React Native TodoList App

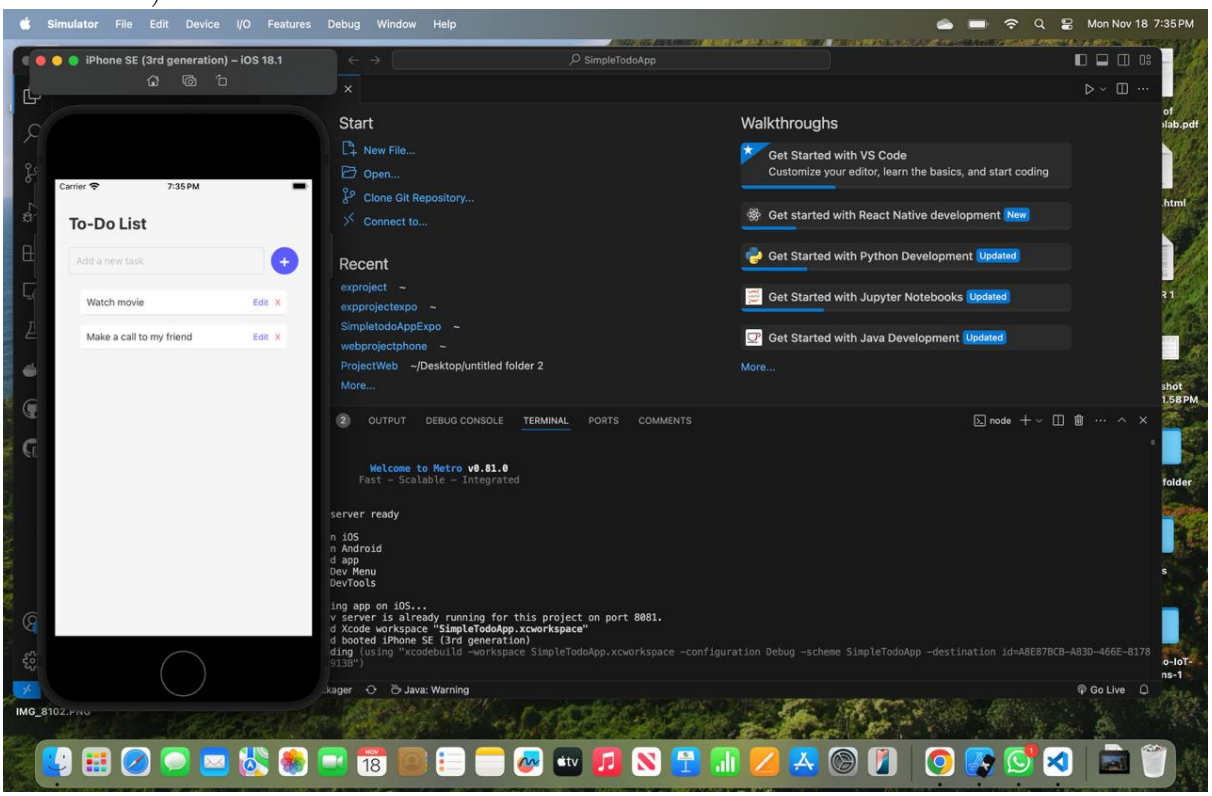
Deepak Naga Sai Vivek Kancharla

November 17th 2024

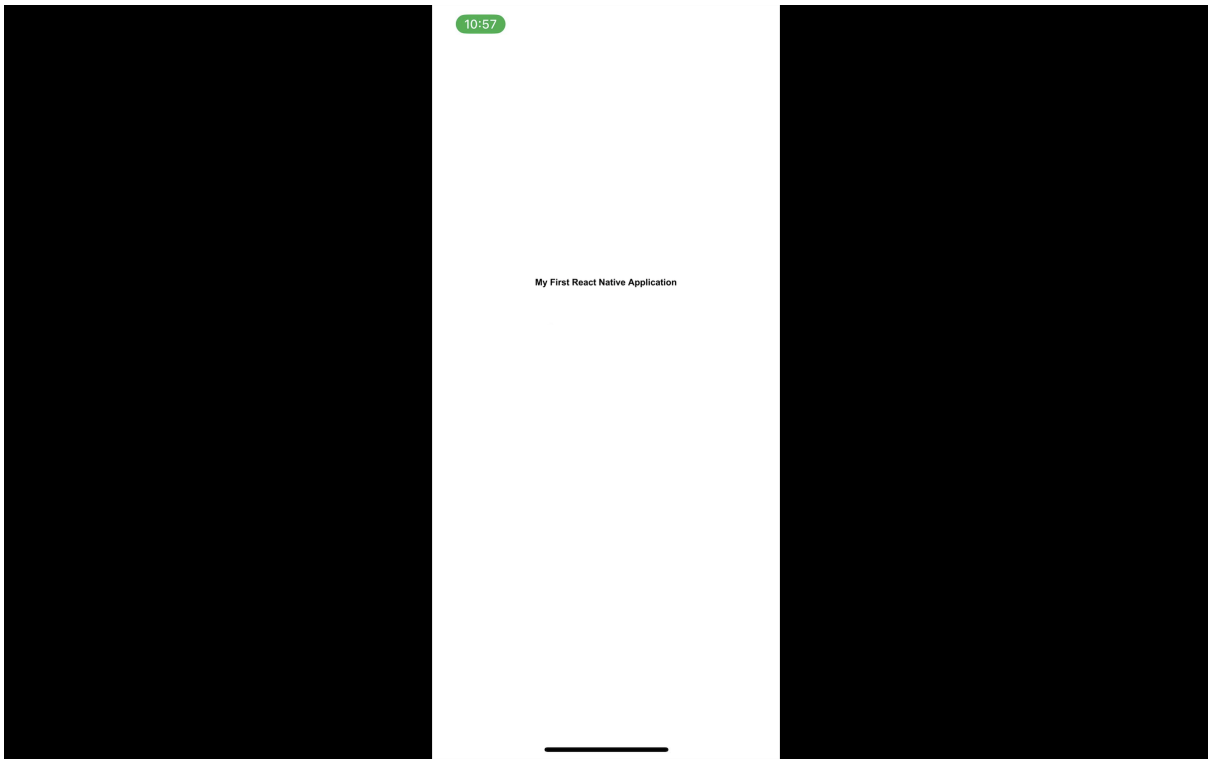
1. Screenshots of Your App

Emulator Screenshot

Take a screenshot of your app running on an emulator (e.g., Android Studio Emulator or iOS Simulator).



Physical Device Screenshot



Differences Observed

- **Performance:** The app typically runs faster on physical devices due to direct hardware usage, while emulators may experience lag.
- **Device-Specific Features:** Physical devices allow testing real-world features like camera, GPS, or touch gestures, which might not behave the same in an emulator.
- **Screen Resolution:** Physical devices display apps at native resolution, which may differ from emulator settings.

2. Setting Up an Emulator

Steps Followed

For iOS:

1. Install Xcode from the App Store.
2. Open Xcode ¿ Preferences ¿ Platforms and download the necessary simulators.
3. Launch the iOS Simulator from Xcode or the terminal with:

```
xcrun simctl list
xcrun simctl boot <device-id>
```

Challenges and Solutions

- **Challenge:** Emulator stuck at boot screen.
  Solution: Increase RAM allocation in the emulator settings or use a different system image.

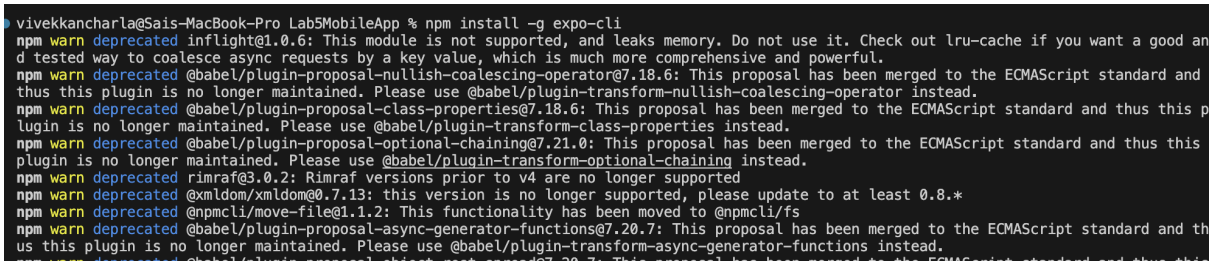
- **Challenge:** Slow emulator performance.
  Solution: Enable hardware acceleration (HAXM for Intel processors or Hypervisor Framework for macOS).

3. Running the App on a Physical Device Using Expo

Steps Followed

1. Install the Expo CLI:

```
npm install -g expo-cli
```

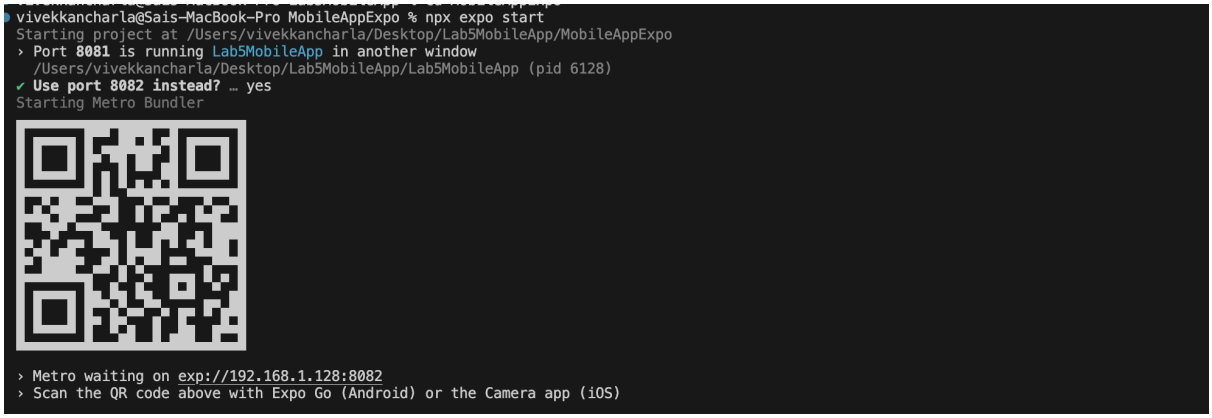


2. Create a project:

```
npx expo init YourProjectName
```

3. Start the Expo development server:

```
npx expo start
```



4. Install the Expo Go app on your device (from App Store or Google Play).
5. Scan the QR code displayed in the Expo CLI using the Expo Go app.
6. Your app runs on the device immediately after scanning.

Troubleshooting Steps

- **Issue:** QR code not working.
  Solution: Ensure both the device and computer are on the same Wi-Fi network.
- **Issue:** App crashes on the device.
  Solution: Check for incompatible dependencies and reinstall node modules.

4. Comparison of Emulator vs. Physical Device

Emulator

Advantages:

- Easy to set up and use.
- Debugging tools are integrated (e.g., logcat in Android Studio).
- No need for physical hardware.

Disadvantages:

- Slower performance, especially with animations or heavy computations.
- Limited support for device-specific features like Bluetooth or real-world GPS.

Physical Device

Advantages:

- Real-world performance and feature testing.
- Faster and smoother UI/UX experience.
- Accurate testing of hardware features (e.g., camera, accelerometer).

Disadvantages:

- Requires USB debugging or Expo setup.
- Dependency on device model for testing compatibility.

5. Troubleshooting a Common Error

Error Encountered

Issue: `EADDRINUSE: address already in use :::8081`.

Cause

The default Metro bundler port `8081` was already being used by another process.

Steps Taken to Resolve

1. Identify the process using the port:

   ```
   lsof -i :8081
   ```

2. Kill the conflicting process:

   ```
   kill -9 <PID>
   ```

3. Alternatively, use a different port for Metro:

   ```
   npx react-native start --port=8088
   ```

Task 2 Submission: Building a Simple To-Do List App

1 Mark Tasks as Complete

1.1 Implementation Details

Toggle Functionality:

- Added a `toggleComplete` function that updates the completion status of tasks in the state.
- Tasks have a `completed` property in their state object, initialized as `false`.
- When a task is toggled, the state is updated using the `setTasks` function.

Styling Completed Tasks:

- Used conditional styling to apply a strikethrough text style or change the text color for completed tasks.
- Example:

```
const taskStyle = {
  textDecorationLine: task.completed ? 'line-through' : 'none',
  color: task.completed ? 'gray' : 'black'
};
```

1.2 State Update Code

```
const toggleComplete = (id) => {
  setTasks(tasks.map(task =>
    task.id === id ? { ...task, completed: !task.completed } : task
  ));
};
```

2 Persist Data Using AsyncStorage

2.1 Implementation Details

Saving Data:

- Used `AsyncStorage` to store the tasks list when the state changes.
- Added a `useEffect` to save data on every update.

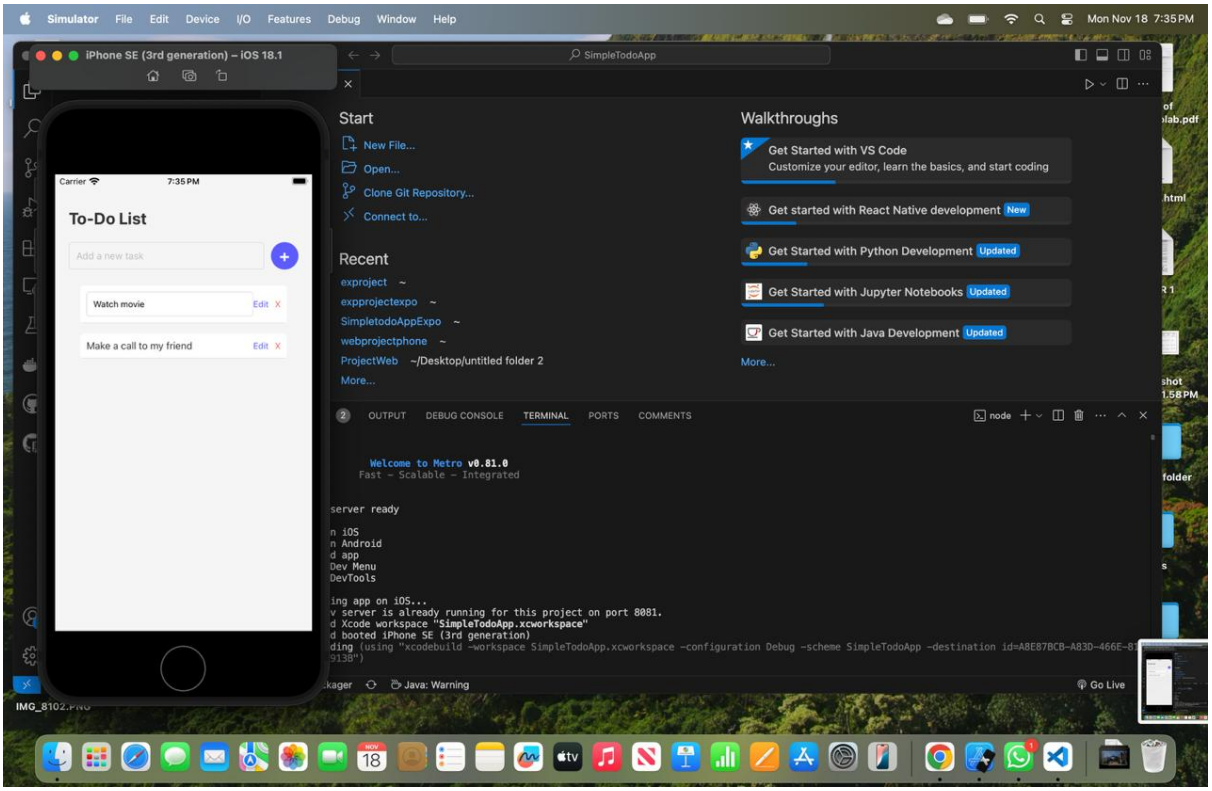
Retrieving Data:

- On app launch, retrieved the stored tasks from `AsyncStorage` and initialized the state.

```
import AsyncStorage from '@react-native-async-storage/async-storage';

useEffect(() => {
  const loadTasks = async () => {
    const storedTasks = await AsyncStorage.getItem('tasks');
    if (storedTasks) setTasks(JSON.parse(storedTasks));
  };
  loadTasks();
}, []);

useEffect(() => {
  AsyncStorage.setItem('tasks', JSON.stringify(tasks));
}, [tasks]);
```



3 Edit Tasks

3.1 Implementation Details

Editing Functionality:

- Added functionality to tap a task to enter edit mode.
- In edit mode, the task's text is replaced with an editable `TextInput` component.

Updating State:

- Implemented an `updateTask` function to modify the content of a task in the state array.
- Used a temporary state variable to track the current editing task.

```
const [editingTask, setEditingTask] = useState(null);
const [newText, setNewText] = useState('');

const updateTask = (id) => {
  setTasks(tasks.map(task =>
    task.id === id ? { ...task, text: newText } : task
  ));
  setEditingTask(null);
};
```

3.2 UI Management

- Rendered a `TextInput` when `editingTask` equals the task ID.
- Rendered regular text otherwise.

4 Add Animations

4.1 Implementation Details

Adding Animations:

- Used the React Native `Animated` API to enhance user experience.
- Added animations for task addition and deletion using `Animated.Value`.

Task Addition:

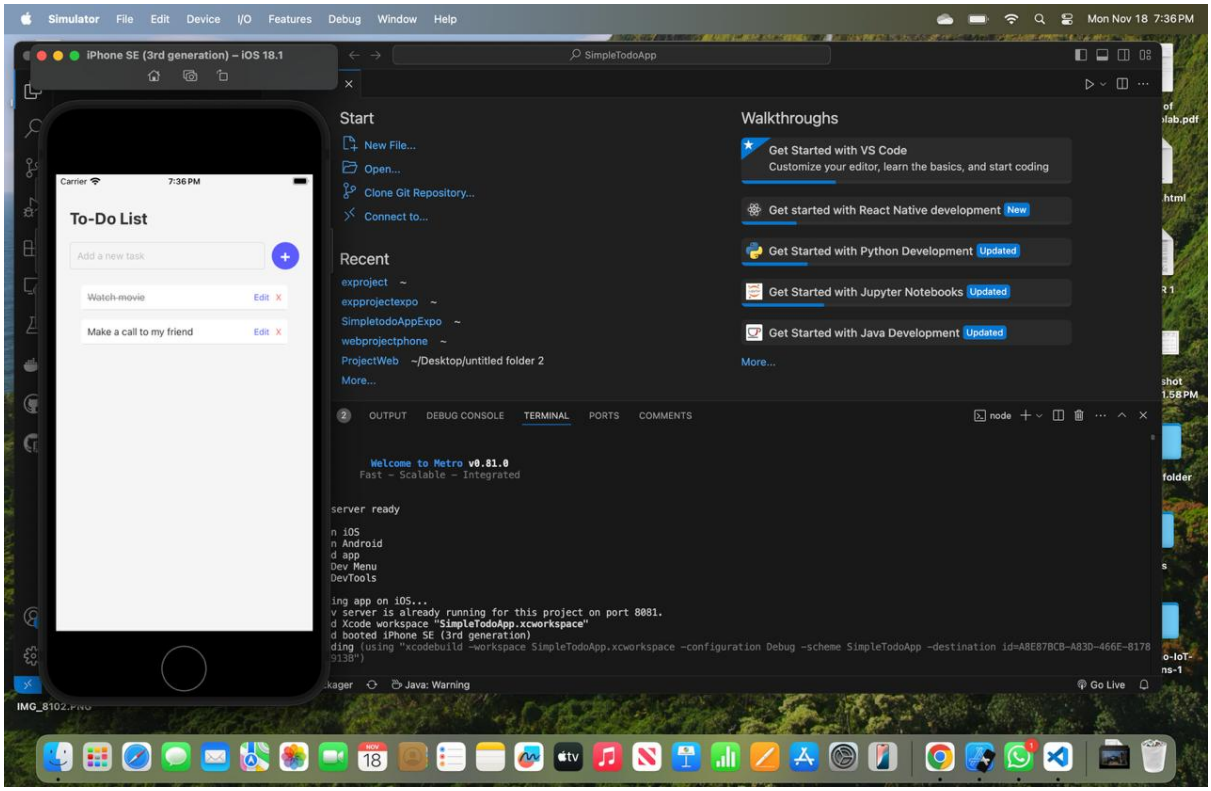
- Applied a fade-in effect using `Animated.timing`.
- Code:

```
const fadeIn = new Animated.Value(0);

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

Task Deletion:

- Implemented a slide-out effect before removing the task from the state.



4.2 Description

- **Fade-in Animation**: Tasks gradually appear when added.
- **Slide-out Animation**: Tasks slide out to the left when deleted.
- These animations make the app feel more dynamic and engaging.

5 Conclusion

The React Native To-Do List app was extended with the following features:

1. **Mark Tasks as Complete**: Users can toggle the completion status of tasks with updated styles.
2. **Persist Data Using AsyncStorage**: Task data is stored and retrieved seamlessly, ensuring persistence.
3. **Edit Tasks**: Tasks can be edited directly within the app, enhancing flexibility.
4. **Add Animations**: Visual effects make the app more engaging and professional.

These enhancements greatly improved the app's usability and user experience.

GitHub Repository

- https://github.com/SaiVivekKancharla/TodoListApp