
const styles = StyleSheet.create({
  container: {
    paddingTop: '15%',
    paddingRight: '5%',
    paddingLeft: '5%',
    flex: 1,
    backgroundColor: '#38b6ff',
    // alignItems: 'center',
    // justifyContent: 'center',
  },
});
```

More relevant code is found below, this is the code different to the default code and is relevant to the first screenshot on the left.

After removing the commented:

```
// alignItems: 'center',
// justifyContent: 'center',
```

and adding those components to my code I have gotten the screenshot of the right hand side.

Description:

Initially I had my background colour to be dodger blue, since I did not match the background colour in my Designs nor the logo I swapped the dodger blue for the Light Blue hexadecimal code which matches the Logo.png icon file colour: '#38b6ff'

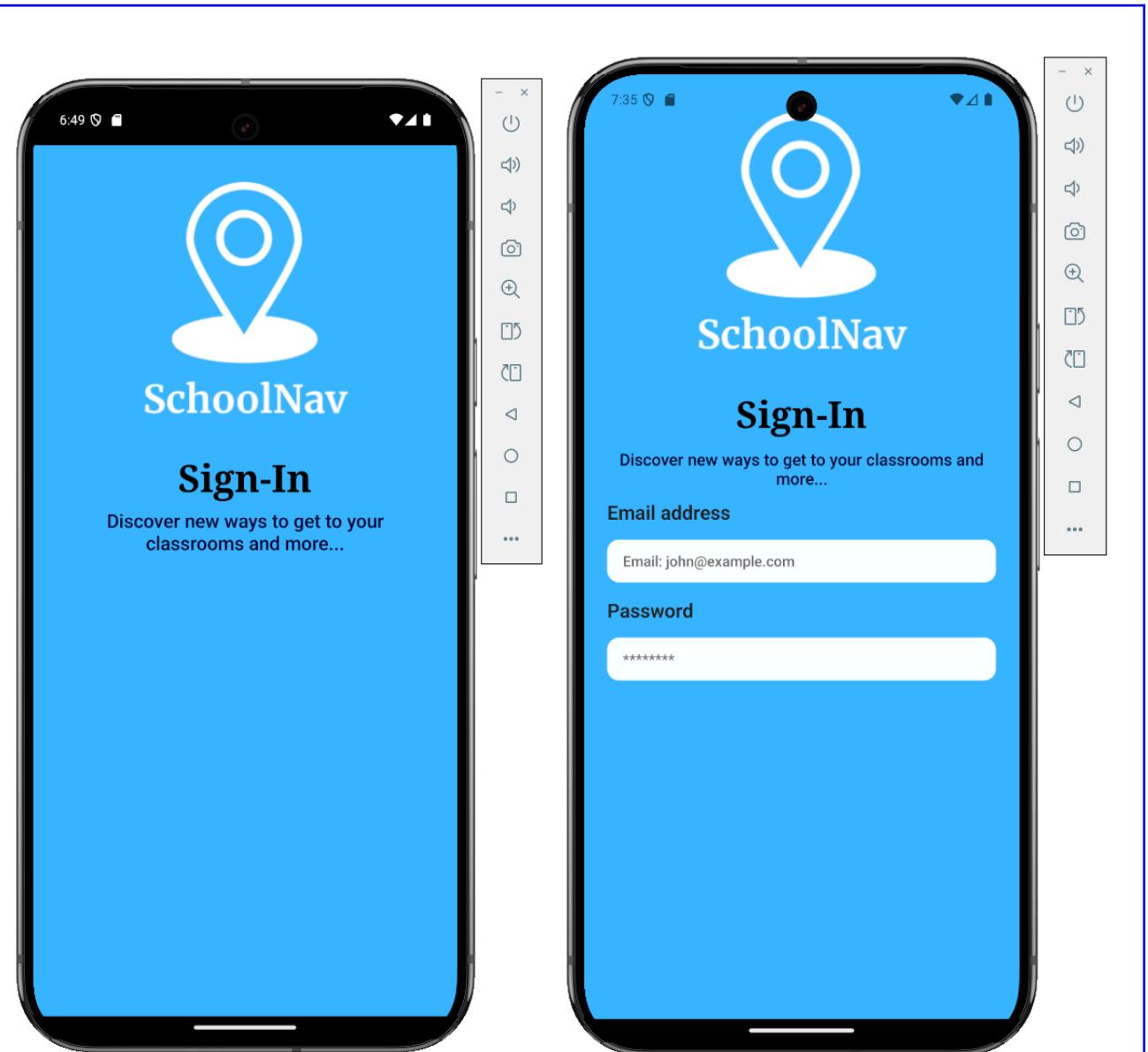
I tested the padding of the screen by removing the AlignItems property using the comment feature. The results that allow readable and clear outputs is the following configuration for the StyleSheet. The padding is to ensure the text and other outputs don't get blocked by the information panel located at the top of the phone and the camera punch hole/notch most phones have. These implementations contribute to the Success of SC.0

I tested the padding of the screen by removing the AlignItems property using the comment feature. The results that allow readable and clear outputs is the following configuration for the StyleSheet. The padding is to ensure the text and other outputs don't get blocked by the information panel located at the top of the phone and the camera punch hole/notch most phones have.

The use of flexbox to position the items in my application, allows

the program to be dynamic and fits to each unique screen. This reduces my project timeline from having to individually set-up each screen proportions.

Get rid of `<StatusBar style="auto" />` since it causes the status bar to be a part of the application and excluding it will assist with viewing the status bar components.



Relevant Source Code:

```
import React, { useState } from 'react';
import { StatusBar } from 'expo-status-bar';
import {
  StyleSheet,
  Text,
```

```
View,
TextInput,
TouchableWithoutFeedback

back,
Image,
TouchableOpacity
} from 'react-native';

export default function App() {

  const [form, setForm] = useState({
    email: '',
    password: '',
  })
  console.log("App executed");

  return (
    <View style={styles.container}>
      <View>
        <TouchableOpacity>
          <Image
            blueRadius={10}
            fadeDuration={1000}
            source={require('./assets/SchoolNavLogo.png')}
            style = {styles.headerImg}
            alt="Logo"
          />
        </TouchableOpacity>
        <Text style={styles.title}>Sign-In</Text>
        <Text style={styles.subtitle}>
          Discover new ways to get to your classrooms and more...
        </Text>
      </View>
    </View>
  );
}
```

```
<View style={styles.form}>
  <View style={styles.input}>
    <Text style={styles.inputLabel}>Email address</Text>

    <TextInput
      autoCapitalize='none'
      autoCorrect= {false}
      style={styles.inputControl}
      placeholder='Email: john@example.com'
      value={form.email}
      onChangeText={email => setForm({ ...form, email})}
    />
  </View>

  <View style={styles.input}>
    <Text style={styles.inputLabel}>Password</Text>

    <TextInput
      secureTextEntry={true}
      style={styles.inputControl}
      placeholder='*****'
      value={form.password}
      onChangeText={email => setForm({ ...form, password})}
    />
  </View>
</View>
);

}

const styles = StyleSheet.create({
  container: {
    padding: 24,
    flex: 1,
```

```
    backgroundColor: "#38b6ff",
},
headerImg: {
  marginBottom: 0,
  alignSelf: 'center',
},
title: {
  color: '#lelele',
  fontSize: 38,
  textAlign: 'center',
  fontFamily: 'serif',
  fontWeight: 'bold',
},
subtitle: {
  marginVertical: 10,
  fontSize: 17,
  fontWeight: '500',
  color: '#000033',
  textAlign: 'center'
},
input: {
  fontSize: 12
},
InputLabel: {
  marginBottom: 15,
  fontSize: 20,
  fontWeight: '600',
  color: '#222'
},
inputControl: {
  height: 44,
  marginBottom: 15,
  backgroundColor: '#ffffff',
  paddingVertical: 10,
  paddingHorizontal: 16,
  borderRadius: 12,
  fontSize: 15,
  fontWeight: '500',
  color: '#222'
```

```

},
form: {
  marginBottom: 24,
  flex: 1,
},
formAction: {
  marginVertical: 24,
},
btn: {
  backgroundColor: '#075eec',
  borderRadius: 8,
  borderWidth: 1,
  borderColor: '#075eec',
  flexDirection: 'row',
  justifyContent: 'center',
  alignItems: 'center',
  paddingVertical: 10,
  paddingHorizontal: 20,
},
btnText: {
  fontSize: 18,
  fontWeight: '600',
  color: '#fff',
} })

```

	<p>Description:</p> <p>To help me know the app is being executed every time I change the code (since Expo has a fast refresh feature which automatically updates the application emulator with changes to the code), I added the “App executed” line to my terminal window using <code>Console.log</code>.</p> <p>The logo image was imported and saved as a png file and is being outputted here. Using the <code>TouchableOpacity</code> feature the logo image is interactable which adds to the user friendly nature of my application.</p>
--	---

I also added an alternative output if the image for whatever reason was not able to be uploaded, which is the text “Logo”.

The addition of the logo and aesthetically pleasing design hence, leads to the completion and meeting of SC.15 and SC.16:

Logo Display should be prominent and visible to users, indicating the app and its purpose.

Aesthetically pleasing and functional design to the user which promotes the user experience.

I added the Textboxes for the title and the subtitle to encourage users to sign-in and also to utilise the application.

For the text-input I made it so autoCapitalisation is off, since for email and password this is not necessary. I also changed the keyboardType to ‘email address’ since this will output certain keys on the keyboard which are most common to emails e.g @. To prevent any confusion on what to input in this box, I added a placeholder holding an example email and set the values of the input to Variable name email.

In React Native, the **keyboardType** prop of a **TextInput** specifies the type of virtual keyboard that should be displayed when the user interacts with the input field.

Setting **keyboardType='email-address'** ensures that the keyboard is optimized for entering email addresses.

The use of TouchableOpacity contributes to the meeting of Success Criteria SC.01 where:

The application should have a smooth and responsive user interface (UI) system.

By initialising the Log-In Page and setting the default page as the Login/Sign-In page we are effectively meeting Success Criteria SC.06 where;

	<p>The Login/Sign-up page should be the first screen for the users of the application, but the user should have the ability to use the app as a guest</p> <p>Later on this project I will return to meeting SC.06 where the guest option is initialised and implemented.</p>
--	--

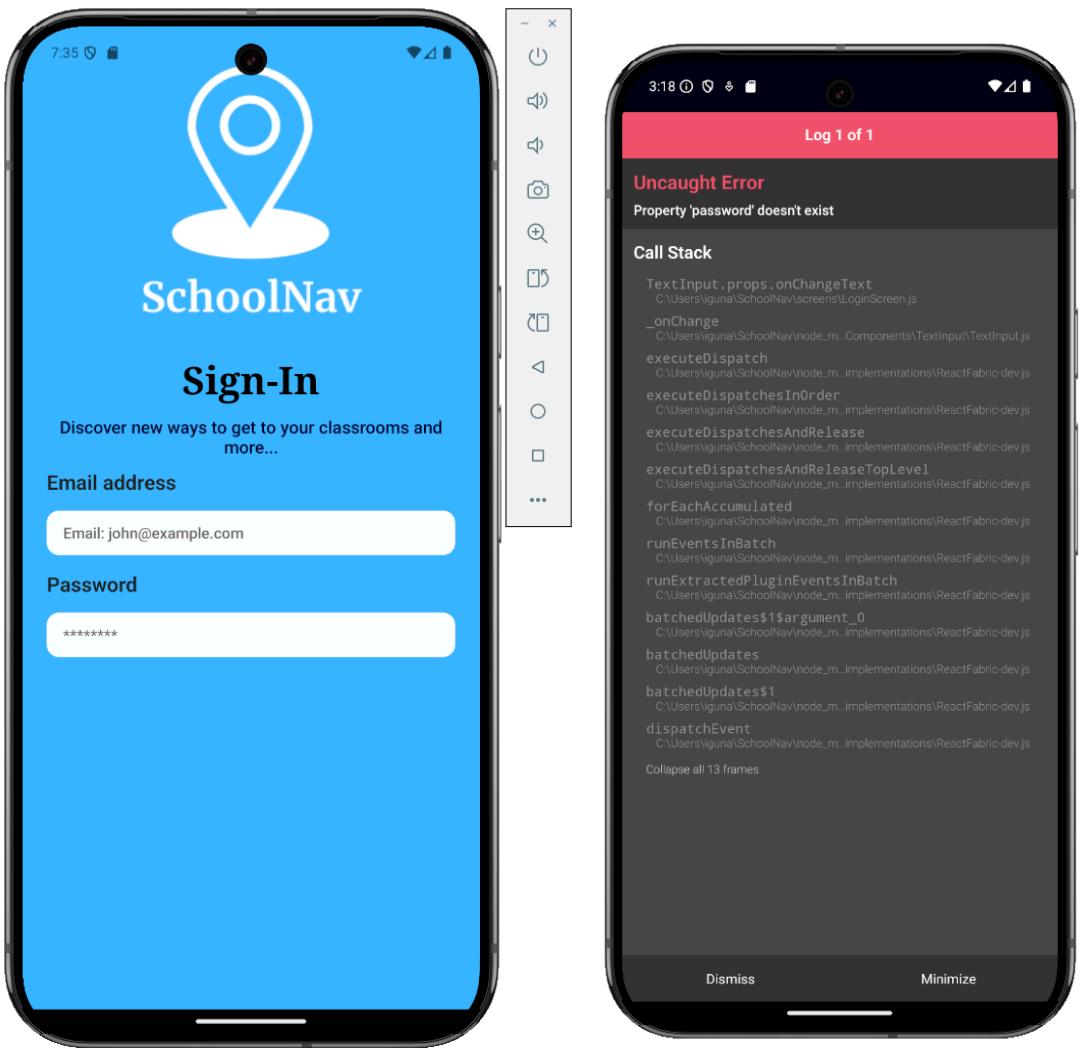
Test Table 1 - Initialising Log-In Screen

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.1	Application should run	Start the application and see if anything is visually seen as expected.	Expected text, images and boxes should be output.	Expected results are presented onto the emulator.	Yes, the application runs!
1.2	Should display background image/colour	Log-in/Sign-Up Page , visual check for background image.	A blue background image presented in the User Interface Section.	A blue color which matches the background of the logo is the background colour of the screen.	Yes, background colour is displayed as the Designs section suggests.
1.3	The Log-in page should be the first page for new users who haven't yet logged-in/created account as stated in SC.06	Log out if logged and relaunch application/relaunch application.	Visual check that the screen present is Log-In/Sign-up Page.	Relaunching the app the Log-In Screen is the first screen.	Yes – but need to revisit this once there are more screens and once the application can save accounts.
1.4	Should display Logo clearly on top of background as	Check if the SchoolNav logo is displayed.	Logo referenced in User Interface	The Logo is cut off by the camera punch hole.	No HyperLink

	a png as stated in SC.15		should be displayed on top of Log-In Page. Use User Interface: Log-in/Sig n-Up Page for reference.		to Solution
1.5	Beneath the Logo, should appear Title for Sign-In	Check if the title to Sign-In is presented.	Sign-In text should appear below Logo Icon. Use User Interface: Log-in/Sig n-Up Page for reference.	The title is presented below the logo with an according to the CSS style.	Yes, Title is positioned and displayed correctly.
1.6	Add a subtitle below the title to greet the users and inform them of the application	Visual check for according subtitle	Subtitles should appear below the title and be descriptive of the app's purpose.	The subtitle is presented and it states: "Discover new ways to get to your classrooms and more..."	Yes, greeting is presented and formatted correctly.
1.7	The input for the email address should be clear and obvious to the user	Visual check for email address box and title.	Present should be a text box for an email address with a placeholder and appropriate title.	The email address input box is clear and presented to the user, supported with the according title. The placeholder used is "Email: john@example.com".	Yes, the placeholders indicate the email address input clearly.

1.8	The input for the password should be clear and obvious to the user	Visual checks for the password input and title.	Present should be a textbox for password with a placeholder and appropriate title.	The password input box is clear and presented to the user, supported with the according title. The placeholder used is “*****”.	Yes, the placeholders indicate the password input clearly.
1.9	Text should be able to be input into email address and password	Attempt typing into each of the fields and the values should be saved as email/password	The text input should open the keyboard to allow typing and the text should appear for each box.	Typing into the email field works properly however the password field was producing error messages.	<p>No</p> <p>HyperLink to Solution</p>

Evidence Table 1

Test ID	Screenshot/Video
Test ID 1.1 – 1.9	
Evaluation:	<p>The application is running and projecting the output onto the emulator/device as expected meeting 1.1.</p> <p>This version fails to meet Test ID 1.4 as some of the logo is being cut off by the camera punchhole.</p> <p>There is an error with Test ID 1.9, the password field, where text cannot be inputted as efficiently.</p> <p>This version covers mainly up to Test ID 1.9.</p>

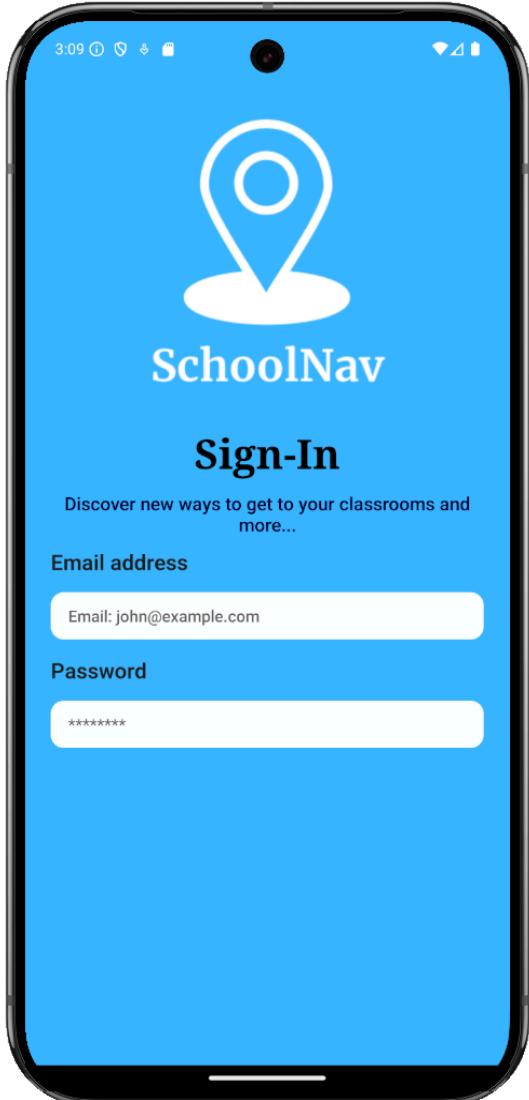
	<p>This is in the process of meeting SC.18 where:</p> <p>The username and password fields should work from the saved register screen</p>
<p>The second image showcases the error message presented when any text is inputted into the password field. This error message was also logged onto my terminal window. The error stating “password” is defined, leading to the fact that I missed to define/properly state the password property. I will solve this issue in the next section.</p>	

Fixing Logo cut off issue: Test ID 1.4	
Relevant Source Code:	<pre>import { SafeAreaProvider, SafeAreaView } from 'react-native-safe-area-context'; <SafeAreaView style={styles.container}> </SafeAreaView></pre>
Description:	<p>I used SafeAreaProvider and SafeAreaView from the react-native-safe-area-context library to ensure the screen content respects device safe areas, such as notches and status bars. Wrapping the main code in SafeAreaView applies automatic padding, creating a consistent layout across devices.</p> <p>This fixed the issue where the logo was outside the SafeAreaView</p>

Test Table 2 - Fixing Logo cut off issue					
Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.4-F IX	Should display Logo clearly on top of background as a png as	Check if the SchoolNav logo is displayed.	Logo referenced in User Interface should be displayed on	The logo can now be seen clearly on the screen and is not cut off by the camera	Yes - fixed from previous error The Logo is

	stated in SC.15	top of Log-In Page. Use User Interface: Log-in/Sign-Up Page for reference.	punchole.	presented as the Designs section requires, no blocking, it now adapts to the user display.
--	-----------------	--	-----------	--

Evidence Table 2

Test ID	Screenshot/Video	Evaluation
1.5		<p>The screen is now fully visible and the logo is now in the bounds of the application. This issue was vital to fix as the logo is a fundamental part of my application and if it is cut off it will look low quality and put off users of the application.</p> <p>This is meeting the SC.16:</p> <p>Aesthetically pleasing and functional design to the user which promotes the user experience.</p> <p>Where you can see how the screen is consistent and aesthetic meeting this SC.</p>

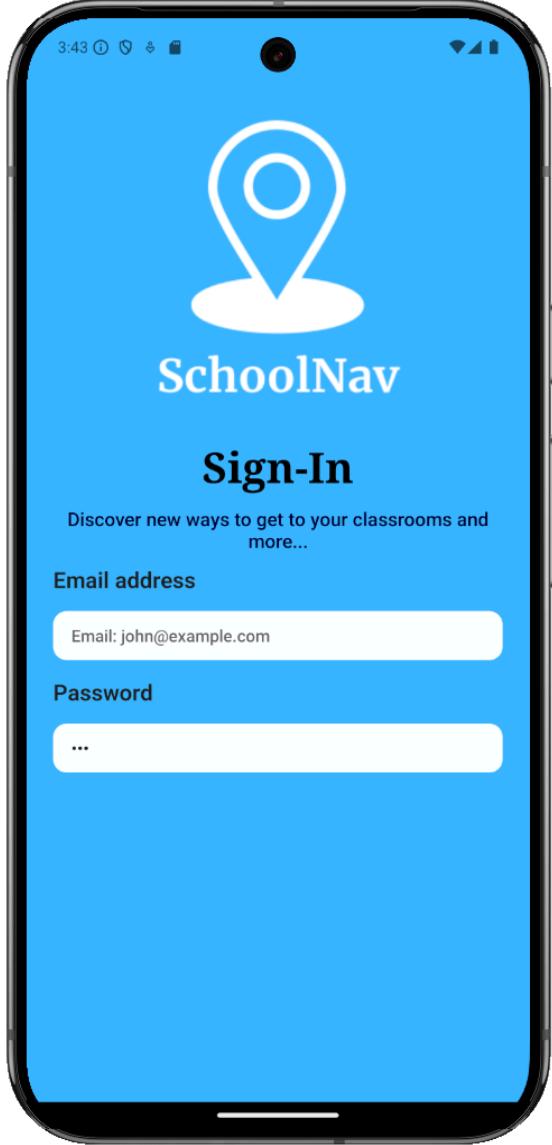
Fixing text input issue with password field: Test ID 1.9

Relevant Source Code:	<pre><TextInput secureTextEntry={true} style={styles.inputControl} placeholder='*****' value={form.password} onChangeText={password => setForm({ ...form, password})}> /></pre>
Description:	<p>Initially the value of the password was set to 'email' hence it was a logical error where it should have been 'password' instead of 'email'. This was the original code section:</p> <pre><TextInput secureTextEntry={true} style={styles.inputControl} placeholder='*****' value={form.password} onChangeText={email => setForm({ ...form, password})}> /></pre>

Test Table 3 - Fixing the input password field error

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.9 - FIX	Text should be able to be input into email address and password	Attempt typing into each of the fields and the values should be saved as email/password	The text input should open the keyboard to allow typing and the text should appear for each box.	Typing into the field now allows me to provide an input value for the password and no error messages are presented.	Yes - fixed from previous error Email address and Password Inputs now work!!

Evidence Table 3

Test ID	Screenshot/Video	Evaluation
1.9	 <p>The screenshot shows the SchoolNav app's sign-in screen on a smartphone. The screen has a blue background with a white location pin icon at the top. Below it, the "SchoolNav" logo is displayed. The main title "Sign-In" is centered in large, bold, white font. A subtitle "Discover new ways to get to your classrooms and more..." is shown below the title. There are two input fields: "Email address" containing "Email: john@example.com" and "Password" containing three dots (...). The phone's status bar at the top shows the time as 3:43.</p>	<p>The text is now able to be logged into the field and no error message appears.</p> <p>This is a fundamental point and fixing it allows the users to successfully login.</p>

Initialising Log-In Screen

Relevant Source Code:

```
import React, { useState } from 'react';
import { StatusBar } from 'expo-status-bar';
import { SafeAreaProvider, SafeAreaView } from
'react-native-safe-area-context';
import {
  StyleSheet,
  Text,
  View,
  TextInput,
  TouchableOpacity,
  Image,
  TouchableOpacity
} from 'react-native';

export default function App() {

  const [form, setForm] = useState({
    email: '',
    password: ''
  })
  console.log("App executed");

  return (
    <SafeAreaProvider>
      <SafeAreaView style={styles.container}>
        <StatusBar style="light" />

        <View>
          <TouchableOpacity>
            <Image
              blueRadius={10}
              fadeDuration={1000}
              source={require('./assets/SchoolNavLogo.png')}
              style = {styles.headerImg}
```

```
        alt="Logo"
      />
    </TouchableOpacity>
    <Text style={styles.title}>Sign-In</Text>
    <Text style={styles.subtitle}>
      Discover new ways to get to your classrooms and
      more...
    </Text>
  </View>

  <View style={styles.form}>
    <View style={styles.input}>
      <Text style={styles.inputLabel}>Email address</Text>

      <TextInput
        autoCapitalize='none'
        autoCorrect= {false}
        style={styles.inputControl}
        placeholder='Email: john@example.com'
        value={form.email}
        onChangeText={email => setForm({ ...form, email})}
      />
    </View>

    <View style={styles.input}>
      <Text style={styles.inputLabel}>Password</Text>

      <TextInput
        secureTextEntry
        style={styles.inputControl}
        placeholder='*****'
        value={form.password}
        onChangeText={email => setForm({ ...form,
password})}
      />
    </View>
```

```
        <View style= {styles.formAction}>
          <TouchableOpacity
            onPress={() => {
              //Handles onPress
            }}>
            <View style={styles.btn}>
              <Text style={styles.btnText}>Sign in</Text>
            </View>
          </TouchableOpacity>
        </View>

      </View>
    </SafeAreaView>
  </SafeAreaProvider>
);
}
const styles = StyleSheet.create({
  container: {
    padding: 24,
    flex: 1,
    backgroundColor: "#38b6ff",
  },
  headerImg: {
    marginBottom: 0,
    alignSelf: 'center',
  },
  title: {
    color: '#lelele',
    fontSize: 38,
    textAlign: 'center',
    fontFamily: 'serif',
    fontWeight: 'bold',
  },
  subtitle: {
    marginVertical: 10,
    fontSize: 17,
    fontWeight: '500',
    color: '#000033',
  }
})
```

```
    textAlign: 'center'
},
input: {
  fontSize: 12
},
inputLabel: {
  marginBottom: 15,
  fontSize: 20,
  fontWeight: '600',
  color: '#222'
},
inputControl: {
  height: 44,
  marginBottom: 15,
  backgroundColor: '#ffffff',
  paddingVertical: 10,
  paddingHorizontal: 16,
  borderRadius: 12,
  fontSize: 15,
  fontWeight: '500',
  color: '#222'
},
form: {
  marginBottom: 24,
  flex: 1,
},
formAction: {
  marginVertical: 24,
},
btn: {
  backgroundColor: '#075eec',
  borderRadius: 8,
  borderWidth: 1,
  borderColor: '#075eec',
  flexDirection: 'row',
  justifyContent: 'center',
  alignContent: 'center',
  paddingVertical: 10,
  paddingHorizontal: 20,
```

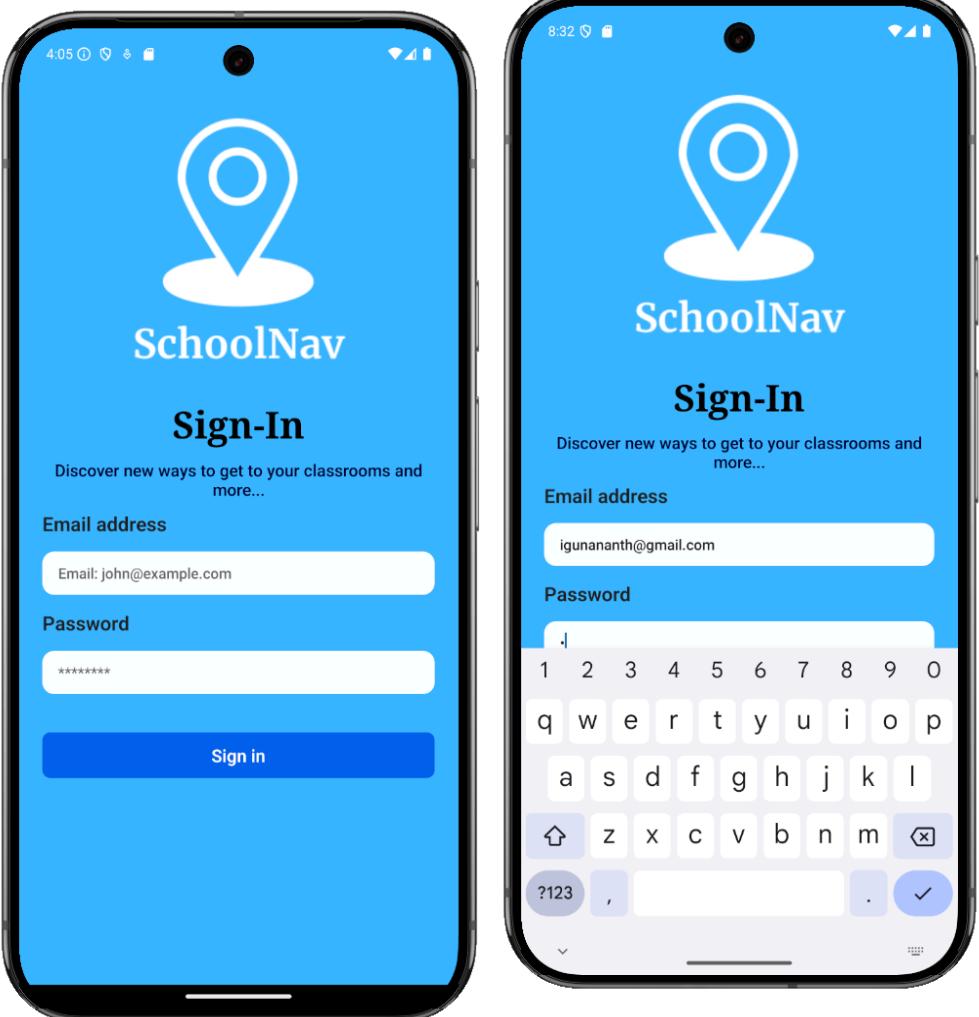
	<pre> }, btnText: { fontSize: 18, fontWeight: '600', color: '#fff', }) </pre>
Description:	<p>Mainly the difference is that the Sign-In button was added and formatted.</p> <p>The button for Sign-In is using Touchable Opacity which allows interaction with the users and it has the onPress function which I will add to later on, this will be used for the navigation between screens.</p> <p>To meet SC.16 and Test ID 1.11 the <code><StatusBar style="light" /></code> was added to ensure the icons at the top of the screen were matching to the theme of the screen (light).</p>

Test Table 4

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.10	Text should be encrypted when writing in the password field	Type into the password field and visual check for encryption symbols to prevent others from seeing the password.	The text typed into the field should be replaced with encryption symbols to prevent others from seeing the password.	Typing into the password field by default it is encrypted.	Yes, the test for the password is now hidden by default to preserve privacy.
1.11	Screen should have matching tone and clear theme	Visual check for consistent blue and white theme for Login Screen.	Consistent blue and white theme for Login Screen.	The phone and the screen both have a clear and consistent theme of blue and white.	Yes, there is a clear theme in the app.

1.12	Sign-In button should be prevalent and clear to the user	Visual check for the Sign-In button	The Sign-In button should be present below the fields for input.	The Sign-In button is not always clear and visible. It is viewable when the keyboard is not in use, hence if users want to actually Sign-In after submitting a password, they wouldn't be able to.	<p>No</p> <p>HyperLink to Solution</p>
------	--	-------------------------------------	--	--	--

Evidence Table 4

Test ID	Screenshot/Video
1.10 - 1.12	
Evaluation:	<p>The icons at the top of the screen often referred to as the Status Bar is now in the theme 'light' and we can customise this dynamically later on in the project.</p> <p>The Sign-In button is initially prevalent and clear to the user and has the option to be touched and interacted with however once you try inputting a password you will find that the Sign-In button gets enveloped with the keyboard and there is no clear solution to manoeuvre out of this.</p>

Fixing Logo cut off issue: Test ID 1.5

Relevant Source Code:

```
import {
  StyleSheet,
  Text,
  View,
  TextInput,
  TouchableWithoutFeedback,
  Image,
  TouchableOpacity,
  Touchable,
  Platform,
  KeyboardAvoidingView,
  ScrollView
} from 'react-native';

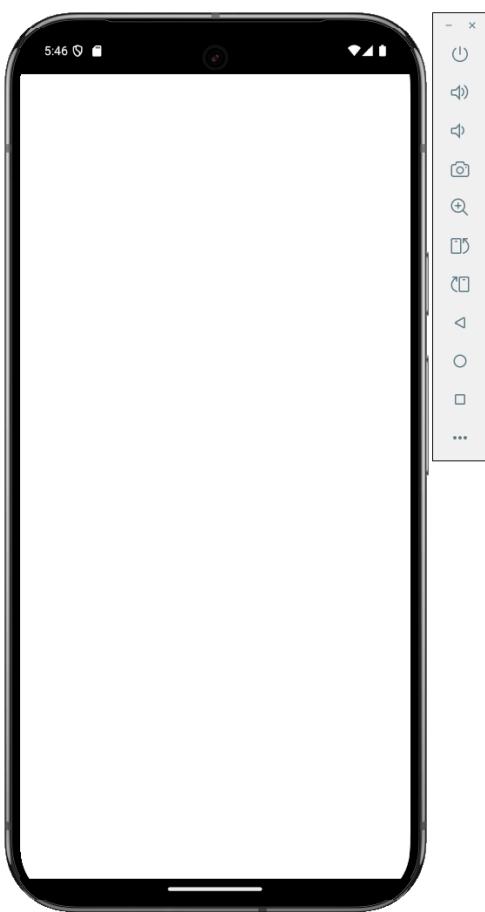
<KeyboardAvoidingView
  behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
  style={{ flex: 1 }}
  keyboardVerticalOffset={20}
>
  <ScrollView
    contentContainerStyle={{ flexGrow: 1 }}
    keyboardShouldPersistTaps="handled"
    showsVerticalScrollIndicator={false}
  >

  </ScrollView>
</KeyboardAvoidingView>
```

Description and images:

The password input does not move up to match the keyboard, the keyboard obstructs the visibility of the application, hence the solution was to use the **KeyboardAvoidingView** component.

Applied keyboardAvoidingView on the outmost parent tag for Login screen return function and got outputted a blank screen.



```
<KeyboardAvoidingView  
    behavior={Platform.OS === 'ios' ? 'padding' : 'height'}  
    style={styles.container}>  
  
</KeyboardAvoidingView>
```

I have included the relevant source code above and will explain it here:

Importing **KeyboardAvoidingView** and **ScrollView** to my screen to allow me to use it.

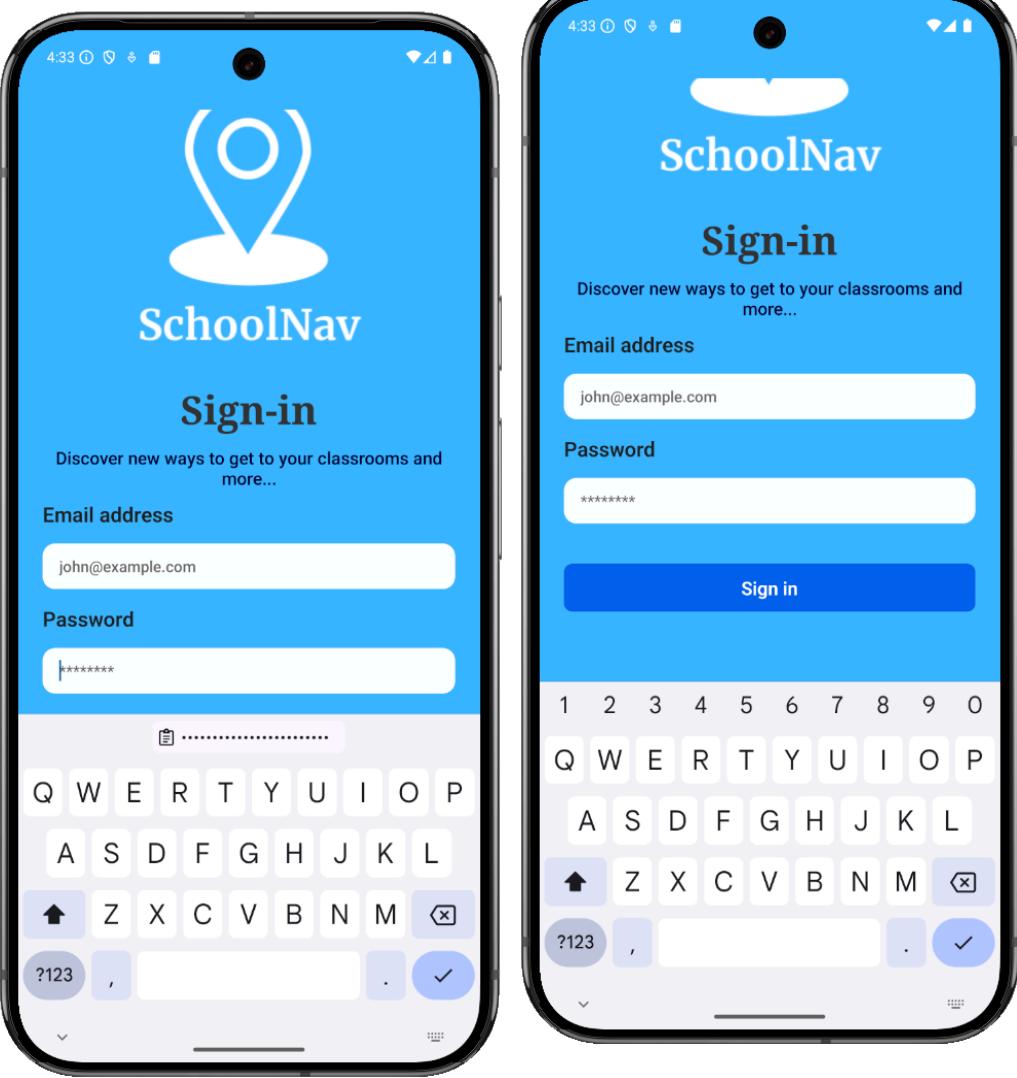
This is in regards to the **KeyboardAvoidingView**, using a ternary operator I can choose padding or height in accordance with the OS of the system.

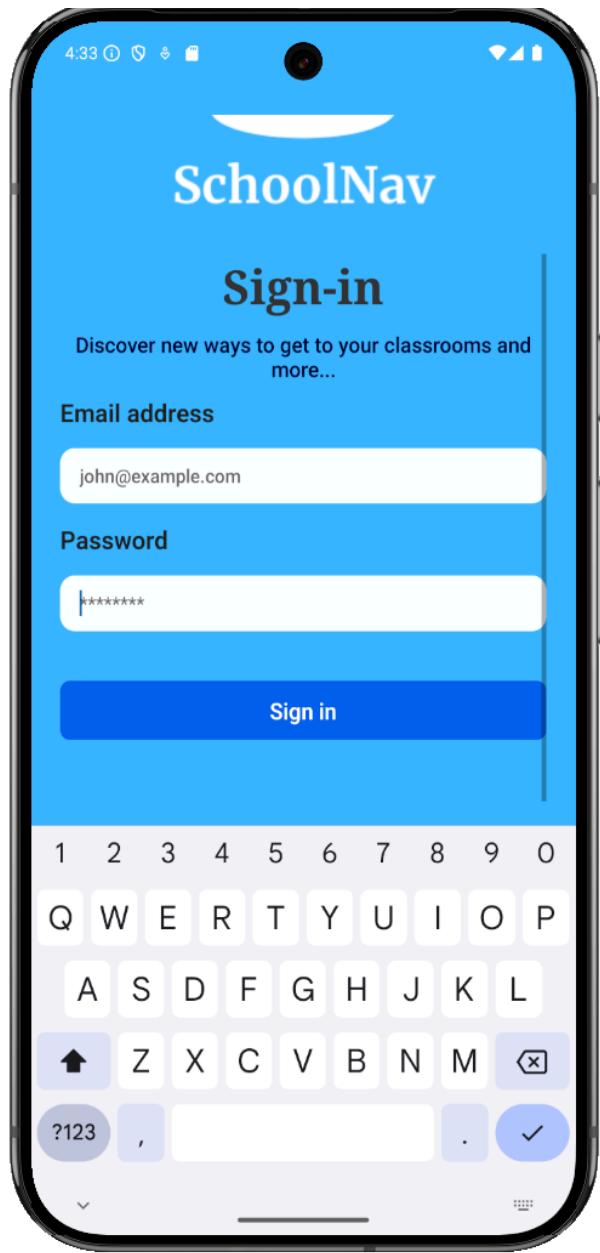
The reason why **KeyboardAvoidingView** wasn't functioning properly was because I did not have the **ScrollView** component wrapped around my parent view function. Hence I wrapped the main elements in a **ScrollView** tag to allow Scrolling to allow the

	<p>KeyboardAvoidingView to work as expected.</p> <p>The code:</p> <pre>keyboardShouldPersistTaps="handled"</pre> <p>Determines when the keyboard should stay visible after a tap. 'handled', the keyboard will not dismiss automatically.</p>
--	--

Test Table 5 - Fixing obstructed Sign-In button due to pop-up keyboard

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.12 -FIX	Sign-In button should be prevalent and clear to the user	Visual check for the Sign-In button	The Sign-In button should be present below the fields for input.	The Sign-In page is always visible and clear to the user even when Keyboard is up as users can now scroll up to the Sign-In button to press it.	Yes - fixed from previous error The Sign-In button can now be accessed at all times and all places in the application.

Test ID	Screenshot/Video
1.12	 <p>The image displays two side-by-side screenshots of a mobile application's sign-in screen. Both phones have a blue header with the SchoolNav logo (a white location pin icon) and the word "SchoolNav". Below the header, both screens show the "Sign-in" title and a subtext: "Discover new ways to get to your classrooms and more...". The left phone shows the initial state of the form fields: "Email address" with placeholder "john@example.com" and "Password" with placeholder "*****". Both fields have a light gray background. The right phone shows the state after the user has entered "john@example.com" into the "Email address" field; the placeholder text is now gone, and the field has a white background. Both phones have a standard QWERTY keyboard at the bottom of the screen.</p>



In this screenshot you can see how the ScrollView component is working and you are able to present scrolling direction and ability to do so through this view. Video evidence of this can be found later on in this documentation.

Initialising Log-In Screen: Adding Sign Up option to the screen and a footer

Relevant Source Code:

```
<TouchableOpacity  
style={{marginTop: 'auto'}}  
onPress={() => {  
//Handles onPress  
}}>  
<View>
```

	<pre> <Text style={styles.formFooter}> Don't have an account? <Text style={{textDecorationLine: 'underline'}}>Sign up</Text> </Text> </View> </TouchableOpacity> </pre> <pre> formFooter: { fontSize: 17, fontWeight: '600', textAlign: 'center', letterSpacing: 0.3, }, </pre>
Description:	<p>This is the addition to the code which is below the Sign-In Button. The use of TouchableOpacity for the footer/sign-up button means it is interactable with the user and provides a confirmation to the user that they have pressed it.</p> <p>The onPress function is there to allow navigation between various screens and we can add to it later on.</p> <p>I have underlined the Sign-Up text specifically to make it stand out and appear more clearer observing Test ID 1.13</p>

Test Table 6 - Initialising Log-In Screen, Buttons etc.

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.13	The Sign-In button should be intractable with user	Press the Sign-In Button	The Sign-In button should be able to be pressed.	When pressing the Sign-In button we get an opacity effect which portrays the fact the button was pressed.	Yes, the Sign-In button is interactable .

1.14	Option to Sign-Up should be present and button should be pressable	Visual check for Sign-Up option and title to clearly define that option.	Present at the footer of the screen should be a Sign-Up option with a description.	At the bottom of the page we see text stating “Don’t have an account? Sign-Up” where the Sign-Up is specifically underlined. The button is pressable and also leaves an opacity effect.	Yes, the Sign-Up button is displayed and interactable.
------	--	--	--	--	--

Evidence Table 6

Test ID	Screenshot/Video	Evaluation
1.13-1.14	https://youtu.be/l31S7TbkRE	The video clearly represents the functionality of the application up to this point. It shows how the Sign-Up Footer/Button is present and pressable and how the keyboard is not a factor to prevent users from signing in correctly without any mishaps.

Version 2 - Validation for Log-In Screen + Other Key Components

Validation for Log-In Screen

Relevant Source Code:

```
// Import necessary React and React Native components
import React, { useState } from 'react';
import { StatusBar } from 'expo-status-bar';
import { SafeAreaProvider, SafeAreaView } from
'react-native-safe-area-context';
import {
  StyleSheet,
  Text,
  View,
  TextInput,
  Image,
  TouchableOpacity,
  Platform,
  KeyboardAvoidingView,
  ScrollView
} from 'react-native';

export default function LoginScreen() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [errors, setErrors] = useState({});

  const validateForm = () => {
    let errors = {};

    if (!email) errors.email = "Email is required.";
    if (!password) errors.password = "Password is required.";

    setErrors(errors);
  }
}
```

```
        return Object.keys(errors).length === 0;
    };

    const handleSubmit = () => {
        if (validateForm()) {
            console.log("Submitted", email, password);
            setEmail('');
            setPassword('');
            setErrors({});
        }
    };

    console.log("App executed");

    return (
        <SafeAreaProvider>
            <SafeAreaView style={styles.container}>
                <KeyboardAvoidingView
                    behavior={Platform.OS === 'ios' ? 'padding' :
                    'height'}
                    style={{ flex: 1 }}
                    keyboardVerticalOffset={20}
                >
                    <StatusBar style="light" />
                    <ScrollView
                        contentContainerStyle={{ flexGrow: 1 }}
                        keyboardShouldPersistTaps="handled"
                        showsVerticalScrollIndicator={false}
                    >
                        <View>
                            <TouchableOpacity>
                                <Image
                                    fadeDuration={1000}

source={require('./assets/SchoolNavLogo.png')} style={styles.headerImg}
```

```
        alt="Logo"
      />
    </TouchableOpacity>
    <Text style={styles.title}>Sign-in</Text>
    <Text style={styles.subtitle}>
      Discover new ways to get to your classrooms
      and more...
    </Text>
  </View>

  <View style={styles.form}>
    <View style={styles.input}>
      <Text style={styles.inputLabel}>Email
      Address</Text>
      <TextInput
        autoCapitalize="none"
        autoCorrect={false}
        style={styles.inputControl}
        placeholder="john@example.com"
        value={email}
        onChangeText={setEmail}
      />
      {errors.email ? (
        <Text
          style={styles.errorText}>{errors.email}</Text>
      ) : null}
    </View>

    <View style={styles.input}>
      <Text
        style={styles.inputLabel}>Password</Text>
      <View style={styles.passwordContainer}>
        <TextInput
          style={[styles.inputControl,
          styles.passwordInput]}
          placeholder="*****"
          value={password}
        </View>
      </View>
    </View>
  </View>
</Image>
```

```
        onChangeText={setPassword}
      />

    </View>
    {errors.password ? (
      <Text
        style={styles.errorText}>{errors.password}</Text>
    ) : null}
  </View>

  <View style={styles.formAction}>
    <TouchableOpacity onPress={handleSubmit}>
      <View style={styles.btn}>
        <Text style={styles.btnText}>Sign
          in</Text>
        </View>
      </TouchableOpacity>
    </View>
  </View>

  <TouchableOpacity
    style={{ marginTop: 'auto' }}
    onPress={() => {
      }
    }>
    <View>
      <Text style={styles.formFooter}>
        Don't have an account or forgotten
        details?{' '}
      <Text style={{ textDecorationLine:
        'underline' }}>
        Sign-up
      </Text>
    </Text>
  </View>

```

```
        </TouchableOpacity>
    </View>
</ScrollView>
</KeyboardAvoidingView>
</SafeAreaView>
</SafeAreaProvider>
);
}

const styles = StyleSheet.create({
  container: {
    paddingHorizontal: 24,
    flex: 1,
    backgroundColor: "#38b6ff",
  },
  headerImg: {
    marginBottom: 0,
    alignSelf: 'center',
  },
  title: {
    color: '#333333',
    fontSize: 38,
    textAlign: 'center',
    fontFamily: 'serif',
    fontWeight: 'bold',
  },
  subtitle: {
    marginVertical: 10,
    fontSize: 17,
    fontWeight: '500',
    color: '#000033',
    textAlign: 'center'
  },
  input: {
    fontSize: 12
  },
  inputLabel: {
    marginBottom: 15,
```

```
    fontSize: 20,
    fontWeight: '600',
    color: '#222'
},
inputControl: {
    height: 44,
    marginBottom: 15,
    backgroundColor: '#ffffff',
    paddingVertical: 10,
    paddingHorizontal: 16,
    borderRadius: 12,
    fontSize: 15,
    fontWeight: '500',
    color: '#222'
},
passwordContainer: {
    flexDirection: 'row',
    alignItems: 'center',
},
passwordInput: {
    flex: 1,
},
form: {
    marginBottom: 24,
    flex: 1,
},
formAction: {
    marginVertical: 24,
},
btn: {
    backgroundColor: '#075eec',
    borderRadius: 8,
    borderWidth: 1,
    borderColor: '#075eec',
    flexDirection: 'row',
    justifyContent: 'center',
    alignContent: 'center',
    paddingVertical: 10,
    paddingHorizontal: 20,
```

```

},
btnText: {
  fontSize: 18,
  fontWeight: '600',
  color: '#fff',
},
formFooter: {
  fontSize: 17,
  fontWeight: '600',
  textAlign: 'center',
  letterSpacing: 0.3,
  paddingHorizontal: 27,
},
errorText: {
  color: 'red',
  marginBottom: 10,
  fontSize: 17,
  backgroundColor: 'white',
  borderRadius: 12,
  padding: 8,
  fontWeight: '400',
  alignSelf: 'flex-start'
},
}) ;

```

Description:	<p>The main purpose of this version of the application is to apply validation rules to the inputs for password and email. This is in order to meet SC.06:</p> <p>The Login/Sign-up page should be the first screen for the users of the application, but the user should have the ability to use the app as a guest and should have appropriate validation.</p> <p>I swapped the setForm values to:</p> <pre>const [email, setEmail] = useState(''); const [password, setPassword] = useState(''); const [errors, setErrors] = useState({});</pre> <p>This will allow me to refer to each variable individually and also update them. I added another variable for errors which will include</p>
--------------	--

any errors to inform the user of.

```
const validateForm = () => {
  let errors = {};

  if (!email) errors.email = "Email is required.";
  if (!password) errors.password = "Password is required.";

  setErrors(errors);
  return Object.keys(errors).length === 0;
};
```

This code forms the main backbone of my validation as it checks if any of the inputs have no text in them and then will output these according error messages e.g “Email is required”.

```
const handleSubmit = () => {
  if (validateForm()) {
    console.log("Submitted", email, password);
    setEmail('');
    setPassword('');
    setErrors({});
  }
};
```

This is to reset the state of errors and email/password when the Sign-In button is pressed and an error is presented.

This function resets the inputs, password email variables and sets errors to empty.

```
showsVerticalScrollIndicator={false}
```

This is to remove the scroll indicator to provide a more sleeker look.

```
{errors.email ? (
  <Text
    style={styles.errorText}>{errors.email}</Text>
) : null}
```

This is a ternary operator and if the error function is true then it will output the message with style `errorText` which is found below:

```
errorText: {
  color: 'red',
  marginBottom: 10,
  fontSize: 17,
  padding: 8,
  fontWeight: '400',
  alignSelf: 'flex-start'
},
```

This same process is applied to the password input:

```
{errors.password ? (
  <Text
    style={styles.errorText}>{errors.password}</Text>
) : null}
```

```
<TouchableOpacity onPress={handleSubmit}>
  <View style={styles.btn}>
    <Text style={styles.btnExit}>Sign in</Text>
  </View>
</TouchableOpacity>
```

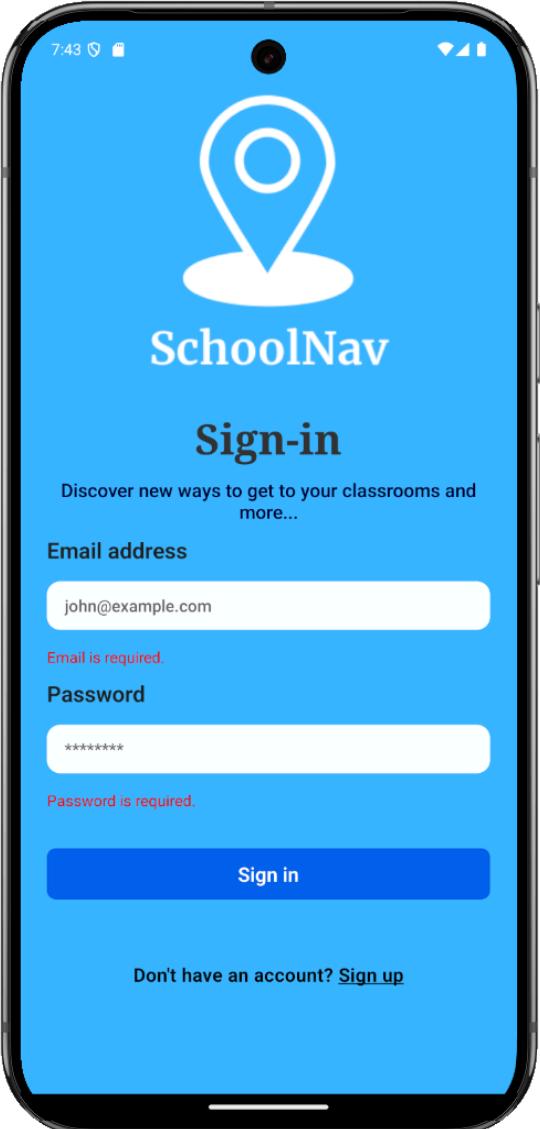
This is for when users press `SignIn` it will handle the Submit and any according error messages. The function is listed above.

Test Table 7 – Validation with the text inputs

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.15 – VALIDATION	If the Sign-In page is pressed without any input for either the email or password fields then the screen should	Sign-In without any input for both fields and visual check for any error messages. Input for email and leave out password and	When attempting to Sign-In without any text input for both fields you should find two error messages accordingly.	The error messages are presented accordingly however they are not clear/visible enough to highlight the issue.	No HyperLink to Solution

	<p>output appropriate error messages to highlight the issue.</p>	<p>then repeat and vice versa to test each case.</p>	<p>If you Sign-In with one input then you should get one error message and if you Sign-In with both inputs, no error messages should be outputted.</p>		
--	--	--	--	--	--

Evidence Table 7

Test ID	Screenshot/Video	Evaluation
1.15		<p>The error messages are being outputted correctly but are not clear/visible enough.</p> <p>This is the test case for when no input is given and the Sign In button is being pressed.</p> <p>Video of all other test cases can be found below.</p>

Fixing Unclear Error Text Messages: Test ID 1.15

Relevant Source Code:	<pre data-bbox="430 1814 1485 2032"> From: errorText: { color: 'red', marginBottom: 10, fontSize: 17, } </pre>
-----------------------	--

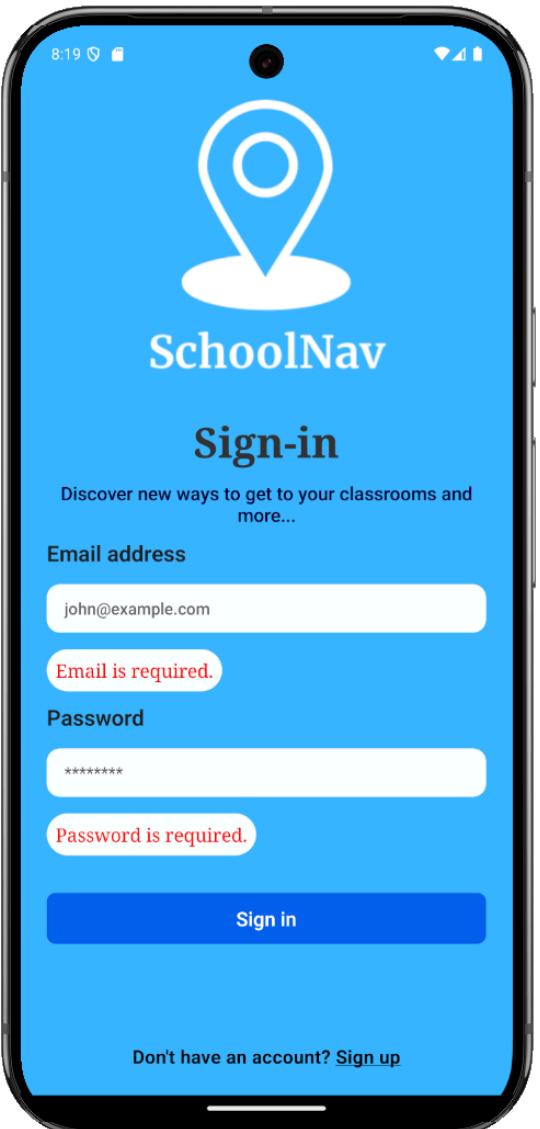
	<pre> padding: 8, alignSelf: 'flex-start' }, To: errorText: { color: 'red', marginBottom: 10, fontSize: 17, backgroundColor: 'white', borderRadius: 12, padding: 8, fontWeight: '400', alignSelf: 'flex-start' }, </pre>
Description:	This allows the text to become more readable on a white background and I softened the box by providing a radius to give off a more friendly and comfortable look. I also added a higher font-weight to give off the message more clearly. Padding was added also to spread out the text from the box it's in.

Test Table 8 – Validation with the text inputs

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.15 – VALIDATION	If the Sign-In page is pressed without any input for either the email or password fields then the screen should output appropriate	Sign-In without any input for both fields and visual check for any error messages. Input for email and leave out password and then repeat and vice versa	When attempting to Sign-In without any text input for both fields you should find two error messages accordingly. If you Sign-In with one input then you should get one error	The error messages are presented accordingly and clear to the user.	Yes – Fixed from previous issue, minor fix An appropriate error message is displayed.

	error messages to highlight the issue.	to test each case.	message and if you Sign-In with both inputs, no error messages should be outputted.		
--	--	--------------------	---	--	--

Evidence Table 8

Test ID	Screenshot/Video	Evaluation
1.15	<p>https://youtu.be/EPCXJ2fJqRU</p>  <p>The screenshot shows the SchoolNav app's sign-in interface. At the top, there is a large white location pin icon with the text "SchoolNav" below it. Below that is the heading "Sign-in". A subtext reads "Discover new ways to get to your classrooms and more...". There are two input fields: "Email address" containing "john@example.com" and "Password" containing "*****". Both fields have red error messages: "Email is required." above the email field and "Password is required." above the password field. At the bottom is a blue "Sign in" button. At the very bottom of the screen, there is a link "Don't have an account? Sign up".</p>	<p>The video presents the test against all possible test cases for the email input and password input and the corresponding error messages.</p> <p>This screenshot shows how the more readable error message is presented.</p> <p>The second image is the values that were attained through the testing, showcasing that form works and does correctly take in data values.</p>

```
(NOBRIDGE) LOG App executed  
(NOBRIDGE) LOG Submitted AAS SDE  
(NOBRIDGE) LOG App executed  
  
(NOBRIDGE) LOG App executed  
(NOBRIDGE) LOG Submitted example.com djhdsjsjd  
(NOBRIDGE) LOG App executed
```

Adding in toggle to switch between Visible and Encrypted passwords + Guest Option

Relevant Source Code:

```
const [isPasswordVisible, setIsPasswordVisible] = useState(false);  
// State to toggle password visibility  
  
<TouchableOpacity  
    onPress={() =>  
        setIsPasswordVisible(!isPasswordVisible)}  
    style={styles.eyeButton}  
>  
    <Text style={styles.eyeText}>  
        {isPasswordVisible ? "👁️" : "🙈"}  
    </Text>  
    </TouchableOpacity>  
</View>  
  
<TouchableOpacity  
    style={{ paddingTop: 25, alignSelf: 'center' }}  
    onPress={() }  
>  
    <View style={styles.btn}>  
        <Text style={styles.btnText}>Guest?</Text>  
    </View>  
</TouchableOpacity>  
  
eyeButton: {  
    flex: 0,  
    marginLeft: 10,  
    marginBottom: 15,  
},
```

```

eyeText: {
  fontSize: 18,
},
btn: {
  backgroundColor: '#075eec',
  borderRadius: 8,
  borderWidth: 1,
  borderColor: '#075eec',
  flexDirection: 'row',
  justifyContent: 'center',
  alignItems: 'center',
  paddingVertical: 10,
  paddingHorizontal: 20,
},
btnText: {
  fontSize: 18,
  fontWeight: '600',
  color: '#fff',
},

```

Description:	<p>This is the relevant code for implementing a solution for the toggle to swap between visible passwords and encrypted password displays.</p> <pre>const [isPasswordVisible, setIsPasswordVisible] = useState(false);</pre> <p>This is the const variable set by default to false where the password is default encrypted.</p> <p>When the toggle is pressed it will mean the function within the onPress object is runned and it will not the original value e.g if value for isPasswordVisible is False it will negate it and set it to True.</p> <p>I chose friendly emojis to represent each of the options for the toggle, this contributes to the friendly environment I am looking to create.</p> <p>I have inserted this on the password input line to associate the toggle with the password.</p> <p>The Guest button was added here with the title/label “Guest?” to</p>

	<p>offer a conversational friendly atmosphere. This button should navigate users directly to the Main Menu if they don't wish to Sign-In/Sign-Up.</p> <p>Below is the CSS Styles for the new elements.</p>
--	--

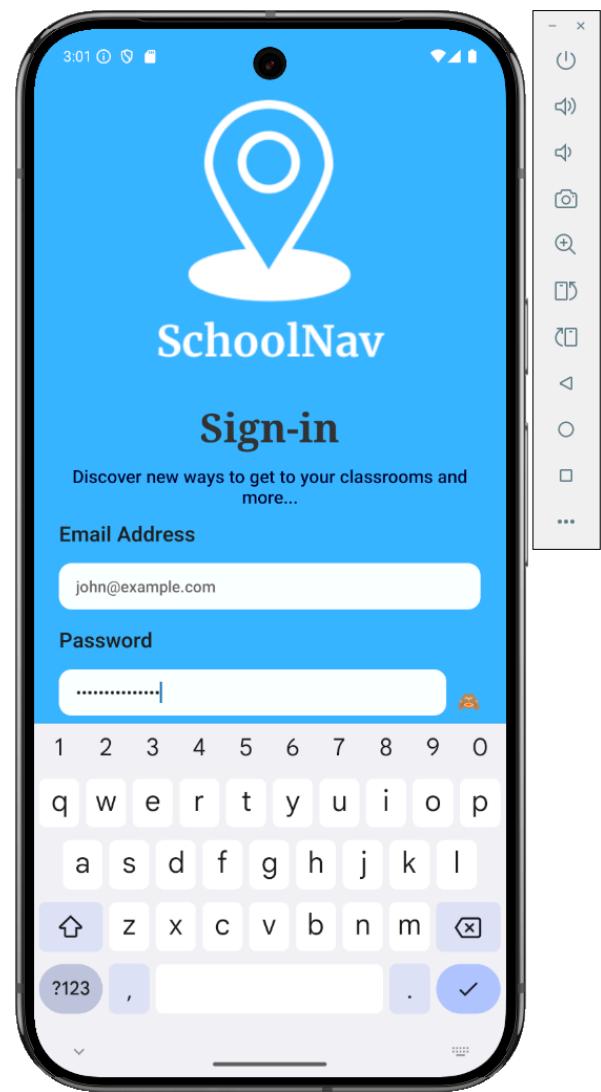
Test Table 9 - Adding in toggle to switch between Visible and Encrypted passwords

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.10 – Revisit due to changing related element	Text should be encrypted when writing in the password field.	Type into the password field and visual check for encryption symbols to prevent others from seeing the password.	The text typed into the field should be replaced with encryption symbols to prevent others from seeing the password.	By default the text remains encrypted, adding the functions doesn't affect the default state of viewing the password.	Yes, the text is encrypted by default for the password field.
1.16	Users should have the option to Sign-In as a guest since not all users would want to Sign-In/Sign-Up. This should be prevalent and clear to the user.	Visual check for Guest option.	There should be an obvious place for the Guest option located in a clear, distinct area on the screen.	The Guest option is labelled with "Guest?" it is prevalent and clear to the user. I am satisfied with this output.	Yes, the Guest option is present.
1.17	Users should be able to toggle between visible	Visual check for toggle and run test cases by	Text should be default encrypted and the toggle when	The toggle is visible and when pressed does lead to a change in the	No HyperL

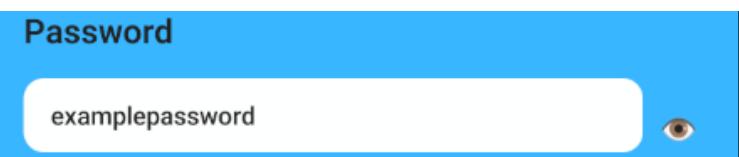
	password fields and encrypted fields.	inputting text to see if it is encrypted and visible accordingly to toggles.	pressed should allow the user to view it and vice versa.	text input accordingly (viewing and encrypting). However the toggle is not in the center and gives off an unprofessional and disorganised look to my application.	Link to Solution
--	---------------------------------------	--	--	---	----------------------------------

Evidence Table 9

Test ID	Screenshot/Video
1.10 (Revisit) + 1.16	



<https://youtu.be/WhBEff682Ac>

Evaluation:	<p>The Guest button is clear and prevalent and is located below the Sign-In button conveniently.</p> <p>The toggle works as expected and is functioning accurately but is located on the screen in an awkward position here is a closer screenshot.</p>  <p>This version doesn't entirely meet 1.16 since users cannot exactly use</p>
-------------	--

	<p>the toggle comfortably and this can hinder and offset a bad impression for the rest of the application.</p> <p>This video shows the toggle function working in action.</p>
--	---

Fixing Placement of Toggle for Viewing Password

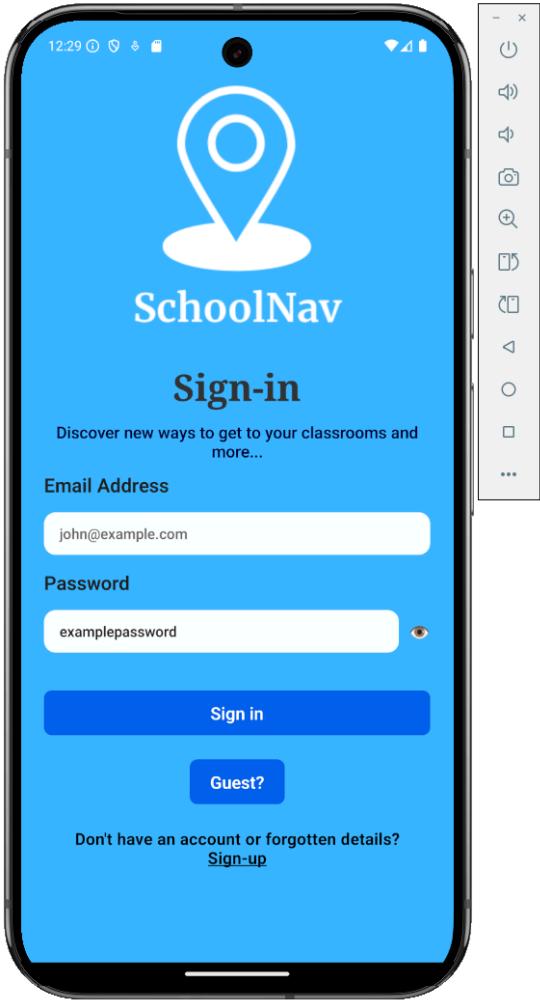
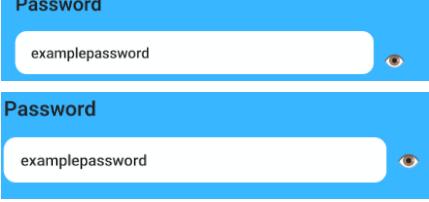
Relevant Source Code:	<p>From:</p> <pre>eyeButton: { marginLeft: 10, }, eyeText: { fontSize: 18, },</pre> <p>To:</p> <pre>eyeButton: { marginLeft: 10, marginBottom: 15, }, eyeText: { fontSize: 18, },</pre>
Description:	I added marginBottom to displace the toggle by 15 pixels upwards. This centered the toggle button in the vertical direction and worked as a solution to the disorganised look it was formulating.

Test Table 10 - Initialising Log-In Screen

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.17 -FIX	Users should be able to toggle between visible	Visual check for toggle and run test cases by inputting text to see if it	Text should be default encrypted and the toggle when	The toggle is now functioning and adding to the organised	Yes - fixed from previous error

	password fields and encrypted fields	is encrypted and visible accordingly to toggles.	pressed should allow the user to view it and vice versa.	look of the application being in the center of the password input box.	Users can now toggle between the encrypted and visible fields.
--	--------------------------------------	--	--	--	--

Evidence Table 10

Test ID	Screenshot/Video	Evaluation
1.17	 <p>The screenshot shows the SchoolNav sign-in interface. At the top is a large blue circular logo with a white location pin icon. Below it is the text "SchoolNav" and "Sign-in". A subtext says "Discover new ways to get to your classrooms and more...". There are two input fields: "Email Address" containing "john@example.com" and "Password" containing "examplepassword". To the right of the password field is an "eye" icon with a small red dot, indicating the password is currently encrypted. Below the fields are two blue buttons: "Sign in" and "Guest?". At the bottom, there's a link "Don't have an account or forgotten details? <u>Sign-up</u>".</p>	<p>These screenshots show the centered toggle for the password viewer.</p> <p>I hadn't tested any functional test cases from the password viewer since we had done so in the previous Test Table and I haven't updated any code relevant to the functioning aspect of it.</p> <p>Below is a more close up screen-shot.</p> <p>This is the before and after respectively:</p> 

Version 3 - Initialising Register Screen

This iteration focuses on setting up the backbone for the Register Screen before implementing any functionality features.

Initialising Register Screen

Relevant Source Code:

```
import React, { useState } from 'react';
import { SafeAreaProvider, SafeAreaView } from
'react-native-safe-area-context';
import {
  StyleSheet,
  Text,
  View,
  TextInput,
  TouchableOpacity,
  Platform,
  KeyboardAvoidingView,
  ScrollView,
} from 'react-native';

export default function RegisterScreen() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');

  return (
    <SafeAreaProvider>
      <SafeAreaView style={styles.container}>
        <KeyboardAvoidingView
          behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
          style={{ flex: 1 }}
          keyboardVerticalOffset={20}
        >
```

```
<ScrollView
    contentContainerStyle={{ flexGrow: 1 }}
    keyboardShouldPersistTaps="handled"
    showsVerticalScrollIndicator={false}
>
<View>
    <Text style={styles.title}>Register your
account</Text>
    <Text style={styles.subtitle}>
        Create an account to access SchoolNav features.
    </Text>
</View>

<View style={styles.form}>
    {/* Name Input */}
    <View style={styles.input}>
        <Text style={styles.inputLabel}>Name</Text>
        <TextInput
            autoCapitalize="True"
            autoCorrect={false}
            style={styles.inputControl}
            placeholder="John Smith"
            value={name}
            onChangeText={setName}
        />
    </View>
    {/* Email Input */}
    <View style={styles.input}>
        <Text style={styles.inputLabel}>Email
Address</Text>
        <TextInput
            autoCapitalize="none"
            autoCorrect={false}
            style={styles.inputControl}
            placeholder="john@example.com"
            value={email}
            onChangeText={setEmail}
        />
    </View>

```

```
/* Password Input */
<View style={styles.input}>
  <Text style={styles.inputLabel}>Password</Text>
  <TextInput
    secureTextEntry
    style={styles.inputControl}
    placeholder="*****"
    value={password}
    onChangeText={setPassword}
  />
</View>

/* Confirm Password Input */
<View style={styles.input}>
  <Text style={styles.inputLabel}>Confirm
  Password</Text>
  <TextInput
    secureTextEntry
    style={styles.inputControl}
    placeholder="*****"
    value={confirmPassword}
    onChangeText={setConfirmPassword}
  />
</View>

/* Submit Button */
<View style={styles.formAction}>
  <TouchableOpacity onPress={() => {}}>
    <View style={styles.btn}>
      <Text style={styles.btnText}>Register</Text>
    </View>
  </TouchableOpacity>
</View>

/* Navigation to Login */
<TouchableOpacity
  onPress={() => {}}
> <View>
```

```
        <Text style={styles.formFooter}>
          Already have an account? <Text style={{textDecorationLine: 'underline' }}>Sign in</Text>
        </Text>
      </View>

    </TouchableOpacity>
  *)
</View>
<ScrollView>
</KeyboardAvoidingView>
<SafeAreaView>
<SafeAreaProvider>
);
}

const styles = StyleSheet.create({
  container: {
    paddingHorizontal: 24,
    flex: 1,
    backgroundColor: '#38b6ff',
    paddingVertical: '70%',
  },
  title: {
    color: '#333333',
    fontSize: 38,
    textAlign: 'center',
    fontFamily: 'serif',
    fontWeight: 'bold',
    paddingBottom: 30,
  },
  subtitle: {
    marginVertical: 10,
    fontSize: 17,
    fontWeight: '500',
    color: '#000033',
    textAlign: 'center',
  },
  input: {
```

```
    fontSize: 12,
},
inputLabel: {
  marginBottom: 15,
  fontSize: 20,
  fontWeight: '600',
  color: '#222',
},
inputControl: {
  height: 44,
  marginBottom: 15,
  backgroundColor: '#ffffff',
  paddingVertical: 10,
  paddingHorizontal: 16,
  borderRadius: 12,
  fontSize: 15,
  fontWeight: '500',
  color: '#222',
},
form: {
  marginBottom: 24,
  flex: 1,
},
formAction: {
  marginVertical: 24,
},
btn: {
  backgroundColor: '#075eec',
  borderRadius: 8,
  borderWidth: 1,
  borderColor: '#075eec',
  flexDirection: 'row',
  justifyContent: 'center',
  alignContent: 'center',
  paddingVertical: 10,
  paddingHorizontal: 20,
},
btnText: {
  fontSize: 18,
```

```

        fontWeight: '600',
        color: '#fff',
    },
    formFooter: {
        fontSize: 17,
        fontWeight: '600',
        textAlign: 'center',
        letterSpacing: 0.3,
        paddingHorizontal: 27,
    },
    errorText: {
        color: 'red',
        marginBottom: 10,
        fontSize: 17,
        backgroundColor: 'white',
        borderRadius: 12,
        padding: 8,
        fontWeight: '400',
        alignSelf: 'flex-start',
    },
});

```

Description:	<p>This is the initial code for the Register Screen. This code has many properties refactored from the Login Screen. The use of refactoring and repurposing older/previous code has reduced my project timeline significantly.</p> <p>I added KeyboardAvoidingView to prevent the Keyboard to obstruct any view of the screen when in use and also the ScrollView to compliment in conjunction with it. Other specific descriptions about this can be found in the Log-In Screen initialisation Version 1.</p> <p>The main title of the screen is implemented here to draw the users attention to the purpose of the screen, complemented with the subtitle which is “Create an account to access SchoolNav features.” This is preceded by the title: “Register your account”.</p>

	<p>Implementing four text inputs. For Name, Email Address, Password and Confirm Password. Where I effectively meet SC.17:</p> <p>Register screen should function and take in username and password and have appropriate validation.</p> <p>All these inputs have according placeholders e.g name: "John Smith" this is to inform the user of what kind of input should be inputted into this field.</p> <p>For the name specifically I have set autoCapitalise to be True since it is a name and should be capitalised and false for all other fields. I also set autoCorrect false to ensure it doesn't correct any names people input.</p> <p>Mainly I have combatted my previous error in Version 1 when setting variable names where I didn't write the correct names, but here I have associated each name input to value and stored it using the useState function.</p> <p>Sometimes people may remember they already have an account so to Sign-In I have created a button for that but no navigation yet. It is under the title: "Already have an account? <u>Sign In</u>"</p> <p>Add name section since I wanted to add customisation and personalisation within the application.</p>
--	--

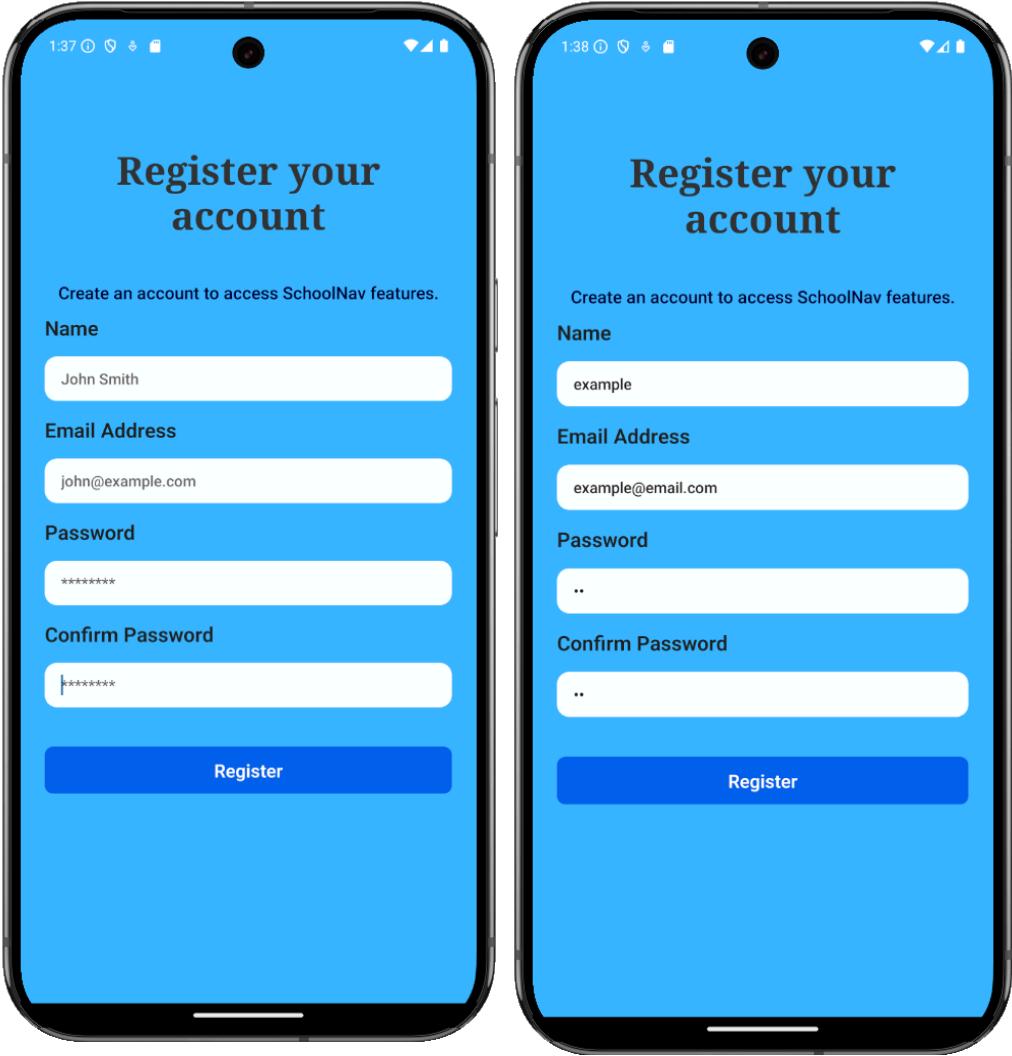
Test Table 11 – Initialising Register Screen

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.18	Users should be able to identify Register Screen/Sign-Up Page through a clear Title.	Visual check for according title to signify Register Screen.	Titles should be prevalent and large to stand out and inform users of the page they are on.	Title is the first element on the screen and is large and prevalent hence clearly defining the page and its purpose.	Yes, Register Screen's purpose is indicative.

1.19	Field for name should be implemented to provide customisation options.	Visual check for Name field and test through writing Text.	The text should save and not produce any error messages and the name field should be present.	The field for name is clear with a placeholder: "John Smith" and the text inputted saves and no error message is produced.	Yes, the name field is there.
1.20	Fields for Email Address and Password should be present and their purpose should be clear using relevant titles.	Visual check for the email address field and the password field with according titles and placeholders. Attempt inputting text.	The fields should be present and clearly visible and their purpose should be identifiable. Text should save and not produce any error messages.	The field for Email Address and Password is clear with a placeholders: "john@example.com" and "*****" respectively and the text inputted saves and no error message is produced.	Yes, email and password fields are present.
1.21	Field for Confirm Password should function and be present	Visual check for the Confirm Password Field and input text into this field.	The fields should be present and clearly visible and their purpose should be identifiable. Text should save and not produce any error messages.	The field for Confirm Password is clear with a placeholder: "*****" and the text inputted saves and no error message is produced.	Yes, confirm password works as desired.
1.22	The register button should be present and pressable.	Visual check for the register button and attempt pressing it.	Register Button should be clear and prevalent and pressable.	The register button matches the theme of previous buttons and is clear and	Yes, the register button is interactable and is present.

				prevalent to users.	
1.23	Option to return back to Sign-In/Guest page	Visual check for options and the button should be pressable.	This option should be visible to users and pressable.	The option to return back to Sign-In is not visible even though it was expected to present the text: "Already have an account? <u>Sign in</u> "	<p>No</p> <p>HyperLink to Solution</p>

Evidence Table 11

Test ID	Screenshot/Video
1.18-1.23	
Evaluation:	<p>1.18 – 1.22 is fulfilled here.</p> <p>All the fields are clear and defined with the text inputs not producing any error messages as before in Version 1.</p> <p>However, the sign-in option, to return to the Log-In page is not present, hence not fulfilling 1.23.</p>

Fixing Logo cut off issue: Test ID 1.5

Relevant Source Code:

From:

```
/* Navigation to Login
  <TouchableOpacity
    onPress={() => {}}
  > <View>
    <Text style={styles.formFooter}>
      Already have an account? <Text style={{textDecorationLine: 'underline'}}>Sign in</Text>
    </Text>
  </View>

  </TouchableOpacity>
*/}
```

To:

```
/* Navigation to Login */
  <TouchableOpacity
    onPress={() => {}}
  > <View>
    <Text style={styles.formFooter}>
      Already have an account? <Text style={{textDecorationLine: 'underline'}}>Sign in</Text>
    </Text>
  </View>

  </TouchableOpacity>
```

Description:

The comment to label each of my functions/elements has interfered with my code and closed off the element presenting the Sign-In Option; this caused that part of the code to not appear and be considered as a comment instead.

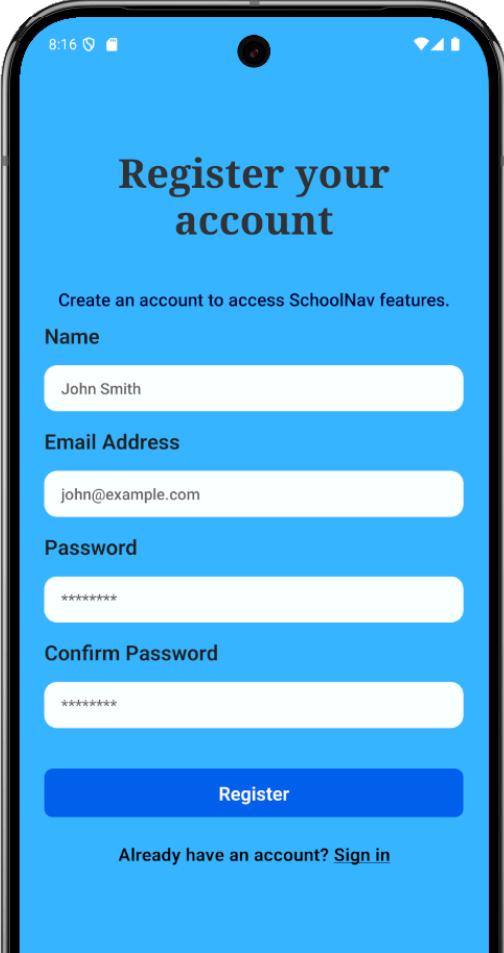
By removing the closing tag of the comment and restoring its intended position the final element is now visible on screen.

This logical error was fixed by manipulating the closing tag for the comment.

Test Table 12 – Initialising Register Screen

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.23-FIX	Option to return back to Sign-In/Guest page	Visual check for options and the button should be pressable.	This option should be visible to users and pressable.	The option is now visible with text: “Already have an account? <u>Sign in</u> ” and is pressable.	Yes – fixed from previous error The option to return to the Sign-In page is present.

Evidence Table 12

Test ID	Screenshot/Video	Evaluation
1.23		<p>Here the option to Sign-In/return to the Login page is found at the bottom of the screen.</p> <p>It is now visible and clear to users with the title:</p> <p>“Already have an account? <u>Sign in</u>”</p>

Version 4 - Validation of Register Screen and Functioning Login System

Validation of Register Screen

Relevant Source Code:

```
export default function RegisterScreen() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [errors, setErrors] = useState({});

  const validateForm = () => {
    let errors = {};
    if (!name) errors.name = 'Name is required.';
    if (!email) errors.email = 'Email is required.';
    if (!password) errors.password = 'Password is required.';
    if (password !== confirmPassword) errors.confirmPassword =
      'Passwords do not match.';
    setErrors(errors);
    return Object.keys(errors).length === 0;
  };

  {errors.name && <Text
    style={styles.errorText}>{errors.name}</Text>}

  {errors.email && <Text
    style={styles.errorText}>{errors.email}</Text>}

  {errors.password && <Text
    style={styles.errorText}>{errors.password}</Text>

  {errors.confirmPassword && (
    <Text
      style={styles.errorText}>{errors.confirmPassword}</Text>
    )}
```

```

/* Submit Button */
        <View style={styles.formAction}>
            <TouchableOpacity onPress={validateForm}>
                <View style={styles.btn}>
                    <Text style={styles.btnText}>Register</Text>
                </View>
            </TouchableOpacity>
        </View>

errorText: {
    color: 'red',
    marginBottom: 10,
    fontSize: 17,
    backgroundColor: 'white',
    borderRadius: 12,
    padding: 8,
    fontWeight: '400',
    alignSelf: 'flex-start',
},

```

	<p>Description:</p> <p>Implemented another variable for errors and utilised the useState function to do so, setting default as none/null.</p> <p>The function validateForm resets any errors initially by setting it to null/empty list and then running conditionals to test if an according error needs to be submitted e.g !name means if name field is empty then errors.name = “Name is required.” This is the error message that would be produced if there is no input in the name field.</p> <p>Equally a set of conditionals and according statements are listed here.</p> <p>For confirmPassword field the logic is different since we want to check if the passwords match hence we state if the password and confirm password does not equal then output an error message saying specifically “Passwords do not match.”</p> <p>This is followed by a handleSubmit function which utilises the above validateForm function. It logs the values of the email and password into the console and once submit is entered sets Name, Email, Password and ConfirmPassword to null and this only happens if there are no error messages and the validateForm function returns an object of empty errors.</p>
--	--

Within all four text input fields before the closing tag for the `<view>` elements

```
{errors.name && <Text  
style={styles.errorText}>{errors.name}</Text>}
```

I implemented this logic to output the error message in the correct positions relatively.

```
{errors.email && <Text  
style={styles.errorText}>{errors.email}</Text>}
```

```
{errors.password && <Text  
style={styles.errorText}>{errors.password}</Text>}
```

```
{errors.confirmPassword && (  
    <Text  
    style={styles.errorText}>{errors.confirmPassword}</Text>  
)}
```

The styles associated with the error messages were refactored from previous code and reduced the potential error of not being able to see the error message we had before.

For the register button:

The button `onPress` runs `handleSubmit` and this will then check the inputs and output error messages etc.

The function is embedded here:

```
<TouchableOpacity onPress={handleSubmit}>
```

Test Table 13 – Validation of Register Screen

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful ?
1.24 - VALIDATION	If the Register Button is pressed without any	Register without any input for all three fields and visual	When attempting to Register without any text input for all fields you	According to each error and missing input there	Yes, there is appropriate validation

	input for either the name, email or password fields then the screen should output appropriate error messages to highlight the issue.	check for any error messages. Input for name and leave out password and then repeat and vice versa to test each case.	should find three error messages accordingly. If you Register with one input then you should get two error messages and if you Sign-In with three inputs, no error messages should be outputted, these error messages are in relation to there needing text in the fields.	is an accurate and relevant error message stating "Name is required." for example. The error message is clear and viewable.	for the inputs.
1.25 - VALIDATION	If the Register Button was pressed without the Password field and the Confirm Password field not matching, appropriate error messages should be outputted.	Register with the passwords matching and register with them not matching visual check for clear and prevalent error messages.	According to both scenarios where they match or don't match, accurate and relevant error messages should not show and show.	With each case the validation works effectively by producing relevant error messages in the right and according places.	Yes, there is appropriate validation for the inputs.
1.26	The Register Button should clear all values of the fields to formulate a clean environment to compliment any error	Press the Register button with inputs and without inputs and visual check if the inputs are no longer in the fields.	With each press of the Register button the fields on the register screen should empty out and be set to empty/null.	Pressing the Register button has no effect on the inputs on screen; the fields are not updated.	No HyperLink to Solution

	messages.				
--	-----------	--	--	--	--

Evidence Table 13

Test ID	Screenshot/Video	Evaluation
1.24 - 1.26	https://youtube.com/shorts/X0_U6kUAe7o	The video tests all cases for the inputs of the Register Screen and its relevant validation error messages however when pressing Register the inputs are not clear when no error messages are present hence forming a disorganised and unclean look.

Fixing Register Screen Input Issue: Test ID 1.26

Relevant Source Code:

```
const handleSubmit = () => {
  if (validateForm()) {
    console.log('Registered:', email, password);
    setName('');
    setEmail('');
    setPassword('');
    setConfirmPassword('');
    setErrors({});
  }
};

/* Submit Button */
<View style={styles.formAction}>
  <TouchableOpacity onPress={handleSubmit}>
    <View style={styles.btn}>
      <Text style={styles.btnClose}>Register</Text>
    </View>
  </TouchableOpacity>
</View>
```

Description:	<p>Creating a new function called handleSubmit to reset the inputs and reset the errors present to maintain and preserve the clean look I want for my application.</p> <p>This function is called the function validateForm so it does two things simultaneously and only does the resetting error messages if the validateForm outputs no error messages.</p> <p>Then I swapped the function being called onPress for the Register button from validateForm to handleSubmit.</p>
--------------	---

Test Table 14 - Fixing Register Screen Input Issue: Test ID 1.26

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.26-FIX	The Register Button should clear all values of the fields to formulate a clean environment to compliment any error messages.	Press the Register button with inputs and without inputs and visual check if the inputs are no longer in the fields.	With each press of the Register button the fields on the register screen should empty out and be set to empty/null.	When there are no errors and there inputs in all fields and the password and confirm Password inputs match, the inputs all reset as expected.	Yes - fixed from previous error

Evidence Table 14

Test ID	Screenshot/Video	Evaluation
1.26	https://youtu.be/LtAVQ9FBMlw	This video showcases the resetting inputs in action as when there are no errors and there inputs in all fields and the password and confirm Password inputs match, the inputs all reset as expected.

Implementing Functioning Login System

Relevant Source Code:

```
// Import necessary React and React Native components
import React, { useState } from 'react';
import { StatusBar } from 'expo-status-bar';
import { SafeAreaProvider, SafeAreaView } from
'react-native-safe-area-context';
import {
  StyleSheet,
  Text,
  View,
  TextInput,
  Image,
  TouchableOpacity,
  Platform,
  KeyboardAvoidingView,
  ScrollView
} from 'react-native';
import AsyncStorage from
'@react-native-async-storage/async-storage';

export default function LoginScreen({route, navigation }) {
  const { name } = route.params || {}; // Retrieve name from
  Register Screen
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [errors, setErrors] = useState({});
  const [isPasswordVisible, setIsPasswordVisible] =
  useState(false); // State to toggle password visibility

  const handleLogin = async () => {
    if (validateForm()) {
      try {
        const storedUser = await
        AsyncStorage.getItem('userDetails');
        const parsedUser = JSON.parse(storedUser);

        if (parsedUser && parsedUser.email === email &&
        parsedUser.password === password) {
```

```

        console.log('Logged in:', email);
        navigation.navigate('Main Menu', { name }); // Navigate
        to the main menu after successful login
    } else {
        setErrors({ ...errors, email: 'Invalid email or
password.' });
    }
} catch (error) {
    console.error('Error fetching user data:', error);
}
};


```

	<p>Description:</p> <p>Here, the primary goal was to implement the Login Screen functionality, enabling integration between the Register Screen and the Main Menu. This ensures users can log in after registration and access a personalized experience, including the display of their name. The development process and decisions made are detailed below, with annotations and justifications provided.</p> <p>To achieve this, I utilized AsyncStorage to store and retrieve user details securely. This library is critical for persisting data between app sessions.</p> <pre> import AsyncStorage from '@react-native-async-storage/async-storage'; </pre> <p>The choice of AsyncStorage was due to its lightweight nature and suitability for storing small amounts of data, such as user credentials.</p> <p>To ensure the name from the Register Screen was accessible on the Login Screen, I used route.params to retrieve the name and navigation to enable seamless transitions between screens.</p> <pre> export default function LoginScreen({route, navigation }) { const { name } = route.params {}; // Retrieve name from Register Screen </pre> <p>This approach ensures a modular and reusable design, allowing data to flow between screens in a structured manner.</p>
--	---

The **handleLogin** function integrates form validation, user authentication using **AsyncStorage**, and navigation. Below is the detailed breakdown:

Form validation is handled by the **validateForm** function (not shown here but assumed to be defined). This ensures users cannot proceed without entering valid credentials.

Using **AsyncStorage**, the stored user details are retrieved and parsed. This allows comparison with the entered credentials.

```
const storedUser = await AsyncStorage.getItem('userDetails');
const parsedUser = JSON.parse(storedUser);
```

The entered email and password are compared with the stored details. If they match, the user is navigated to the Main Menu with their **name** passed as a parameter.

```
if (parsedUser && parsedUser.email === email &&
parsedUser.password === password) {
    console.log('Logged in:', email);
    navigation.navigate('Main Menu', { name }); // Navigate
to the main menu after successful login
} else {
    setErrors({ ...errors, email: 'Invalid email or
password.' });
}
```

If credentials do not match or there is an issue fetching user data, appropriate error messages are displayed.

```
setErrors({ ...errors, email: 'Invalid email or password.' });
```

This iterative process of development, validation, and review ensured a well-structured and user-friendly solution. Adjustments were made based on testing (e.g., improving password visibility and error handling), and decisions were justified to align with the success criteria.

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.30	User authentication with stored credentials	Enter a registered email and password for login. Simulate invalid login attempts with incorrect credentials.	Successful login for correct credentials redirects the user to Main Menu. Invalid credentials display an error message.	Correct credentials produce no error messages; incorrect credentials show "Invalid email or password" error.	Yes, the user is able to login effectively.
1.31	Store and retrieve user data securely	Simulate saving user credentials during registration and retrieving them during login using persistent storage.	Stored user data matches inputted registration data and can be retrieved during login for authentication.	Credentials are securely stored and retrieved successfully for valid users.	Yes, the user can then access that data as expected.
1.32-VALIDATION	Error handling for unregistered users	Enter an unregistered email during login.	The system displays an error message like "Invalid email or password."	Error message "Invalid email or password" displayed for unregistered email.	Yes, unregistered users experience an error message.
1.33-REVIST once the Main Menu has been initialized.	Navigation to Main Menu after successful login	Simulate a login process with valid email and password.	The app should navigate to the Main Menu page and display the user's name.	Navigation to the Main Menu occurs, and the user's name is displayed.	REVISIT

1.34-VALIDATION	Error message display for failed login	Enter incorrect email or password during login attempts.	An appropriate error message is displayed below the email or password field.	Error messages appear in the expected location and format.	Yes, failed logins result in error messages.
1.35	Clear feedback for successful login	Complete login with valid credentials.	A message (or navigation) confirms login success and displays user-specific content on the next screen.	Feedback is clear; user-specific content is shown after successful login. User is navigated to the Main Menu.	Yes, the user is taken to the Main Menu when successfully logged in.
1.36-VALIDATION	Incorrect email formats should throw error messages - Validation	<p>Any string without the character @ is invalid for the email address.</p> <p>Any string with more than one @ is invalid.</p> <p>Any string with no domain name after @ character is invalid.</p> <p>Any strings with any symbols other than these: (),;;<>@[\].</p>	Error message should be shown, incorrect format for email input.	The email format validation works as intended, displaying the error message "Incorrect email format" for strings without the @ character, with more than one @, missing a domain name after the @, or containing invalid symbols. For valid email formats, no error message appears, and the form submission proceeds as	Yes, there is appropriate validation for the inputs.

expected.

Version 5 – Initialising Main Menu

Navigation Between Different Screens

Relevant
Source
Code:

```
import { NavigationContainer } from "@react-navigation/native";
import { createNativeStackNavigator } from
"@react-navigation/native-stack";
import LoginScreen from "../SchoolNav/screens/LoginScreen";
import RegisterScreen from "../SchoolNav/screens/RegisterScreen";
import MainMenu from "../SchoolNav/screens/MainMenu";

const Stack = createNativeStackNavigator()
export default function App() {
  return(
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Login Page"
          component={LoginScreen}
          options={{ headerShown: false }} // Hide the header
        />
        <Stack.Screen
          name="Register Page"
          component={RegisterScreen}
          options={{ headerShown: false }} // Show the header
          initialParams={{
            ...
          }}
        />
        <Stack.Screen
          name="Main Menu"
          component={MainMenu}
          options={{ headerShown: true }} // Hide the header
          initialParams={{
            name: 'Guest',
          }}
        />
    
```

```

        </Stack.Navigator>
    </NavigationContainer>
)
}

export default function LoginScreen({ navigation }) {
    <TouchableOpacity
        style={{ paddingTop: 25, alignSelf: 'center' }}
        onPress={() => {
            navigation.navigate("Main Menu");
        }}
    >

    <TouchableOpacity
        style={{ marginTop: 'auto' }}
        onPress={() => {
            navigation.navigate('Register Page', {});
        }}
    >

    <TouchableOpacity
        onPress={() => {
            navigation.navigate('Login Page', {});
        }}
        style={{ marginTop: 'auto', alignSelf: 'center' }}
    >
        <View>
            <Text style={styles.formFooter}>
                Already have an account? <Text style={{
textDecorationLine: 'underline' }}>Sign in</Text>
            </Text>
        </View>
    </TouchableOpacity>
}

export default function MainMenu({route}) {
    const {name} = route.params;
}

```

Description:	<p>The navigation between screens was achieved using a Stack system.</p> <p>React Navigation:</p> <p>Using Stack, Drawer and Tab navigators.</p> <p>Stack Navigator provides a way for your app to transition between screens where each new screen is placed on top of a stack. This is in addition to the completion of SC.19: The different pages should be easy to navigate through.</p> <p>The name variable is dynamic and is set as default Guest and this happens when User chooses the log-in as a guest.</p> <p>This part of the project focuses on setting up and implementing navigation between the Login Page, Register Page, and Main Menu screens using React Navigation. The code primarily uses <code>createNativeStackNavigator</code> to organize these screens within a navigation stack and manages transitions between them seamlessly. The functionality is centered around allowing users to move between different pages of the app based on their actions.</p> <p>At the core of the implementation is the NavigationContainer component, which acts as the main wrapper for the entire navigation system. Inside it, a Stack Navigator is initialized with three key screens: the Login Page, Register Page, and Main Menu. Each screen is configured with options to either hide or show the header. For example, the Login Page and Register Page have their headers hidden using <code>options={{ headerShown: false }}</code> to maintain a cleaner, minimalistic user interface. In contrast, the Main Menu screen shows its header by default.</p> <p>The Login Page serves as the starting screen where users can choose to either navigate to the Main Menu or the Register Page. The navigation is enabled through buttons implemented using <code>TouchableOpacity</code>, which provides a responsive interaction when pressed. For instance, when the user taps on the Main Menu button, the <code>navigation.navigate("Main Menu")</code> function triggers a transition to the Main Menu screen. Similarly, there is another button that allows the user to navigate to the Register Page.</p> <p>The Register Page is designed to handle new user registration. While its exact functionality is not detailed in this segment of the code, the navigation system ensures that it is easily accessible through the Login Page. Users can also navigate back to the Login Page if needed,</p>
--------------	---

providing flexibility in the flow between these screens, which contributes to SC.16:

Aesthetically pleasing and functional design to the user which promotes the user experience.

The **Main Menu** screen introduces a customizable feature to enhance the user experience. It utilizes parameters passed during navigation to display personalized content. By default, the **initialParams** property sets a value of **name: "Guest"**, which is then displayed on the screen through a simple text component. For example, the **route.params** object is used to extract the name parameter, which is then displayed as **Hello {name}!**. This simple personalization provides a welcoming and user-friendly touch to the app. If no specific name is passed, it defaults to "Guest," ensuring the screen remains functional. By implementing a greet functionality, which is personalised in the process of meeting SC.16:

Aesthetically pleasing and functional design to the user which promotes the user experience.

Initialising Main Menu

Relevant Source Code:

```
import React from 'react';
import { StatusBar } from 'expo-status-bar';
import { SafeAreaProvider, SafeAreaView } from
'react-native-safe-area-context';
import {
  StyleSheet,
  Text,
  View,
  Image,
  TouchableOpacity,
  Platform,
  KeyboardAvoidingView,
} from 'react-native';

export default function MainMenu({route}) {
```

```
const {name} =route.params;

console.log('App executed');

return (
  <SafeAreaProvider>
    <SafeAreaView style={styles.container}>
      <KeyboardAvoidingView
        behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
        style={{ flex: 1 }}
        keyboardVerticalOffset={20}
      >
        <StatusBar style="light" />

        {/* Weather and Settings Section */}
        <View style={styles.WSContainer}>
          <TouchableOpacity style={styles.WeatherOutput}>
            <Image
              source={require('./assets/weathericonsn.png')}
              fadeDuration={500}
              style={styles.WeatherImg}
              alt="Weather"
            />
          </TouchableOpacity>

          <TouchableOpacity style={styles.SettingsOutput}>
            <Image
              source={require('./assets/settingsiconsn.png')}
              fadeDuration={500}
              style={styles.SettingsImg}
              alt="Settings"
            />
          </TouchableOpacity>
        </View>

        <View>
          <Text style={styles.Text}>
            Hello {name}!
          </Text>
        </View>
      </KeyboardAvoidingView>
    </SafeAreaView>
  </SafeAreaProvider>
)
```

```
</View>

/* School Map Section */
<View style={styles.Map}>
  <Text style={styles.MapText}>School Map</Text>
</View>

/* Features Section */
<View style={styles.Features}>
  <View style={styles.RowContainer}>
    /* Report Box */
    <TouchableOpacity>
      <View style={styles.Box}>
        <Text style={styles.Label}>Report</Text>
      </View>
    </TouchableOpacity>

    /* ETA Box */
    <View style={styles.ETAWrap}>
      <Text style={styles.Label}>ETA</Text>
      <View style={styles.ETABox}></View>
    </View>

    /* Common Places */
    <TouchableOpacity>
      <View style={styles.Box}>
        <Text style={styles.Label}>Common Places</Text>
      </View>
    </TouchableOpacity>
  </View>
</View>

/* Search Location Section */
<View style={styles.SearchLocation}>
  <TouchableOpacity>
    <Text style={styles.SearchLabel}>Search
    Location</Text>
  </TouchableOpacity>
</View>
```

```
        </KeyboardAvoidingView>
      </SafeAreaView>
    </SafeAreaProvider>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#38b6ff',
    paddingHorizontal: 24,
  },
  WSContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    marginVertical: 20,
  },
  WeatherOutput: {
    flex: 1,
    justifyContent: 'center',
  },
  SettingsOutput: {
    flex: 1,
    alignItems: 'flex-end',
  },
  WeatherImg: {
    width: 80,
    height: 80,
    borderRadius: 40,
  },
  SettingsImg: {
    width: 70,
    height: 70,
    borderRadius: 35,
  },
  Text: {
    fontSize: 36,
    fontWeight: 'bold',
  },
  Map: {
```

```
    height: '45%',
    marginVertical: 10,
    borderColor: 'black',
    borderWidth: 2,
    borderRadius: 10,
    justifyContent: 'center',
    alignItems: 'center',
  },
  MapText: {
    fontSize: 36,
    fontWeight: 'bold',
  },
  Features: {
    height: '20%',
    borderColor: 'black',
    borderWidth: 2,
    borderRadius: 10,
    justifyContent: 'center',
  },
  RowContainer: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    alignItems: 'center',
  },
  Box: {
    width: 110,
    height: 140,
    borderWidth: 2,
    borderRadius: 10,
    justifyContent: 'center',
    alignItems: 'center',
    borderColor: 'black',
  },
  ETAWrap: {
    justifyContent: 'center',
    alignItems: 'center',
  },
  ETABox: {
    width: 140,
```

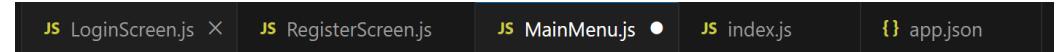
```

        height: 114,
        borderWidth: 2,
        borderRadius: 10,
        borderColor: 'black',
        marginTop: 5,
    },
    Label: {
        fontSize: 20,
        fontWeight: 'bold',
    },
    SearchLocation: {
        height: '10%',
        marginTop: 10,
        borderColor: 'black',
        borderWidth: 2,
        borderRadius: 10,
        justifyContent: 'center',
        alignItems: 'center',
    },
    SearchLabel: {
        fontSize: 18,
        fontWeight: 'bold',
    },
});

```

Description:

This is at MainMenu.js file:



This version is mainly composed of initialising and setting up the Main Menu page, structuring it in a way so that all the buttons and design features match the plans formulated in the designs section. Most of functionality is through using the `</TouchableOpacity>` Feature, this gives the user something to press and provides a responsive output, we can then later on implement any functions embedded in the press. Hence, this part of the project is a vital component which many other parts depend on.

The weather and Settings Section is located at the top of the screen. Since the positioning of the items are unorthodox, I had to use a `flexDirection` component and set it to be 'row' to ensure this is

setup correctly.

I used images:



My images were obtained from the internet and I used Canva to set the background to match the screen background.

Using alternatives, in the chance the image isn't uploaded with "Weather" and "Settings" respectively. This is to allow users to go into the Settings Page, weather Modal etc. meeting SC.19
The different pages should be easy to navigate through.

Using a routing function, I used the default value of "name"(Guest) to present a customisable/personalised feature to the app. This can provide a more comfortable user experience for the user.

I implemented the personalising using a simple Text component with the text Hello {name}! where the "name" is the variable.

The School Map Section is bare bones at this point of the project but does have a title to signify its purpose and the size of the map is set as final.

Now implementing the Features section and the sectioning of the boxes and labels was a feat. I used various containers to set the boxes customly and to ensure the specific boxes were touchable opacity and the others were not. This is important as it meets SC.16
- Aesthetically pleasing and functional design to the user which promotes the user experience. Also to size it in a design perspective, simple and sleek manner I used justifyContent components etc.

Test Table 16 – Navigation Between Screens

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.27	Navigation between different screens. “Sign-Up” button should navigate to the Register Screen etc.	Press each button that is meant to navigate to different screens and see if it navigates you to each screen respectively.	The buttons once pressed should navigate users to appropriate screens, once checked if no errors are present.	The “Sign-Up” button navigates the user to the Register Screen.	Yes, the navigation works as expected.
1.28	The Guest Button should function and be pressable. It should take users directly to the Main Menu and set the name to “Guest”.	Press the Guest button and check if it takes the user to the Main Menu and the name is set to “Guest”.	The Main Menu should be presented to the user by pressing the Guest button and the name is set to “Guest”.	When the Guest button is pressed users are taken to the Main Menu straight away and the name is set to “Guest”.	Yes, the Guest button directs users to the Main Menu and the name is set to Guest.
1.29	Users should be able to navigate back to the Login Page from the Register Screen.	Press the Register Screen to navigate to the relevant screen and then visual check for the option to return to the Login Page and press it.	The user should be navigated to the Login Page and the button should be visible and clear to users.	The button is clear and visible with text: “Already have an account? <u>Sign in</u> ”. However on press the program presents an error message and doesn’t run expectantly.	No HyperLink to Solution

Evidence Table 16

Test ID	Screenshot/Video	Evaluation
1.27 - 1.29 + error in 1.29	<p>https://youtu.be/VW7UP0UFq28</p>  <pre data-bbox="410 698 891 1417"> Touchableopacity.props.onPress C:\Users\iguna\SchoolNav\screens\RegisterScreen.js _performTransitionSideEffects C:\Users\iguna\SchoolNav\node_mo...Libraries\Pressability\Pressability.js _receiveSignal C:\Users\iguna\SchoolNav\node_mo...Libraries\Pressability\Pressability.js responderEventHandlers.onResponderRelease C:\Users\iguna\SchoolNav\node_mo...Libraries\Pressability\Pressability.js executeDispatch C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js executeDispatchesInOrder C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js executeDispatchesAndRelease C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js executeDispatchesAndReleaseTopLevel C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js forEachAccumulated C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js runEventsInBatch C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js runExtractedPluginEventsInBatch C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js batchedUpdates\$1\$argument_0 C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js batchedUpdates\$1 C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js dispatchEvent C:\Users\iguna\SchoolNav\node_m...implementations\ReactFabric-dev.js Collapse all 15 frames </pre>	<p>The video presents the navigation system within the application and how the buttons can navigate users to the different screens. From the Login in screen to the Register screen as in Test ID 1.27 or the Login in directly to the Main Menu screen through the Guest function as in Test ID 1.28.</p> <p>There is a major error as the only way for users to return to the Login Page is either by entering the Register Screen field details or the “Already have an account? <u>Sign in</u>” button, so it is vital this works but this is outputting an error stating “Property ‘navigation’ doesn’t exist”.</p>

Error in returning to Login Screen from the Register Screen: Test ID 1.29

Relevant Source Code:	<p>From:</p> <pre>export default function RegisterScreen() { const [name, setName] = useState(''); const [email, setEmail] = useState(''); const [password, setPassword] = useState(''); const [confirmPassword, setConfirmPassword] = useState(''); const [errors, setErrors] = useState({});</pre> <p>To:</p> <pre>export default function RegisterScreen({navigation}) { const [name, setName] = useState(''); const [email, setEmail] = useState(''); const [password, setPassword] = useState(''); const [confirmPassword, setConfirmPassword] = useState(''); const [errors, setErrors] = useState({});</pre>
Description:	<p>The change was in defining the navigation component in the function for the screen. This small implementation was a breakthrough for this error in this part of the project, allowing seamless navigation between different screens as the navigation tag is now defined and included.</p> <p>Resolving the error where it was outputted an error stating “Property ‘navigation’ doesn’t exist”.</p>

Test Table 17 – Initialising Log-In Screen

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
1.29-FIX	Users should be able to navigate back to the Login Page from the Register Screen.	Press the Register Screen to navigate to the relevant screen and then visual	The user should be navigated to the Login Page and the button should be	The button is clear and visible with text: “Already have an account? <u>Sign in</u> ”. On press	Yes – fixed from previous error

		check for the option to return to the Login Page and press it.	visible and clear to users.	the program directs the user to the Login Screen.	
--	--	--	-----------------------------	---	--

Evidence Table 17

Test ID	Screenshot/Video	Evaluation
1.29 - FIX	https://youtu.be/N9bkb8lZeI4	The navigation between the Register Screen and the Login In Screen is now fixed and works as expected.

Test Table 18 - Initialising Main Menu

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
2.1	Main Menu Page should be initialised and set up to take in name from database/other screens.	Menu Menu page should open when pressed. Visual check for name output.	The Main Menu should be running and it should be initialised.	The Main Menu runs successfully and is presented on screen with the name from the database/asynchronous storage.	Yes, the Main Menu takes in the correct name for the greeting.
2.2	The Weather and Settings sections should appear side by side at the top of the screen.	Open the screen and visually check if the two sections are positioned horizontally next to each other.	The two sections should be aligned properly, without any overlap or gaps between them.	The Weather and Settings Icon of the Main Menu Screen is presented aesthetically and in a quality manner,	Yes, the icons are present.

				where they are presented next to each other, please have a look below.	
2.3	The Weather section should be clear in purpose and allow user interaction.	Examine the design of the Weather section to ensure it reflects its purpose. Test interaction by attempting to access weather information.	The section should clearly indicate it is for weather updates and should respond appropriately to user interaction.	The Weather section is presented using a high quality vector image which is a clear weather icon.	Yes, the weather icons are clear in their purpose.
2.4	The Settings section should be clear in purpose and allow user interaction.	Review the appearance of the Settings section to confirm its purpose. Test it by attempting to access settings options.	The section should clearly represent its function and respond appropriately to user interaction.	The Settings Section is presented with the standard Settings Cog which is clear to the purpose of the section.	Yes, the settings icon is clear in its purpose.
2.5	The title for the "Greeting" section should be visible and clearly indicate its purpose.	Open the screen and check if the title for the map section is displayed prominently and is easy to read.	The title should be clearly visible and help users understand the purpose of the section immediately.	The greeting is large and present on the screen and followed by a large chunk of the map, which is clear to the purpose and necessity of that part.	Yes, the greeting is displayed and clear.
2.6	The layout should maintain a professional and	Review the design and placement of elements on the screen to	The screen should look organized and visually appealing,	The layout is consistent and usable, the buttons are pressable	Yes, there is a professional tone to the

	user-friendly appearance.	check for consistency and usability.	with all elements positioned intuitively.	and have feedback and the theme is consistent and user-friendly as the Test ID desires.	application.
2.7	Navigation back to the Login Page should work correctly.	Test the functionality by attempting to navigate from the Main Menu to the Login Page.	The app should seamlessly return to the Login Page without any disruptions or delays. This is for users who use Guest mode, but decide to return back to the Login In page.	Users are able to navigate back to the Log-in page if they decide to Sign-Up or Log-In to an existing account. The navigation is clear, located at the top and successfully navigates users back to the Login Screen.	Yes, users can navigate to the Login Page successfully.
2.8	The Features for Report and Common Places should be present and pressable.	Visual check for the Report and Common Places button and press it to see if it is interactable.	There should be two buttons reflecting the purpose of Reporting and for Common Places respectively.	The Button for Report and Common Places are successfully rendered and present, they are sandwiched by the ETA sign, just as the Designs implies.	Yes, the features are present.
2.9	The ETA should be a prominent section which is not	Visual check for an ETA box with sufficient space to present	There should be a prominent box on the Main Menu screen	The ETA is a prominent button on the page which is a contrasting	Yes, the ETA is present and not pressable.

	pressable but is to present values.	information.	for the ETA.	white colour, with a clear title “ETA”. It is not pressable.	
2.10	The Search Location button should be visual and prominent on screen.	Visual check for Search Location button and press to test interactability.	There should be a box for Search Location and it should be interactable.	The Search Location is located prominently at the bottom of the page and has a clear title for “Search Location”	Yes, the search location feature is present!
2.11	Retrieve name from Register Screen and display on Main Menu	Simulate navigating from Register Screen to Main Menu with a user-defined name inputted during registration.	The name should be displayed on the Main Menu after a successful login.	The name is correctly displayed on the Main Menu if the user Registers straight away then Logs in. But if they Register and exit the application, users are able to enter the Main Menu but the name doesn't appear, it presents as null.	No HyperLink to Solution
1.33 - REVISIT	Navigation to Main Menu after successful login	Simulate a login process with valid email and password.	The app should navigate to the Main Menu page and display the user's name.	Navigation to the Main Menu occurs, and the user's name is displayed.	YES - confirmation of completion was delayed due to the need to Initialise

					Main Menu.
--	--	--	--	--	---------------

Evidence Table 18

Test ID	Screenshot/Video
2.1-2.10	<p>https://youtu.be/NOXF2iDBZ50</p> <p>https://youtu.be/_77OF7h-LQU</p>
2.11	https://youtube.com/shorts/dDt4ghv88A4
Evaluation:	<p>The first video is an overall video showcasing the project up to this point, the Main Menu is in a rough place right now, but the second video goes more in depth of the Main Menu.</p> <p>It meets Test ID 2.01 because the Main Menu runs successfully and is presented on the screen with the correct name. By presenting the icons and sections for Settings and Weather it successfully meets Test ID 2.02.</p> <p>The use of an image clearly indicative and purposeful is chosen for the Weather and Settings Buttons, meeting both Test ID 2.03 and Test ID 2.04.</p> <p>The greeting is very prominent and indicative of purpose meeting Test ID 2.05. Alongside, the use of padding and comfortable and professional structure ensures the meeting of Test ID 2.06 and SC.16 - Aesthetically pleasing and functional design to the user which promotes the user experience. Navigation is a vital component of this app as users should be able to navigate and manoeuvre around different screens, signified in Test ID 2.07 and since the Navigation back to Login page works, this is met.</p> <p>The Features, Report, Common places are present and clear in what they do and interactable, meeting Test ID 2.08 and Test ID 2.09.</p> <p>The retrieval of name from the Main Menu was successful and accurate and the Search Location button is prominent and touchable, Test ID 2.10/2.11.</p> <p>A very important part is if the user can be navigated to the Main Menu once logged in successfully and this works as expected, hence meeting 1.33 Revist.</p>

	There was an error as the name is correctly displayed on the Main Menu if the user Registers straight away then Logs in. But if they Register and exit the application, users are able to enter the Main Menu but the name doesn't appear, it presents as null.
--	---

Test Table 19 – Fixing name personalisation feature

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
2.11-FIX	Retrieve name from Register Screen and display on Main Menu	Simulate navigating from Register Screen to Main Menu with a user-defined name inputted during registration.	The name should be displayed on the Main Menu after a successful login.	The name is correctly displayed on the Main Menu in all cases.	Yes – fixed from previous error

Version 6 - Implementing Map of School

Implementing the Map of the School

Relevant Source Code:

```
/* School Map Section */
<View style={styles.Map}>
  <MapView
    style={{ flex: 1, width: '100%', borderRadius: 10 }}
  // Ensures MapView fills the View
  initialRegion={{
    latitude: 51.568352,
    longitude: 0.085941,
    latitudeDelta: 0.005,
    longitudeDelta: 0.005,
  }}
  />
</View>
```

```
Map: {
  height: '45%', // Ensures the container height
  marginVertical: 10,
  borderColor: 'black',
  borderWidth: 2,
  borderRadius: 10,
  overflow: 'hidden', // Prevents the map from overflowing
  rounded corners
},
```

```
mapType="hybrid" // Use "hybrid" for satellite + labels
```

```
import React, { useEffect, useState, useRef } from 'react';
import MapView, { PROVIDER_GOOGLE, Marker } from
'react-native-maps';
import { StatusBar } from 'expo-status-bar';
import { SafeAreaProvider, SafeAreaView } from
'react-native-safe-area-context';
import {
  StyleSheet,
  Text,
```

```
View,
Image,
TouchableOpacity,
Platform,
KeyboardAvoidingView,
Dimensions,
} from 'react-native';
import * as Location from 'expo-location';

// Get screen dimensions
const { width, height } = Dimensions.get('window');

// Scaling functions
const BASE_WIDTH = 375; // Reference width (e.g., iPhone 8)
const BASE_HEIGHT = 667; // Reference height (e.g., iPhone 8)
const scaleWidth = (size) => (width / BASE_WIDTH) * size;
const scaleHeight = (size) => (height / BASE_HEIGHT) * size;

export default function MainMenu({ route }) {
  const { name } = route.params;
  const [userLocation, setUserLocation] = useState(null);
  const [locationPermission, setLocationPermission] =
  useState(false);
  const mapRef = useRef(null); // Reference to the MapView
component

  // Request location permissions and fetch user location with
highest accuracy
  useEffect(() => {
    (async () => {
      const { status } = await
Location.requestForegroundPermissionsAsync();
      setLocationPermission(status === 'granted');
      if (status === 'granted') {
        const location = await Location.getCurrentPositionAsync({
          accuracy: Location.Accuracy.Highest, // Request highest
accuracy
        });
        setUserLocation({
```

```

        latitude: location.coords.latitude,
        longitude: location.coords.longitude,
        latitudeDelta: 0.0004,
        longitudeDelta: 0.0004,
    });
}
})());
}, []);
// Function to recenter the map
const recenterMap = () => {
    if (mapRef.current && userLocation) {
        mapRef.current.animateToRegion(userLocation, 1000); // Smooth
        animation to user location
    }
};

return (
<SafeAreaProvider>
<SafeAreaView style={styles.container}>
<KeyboardAvoidingView
    behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
    style={{ flex: 1 }}
    keyboardVerticalOffset={20}
>
<StatusBar style="light" />

{/* Weather and Settings Section */}
<View style={styles.WSContainer}>
    <TouchableOpacity style={styles.WeatherOutput}>
        <Image
            source={require('./assets/weathericonsn.png')}
            fadeDuration={500}
            style={styles.WeatherImg}
            alt="Weather"
        />
    </TouchableOpacity>

    <TouchableOpacity style={styles.SettingsOutput}>
        <Image

```

```
        source={require('./assets/settingsiconsn.png')}
        fadeDuration={500}
        style={styles.SettingsImg}
        alt="Settings"
      />
    </TouchableOpacity>
  </View>

  {/* Greeting */}
  <View>
    <Text style={styles.Text}>Hello {name}!</Text>
  </View>

  {/* Map Section */}
  <View style={styles.Map}>
    <MapView
      ref={mapRef} // Attach the ref to the MapView
      style={StyleSheet.absoluteFillObject}
      provider={PROVIDER_GOOGLE}
      mapType="hybrid"
      userLocationUpdateInterval={1000000}
      initialRegion={{
        latitude: 51.5685,
        longitude: 0.0859,
        latitudeDelta: 0.001,
        longitudeDelta: 0.001,
      }}
      region={userLocation} // Center the map on the user's
location
      showsUserLocation={true} // Show the default blue dot
      followsUserLocation={true} // Automatically follow
the user's location
      pitch={45} // Add a 3D tilt to the map
      heading={0} // Set the map's heading (0 degrees means
north-up)
    >
      {userLocation && (
        <Marker
          coordinate={{


```

```
        latitude: userLocation.latitude,
        longitude: userLocation.longitude,
    )}
    title="You are here"
/>
)}
```

```
</MapView>
<TouchableOpacity style={styles.RecenterButton}
onPress={recenterMap}>
    <Text
style={styles.RecenterButtonText}>Recenter</Text>
</TouchableOpacity>
</View>

/* Features Section */
<View style={styles.Features}>
    <View style={styles.RowContainer}>
        <TouchableOpacity>
            <View style={styles.Box}>
                <Text style={styles.Label}>Report</Text>
            </View>
        </TouchableOpacity>

        <View style={styles.ETAWrap}>
            <Text style={styles.Label}>ETA</Text>
            <View style={styles.ETABox}></View>
        </View>

        <TouchableOpacity>
            <View style={styles.Box}>
                <Text style={styles.Label}>Common Places</Text>
            </View>
        </TouchableOpacity>
    </View>
</View>

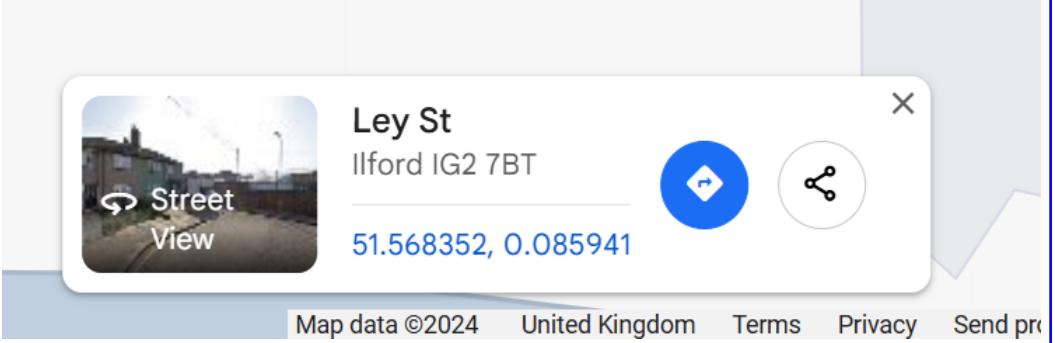
/* Search Location Section */
<View style={styles.SearchLocation}>
    <TouchableOpacity>
```

```
        <Text style={styles.SearchLabel}>Search
Location</Text>
        </TouchableOpacity>
    </View>
</KeyboardAvoidingView>
</SafeAreaView>
</SafeAreaProvider>
);
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#38b6ff',
    paddingHorizontal: scaleWidth(16),
  },
  WSContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    marginVertical: scaleHeight(16),
  },
  WeatherOutput: {
    flex: 1,
    justifyContent: 'center',
  },
  SettingsOutput: {
    flex: 1,
    alignItems: 'flex-end',
  },
  WeatherImg: {
    width: scaleWidth(70),
    height: scaleWidth(70),
    borderRadius: scaleWidth(35),
  },
  SettingsImg: {
    width: scaleWidth(60),
    height: scaleWidth(60),
    borderRadius: scaleWidth(30),
  },
});
```

```
Text: {
  fontSize: scaleHeight(24),
  fontWeight: 'bold',
  textAlign: 'center',
  marginVertical: scaleHeight(8),
},
Map: {
  flex: 1,
  marginVertical: scaleHeight(10),
  borderRadius: scaleWidth(10),
  overflow: 'hidden',
},
RecenterButton: {
  position: 'absolute',
  bottom: scaleHeight(10),
  right: scaleWidth(10),
  backgroundColor: '#fff',
  padding: scaleHeight(10),
  borderRadius: scaleWidth(10),
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.25,
  shadowRadius: 3.84,
  elevation: 5,
},
RecenterButtonText: {
  fontSize: scaleHeight(14),
  fontWeight: 'bold',
  color: '#000',
},
Features: {
  height: scaleHeight(120),
  borderColor: 'black',
  borderWidth: 2,
  borderRadius: scaleWidth(10),
  justifyContent: 'center',
  marginVertical: scaleHeight(8),
},
RowContainer: {
```

```
    flexDirection: 'row',
    justifyContent: 'space-around',
    alignItems: 'center',
},
Box: {
    width: scaleWidth(90),
    height: scaleHeight(100),
    borderWidth: 2,
    borderRadius: scaleWidth(10),
    justifyContent: 'center',
    alignItems: 'center',
},
ETAWrap: {
    justifyContent: 'center',
    alignItems: 'center',
},
ETABox: {
    width: scaleWidth(110),
    height: scaleHeight(90),
    borderWidth: 2,
    borderRadius: scaleWidth(10),
    borderColor: 'black',
    marginBottom: scaleHeight(5),
},
Label: {
    fontSize: scaleHeight(16),
    fontWeight: 'bold',
},
SearchLocation: {
    height: scaleHeight(60),
    borderColor: 'black',
    borderWidth: 2,
    borderRadius: scaleWidth(10),
    justifyContent: 'center',
    alignItems: 'center',
    marginVertical: scaleHeight(8),
},
SearchLabel: {
    fontSize: scaleHeight(16),
```

	<pre> fontWeight: 'bold', },); </pre>
Description:	<p>Installed the react native map component in the bash using this command: “npm install react-native-maps” to allow use of it in my application.</p> <p>Used google maps to find the initial location using longitude and latitude.</p>  <p>I was receiving a zoomed out map of London rather than a zoomed in map of the school, hence I had to adjust my latitudeDelta and longitudeDelta parameters, this is to meet one of the more important Success Criterias: SC.02</p> <p>The map should be accurate, easy to understand and reflective of the building.</p> <p>I don't have to set the Provider since, the default which is Google Maps is suited for this application. For IOS, the map used would be Apple Maps.</p> <p>Decided to use mapType to be hybrid as it's a mix of Standard map type and Satellite map type so the user can easily understand the map and what it's representing.</p> <p>This meant that I had to adjust the longitudeDelta and latitudeDelta to zoom accordingly.</p> <p>e.g</p> <pre> latitudeDelta: 0.002, longitudeDelta: 0.002, </pre> <p>is too zoomed out so I increased it to...</p> <pre> latitudeDelta: 0.001, </pre>

```
longitudeDelta: 0.001,
```

This presents the school and users can manoeuvre around the application as needed.

I used this documentation on GitHub to assist me in meeting my Test Plans for this application:

[react-native-maps/docs/mapview.md at master](https://react-native-maps.github.io/react-native-maps/docs/mapview.md)

I wanted to make the application dynamic for different screen sizes so I created variables and functions to assist with this, where I can then use percentage values to set the position of elements.

Since we are using location for the map, I set up a function to request location permissions and fetch user location with highest accuracy, since my application is in a small area and needs a really precise GPS signal. Then I set the user's location to be the current location they are at.

I began by installing the `react-native-maps` component using the command `npm install react-native-maps`, which allowed me to integrate Google Maps into the application. This was essential because I needed a reliable map service to display the school location and provide an interactive interface for users.

I didn't need to specify the map provider explicitly because the default provider, Google Maps, worked well for my needs. For iOS users, Apple Maps would be used automatically, so I didn't need to worry about cross-platform differences. I also chose the `hybrid` map type because it offers a mix of both the standard and satellite map views. This choice made the map more informative, as users could better understand their surroundings with satellite imagery, while still having clear road and landmark information from the standard view.

Since the map is location-based, I needed to ensure that the application accurately reflected the user's current position. To achieve this, I set up a function to request location permissions and fetch the user's location with the highest level of accuracy. This was especially important because the app is focused on a small area (such as a school), where precise GPS data is crucial for the navigation to work properly. Once the user's location was fetched, I used this data to update the map's center so that users would always be able to see their current position.

I also wanted the map to be dynamic across different screen sizes.

To address this, I created variables and functions to scale the layout of the app, adjusting the positions of elements using percentage values. This ensured that the app would be responsive and visually appealing, no matter what device the user was on.

Finally, I wanted to give the map a more immersive feel by adding a 3D effect. By adding a 3D effect I am effectively meeting SC.01 **The application should have a smooth and responsive user interface (UI) system.** To do this, I used the **pitch** property to tilt the map to 45 degrees. This created a pseudo-3D effect, which made the map appear more engaging and easier to navigate, especially for users unfamiliar with the area. Additionally, I kept the map's heading pointing **North (heading={0})** to maintain consistency and ensure users always know which way they're facing.

By making these adjustments, I was able to create an intuitive and responsive map that provides users with a dynamic navigation experience, while ensuring that it works smoothly across different devices and accurately reflects the user's current location.

To enhance the user experience, I added a recenter feature that allows users to quickly return the map's focus to their current location. This was an important addition because, while navigating the map, users might zoom in or move around, potentially losing track of their initial position. The recenter feature ensures that, with a simple tap, the map will smoothly adjust to display the user's current coordinates.

To implement this, I used a **recenterMap** function to meet SC.10 **Compass feature/Recenter feature and map adapting to user orienting the phone.** This function checks if the map reference (**mapRef.current**) is available and if the user's location (**userLocation**) is set. If both conditions are met, the function uses the **animateToRegion** method on the map reference, which smoothly animates the map's view to the user's current location. I set the duration of the animation to 1000 milliseconds to give it a smooth, fluid effect, making the transition feel more natural.

I positioned the recenter button in the bottom-right corner of the screen for easy access. This placement is intuitive, as it's where users often expect to find common actions, and it doesn't obstruct the map view. The button is styled with a white background, rounded corners, and a shadow effect, making it visually distinct and easy to press, the addition of all these features contributes heavily to SC.16 - **Aesthetically pleasing and functional design to the user which promotes the user experience.**

By adding this recenter feature, I ensured that users could always

	quickly return to their location, improving the app's usability, especially when they are navigating through different areas of the map. It also helps users stay oriented within the map, enhancing the overall navigation experience.
--	---

Test Table 20 – Implementing Map

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
2.12	Verify that the map displays the user's current location correctly.	Check if the map is initialized to show the user's location as the center of the map. This can be tested by allowing the app to request location access and then confirming that the user's location is shown on the map with a visual marker or blue dot.	The map should center on the user's current location, with a marker or blue dot clearly indicating the position.	The map correctly centers on the user's location, and a marker or blue dot is displayed, confirming that the user's location is accurately shown on the map.	Yes, the map displays the accurate location.
2.13	Verify that the recenter button works as intended.	After manually navigating away from the user's location, press the recenter button and observe if the	The recenter button should recenter the map on the user's location, with a smooth animation back to the user's position.	The recenter button works as expected, animating the map smoothly to the user's location.	Yes, the recenter buttons do work as intended.

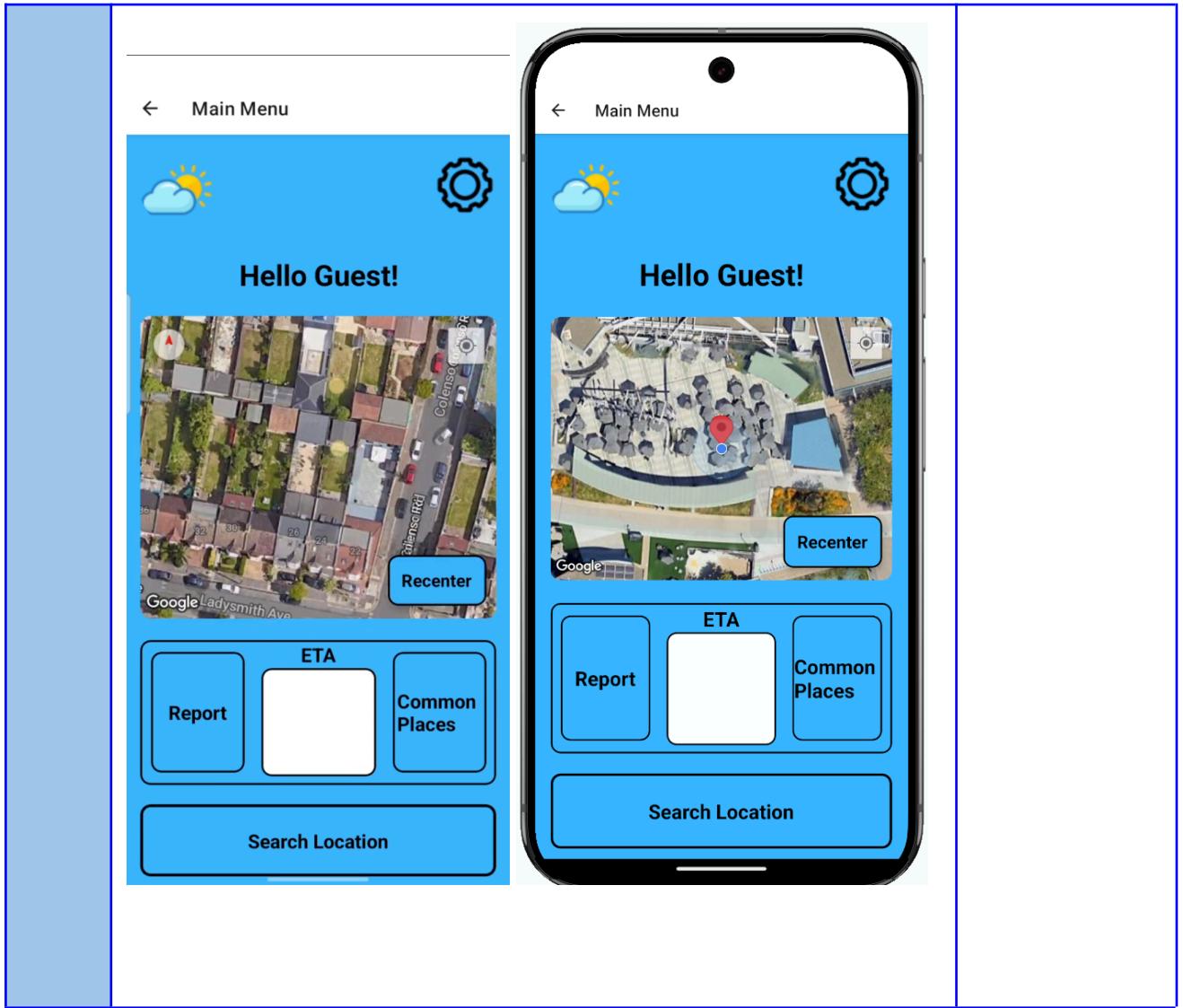
		map smoothly animates back to the user's current position.			
2.14	Verify the map's zoom level is appropriate for the current location.	Check if the map is zoomed in enough to show detailed information for the user's location. This can be tested by visually inspecting the map after it loads to ensure it is neither too zoomed out nor too zoomed in.	The map should be zoomed in sufficiently to show a clear and detailed view of the user's location, without being too zoomed out or too zoomed in.	The map zoom level is set correctly, providing a clear and detailed view of the user's location without distortion or excessive zoom.	Yes, the map is presentable.
2.15	Verify that the map type is set correctly.	Check if the map is rendered with the correct type, such as hybrid or satellite, depending on the requirements.	The map should display in the selected map type (e.g., hybrid, which combines standard map and satellite view).	The map type is set correctly, and the map displays in the intended view, providing both map details and satellite imagery.	Yes, the map type is correct.
2.16	Verify the correct handling of location permissions.	Test if the application asks for location permissions when needed and if the map displays the	The app should request location permissions when necessary. If permission is granted, the	The app correctly handles location permissions, requesting them when necessary,	Yes, location permissions are handled appropriately.

		<p>user's location only after permission is granted. If permissions are denied, the app should not attempt to access the user's location.</p>	<p>map should display the user's location; if permission is denied, the app should handle it gracefully without attempting to fetch location data.</p>	<p>and displays the user's location only after permission is granted.</p>	
2.17	Verify that the map is responsive and interactive.	<p>Test whether the user can interact with the map, including zooming in/out, panning, and moving around the map without issues.</p>	<p>The map should respond to user interactions such as zooming and panning. The map should be interactive and smooth when the user zooms or drags the map.</p>	<p>The map is responsive and interactive, with smooth zooming and panning as expected.</p>	<p>Yes, the map can be maneuvered around.</p>
2.18	Verify that the app adjusts the map display based on the device's screen size.	<p>Test the map on various devices with different screen sizes to ensure the map scales properly to fit the screen without distortion.</p>	<p>The map should adjust its size and scaling dynamically to fit different screen sizes, ensuring it looks properly scaled on all devices.</p>	<p>The map scales appropriately on different devices, displaying correctly without clipping or distortion.</p>	<p>Yes, the app does change and adjust depending on the screen size.</p>

2.19	Verify the user's location updates dynamically as they move.	Move the device to a different location and check if the map updates the user's position in real-time.	The map should update dynamically as the user moves, with the location marker following the user in real-time.	The map does not dynamically update as the user moves, and the location marker doesn't follow the user's position. It seems the location isn't refreshing.	No - the marker doesn't present the current location as I move around. HyperLink to Solution
------	--	--	--	--	---

Evidence Table 20

Test ID	Screenshot/Video	Evaluation
2.12-2 .19	https://youtube.com/shorts/jPi-gfyAeTk https://youtube.com/shorts/wux6yhWX2eI https://youtube.com/shorts/2i71Tp35bVs?feature=share	The first video is for Test ID 2.12 This is for Test IDs: 2.13 - 2.15, 2.17 Test ID 2.16 Test ID 2.18 Here are two different phone screens and the Main Menu and the Map is presented accurately on them.



Fixing the map not dynamically adjusting the location marker as you move: Test ID 2.19

Relevant Source Code:

```
From: <MapView
      ref={mapRef} // Attach the ref to the MapView
      style={StyleSheet.absoluteFillObject}
      provider={PROVIDER_GOOGLE}
      mapType="hybrid"
      userLocationUpdateInterval={1000000}
      region={userLocation} // Center the map on the user's
      location
      showsUserLocation={true} // Show the default blue dot
      followsUserLocation={true} // Automatically follow
```

```
the user's location
    pitch={45} // Add a 3D tilt to the map
    heading={0} // Set the map's heading (0 degrees means
north-up)
    >
```

To:

```
<MapView
    ref={mapRef} // Attach the ref to the MapView
    style={StyleSheet.absoluteFillObject}
    provider={PROVIDER_GOOGLE}
    mapType="hybrid"
    userLocationUpdateInterval={1000}
    region={userLocation} // Center the map on the user's
location
    showsUserLocation={true} // Show the default blue dot
    followsUserLocation={true} // Automatically follow
the user's location
    pitch={45} // Add a 3D tilt to the map
    heading={0} // Set the map's heading (0 degrees means
north-up)
    >
```

Description: There was a logic error within the MapView component, it seems the base default value for the units was not micro seconds but 1/1000 of a second. This meant my original value 1000000 was too long of a refresh time, since I wanted the screen to refresh every second.

So I switched the userLocationUpdateInterval={1000} to contain a value of 1000 than 1000000, since the original 1000000 meant the marker was refreshing every 1000 seconds which was too long, hence why we couldn't see the marker refresh as I moved around.

Test Table 21 – Fixing the map not dynamically adjusting the location marker as you move: Test ID 2.19

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
2.19 -FIX	Verify the user's location updates dynamically as they move.	Move the device to a different location and check if the map updates the user's position in real-time.	The map should update dynamically as the user moves, with the location marker following the user in real-time.	The map dynamically updates as the user moves, and the location marker follows the user's position.	Yes – fixed from previous error

Evidence Table 21

Test ID	Screenshot/Video	Evaluation
2.19	https://youtube.com/shorts/62eYNTxhhI4?feature=share	This video shows how the marker changes as I move around a position on the map. This is now where my position on the map refreshes every second. This is important as this is what will inform the user of their current location, a vital component for the main purpose of SchoolNav which routes users.

Implementing Search Location Feature and Map of School

Relevant
Source
Code:

```
// Hardcoded places
const places = [
  { id: 1, name: 'Main Hall', latitude: 51.56853759709549,
    longitude: 0.08591651916503908 },
  { id: 2, name: 'Library', latitude: 51.56863096032261, longitude:
    0.08657097816467287 },
  { id: 3, name: 'Gym', latitude: 51.56893105511125, longitude:
    0.08645296096801759},
  { id: 4, name: 'Dining Hall', latitude: 51.56845423690923 ,
    longitude: 0.08490264415740968 },
  { id: 5, name: 'Main Reception', latitude: 51.56825083741333 ,
    longitude: 0.08591651916503908 },
  { id: 6, name: 'Astro Turf', latitude: 51.56926115709083,
    longitude: 0.08649051189422609 },
  { id: 7, name: 'Sports Hall', latitude: 51.568704317009434,
    longitude: 0.08694648742675783},
  { id: 8, name: '6th Form Common Room', latitude:
    51.56905109247207, longitude: 0.08697867393493652 },
  { id: 9, name: 'Boys Toilet', latitude: 51.5687476640869,
    longitude: 0.08532643318176271},
  { id: 10, name: 'Girls Toilet', latitude: 51.5687509984758,
    longitude: 0.08657097816467287 },
  { id: 11, name: 'Hub', latitude: 51.56813413237369,
    longitude: 0.08723616600036622},
  { id: 12, name: 'Lecture Theatre', latitude: 51.56904091917293,
    longitude: 0.08667417918180487},
  { id: 13, name: 'Margaret Evans Centre', latitude:
    51.56799613868321, longitude: 0.08479976208794593},
  { id: 14, name: 'Staff Room', latitude: 51.56857966263135 ,
    longitude: 0.08640148787738777},
  { id: 14, name: '', latitude: 51.56870136811619,
    longitude: 0.08634069124885624},
  { id: 16, name: 'Art Tech', latitude: 51.56870636967905,
    longitude: 0.08564644612155004},
  { id: 17, name: 'DT Tech', latitude: 51.56885641704665,
```

```
longitude:0.08710869704221746},
  { id: 18, name: 'SCI Staff', latitude: 51.56896422857994,
longitude:0.08610331583289277},
  { id: 19, name: '6th Form Private Study', latitude:
51.568921993176176, longitude:0.0870586292497455},
  { id: 20, name: 'Rm 1', latitude: 51.568351260168114,
longitude:0.08608958446001402},
  { id: 21, name: 'Rm 2', latitude: 51.568356084551695,
longitude:0.08615777483627962},
  { id: 22, name: 'Rm 3', latitude: 51.56835263854545,
longitude:0.08623483535680432},
  { id: 23, name: 'Rm 4', latitude: 51.568347814161484,
longitude:0.08632741893440876},
  { id: 24, name: 'Rm 5', latitude: 51.568358841306,
longitude:0.08646934327336986},
  { id: 25, name: 'Rm 6', latitude: 51.56839536885297, longitude:
0.08656081813365635},
  { id: 26, name: 'Rm 7', latitude: 51.56846911301434, longitude:
0.0865597094165782},
  { id: 27, name: 'Rm 8', latitude: 51.56870137193664,
longitude:0.08606907217542314},
  { id: 28, name: 'Rm 10', latitude: 51.56869930432338,
longitude:0.08578245129843065},
  { id: 29, name: 'Rm 11', latitude: 51.568712398995395,
longitude:0.08550691820333078},
  { id: 30, name: 'Rm 12', latitude: 51.56866070940558,
longitude:0.08528904206124288},
  { id: 31, name: 'Rm 13', latitude: 51.56861039818962,
longitude:0.08527518228315767},
  { id: 32, name: 'Rm 14', latitude: 51.568575249185976,
longitude:0.08527241028696508},
  { id: 33, name: 'Rm 15', latitude: 51.568493234801586,
longitude:0.08527407356591965},
  { id: 34, name: 'Rm 16', latitude: 51.56836366559432,
longitude:0.08527739992083117},
  { id: 35, name: 'Rm 17', latitude: 51.56836021958865,
longitude:0.08537441857054517},
  { id: 36, name: 'Rm 18', latitude: 51.56836228715423,
longitude:0.0855734455037993},
```

```

        { id: 37, name: 'Rm 19', latitude: 51.56844154496262,
longitude:0.08540435576434913},
        { id: 38, name: 'Rm 20', latitude: 51.568522870191075,
longitude:0.08540768211926064},
        { id: 39, name: 'Rm 21', latitude: 51.56859454655905,
longitude:0.08542209625602482},
        { id: 41, name: 'Rm 23', latitude: 51.56850701862465,
longitude:0.08718860004323249},
        { id: 42, name: 'Rm 24', latitude: 51.56850770777942,
longitude:0.08683967778706503},
        { id: 43, name: 'Rm 25', latitude: 51.568336097446995,
longitude:0.08681916529923672},
        { id: 44, name: 'Rm 26', latitude: 51.56822927136887,
longitude:0.08682304601274726},
        { id: 45, name: 'Rm 27', latitude: 51.56826442060845,
longitude:0.08698437432521633},
        { id: 46, name: 'Rm 28', latitude: 51.568299569631534,
longitude:0.08696829711257514},
        { id: 47, name: 'Rm 29', latitude: 51.56834298920933,
longitude:0.08696885147121414},
        { id: 48, name: 'Rm 30', latitude: 51.568522180657965, longitude:
0.08696774275393615},
        // { id: 50, name: 'Rm 32', latitude: , longitude:},
        // { id: 51, name: 'Rm 33', latitude: , longitude:},
        // { id: 52, name: 'Rm 34', latitude: , longitude:},
        // { id: 53, name: 'Rm 35', latitude: , longitude:},
        // { id: 54, name: 'Rm 36', latitude: , longitude:},
        // { id: 55, name: 'Rm 37', latitude: , longitude:},
        // { id: 56, name: 'Rm 38', latitude: , longitude:},
];
export default function MainMenu({ route, navigation }) {
  const { name } = route.params;
  const [userLocation, setUserLocation] = useState(null);
  const [locationPermission, setLocationPermission] =
  useState(false);
  const [searchQuery, setSearchQuery] = useState('');
  const [searchResults, setSearchResults] = useState([]);
  const [isModalVisible, setIsModalVisible] = useState(false);
  const mapRef = useRef(null);

```

```
// Request location permissions and fetch user location with
// highest accuracy
useEffect(() => {
  (async () => {
    const { status } = await
    Location.requestForegroundPermissionsAsync();
    setLocationPermission(status === 'granted');
    if (status === 'granted') {
      const location = await Location.getCurrentPositionAsync({
        accuracy: Location.Accuracy.Highest, // Request highest
accuracy
      });
      setUserLocation({
        latitude: location.coords.latitude,
        longitude: location.coords.longitude,
        latitudeDelta: 0.0004,
        longitudeDelta: 0.0004,
      });
    }
  })();
}, []);

// Handle search
const handleSearch = (query) => {
  setSearchQuery(query);
  if (query.trim() === '') {
    setSearchResults([]);
    return;
  }
  const results = places.filter((place) =>
    place.name.toLowerCase().includes(query.toLowerCase())
  );
  setSearchResults(results);
};

// Center map on selected place
const goToPlace = (place) => {
  if (mapRef.current) {
```

```
mapRef.current.animateToRegion(
  {
    latitude: place.latitude,
    longitude: place.longitude,
    latitudeDelta: 0.0004,
    longitudeDelta: 0.0004,
  },
  1000
);
}

setIsModalVisible(false); // Close modal after selecting a place
setSearchQuery('');
setSearchResults([]);
};

}

{showMarkers &&
  places.map((place) => (
    <Marker
      key={place.id}
      coordinate={{{
        latitude: place.latitude,
        longitude: place.longitude,
      }}}
      title={place.name} // Shows the name on marker
    click
      >
      </Marker>
    ))}

<TouchableOpacity
  style={styles.ToggleMarkersButton}
  onPress={() => setShowMarkers(!showMarkers)}
>
  <Text style={styles.ToggleMarkersButtonText}>
    {showMarkers ? 'Hide Markers' : 'Show Markers'}
  </Text>
</TouchableOpacity>
```

```
/* Search Location Section */
    <View style={styles.SearchLocation}>
        <TouchableOpacity onPress={() =>
setIsModalVisible(true)}>
            <Text style={styles.SearchLabel}>Search
Location</Text>
        </TouchableOpacity>
    </View>

/* Modal for Searching Locations */
<Modal
    visible={isModalVisible}
    transparent={true}
    animationType="slide"
>
    <View style={styles.modalContainer}>
        <View style={styles.modalContent}>
            <TextInput
                KeyboardAvoidingView = 'false'
                style={styles.searchInput}
                placeholder="Search for a place..."
                value={searchQuery}
                onChangeText={handleSearch}
            />
            <FlatList
                data={searchResults}
                keyExtractor={(item) => item.id.toString()}
                renderItem={({ item }) => (
                    <TouchableOpacity
                        style={styles.resultItem}
                        onPress={() => goToPlace(item)}
                    >
                        <Text
style={styles.resultText}>{item.name}</Text>
                    </TouchableOpacity>
                )}
            />
            <TouchableOpacity
                style={styles.closeButton}>
```

```
        onPress={() => setIsModalVisible(false)}
      >
      <Text style={styles.closeButtonText}>Close</Text>
    </TouchableOpacity>
  </View>
</View>
</Modal>

ToggleMarkersButton: {
  position: 'absolute',
  bottom: scaleHeight(5),
  left: scaleWidth(5),
  backgroundColor: '#38b6ff',
  padding: scaleHeight(3),
  borderRadius: scaleWidth(10),
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.25,
  shadowRadius: 3.84,
  elevation: 5,
  borderColor: 'black',
  borderWidth: 2,
},
ToggleMarkersButtonText: {
  fontSize: scaleHeight(14),
  fontWeight: 'bold',
  color: '#000',
},
SearchLocation: {
  height: scaleHeight(60),
  borderColor: 'black',
  borderWidth: 3,
  borderRadius: scaleWidth(10),
  justifyContent: 'center',
  alignItems: 'center',
  marginVertical: scaleHeight(8),
},
SearchLabel: {
```

```
    fontSize: scaleHeight(16),
    fontWeight: 'bold',
  },
  modalContainer: {
    flex: 1,
    backgroundColor: 'rgba(0,0,0,0.5)',
    justifyContent: 'center',
    alignItems: 'center',
  },
  modalContent: {
    width: '80%',
    backgroundColor: '#38b6ff',
    borderRadius: scaleWidth(10),
    padding: scaleWidth(16),
    borderWidth: 3,
    borderColor: 'black'
  },
  searchInput: {
    borderBottomWidth: 1,
    marginBottom: scaleHeight(10),
    fontSize: scaleHeight(16),
  },
  resultItem: {
    padding: scaleHeight(10),
    borderBottomWidth: 1,
  },
  resultText: {
    fontSize: scaleHeight(16),
  },
  closeButton: {
    marginTop: scaleHeight(10),
    alignSelf: 'center',
  },
  closeButtonText: {
    color: 'black',
    fontWeight: 'bold',
    fontSize: scaleHeight(16),
  }
}
```

Description :	<p>I am going to assign each room on this SchoolMap to a specific coordinate on the map, then route using GoogleMaps, using this image:</p> <p>I went around to each classroom and measured the precise coordinates of each room.</p> <p>I also used these websites to assist me with the precise coordinates:</p> <p>https://www.maps.ie/coordinates.html#google vignette</p> <p>https://www.gps-coordinates.net/</p> <p>Any rooms on a floor above ground level I commented out, will find a clear solution for this in later iterations.</p> <p>Initially I decided to hardcode all the locations but this would be time-consuming so I abstracted the project to the first 25 Rooms and all major locations. These locations also include classrooms that are on upper floors, to demonstrate the method of coding these routes. However these places are coded for now.</p> <p>Some numbers are missing because that room doesn't exist e.g Rm 22 or 31.</p> <p>I removed the parent tag KeyboardAvoidingView since it interfered when I was opening the ModalContainer.</p> <p>To implement the search location feature and allow marking of different places on the map, I began by creating an array containing the names and coordinates (latitude and longitude) of various locations within the school. This array serves as the data source for both displaying map markers and filtering results when users search for specific locations. Using the <code>useEffect</code> hook, I requested foreground location permissions and retrieved the user's current position with high accuracy through <code>Location.requestForegroundPermissionsAsync</code> and <code>Location.getCurrentPositionAsync</code>. This ensured that the application could display the user's real-time location with precision, which is especially critical for navigation within a small, defined area like a school. This is vital as it meets: SC.02</p> <p>The map should be accurate, easy to understand and reflective of the building.</p> <p>To enhance my map application, I implemented a toggleable marker feature using React Native's <code>Marker</code> component, dynamically</p>
---------------	---

rendering markers based on user interaction. Using `places.map()`, I iterated through an array of locations, rendering each marker with precise latitude and longitude coordinates and displaying a title on click for clarity. Each marker was assigned a unique key for efficient rendering. To manage marker visibility, I introduced a `<TouchableOpacity>` button with an `onPress` handler that toggles the `showMarkers` state, dynamically updating the button text between "Show Markers" and "Hide Markers." This intuitive functionality reduces map clutter and allows users to focus on specific areas. Iterative improvements included addressing initial feedback by introducing the toggle and refining button styling for visibility and usability. I ensured descriptive naming, such as `showMarkers`, for clarity and added comments to aid future maintenance. Validation involved testing marker visibility, title accuracy, and responsiveness across screen sizes, ensuring a user-friendly and adaptable solution.

To visualize these locations interactively, I integrated the `react-native-maps` library into the project. The `MapView` component was configured to display the school area effectively by setting appropriate values for `latitudeDelta` and `longitudeDelta` to control the zoom level. For dynamic map interactions, I used a `mapRef` to implement an `animateToRegion` function, enabling smooth transitions as the map centers on a selected location. This functionality enhances user experience by making it easier to navigate to the desired spot within the map.

For the search feature, I designed a modal interface where users could type in a location name. A `TextInput` field captures the user's query, and a filtered list of matching locations is displayed using a `FlatList` component. Upon selecting a location, the map automatically zooms into the corresponding coordinates using the `animateToRegion` function, and the modal closes to allow users to view the selected location seamlessly. Additionally, a close button within the modal provides an option for users to exit the search interface if needed, improving usability.

Throughout the development process, I applied scalable styling techniques to ensure that all interface elements, including the search bar, modal, and list, adapt effectively to different screen sizes. By testing the feature incrementally and consulting library documentation, I validated each component to align with the application's overall goals and the predefined success criteria. This modular and structured approach allowed me to create a functional and intuitive search and navigation feature tailored for a school environment.

Test Table 22 – Implementing the Search Feature and Map of School

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
2.23	Users can view the map interface with markers for all mapped classroom locations.	Open the app and visually check the map. Ensure all predefined locations have a marker displayed.	Markers for all classrooms are visible on the map in their correct positions.	Markers are visible, but one marker (Room 23) appears slightly offset, and is off grounds.	No HyperLink to Solution
2.24	The Search Location button is visible and functional.	Launch the app, check for the search button at the bottom of the interface. Input a search query.	The search bar is visible. Entering text displays a dropdown list of matching locations based on the input.	Search bar is visible and functional, showing correct results for most queries. An error message is being outputted on the main screen. However, one classroom ("IT Tech") was not shown.	No HyperLink to Solution
2.25	Modal displays a list of locations matching the search query.	Input part of a classroom name in the search bar (e.g., "Toilet"). Verify if a list of matching locations appears.	A filtered list of classroom names matching the input query is displayed within the modal.	Modal displays filtered results correctly. No issues observed.	Yes, the modal is displayed correctly.

2.26	Selecting a location from the search results zooms in on the selected location and after a short period of time, returns to the current location.	Tap on a location name in the search result list. Check if the map centers and zooms into the corresponding marker for that location.	Map smoothly transitions to the selected location and centers on it, for 2 seconds, then back to the current location of the user.	Selecting a location works as expected, and zooms into the location.	Yes, selecting a location from the search location works as expected.
2.27	Users can dismiss the search modal if no selection is needed.	Open the search modal and press the close button or outside the modal area.	The search modal closes, and the user returns to the map interface.	Modal closes correctly using both methods.	Yes, users can escape the modal.

Evidence Table 22

Test ID	Screenshot/Video
2.23	https://youtube.com/shorts/KQK9rnMLWsc?feature=share
2.24	https://youtube.com/shorts/uVzFJ0MBPmQ?feature=share https://youtube.com/shorts/YJdknYfaPX0
2.25-2.27	
Evaluation:	In this first video, we can see the Rooms/Places all being accurately plotted on the map however it seems as though for Rm23 the location is offset as seen in the video. The rest of the Places being tested can be found below. Test ID 2.24 is not being met as we see in the second video how there is an error message coming up due to the search modal and how there is a bug in the code in regards to it. The error message shows “Encountered two children with the same key” and I will solve this bug/problem in the next sections. On top of this we can

see how IT Tech which is a place in the real life school map, which is meant to be mapped in is not in the Search Modal or Map.

In the third video we see how Test IDs 2.25 up to 2.27 are being met. The modal closes with the “close” button and opens up a list which your text input filters the results shown to be relevant to what your typing and the text input works for both capital upper case characters and lower case, emphasising the robustness of this module.

Fixing Error Message for Key and Missing “IT Tech” and Rm23 offset: Test ID 2.23 and 2.24

Relevant Source Code:

From:

```
const places = [
  { id: 1, name: 'Main Hall', latitude: 51.56853759709549,
longitude: 0.08591651916503908 },
  { id: 2, name: 'Library', latitude: 51.56863096032261, longitude:
0.08657097816467287 },
  { id: 3, name: 'Gym', latitude: 51.56893105511125, longitude:
0.08645296096801759},
  { id: 4, name: 'Dining Hall', latitude: 51.56845423690923 ,
longitude: 0.08490264415740968 },
  { id: 5, name: 'Main Reception', latitude: 51.56825083741333 ,
longitude: 0.08591651916503908 },
  { id: 6, name: 'Astro Turf', latitude: 51.56926115709083,
longitude: 0.08649051189422609 },
  { id: 7, name: 'Sports Hall', latitude: 51.568704317009434,
longitude: 0.08694648742675783},
  { id: 8, name: '6th Form Common Room', latitude:
51.56905109247207, longitude: 0.08697867393493652 },
  { id: 9, name: 'Boys Toilet', latitude: 51.5687476640869,
longitude: 0.08532643318176271},
  { id: 10, name: 'Girls Toilet', latitude: 51.5687509984758,
longitude: 0.08657097816467287 },
  { id: 11, name: 'Hub', latitude: 51.56813413237369,
longitude: 0.08723616600036622},
  { id: 12, name: 'Lecture Theatre', latitude: 51.56904091917293,
longitude: 0.08667417918180487},
  { id: 13, name: 'Margaret Evans Centre', latitude:
51.56799613868321, longitude: 0.08479976208794593},
```

```
{ id: 14, name: 'Staff Room', latitude:51.56857966263135 , longitude: 0.08640148787738777}, { id: 14, name: '', latitude: 51.56870136811619, longitude:0.08634069124885624}, { id: 16, name: 'Art Tech', latitude: 51.56870636967905, longitude:0.08564644612155004}, { id: 17, name: 'DT Tech', latitude: 51.56885641704665, longitude:0.08710869704221746}, { id: 18, name: 'SCI Staff', latitude: 51.56896422857994, longitude:0.08610331583289277}, { id: 19, name: '6th Form Private Study', latitude: 51.568921993176176, longitude:0.0870586292497455}, { id: 20, name: 'Rm 1', latitude: 51.568351260168114, longitude:0.08608958446001402}, { id: 21, name: 'Rm 2', latitude: 51.568356084551695, longitude:0.08615777483627962}, { id: 22, name: 'Rm 3', latitude: 51.56835263854545, longitude:0.08623483535680432}, { id: 23, name: 'Rm 4', latitude: 51.568347814161484, longitude:0.08632741893440876}, { id: 24, name: 'Rm 5', latitude: 51.568358841306, longitude:0.08646934327336986}, { id: 25, name: 'Rm 6', latitude: 51.56839536885297, longitude: 0.08656081813365635}, { id: 26, name: 'Rm 7', latitude: 51.56846911301434, longitude: 0.0865597094165782}, { id: 27, name: 'Rm 8', latitude: 51.56870137193664, longitude:0.08606907217542314}, { id: 28, name: 'Rm 10', latitude: 51.56869930432338, longitude:0.08578245129843065}, { id: 29, name: 'Rm 11', latitude: 51.568712398995395, longitude:0.08550691820333078}, { id: 30, name: 'Rm 12', latitude: 51.56866070940558, longitude:0.08528904206124288}, { id: 31, name: 'Rm 13', latitude: 51.56861039818962, longitude:0.08527518228315767}, { id: 32, name: 'Rm 14', latitude: 51.568575249185976, longitude:0.08527241028696508}, { id: 33, name: 'Rm 15', latitude: 51.568493234801586,
```

```
longitude:0.08527407356591965},
  { id: 34, name: 'Rm 16', latitude: 51.56836366559432,
longitude:0.08527739992083117},
  { id: 35, name: 'Rm 17', latitude: 51.56836021958865,
longitude:0.08537441857054517},
  { id: 36, name: 'Rm 18', latitude: 51.56836228715423,
longitude:0.0855734455037993},
  { id: 37, name: 'Rm 19', latitude: 51.56844154496262,
longitude:0.08540435576434913},
  { id: 38, name: 'Rm 20', latitude: 51.568522870191075,
longitude:0.08540768211926064},
  { id: 39, name: 'Rm 21', latitude: 51.56859454655905,
longitude:0.08542209625602482},
  { id: 41, name: 'Rm 23', latitude: 51.56850701862465,
longitude:0.08718860004323249},
  { id: 42, name: 'Rm 24', latitude: 51.56850770777942,
longitude:0.08683967778706503},
  { id: 43, name: 'Rm 25', latitude: 51.568336097446995,
longitude:0.08681916529923672},
  { id: 44, name: 'Rm 26', latitude: 51.56822927136887,
longitude:0.08682304601274726},
  { id: 45, name: 'Rm 27', latitude: 51.56826442060845,
longitude:0.08698437432521633},
  { id: 46, name: 'Rm 28', latitude: 51.568299569631534,
longitude:0.08696829711257514},
  { id: 47, name: 'Rm 29', latitude: 51.56834298920933,
longitude:0.08696885147121414},
  { id: 48, name: 'Rm 30', latitude: 51.568522180657965, longitude:
0.08696774275393615},
  // { id: 50, name: 'Rm 32', latitude: , longitude:},
  // { id: 51, name: 'Rm 33', latitude: , longitude:},
  // { id: 52, name: 'Rm 34', latitude: , longitude:},
  // { id: 53, name: 'Rm 35', latitude: , longitude:},
  // { id: 54, name: 'Rm 36', latitude: , longitude:},
  // { id: 55, name: 'Rm 37', latitude: , longitude:},
  // { id: 56, name: 'Rm 38', latitude: , longitude:},
];

];
```

To:

```
const places = [
    { id: 1, name: 'Main Hall', latitude: 51.56853759709549,
longitude: 0.08591651916503908 },
    { id: 2, name: 'Library', latitude: 51.56863096032261, longitude:
0.08657097816467287 },
    { id: 3, name: 'Gym', latitude: 51.56893105511125, longitude:
0.08645296096801759},
    { id: 4, name: 'Dining Hall', latitude: 51.56845423690923 ,
longitude: 0.08490264415740968 },
    { id: 5, name: 'Main Reception', latitude: 51.56825083741333 ,
longitude: 0.08591651916503908 },
    { id: 6, name: 'Astro Turf', latitude: 51.56926115709083,
longitude: 0.08649051189422609 },
    { id: 7, name: 'Sports Hall', latitude: 51.568704317009434,
longitude: 0.08694648742675783},
    { id: 8, name: '6th Form Common Room', latitude:
51.56905109247207, longitude: 0.08697867393493652 },
    { id: 9, name: 'Boys Toilet', latitude: 51.5687476640869,
longitude: 0.08532643318176271},
    { id: 10, name: 'Girls Toilet', latitude: 51.5687509984758,
longitude: 0.08657097816467287 },
    { id: 11, name: 'Hub', latitude: 51.56813413237369,
longitude: 0.08723616600036622},
    { id: 12, name: 'Lecture Theatre', latitude: 51.56904091917293,
longitude: 0.08667417918180487},
    { id: 13, name: 'Margaret Evans Centre', latitude:
51.56799613868321, longitude: 0.08479976208794593},
    { id: 14, name: 'Staff Room', latitude: 51.56857966263135 ,
longitude: 0.08640148787738777},
    { id: 15, name: 'IT Tech', latitude: 51.56870136811619,
longitude: 0.08634069124885624},
    { id: 16, name: 'Art Tech', latitude: 51.56870636967905,
longitude: 0.08564644612155004},
    { id: 17, name: 'DT Tech', latitude: 51.56885641704665,
longitude: 0.08710869704221746},
    { id: 18, name: 'SCI Staff', latitude: 51.56896422857994,
longitude: 0.08610331583289277},
    { id: 19, name: '6th Form Private Study', latitude:
```

```
51.568921993176176, longitude:0.0870586292497455},  
    { id: 20, name: 'Rm 1', latitude: 51.568351260168114,  
longitude:0.08608958446001402},  
    { id: 21, name: 'Rm 2', latitude: 51.568356084551695,  
longitude:0.08615777483627962},  
    { id: 22, name: 'Rm 3', latitude: 51.56835263854545,  
longitude:0.08623483535680432},  
    { id: 23, name: 'Rm 4', latitude: 51.568347814161484,  
longitude:0.08632741893440876},  
    { id: 24, name: 'Rm 5', latitude: 51.568358841306,  
longitude:0.08646934327336986},  
    { id: 25, name: 'Rm 6', latitude: 51.56839536885297, longitude:  
0.08656081813365635},  
    { id: 26, name: 'Rm 7', latitude: 51.56846911301434, longitude:  
0.0865597094165782},  
    { id: 27, name: 'Rm 8', latitude: 51.56870137193664,  
longitude:0.08606907217542314},  
    { id: 28, name: 'Rm 10', latitude: 51.56869930432338,  
longitude:0.08578245129843065},  
    { id: 29, name: 'Rm 11', latitude: 51.568712398995395,  
longitude:0.08550691820333078},  
    { id: 30, name: 'Rm 12', latitude: 51.56866070940558,  
longitude:0.08528904206124288},  
    { id: 31, name: 'Rm 13', latitude: 51.56861039818962,  
longitude:0.08527518228315767},  
    { id: 32, name: 'Rm 14', latitude: 51.568575249185976,  
longitude:0.08527241028696508},  
    { id: 33, name: 'Rm 15', latitude: 51.568493234801586,  
longitude:0.08527407356591965},  
    { id: 34, name: 'Rm 16', latitude: 51.56836366559432,  
longitude:0.08527739992083117},  
    { id: 35, name: 'Rm 17', latitude: 51.56836021958865,  
longitude:0.08537441857054517},  
    { id: 36, name: 'Rm 18', latitude: 51.56836228715423,  
longitude:0.0855734455037993},  
    { id: 37, name: 'Rm 19', latitude: 51.56844154496262,  
longitude:0.08540435576434913},  
    { id: 38, name: 'Rm 20', latitude: 51.568522870191075,  
longitude:0.08540768211926064},
```

	<pre> { id: 39, name: 'Rm 21', latitude: 51.56859454655905, longitude:0.08542209625602482}, { id: 41, name: 'Rm 23', latitude: 51.56850701862465, longitude:0.08708860004323249}, { id: 42, name: 'Rm 24', latitude: 51.56850770777942, longitude:0.08683967778706503}, { id: 43, name: 'Rm 25', latitude: 51.568336097446995, longitude:0.08681916529923672}, { id: 44, name: 'Rm 26', latitude: 51.56822927136887, longitude:0.08682304601274726}, { id: 45, name: 'Rm 27', latitude: 51.56826442060845, longitude:0.08698437432521633}, { id: 46, name: 'Rm 28', latitude: 51.568299569631534, longitude:0.08696829711257514}, { id: 47, name: 'Rm 29', latitude: 51.56834298920933, longitude:0.08696885147121414}, { id: 48, name: 'Rm 30', latitude: 51.568522180657965, longitude: 0.08696774275393615}, // { id: 50, name: 'Rm 32', latitude: , longitude:}, // { id: 51, name: 'Rm 33', latitude: , longitude:}, // { id: 52, name: 'Rm 34', latitude: , longitude:}, // { id: 53, name: 'Rm 35', latitude: , longitude:}, // { id: 54, name: 'Rm 36', latitude: , longitude:}, // { id: 55, name: 'Rm 37', latitude: , longitude:}, // { id: 56, name: 'Rm 38', latitude: , longitude:},]; </pre>
Description:	<p>There seems to be a precision error within the original code prohibiting the Room 23 location to be plotted in the correction location, hence I readjusted the coordinates by using the website mentioned earlier.</p> <p>In regards to the collision with keys, in writing the code I repeated the IDs 14 twice, hence the reason for the error message for key 14 and in place of "IT Tech" was null, leading to not being able to find the IT Tech location in the Search Modal or the Map.</p>

Test Table 23 – Fixing errors in Test ID 2.23, 2.24

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
2.23 -FIX	Users can view the map interface with markers for all mapped classroom locations.	Open the app and visually check the map. Ensure all predefined locations have a marker displayed.	Markers for all classrooms are visible on the map in their correct positions.	ALL markers including Rm23 are on the Map and can be identified using the Search Modal.	Yes – fixed from previous error
2.24 -FIX	The Search Location button is visible and functional.	Launch the app, check for the search button at the bottom of the interface. Input a search query.	The search bar is visible. Entering text displays a dropdown list of matching locations based on the input.	There are no error messages within the Search Modal and the components and the program runs as expected.	Yes – fixed from previous error

Evidence Table 23

Test ID	Screenshot/Video	Evaluation
2.23 - 2.24 FIX	https://youtube.com/shorts/FNo9 UKaZ50	<p>This video showcases how the previous error of collisions in the places array has been eradicated, allowing the smooth use of the application and how a key function like Search Location does not have any errors now and works successfully.</p> <p>Rm 23 is now in its rightful place and not in an offset location, this preserves the integrity and quality of the application.</p>

Version 7 – Routing Users to Selected Locations + Corridors

This iteration primarily focuses on our goal to add the **corridors** which will act as edges for the **routing algorithm** and then using the routing algorithm to find shortest paths from and to different places. Then I need to aesthetically output the route when the user presses to go there! To meet elements from the **success criteria and the test plans** in the Designs section the line should be vivid and visible for all users for the corridors/route. Wherever you are it will choose the closest node! Then route from there. To meet the routing element of my application I chose to use **A* Algorithm** after performing research and analysis with stakeholders and past experience of other developers, developing a mapping application. The function to route should appear when users select a marker(Classroom) on the map and when they search for a location using the Search Location button (coded in previous iterations). To implement the **A* Algorithm** to find a path, I need to add a **graph structure** with nodes and edges. As an artistic flair when users route to a place using Search Location I wanted the map to pan to the destination then to the closest node following a set route after a set period of time. Meeting the majority of **SC.02 – The map should be accurate, easy to understand and reflective of the building.**

Routing Users from Closest Point on Edge to Node

Relevant Source Code:

```
const nodes = [
  {id:0},
  { id: 1, latitude: 51.56816849386687, longitude: 0.0868893930876613 },
  { id: 2, latitude: 51.568546220862, longitude: 0.08690347468498949 },
  { id: 3, latitude: 51.568366371103366, longitude: 0.0868981102669597 },
  { id: 4, latitude: 51.56849099476613, longitude: 0.08689676916245226 },
  { id: 5, latitude: 51.56836064007387, longitude: 0.0867298016512752 },
  { id: 6, latitude: 51.56836574590019, longitude: 0.08657088076714281 },
  { id: 7, latitude: 51.568554244270274, longitude: 0.08657021021488909 },
  { id: 8, latitude: 51.56855601567193, longitude: 0.08668487465027575 },
  { id: 9, latitude: 51.56869366378912, longitude: 0.08649913167599443 },
  { id: 10, latitude: 51.568753891221824, longitude: }
```

```
0.08650583719853167},
  { id: 11, latitude: 51.56869814437943, longitude:
0.08592715060356859},
  { id: 12, latitude: 51.568704083765894, longitude:
0.08540076708439592},
  { id: 13, latitude: 51.568675116224554, longitude:
0.08528006767872576},
  { id: 14, latitude: 51.56835178302684, longitude:
0.08527000939491991},
  { id: 15, latitude: 51.568335840337895, longitude:
0.08592245673779253},
  { id: 16, latitude: 51.56834198817285, longitude:
0.08510371243599657},
  { id: 17, latitude: 51.56850891720453, longitude:
0.08512382900360826},
  { id: 18, latitude: 51.568749931635246, longitude:
0.08538065051678423},
  { id: 19, latitude: 51.5682298682045, longitude:
0.08592312729004625},
  { id: 20, latitude: 51.568813701776705, longitude:
0.08638111447933916},
  { id: 21, latitude: 51.56892279852696, longitude:
0.08638178503159288},
  { id: 22, latitude: 51.568929675677246, longitude:
0.08615245616081957},
  { id: 23, latitude: 51.56900428227089, longitude:
0.0863737384045482},
  { id: 24, latitude: 51.569203927246136, longitude:
0.08627181446198229},
  { id: 25, latitude: 51.56901313919077, longitude:
0.08681160902622942},
  { id: 26, latitude: 51.56906753106095, longitude:
0.08681831454876665},
  { id: 27, latitude: 51.56905929934531, longitude:
0.08696784770134691},
  { id: 28, latitude: 51.56865854844982, longitude:
0.0865762451851726},
  { id: 29, latitude: 51.56852548503379, longitude:
0.08702350353840593},
```

```
];
const edges = [
    { from: 1, to: 3, weight: calculateDistance(nodes[1], nodes[3]) },
    { from: 3, to: 4, weight: calculateDistance(nodes[3], nodes[4]) },
    { from: 4, to: 2, weight: calculateDistance(nodes[4], nodes[2]) },
    { from: 3, to: 5, weight: calculateDistance(nodes[3], nodes[5]) },
    { from: 5, to: 6, weight: calculateDistance(nodes[5], nodes[6]) },
    { from: 6, to: 7, weight: calculateDistance(nodes[6], nodes[7]) },
    { from: 7, to: 8, weight: calculateDistance(nodes[7], nodes[8]) },
    { from: 7, to: 28, weight: calculateDistance(nodes[7], nodes[28]) }
],
    { from: 9, to: 10, weight: calculateDistance(nodes[9], nodes[10]) }
},
    { from: 9, to: 11, weight: calculateDistance(nodes[9], nodes[11]) }
},
    { from: 11, to: 12, weight: calculateDistance(nodes[11], nodes[12]) }
},
    { from: 11, to: 15, weight: calculateDistance(nodes[11], nodes[15]) }
},
    { from: 12, to: 18, weight: calculateDistance(nodes[12], nodes[18]) }
},
    { from: 12, to: 13, weight: calculateDistance(nodes[12], nodes[13]) }
},
    { from: 13, to: 14, weight: calculateDistance(nodes[13], nodes[14]) }
},
    { from: 14, to: 16, weight: calculateDistance(nodes[14], nodes[16]) }
},
    { from: 16, to: 17, weight: calculateDistance(nodes[16], nodes[17]) }
},
    { from: 14, to: 15, weight: calculateDistance(nodes[14], nodes[15]) }
},
    { from: 15, to: 19, weight: calculateDistance(nodes[15], nodes[19]) }
},
    { from: 15, to: 6, weight: calculateDistance(nodes[15], nodes[6]) }
},
    { from: 10, to: 20, weight: calculateDistance(nodes[10], nodes[20]) }
},
    { from: 20, to: 21, weight: calculateDistance(nodes[20], nodes[21]) }
},
```

```

        { from: 21, to: 22, weight: calculateDistance(nodes[21], nodes[22]) },
        { from: 21, to: 23, weight: calculateDistance(nodes[21], nodes[23]) },
        { from: 23, to: 24, weight: calculateDistance(nodes[23], nodes[24]) },
        { from: 23, to: 25, weight: calculateDistance(nodes[23], nodes[25]) },
        { from: 25, to: 26, weight: calculateDistance(nodes[25], nodes[26]) },
        { from: 26, to: 27, weight: calculateDistance(nodes[26], nodes[27]) },
        { from: 28, to: 9, weight: calculateDistance(nodes[28], nodes[9]) },
        { from: 29, to: 2, weight: calculateDistance(nodes[29], nodes[2]) },
        { from: 8, to: 2, weight: calculateDistance(nodes[8], nodes[2]) },
    ];
}

function calculateDistance(node1, node2) {
    const R = 6371; // Earth's radius in km
    const dLat = (node2.latitude - node1.latitude) * (Math.PI / 180);
    const dLon = (node2.longitude - node1.longitude) * (Math.PI / 180);
    const a =
        Math.sin(dLat / 2) * Math.sin(dLat / 2) +
        Math.cos(node1.latitude * (Math.PI / 180)) *
        Math.cos(node2.latitude * (Math.PI / 180)) *
        Math.sin(dLon / 2) *
        Math.sin(dLon / 2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    return R * c;
}

// A* Algorithm to find the shortest path
function findPath(startId, endId) {
    const openSet = [startId]; // Nodes to be evaluated
    const cameFrom = {}; // Map to store the best path
    const gScore = {}; // Cost from start to a node
    const fScore = {}; // Total cost (gScore + heuristic)
}

```

```

    nodes.forEach((node) => {
      gScore[node.id] = Infinity;
      fScore[node.id] = Infinity;
    });
    gScore[startId] = 0;
    fScore[startId] = heuristic(startId, endId);

    while (openSet.length > 0) {
      // Get node with the lowest fScore
      const current = openSet.sort((a, b) => fScore[a] - fScore[b])[0];
      if (current === endId) break; // Found the target

      // Remove current from openSet
      openSet.splice(openSet.indexOf(current), 1);

      // Get neighbors of the current node
      edges
        .filter((edge) => edge.from === current || edge.to === current)
        .forEach((edge) => {
          const neighbor = edge.from === current ? edge.to : edge.from;
          const tentativeGScore = gScore[current] + edge.weight;

          if (tentativeGScore < gScore[neighbor]) {
            cameFrom[neighbor] = current;
            gScore[neighbor] = tentativeGScore;
            fScore[neighbor] = gScore[neighbor] + heuristic(neighbor, endId);

            if (!openSet.includes(neighbor)) openSet.push(neighbor);
          }
        });
    }

    // Reconstruct path
    const path = [];
    let current = endId;
    while (current) {
      path.unshift(current);
      current = cameFrom[current];
    }
  }
}

```

```

    }
    return path;
}
// Heuristic function (straight-line distance to end node)
function heuristic(nodeId, endId) {
  const node = nodes.find((n) => n.id === nodeId);
  const endNode = nodes.find((n) => n.id === endId);
  return calculateDistance(node, endNode);
}

const handleCalculateRoute = () => {
  const lastNode = nodes.length - 1; // Last node
  const secondLastNode = nodes.length - 2; // Second-last node
  const path = findPath(lastNode, secondLastNode); // Example: from
node 1 to node 4
  const coordinates = path.map((id) => {
    const node = nodes.find((n) => n.id === id);
    return { latitude: node.latitude, longitude: node.longitude };
  });
  setPathCoordinates(coordinates);
};

function findClosestNode(userLocation, nodes) {
  if (!userLocation || !nodes || nodes.length === 0) {
    throw new Error("Invalid userLocation or nodes array.");
  }

  let closestNode = null;
  let shortestDistance = Infinity;

  nodes.forEach((node) => {
    const distance = calculateDistance(
      userLocation.latitude,
      userLocation.longitude,
      node.latitude,
      node.longitude
    );
    if (distance < shortestDistance) {
      shortestDistance = distance;
      closestNode = node;
    }
  });
  return closestNode;
}

```

```

        }
    });
    return closestNode;
}

/* Render polylines for edges */
edges.map((edge, index) => {
    const fromNode = nodes.find((node) => node.id ===
edge.from);
    const toNode = nodes.find((node) => node.id ===
edge.to);

    // Ensure both nodes exist before rendering the
polyline
    if (!fromNode || !toNode) return null;

    return (
        <Polyline
            key={index}
            coordinates={[
                { latitude: fromNode.latitude, longitude:
fromNode.longitude },
                { latitude: toNode.latitude, longitude:
toNode.longitude },
            ]}
            strokeColor="blue" // Color of the line
            strokeWidth={7} // Thickness of the line
        />
    );
})}

{pathCoordinates.length > 0 && (
<Polyline
    coordinates={pathCoordinates} // Array of
latitude/longitude points
    strokeColor="yellow" // A nice yellow color
    strokeWidth={12} // Thicker line for visibility
    lineDashPattern={[8, 6]} // Dashed line: 4px dash, 2px
gap
/>
)
}

```

```

        )}

<TouchableOpacity
    style={styles.resultItem}
    onPress={() => {
        goToPlace(item);

        setTimeout(() => {
            handleCalculateRoute();
        }, 2500);
    }}
>

```

Description:

To implement the A* Algorithm, as we discussed at the preface of this iteration we needed a graph structure. In addition to the places array we added before, I added a nodes array containing the turning points of each corridor(excluding classrooms due to the places array being previously coded).

```

const nodes = [
    {id:0},
    { id: 1, latitude: 51.56816849386687, longitude: 0.0868893930876613 },
    { id: 2, latitude: 51.568546220862, longitude: 0.08690347468498949 },
]

```

The structure is of Id, then latitude and longitude, which can be later used to manipulate routes and calculations and even be plotted on the map.

To complement the nodes array I added an edges array which links with the nodes array so that the connections can be graphed and visualised.

```

const edges = [
    { from: 1, to: 3, weight: calculateDistance(nodes[1], nodes[3]) },
    { from: 3, to: 4, weight: calculateDistance(nodes[3], nodes[4]) },
]

```

From and to represents a bidirectional edge and then the weight is calculated using a function which can be seen below (using the longitude, latitude details of the nodes connecting the edges).

To calculate the weight, which will be necessary for algorithms like A*I use a function which utilises the Haversine formula which takes the radius of the Earth and performs mathematical calculations using the longitude and latitude to provide the Euclidean distance between

two nodes.

I used this website to assist me with the code for this:

<https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128>

Which has Python code which I used in conjunction with my understanding of the Haversine formula to produce this:

```
function calculateDistance(node1, node2) {  
    const R = 6371; // Earth's radius in km  
    const dLat = (node2.latitude - node1.latitude) * (Math.PI / 180);  
    const dLon = (node2.longitude - node1.longitude) * (Math.PI / 180);  
    const a =  
        Math.sin(dLat / 2) * Math.sin(dLat / 2) +  
        Math.cos(node1.latitude * (Math.PI / 180)) *  
        Math.cos(node2.latitude * (Math.PI / 180)) *  
        Math.sin(dLon / 2) *  
        Math.sin(dLon / 2);  
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));  
    return R * c;
```

Where $R \cdot c$ the output is the distance/weight.

The actual A* Algorithm is presented here below and takes in inputs **startID** and **endID** referring to the ID attribute we discussed and assigned earlier to the nodes array. Overall this first part of the algorithm finds the shortest path between two nodes in a graph. **openSet** is an array that starts with the starting node **startID** and **cameFrom** maps each node to its previous one for reconstructing the path later; it's similar to backtracking on a depth-first traversal. **gScore** holds the cost from the start node to each node, this starts at **infinity** for all nodes except the starting node (0). **fScore** combines **gScore** with a heuristic estimate to the goal. A* is all about finding the shortest path with a heuristic, in this case **gScore** and **fScore**, this implementation of the A* Algorithm is a progression towards the completion of SC.21 – The route should take the shortest distance and be accurate and precise. If there are nodes in **openSet** the algorithm picks the nodes with the smallest **fScore**. This is sorted using **fScore[a] - fScore[b]**. If the current node matches the target node (**endID**) the loop stops.

The current node is then removed from **openSet** to let the algorithm examine its neighbours (nodes connected by edges). For each neighbour it calculates a tentative **gScore** by adding the current node's **gScore** to the weight of the edge connecting them. If this new

`gScore` is better, the neighbour is updated, `camFrom` stores the current node as the path to this neighbour and `gScore` and `fScore` for the neighbour are updated. The neighbor is added to `openSet` if not already in it.

Then there is the path reconstruction, where once the end node is reached, the path is built by tracing back through `cameFrom` as we discussed earlier.

The approach is efficient and uses the heuristic to guide the search, which helps reduce unnecessary checks and speeds up finding the optimal path.

```
function findPath(startId, endId) {  
    const openSet = [startId]; // Nodes to be evaluated  
    const cameFrom = {}; // Map to store the best path  
    const gScore = {}; // Cost from start to a node  
    const fScore = {}; // Total cost (gScore + heuristic)  
  
    nodes.forEach((node) => {  
        gScore[node.id] = Infinity;  
        fScore[node.id] = Infinity;  
    });  
    gScore[startId] = 0;  
    fScore[startId] = heuristic(startId, endId);  
  
    while (openSet.length > 0) {  
        // Get node with the lowest fScore  
        const current = openSet.sort((a, b) => fScore[a] - fScore[b])[0];  
        if (current === endId) break; // Found the target  
  
        // Remove current from openSet  
        openSet.splice(openSet.indexOf(current), 1);  
  
        // Get neighbors of the current node  
        edges  
            .filter((edge) => edge.from === current || edge.to === current)  
            .forEach((edge) => {  
                const neighbor = edge.from === current ? edge.to : edge.from;  
                const tentativeGScore = gScore[current] + edge.weight;  
  
                if (tentativeGScore < gScore[neighbor]) {  
                    gScore[neighbor] = tentativeGScore;  
                    fScore[neighbor] = gScore[neighbor] + heuristic(neighbor, endId);  
                    cameFrom[neighbor] = current;  
                    openSet.push(neighbor);  
                }  
            });  
    }  
    return cameFrom;  
}
```

```

        cameFrom[neighbor] = current;
        gScore[neighbor] = tentativeGScore;
        fScore[neighbor] = gScore[neighbor] + heuristic(neighbor,
endId);

        if (!openSet.includes(neighbor)) openSet.push(neighbor);
    }
});

}

// Reconstruct path
const path = [];
let current = endId;
while (current) {
    path.unshift(current);
    current = cameFrom[current];
}
return path;
}

// Heuristic function (straight-line distance to end node)
function heuristic(nodeId, endId) {
    const node = nodes.find((n) => n.id === nodeId);
    const endNode = nodes.find((n) => n.id === endId);
    return calculateDistance(node, endNode);
}

```

```

const handleCalculateRoute = () => {
    const lastNode = nodes.length - 1; // Last node
    const secondLastNode = nodes.length - 2; // Second-last node
    const path = findPath(lastNode, secondLastNode); // Example: from
node 1 to node 4
    const coordinates = path.map((id) => {
        const node = nodes.find((n) => n.id === id);
        return { latitude: node.latitude, longitude: node.longitude };
    });
    setPathCoordinates(coordinates);
};

function findClosestNode(userLocation, nodes) {
    if (!userLocation || !nodes || nodes.length === 0) {

```

```

        throw new Error("Invalid userLocation or nodes array.");
    }

    let closestNode = null;
    let shortestDistance = Infinity;

    nodes.forEach((node) => {
        const distance = calculateDistance(
            userLocation.latitude,
            userLocation.longitude,
            node.latitude,
            node.longitude
        );
        if (distance < shortestDistance) {
            shortestDistance = distance;
            closestNode = node;
        }
    });
    return closestNode;
}

```

I have chosen polylines to render the route and have set it up right now to show all the corridors/edges for testing and visualisation purposes, this will allow clear visualisation of the route, enabling the completion of SC.20 - The route should be clear and presented on the Map.

```

/* Render polylines for edges */
edges.map((edge, index) => {
    const fromNode = nodes.find((node) => node.id ===
edge.from);
    const toNode = nodes.find((node) => node.id ===
edge.to);

    // Ensure both nodes exist before rendering the
polyline
    if (!fromNode || !toNode) return null;

    return (
        <Polyline
            key={index}

```

```

        coordinates={[
          { latitude: fromNode.latitude, longitude:
fromNode.longitude },
          { latitude: toNode.latitude, longitude:
toNode.longitude },
        ]}
      strokeColor="blue" // Color of the line
      strokeWidth={7} // Thickness of the line
    />
);
})}

```

This code renders polylines on a map to represent edges between nodes. It starts by iterating over the `edges` array using `map()`, where each edge has a `from` node and a `to` node. For each edge, the code finds the corresponding nodes from the `nodes` array using `find()`, which locates the nodes based on their `id`. If either of the nodes doesn't exist (i.e., one or both nodes are not found), it returns `null` to avoid rendering the polyline. If both nodes are found, it proceeds to render a `<Polyline>` component, passing the coordinates of the `fromNode` and `toNode` as an array of objects with latitude and longitude. The polyline is styled with a blue color (`strokeColor="blue"`) and a line thickness of 7 pixels (`strokeWidth={7}`). The `key={index}` ensures that each polyline has a unique identifier for React's reconciliation process. This setup visualizes the connections between nodes as blue lines on the map.

```

{pathCoordinates.length > 0 && (
  <Polyline
    coordinates={pathCoordinates} // Array of
    latitude/longitude points
    strokeColor="yellow" // A nice yellow color
    strokeWidth={12} // Thicker line for visibility
    lineDashPattern={[8, 6]} // Dashed line: 4px dash, 2px
    gap
  />
)}

```

This code snippet conditionally renders a polyline on the map when `pathCoordinates` contains any points. It first checks if the `pathCoordinates` array has a length greater than 0, indicating that there is a valid path to display. If the condition is met, it renders a `<Polyline>` component with the following properties: the `coordinates` prop is set to `pathCoordinates`, which is an array of latitude and longitude points defining the path. The polyline's appearance is

	customized by setting the <code>strokeColor</code> to "yellow", giving the line a bright, noticeable color. The <code>strokeWidth</code> is set to 12, making the line thicker for better visibility. Additionally, the <code>lineDashPattern</code> property is used to create a dashed line, where the pattern [8, 6] means an 8-pixel dash followed by a 6-pixel gap, giving the line a dashed effect. The result is a visually distinctive, dashed yellow path drawn on the map when there are valid coordinates to display. I experimented with these settings to meet the test plans where this route line is clear and visible, so that this SC.20 - The route should be clear and presented on the Map, can be met.
--	---

Test Table 24 – Initialising Log-In Screen

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
2.28	Testing the route generation functionality	Select a marker (e.g., Classroom) on the map or use the "Search Location" button to input a destination.	When the user selects a marker or searches for a location, the application should calculate and display a route from the closest node to the destination using the A* Algorithms.	The item you search for when you press it, the route presented is not the route to that node, rather it is a route from ID 29 to ID 28 and presents that route for any classroom search.	No HyperLink to Solution
2.29	Ensuring the route line is vivid and visible	Trigger a route calculation by selecting a destination or marker.	The route line should be visible, vivid (e.g., yellow or other contrasting color), and thick enough to be easily seen by	The route line should be displayed in a vivid color and with sufficient thickness to be clearly visible to users.	Yes, you can vividly see the route line.

			users.		
2.30	Testing route update based on user movement	Move the map view or change location after the initial route is drawn (e.g., by panning or zooming on the map).	The route line should update dynamically to reflect the new path based on the user's movement, maintaining accuracy.	The route line does not update as the user moves around.	No HyperLink to Solution
2.31	Testing automatic panning aesthetically and smoothly to the destination and closest node	After selecting a location, ensure the map pans smoothly to the destination and the closest node within a set time (e.g., 3 seconds).	The map should smoothly pan to the destination and then to the closest node, following a clear route after a specified time.	The map does pan to the destination but fails to delay and then pan to the current node/startng node.	No HyperLink to Solution
2.32	Testing the calculation of the shortest path	Select different destination locations and check if the algorithm calculates and displays the shortest path.	The A* algorithm should calculate and render the shortest possible path from the current position or closest node to the destination.	The algorithm should correctly display the shortest path, calculating the optimal route based on the input data.	Yes - but needs to test once Test ID 2.28 is resolved.
2.33	Ensuring user feedback on path calculation status	After clicking the "Search Location" button or selecting a marker, verify	The app should indicate the status of the route calculation	Need to add, once the route works properly, and can properly route users	No - Need to add once Test ID 2.28 has been met.

		the app shows a loading indicator or similar status while calculating the route.	process (e.g., loading spinner, message).	from the closest node to a classroom.	
2.34	Button to Clear Route should be visible and prevalent and functional	Attempt pressing Clear Route with no route on screen, to detect errors. Search for a couple of routes and see if the route is being set onPress of the Clear Route button.	The Clear Route button should be obvious and present to the user and it should be on the map clearly. onPress it should remove any route lines.	There is no Clear Route button below the Recenter button!	No - Need to add preceding completion of Test ID 2.28.

Evidence Table 24

Test ID	Screenshot/Video
2.28 – 2.33	<p>Narrated Video:</p> <p>https://youtube.com/shorts/AQKvBbsqq6g</p>
Evaluation:	<p>This test evaluates the navigation system's functionality, specifically for test IDs 2.28 to 2.33. When accessing the system as a guest, the app successfully pans to the user's current location, and markers representing corridors and edges are visible. These edges, plotted using the polyline function, provide a visual representation of nodes and their connections, helping to verify placement accuracy. However, test ID 2.28, which focuses on route visibility, is not met in this instance since no route is displayed. The edges shown serve only for testing purposes rather than representing the intended route.</p> <p>For test ID 2.29, which assesses whether the route line is vivid and easily distinguishable, the system meets expectations. The route is displayed as a yellow dashed line, ensuring visibility even from a distance, making it clear for the user. The system also calculates the shortest path correctly,</p>

satisfying test ID 2.32. The algorithm selects the second-to-last nodes, such as node IDs 29 and 28, to determine the most efficient route. However, additional routes need testing to ensure consistent performance.

Test ID 2.30, which checks if the route updates dynamically based on user movement, is not met, as the route remains static rather than adjusting in real-time. Additionally, while the system successfully pans to a selected destination when searching for a location (e.g., "Sports Hall"), it does not pan to the user's closest node, meaning test ID 2.31 is not fully satisfied. Overall, while some test criteria are met—such as route visibility and shortest path calculation—others require further refinement to improve user interaction and real-time updates.

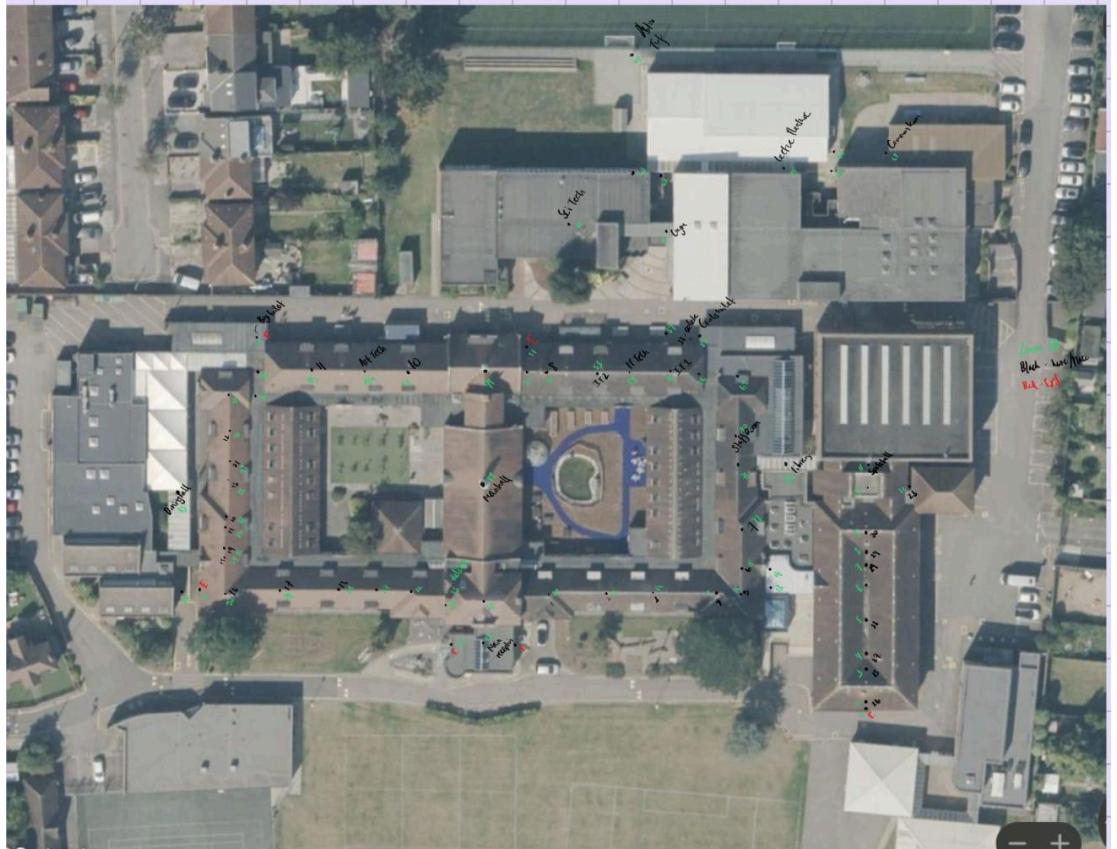
Fixing the route being fixed: Test ID 2.28, 2.30 and 2.31, implementing Test ID 2.33

Relevant Source Code:

This is where I define my graph and the places array, which is necessary for the goToPlace function which plays an important role in my code. I had to scrap the previous coordinates and start fresh, these are all new coordinates and I did it in a form to replicate the actual corridors and where the classrooms lie on the corridors in this way the A* Algorithm will successfully be able to run, to aid the successful completion of SC.21
- The route should take the shortest distance and be accurate and precise.

The graph is to replicate this:

Black is the name of the place, green is the ID for future reference. I used console.log() functions to test the IDs and locations, and used the same method for the previous section to calculate the coordinates.

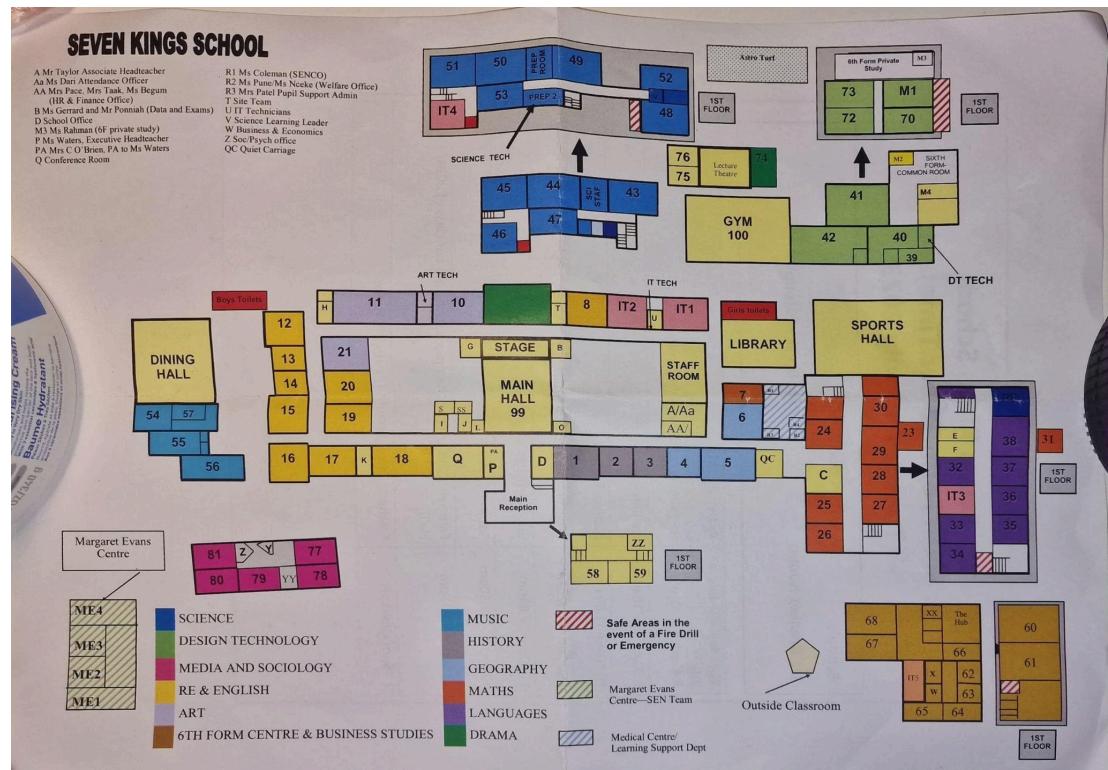


It may look a bit small, but there are annotations on this image, just for my reference when designing the graph.



This is a zoomed in image.

The map I used for reference is here: This was done to meet accurately the SC.02 - The map should be accurate, easy to understand and reflective of the building so I utilised the actual map alongside the satellite imagery to form the virtual map accurately.



```
// Hardcoded places

const places = [
{id: 2, name: 'Rm 26',latitude: 51.568172547493674,longitude:
0.0868942453397814},
{id: 3, name: 'Rm 25',latitude:
51.56823145571189,longitude:0.08689513960044692 },
{id: 4, name: 'Rm 27', latitude: 51.56826029831658,longitude:
0.08689678819390867},
{id: 5, name: 'Rm 28', latitude: 51.568304201545864,longitude:
0.08689678835761772},
{id: 7, name: 'Rm 29', latitude: 51.568384227566234,longitude:
0.08689857663338518},
{id: 8, name: 'Rm 24', latitude: 51.568413681519125,longitude:
0.08689678860318129},
{id: 9, name: 'Rm 30', latitude: 51.56844146824725,longitude:
0.08689768270013776},
{id: 11, name: 'Sports Hall',latitude: 51.568549280637356,longitude:
0.08690841169990637},
{id: 12, name: 'Rm 23',latitude: 51.56850037611065,longitude:
0.0870148059635456},
{id: 15, name: 'Rm 6', latitude: 51.56838978470056,longitude:
0.08657671196087069},
```

```
{id: 16, name: 'Rm 5',latitude:  
51.56834754872497,longitude:0.08658028826684205 },  
{id: 17, name: 'Rm 4',latitude: 51.56834588150305,longitude:  
0.0864998220782498},  
{id: 18, name: 'Rm 3',latitude: 51.56834365852768,longitude:  
0.08635766508231502},  
{id: 19, name: 'Rm 2',latitude: 51.568346992933456,longitude:  
0.08623785977360132},  
{id: 20, name: 'Rm 1',latitude: 51.56834254704632,longitude:  
0.08609570277766654},  
{id: 22, name: 'Main Reception', latitude:51.56826418811395 ,longitude:  
0.08591957116149462},  
{id: 28,name: 'Rm 18', latitude: 51.568353661617074,longitude:  
0.08556373178688848},  
{id: 29,name: 'Rm 17', latitude:  
51.56835532881327,longitude:0.08540190514903845 },  
{id: 30, name: 'Rm 16',latitude: 51.56835032717761,longitude:  
0.08526779478014834},  
{id: 33,name: 'Dining Hall', latitude: 51.5685076002505,longitude:  
0.0851408367776374},  
{id: 34, name: 'Rm 15',latitude:51.56839645321098 ,longitude:  
0.08526868871339577},  
{id: 35, name: 'Rm 19',latitude:51.568419794090715 ,longitude:  
0.08526779469829382},  
{id: 36, name: 'Rm 14',latitude: 51.56845202671261,longitude:  
0.08527137092241066},  
{id: 37, name: 'Rm 20',latitude: 51.568480924877015,longitude:  
0.08527584124348397},  
{id: 38,name: 'Rm 13', latitude:51.568527606495635 ,longitude:  
0.08527673550414949},  
{id: 39, name: 'Rm 21',latitude: 51.568551503058885,longitude:  
0.08528120590707733 },  
{id: 40, name: 'Rm 12',latitude: 51.568602630514086,longitude:  
0.08527852369806244},  
{id: 44, name: 'Boys Toilet', latitude:  
51.568752122405726,longitude:0.08535094366888263 },  
{id: 45, name: 'Rm 11',latitude: 51.56869988361901,longitude:  
0.08549757114959977},  
{id: 46, name: 'Art Tech', latitude:
```

```

51.56869988361027,longitude:0.08563436389121382 },
{id: 47, name: 'Rm 10',latitude: 51.56869599348525,longitude:
0.08573539373682326},
{id: 49, name: 'Main Hall',latitude:51.56851871458137 ,longitude:
0.08592672475569163},
{id: 52, name: 'Rm 8',latitude: 51.568693770453066,longitude:
0.08608586934823581},
{id: 53, name: 'IT2',latitude: 51.56869265897152,longitude:
0.08624233156805605},
{id: 54, name: 'IT Tech',latitude: 51.56869377043558,longitude:
0.08630491647235505},
{id: 55, name: 'IT1',latitude:
51.56868988031005,longitude:0.08641131073599428 },
{id: 57, name: 'Girls Toilet',latitude: 51.56874767637935,longitude:
0.08646942523736545},
{id: 60,name: 'Gym' , latitude:
51.56891550688693,longitude:0.08638448872769988 },
{id: 61, name: 'SCI Staff',latitude: 51.56892439853998,longitude:
0.08611358569905025},
{id: 64, name: 'Astro Turf',latitude: 51.569202817912135,longitude:
0.08631296350867235},
{id: 65, name: 'Lecture Theatre', latitude: 51.569019983695,longitude:
0.08667684983774127},
{id: 68, name: '6th Form Common Room',latitude:
51.569051660128366,longitude: 0.08694954138772193},
{id: 70, name: 'Staff Room',latitude: 51.568599851179364,longitude:
0.08656956188641818},
{id: 72, name: 'Rm 7',latitude:
51.56844480127132,longitude:0.08657939678923032 },
{id: 73, name: 'Library', latitude:51.56854927938607
,longitude:0.0866893672769864 }

];

```

The graph structure works so that it has nodes connected by edges, in which the edges have to be traversable, e.g. if the path between two nodes is blocked by a building it can't be an edge, since it's not traversable.

```
const nodes = [
```

```
{id:0, latitude: 51.568163655684835 ,longitude: 0.08689513935488336},  
{id: 1, latitude: 51.568163655684835 ,longitude: 0.08689513935488336},  
{id: 2, latitude: 51.568172547493674,longitude: 0.0868942453397814},  
{id: 3, latitude: 51.56823145571189,longitude:0.08689513960044692 },  
{id: 4, latitude: 51.56826029831658,longitude: 0.08689678819390867},  
{id: 5, latitude: 51.568304201545864,longitude: 0.08689678835761772},  
{id: 6, latitude: 51.568360886676196,longitude: 0.08689678843947224},  
{id: 7, latitude: 51.568384227566234,longitude: 0.08689857663338518},  
{id: 8, latitude: 51.568413681519125,longitude: 0.08689678860318129},  
{id: 9, latitude: 51.56844146824725,longitude: 0.08689768270013776},  
{id: 10, latitude: 51.56850259901476,longitude: 0.08690125908796364},  
{id: 11, latitude: 51.568549280637356,longitude: 0.08690841169990637},  
{id: 12, latitude: 51.56850037611065,longitude: 0.0870148059635456},  
{id: 13, latitude: 51.568355884808206,longitude: 0.08665091979818573},  
{id: 14, latitude: 51.56838533883034,longitude: 0.08665091979818573},  
{id: 15, latitude: 51.56838978470056,longitude: 0.08657671196087069},  
{id: 16, latitude: 51.56834754872497,longitude:0.08658028826684205 },  
{id: 17, latitude: 51.56834588150305,longitude: 0.0864998220782498},  
{id: 18, latitude: 51.56834365852768,longitude: 0.08635766508231502},  
{id: 19, latitude: 51.568346992933456,longitude: 0.08623785977360132},  
{id: 20, latitude: 51.56834254704632,longitude: 0.08609570277766654},  
{id: 21, latitude: 51.5683342109861,longitude: 0.08592672369158283},  
{id: 22, latitude:51.56826418811395 ,longitude: 0.08591957116149462},  
{id: 23, latitude: 51.56826641106776,longitude: 0.08584446939093215},  
{id: 24, latitude:51.56833309947904 ,longitude: 0.0858283761368428},  
{id: 25, latitude:51.56833309947904 ,longitude: 0.0858283761368428},  
{id: 26, latitude: 51.5683492158069,longitude: 0.08572377006711651},  
{id: 27, latitude: 51.56835366165524,longitude: 0.08566386745368693},  
{id: 28, latitude: 51.568353661617074,longitude: 0.08556373178688848},  
{id: 29, latitude: 51.56835532881327,longitude:0.08540190514903845 },  
{id: 30, latitude: 51.56835032717761,longitude: 0.08526779478014834},  
{id: 31, latitude: 51.56834921568767,longitude: 0.08519716324880466},  
{id: 32, latitude: 51.568351994369166,longitude:0.08515067159859502 },  
{id: 33, latitude: 51.5685076002505,longitude: 0.0851408367776374},  
{id: 34, latitude:51.56839645321098 ,longitude: 0.08526868871339577},  
{id: 35, latitude:51.568419794090715 ,longitude: 0.08526779469829382},  
{id: 36, latitude: 51.56845202671261,longitude: 0.08527137092241066},  
{id: 37, latitude: 51.568480924877015,longitude: 0.08527584124348397},  
{id: 38, latitude:51.568527606495635 ,longitude: 0.08527673550414949},
```

```
{id: 39, latitude: 51.568551503058885,longitude: 0.08528120590707733 },  
{id: 40, latitude: 51.568602630514086,longitude: 0.08527852369806244},  
{id: 41, latitude: 51.56865987097896,longitude: 0.08528299410099027},  
{id: 42, latitude: 51.568662093908664,longitude: 0.08535094350517358 },  
{id: 43, latitude: 51.56869988367147,longitude: 0.08535183768398458},  
{id: 44, latitude: 51.568752122405726,longitude: 0.08535094366888263 },  
{id: 45, latitude: 51.56869988361901,longitude: 0.08549757114959977},  
{id: 46, latitude: 51.56869988361027,longitude: 0.08563436389121382 },  
{id: 47, latitude: 51.56869599348525,longitude: 0.08573539373682326},  
{id: 48, latitude: 51.5687009950654 ,longitude: 0.08592851286775005},  
{id: 49, latitude: 51.56851871458137 ,longitude: 0.08592672475569163},  
{id: 50, latitude: 51.56869599345028,longitude: 0.08604027171312811},  
{id: 51, latitude: 51.568738784769636,longitude: 0.08603937769802616},  
{id: 52, latitude: 51.568693770453066,longitude: 0.08608586934823581},  
{id: 53, latitude: 51.56869265897152,longitude: 0.08624233156805605},  
{id: 54, latitude: 51.56869377043558,longitude: 0.08630491647235505},  
{id: 55, latitude: 51.56868988031005,longitude: 0.08641131073599428 },  
{id: 56, latitude: 51.568688768828416 ,longitude: 0.08648462447635286},  
{id: 57, latitude: 51.56874767637935,longitude: 0.08646942523736545},  
{id: 58, latitude: 51.56833309947904 ,longitude: 0.0858283761368428},  
{id: 59, latitude: 51.568753233625074,longitude: 0.08638627692161283 },  
{id: 60, latitude: 51.56891550688693,longitude: 0.08638448872769988 },  
{id: 61, latitude: 51.56892439853998,longitude: 0.08611358569905025},  
{id: 62, latitude: 51.569009980751396,longitude: 0.08637912439152462 },  
{id: 63, latitude: 51.56901275937503,longitude: 0.08630759884507899},  
{id: 64, latitude: 51.569202817912135,longitude: 0.08631296350867235},  
{id: 65, latitude: 51.569019983695,longitude: 0.08667684983774127},  
{id: 66, latitude: 51.56901553786016,longitude: 0.08681006635523847 },  
{id: 67, latitude: 51.56904332421962,longitude: 0.08681543085512278 },  
{id: 68, latitude: 51.569051660128366,longitude: 0.08694954138772193},  
{id: 69, latitude: 51.568682655383554,longitude: 0.08656956180456366},  
{id: 70, latitude: 51.568599851179364,longitude: 0.08656956188641818},  
{id: 71, latitude: 51.568548723669366,longitude: 0.0865686677894617 },  
{id: 72, latitude: 51.56844480127132,longitude: 0.08657939678923032 },  
{id: 73, latitude: 51.56854927938607 ,longitude: 0.0866893672769864 }  
];
```

```
const edges = [
    {from: 1,to: 2,weight: calculateDistance(nodes[1], nodes[2]) },
    {from: 2,to: 3,weight: calculateDistance(nodes[ 2], nodes[ 3]) },
    {from: 3,to: 4,weight: calculateDistance(nodes[3 ], nodes[ 4]) },
    {from: 4,to: 5,weight: calculateDistance(nodes[ 4], nodes[ 5]) },
    {from: 5,to: 6,weight: calculateDistance(nodes[5 ], nodes[ 6]) },
    {from: 6,to: 7,weight: calculateDistance(nodes[6 ], nodes[7 ]) },
    {from: 7,to: 8,weight: calculateDistance(nodes[7 ], nodes[ 8]) },
    {from: 8,to: 9,weight: calculateDistance(nodes[ 8], nodes[9 ]) },
    {from: 9,to: 10,weight: calculateDistance(nodes[ 9], nodes[10 ]) },
    {from: 10,to: 11,weight: calculateDistance(nodes[ 10], nodes[11 ]) },
    {from:10 ,to: 12,weight: calculateDistance(nodes[10 ], nodes[12 ]) },
    {from: 6,to: 13,weight: calculateDistance(nodes[6 ], nodes[13 ]) },
    {from: 13,to: 14,weight: calculateDistance(nodes[ 13], nodes[14 ]) },
    {from: 14,to: 15,weight: calculateDistance(nodes[ 14], nodes[15 ]) },
    {from: 15,to:16 ,weight: calculateDistance(nodes[15 ], nodes[16 ]) },
    {from:16 ,to: 17,weight: calculateDistance(nodes[ 16], nodes[ 17]) },
    {from:17 ,to: 18,weight: calculateDistance(nodes[ 17], nodes[ 18]) },
    {from: 18,to:19 ,weight: calculateDistance(nodes[18 ], nodes[19 ]) },
    {from: 19,to:20 ,weight: calculateDistance(nodes[ 19], nodes[20 ]) },
    {from: 20,to: 21,weight: calculateDistance(nodes[ 20], nodes[21 ]) },
    {from: 21,to: 22,weight: calculateDistance(nodes[ 21], nodes[22 ]) },
    {from: 21,to: 24,weight: calculateDistance(nodes[21 ], nodes[24 ]) },
    {from: 22,to: 23,weight: calculateDistance(nodes[ 22], nodes[23 ]) },
    {from: 24,to: 26,weight: calculateDistance(nodes[ 24], nodes[ 26]) },
    {from: 27,to: 26,weight: calculateDistance(nodes[ 27], nodes[ 26]) },
    {from: 27,to: 28,weight: calculateDistance(nodes[ 27], nodes[28 ]) },
    {from: 28,to: 29,weight: calculateDistance(nodes[28 ], nodes[29 ]) },
    {from: 29,to: 30,weight: calculateDistance(nodes[29 ], nodes[ 30]) },
    {from: 30,to: 31,weight: calculateDistance(nodes[ 30], nodes[31 ]) },
    {from: 31,to: 32,weight: calculateDistance(nodes[31 ], nodes[32 ]) },
    {from: 32,to: 33,weight: calculateDistance(nodes[ 32], nodes[33 ]) },
    {from: 30,to: 34,weight: calculateDistance(nodes[30 ], nodes[34 ]) },
    {from: 34,to: 35,weight: calculateDistance(nodes[34 ], nodes[35 ]) },
    {from: 35,to: 36,weight: calculateDistance(nodes[ 35], nodes[ 36]) },
    {from: 36,to: 37,weight: calculateDistance(nodes[36 ], nodes[ 37]) },
    {from: 38,to:37 ,weight: calculateDistance(nodes[37 ], nodes[38 ]) },
    {from: 38,to: 39,weight: calculateDistance(nodes[38], nodes[ 39]) },
]
```

```

{from: 39,to: 40,weight: calculateDistance(nodes[ 39], nodes[ 40]) },
{from: 40,to: 41,weight: calculateDistance(nodes[40 ], nodes[ 41]) },
{from: 41,to: 42,weight: calculateDistance(nodes[ 41], nodes[42 ]) },
{from: 42,to: 43,weight: calculateDistance(nodes[42 ], nodes[43 ]) },
{from: 43,to:44 ,weight: calculateDistance(nodes[ 43], nodes[44 ]) },
{from: 43,to: 45,weight: calculateDistance(nodes[43 ], nodes[45 ]) },
{from: 45,to: 46,weight: calculateDistance(nodes[ 45], nodes[46 ]) },
{from: 46,to: 47,weight: calculateDistance(nodes[46 ], nodes[47 ]) },
{from: 47,to: 48,weight: calculateDistance(nodes[47 ], nodes[48 ]) },
{from: 48,to: 49,weight: calculateDistance(nodes[48 ], nodes[ 49]) },
{from: 49,to: 21,weight: calculateDistance(nodes[49 ], nodes[21 ]) },
{from: 48,to: 50,weight: calculateDistance(nodes[48 ], nodes[50 ]) },
{from: 50,to: 51,weight: calculateDistance(nodes[50 ], nodes[51 ]) },
{from: 50,to: 52,weight: calculateDistance(nodes[50 ], nodes[52 ]) },
{from: 52,to: 53,weight: calculateDistance(nodes[52 ], nodes[53 ]) },
{from: 53,to: 54,weight: calculateDistance(nodes[ 53], nodes[ 54]) },
{from: 54,to: 55,weight: calculateDistance(nodes[ 54], nodes[55 ]) },
{from: 55,to: 56,weight: calculateDistance(nodes[ 55], nodes[ 56]) },
{from: 57,to: 56,weight: calculateDistance(nodes[ 57], nodes[ 56]) },
{from: 57,to: 59,weight: calculateDistance(nodes[ 57], nodes[59 ]) },
{from: 59,to: 60,weight: calculateDistance(nodes[59 ], nodes[ 60]) },
{from: 60,to:61 ,weight: calculateDistance(nodes[60 ], nodes[61 ]) },
{from: 60,to:62 ,weight: calculateDistance(nodes[60 ], nodes[62 ]) },
{from: 62,to: 63,weight: calculateDistance(nodes[62], nodes[63]) },
{from: 63,to:64 ,weight: calculateDistance(nodes[63], nodes[64]) },
{from: 62,to: 65,weight: calculateDistance(nodes[62], nodes[65]) },
{from: 65,to:66 ,weight: calculateDistance(nodes[65], nodes[66]) },
{from:66 ,to: 67,weight: calculateDistance(nodes[66], nodes[67]) },
{from:67 ,to:68 ,weight: calculateDistance(nodes[67], nodes[68]) },
{from: 56,to: 69,weight: calculateDistance(nodes[56], nodes[69]) },
{from: 70,to: 69,weight: calculateDistance(nodes[70], nodes[69]) },
{from: 70,to: 71,weight: calculateDistance(nodes[70], nodes[71]) },
{from: 71,to: 72,weight: calculateDistance(nodes[71], nodes[72]) },
{from: 72,to: 15,weight: calculateDistance(nodes[72], nodes[15]) },
{from: 71,to: 73,weight: calculateDistance(nodes[71], nodes[73]) },
{from: 73,to: 11,weight: calculateDistance(nodes[73], nodes[11]) },
{from: 59,to: 51,weight: calculateDistance(nodes[59], nodes[51]) },
{from: 51,to: 60,weight: calculateDistance(nodes[51], nodes[60]) },
//{from: ,to: ,weight: calculateDistance(nodes[ ], nodes[ ]) },

```

```
];
```

This is a function to openExternalLink, with Validation and error checking. It takes a parameter url, which is:

<https://maps.app.goo.gl/vWdmYnToUYPwMJbYA>

This is the link for the google maps Seven Kings School Address. I created this short function to act as a means to direct users to the School if they are a set distance away from the school, for testing purposes (since I am not close to school), I commented out this feature, to prevent always linking to google maps app.

The function is called later on!

Any errors are validated and presented with an Alert which informs the user of any problems, so that the URL linked is not an invalid url and that the user is safe.

```
function openExternalLink(url) {  
  
    // Check if the URL is valid  
    Linking.canOpenURL(url)  
        .then((supported) => {  
            if (supported) {  
                // Open the URL if it's valid  
                Linking.openURL(url)  
                    .catch((err) => {  
                        console.error("Failed to open URL:", err);  
                        Alert.alert("Error", "Something went wrong while opening  
the link.");  
                    });  
            } else {  
                Alert.alert("Error", "This URL cannot be opened.");  
            }  
        })  
        .catch((err) => {  
            console.error("Error checking URL:", err);  
            Alert.alert("Error", "Something went wrong.");  
        });  
}
```

The function for handleCalculateRoute() is the most important function in the entire application. It takes in an ID of the node and performs an A* algorithm from the closest node to userLocation to the ID desired by the user.

Changes to this function from the previous section was that cNodeID is the closest node ID and the function now takes in an input, which allows it to be dynamic and route various paths, rather than just be static and route the same nodes, like the error we had previously.

Now the animation is presented in the second half of the code and primarily uses two functions to do so, rotateMapToNode and rotateMapToNode2, one is for aesthetics and one is to reorient the user back to its original view, since a tilt effect was added in the first one for an aesthetic appeal meeting Test ID 2.31, the function primarily aims to orient the user to face the direction of the target node, this helps the user know which direction to be heading, using this animation I am successfully meeting SC.01.

The application should have a smooth and responsive user interface (UI) system.

```
const handleCalculateRoute = (id) => {
    const cNodeID = (findClosestNodeId(userLocation, nodes))
    goToPlace(nodes[cNodeID])
    console.log(cNodeID)
    const path = findPath(cNodeID, id); // Example: from node 1 to node
4
    const coordinates = path.map((id) => {
        const node = nodes.find((n) => n.id === id);
        return { latitude: node.latitude, longitude: node.longitude };
    });
    setPathCoordinates(coordinates);
    // Rotate the map towards the target node
    setTimeout(() => {
        rotateMapToNode(nodes[cNodeID], nodes[id]);
        // Delay second rotation after first completes
        setTimeout(() => {
            rotateMapToNode2(nodes[cNodeID], nodes[id]);
        }, 1500);
    });
}
```

```
}, 1000);  
};
```

Although the second half of the `findClosestNodeId()` function was commented out for testing features it has been tested and works as intended, where if the distance was greater than a set distance I chose in this case from node 11 and 17, the user should be linked to google maps to arrive at the school first then use the application.

Other than that, no changes to the previous sections.

```
function findClosestNodeId(userLocation, nodes, places) {  
    if (!userLocation || !nodes || nodes.length === 0) {  
        throw new Error("Invalid userLocation or nodes array.");  
    }  
  
    let closestNodeId = null;  
    let shortestDistance = Infinity;  
  
    // Loop through all nodes and calculate the distance to the user  
    location  
    nodes.forEach((node) => {  
        const distance = calculateDistance(userLocation, node);  
  
        // If the calculated distance is smaller than the shortestDistance,  
        update  
        if (distance < shortestDistance) {  
            shortestDistance = distance;  
            closestNodeId = node.id;  
        }  
    });  
  
    // After the loop, check if the shortestDistance exceeds the distance  
    between nodes[11] and nodes[71]  
    if (shortestDistance > calculateDistance(nodes[11], nodes[71])) {  
        console.log("done");  
        // Trigger navigation to a specific place and open the external  
        Google Maps link  
        //goToPlace(places[15]);  
        openExternalLink("https://maps.app.goo.gl/vWdmYnToUYPwMJbYA");  
    }  
}
```

```

        return;
    }
    return closestNodeId;
}

```

resetRoute is a short function which resets the Path and PathCoordinates to allow the map to be cleared of any route lines, by setting it to null.

```

function resetRoute() {
    setPath([]);
    setPathCoordinates([])
}

```

The calculate bearing function is a mathematical function which links to the Haversine formula and is very similar to the CalculateDistance function we coded earlier, however in this case the return value is an angle, to present the angle/bearing the user should be facing.

The rotateMapToNode, adjusts the map setting to go to the bearing orientation aesthetically with set timings using a timeout feature.

```

const calculateBearing = (start, end) => {
    const lat1 = (Math.PI / 180) * start.latitude;
    const lat2 = (Math.PI / 180) * end.latitude;
    const lon1 = (Math.PI / 180) * start.longitude;
    const lon2 = (Math.PI / 180) * end.longitude;

    const dLon = lon2 - lon1;

    const y = Math.sin(dLon) * Math.cos(lat2);
    const x =
        Math.cos(lat1) * Math.sin(lat2) -
        Math.sin(lat1) * Math.cos(lat2) * Math.cos(dLon);
    let bearing = Math.atan2(y, x) * (180 / Math.PI);

    return (bearing + 360) % 360; // Normalize between 0-360
};

const rotateMapToNode = (currentNode, targetNode) => {
    if (!mapRef.current || !currentNode || !targetNode) return;

```

```

const bearing = calculateBearing(currentNode, targetNode);

mapRef.current.animateCamera({
  center: currentNode, // Keep centered on current location
  heading: bearing, // Rotate towards target node
  pitch: 45, // Slight tilt for a 3D effect
  zoom: 18, // Adjust zoom level
}, { duration: 2500 }); // Smooth animation
};

const rotateMapToNode2 = (currentNode, targetNode) => {
  if (!mapRef.current || !currentNode || !targetNode) return;

  const bearing = calculateBearing(currentNode, targetNode);

  mapRef.current.animateCamera({
    center: currentNode, // Keep centered on current location
    heading: bearing, // Rotate towards target node
    pitch: 0, // Reset tilt for a 3D effect
    zoom: 25, // Adjust zoom level
  }, { duration: 1000 }); // Smooth animation
};

```

I have removed the section where I had the map and have the polyline for all the edges, since I was satisfied with how it looks and had no reason to keep it.

When pathCoordinates has a value and path has a value the polyline is initiated with aesthetic colours and styling, with a prominent touch.

```

{pathCoordinates.length > 0 && (
  <Polyline
    coordinates={pathCoordinates}
    strokeColor="rgba(66, 133, 244, 0.3)" // Light blue
    transparentGlow
    strokeWidth={20} // Bigger for a glowing effect
  />, 
  <Polyline
    coordinates={pathCoordinates}
    strokeColor="white" // White
    strokeDash={[4, 4]} // Dashed style
    strokeWidth={20} // Bigger for a glowing effect
  />
);

```

```
        strokeColor="white"
        strokeWidth={16} // Thicker white outline for contrast
      />,
    <Polyline
      coordinates={pathCoordinates}
      strokeColor="#4285F4"
      strokeWidth={12} // Slightly smaller than the outline
    />
  )}
```

This button was a new addition which I added below the recenter button which resets the routes, and utilises the function we mentioned earlier.

```
<TouchableOpacity style={styles.resetRoute} onPress={resetRoute}>
  <Text style={styles.RecenterButtonText}>Clear
  Route</Text>
</TouchableOpacity>
```

The final addition was to adjust the onPress function for the button which calculates the entire route and I set a timeout function to add animation effects as the map goToPlace and then routes. I added a testing method of logging the Item.id to ensure the correct routes are being formed.

```
<TouchableOpacity
  style={styles.resultItem}
  onPress={() => {
    goToPlace(item);

    setTimeout(() => {

      console.log(`Item ID: ${item.id}`)
      setPath([])
      handleCalculateRoute(item.id)
      ;
    }, 2500);
  }>
```

These variables need to be set to initiate the view for the model of Common Places and Search Location separately.

This next part is in relation to the addition of the Common Places Section.

```
const [isCommonPlacesModalVisible, setIsCommonPlacesModalVisible] =  
useState(false);  
  
const [isSearchLocationModalVisible, setIsSearchLocationModalVisible]  
= useState(false);
```

The modal for the Common Places follows a similar structure for the Search Location but its not a search input, but rather direct buttons, I implemented another array here:

```
const commonPlaces = [  
    { id: 73, name: 'Library' },  
    { id: 49, name: 'Main Hall' },  
    { id: 60, name: 'Gym' },  
    { id: 22, name: 'Main Reception' },  
    { id: 11, name: 'Sports Hall' },  
    { id: 33, name: 'Dining Hall' },  
    { id: 44, name: 'Boys Toilet' },  
    { id: 57, name: 'Girls Toilet' },  
    { id: 70, name: 'Staff Room' },  
    { id: 65, name: 'Lecture Theatre' },  
];
```

To set the markers which are Common Places.

```
<TouchableOpacity onPress={() => setIsCommonPlacesModalVisible(true)}>  
    <View style={styles.Box}>  
        <Text style={styles.Label}>Common Places</Text>  
    </View>  
    </TouchableOpacity>  
    {/* Modal for Common Places */}  
    <Modal  
        visible={isCommonPlacesModalVisible}  
        transparent={true}  
        animationType="slide"  
    >
```

```

        <View style={styles.modalContainer}>
            <View style={styles.modalContent}>
                <ScrollView
contentContainerStyle={styles.scrollViewContainer}>
                {/* Preset buttons for each common place */}
                {commonPlaces.map((place) => (
                    <TouchableOpacity
                        key={place.id}
                        style={styles.placeButton}
                        onPress={() => {
                            setIsCommonPlacesModalVisible(false)
                            goToPlace(nodes[place.id]);
                            setTimeout(() => {
                                console.log(`Selected Place ID:
${place.id}`);
                                setPath([]);
                                handleCalculateRoute(place.id);
                            }, 2000);
                        }}
                    >
                        <Text
style={styles.placeButtonText}>{place.name}</Text>
                    </TouchableOpacity>
                ))}
            </ScrollView>
            <TouchableOpacity
                style={styles.closeButton}
                onPress={() =>
setIsCommonPlacesModalVisible(false)}
            >
                <Text style={styles.closeButtonText}>Close</Text>
            </TouchableOpacity>
        </View>
    </View>
</Modal>
</View>

/* Search Location Section */

```

```
<View style={styles.SearchLocation}>
    <TouchableOpacity onPress={() =>
setIsSearchLocationModalVisible(true)}>
        <Text style={styles.SearchLabel}>Search Location</Text>
    </TouchableOpacity>
</View>

{/* Modal for Searching Locations */}
<Modal
    visible={isSearchLocationModalVisible}
    transparent={true}
    animationType="slide"
>
    <View style={styles.modalContainer}>
        <View style={styles.modalContent}>
            <TextInput
                KeyboardAvoidingView = 'false'
                style={styles.searchInput}
                placeholder="Search for a place..."
                value={searchQuery}
                onChangeText={handleSearch}
            />
            <FlatList
                data={searchResults}
                keyExtractor={(item) => item.id.toString()}
                renderItem={({ item }) => (
                    <TouchableOpacity
                        style={styles.resultItem}
                        onPress={() => {
                            setIsSearchLocationModalVisible(false)
                            goToPlace(item);

                            setTimeout(() => {
                                console.log(`Item ID: ${item.id}`)
                                setPath([])
                                handleCalculateRoute(item.id)
                                ;
                            }, 2500);
                        }}
                )}
```

```

        >
        <Text
style={styles.resultText}>{item.name}</Text>
        </TouchableOpacity>
    )}
/>
<TouchableOpacity
style={styles.closeButton}
onPress={() =>
setIsSearchLocationModalVisible(false)}
>
    <Text style={styles.closeButtonText}>Close</Text>
    </TouchableOpacity>
</View>
</View>
</Modal>

```

This modal is a pop-up that appears when `isCommonPlacesModalVisible` is true, with a transparent background and a slide-in animation. Inside, it contains a scrollable list of buttons representing common places. When a button is pressed, the modal closes (`setIsCommonPlacesModalVisible(false)`), the selected place's location is set with `goToPlace(nodes[place.id])`, and after a 2-second delay, the route is recalculated by clearing the current path and calling `handleCalculateRoute(place.id)`. This ensures a smooth transition from selection to navigation, meeting SC.16:

Aesthetically pleasing and functional design to the user which promotes the user experience.

Evaluation:	Overall, this was a great fixing duty to the error in the path finding algorithm not working, by creating a new graph and updating our inputs and outputs of certain functions we ensure the functionality of the app. The designs were focussed using animation tricks and the addition of the Reset Route button, which is most necessary.
-------------	--

Test Table 25 – Initialising Log-In Screen

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
2.28-FIX	Testing the route generation functionality	Select a marker (e.g., Classroom) on the map or use the "Search Location" button to input a destination.	When the searches for a location, the application should calculate and display a route from the closest node to the destination using the A* Algorithms.	The route works as expected! It finds the shortest route between a closest node and the target node, displaying the route successfully, this happens after the user searches for a location.	Yes - fixed from previous error -SUCCESS!
2.30-FIX	Ensuring the route line is vivid and visible	Trigger a route calculation by selecting a destination or marker.	The route line should be visible, vivid (e.g., yellow or other contrasting color), and thick enough to be easily seen by users.	The line presented is a vivid blue colour which is thick and clear to users throughout the map. Since the map is not entirely blue, there is great contrast meeting the Test ID 2.30.	Yes - fixed from previous error
2.31	Testing the calculation of the shortest path	Select different destination locations and check if the algorithm calculates and	The A* algorithm should calculate and render the shortest possible path	The algorithm correctly displays the shortest path, calculating the optimal route based on the	Yes, the shortest path is presented.

		displays the shortest path.	from the current position or closest node to the destination.	input data.	
2.33	Ensuring user feedback on path calculation status	After clicking the "Search Location" button or selecting a marker, verify the app shows a loading indicator or similar status while calculating the route.	The app should indicate the status of the route calculation process (e.g., loading spinner, message).	The feedback is presented in an aesthetic, appealing fashion where it pans and zooms into the target location, showing clear feedback that the calculation has been successful and the route is ready to traverse.	Yes, the user will know when the calculation is complete and the route is found.
2.34	Button to Clear Route should be visible and prevalent and functional	Attempt pressing Clear Route with no route on screen, to detect errors. Search for a couple of routes and see if the route is being set onPress of the Clear Route button.	The Clear Route button should be obvious and present to the user and it should be on the map clearly. onPress it should remove any route lines.	There is a Clear Route button below the Recenter button and onPress it clears any routes, pressing it on its own leads to no errors.	Yes, the clear route button is present and functional.
2.35	Route resets when the reset option is selected	Visual check for a reset button. Click the button after selecting a route.	The route should disappear from the screen, and no polyline should be visible.	When the reset route button is pressed, the polyline disappears and is not to be found, as expected!	Yes

2.36	Resetting the route does not affect user location or school marker	Click the reset button and check that the user's location and school marker remain on the map.	Only the route should disappear, but user and school markers should remain unchanged.	When the reset route button is pressed, only the route disappears, the user and school markers remain unchanged as expected.	Yes
2.37	If the user's distance from school exceeds a threshold, prompt Google Maps redirection	Simulate a location far from the school (e.g., 5+ km away). Then initiate the route finding algorithm.	A prompt should appear, offering an option to open Google Maps for navigation.	When the conditions are satisfied the user is directly taken to the Google Maps app and the Seven Kings Map is presented.	Yes, any distance too far, the user will be routed to the School via Google Maps.
2.38	Google Maps opens with correct school destination	Click the Google Maps redirection button when prompted.	Google Maps should launch with directions pre-set to the school's location.	The map presented is for the route to School (Seven Kings).	Yes, linked to a correct location.
2.39	The Common Places button is pressable and triggers the modal.	Attempt to press the "Common Places" button.	Pressing the button should open a modal displaying preset locations.	Pressing the button successfully opens the modal with a list of common places.	Yes, it does trigger a modal.
2.40	Common Places modal has a list of preset locations.	Visual check inside the modal for multiple location buttons.	The modal should display a scrollable list of predefined locations.	The modal contains a correctly formatted, scrollable list of common places.	Yes, preset common locations are presented.

2.41	Selecting a place closes the modal.	Press a place button inside the modal and observe if it closes.	The modal should close immediately after selection.	The modal closes successfully when a place is selected.	Yes, selecting a place does close a modal.
2.42	The new route is visually displayed on the screen.	Select a common place and check for a visual path update.	The screen should display a highlighted path to the selected destination.	The route appears correctly as a blue path leading to the chosen place.	Yes
2.43	Close button in the modal is visible and pressable.	Visual check and attempt to press the "Close" button in the modal.	The "Close" button should be visible and functional.	The "Close" button is clearly visible and successfully closes the modal when pressed.	Yes

Evidence Table 25

Test ID	Screenshot/Video
Test ID 2.28, 2.30 FIX 2.31,2.3 3 - 2. 2.39 - 2.43	<p>Narrated Videos:</p> <p>https://youtu.be/F5nS_OU1M8I</p> <p>https://youtu.be/PXFY50o1sGc</p>
Evaluation:	<p>After resetting the application, this test evaluates functionalities from test IDs 2.28 to 2.38. Test IDs 2.28 and 2.30 address previous issues, while the rest introduce new features.</p> <p>Test ID 2.28, which ensures the route functions correctly, is now successfully met. The application accurately determines the closest node to the user's location and provides the shortest route to the selected destination. Panning to the gym confirms that the route is optimal by distance and weight, marking a major milestone in the app's</p>

development.

For test ID 2.30, the route line has been improved from yellow to a vivid blue, aligning better with the app's theme. It is also thicker, enhancing visibility. Additionally, unnecessary edges have been removed, ensuring only the relevant path is displayed. These changes confirm that the route is now clearly visible, meeting the test criteria.

Test ID 2.31, verifying the shortest path calculation, is also met, as the app consistently determines and displays the most efficient route. Meanwhile, test ID 2.33 focuses on providing feedback during route calculation. This is achieved through a panning animation that visually represents the route being processed. A slight delay has also been introduced to simulate realistic processing time, ensuring the user perceives the calculation progress effectively.

Test IDs 2.34 and 2.35 assess the "Clear Route" button. The button is visible, clearly labeled, and functional, successfully removing the displayed route upon selection. This confirms that both its visibility (2.34) and functionality (2.35) meet the required standards.

For test ID 2.36, resetting the route does not affect the user's location or school markers. A comparison of the markers before and after resetting confirms that their positions remain unchanged, ensuring route clearing does not interfere with other map elements.

Lastly, test ID 2.37, which involves the distance exceeding a predefined threshold, is currently disabled for testing. The feature has been commented out as the user is not physically close to the school. However, it can be re-enabled when needed for further validation.

This test covers the implementation of the "Common Places" button, focusing on test IDs 2.39 to 2.43.

Test ID 2.39 ensures that the button is pressable and successfully triggers a modal. Upon pressing the button, the modal opens as expected, displaying a list of common places. This confirms that the functionality is correctly implemented.

Test ID 2.40 checks whether the list contains preset locations. The modal does display a predefined set of locations, although scrolling is not currently necessary due to the limited number of entries. However, the presence of a structured list confirms that the test is met.

Test ID 2.41 verifies that selecting a location closes the modal. This is confirmed by selecting "Sport Hall," which causes the modal to close while simultaneously generating a new route. This also satisfies test ID 2.42, which ensures that selecting a location correctly forms and displays

a route on the screen.

For test ID 2.43, the modal's close button is tested for visibility and functionality. The button is clearly visible, and pressing it successfully closes the modal, confirming that it performs as intended.

Further testing was conducted on various locations, including the library, main hall, and dining hall, confirming that the feature consistently works as expected. The longest route was also tested, and the upcoming addition of an estimated time of arrival (ETA) function is expected to enhance the visualization of travel times. Overall, the implementation of the "Common Places" button is complete and fully functional.

Version 8 - Initialising Settings Page

Initialising Log-In Screen

Relevant Source Code:

```
import React, { useState } from "react";
import { View, Text, TouchableOpacity, ScrollView, TextInput,
Switch, StyleSheet, Modal, FlatList, Dimensions } from
"react-native";
import Icon from "react-native-vector-icons/MaterialIcons"; // Choose any icon set
```

This is to import any components I am using for the Settings Page, these are predefined libraries and functions. Although most of it is in common with the imported components for the Main menu screen, I did not refactor this, since by adding excessive unused functions, I could slow the program down.

```
// Get screen dimensions
const { width, height } = Dimensions.get('window');

// Scaling functions
const BASE_WIDTH = 375; // Reference width (e.g., iPhone 8)
const BASE_HEIGHT = 667; // Reference height (e.g., iPhone 8)
const scaleWidth = (size) => (width / BASE_WIDTH) * size;
const scaleHeight = (size) => (height / BASE_HEIGHT) * size;
```

This code was refactored from the previous pages since it was needed for the styles component when portraying the buttons just as the designs section implies for the settings page.

```
const SettingsPage = ({navigation }) => {
  const [schoolInfoExpanded, setSchoolInfoExpanded] =
useState(false);
  const [personalInfoExpanded, setPersonalInfoExpanded] =
useState(false);
  const [additionalSettingsExpanded, setAdditionalSettingsExpanded] =
useState(false);
  const [formRoom, setFormRoom] = useState("");
  const [name, setName] = useState("");
  const [accessibilityNeeds, setAccessibilityNeeds] =
```

```

useState(false);
const [theme, setTheme] = useState("Light");
const [isSearchSchoolModalVisible, setIsSearchSchoolModalVisible] = useState(false);
const [searchQuery, setSearchQuery] = useState("");
const [searchResults, setSearchResults] = useState([]);
const [isDarkMode, setIsDarkMode] = useState(false);

```

Here I set the variable names and define a few of them like setName is useState() which updates and listens to any changes to the variable.

Although you can see I attempted to implement the theme functionality, but as a revised test plan I chose to remove it due to time constraints.

```

const schools = [
  { id: 1, name: "Seven Kings High School" },
  { id: 2, name: "Oaks Park High School" },
  { id: 3, name: "Brampton Manor Academy" },
  { id: 4, name: "Valentines High School" },
  { id: 5, name: "NCS" },
];

```

These are the schools defined in an array for the search modal when you see the option to Search for a Specific school.

```

const handleSearch = (query) => {
  setSearchQuery(query);
  if (query.trim() === "") {
    setSearchResults(schools); // Reset to full list if query is empty
  } else {
    const filteredSchools = schools.filter((s) =>
      s.name.toLowerCase().includes(query.toLowerCase())
    );
    setSearchResults(filteredSchools);
  }
};

const selectSchool = (schools) => {

```

```

    setIsSearchSchoolModalVisible(false);
    console.log(`Selected School ID: ${schools.name}`);
    // Implement school selection logic here
};
```

This function to SearchQuery and filter lists, was completely refactored from the Main Menu, since it uses a very similar function to the Search Location function in Main Menu. Through refactoring I reduced my time conditions and restrictions on this section of the application.

```

const toggleTheme = () => {
    setIsDarkMode((prevMode) => !prevMode);
};

const lightTheme = {
    backgroundColor: "#f0f0f0",
    textColor: "#000",
    buttonColor: "#38b6ff",
};

const darkTheme = {
    backgroundColor: "#121212",
    textColor: "#fff",
    buttonColor: "#1e90ff",
};

const currentTheme = isDarkMode ? darkTheme : lightTheme;

return (
    <View style={styles.container}>
        <ScrollView showsVerticalScrollIndicator={false}>
            <Text style={styles.title}>Settings</Text>

            {/* School Information Section */}
            <TouchableOpacity onPress={() =>
                setSchoolInfoExpanded(!schoolInfoExpanded)} style={styles.button}>
                <View style={styles.buttonInner}>
                    <Text style={styles.buttonText}>School
```

```

Information</Text>
    <Icon name={schoolInfoExpanded ? "keyboard-arrow-up" :
"keyboard-arrow-down"} size={24} color="black" />
</View>
</TouchableOpacity>
{schoolInfoExpanded && (
    <View style={styles.dropdownContent}>
        <TouchableOpacity style={styles.subButton} onPress={() => setIsSearchSchoolModalVisible(true)}>
            <Text style={styles.subButtonText}>Select
School</Text>
        </TouchableOpacity>
        <TextInput
            style={styles.input}
            placeholder="Enter Form Room Number"
            keyboardType="numeric"
            value={formRoom}
            onChangeText={setFormRoom}
        />
        <TouchableOpacity style={styles.subButton}>
            <Text style={styles.subButtonText}>Time-Table
Information</Text>
        </TouchableOpacity>
    </View>
)}
/* Modal for Searching Schools */
<Modal visible={isSearchSchoolModalVisible}
transparent={true} animationType="slide">
    <View style={styles.modalContainer}>
        <View style={styles.modalContent}>
            <TextInput
                style={styles.searchInput}
                placeholder="Search for a school..."
                value={searchQuery}
                onChangeText={handleSearch}
            />
            <FlatList
                data={searchResults}
                keyExtractor={(item) => item.id.toString()}>

```

```

        renderItem={({ item }) => (
          <TouchableOpacity
            style={styles.resultItem}
            onPress={() => selectSchool(item)}
          >
            <Text
              style={styles.resultText}>{item.name}</Text>
            </TouchableOpacity>
          )
        />
        <TouchableOpacity style={styles.closeButton}
          onPress={() => setIsSearchSchoolModalVisible(false)}>
          <Text style={styles.closeButtonText}>Close</Text>
        </TouchableOpacity>
      </View>
    </View>
  </Modal>

```

This entire chunk of code was to code the functionality of the School information Section as indicated by the commenting feature I mentioned in the beginning of development.

It features the use of a modal which was refactored from the main menu and it shows a list of schools to choose from, just as stated in test plans.

```

/* Personal Information Section */
<TouchableOpacity onPress={() =>
setPersonalInfoExpanded(!personalInfoExpanded)}
style={styles.button}>
  <View style={styles.buttonInner}>
    <Text style={styles.buttonText}>Personal
Information</Text>
    <Icon name={personalInfoExpanded ? "keyboard-arrow-up"
: "keyboard-arrow-down"} size={24} color="black" />
  </View>
</TouchableOpacity>
{personalInfoExpanded && (
  <View style={styles.dropdownContent}>
    <TextInput

```

```

        style={styles.input}
        placeholder="Enter Your Name"
        value={name}
        onChangeText={setName}
    />
    <View style={styles.toggleContainer}>
        <Text style={[styles.subButtonText, {color:
"white"}]}>Accessibility Needs</Text>
        <Switch value={accessibilityNeeds}
onValueChange={setAccessibilityNeeds} />
    </View>
</View>
)}
```

{/* Additional Settings Section */}

```

<TouchableOpacity onPress={() =>
setAdditionalSettingsExpanded(!additionalSettingsExpanded)}
style={styles.button}>
    <View style={styles.buttonInner}>
        <Text style={styles.buttonText}>Additional
Settings</Text>
        <Icon name={additionalSettingsExpanded ?
"keyboard-arrow-up" : "keyboard-arrow-down"} size={24}
color="black" />
    </View>
</TouchableOpacity>
{additionalSettingsExpanded && (
    <View style={styles.dropdownContent}>
        <TouchableOpacity onPress={() => setTheme(theme ===
"Light" ? "Dark" : "Light")}>
            <Text style={styles.subButton}>
                <Text style={styles.subButtonText}>Theme:
{theme}</Text>
            </Text>
        </TouchableOpacity>
        <TouchableOpacity style={styles.subButton}>
            <Text style={styles.subButtonText}>ETA Measure</Text>
        </TouchableOpacity>
    </View>
)}
```

```

<TouchableOpacity onPress={() => navigation.navigate('Main
```

```
Menu', {name})} style={{ marginTop: 'auto', alignSelf: 'center' }}>
    <View>
        <Text style={styles.formFooter}>Return to Main
    Menu?</Text>
    </View>
</TouchableOpacity>
</ScrollView>
</View>
);
};

const styles = StyleSheet.create({
    container: {
        flex: 1,
        backgroundColor: "#38b6ff",
        paddingHorizontal: 16,
        paddingVertical: 40,
    },
    title: {
        color: "#333333",
        fontSize: 38,
        textAlign: "center",
        fontFamily: "serif",
        fontWeight: "bold",
        textDecorationLine: 'underline', // Underlines the text
        marginVertical: 30,
    },
    buttonContent: {
        flexDirection: "row", // Align text and icon horizontally
        alignItems: "center", // Center vertically
    },
    button: {
        backgroundColor: "#38b6ff",
        padding: 15,
        borderRadius: 10,
        borderWidth: 2,
        marginVertical: 8,
    },
    buttonText: {
```

```
    textAlign: "center",
    fontSize: 22,
    fontWeight: 'bold',
},
dropdownContent: {
    backgroundColor: "rgba(0,0,0,0.5)",
    padding: 10,
    borderRadius: 10,
    marginVertical: 8,
},
subButton: {
    backgroundColor: "#38b6ff",
    padding: 10,
    borderRadius: 8,
    marginVertical: 5,
},
subButtonText: {
    textAlign: "center",
    fontSize: 18,
},
input: {
    backgroundColor: "white",
    padding: 10,
    borderRadius: 8,
    marginVertical: 5,
    borderWidth: 2,
    borderColor: "black",
    fontSize: 14,
},
toggleContainer: {
    flexDirection: "row",
    justifyContent: "space-between",
    alignItems: "center",
    padding: 10,
},
formFooter: {
    fontSize: 17,
    fontWeight: '600',
    textAlign: 'center',
```

```
letterSpacing: 0.3,  
padding: 20,  
  
, SearchLocation: {  
height: scaleHeight(60),  
borderColor: 'black',  
borderWidth: 3,  
borderRadius: scaleWidth(10),  
justifyContent: 'center',  
alignItems: 'center',  
marginVertical: scaleHeight(8),  
},  
SearchLabel: {  
fontSize: scaleHeight(16),  
fontWeight: 'bold',  
},  
modalContainer: {  
flex: 1,  
backgroundColor: 'rgba(0,0,0,0.5)',  
justifyContent: 'center',  
alignItems: 'center',  
},  
modalContent: {  
width: '80%',  
backgroundColor: '#38b6ff',  
borderRadius: scaleWidth(10),  
padding: scaleWidth(16),  
borderWidth: 3,  
borderColor: 'black'  
},  
searchInput: {  
borderBottomWidth: 1,  
marginBottom: scaleHeight(10),  
fontSize: scaleHeight(16),  
},  
resultItem: {  
padding: scaleHeight(10),  
borderBottomWidth: 1,  
},
```

```

    resultText: {
      fontSize: scaleHeight(16),
    },
    closeButton: {
      marginTop: scaleHeight(10),
      alignSelf: 'center',
    },
    closeButtonText: {
      color: 'black',
      fontWeight: 'bold',
      fontSize: scaleHeight(16),
    },
  });
}

export default SettingsPage;

```

Description: This is the entire code for the Settings Page in order, I have cut out sections to annotate and describe the functions happening.

To get to the Settings Page I have implemented this:

```

<TouchableOpacity style={styles.SettingsOutput}
  onPress={() => navigation.navigate("Settings Page")}>
  <Image
    source={require('../assets/settingsicons.png')}
    fadeDuration={500}
    style={styles.SettingsImg}
    alt="Settings"
  />
</TouchableOpacity>

```

This is for the Main Menu to get to the Settings Page.

Revised Test Plans for Settings Page:

Test Table 26 – Initialising Settings Page					
Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful?
3.01	Settings Page Label should be displayed	Visual check for the Settings Page label at the top	The label should be displayed at the top of the page and clearly visible to the user	The "Settings" label is displayed at the top of the page and is clearly visible.	Yes, the settings page title is displayed.
3.02	Background image should be displayed	Visual check for background image consistency	The same background image used in the app should be present on the Settings Page	The background image is displayed correctly and matches the theme.	Yes, the background image is displayed consistently.
3.03	School Information Button should be displayed	Visual check for the "School Information" button	The button labeled "School Information" should be visible and displayed as the first option	The "School Information" button is visible and correctly positioned as the first option.	Yes, the button is displayed.
3.04	Personal Information Button should be displayed	Visual check for the "Personal Information" button	The button labeled "Personal Information" should be visible and displayed as the second option	The "Personal Information" button is visible and correctly positioned as the second option.	Yes, the button is displayed.

3.05	Additional Settings Button should be displayed	Visual check for the "Additional Settings" button	The button labeled "Additional Settings" should be visible and displayed as the third/final option	The "Additional Settings" button is visible and correctly positioned as the third option.	Yes, the button is displayed.
3.06	School Information Button should function to open a drop-down menu	Press the "School Information" button	The button should expand to reveal the "Select School," "Form Room," and "Time-Table Information" options	Pressing the "School Information" button successfully expands to reveal the expected options.	Yes, the button opens a drop-down .
3.07	Personal Information Button should function to open a drop-down menu	Press the "Personal Information" button	The button should expand to reveal "Name" and "Accessibility Needs" sections	Pressing the "Personal Information" button successfully expands to reveal the expected sections.	Yes, the button opens a drop-down .
3.08	Additional Settings Button should function to open a drop-down menu	Press the "Additional Settings" button	The button should expand to reveal "Theme: Light/Dark" and "ETA Measure" sections	Pressing the "Additional Settings" button successfully expands to reveal the expected sections.	Yes, the button opens a drop-down .
3.09	Settings Page should have a functioning scroll feature	Attempt to scroll up and down the page	The scroll function should work smoothly, allowing the user to navigate all settings	Scrolling works smoothly, and all settings can be accessed without issues.	Yes, the settings has a scroll feature to be able to find all the buttons.

3.10	School Name Button should function properly	Press the "Select School" button	A list of schools should be presented for selection from the SchoolNav Map database	Pressing "Select School" opens a modal displaying a list of schools for selection.	Yes, the Select School button opens a list of schools for selection.
3.11	Form Room Input should function properly	Press the "Form Room" input field	A field should be displayed with a numeric keyboard for input	Pressing the "Form Room" input field correctly opens a numeric keyboard for input.	Yes, this button functions expectedly.
3.12	Name Input should function properly	Press the "Name" input field, and return to Main Menu for a visual check to see if Name is updated.	A text input field and keyboard should appear, allowing users to enter their name	Pressing the "Name" input field correctly opens a keyboard and allows text input. And if any name is entered the name will then be routed to the Main Menu and presented, however no name was entered, going back to the Main Menu leads to "Hello !" for the greeting.	No HyperLink to Solution
3.13 - VALIDATION	The name field should have proper validation to prevent any invalid inputs that are not of form name.	Enter a variety of names and check if an error message is produced, e.g. "123", "**", "	Any invalid inputs should output error messages to inform the user of the incorrect format.	This has not been implemented!	No – Not implemented yet.

		should not be accepted.			
3.14	Accessibility Needs Toggle should function properly	Press the "Accessibility Needs" toggle	A toggle should appear that allows users to enable or disable accessibility needs	Pressing the "Accessibility Needs" toggle correctly switches between enabled and disabled states.	Yes, this button functions expectedly.
3.16	The ETA Measure Button should function properly	Press the ETA Measure button	The ETA Measure button should be responsive and interactable.	The ETA Measure button behaves like a normal button and is interactable, providing feedback to users that they pressed it.	Yes, this button functions expectedly.
3.17	The Theme selector should be functional and working as expected, clear and relevant toggle to switch between themes.	Visual check for the Theme Selector and press it.	The Theme Selector should alternate as you toggle between it, from Light and Dark and this should be obvious and clear to the user.	The button is presented with Theme: Light e.g. and this is clear and relevant, each time a user presses it, the toggle changes to alternate to Dark vice versa.	Yes, this button functions expectedly.
3.18	The button to return to the Main Menu should function and be visible to users.	Visual check for option to return to Main Menu from Settings Page, Press the button.	The option should be clear and present to the user and it should take the user to the Main Menu when pressed.	There is a clear option on the bottom of the Settings page, where the user can "Return to the Main Menu". On press the user is appropriately taken to the Main Menu,	Yes, the return to Main Menu option is present.

				signifying the completion of this Test ID.	
--	--	--	--	--	--

I chose to remove Test IDs 3.19 – 3.23, where:

For reasons I further explain in the Evaluation section.

3.19	Accessibility Needs Button should function accordingly v2	Press the Accessibility Needs button and toggle Yes Accessibility is necessary.	The routes which require use of staircases/other unusable means, should be rerouted to nearest ramp/lifts.
3.20	Accessibility Needs Button should function accordingly v3	Press the Accessibility Needs button and toggle No Accessibility is necessary.	The routes which require use of staircases/other unusable means, should not be rerouted.
3.21	ETA Measure Button should function accordingly v2	Press the ETA Measure button and choose Hours, select a route to specific location.	The route should be calculated and the ETA should be in Hours (to the nearest 10 minutes).
3.22	ETA Measure Button should function accordingly v3	Press the ETA Measure button and choose Minute, select a route to specific location.	The route should be calculated and the ETA should be in Minutes (to the Nearest Minute).
3.23	ETA Measure Button should function accordingly v4	Press the ETA Measure button and choose Seconds, select a route to a specific location.	The route should be calculated and the ETA should be in Seconds, (to the nearest Minute)

Evidence Table 26

Test ID	Screenshot/Video	Evaluation
3.01 - 3.17	Narrated Video: https://youtu.be/5ZCG_ZNhGJ4	Both videos are narrated in great detail, the second video is a focus on the error, where the first video talks through the entire test IDs from 3.01 to 3.17.
3.12	In Depth Review of error in Test ID 3.12: https://youtube.com/shorts/IgpCsyTsx8I	The error is where: Pressing the "Name" input field correctly opens a keyboard and allows text input. And if any name is entered the name will then be routed to the Main Menu and presented, however no name was entered, going back to the Main Menu leads to "Hello!" for the greeting.

Fixing the name not being presented accurately and issues entering name in settings: Test ID 3.12

Relevant Source Code with Annotations:

```
const [error, setError] = useState("");
```

This defines the variables error and setError and initialises it as ""(empty), so that it can be added to through the program. The useState("") is used to show that the variable can update throughout the program and the program should listen to these updates. This is relevant when we see how the error messages are only presented when there is a value in errors.

```
const validateName = (text) => {
    const nameRegex = /^[A-Za-z]+(?: [A-Za-z]+)+$/; // Ensures at least two words with only letters
    if (!nameRegex.test(text)) {
        setError("Enter a valid full name (first and last name, letters only).");
    } else {
```

```
        setError("");
    }
    setName(text);
};
```

This is the function which is called later on, we can see how this is to validate the name input, through the function Regular Expression which is tested against the “text” parameter in which we see the name variable is then passed in.

Here is the website I used for guidance in using this:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions

Then if the name is of Regular Expression (a first name and a last name) then the setError is empty, else there is an error!

```
/* Personal Information Section */
<TouchableOpacity onPress={() =>
setPersonalInfoExpanded(!personalInfoExpanded)}
style={styles.button}>
    <View style={styles.buttonInner}>
        <Text style={styles.buttonText}>Personal
Information</Text>
        <Icon name={personalInfoExpanded ? "keyboard-arrow-up"
: "keyboard-arrow-down"} size={24} color="black" />
    </View>
</TouchableOpacity>
{personalInfoExpanded && (
    <View style={styles.dropdownContent}>
        <Text style={[styles.subButtonText, { color: "white",
textAlign: "left", paddingHorizontal: 10, paddingVertical:
5, }]}>Name:</Text>
        <TextInput
            style={[styles.input, error ? styles.inputError :
null]}
            placeholder={name === "Guest" ? "Enter your full
name" : "Change your name?"}
            value={name}
            onChangeText={validateName}
            autoCapitalize="words"
```

```

        />
        {error ? <Text style={styles.errorText}>{error}</Text>
: null}
        <View style={styles.toggleContainer}>
            <Text style={[styles.subButtonText, { color: "white"
}]}>Accessibility Needs</Text>
            <Switch value={accessibilityNeeds}>
                <onValueChange={setAccessibilityNeeds} />
            </Switch>
        </View>
    )
}

```

Added a name title and we see how the error text is presented when errors exist using the ternary operator.

`onChangeText={validateName}`

This shows how when the text input changes, the function `validateName` is run, passing through the text in the text input.

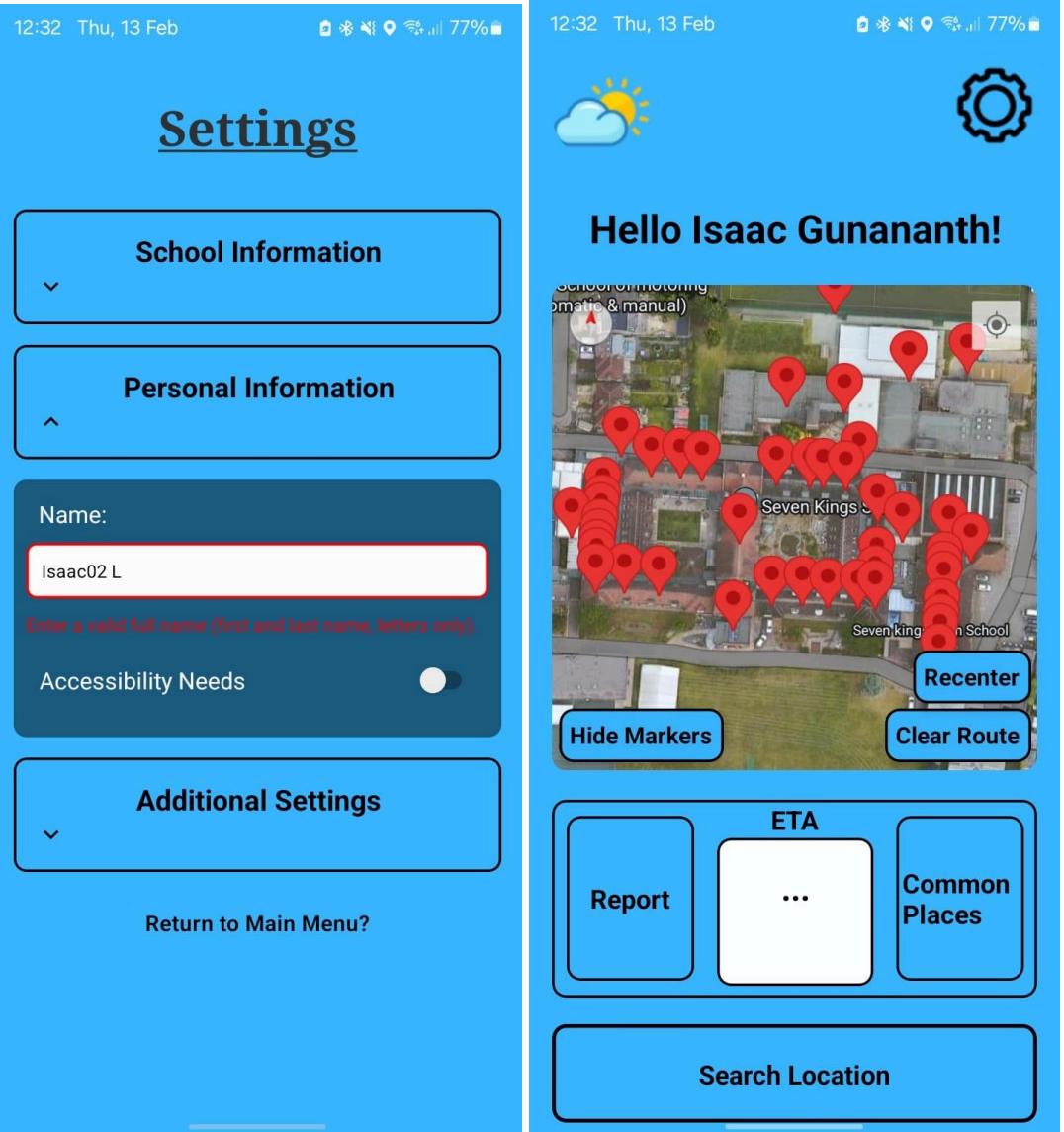
We see how using the ternary operators, the error messages are controlled by the existence of error messages, using styles the errors are then presented onto the input.

Test Table 27 – Initialising Settings Page and Cleaning the name, greeting function.

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful ?
3.12 -FIX	Name Input should function properly	Press the "Name" input field, and return to Main Menu for a visual check to see if Name is updated.	A text input field and keyboard should appear, allowing users to enter their name	Pressing the "Name" input field correctly opens a keyboard and allows text input. And if any name is entered the name will then be routed to the Main Menu and presented and presents the name	Yes - fixed from previous error The input now allows the name to be adjusted and presented

				consistently once entered!	on the Main Menu.
3.13	The name field should have proper validation to prevent any invalid inputs that are not of form name.	Enter a variety of names and check if an error message is produced, e.g. "123", "**", " " should not be accepted.	Any invalid inputs should output error messages to inform the user of the incorrect format.	With each test case the solution appropriately displays error messages with invalid input and no error messages with correct input. E.g. any inputs with a symbol e.g "Isaac**" would be presented with an error message.	Yes, there is appropriate validation.

Evidence Table 27

Test ID	Screenshot/Video
<p>3.12 - FIX Implem- entatio- n of 3.13</p>	<p>Narrated Video: https://youtube.com/shorts/S6-XlQKMJ2Q?feature=share</p> 
Evaluation:	<p>This video uncovers and talks through how it fixes and meets Test ID 3.12 and implements Test ID 3.13. The Screenshots show how the error message is presented when the format of the text input for name is not met and also notice how when I type in a valid name, the name is presented as the greeting.</p>

Version 9 – Add Commenting/Report Feature for Users and Adding ETA System

Initialising Log-In Screen

Relevant Source Code:

```
const [reportMode, setReportMode] = useState(false); // Track report mode
const [marker, setMarker] = useState(null); // Store the reported location
const [description, setDescription] = useState(""); // Store user input
const [reportModalVisible, setReportModalVisible] = useState(false); // Control modal
```

This the defining and initialising of reportMode, marker, description and reportModalVisible, which are all very important for the purpose of this iteration.

```
const handleMapPress = (marker) => {
  if (reportMode) {
    setMarker({
      latitude: marker.nativeEvent.coordinate.latitude,
      longitude: marker.nativeEvent.coordinate.longitude,
    });
    setReportModalVisible(true); // Open input modal
    setReportMode(false); // Disable report mode after selection
  }
};

const submitReport = () => {
  setReportModalVisible(false); // Close modal
};
```

These two functions: contribute to the report function, I am implementing here. The first function is to handleMapPress and later on we see how this is only triggered when the ReportMode is True, this forms a marker to appear in the place the user presses a place on the map. This sets the ReportModal visible so then the user can type in and set a description.

The submitReport function is formulated to close the modal once the description has been set and the user is done setting a report.

```

<MapView
    ref={mapRef} // Attach the ref to the MapView
    style={StyleSheet.absoluteFillObject}
    provider={PROVIDER_GOOGLE}
    mapType="hybrid"
    userLocationUpdateInterval={1000}
    region={userLocation} // Center the map on the user's
    location
    showsUserLocation={true} // Show the default blue dot
    followsUserLocation={true} // Automatically follow
    the user's location
    pitch={45} // Add a 3D tilt to the map
    heading={0} // Set the map's heading (0 degrees means
    north-up)

    onPress={handleMapPress} // Listen for user tap
>

```

We see the addition of `onPress={handleMapPress}` which listens for the user's tap on the map. This is only initiated when ReportMode is set. We will see here where we set it `onPress` with the Report button.

```

/* Input Modal for Report Description */
<Modal visible={reportModalVisible} transparent
animationType="slide">
    <View style={styles.modalContainer}>
        <View style={styles.modalContent}>
            <Text style={styles.searchInput}>Enter Report
Details</Text>
            <TextInput
                placeholder="Describe the issue..."
                value={description}
                onChangeText={setDescription}
                style={{
                    borderWidth: 1,
                    borderColor: "black",
                    padding: 8,
                    marginBottom: 10,
                    borderRadius: 5,

```

```

        }
      />
      <Button title="Submit" onPress={submitReport} />
    </View>
  </View>
</Modal>

```

We can see here the Input Modal for the Report Description and how the Modal is set up. It has values of description and we see how this changes on text, updating the description variable. You can see the Button function being implemented here, where onPress triggers the submitReport function I mentioned above.

```

const [etaValue, setEtaValue] = useState("..."); // Initially show
"..."

```

Here we are initialising the values of etaValue and defaulting it to “...” so it can present “...” when no route is selected.

```

const handleCalculateRoute = (id) => {
  const cNodeID = (findClosestNodeId(userLocation, nodes))
  goToPlace(nodes[cNodeID])
  console.log(cNodeID)
  const path = findPath(cNodeID, id); // Example: from node 1 to
node 4
  const coordinates = path.map((id) => {
    const node = nodes.find((n) => n.id === id);
    return { latitude: node.latitude, longitude: node.longitude
  };
  });
  setPathCoordinates(coordinates);
  // Calculate the total weight of the path
  let totalWeight = 0;
  for (let i = 0; i < path.length - 1; i++) {
    const fromNodeId = path[i];
    const toNodeId = path[i + 1];

    // Find the edge between these two nodes
    const edge = edges.find(
      (e) => (e.from === fromNodeId && e.to === toNodeId) ||
(e.from === toNodeId && e.to === fromNodeId)
    );
    if (edge) {
      totalWeight += edge.weight;
    }
  }
  setTotalWeight(totalWeight);
}

```

```

    );

    if (edge) {
        totalWeight += edge.weight;
    }
}

console.log(`Total weight of the path: ${totalWeight}`);
const scaleFactor = 924;

// Calculate ETA in seconds
const etaValue = Math.round(totalWeight * scaleFactor);

console.log(`ETA (in seconds): ${etaValue}`);
// Set the etaValue to the calculated ETA (in seconds)
setEtaValue(`${etaValue} Seconds`);

// Rotate the map towards the target node
setTimeout(() => {
    rotateMapToNode(nodes[cNodeID], nodes[id]);

    // Delay second rotation after first completes
    setTimeout(() => {
        rotateMapToNode2(nodes[cNodeID], nodes[id]);
    }, 1500);
}, 1000);
};

```

I have updated the handleRoute function, just by adding a function which adds the weights of the path and defines it to totalWeight and then this is multiplied by a scale factor and rounded to integer, which is then presented and set to etaValue.

The scale factor was determined by using a known distance and the time it takes to get there usually.

```

<View style={styles.ETAWrap}>
    <Text style={styles.Label}>ETA</Text>
    <View style={styles.ETABox}>
        {/* Display the ETA value or "..." if no route

```

	<pre>selected */> <Text style={styles.ETA}>{etaValue}</Text> </View> </View></pre>
	This is the implementation from the function: we can see how now the ETA now presents etaValue instead of null.
Description:	The ETA function calculates travel time by summing path weights and applying a scale factor, ensuring a meaningful estimate is displayed instead of null. The value updates dynamically and integrates smoothly into the UI. The Report button enables users to tap the map (when ReportMode is active), triggering a modal where they can enter issue details. The Submit button then processes the input via submitReport, ensuring efficient report handling. Both functions enhance user interaction by providing real-time feedback and intuitive controls.

Revised Test Plans:

Test Table 28 – Add Commenting/Report Feature for Users and Adding ETA System

Test ID	Test Description	Test Method/Data	Expected Result	Actual Result	Successful ?
2.44	Routes' ETA proportional to Distance	Select a route and note down ETA time, then choose a route which is shorter and note ETA Time, and choose a route which is longer and note the ETA Time	The shorter route ETA Time should be lower than the initial route and the longer route ETA Time should be greater than the initial route.	When testing with example locations e.g. Lecture Theatre and Boys Toilet, the ETA Value is proportional to time and is a reasonable value.	Yes, the ETA is correctly calculated.

2.45	ETA value should be a reasonable time estimate for the route.	Find a route and check if the ETA Value adjusts accordingly and is realistic, and time is with an actual test.	The ETA Value should be accurate and it should be reasonable, it is relative and proportional to the distance.	The ETA Value is appropriate, we used Lecture Theatre as an example and it shows 20 seconds, which is a reasonable ETA from the Common Room.	Yes, the ETA is correctly calculated.
2.46	The ETA Value should be clear and visible and the value/unit should be clear and appropriate.	Visual check for the ETA Value and unit. User's should understand and infer the unit and meaning.	The ETA Value should be bold/clear in any way to the user. The unit should be defined either, minutes, seconds or hours.	The ETA Value is clear and bold and in the center of the ETA box. The unit is by default seconds, this is shown in a complementing bold font.	Yes, the ETA is an appropriate unit.
2.47	Search Box should function properly within Search Location Pop-Up	Press the Search Box button.	Keyboard should be presented to type in location.	The keyboard is presented and users can type in their locations as desired.	Yes, the Search Box functions as expected.
2.48	Search Box should function properly within Search Location Pop-Up v2 - VALIDATION	Press Search Box and then type in a valid location	Direct to School Map with a route and ETA.	The ETA is now presented alongside the route, and only works with a valid location.	Yes, ETA and route are both presented.

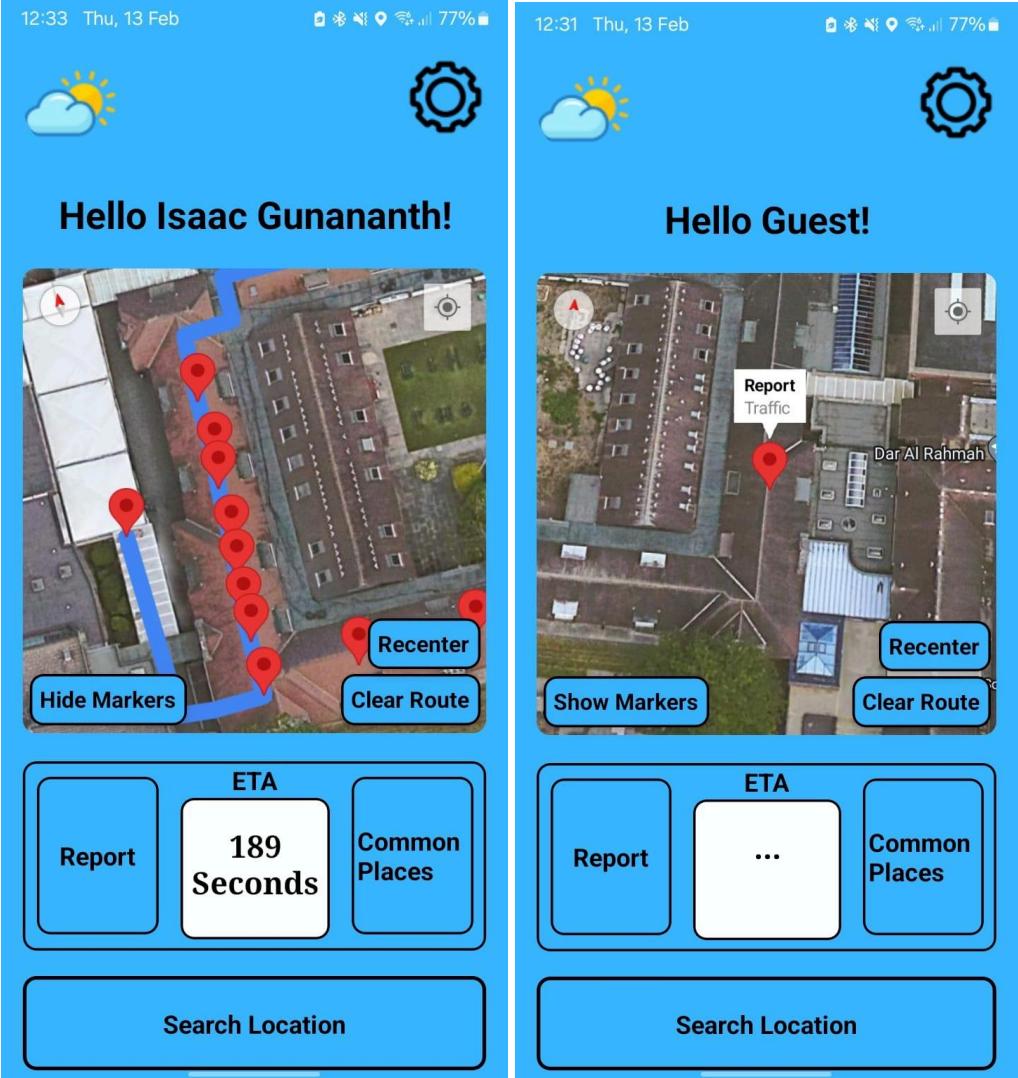
2.49	Search Box should function properly within Search Location Pop-Up v3 - VALIDATION	Press search Box and then type in an Invalid Location	Error/message to say Location not found, perhaps “input” is not correctly typed.	No ETA is presented alongside, and with invalid location, no locations can be pressed.	Yes, no route nor an ETA is presented when invalid location is input.
2.50	Report/Comment button, when pressed should allow the user to click on a point on a map to place a marker and type in its description.	Press and attempt placing markers on various different locations and then, try to type in different inputs and see if they get saved and remain on map.	The report button should work and when pressed allows me to place a marker on the map and type in the label, it saves and I can see it as I traverse through the map.	The report button when pressed allows users to select a place on the map to place a Report and type a description accordingly.	Yes, users can report as expected.
2.51	The Report Marker is clear and visible to users.	Visual check for the Report Marker once placed. Zoom out and in to see if it's constant. Visually check for the description.	The Report should be clear and prevalent to users. It should pop out of the page and the description readable to all.	The description is clear and visible, the report is of bright red marker colour and can be interacted with.	Yes, reports are visible to users.
2.52	Users can Type and enter their suggestions and reports, and it will be presented and readable by users, on the marker.	Place Report and type in description/suggestion and see if it is presented on the report.	The Report has the appropriate description when initialised and is present and visually detectable.	The description is relevant and appropriate, it matches with whatever the user types in and can be readable by users on the	Yes, their reports are readable on the map.

				marker.	
--	--	--	--	---------	--

I have removed these Test IDs, due to time restrictions, which I will expand on in the Evaluation section of this project:

Test ID	Test Description	Test Method/Data	Expected Result
2.53	The Weather Information Button should open up Weather Information Pop-Up	Press the weather information button	The weather information screen should pop-up on top of the Main-Menu Screen
2.54	Weather Information should be accurate.	Look at the details provided in the weather information page, cross reference with other online resources e.g. Google Weather.	The weather should match and correspond to the online stating weather.
3.21	ETA Measure Button should function accordingly v2	Press the ETA Measure button and choose Hours, select a route to specific location.	The route should be calculated and the ETA should be in Hours (to the nearest 10 minutes).
3.22	ETA Measure Button should function accordingly v3	Press the ETA Measure button and choose Minute, select a route to specific location.	The route should be calculated and the ETA should be in Minutes (to the Nearest Minute).
3.23	ETA Measure Button should function accordingly v4	Press the ETA Measure button and choose Seconds, select a route to a specific location.	The route should be calculated and the ETA should be in Seconds, (to the nearest Minute)

Evidence Table 28

Test ID	Screenshot/Video
2.44 - 2.49	<p>Narrated Video:</p> <p>Going through the addition of the ETA Measure:</p> <p>https://youtu.be/APB_MxDax-I</p>
2.50 - 2.52	<p>Report Function:</p> <p>https://youtube.com/shorts/pNB6-GnrRcs</p> 

Evaluation:	<p>The Narrated Videos uncovers how the solution meets the Test IDs from 2.44 to 2.49 and 2.50 to 2.52 appropriately.</p> <p>We can see in the screenshots the visibility, how the program meets this.</p> <p>This test focuses on implementing the main menu, adding a commenting and reporting feature, and integrating the ETA system.</p> <p>For test ID 2.44, the goal is to verify that ETA values are proportional to the route distance. Initially, the system displays ellipses when no route is selected. The test involves selecting different routes and comparing their ETA values. The results confirm that shorter routes have lower ETA values, while longer routes have higher ones. For example, the route to the lecture theater has an ETA of 20 seconds, while a longer route to the boys' toilet has an ETA of 131 seconds. A mid-length route to the gym shows 49 seconds, confirming that ETA values scale appropriately with distance.</p> <p>Test ID 2.45 ensures that the ETA values are reasonable. The lecture theater route, serving as a base reference, consistently displays 20 seconds, matching the expected straight-line travel time. The same scaling factor is applied to all other routes, aligning with the design and analysis phase.</p> <p>Test ID 2.46 checks the clarity and visibility of ETA values, ensuring that the displayed time is bold, clear, and properly labeled with the unit of measurement. The test confirms that ETA values are both visible and appropriately formatted.</p> <p>Test ID 2.47 verifies that the search box functions correctly within the location search popup. The system successfully processes valid search terms, presenting the corresponding locations and their ETA values.</p> <p>Test ID 2.48 focuses on input validation, ensuring that only valid locations return an ETA and a route. For example, searching for "Sports Hall" successfully generates an ETA and route.</p> <p>Test ID 2.49 confirms that invalid locations, such as "Canteen," do not trigger any response, meaning the system correctly ignores invalid inputs while still displaying results for valid locations like "Dining Hall," which shows an ETA of 189 seconds.</p> <p>With these tests completed, the next phase involves testing the commenting and reporting feature.</p> <p>For test ID 2.50, the goal is to ensure that the report comment button allows users to click on a point on the map, place a marker, and type a</p>
-------------	---

description. The test confirms that when a user clicks a location (e.g., a congested area), they can place a report marker and type "traffic." The marker remains visible even when classrooms are hidden, and users can clear routes while still seeing their reports. The report is also accessible and interactive.

For test ID 2.51, the focus is on visibility and readability of report markers. A user successfully marks the Dining Hall and reports "Food is out of stock." The system correctly displays this report, making it visible to other users, ensuring effective communication about real-time school conditions. The test confirms that report markers remain clear, readable, and functional.

This concludes the validation of the commenting and reporting feature.

Post Development Testing

Review of Iterative Development Tests

This is a summary of the Test Plans in the Designs section and its progress within the iterative development process. With cross reference to the success criteria, this also presents the success I faced and also the possible improvements to these Test IDs and their functions.

Test Description + ID + Success Criteria	Successful?	Relevant Test Table	Possible Improvements
Login-In/Sign-Up Page			
Application should run (Test ID 1.1 S.C. N/A)	Yes, the application runs!	Test Table 1	N/A
Should display background image/colour (Test ID 1.2 S.C.15 and SC.16)	Yes, background colour is displayed as the Designs section suggests.	Test Table 1	N/A
The Log-in page should be the first page for new users who haven't yet logged-in/created account as stated in SC.06 (Test ID 1.3 S.C.06)	Yes, the Log-In page is the first page for new users.	Test Table 1	N/A
Should display Logo clearly on top of background as a png as stated in SC.15 (Test ID 1.4 S.C.15)	Yes, The Logo is presented as the Designs section requires, no blocking, it now adapts to the user display.	Test Table 2	N/A
Beneath the Logo, should appear Title for Sign-In (Test ID 1.5 S.C. 15)	Yes, Title is positioned and displayed correctly.	Test Table 1	N/A

Add a subtitle below the title to greet the users and inform them of the application (Test ID 1.6 S.C. 16)	Yes, greeting is presented and formatted correctly.	Test Table 1	N/A
The input for the email address should be clear and obvious to the user (Test ID 1.17 S.C. N/A)	Yes, the placeholders indicate the email address input clearly.	Test Table 1	N/A
The input for the password should be clear and obvious to the user (Test ID 1.8 S.C. N/A)	Yes, the placeholders indicate the password input clearly.	Test Table 1	N/A
Text should be able to be input into email address and password (Test ID 1.9 S.C. 18)	Yes, Email address and Password Inputs work!	Test Table 3	N/A
Text should be encrypted when writing in the password field (Test ID 1.10 S.C. N/A)	Yes, the test for the password is now hidden by default to preserve privacy.	Test Table 9	N/A
Screen should have matching tone and clear theme (Test ID 1.11 S.C.15 and S.C.16)	Yes, there is a clear theme in the app.	Test Table 4	Perhaps implement the theme selector, so users can customise the theme and tone of Screens.
Sign-In button should be prevalent and clear to the user (Test ID 1.12 S.C. 16)	Yes, the Sign-In button can now be accessed at all times and all places in the application.	Test Table 5	N/A
The Sign-In button should be intractable with user (Test ID 1.13	Yes, the Sign-In button is interactable.	Test Table 6	N/A

S.C.16)			
Option to Sign-Up should be present and button should be pressable (Test ID 1.14 S.C. N/A)	Yes, the Sign-Up button is displayed and interactable.	Test Table 6	N/A
If the Sign-In page is pressed without any input for either the email or password fields then the screen should output appropriate error messages to highlight the issue. (Test ID 1.15 – VALIDATION S.C. 06)	Yes, an appropriate error message is displayed.	Test Table 8	N/A
Users should have the option to Sign-In as a guest since not all users would want to Sign-In/Sign-Up. This should be prevalent and clear to the user. (Test ID 1.16 S.C. 16)	Yes, the Guest option is present.	Test Table 9	N/A
Users should be able to toggle between visible password fields and encrypted fields. (Test ID 1.17 S.C. N/A)	Yes, users can now toggle between the encrypted and visible fields.	Test Table 10	N/A
Users should be able to identify Register Screen/Sign-Up Page through a clear Title. (Test ID 1.18 S.C. 19 and S.C.16)	Yes, Register Screen's purpose is indicative.	Test Table 11	N/A
Field for name should be implemented to provide customisation options. (Test ID 1.19 S.C. 17)	Yes, the name field is there.	Test Table 11	N/A
Fields for Email Address and Password should be present and their purpose should be clear using relevant titles. (Test ID 1.20	Yes, email and password fields are present.	Test Table 11	N/A

S.C. 06)			
Field for Confirm Password should function and be present (Test ID 1.21 S.C. 06)	Yes, confirm password works as desired.	Test Table 11	N/A
The register button should be present and pressable. (Test ID 1.22 S.C. 16)	Yes, the register button is interactable and is present.	Test Table 11	N/A
Option to return back to Sign-In/Guest page (Test ID 1.23 S.C. 19)	Yes, The option to return to the Sign-In page is present.	Test Table 12	N/A
If the Register Button is pressed without any input for either the name, email or password fields then the screen should output appropriate error messages to highlight the issue. (Test ID 1.24 - VALIDATION S.C. 06)	Yes, there is appropriate validation for the inputs.	Test Table 13	N/A
If the Register Button was pressed without the Password field and the Confirm Password field not matching, appropriate error messages should be outputted. (Test ID 1.25 - VALIDATION S.C.06)	Yes, there is appropriate validation for the inputs.	Test Table 13	N/A
The Register Button should clear all values of the fields to formulate a clean environment to compliment any error messages. (Test ID 1.26 S.C. 16)	Yes, the register button clears the fields once registered.	Test Table 14	N/A
Navigation between different screens. "Sign-Up" button should navigate to the Register Screen etc. (Test ID 1.27	Yes, the navigation works as expected.	Test Table 16	Smooth animation when navigating between

S.C. 19)			screens.
The Guest Button should function and be pressable. It should take users directly to the Main Menu and set the name to “Guest”. (Test ID 1.28 S.C. 19 and S.C.16)	Yes, the Guest button directs users to the Main Menu and the name is set to Guest.	Test Table 16	N/A
Users should be able to navigate back to the Login Page from the Register Screen. (Test ID 1.29 S.C. 19)	Yes, users can navigate back to the Register Screen.	Test Table 17	N/A
User authentication with stored credentials (Test ID 1.30 S.C. 06 and S.C.17)	Yes, the user is able to login effectively.	Test Table 15	Add option for Google Sign-In.
Store and retrieve user data securely (Test ID 1.31 S.C. 06)	Yes, the user can then access that data as expected.	Test Table 16	N/A
Error handling for unregistered users (Test ID 1.32 – VALIDATION S.C. 06)	Yes, unregistered users experience an error message.	Test Table 16	N/A
Navigation to Main Menu after successful login (Test ID 1.33 S.C. 19)	Yes	Test Table 18	N/A
Error message display for failed login (Test ID 1.34 S.C. N/A)	Yes, failed logins result in error messages.	Test Table 16	N/A
Clear feedback for successful login (Test ID 1.35 S.C. N/A)	Yes, the user is taken to the Main Menu when successfully logged in.	Test Table 16	N/A

Incorrect email formats should throw error messages - Validation (Test ID 1.36 -VALIDATION S.C. 06)	Yes, there is appropriate validation for the inputs.	Test Table 16	N/A
Main Menu			
Main Menu Page should be initialised and set up to take in name from database/other screens. (Test ID 2.1 S.C. 17)	Yes, the Main Menu takes in the correct name for the greeting.	Test Table 18	N/A
The Weather and Settings sections should appear side by side at the top of the screen. (Test ID 2.2 S.C. 16)	Yes, the icons are present.	Test Table 18	N/A
The Weather section should be clear in purpose and allow user interaction. (Test ID 2.3 S.C. 08)	Yes, the weather icons are clear in their purpose.	Test Table 18	N/A
The Settings section should be clear in purpose and allow user interaction. (Test ID 2.4 S.C. 03)	Yes, the settings icon is clear in its purpose.	Test Table 18	N/A
The title for the "Greeting" section should be visible and clearly indicate its purpose. (Test ID 2.5 S.C. N/A)	Yes, the greeting is displayed and clear.	Test Table 18	N/A
The layout should maintain a professional and user-friendly appearance. (Test ID 2.6 S.C. 16)	Yes, there is a professional tone to the application.	Test Table 18	N/A
Navigation back to the Login Page should work correctly. (Test ID 2.7)	Yes, users can navigate to the Login Page	Test Table 18	N/A

S.C. 19)	successfully.		
The Features for Report and Common Places should be present and pressable. (Test ID 2.8 S.C. 07 and S.C.09)	Yes, the features are present.	Test Table 18	N/A
The ETA should be a prominent section which is not pressable but is to present values. (Test ID 2.9 S.C. 04)	Yes, the ETA is present and not pressable.	Test Table 18	N/A
The Search Location button should be visual and prominent on screen. (Test ID 2.10 S.C. N/A)	Yes, the search location feature is present!	Test Table 18	N/A
Retrieve name from Register Screen and display on Main Menu (Test ID 2.11 S.C. 19)	Yes, the name is successfully retrieved to the Main Menu.	Test Table 19	N/A
Verify that the map displays the user's current location correctly. (Test ID 2.12 S.C. 10)	Yes, the map displays the accurate location.	Test Table 20	N/A
Verify that the recenter button works as intended. (Test ID 2.13 S.C.10)	Yes, the recenter buttons do work as intended.	Test Table 20	N/A
Verify the map's zoom level is appropriate for the current location. (Test ID 2.14 S.C. 10)	Yes, the map is presentable.	Test Table 20	N/A
Verify that the map type is set correctly. (Test ID 2.15 S.C.10)	Yes, the map type is correct.	Test Table 20	Better satellite imagery of school specifically.
Verify the correct handling of location permissions.	Yes, location permissions are	Test Table 20	N/A

(Test ID 2.16 S.C.N/A)	handled appropriately.		
Verify that the map is responsive and interactive. (Test ID 2.17 S.C. 01)	Yes, the map can be maneuvered around.	Test Table 20	N/A
Verify that the app adjusts the map display based on the device's screen size. (Test ID 2.18 S.C. 16)	Yes, the app does change and adjust depending on the screen size.	Test Table 20	N/A
Verify the user's location updates dynamically as they move. (Test ID 2.19 S.C. 01)	Yes, the user location updates as they move.	Test Table 21	N/A
Reporting Feature Button (Test ID 2.20 S.C.07)	Yes, the button is displayed on the page.	Test Table 22	N/A
The ETA section should be displayed (Test ID 2.21 S.C. 04)	Yes, the feature is displayed on the page.	Test Table 22	N/A
ETA is not interactable, presenting a value alone. (Test ID 2.22 S.C. 04)	Yes, it is not interactable.	Test Table 22	N/A
Users can view the map interface with markers for all mapped classroom locations. (Test ID 2.23 S.C. 02)	Yes, the user can look around the map and interact with the classroom locations.	Test Table 23	N/A
The Search Location button is visible and functional. (Test ID 2.24 S.C. N/A)	Yes, the button is displayed on the page.	Test Table 23	N/A
Modal displays a list of locations matching the search query. (Test ID 2.25	Yes, the modal is displayed correctly.	Test Table 22	N/A

S.C. 02)			
Selecting a location from the search results zooms in on the selected location and after a short period of time, returns to the current location. (Test ID 2.26 S.C. 02 and S.C.01)	Yes, selecting a location from the search location works as expected.	Test Table 22	N/A
Users can dismiss the search modal if no selection is needed. (Test ID 2.27 S.C. N/A)	Yes, users can escape the modal.	Test Table 22	N/A
Testing the route generation functionality (Test ID 2.28 S.C. 20)	Yes, the route is generated successfully.	Test Table 25	N/A
Ensuring the route line is vivid and visible (Test ID 2.29 S.C.20)	Yes, you can vividly see the route line. Yes	Test Table 24	N/A
Testing route update based on user movement (Test ID 2.30 S.C. 01)	Yes, when the user moves around the marker representing their location changes.	Test Table 25	N/A
Testing automatic panning aesthetically and smoothly to the destination and closest node (Test ID 2.31 S.C. 16)	Yes, aesthetic panning precedes the generation of the route on screen.	Test Table 25	N/A
Testing the calculation of the shortest path (Test ID 2.32 S.C. 21)	Yes, the shortest path is presented.	Test Table 24	N/A
Ensuring user feedback on path calculation status (Test ID 2.33 S.C. 21)	Yes, the user will know when the calculation is complete and the route is found.	Test Table 25	N/A

Button to Clear Route should be visible and prevalent and functional (Test ID 2.34 S.C. N/A)	Yes, the clear route button is present and functional.	Test Table 25	N/A
Route resets when the reset option is selected (Test ID 2.35 S.C. N/A)	Yes, the route is cleared when pressed.	Test Table 25	N/A
Resetting the route does not affect user location or school marker (Test ID 2.36 S.C. N/A)	Yes, the button only works as intended, and doesn't interfere with anything else.	Test Table 25	N/A
If the user's distance from school exceeds a threshold, prompt Google Maps redirection (Test ID 2.37 S.C. N/A)	Yes, any distance too far, the user will be routed to the School via Google Maps.	Test Table 25	Embed route into the actual app, instead of redirecting.
Google Maps opens with correct school destination (Test ID 2.38 S.C. N/A)	Yes, linked to a correct location.	Test Table 25	N/A
The Common Places button is pressable and triggers the modal. (Test ID 2.39 S.C. 09)	Yes, it does trigger a modal.	Test Table 25	N/A
Common Places modal has a list of preset locations (Test ID 2.40 S.C. 09)	Yes, preset common locations are presented.	Test Table 25	N/A
Selecting a place closes the modal. (Test ID 2.41 S.C. 01)	Yes, selecting a place does close a modal.	Test Table 25	N/A
The new route is visually displayed on the screen. (Test ID 2.42 S.C. 20)	Yes, the route is displayed.	Test Table 25	N/A

Close button in the modal is visible and pressable. (Test ID 2.43 S.C. 09)	Yes, the close button for the modal is interactable.	Test Table 25	N/A
Routes' ETA proportional to Distance (Test ID 2.44 S.C. 04)	Yes, the ETA is correctly calculated.	Test Table 28	N/A
ETA value should be a reasonable time estimate for the route. (Test ID 2.45 S.C. 04)	Yes, the ETA is correctly calculated.	Test Table 28	N/A
The ETA Value should be clear and visible and the value/unit should be clear and appropriate (Test ID 2.46 S.C. 06)	Yes, the ETA is an appropriate unit.	Test Table 28	N/A
Search Box should function properly within Search Location Pop-Up (Test ID 2.47 S.C. 01)	Yes, the Search Box functions as expected.	Test Table 28	N/A
Search Box should function properly within Search Location Pop-Up v2 - VALIDATION (Test ID 2.48 S.C. 01)	Yes, ETA and route are both presented.	Test Table 28	N/A
Search Box should function properly within Search Location Pop-Up v3 - VALIDATION (Test ID 2.49 S.C. 01)	Yes, no route nor an ETA is presented when invalid location is input.	Test Table 28	N/A
Report/Comment button, when pressed should allow the user to click on a point on a map to place a marker and type in its description. (Test ID 2.50 S.C. 07)	Yes, users can report as expected.	Test Table 28	N/A

The Report Marker is clear and visible to users. (Test ID 2.51 S.C. 07)	Yes, reports are visible to users.	Test Table 28	Add an information icon with description to explain this functionality.
Users can Type and enter their suggestions and reports, and it will be presented and readable by users, on the marker. (Test ID 2.52 S.C. 07)	Yes, their reports are readable on the map.	Test Table 28	N/A
Settings Page			
Settings Page Label should be displayed (Test ID 3.1 S.C. 03)	Yes, the settings page title is displayed.	Test Table 26	N/A
Background image should be displayed (Test ID 3.2 S.C. 15)	Yes, the background image is displayed consistently.	Test Table 26	N/A
School Information Button should be displayed (Test ID 3.3 S.C. 03)	Yes, the button is displayed.	Test Table 26	N/A
Personal Information Button should be displayed (Test ID 3.4 S.C. 03)	Yes, the button is displayed.	Test Table 26	N/A
Additional Settings Button should be displayed (Test ID 3.5 S.C. 03)	Yes, the button is displayed.	Test Table 26	N/A
School Information Button should function to open a drop-down menu (Test ID 3.6 S.C. 03)	Yes, the button opens a drop-down of schools.	Test Table 26	Perhaps improve to include more schools.

Personal Information Button should function to open a drop-down menu (Test ID 3.7 S.C. 05)	Yes, the button opens a drop-down.	Test Table 26	N/A
Additional Settings Button should function to open a drop-down menu (Test ID 3.8 S.C.05)	Yes, the button opens a drop-down.	Test Table 26	N/A
Settings Page should have a functioning scroll feature (Test ID 3.9 S.C. N/A)	Yes, the settings has a scroll feature to be able to find all the buttons.	Test Table 26	N/A
School Name Button should function properly (Test ID 3.10 S.C. 05)	Yes, the Select School button opens a list of schools for selection.	Test Table 26	N/A
Form Room Input should function properly (Test ID 3.11 S.C. 05)	Yes, this button functions expectedly.	Test Table 26	N/A
Name Input should function properly (Test ID 3.12 S.C. 05)	Yes, users can input their names successfully.	Test Table 27	N/A
The name field should have proper validation to prevent any invalid inputs that are not of form name. (Test ID 3.13 – VALIDATION S.C. 05)	Yes, there is appropriate validation.	Test Table 27	N/A
Accessibility Needs Toggle should function properly (Test ID 3.14 S.C. 05)	Yes, this button functions expectedly.	Test Table 26	N/A
The ETA Measure Button should function properly (Test ID 3.15	Yes, this button functions expectedly.	Test Table 26	N/A

S.C. 04)			
Form Room Button should function accordingly v2 (Test ID 3.16 S.C. 05)	Yes, this button functions expectedly.	Test Table 26	N/A
The Theme selector should be functional and working as expected, clear and relevant toggle to switch between themes. (Test ID 3.17 S.C. 05)	Yes, the return to Main Menu option is present.	Test Table 26	N/A
The button to return to the Main Menu should function and be visible to users. (Test ID 3.18 S.C. 19)	Yes, The input allows the name to be adjusted and presented on the Main Menu.	Test Table 26	N/A
Accessibility Needs Button should function accordingly v2 (Test ID 3.19 S.C. 19)	No, due to time-restrictions and lack of resources, I wasn't able to complete these Test Plans.	N/A	N/A
Accessibility Needs Button should function accordingly v3 (Test ID 3.20 S.C. 05)	No	N/A	N/A
ETA Measure Button should function accordingly v2 (Test ID 3.21 S.C. 04)	No	N/A	N/A
ETA Measure Button should function accordingly v3 (Test ID 3.22 S.C. 04)	No	N/A	N/A
ETA Measure Button should function accordingly v4 (Test ID 3.23 S.C. 04)	No	N/A	N/A

Robustness/Functionality Testing

Robustness/Functionality Testing here aims to resolve the Post-Development Testing in the perspective of the user. To ensure the program runs in every circumstance smoothly and appropriately. The vitality of this section is due to the practical and final stage of the testing process, where we can now test the entire app and assess both its robustness and functionality. This phase ensures that the app performs well under various real-world conditions, including unexpected user inputs, system failures, and resource limitations. It is crucial for identifying potential issues that could impact user experience or overall system performance, guaranteeing reliability and stability before the app's release.

Test ID	Test Description	Test Method/Data	Expected Result	Successful?
R1	<p>At the initial start of the application, the user should be prompted to Login/Sign-up.</p> <p>This function should work effectively and settings/searches within the application should be saved into account details.</p>	<p>Login/Sign-up Page should be the first page on initial start-up.</p> <p>I should be able to Log-in to the application and save my account settings to that account, which I can access if I log-out and log-back in.</p>	<p>The Login/Sign-up page appears immediately when the application is launched for the first time. No other pages should be accessible until login/sign-up is completed.</p>	<p>Yes, when the application starts, new users are prompted to Login/Sign-up. Their name and account details are saved. This is important as it deems the success of SC.06</p>
R2	<p>After logged-in, subsequent program launches should open in Main-Menu</p>	<p>When a user logs in, they should be directed to the Main Menu Page.</p> <p>After relaunching the application, users should be directed to Main-Menu straight away.</p>	<p>After the initial login, the next time the application is launched, it bypasses the login screen and opens directly to the Main Menu unless manually logged out.</p>	<p>Yes, meeting SC.19 the user is directed to the Main Menu after Logging in.</p>

R3	The account user is on should not reset at any point.	For every launch of the application after the initial launch (where the user logs in/creates an account) the application should not log out and continue on that account.	The user remains logged into their account unless they explicitly log out. Unexpected logouts or resets do not occur.	Yes, the account user does not reset and stays constant e.g. set named, in the process, meeting SC.03 and SC.05
R4	The settings of user should not change/reset once application is relaunched	The settings should be linked to the user account and since the account should be constant, unless logged out manually, the settings should be constant and not reset.	User preferences (e.g., language, theme, navigation settings) remain unchanged after restarting the app.	Yes, relaunching the application does not affect settings, again meeting SC.03 and SC.05
R5	Reported features within the School Map should be constant, until time-limit expires	The reported features, reported by users should appear and be constant, even if application resets, it should be updated to the school map online and be reset after 24 hours.	Any reports submitted by users (e.g., broken paths, closed areas) remain visible and unchanged until their expiration time (e.g., 24 hours) is reached.	Yes, the report features do not get removed (SC.07) when the user resets and stays constant for a set time-period .
R6	Routes should reset/stop if application is reset	If the user escapes the application and closes it whilst a route was running/calculated, the application should cancel the application and flush any routes.	If the app is closed while a route is in progress, reopening the app does not resume the previous route. Users must input	Yes, when the application is reset the routes are reset and cleared. (SC.16)



Evidence of Robustness/Functionality Testing Working:

Test ID	Evidence	Evaluation
R1	<p>https://youtube.com/shorts/TxX7uiwONuo?feature=share</p> 	<p>This test aims to evaluate the app's ability to function correctly under various conditions. Upon initialization, the app successfully launches and directs the user to the login/sign-up page, ensuring that the startup process proceeds as expected.</p> <p>This meets the success criteria SC.06</p> <p>The Login/Sign-up page should be the first screen for the users of the application, but the user should have the ability to use the app as a guest and should have appropriate validation.</p> <p>This screenshot is the first page for new users.</p>
R2	<p>https://youtube.com/shorts/7t_bBu6OCpg</p>	<p>The robustness test verifies R1 (app prompts the login page on launch) and R2 (user remains logged in after re-entering credentials). After registering and logging in, user details persist correctly, even after restarting the app. The next step involves testing the report function.</p>

		<p>The consistency of the details and how users are directed to the Main Menu meets the SC.19</p> <p>The different pages should be easy to navigate through.</p>
R3	https://youtube.com/shorts/7t_bBu6OCpg 	<p>During the test, after registering the account with the password "123", the user logs in and is correctly greeted with their name, "Isaac", at the main menu. To verify R3, the app is restarted, and the user logs in again using the same credentials. The test confirms that my name and settings remain unchanged, proving that the account has not been reset.</p> <p>Additionally, before restarting, the user checks if they can exit and return to the app without losing progress. The app successfully resumes from its last known state, further confirming that user data is retained. When the app is restarted completely, it still directs the user to the login page as expected, and upon logging in again, the saved user settings (name, preferences) are restored.</p> <p>Since the Settings are saved and the details are all saved in this robustness test it meets aspects of SC.03</p> <p>Settings should allow updating and meet all different user needs.</p> <p>and SC.05</p>

		The application must be responsive to certain requirements/details the user may have.
R4	<p>https://youtube.com/shorts/7t_bBu6OCpg</p> 	<p>Further testing ensures that the application's settings and user data remain unchanged after relaunching. The app is restarted, and upon reopening, it correctly returns to the login page. After logging in again, the previously saved user details are retained, confirming that the system does not reset user accounts unnecessarily. This satisfies R4, ensuring that user settings remain constant until an explicit reset or a time limit expiration.</p> <p>Since the Settings are saved and the details are all saved in this robustness test it meets aspects of SC.03</p> <p>Settings should allow updating and meet all different user needs.</p> <p>and SC.05</p> <p>The application must be responsive to certain requirements/details the user may have.</p>
R5	<p>https://youtube.com/shorts/pjSAP5yc3G8</p>	<p>The test verifies R5, confirming that reports remain unchanged until a predefined time limit expires. The "chairs are broken" report in the main hall remained constant after two minutes, demonstrating that the feature functions correctly.</p>

		<p>The test now focuses on R5, which ensures that reported issues within the school map remain constant until a set time limit expires. This means that once a report is submitted, it should persist without any unintended changes or deletions until the system's predefined expiration period is reached.</p> <p>To test this, a report is submitted in the main hall, describing that "chairs are broken." I then leave the report untouched for approximately two minutes to observe whether any unexpected changes occur.</p> <p>After waiting, the report is checked, and it remains exactly as it was originally submitted. Since the report has neither disappeared nor been altered, R5 is confirmed to be met—the reported features within the school map remain constant as expected.</p> <p>This meets the success criteria of SC.07</p> <p>Reporting feature – to highlight issues and factors that exist on the map and present them to other users.</p>
R6	https://youtube.com/shorts/WRPqnC81rvU	<p>The test verifies R6, confirming that routes reset when the app is restarted. The previous route from main menu to main hall was cleared after reopening,</p>

allowing users to start fresh. A new route to the **dining hall** was then successfully created, ensuring proper functionality.

After logging in, they check whether the previous route is still present. The test confirms that the **route has been reset**, meaning users do not see the old route upon reopening the app. This behavior aligns with the intended functionality, ensuring that each session begins with a fresh state rather than retaining previous navigation paths.

To further confirm R6, the tester selects a new route (to the **dining hall**) and verifies that it works without errors, reinforcing that the reset mechanism functions correctly.

Meeting the success criteria
SC.16

Aesthetically pleasing and functional design to the user which promotes the user experience.

Usability Testing

Usability Testing focuses on evaluating the app from the user's perspective to ensure it is intuitive, user-friendly, and meets their needs effectively. This stage aims to identify any obstacles users might encounter when interacting with SchoolNav, ensuring that navigation, design, and overall user experience are smooth and efficient. The importance of usability testing lies in its ability to highlight areas for improvement, ensuring the app is accessible and easy to use for its target audience. It helps refine the app's interface, reducing frustration and enhancing satisfaction, ultimately improving user adoption and engagement.

Test ID	Test Description	Test Method/Data	Expected Result	Successful?
U1	The application has functional buttons and responsive controls.	Each button should take you to the appropriate page or perform a specific function.	All buttons respond instantly to user interaction, with no delays or incorrect responses. Clicking a button takes the user to the correct page or executes the expected function. Buttons provide visual feedback (e.g., color change, slight animation) when pressed.	Yes, the stakeholders find the buttons are responsive and functional. U1 solely ticks off the completion of SC.01.
U2	Settings should be easily accessible and visible, when in Main Menu Screen	The settings icon should function to present user Settings's Page and it should be apparent to where the setting's can be found, it should be in the top-right corner.	The settings icon is clearly visible and located in the top-right corner as expected. Users can access settings in one click without confusion.	Yes, the stakeholders can find the settings page without any guidance. Through this meeting: SC.19 and SC.03

U3	<p>Easy to understand and self-explanatory sections and buttons</p>	<p>Every button should be clearly labelled, with accurate and descriptive labels which reflect the functionality of the button.</p> <p>Each page should be labelled and described to ensure usability within the application.</p>	<p>Users can identify the purpose of each button without guessing. Buttons have clear labels with concise descriptions of their function. There is no ambiguity or misleading text.</p>	<p>Yes, each button was understood by the stakeholders intuitively.</p> <p>Meeting SC.16</p>
U4	<p>The school map should reflect accurately and descriptively the location of the user and route to take</p>	<p>Clear arrows and routes with highlighted lines across the map corridors.</p> <p>Description and labelling of each location/room user crosses/will cross.</p>	<p>The user's current location is correctly displayed with a clear indicator. The map updates in real-time as the user moves. Routes appear correctly and lead to the intended destinations.</p>	<p>Yes, stakeholders were able to recognise and found the map to be accurate and the routes to be effective.</p> <p>This finalises the completion of SC.02</p>
U5	<p>Each screen should be easy to get to and indicative of its purpose.</p>	<p>Check the buttons which lead to each page to ensure they are easily identifiable and functional.</p> <p>Test the responsiveness of the buttons, ensuring they work across different devices and screen sizes.</p>	<p>Each page should be clear in purpose and the routing to each different page is clear and usable. The user should be able to understand and use the application inherently.</p>	<p>Yes, the stakeholders understood the purpose of each screen and were able to get to the pages they wanted to easily.</p> <p>Completing SC.19</p>

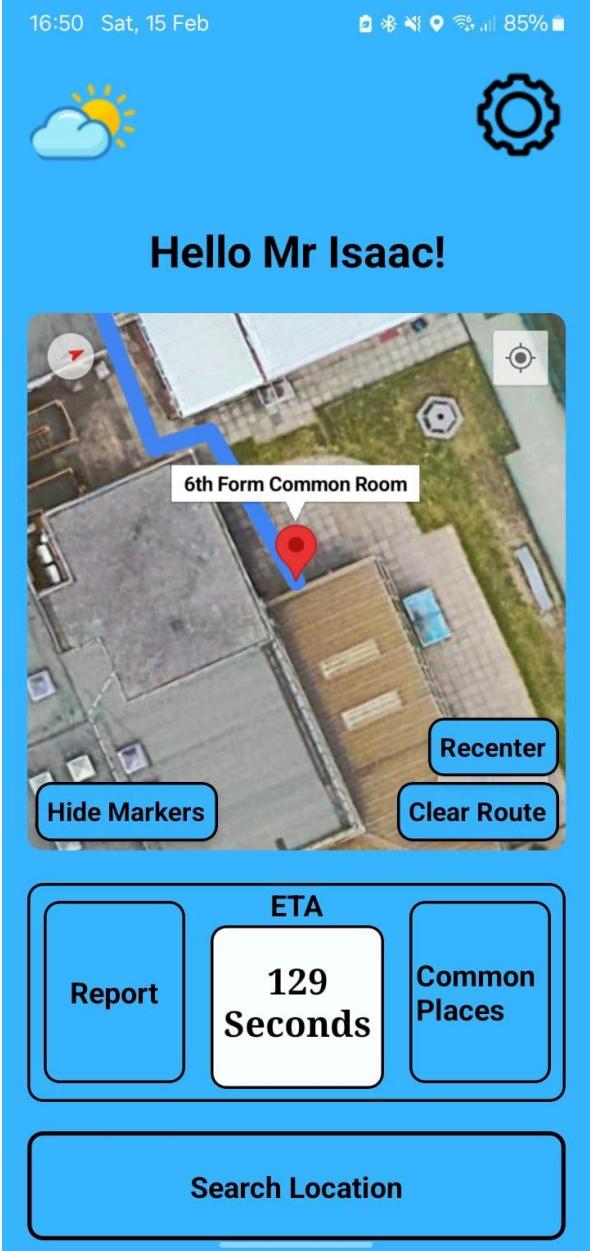
		<p>Test the flow of navigation between pages to ensure it is logical and seamless, with minimal effort required to move from one section to another.</p>		
U6	<p>The routes should be easy to follow and use, to find a route and traverse it efficiently.</p>	<p>Ensure the routes within the app are easy to follow, with clear, step-by-step directions or indicators that guide users seamlessly. Test if users can easily find a route and quickly understand how to traverse it, minimizing any confusion or frustration.</p>	<p>Users should be able to easily locate and select routes without confusion, with intuitive navigation options clearly guiding them to their destination. Routes should be presented in a clear, organized manner, with labels or icons that are easy to understand and follow. The app should display accurate, step-by-step directions or indicators, allowing users to traverse the route without getting lost or needing to backtrack.</p>	<p>Yes, the routes were easy to follow and stakeholders found it efficient to traverse along.</p> <p>Meeting a fundamental Success Criteria:</p> <p>SC.21</p>

Evidence of Usability Testing:

Exploring the evidence for the Usability Testing I have included this walk-through (which serves as evidence of a key stakeholder – a Year 7 student at Seven Kings High School)—using the application for the first time without any help or guidance. Conducted before the final iteration, the walk-through still aligns with the six usability points. Any features the user did not explore are covered in a separate video below the Stakeholder walkthrough. I have provided screenshots where appropriate to also provide further evidence of the usability features.

Test ID	Evidence	Evaluation
U1	<p>This walk-through is evidence of an important stakeholder using the application for the first time with no help or guidance. The stakeholder running the walk-through is a Year 7 Current Student at Seven Kings High School. This walk-through was before the completion of the final iteration but still meets the 6 Usability points. Any features the user missed out I went through in a separate video below.</p> <p>https://youtu.be/94u2H1NJnAM</p> <p>This narrated video is then me going through the application in the perspective of the user for some added evidence.</p> <p>https://youtu.be/2UUiMiUFqmy</p>	<p>The text describes usability testing for the school navigation app. The testing covers various elements: U1 focuses on functional buttons and responsive controls, including buttons like the interactive logo, sign-in, settings, recenter, and show markers. Other tests include selecting a school, changing an account name, and toggling accessibility settings for features like light/dark mode and ETA formatting. The user also tests map interactions, report features (like reporting a broken pull-up bar), and the compass for directional guidance, ensuring interactivity and responsiveness throughout the app.</p> <p>From what the input suggests, the testing covers functionality of buttons, controls, and various interactive features. The user is examining different elements like the age button, settings page, and interactive icons such as the recenter and hide/show markers. It also explores features like search location, school information selection, and customization (such as dark/light mode). The user also addresses small issues, like broken icons</p>

		<p>and testing the compass feature for navigation while confirming the responsiveness of the map. Overall, it's focused on testing user interactivity and accessibility.</p> <p>The screenshot shows me pressing the Logo, and how it is interactable.</p> <p>This Usability Test meets SC.01</p> <p>The application should have a smooth and responsive user interface (UI) system.</p>
U2	<p>This is the stakeholder walkthrough. https://youtu.be/94u2H1NJnAM</p> <p>This narrated video is then me going through the application in the perspective of the user for some added evidence.</p> <p>https://youtube.com/shorts/nv3nfT5nNJM</p>	<p>In this segment of usability testing, the focus is on U2, which evaluates the accessibility and visibility of the settings within the app. The test begins with checking the settings button to ensure it is easily accessible from the main screen. The button is located prominently and is functional, allowing users to open the settings page without difficulty. Once the settings page is accessed, the tester explores</p>



The screenshot shows a mobile application interface with a blue header bar. The top left displays the time "16:50" and date "Sat, 15 Feb". The top right shows battery level "85%" and signal strength. Below the header is a weather icon of a sun and cloud. A gear icon is in the top right corner of the main screen area.

The main content area features a heading "Hello Mr Isaac!" in bold black text. Below it is a map showing a building complex with a red location marker pointing to the "6th Form Common Room". On the map are buttons for "Hide Markers", "Recenter", and "Clear Route".

Below the map is a section with three buttons: "Report", "ETA 129 Seconds", and "Common Places". At the bottom is a large "Search Location" button.

Text on the right:

the options available, confirming that users can make several important changes, such as updating their name and adjusting accessibility settings based on their needs. For example, a user may want to change their name from "Guest" to their personal name, and this can be done easily. Similarly, accessibility settings like changing the display preferences for dark or light mode are also available.

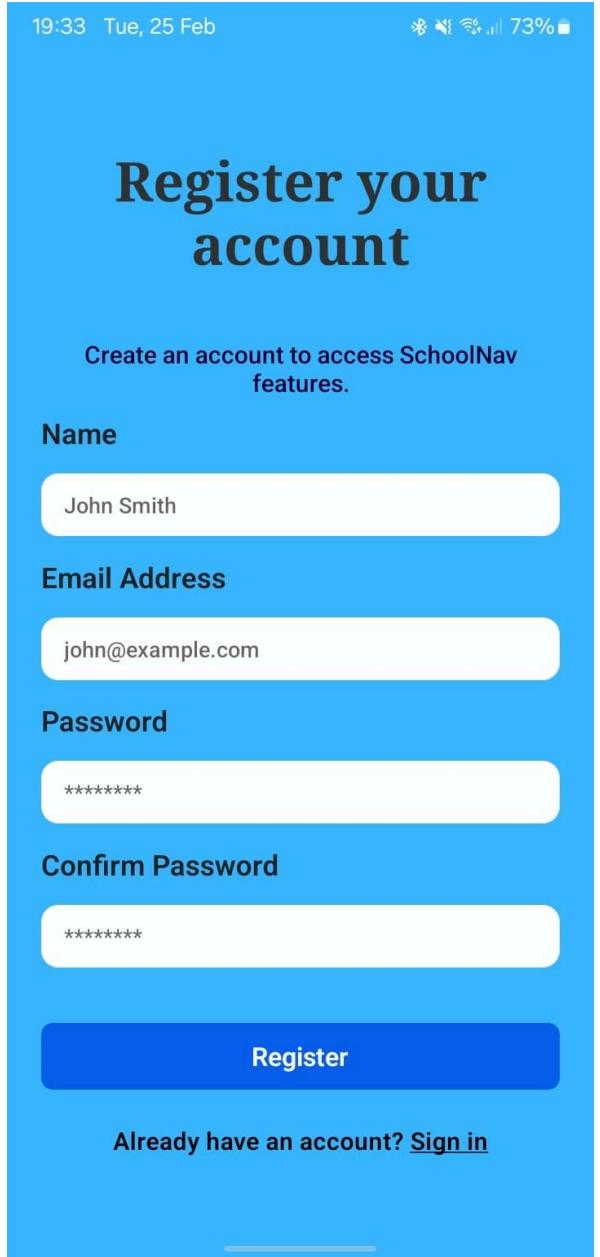
Another key feature tested is the school selection option within the settings. The tester ensures that the user can select their school from a list. If a user needs to update their school, they can select a new one from the available options. However, it is noted that certain schools, such as Brampton, do not yet have map features integrated, which is a limitation. While this does not impact the overall functionality, it's acknowledged as an area for future improvement, and the development team plans to address this limitation in subsequent updates.

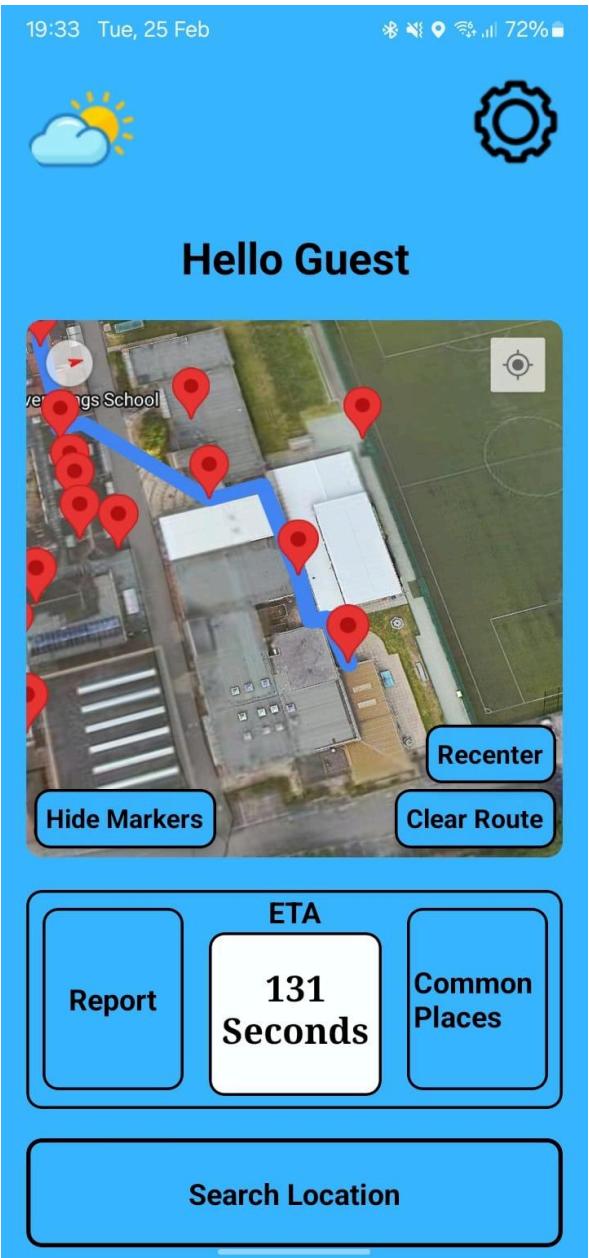
Additionally, the tester verifies that after making any changes in the settings, users can easily return to the main menu. The transition is smooth, ensuring that users do not encounter any navigation issues. This is important for maintaining a seamless experience between different parts of the app. The tester also takes a screenshot to document the result of the test, capturing the settings page with

	<p>16:51 Sat, 15 Feb</p> <p>85%</p> <p>00:48</p> <h2>Settings</h2> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> School Information </div> <div style="background-color: #0070C0; color: white; padding: 5px; border-radius: 5px;"> Select School </div> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> Enter Form Room Number </div> <div style="background-color: #0070C0; color: white; padding: 5px; border-radius: 5px; margin-top: 10px;"> Time-Table Information </div> <div style="border: 1px solid black; padding: 5px; margin-top: 20px;"> Personal Information </div> <div style="background-color: #0070C0; color: white; padding: 5px; border-radius: 5px; margin-top: 5px;"> Name: </div> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> Mr Isaac </div> <div style="background-color: #0070C0; color: white; padding: 5px; border-radius: 5px; margin-top: 10px;"> Accessibility Needs </div> <div style="border: 1px solid black; padding: 5px; margin-top: 20px;"> Additional Settings </div>	<p>all the options visible and fully functional.</p> <p>In conclusion, the usability test confirms that U2 has been successfully met, as users can easily access the settings and make necessary adjustments without encountering issues. While there are minor limitations, such as the absence of map features for some schools, these are noted for future improvements. The overall experience is intuitive and user-friendly, ensuring that users can customize their app settings with ease.</p> <p>From these results we can conclude that these Success Criteria has been met:</p> <p>SC.19 and SC.13</p> <p>The different pages should be easy to navigate through.</p> <p>Settings should allow updating and meet all different user needs</p>
U3	<p>Stakeholder walkthrough: https://youtu.be/94u2H1NJnAM</p> <p>This narrated video is then me going through the application in the perspective of the user for some added evidence. https://youtube.com/shorts/JoNATJN6Big</p>	<p>In this usability testing, the focus is on evaluating the clarity and intuitiveness of the app's buttons and functions. The test begins by confirming that the email and password fields on the login page are clearly labeled and self-explanatory. The Sign In button functions as expected, leading to the main menu. Once logged in, features like Recenter, Clear Route, and Hide Marker are tested, all of which perform</p>

		<p>intuitively, with clear effects such as resetting the map or hiding markers. The settings menu allows easy adjustments to the guest name, accessibility options (like dark or light mode), and time units. The common places feature is also tested, showing that users can quickly route to familiar locations like the Library. Overall, the app's buttons and features are user-friendly, with clear labels and easy navigation.</p> <p>From these results we can see the completion of SC.16</p> <p>Aesthetically pleasing and functional design to the user which promotes the user experience.</p>
U4	<p>Stakeholder walkthrough:</p> <p>https://youtu.be/94u2H1NJnAM</p> <p>This narrated video is then me going through the application in the perspective of the user for some added evidence.</p> <p>https://youtube.com/shorts/yX1RLyFLZGM</p>	<p>In this usability testing for U4, the focus is on ensuring that the school map accurately reflects the user's location and provides a clear route to their destination. Since the tester is testing from home and far from the school, the app routes users to Google Maps if they are outside the school. However, if they were within the school, the app would use the closest node to determine the starting point. The tester checks this by selecting Main Reception as a location, and the app correctly displays a route from the closest node, guiding the user toward the reception.</p> <p>The app not only provides the correct routing but also labels key locations along the path, such as Main Reception, Main Hall, Gym, and Lecture Theater,</p>

		<p>so users can easily track their progress and verify they are on the right path. The satellite imagery feature is also functional, offering an enhanced view of the school layout, which further helps the user navigate effectively. The test confirms that the app accurately reflects the user's location and provides clear, descriptive routing.</p> <p>From these results we can see the completion of SC.02</p> <p>The map should be accurate, easy to understand and reflective of the building</p>
U5	<p>Stakeholder walkthrough:</p> <p>https://youtu.be/94u2H1NJnAM</p> <p>This narrated video is then me going through the application in the perspective of the user for some added evidence.</p> <p>https://youtube.com/shorts/H_hlVDfcxq4</p>	<p>In this Usability Testing for U5, the focus is on evaluating whether each screen is easy to access and clearly indicates its purpose. The test begins with the login screen, which is simple and straightforward, making it easy to access and clearly marked with a "Sign In" label. If the user wishes to sign up, the sign-up button is located at the bottom of the screen, clearly labeled "Sign Up", and the title changes to</p>

	<p>"Register Account", clearly indicating the purpose of the screen. If the user decides to go back, the Sign In button is easily accessible.</p> <p>The test further examines navigation, confirming that users can easily switch between screens. For example, after entering an email address, users can scroll or use the keyboard's down button to easily navigate to the Sign Up button. The test also confirms that users can navigate directly to the main menu by selecting the Guest option, which brings them to the main menu screen. The main menu is intuitive, with a map and several clearly labeled features, including the settings icon, weather icon, ETA, and options like report, search location, and common places.</p> <p>Finally, the settings page is also reviewed, with a clear title and icon that indicate its purpose. The settings cog icon is easily accessible from the main menu, making the navigation seamless and intuitive. Overall, the test confirms that all screens are user-friendly, easy to access, and clearly indicative of their respective purposes.</p> <p>From these results we can see the completion of SC.19</p> <p>The different pages should be easy to navigate through.</p>
--	---

U6	<p>Stakeholder walkthrough:</p> <p>https://youtu.be/94u2H1NJnAM</p> <p>This narrated video is then me going through the application in the perspective of the user for some added evidence.</p> <p>https://youtube.com/shorts/mumlpJCCmGQ</p> 	<p>In this Usability Testing for U6, the focus is on verifying that the routes within the app are easy to follow and help users navigate the school efficiently. The test begins by selecting a route from the Sixth Form Common Room to Room 12. The app provides a clear route with the user icon indicating the user's current location. The icon moves along the route as the user progresses, allowing them to track their movement. Additionally, the ETA is displayed, showing the estimated time to reach the destination.</p> <p>The app effectively shows the fastest route to Room 12, and the test verifies that it provides an efficient and clear way to traverse the school. The tester then tests a different route, from the Sixth Form Common Room to the Dining Hall, which also displays a route with an estimated time of 189 seconds. The routes are easy to follow, with clear guidance provided for users to efficiently reach their destinations. This test confirms that the app allows users to find and traverse routes within the school effectively.</p> <p>From these results we can see the completion of SC.21</p> <p>The route should take the shortest distance and be accurate and precise.</p>
----	--	---

Evaluation

After completing the development phase, I will evaluate my navigation application on four critical aspects: Functionality, Usability, Robustness, Maintenance and Limitations.

To do this, I will use the test plans outlined in the Design section as a guide. This evaluation will involve conducting a comprehensive run-through of the application to simulate the experience of a typical user, e.g. searching and achieving a route and an ETA, allowing me to test various features and elements of SchoolNav against the success criteria established in the analysis phase. During this process, I will identify any errors or issues that need addressing. Additionally, I will involve multiple stakeholders in testing the app, gathering valuable feedback to gain different perspectives on its performance, and using their input to make any necessary improvements. This thorough testing process will ensure that the game is fully refined and ready for release.

Demonstration

Here I have done a full walkthrough of the game and tested the different aspects - from the Login Page to the Settings Page. This allowed me to test the overall functionality of the application, and understand from a user's point of view.

<https://youtu.be/ud2fu4B4Xs4>

In this demonstration, I walk you through the SchoolNav application, covering everything from the login process to various features like route navigation, settings, and reporting. The walkthrough begins with the login page, where I explain the options available for signing in, including the "Guest" feature, which provides quick access to the app without an account (**SC.19**).

Once logged in, the main interface presents the school map, where you can search for locations, calculate the shortest route, and view the estimated time of arrival (**SC.21**). I demonstrate how the "Search Location" feature works, navigating from places like the Main Hall to the Dining Hall and more. The app intelligently selects the nearest node to begin the route and displays it efficiently.

The "Common Places" feature is also highlighted, where I show how selecting popular areas like the library or lecture halls allows users to quickly get directions. Additionally, I explain the "Report" feature, which lets users create markers on the map to report issues, like food being finished in the dining hall.

The settings section allows for personalized adjustments which links to the success of **SC.03**, such as changing the user's name, setting accessibility options, and switching between light and dark themes. There are also placeholders for future features like timetable integration and saving personal room numbers.

Lastly, I demonstrate the login validation system, confirming that the app ensures proper email formatting and password matching. Overall, the app functions smoothly with intuitive navigation, user-friendly features, and effective error handling.

Overall, the application has met the requirements of the initial analysis, effectively addressing the problem definition, which focuses on assisting users with navigation through the school. There is a solid set of features included, though some were excluded due to time constraints, which I plan to explore in more detail later on. Even with these features not being fully implemented, the majority of the code is relevant and necessary for the core functionality of the application. The navigation system, user customisation, and reporting features are all operational, offering a strong foundation. The app demonstrates practical use and flexibility, aligning well with the original goals, and I am confident that the remaining features, once completed, will further enhance its functionality and user experience.

Final Stakeholder Testing

SchoolNav is an application which has a specific target market, this makes the Stakeholder Testing much more significant as it determines the quality and usability of my application. Most of these stakeholders will rely on my application to traverse their schools, so ensuring feedback on this with stakeholders is important. My stakeholders include teachers, students, parents and visitors/guests to School. I asked a Year 7 student to use the app for the first time and to tell me his thoughts at the time of using each feature on the app. The video to this is found above in the Usability Evidence section. The evidence is found in (page 345 -352). I also asked 2 teachers and 10 other students to test my application.

Here were the overall results:

<u>Stakeholder Usability Test (Year 7 User):</u>		
Test Description	Successful?	Result
The application has functional buttons and responsive controls.	Yes	The stakeholder was able to interact with the app's main features, including signing up, logging in, and navigating the school map. Key buttons like "Recenter," "Clear Route," and "Hide Markers" functioned correctly, improving usability. However, minor issues were noted, such as difficulty pressing the "Search Location" button, suggesting that making the entire button clickable would improve usability.
Settings should be easily accessible and visible, when in the Main Menu Screen.	Yes	The stakeholder successfully accessed the settings, tested features like updating personal details, and noted accessibility options. The ability to change themes (light/dark mode) and input personal information demonstrates that the settings are easily accessible and functional.
Easy to understand and self-explanatory sections and buttons.	Yes	The stakeholder found the app intuitive, mentioning that features like the "Common Places" function and

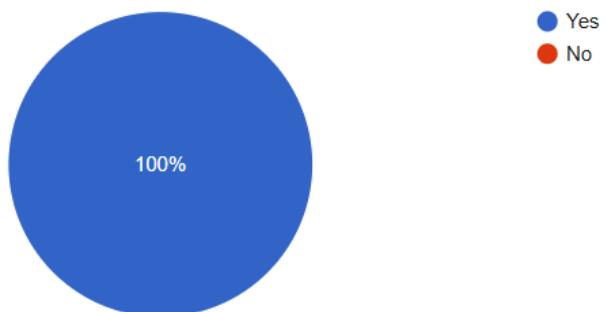
		navigation options were easy to understand. They recognized the design choice of shortening "Nav" in the title as effective. However, they noticed that the search button could be more user-friendly, indicating a minor area for improvement.
The school map should reflect accurately and descriptively the location of the user and route to take.	Yes	The stakeholder confirmed that the app accurately displayed locations like the boys' toilets and library, matching real-world locations. They appreciated the ability to visually verify locations, reinforcing the accuracy of the map data.
Each screen should be easy to get to and indicative of its purpose.	Yes	The stakeholder moved through different sections (login, settings, navigation, and common places) without confusion. They highlighted that features such as "Recenter" and "Clear Route" worked as expected, helping them manage their navigation efficiently.
The routes should be easy to follow and use, to find a route and traverse it efficiently.	Yes	The stakeholder successfully searched for locations and verified the app's ability to guide them effectively. They specifically noted how the app would be beneficial for transfer students who are unfamiliar with the school layout, showing that the routing system is practical and user-friendly.

Using a combination of qualitative and quantitative questions I have collated data from 12 stakeholders (2 Teachers and 10 Students). I primarily got data from students and a few teachers. I have made best use of sections in this questionnaire which make sure the relevant precise questions are going to the right group of people. This data will allow me to evaluate and analyse the final product of my program and how usable it is, through direct observation and analysis of user interaction. My analysis assumes the entire questionnaire and its respondents completed the survey truthfully.

Stakeholder Usability Test (Questionnaire):											
Question:	Justification and analysis:										
<p>How responsive were the buttons on a scale of 1 - 10?</p> <p>12 responses</p> <table border="1"> <thead> <tr> <th>Rating</th> <th>Count (%)</th> </tr> </thead> <tbody> <tr> <td>9</td> <td>5 (41.7%)</td> </tr> <tr> <td>10</td> <td>7 (58.3%)</td> </tr> <tr> <td>1, 2, 3, 4, 5, 6, 7, 8</td> <td>0 (0%)</td> </tr> </tbody> </table>	Rating	Count (%)	9	5 (41.7%)	10	7 (58.3%)	1, 2, 3, 4, 5, 6, 7, 8	0 (0%)	<p>Justification:</p> <p>I asked this question to test against the completion of U1 where the application has functional and responsive controls.</p> <p>Analysis:</p> <p>All the results were either 10 or 9 where 10 meant Most responsive, so this shows that the responsive aspect of U1 has been met here.</p>		
Rating	Count (%)										
9	5 (41.7%)										
10	7 (58.3%)										
1, 2, 3, 4, 5, 6, 7, 8	0 (0%)										
<p>How functional were the buttons on a scale of 1 - 10?</p> <p>12 responses</p> <table border="1"> <thead> <tr> <th>Rating</th> <th>Count (%)</th> </tr> </thead> <tbody> <tr> <td>8</td> <td>2 (16.7%)</td> </tr> <tr> <td>9</td> <td>3 (25%)</td> </tr> <tr> <td>10</td> <td>7 (58.3%)</td> </tr> <tr> <td>1, 2, 3, 4, 5, 6, 7</td> <td>0 (0%)</td> </tr> </tbody> </table>	Rating	Count (%)	8	2 (16.7%)	9	3 (25%)	10	7 (58.3%)	1, 2, 3, 4, 5, 6, 7	0 (0%)	<p>Justification:</p> <p>I asked this question to test against the completion of U1 where the application has functional and responsive controls.</p> <p>Analysis:</p> <p>We see a little bit more spread, however the results are skewed towards the most functional end, suggesting the successful completion of U1.</p>
Rating	Count (%)										
8	2 (16.7%)										
9	3 (25%)										
10	7 (58.3%)										
1, 2, 3, 4, 5, 6, 7	0 (0%)										

Was the Settings Page easy to find and access?

12 responses



Justification:

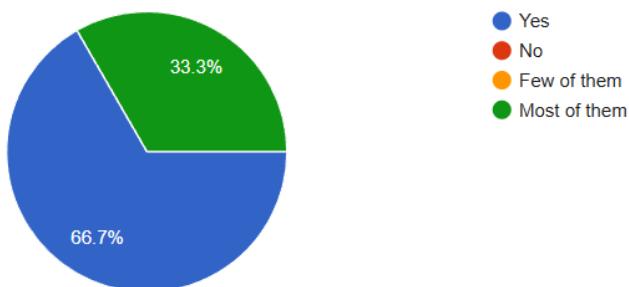
This question tests the completion and success of [U2](#) where it tests the visibility and accessibility of the Settings Page as it is a crucial component of my application.

Analysis:

Out of the 12 Stakeholders who responded all of them were able to access and find the settings page, suggesting the completion and success of [U2](#).

Did you understand each button and section of SchoolNav?

12 responses



Justification:

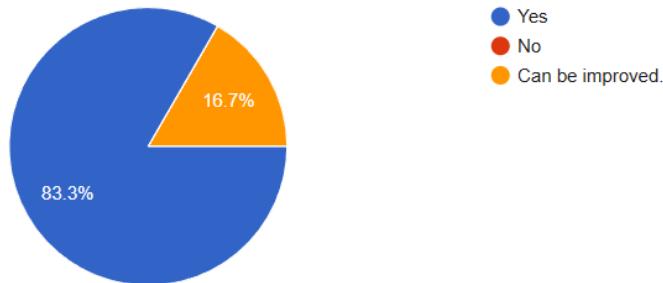
This question tests the completion and success of [U3](#) where it checks against the Success Criteria: SC.16, to see if it is self-explanatory and understandable.

Analysis:

All results vary between Yes and Most of them, suggesting that users found the buttons understandable generally with a few exceptions. A follow-up survey can determine the specific buttons for further development to completely meet [U3](#), however the majority success finalises this completion.

Was the Map and all the markers/routes reflective of the school?

12 responses



Justification:

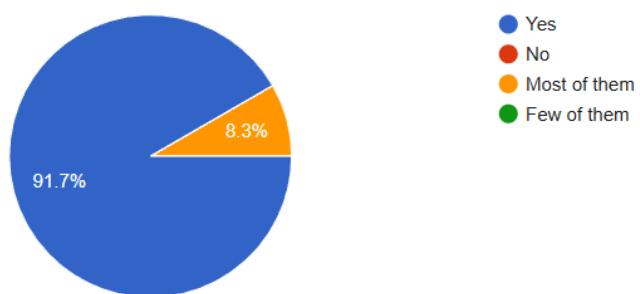
This question is to assess the completion and success of [U4](#), this is important and significant because it tests against the Map which is a fundamental part of my application.

Analysis:

Over 80% said that the map was reflective of the school, which is a great majority with around 20% saying it could be improved. The most important fact is that no stakeholder said the map was utterly not reflective of the school so this marks the completion of [U4](#), further development could better plot the map and label it accordingly.

Was each screen (e.g. Main Menu) easy to get to and indicative of its purpose?

12 responses



Justification:

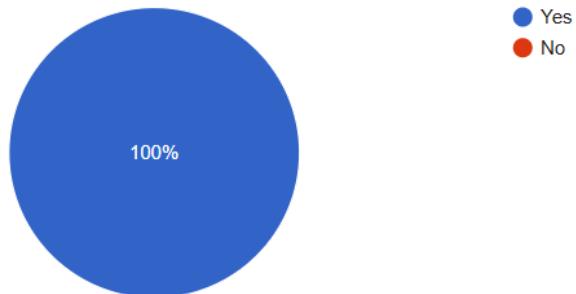
This query assesses the success of [U5](#) where it checks the easy to understand nature of each screen and the supporting navigation to get there.

Analysis:

With nearly 100% of the stakeholders saying that each screen was easy to get to and some saying that Most of the Screens were easy to get to, this marks the success and completion of [U5](#).

Was the routes easy to follow?

12 responses



Justification:

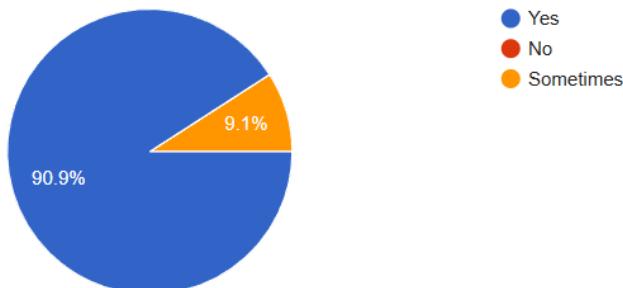
This question checks against the completion and success of [U6](#) arguably the most important usability test which checks the route and ease of following it and the efficiency of it.

Analysis:

Glad to see that 100% of the stakeholders said the routes were easy to follow! This is a major win and exhibits the success of [U6](#).

Was the routes efficient (fastest route)?

11 responses



Justification:

This question checks against the completion and success of [U6](#) arguably the most important usability test which checks the route and ease of following it and the efficiency of it.

Analysis:

Similar to the previous question, the result was positive with no stakeholder saying the route was not efficient (fastest). Some did say the routes sometimes were not efficient, so further development can perhaps look to improve upon this.

Further evaluation suggests a few areas for improvement. Expanding the clickable area of the search button would enhance usability, as the stakeholder found it difficult to press. While location accuracy was confirmed, regular updates to school maps would help maintain reliability. The accessibility settings are a valuable feature, and future enhancements like text-to-speech or font adjustments could improve usability for visually impaired users. Additionally, implementing a check to prevent blank name entries after changes in settings would strengthen error handling.

Overall, the application meets the test plan criteria well, offering a functional, intuitive, and responsive user experience with minor refinements needed.

Success Criteria Evaluation

Here is my success criteria I have established previously, the purpose of this section is to comment and evaluate on the success/impartial/failure of these outcomes, denoted by a traffic colour system and also explanations. I have provided comments here on how these failures/impartial success can be addressed in further development here as well.

No. (S.C)	Requirement s' Description	Justification	How do we measure success?	Outcome + Evaluation
SC.01	The application should have a smooth and responsive user interface (UI) system.	Only if the UI is responsive can users fully use the application and find the directions to where they want to go, adjust the settings and several other features all depend on receiving user input and making use of it, hence needs to be responsive. This allows users to smoothly access and route the paths they need to along as fast and convenient as possible. This is a major factor since the application aims to reduce time traversing, but by not having a responsive UI system the application wastes further time doing its job. Hence should be fast, smooth and responsive to user input.	<ul style="list-style-type: none"> ❖ Manipulated and traversed ❖ Easy to go to different pages of the application ❖ Different processes of the application for e.g changing Maps, able to be school name, should be fast and easy to perform 	<p><u>As seen in UI</u> and in <u>R1</u> the application in its entirety is smooth and responsive with functional buttons and responsive controls, and each screen is appropriately presented and displayed in the correct order.</p> <p>A few places of improvement could be how each marker on the screen can be routes to from the closest place just by a click on the map rather than the Search Location process to find a route.</p>

SC.02	<p>The map should be accurate, easy to understand and reflective of the building</p>	<p>The map is the most fundamental aspect of my application, with most of my primary research showing how a clear and distinct map is vital to creating a successful navigation app. Even though researching similar applications like Google Maps and Waze, they have an apparent and easy-to-understand map, with little to no learning curve.</p> <p>Users should be able to know where they are in the building by looking at the map and seeing which direction they are going along.</p>	<ul style="list-style-type: none"> ❖ Room numbers are shown with each room ❖ Common places highlighted and named on the map ❖ Inactive corridors marked with red + white chequered line ❖ Pictures should be shown at each junction or significant room to help direct users. 	<p><u>As seen in U4</u> the map is accurate and is understood by stakeholders and users are able to understand and traverse the map on screen successfully.</p> <p>This is a great success as it is not only a fundamental part of my application but the most dependent part, so the fact the map on the screen is easy to understand and reflective of the building shows how there is success in this region of the application.</p>
SC.03	<p>Settings should allow updating and meet all different user needs.</p>	<p>This is because in settings, there will be an option to choose if accessible routes are necessary for the user or not. In addition, the particular subjects/timetable of the user, alongside the form room. These are common places which by using the settings I can input the locations and where the user should be at each certain time. Allowing faster route finding, which also acts as a reminder for the student of where to go.</p>	<ul style="list-style-type: none"> ❖ Settings are allowed to be changed and stored with the user's account ❖ Settings should be acted upon by the routes provided and the maps for the users. E.g Ramps and lifts should be highlighted on the map for users. 	<p><u>As seen in U2 and R3, R4</u> the settings are not only adjustable and can be altered according to user needs and wants, the Main Menu and all other functions adjust in response to those changes.</p> <p>Potential improvements could be in where the accessibility routes function could be added in, the light/dark themes for the app (desirable features).</p> <p>Overall, the settings allow updating and</p>

				meet the user needs. E.g. with the Settings users can change if they need accessibility needs or if they need to change the ETA Measure to Minutes instead of Seconds.
SC.04	Estimated Time of Arrival should be accurate and realistic	This will be shown in conjunction with each route mapping. This is to give a rough idea to the user when they would arrive at their location.	❖ ETA should be tested and timed, accurately giving the time you would arrive at your destination.	<p><u>As seen in U6</u> we see how the ETA is accurate and realistic. The user tested for the time it takes to traverse a route and compared it against the presented ETA time in seconds, hence making it accurate and realistic.</p> <p>This is important because ETA is a number or value most people in schools wouldn't know off the top of their head and we talked about how this is an essential feature in the analysis. ETA features are a fundamental part of this application and the fact that it is accurate and functioning accurately boosts the potential of the application severely.</p>
SC.05	The application must be responsive to certain requirements /details the	Majorly disabled users of this application would need routes to involve a lift or ramp system. Steps have to be avoided on routes and pointed out on the maps of these users.	❖ After specifying the Disabled Accessibility routes, users should be presented with routes with lifts/ramps and not any steps.	<p><u>As seen in R3, R4</u> the settings is saved and the application is responsive to certain requirements.</p> <p>Although, we have left out any accessibility routes and how the</p>

	<p>user may have.</p>	<p>The particular school of the user should be specified in the settings page and the map of the specific school should be accordingly presented.</p>	<ul style="list-style-type: none"> ❖ If the school they have inputted as the current school is not mapped/in the application it should show an error message. 	<p>users from room and time-table information can be saved due to time restrictions. I havestill added the options to toggle and input values, and these options can be then used to change the main functionality.</p> <p>Similar to how the name can be changed and we saw the success of this in the stakeholder testing these settings can be changed also given some more time and resources.</p> <p>After coding the majority of the application, I realised creating this feature and adjusting each route would take too much time and decided to exclude this feature.</p>
SC.06	<p>The Login/Sign-up page should be the first screen for the users of the application, but the user should have the ability to use the app as a guest and should have</p>	<p>This allows new users to log into their account and access their saved settings like what school they go to, their form room and if they need disabled access. Also, to allow saved searches in the search bar.</p> <p>Users should have the option to sign in as a guest since it would faster access to the app and users can then</p>	<ul style="list-style-type: none"> ❖ Should successfully create a new account. ❖ Should set certain settings, log out and then log back in to see if the settings are saved. ❖ After logging in to search for some specific places logout and log back in to see if those searches are still present. 	<p>As seen in R1, we can see how the first screen of the application is the Login page and users have the option to Sign-up and save their details e.g. Name which will be transferred onto the Main menu for customisation. The fact this is functioning as expected links back to the flow chart I formulated in the designs section and how the first screen is fundamental as it</p>

	appropriate validation.	sign in later on to save their details. As a guest, users can also view and enjoy traversing the map of schools allowing them to explore their options in future schools.	<ul style="list-style-type: none"> ❖ The guest mode should be tested to see if I can easily traverse and route paths without the features of saved searches or settings. 	allows users to log in to their saved accounts or create accounts, as School information can be private and needs to be protected by some sort of authentication, as well as how users wouldn't need to re-enter certain details.
SC.07	Reporting feature - to highlight issues and factors that exist on the map and present them to other users	This is to allow users to report certain factors which minimises the work that needs to be done by maintenance and easily provides information to be passed on to other users.	<ul style="list-style-type: none"> ❖ Test if the report function is responsive and allows an array of factors to be reported ❖ If reported issues by another user are visible to other users 	<p><u>As seen in U3</u>, and <u>R5</u> how the reporting feature works as intended, it is constant on the map and visible to users, the reporting feature has a description feature as we saw in the stakeholder testing.</p> <p>Overall, evaluatively speaking the report function is good and meets the success criteria but further improvement would be to save it to a database and allow users from other accounts to see it, as right now only the user who put the report can see it.</p>
SC.08	The weather shown in the app should be accurate and have the appropriate image to represent the climate for that day and	This is particularly handy for students since they would have to traverse an entire day in that climate and especially if they live further away, it will be harder to present the weather at the school. Hence appropriately students/users can wear suitable clothing	<ul style="list-style-type: none"> ❖ Screenshot weather shown by app, and weather shown through other reliable weather apps. ❖ Screenshot the weather shown for the following days, and see if it matches. 	<p><u>As seen in U3</u>, the weather section is representative and clear of its purpose but the main functionality has not been implemented.</p> <p>The icon and the image has been presented and shown at the top left corner but pressing on</p>

	the following days.	and bring an umbrella if necessary.		it doesn't lead to any modal or pop-up with weather information. I haven't implemented this due to restrictions within the application. After coding the majority of the application, I realised creating this feature and adjusting each route would take too much time and decided to exclude this feature.
SC.09	Common places tabs where it presents the ability to access the common places of the map.	This is a desirable feature for my application since it simplifies user's having to search for these common places such as toilets. The search can be mainly used for niche examples like rooms. This simplifies the user experience and makes the app more comfortable to use.	❖ After clicking on a place on the common places tab, the user is routed to that place from the start point. ❖ There should be a wide range of places which meet the needs of all students. E.g. toilets, main hall or reception.	<u>As seen in U4</u> , the common places tab has a modal of all the common locations and users can select the location to be routes to that place successfully. This is a great success to further improvements from there.
SC.10	Compass feature/Recenter feature and map adapting to user orienting the phone.	This is to orient users. Reduce the confusion and promote clarity on where users are going. Compasses have niche case uses in camping or other treks, hence can be utilised then, perhaps on a school trip. But majorly this feature is to meet all the needs of my stakeholders and to improve the clarity of the visual representation of the	❖ As I move the phone around the visual point representing myself, turn accordingly and face the appropriate buildings. ❖ If I press the compass button, I should get the bearing and also the primary direction I am facing.	<u>As seen in U3</u> , the map presents the recenter button which has been successfully implemented. The recenter button animated users to the current location and zooms in on their whereabouts. Again, a successful outcome for the recenter button and how it orients the map and the phone.

		user.		
SC.11	Traffic colour system for corridors	<p>This is to inform users of the state of certain corridors, it can be slow and cramped to traverse in packed corridors, hence this feature can help users inform decisions on where to go and my algorithm can also use this information to reroute certain paths with high traffic.</p>	<ul style="list-style-type: none"> ❖ Corridors with a larger number of users on it should be presented with a red colour rather than a corridor with no traffic as green or uncoloured. Mild traffic is offered with amber. ❖ The number of users for each threshold is reasonable. 	<p>As seen in U4, the routes are implemented with a polyline and the corridors are coloured to make the route visible.</p> <p>However, the success criteria talks about how the Traffic Colours should be relative to the amount of users in that corridor. With more time and resources I would be able to implement this by adding a server where users can interact with each other etc.</p> <p>Since we can see how the corridor can be colourized using a polyline for the route similarly it can be implemented for the traffic system.</p> <p>After coding the majority of the application, I realised creating this feature and adjusting each route would take too much time and decided to exclude this feature.</p>
SC.12	Response to if user is heading in wrong direction	If there is clear indication, that the user is heading in the wrong way. The user can be led to go more in the incorrect direction. This leads to	<ul style="list-style-type: none"> ❖ Red alert or vibration is produced if the user heads in the opposing direction on a set route. 	<p>As seen in U6 and in R6, the user can identify if they are going in the wrong direction since the user location wouldn't be going in the direction</p>

		more confusion and more anxiety which contradicts the purpose of this solution entirely.		of the path.
SC.13	Product can handle multiple responses at the same time.	This is because, in schools, users/students utilise and make use of the app around the same time e.g at the start of the day or after break. Hence, my application needs to be able to handle different user requests simultaneously.	<ul style="list-style-type: none"> ❖ Testing with different devices using the system at the same time and seeing if the application works in sync with other users. 	<p>As presented in R4, the application can handle different settings and resetting of the application.</p> <p>However, utilising servers to add multiple users has not been implemented here.</p>
SC.14	The maps should update according to the user commenting feature.	Factoring important changes to the school map is a vital component. Since the route may include corridors which are inactive. Most often corridors or paths which exams take place, shouldn't be traversed along, hence should those particular areas should be updated for the application to input and base routes off of..	<ul style="list-style-type: none"> ❖ Comment as a user, corridor is inactive or that an exam session is active in that vicinity. This will result in a change within the map, a visual clue to the inactivity or certain event, also routes will be changed and altered depending on these comments. ❖ Ability to report and see your report affect routes 	We saw in R5 the commenting feature stays on the map and the maps update according to the user comments. This is important as only this feature allows the users to communicate certain news and events within the map/location and the fact this is working secures the success of this success criteria.
SC.15	Logo Display should be prominent and visible to users, indicating	It allows the user to associate a brand with the app. Allowing easy access to the application, even in the midst of several other apps.	<ul style="list-style-type: none"> ❖ A clear logo at the top of the screen and when the user logs on. 	<p>As seen in U5 and in R1, we can see how the Logo is displayed and represents the purpose and function of the app.</p> <p>This is clearly seen in the usability stakeholder testing where the Year 7</p>

	the app and its purpose.			Stakeholder was able to understand and derive the purpose of this app just from the logo, presenting how it indicates the purpose of the application and the screens. The logo also helps with brand recognition.
SC.16	Aesthetically pleasing and functional design to the user which promotes the user experience.	For SchoolNav , an aesthetically pleasing and functional design is essential to enhance the user experience. A visually appealing interface keeps users engaged while ensuring that features are easy to use and accessible, promoting a smooth and enjoyable navigation process.	<ul style="list-style-type: none"> ❖ Colourful and appropriate theme, which compliments the application. ❖ The information should be legible and easy to understand. 	<p><u>As seen in UI</u> and the buttons and the screens are functional and responsive and are aesthetic to the users.</p> <p>This is important as it will incline users to come back to the application and keep using it as users can be easily bored by a bad design.</p>
SC.17	Register screen should function and take in username and password and have appropriate validation.	<p>This is to ensure that the application's account details and login verification is appropriate and the inputs are valid. This is to ensure the program works as intended.</p> <p>Ensures that only valid usernames and passwords are accepted, preventing errors and security risks. Proper validation improves data integrity and enhances user security.</p>	<ul style="list-style-type: none"> ❖ Users can successfully register with a valid username and password. ❖ Invalid inputs (e.g., empty fields, weak passwords) trigger appropriate error messages. ❖ Duplicate usernames are not allowed, and users are prompted to choose a different one. ❖ Passwords meet complexity 	<p><u>As seen in R1</u>, the validation is appropriate and is up to standard. The Register Screen functions perfectly and presents error messages when necessary e.g. when the password and confirm password fields don't match. The validation is readable and can be used to inform the user of what to improve on.</p>

			requirements (e.g., minimum length, special characters).	
SC.18	The username and password fields should work from the saved register screen.	Users should be able to log in seamlessly using their registered credentials. A functional authentication system ensures secure access and prevents unauthorized use.	<ul style="list-style-type: none"> ❖ Users can successfully log in with their registered credentials without errors or delays. ❖ Incorrect credentials trigger appropriate error messages (e.g., "Invalid username or password"). ❖ Multiple users can log in and out without issues. 	<p>Linking on from the previous SC.17 we see how in R1 and R2, users can successfully log in using the registered details and save their account details e.g. name.</p> <p>This is successful as it allows successful Login using Registered details.</p>
SC.19	The different pages should be easy to navigate through.	<p>For SchoolNav, it's crucial that different pages are easy to navigate, allowing users to quickly access features like the map, settings, and common places. Intuitive navigation ensures that students can seamlessly switch between sections without confusion, improving the overall user experience.</p> <p>Users should be able to move between different sections of the app effortlessly. A well-structured navigation system ensures a smooth user experience, allowing quick access to key features.</p>	<ul style="list-style-type: none"> ❖ Users can switch between pages without confusion or unnecessary steps. ❖ Navigation buttons and menus function correctly on different devices. ❖ Page transitions are smooth and responsive, with minimal load time. ❖ User feedback (e.g., usability testing) confirms that navigation is intuitive. 	<p>As seen in U5 the screens are indicative of their purpose and are easy to get to/navigate to. The stakeholders had no problem navigating to each screen in the application e.g. Settings. Users knew how to get to the Settings page, showing how the screen is easy to get to and navigate to.</p> <p>This is successful and the icons reflect the purpose of the app and the app location/navigation to the app from said place.</p>

SC.20	<p>The route should be clear and presented on the Map.</p>	<p>Users need to visually see their selected route to follow it correctly. A well-displayed route prevents confusion and enhances the app's usability for efficient navigation.</p> <p>For SchoolNav, the route should be clearly displayed on the map to guide users effectively. A visible, straightforward route ensures that students can easily follow the path to their destination, reducing confusion and improving the overall navigation experience.</p>	<ul style="list-style-type: none"> ❖ The route is displayed accurately on the map with a clear path, markers, and visual indicators. ❖ Users can distinguish different routes using colors or labels. ❖ Zooming and panning do not distort or hide route information. ❖ Users can successfully follow the route without ambiguity. 	<p><u>As seen in U4 and R6</u> the route is clear and presented on the map using a bright blue polyline feature. This is a clear route from A to B e.g. Dining hall to the Common room. The route is presented on the map and is a clear and correct route.</p> <p>This is successful and a fundamental success to the program as it links to the main functionality of the program.</p>
SC.21	<p>The route should take the shortest distance and be accurate and precise.</p>	<p>The app should calculate the most efficient path to save time for users. Accurate routing ensures reliability and effectiveness in navigation.</p> <p>For SchoolNav, routes must be accurate, precise, and take the shortest distance to ensure efficient navigation. Incorrect or inefficient paths could confuse users and waste time, especially in a busy school setting. Optimizing routing algorithms enhances reliability, helping students reach</p>	<ul style="list-style-type: none"> ❖ The shortest path is consistently chosen, and users reach their destinations efficiently without unnecessary detours. ❖ The calculated route matches real-world paths and avoids obstacles. ❖ Time estimates align closely with actual travel times. ❖ User testing confirms the route is logical and efficient. 	<p><u>As seen in U4 and R6</u>, being very similar to the previous Success Criteria, we see how the routes are displayed and also displayed in the shortest distance/route. Having cross-checked with stakeholders the routes are traversable and are the shortest route from A to B so meeting SC.21.</p> <p>The reason why this Success criteria is so important is because if the route is not accurate or precise the users wouldn't be able to traverse any routes or get to a place in the school, since this is the main purpose of</p>

		destinations quickly and easily.		SchoolNav this success criteria is vital!
--	--	----------------------------------	--	---

Essential and Desirable Features

Here are all the features I chose to include and develop in my application during the Analysis section. Although the final product has much more of these features, I chose to include these additional features during the development process as I realised the necessity of that feature. But, overall here are the main features with evaluation and potential improvements.

Feature	Essential/ Desirable	Successful ?	Evaluation/Improvements
<input type="checkbox"/> Map of school: this should be up-to-date and a clear representation of the school. The appropriate room numbers should also be enlisted.	Essential	Yes	<p>The map is clear and up-to-date. It is a valid representation being a 3D overview of the school using satellite imagery, the map is clear and the room numbers and locations are listed and labelled. This is important as users will use this map to actually traverse their route and visualise their place in the school.</p> <p>Improvements could be to add better satellite imagery for the school specifically instead of using pre-existing google imagery.</p>
<input type="checkbox"/> Pictures associated with each room: It needs to have pictures associated with each room, which can be accessed when tapped on but also when a route points to that room.	Desirable	No	<p>This was not implemented due to a lack of time within the application.</p> <p>However, users can see their location using the current location marker and the 3D satellite imagery helps users see where they are in the map.</p>
<input type="checkbox"/> Login Page: this should be the opening page to the application, ensuring saved	Essential	Yes	This is a successful feature in the application, the Login Page is the opening

<p>information in settings so appropriate school maps can also be accessed. Further, saved searches of particular rooms or places.</p>			<p>page and allows people to log in to their accounts and find their saved information. This is important as it prevents users from having to reenter all their details and also to add a layer of protection and authentication for the app.</p> <p>Improvements could be to save these Login Details to a database rather than using asynchronous storage as users will only be able to login on the device they registered on.</p>
<p><input type="checkbox"/> Home Page: this should show the option to “Navigate school” and to also access the “Settings” page.</p>	Essential	Yes	<p>The Home Page/main Menu is presented successfully and has a set of features which allow the functionality of the application. The navigate to school feature is presented and users can also access the Settings page from the Main Menu, the reason this feature is important is because it houses the main components of my program.</p> <p>Further improvements could be to add an information page to show the features and screens and their purposes descriptively.</p>
<p><input type="checkbox"/> Audio-based directions: making use of audial directions e.g. “Take the next left” can be useful and has been deemed necessary in the</p>	Desirable	No	<p>This was not implemented due to a lack of time within the application.</p> <p>Being a desirable feature I neglected this feature and</p>

questionnaire conducted above, hence desirable.			prioritised my time in more important features like the Report feature.
<input type="checkbox"/> Compass: users can use the visual compass to see which direction they are going.	Desirable	Yes	This is a successful feature in my application, users can't only have access to a compass to find out which direction you are going, similar to any other navigation app. But it also features a recenter button which allows users to recenter back to their current location.
<input type="checkbox"/> Settings Page: this allows users to specify their school, if they require the use of accessibility routes, their particular subjects and appropriate room numbers, form room and timetable information. Users can also choose to log out here.	Essential	Yes	The Settings Page allows users to customise their experience by specifying their school, subjects, room numbers, accessibility needs, and timetable information. It also provides the option to log out, ensuring both convenience and security. This feature is important as it enhances personalization, accessibility, and efficiency, helping users easily manage and access their relevant school details.
<input type="checkbox"/> Common places tab: this shows the common places users go, allowing quick, single-tap access. This is desirable.	Desirable	Yes	The Common Places Tab provides users with quick, single-tap access to frequently visited locations. This feature is desirable because it saves time and improves convenience, allowing users to easily navigate to their most-used places without having to search or manually input information. Although it is a desirable

			feature, I had the time and coding ability to implement this into my code. A potential improvement could be to customise the common places and add to it individually.
<input type="checkbox"/> Pointer to show where the user is: this is vital since it has been mentioned several times by the stakeholders in this application and is an important component in similar software around the market.	Essential	Yes	The Pointer to Show Where the User Is is a vital feature, as it provides users with real-time location tracking within the app. This has been emphasised by stakeholders and is a key component in similar software, helping users easily orient themselves and navigate efficiently, improving the overall user experience and functionality.
<input type="checkbox"/> Reporting feature: primarily the information for the application is going to be derived from users reporting incidents and activity around the school, hence vital.	Essential	Yes	The Reporting Feature is essential as it enables users to report incidents and activities around the school, which forms the core data for the application. This functionality is vital for gathering accurate, real-time information, helping to ensure that the app remains responsive and relevant to users' needs.
<input type="checkbox"/> Weather for the area: this shows the weather for the school area for that day, and other days once pressed. A visual representation of the weather, alongside the degrees for the temperature, can be utilised.	Desirable	No	I wasn't able to implement the functionality for this feature due to time constraints. I have included the icon and button to access the weather but haven't added the actual functionality.

<input type="checkbox"/> 3D Mode: presents the 3D scale-up of the school map. Shows the height of rooms and buildings, which can be significant with buildings with more than one floor.	Desirable	Yes	<p>Although this was desirable I was able to include this feature.</p> <p>The 3D Mode presents a scaled-up version of the school map, showcasing the height of rooms and buildings. This is particularly significant for multi-story buildings, as it provides users with a more accurate and intuitive understanding of the school's layout, enhancing navigation and spatial awareness within the app.</p>
<input type="checkbox"/> Red and white chequered line: this is to show a corridor in particular is blocked/inactive.	Desirable	No	<p>After coding the majority of the application, I realised creating this feature and adjusting each route would take too much time and decided to exclude this feature. This feature to implement meant I would have to adjust the map permanently since the corridor is inactive but wasn't able to do this.</p>
<input type="checkbox"/> Traffic colour system: this can show the traffic density of certain corridors and represent them visually by use of colours.	Desirable	No	<p>After coding the majority of the application, I realised creating this feature and adjusting each route would take too much time and decided to exclude this feature.</p>
<input type="checkbox"/> XP points system: points can be generated by users who report factors which receive an amount of likes and/or comments. This can improve the validity of the reports.	Desirable	No	<p>This would take a lot of time and resources as the XP points would have to be tracked and saved on a database which I wasn't able to create.</p>

As we can see, there are a plethora of unsuccessful functions, but all of these are desirable. Meeting the essential features of my program is fundamental and the success of these is most important.

Further Development

Functionality/Meeting the Success Criteria

As in the Review of Iterative Development Tests table, there are quite a few areas of improvement that I can implement into my game. The reason I haven't been able to cater to these improvements are mainly due to time constraints, as adding these features would require further iterative testing and development, which requires a lot more time than that allowed for this project. Furthermore, there is the risk of the file growing to be too large and causing the game to crash because of all the lines of code and functions that will need to be implemented into react.native in order for these improvements to work.

The only Success Criteria I wasn't able to touch at all was the one pertaining to a Traffic Colour system for corridors (S.C. 11) - this is because, as previously mentioned, it would take too much time to implement a map where the corridors colour relate to the amount of traffic along that corridor, this is difficult to code and time consuming as it would mean I would have to add a server which tracks and links the location of each user and which corridor they are on. This process in and of itself can be very taxing and demand a large amount of time that I can't afford to give. Other potential improvements have been listed in the Review of Iterative Development Tests above.

I have met the majority of my success criteria with great results and have explored this in the previous table. Reasons for excluding certain features like the adjusted routes for the Accessibility Needs which links to SC.03 and how the settings need to adjust according to user needs. This wasn't included since I would need more time to change the routes so that it changes according to each individual needs. Further development would mean I can add these features and perhaps better improve my application to meet the needs of more users.

Additionally, I haven't succeeded in meeting SC.08 which talks about the weather functionality. This is important as it can allow users to know if they would need to take an umbrella or anything weather related to help with their journey. However due to time restrictions I prioritised my time in other more important features which compose more larger parts of my application. To implement this I would've had to get an API and implement it into my application which may have cost me and have taken time to add this to the application. Since I didn't have resources or even time in this stage I wasn't able to implement this successfully as to present the weather it would need to get it from the API then display it on the application, which is time-consuming to implement.

Another I wasn't able to include was found within the Success Criteria SC.07 which talks about the reporting feature being visible to other users. Although the report is able to be presented and displayed on the map, other users would not be able to check the reports made by the user since the report is not logged onto a server which other users can access, but rather to the specific map the user is using right there and then. I wasn't able to add this due to time constrictions as I would have to learn how to implement a server and save the information there and link to each user individually. This links also to SC.13, which talks about multiple users on the application and how the app handles it, and users can handle it but users can use the app at the same time but aren't able to interact with each other due to reasons mentioned above.

Overall, the success far outweighs the features missed out, and we see this success in the success criteria evaluation. All these unsuccessful features link to the nature of my analysis and targets as they weren't realistic given the time and resources I had. Better next time would be to consider my time more realistically. But overall the functionality is up to a high standard, near perfect and the success criteria is met completely.

Robustness

Generally the testing for robustness demonstrates the robust and secure nature of the application here, however, certain improvements can be made to the program entirely. The tests show how the Login Function and the Application Settings (Name) is saved by linking it to the account and this is constant. Further we see how routes are automatically cleared and reset when users exit and relaunch the application, verifying the robustness with routes. Although all Robustness Test IDs have been met, it can be argued that these tests can be amplified and extended to the program's future, through simulating high traffic and also with a better/worser GPS system. These will test the robustness and really verify if the application can withstand all conditions and inputs. Some improvements to the current robustness of the program can be to implement saving of other settings within the application e.g. Accessibility Needs and save that to the Setting. Further, as of now any Reports made are cleared when the user resets/relaunches the application, a potential solution would be to implement a server and database to store the Reports online so they can be retrieved later on/stay constant on the map until time limit expires.

Usability

We saw during the Stakeholder testing the user was able to navigate and manoeuvre around the map with no assistance, however there were a few buttons the user couldn't understand. The Report button for instance, the stakeholder believed this was to report any bugs within the application, which fair enough, makes sense but isn't the intended purpose of that button (rather to report an issue within the school on the map). Implementing a tutorial/instructions on how to use the application and what functions there can be a major change/improvement on the Usability of SchoolNav. Overall, the other main functions like routing users to different places of the school, worked effectively and had no problems with the user. The user interface was appealing and aesthetically pleasing, linking back to a few success criteria which focus on this. Further, the settings page and changing name and other settings works as intended. Perhaps, implementing additional settings and options which can better suit more specific users e.g. color blind support or high-contrast text.

Limitations

Throughout the development of SchoolNav, I encountered several limitations that prevented me from fully implementing certain features. As outlined in the Review of Iterative Development Tests table, many of these challenges stemmed from time constraints, as adding new features required extensive iterative testing and refinement, far beyond what the project timeframe allowed. Additionally, expanding the functionality risked increasing the file size to a point where performance issues, such as crashes, could arise due to the sheer volume of code and functions needed in React Native.

Another key limitation was the complexity of certain implementations. Some features required advanced technical knowledge that I had not yet fully developed, meaning additional learning time would have been necessary before even beginning implementation. Others demanded external resources, such as APIs or servers, which introduced additional constraints, including costs, integration challenges, and security considerations. Furthermore, real-time functionality and multi-user features would have required backend infrastructure, significantly increasing the development workload beyond what was feasible within the given timeframe.

Ultimately, these limitations were not due to a lack of vision or planning but rather the practical constraints of time, technical complexity, and available resources. Despite these challenges, I prioritised core features to ensure the application met its primary objectives while maintaining a high standard of functionality and usability.

Maintenance

Maintenance is a fundamental part of my application and project as it secures the longevity of my application e.g. fixing bugs and also adding new features/updating the map. To assist with the maintenance I have added several comments to highlight and pinpoint the purpose of certain sections of code and used suitable variable names (using CamelCase) to understand the purpose and need for those variables and what they represent. For example the nodes array, which was implemented using the longitude and latitude and how it is precisely for each location. If a new room were to come about, the maintenance team or I would add it to the nodes array and label it etc with the precise coordinates. This is an example of the maintenance that could happen to the application over time. I have also described my code and annotated it with great detail in the development section of this application, anyone maintaining the application will be able to refer to the annotations and descriptions for guidance and assistance in developing/fixing bugs. Further I have added video evidence for each feature and a walk-through so anyone in the future would be able to see how the application is meant to run and how each specific feature should work. Further, I have added in console.log() snippets throughout the code to help the developer test the functions and if they work as intended. Overall the maintenance of the program has been made easier due to the additional help and assisting parts of this project.

Final Code

Login Screen Code:

```
// Import necessary React and React Native components
import React, { useState } from 'react';
import { StatusBar } from 'expo-status-bar';
import { SafeAreaProvider, SafeAreaView } from 'react-native-safe-area-context';
import {
  StyleSheet,
  Text,
  View,
  TextInput,
  Image,
  TouchableOpacity,
  Platform,
  KeyboardAvoidingView,
  ScrollView
```

```
} from 'react-native';

import AsyncStorage from '@react-native-async-storage/async-storage';

export default function LoginScreen({route, navigation }) {
  const { name } = route.params || {};
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [errors, setErrors] = useState({});
  const [isPasswordVisible, setIsPasswordVisible] = useState(false);
  // State to toggle password visibility

  const validateForm = () => {
    let errors = {};

    // Check if email is provided and validate email format
    if (!email) {
      errors.email = "Email is required.";
    }

    // Check if password is provided
    if (!password) {
      errors.password = "Password is required.";
    }

    setErrors(errors);
    return Object.keys(errors).length === 0;
  };

  const handleLogin = async () => {
    if (validateForm()) {
      try {
        const storedUser = await AsyncStorage.getItem('userDetails');
        const parsedUser = JSON.parse(storedUser);

        if (parsedUser && parsedUser.email === email && parsedUser.password ===

```

```

password) {
    console.log('Logged in:', email);
    navigation.navigate('Main Menu', { name }); // Navigate to the main menu
after successful login
} else {
    setErrors({ ...errors, email: 'Invalid email or password.' });
}
} catch (error) {
    console.error('Error fetching user data:', error);
}
};

console.log("App executed");

return (
<SafeAreaProvider>
<SafeAreaView style={styles.container}>
<KeyboardAvoidingView
behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
style={{ flex: 1 }}
keyboardVerticalOffset={20}
>
<StatusBar style="light" />
<ScrollView
contentContainerStyle={{ flexGrow: 1 }}
keyboardShouldPersistTaps="handled"
showsVerticalScrollIndicator={false}
>
<View>
<TouchableOpacity>
<Image
fadeDuration={1000}
source={require('./assets/SchoolNavLogo.png')}
style={styles.headerImg}
alt="Logo"
/>
</TouchableOpacity>

```

```

        <Text style={styles.title}>Sign-in</Text>
        <Text style={styles.subtitle}>
            Discover new ways to get to your classrooms and more...
        </Text>
    </View>

    <View style={styles.form}>
        <View style={styles.input}>
            <Text style={styles.inputLabel}>Email Address</Text>
            <TextInput
                autoCapitalize="none"
                autoCorrect={false}
                style={styles.inputControl}
                placeholder="john@example.com"
                value={email}
                onChangeText={setEmail}
            />
            {errors.email ? (
                <Text style={styles.errorText}>{errors.email}</Text>
            ) : null}
        </View>

        <View style={styles.input}>
            <Text style={styles.inputLabel}>Password</Text>
            <View style={styles.passwordContainer}>
                <TextInput
                    secureTextEntry={!isPasswordVisible} // Toggle visibility
                    style={[styles.inputControl, styles.passwordInput]}
                    placeholder="*****"
                    value={password}
                    onChangeText={setPassword}
                />
                {/* Toggle eye button */}
                <TouchableOpacity
                    onPress={() => setIsPasswordVisible(!isPasswordVisible)}
                    style={styles.eyeButton}
                >

```

```
<Text style={styles.eyeText}>
  {isPasswordVisible ? "👁️" : "🙈"}
</Text>
</TouchableOpacity>
</View>
{errors.password ? (
  <Text style={styles.errorText}>{errors.password}</Text>
) : null}
</View>

<View style={styles.formAction}>
<TouchableOpacity onPress={handleLogin}>
  <View style={styles.btn}>
    <Text style={styles.btnText}>Sign-in</Text>
  </View>
</TouchableOpacity>

<TouchableOpacity
  style={{ paddingTop: 25, alignSelf: 'center' }}
  onPress={() => {
    navigation.navigate("Main Menu");
  }}
>
  <View style={styles.btn}>
    <Text style={styles.btnText}>Guest?</Text>
  </View>
</TouchableOpacity>
</View>

<TouchableOpacity
  style={{ marginTop: 'auto' }}
  onPress={() => {
    navigation.navigate('Register Page', {});
  }}
>
  <View>
```

```

        <Text style={styles.formFooter}>
          Don't have an account or forgotten details?{' '}
          <Text style={{ textDecorationLine: 'underline' }}>
            Sign-up
          </Text>
        </Text>
      </View>
    </TouchableOpacity>
  </View>
</ScrollView>
</KeyboardAvoidingView>
</SafeAreaView>
</SafeAreaProvider>
);
}

const styles = StyleSheet.create({
  container: {
    paddingHorizontal: 24,
    flex: 1,
    backgroundColor: "#38b6ff",
  },
  headerImg: {
    marginBottom: 0,
    alignSelf: 'center',
  },
  title: {
    color: '#333333',
    fontSize: 38,
    textAlign: 'center',
    fontFamily: 'serif',
    fontWeight: 'bold',
  },
  subtitle: {
    marginVertical: 10,
    fontSize: 17,
    fontWeight: '500',
    color: '#000033',
  }
})

```

```
    textAlign: 'center'
},
input: {
  fontSize: 12
},
inputLabel: {
  marginBottom: 15,
  fontSize: 20,
  fontWeight: '600',
  color: '#222'
},
inputControl: {
  height: 44,
  marginBottom: 15,
  backgroundColor: '#ffffff',
  paddingVertical: 10,
  paddingHorizontal: 16,
  borderRadius: 12,
  fontSize: 15,
  fontWeight: '500',
  color: '#222'
},
passwordContainer: {
  flexDirection: 'row',
  alignItems: 'center',
},
passwordInput: {
  flex: 1,
},
eyeButton: {
  flex: 0,
  marginLeft: 10,
  marginBottom: 15,
},
eyeText: {
  fontSize: 18,
},
form: {
  marginBottom: 24,
```

```
        flex: 1,
    },
    formAction: {
        marginVertical: 24,
    },
    btn: {
        backgroundColor: '#075eec',
        borderRadius: 8,
        borderWidth: 1,
        borderColor: '#075eec',
        flexDirection: 'row',
        justifyContent: 'center',
        alignContent: 'center',
        paddingVertical: 10,
        paddingHorizontal: 20,
    },
    btnText: {
        fontSize: 18,
        fontWeight: '600',
        color: '#fff',
    },
    formFooter: {
        fontSize: 17,
        fontWeight: '600',
        textAlign: 'center',
        letterSpacing: 0.3,
        paddingHorizontal: 27,
    },
    errorText: {
        color: 'red',
        marginBottom: 10,
        fontSize: 17,
        backgroundColor: 'white',
        borderRadius: 12,
        padding: 8,
        fontWeight: '400',
        alignSelf: 'flex-start'
    },
});
```

Register Screen Code:

```
import React, { useState } from 'react';
import { SafeAreaProvider, SafeAreaView } from 'react-native-safe-area-context';
import {
  StyleSheet,
  Text,
  View,
  TextInput,
  TouchableOpacity,
  Platform,
  KeyboardAvoidingView,
  ScrollView,
} from 'react-native';

import AsyncStorage from '@react-native-async-storage/async-storage';

export default function RegisterScreen({ navigation }) {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [errors, setErrors] = useState({});

  const validateForm = () => {
    let errors = {};
    if (!name) errors.name = 'Name is required.';
    if (!email) {
      errors.email = 'Email is required.';
    } else if (!/\S+@\S+\.\S+/.test(email)) {
      errors.email = 'Please enter a valid email address.';
    }
    if (!password) errors.password = 'Password is required.';
    if (password !== confirmPassword) errors.confirmPassword = 'Passwords do not
match.';
```

```

        setErrors(errors);
        return Object.keys(errors).length === 0;
    };

    const handleSubmit = async () => {
        if (validateForm()) {
            try {
                // Save user data to AsyncStorage
                await AsyncStorage.setItem('userDetails', JSON.stringify({ name, email,
password }));
                console.log('Registered:', email, password);

                // Clear the form fields
                setName('');
                setEmail('');
                setPassword('');
                setConfirmPassword('');
                setErrors({});

                // Navigate to Login page
                navigation.navigate('Login Page', {name}); //Pass name as a parameter
            } catch (error) {
                console.error('Error saving data:', error);
            }
        }
    };
}

return (
    <SafeAreaProvider>
    <SafeAreaView style={styles.container}>
        <KeyboardAvoidingView behavior={Platform.OS === 'ios' ? 'padding' :
'height'} style={{ flex: 1 }} keyboardVerticalOffset={20}>
            <ScrollView contentContainerStyle={{ flexGrow: 1 }}>
                <View>
                    <Text style={styles.title}>Register your account</Text>
                    <Text style={styles.subtitle}>Create an account to access SchoolNav
features.</Text>

```

```

        </View>

        <View style={styles.form}>
            {/* Name Input */}
            <View style={styles.input}>
                <Text style={styles.inputLabel}>Name</Text>
                <TextInput autoCapitalize="True" autoCorrect={false}
style={styles.inputControl} placeholder="John Smith" value={name}
onChangeText={setName} />
                    {errors.name && <Text
style={styles.errorText}>{errors.name}</Text>}
                </View>

            {/* Email Input */}
            <View style={styles.input}>
                <Text style={styles.inputLabel}>Email Address</Text>
                <TextInput autoCapitalize="none" autoCorrect={false}
style={styles.inputControl} placeholder="john@example.com" value={email}
onChangeText={setEmail} />
                    {errors.email && <Text
style={styles.errorText}>{errors.email}</Text>}
                </View>

            {/* Password Input */}
            <View style={styles.input}>
                <Text style={styles.inputLabel}>Password</Text>
                <TextInput secureTextEntry style={styles.inputControl}
placeholder="*****" value={password} onChangeText={setPassword} />
                    {errors.password && <Text
style={styles.errorText}>{errors.password}</Text>}
                </View>

            {/* Confirm Password Input */}
            <View style={styles.input}>
                <Text style={styles.inputLabel}>Confirm Password</Text>
                <TextInput secureTextEntry style={styles.inputControl}
placeholder="*****" value={confirmPassword} onChangeText={setConfirmPassword}
/>
                    {errors.confirmPassword && <Text

```

```

        style={styles.errorText}>{errors.confirmPassword}</Text>
    </View>

    /* Submit Button */
    <View style={styles.formAction}>
        <TouchableOpacity onPress={handleSubmit}>
            <View style={styles.btn}>
                <Text style={styles.btnText}>Register</Text>
            </View>
        </TouchableOpacity>
    </View>

    <TouchableOpacity onPress={() => navigation.navigate('Login Page')}>
        <Text style={{ marginTop: 'auto', alignSelf: 'center' }}>
            <View>
                <Text style={styles.formFooter}>Already have an account? <Text
                    style={{ textDecorationLine: 'underline' }}>Sign in</Text></Text>
            </View>
        </TouchableOpacity>
    </View>
    </ScrollView>
</KeyboardAvoidingView>
</SafeAreaView>
</SafeAreaProvider>
);
}

const styles = StyleSheet.create({
    container: {
        paddingHorizontal: 24,
        flex: 1,
        backgroundColor: '#38b6ff',
        paddingVertical: '70%',
    },
    title: {
        color: '#333333',
        fontSize: 38,
        textAlign: 'center',
        fontFamily: 'serif',
    },
})

```

```
fontWeight: 'bold',
paddingBottom: 30,
},
subtitle: {
  marginVertical: 10,
  fontSize: 17,
  fontWeight: '500',
  color: '#000033',
  textAlign: 'center',
},
input: {
  fontSize: 12,
},
InputLabel: {
  marginBottom: 15,
  fontSize: 20,
  fontWeight: '600',
  color: '#222',
},
inputControl: {
  height: 44,
  marginBottom: 15,
  backgroundColor: '#ffffff',
  paddingVertical: 10,
  paddingHorizontal: 16,
  borderRadius: 12,
  fontSize: 15,
  fontWeight: '500',
  color: '#222',
},
form: {
  marginBottom: 24,
  flex: 1,
},
formAction: {
  marginVertical: 24,
},
btn: {
  backgroundColor: '#075eec',
```

```
borderRadius: 8,  
borderWidth: 1,  
borderColor: '#075eec',  
flexDirection: 'row',  
justifyContent: 'center',  
alignContent: 'center',  
paddingVertical: 10,  
paddingHorizontal: 20,  
},  
btnText: {  
  fontSize: 18,  
  fontWeight: '600',  
  color: '#fff',  
},  
formFooter: {  
  fontSize: 17,  
  fontWeight: '600',  
  textAlign: 'center',  
  letterSpacing: 0.3,  
  paddingHorizontal: 27,  
},  
errorText: {  
  color: 'red',  
  marginBottom: 10,  
  fontSize: 17,  
  backgroundColor: 'white',  
  borderRadius: 12,  
  padding: 8,  
  fontWeight: '400',  
  alignSelf: 'flex-start',  
},  
});
```

Main Menu Screen Code:

```

import React, { useEffect, useState, useRef } from 'react';
import { Linking, Alert } from 'react-native';
import MapView, { PROVIDER_GOOGLE, Marker, Callout, Polyline } from
'react-native-maps';
import { StatusBar } from 'expo-status-bar';
import { SafeAreaProvider, SafeAreaView } from 'react-native-safe-area-context';
import {
  StyleSheet,
  Text,
  View,
  Image,
  TouchableOpacity,
  TextInput,
  Dimensions,
  Modal,
  FlatList,
  ScrollView,
  Button,
} from 'react-native';
import * as Location from 'expo-location';

// Get screen dimensions
const { width, height } = Dimensions.get('window');

// Scaling functions
const BASE_WIDTH = 375; // Reference width (e.g., iPhone 8)
const BASE_HEIGHT = 667; // Reference height (e.g., iPhone 8)
const scaleWidth = (size) => (width / BASE_WIDTH) * size;
const scaleHeight = (size) => (height / BASE_HEIGHT) * size;

// Hardcoded places
const places = [
{id: 2, name: 'Rm 26', latitude: 51.568172547493674, longitude: 0.0868942453397814},
{id: 3, name: 'Rm 25', latitude: 51.56823145571189, longitude: 0.08689513960044692},
{},
{id: 4, name: 'Rm 27', latitude: 51.56826029831658, longitude: 0.08689678819390867},
{id: 5, name: 'Rm 28', latitude: 51.568304201545864, longitude:
0.08689678835761772},
]

```

```
{id: 7, name: 'Rm 29', latitude: 51.568384227566234, longitude:  
0.08689857663338518},  
{id: 8, name: 'Rm 24', latitude: 51.568413681519125, longitude:  
0.08689678860318129},  
{id: 9, name: 'Rm 30', latitude: 51.56844146824725, longitude: 0.08689768270013776},  
{id: 11, name: 'Sports Hall', latitude: 51.568549280637356, longitude:  
0.08690841169990637},  
{id: 12, name: 'Rm 23', latitude: 51.56850037611065, longitude: 0.0870148059635456},  
{id: 15, name: 'Rm 6', latitude: 51.56838978470056, longitude:  
0.08657671196087069},  
{id: 16, name: 'Rm 5', latitude: 51.56834754872497, longitude: 0.08658028826684205 },  
{id: 17, name: 'Rm 4', latitude: 51.56834588150305, longitude: 0.0864998220782498},  
{id: 18, name: 'Rm 3', latitude: 51.56834365852768, longitude: 0.08635766508231502},  
{id: 19, name: 'Rm 2', latitude: 51.568346992933456, longitude:  
0.08623785977360132},  
{id: 20, name: 'Rm 1', latitude: 51.56834254704632, longitude: 0.08609570277766654},  
{id: 22, name: 'Main Reception', latitude: 51.56826418811395 ,longitude:  
0.08591957116149462},  
{id: 28, name: 'Rm 18', latitude: 51.568353661617074, longitude:  
0.08556373178688848},  
{id: 29, name: 'Rm 17', latitude: 51.56835532881327, longitude: 0.08540190514903845  
},  
{id: 30, name: 'Rm 16', latitude: 51.56835032717761, longitude:  
0.08526779478014834},  
{id: 33, name: 'Dining Hall', latitude: 51.5685076002505, longitude:  
0.0851408367776374},  
{id: 34, name: 'Rm 15', latitude: 51.56839645321098 ,longitude:  
0.08526868871339577},  
{id: 35, name: 'Rm 19', latitude: 51.568419794090715 ,longitude:  
0.08526779469829382},  
{id: 36, name: 'Rm 14', latitude: 51.56845202671261, longitude:  
0.08527137092241066},  
{id: 37, name: 'Rm 20', latitude: 51.568480924877015, longitude:  
0.08527584124348397},  
{id: 38, name: 'Rm 13', latitude: 51.568527606495635 ,longitude:  
0.08527673550414949},  
{id: 39, name: 'Rm 21', latitude: 51.568551503058885, longitude: 0.08528120590707733  
},  
{id: 40, name: 'Rm 12', latitude: 51.568602630514086, longitude:
```

```

0.08527852369806244},
{id: 44, name: 'Boys Toilet', latitude:
51.568752122405726,longitude:0.08535094366888263 },
{id: 45, name: 'Rm 11',latitude: 51.56869988361901,longitude:
0.08549757114959977},
{id: 46, name: 'Art Tech', latitude:
51.56869988361027,longitude:0.08563436389121382 },
{id: 47, name: 'Rm 10',latitude: 51.56869599348525,longitude:
0.08573539373682326},
{id: 49, name: 'Main Hall',latitude:51.56851871458137 ,longitude:
0.08592672475569163},
{id: 52, name: 'Rm 8',latitude: 51.568693770453066,longitude:
0.08608586934823581},
{id: 53, name: 'IT2',latitude: 51.56869265897152,longitude: 0.08624233156805605},
{id: 54, name: 'IT Tech',latitude: 51.56869377043558,longitude:
0.08630491647235505},
{id: 55, name: 'IT1',latitude: 51.56868988031005,longitude:0.08641131073599428 },
{id: 57, name: 'Girls Toilet',latitude: 51.56874767637935,longitude:
0.08646942523736545},
{id: 60, name: 'Gym', latitude: 51.56891550688693,longitude:0.08638448872769988 },
{id: 61, name: 'SCI Staff',latitude: 51.56892439853998,longitude:
0.08611358569905025},
{id: 64, name: 'Astro Turf',latitude: 51.569202817912135,longitude:
0.08631296350867235},
{id: 65, name: 'Lecture Theatre', latitude: 51.569019983695,longitude:
0.08667684983774127},
{id: 68, name: '6th Form Common Room',latitude: 51.569051660128366,longitude:
0.08694954138772193},
{id: 70, name: 'Staff Room',latitude: 51.568599851179364,longitude:
0.08656956188641818},
{id: 72, name: 'Rm 7',latitude: 51.56844480127132,longitude:0.08657939678923032 },
{id: 73, name: 'Library', latitude:51.56854927938607 ,longitude:0.0866893672769864
}

];

const nodes = [
{id:0, latitude: 51.568163655684835 ,longitude: 0.08689513935488336},
{id: 1, latitude: 51.568163655684835 ,longitude: 0.08689513935488336},

```

```
{id: 2, latitude: 51.568172547493674,longitude: 0.0868942453397814},  
{id: 3, latitude: 51.56823145571189,longitude:0.08689513960044692 },  
{id: 4, latitude: 51.56826029831658,longitude: 0.08689678819390867},  
{id: 5, latitude: 51.568304201545864,longitude: 0.08689678835761772},  
{id: 6, latitude: 51.568360886676196,longitude: 0.08689678843947224},  
{id: 7, latitude: 51.568384227566234,longitude: 0.08689857663338518},  
{id: 8, latitude: 51.568413681519125,longitude: 0.08689678860318129},  
{id: 9, latitude: 51.56844146824725,longitude: 0.08689768270013776},  
{id: 10, latitude: 51.56850259901476,longitude: 0.08690125908796364},  
{id: 11, latitude: 51.568549280637356,longitude: 0.08690841169990637},  
{id: 12, latitude: 51.56850037611065,longitude: 0.0870148059635456},  
{id: 13, latitude: 51.568355884808206,longitude: 0.08665091979818573},  
{id: 14, latitude: 51.56838533883034,longitude: 0.08665091979818573},  
{id: 15, latitude: 51.56838978470056,longitude: 0.08657671196087069},  
{id: 16, latitude: 51.56834754872497,longitude:0.08658028826684205 },  
{id: 17, latitude: 51.56834588150305,longitude: 0.0864998220782498},  
{id: 18, latitude: 51.56834365852768,longitude: 0.08635766508231502},  
{id: 19, latitude: 51.568346992933456,longitude: 0.08623785977360132},  
{id: 20, latitude: 51.56834254704632,longitude: 0.08609570277766654},  
{id: 21, latitude: 51.5683342109861,longitude: 0.08592672369158283},  
{id: 22, latitude:51.56826418811395 ,longitude: 0.08591957116149462},  
{id: 23, latitude: 51.56826641106776,longitude: 0.08584446939093215},  
{id: 24, latitude:51.56833309947904 ,longitude: 0.0858283761368428},  
{id: 25, latitude:51.56833309947904 ,longitude: 0.0858283761368428},  
{id: 26, latitude: 51.5683492158069,longitude: 0.08572377006711651},  
{id: 27, latitude: 51.56835366165524,longitude: 0.08566386745368693},  
{id: 28, latitude: 51.568353661617074,longitude: 0.08556373178688848},  
{id: 29, latitude: 51.56835532881327,longitude:0.08540190514903845 },  
{id: 30, latitude: 51.56835032717761,longitude: 0.08526779478014834},  
{id: 31, latitude: 51.56834921568767,longitude: 0.08519716324880466},  
{id: 32, latitude: 51.568351994369166,longitude:0.08515067159859502 },  
{id: 33, latitude: 51.5685076002505,longitude: 0.0851408367776374},  
{id: 34, latitude:51.56839645321098 ,longitude: 0.08526868871339577},  
{id: 35, latitude:51.568419794090715 ,longitude: 0.08526779469829382},  
{id: 36, latitude: 51.56845202671261,longitude: 0.08527137092241066},  
{id: 37, latitude: 51.568480924877015,longitude: 0.08527584124348397},  
{id: 38, latitude:51.568527606495635 ,longitude: 0.08527673550414949},  
{id: 39, latitude: 51.568551503058885,longitude: 0.08528120590707733 },  
{id: 40, latitude: 51.568602630514086,longitude: 0.08527852369806244},
```

```

{id: 41, latitude: 51.56865987097896,longitude: 0.08528299410099027},
{id: 42, latitude: 51.568662093908664,longitude:0.08535094350517358 },
{id: 43, latitude: 51.56869988367147,longitude: 0.08535183768398458},
{id: 44, latitude: 51.568752122405726,longitude:0.08535094366888263 },
{id: 45, latitude: 51.56869988361901,longitude: 0.08549757114959977},
{id: 46, latitude: 51.56869988361027,longitude:0.08563436389121382 },
{id: 47, latitude: 51.56869599348525,longitude: 0.08573539373682326},
{id: 48, latitude:51.5687009950654 ,longitude: 0.08592851286775005},
{id: 49, latitude:51.56851871458137 ,longitude: 0.08592672475569163},
{id: 50, latitude: 51.56869599345028,longitude: 0.08604027171312811},
{id: 51, latitude: 51.568738784769636,longitude: 0.08603937769802616},
{id: 52, latitude: 51.568693770453066,longitude: 0.08608586934823581},
{id: 53, latitude: 51.56869265897152,longitude: 0.08624233156805605},
{id: 54, latitude: 51.56869377043558,longitude: 0.08630491647235505},
{id: 55, latitude: 51.56868988031005,longitude:0.08641131073599428 },
{id: 56, latitude:51.568688768828416 ,longitude: 0.08648462447635286},
{id: 57, latitude: 51.56874767637935,longitude: 0.08646942523736545},
{id: 58, latitude:51.56833309947904 ,longitude: 0.0858283761368428},
{id: 59, latitude: 51.568753233625074,longitude:0.08638627692161283 },
{id: 60, latitude: 51.56891550688693,longitude:0.08638448872769988 },
{id: 61, latitude: 51.56892439853998,longitude: 0.08611358569905025},
{id: 62, latitude: 51.569009980751396,longitude:0.08637912439152462 },
{id: 63, latitude: 51.56901275937503,longitude: 0.08630759884507899},
{id: 64, latitude: 51.569202817912135,longitude: 0.08631296350867235},
{id: 65, latitude: 51.569019983695,longitude: 0.08667684983774127},
{id: 66, latitude: 51.56901553786016,longitude:0.08681006635523847 },
{id: 67, latitude: 51.56904332421962,longitude:0.08681543085512278 },
{id: 68, latitude: 51.569051660128366,longitude: 0.08694954138772193},
{id: 69, latitude: 51.568682655383554,longitude: 0.08656956180456366},
{id: 70, latitude: 51.568599851179364,longitude: 0.08656956188641818},
{id: 71, latitude: 51.568548723669366,longitude:0.0865686677894617 },
{id: 72, latitude: 51.56844480127132,longitude:0.08657939678923032 },
{id: 73, latitude:51.56854927938607 ,longitude:0.0866893672769864 }

];

const edges = [
  {from: 1,to: 2,weight: calculateDistance(nodes[1], nodes[2]) },

```

```

{from: 2,to: 3,weight: calculateDistance(nodes[ 2], nodes[ 3]) },
{from: 3,to: 4,weight: calculateDistance(nodes[3 ], nodes[ 4]) },
{from: 4,to: 5,weight: calculateDistance(nodes[ 4], nodes[ 5]) },
{from: 5,to: 6,weight: calculateDistance(nodes[5 ], nodes[ 6]) },
{from: 6,to: 7,weight: calculateDistance(nodes[6 ], nodes[7 ]) },
{from: 7,to: 8,weight: calculateDistance(nodes[7], nodes[ 8]) },
{from: 8,to: 9,weight: calculateDistance(nodes[ 8], nodes[9 ]) },
{from: 9,to: 10,weight: calculateDistance(nodes[ 9], nodes[10 ]) },
{from: 10,to: 11,weight: calculateDistance(nodes[ 10], nodes[11 ]) },
{from:10 ,to: 12,weight: calculateDistance(nodes[10 ], nodes[12 ]) },
{from: 6,to: 13,weight: calculateDistance(nodes[6 ], nodes[13 ]) },
{from: 13,to: 14,weight: calculateDistance(nodes[ 13], nodes[14 ]) },
{from: 14,to: 15,weight: calculateDistance(nodes[ 14], nodes[15 ]) },
{from: 15,to:16 ,weight: calculateDistance(nodes[15 ], nodes[16 ]) },
{from:16 ,to: 17,weight: calculateDistance(nodes[ 16], nodes[ 17]) },
{from:17 ,to: 18,weight: calculateDistance(nodes[ 17], nodes[ 18]) },
{from: 18,to:19 ,weight: calculateDistance(nodes[18 ], nodes[19 ]) },
{from: 19,to:20 ,weight: calculateDistance(nodes[ 19], nodes[20 ]) },
{from: 20,to: 21,weight: calculateDistance(nodes[ 20], nodes[21 ]) },
{from: 21,to: 22,weight: calculateDistance(nodes[ 21], nodes[22 ]) },
{from: 21,to: 24,weight: calculateDistance(nodes[21 ], nodes[24 ]) },
{from: 22,to: 23,weight: calculateDistance(nodes[ 22], nodes[23 ]) },
{from: 24,to: 26,weight: calculateDistance(nodes[ 24], nodes[ 26]) },
{from: 27,to: 26,weight: calculateDistance(nodes[ 27], nodes[ 26]) },
{from: 27,to: 28,weight: calculateDistance(nodes[ 27], nodes[28 ]) },
{from: 28,to: 29,weight: calculateDistance(nodes[28 ], nodes[29 ]) },
{from: 29,to: 30,weight: calculateDistance(nodes[29 ], nodes[ 30]) },
{from: 30,to: 31,weight: calculateDistance(nodes[ 30], nodes[31 ]) },
{from: 31,to: 32,weight: calculateDistance(nodes[31 ], nodes[32 ]) },
{from: 32,to: 33,weight: calculateDistance(nodes[ 32], nodes[33 ]) },
{from: 30,to: 34,weight: calculateDistance(nodes[30 ], nodes[34 ]) },
{from: 34,to: 35,weight: calculateDistance(nodes[34 ], nodes[35 ]) },
{from: 35,to: 36,weight: calculateDistance(nodes[ 35], nodes[ 36]) },
{from: 36,to: 37,weight: calculateDistance(nodes[36 ], nodes[ 37]) },
{from: 38,to:37 ,weight: calculateDistance(nodes[37 ], nodes[38 ]) },
{from: 38,to: 39,weight: calculateDistance(nodes[ 38], nodes[ 39]) },
{from: 39,to: 40,weight: calculateDistance(nodes[ 39], nodes[ 40]) },
{from: 40,to: 41,weight: calculateDistance(nodes[40 ], nodes[ 41]) },
{from: 41,to: 42,weight: calculateDistance(nodes[ 41], nodes[42 ]) },

```

```

{from: 42,to: 43,weight: calculateDistance(nodes[42 ], nodes[43 ]) },
{from: 43,to:44 ,weight: calculateDistance(nodes[ 43], nodes[44 ]) },
{from: 43,to: 45,weight: calculateDistance(nodes[43 ], nodes[45 ]) },
{from: 45,to: 46,weight: calculateDistance(nodes[ 45], nodes[46 ]) },
{from: 46,to: 47,weight: calculateDistance(nodes[46 ], nodes[47 ]) },
{from: 47,to: 48,weight: calculateDistance(nodes[47 ], nodes[48 ]) },
{from: 48,to: 49,weight: calculateDistance(nodes[48 ], nodes[ 49]) },
{from: 49,to: 21,weight: calculateDistance(nodes[49 ], nodes[21 ]) },
{from: 48,to: 50,weight: calculateDistance(nodes[48 ], nodes[50 ]) },
{from: 50,to: 51,weight: calculateDistance(nodes[50 ], nodes[51 ]) },
{from: 50,to: 52,weight: calculateDistance(nodes[50 ], nodes[52 ]) },
{from: 52,to: 53,weight: calculateDistance(nodes[52 ], nodes[53 ]) },
{from: 53,to: 54,weight: calculateDistance(nodes[ 53], nodes[ 54]) },
{from: 54,to: 55,weight: calculateDistance(nodes[ 54], nodes[55 ]) },
{from: 55,to: 56,weight: calculateDistance(nodes[ 55], nodes[ 56]) },
{from: 57,to: 56,weight: calculateDistance(nodes[ 57], nodes[ 56]) },
{from: 57,to: 59,weight: calculateDistance(nodes[ 57], nodes[59 ]) },
{from: 59,to: 60,weight: calculateDistance(nodes[59 ], nodes[ 60]) },
{from: 60,to:61 ,weight: calculateDistance(nodes[60 ], nodes[61 ]) },
{from: 60,to:62 ,weight: calculateDistance(nodes[60 ], nodes[62 ]) },
{from: 62,to: 63,weight: calculateDistance(nodes[62], nodes[63]) },
{from: 63,to:64 ,weight: calculateDistance(nodes[63], nodes[64]) },
{from: 62,to: 65,weight: calculateDistance(nodes[62], nodes[65]) },
{from: 65,to:66 ,weight: calculateDistance(nodes[65], nodes[66]) },
{from:66 ,to: 67,weight: calculateDistance(nodes[66], nodes[67]) },
{from:67 ,to:68 ,weight: calculateDistance(nodes[67], nodes[68]) },
{from: 56,to: 69,weight: calculateDistance(nodes[56], nodes[69]) },
{from: 70,to: 69,weight: calculateDistance(nodes[70], nodes[69]) },
{from: 70,to: 71,weight: calculateDistance(nodes[70], nodes[71]) },
{from: 71,to: 72,weight: calculateDistance(nodes[71], nodes[72]) },
{from: 72,to: 15,weight: calculateDistance(nodes[72], nodes[15]) },
{from: 71,to: 73,weight: calculateDistance(nodes[71], nodes[73]) },
{from: 73,to: 11,weight: calculateDistance(nodes[73], nodes[11]) },
{from: 59,to: 51,weight: calculateDistance(nodes[59], nodes[51]) },
{from: 51,to: 60,weight: calculateDistance(nodes[51], nodes[60]) },
//{from: ,to: ,weight: calculateDistance(nodes[ ], nodes[ ]) },
];

```

```

function calculateDistance(node1, node2) {
  const R = 6371; // Earth's radius in km
  const dLat = (node2.latitude - node1.latitude) * (Math.PI / 180);
  const dLon = (node2.longitude - node1.longitude) * (Math.PI / 180);
  const a =
    Math.sin(dLat / 2) * Math.sin(dLat / 2) +
    Math.cos(node1.latitude * (Math.PI / 180)) * 
      Math.cos(node2.latitude * (Math.PI / 180)) * 
      Math.sin(dLon / 2) * 
      Math.sin(dLon / 2);
  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
  return R * c;
}

// A* Algorithm to find the shortest path
function findPath(startId, endId) {
  const openSet = [startId]; // Nodes to be evaluated
  const cameFrom = {}; // Map to store the best path
  const gScore = {}; // Cost from start to a node
  const fScore = {} // Total cost (gScore + heuristic)

  nodes.forEach((node) => {
    gScore[node.id] = Infinity;
    fScore[node.id] = Infinity;
  });

  gScore[startId] = 0;
  fScore[startId] = heuristic(startId, endId);

  while (openSet.length > 0) {
    // Get node with the lowest fScore
    const current = openSet.sort((a, b) => fScore[a] - fScore[b])[0];
    if (current === endId) break; // Found the target

    // Remove current from openSet
    openSet.splice(openSet.indexOf(current), 1);

    // Get neighbors of the current node
  }
}

```

```

edges
    .filter((edge) => edge.from === current || edge.to === current)
    .forEach((edge) => {
        const neighbor = edge.from === current ? edge.to : edge.from;
        const tentativeGScore = gScore[current] + edge.weight;

        if (tentativeGScore < gScore[neighbor]) {
            cameFrom[neighbor] = current;
            gScore[neighbor] = tentativeGScore;
            fScore[neighbor] = gScore[neighbor] + heuristic(neighbor, endId);

            if (!openSet.includes(neighbor)) openSet.push(neighbor);
        }
    });
}

// Reconstruct path
const path = [];
let current = endId;
while (current) {
    path.unshift(current);
    current = cameFrom[current];
}

return path;
}

// Heuristic function (straight-line distance to end node)
function heuristic(nodeId, endId) {
    const node = nodes.find((n) => n.id === nodeId);
    const endNode = nodes.find((n) => n.id === endId);
    return calculateDistance(node, endNode);
}

function openExternalLink(url) {
    // Validate the URL
    if (!url) {
        Alert.alert("Error", "No URL provided.");
        return;
    }
}

```

```

}

// Check if the URL is valid
Linking.canOpenURL(url)
  .then((supported) => {
    if (supported) {
      // Open the URL if it's valid
      Linking.openURL(url)
        .catch((err) => {
          console.error("Failed to open URL:", err);
          Alert.alert("Error", "Something went wrong while opening the link.");
        });
    } else {
      Alert.alert("Error", "This URL cannot be opened.");
    }
  })
  .catch((err) => {
    console.error("Error checking URL:", err);
    Alert.alert("Error", "Something went wrong.");
  });
}

const filterPlacesByRoomNumber = (formRoom, places) => {
  return places.find(place => {
    const match = place.name.match(/\d+/); // Extract numeric part from name
    return match && parseInt(match[0], 10) === formRoom;
  });
};

const updateCommonPlaces = (formRoom, places, commonPlaces) => {
  const matchedPlace = filterPlacesByRoomNumber(formRoom, places);
  if (matchedPlace) {
    const updatedCommonPlaces = [{ id: matchedPlace.id, name: 'Form Room' },
...commonPlaces];
    return updatedCommonPlaces.filter((place, index, self) =>
      index === self.findIndex(p => p.id === place.id)); // Remove duplicates
  }
  return commonPlaces;
};

```

```

const handleOpenCommonPlacesModal = () => {
  const updatedCommonPlaces = updateCommonPlaces(formRoom, places, commonPlaces);
  setCommonPlaces(updatedCommonPlaces);
  setIsCommonPlacesModalVisible(true);
};

export default function MainMenu({ route, navigation }) {
  const { name, etaUnit } = route.params;
  const [userLocation, setUserLocation] = useState(null);
  const [locationPermission, setLocationPermission] = useState(false);
  const [selectedPlace, setSelectedPlace] = useState(null);
  const [searchQuery, setSearchQuery] = useState('');
  const [searchResults, setSearchResults] = useState([]);
  const [isModalVisible, setIsModalVisible] = useState(false);
  const [showMarkers, setShowMarkers] = useState(true); // State to toggle markers
  const mapRef = useRef(null);
  const [path, setPath] = useState([]);
  const [pathCoordinates, setPathCoordinates] = useState([]);
  const [isCommonPlacesModalVisible, setIsCommonPlacesModalVisible] =
    useState(false);
  const [isSearchLocationModalVisible, setIsSearchLocationModalVisible] =
    useState(false);
  const [formRoom, setFormRoom] = useState(route.params?.formRoom || null);
  const [reportMode, setReportMode] = useState(false); // Track report mode
  const [marker, setMarker] = useState(null); // Store the reported location
  const [description, setDescription] = useState(""); // Store user input
  const [reportModalVisible, setReportModalVisible] = useState(false); // Control modal
  const [etaValue, setEtaValue] = useState("..."); // Initially show ...
  const [commonPlaces, setCommonPlaces] = useState([
    { id: 73, name: 'Library' },
    { id: 49, name: 'Main Hall' },
    { id: 60, name: 'Gym' },
    { id: 22, name: 'Main Reception' },
    { id: 11, name: 'Sports Hall' },
    { id: 33, name: 'Dining Hall' },
    { id: 44, name: 'Boys Toilet' },
  ]);
}

```

```

        { id: 57, name: 'Girls Toilet' },
        { id: 70, name: 'Staff Room' },
        { id: 65, name: 'Lecture Theatre' },
    ]);

// Request location permissions and fetch user location with highest accuracy
useEffect(() => {
    (async () => {
        const { status } = await Location.requestForegroundPermissionsAsync();
        setLocationPermission(status === 'granted');
        if (status === 'granted') {
            const location = await Location.getCurrentPositionAsync({
                accuracy: Location.Accuracy.Highest, // Request highest accuracy
            });
            setUserLocation({
                latitude: location.coords.latitude,
                longitude: location.coords.longitude,
                latitudeDelta: 0.0004,
                longitudeDelta: 0.0004,
            });
        }
    })();
}, []);

useEffect(() => {
    if (route.params?.formRoom) {
        setFormRoom(route.params.formRoom);
        setCommonPlaces(prev => updateCommonPlaces(route.params.formRoom, places, prev));
    }
}, [route.params?.formRoom]);

const handleOpenCommonPlacesModal = () => {
    setIsCommonPlacesModalVisible(true);
};

// Handle search
const handleSearch = (query) => {
    setSearchQuery(query);
}

```

```

    if (query.trim() === '') {
      setSearchResults([]);
      return;
    }
    const results = places.filter((place) =>
      place.name.toLowerCase().includes(query.toLowerCase())
    );
    setSearchResults(results);
  };

// Center map on selected place

const goToPlace = (place) => {
  if (mapRef.current) {
    mapRef.current.animateToRegion(
      {
        latitude: place.latitude,
        longitude: place.longitude,
        latitudeDelta: 0.0004,
        longitudeDelta: 0.0004,
      },
      2000
    );
  }
  setSelectedPlace(place); // Set the selected place
  setIsModalVisible(false); // Close modal
  setSearchQuery('');
  setSearchResults([]);
};

// Function to recenter the map
const recenterMap = () => {
  if (mapRef.current && userLocation) {
    mapRef.current.animateToRegion(userLocation, 1000); // Smooth animation to
    user location
  }
};

```

```

const handleCalculateRoute = (id) => {
  const cNodeID = (findClosestNodeId(userLocation, nodes))
  goToPlace(nodes[cNodeID])
  console.log(cNodeID)
  const path = findPath(cNodeID, id); // Example: from node 1 to node 4
  const coordinates = path.map((id) => {
    const node = nodes.find((n) => n.id === id);
    return { latitude: node.latitude, longitude: node.longitude };
  });
  setPathCoordinates(coordinates);
  // Calculate the total weight of the path
  let totalWeight = 0;
  for (let i = 0; i < path.length - 1; i++) {
    const fromNodeId = path[i];
    const toNodeId = path[i + 1];

    // Find the edge between these two nodes
    const edge = edges.find(
      (e) => (e.from === fromNodeId && e.to === toNodeId) || (e.from === toNodeId
      && e.to === fromNodeId)
    );

    if (edge) {
      totalWeight += edge.weight;
    }
  }

  console.log(`Total weight of the path: ${totalWeight}`);
  const scaleFactor = 924;

  // Calculate ETA in seconds
  const etaValue = Math.round(totalWeight * scaleFactor);

  console.log(`ETA (in seconds): ${etaValue}`);
  // Set the etaValue to the calculated ETA (in seconds)
  setEtaValue(`${etaValue} Seconds`);

  // Rotate the map towards the target node
  setTimeout(() => {

```

```

rotateMapToNode(nodes[cNodeID], nodes[id]);

    // Delay second rotation after first completes
    setTimeout(() => {
        rotateMapToNode2(nodes[cNodeID], nodes[id]);
    }, 1500);
}, 1000);
};

// Function to open external link (Google Maps)
function openExternalLink(url) {
    if (!url) {
        Alert.alert("Error", "No URL provided.");
        return;
    }

    Linking.canOpenURL(url)
        .then((supported) => {
            if (supported) {
                Linking.openURL(url)
                    .catch((err) => {
                        console.error("Failed to open URL:", err);
                        Alert.alert("Error", "Something went wrong while opening the link.");
                    });
            } else {
                Alert.alert("Error", "This URL cannot be opened.");
            }
        })
        .catch((err) => {
            console.error("Error checking URL:", err);
            Alert.alert("Error", "Something went wrong.");
        });
}

// The main function
function findClosestNodeId(userLocation, nodes, places) {
    if (!userLocation || !nodes || nodes.length === 0) {
        throw new Error("Invalid userLocation or nodes array.");
}

```

```

}

let closestNodeId = null;
let shortestDistance = Infinity;

// Loop through all nodes and calculate the distance to the user location
nodes.forEach((node) => {
  const distance = calculateDistance(userLocation, node);

  // If the calculated distance is smaller than the shortestDistance, update
  if (distance < shortestDistance) {
    shortestDistance = distance;
    closestNodeId = node.id;
  }
});

// After the loop, check if the shortestDistance exceeds the distance between
nodes[11] and nodes[71]
if (shortestDistance > calculateDistance(nodes[11], nodes[71])) {
  console.log("done");
  // Trigger navigation to a specific place and open the external Google Maps
link
  //goToPlace(places[15]);
  openExternalLink("https://maps.app.goo.gl/vWdmYnToUYPwMJbYA");
  return;
}

return closestNodeId;
}

function resetRoute() {
  setPath([]);
  setPathCoordinates([])
}

const calculateBearing = (start, end) => {
  const lat1 = (Math.PI / 180) * start.latitude;
  const lat2 = (Math.PI / 180) * end.latitude;
  const lon1 = (Math.PI / 180) * start.longitude;

```

```

const lon2 = (Math.PI / 180) * end.longitude;

const dLon = lon2 - lon1;

const y = Math.sin(dLon) * Math.cos(lat2);
const x =
  Math.cos(lat1) * Math.sin(lat2) -
  Math.sin(lat1) * Math.cos(lat2) * Math.cos(dLon);
let bearing = Math.atan2(y, x) * (180 / Math.PI);

return (bearing + 360) % 360; // Normalize between 0-360
};

const rotateMapToNode = (currentNode, targetNode) => {
  if (!mapRef.current || !currentNode || !targetNode) return;

  const bearing = calculateBearing(currentNode, targetNode);

  mapRef.current.animateCamera({
    center: currentNode, // Keep centered on current location
    heading: bearing, // Rotate towards target node
    pitch: 45, // Slight tilt for a 3D effect
    zoom: 18, // Adjust zoom level
  }, { duration: 2500 }); // Smooth animation
};

const rotateMapToNode2 = (currentNode, targetNode) => {
  if (!mapRef.current || !currentNode || !targetNode) return;

  const bearing = calculateBearing(currentNode, targetNode);

  mapRef.current.animateCamera({
    center: currentNode, // Keep centered on current location
    heading: bearing, // Rotate towards target node
    pitch: 0, // Reset tilt for a 3D effect
    zoom: 25, // Adjust zoom level
  }, { duration: 1000 }); // Smooth animation
};

```

```

const handleMapPress = (marker) => {
  if (reportMode) {
    setMarker({
      latitude: marker.nativeEvent.coordinate.latitude,
      longitude: marker.nativeEvent.coordinate.longitude,
    });
    setReportModalVisible(true); // Open input modal
    setReportMode(false); // Disable report mode after selection
  }
};

const submitReport = () => {
  setReportModalVisible(false); // Close modal
};

return (
  <SafeAreaProvider>
    <SafeAreaView style={styles.container}>
      <StatusBar style="light" />

      {/* Weather and Settings Section */}
      <View style={styles.WSContainer}>
        <TouchableOpacity style={styles.WeatherOutput}>
          <Image
            source={require('./assets/weathericonsn.png')}
            fadeDuration={500}
            style={styles.WeatherImg}
            alt="Weather"
          />
        </TouchableOpacity>

        <TouchableOpacity style={styles.SettingsOutput}
          onPress={() => navigation.navigate("Settings Page", {name})}>
          <Image
            source={require('./assets/settingsiconsn.png')}
            fadeDuration={500}
            style={styles.SettingsImg}
            alt="Settings"
          />
        </TouchableOpacity>
      </View>
    </SafeAreaView>
  </SafeAreaProvider>
);

```

```
        />
      </TouchableOpacity>
    </View>

    {/* Greeting */}
    <View>
      <Text style={styles.Text}>Hello {name}>
    </Text>
  </View>

  {/* Map Section */}
  <View style={styles.Map}>

    <MapView
      ref={mapRef} // Attach the ref to the MapView
      style={StyleSheet.absoluteFillObject}
      provider={PROVIDER_GOOGLE}
      mapType="hybrid"
      userLocationUpdateInterval={1000}
      region={userLocation} // Center the map on the user's location
      showsUserLocation={true} // Show the default blue dot
      followsUserLocation={true} // Automatically follow the user's
location
      pitch={45} // Add a 3D tilt to the map
      heading={0} // Set the map's heading (0 degrees means north-up)

      onPress={handleMapPress} // Listen for user tap
    >
      {/* Render the user's location marker */}
      {userLocation && (
        <Marker
          coordinate={{
            latitude: userLocation.latitude,
            longitude: userLocation.longitude,
          }}
          title="You are here!">
      )}
    
```

```
    />
  )}

  {/* Render all place markers */}
  {showMarkers &&
    places.map((place) => (
      <Marker
        key={place.id}
        coordinate={{{
          latitude: place.latitude,
          longitude: place.longitude,
        }}}
        title={place.name} // Shows the name on marker click
      />
    )));
  }

  {/* Render polylines for edges */}
  {edges.map((edge, index) => {
    const fromNode = nodes.find((node) => node.id === edge.from);
    const toNode = nodes.find((node) => node.id === edge.to);

    // Ensure both nodes exist before rendering the polyline
    if (!fromNode || !toNode) return null;

    return (
      <Polyline
        key={index}
        coordinates={[
          { latitude: fromNode.latitude, longitude: fromNode.longitude },
          { latitude: toNode.latitude, longitude: toNode.longitude },
        ]}
        strokeColor="blue" // Color of the line
        strokeWidth={7} // Thickness of the line
      />
    );
  ))} */}

  {pathCoordinates.length > 0 && (
    <Polyline
```

```

        coordinates={pathCoordinates}
        strokeColor="rgba(66, 133, 244, 0.3)" // Light blue transparent glow
        strokeWidth={20} // Bigger for a glowing effect
    />,
    <Polyline
        coordinates={pathCoordinates}
        strokeColor="white"
        strokeWidth={16} // Thicker white outline for contrast
    />,
    <Polyline
        coordinates={pathCoordinates}
        strokeColor="#4285F4"
        strokeWidth={12} // Slightly smaller than the outline
    />
)}
```

/* Show marker if placed */

```

{marker && (
    <Marker coordinate={marker} title="Report" description={description}
/>
)}
```

</MapView>

/* Input Modal for Report Description */

```

<Modal visible={reportModalVisible} transparent animationType="slide">
    <View style={styles.modalContainer}>
        <View style={styles.modalContent}>
            <Text style={styles.searchInput}>Enter Report Details</Text>
            <TextInput
                placeholder="Describe the issue..."
                value={description}
                onChangeText={setDescription}
                style={{
                    borderWidth: 1,
                    borderColor: "black",
                    padding: 8,
                    marginBottom: 10,
                    borderRadius: 5,

```

```

        }
    />
    <Button title="Submit" onPress={submitReport} />
</View>
</View>
</Modal>

<TouchableOpacity style={styles.resetRoute} onPress={resetRoute}>
    <Text style={styles.RecenterButtonText}>Clear Route</Text>
</TouchableOpacity>
<TouchableOpacity style={styles.RecenterButton} onPress={recenterMap}>
    <Text style={styles.RecenterButtonText}>Recenter</Text>
</TouchableOpacity>
<TouchableOpacity
    style={styles.ToggleMarkersButton}
    onPress={() => setShowMarkers(!showMarkers)}
>
    <Text style={styles.ToggleMarkersButtonText}>
        {showMarkers ? 'Hide Markers' : 'Show Markers'}
    </Text>
</TouchableOpacity>
</View>

/* Features Section */
<View style={styles.Features}>
    <View style={styles.RowContainer}>
        <TouchableOpacity
            onPress={() => setReportMode(true)}
>
        <View style={styles.Box}>
            <Text style={styles.Label}>Report</Text>
        </View>
</TouchableOpacity>

        <View style={styles.ETAWrap}>
            <Text style={styles.Label}>ETA</Text>
            <View style={styles.ETABox}>
                /* Display the ETA value or "..." if no route selected */
                <Text style={styles.ETA}>{etaValue}</Text>
            </View>
        </View>
    </View>
</View>

```

```
</View>
</View>

<TouchableOpacity onPress={() =>
setIsCommonPlacesModalVisible(true)}>
  <View style={styles.Box}>
    <Text style={styles.Label}>Common Places</Text>
  </View>
</TouchableOpacity>
 {/* Modal for Common Places */}
<Modal
  visible={isCommonPlacesModalVisible}
  transparent={true}
  animationType="slide"
>
  <View style={styles.modalContainer}>
    <View style={styles.modalContent}>
      <ScrollView
contentContainerStyle={styles.scrollViewContainer}>
      {/* Preset buttons for each common place */}
      {commonPlaces.map((place) => (
        <TouchableOpacity
          key={place.id}
          style={styles.placeButton}
          onPress={() => {
            setIsCommonPlacesModalVisible(false)
            goToPlace(nodes[place.id]);
            setTimeout(() => {
              console.log(`Selected Place ID: ${place.id}`);
              setPath([]);
              handleCalculateRoute(place.id);
            }, 2000);
          }}
        >
          <Text style={styles.placeButtonText}>{place.name}</Text>
        </TouchableOpacity>
      ))}
    </ScrollView>
    <TouchableOpacity
```

```

        style={styles.closeButton}
        onPress={() => setIsCommonPlacesModalVisible(false)}
      >
      <Text style={styles.closeButtonText}>Close</Text>
    </TouchableOpacity>
  </View>
</View>
</Modal>
</View>
</View>

/* Search Location Section */
<View style={styles.SearchLocation}>
  <TouchableOpacity onPress={() =>
setIsSearchLocationModalVisible(true)}>
    <Text style={styles.SearchLabel}>Search Location</Text>
  </TouchableOpacity>
</View>

/* Modal for Searching Locations */
<Modal
  visible={isSearchLocationModalVisible}
  transparent={true}
  animationType="slide"
>
<View style={styles.modalContainer}>
  <View style={styles.modalContent}>
    <TextInput
      KeyboardAvoidingView = 'false'
      style={styles.searchInput}
      placeholder="Search for a place..."
      value={searchQuery}
      onChangeText={handleSearch}
    />
    <FlatList
      data={searchResults}
      keyExtractor={(item) => item.id.toString()}
      renderItem={({ item }) => (
        <TouchableOpacity

```

```

        style={styles.resultItem}
        onPress={() => {
          setSearchLocationModalVisible(false)
          goToPlace(item);

          setTimeout(() => {
            console.log(`Item ID: ${item.id}`)
            setPath([])
            handleCalculateRoute(item.id)
            ;
          }, 2500);
        }}

      >
        <Text style={styles.resultText}>{item.name}</Text>
      </TouchableOpacity>
    )}
  />
  <TouchableOpacity
    style={styles.closeButton}
    onPress={() => setSearchLocationModalVisible(false)}
  >
    <Text style={styles.closeButtonText}>Close</Text>
  </TouchableOpacity>
</View>
</View>
</Modal>

</SafeAreaView>
</SafeAreaProvider>
);
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#38b6ff',
    paddingHorizontal: scaleWidth(16),
  },

```

```
WSContainer: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  marginVertical: scaleHeight(16),
},
WeatherOutput: {
  flex: 1,
  justifyContent: 'center',
},
SettingsOutput: {
  flex: 1,
  alignItems: 'flex-end',
},
WeatherImg: {
  width: scaleWidth(70),
  height: scaleWidth(70),
  borderRadius: scaleWidth(35),
},
SettingsImg: {
  width: scaleWidth(60),
  height: scaleWidth(60),
  borderRadius: scaleWidth(30),
},
Text: {
  fontSize: scaleHeight(24),
  fontWeight: 'bold',
  textAlign: 'center',
  marginVertical: scaleHeight(8),
},
Map: {
  flex: 1,
  marginVertical: scaleHeight(10),
  borderRadius: scaleWidth(10),
  overflow: 'hidden',
},
RecenterButton: {
  position: 'absolute',
  bottom: scaleHeight(40),
  right: scaleWidth(5),
}
```

```
backgroundColor: '#38b6ff',
padding: scaleHeight(5),
borderRadius: scaleWidth(10),
shadowColor: '#000',
shadowOffset: { width: 0, height: 2 },
shadowOpacity: 0.25,
shadowRadius: 3.84,
elevation: 5,
borderColor: 'black',
borderWidth: 2,
},
RecenterButtonText: {
  fontSize: scaleHeight(14),
  fontWeight: 'bold',
  color: '#000',
},
ToggleMarkersButton: {
  position: 'absolute',
  bottom: scaleHeight(5),
  left: scaleWidth(5),
  backgroundColor: '#38b6ff',
  padding: scaleHeight(5),
  borderRadius: scaleWidth(10),
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.25,
  shadowRadius: 3.84,
  elevation: 5,
  borderColor: 'black',
  borderWidth: 2,
},
ToggleMarkersButtonText: {
  fontSize: scaleHeight(14),
  fontWeight: 'bold',
  color: '#000',
},
Features: {
  height: scaleHeight(120),
  borderColor: 'black',
```

```
borderWidth: 2,  
borderRadius: scaleWidth(10),  
justifyContent: 'center',  
marginVertical: scaleHeight(8),  
},  
RowContainer: {  
  flexDirection: 'row',  
  justifyContent: 'space-around',  
  alignItems: 'center',  
},  
Box: {  
  width: scaleWidth(90),  
  height: scaleHeight(100),  
  borderWidth: 2,  
  borderRadius: scaleWidth(10),  
  justifyContent: 'center',  
  alignItems: 'center',  
},  
ETAWrap: {  
  justifyContent: 'center',  
  alignItems: 'center',  
},  
ETABox: {  
  width: scaleWidth(110),  
  height: scaleHeight(90),  
  borderWidth: 2,  
  borderRadius: scaleWidth(10),  
  borderColor: 'black',  
  
  marginBottom: scaleHeight(5),  
  backgroundColor: 'white',  
},  
Label: {  
  fontSize: scaleHeight(16),  
  fontWeight: 'bold',  
},  
SearchLocation: {  
  height: scaleHeight(60),  
  borderColor: 'black',
```

```
borderWidth: 3,
borderRadius: scaleWidth(10),
justifyContent: 'center',
alignItems: 'center',
marginVertical: scaleHeight(8),
},
SearchLabel: {
  fontSize: scaleHeight(16),
  fontWeight: 'bold',
},
modalContainer: {
  flex: 1,
  backgroundColor: 'rgba(0,0,0,0.5)',
  justifyContent: 'center',
  alignItems: 'center',
},
modalContent: {
  width: '80%',
  backgroundColor: '#38b6ff',
  borderRadius: scaleWidth(10),
  padding: scaleWidth(16),
  borderWidth: 3,
  borderColor: 'black'
},
searchInput: {
  borderBottomWidth: 1,
  marginBottom: scaleHeight(10),
  fontSize: scaleHeight(16),
},
resultItem: {
  padding: scaleHeight(10),
  borderBottomWidth: 1,
},
resultText: {
  fontSize: scaleHeight(16),
},
closeButton: {
  marginTop: scaleHeight(10),
  alignSelf: 'center',
}
```

```
},
closeButtonText: {
  color: 'black',
  fontWeight: 'bold',
  fontSize: scaleHeight(16),
},
resetRoute: {
  position: 'absolute',
  bottom: scaleHeight(5),
  right: scaleWidth(5),
  backgroundColor: '#38b6ff',
  padding: scaleHeight(5),
  borderRadius: scaleWidth(10),
  shadowColor: '#000',
  shadowOffset: { width: 0, height: 2 },
  shadowOpacity: 0.25,
  shadowRadius: 3.84,
  elevation: 5,
  borderColor: 'black',
  borderWidth: 2,
},
scrollViewContainer: {
  width: '100%',
  flexDirection: 'row',
  flexWrap: 'wrap',
  justifyContent: 'center',
  paddingVertical: scaleWidth(10),
},
placeButton: {
  backgroundColor: '#38b6ff', // Blue background matching the modal content
  paddingVertical: scaleWidth(12),
  paddingHorizontal: scaleWidth(20),
  borderRadius: scaleWidth(8),
  margin: scaleWidth(8),
  alignItems: 'center',
  justifyContent: 'center',
  borderWidth: 2,
  borderColor: 'black',
  fontWeight: 'bold',
}
```

```

},
placeButtonText: {
  color: 'black',
  fontSize: 16,
  textAlign: 'center',
},CommonPlaces: {
  // Style for the section where Common Places button is
  marginTop: 20,
  fontWeight: 23,
}

},
ETA: {
  fontSize: 25, // Large font size for prominence
  fontWeight: 'bold', // Bold text
  fontFamily: 'serif', // Serif font style
  paddingVertical: 20, // Add padding around the text
  textAlign: 'center', // Center align the text horizontally
},
});

```

Settings Screen Code:

```

import React, { useState } from "react";
import { View, Text, TouchableOpacity, ScrollView, TextInput, Switch, StyleSheet, Modal, FlatList, Button, Dimensions } from "react-native";
import Icon from "react-native-vector-icons/MaterialIcons"; // Choose any icon set
import { Picker } from "@react-native-picker/picker"; // Import Picker for dropdown
// Get screen dimensions
const { width, height } = Dimensions.get('window');

// Scaling functions
const BASE_WIDTH = 375; // Reference width (e.g., iPhone 8)
const BASE_HEIGHT = 667; // Reference height (e.g., iPhone 8)
const scaleWidth = (size) => (width / BASE_WIDTH) * size;
const scaleHeight = (size) => (height / BASE_HEIGHT) * size;

const SettingsPage = ({navigation, route }) => {

```

```

const [name, setName] = useState(route.params?.name || "Guest");
const [schoolInfoExpanded, setSchoolInfoExpanded] = useState(false);
const [personalInfoExpanded, setPersonalInfoExpanded] = useState(false);
const [additionalSettingsExpanded, setAdditionalSettingsExpanded] =
useState(false);
const [formRoom, setFormRoom] = useState("");
const [error, setError] = useState("");
const [accessibilityNeeds, setAccessibilityNeeds] = useState(false);
const [theme, setTheme] = useState("Light");
const [isSearchSchoolModalVisible, setSearchSchoolModalVisible] =
useState(false);
const [searchQuery, setSearchQuery] = useState("");
const [searchResults, setSearchResults] = useState([]);
const [isDarkMode, setIsDarkMode] = useState(false);
const [modalVisible, setModalVisible] = useState(false); // Control modal
const [etaUnit, setEtaUnit] = useState("Minutes"); // Default ETA unit
const [etaDropdownVisible, setEtaDropdownVisible] = useState(false); // Control
ETA picker

const schools = [
  { id: 1, name: "Seven Kings High School" },
  { id: 2, name: "Oaks Park High School" },
  { id: 3, name: "Brampton Manor Academy" },
  { id: 4, name: "Valentines High School" },
  { id: 5, name: "NCS" },
];

const handleSearch = (query) => {
  setSearchQuery(query);
  if (query.trim() === "") {
    setSearchResults(schools); // Reset to full list if query is empty
  } else {
    const filteredSchools = schools.filter((s) =>
      s.name.toLowerCase().includes(query.toLowerCase())
    );
    setSearchResults(filteredSchools);
  }
};

```

```

const selectSchool = (schools) => {
  setIsSearchSchoolModalVisible(false);
  console.log(`Selected School ID: ${schools.name}`);
  // Implement school selection logic here
};

const validateName = (text) => {
  const nameRegex = /^[A-Za-z]+(?: [A-Za-z]+)+$/; // Ensures at least two words
  with only letters
  if (!nameRegex.test(text)) {
    setError("Enter a valid full name (first and last name, letters only).");
  } else {
    setError("");
  }
  setName(text);
};

return (
  <View style={styles.container}>
    <ScrollView showsVerticalScrollIndicator={false}>
      <Text style={styles.title}>Settings</Text>

      {/* School Information Section */}
      <TouchableOpacity onPress={() =>
        setSchoolInfoExpanded(!schoolInfoExpanded)} style={styles.button}>
        <View style={styles.buttonInner}>
          <Text style={styles.buttonText}>School Information</Text>
          <Icon name={schoolInfoExpanded ? "keyboard-arrow-up" :
            "keyboard-arrow-down"} size={24} color="black" />
        </View>
      </TouchableOpacity>
      {schoolInfoExpanded && (
        <View style={styles.dropdownContent}>
          <TouchableOpacity style={styles.subButton} onPress={() =>
            setIsSearchSchoolModalVisible(true)}>
            <Text style={styles.subButtonText}>Select School</Text>
          </TouchableOpacity>
        </View>
      )}
    </ScrollView>
  </View>
);

```

```

<TextInput
    style={styles.input}
    placeholder="Enter Form Room Number"
    keyboardType="numeric"
    value={formRoom}
    onChangeText={setFormRoom}
/>
<TouchableOpacity style={styles.subButton}>
    <Text style={styles.subButtonText}>Time-Table Information</Text>
</TouchableOpacity>
</View>
)}

/* Modal for Searching Schools */
<Modal visible={isSearchSchoolModalVisible} transparent={true}
animationType="slide">
    <View style={styles.modalContainer}>
        <View style={styles.modalContent}>
            <TextInput
                style={styles.searchInput}
                placeholder="Search for a school..."
                value={searchQuery}
                onChangeText={handleSearch}
            />
            <FlatList
                data={searchResults}
                keyExtractor={(item) => item.id.toString()}
                renderItem={({ item }) => (
                    <TouchableOpacity
                        style={styles.resultItem}
                        onPress={() => selectSchool(item)}
                    >
                        <Text style={styles.resultText}>{item.name}</Text>
                    </TouchableOpacity>
                )}
            />
            <TouchableOpacity style={styles.closeButton} onPress={() =>
setIsSearchSchoolModalVisible(false)}>
                <Text style={styles.closeButtonText}>Close</Text>

```

```

        </TouchableOpacity>
    </View>
</View>
</Modal>

/* Personal Information Section */
<TouchableOpacity onPress={() =>
setPersonalInfoExpanded(!personalInfoExpanded)} style={styles.button}>
    <View style={styles.buttonInner}>
        <Text style={styles.buttonText}>Personal Information</Text>
        <Icon name={personalInfoExpanded ? "keyboard-arrow-up" :
"keyboard-arrow-down"} size={24} color="black" />
    </View>
</TouchableOpacity>
{personalInfoExpanded && (
    <View style={styles.dropdownContent}>
        <Text style={[styles.subButtonText, { color: "white", textAlign:
"left", paddingHorizontal: 10, paddingVertical: 5,}]}>Name:</Text>
        <TextInput
            style={[styles.input, error ? styles.inputError : null]}
            placeholder={name === "Guest" ? "Enter your full name" : "Change
your name?"}
            value={name}
            onChangeText={validateName}
            autoCapitalize="words"
        />
        {error ? <Text style={styles.errorText}>{error}</Text> : null}
        <View style={styles.toggleContainer}>
            <Text style={[styles.subButtonText, { color: "white"
}]}>Accessibility Needs</Text>
            <Switch value={accessibilityNeeds}
onValueChange={setAccessibilityNeeds} />
        </View>
    </View>
)
/* Additional Settings Section */
<TouchableOpacity onPress={() =>
setAdditionalSettingsExpanded(!additionalSettingsExpanded)} style={styles.button}>
    <View style={styles.buttonInner}>

```

```

        <Text style={styles.buttonText}>Additional Settings</Text>
        <Icon name={additionalSettingsExpanded ? "keyboard-arrow-up" :
"keyboard-arrow-down"} size={24} color="black" />
    </View>
</TouchableOpacity>
{additionalSettingsExpanded && (
    <View style={styles.dropdownContent}>
        <TouchableOpacity onPress={() => setTheme(theme === "Light" ? "Dark" :
"Light")}>{styles.subButton}>
            <Text style={styles.subButtonText}>Theme: {theme}</Text>
        </TouchableOpacity>
    </View>
    <TouchableOpacity style={styles.subButton}
onPress={() => setEtaDropdownVisible(true)} // Show dropdown on press
    >
        <Text style={styles.subButtonText}>ETA Measure</Text>
    </TouchableOpacity>
</View>
)};

/* Dropdown Modal for ETA Unit Selection */
<Modal visible={etaDropdownVisible} transparent animationType="slide">
    <View style={styles.modalContainer}>
        <View style={styles.modalContent}>
            <Text style={styles.searchInput}>Select ETA Unit</Text>
            <Picker selectedValue={etaUnit} onValueChange={(itemValue) =>
setEtaUnit(itemValue)}>
                <Picker.Item label="Hours" value="Hours" />
                <Picker.Item label="Minutes" value="Minutes" />
                <Picker.Item label="Seconds" value="Seconds" />
            </Picker>
            <Button title="Confirm" onPress={() => setEtaDropdownVisible(false)}>
                </View>
            </View>
        </Modal>

<TouchableOpacity onPress={() => navigation.navigate('Main Menu', {name,

```

```
etaUnit})}
      style={{ marginTop: 'auto', alignSelf: 'center' }}
      disabled={!error}>
    <View>
      <Text style={styles.formFooter}>Return to Main Menu?</Text>
    </View>
  </TouchableOpacity>
</ScrollView>
</View>
);
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#38b6ff",
    paddingHorizontal: 16,
    paddingVertical: 40,
  },
  title: {
    color: "#333333",
    fontSize: 38,
    textAlign: "center",
    fontFamily: "serif",
    fontWeight: "bold",
    textDecorationLine: 'underline', // Underlines the text
    marginVertical: 30,
  },
  buttonContent: {
    flexDirection: "row", // Align text and icon horizontally
    alignItems: "center", // Center vertically
  },
  button: {
    backgroundColor: "#38b6ff",
    padding: 15,
    borderRadius: 10,
    borderWidth: 2,
    marginVertical: 8,
  },
});
```

```
buttonText: {
  textAlign: "center",
  fontSize: 22,
  fontWeight: 'bold',
},
dropdownContent: {
  backgroundColor: "rgba(0,0,0,0.5)",
  padding: 10,
  borderRadius: 10,
  marginVertical: 8,
},
subButton: {
  backgroundColor: "#38b6ff",
  padding: 10,
  borderRadius: 8,
  marginVertical: 5,
},
subButtonText: {
  textAlign: "center",
  fontSize: 18,
},
input: {
  backgroundColor: "white",
  padding: 10,
  borderRadius: 8,
  marginVertical: 5,
  borderWidth: 2,
  borderColor: "black",
  fontSize: 14,
},
toggleContainer: {
  flexDirection: "row",
  justifyContent: "space-between",
  alignItems: "center",
  padding: 10,
},
formFooter: {
  fontSize: 17,
  fontWeight: '600',
```

```
    textAlign: 'center',
    letterSpacing: 0.3,
    padding: 20,

}, SearchLocation: {
    height: scaleHeight(60),
    borderColor: 'black',
    borderWidth: 3,
    borderRadius: scaleWidth(10),
    justifyContent: 'center',
    alignItems: 'center',
    marginVertical: scaleHeight(8),
},
SearchLabel: {
    fontSize: scaleHeight(16),
    fontWeight: 'bold',
},
modalContainer: {
    flex: 1,
    backgroundColor: 'rgba(0,0,0,0.5)',
    justifyContent: 'center',
    alignItems: 'center',
},
modalContent: {
    width: '80%',
    backgroundColor: '#38b6ff',
    borderRadius: scaleWidth(10),
    padding: scaleWidth(16),
    borderWidth: 3,
    borderColor: 'black'
},
searchInput: {
    borderBottomWidth: 1,
    marginBottom: scaleHeight(10),
    fontSize: scaleHeight(16),
},
resultItem: {
    padding: scaleHeight(10),
    borderBottomWidth: 1,
```

```

    },
    resultText: {
      fontSize: scaleHeight(16),
    },
    closeButton: {
      marginTop: scaleHeight(10),
      alignSelf: 'center',
    },
    closeButtonText: {
      color: 'black',
      fontWeight: 'bold',
      fontSize: scaleHeight(16),
    },
    inputError: {
      borderColor: "red"
    },
    errorText: {
      color: "red", marginTop: 5
    },
  });
}

export default SettingsPage;

```

Screens Navigation Stack (App.js) Code:

```

import { NavigationContainer } from "@react-navigation/native";
import { createNativeStackNavigator } from "@react-navigation/native-stack";
import LoginScreen from "../SchoolNav/screens/LoginScreen";
import RegisterScreen from "../SchoolNav/screens/RegisterScreen";
import MainMenu from "../SchoolNav/screens/MainMenu";
import Settings from "../SchoolNav/screens/SettingsScreen";

const Stack = createNativeStackNavigator()

export default function App() {
  return(
    <NavigationContainer>
      <Stack.Navigator screenOptions={{ animationEnabled: true }}>

```

```
<Stack.Screen
  name="Login Page"
  component={LoginScreen}
  options={{ headerShown: false }} // Hide the header
/>
<Stack.Screen
  name="Register Page"
  component={RegisterScreen}
  options={{ headerShown: false }} // Show the header
  initialParams={{

}}
/>
<Stack.Screen
  name="Main Menu"
  component={MainMenu}
  options={{ headerShown: false }} // Hide the header
  initialParams={{
    name: 'Guest',
}}
/>
<Stack.Screen
  name="Settings Page"
  component={Settings}
  options={{ headerShown: false }} // Hide the header
  initialParams={{
    name: 'Guest',
}}
/>
</Stack.Navigator>
</NavigationContainer>
)
}
```

Sources

Assisting with the Login Page and Register Page:

- ▶ #1 Create a Simple Login Screen in React Native
- ▶ React Native Login Authentication PART 1 (step-by-step tutorial)
- ▶ Signup, Login and Welcome Screen in React Native || React Native Login & Sign...

Navigation between different Screens:

- ▶ React Native Tutorial - 74 - Navigation between Screens

I used inspiration and small snippets from this tutorial:

- ▶ React Native Tutorial for Beginners - Build a React Native App

Github page which informed me of the different map functions I can use:

[react-native-maps/docs/mapview.md at master](https://react-native-maps.github.io/docs/mapview.md)

I also used these websites to assist me with the locations of each point/corridor/turning point for the precise coordinates:

[Find GPS coordinates on Google maps | Latitude Longitude Search](https://www.google.com/maps/search/?api=1&q=1600+Ames+St,+Washington,+DC)

[GPS Coordinates](https://www.latlong.net/)

Mapping Algorithm:

[Distance on a sphere: The Haversine Formula - Esri Community](https://www.esri.com/arcgis-blog/arcgis-enterprise/2017/06/using-the-haversine-formula-to-calculate-distance-on-a-sphere/)

Which has Python code which I used in conjunction with my understanding of the Haversine formula.

[A* Pathfinding in React JS Web Development](https://www.geeksforgeeks.org/a-pathfinding-algorithm-in-react-javascript-development/)

Additionally I utilised [ChatGPT](https://chat.openai.com/) to help me with the A* Algorithm to format the code in React Native.

