React Native tutorial

A tutorial by Eduard Shkulipa

Contents

React Native is a powerful open-source framework developed by Facebook for building high-performance mobile applications using JavaScript and React, enabling cross-platform development for both iOS and Android with a single codebase. By utilizing native components, React Native ensures smooth, responsive user experiences comparable to those of native apps. This tutorial will guide you through setting up your development environment, creating and styling user interfaces, implementing navigation, managing state, fetching data, and deploying your app. With React Native's vibrant ecosystem, hot reloading feature, and robust community support, you can efficiently develop stunning mobile apps while leveraging your existing JavaScript and React skills.

Installation

The simplest way to install the react native is to use the expo framework. The regular installation is too confusing and you should move o native development after learning this tutorial. However, these are the steps to install the react native with the expo framework.

Step 1: Install Homebrew

Homebrew is a package manager for macOS that makes it easy to install and manage software. Open your terminal and run the following command to install Homebrew:

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

After the installation is complete, ensure Homebrew is ready to use:

Step 2: Install Node.js and npm

Node.js and npm are essential for React Native development. You can install them via Homebrew:

```
brew install node```
```

Step 3: Install Watchman

Watchman is a tool by Facebook for watching changes in the filesystem. It is highly recommended for React Native development:

```
brew install watchman
```

Step 4: Install Expo CLI

Expo CLI is a command-line tool that makes it easy to set up a React Native project without worrying about complex configurations. Install Expo CLI globally using npm:

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

Step 5: Create a New React Native Project

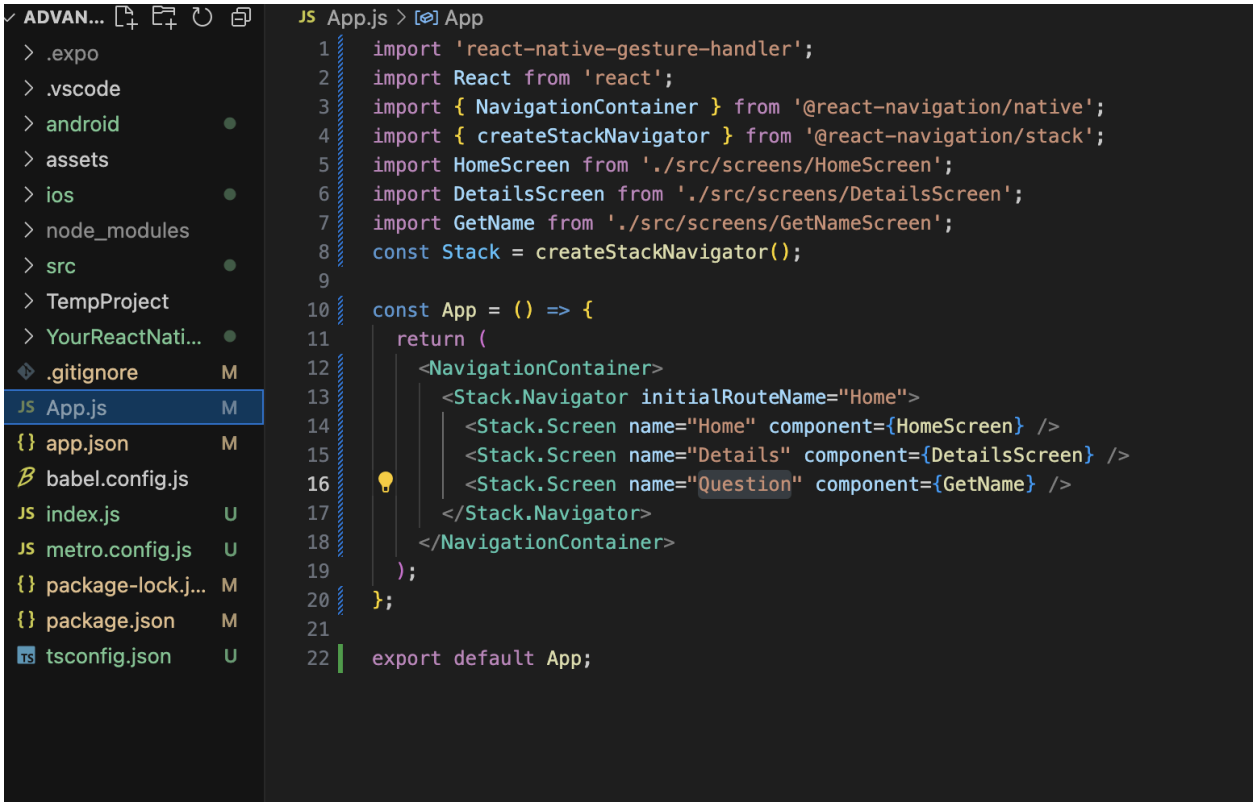
Now that you have all the necessary tools installed, you can create a new React Native project using Expo. Run the following command in your terminal:

```
expo init MyNewProject
```

Navigate into your project directory:

```
cd MyNewProject
```

After that, the project will have already some files and code, and will generally look like this:



Step 6: Start the Development Server

To start the development server, run:

```
expo start
```

Once the project starts, you will see the directions to run the app on the web, the phone, or other options:

Building a simple application

The react native is set up so that you have a main procedure in the App.js file, and each view is a separate object. Each object contains a view that is set up very similar to HTML, and there is a style sheet that is attached to each object, that is similar to css. Similarly to web development, there are event handlers, different objects, and so on.
A very useful feature of the set up as this one is that each object has states and the states can change once the input changes, or anything similar.

Example with different pages:

React Native is a framework that allows developers to build mobile applications using JavaScript and React. The structure of a React Native application is designed to be intuitive and similar to web development, making it accessible for developers with a background in web

technologies. Here is an expanded and improved description of the structure of the React Native framework:

Main Component

The entry point of a React Native application is typically the App.js file. This file contains the main component, often named App, which serves as the root component of the application. Within this main component, you define the overall structure and layout of your app, and it acts as a container for other components (views).

Components and Views

In React Native, each view is represented as a separate component. Components in React Native are analogous to classes or functions that define a portion of the user interface. These components can be nested within each other to create complex UIs. Each component corresponds to a view, similar to how elements are defined in HTML. Here is a simple example:

```
import React from 'react';
import { View, Text } from 'react-native';

const MyComponent = () => {
 return (
   <View>
     <Text>Hello, this is a component!</Text>
   </View>
 );
};

export default MyComponent;
```

Styling

Styling in React Native is handled using a syntax similar to CSS, but it is done through JavaScript objects. Each component can have an associated stylesheet that defines its appearance. This is similar to attaching CSS styles to HTML elements. Here is an example:

```
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const MyStyledComponent = () => {
 return (
   <View style={styles.container}>
     <Text style={styles.text}>This is a styled component!</Text>
```

```
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#f0f0f0',
  },
  text: {
    fontSize: 20,
    color: '#333',
  },
});

export default MyStyledComponent;
```

Event Handling

Just like in web development, React Native allows you to handle events such as user interactions. Event handlers can be attached to components to respond to various events like button presses, text input changes, and more. Here is an example:

```
import React, { useState } from 'react';
import { View, Button, Alert } from 'react-native';

const MyEventComponent = () => {
  const handlePress = () => {
    Alert.alert('Button pressed!', 'You have pressed the button.');
  };

  return (
    <View>
      <Button title="Press Me" onPress={handlePress} />
    </View>
  );
};

export default MyEventComponent;
```

State Management

A powerful feature of React Native is its state management. Each component can maintain its own state, which allows the UI to update dynamically in response to user input or other changes. State in React Native is managed using the useState hook or the setState method in class components. Here is an example using the useState hook:

```
import React, { useState } from 'react';
import { View, Text, TextInput } from 'react-native';

const MyStatefulComponent = () => {
 const [text, setText] = useState('');

 return (
   <View>
     <TextInput
       placeholder="Type here"
       onChangeText={newText => setText(newText)}
       value={text}
     />
     <Text>You typed: {text}</Text>
   </View>
 );
};

export default MyStatefulComponent;
```

References

https://www.youtube.com/watch?v=YysKbNk1tj0&ab_channel=Indently
https://www.geeksforgeeks.org/how-to-create-button-in-react-native-app/
https://stackoverflow.com/questions/48726288/how-to-import-and-export-styles-in-react-native
https://blog.logrocket.com/using-react-native-ble-manager-mobile-app/