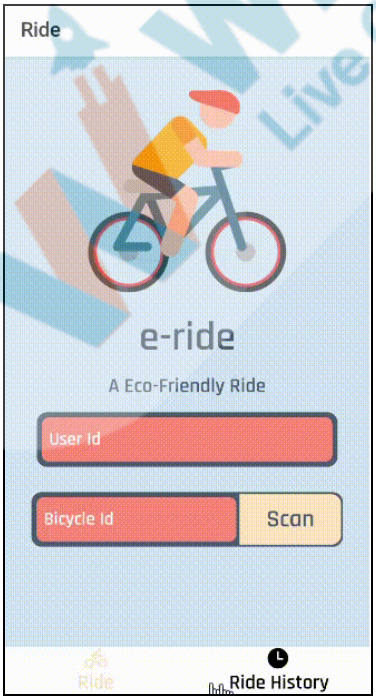
Topic	AUTHENTICATING USERS	
Class Description	**The student learns to authenticate a user in their app by asking them to enter their username and password.** **The student also learns to change security rules for their database so that only authenticated users can access - read and write values - to the database.**	
Class	**C75**	
Class time	**40 mins**	
Goal	● Create an authentication page for the user to authenticate. ● Use firebase authentication service to authenticate a user. ● Modify security rules in the firebase database to allow only authenticated users to access the database.	
Resources Required	● Teacher Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Android/iOS Smartphone with Expo App installed ● Student Resources ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Android/iOS Smartphone with Expo App installed	
Class structure	**Warm-Up** **Teacher-led Activity** **Student-led Activity** **Wrap-Up**	**5 min** **15 min** **15 min** **5 min**
WARM-UP SESSION - 5 mins		
CONTEXT ● **Get the student to think about the problem of allowing any user to access the app and the app database.**		

Teacher starts slideshow from slides 1 to 10 Refer to speaker notes and follow the instructions on each slide.	
Activity details	**Solution/Guidelines**
Hey <student's 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:** • **Greet the student.** • **Revision of previous class activities.** • **Quizzes**	**ESR**: Hi, thanks, Yes I am excited about it! Click on the slide show tab and present the slides
QnA Session	
Question	**Answer**
Choose the right block of code that can be used to render the FlatList component.  	**A**

A. ``` {/* <FlatList data={allTransactions} renderItem={this.renderItem} keyExtractor={(item, index) => index.toString()} onEndReached={() => this.fetchMoreTransactions(searchText)} onEndReachedThreshold={0.7} /> */} ``` B. ``` {/* <FlatList data=allTransactions renderItem=this.renderItem keyExtractor={(item, index) => index.toString()} onEndReached={() => this.fetchMoreTransactions(searchText)} onEndReachedThreshold={0.7} /> */} ``` C. ``` {/* <FlatList data:{allTransactions} renderItem:{this.renderItem} keyExtractor:{(item, index) => index.toString()} onEndReached:{() => this.fetchMoreTransactions(searchText)} onEndReachedThreshold={0.7} /> */} ``` D. ``` {/* <FlatList data={"allTransactions"} renderItem={"this.renderItem"} keyExtractor={(item, index) => index.toString()} onEndReached= {this.fetchMoreTransactions(searchText)} onEndReachedThreshold={0.7} /> */} ``` 	

Choose the right block of code that describes the fetchMoreTransactions() function.
A.
``
fetchMoreTransactions = async text => {
var enteredText = text.toUpperCase().split("");
text = text.toUpperCase();
}
``
B.
``
fetchMoreTransactions = async text => (
var enteredText = text.toUpperCase().split("");
text = text.toUpperCase();
)
``
 	
	C

C.
```
fetchMoreTransactions = async () text => {
    var enteredText = text.toUpperCase().split("");
    text = text.toUpperCase();
}
```

D.
```
fetchMoreTransactions () = async text => {
    var enteredText = text.toUpperCase().split("");
    text = text.toUpperCase();
}
```

Continue the WARM-UP session

Activity details	Solution/Guidelines
Run the presentation from slide 4 to slide 10 to set the problem statement. **Following are the WARM-UP session deliverables:** ● Appreciate the student. ● Authentication page for e-library app.	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

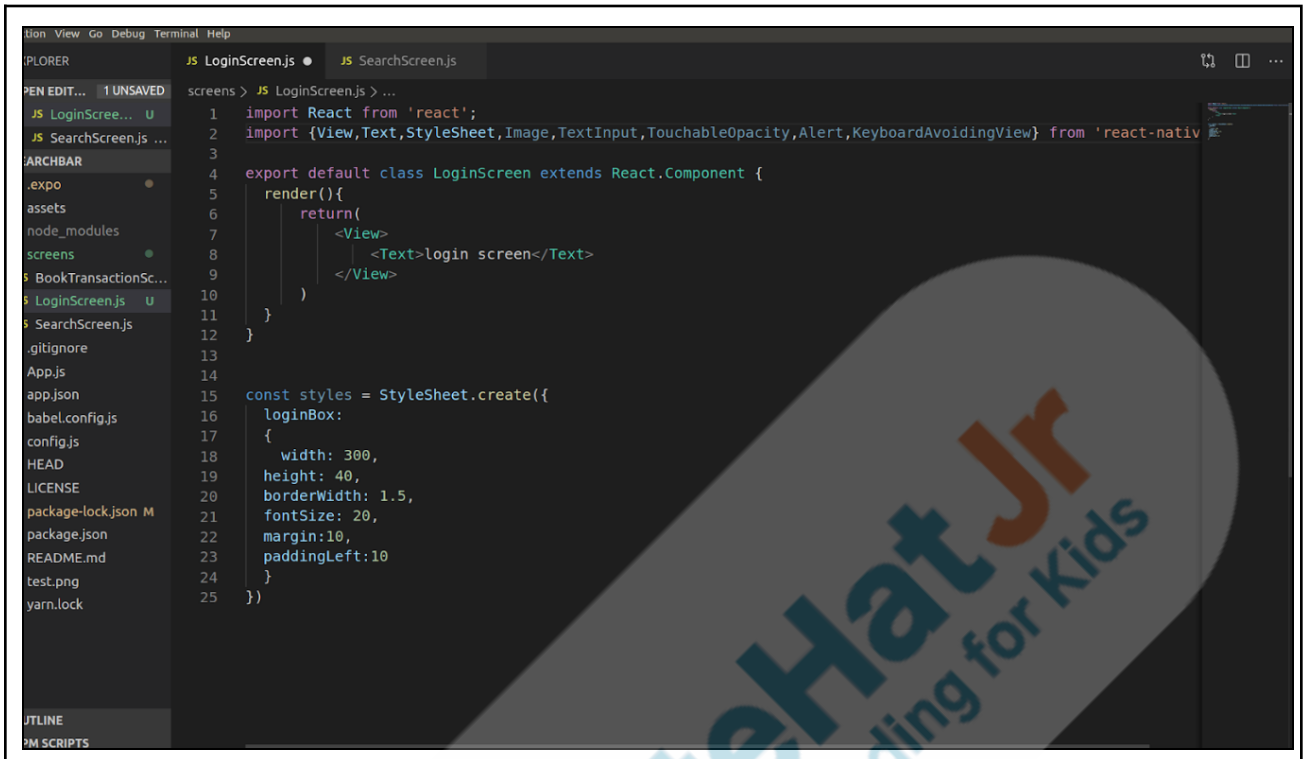
Teacher Initiates Screen Share

CHALLENGE

- **Design an authentication page for the user to authenticate using email and password.**
- **Use firebase authentication service to authenticate the user.**

Teacher Action	Student Action
The teacher opens the code from the previous class or clones the code from [Teacher Activity 1](#).	

Steps to clone the project:- **git clone <projectURL>** **cd <projectFolder>** **npm install** Let's start by creating a new screen for the login page. We can simply create a new file inside our **screens** folder called **Login.js.** For now, we can write a simple code to display some text on this page. This simple code will contain a **View** component and a **Text** component to add a text on the screen. And some styling for it. *The teacher codes for **Login.js** with the help of students' inputs.*	



We are using **TabNavigation** in our app. However, we do not want the login screen as a tab in our application. This should be the first screen that the users must-see. How should the user move to the next screen?	**ESR:** By entering email, password and hitting a login button.
What kind of navigation supports switching to another screen with the press of a button?	**ESR:** The **Switch** Navigator.
Excellent!! Since we do not want the login screen as a tab in our application, we are going to create a **SwitchNavigator** which will contain both - the login page and the **TabNavigator** - we had created earlier. Can you help me in creating the **SwitchNavigator**?	*The student gives inputs while the teacher writes the code.*

Now we import **createSwitchNavigator** from 'react-navigation'.

Then we create a **SwitchNavigator** containing **BottomTabNavigator** and **LoginScreen**.

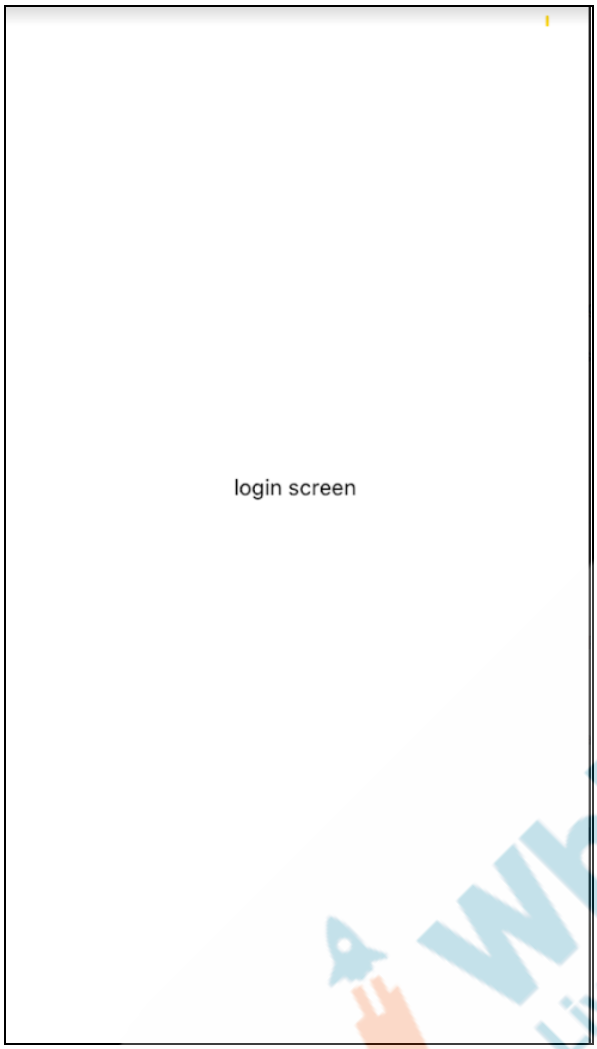
We will also modify **AppContainer** to contain the **SwitchNavigator.**

The teacher writes the code.

```
JS App.js > ...
import React, { Component } from "react";
import { Rajdhani_600SemiBold } from "@expo-google-fonts/rajdhani";
import * as Font from "expo-font";
import db from "./config";
import LoginScreen from "./screens/Login";
import BottomTabNavigator from "./components/BottomTabNavigator";

import { createSwitchNavigator, createAppContainer } from "react-navigation";

const AppSwitchNavigator = createSwitchNavigator(
  {
    Login: {
      screen: LoginScreen
    },
    BottomTab: {
      screen: BottomTabNavigator
    }
  },
  {
    initialRouteName: "Login"
  }
);

const AppContainer = createAppContainer(AppSwitchNavigator);
```

```
render() {
  const { fontLoaded } = this.state;
  if (fontLoaded) {
    return <AppContainer />;
  }
  return null;
}
```

Let's test our app and see if we see a login page when the user opens the app. *The teacher runs the code to test the output.*	*The student observes the output and comments.*



How do we move to the next screen now?	**ESR:** We will have to create **TextInput** buttons to collect email and password from the user. We will also have to create a button to navigate to the next screen.
Let's design the login screen to contain the two **TextInput** components and one **TouchableOpacity** button.	

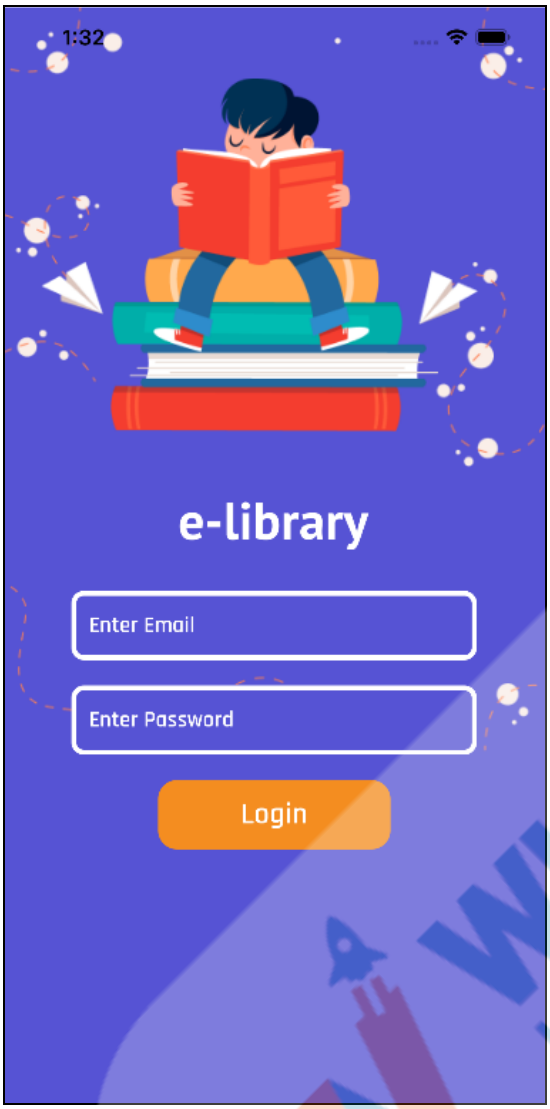
*The teacher introduces the props **keyBoardType** and **secureTextEntry** for the **TextInput** Component to collect email and password from the user.* **keyBoardType** prop opens the keyboard based on the type of input to be taken from the user, for example, if the input to be taken is a name the alphabetic keyboard will open. If the input to be taken is a number format, it will open a numeric keyboard. **secureTextEntry** prop helps to maintain the secrecy of the password that has been entered. It will show stars in place of the password. *The teacher writes the code to create the UI for the login screen as follows:*	

```
<KeyboardAvoidingView behavior="padding" style={styles.container}>
  <ImageBackground source={bgImage} style={styles.bgImage}>
    <View style={styles.upperContainer}>
      <Image source={appIcon} style={styles.appIcon} />
      <Image source={appName} style={styles.appName} />
    </View>
    <View style={styles.lowerContainer}>
      <TextInput
        style={styles.textinput}
        onChangeText={text => this.setState({ email: text })}
        placeholder={"Enter Email"}
        placeholderTextColor={"#FFFFFF"}
        autoFocus
      />
      <TextInput
        style={[styles.textinput, { marginTop: 20 }]}
        onChangeText={text => this.setState({ password: text })}
        placeholder={"Enter Password"}
        placeholderTextColor={"#FFFFFF"}
        secureTextEntry
      />
      <TouchableOpacity
        style={[styles.button, { marginTop: 20 }]}
        onPress={() => this.handleLogin(email, password)}
      >
        <Text style={styles.buttonText}>Login</Text>
      </TouchableOpacity>
    </View>
  </ImageBackground>
</KeyboardAvoidingView>
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#FFFFFF"
  },
  bgImage: {
    flex: 1,
    resizeMode: "cover",
    justifyContent: "center"
  },

  upperContainer: {
    flex: 0.5,
    justifyContent: "center",
    alignItems: "center"
  },
  appIcon: {
    width: 280,
    height: 280,
    resizeMode: "contain",
    marginTop: 80
  },
  appName: {
    width: 130,
    height: 130,
    resizeMode: "contain"
  },
  lowerContainer: {
    flex: 0.5,
    alignItems: "center"
  },
```

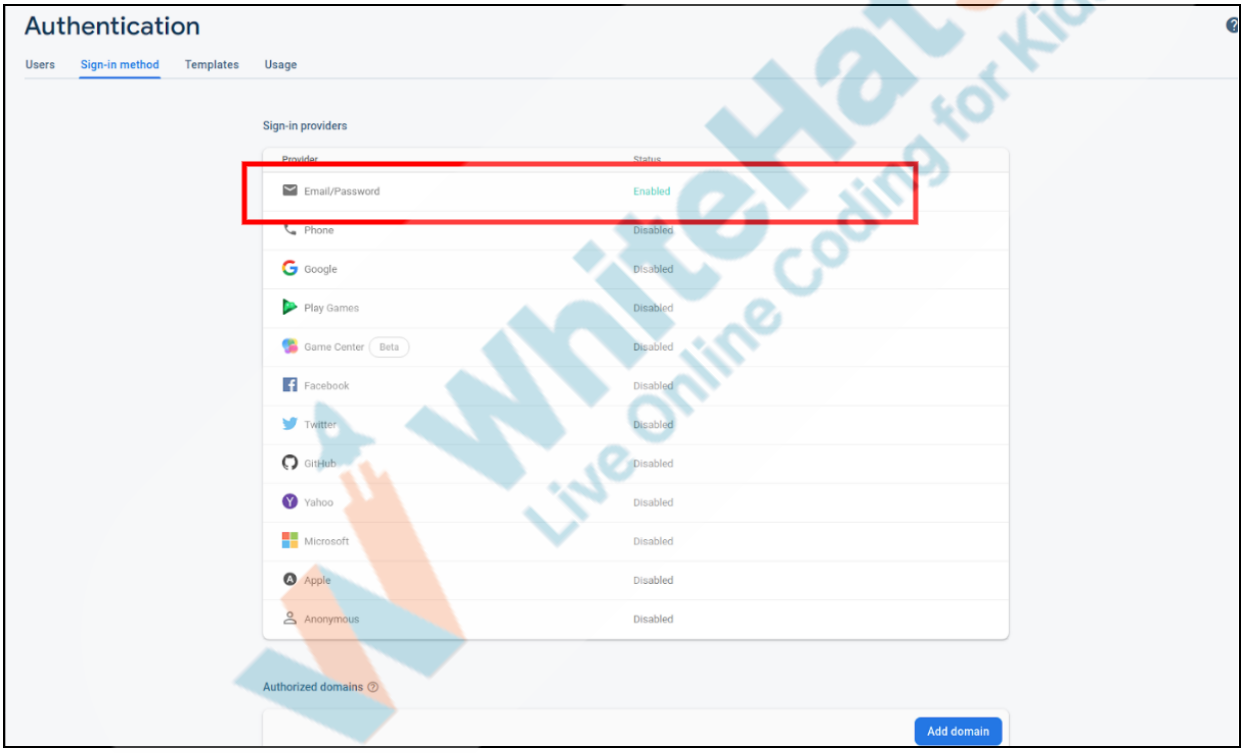
```
  lowerContainer: {
    flex: 0.5,
    alignItems: "center"
  },
  textinput: {
    width: "75%",
    height: 55,
    padding: 10,
    borderColor: "#FFFFFF",
    borderWidth: 4,
    borderRadius: 10,
    fontSize: 18,
    color: "#FFFFFF",
    fontFamily: "Rajdhani_600SemiBold",
    backgroundColor: "#5653D4"
  },
  button: {
    width: "43%",
    height: 55,
    justifyContent: "center",
    alignItems: "center",
    backgroundColor: "#F48D20",
    borderRadius: 15
  },
  buttonText: {
    fontSize: 24,
    color: "#FFFFFF",
    fontFamily: "Rajdhani_600SemiBold"
  }
});
```

Output:

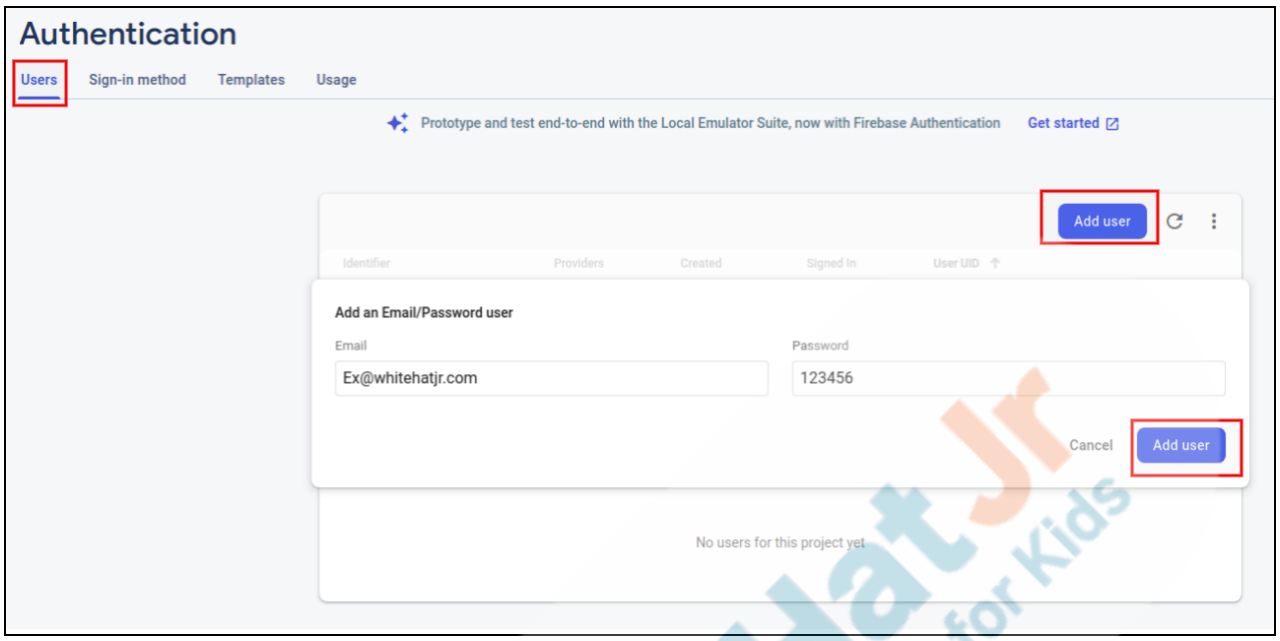


We are done with designing the login screen. When the user enters the email ID and password, there should be something to check if the email ID and password are registered. Firebase provides an authentication service that helps us do that. *The teacher can refer to the authentication document from [Teacher Activity 2](#).*	*The student can refer to the authentication document from [Student Activity 2](#).*

*The teacher shows how to enable email '**sign in**' inside the authentication tab in the database for the e-library library.* Login to the firebase website using your registered email and password. Open the **e-library database**. Click on **Authentication** in the side panel. Click on the **Sign-in** method.	*The student observes how different authentication services are enabled inside the firebase database.*



Let's select **Email/Password**, to register an email ID and password for test purposes. *The teacher registers an email and password by adding a user.*	



Now, in our app code, when the user presses the button, we will call the **login()** function, which will check if the username and password are registered. Firebase provides us an auth service to do that using **firebase.auth().signInWithEmailAndPassword()** Let's do that. Our **handleLogin()** function is going to take some time to authenticate. *The teacher creates the **handleLogin()** function and uses **firebase.auth().signInWithEmailAndPassword()** to authenticate the registered user and navigate to the next screen if the authentication is successful.*	*The student helps the teacher write the code.*

```
<KeyboardAvoidingView behavior="padding" style={styles.container}>
  <ImageBackground source={bgImage} style={styles.bgImage}>
    <View style={styles.upperContainer}>
      <Image source={appIcon} style={styles.appIcon} />
      <Image source={appName} style={styles.appName} />
    </View>
    <View style={styles.lowerContainer}>
      <TextInput
        style={styles.textinput}
        onChangeText={text => this.setState({ email: text })}
        placeholder={"Enter Email"}
        placeholderTextColor={"#FFFFFF"}
        autoFocus
      />
      <TextInput
        style={[styles.textinput, { marginTop: 20 }]}
        onChangeText={text => this.setState({ password: text })}
        placeholder={"Enter Password"}
        placeholderTextColor={"#FFFFFF"}
        secureTextEntry
      />
      <TouchableOpacity
        style={[styles.button, { marginTop: 20 }]}
        onPress={() => this.handleLogin(email, password)}
      >
        <Text style={styles.buttonText}>Login</Text>
      </TouchableOpacity>
    </View>
  </ImageBackground>
</KeyboardAvoidingView>
```

```
handleLogin = (email, password) => {
  firebase
    .auth()
    .signInWithEmailAndPassword(email, password)
    .then(() => {
      this.props.navigation.navigate("BottomTab");
    })
    .catch(error => {
      Alert.alert(error.message);
    });
};
```

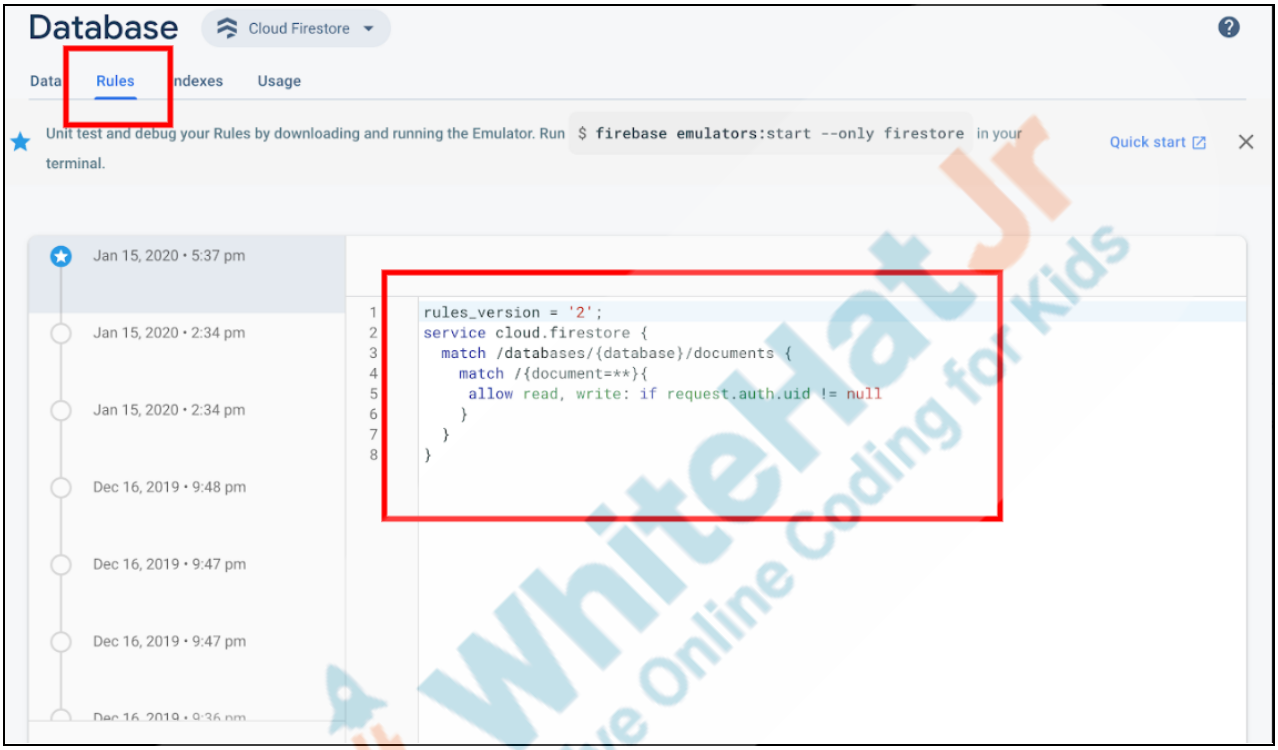
Let's now test our app and see if our login works. *The teacher runs the app and checks by logging in using the registered ID and password.*	*The student observes the output.*

Great! We now have user authentication ready.

One final change. We need to change our database rules to allow only authenticated users to access, read and modify our database.

There is a separate way for writing complex database rules, which we will learn later. For now, we can just make

a small change to allow everyone who is authenticated to read and write values in our database. Here, **auth.uid!=null** means to allow a person who is authenticated to read and write values in our database.	*The student observes how to make these changes in the database security rules.*



Awesome! Great work! Now, can you do this in your own app?	**ESR**: Yes!

Teacher Stops Screen Share

	Now it's your turn. Please share your screen with me.	

STUDENT-LED ACTIVITY - 15 mins

- **Ask Student to press ESC key to come back to panel**
- **Guide Student to start Screen Share**
- **Teacher gets into Fullscreen**

ACTIVITY
• **Recreate the login screen and its functionality in the app.** • **Try login with an unauthorized user ID to observe the error toast message.**
Teacher starts slideshow : Slide 11 to 12 Refer to speaker notes and follow the instructions on each slide.

Teacher Action	Student Action
Now you will be recreating the login screen and its functionalities in the app by yourself. Begin by creating the **SwitchNavigator** containing **BottomTabNavigator** and the Login Screen Create the basic Login Screen UI.	*Student can use the code from previous class or clone the code from Student Activity 1.* *The student writes code for the Login Screen.*

```
import React from 'react';
import {View,Text,StyleSheet,Image,TextInput,TouchableOpacity,Alert,KeyboardAvoidingView} from 'react-nativ

export default class LoginScreen extends React.Component {
  render(){
      return(
          <View>
              <Text>login screen</Text>
          </View>
      )
  }
}


const styles = StyleSheet.create({
  loginBox:
  {
    width: 300,
  height: 40,
  borderWidth: 1.5,
  fontSize: 20,
  margin:10,
  paddingLeft:10
  }
})
```

Create **SwitchNavigator** containing **BottomTabNavigator** and the Login Screen.	*The student creates a **SwitchNavigator** which contains **BottomTabNavigator** and LoginScreen.*

```
import React, { Component } from "react";
import { Rajdhani_600SemiBold } from "@expo-google-fonts/rajdhani";
import * as Font from "expo-font";
import db from "./config";
import LoginScreen from "./screens/Login";
import BottomTabNavigator from "./components/BottomTabNavigator";

import { createSwitchNavigator, createAppContainer } from "react-navigation";
```

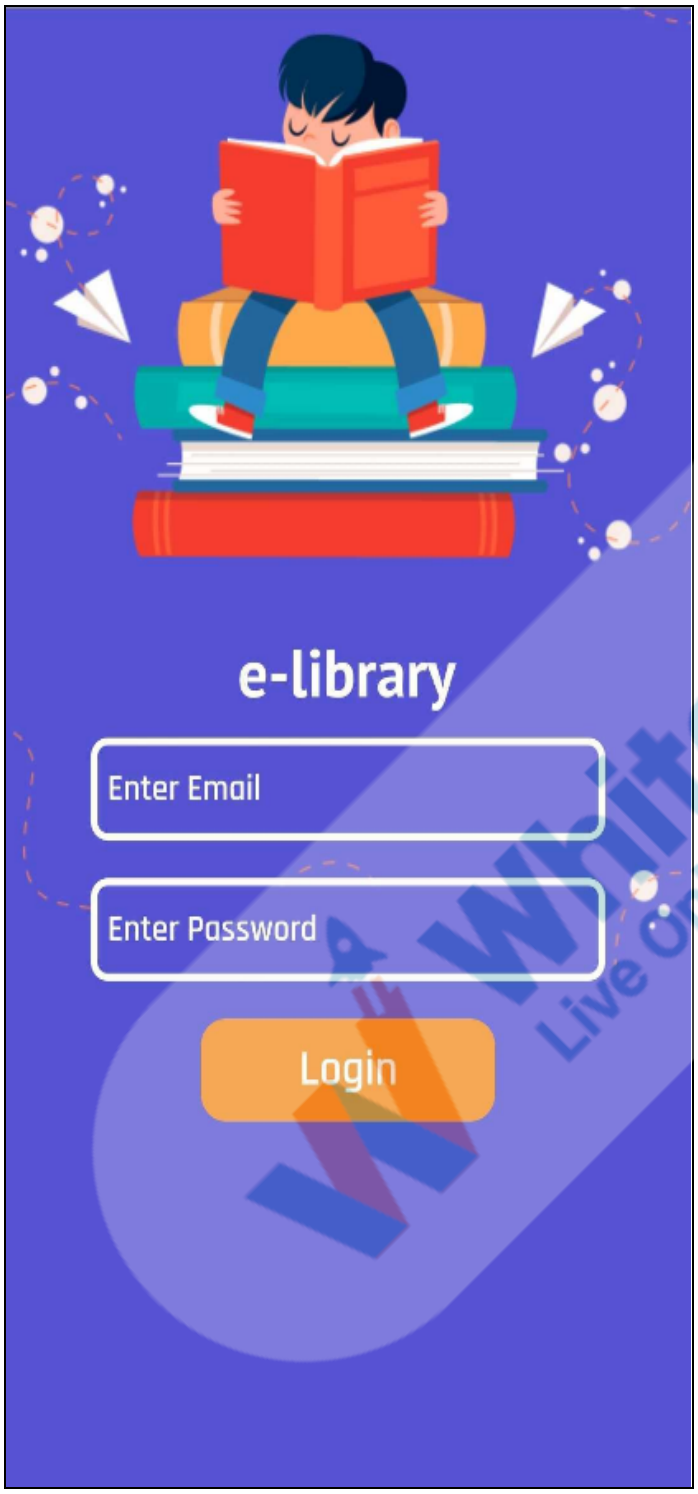
```
const AppSwitchNavigator = createSwitchNavigator(
  {
    Login: {
      screen: LoginScreen
    },
    BottomTab: {
      screen: BottomTabNavigator
    }
  },
  {
    initialRouteName: "Login"
  }
);

const AppContainer = createAppContainer(AppSwitchNavigator);
```

```
  render() {
    const { fontLoaded } = this.state;
    if (fontLoaded) {
      return <AppContainer />;
    }
    return null;
  }
```

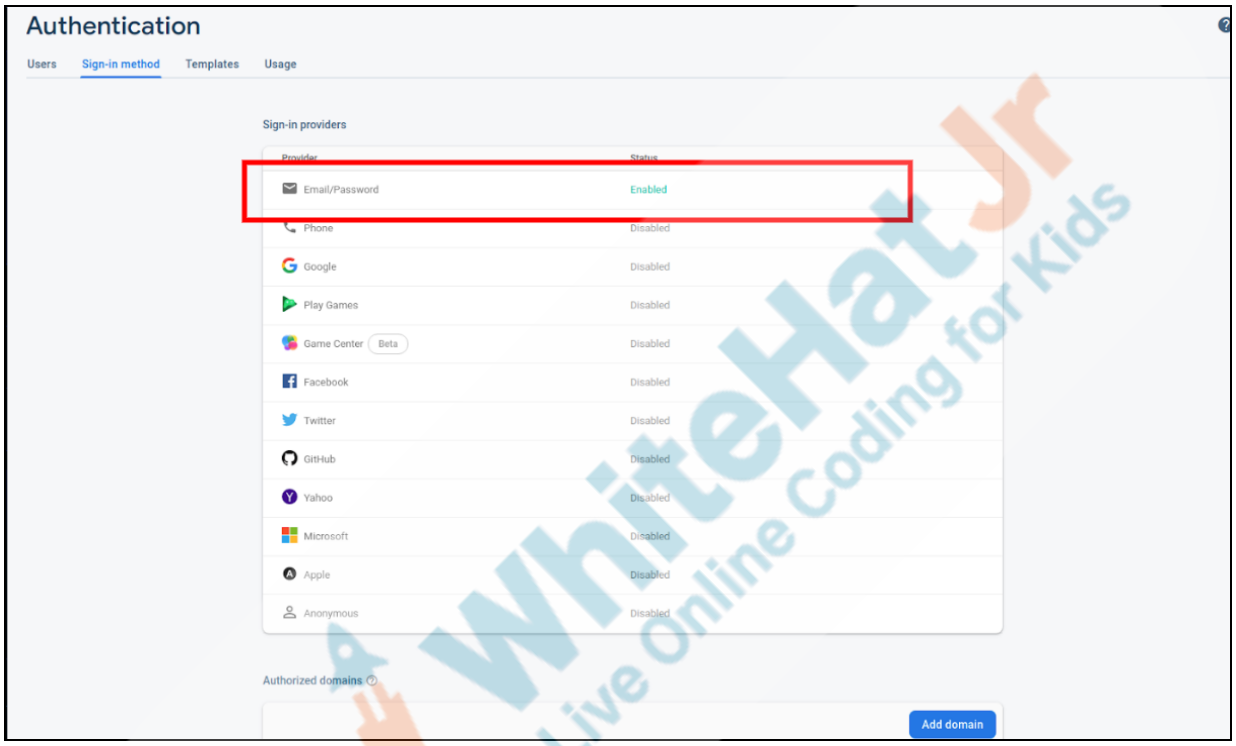
Now, create the Login Screen UI.	*The student creates the Login Screen User Interface containing the 2 TextInput Components and a Touchable Opacity Button.*

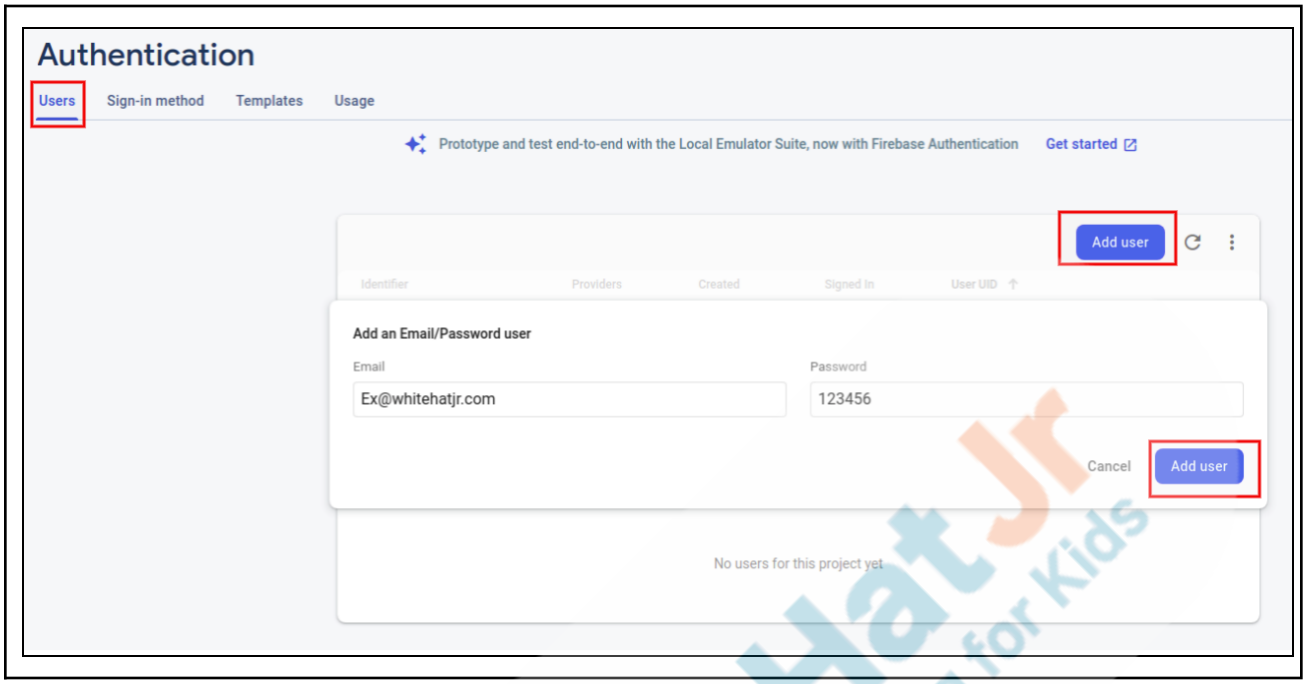
```
<KeyboardAvoidingView behavior="padding" style={styles.container}>
  <ImageBackground source={bgImage} style={styles.bgImage}>
    <View style={styles.upperContainer}>
      <Image source={appIcon} style={styles.appIcon} />
      <Image source={appName} style={styles.appName} />
    </View>
    <View style={styles.lowerContainer}>
      <TextInput
        style={styles.textinput}
        onChangeText={text => this.setState({ email: text })}
        placeholder={"Enter Email"}
        placeholderTextColor={"#FFFFFF"}
        autoFocus
      />
      <TextInput
        style={[styles.textinput, { marginTop: 20 }]}
        onChangeText={text => this.setState({ password: text })}
        placeholder={"Enter Password"}
        placeholderTextColor={"#FFFFFF"}
        secureTextEntry
      />
      <TouchableOpacity
        style={[styles.button, { marginTop: 20 }]}
        onPress={() => this.handleLogin(email, password)}
      >
        <Text style={styles.buttonText}>Login</Text>
      </TouchableOpacity>
    </View>
  </ImageBackground>
</KeyboardAvoidingView>
```

Output for the UI:



Now, enable firebase authorization for email sign-ins and add registered users.	*The student enables the firebase authorization service for email sign-ins. He/She creates a registered email id and password for testing.*



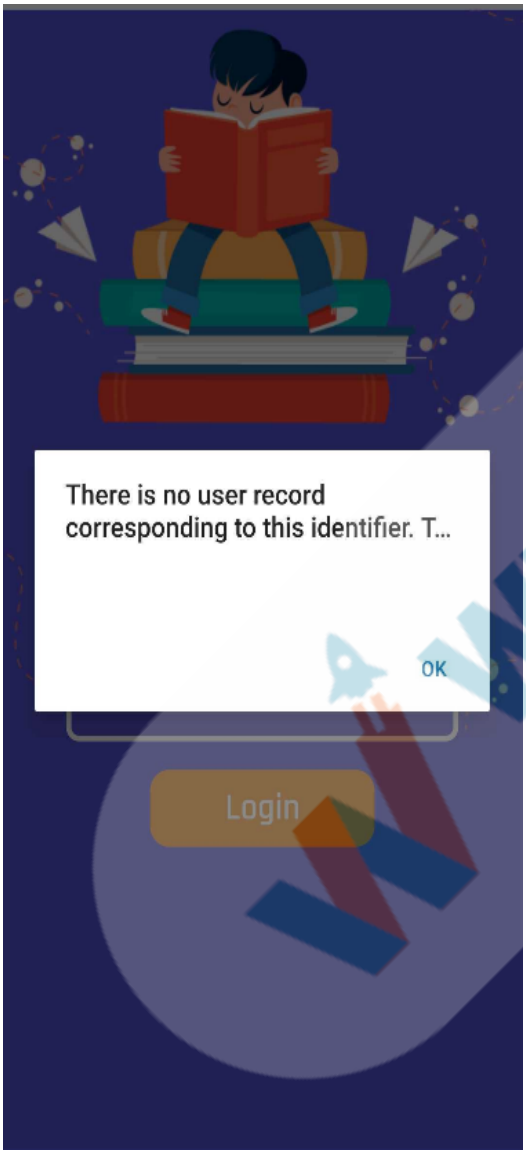


```
<KeyboardAvoidingView behavior="padding" style={styles.container}>
  <ImageBackground source={bgImage} style={styles.bgImage}>
    <View style={styles.upperContainer}>
      <Image source={appIcon} style={styles.appIcon} />
      <Image source={appName} style={styles.appName} />
    </View>
    <View style={styles.lowerContainer}>
      <TextInput
        style={styles.textinput}
        onChangeText={text => this.setState({ email: text })}
        placeholder={"Enter Email"}
        placeholderTextColor={"#FFFFFF"}
        autoFocus
      />
      <TextInput
        style={[styles.textinput, { marginTop: 20 }]}
        onChangeText={text => this.setState({ password: text })}
        placeholder={"Enter Password"}
        placeholderTextColor={"#FFFFFF"}
        secureTextEntry
      />
      <TouchableOpacity
        style={[styles.button, { marginTop: 20 }]}
        onPress={() => this.handleLogin(email, password)}
      >
        <Text style={styles.buttonText}>Login</Text>
      </TouchableOpacity>
    </View>
  </ImageBackground>
</KeyboardAvoidingView>
```

Check the output and debug the code if needed.	*The student runs the app to check the output.*
Change the security rules for the application. The rule that you will be changing is to allow only the authenticated user to make changes to the data of the database. Do you recall what does **auth.uid!=null** helps us with?	*The student modifies the security rules to allow only authenticated users to access the database.*

Here, **auth.uid!=null** means to allow a person who is authenticated to read and write values in our database. Now, try logging in using an unauthorized email and password to check the output.	

If any unauthorized user tries to log in he/she will see the following output:



Great work today! We have learned to create a login page and add the firebase authentication so that only the registered users can make changes to our data.
If the non-existent user tries to log in, he/she will see a toast message.

Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 5 Mins	
Teacher starts slideshow from: Slide 13 to slide 23	
Activity details	**Solution/Guidelines**
Run the presentation from slide 13 to slide 23. **Following are the WRAP-UP session deliverables:** • **Appreciate the student.** • **Revise the current class activities.** • **Discuss the quizzes.** • **Assignment discussion.**	Discuss with the student the current class activities and the student will ask doubts related to the activities.
Quiz time - Click on in-class quiz	
Question	**Answer**
Which of the following is true for try-catch block? A. The try statement allows you to define a block of code to be tested for errors while it is being executed. B. The catch statement allows you to define a block of code to be executed if an error occurs in the try block. C. The try-catch block is used for exception handling D. All of the above	**D**
Which of the following functions did we use to log in the authenticated user? A. firebase.auth().signInWthEmailAndPassword() B. firebase.auth().logInWthEmailAndPassword() C. firestore.auth().signInWthEmailAndPassword() D. firestore.auth().signUpWthEmailAndPassword()	**A**
Which of the following rules in the database will allow only	**A**

the authenticated user to log in? A. request.auth.uid != null; B. request.time < timestamp.date(2020, 11, 14); C. request.authentication == true D. request.uid == “authenticated”	
End the quiz panel	
FEEDBACK ● **Encourage the student to look at other modes of authentication services firebase provides and read their documentation.**	
Wow! We finally completed our e-library application which is ready to be published. How many hours did it take to create our application?	**ESR:** 8 hours!
Isn't it amazing! What did we learn while creating this application?	**ESR:** ● Use of different React Native Components. ● Creating Tab Navigation and Switch Navigation in our app. ● Scanning QR code ● Lazy loading through FlatLists. ● Creating a database in firestore. ● Querying data from firestore. ● Authenticating our users through email

	sign in.
Amazing! In the next class, we are going to start another application based on another problem statement. Do you want to find out what we will be creating? This time we will be creating a space-related application called the ISS Tracker. We will find out in the next class!!	**ESR:** Yes!
You get a “hats off”. Till next class then. See you. Bye!	*Make sure you have given at least 2 Hats Off during the class for:*   
**** This Project will take only 30 mins to complete. Motivate students to try and finish it immediately after the class.*** **Project Overview** E-RIDE STAGE 8 **Goal of the Project:**	**Note: You can assign the project to the student in class itself by clicking on the Assign Project button which is available under the projects tab.**

In class 75, you learned how to authenticate a user in their app by asking them to enter their username and password. You will be implementing and creating an authentication page in the e-ride app. * This is a continuation of Project-68, 69, 70, 71,72,73,& 74 to make sure you have completed and submitted that before attempting this one. This is the last part of the e-ride. **Story:** The e-ride app is working fantastic! You are receiving great feedback from the users. Now that the app is working great, you would like to work on data security in the app, you proposed to create an Authentication page in the app as well as in the database. This would allow only an authorized user to log in to your app. I am very excited to see your project solution and I know you both will do really well. Bye Bye!	
Teacher ends slideshow	
Teacher Clicks ✖ End Class	
ADDITIONAL ACTIVITIES	
Additional Activities *Encourage the student to write reflection notes in their reflection journal using Markdown.* Use these as guiding questions: • What happened today?	*The student uses the Markdown editor to write her/his reflection as a reflection journal.*

○ Describe what happened ○ 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?	

Links:

Activity	Activity Name	Links
Teacher Activity 1	Previous class code	https://github.com/procodingclass/e-library-75-TA
Teacher Activity 2	Firebase Authentication Document	https://firebase.google.com/docs/auth/web/password-auth
Teacher Activity 3	Teacher Reference	https://github.com/procodingclass/e-library-PRO-C75
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/e-library-C75-SA
Student Activity 2	Firebase Authentication Document	https://firebase.google.com/docs/auth/web/password-auth
Visual Aid Link	VA Link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C75-withcues.html
In-Class Quiz	In-Class quiz doc	https://s3-whjr-curriculum-uploads.whjr.online/809a3bbc-d333-4e42-8069-dd1e9e4e6279.pdf
Project Solution	E-Ride Stage-8	https://github.com/procodingclass/PRO-C75-PROJECT