Lab 3: Building a Mobile App with React Native

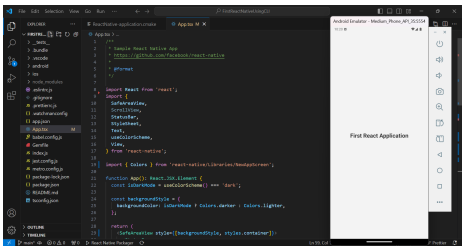
Task 1: Set Up the Dev Environment

Q1: **Attach screenshots of your app running on an emulator and on a physical Android or iOS device**

Answer: The image below shows the application running through React-Native CLI and I used a little different installation as the document given to us mentions to use
npm install -g react-native-cli
Note: If the global `react-native-cli` package has been deprecated, you can use the local version with `npx`:
npx react-native init YourProjectName

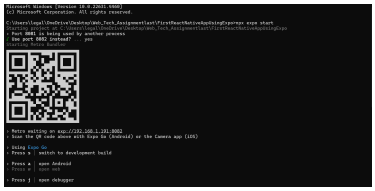
But the above commands didn't work it says this package is deprecated so instead i used these commands

npx @react-native-community/cli init MyProject



Project Running on Physical Android and Expo

Command for Running Expo



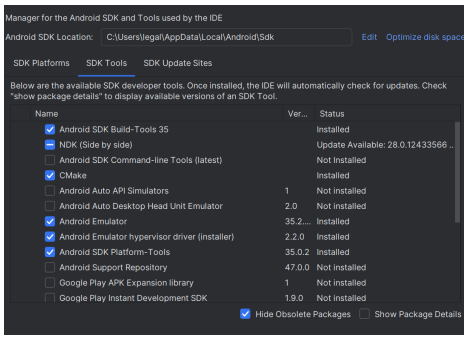
Setting Up an Emulator

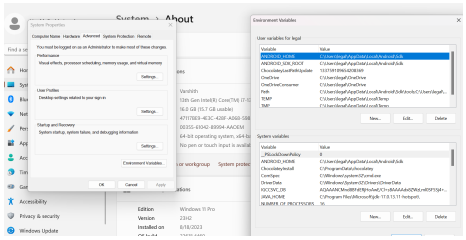
Q2: **Explain the steps you followed to set up an emulator in Android Studio or Xcode?**

1.First I Created an React-Native Cli Application using the command npx @react-native-community/cli init MyProject

2.Download andriod studio Application and download the required SDK tools

3.After downloading Android Studio then add ANDROID-HOME to the envirnoment variables





4.Navigate to the Application and run the application using the command npx react-native start

5.Click on "a" to run as an andriod application

Q3: **Discuss any challenges you faced during the setup and how you overcame them?**

There were many challenges while setting up the react-native application the command "npx react-native init Projectname" is depreceted so i found out an identical command "npx @react-native-community/cli init MyProject" which helped me to create a project. but while running the application using the command "npx react-native start" there were some configuration issues in "Cmake" so always to find out the configuration issues try this command "npx react-native doctor" this command help me find there are some configuration issues of Android SDK tools. then i fixed that by adding the "ANDROID-HOME" to the environment variables.

Q4: **Describe how you connected your physical device to run the app using Expo.?**

Connecting to the Expo is rather simple there were not much challenges.

1.Firstly install expo by using the command "npm install -g expo-cli"

2.Create the Application by using the command "npx expo init YourProjectName"

3.Navigate into the project

4.Run the command "npx expo start" to start the application and click on "a" to run on andriod and download the expo application in your mobile and there would a QR code on the command prompt scan the QR and the application opens in Expo

Q5: **Include any troubleshooting steps if you encountered issues?** There are not many issues i encountered in this setup but i got some syntactical errors.

Q6: **Compare and contrast using an emulator versus a physical device for React Native development. and discuss the advantages and disadvantages of each option?**

Emulator:
advantages:
The emulator offers quick testing of apps without needing a physical device, and it allows simulating different device configurations such as screen sizes, resolutions, and Android versions for cross-platform compatibility. It is also free from hardware dependency, as long as the system meets the requirements.
Disadvantages:
emulators tend to be slower than physical devices, with potential lag in animations and overall performance. They are also resource-intensive, consuming significant CPU and memory, which can slow down the host system. Additionally, certain real-world features like camera access, GPS, and physical sensors may not function accurately on an emulator.

Physical Device:
Advantages:
Running the app on a physical device provides a more accurate representation of the user experience, including smoother performance, better responsiveness, and real sensor data. It also enables testing of physical features such as the camera, GPS, and touch input, which may not be replicated on an emulator.
Disadvantages:
On the downside, setting up a physical device for testing requires additional steps like installing Expo or USB drivers, and the testing is limited to the specific device you have. If you need to test on different screen sizes or Android versions, you must own multiple devices. Additionally, the availability of the device can be a limitation, as it may be in use for other tasks.

Q7: **Identify a common error you encountered when starting your React Native app.Note that it is very unlikely that everyone**

will get the same error here and Explain the cause of the error and the steps you took to resolve it.

Configuration Issues with Android SDK Tools (CMake)

Cause: While running the app using the command npx react-native start, you encountered configuration issues related to CMake (a build tool). This is a common issue related to missing or misconfigured Android SDK tools.

Steps to Resolve:
1.Use the npx react-native doctor command, which helps identify and report any configuration issues related to the Android SDK or other dependencies.

2.The output of this command pointed out that the Android SDK tools were not properly configured.

3.To fix this, I added the ANDROIDHOME environment variable to the system's environment variables to ensure that the Android SDK was properly recognized by React Native. After setting the environment variable correctly, I ran the application again, and the issue was resolved.

Task 2: Building a Simple To-Do List App

Mark Tasks as Complete

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. Explain how you updated the state to reflect the completion status of tasks

In the code, the state reflecting the completion status of tasks is updated using the toggleTaskCompletion function, which maps through the existing tasks array and checks if the id of a task matches the taskId passed to it. If a match is found, it toggles the completed status of that task by using !item.completed. This update is applied to the state by calling setTasks, which replaces the old array with a new one that contains the updated task. The toggle function is triggered when a user taps on a task, as each task is wrapped in a TouchableOpacity that calls toggleTaskCompletion(item.id) upon press. The task's completion status is reflected visually by conditionally applying a completedTask style, which adds a strikethrough and changes the color of the text when the task is marked as completed.

Persist Data Using AsyncStorage.

Implement data persistence so that tasks are saved even after the app is closed.Use AsyncStorage to store and retrieve the tasks list.

I implemented the data persistence using AsyncStorage to ensure tasks are saved even after the app is closed. After Running the Application, the useEffect hook retrieves tasks from AsyncStorage using AsyncStorage.getItem and updates the tasks state with the parsed data. Any changes to the tasks, such as adding, deleting, or toggling completion, are automatically saved to AsyncStorage via another useEffect that calls AsyncStorage.setItem whenever the tasks state changes. This ensures that the task list is consistently stored and restored, providing seamless data persistence across app sessions.

Edit Tasks

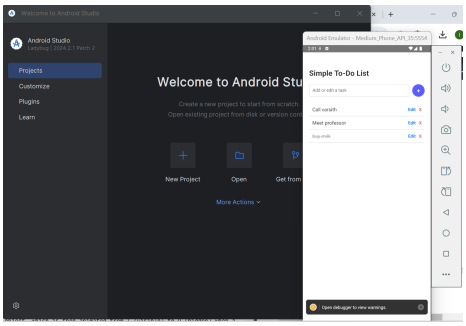
Allow users to tap on a task to edit its content.Implement an update function that modifies the task in the state array.Explain how you managed the UI for editing tasks.

Edit button was added next to each task. When the button is clicked, the task's content is pre-filled into the input field, and the app enters an "editing mode." This is managed using two state variables: isEditing, which tracks whether the user is editing or adding a task, and editingTaskId, which stores the ID of the task being edited. The Save button replaces the Add button during editing. When the changes are saved, the saveEditedTask function updates the task content in the state array. This approach provides a clear and intuitive UI for users to modify tasks without confusion.

Add Animations

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.

Animation is implemented using React Native's Animated API. Specifically, two types of animations are used: scale animation and fade animation. The scaleAnimation is triggered when a new task is added, creating a visual effect where the task "grows" in size as it is introduced into the list. This is done using the Animated.timing method with a scaleAnimation value that starts at 0 and animates to 1. The fadeAnimation is applied to each task in the list, where a fade-in effect occurs when a task is added, and a fade-out effect is triggered when a task is deleted. The fade effect is achieved by assigning a unique Animated.Value to each task in the fadeAnimation object, which is then animated from 1 (visible) to 0 (hidden) when a task is deleted. These animations enhance the user experience by making the interface feel more dynamic and responsive, adding smooth transitions that improve the visual appeal of task management actions like adding and deleting tasks.



Link For GitHub
https://github.com/VarsithReddyLegala/WebTechnologies-Lab3

References
Took the references of online sources and also used AI tools for syntax and also for setting up the environment