Lab 5: React Native TODO List

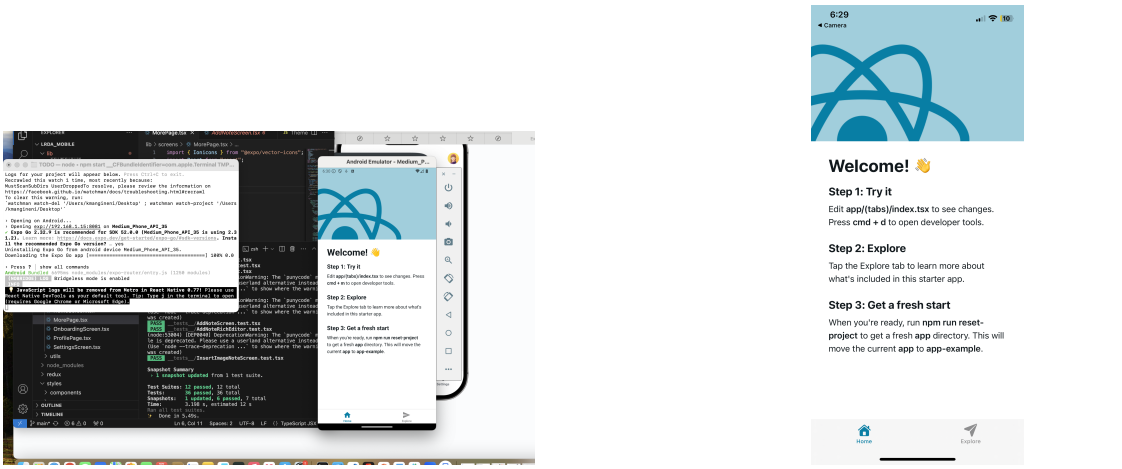
Karthik Mangineni

November 17, 2024

Task 1: React Native TODO List

(1) Screenshots of Your App

- **Screenshots:**



(a) App running on an emulator.

(b) App running on a physical device.

Figure 1: Screenshots of the app running on an emulator and a physical device.

- **Observed Differences:**
  - The app ran slower on the emulator, and touch gestures were simulated using the mouse, which felt unnatural.
  - The physical device provided a more fluid experience with real touch interactions.
  - Colors and brightness appeared more accurate on the physical device compared to the emulator.

(2) Setting Up an Emulator

- **Steps to Set Up an Emulator:**
  1. Opened Android Studio and selected **Virtual Device Manager**.
  2. Created a new virtual device using **API Level 35**.
  3. Adjusted settings such as RAM allocation and screen orientation.
  4. Started the emulator by clicking the **Play** button.
- **Challenges Faced:**
  - Downloading the system image for API Level 35 took longer due to a slow internet connection.

- The emulator initially failed to start due to insufficient RAM allocation. This was resolved by increasing RAM in the AVD Manager settings.

(3) Running the App on a Physical Device Using Expo

- **Steps to Run on a Physical Device:**
  1. Installed the **Expo Go** app on the phone.
  2. Ensured both the phone and computer were connected to the same Wi-Fi network.
  3. Ran `npx expo start` on the terminal to start the development server and scanned the displayed QR code using the Expo Go app.
- **Troubleshooting Steps:**
  - Encountered an issue where the Expo Go app couldn't detect the app because the devices were on different networks.
  - Resolved the issue by reconnecting both devices to the same network and restarting the Expo server.

(4) Comparison of Emulator vs. Physical Device

- **Advantages of an Emulator:**
  - Quick setup and accessible without physical hardware.
  - Suitable for testing basic app functionalities.
- **Disadvantages of an Emulator:**
  - Slower performance compared to physical devices.
  - Limited ability to test hardware-dependent features like touch gestures and sensors.
- **Advantages of a Physical Device:**
  - Provides an authentic user experience with accurate touch interactions.
  - Enables testing of hardware features like camera and sensors.
- **Disadvantages of a Physical Device:**
  - Requires physical hardware, which may not always be accessible.
  - Setup can be time-consuming, especially for debugging permissions and network connectivity.

(5) Troubleshooting a Common Error

- **Error Encountered:** The Expo Go app on the physical device failed to detect the app.
- **Cause of the Error:** The computer and phone were not on the same Wi-Fi network.
- **Resolution:**
  1. Reconnected both the computer and phone to the same Wi-Fi network.
  2. Restarted the Expo server using `npx expo start`.
  3. Scanned the QR code again using the Expo Go app, which resolved the issue.

2.7 Extending Functionality

(a) Mark Tasks as Complete

- **Implementation:**
  - Added a `markAsCompleted` function that toggles the `status` property of a task to `true`.
  - Styled completed tasks using a strikethrough effect with the following code:

    ```
    textDecorationLine: item.status ? 'line-through' : 'none'
    ```

- **Screenshot:**

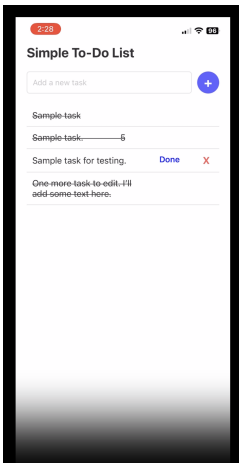


Figure 2: Task marked as completed with strikethrough styling.

(b) Persist Data Using AsyncStorage

- **Implementation:**
  - Used `AsyncStorage.setItem` to store tasks and `AsyncStorage.getItem` to retrieve them when the app is loaded.

(c) Edit Tasks

- **Implementation:**
  - Used an `isEditClicked` state to track when a user is editing a task.
  - Modified the task in the state array upon clicking the `Done` button.
  - The editTaskFun function is responsible for initiating the editing process of a task. It searches the tasks array for a task with the specified taskId using the find() method. If the task is found, it updates the editTask state with the selected task, sets the task state to the current text of the task, and toggles the isEditClicked state to true to enable the editing UI. If the task does not exist, it triggers an alert notifying the user. On the other hand, the editDone function is used to save the changes made to the task. It checks if an editTask exists and ensures that the task text is not empty after trimming. The function then updates the tasks array by replacing the text of the edited task using the map() method, clears the input field, and resets isEditClicked to false to close the editing UI. If no editTask is found, it displays an alert to inform the user that the task does not exist. Together, these functions manage the editing flow of tasks, ensuring a seamless user experience.

- **Screenshot:**

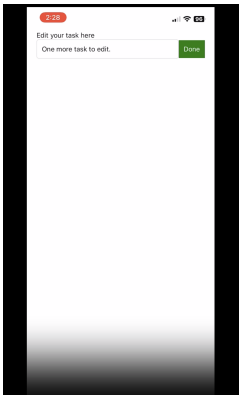


Figure 3: Editing a task in the TODO list app.

(d) Add Animations

- **Implementation:**
  - Added a fade-in effect using the Animated API for new tasks.
  - The animation transitioned the opacity of a task from 0 to 1 over 500ms.
- **Screenshot:**

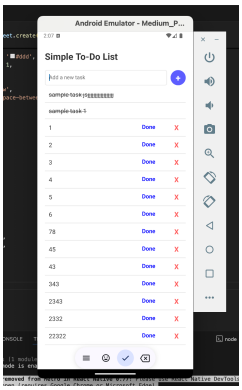


Figure 4: Fade-in animation for task addition.

Acknowledgment

I utilized ChatGPT to learn about and implement AsyncStorage in my project. This guidance helped me understand how to persist data in a React Native application effectively.

GitHub Repository

The complete project code, including the LaTeX file and the PDF report, can be accessed via the public GitHub repository:
https://github.com/rcAsironman/sample-To-Do-app.git.