React Native Development

Task 1

GitHub Repository: React Native Apps Folder Github Link

1) Screenshots of Your App

I ran the React Native app on both emulator and a physical mobile device.

- On the emulator, the app runs slower, especially when starting and transitioning. I believe this is due to the complexity of Android Studio.
- On the physical mobile device, the app is more responsive and easier to handle, with faster access via QR code.

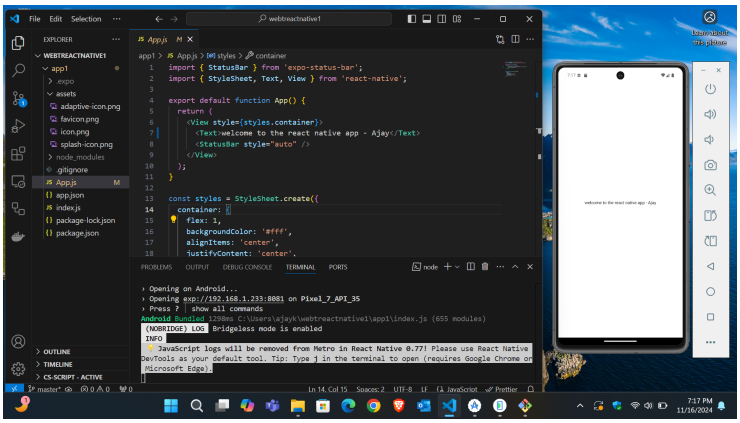

Figure 1: Screenshot of the app running on the Android emulator

2) Setting Up an Emulator

I am using Windows, so I downloaded and installed Android Studio. Then, I navigated to the AVD Manager and created a new virtual device, selecting the platform, version, tools, and orientation. Afterward, I clicked the "Start Emulator" button to begin the emulator.

Issues: When I clicked "Start Emulator," I encountered an error due to low disk space. Upon checking, I found only 2GB of free space left on my C drive. After deleting unwanted files and freeing up around 10GB, I was able to start the emulator successfully. However, the laptop's performance slowed due to the emulator running.

3) Running the App on a Physical Device Using Expo

As described in the PDF, I created an Expo project using the specified commands and started the project with `npm expo start`, which generated a QR code in my Git Bash terminal. I opened the Expo app on my mobile device and scanned the QR code to launch the app.

While using Expo, I did not encounter any issues. The problems I faced occurred while working with React Native CLI, which I will explain in the next section.

4) Comparison of Emulator vs. Physical Device

The emulator is good for testing, but the laptop performance becomes slow when using it, and the app takes time to load after each change. Every time I want to see changes, I must rebuild the app. On the mobile device, the performance is much better, and connecting is easier. It is also faster and more convenient for debugging the app.

Advantages of Emulator:

- Useful for testing on various device configurations and OS versions, which is helpful for large apps.

Disadvantages of Emulator:

- Slower performance and inability to test camera, maps, and touch responsiveness.

5) Troubleshooting a Common Error

While working with React Native CLI, I encountered several issues. When I ran `npx react-native init app`, I ended up with a blanck folder containing only the `node_modules` and `package.json` files, with no other folder structures. After some time, basic React Native CLI commands stopped working.

To resolve this, I referred to the React Native documentation for commands, which helped me recreate the basic structure. I also encountered issues with environment variables and the AVD not being detected. I resolved these by setting the path variables and restarting the AVD by installing additional files in the SDK platform tools.

Task 2

a) Mark Tasks As Complete

I have updated the task interface and implemented a done button. When a task is done, the task content will be strikethrough.

I have implemented the `toggleTaskCompletion` function which takes a `taskId` as an argument. It updates the task state using the `setTasks` function. Using the map function, it iterates through each task and toggles the completed status for the task with the matching `taskId`. If the task's `id` matches `taskId`, it will update the task's `completed` property to the opposite of its current value.

b) Persist Data Using AsyncStorage

`saveTasksToStorage` and `loadTasksFromStorage` functions handle the tasks data. They store the data into a JSON file and load the stored data when reopening the app.

With this implementation, after I close the app and the emulator, when I reload, the task data is still visible.

c) Edit Tasks

The `startEditing` function sets the `editingTask` state to the id of the task being edited and initializes `editedText` with the task's current text. The `updateTask` function updates the task text in the tasks array. It maps over each task and updates the task with the matching `editingTask` id, setting the text to `editedText`. After updating, it clears the `editingTask` and `editedText` states. This allows the text to be updated.

The edit button works such that when I long-press on tasks, the content becomes editable, and I can change the text.

d) Add Animations

I have implemented the `fadeIn` function which animates the fade-in effect by changing the `fadeAnim` value from 0 to 1 over a duration of 3500ms. I also implemented the `fadeOut` function which animates the fade-out effect by changing the `fadeAnim` value from 1 to 0 over a duration of 500ms. It then triggers the deletion of the task after the animation completes.

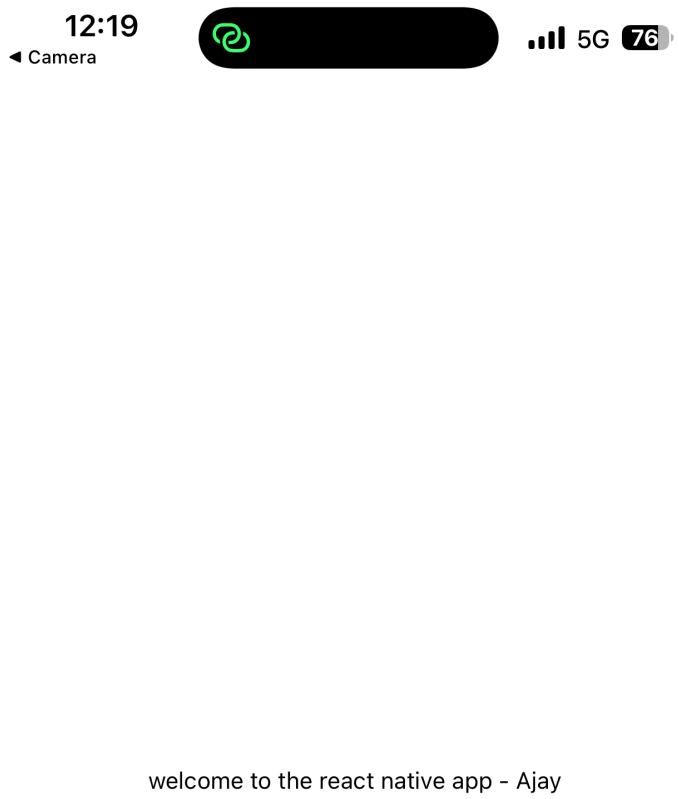

Figure 2: Screenshot of the app running on the mobile device.

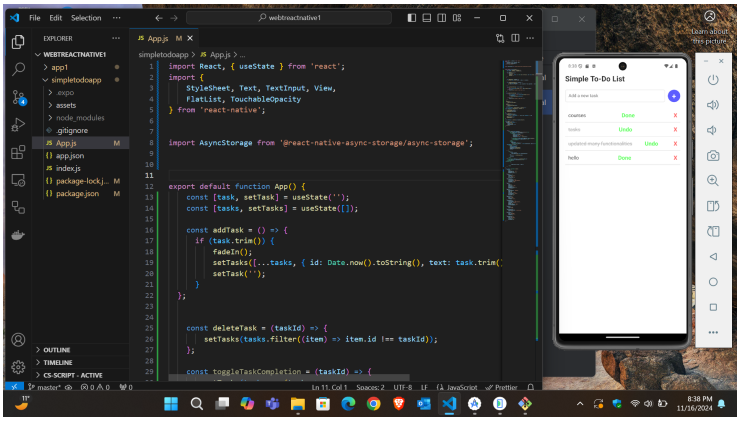

Figure 3: simple app running on the emulator with all functionalities.