Web Technologies - Lab 3

Manoj Sandadi

November 17, 2024

1 Task-1

1.1 Screenshots of App

Below are the screenshots of the application:

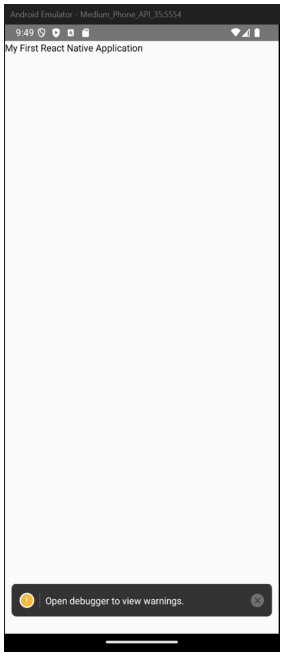

Figure 1: React Native App on Emulator

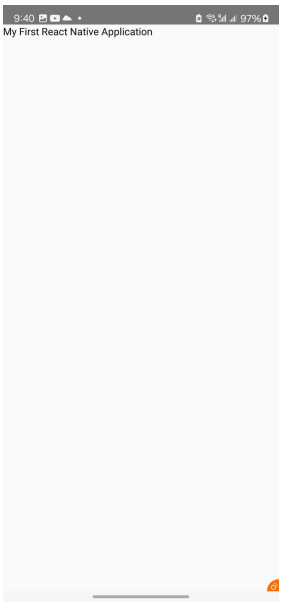

Figure 2: React Native App on a Physical Device

The emulator is slower compared to the physical device. Running the app on a physical device is faster and more convenient.

1.2 Setting up the Emulator

The steps to set up an Android emulator and configure the React Native environment are as follows:

1. Install Android Studio and add the Android SDK's `Android_Home` to the system `PATH` environment variable.
2. Install Java, `npm`, and the React Native CLI globally on your system.
3. Set paths for Java and `npm` in the system environment variables to ensure they are accessible from any directory.
4. Run the following command to create a new React Native app named `app`:

```
npx react-native init app
```

5. Navigate to the app directory:

```
cd app
```

6. Start the Metro development server by running:

```
npx react-native start
```

7. Select the option `"a"` to run the Android emulator.

1.2.1 Error: Metro Bundler Failed to Start

The Metro bundler fails to start or crashes when running `npx react-native start`.

1. Ensure no other instance of the Metro bundler is running. Close all existing terminals or processes using the following command:

```
killall -9 node
```

2. Clear the Metro bundler cache:

```
npx react-native start --reset-cache
```

3. Restart the Metro bundler and verify if it works correctly.

1.3 Running the App on a Physical Device Using Expo

To run the app on a physical device, install the Expo Go app on your mobile device. Start the development server and scan the QR code displayed in the terminal or Metro bundler dashboard. This allows you to run the app directly on the device for testing.

1.4 Comparison of Emulator vs. Physical Device

React Native development can be performed using either an emulator or a physical device. Below is a comparison of the two options:

- **Emulator:**
  - **Advantages:**
    - Easily accessible and does not require additional hardware.
    - Supports testing on multiple virtual devices with different screen sizes and Android versions.

    - Built-in tools allow debugging directly in the development environment.
  - **Disadvantages:**
    - Slower performance compared to physical devices, especially on machines with limited resources.
    - Limited to mouse and keyboard inputs, making it less ideal for testing touch gestures.

- **Physical Device:**
  - **Advantages:**
    - Provides accurate performance metrics and real-world user experience.
    - Allows testing of hardware features like GPS, camera, and accelerometer.
    - Faster and smoother compared to emulators for app interactions.
  - **Disadvantages:**
    - Requires USB debugging setup and drivers for connectivity.
    - Limited to the specific hardware model and Android version of the physical device.

1.5 Troubleshooting a Common Error

Error Encountered: Node.js Version 22 Compatibility

While attempting to initialize a new React Native app using the command:

```
npx react-native init NamelessApp
```

The following issues and warnings were encountered:

- Deprecated module warnings, such as:

```
inflight@1.0.6: This module is not supported,
    and leaks memory.}
@babel/plugin-proposal-class-properties: This
    proposal has been merged to ECMAScript.
rimraf versions prior to v4 are no longer
    supported.}
 Deprecation notice for the init command:
The init command is deprecated. Switch to npx
    @react-native-community/cli init
Fatal error:
Error: spawn npx ENOENT
Node.js v22.11.0
```

This error indicates a compatibility issue with Node.js version 22.

Cause:
The error occurred due to the usage of an incompatible Node.js version (v22.11.0). React Native and its dependencies require a stable and supported version of Node.js .

Steps to Resolve:

1. **Check the Current Node.js Version:**
   Verify the installed Node.js version using:

   ```
   node -v
   ```

   In this case, the version was `v22.11.0`.

2. **Uninstall the Unsupported Node.js Version:**
   Uninstall Node.js v22 using your system's package manager. For example, on a Unix-based system:

   ```
   sudo apt remove nodejs
   ```

3. **Install a Supported Node.js Version:**
   Download and install a stable LTS version of Node.js (e.g., v18 or v20) from the official website (`https://nodejs.org/`) or using `nvm`:

   ```
   nvm install --lts
   ```

4. **Clear the Cache and Reinstall Dependencies:**
   After updating Node.js, clear the npm cache and reinstall React Native CLI:

   ```
   npm cache clean --force
   npm install -g react-native-cli
   ```

5. **Reinitialize the React Native App:**
   Use the updated initialization command as per the deprecation notice:

   ```
   npx @react-native-community/cli init NamelessApp
   ```

6. **Verify the Installation:**
   Start the Metro bundler and run the app to ensure everything works correctly:

   ```
   cd NamelessApp
   npx react-native start
   ```

2 Task-2

This section covers the additional features implemented in the To-Do List app.

2.1 Mark Tasks as Complete

Implementation:

1. Added a `Switch` component to toggle the completion status.
2. Updated the state using `setTasks()` to store the updated task status.
3. Styled completed tasks conditionally with strikethrough text:

Code:

```
const toggleComplete = (taskId) => {
    setTasks((prevTasks) =>
        prevTasks.map((item) =>
            item.id === taskId ? { ...item, completed: !item
                .completed } : item
        )
    );
};
```

Screenshots:

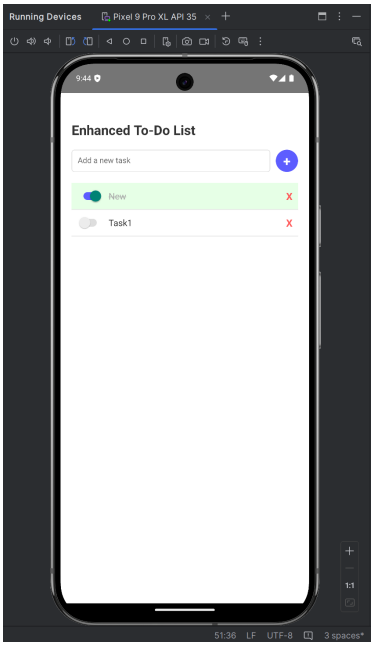

Figure 3: Marking a Task as Complete

2.2 Persist Data Using AsyncStorage

Implementation:

1. Used `AsyncStorage.getItem()` to retrieve tasks when the app loads.
2. Used `AsyncStorage.setItem()` to save tasks whenever the state updates.
3. Ensured the tasks state synchronizes with storage.

Code Snippet:

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

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

Screenshots:

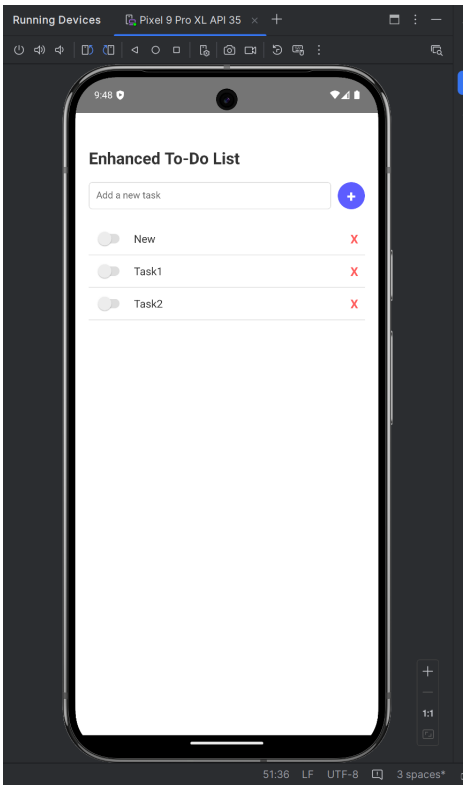

Figure 4: Data Persistence with AsyncStorage

2.3 Edit Tasks

Implementation:

1. Used a `TextInput` to replace the task text when editing.
2. Managed the editing state with `editingTaskId`.
3. Updated the task content in the state array using `updateTask()`.

Code Snippet:

```
const updateTask = (taskId, newText) => {
    setTasks((prevTasks) =>
        prevTasks.map((item) =>
            item.id === taskId ? { ...item, text: newText }
                : item
        )
    );
    setEditingTaskId(null); // Exit editing mode
    setEditingText(''); // Clear editing text
```

```
};
```

Screenshots:

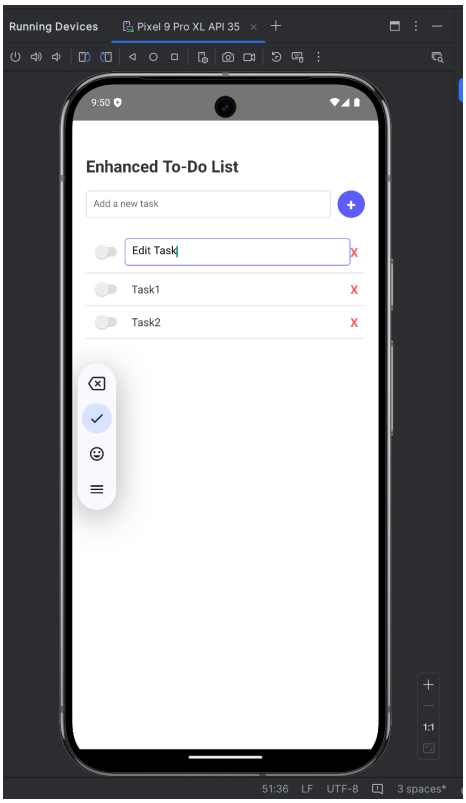

Figure 5: Editing a Task

2.4 Add Animations

Implementation:

1. Added fade-in and scale-up animations for new tasks using `Animated.timing()`.
2. Added shrink-and-fade-out animations for deleted tasks.

Code Snippet:

```
const runAddAnimation = () => {
    fadeAnim.setValue(0);
    Animated.timing(fadeAnim, {
        toValue: 1,
        duration: 300,
        useNativeDriver: true,
```

```
    }).start();
};
```

GitHub Repository: https://github.com/manojSandadi/SimpleToDOApp
AI Tools Used: I have used ChatGpt to debug the code errors and research about the advantages and disadvantages about the reactive native on emulator vs physical Device