Topic	**LOGIN AND PROFILE SCREEN**	
Class Description	**In this class, the student would be completing their Login and the Profile Screen of the App.**	
Class	**C86**	
Class time	**45 mins**	
Goal	● Complete the login screen of the App. ● 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.**

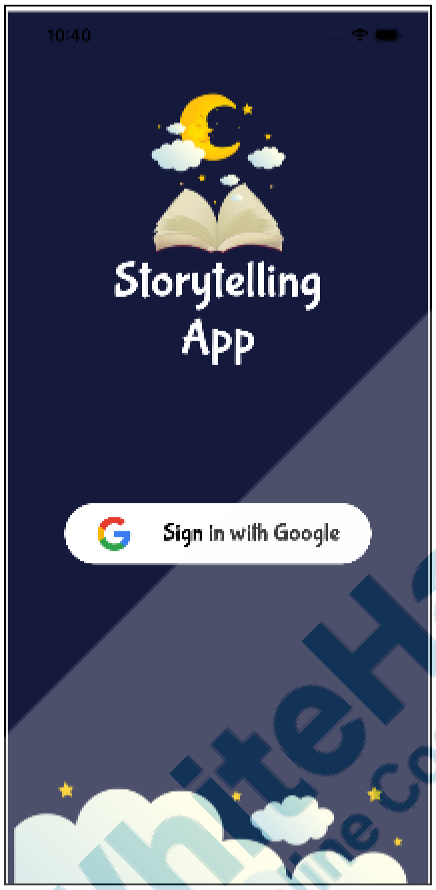
Teacher starts slideshow from slides 1 to 8

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.	
QnA Session	
Question	**Answer**
Under what reference name we are storing the new user's information in our database? A. users B. newUser C. currentUser D. USER	**A**
Which function can be used to find details of user logged in from firebase Authentication? A. firebase.auth().onAuthStateChanged() B. firebase.auth().currentUser() C. firebase.auth().UserLoggedIn D. firebase.auth().onAuthChanged()	**B**
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.

WhiteHat Jr
Live online Coding for Kids

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, as we know an attractive **UI** keeps users hooked and customers satisfied. *For reference you can access the previous class code in Teacher Activity 1* ***Note*** - *If the student and/or teacher is using the snack editor for these classes, please refer to the support document in Teacher Activity 4* Today let's add some life to the Login Screen. We want to make our Login Screen look something like this -	



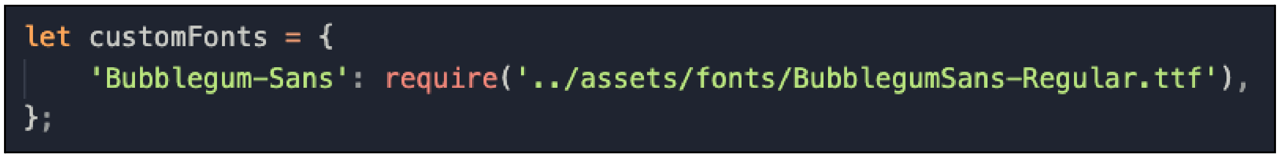
	Looks nice, right? We already have a template given to us in the boilerplate code for this -	

Teacher explains the boilerplate code to the student

We can see that the fonts are already imported into the code in the boilerplate provided.

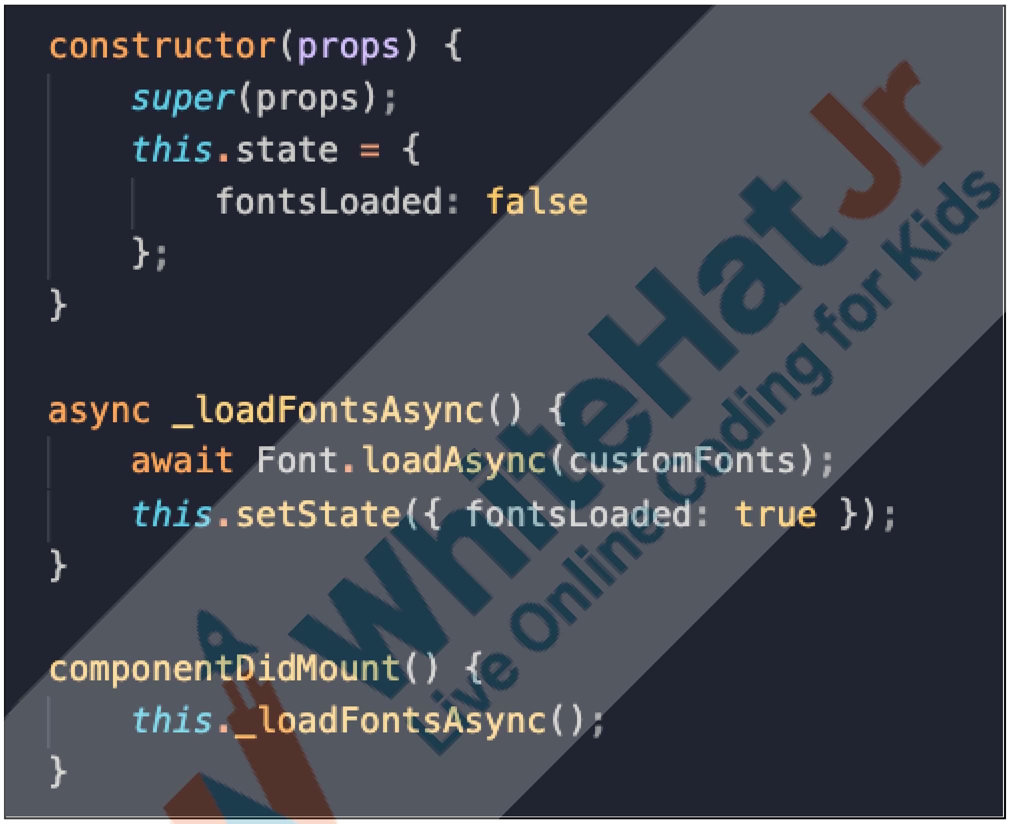
```
import AppLoading from 'expo-app-loading';
import * as Font from 'expo-font';
```

The fonts are also used -



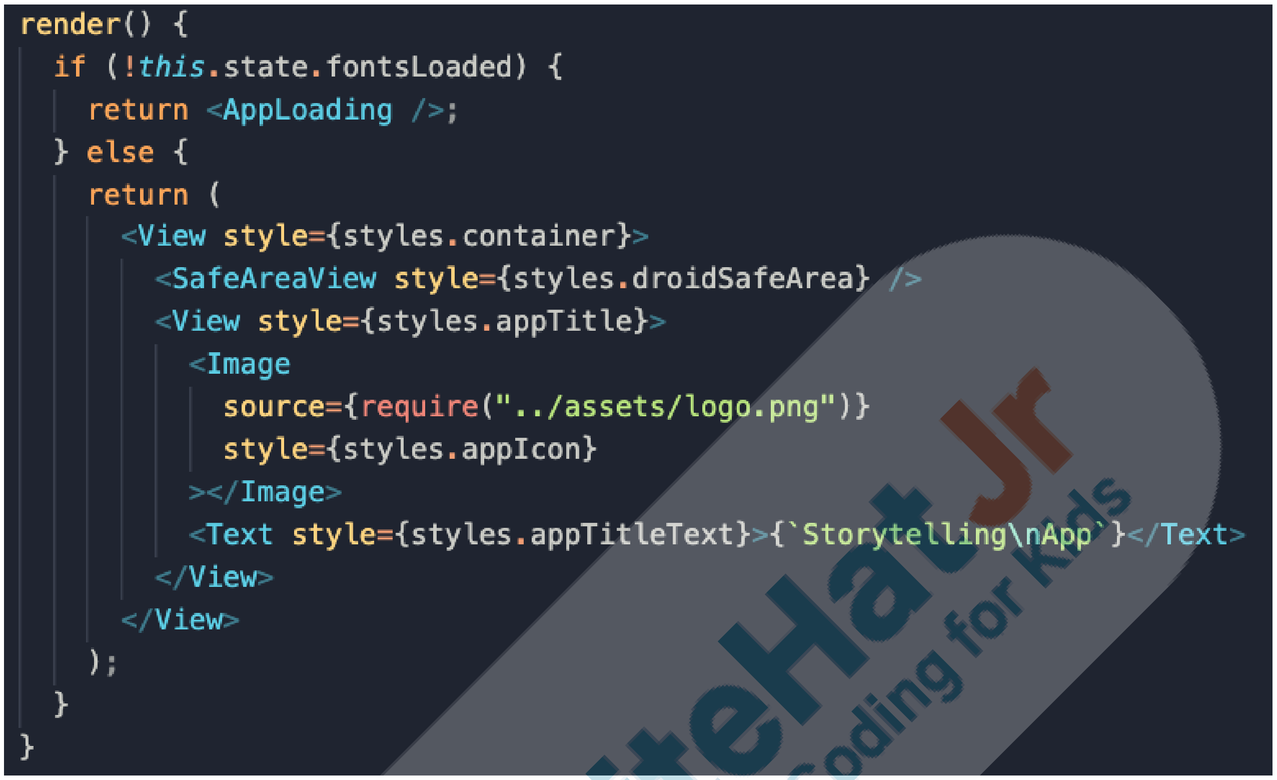
```
let customFonts = {
    'Bubblegum-Sans': require('../assets/fonts/BubblegumSans-Regular.ttf'),
};
```

We also have a **constructor()** and **loadFontsAsync()** functions and call it on **componentDidMount()**.



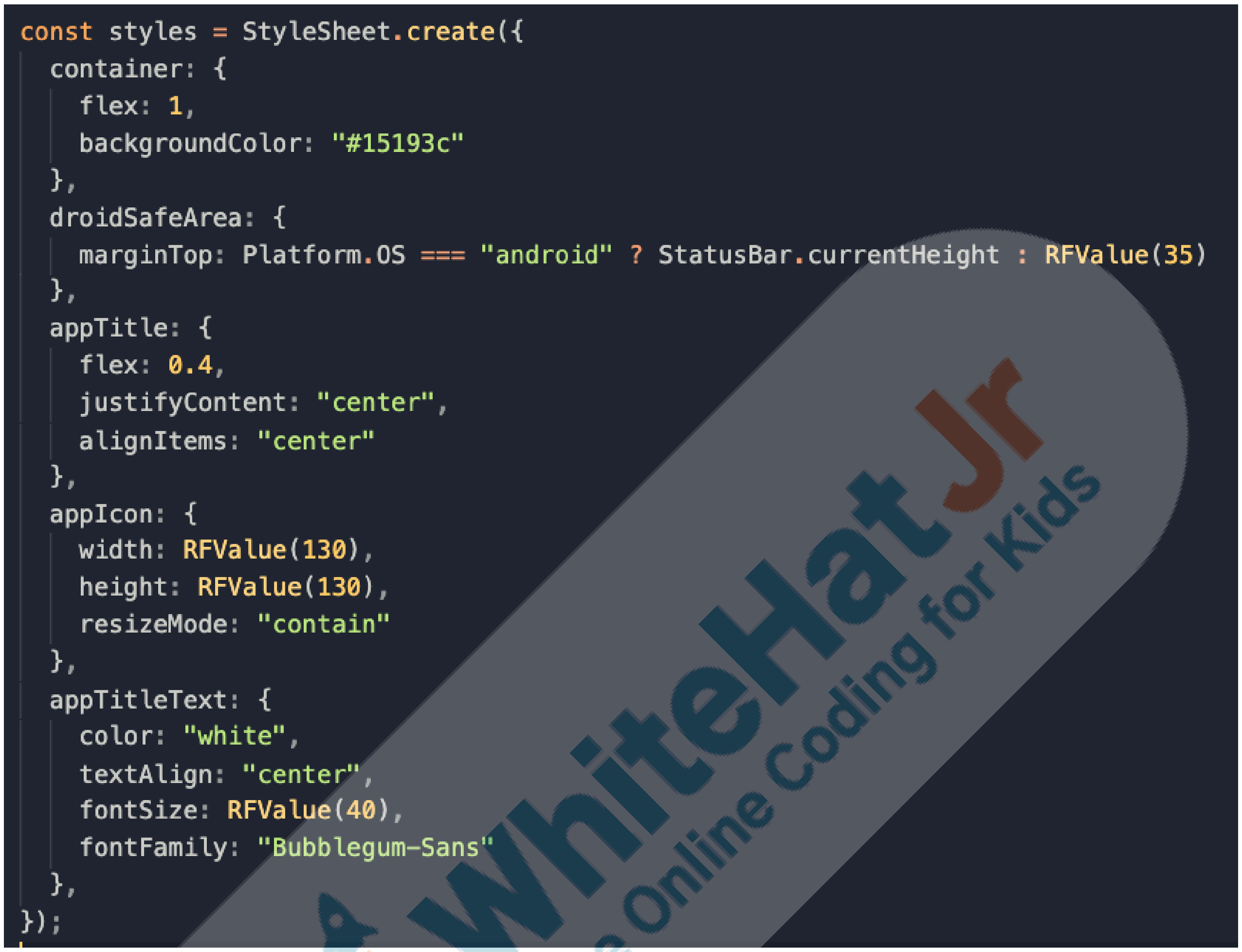
```
constructor(props) {
    super(props);
    this.state = {
        fontsLoaded: false
    };
}

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

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

Inside the render function, as done in previous screens, we have the code for title and app's image in the main container -



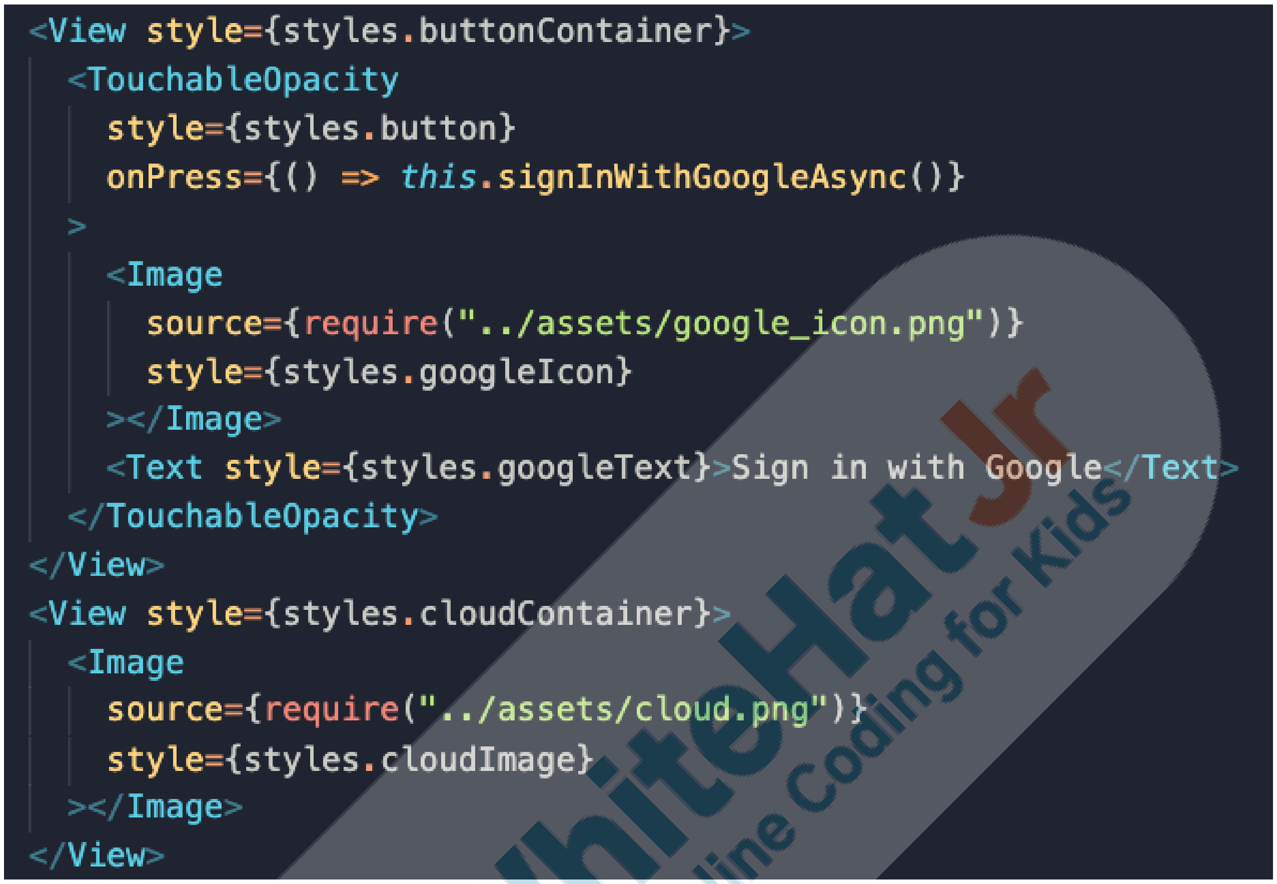
```
render() {
  if (!this.state.fontsLoaded) {
    return <AppLoading />;
  } else {
    return (
      <View style={styles.container}>
        <SafeAreaView style={styles.droidSafeArea} />
        <View style={styles.appTitle}>
          <Image
            source={require("../assets/logo.png")}
            style={styles.appIcon}
          ></Image>
          <Text style={styles.appTitleText}>{`Storytelling\nApp`}</Text>
        </View>
      </View>
    );
  }
}
```

We finally have the styling that goes for the render function above -



```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#15193c"
  },
  droidSafeArea: {
    marginTop: Platform.OS === "android" ? StatusBar.currentHeight : RFValue(35)
  },
  appTitle: {
    flex: 0.4,
    justifyContent: "center",
    alignItems: "center"
  },
  appIcon: {
    width: RFValue(130),
    height: RFValue(130),
    resizeMode: "contain"
  },
  appTitleText: {
    color: "white",
    textAlign: "center",
    fontSize: RFValue(40),
    fontFamily: "Bubblegum-Sans"
  },
});
```

	With our boilerplate now, we have the basic template of the app. According to the output, we now have to display the google login button and the clouds at the bottom of the screen! Let's start working on our render function to make it look like the UI we saw as shown in the below code snippet - *(Ask the student to help you out with the code)*	*Student and teacher work together to build the UI.*



```
<View style={styles.buttonContainer}>
  <TouchableOpacity
    style={styles.button}
    onPress={() => this.signInWithGoogleAsync()}
  >
    <Image
      source={require("../assets/google_icon.png")}
      style={styles.googleIcon}
    ></Image>
    <Text style={styles.googleText}>Sign in with Google</Text>
  </TouchableOpacity>
</View>
<View style={styles.cloudContainer}>
  <Image
    source={require("../assets/cloud.png")}
    style={styles.cloudImage}
  ></Image>
</View>
```

Below the Title Container view, we will add a **<TouchableOpacity>** component containing our button's code with Google's Icon and Text. We have also called our **signInWithGoogleAsync()** function which we created in the last class on the **onPress** event of our **<TouchableOpacity>**.

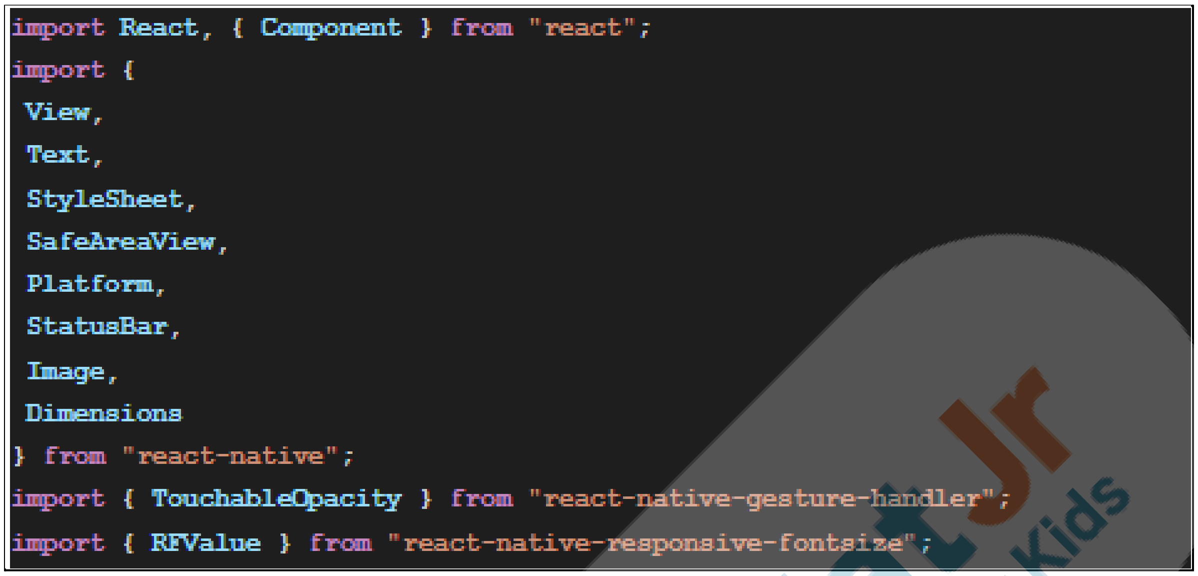
Next, we finally add a cloud image inside the **View** container with **cloudContainer** styles.

The styles for the button and clouds would be the following:

```
buttonContainer: {
   flex: 0.3,
   justifyContent: "center",
   alignItems: "center"
 },
 button: {
   width: RFValue(250),
   height: RFValue(50),
   flexDirection: "row",
   justifyContent: "space-evenly",
   alignItems: "center",
   borderRadius: RFValue(30),
   backgroundColor: "white"
 },
 googleIcon: {
   width: RFValue(30),
   height: RFValue(30),
   resizeMode: "contain"
 },
 googleText: {
   color: "black",
   fontSize: RFValue(20),
   fontFamily: "Bubblegum-Sans"
 },
 cloudContainer: {
   flex: 0.3
 },
 cloudImage: {
   position: "absolute",
   width: "100%",
   resizeMode: "contain",
   bottom: RFValue(-5)
 }
```

Note - *The styling is expected to be copy pasted.*

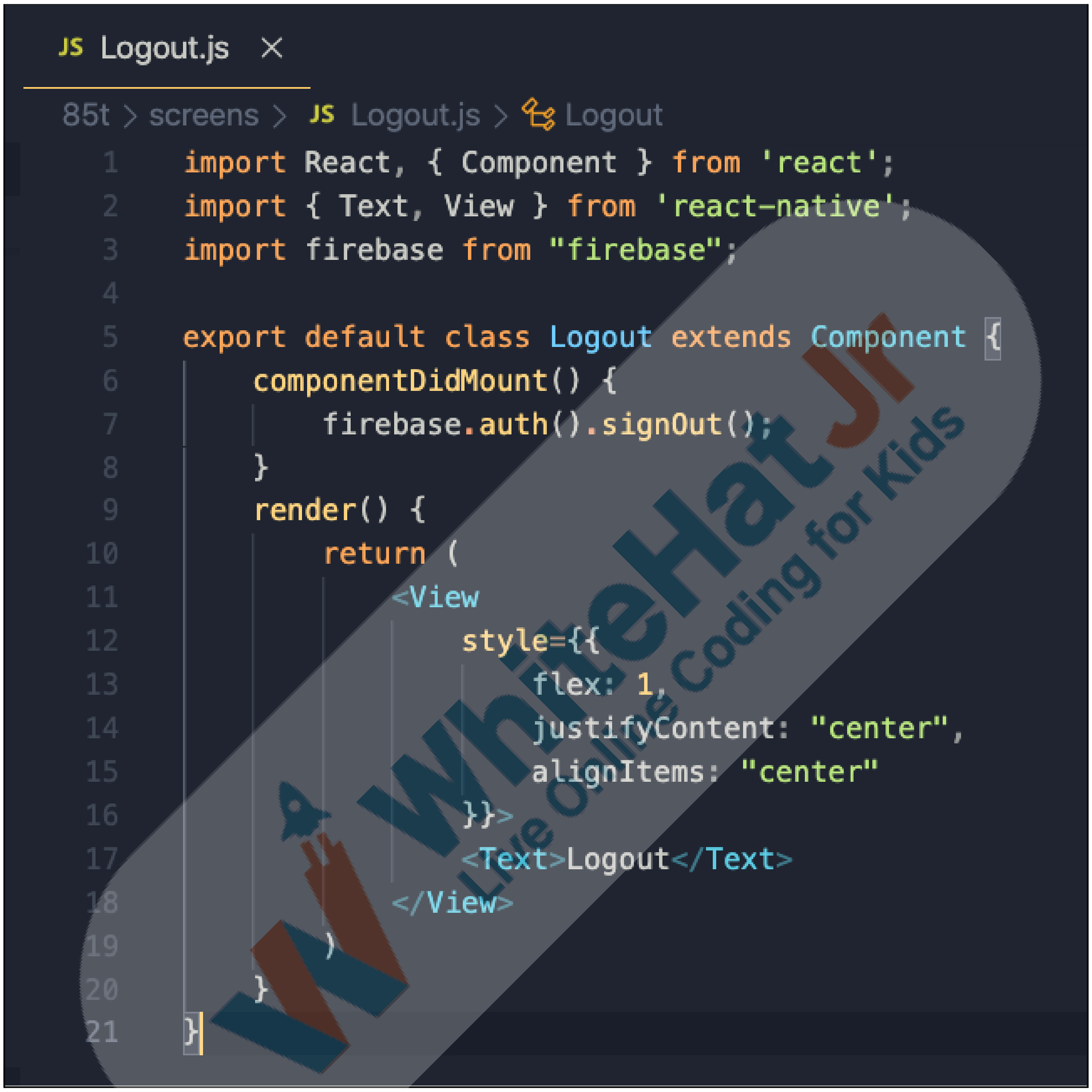
The imports for the Login and the Profile screen is the same as other screens created so far; see below for quick code reference:



```
import React, { Component } from "react";
import {
 View,
 Text,
 StyleSheet,
 SafeAreaView,
 Platform,
 StatusBar,
 Image,
 Dimensions
} from "react-native";
import { TouchableOpacity } from "react-native-gesture-handler";
import { RFValue } from "react-native-responsive-fontsize";
```

	Great! Now if we open our App, it looks just how we wanted it to!	
	With this, our login screen is complete. We will not be adding the light theme for this screen as we don't know what user has logged in and what theme they prefer.	
	What do you do when you are done working on any app which requires you to login? Logout from the app. 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.	**ESR**: Close it / Logout / Varied.

	Let’s quickly create a **Logout.js** in our **screens** folder for that. Here, in the **componentDidMount()** function, we will use **firebase.auth().signOut()**, 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.	



Logout.js

85t > screens > Logout.js > Logout

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

export default class Logout extends Component {
    componentDidMount() {
        firebase.auth().signOut();
    }
    render() {
        return (
            <View
                style={{
                    flex: 1,
                    justifyContent: "center",
                    alignItems: "center"
                }}>
                <Text>Logout</Text>
            </View>
        )
    }
}
```

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

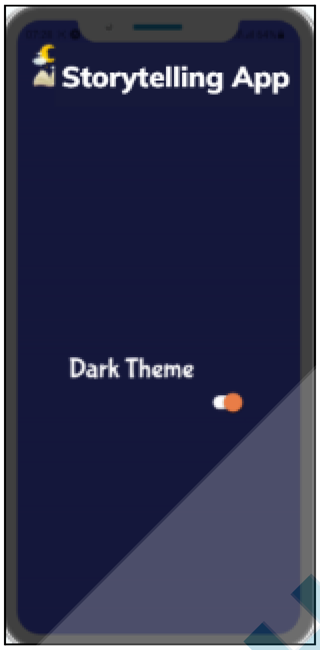
```
import React from "react";
import { createDrawerNavigator } from "@react-navigation/drawer";
import StackNavigator from "./StackNavigator";
import Profile from "../screens/Profile";
import Logout from "../screens/Logout";

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 our Login screen and added logout in our side drawer.	
	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. You will also have to update the OAuth IDs in the login screen.	*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 login and logout screen. Now you will be creating the profile screen. That screen would look something like this -	

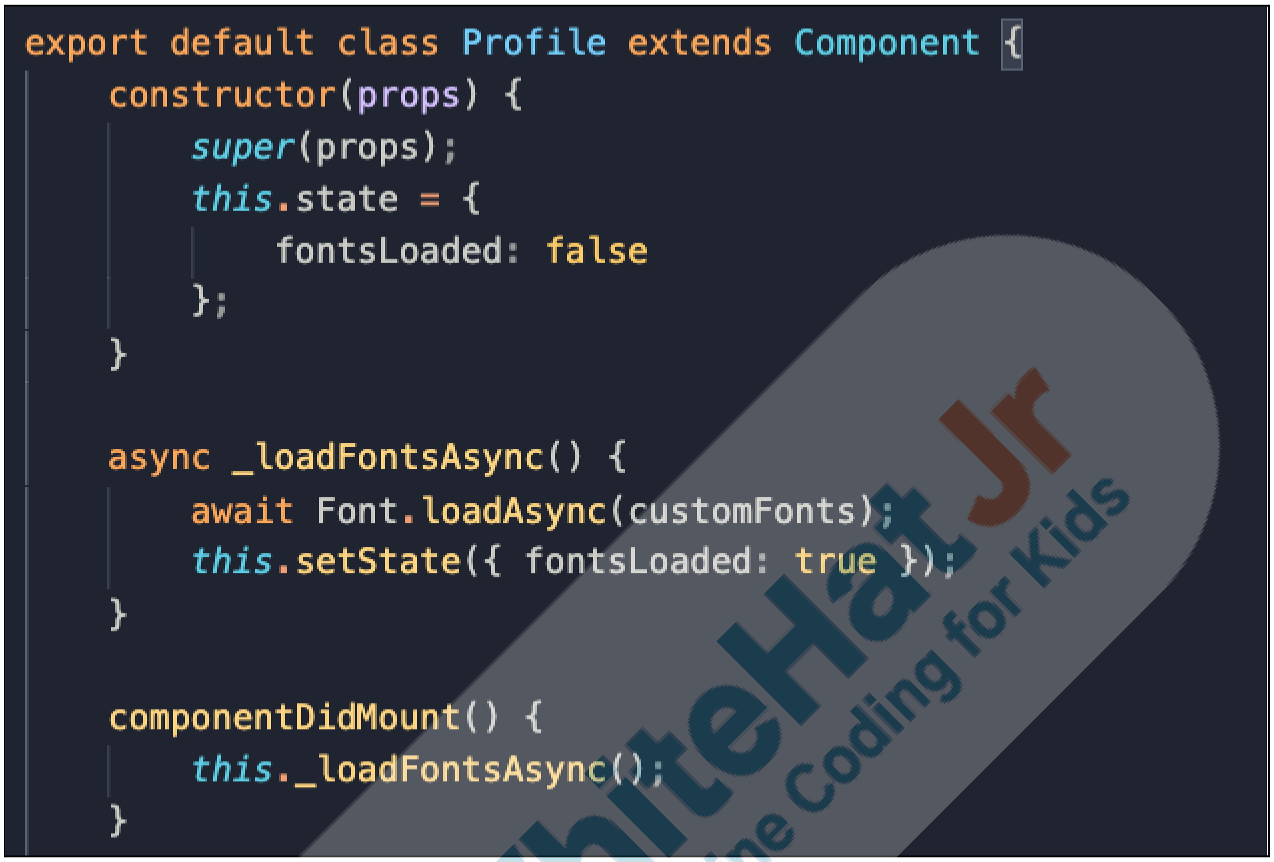


Now let’s code for the same!

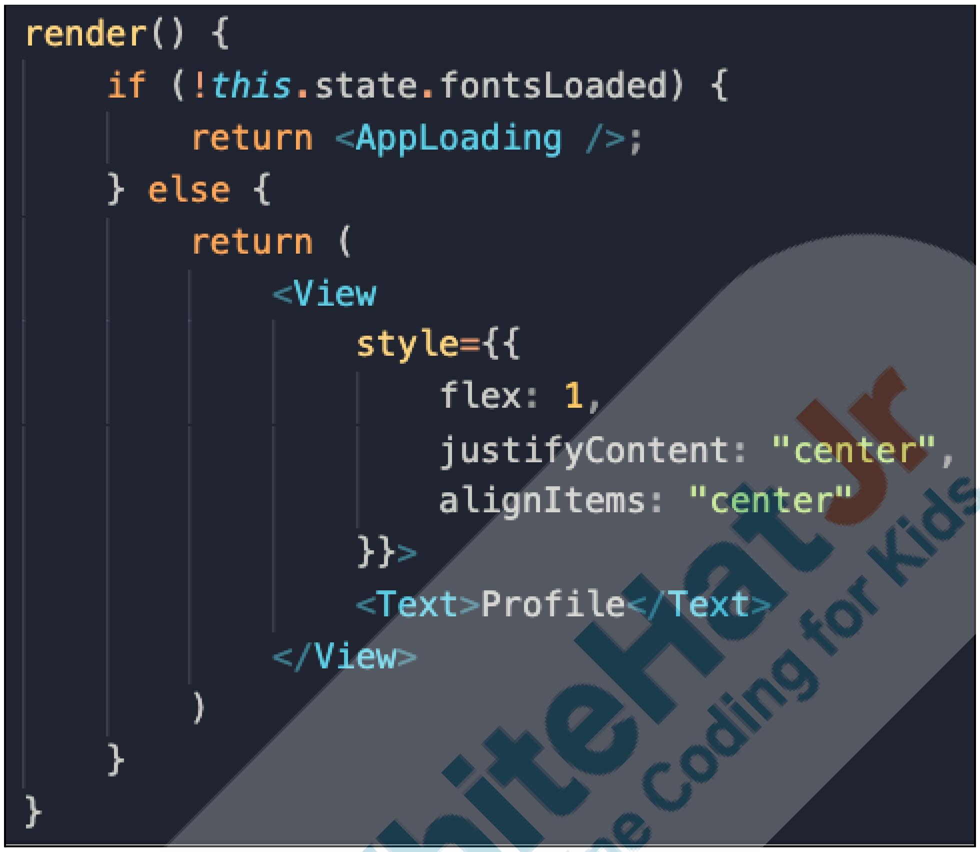
This time around, we will be coding differently.

We already have the fonts added to the screen from the boilerplate code we just cloned.

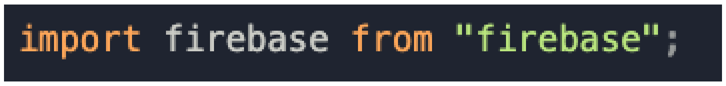
```
import React, { Component } from "react";
import {
  View,
  Text
} from "react-native";
import { RFValue } from "react-native-responsive-fontsize";
import AppLoading from "expo-app-loading";
import * as Font from "expo-font";

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



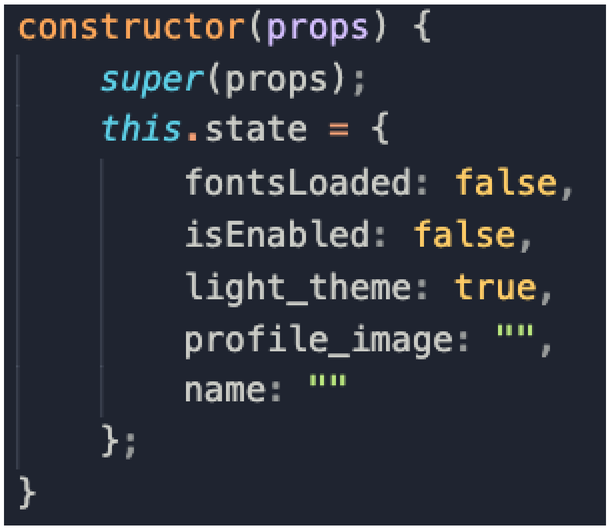
```
export default class Profile extends Component {
    constructor(props) {
        super(props);
        this.state = {
            fontsLoaded: false
        };
    }

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

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


```
render() {
    if (!this.state.fontsLoaded) {
        return <AppLoading />;
    } else {
        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 **firebase** and write a function to fetch the user - *The teacher helps the student.*	*The student writes the code.*


```
import firebase from "firebase";
```

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



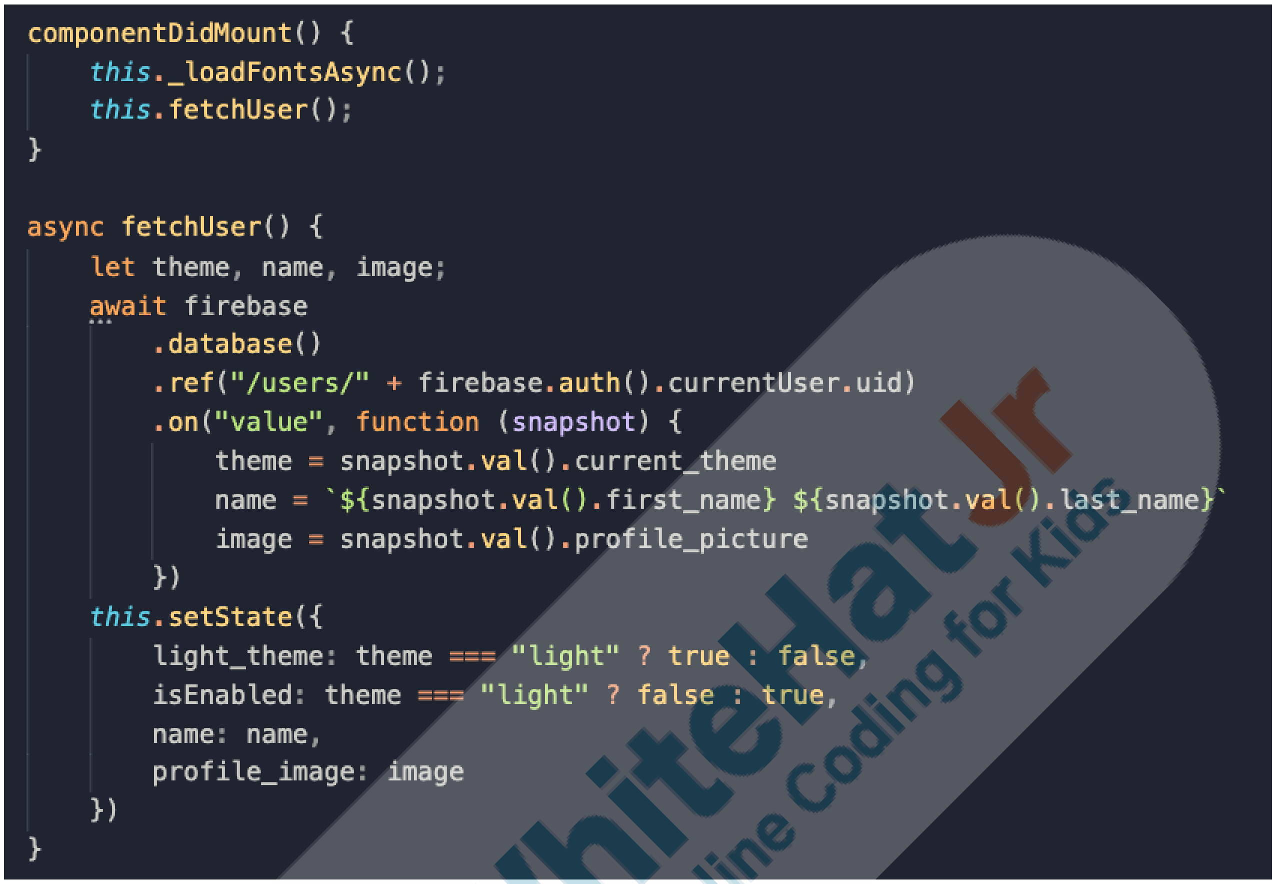
```
constructor(props) {
    super(props);
    this.state = {
        fontsLoaded: false,
        isEnabled: false,
        light_theme: true,
        profile_image: "",
        name: ""
    };
}
```

Teacher explains the code to the student

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 **profile_image** and **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 **firebase.auth().currentUser.uid**.



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

async fetchUser() {
    let theme, name, image;
    await firebase
        .database()
        .ref("/users/" + firebase.auth().currentUser.uid)
        .on("value", function (snapshot) {
            theme = snapshot.val().current_theme
            name = `${snapshot.val().first_name} ${snapshot.val().last_name}`
            image = snapshot.val().profile_picture
        })
    this.setState({
        light_theme: theme === "light" ? true : false,
        isEnabled: theme === "light" ? false : true,
        name: name,
        profile_image: image
    })
}
```

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

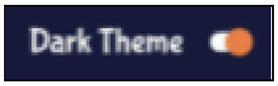
	Now we have all the data! Let's build the UI! For our switch that we have for selecting themes, you can use the **<Switch>** component from **react-native.** *Teacher guides and helps the student.*	*Student builds the UI.*

```
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)
  }
});
```

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"
        var updates = {}
        updates["/users/" + firebase.auth().currentUser.uid + "/current_theme"] = theme
        firebase.database().ref().update(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 -

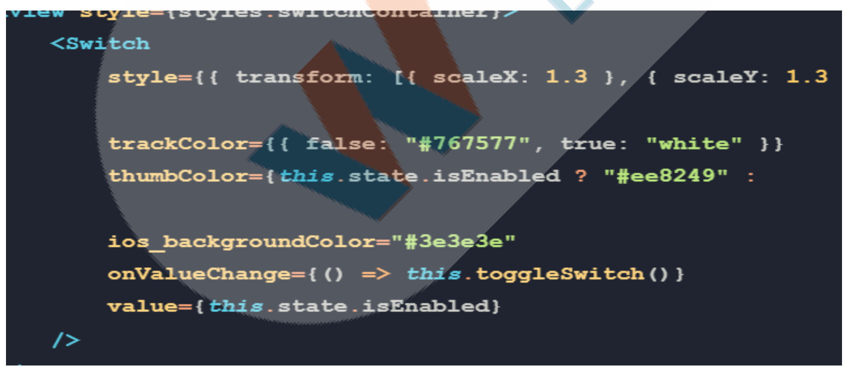
```
import {
 View,
 Text,
 StyleSheet,
 SafeAreaView,
 Platform,
 StatusBar,
 Image,
 Switch
} from "react-native";
```

	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.	

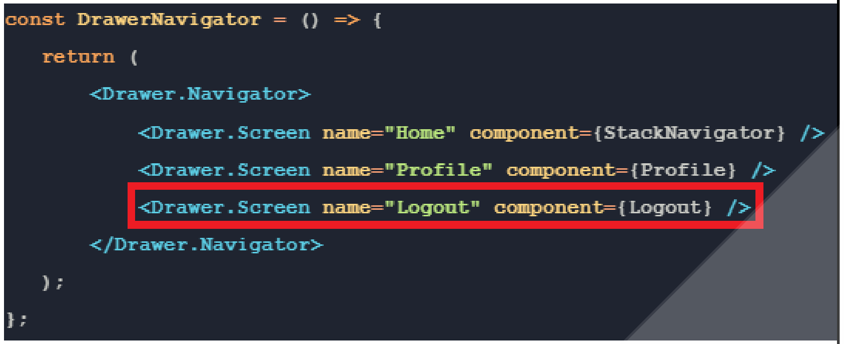
Teacher Guides Student to Stop Screen Share

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** ● **Additional Activity**	Guide the student to develop the project and share it with us.
Quiz time - Click on the in-class quiz	
Question	**Answer**
To toggle the switch between the two themes, which property is added in the constructor? A. dark_theme B. theme C. light_theme D. isEnabled	**D**

What does the <Switch> component in the following snippet of code do?

`view style={styles.switchContainer}>`
`  <Switch`
`      style={{ transform: [{ scaleX: 1.3 }, { scaleY: 1.3`
`      trackColor={{ false: "#767577", true: "white" }}`
`      thumbColor={this.state.isEnabled ? "#ee8249" :`
`      ios_backgroundColor="#3e3e3e"`
`      onValueChange={() => this.toggleSwitch()}`
`      value={this.state.isEnabled}`
`  />`
A. We are using the **<Switch>** component to change the theme.
B. We are using the **<Switch>** component to toggle between themes.
C. We are using the **<Switch>** component to change the background image.
	B

D. None of the above.		

What does the highlighted piece of code in the following snippet do?

`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>`
`  );`
`};`
A. We have added the <Drawer.Screen> component to have a logout functionality in the Drawer Screen.
B. We are navigating to the logout screen.
C. We are allowing the user to logout from the app.
D. None of the above.
		A

End the quiz panel		
FEEDBACK • **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 create a Login and the Profile Screen also improvised UI of these screens In this project, you will practice the concepts learned in the class to create Login and Profile screens for the Spectagram app and include 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. ● 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?	*The student uses the markdown editor to write their reflections in a reflection journal.*

Links:

Activity	Activity Name	Links
Teacher Activity 1	Teacher Boilerplate Code	https://github.com/pro-whitehatjr/Story-Telling-App-86-TB
Teacher Activity 2	Reference Code	https://github.com/pro-whitehatjr/ST-86-Solution
Teacher Activity 3	Teacher Aid	https://drive.google.com/file/d/1WA1

		BQff4dmgv5BInU3f_imk4vlpvAyMa/view?usp=sharing
Teacher Activity 4	Snack Support Document	https://docs.google.com/document/d/11vq49uJQCfdxaUUzOoY7A65aau0kZqNMFhObZH-e71Y/edit?usp=sharing
Student Activity 1	Boilerplate Code	https://github.com/pro-whitehatjr/ST-86-Boilerplate
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/PRO_V3_C86_LITE_withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/60e0c36e-28a1-4e37-81d2-8173217e64b9.pdf