Web Technologies Lab 3
React Native Application

Contents

1 Introduction

This document presents the steps and insights gained during Lab 3 of Web Technologies, focusing on running a React Native application on both emulators and physical devices. It also includes troubleshooting common errors encountered during the process.

2 Task 1: Set Up the Dev Environment

2.1 App Running on Emulator and Physical Device

Screenshot of App Running on Emulator

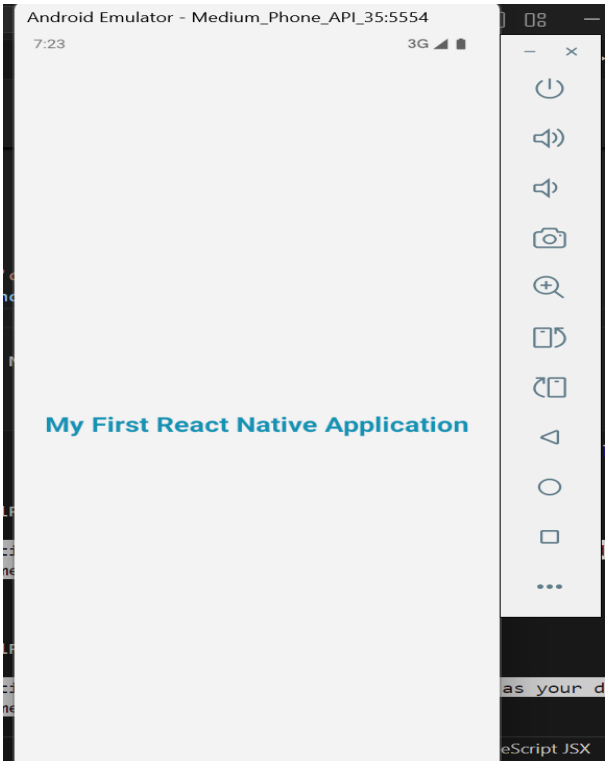

Figure 1: App running on an emulator.

Screenshot of App Running on Physical Android Device

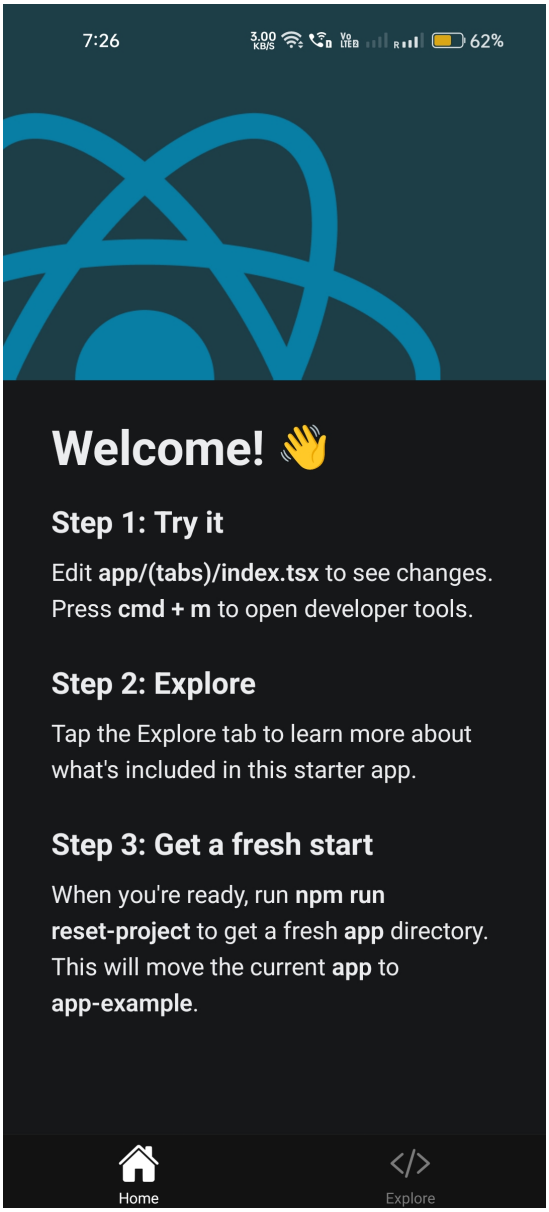

Figure 2: Device Screen 1.

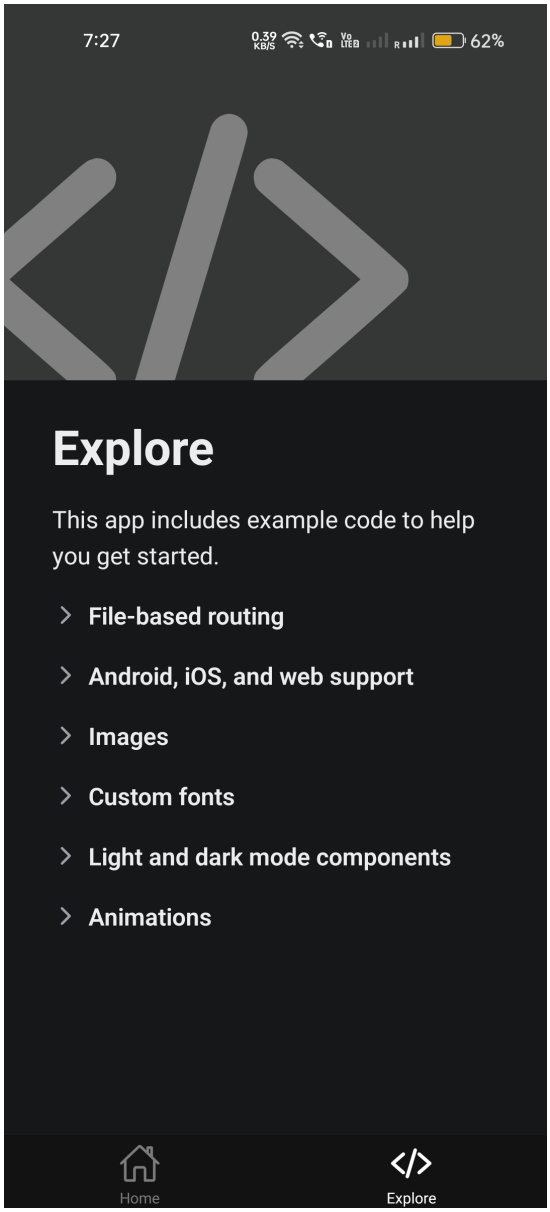

Figure 3: Device Screen 2.

Differences Between Emulator and Physical Device Testing:

There are a few key differences running an app on an emulator versus running the app on an actual, real-life physical device. Physical devices normally give much faster performances and more consistent speeds, since they are not dependent on the computer's resources, unlike emulators, which may lag when apps become complex. Hardware sensors, such as GPS, accelerometers, and cameras, also show accurate real-time data in physical devices, while on emulators such features are only simulated and often hardly accurate. Emulation, on the other hand, also uncovers the impact of an app on real battery life, memory, and storage-emulators do not really simulate such low-resource or power-drain conditions. Testing in both environments is rather mandatory for capturing a complete picture of application behaviour and user experience.

2.2 Setting Up an Emulator

Steps to Set Up an Emulator

1. Open Android Studio and click on "More Options."
2. Select "Virtual Device Manager" to view default virtual devices.
3. Click the "+" symbol to add a new device.
4. Choose a device from the list and click "Next."
5. Select the desired Android version and verify configurations.
6. Click "Finish" to add the device to the manager.
7. To run the device, click the "Run Device" icon.

Challenges Faced: I initially selected the Pixel 9 Pro device, but it did not display on the screen. After trying different devices, I found the default medium phone worked. ChatGPT helped identify that the issue was with the screen resolution or aspect ratio of the devices I tried earlier.

2.3 Running the App on a Physical Device Using Expo

Steps:

1. Installed the Expo Go app on my Android device.
2. Connected both the computer and the device to the same Wi-Fi network.
3. Started the Expo server with the command `npx expo start`.
4. Scanned the QR code displayed on the terminal with the Expo Go app. It took some time to build the app on my device.

Challenges Faced: I didn't faced challenges while running the app on a physical device.

2.4 Comparison of Emulator vs. Physical Device

1. **Performance:** Physical devices can perform much faster, smoother, and more consistently without the constraints of needing to take up computer resources. Emulators are laggy, especially for slower computers, as they use extra CPU and memory that affect performance means.
2. **User Experience:** Physical devices represent an authentic experience in terms of real-world touch sensitivity, various gestures, and screen responsiveness, while emulators use mouse input to simulate touches, which is not that accurate. On a physical device, you can test how users would interact with the app.
3. **Battery and Resources:** Physical devices allow for real testing of battery consumption, memory usage, and performance under low-resource scenarios. Emulators can't fully simulate battery drain or the app's behaviour in low-resource conditions; hence, it limits resource optimization testing.

4. **UI Rendering:** Emulators do try to emulate several screen sizes and resolutions, but physical devices will really show how the UI would look regarding the scaling of various elements and their layout on screens of different sizes. This comes to be very important for ensuring that fonts, images, and interactive elements portray exactly as they should.

5. **Convenience and Accessibility:** Emulators are more convenient and allow the testing of wide ranges of virtual devices. However, they cannot provide complete user experience; therefore, their counterpart, physical devices, becomes necessary in the last phases of testing for quality assurance.

Overall, the development of React Native is good in both aspects: emulators and physical devices. Emulators give flexibility and easy access to various device configurations, while physical devices provide accuracy regarding performance insights and real-world interaction feedback.

- **Advantages and disadvantages:**

  One of the biggest advantages of using an emulator in React Native development is not only convenient but also cost-effective. It usually comes with one's development environment and saves a developer from having to purchase physical models of different devices. They also have integrated debugging tools that make logs and outputs, along with network requests, easier to inspect, therefore speeding up the debugging. Also, emulators support fast resets and configurations. Therefore, with emulators, a developer can swiftly change between devices and configurations. However, there are limitations to using emulators: They usually experience performance and accuracy limitations because they tend to rely on the computer for their resources and therefore lag or run slower, especially on lower-spec computers; this makes the test less smooth as compared to a physical device. Emulators will only simulate touch and gesture interactions, which can miss much of the nuance of real-world user interactions, and can't reliably represent battery usage, memory drain, or low-resource scenarios to limit insight into real-world performance.

  On the other hand, physical device testing with React Native development promises to be amazingly realistic: real-world performance, accurate user interaction, precise sensor data-all on your real device. Physical devices give an actual feel where actual touch sensitivity, multi-touch, and swipe responsiveness can be checked by the developers. The actual running of the application on actual hardware provides a better understanding to the developers for battery and memory consumption, thereby optimizing resource utilization. This becomes quite essential because low-level battery or heavy requirements of memory of the application cannot be emulated precisely using these emulators. However, there are also challenges with the testing on physical devices: the cost of and access to acquiring multiple devices that represent a wide range of screen sizes, operating systems, and versions. Additionally, setting up and configuring physical devices can be pretty time-consuming because one is forced to install and connect cables manually for each test. Despite the drawbacks, a physical device offers incomparable accuracy in final-stage testing, where authentic user experience and performance insights are critical.

2.5 Troubleshooting a Common Error

Error Description: While starting the React Native app, I encountered the error:

```
Unable to load script from assets 'index.android.bundle'.
```

Resolution: The above mentioned error generally happens when the React Native app fails to locate the bundled JavaScript file needed for rendering. This error often happens if the Metro bundler isn't running, if the bundling process didn't complete successfully, or if there's a misconfiguration that prevents the app from connecting to the local development server. Sometimes, it can also occur after changes in the project structure or configuration files.
I resolved this error by:

- Ensure the Metro bundler is running.
- Clearing the cache with `npx react-native start --reset-cache`.
- Restarting the app.

3 Task 2: Building a Simple To-Do List App

3.1 Mark Tasks as Complete

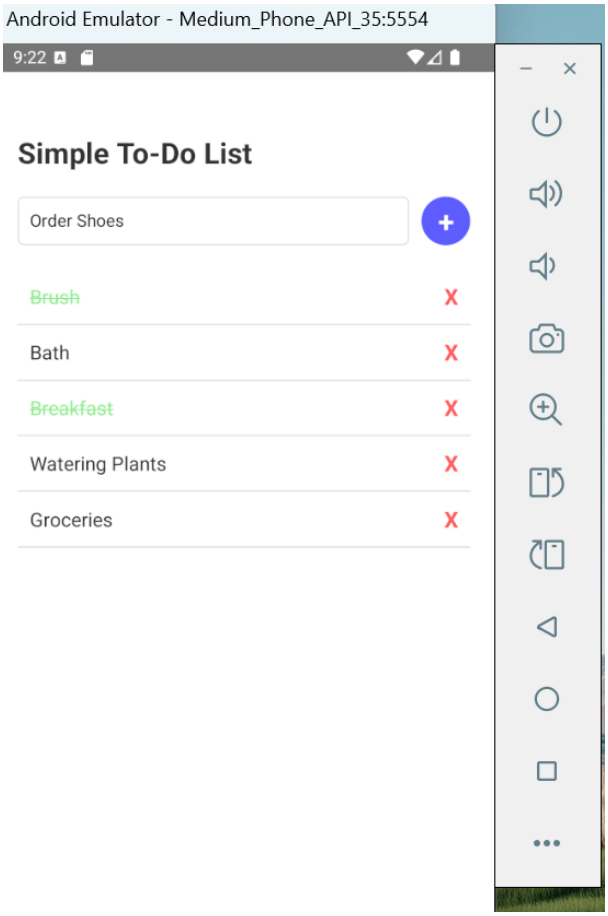

Figure 4: Marked task as completed.

I accomplished updating state with the completion status of tasks by implementing the function ‘toggleTaskCompletion‘: It maps over the ‘tasks‘ array in order to find a task by its ‘id‘ and, if it finds it, toggles its property ‘completed‘ between true and false, keeping the other tasks unchanged. After that, it calls the function ‘setTasks‘, passing the result as an argument. React then automatically re-renders the component, and I conditionally styled completed tasks with text-decoration: line-through and a color change so that the UI reflects the changed state.

3.2 Persist Data Using AsyncStorage

I used AsyncStorage to save and retrieve the list of tasks. Every time the state of tasks changes, I save the updated list in AsyncStorage using ‘AsyncStorage.setItem‘. This way, when the app starts, the saved tasks in AsyncStorage are fetched using ‘AsyncStorage.getItem‘ and set into state. This means tasks persist across sessions of running the app and are even there even after the app is closed and reopened.

3.3 Edit Tasks

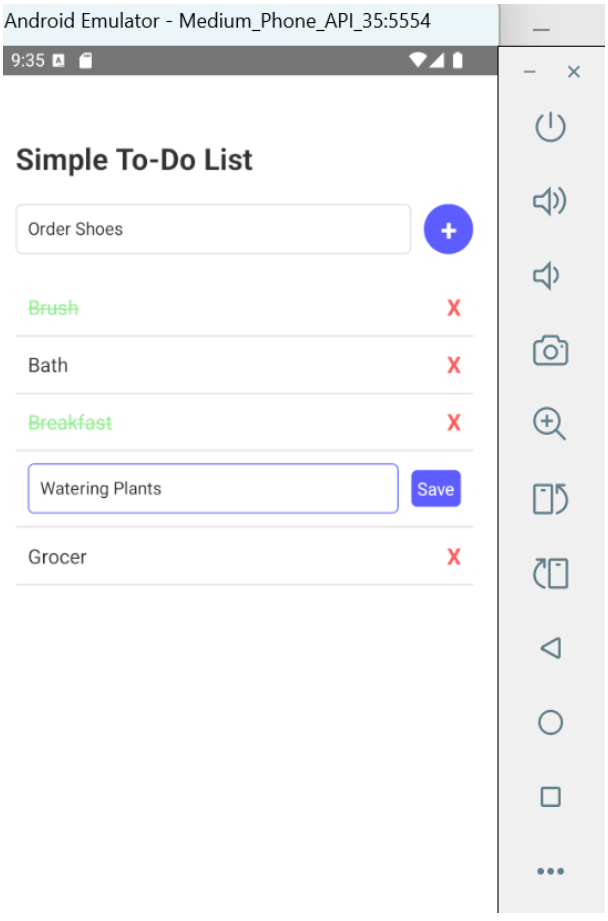

Figure 5: Edit the task.

I allowed the editing of tasks by enabling users to tap on a task to edit its content. This is by setting in state variables the ‘id‘ and the current text of the task in the ‘editingTaskId‘ and ‘editingTaskText‘, respectively, upon tapping a task. This will toggle the UI to the

editing mode. Finally, in the update function, a task was changed in the state array by mapping over the tasks in search of the one whose 'id' matched and updating the 'text'. And finally, with saved changes retained, the 'editingTaskId' gets cleared, and the UI switches to normal view. This was handled through conditional rendering, showing a 'TextInput' if a task is being edited to allow for seamless inline editing.

3.4 Add Animations

I've implemented the Animated API from React Native to be able to apply fade-in and fade-out to tasks upon their addition or deletion. While a new task is added, an animation has to be performed related to the 'opacity' property-from 0 to 1 in 500 milliseconds-every time a new task is rendered. In this case, while deleting a task, this should fade out before the actual removal of this task from the state. In that way, it creates an effect of transitions in the app, making it more interactive by providing some kind of transitions or feedback to users' actions, so that users can feel polished.

4 GitHub Link

The link to github repository: `https://github.com/saidugyala/Web_Tech_Lab3`

Use Of AI generated tools I used ChatGPT for resolving errors while setting up the environment and also used it to resolve some issues in the code.