Logicbit Solutions

DEKAR PROJECT 2023

Developed by :

Bibek Dhimal & Nikesh Khadka

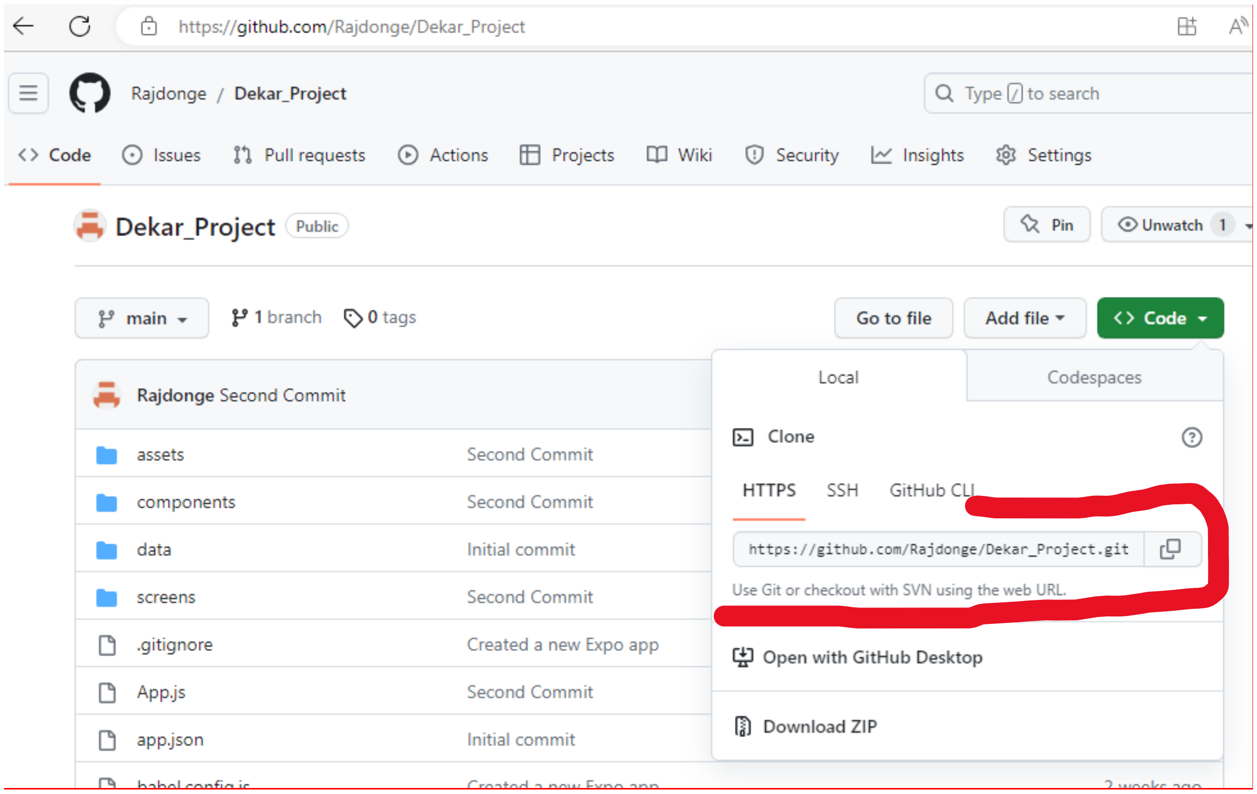
Contents

A. Introduction to React Native

React Native is an open-source JavaScript framework that lets developers build native apps for iOS and Android using a single code base. It is based on React JS, a JavaScript library for building user interfaces. It is used by some popular apps like Facebook, Instagram, Pinterest, and Skype.

B. Steps to clone the Project

1. Install node, and JDK
2. Install expo client from Play Store (android) or App Store (IOS)
3. Go to the terminal or command prompt and write command → npm install -g expo-cli
4. Copy HTTPS URL from GitHub repository



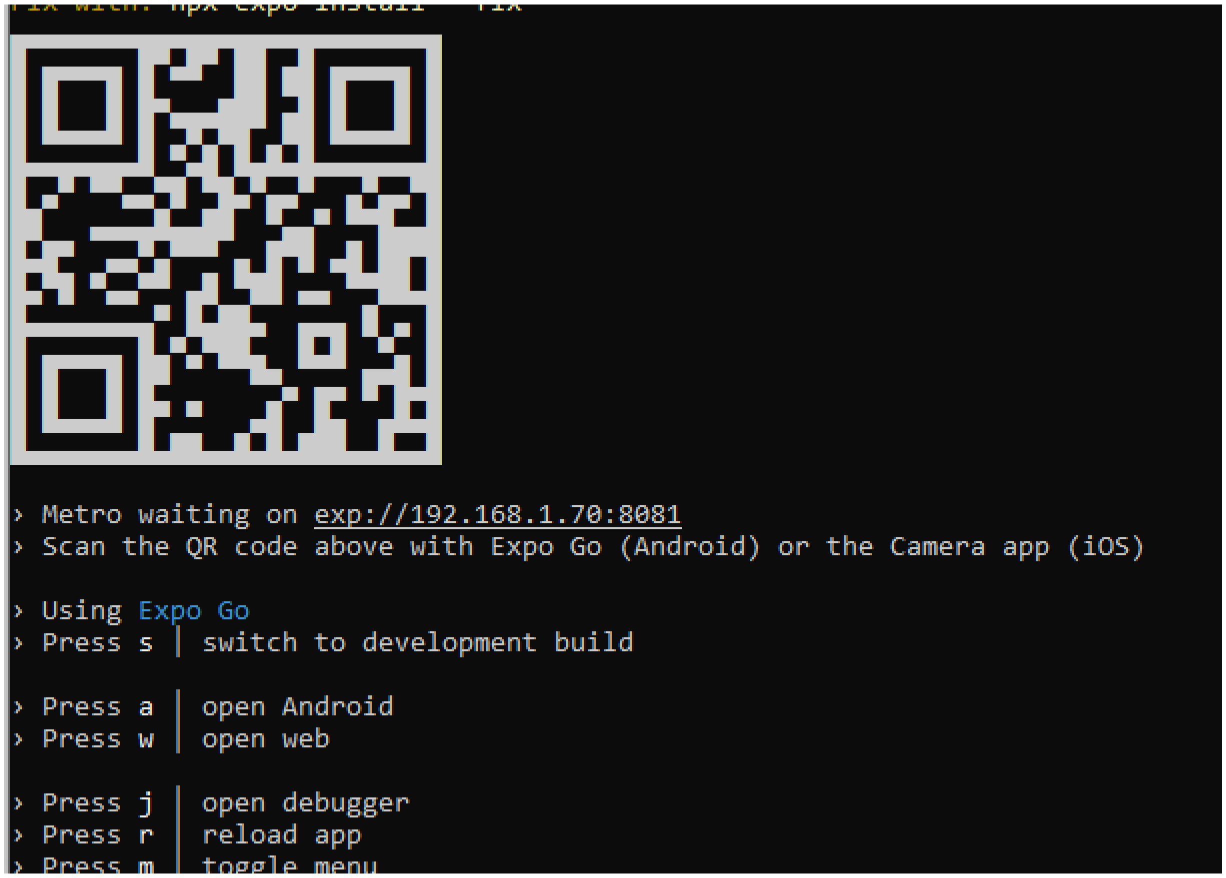
assets	Second Commit
components	Second Commit
data	Initial commit
screens	Second Commit
.gitignore	Created a new Expo app
App.js	Second Commit
app.json	Initial commit

```
https://github.com/Rajdonge/Dekar_Project.git
```

Use Git or checkout with SVN using the web URL.

5. Open Visual Studio Code (VS Code) → code editor
6. Open terminal (CTRL + `): write command → git clone <URL>
7. Open the file in VS code and go to the terminal and write command → npm install (to install all the dependencies used in a project)
8. Write the command → npm start

9. open the expo app and scan QR code from the terminal or also you can enter URL manually



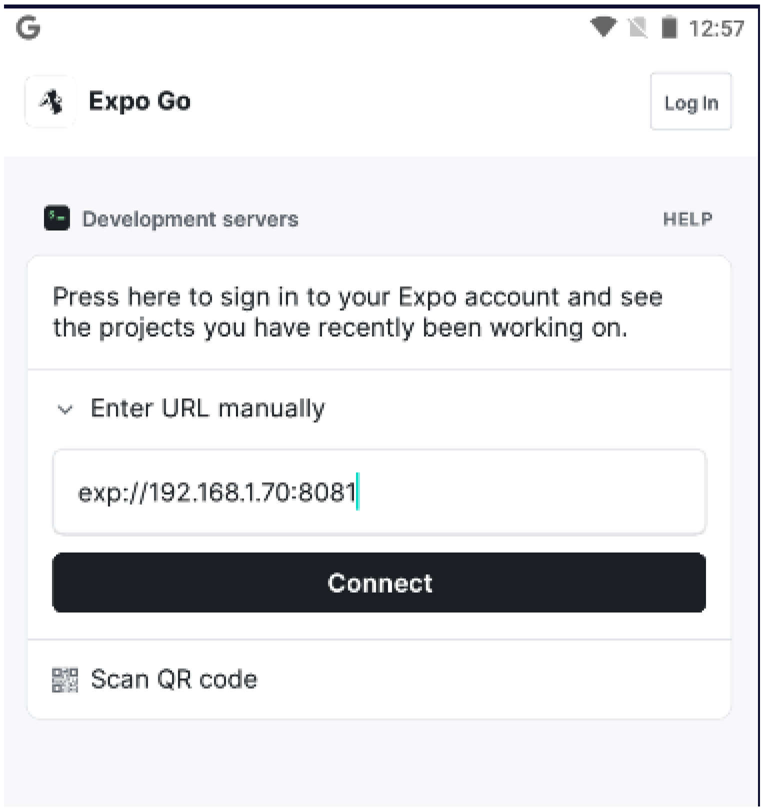
```
Fix with: npx expo install --fix

› Metro waiting on exp://192.168.1.70:8081
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Using Expo Go
› Press s │ switch to development build

› Press a │ open Android
› Press w │ open web

› Press j │ open debugger
› Press r │ reload app
› Press m │ toggle menu
```



12:57

Expo Go

Log In

Development servers

HELP

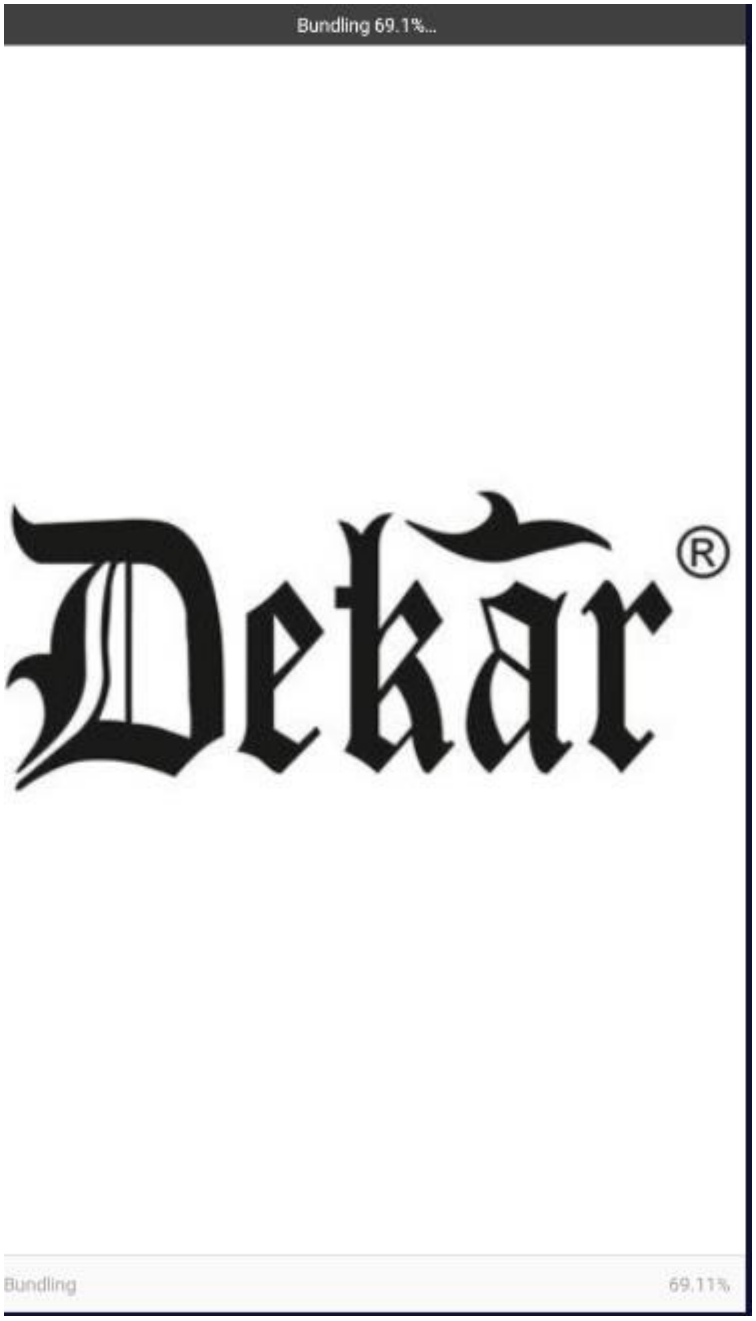
Press here to sign in to your Expo account and see the projects you have recently been working on.

Enter URL manually

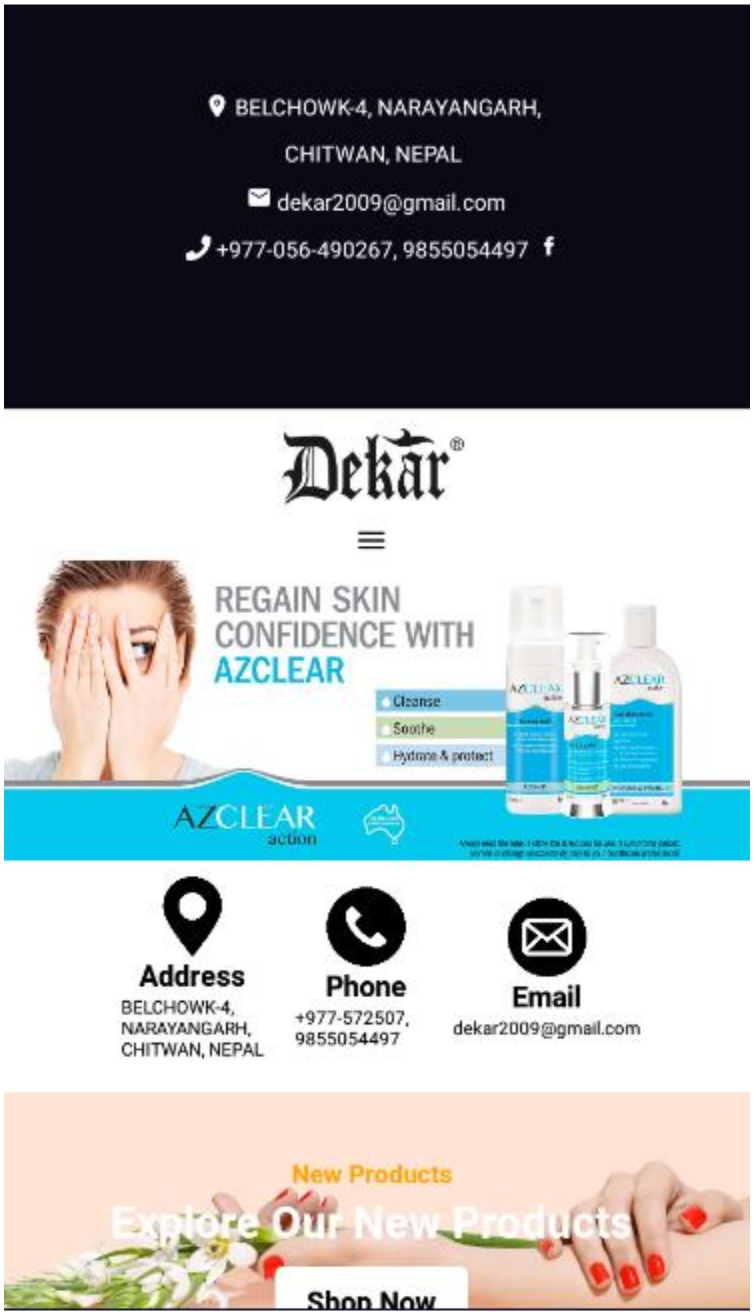
exp://192.168.1.70:8081

Connect

Scan QR code



10. Finally App started



BELCHOWK-4, NARAYANGARH, CHITWAN, NEPAL

dekar2009@gmail.com

+977-056-490267, 9855054497

Dekar®

REGAIN SKIN CONFIDENCE WITH AZCLEAR

Cleanse

Soothe

Hydrate & protect

AZCLEAR action

Address

BELCHOWK-4, NARAYANGARH, CHITWAN, NEPAL

Phone

+977-572507, 9855054497

Email

dekar2009@gmail.com

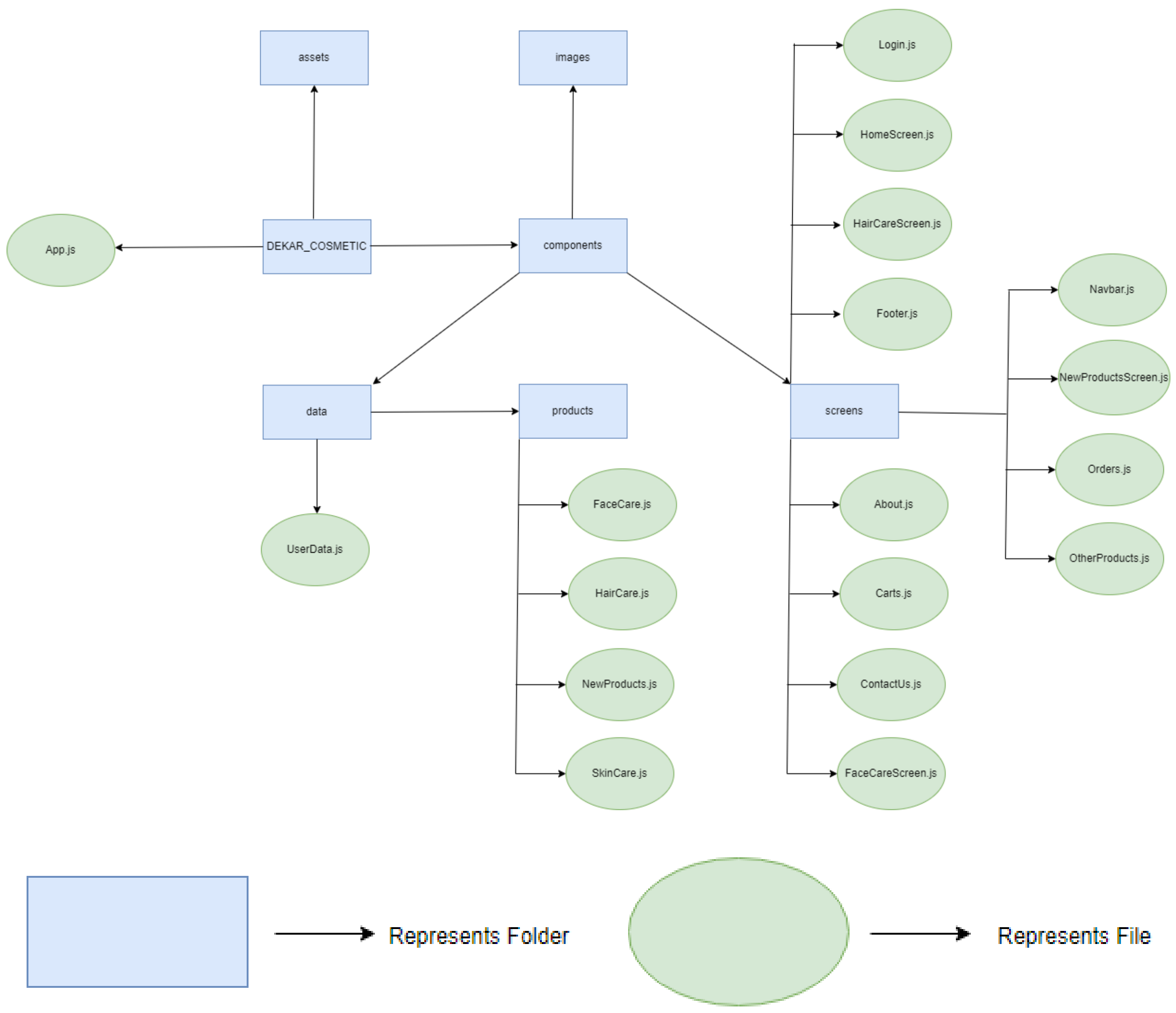
New Products

Explore Our New Products

Shop Now

C. Introduction to the Project

The "DEKAR" project is a React Native application designed to showcase fundamental concepts of mobile app development, including user authentication and product listing. The app is aimed at providing users with a seamless experience while browsing and exploring a selection of products. It's folder structure:



Here's a brief overview of the main components and their functionalities:

1. App.js

This is the entry point of the application. It sets up the navigation stack using the "@react-navigation/native" library and defines different screens (components) that can be navigated to. It also manages the user's login status and shopping cart.

2. Navbar.js
This component represents the top navigation bar. It displays contact information, the company logo, and a menu icon for navigation. The menu icon triggers a modal menu that provides links to different sections of the app.

3. HomeScreen.js
This component represents the main landing page of the app. It displays a slideshow of images, contact details, and various product categories (FaceCare, SkinCare, HairCare, etc.) that users can navigate to. It also renders the "New Products" section and the footer.

4. FaceCareScreen.js
This component displays the list of face care products available for purchase. It maps over the `face_products` data and displays each product's name, image, price, and quantity. Users can add items to their shopping cart using the "ADD TO CART" button.

5. Carts.js
This component represents the user's shopping cart. It displays the products that the user has added to their cart along with quantities and prices. Users can adjust quantities and select items for checkout. It calculates and displays the total price of selected items.

6. Orders.js
This component displays the ordered items after the user has proceeded with the checkout process. It receives the selected items and cart data as props from the `Carts.js` component. It displays the ordered items' names, prices, and quantities, as well as the total order price.

7. Login.js
This component provides a simple login interface. It allows users to enter their email and password, then checks against a predefined list of users to determine whether the login is successful.

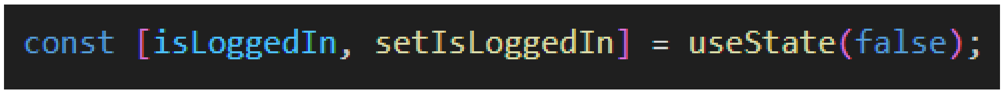
8. Footer.js
This component displays the footer at the bottom of various screens. It contains links to the "About Us" and "Contact Us" sections.

These components work together to create an e-commerce app with navigation, product browsing, shopping cart management, and user authentication features. The code structure seems well-organized, and the components have clear responsibilities. Proper documentation and comments would help improve code readability and maintainability.

D. Functions and methods used in this project include

1. useState

This is a React hook used to manage state within functional components. It allows you to declare a state variable and a function to update it. For example:

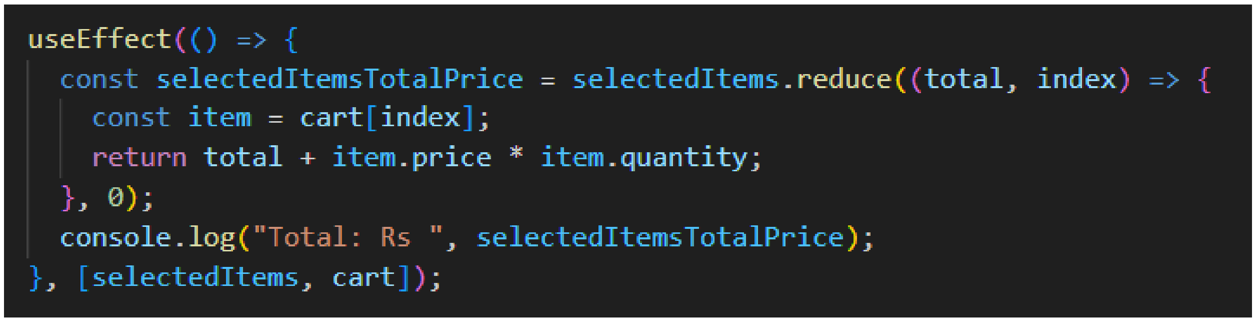


```
const [isLoggedIn, setIsLoggedIn] = useState(false);
```

Here, ‘isLoggedIn’ is a state variable representing the user’s login status, and ‘setIsLoggedIn’ is the function to update that status.

2. useEffect

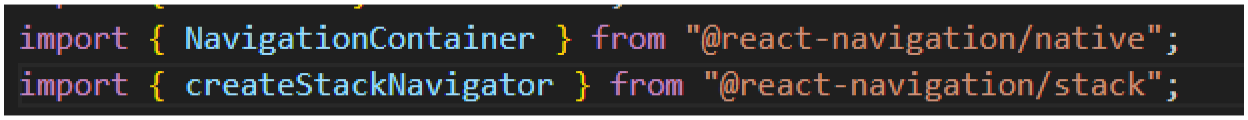
This is another React hook used for handling side effects in functional components. It’s commonly used to perform actions like data fetching, subscriptions, DOM manipulations after the component has rendered. For example:



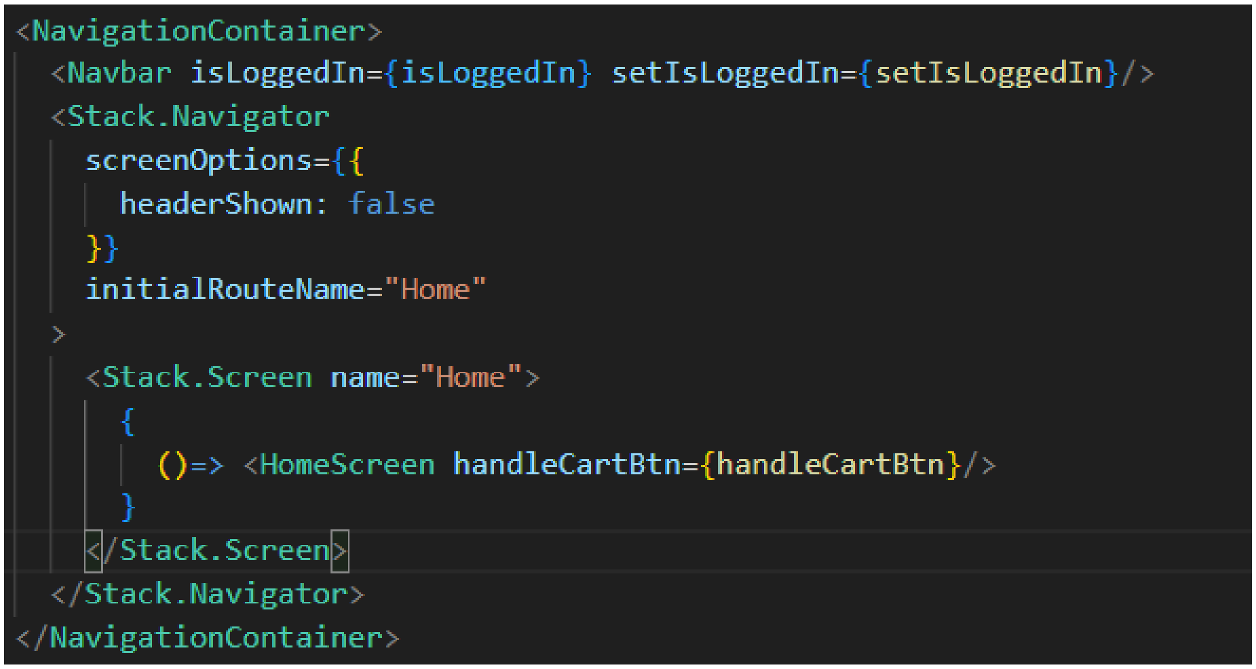
```
useEffect(() => {
  const selectedItemsTotalPrice = selectedItems.reduce((total, index) => {
    const item = cart[index];
    return total + item.price * item.quantity;
  }, 0);
  console.log("Total: Rs ", selectedItemsTotalPrice);
}, [selectedItems, cart]);
```

3. NavigationContainer, createStackNavigator

These are components from the ‘@react-navigation/native’ library that help with the navigation within the app. ‘NavigationContainer’ is the root component that manages the navigation tree, and ‘createStackNavigator’ creates a stack of screens for navigation.

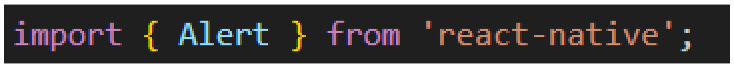


```
import { NavigationContainer } from "@react-navigation/native";
import { createStackNavigator } from "@react-navigation/stack";
```

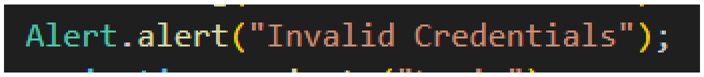


```
<NavigationContainer>
  <Navbar isLoggedIn={isLoggedIn} setIsLoggedIn={setIsLoggedIn}/>
  <Stack.Navigator
    screenOptions={{
      headerShown: false
    }}
    initialRouteName="Home"
  >
    <Stack.Screen name="Home">
      {
        ()=> <HomeScreen handleCartBtn={handleCartBtn}/>
      }
    </Stack.Screen>
  </Stack.Navigator>
</NavigationContainer>
```

4. Alert.alert
A method provided by 'react-native' that shows an alert dialog with a message. It's used to display notification to users. For example:

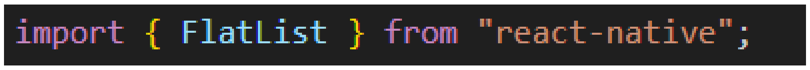


```
import { Alert } from 'react-native';
```

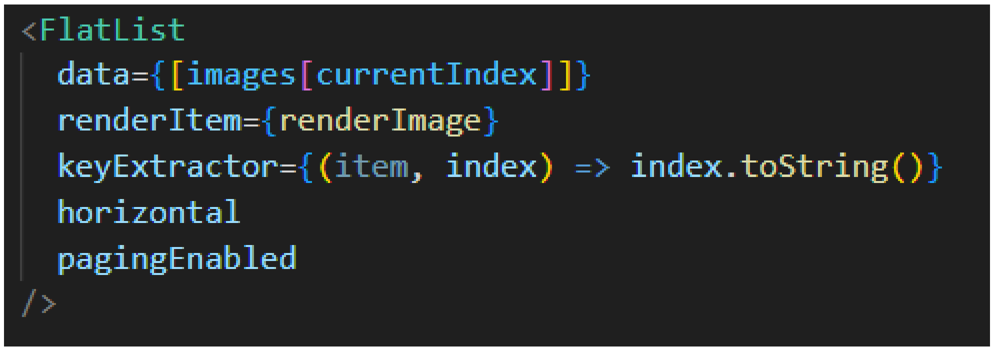


```
Alert.alert("Invalid Credentials");
```

5. FlatList
A component used to render a list of items with improved performance over 'ScrollView'. It's particularly useful for rendering large lists efficiently. For example:



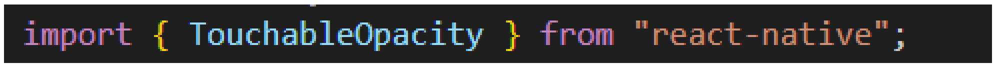
```
import { FlatList } from "react-native";
```



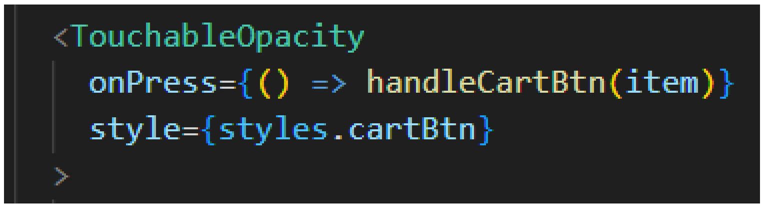
```
<FlatList
  data={[images[currentIndex]]}
  renderItem={renderImage}
  keyExtractor={(item, index) => index.toString()}
  horizontal
  pagingEnabled
/>
```

6. TouchableOpacity

A touchable component that provides feedback when pressed. It's often used for buttons and interactive elements. For example:



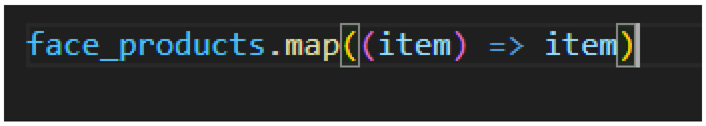
```
import { TouchableOpacity } from "react-native";
```



```
  <TouchableOpacity
    onPress={() => handleCartBtn(item)}
    style={styles.cartBtn}
  >
```

7. map

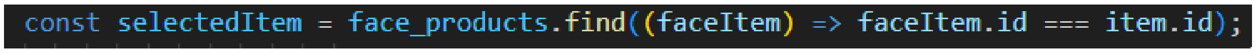
A method used with arrays to iterate over each item and return a new array based on the transformation applied to each item. It's often used to render lists of items. For example:



```
face_products.map((item) => item)
```

8. find

A method used with arrays to find the first element that matches a specific condition. It returns the element found or 'undefined' if not found. For example:



```
const selectedItem = face_products.find((faceItem) => faceItem.id === item.id);
```

E. Other functions:

1. handleLoginBtn

This function is called when the "Login" button is pressed. It retrieves the values entered in the username and password fields, and then compares them with predefined values (hardcoded credentials) to determine whether the login is successful or not. If successful, it sets the “isLoggedIn” state to “true” using the “setIsLoggedIn” function. If not successful, it displays an alert using “Alert.alert” to notify the user about invalid credentials.

2. handleLogoutBtn

This function is called when the "Logout" button is pressed. It sets the “isLoggedIn” state to `false`, effectively logging the user out.

3. renderProducts

This function takes an array of product objects and uses the “map” function to iterate over each product. For each product, it renders a “ProductComponent” (which is assumed to be a separate component responsible for rendering product details) with the “product” data passed as a prop.

4. handleProductPress

This function is likely associated with the behavior of selecting a product from the list. It takes a product ID as a parameter and is intended to be used as an event handler when a product is pressed. Inside the function, it uses the “find” function to locate the product object in the “products” array based on the provided product ID. The specific behavior beyond this point isn't provided in the code snippet, but it could involve navigating to a product details screen or displaying additional information about the selected product.

5. useEffect for products

This “useEffect” hook is likely used to simulate a data fetching scenario. It runs when the component mounts and updates the “products” state with an array of predefined product objects. This is typically where an API call to fetch actual product data would occur in a real-world scenario.

6. “useEffect” for “isLoggedIn”

This “useEffect” hook watches the “isLoggedIn” state. If “isLoggedIn” becomes “true”, it likely indicates that the user has successfully logged in. Depending on the application's design, this effect might trigger a navigation action to redirect the user to a dashboard or main screen.

7. “render” method
This is the main render method of the component. It returns JSX that defines the structure and layout of the component. It conditionally renders different components based on the “isLoggedIn” state. If the user is logged in, it renders a list of products using the “renderProducts” function. If the user is not logged in, it renders a login form with input fields for username and password, as well as a "Login" button.

F. Push code to GitHub

1. Create a GitHub Account
If you don't have one already, go to https://github.com and create an account.

2. Create a New Repository
- Click the "+" icon in the top-right corner and select "New repository".
- Give your repository a name, description, and choose other settings as needed.
- Click "Create repository".

3. Clone the Repository
- Once the repository is created, you can clone it to your local machine. To do this, copy the repository URL from the repository's main page, go to the terminal, and enter the command → git clone <repository-URL>

4. Add Your Code
- Navigate to the cloned repository using your terminal or command prompt.
- Add your code files to this directory.

5. Stage and Commit Changes
- Use the following commands to stage and commit your changes using following commands →

```
git add .   ( This stages all the changes in the directory)

git commit -m "Your commit message here"
```

6. Push to GitHub

- After committing your changes, you can push them to GitHub using:

```
git push origin main   #(Assuming your main branch is named 'main')
```

If you're pushing to a different branch, replace "main" with the appropriate branch name.

7. Provide GitHub Credentials

- If you're pushing for the first time from your machine, GitHub might prompt you to provide your GitHub credentials.

8. View Changes on GitHub

- Go to your repository on GitHub and you should see the changes you've pushed.

Note: Remember to replace `<repository-URL>` with the actual URL of your GitHub repository.

Please note that these steps assume you have Git installed on your local machine. If you don't have Git installed, you can download and install it from https://git-scm.com/downloads.

Also, be cautious while pushing code to public repositories, especially if they contain sensitive information. It's a good practice to avoid including sensitive data like passwords or API keys in your public code. Use environment variables or configuration files for such sensitive information.