

React Native Development Environment Setup and Todo List Application

Jeevan Tubinakere Govindaraju

November 18, 2024

GitHub Repository: <https://github.com/Jeevan-369/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
```

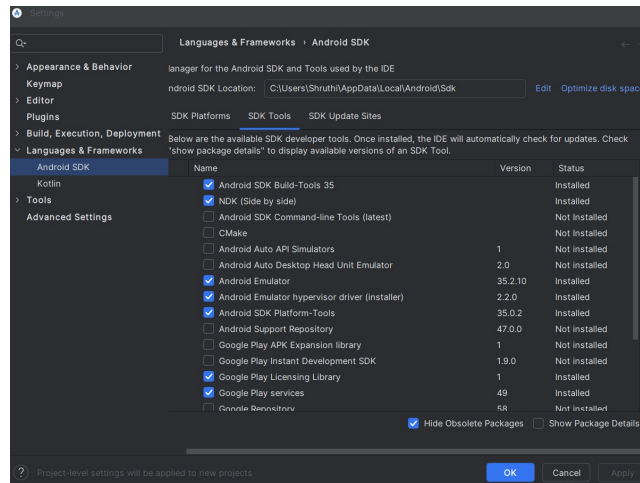


Figure 1: Android SDK Setup in Android Studio

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
```

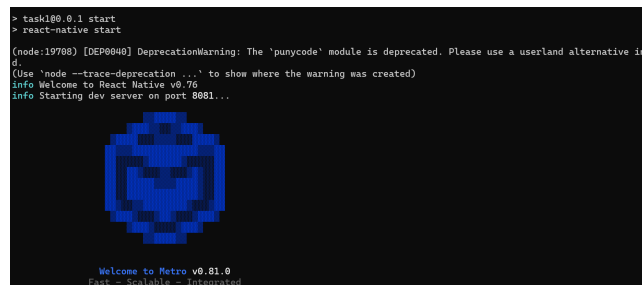


Figure 2: Metro Bundler Started in Terminal

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
```

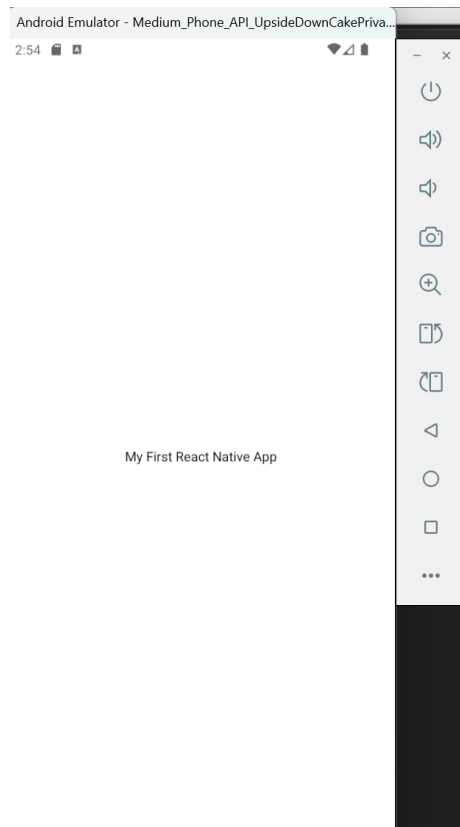


Figure 3: App running on Android Emulator

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.

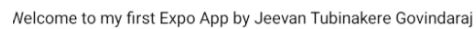
A screenshot of an Android device's status bar. On the left, the time is 2:42. To its right are two small square icons. On the right side of the bar are icons for Wi-Fi, cellular signal, and battery level.A screenshot of an Android application interface. It displays a single line of text: "Welcome to my first Expo App by Jeevan Tubinakere Govindaraj".

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 that sometimes, the emulator

might fail to start due to insufficient hardware acceleration. Ensure that virtualization support is enabled in the BIOS.

- **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 Ensuring your phone and development machine are on the same Wi-Fi network. If you encounter issues, restarting the Expo server (`expo start -c`) or restarting your device can resolve the problem..
- **Comparison of Emulator vs. Physical Device:**
 - Advantages of Emulator: Easy to set up, does not require physical hardware, good for initial testing without requiring a physical device.
 - Disadvantages of Emulator: Slower performance, less accurate touch input, may not simulate hardware features accurately
 - Advantages of Physical Device: Faster performance, real-world testing of touch gestures, hardware features (camera, GPS, sensors) are fully functional, better representation of the final user experience.
 - Disadvantages of Physical Device: Requires a physical device, sometimes issues with Expo or connection setup, not as easy to test on multiple devices.
- **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.

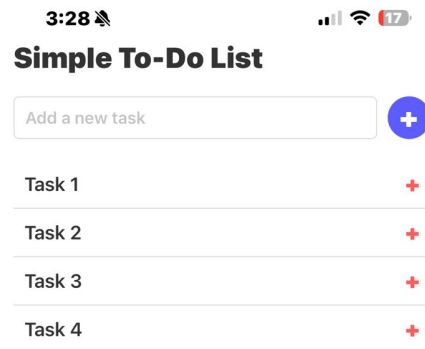


Figure 5: Adding Task

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

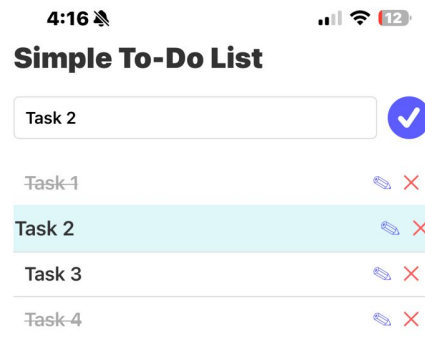


Figure 6: Editing Task

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

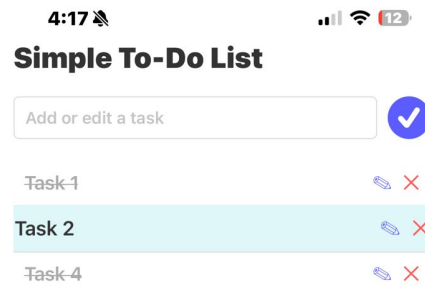


Figure 7: Deleting Task

- **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\bharg\Lab3>npm 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:

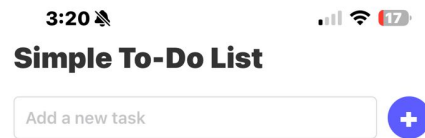


Figure 9: Initial View of To-Do List

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.

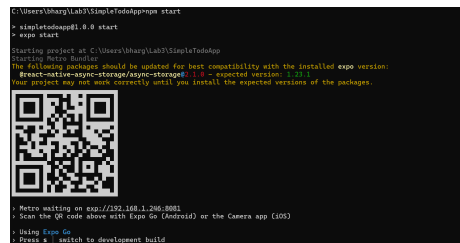


Figure 10: Running the To-Do List

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.

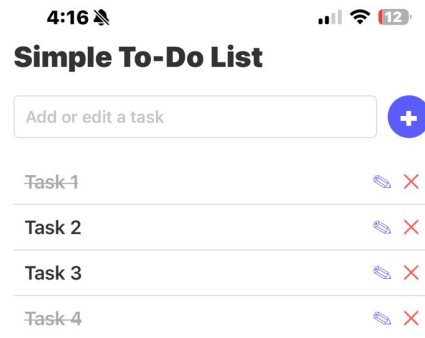


Figure 11: Task Marked as Complete

- Explain how you updated the state to reflect the completion status of tasks.
- To update the state and reflect the completion status of tasks, a new property 'isCompleted' was added to each task object in the 'tasks' array, which tracks whether a task is marked as completed.

The 'toggleTaskCompletion' function was implemented to toggle this property by creating a new array where only the targeted task's 'isCompleted' value is updated. This new array is then set to the 'tasks' state using 'setTasks', and the updated list is persisted to AsyncStorage for data retention. To visually differentiate completed tasks, a conditional style was applied, adding a strikethrough and changing the text color to gray if 'isCompleted' is true. React's state management automatically re-renders the 'FlatList' with these changes, providing immediate feedback to the user.

- **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.

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

  loadTasks();
}, []);

const saveTasks = async (newTasks) => {
  try {
    await AsyncStorage.setItem('tasks', JSON.stringify(newTasks));
  } catch (error) {
    console.error('Error saving 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.

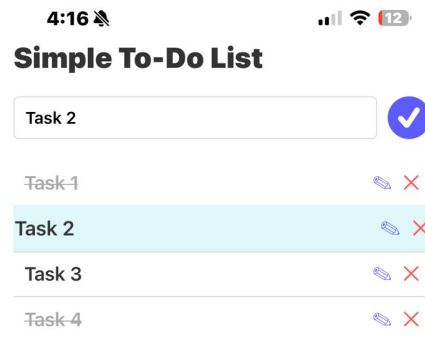


Figure 13: Editing Task in the To-Do List

- Explain how you managed the UI for editing tasks.
- To manage the UI for editing tasks, I implemented a system where tapping on a task allows users to modify its content. The task being edited is identified using the `‘editingTaskId‘` state, which stores the ID of the task currently being edited. When a task is selected for editing, its text is pre-filled into the input field, allowing users to make changes. The input field’s value is controlled by the `‘task‘` state, which holds the current text being edited. Upon saving the changes, the `‘updateTask‘` function is called to update the task’s text in the state by mapping over the `‘tasks‘` array and replacing the old text with the new one. The task list is then updated, and the changes

are persisted to AsyncStorage. Additionally, the add button's text changes to a checkmark when editing, giving users a visual cue that the task is being edited. After updating, the input field is cleared, and 'editingTaskId' is reset to 'null' to signal the end of the editing process. This approach ensures a smooth and intuitive editing experience while maintaining state consistency.

- **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.
- I implemented two key animations in the app using React Native's 'Animated' API: fade-in and fade-out. When a task is added, it smoothly fades into view, enhancing the user experience by making the addition feel polished and dynamic. Similarly, when a task is deleted, it fades out before being removed, providing a smoother transition and visual feedback. These animations make the app feel more fluid and responsive, reducing abrupt changes and creating a more engaging interaction. Overall, they contribute to a polished user interface by providing clear and visually appealing feedback during task additions and deletions.

```
const fadeIn = () => {
  Animated.timing(fadeAnim, {
    toValue: 1,
    duration: 500,
    useNativeDriver: true,
  }).start();
};

const fadeOut = (taskId) => {
  Animated.timing(fadeAnim, {
    toValue: 0,
    duration: 500,
    useNativeDriver: true,
  }).start(() => {
    const updatedTasks = tasks.filter((item) => item.id !== taskId);
    setTasks(updatedTasks);
    saveTasks(updatedTasks);
  });
};
```

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.