Task 1 Submission

1. Screenshots of Your App

Emulator Screenshot

Take a screenshot of your app running on an emulator (e.g., Android Studio Emulator or iOS Simulator).

Physical Device Screenshot

Take a screenshot of your app running on an actual device (via Expo or USB debugging).

Differences Observed

- **Performance:** The app typically runs faster on physical devices due to direct hardware usage, while emulators may experience lag.
- **Device-Specific Features:** Physical devices allow testing real-world features like camera, GPS, or touch gestures, which might not behave the same in an emulator.
- **Screen Resolution:** Physical devices display apps at native resolution, which may differ from emulator settings.

2. Setting Up an Emulator

Steps Followed

For Android:

1. Install Android Studio from the official website.
2. Open Android Studio ¿ Tools ¿ AVD Manager ¿ Create Virtual Device.
3. Choose a hardware profile (e.g., Pixel 6) and select a system image.
4. Configure the emulator settings (RAM, storage, etc.) and launch the emulator.

For iOS:

1. Install Xcode from the App Store.
2. Open Xcode ¿ Preferences ¿ Platforms and download the necessary simulators.

3. Launch the iOS Simulator from Xcode or the terminal with:

```
xcrun simctl list
xcrun simctl boot <device-id>
```

Challenges and Solutions

- **Challenge:** Emulator stuck at boot screen.
  Solution: Increase RAM allocation in the emulator settings or use a different system image.
- **Challenge:** Slow emulator performance.
  Solution: Enable hardware acceleration (HAXM for Intel processors or Hypervisor Framework for macOS).

3. Running the App on a Physical Device Using Expo

Steps Followed

1. Install the Expo CLI:

```
npm install -g expo-cli
```

2. Create a project:

```
npx expo init YourProjectName
```

3. Start the Expo development server:

```
npx expo start
```

4. Install the Expo Go app on your device (from App Store or Google Play).
5. Scan the QR code displayed in the Expo CLI using the Expo Go app.
6. Your app runs on the device immediately after scanning.

Troubleshooting Steps

- **Issue:** QR code not working.
  Solution: Ensure both the device and computer are on the same Wi-Fi network.
- **Issue:** App crashes on the device.
  Solution: Check for incompatible dependencies and reinstall node modules.

4. Comparison of Emulator vs. Physical Device

Emulator

Advantages:

- Easy to set up and use.
- Debugging tools are integrated (e.g., logcat in Android Studio).
- No need for physical hardware.

Disadvantages:

- Slower performance, especially with animations or heavy computations.
- Limited support for device-specific features like Bluetooth or real-world GPS.

Physical Device

Advantages:

- Real-world performance and feature testing.
- Faster and smoother UI/UX experience.
- Accurate testing of hardware features (e.g., camera, accelerometer).

Disadvantages:

- Requires USB debugging or Expo setup.
- Dependency on device model for testing compatibility.

5. Troubleshooting a Common Error

Error Encountered

Issue: `EADDRINUSE: address already in use :::8081`.

Cause

The default Metro bundler port `8081` was already being used by another process.

Steps Taken to Resolve

1. Identify the process using the port:

```
lsof -i :8081
```

2. Kill the conflicting process:

```
kill -9 <PID>
```

3. Alternatively, use a different port for Metro:

```
npx react-native start --port=8088
```

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

1. App Features

- **Add New Tasks:** Users can input text into a form and add it as a task to the to-do list.
- **Update Existing Tasks:** Users can modify tasks they have already created.
- **Delete Tasks:** Users can remove tasks from the list.
- **Scrollable Task List:** The to-do list supports scrolling, allowing navigation through a large number of tasks.
- **User-Friendly Interface:** The app provides a simple and intuitive interface for managing tasks.

2. Setting Up the Project

1. Create a new React Native project:

```
npx react-native init SimpleTodoApp
cd SimpleTodoApp
```

2. Open the project in Visual Studio Code:

```
code .
```

3. App Implementation

Basic To-Do List Structure

- Replace the content of `App.js` with the provided code that handles adding, deleting, and rendering tasks.
- Use `useState` for managing task states and implement `addTask` and `deleteTask` functions.

Styling

- Add personalized styles to the bottom of `App.js` using the `StyleSheet` module.
- Ensure a user-friendly design for task input, display, and interactions.

4. Running the App

1. Run the app on your platform:

   ```
   npx react-native run-android
   ```

   or

   ```
   npx react-native run-ios
   ```

2. Verify that the app compiles and runs correctly on the selected platform.

5. Extending Functionality

(a) Mark Tasks as Complete (15 Points)

- Implement a toggle function to mark tasks as completed.
- Style completed tasks differently, such as using strikethrough text or changing text color.
- Update the task state to include a `completed` property.

(b) Persist Data Using AsyncStorage (15 Points)

- Use `AsyncStorage` to store the task list persistently.
- Implement functions to save tasks to storage and retrieve them when the app launches.

(c) Edit Tasks (10 Points)

- Allow users to tap on a task to edit its content.
- Create an `editTask` function to update the state and reflect changes in the UI.
- Provide a seamless user interface for editing tasks.

(d) Add Animations (10 Points)

- Use the `Animated` API from React Native to add animations when tasks are added or removed.
- Describe how the animations enhance the user experience by making the app more interactive and visually appealing.

GitHub Repository

- https://github.com/SaiVivekKancharla/MobileAppLab5