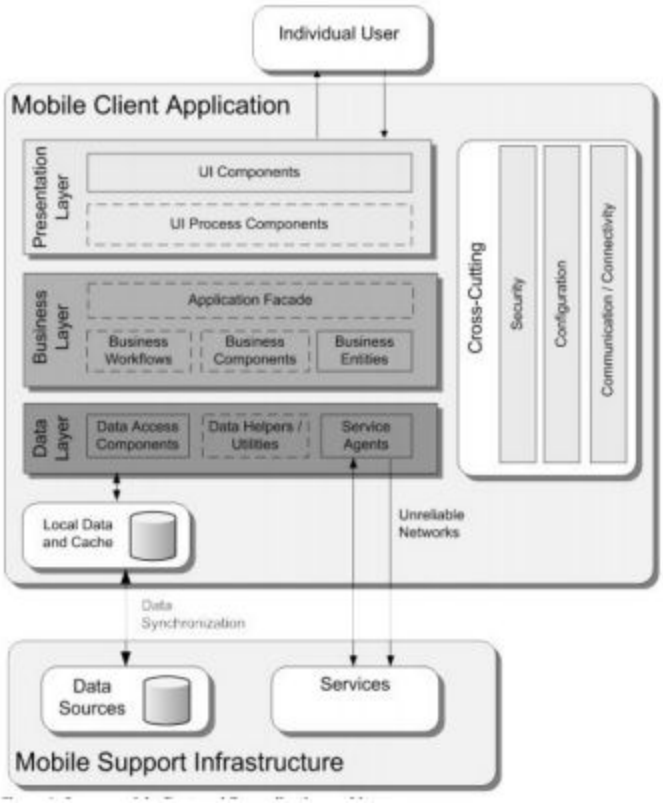
SWEN 325

Gabriel Tennent
300393699
Assignment 2 Report

Overall application architecture:

During the design of this mobile application I tried my best to follow the guidance of the microsoft - “Mobile Application Architecture Guide” linked in the lecture slides. This design splits the application into three primary parts, the presentation layer, business layer, and data layer. Each layer is independent and is only aware of and communicates with the neighbouring layers, which inturn helps prevent coupling. The three layers are unique in their purpose and paired together provide the functionality needed for a mobile application. Each layer being unique and independent in turn makes modifications to each layer easier, because you can modify a specific aspect of the mobile application without affecting others.



For example, if you wanted to modify the data layer and introduce a new storage system other than the temporary storage of data to the application (which I did) you do not want to have also modify the presentation layer to account for this change in the data layer. Using this design application you will change the way you are accessing the data and even where the data is stored but the data being provided will still be the same and inturn will still be presented the same way without changes. If you were not using this application architecture changing the data layer could affect the presentation of the data and in turn lead to huge amounts of time being wasted on adjusting the code for the presentation as aspects of the presentation were

dependant on the previous data system. Ideally I wanted as much of the code to not be dependant on other pieces of code as possible.

The application I designed was a rich client application, meaning that the data and business services were located locally on the users device itself. I tried my best to follow the architecture described above and managed to split a lot of the displaying of data into their own components/layers but struggled to completely split the data service from the page where all of the display data for the home page is returned.

The presentation layer is what the user sees and interacts with on their mobile device. In my application this was the return method of all components I developed as I failed to split the functions used by the return methods into their own components entirely. I did however split the input of information that will be stored permanently into its own component - HabitInput.js. This component did not have direct access to the business/data layer. When designing the presentation layer as a whole I tried to take into account who would be using my application and the environment they would be using it in. With this in mind I made the application navigation self explanatory and simple by using buttons that described their purpose through simple images. Visually I tried to make the application simple and easy to understand by reducing clutter and having lots of white space which in turn guides the eyes towards the aspects of the application that displays the relevant information.

The business layer and storage layer of the application architecture were merged quite heavily. If I was to continue working on this application I would spend more time separating the layers accordingly by introducing an entirely new component for all of the permanent data storage functions. These layers consisted of the functions located in the HabitPage.js. These functions handled the data and the permanent storage of the data. Some functions took the data inputted and stored it in temporary storage. Other functions were used to take the temporary data and format it for permanent storage. Further functions in this component were used to retrieve the data from permanent storage for temporary storage. The presentation layer of the application utilised the temporary storage when displaying the data.

Description on external component:

The external component I used was the React Native ASyncStorage system that allowed for permanent storage of data processed on the application pages to the users device. This was imported into the application in the command line and then imported into the pages using imports located at the top of the .js pages. The data layer that made a connection between the storage component and the business layer was made by adding functions to the HomePage.js file. These functions provided all of the accessibility for storing and retrieving data from the component, and had the relevant methods for retrieving all of the data stored inside of the storage component for list display on the presentation layer. Functions were also developed to delete specific items when the user completes or wants to remove the information stored.

This implementation external component was satisfactory as it functioned correctly allowing me to update the page with new data without reloading the page and did not lead to me needing to rewrite code in the presentation layer. But the functions of the data layer were slightly integrated with the business layer which was no ideal and sometimes lead to rewriting code where I could have prevented it. Furthermore the code was harder to digest than if I had the data layer functions in their own separate component.

React Native framework - Advantages/Disadvantages:

React Native was a framework I found confusing at first, but with further learning and time it felt like a framework with a lot of possibilities. Initially the framework felt intimidating but with time the pros of this framework greatly outway the cons.

Advantages:

- All components and especially style sheets can be used for multiple display components. This is awesome because it helps prevent the coder from having to repeat themselves.

- Components have a very large range of customization. I was not very component at developing the CSS for this but the potential is there to make complex and very aesthetically pleasing displays. Simple but attractive displays are also easy to develop.

- Online community is really good and well supported. I found information on almost every bug/problem I had with my application that helped me resolve issues quickly.

- React native uses native components for each mobile platform. So you can develop an application for both Android and IOS at the same time. Since the components are native to each device they will look slightly different but follow and identical structure and look very professional.

- The simulation of the application was very easy to add to an Android emulator of any type. I found the emulator connection a click away and when using React Native Expo it automatically installs the required software on your emulator to emulate. This was an awesome attribute because instead of looking at the application through the web in a phone format I could see the application on a simulated phone.

- Visuals and logic of the application is compact together. The embedded logic and visuals allows for dynamic changes without side effects as the framework has low coupling.

- Error messages where clearly displayed and easy to follow. As opposed to Ionic where I found it unclear where the error was originating from I was almost always provided with code line and code string. Making debugging a lot easier.

Disadvantages:

The react native framework is constantly being updated which has more pros than cons. But couple that with the fact a large change was brought to the framework a little over 6 months ago and you find a lot of information provided by the community online to be out of date. This lead to hours spent researching and finding that what I had learnt was no longer supported. A good example of this was the movement from classes to functional components changing a lot of the

way .js files are designed. Almost all old guides containing classes were not really applicable when designing my application.

Learning React native was harder than Ionic as the structure of the application was less established at the start. This is good because you have more freedom but it made learning the framework more of a struggle. Furthermore learning functional components was the new process over classes was disheartening. I do think they are a good addition to application development and I am glad I did but they were very frustrating at first and lead to countless hours spent learning a system that would of been a lot easier if it had just used classes that I already have experience with.

Conclusion:

The disadvantages I had with the framework were more learning struggles. I really like the framework and if I was to pick between Ionic and React Native I would choose to continue using React Native as I think it provides you application with more potential.

3 UX Decisions:

Left -> Right (Right being final product)

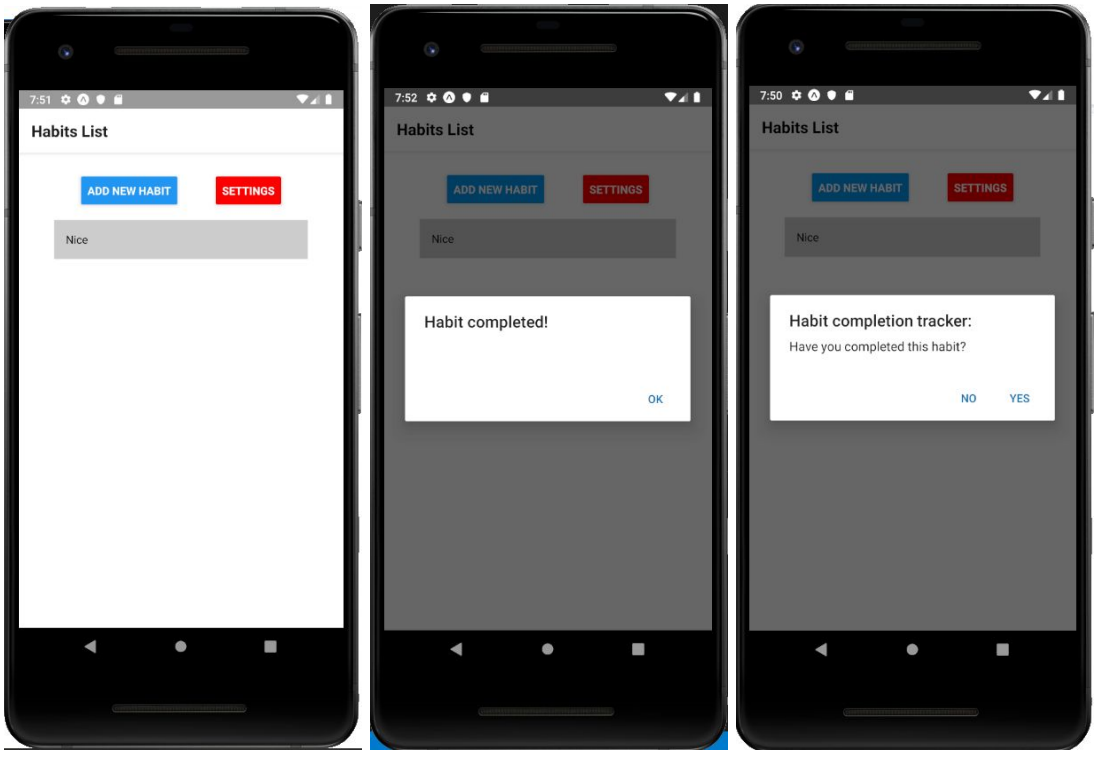
1st UX design decision:

A UX decision I made was about how the user is notified that they have completed a habit. This UX decision had a large impact on the feeling received from the application. I utilised this notification process on a couple of my pages but it was most significantly noticed on the home page.

Initially tapping the habit just removed it without any indication of what happened. This was a problem because the user could accidentally tap the habit without realising causing it to disappear and therefore leading to confusion. This was amplified when many habits are on the screen as the list is scrollable to see all of the habits in the list and well scrolling the user may forget to slide their finger in turn just tapping the habit and removing it.

The second option notified the user that the habit has been completed and therefore removed from the list. But still didn't prevent the user from accidentally completing a habit when using the application.

The third and final notified the user that a habit was being completed but then gave the option for the user to not follow through if they had accidentally completed the habit. This check was also applied to the settings page delete all habits action, so that users could backtrack if they had accidentally tapped the button.



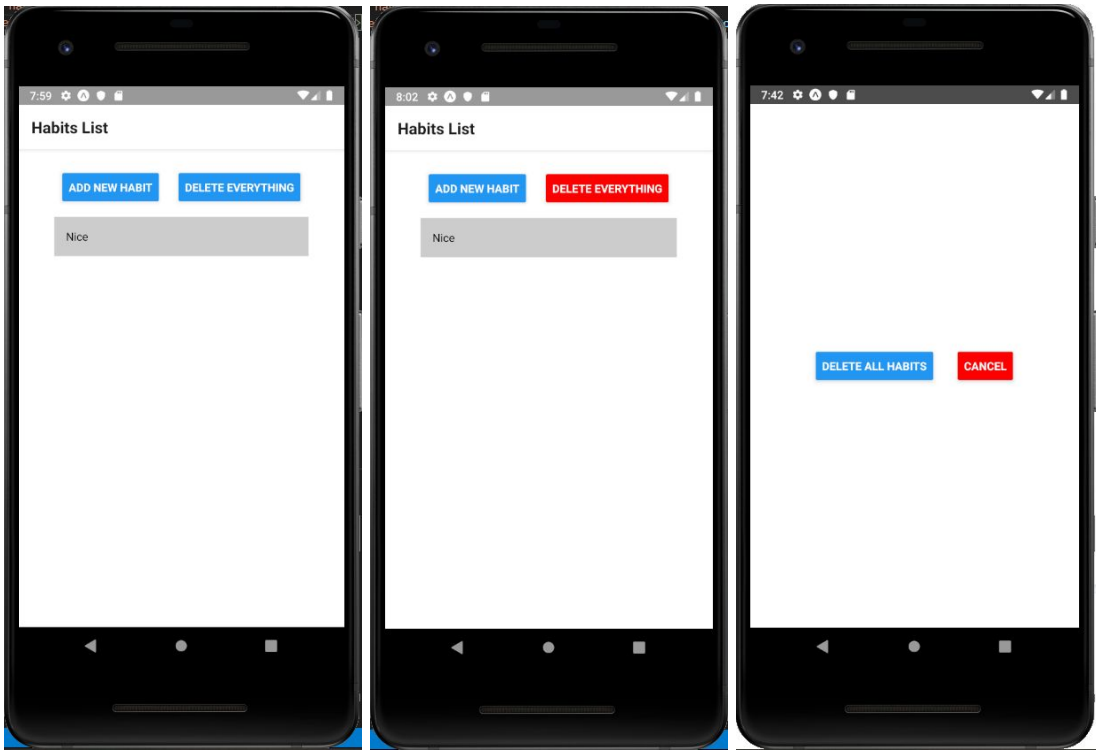
2nd UX design decision:

Another UX decision I made was with button placement. This decision once again had an impact on the entire application as there were buttons through it all. But was most clearly scene with the wipe all data button.

Initially the delete everything button was located next to the add new habit button but was something that was not going to be pushed very often and yet made it seem like a more prominent part of the application. It could also be accidentally pushed which lead to wiping all the data by accident.

The second option I went with clearly defined that this button had a severe impact on the current state of the application because it was labeled red. But it still seemed like an odd place to have a delete everything button - right next to the add habit button. It could still also be pushed accidentally which was not ideal.

The final option I went with was moving the button into its own page. Then replacing the delete everything button with a settings page button. This page in the future could contain more information and options for the application making its placement suitable. Making the user navigate to the settings page so they could find the delete everything button felt right as it is an action you would find most commonly in a settings page. Coupling this with the design decision above about notifications meant that the user could no longer accidentally delete all of the data currently stored.



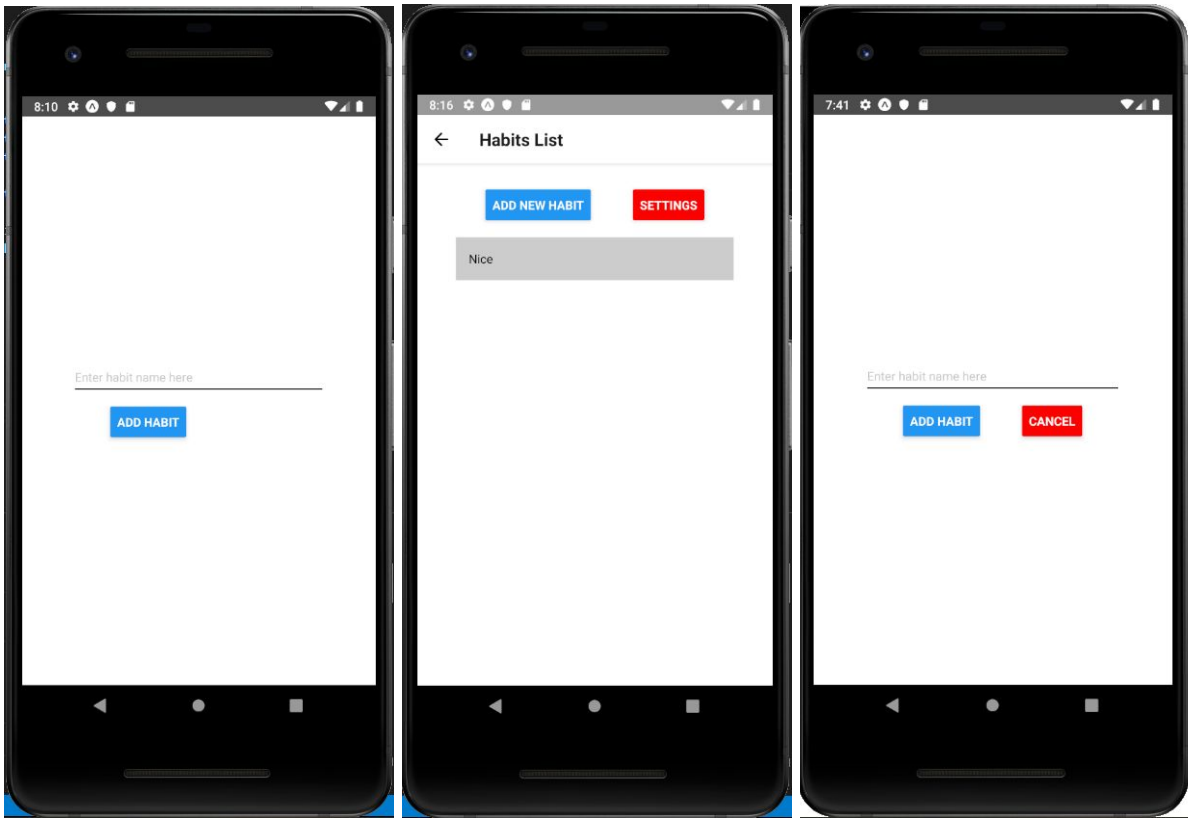
3rd UX design decision:

The third and final UX decision I am going to talk about was the back navigation methods. This UX decision influenced both the adding habit and settings page. I will demonstrate the back navigation using the add habit page.

Initially I had no way of returning to the home page upon navigation to the add habit page which was an issue as it meant the user had to input a habit and then complete it if they wanted to return to the home page without restarting the application.

My second approach was to use the inbuilt back navigation button provided by react native navigation which is shown in the top left of the second picture. This had two issues the first being it let the user return to the entry page which is not functionality I wanted in the application. The second and primary problem with this navigation was that did not appear on the add habit page was I was using modals to display pop up pages over the home page which do not by default include the react native navigation title system.

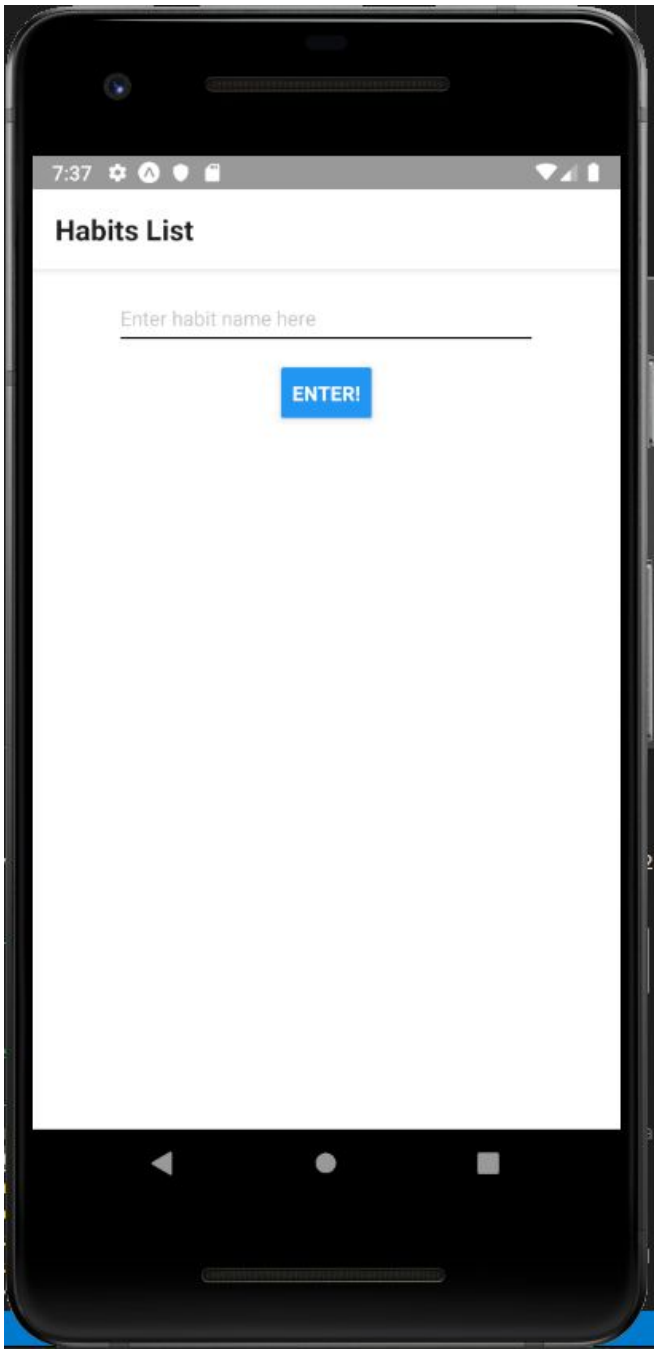
The third and final option I went with was removing the back navigation functionality and replacing it with cancel buttons on each page. These buttons were red so they would stand out and it very easy for the user to navigate the application.



Appendix:

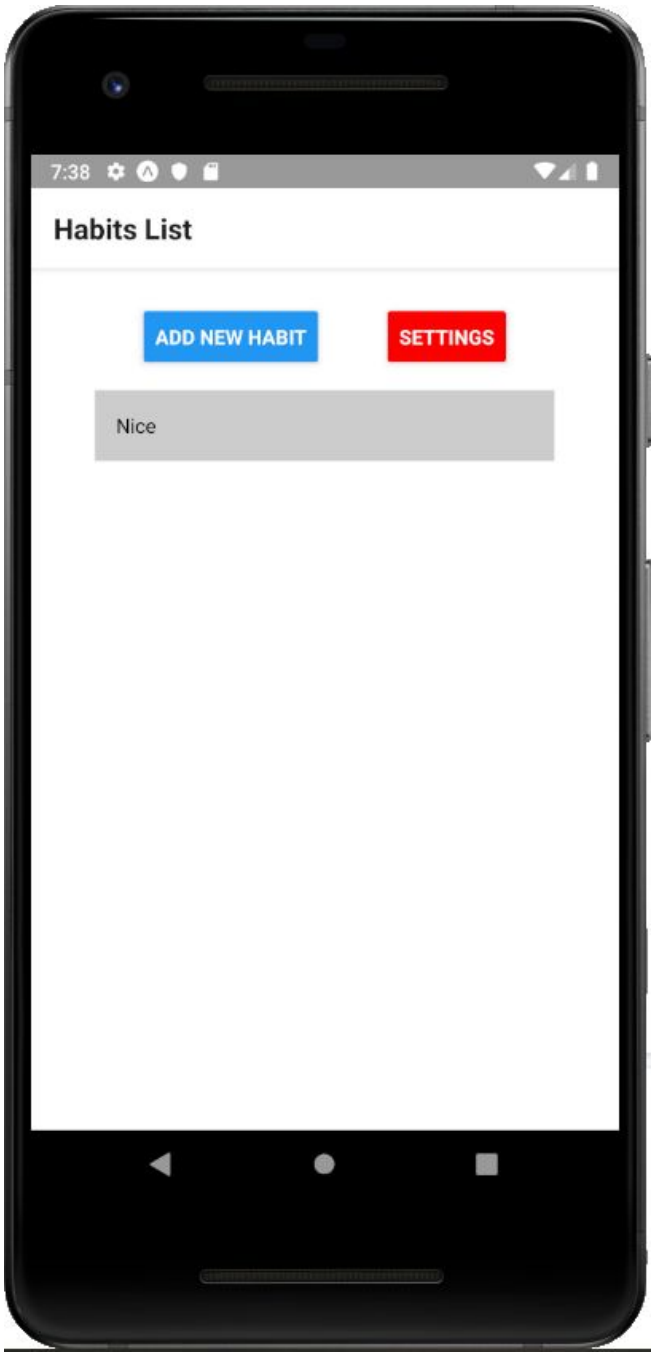
Entry Page:

This is essentially the login page for the application. It prompts the user to enter their name when opening the application. Upon using the enter button the user is navigated to the habit list home page where a pop up message appears welcoming the user by name.



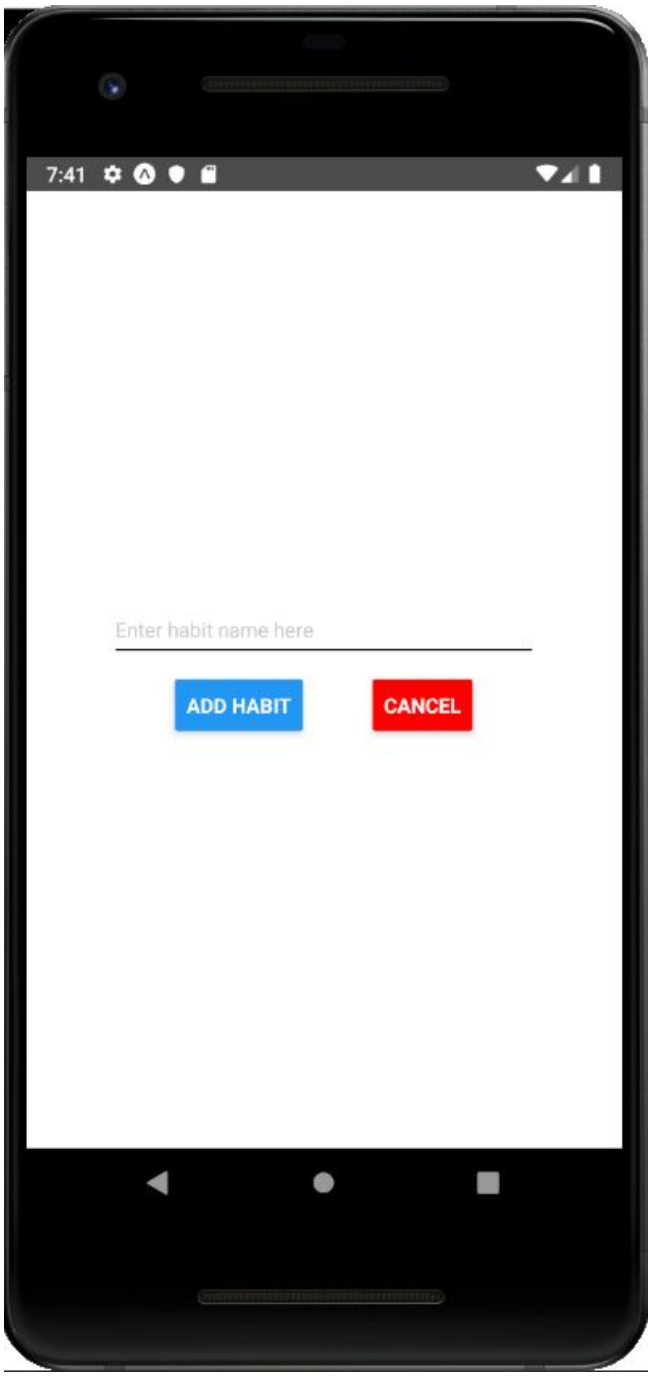
Home page:

This is the home page for the application, it includes navigation to the add new habit page and settings page. It also lists all of the habits stored by the user and lets the user touch the habits to complete them.



Add Habit Page:

This is the add habit page of the application. It prompts the user to enter the habit name upon which they can add the habit entered or cancel.



Settings page:

This is the settings page of the application. It allows the user to clear all data/habits currently stored on the application.