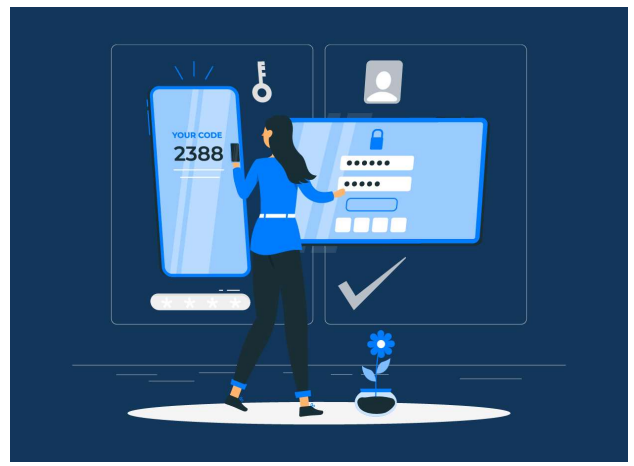


GOOGLE AUTHENTICATION AND DB INTEGRATION



What is our GOAL for this MODULE?

In this class, we continued to build the Storytelling application. We learned to implement Google login and integrate the App with the Firebase database using Firebase Authentication.

What did we ACHIEVE in the class TODAY?

Designed switch navigation to navigate from Loading screen to Login screen to the Dashboard screen.

Implemented Google Login for the App by setting up the credentials in our Firebase App on the Firebase console.

Integrated the App with Firebase using Firebase services for login authentication.

Which CONCEPTS/ CODING BLOCKS did we cover today?

Creating Loading, Login and Dashboard Screens

Function to decide which screen to display based on user's login status

Writing complex functions to implement Google authenticated login

How did we DO the activities?

1. Write code to create switch Navigator and App container using **createSwitchNavigator()** and **createAppContainer()** functions which would contain:
 - Loading Screen
 - Login Screen
 - Dashboard Screen
 - Call switch navigator **AppSwitchNavigator** and navigation container **AppNavigator** in App.js

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

import LoginScreen from "./screens/LoginScreen";
import LoadingScreen from "./screens/LoadingScreen";
import DashboardScreen from "./screens/DashboardScreen";

import * as firebase from "firebase";
import { firebaseConfig } from "./config";

if (!firebase.apps.length) {
  firebase.initializeApp(firebaseConfig);
} else {
  firebase.app();
}

const AppSwitchNavigator = createSwitchNavigator({
  LoadingScreen: LoadingScreen,
  LoginScreen: LoginScreen,
  DashboardScreen: DashboardScreen
});

const AppNavigator = createAppContainer(AppSwitchNavigator);

export default function App() {
  return <AppNavigator />;
}
```

2. Create 3 files LoadingScreen.js, LoginScreen.js and DashboardScreen.js.

LoadingScreen.js

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

export default class LoadingScreen extends Component {
  render() {
    return (
      <View
        style={{
          flex: 1,
          justifyContent: "center",
          alignItems: "center"
        }}>
        <Text>Loading</Text>
      </View>
    )
  }
}
```

LoginScreen.js

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

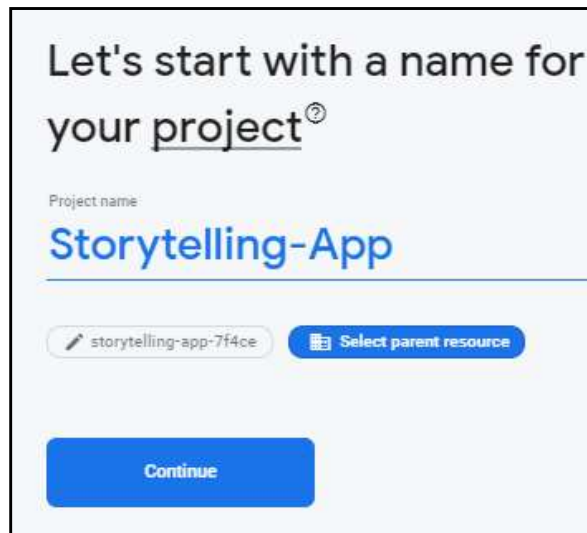
export default class LoginScreen extends Component {
  render() {
    return (
      <View
        style={{
          flex: 1,
          justifyContent: "center",
          alignItems: "center"
        }}>
        <Text>LoginScreen</Text>
      </View>
    )
  }
}
```

```
)  
}  
}
```

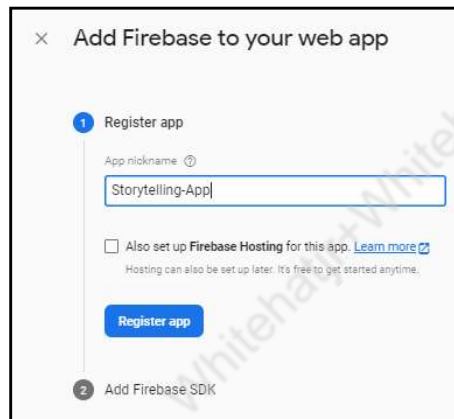
DashboardScreen.js

```
import React, { Component } from 'react';  
import { Text, View } from 'react-native';  
  
export default class DashboardScreen extends Component {  
  render() {  
    return (  
      <View  
        style={{  
          flex: 1,  
          justifyContent: "center",  
          alignItems: "center"  
        }}>  
        <Text>DashboardScreen</Text>  
      </View>  
    )  
  }  
}
```

3. Create a new Firebase project and create configuration keys and save them in the config.js file of the project to use this database.
 - Create Firebase project.



- Create App and configuration keys and copy the SDK code from step 2; (it will be visible once you click on Register App).



4. Save **config.js** file in **.gitignore** so that our Firebase project will not be blocked by Google, and paste SDK code in config.js.

```
export const firebaseConfig = {  
  apiKey: "AIzaSyDce_gGywAiuJEftp4Ccbt9odCV5y7rZiI",  
  authDomain: "storytelling-app-cab54.firebaseio.com",  
  projectId: "storytelling-app-cab54",  
  storageBucket: "storytelling-app-cab54.appspot.com",  
  messagingSenderId: "843153669971",  
  appId: "1:843153669971:web:05101931886d9498266ba6"  
};
```

5. Import **firebase** and **firebaseConfig** into **App.js** and write an **if** statement to initialize firebase.

```
4 import LoginScreen from "../screens/LoginScreen";
5 import LoadingScreen from "../screens/LoadingScreen";
6 import DashboardScreen from "../screens/DashboardScreen";
7
8 import * as firebase from "firebase";
9 import { firebaseConfig } from "../config";
10
11 if (!firebase.apps.length) {
12   firebase.initializeApp(firebaseConfig);
13 } else {
14   firebase.app();
15 }
16
```

6. Write function **checkIfLoggedIn()** and call it in **componentDidMount()** in **loginScreen.js** to load the dashboard screen if the user is already logged in otherwise display the login screen.

Use **firebase.auth ().onAuthStateChanged ()** to check user exists or not

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

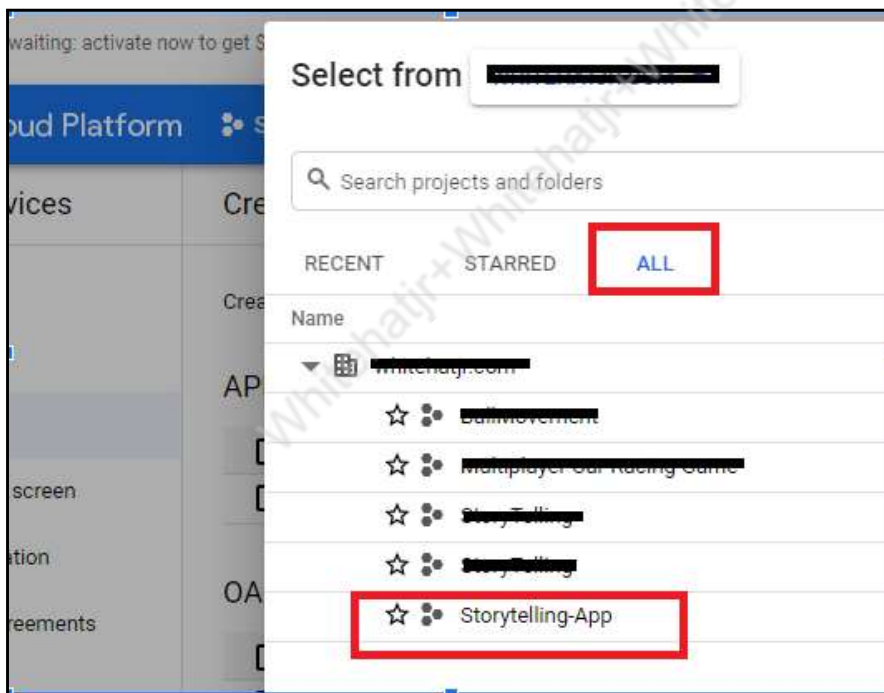
```
export default class LoadingScreen extends Component {

  componentDidMount() {
    this.checkIfLoggedIn()
  }

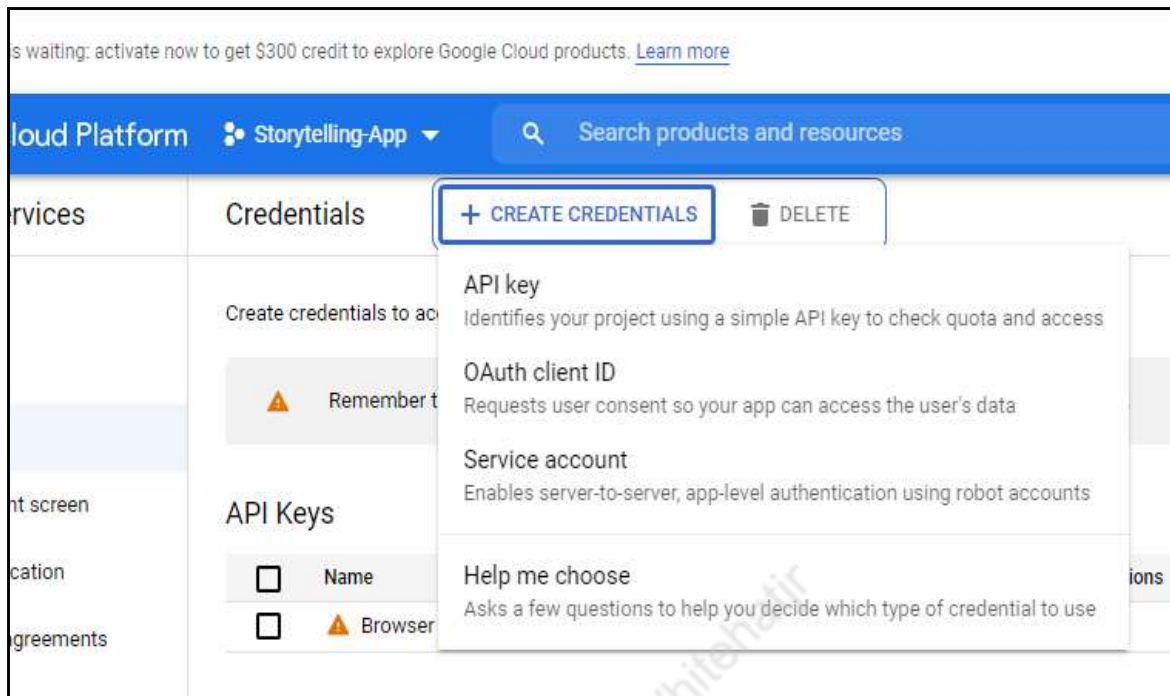
  checkIfLoggedIn = () => {
    firebase.auth().onAuthStateChanged((user) => {
      if (user) {
        this.props.navigation.navigate('DashboardScreen')
      } else {
        this.props.navigation.navigate('LoginScreen')
      }
    })
  }

  render() {
    return (
      <View
        style={{
          flex: 1,
          justifyContent: "center",
          alignItems: "center"
        }}
      >
        <ActivityIndicator size="large" />
      </View>
    )
  }
}
```

7. Implement Google login using [expo's documentation](#) for Google Sign in.
Click on the Credentials page.
Select the Firebase project "Storytelling-App".



8. Create the credentials for the app.

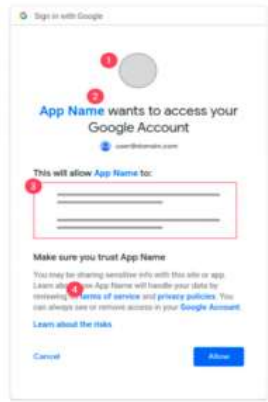


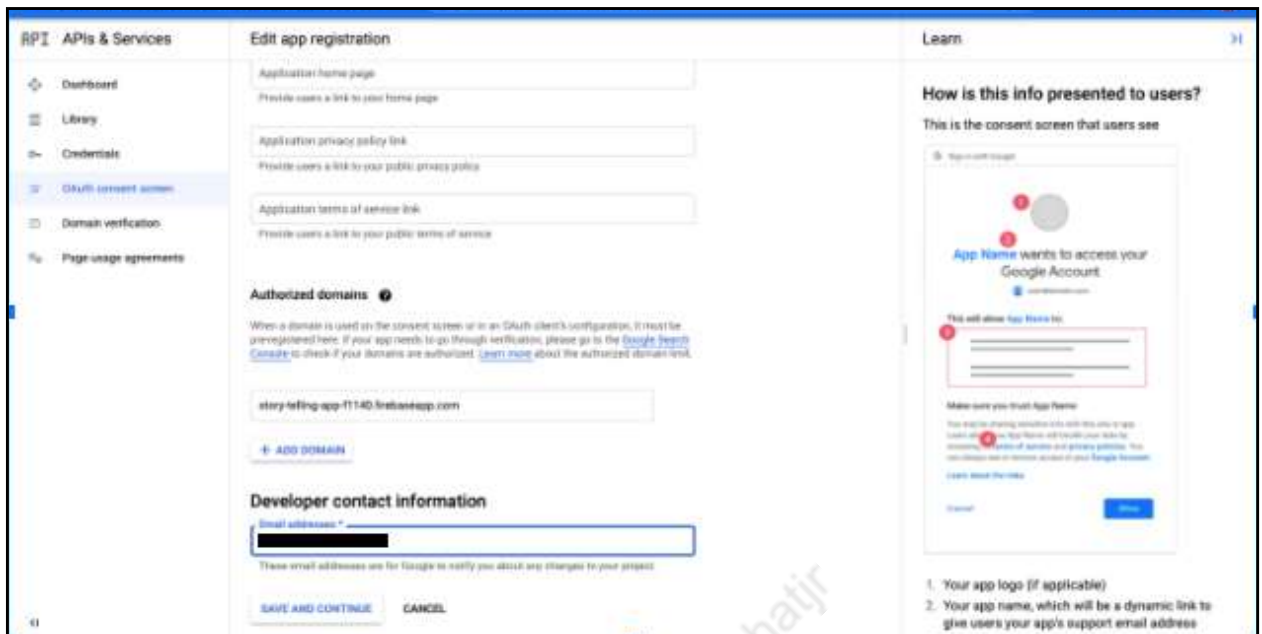
9. Create **OAuth client ID** and configure the consent screen by selecting **External** option.



API	APIs & Services	OAuth consent screen	Learn
Dashboard	Choose how you want to configure and register your app, including your target users. You can only associate one app with your project.	<p>User Type</p> <p><input type="radio"/> Internal ⓘ</p> <p>Only available to users within your organization. You will not need to submit your app for verification.</p> <p><input checked="" type="radio"/> External ⓘ</p> <p>Available to any test user with a Google Account. Your app will start in testing mode and will only be available to users you add to the list of test users. Once your app is ready to push to production, you may need to verify your app.</p> <p>CREATE</p> <p>Let us know what you think about our OAuth experience</p>	<p>Google OAuth consent screen</p> <p>What is the OAuth consent screen? ▾</p> <p>What are OAuth consent scopes? ▾</p> <p>What are sensitive API scopes? ▾</p> <p>What are restricted API scopes? ▾</p> <p>The app registration process</p> <p>What information do I need? ▾</p> <p>Will my app need to be verified by Google? ▾</p> <p>What if I don't verify my app? ▾</p> <p>How long does the verification process take? ▾</p> <p>How many users can use my app? ▾</p> <p>Domain verification ▾</p>

10. Enter your email ID for user support along with our developer information.

API	APIs & Services	Edit app registration	Learn
Dashboard	<p>1 OAuth consent screen — 2 Scopes — 3 Optional info — 4 Summary</p> <p>App information</p> <p>This shows in the consent screen, and helps end users know who you are and contact you</p> <p>App name *</p> <p>project-72696421845</p> <p>The name of the app asking for consent</p> <p>User support email *</p> <p>apoorvelous@gmail.com</p> <p>For users to contact you with questions about their consent</p> <p>App logo BROWSE</p> <p>Upload an image, not larger than 1MB on the consent screen that will help users recognize your app. Allowed image formats are JPG, PNG, and BMP. Logos should be square and 120px by 120px for the best results.</p> <p>App domain</p> <p>To protect you and your users, Google only allows apps using OAuth to use Authorized Domains. The following information will be shown to your users on the consent screen.</p> <p><input type="text"/></p>	<p>How is this info presented to users?</p> <p>This is the consent screen that users see</p>  <p>1 Your app logo (if applicable)</p> <p>2 Your app name, which will be a dynamic link to give users your app's support email address</p> <p>3 The data you are requesting, or scopes, which you</p>	



API APIs & Services

Edit app registration

Application home page
Provide users a link to your home page

Application privacy policy link
Provide users a link to your public privacy policy

Application terms of service link
Provide users a link to your public terms of service

Authorized domains

When a domain is used on the consent screen or in an OAuth client's configuration, it must be pre-registered here. If your app needs to go through verification, please go to the [Google Search Console](#) to check if your domains are authorized. [Learn more](#) about the authorized domain tool.

story-telling-app-f1140.firebaseio.com

[+ ADD DOMAIN](#)

Developer contact information

Email addresses *

These email addresses are for Google to notify you about any changes to your project.

[SAVE AND CONTINUE](#) [CANCEL](#)

Learn

How is this info presented to users?

This is the consent screen that users see

[App consent screen](#)

App Name wants to access your Google Account

This will allow App Name to:

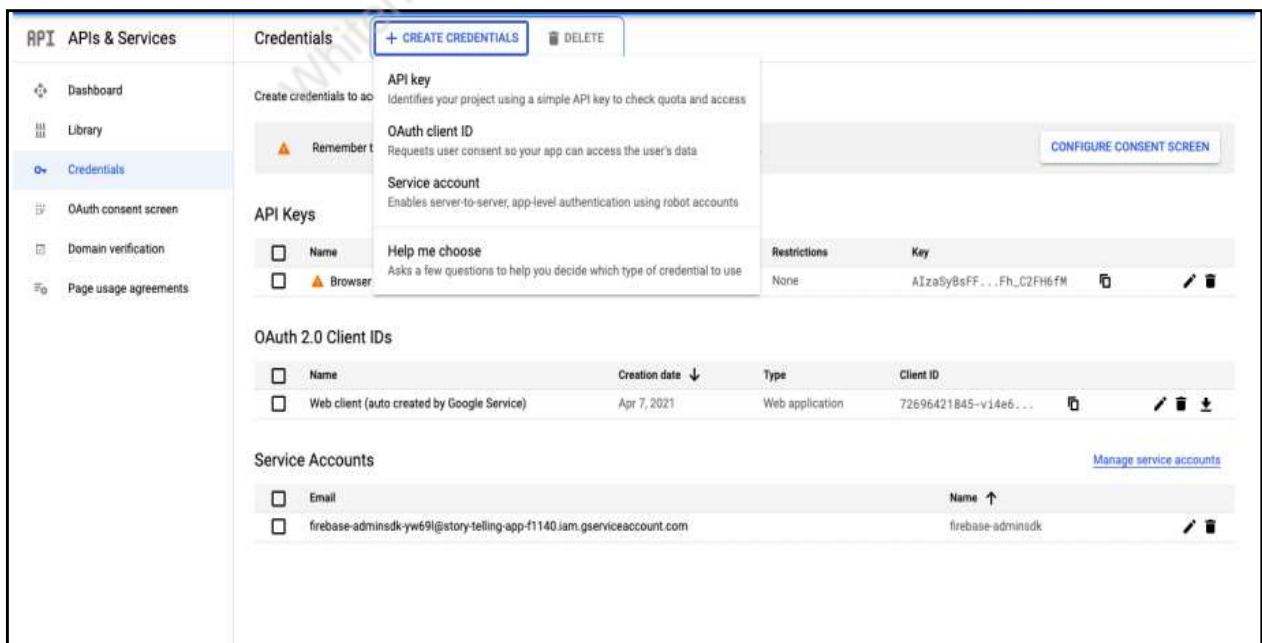
Make sure you trust App Name:

You should be sharing sensitive info with this app to help it work properly. App Name will handle your data for you. [Learn more about App Name's privacy policy.](#) This information will be shared with other Google services.

[Cancel](#) [Allow](#)

1. Your app logo (if applicable)
2. Your app name, which will be a dynamic link to give users your app's support email address

11. Go back to the dashboard and click on OAuth Client ID from Create Credentials.



API APIs & Services

Credentials [+ CREATE CREDENTIALS](#) [DELETE](#)

Create credentials to use

Remember to

API Keys

☐ Name

☐ [Browser](#)

Help me choose

Asks a few questions to help you decide which type of credential to use

OAuth 2.0 Client IDs

Name	Creation date	Type	Client ID
<input type="checkbox"/> Web client (auto created by Google Service)	Apr 7, 2021	Web application	72696421845-v14e6...

Service Accounts [Manage service accounts](#)

Email	Name
<input type="checkbox"/> firebase-adminsdk-yw69l@story-telling-app-f1140.iam.gserviceaccount.com	firebase-adminsdk

12. Follow the instructions to generate OAuth IDs for both Android and iOS.

- **Create an iOS OAuth Client ID**
 - Select "iOS Application" as the Application Type. Give it a name if you want (e.g. "iOS Development").
 - Use `host.exp.exponent` as the bundle identifier.
 - Click "Create"
 - You will now see a modal with the client ID.
 - The client ID is used in the `iosClientId` option for `Google.loginAsync` (see code example below).
- **Create an Android OAuth Client ID**
 - Select "Android Application" as the Application Type. Give it a name if you want (maybe "Android Development").
 - Run `openssl rand -base64 32 | openssl sha1 -c` in your terminal, it will output a string that looks like `A1:B2:C3` but longer. Copy the output to your clipboard.
 - Paste the output from the previous step into the "Signing-certificate fingerprint" text field.
 - Use `host.exp.exponent` as the "Package name".
 - Click "Create"
 - You will now see a modal with the Client ID.
 - The client ID is used in the `androidClientId` option for `Google.loginAsync` (see code example below).

Here are it's installation steps -

To check if OpenSSL is installed in your system or not, run the following in a terminal / command prompt -

`openssl --version`

If it is not installed, which is highly unlikely, then see the following steps -

MacOS -

In a new terminal, run the following command -

`export ACL_OPENSSL_VERSION=11`

After that, run the following command -

`brew install openssl`

Ubuntu -

Run the following commands in a new Terminal -

`sudo apt-get install libssl-dev`

Windows -

Navigate to **`C:\Program Files\Git\usr\bin`**

Run the command - **`openssl rand -base64 32 | openssl sha1 -c`**

This will generate device fingerprint required to create OAuth ID for Android.

13. Add a **signInWithGoogleAsync()** function in login screen class, and update **androidClientId** and **iosClientId**.

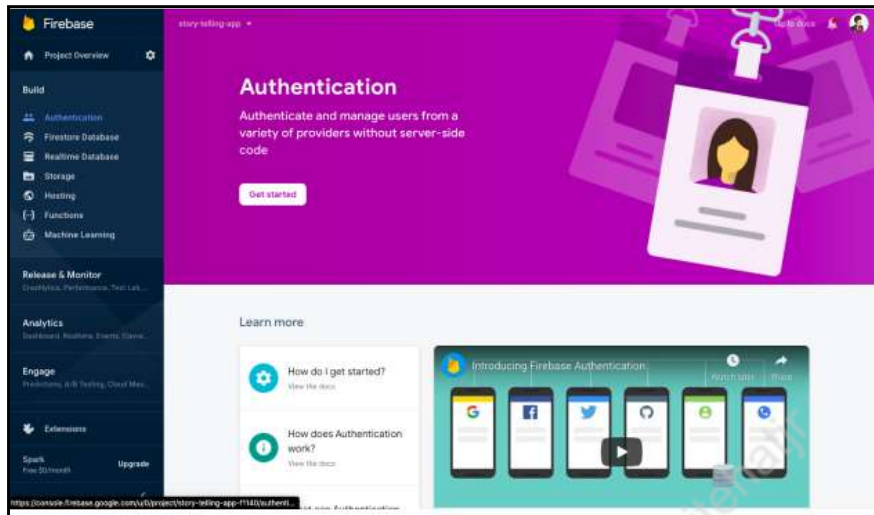
```
import React, { Component } from 'react';
import { View, Button } from 'react-native';

export default class LoginScreen extends Component {
  signInWithGoogleAsync = async () => {
    try {
      const result = await Google.logInAsync({
        behaviour: "web",
        androidClientId: "512716633783-
rkrc8tpbrknrpnt0akjbqqo0v4ukafkc.apps.googleusercontent.com",
        iosClientId: "512716633783-
n6qco4smqeh9g64ffalebfrsepq0nvlf.apps.googleusercontent.com",
        scopes: ['profile', 'email'],
      });
    }
  };
}
```

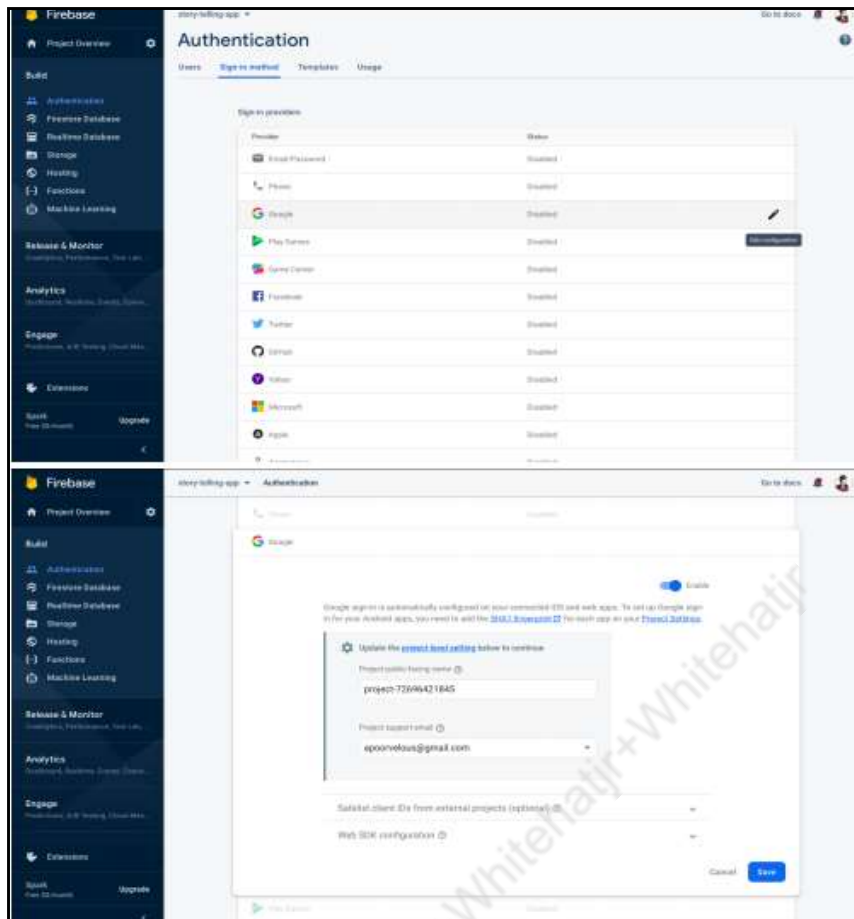
14. Create a button in the Login screen. On pressing on this button call **signInWithGoogleAsync()** function.

```
render() {
  return (
    <View
      style={{
        flex: 1,
        justifyContent: "center",
        alignItems: "center"
      }}>
      <Button
        title="Sign in with Google"
        onPress={() => this.signInWithGoogleAsync()}></Button>
    </View>
  )
}
```

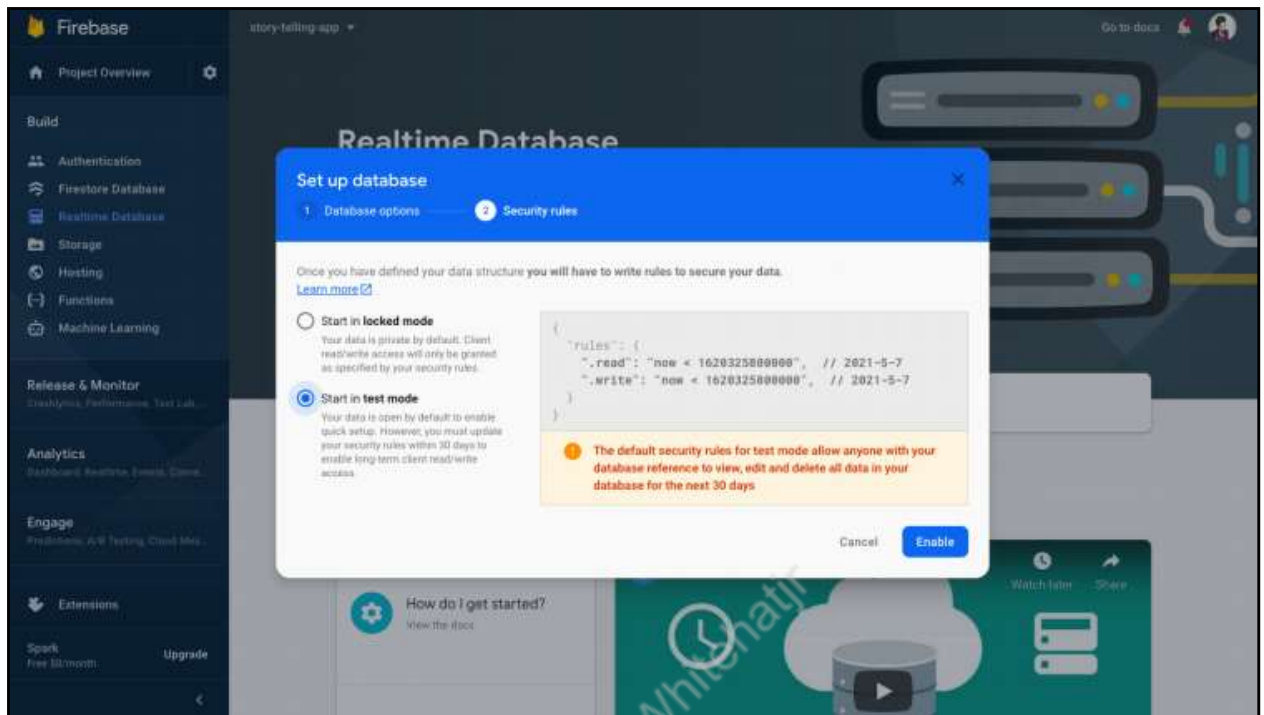
15. Fetch the user's data from the database and change the settings in Authentication of Firebase.



16. Enable Google Sign In.



17. Create a Realtime Database in test mode.



18. Install **expo install expo-google-auth** dependency.

19. Write code for function **OnSignIn** .

- Check if the App is already signed in with the correct user using **firebase.auth().onAuthStateChanged()**.
- Use **isUserEqual()** to check the same user signed in.
- Create the credentials for the user so they can use our app as a signed in user with **firebase.auth().GoogleAuthProvider.credential()** function. For setting up the credential, use the **id_token** and the **access_token** of our **googleUserSign** in with credentials from the Google user.

```
onSignIn = googleUser => {
  console.log('Google Auth Response', googleUser);
  // We need to register an Observer on Firebase Auth to make sure auth is
  initialized.
  var unsubscribe = firebase.auth().onAuthStateChanged((firebaseUser) => {
    unsubscribe();
    // Check if we are already signed-in Firebase with the correct user.
    if (!this.isUserEqual(googleUser, firebaseUser)) {
      // Build Firebase credential with the Google ID token.
      var credential = firebase.auth.GoogleAuthProvider.credential(
```



```

    googleUser.idToken,
    googleUser.accessToken
  );

  // Sign in with credential from the Google user.
  firebase
    .auth()
    .signInWithCredential(credential)
    .then(function (result) {
      if (result.additionalUserInfo.isNewUser) {
        firebase
          .database()
          .ref("/users/" + result.user.uid)
          .set({
            gmail: result.user.email,
            profile_picture: result.additionalUserInfo.profile.picture,
            locale: result.additionalUserInfo.profile.locale,
            first_name: result.additionalUserInfo.profile.given_name,
            last_name: result.additionalUserInfo.profile.family_name,
            current_theme: "dark"
          })
          .then(function (snapshot) {

          })
        }
      })
    .catch((error) => {
      // Handle Errors here.
      var errorCode = error.code;
      var errorMessage = error.message;
      // The email of the user's account used.
      var email = error.email;
      // The firebase.auth.AuthCredential type that was used.
      var credential = error.credential;
      // ...
    });
  } else {
    console.log("User already signed-in Firebase.");
  }
});

```

```

    }

    signInWithGoogleAsync = async () => {
      try {
        const result = await Google.signInAsync({
          behaviour: "web",
          androidClientId: "512716633783-
rkr8tpbrknrpnt0akjbqqo0v4ukafkc.apps.googleusercontent.com",
          iosClientId: "512716633783-
n6qco4smqeh9g64ffalebfrsepq0nvlf.apps.googleusercontent.com",
          scopes: ['profile', 'email'],
        });

        if (result.type === 'success') {
          this.signIn(result)
          return result.accessToken;
        } else {
          return { cancelled: true };
        }
      } catch (e) {
        return { error: true };
      }
    }

    render() {
      return (
        <View
          style={{
            flex: 1,
            justifyContent: "center",
            alignItems: "center"
          }}>
          <Button
            title="Sign in with Google"
            onPress={() => this.signInWithGoogleAsync()}></Button>
        </View>
      )
    }
  }
}

```

20. Sign in the user with the Firebase authentication system using **firebase.auth().signInWithCredential()** function.

Use a **.then()** function on it in which we are checking if the user is a new user or not with **result.additionalUserInfo.isNewUser**.

```
// Sign in with credential from the Google user.
firebase
  .auth()
  .signInWithCredential(credential)
  .then(function (result) {
    if (result.additionalUserInfo.isNewUser) {
      firebase
        .database()
        .ref("/users/" + result.user.uid)
        .set({
          gmail: result.user.email,
          profile_picture: result.additionalUserInfo.profile.picture,
          locale: result.additionalUserInfo.profile.locale,
          first_name: result.additionalUserInfo.profile.given_name,
          last_name: result.additionalUserInfo.profile.family_name,
          current_theme: "dark"
        })
        .then(function (snapshot) { });
    }
  })
```

21. Call **onSignIn()** function inside **signInWithGoogleAsync ()** function on finding the user. Use **Drawer Navigation** in the Dashboard screen to display the main screen.

```
import * as React from "react";
import { NavigationContainer } from "@react-navigation/native";
import DrawerNavigator from "../navigation/DrawerNavigator";

export default function DashboardScreen() {
  return (
    <NavigationContainer>
      <DrawerNavigator />
    </NavigationContainer>
  );
}
```

```
);  
}
```

22. Test the App by running the app on Mobile and login into it using a Gmail Id.. It should register the Email ID in the database.



OUTPUT:



What's next?

In the next class, we will complete the UI of the Login screen and allow the user to set the theme for the profile screen.

Expand your knowledge:

1. To explore more options for creating Authentication in Firebase:
<https://firebase.google.com/docs/auth?authuser=0>

Whitehatjr+Whitehatjr+Whitehatjr