React Native Development Environment Setup and Todo List Application

Gowtham Sai Mukthineni

November 18, 2024

GitHub Repository: https://github.com/gowthamsai-mukthineni/Lab3

1 Introduction to React Native

React Native is an open-source framework developed by Facebook that allows developers to create mobile applications using JavaScript and React. Its primary advantage is the ability to develop apps for both iOS and Android platforms from a single codebase, significantly reducing development time and effort. This means you can write your application once and deploy it on both platforms, taking advantage of native performance and user experience.

1.1 Key Features of React Native

- **Cross-Platform Compatibility:** Build applications that run seamlessly on both iOS and Android.
- **Hot Reloading:** Instantly see the results of changes made to your code, speeding up the development process.
- **Rich Ecosystem:** Access a wide range of libraries and components, making it easier to add complex functionalities.

Task 1: Set Up the Development Environment (50 Points)

In this task, I set up the development environment to build React Native applications.

1.2 Step 1: Install Node.js and Watchman

To complete this step:

1. Installed Node.js from the official website, which also included npm.
2. Installed Watchman (optional) using Homebrew on macOS:

```
brew install watchman
```

1.3 Step 2: Install React Native CLI

Installed the React Native CLI using:

```
npm install -g react-native-cli
```

Alternatively, used npx:

```
npx react-native init YourProjectName
```

1.4 Step 3: Set Up Android Studio (or Xcode for iOS)

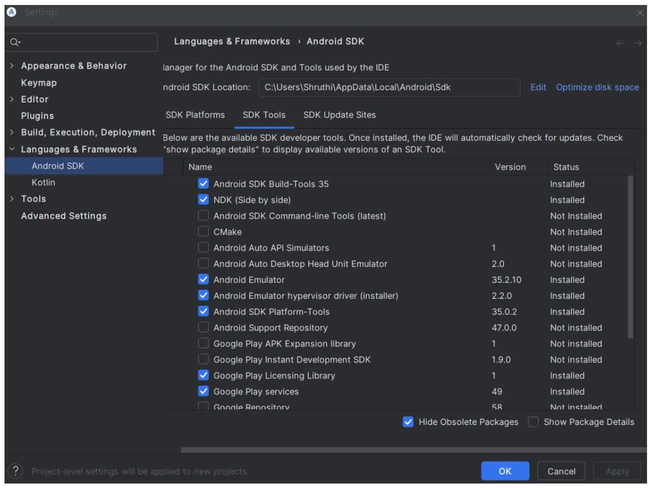
For Android:

1. Installed Android Studio and enabled SDK tools including Android SDK Build-Tools, Platform-Tools, Emulator, and Google Play Services.
2. The setup was verified as shown in Figure 1.

For iOS:

- Installed Xcode and command line tools using:

```
xcode-select --install
```



Languages & Frameworks › Android SDK

lanager for the Android SDK and Tools used by the IDE

ndroid SDK Location: C:\Users\Shruthi\AppData\Local\Android\Sdk Edit Optimize disk space

SDK Platforms | SDK Tools | SDK Update Sites

Below are the available SDK developer tools. Once installed, the IDE will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
✓ Android SDK Build-Tools 35		Installed
✓ NDK (Side by side)		Installed
☐ Android SDK Command-line Tools (latest)		Not Installed
☐ CMake		Not Installed
☐ Android Auto API Simulators	1	Not installed
☐ Android Auto Desktop Head Unit Emulator	2.0	Not installed
✓ Android Emulator	35.2.10	Installed
✓ Android Emulator hypervisor driver (installer)	2.2.0	Installed
✓ Android SDK Platform-Tools	35.0.2	Installed
☐ Android Support Repository	47.0.0	Not installed
☐ Google Play APK Expansion library	1	Not installed
☐ Google Play Instant Development SDK	1.9.0	Not installed
✓ Google Play Licensing Library	1	Installed
✓ Google Play services	49	Installed
☐ Google Repository	58	Not installed

✓ Hide Obsolete Packages ☐ Show Package Details

Project-level settings will be applied to new projects

Figure 1: Android SDK Setup in Android Studio

1.5 Step 4: Create a New React Native Project

Initialized a new project using:

```
npx react-native init YourProjectName
cd YourProjectName
```

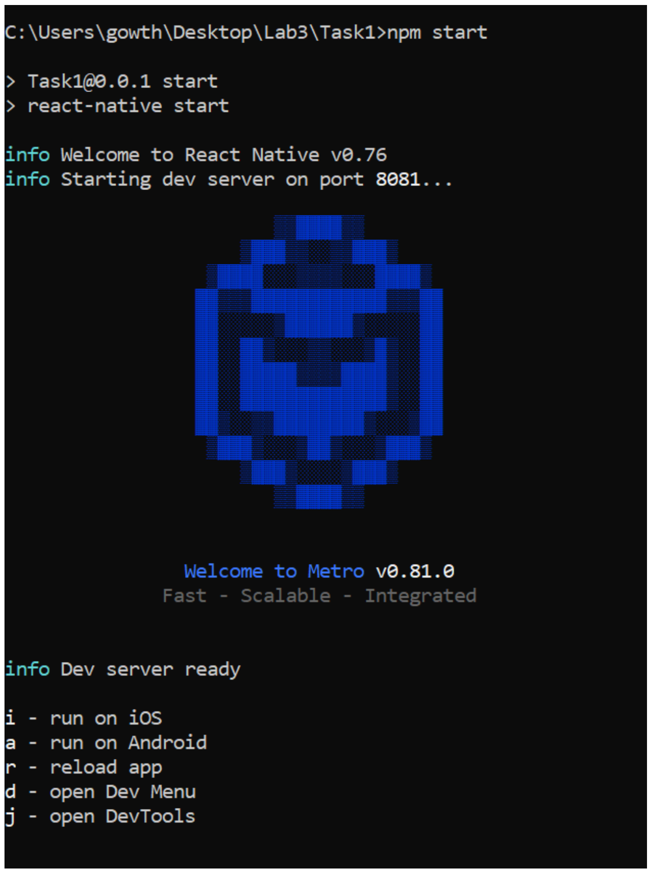
1.6 Step 5: Open the Project in Visual Studio Code

Opened the folder in VS Code and installed the React Native Tools extension.

1.7 Step 6: Start the Metro Bundler

Started the Metro Bundler using:

```
npx react-native start
```



```
C:\Users\gowth\Desktop\Lab3\Task1>npm start

> Task1@0.0.1 start
> react-native start

info Welcome to React Native v0.76
info Starting dev server on port 8081...

                Welcome to Metro v0.81.0
              Fast - Scalable - Integrated


info Dev server ready

i - run on iOS
a - run on Android
r - reload app
d - open Dev Menu
j - open DevTools
```

Figure 2: Metro Bundler Started in Terminal

1.8 Step 7: Run the App on Emulator or Device

For Android:

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

For iOS:

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

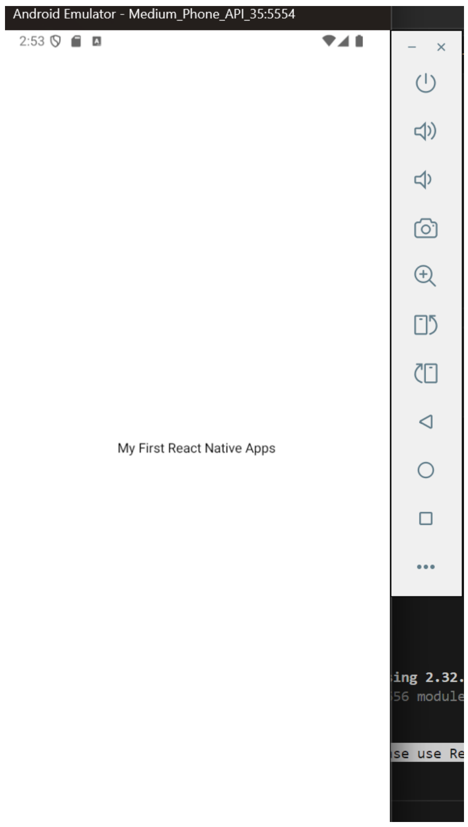

Android Emulator - Medium_Phone_API_35:5554
2:53
My First React Native Apps
ing 2.32.
56 module
se use Re

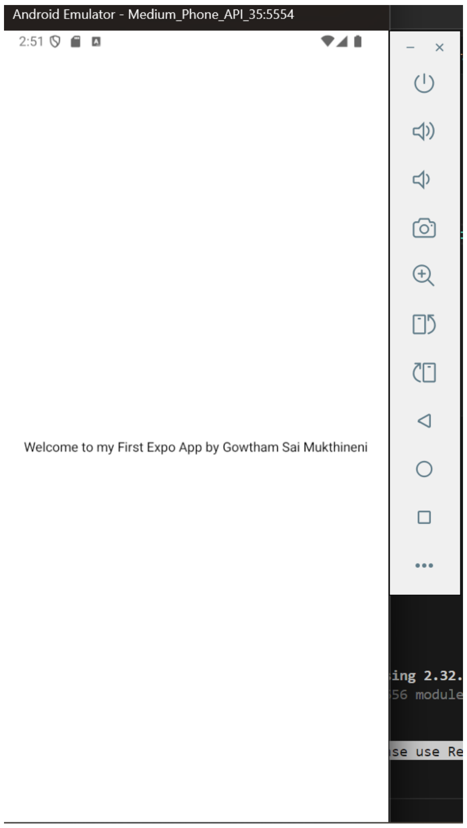
Figure 3: App running on Android Emulator

1.9 Step 8: Run the App Using Expo

Installed and created a new Expo project:

```
npm install -g expo-cli
npx expo init YourProjectName
npx expo start
```

Connected a physical device using the Expo Go app.



Welcome to my First Expo App by Gowtham Sai Mukthineni

Figure 4: App running on Physical Device using Expo

1.10 Submission Requirements for Task 1

- **Screenshots**:
  - Figure 3 shows the app running on the Android emulator.
  - Figure 4 shows the app running on a physical device using Expo.
  - Figure 2 shows the Metro Bundler running in the terminal.
- **Setting Up an Emulator**: Steps to set up the emulator are explained in Section 1. Challenges faced included issues with Emulator failed to launch due to insufficient system memory which was resolved adjusting emulator settings to reduce RAM usage.
- **Running on a Physical Device Using Expo**: The process for running the app on a physical device is explained in Step 8, Challenges faced included issues with Device and development system weren't on the same Wi-Fi network which was resolved connecting both to the same network.

- **Comparison of Emulator vs. Physical Device**:
  - Advantages of Emulator: Easy to test various screen sizes and Android/iOS versions.
  - Disadvantages of Emulator: Slower performance compared to physical devices.
  - Advantages of Physical Device: Provides a realistic testing environment with actual touch inputs.
  - Disadvantages of Physical Device: Limited to the specific hardware model and version.
- **Troubleshooting Common Errors**:
  - Encountered an issue with the default `App.tsx` file. Resolved by changing the extension to `App.js`.
  - JAVA_HOME path was not being verified. Used an LLM to obtain the correct command for Visual Studio to fix the path.

2 Task 2: Building a Simple To-Do List App (60 Points)

In this task, I built a simple To-Do List application using React Native.

2.1 App Features

- **Add New Tasks:** Users can input text into a form and add it as a task to the to-do list.

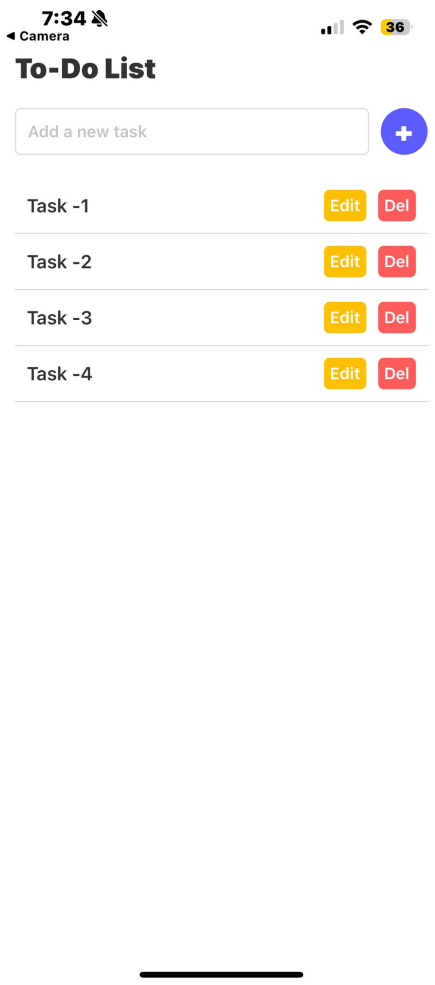


Figure 5: Adding Task

- **Update Existing Tasks:** Users can modify tasks they have already created.

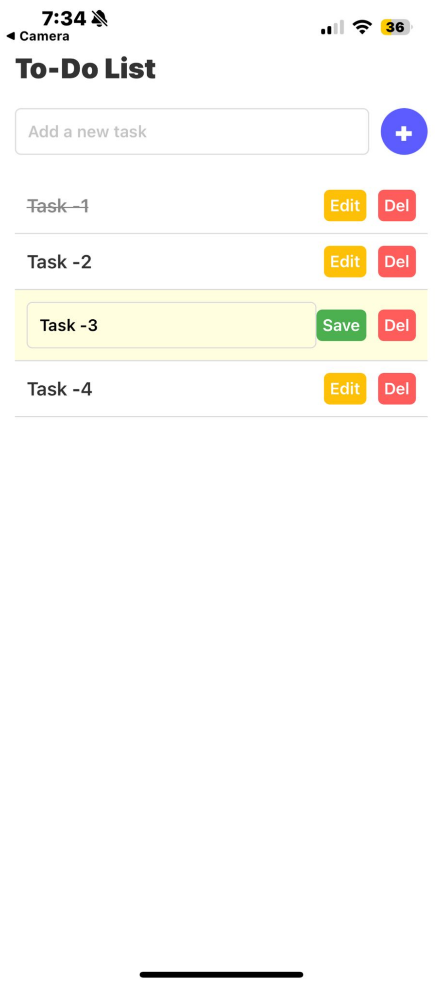


Figure 6: Editing Task

- **Delete Tasks:** Users can remove tasks from the list.

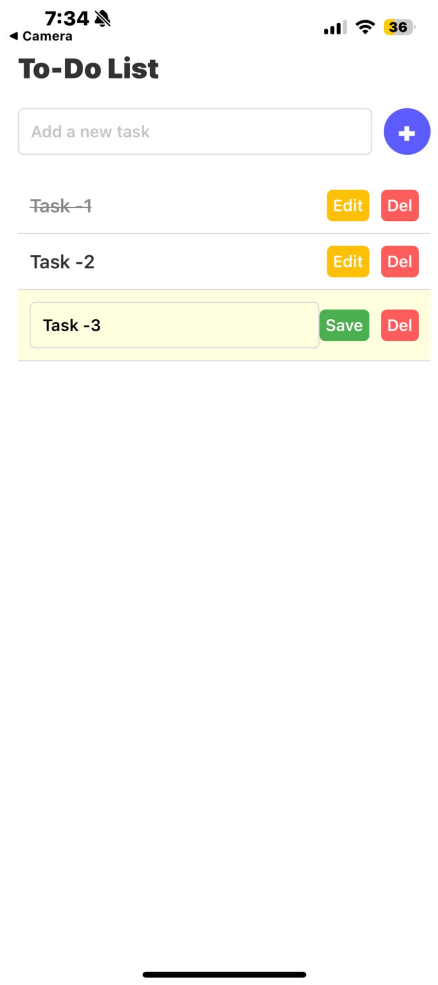


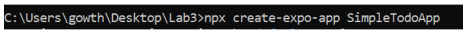
Figure 7: Deleting Task

- **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.

2.2 Step 1: Set Up the Project

1. Create and navigate to the new project:

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



```
C:\Users\gowth\Desktop\Lab3>npx create-expo-app SimpleTodoApp
```

Figure 8: Setting up the To-Do List Project

2. Open the project in Visual Studio Code:

```
code .
```

2.3 Step 2: Create the Basic To-Do List Structure

Replace the content of `App.js` with the following code:

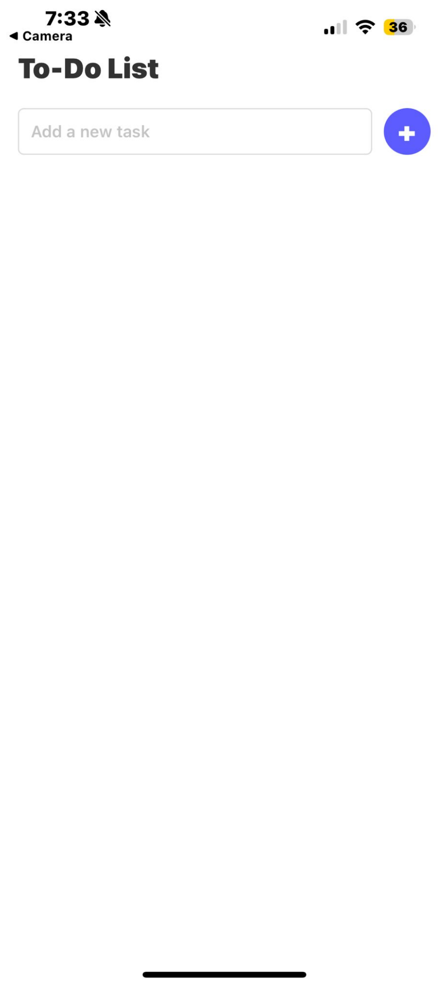


Figure 9: Initial View of To-Do List

2.4 Explanation of the Code

- **State Management**
  - The `useState` hook is used to manage the state of the input field (task) and the list of tasks (tasks).
  - When a new task is added, it updates the tasks array, and the input field is cleared.
- **Adding a Task**
  - The `addTask` function checks if the input is not empty.

    - It adds a new task with a unique ID (using the current timestamp) to the tasks array.
    - The input field is then reset to an empty string.

- **Deleting a Task**
    - The `deleteTask` function filters out the task with the specified ID from the tasks array.
    - This updates the state and re-renders the list without the deleted task.

- **Rendering the List**
    - The `FlatList` component efficiently renders the list of tasks.
    - Each item in the list displays the task text and a delete button.

2.5 Step 4: Running the App

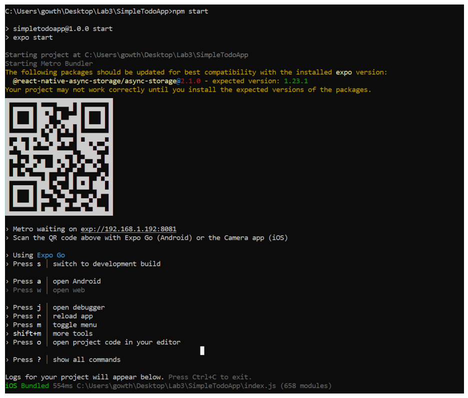
1. In your terminal, run:

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

    or

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

2. This compiles and runs your app on the selected platform.



```
C:\Users\gowth\Desktop\Lab3\SimpleTodoApp>npm start

> simpletodoapp@1.0.0 start
> expo start

Starting project at C:\Users\gowth\Desktop\Lab3\SimpleTodoApp
Starting Metro Bundler
The following packages should be updated for best compatibility with the installed expo version:
  @react-native-async-storage/async-storage@2.1.0 - expected version: 1.23.1
Your project may not work correctly until you install the expected versions of the packages.

› Metro waiting on exp://192.168.1.192:8081
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Using Expo Go
› Press s │ switch to development build

› Press a │ open Android
› Press w │ open web

› Press j │ open debugger
› Press r │ reload app
› Press m │ toggle menu
› shift+m │ more tools
› Press o │ open project code in your editor

› Press ? │ show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
iOS Bundled 554ms C:\Users\gowth\Desktop\Lab3\SimpleTodoApp\index.js (658 modules)
```

Figure 10: Running the To-Do List

2.6 Submission (Total 60 Points)

Provide detailed answers to the following questions, including any necessary screenshots:

Extending Functionality (60 Points)

- **Mark Tasks as Complete (15 Points)**
  - Add a toggle function that allows users to mark tasks as completed.
  - Style completed tasks differently, such as displaying strikethrough text or changing the text color.

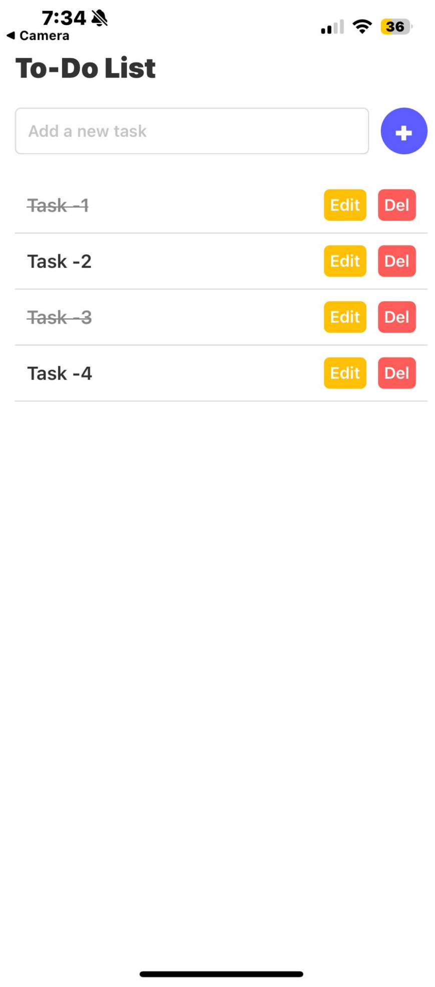
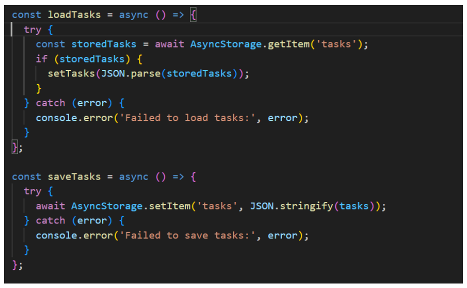


Figure 11: Task Marked as Complete

- Explain how you updated the state to reflect the completion status of tasks.
- To update the state and reflect the completion status of tasks, the toggleCompleteTask function was implemented. This function takes a task's ID as input and maps through the tasks array, checking for the task with the matching ID. When found, the completed property of the task is toggled between true and false, indicating its completion status. The updated array is then saved back into the tasks state using the setTasks function, ensuring that the component re-renders with the updated status. This dynamic update ensures the task list visually reflects completion with styling changes such as strikethrough text.

- **Persist Data Using AsyncStorage (15 Points)**
  - Implement data persistence so that tasks are saved even after the app is closed.
  - Use AsyncStorage to store and retrieve the tasks list.



```
const loadTasks = async () => {
  try {
    const storedTasks = await AsyncStorage.getItem('tasks');
    if (storedTasks) {
      setTasks(JSON.parse(storedTasks));
    }
  } catch (error) {
    console.error('Failed to load tasks:', error);
  }
};

const saveTasks = async () => {
  try {
    await AsyncStorage.setItem('tasks', JSON.stringify(tasks));
  } catch (error) {
    console.error('Failed to save tasks:', error);
  }
};
```

Figure 12: AsyncStorage Code Snippet

- **Edit Tasks (10 Points)**
  - Allow users to tap on a task to edit its content.
  - Implement an update function that modifies the task in the state array.

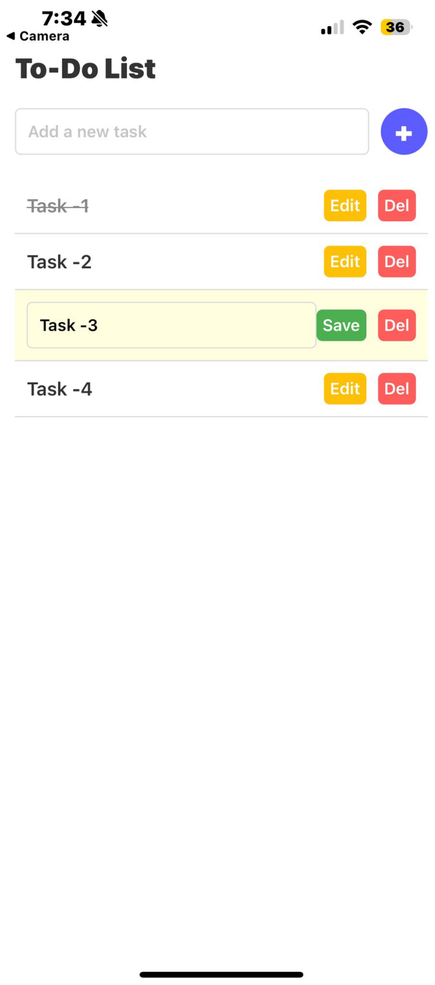
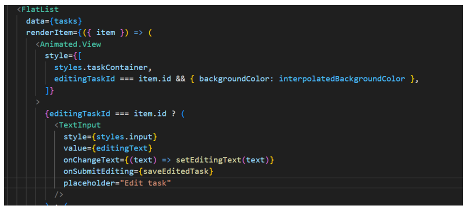


Figure 13: Editing Task in the To-Do List

– Explain how you managed the UI for editing tasks.

– The UI for editing tasks was managed by toggling between a TextInput component and a Text component based on the task's editing state. When the "Edit" button is pressed, the editingTaskId is set to the task's ID, and the corresponding task's current text is loaded into the editingText state. If the editingTaskId matches the task ID, a TextInput is displayed, allowing the user to edit the text. The "Save" button replaces the "Edit" button during this mode, and pressing it triggers the saveEditedTask function to update the task in the state and exit editing mode. This approach ensures a seamless transition between viewing and editing, maintaining a user-friendly interface.

- **Add Animations (10 Points)**
  - Use the `Animated` API from React Native to add visual effects when adding or deleting tasks.
  - Describe the animations you implemented and how they enhance user experience.
  - The implemented animations enhance the user experience by providing smooth visual feedback for adding and deleting tasks. When a task is added, it scales up and fades in, creating a polished and fluid appearance. Deleted tasks fade out and shrink before being removed, ensuring a natural and clear transition. These animations make the app feel responsive, professional, and engaging, avoiding abrupt changes in the UI. Overall, they improve clarity and interaction, making the app more user-friendly and aesthetically appealing.



```
<FlatList
  data={tasks}
  renderItem={({ item }) => (
    <Animated.View
      style={[
        styles.taskContainer,
        editingTaskId === item.id && { backgroundColor: interpolatedBackgroundColor },
      ]}
    >
      {editingTaskId === item.id ? (
        <TextInput
          style={styles.input}
          value={editingText}
          onChangeText={(text) => setEditingText(text)}
          onSubmitEditing={saveEditedTask}
          placeholder="Edit task"
        />
```

Figure 14: Code Snippet for Adding Animations Using Animated API

3 Conclusion

This report provides an overview of setting up a development environment for React Native, configuring an emulator, comparing development options, and building a simple To-Do List app with extended functionality. The setup allows for efficient app development, testing, and deployment on both emulators and physical devices.

4 Acknowledgment of LLM Assistance

During the course of setting up the development environment and fixing certain errors, I utilized a Language Learning Model (LLM) for assistance. Specifically, the LLM provided solutions for resolving JAVA_HOME path verification issues and offered advice on handling the default `App.tsx` compilation problem.