Lab3-Web Technology

Sudheepa Katakam

001282273

Task-1

(1) Screenshots of Your App (5 Points)

- Attach screenshots of your app running on an emulator and on a physical Android or iOS device.

- Describe any differences you observed between running the app on an emulator versus a physical device.

  The application runs more smoothly and responsively on a physical device, and touch gestures and other interactions feel more natural. Performance can lag and the emulator is generally slower, particularly when it comes to hardware-dependent functionality or real-world testing.

(2) Setting Up an Emulator (10 Points)

- Explain the steps you followed to set up an emulator in Android Studio or Xcode. Discuss any challenges you faced during the setup and how you overcame them.

  After downloading Xcode, I downloaded the simulator and selected the version. The simulator ran when I tried to run the project. Due to the outdated version, I had to download it again and choose the correct version.

(3) Running the App on a Physical Device Using Expo (10 Points)

- Describe how you connected your physical device to run the app using Expo. Include any troubleshooting steps if you encountered issues.

  First, I installed the Expo Go app on the phone, then ran `npx run start` in the terminal. I scanned the QR code, which loaded the app on the device. The app showed a network issue when I scanned the QR code.

Figure 1: .

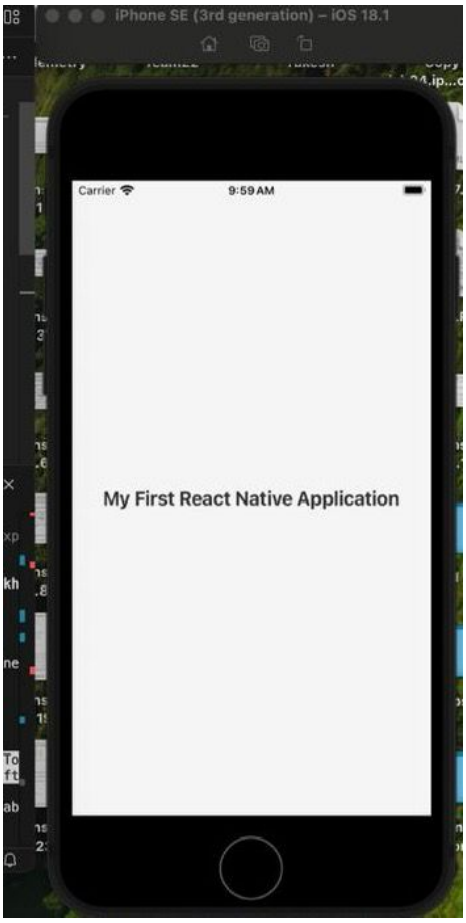

iPhone SE (3rd generation) – iOS 18.1
Carrier
9:59 AM
My First React Native Application

Figure 2: 2

(4) Comparison of Emulator vs. Physical Device (10 Points)

- Compare and contrast using an emulator versus a physical device for React Native development.

  Although using an emulator can be faster and may not always accurately represent how the app will function in real life, it is useful for rapidly testing your app across multiple devices without having to switch between physical devices. Although it can be a little difficult to set up and requires a connection, a physical device provides a more authentic experience by demonstrating how your program works with real hardware and under real-world circumstances. Physical devices are necessary when precise testing is required prior to deployment, but emulators are excellent for rapid iterations.

5) Troubleshooting a Common Error (5 Points)

- 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. Explain the cause of the error and the steps you took to resolve it.

  When I tried to launch the application, I frequently got the "Could not connect to the server" error. This was caused by a problem with the connection between my development server and the emulator or actual device. To fix it, I made sure my device and PC were connected to the same Wi-Fi network, used `npx expo start` to restart the Expo server, and verified that the device could reach the local server by scanning the QR code from Expo Go.

Task-2

(a) 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.
- Explain how you updated the state to reflect the completion status of tasks.

  I included a function called `toggleComplete` that, when clicked, modifies the task's completion state so that people can mark it as completed. Additionally, the task's appearance changes to indicate whether it has been finished, for example by turning the text gray or displaying a strikethrough. To keep the user interface (UI) in sync with the data, the state is updated whenever a task is designated as complete.

(b) 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.

  I used AsyncStorage, which lets us save key-value pairs, to ensure tasks are saved even after the app has been closed. I save the task list to AsyncStorage whenever it updates. I restore the tasks by loading them from storage when the app launches. This ensures that when users reopen the app, their data is preserved. I store the data as a string and retrieve it using `JSON.parse` and `JSON.stringify`.

(c) 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.
- Explain how you managed the UI for editing tasks.

  I gave users the option to tap on a task to make a direct content modification. Each task now has a `TextInput` that allows users to enter new content. When the text changes, the state updates the task, and the user interface is updated instantly. The text property in the state associated with the task is updated when the user makes changes to the text. This allows the user to view their changes in real time.

(d) Add Animations (10 Points)

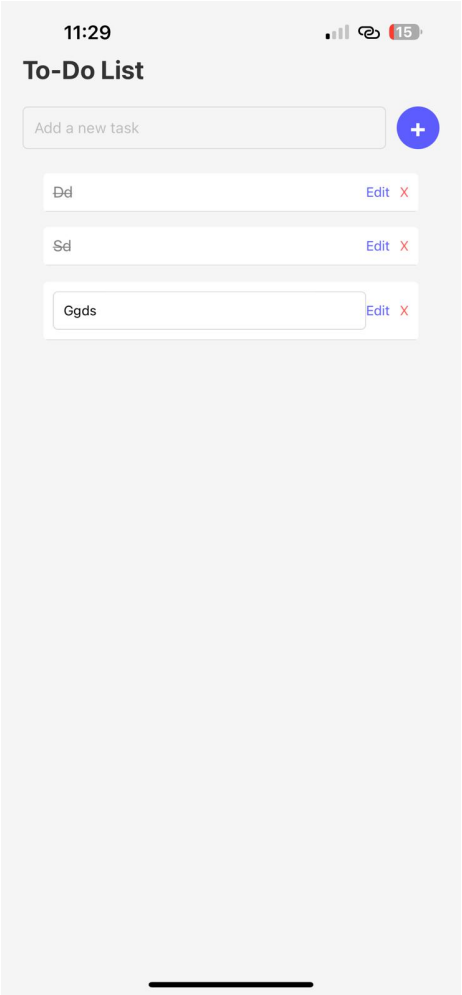
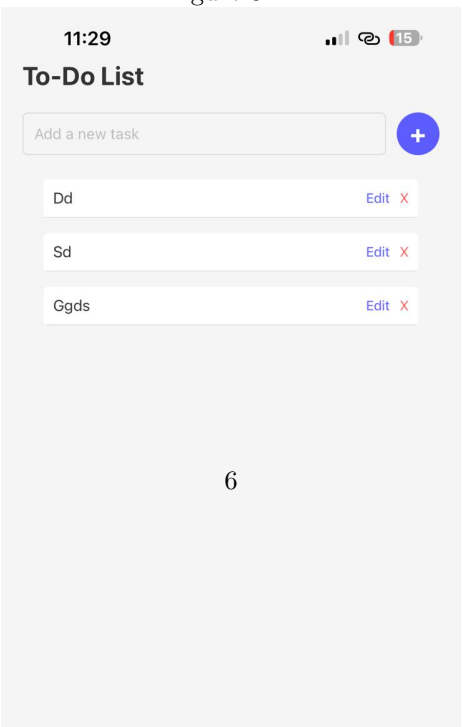

11:29
15
To-Do List
Add a new task
Dd
Edit X
Sd
Edit X
Ggds
Edit X

Figure 3: 1


11:29
15
To-Do List
Add a new task
Dd
Edit X
Sd
Edit X
Ggds
Edit X

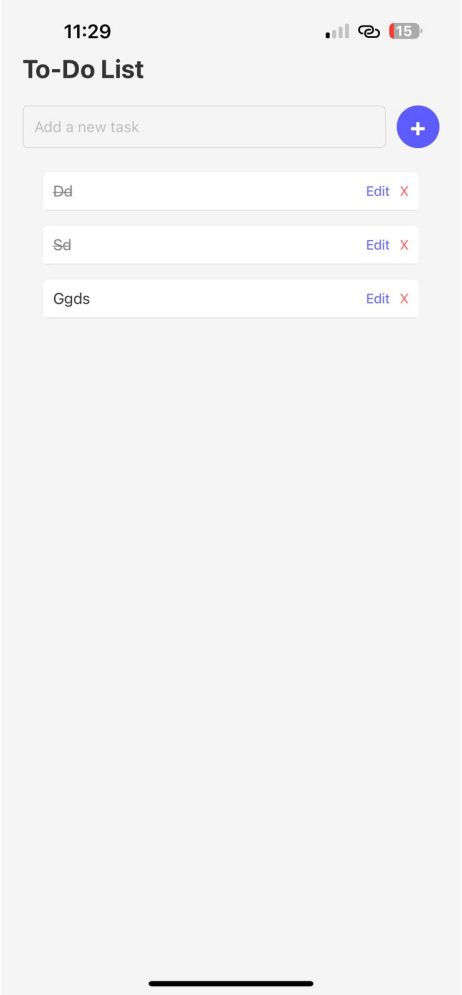


Figure 5: 3

- 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 used React Native's Animated API to create animations to give the app a more dynamic feel. I applied a fade effect to tasks when they were added or removed. The task's opacity is smoothly changed by the animation, producing a pleasing transition. Instead of tasks appearing and disappearing abruptly, this makes the app feel more responsive and polished, enhancing the user experience.

I have used Chatgpt to check my code errors , latex conversion
This is my GitHub link for my Assignment
`https://github.com/katakamsudheepa/project1w`