22 SEP

How to Select and Use any Currency

Learn to Use Navigation, Setup Redux, Design Components, Work with a Remote API, and More

https://learn.reactnativeschool.com/courses/175915/lectures/17374107

▼ React native CLI vs EXPO

Expo is a framework and platform for building React Native applications. It aims to provide a simplified development experience by abstracting away some of the complexities involved in setting up and configuring a React Native project.

Advantages of Expo

1. **Quick Start**:
2. **Over-the-Air Updates**:
3. **Access to Pre-built Libraries**:
4. **Simplified Build Process:**
5. **Native Physical Device:**

React Native CLI is the traditional approach for building React Native apps. It provides maximum flexibility and control over your project setup and configuration.

Advantages of React Native CLI:

1. **Full Native Module Support:**
2. **Customization Options:**
3. **Ecosystem Compatibility:**

▼ Project development and enviornment config

- `android`: Contains Android-specific code and configuration.
- `ios`: Contains iOS-specific code and configuration.
- `node_modules`: Where your project's dependencies are stored.
- `src`: Typically, this is where you'll place your React Native source code.
- `App.js` or `index.js`: The entry point of your React Native application.

You can organize your project further based on your requirements and preferences.

▼ File structure

```
my-react-native-app/
├── android/                  // Android specific files and build configuration
│   ├── app/
│   ├── gradle/
│   ├── build.gradle
│   └── ...
├── ios/                      // iOS specific files and build configuration
│   ├── Pods/
│   ├── MyReactNativeApp/
│   ├── MyReactNativeApp.xcodeproj/
│   ├── Podfile
│   └── ...
├── node_modules/             // Dependencies installed via npm or yarn
├── src/                      // Source code for your React Native app
│   ├── assets/               // Static assets like images, fonts, and icons
│   ├── components/           // Reusable UI components
│   ├── screens/              // App screens or views
│   ├── navigation/           // Navigation configuration and components
│   ├── redux/                // Redux store configuration and actions
│   ├── services/             // API services, Firebase, or other external services
│   ├── utils/                // Utility functions and helpers
│   ├── App.js                // Entry point of your app
│   └── ...
├── index.js                  // Entry point for the React Native application
├── package.json              // Project dependencies and scripts
├── yarn.lock or package-lock.json  // Lock files for dependency versions
├── metro.config.js           // Metro Bundler configuration (optional)
└── ...
```

▼ Basics of building screens

```
import React from 'react';
import MyScreen from './components/MyScreen';
```

```
function App() {
  return (
    <View style={{ flex: 1 }}>
      <MyScreen />
    </View>
  );
}

export default App;
```

```
function MyScreen() {
  return (
    <View style={styles.container}>
      <Text style={styles.heading}>Welcome to My Screen</Text>
      <Text>This is a simple React Native screen.</Text>
    </View>
  );
}

const styles = {
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  heading: {
    fontSize: 24,
    fontWeight: 'bold',
  },
};
```

▼ Navigation between screen using react navigation

To navigate between screens, you can use the `navigation` prop provided by React Navigation. In the example components above, we use the `navigation.navigate()` method to move between the `Home` and `Details` screens. The `navigation.goBack()` method takes you back to the previous screen.

Need to install

```
# Using npm
npm install @react-navigation/native @react-navigation/stack
```

```
# Using yarn
yarn add @react-navigation/native @react-navigation/stack
```