300524008 Joel Chu Assignment 2

React Native Review

Having briefly used react in the past, I came into building this app already having a general understanding of how react works. This helped me when it came to creating the file structure to begin the project. I knew that pages, components and styles had to be separated, this made it easier to organise the whole project.

Styling:

The react native syntax was fairly easy to follow, using mainly View and Text components throughout the app, this kept the files for the pages very clean and made it easy to read. Doing some research on project structure, there was a general consensus that the Stylesheet files were kept separate from the file with the page structure. I made a stylesheet for each page, so I had different styles for different components in the page. This kept the stylesheet page tidy as each style belong to each page's stylesheet meaning I was able to decouple styles between each page. Having to only use one import to reference each stylesheet was very nice as it kept the imports clutter-free. One of the problems I encountered during my development was the syntax within the styles. I knew that react native styling syntax was similar to CSS but there were still some differences. Things like text-align in css would be textAlign in React Native. This made things more confusing that I often found myself looking at the stylesheets wondering if I was using the correct syntax or not. Another problem that I found with the styling was that some styling functions could not be used. In CSS I would normally use margin-auto: "center" to center divs. I often used this command in React Native out of force of habit and I would get confused when it told me that the syntax was invalid.

Language

I used JavaScript for my app due to simplicity since JavaScript is the default language. JavaScript is also one of the most popular programming languages which could contribute to React Natives popularity. React Native supports development in Typescript and Flow. An advantage of this is that users have the choice between languages. If I was to pick a language to develop the app in, in hindsight, I would have done it in TypeScript. The disadvantage of supporting multiple languages is that instead of having one large community, it is split into smaller communities split by each language. This would mean that there are less online resources for each language compared to if there was only one supported language. Framework developers need to very careful in picking the languages that they support. React Native supports JS, TS and Flow, these languages are all very similar so documentation and resources would all be very similar, meaning some documentation can be shared between the languages.

Hooks:

Hooks are a big part of the React Native framework, they handle state between and within classes and functions. I have never used hooks before do it took me some time to learn the concept before I could start using it. I mainly used useState and useEffect as they were simpler to understand than the other hooks. Due to the simplicity of my App, there was not much need for state storage so the simpler hooks worked fine for me. In my opinion, since I'm not familiar with hooks, they did confuse me but if I did some more react native development, this would not be an issue.

Functions vs Classes:

In my application, I used functions instead of classes for my pages. I was a confused when I was that on the react native documentation site, it almost appeared that they could both be used interchangeably. After doing some research, it appeared that most people were using functional components for their applications so I followed them. After having made the Application, I now know that this is not the case. Since there was not much state storage, using functions worked for me, although a more complicated application with more state would have to use class components. It didn't help that when I was looking online for help, some people had solutions which were made using class components so I had to ignore their solutions. For expandability, using a class component would have made things easier I the future. It would be good to have some sort of more concrete standard to use either class and functional components and their purposes. Even though functional components are meant to be stateless, it is still possible to give them state by implementing hooks to store state. It would have been good to have a clear boundary between the two components.

Community:

Being one of the most popular mobile development frameworks, there is a large online community which made finding resources and tutorials very easy. I encountered a lot of problems when I was setting up the environment and starting the project. I was able to google the error messages that I was getting and there was almost always another person who had come across the same problem as me. This made it easy to find bugs and fix them. Since the community is so large, I could also search for examples and tutorials for just about anything. I was researching a way to stack all the transport screens within the same bar tab and I wanted the settings tab to always be available. It seemed to be a problem that many other had. Due to the size of the community, I was able to find a solution that I was happy with there I could nest a stack navigator inside of a tab navigator.

React native is an open source framework. This means that people are able to view the source code and modify it to make it better. With a large userbase and open source software, it will also reduce the number of bugs since there is a good change that bugs will be discovered by the community. Open source software in general is a good idea. Development frameworks should follow this approach and make their software open source. It tends to make their software more secure due to its nature.

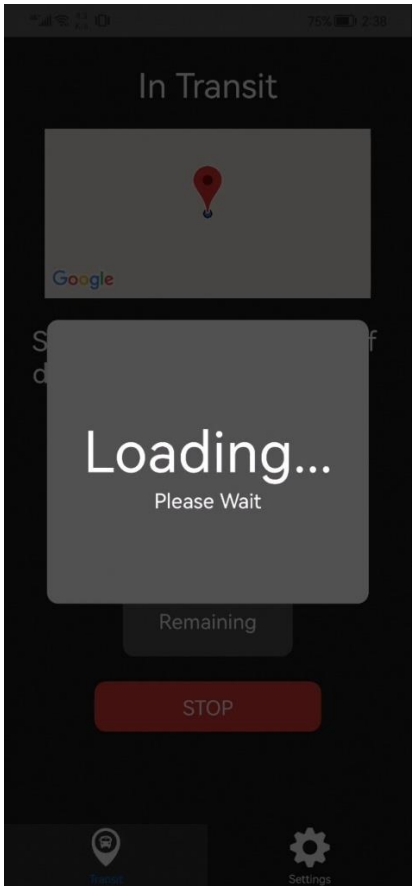
Documentation:

I was impressed by React Natives documentation. I often went on the react native docs website to read the documentation when I wasn't sure how to use their components. A framework with good documentation makes the framework much easier to learn and easier to use. The layout of the website is tidy and easy to read. A bonus was their addition of example code. There would often be example code for the components or API's. This helped me to understand the code better and it often makes more sense seeing code used in context. The website also had an extension which could run the example code snippets in expo. This gave me a visual representation what I could interact with. React native does a great job on providing comprehensive documentation for all their components, APIs and architecture.

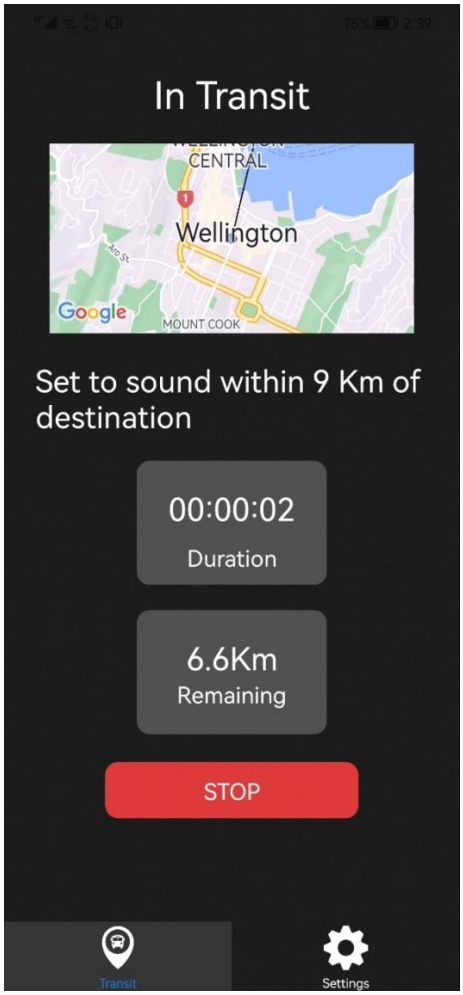
Dependencies/libraries:

React Native does a good job of making a library with all the basic components needed for common components needed in most applications. Due to React Natives community, there is a large number of libraries created by the React Native community. Whenever there is a component that does not exist, there is almost always a library for the component. I have used libraries made by others for location, audio, and modals. This made development much easier compared to if I had to create the components myself. More complicated components are usually made by private users and not organisations. Out of 31 dependencies, 18 of them are not made by organisations. These libraries support functionalities that are not used as much and are only used for specific use cases. More commonly used functionalities are implemented in the React Native library. The problem with letting users create their own libraries is that there is an amount of trust that needs to be made when using their libraries. Each dependency will also have interdependencies which it will rely on creating a large web of dependencies. At any point, the users managing the libraries can edit their code and break applications. Even for such a small application, I have so many dependencies, for larger applications, this number will only grow. Luckily with npm, when users use the npm repo, it often tells them if there are any known vulnerabilities associated with the libraries. Security features like these will give users more piece of mine when they use the libraries. Framework designers need to be aware of external libraries and the possible security risks associated with them. Especially for open source software like React Native, this could be a huge problem without the right security measures in place.

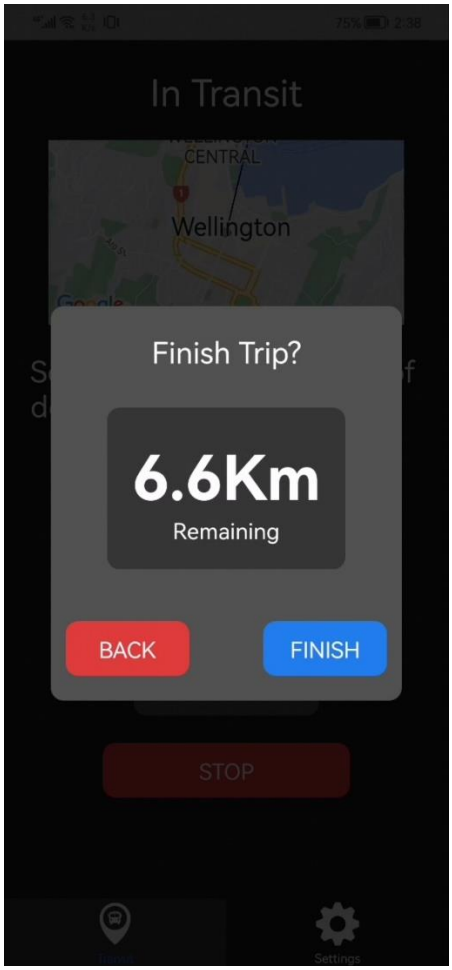
Home screen contains a header with the words “Select your destination”. The user is then able to interact with the map which takes up the majority of the screen. The map component is the main feature of the screen. Since the map is small, it had to be the biggest component on the screen, The use is able to drag the pointer to their desired location. From the home page they can navigate to the settings page. When the user clicks start journey, it will take them to the transit page



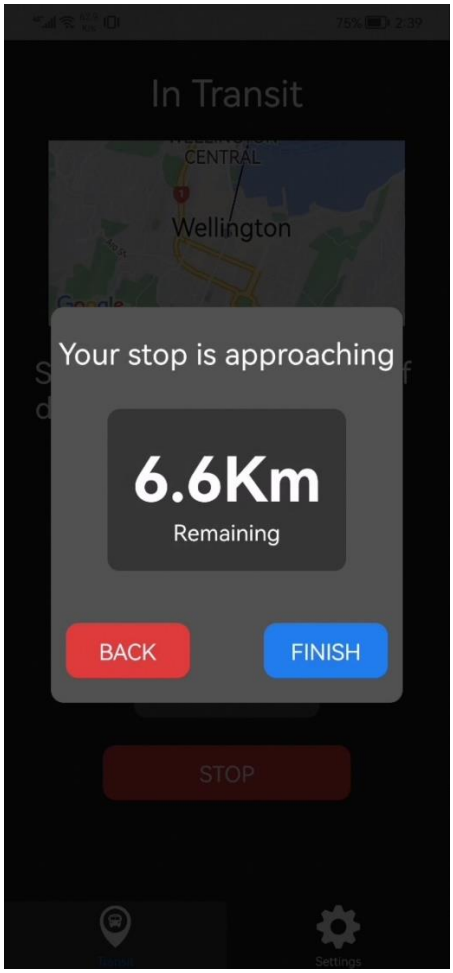
When the user first opens the transit screen, they are greeted by a loading screen. This is because the transit page needs to retrieve data from the firebase connection to work. Once the data is retrieved, the loading screen does away.



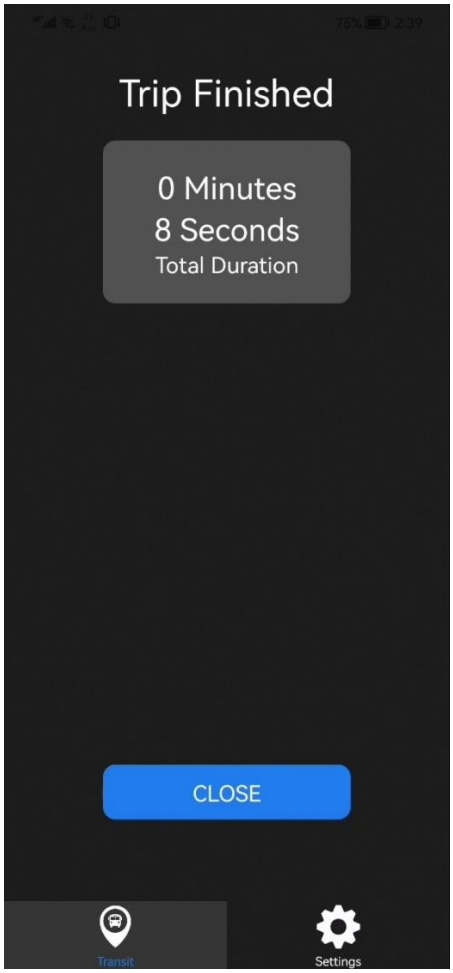
In the transit screen, the user is shown an interactive map. There is a line drawn between the users current location and their destination. The text Set to sound within 9 km of destination is derived from the users settings. The text changes depending on the user settings. The user is shown the duration of the trip as well as the distance from their current location and the destination. From this screen, when the user is within a certain radius of the destination, there will be a pop-up screen. There is another pop-up screen for when the user manually stops the trip.



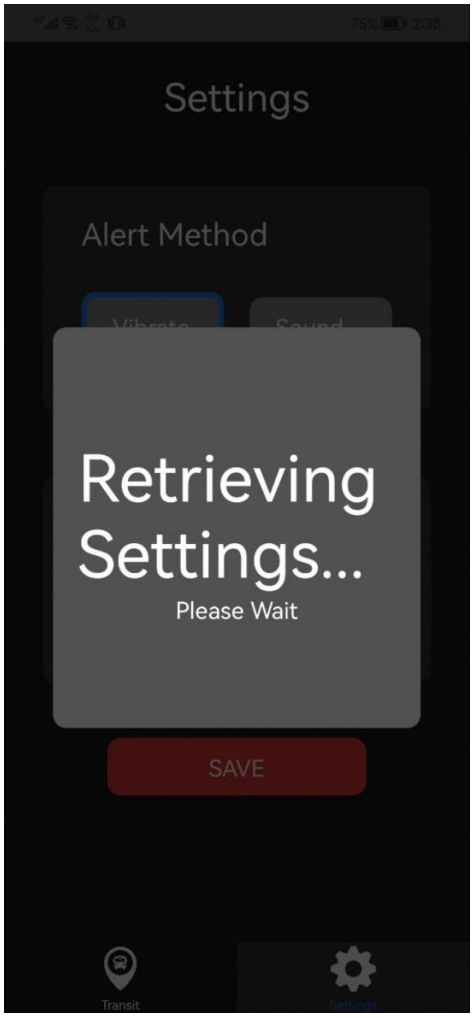
When the user manually stops the trip, they are shown a confirmation screeen asking if they are sure they want to finish the trip. The distance remaining is displayed as well as two buttons; back and finish. Clicking the back button returns the user to the previous transit screen. Clicking finish will end the trip and will take the user to the summary page.



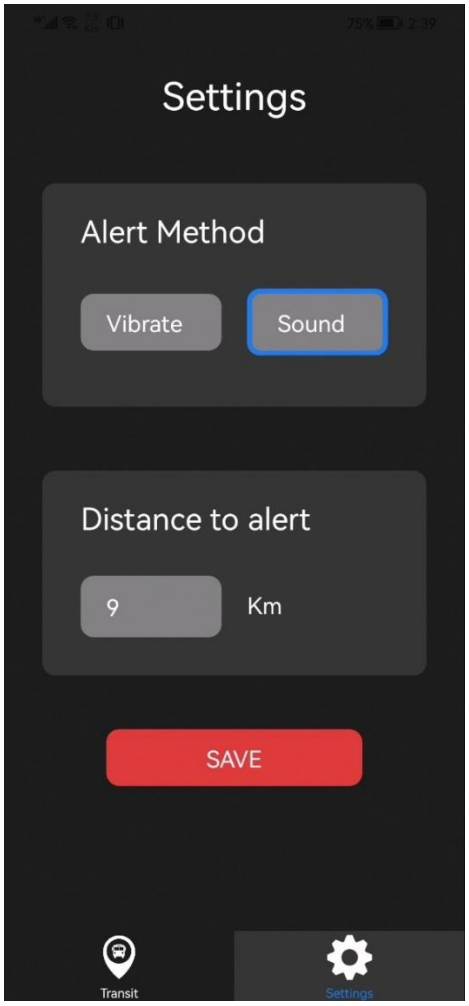
This screen is shown when the user is in transit and they are within the set radius of the destination. If the user has set vibrate for their mode of transport. If the user has selected the sound setting, a song will be played in this screen. This is to wake up the user or get the users attention. The ringing or vibration will stop when either back or finish is pressed.



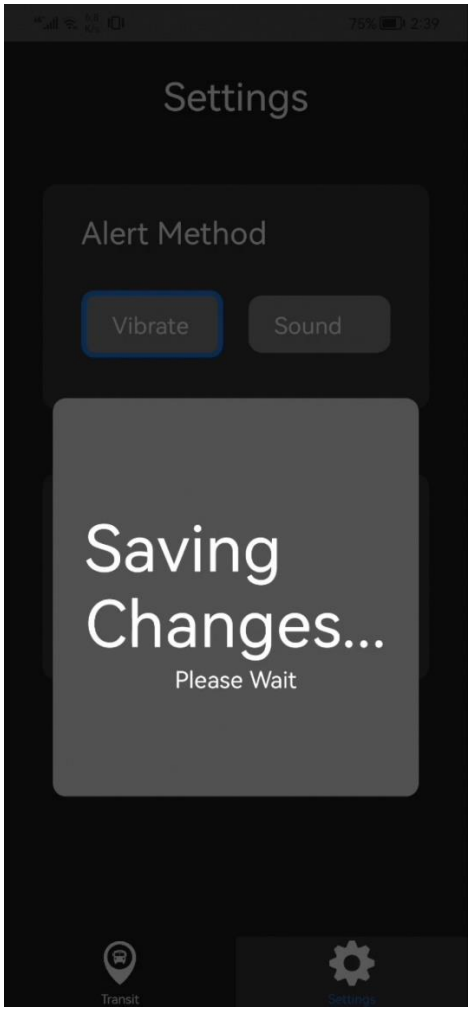
This is the Trip finished screen. The trip finished screen is displayed when the user has ended their transit. The trip finished page displays a summary of the trip. At the moment, the duration in minutes and seconds is displauyed to the user. Pressing the close button will take the user back to the first page (Select your destination).



From any transit screen, the user is able to enter the settings page. When they tap the settings screen the first time, there will be a loading screen for when the application is retrieving the settings for the screen. The retrieving settings page will go away when the settings have been retrieved from the firebase database.



In the settings screen, the user is able to select their alert method and set the distance to alert. In the Alert Method section, the selected method has a blue colour border. The user is able to select either vibrate or sound as their alert method. In the Distance to alert section, they can enter their preferred distance from the destination until they want to be alerted. The distance is in Km. When the user presses save, their current settings will be saved and they will be shown a saving screen.



The users changes are only saved if they tap the save button. When they do this, the current selected settings are sent to the firebase database and the alert method and distance are updated in the database. When this is complete. The saving changes screen will close and they will return to the settings screen.