Lab 3: Building a Mobile App

Mary Sreeja Thirumala Reddy

November 17, 2024

Task 1: Screenshots of Your App

Attach screenshots of your app running on an emulator and on a physical Android or iOS device.

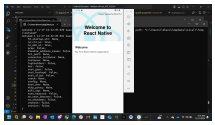

Figure 1: App running on Emulator

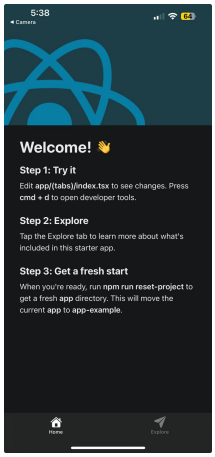

Figure 2: App running on Physical Device

Observations

1. When running the app in the emulator, it is slower than on a physical device.
2. The emulator does not have full access to the hardware properties compared to a physical device.
3. The user experience on an emulator does not feel real as it uses a touchpad, while a physical device provides real-world usage.
4. The emulator uses the host network, making real-world testing problematic, whereas a physical device connects through Wi-Fi for realistic problem-solving.

Setting Up an Emulator

Step 1: Install Android Studio

- Download and install Android Studio.

Step 2: Set Up Android SDK

1. Go to `Settings`.
2. Navigate to `Appearance & Behavior > System Settings > Android SDK`.
3. Under the `SDK Platforms` tab:
    - Select the latest Android version.
    - Check the box for Google APIs.
4. Under the `SDK Tools` tab:
    - Check `Android SDK Build-Tools` and `Android Emulator`.
    - Install them.

Step 3: Create an Emulator

1. Open Android Studio.

2. Go to `Tools > Device Manager` (or click the Device Manager icon).
3. Click `Create Device`.
4. Select a device model (e.g., Pixel 5) and click `Next`.
5. Choose a system image that matches the latest Android version.
6. Customize emulator settings if needed, then click `Finish`.

Step 4: Start the Emulator

- In the Device Manager, find the created device and click the `Play` button.

Step 5: Link Emulator to React Native

1. Start the Emulator: Ensure the emulator is running.
2. Start Metro Bundler: Run `npx react-native start`.
3. Run the app: Open a new terminal and run `npx react-native run-android`.

Step 6: Verify Emulator Setup

- The app should load on the emulator.

Challenges

- Version incompatibility and environment variables (`ANDROID_HOME`, `JAVA_HOME`).
- Dependency management issues due to frequent updates in React Native.
- Configuring the AVD and linking it to the React Native project.
- Issues with CMake and state persistence libraries.

Comparison: Emulator vs. Physical Device

Aspect	Emulator	Physical Device
Setup	Requires installing Android Studio	Requires installing Expo Go app
Performance	Slower due to virtualization	Faster due to hardware access
Device Access	Limited and has path issues	Full access and easy to connect
Debugging	Built-in tools	Requires separate setup
Screen Resolution	Multiple options available	Fixed to the device screen

Advantages of Emulators

- Cost-effective.
- Versatile testing across different models and configurations.
- Built-in debugging tools.
- Can operate without physical equipment.

Disadvantages of Emulators:

- Slower performance.
- Limited hardware access.
- Requires a development environment.

Advantages of Physical Devices:

- Real user experience.
- High performance accuracy.
- Full hardware access (e.g., sensors, camera, GPS).

Disadvantages of Physical Devices:

- Requires separate setup.
- Device ownership required.
- Battery drain during tests.

Task 2

(a) Mark Tasks as Complete (15 Points)

Feature: Add a toggle function to mark tasks as completed.

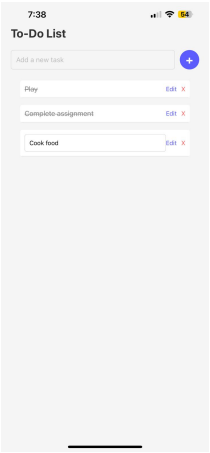

Figure 3: added tasks

- Explanation:The feature simply 'toggles' the completion state of the task on and off.On click of a task the toggleComplete function is triggered which updates the task list state by calling setTasks .nction is triggered, which updates the task list state using setTasks. In this function we check if the task people have clicked on is the same as the task's ID and then toggle the task's completion status. Based on whether or not tasks are completed or pending they are styled in different ways.The styles.completedTask adds a strikethrough effect and provides grey text for completed tasks and checks black text for pending tasks with the styles.pendingTask.his function checks if the clicked task matches the task's ID and then toggles its completion status. The tasks are styled differently based on whether they are completed or pending. The styles.completedTask applies a strikethrough effect and changes the text color to gray for completed tasks, while styles.pendingTask keeps the text black for tasks that are still pending. This visualization allows users to easily tell where they are and where they need to be, and it updates the state each time the user interacts with the task list.

(b) Persist Data Using AsyncStorage (15 Points)

- As for the storage of information within a React Native app, we can use the AsyncStorage to then save and load our tasks. First, you have

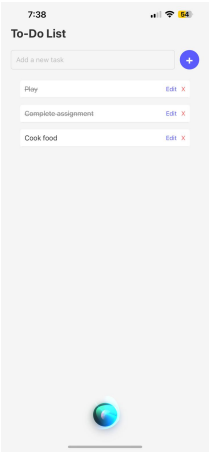

Figure 4: completed

to make make sure you have put the package '@react-native-async-storage/async-storage' install. age. Tasks have to be kept as a JSON string with 'AsyncStorage.setItem' To store, there is the 'AsyncStorage.set' and for viewing, there's 'AsyncStorage.getItem'. When the app is started load tasks with the help of 'useEffect' and put them into the state. To achieve this, use another 'useEffect' to save tasks, but the present one is reserved for passing parameters only activating the second one for saving the tasks. because tasks will be different it will, time since. This guarantees that they are briefly stored locally and remain the same after administrations make their rounds across local communities. ising the usage strong and results permanent insofar as the application is closed and reopened.

(c) Edit Tasks (10 Points)

- When there are tasks, they can be edited by 'tapping' them. The editingId identifies the type of work edited; the TextInput com r, the operations penticate allows the modification of the tasks text. Whenever a task is touched, the method stardedEditing is called, linking editingId and load- hifting the current text of the task into the task-Text. The UI conditionally shows a TextInput if the task is being edited (if the task has ID that matches the editingId). When the user is done in modifying and adding up new information on the report I

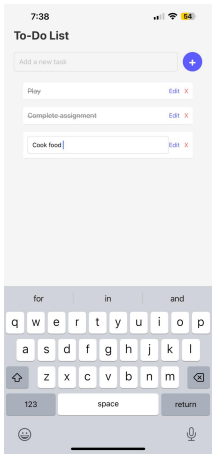

Figure 5: editable

had in mind anywhere outside the TextInput taps the saveEdit function is called to Then the changed task is set and the state editor to false for the edit to stop.

(d) Add Animations (10 Points)

- Explanation: Most of the animations used here are under React Native's Animated API, enhancing the UX of task interactions through smooth transitions. When a task is added, the new window just pops in with some kind of animation giving the impression of added task. Likewise, when a task is archived then it gradually vanishes and it has a nice way of being removed other than having it suddenly vanish.

  These animations enhance the interactivity of the tasks, making it appear that the app is more professional and smooth each time a user wraps each task in an Animated.View. The transitions do not only minimize continuity jarring effects but also offer valuable screen feedback, which improves accessibility of the gadgets. Also, they can guard that when a task is clicked, the brightness of a task is changed at once or step by step, which makes the interaction more interesting.

GitHub Repository

You can access it using the following link:
https://github.com/SreejaReddytr/SimpleToDoAppExpoSimpleToDoApp GitHub Repository

Acknowledgment of Learning Assistance

I would like to acknowledge the use of ChatGPT, an AI language model, as a valuable learning tool during the development of this lab report. ChatGPT provided insights and guidance on formatting this document in LaTeX, as well as assistance in structuring and explaining technical content effectively.and it also have helped a lot in rectifing all the errors and getting the output