Web Technologies Lab – 3

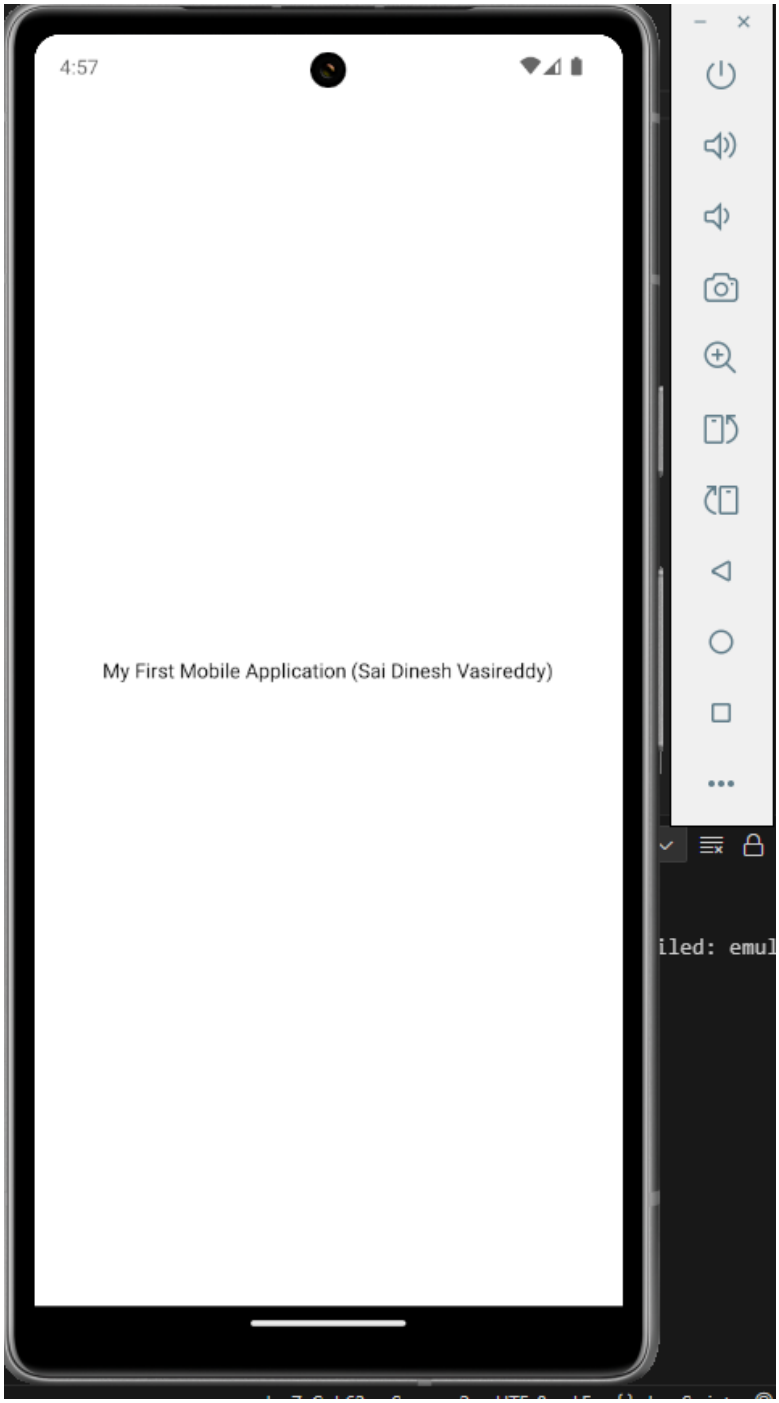
Sai Dinesh Vasireddy
001271330

November 2022

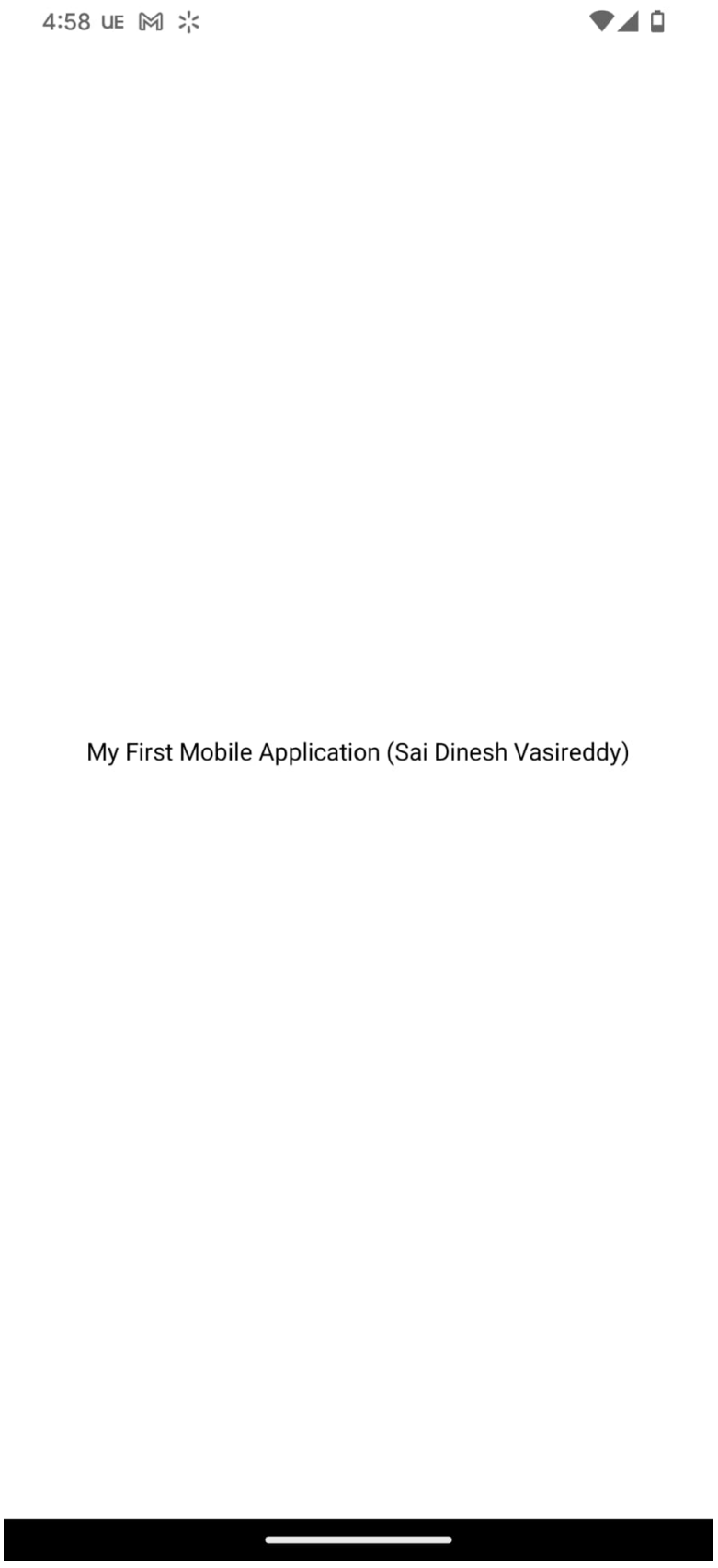
Task 1: Set Up the Development Environment

1. Screenshots of Your App (5 Points)

Emulator:



Android Device:



Differences between Emulator and Physical Device:

A few notable differences are:

- **Performance:** Emulators are slower due to factors like system load and app complexity.
- **Accessibility:** Navigation may be slower on the emulator compared to the physical device.
- **Orientation:** Changing orientation on an emulator can take some time to get used to, while rotating a physical device is much more intuitive.

2. Setting Up an Emulator (10 Points)

Steps to Set Up the Emulator in Android Studio:

After installing Android Studio, I set up the Android SDK by selecting the required SDK checkboxes. Key steps include:

- Setting up the `ANDROID_HOME` environment variable.
- Selecting a device in `Virtual Device Manager` (I selected my physical device "Pixel 7a" with an outdated Android version 14).
- Setting up `build-tools` and `emulator` path variables.

Challenges Faced:

During setup, I encountered an issue with the deprecation of the `react-native` command, which caused `npx react-native init` to fail. After referring to the official React Native documentation, I switched to `npx @react-native-community/cli@latest init` to resolve the issue.

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

Steps to Connect Physical Device:

I ensured both my PC and mobile were on the same Wi-Fi network, then followed these steps:

- `npm install -g expo-cli`
- `npx expo init myMobileApp`
- `cd myMobileApp`
- `npx expo start`

A QR code was generated which I scanned with my mobile device, after which the app built and started running.

Troubleshooting:

No major issues, but occasionally the app would not load if Wi-Fi connectivity was unstable.

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

Emulator:

- **Advantages:**
  - Cost-effective: No need for a physical device.
  - Accessible on any PC.
  - Test on multiple device configurations.
- **Disadvantages:**
  - Slower performance, especially for complex apps.
  - Doesn't replicate real-world usage (e.g., GPS, camera).
  - Limited features like Bluetooth support.

Physical Device:

- **Advantages:**
  - Realistic testing with sensors, GPS, etc.
  - Faster performance and smoother apps.
  - Easier debugging.
- **Disadvantages:**
  - Requires purchasing a physical device.
  - Setup hassles like enabling developer mode.
  - Limited to testing on a single device.

In short, emulators are great for quick testing, while physical devices offer more realistic testing.

5. Troubleshooting a Common Error (5 Points)

I encountered a common error when trying to run my app due to the deprecation of the `react-native init` command. The error was resolved by using `npx @react-native-community/cli@latest init` instead. Additionally, there was an issue with `cmake` during the build process, where the wrong directory separator ( ) was used. The fix involved editing the `ReactNative-application.cmake` file to replace with `/`.

Task 2: Building a Simple To-Do List App

a. Mark Tasks as Complete (15 Points)

I implemented a toggle function that allows users to mark tasks as completed. The tasks are displayed with a strikethrough and a change in background color to light green once marked complete.

State Management for Task Completion:

To manage the completion status, I used React's `useState` to update the state when a task is toggled. Here's a sample code snippet i reffered to:

```
const [tasks, setTasks] = useState([]);
const toggleComplete = (index) => {
    const newTasks = [...tasks];
    newTasks[index].completed = !newTasks[index].completed;
    setTasks(newTasks);
};
```

b. Persist Data Using AsyncStorage (15 Points)

I used `AsyncStorage` to store and retrieve tasks, ensuring that the tasks persist even after the app is closed. Here's how:

- Tasks are saved to AsyncStorage whenever the state updates.
- On app load, the task list is fetched from AsyncStorage and displayed.

c. Edit Tasks (10 Points)

Users can edit tasks by tapping on them. I provided an input field that repopulates with the selected task's text. Once edited, the task is updated in the state array.

d. Add Animations (10 Points)

I used the `Animated` API from React Native to add visual effects when adding or deleting tasks. The tasks fade out smoothly when deleted, providing a better user experience.

Usage of LLM (ChatGPT)

I used ChatGPT during the initial setup to resolve issues related to the deprecated `react-native` command. Later, for the `cmake` issue, I relied on GitHub discussions and documentation for the correct solution.

Demonstration Video and GitHub Repository Link

Demonstration Video: You can view the video demonstration of the app's functionalities here:
`https://drive.google.com/drive/folders/1WKLvGSkZYBchFctsADTpRQOHJVQMc1xw?usp=sharing`
GitHub Repository: The source code of the project is available on my GitHub repository:
`https://github.com/DineshVasireddy/MyToDo`