Themes in Mobile Application

1. Introduction

Themeing is critical in every application (not only mobile applications)! A good theme attracts potential users to use your application and constitutes a good user experience. As a developer, we should not declare color values one by one in each *CSS* class. In this tutorial, we are going to introduce proper theme construction in a React Native app to build themes for our mobile application on top of what you have learnt in Tutorial 6.

We are going to utilize two libraries, namely *react-native-paper* and *react-native-navigation*. After completing this tutorial, you will learn how to create your own theme and make it consistent throughout the whole application.

react-native-paper: provides pre-designed components including buttons, cards, etc. and theming that follows Material Design guidelines (developed by Google a set of guidelines for creating visually appealing UI across different platforms).

react-native-navigation: an efficient solution focusing on smooth transitions between scenarios, tabs and menus.

2. Setup our app

Before proceeding to constructing themes, we will first have to set up our *react-native* application. In this tutorial, we will be using Expo framework for development. Expo is a multi-platform framework which allows developers to develop and test cross-platform applications easily.

2.1. Open the desired code editor, and go to the directory where you would like to put the root folder of the project

2.2. Run the following command:

```
npx create-expo-app comp3330-tutorial
```

2.3. Replace *App.js* with the following codes:

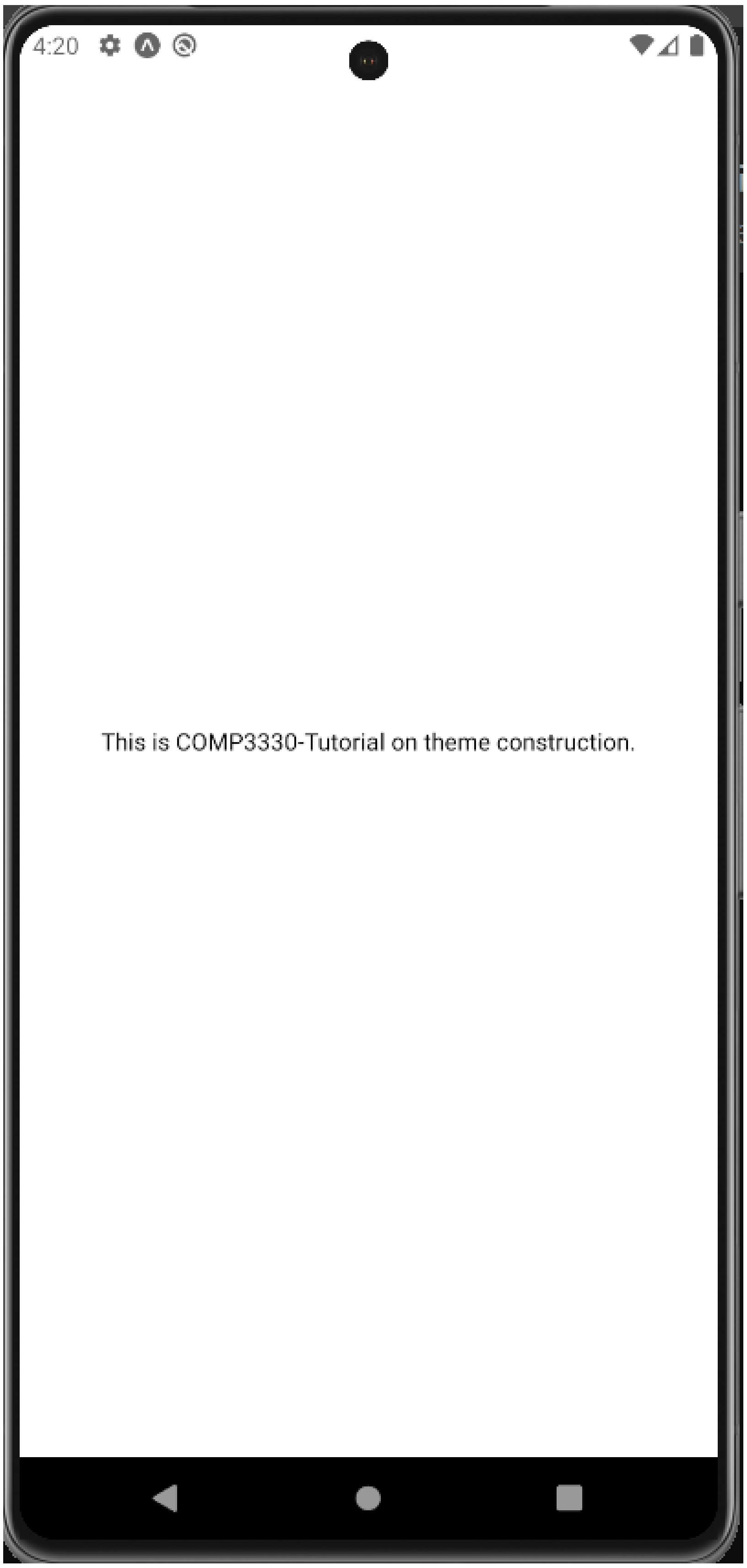
```
import { StatusBar } from "expo-status-bar";
import { StyleSheet, Text, View } from "react-native";
```

```

export default function App() {
  return (
    <View style={styles.root}>
      <Text>This is COMP3330-Tutorial on theme construction.</Text>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  root: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
  },
});

```

2.4. Run the following command:

```
npm start
```

You will see a QR code prompted in your console. Scan it with your own device or open it with an emulator from Android Studio. You shall see a simple UI as follows:



This is COMP3330-Tutorial on theme construction.

Everything is set! Let's go to the main content of this tutorial.

3. Setup theme context

Next, in order to set up a proper theme context, we will need to make use of the *createContext()* method provided in the *react* framework.

A react context is a way to share values, such as state or functions between components without having to pass props through every level of the component tree.

3.1. Insert the following codes to the corresponding place in *App.js.*

```
import { createContext, useState } from "react";
import { StatusBar } from "expo-status-bar";
import { StyleSheet, Text, View } from "react-native";

export default function App() {
  const ThemeContext = createContext({
    isDarkTheme: false,
    toggleTheme: () => {},
    theme: {}
  });
  const [isDarkTheme, setIsDarkTheme] = useState(false);
  const toggleTheme = () => setIsDarkTheme(!isDarkTheme);
  return (
  // provide theme-related context to component tree
    <ThemeContext.Provider value={{ isDarkTheme, toggleTheme }}>
      <View style={styles.root}>
        <Text>This is COMP3330-Tutorial on theme construction.</Text>
        <StatusBar style="auto" />
      </View>
    </ThemeContext.Provider>
  );
}
```

3.2. After implementing the additional codes, you may find that the app still does not have a theme. We are going to add a screen-level pre-designed theme from *react-navigation* to the app.

3.2.1. *React-navgation* library enables us to implement screen-level themeing. In order to use the features, we have to first install the required packages with the following commands.

```
npm install @react-navigation/native @react-navigation/stack
npx expo install react-native-screens react-native-safe-area-context react-native-gesture-handler
```

3.2.2. Afterwards, we can import the package to our *App.js.* In order to implement screen-level rendering, we have to make use of stack navigators. Meanwhile, *react-native-gesture-handler* is a package required by stack navigators for gesture handling. We can develop our stacked home page by adding the following codes in *App.js*. Moreover, the stacked screen must be wrapped in the container provided by *react-navigation*.

Stack navigator refers to the way that screens are managed in a stack-like data structure (Last-In, First-Out)

```
// other import statements
import { createStackNavigator } from "@react-navigation/stack";
import { NavigationContainer } from "@react-navigation/native";
import "react-native-gesture-handler";

export default function App() {
    // other codes
    const Stack = createStackNavigator();

    // Component displaying a text message and the status bar
    const HomeScreen = () => (
    <View style={styles.root}>
        <Text>This is COMP3330-Tutorial on Theme Construction</Text>
        <StatusBar style="auto" />
    </View>
    )

    // Defines the navigation stack, includes a screen named 'Home'
    // corresponding to 'Homescreen' component
    const Router = () => (
        <Stack.Navigator>
            <Stack.Screen
                name="Home"
                component={HomeScreen}
            />
        </Stack.Navigator>
    )
    return (
        <ThemeContext.Provider value={{ isDarkTheme, toggleTheme }}>
            <NavigationContainer>
                <Router />
            </NavigationContainer>
        </ThemeContext.Provider>
    )
}
```

After adding the codes above, you will find that your app now has got one screen header. That means you are on the right track.

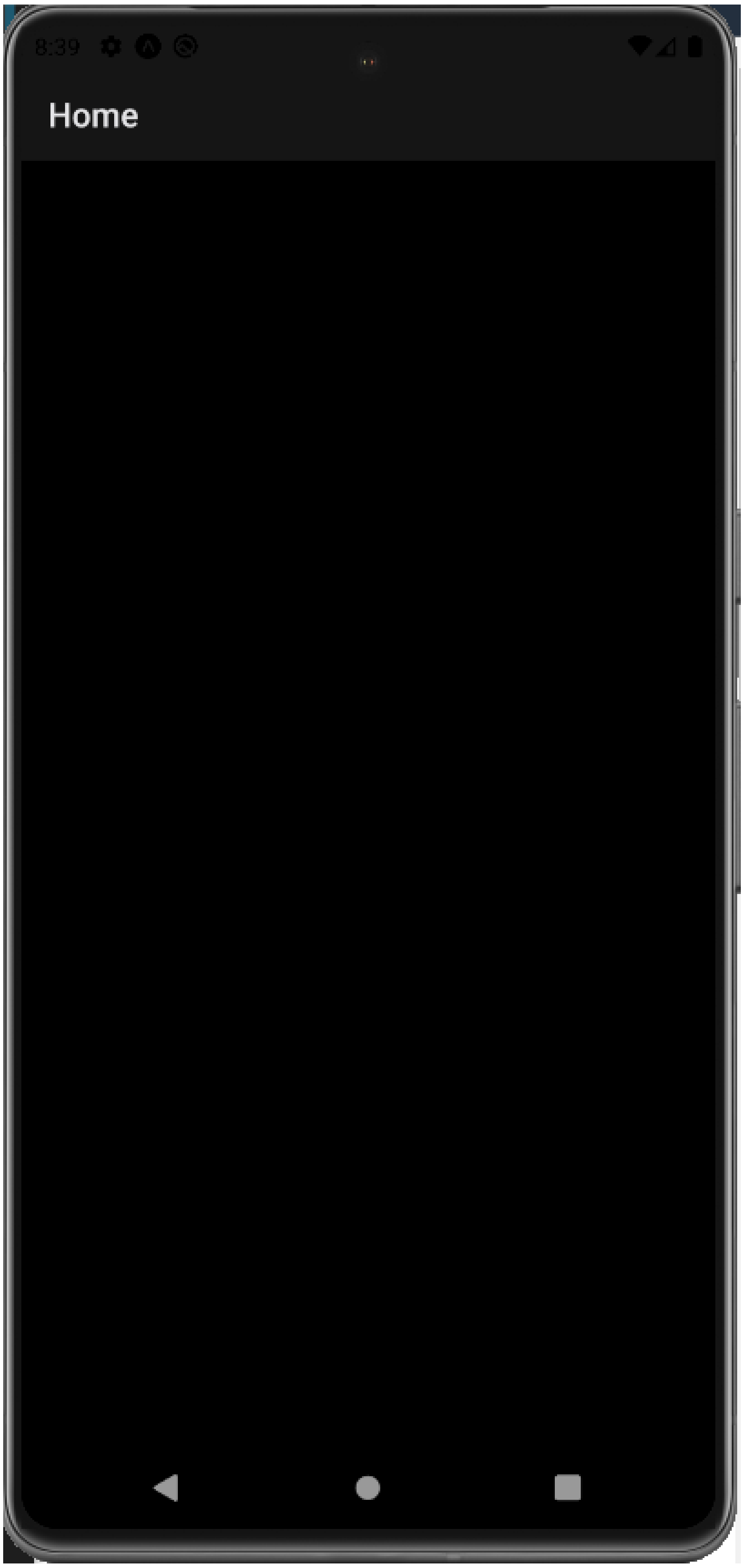
3.2.3 Afterwards, we can add themes to our screen-level components. We have to modify the import statement of "@react-navigation/native", and provide the pre-designed themes to the *NavigationContainer* that we constructed in section 3.2.2.

```
// other import statements
import { NavigationContainer, DefaultTheme, DarkTheme } from "@react-navigati

export default function App() {
    // ...
    return (
    <ThemeContext.Provider value={{ isDarkTheme, toggleTheme }}>
        <NavigationContainer theme={!isDarkTheme ? DefaultTheme : DarkTheme}>
            <Router />
        </NavigationContainer>
    </ThemeContext.Provider>
    )
}

```

3.2.4. Let's try to look at our dark theme! Modify the default value of *isDarkTheme* to *true* and see what happens.

```
const [isDarkTheme, setIsDarkTheme] = useState(true);
```

Oh! The text "This is COMP3330-Tutorial on Theme Construction" seems to have disappeared. How can we fix it?



3.3. The text is actually still there, but it is not visible because it is not theme responsive (Yes, you may notice that the header "Home" becomes white in color because it is associated with the screen-level container). The text color is not changed, i.e., it's black. Therefore, we have to make use of *react-native-paper* to implement component-level themeing.

3.3.1. First, install the required package with the following command in the command line interface.

```
npm install react-native-paper
```

3.3.2. Next, we have to wrap the components with the container provided by the library.

```
// other import statements
import { PaperProvider, DefaultTheme as PaperDefaultTheme, MD3DarkTheme as
PaperDarkTheme} from "react-native-paper";
```

3.3.3. Afterwards wrap the *<Router/>* component with the *PaperProvider* imported.

```
// ...
    return (
        <ThemeContext.Provider value={{ isDarkTheme, toggleTheme }}>
            <NavigationContainer theme={!isDarkTheme ? DefaultTheme : DarkThe
                <PaperProvider theme={!isDarkTheme ? PaperDefaultTheme : Pape
                    <Router />
                </PaperProvider>
            </NavigationContainer>
        </ThemeContext.Provider>
    )
}
```

3.3.4. Last but not least, we import *Text* from "react-native-paper" instead of "react-native".

```
// other import statements
import { StyleSheet, ~~Text~~, View } from "react-native";
import {
  PaperProvider,
  DefaultTheme as PaperDefaultTheme,
  MD3DarkTheme as PaperDarkTheme,
  Text
} from "react-native-paper";
```

All set! You should see your words on the home screen in the dark theme! Therefore, we should keep in mind that we should always import components from *react-native-paper* if available to facilitate theme management.

4. Toggling Themes

There is a common use-case where the app user will like to change the theme layout of the app. In this section, we will be demonstrating how to implement theme toggling. With the *App.js* created in section 3, we will be modifying the default value of *isDarkTheme* as *true*. Let's see what will happen.

4.1. In the *HomeScreen* component, we can add a button to toggle the theme.

```
// other import statements
import { ..., Button } from "react-native-paper"

export default function App() {
    ...
    const HomeScreen = () => (
        <View style={styles.root}>
            <Text>This is COMP3330-Tutorial on Theme Construction</Text>
                <Button onPress={toggleTheme}>
                    <Text>Change Theme</Text>
                </Button>
            <StatusBar style="auto" />
        </View>
    )
    return (...)
}
```

Yay! You should be able to toggle the theme easily!

5. Theme Customization

From toggling theme in section 4, you may notice that the toggle button actually is not user-friendly, as it does not have color difference with the background. The reason is that the primary colors and screen colors in the default theme are configured to be the same. In some other use cases, you may also want to include your own theme colors! In this section, we will be covering how to customize themes.

5.1. Let's open *App.js* again. Instead of customizing every single color, we will use primary color as a demonstration of theme customization in this section. You can always refer to the official documentation of *react-native-paper* and/or *react-navigation* for more guidelines:

https://reactnavigation.org/

React Navigation | React Navigation

Routing and navigation for your React Native apps

https://reactnativepaper.com/

React Native Paper

React Native Paper is a high-quality, standard-compliant Material Design library that has you covered in all major use-cases.

5.2. In *App.js*, we define our own light theme and dark theme as follows. We will treat theme as a constant object which can be placed outside our *App.*

```
// import statements
const MyLightTheme = {
  ...PaperDefaultTheme,
  primary: "#6767fe",
}

const MyDarkTheme = {
  ...PaperDarkTheme,
  primary: "#6767fe",
}
```

5.3. Next, we replace the *PaperTheme* and *PaperDarkTheme* with *MyLightTheme* and *MyDarkTheme* respectively.

```
export function App() {
    ...
    return (
        <ThemeContext.Provider value={{ isDarkTheme, toggleTheme }}>
            <NavigationContainer theme={!isDarkTheme ? DefaultTheme : DarkTheme}
                <PaperProvider theme={!isDarkTheme ? MyLightTheme : MyDarkTheme}
                    <Router />
                </PaperProvider>
            </NavigationContainer>
        </ThemeContext.Provider>
    )
}
```

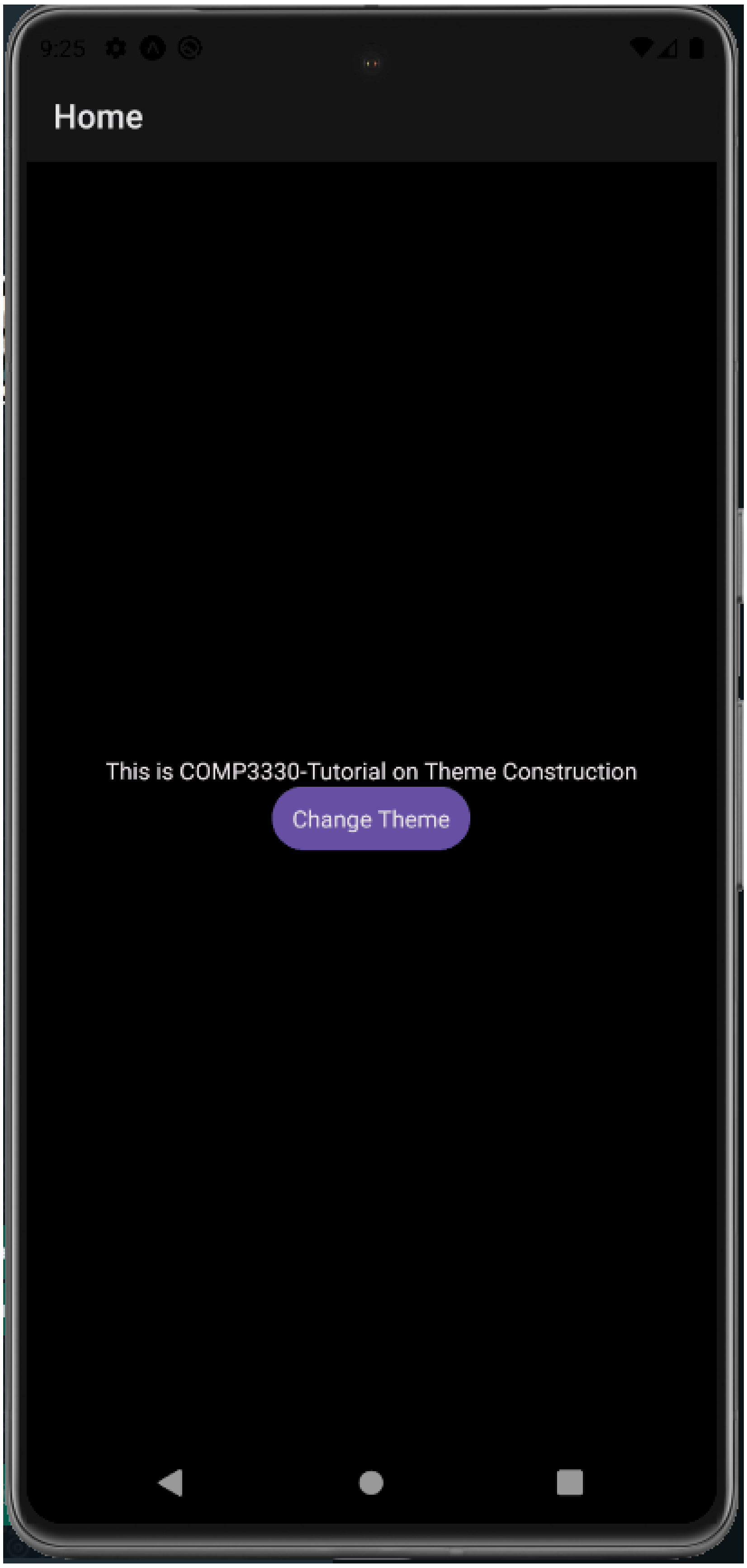
5.4. Finally, we can use a hook function provided by *react-native-paper* to get the theme *colors*. It can be useful when you have many screens.

```
// other import statements
import { ..., useTheme } from "react-native-paper";

export default function App() {
    const { colors } = useTheme();
    ...
}
```

5.5 The *colors* object consists of different colors you defined in the theme object. Now, we try to modify the style of the button to have a color with our primary color.

```
// import statements

export default App() {
    ...
    const HomeScreen = () => (
        <View style={styles.root}>
            <Text>This is COMP3330-Tutorial on Theme Construction</Text>
                <Button onPress={toggleTheme} style={{ backgroundColor: colors.p
                    <Text>Change Theme</Text>
                </Button>
            <StatusBar style="auto" />
        </View>
    )
    ...
}
```

Looks good! Your app could use the theme color right now! With this tutorial, I hope you can learn how to manage the themes easily with the mentioned libraries.



Home

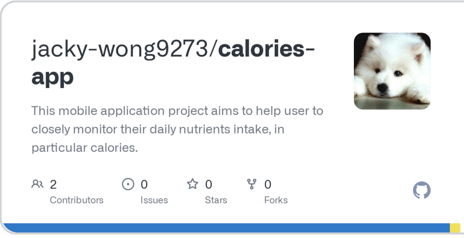
This is COMP3330-Tutorial on Theme Construction

Appendix

Your *App.js* should look like this at the end:

```
import { StatusBar } from "expo-status-bar";
import { StyleSheet, View } from "react-native";
import { createContext, useState } from "react";
import { createStackNavigator } from "@react-navigation/stack";
import {
  NavigationContainer,
  DefaultTheme,
  DarkTheme,
} from "@react-navigation/native";
import "react-native-gesture-handler";
import {
  PaperProvider,
  DefaultTheme as PaperDefaultTheme,
  MD3DarkTheme as PaperDarkTheme,
  Text,
  Button,
  useTheme,
} from "react-native-paper";

const MyLightTheme = {
  ...PaperDefaultTheme,
  primary: "#ff0000",
};

const MyDarkTheme = {
  ...PaperDarkTheme,
  primary: "#6767fe",
};

export default function App() {
  const { colors } = useTheme();
  const ThemeContext = createContext({
    isDarkTheme: false,
    toggleTheme: () => {},
    theme: {},
  });
  const [isDarkTheme, setIsDarkTheme] = useState(true);
  const toggleTheme = () => setIsDarkTheme(!isDarkTheme);
  const Stack = createStackNavigator();
  const HomeScreen = () => (
    <View style={styles.root}>
      <Text>This is COMP3330-Tutorial on Theme Construction</Text>
      <Button onPress={toggleTheme} style={{ backgroundColor: colors.primary }}>
        <Text>Change Theme</Text>
      </Button>
      <StatusBar style="auto" />
    </View>
  );
  const Router = () => (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={HomeScreen} />
    </Stack.Navigator>
  );

  return (
    <ThemeContext.Provider value={{ isDarkTheme, toggleTheme }}>
      <NavigationContainer theme={!isDarkTheme ? DefaultTheme : DarkTheme}>
        <PaperProvider theme={!isDarkTheme ? MyLightTheme : MyDarkTheme}>
          <Router />
        </PaperProvider>
      </NavigationContainer>
    </ThemeContext.Provider>
  );
}

const styles = StyleSheet.create({
  root: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
  },
});

```

Reproducible source codes:


jacky-wong9273/**calories-app**

This mobile application project aims to help user to closely monitor their daily nutrients intake, in particular calories.

2 Contributors 0 Issues 0 Stars 0 Forks

https://github.com/jacky-wong9273/calories-app

GitHub - jacky-wong9273/calories-app: This mobile application project aims to help user to closely m

This mobile application project aims to help user to closely monitor their daily…