WhiteHat Jr
Live online Coding for Kids

Topic	LOGOUT AND PROFILE SCREEN	
Class Description	**In this class, the student would be adding logout functionality and the Profile Screen of the App.**	
Class	**C86**	
Class time	**45 mins**	
Goal	● Add logout functionality ● Complete the profile screen of the App.	
Resources Required	● Teacher Resources ○ Visual Studio Code Editor ○ laptop with internet connectivity ○ earphones with mic ○ notebook and pen ● Student Resources ○ Visual Studio Code Editor ○ laptop with internet connectivity ○ earphones with mic ○ notebook and pen	
Class structure	**Warm-Up** **Teacher-led Activity** **Student-led Activity** **Wrap-Up**	**5 mins** **15 mins** **20 mins** **5 mins**

WARM-UP SESSION - 5 mins

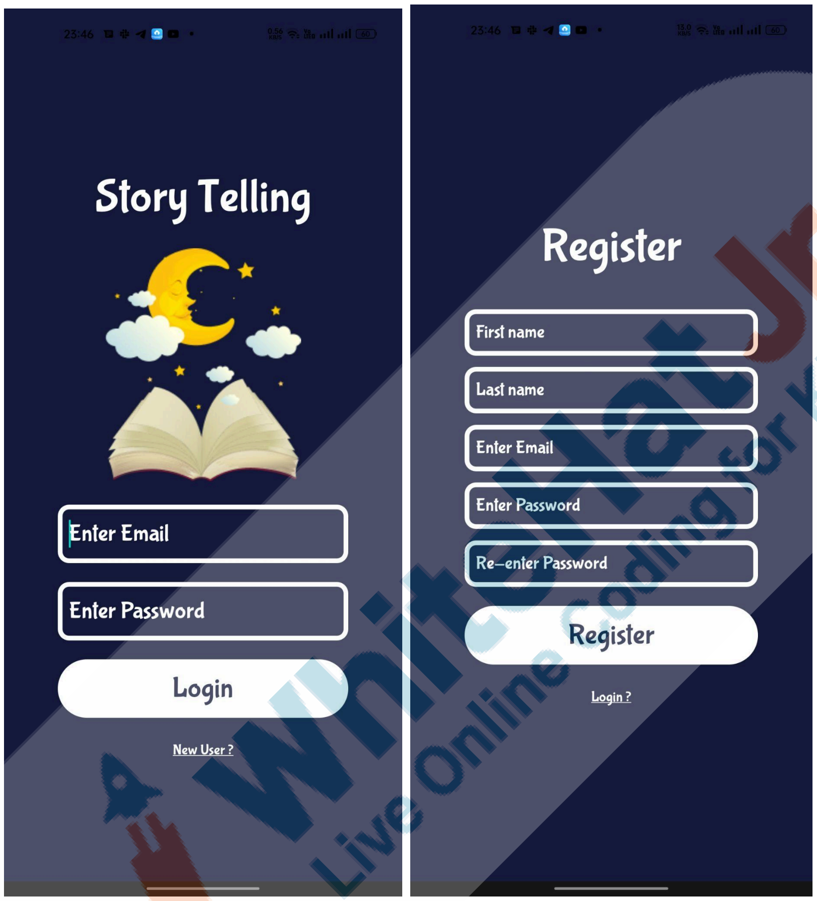
CONTEXT

- **Discuss the importance of UI in an App.**

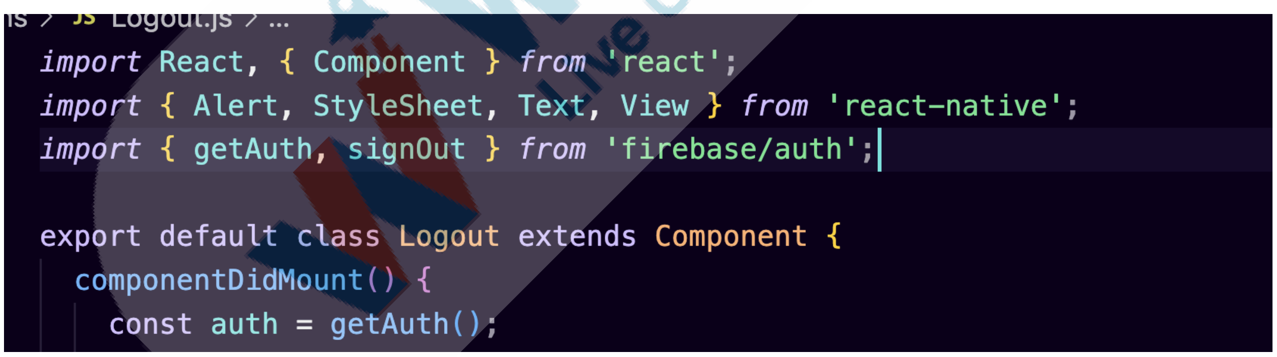
Teacher starts slideshow from slides 1 to 8
Refer to speaker notes and follow the instructions on each slide.

Activity details	
Hey <student name>. How are you? It's great to see you! Are you excited to learn something new today? ***Run the presentation from slide 1 to slide 3.*** **The following are the warm-up session deliverables:** ● Connecting students to the previous class.	
Continue the warm-up session	
Activity details	**Solution/Guidelines**
Run the presentation from slide 4 to slide 8 to set the problem statement. **The following are the warm-up session deliverables:** ● Explain about UI designing for Login Screen.	Narrate the slides by using hand gestures and voice modulation methods to bring in more interest in students.
Teacher ends slideshow	
TEACHER-LED ACTIVITY - 15 mins	
Teacher Initiates Screen Share	
CHALLENGE ● **Completing the UI for the Login Screen.**	

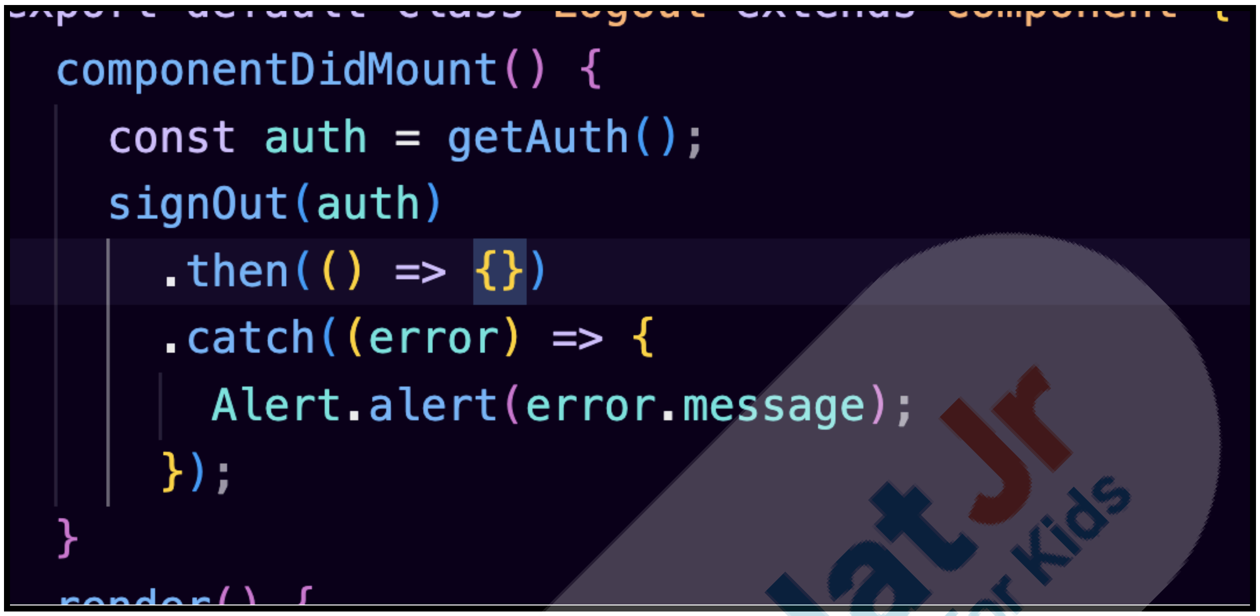
Step 2: Teacher-led Activity (15 min)	In the last class, we created a plain login screen for storytelling. Also we created a functionality to register a new user in the app *<For reference you can access the previous class code in Teacher Activity 1>*	

	Our Login Screen look something like this -	
		
	With this, our login and registration screen was complete.	
	What do you do when you are done working on any app which requires you to login? Logout from the app.	**ESR**: Close it / Logout / Varied.

	Today we will add the logout functionality and create a profile screen for the user. Logging out from an app that is not in use is the safest way to protect the data. It is a good security measure for the user, the user is not always connected to the DB thus not utilizing a lot of DB resources and so on. Let's quickly create a **Logout.js** in our **screens** folder for that. Here, in the **componentDidMount()** function, we will use **signOut((auth)=>{})**, which will sign out the user from our firebase app, therefore the user will not be able to access the App and will directly be logged out and navigated to the Login Screen.	



```
import React, { Component } from 'react';
import { Alert, StyleSheet, Text, View } from 'react-native';
import { getAuth, signOut } from 'firebase/auth';

export default class Logout extends Component {
  componentDidMount() {
    const auth = getAuth();
```

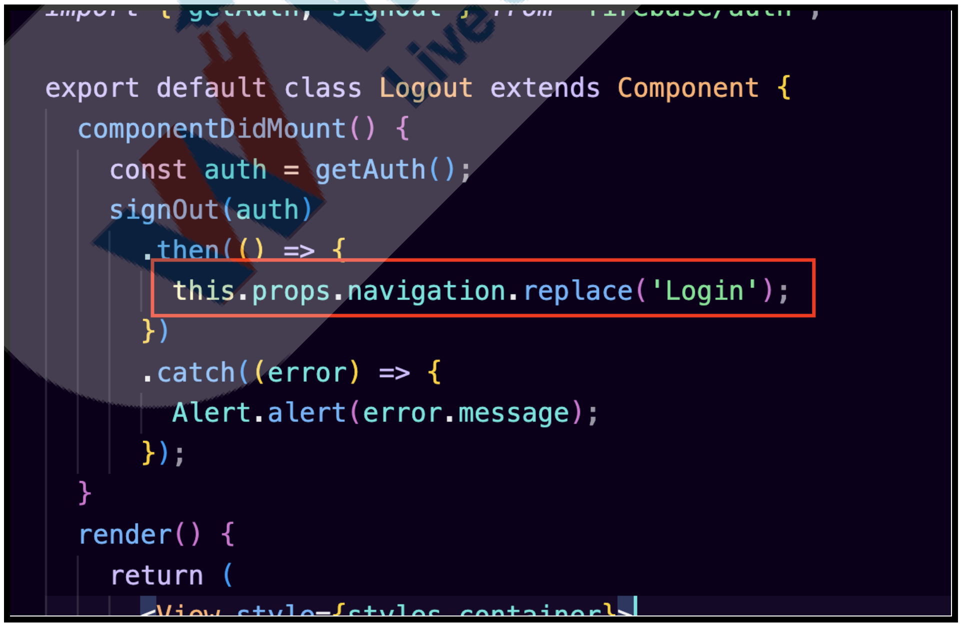


```
export default class Logout extends Component {
  componentDidMount() {
    const auth = getAuth();
    signOut(auth)
      .then(() => {})
      .catch((error) => {
        Alert.alert(error.message);
      });
  }
  render() {
```

In order to access this logout text we will add **Logout.js** file inside our Drawer Navigation, so let's do that -

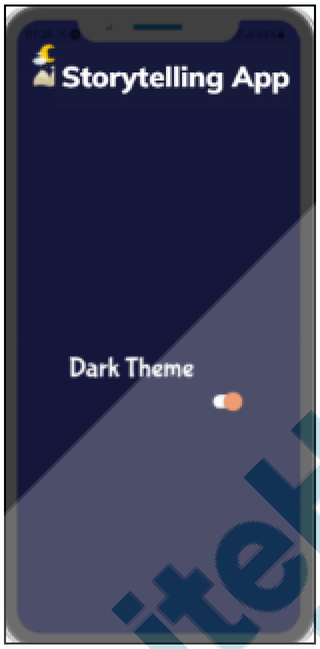
```
import React from "react";
import { createDrawerNavigator } from "@react-navigation/drawer";
import StackNavigator from "./StackNavigator";
import Profile from "../screens/Profile";
import Logout from "../screens/Logout";

const Drawer = createDrawerNavigator();

const DrawerNavigator = () => {
  return (
    <Drawer.Navigator>
      <Drawer.Screen name="Home" component={StackNavigator} />
      <Drawer.Screen name="Profile" component={Profile} />
      <Drawer.Screen name="Logout" component={Logout} />
    </Drawer.Navigator>
  );
};

export default DrawerNavigator;
```

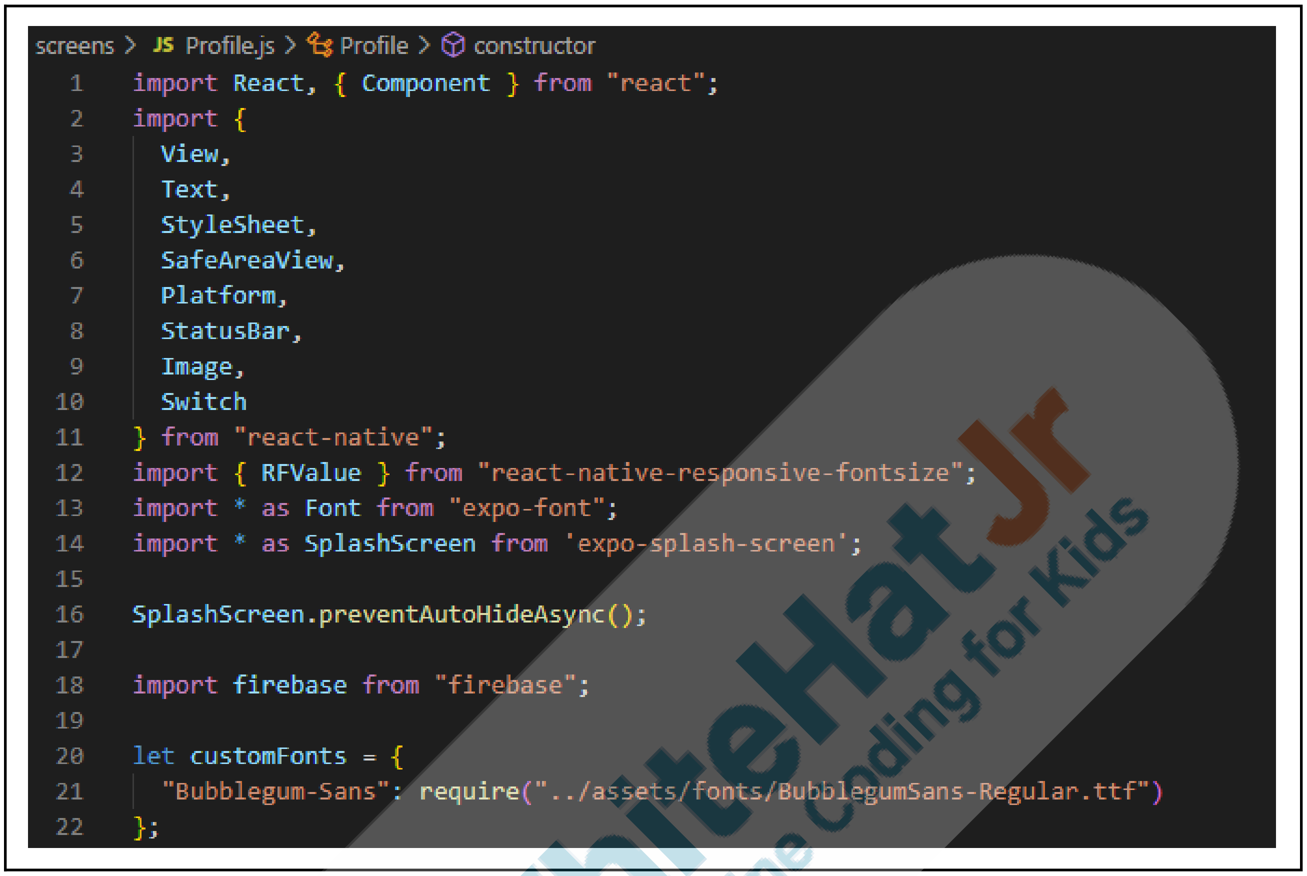
Here, we have imported the **Logout** from **screens** and added a new **<Drawer.Screen>** component to include **Logout** text from the **Logout.js** file.		
	Teacher and student test the app. Now we have completed adding logout in our side drawer.	
	What is happening on logging out? To avoid this we can replace the screen with some other screen, so which screen should we open after logging out? Correct! Let redirect to the login screen after logging out.	**ESR:** Logout screen comes when we try to logout but we can still press the back button to open the profile. **ESR:** Login Screen.



```
import { getAuth, signOut } from 'firebase/auth';

export default class Logout extends Component {
  componentDidMount() {
    const auth = getAuth();
    signOut(auth)
      .then(() => {
        this.props.navigation.replace('Login');
      })
      .catch((error) => {
        Alert.alert(error.message);
      });
  }
  render() {
    return (
      <View style={styles.container}>
```

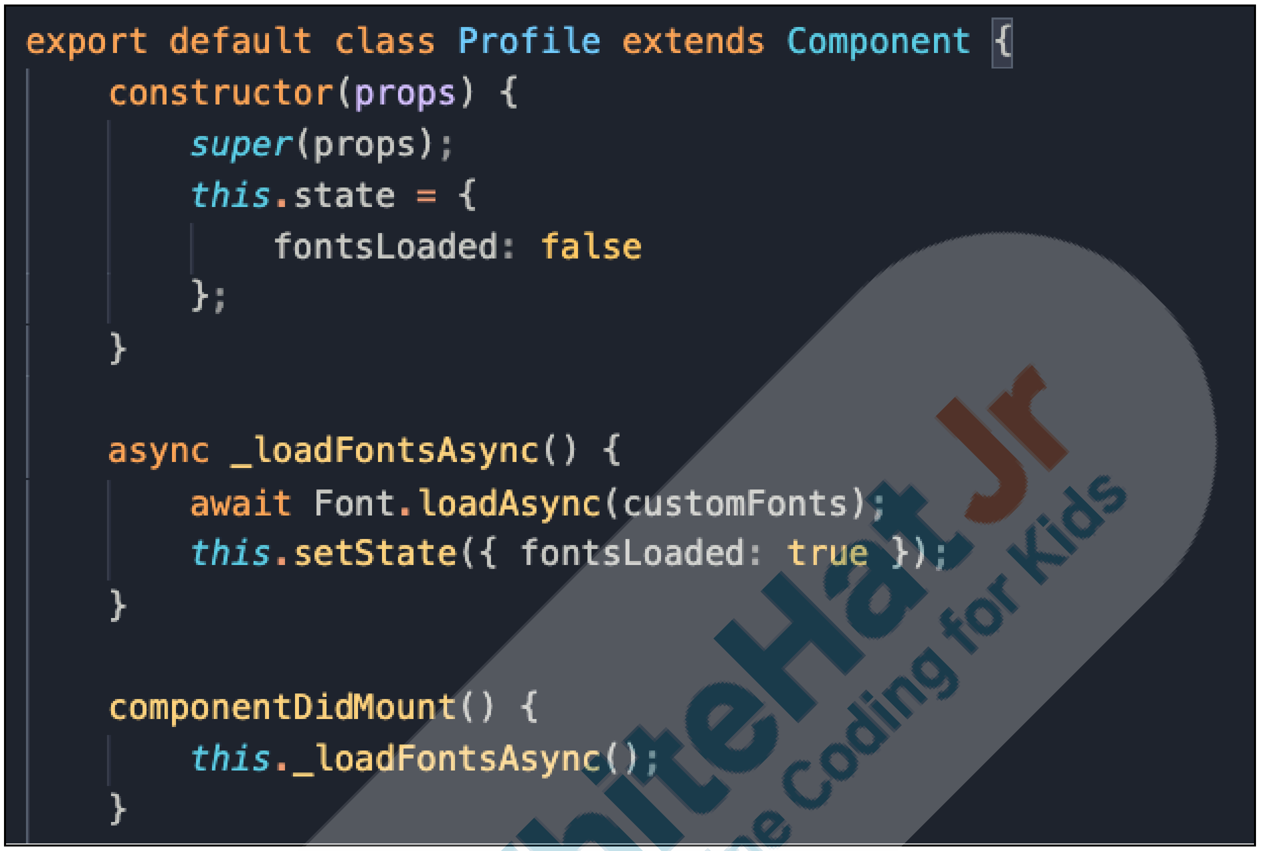
	Teacher and student test the app. Now we have completed the logout functionality. Next, you will be working on the profile screen.	
Teacher Stops Screen Share		
	Now it's your turn. Please share your screen with me.	
STUDENT-LED ACTIVITY - 20 mins		
● **Ask the student to press the ESC key to come back to the panel.** ● **Guide the student to start screen share.** ● **Teacher gets into fullscreen.**		
Teacher starts slideshow from slide 9 to slide 11		
ACTIVITY ● **Creating and completing the UI by adding themes to the profile screen.**		
Step 3: Student-Led Activity (20 mins)	Please refer to *Student Activity 1* to clone the boilerplate code. Don't forget to add the **config.js** with your credentials in it.	*The student refers to Student Activity 1, clones the repo, and adds config.js.*
	The boilerplate code contains the code we just did for the logout screen. Now you will be creating the profile screen.	

	That screen would look something like this -	
		
	Now let's code for the same! This time around, we will be coding differently. First, let's add the fonts to this screen. Note, this is the last time we are adding fonts to any screen.	*The student adds the fonts.*

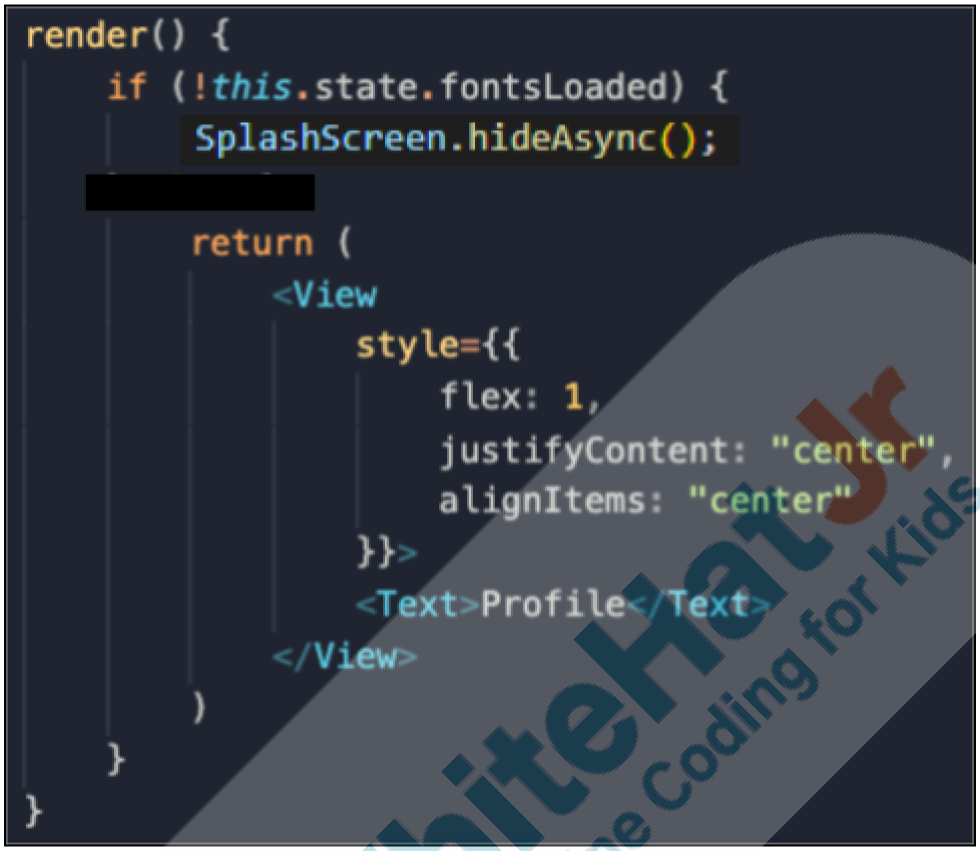


screens > Profile.js > Profile > constructor

```
import React, { Component } from "react";
import {
  View,
  Text,
  StyleSheet,
  SafeAreaView,
  Platform,
  StatusBar,
  Image,
  Switch
} from "react-native";
import { RFValue } from "react-native-responsive-fontsize";
import * as Font from "expo-font";
import * as SplashScreen from 'expo-splash-screen';

SplashScreen.preventAutoHideAsync();

import firebase from "firebase";

let customFonts = {
  "Bubblegum-Sans": require("../assets/fonts/BubblegumSans-Regular.ttf")
};
```

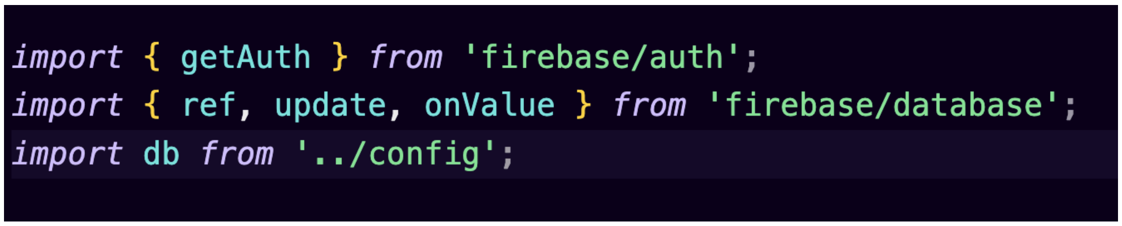


```
export default class Profile extends Component {
    constructor(props) {
        super(props);
        this.state = {
            fontsLoaded: false
        };
    }

    async _loadFontsAsync() {
        await Font.loadAsync(customFonts);
        this.setState({ fontsLoaded: true });
    }

    componentDidMount() {
        this._loadFontsAsync();
    }
```

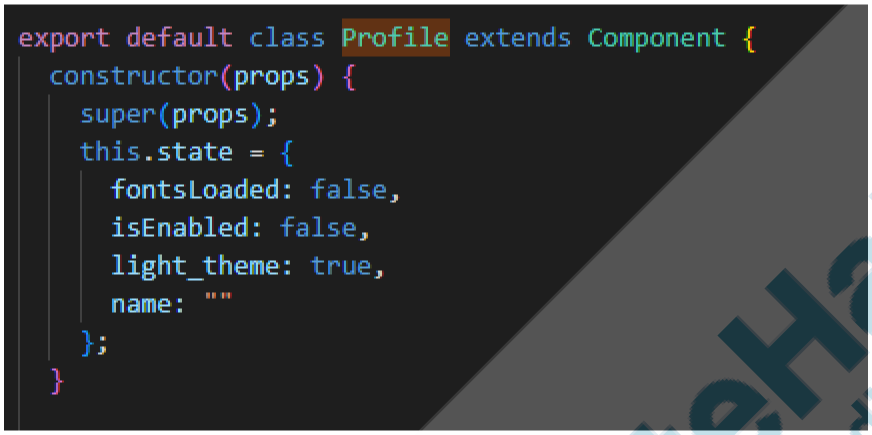


```
render() {
    if (!this.state.fontsLoaded) {
        SplashScreen.hideAsync();
    [redacted]
        return (
            <View
                style={{
                    flex: 1,
                    justifyContent: "center",
                    alignItems: "center"
                }}>
                <Text>Profile</Text>
            </View>
        )
    }
}
```

	Now this time, since we already have a user stored in firebase, we will be fetching the user's details first to display their name, profile image, and their preferred theme (*dark by default*). Import **getAuth and ref, update onValue** from **firebase/auth** and **firebase/database** respectively and write a function to fetch the user - *The teacher helps the student.*	*The student writes the code.*


```
import { getAuth } from 'firebase/auth';
import { ref, update, onValue } from 'firebase/database';
import db from '../config';
```

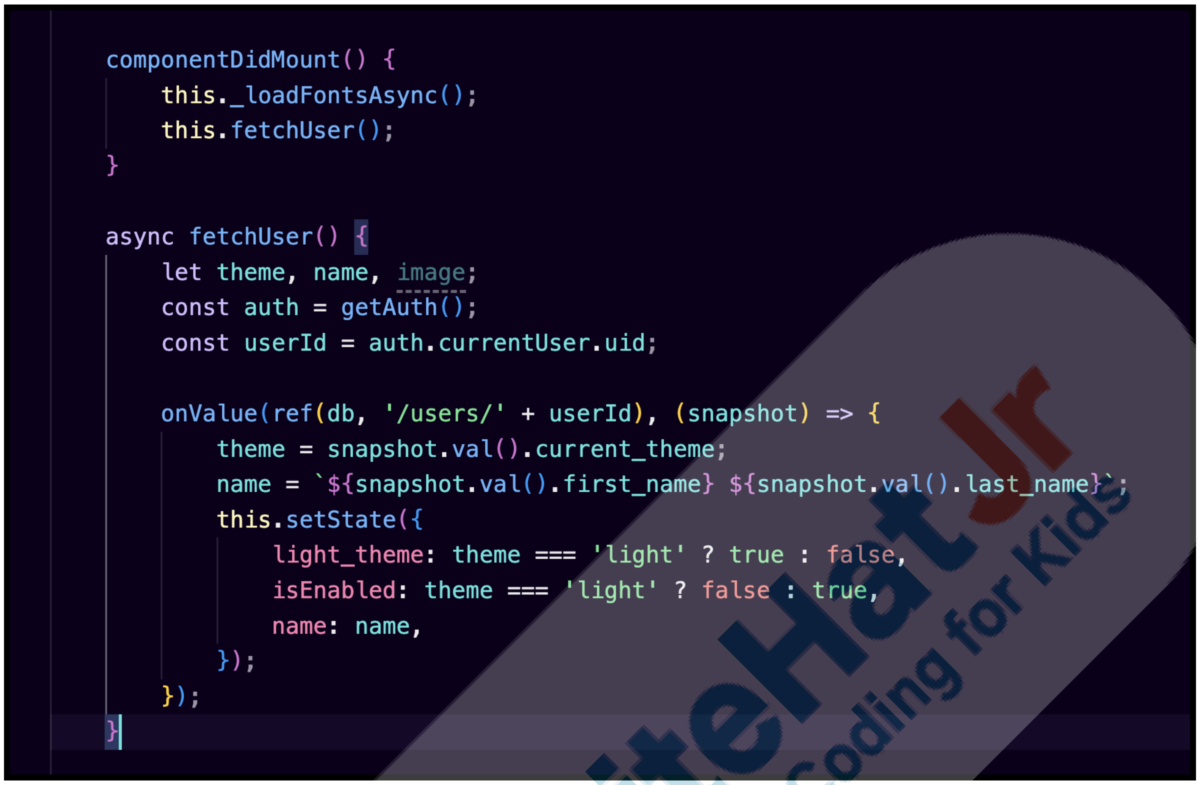
Now create properties inside the **constructor()**.


```
export default class Profile extends Component {
  constructor(props) {
    super(props);
    this.state = {
      fontsLoaded: false,
      isEnabled: false,
      light_theme: true,
      name: ""
    };
  }
```

Here, in the constructor, we will add **isEnabled**, for our toggle switch which we will create to toggle between themes.

We also have **light_theme** set to **true** by default. This will make sure that the App initially will have a light theme. We then have **name** as empty strings in the state to store data from **firebase** using the command **fetchUser**.

Next create a **fetchUser()** function which we are calling in our **componentDidMount()** function. This function fetches the user for us based on their **unique id**, which we can find from the **auth.currentUser.uid;**

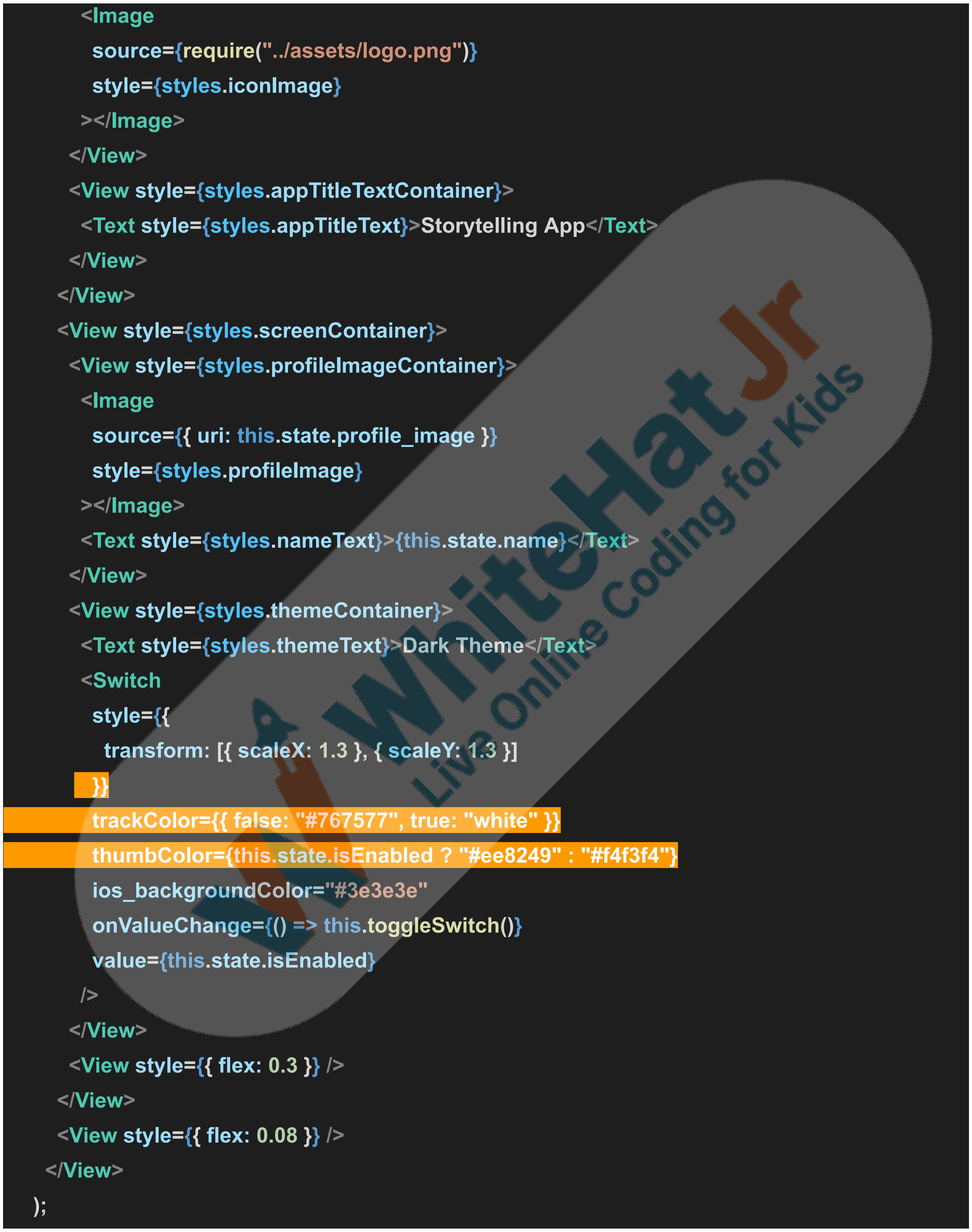


```
componentDidMount() {
    this._loadFontsAsync();
    this.fetchUser();
}

async fetchUser() {
    let theme, name, image;
    const auth = getAuth();
    const userId = auth.currentUser.uid;

    onValue(ref(db, '/users/' + userId), (snapshot) => {
        theme = snapshot.val().current_theme;
        name = `${snapshot.val().first_name} ${snapshot.val().last_name}`;
        this.setState({
            light_theme: theme === 'light' ? true : false,
            isEnabled: theme === 'light' ? false : true,
            name: name,
        });
    });
}
```

We are saving their preferred theme and their name from the data that we receive and are setting the **name** states based on that.

	Now we have all the data! Let's build the UI! For our switch that we have for selecting themes, you can use the **<Switch>** component from **react-native.** *Teacher guides and helps the student.*	*Students build the UI.*

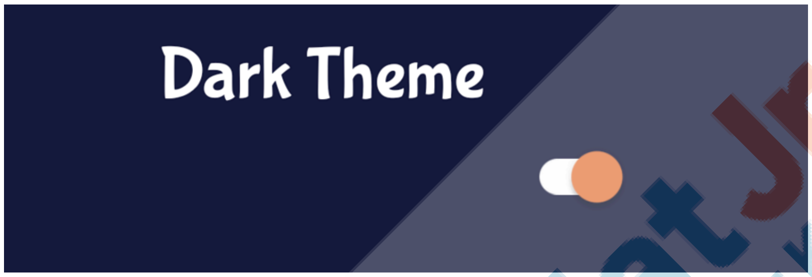
```
return (
   <View style={styles.container}>
    <SafeAreaView style={styles.droidSafeArea} />
    <View style={styles.appTitle}>
     <View style={styles.appIcon}>
```



```
          <Image
            source={require("../assets/logo.png")}
            style={styles.iconImage}
          ></Image>
        </View>
        <View style={styles.appTitleTextContainer}>
          <Text style={styles.appTitleText}>Storytelling App</Text>
        </View>
      </View>
      <View style={styles.screenContainer}>
        <View style={styles.profileImageContainer}>
          <Image
            source={{ uri: this.state.profile_image }}
            style={styles.profileImage}
          ></Image>
          <Text style={styles.nameText}>{this.state.name}</Text>
        </View>
        <View style={styles.themeContainer}>
          <Text style={styles.themeText}>Dark Theme</Text>
          <Switch
            style={{
              transform: [{ scaleX: 1.3 }, { scaleY: 1.3 }]
            }}
            trackColor={{ false: "#767577", true: "white" }}
            thumbColor={this.state.isEnabled ? "#ee8249" : "#f4f3f4"}
            ios_backgroundColor="#3e3e3e"
            onValueChange={() => this.toggleSwitch()}
            value={this.state.isEnabled}
          />
        </View>
        <View style={{ flex: 0.3 }} />
      </View>
      <View style={{ flex: 0.08 }} />
    </View>
  );
```

Here, we are using the **profile image** and **name of the user**. We are also using the **<Switch>** component to toggle between themes.

In this component, the attribute **trackColor** is for the color of the track (**ToggleSwitch**) when it’s **true** or **false**. Similarly, **thumbColor** is the color of the circle on the switch. (*Highlighted in the above code snippet*)



The styles for this would look like this -

```
const styles = StyleSheet.create({
 container: {
  flex: 1,
  backgroundColor: "#15193c"
 },
 droidSafeArea: {
  marginTop: Platform.OS === "android" ? StatusBar.currentHeight : 0
 },
 appTitle: {
  flex: 0.07,
  flexDirection: "row"
 },
 appIcon: {
  flex: 0.3,
  justifyContent: "center",
  alignItems: "center"
 },
 iconImage: {
  width: "100%",
  height: "100%",
  resizeMode: "contain"
},
appTitleTextContainer: {
  flex: 0.7,
  justifyContent: "center"
},
appTitleText: {
  color: "white",
  fontSize: RFValue(28),
  fontFamily: "Bubblegum-Sans"
},
screenContainer: {
  flex: 0.85
},
profileImageContainer: {
  flex: 0.5,
  justifyContent: "center",
  alignItems: "center"
},
profileImage: {
  width: RFValue(140),
  height: RFValue(140),
  borderRadius: RFValue(70)
},
nameText: {
  color: "white",
  fontSize: RFValue(40),
  fontFamily: "Bubblegum-Sans",
  marginTop: RFValue(10)
},
themeContainer: {
  flex: 0.2,
  flexDirection: "row",
  justifyContent: "center",
  marginTop: RFValue(20)
```

```
},
themeText: {
  color: "white",
  fontSize: RFValue(30),
  fontFamily: "Bubblegum-Sans",
  marginRight: RFValue(15)
}
});
```

Import the all required firebase methods as below -

```
import { getAuth } from 'firebase/auth';
import { ref, update, onValue } from 'firebase/database';
import db from '../config';
```

In **<Switch>**, we are using a function **toggleSwitch()** for the **onPress** event. That function will be coded as follows -

```
toggleSwitch() {
       const previous_state = this.state.isEnabled;
       const theme = !this.state.isEnabled ? 'dark' : 'light';

       const auth = getAuth();
       const user = auth.currentUser;

       if (user) {
           var updates = {};
           updates['users/' + user.uid + '/current_theme'] = theme;

           const dbRef = ref(db, '/');
           update(dbRef, updates);

           this.setState({
              isEnabled: !previous_state,
              light_theme:previous_state
```

```
            });
        }

};
```

Here, we are checking for the theme that the user has toggled to, if it's dark or light, and based on that, we are creating an object **updates** with the user's **reference in the database as the key** and the **theme they chose as the value**.

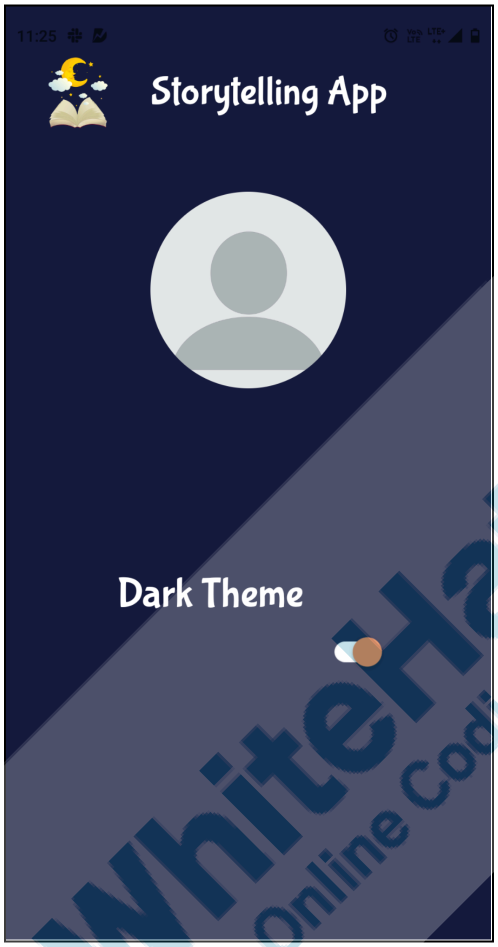
Based on that, we are updating their preferred theme in the database and changing the state.

The imports for these looks like this -

```
import {
 View,
 Text,
 StyleSheet,
 SafeAreaView,
 Platform,
 StatusBar,
 Image,
 Switch
} from "react-native";
```

	Now our screen looks perfect, however, we are still not able to toggle between the themes because we haven't built the light theme yet and added the functionality for it.	

Expected Output:



Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 5 Mins	
Teacher starts slideshow from slide 12 to slide 22	
Activity details	Solution/Guidelines
Run the presentation from slide 12 to slide 22. **Following are the wrap-up session deliverables:** ● **Explain the facts and trivias** ● **Next class challenge** ● **Project for the day**	Guide the student to develop the project and share it with us.

● **Additional Activity**		
End the quiz panel		
<u>FEEDBACK</u> ● **Appreciate the student for their class** ● **Get them to play around with different ideas**		
	Amazing work today! You get a "hats-off". Alright. In the next class, we will be building and then integrating the light theme into our app, so that the users can choose between the themes they prefer.	*Make sure you have given at least 2 Hats Off during the class for:* Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
	Project Overview **Spectagram Stage 6** **Goal of the Project:** In Class 86, we learned how to logout and created the Profile Screen. In this project, you will practice the concepts learned in the class to create Logout and Profile screens for the Spectagram app and include an attractive UI for the same. *This is a continuation project of 81 to 85, please make sure to finish that before attempting this one.	

	Story: Jenny is a photographer. She wants to share pictures taken by her with others. At the same time, she wants to create a space for others to share their talent too. She has decided to create a social media app for her and all upcoming talents. She has asked for your help to create an app. Guide Jenny to create an attractive Login and Profile screen for her app. I am very excited to see your project solution and I know you will do really well. Bye Bye!	
Teacher ends slideshow		
Teacher Clicks ✖ End Class		
ADDITIONAL ACTIVITY		
Additional Activities	*Encourage the student to write reflection notes in their reflection journal using Markdown.* Use these as guiding questions: ● What happened today? ○ Describe what happened. ○ The code I wrote.	*The student uses the markdown editor to write their reflections in a reflection journal.*

	• How did I feel after the class? • What have I learned about programming and developing games? • What aspects of the class helped me? What did I find difficult?	

Links:

Activity	Activity Name	Links
Teacher Activity 1	Previous class code	https://github.com/React-Native-Frontier/PRO-C85-Story-Teller-TA-Solution
Teacher Activity 2	Reference Code	https://github.com/React-Native-Frontier/PRO-C86-Story-Teller-TA-Solution
Teacher Activity 3	Teacher Aid	https://drive.google.com/file/d/1WA1BQff4dmgv5BInU3f_imk4vlpvAyMa/view?usp=sharing
Student Activity 1	Boilerplate Code	https://github.com/React-Native-Frontier/PRO-C86-Story-Teller-SA-Boilerplate
Teacher Reference visual aid link	Visual aid link	https://s3-whjr-curriculum-uploads.whjr.online/1ad9fdd5-83bf-4883-99e4-ba053236cee3.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/136e27e1-505e-4090-80d0-d4fe575884a3.pdf