Topic	GOOGLE AUTHENTICATION AND DB INTEGRATION	
Class Description	**In this class, the students will be implementing Google Authentication and integrate the app with Firebase.**	
Class	**C85**	
Class time	**55 mins**	
Goal	● Using Google Authentication to authenticate the users. ● Integrating the Firebase database with 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-Student Collaborative Activity** **Wrap-Up** **This class requires database configuration and SSL installation. Request students to live share VSC and perform activities to avoid writing the same code twice at both ends.*	**5 mins** **45 mins** **5 mins**
Credit:	**Code samples used for Firebase-Google Authentication are licensed under the [Apache 2.0 License](#).** **Expo documentation used from - https://expo.io**	

WARM-UP SESSION - 5 mins	
Teacher starts slideshow from slides 1 to 12 Refer to speaker notes and follow the instructions on each slide.	
Activity details	**Solution/Guidelines**
Run the presentation from slide 1 to slide 4. **The following are the warm-up session deliverables:** ● Revision ● Warm-Up Quiz Session	Click on the slide show tab and present the slides.
QnA Session	
Question	**Answer**
Which function is used to stop the speech from playing? A. Speech.end() B. Speech.stop() C. Speech.over() D. Speech.terminate()	**B**
Since we've passed the story from one screen to another through navigation and not from the parent component to the child component directly, we need to use_____ A. this.props.story B. this.props.navigation.navigate C. this.props.route.params D. this.props.params	**C**

Continue the warm-up session	
Activity details	**Solution/Guidelines**
Run the presentation from slide 4 to slide 12 to set the problem statement. **The following are the warm-up session deliverables:** • Introduce students to the coding environment - Workspace, blocks, and output. • Steps to write and run the code. • Introduce the concepts of Teacher-led Activity.	Narrate the story by using hand gestures and voice modulation methods to bring in more student interest.
• **Teacher ends slideshow**	
• **Teacher-Student Collaborative Activity - 45 mins**	
Teacher Initiates Screen Share	
<u>CHALLENGE</u> • **Adding Google Authentication to authenticate and Login the user.** • **Integrating the Firebase database to the App.**	

Step 2: Teacher-led Activity (15 min)	Since Google Authentication is a completely new concept, it might look a bit tricky in the beginning but it's not. For this class, we will be implementing the Google Login collaboratively. *(Ask the student to observe closely as all the changes should be made on both, the student's and teacher's codes.)*	

	(There are no separate Teacher and Student activities in this class.) ***Note -*** *If the student and/or teacher is using the snack editor for these classes, please refer to the support document in Teacher Activity 6*	
	Since most of our steps would involve dealing with steps to enable Google authentication, we will use boilerplate code for this class. *Teacher refers to Teacher Activity 4 and clones the boilerplate code.* Let's start by installing Firebase and react-navigation- *yarn add firebase@^8.2.10* *yarn add react-navigation* *(Teacher installs Firebase and react-navigation.)* Remember that up until this point, we have installed specific navigation - Tab, Drawer and Stack but this time, we are installing the entire react-navigation dependency. Now the idea for implementing Google Login is that we have 3 parts - 1. The Login screen where the user will Login from. 2. The loading screen, while the user is logging in.	*Student refers to Student Activity 4 and clones the boilerplate code.* *Student installs Firebase and react-navigation.*

	3. The dashboard screen (or the Feed screen in our case) that the user will see once they are logged in. For this, we will be using the Switch Navigator. How many navigation methods have we implemented in our app so far? We already have the code for this provided to us in the boilerplate code that we just coded. Let's quickly go through it. Inside the file ***App.js -***	ESR: 3 navigations - Stack, Drawer and Tab.

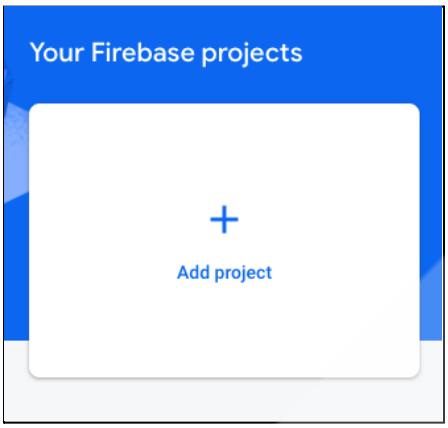
```
import * as React from "react";
import { createSwitchNavigator, createAppContainer } from "react-navigation";

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 />;
}
```

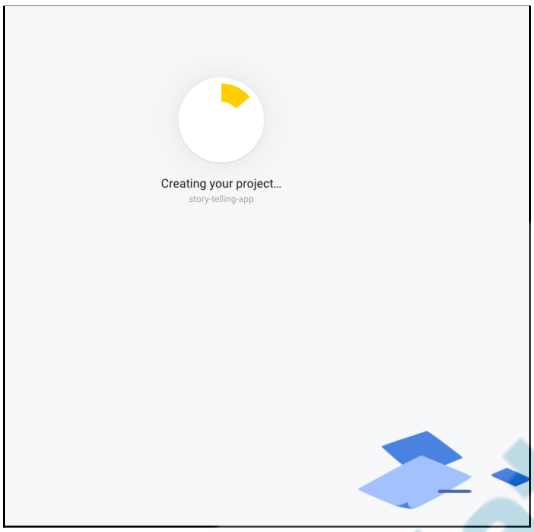
Teacher explains the code to the student.

Here, we are importing the ***createSwitchNavigator*** *and* ***createAppContainer*** components from the **react-navigation** and we are creating our Switch Navigator and App Navigator with the functions.

If you remember, we discussed that it is important to have our navigation in the **<NavigationContainer>** component. Here, our **createAppContainer** component does that for us and wraps our navigator inside a container.

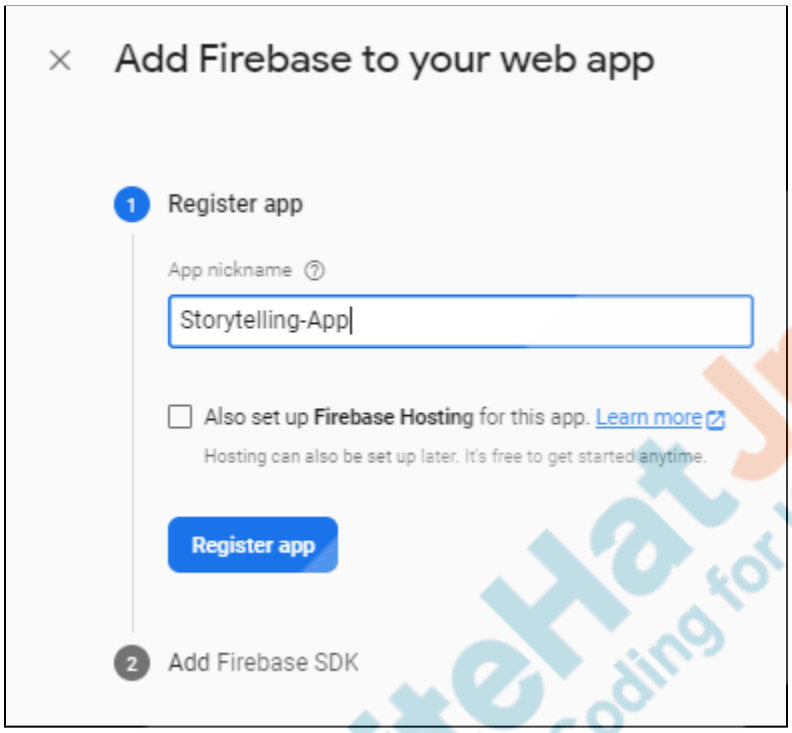
In the Switch Navigator, we are using **LoadingScreen, LoginScreen and DashboardScreen.** Note that we don't have these screens yet, so we'll create them next. Ignore the code for the Firebase in this file right now. We will go over it again once we have the firebase database in place.		
	Now let's see in our ***screens*** folder. There should be 3 files, that we just discussed - 1. **LoadingScreen.js** 2. **LoginScreen.js** 3. **DashboardScreen.js** All these 3 files have some boilerplate code added in them, such that the **LoadingScreen** will decide if the user should go to the ***DashboardScreen*** or the ***LoginScreen***.	*Student observes.*
	Now we have already installed Firebase into our project. Let's set up a new Firebase DB. *Teacher refers to Teacher Activity 1* *Teacher tells the student to open the Firebase console.* Here, let's create a new project by clicking on the **Add project** button. *Teacher creates a new Firebase project.*	*Student refers to Student Activity 1.* *Student creates a new Firebase project.*





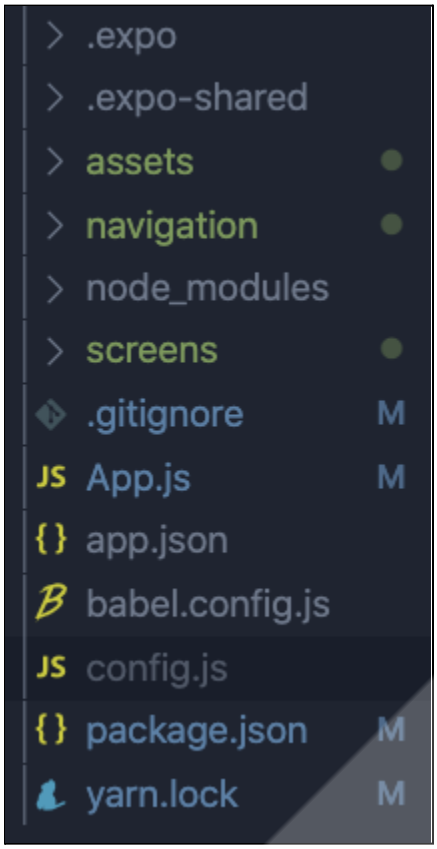
Great! Now that we have our project ready, we need to create our configuration keys to use this database.

For that, we will click on the web button, register our app by giving it a name and copy the config keys from there -



Copy the SDK code from step 2; *(It will be visible once you click on Register App.)*

	Now, to save these config keys, let's create a new file ***config.js*** in our root folder of the project. Also don't forget to enter the ***config.js*** file in ***.gitignore***, or else your Firebase project will be blocked. It is a poor practice in software development to expose your authentication keys on github, and opens a door for hackers to access and view sensitive information from your database. *Teacher copies the config in **config.js** and adds the filename in **.gitignore.***	*Student copies the config in **config.js** and adds the file name in **.gitignore.***



```
85t > .gitignore
*.jks
*.p8
*.p12
*.key
*.mobileprovision
*.orig.*
web-build/

# macOS
.DS_Store

config.js
```

Create config.js

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

Here, please note that we are using **export const** for our config keys, since we want to export it as a constant in our app.

	Great! Now our Firebase database will be available to our app. Now let's refer back to our ***App.js***	

	We have a few lines of code where we were importing Firebase.	

```
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();
}
```

We are importing the Firebase database and our config and initializing the app with it. We have the **if-else** condition to check if we already have the Firebase app initialized. If not, we are initializing the Firebase app otherwise, we are using the already initialized app.

	Now we are pretty much halfway through. We place our Loading Screen as the first screen in our switch navigator. Do you know why?	**ESR:** Varied.
	We did it because we want to first check if a user is already logged in or not. If they are already logged in, we don't want to take them to the Login screen again but instead to the dashboard screen.	

	If, however, they are not logged in, we want to take them to the Login screen. This is the reason why we have the Loading Screen as our first screen. We will be checking it on our loading screen. Let's refer to the code provided in our ***LoadingScreen.js*** and over it to understand how we achieved the functionality we talked about -	

Import statements -

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

LoadingScreen component -

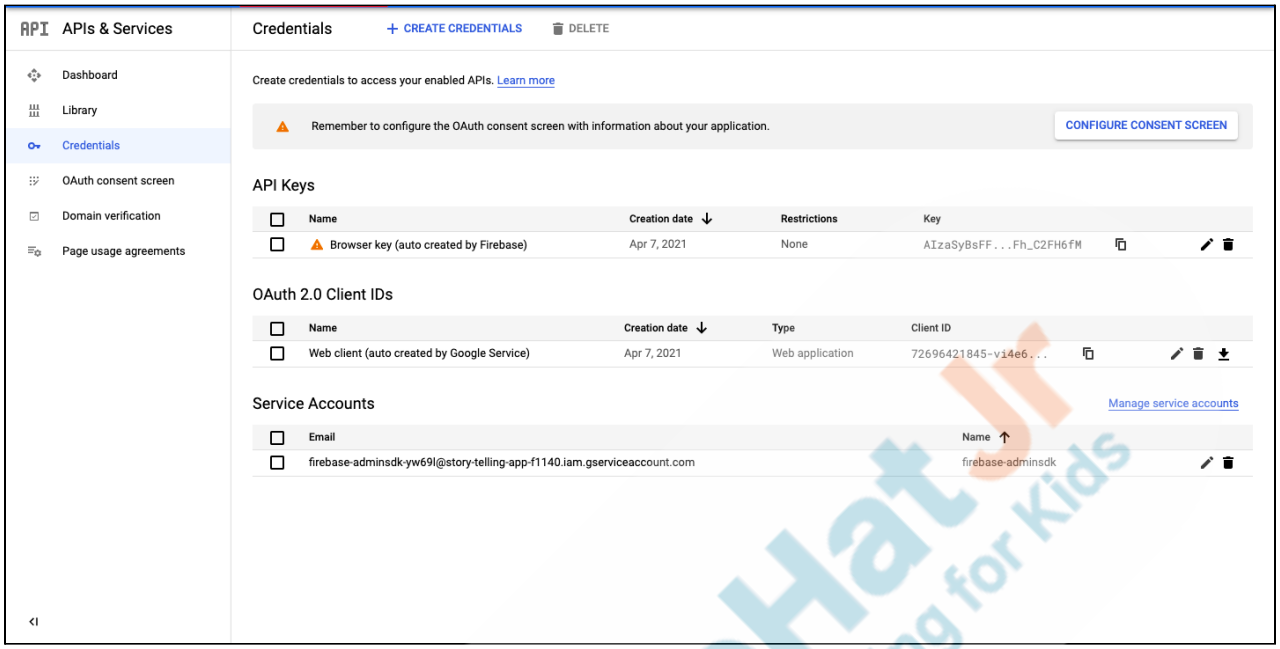
```
export default class LoadingScreen extends Component {
  componentDidMount() {
    this.checkIfLoggedIn();
  }

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

  render() {
    return (
      <View style={styles.container}>
        <Text>Loading</Text>
      </View>
    );
  }
}
```

Teacher explains the code to the student

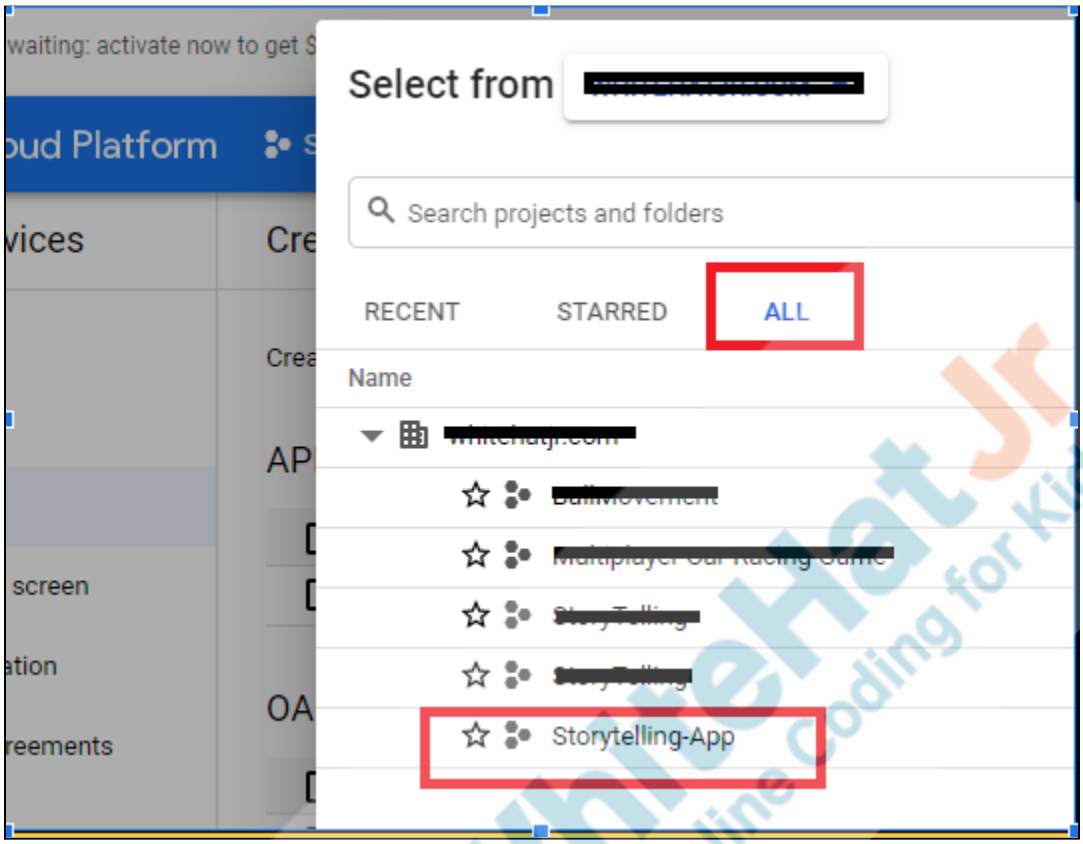
We have now imported Firebase into our loading screen and we have created a function **checkedIfLoggedIn().** We are calling this function in **componentDidMount()**.

Now **firebase.auth().onAuthStateChanged()** returns the user that has logged in. Inside this function, we are checking if the **user** exists or not.

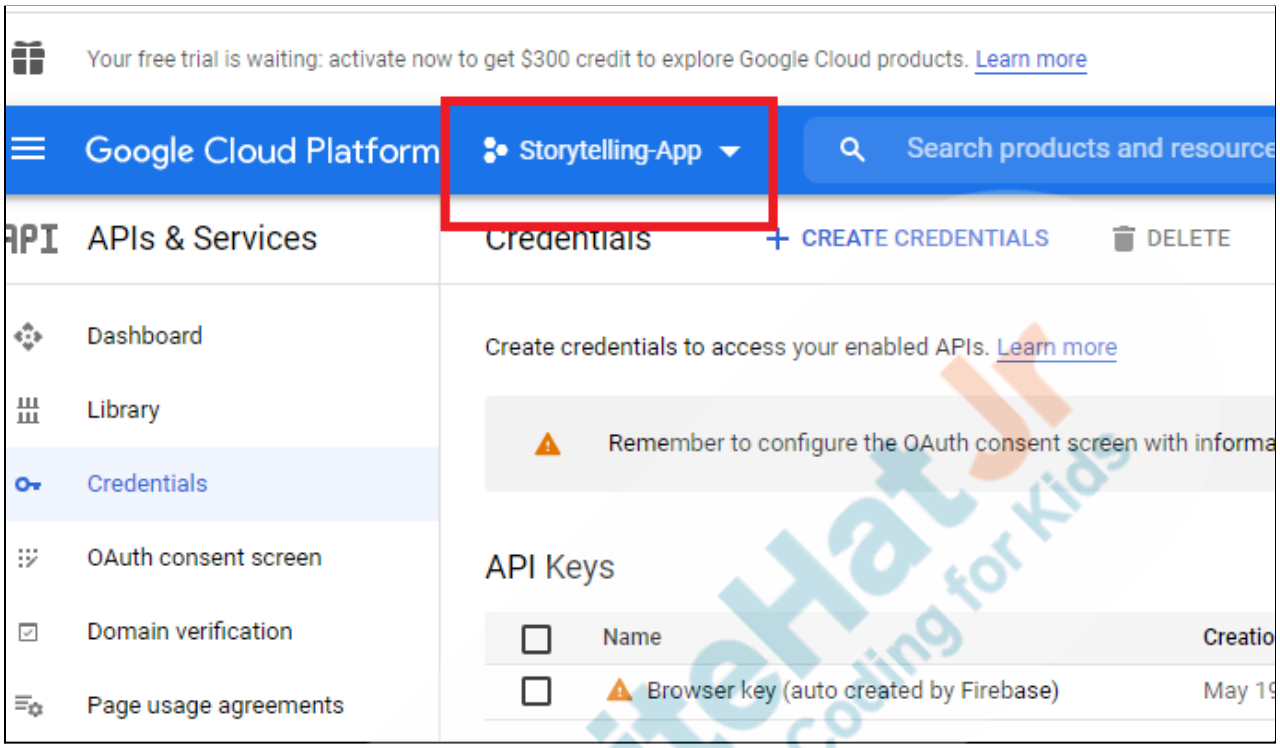
If we find the user, we are navigating them to the **DashboardScreen**, else we are navigating them to the **LoginScreen**.		
	With this, we have a lot of our functionality ready. If we open the app now, we will see that instead of the **LoadingScreen,** we are navigated to the **LoginScreen** by default!	
	Now comes the part where we will be implementing our Google Login. Let's take a look at the expo's documentation for Google Sign in - *Teacher refers to Teacher Activity 2.*	*Student refers to Student Activity 2.*
	If we take a look at the page, it tells us that we need to set up credentials for our specific app. For that, let's go to the credentials page. *Teacher clicks on the credential page.*	*Student follows the instructions.*
1. We will click on the Credentials Page.		



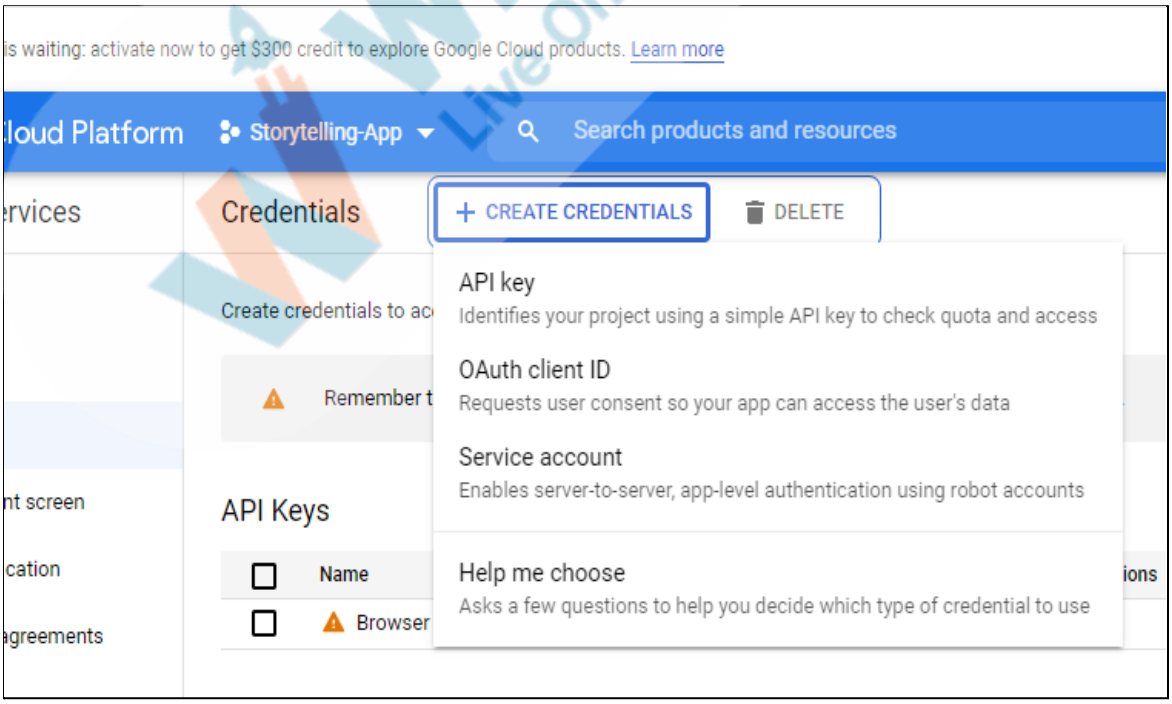
2. We'll click on the project to select the project (I already have the project selected but you might see some other project or nothing at all).



3. We go on all in the modal box that appears and select our project.



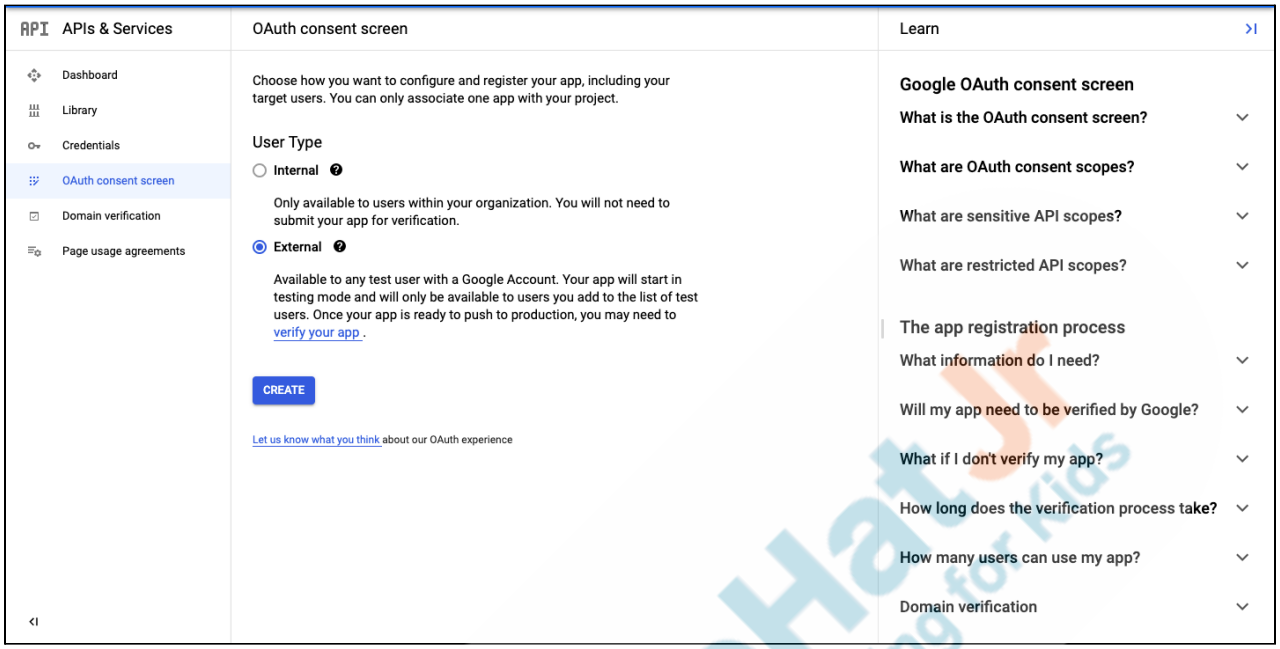
4. We get back to this page with our project selected and click on create credential



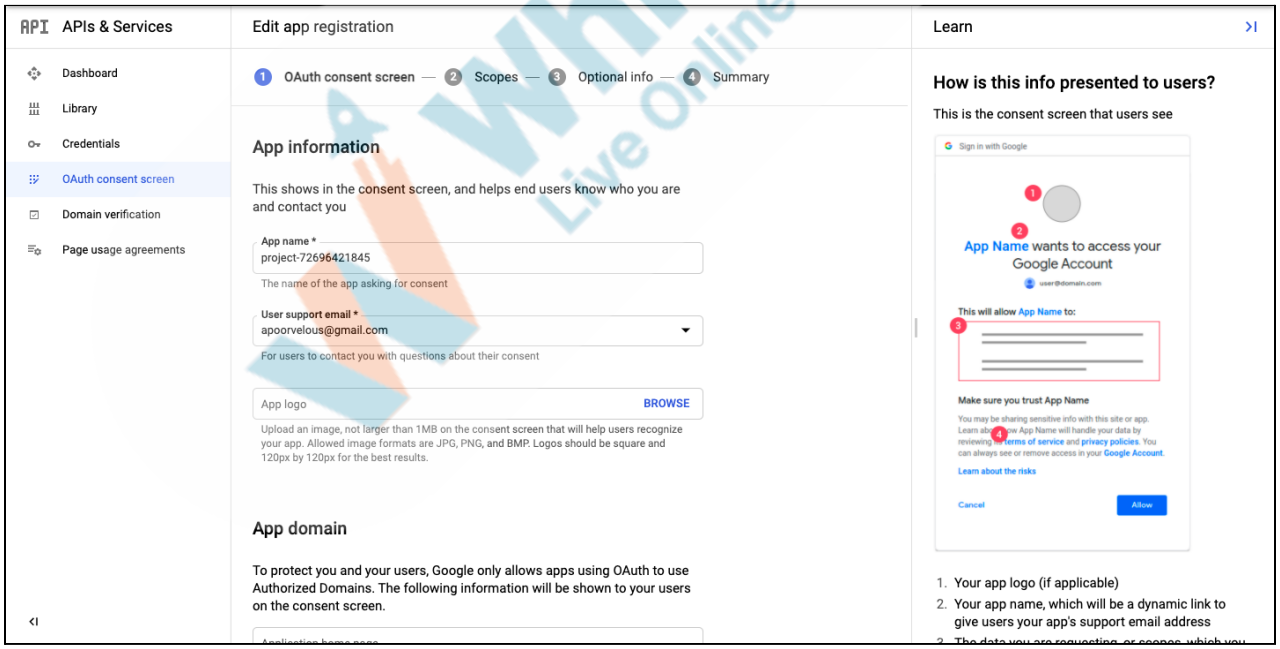
	Okay, now we are ready to create the credentials for our app.	*Student follows the teacher.*

5. We'll click on the Create Credentials button and then click on OAuth Client ID. It will prompt us to the following page -

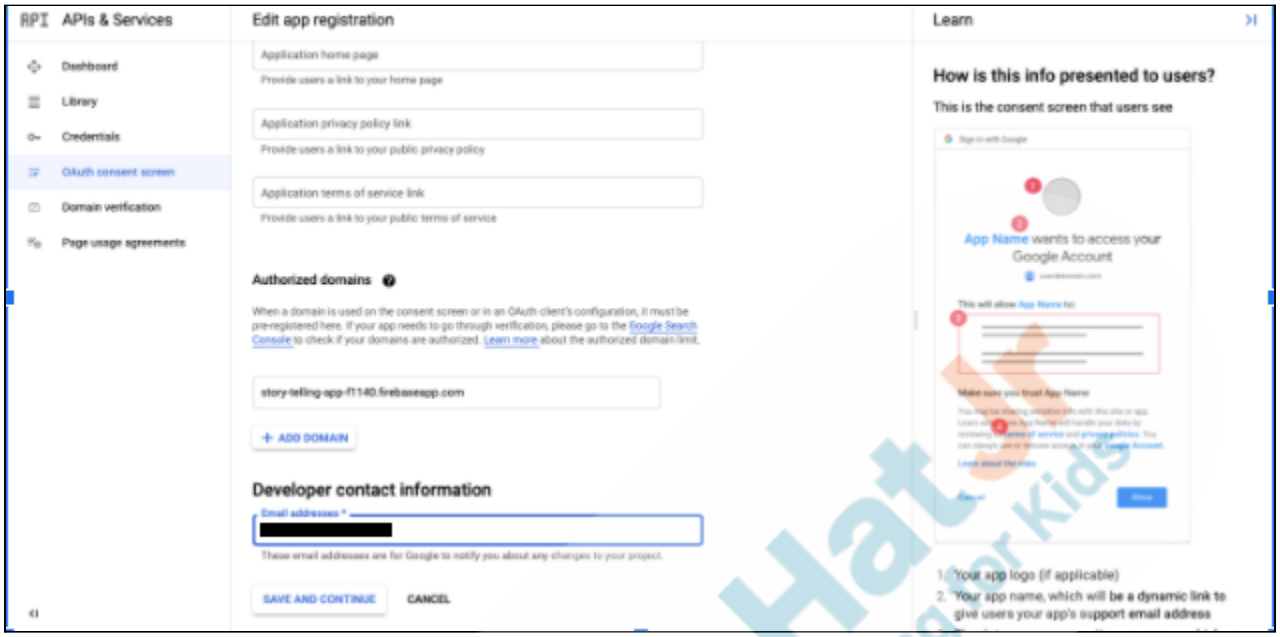
6. Here, we will first need to configure our Consent Screen. Let's click on the blue button to configure it -



7. We will select "**External**" and click on Create.

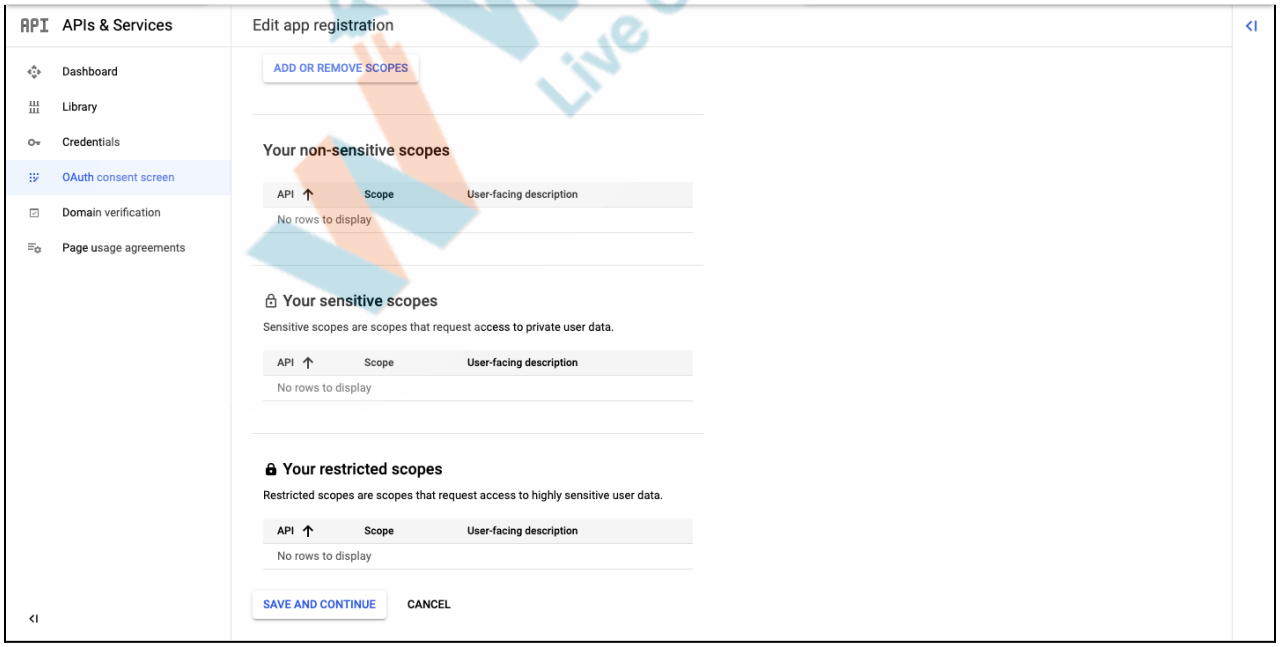


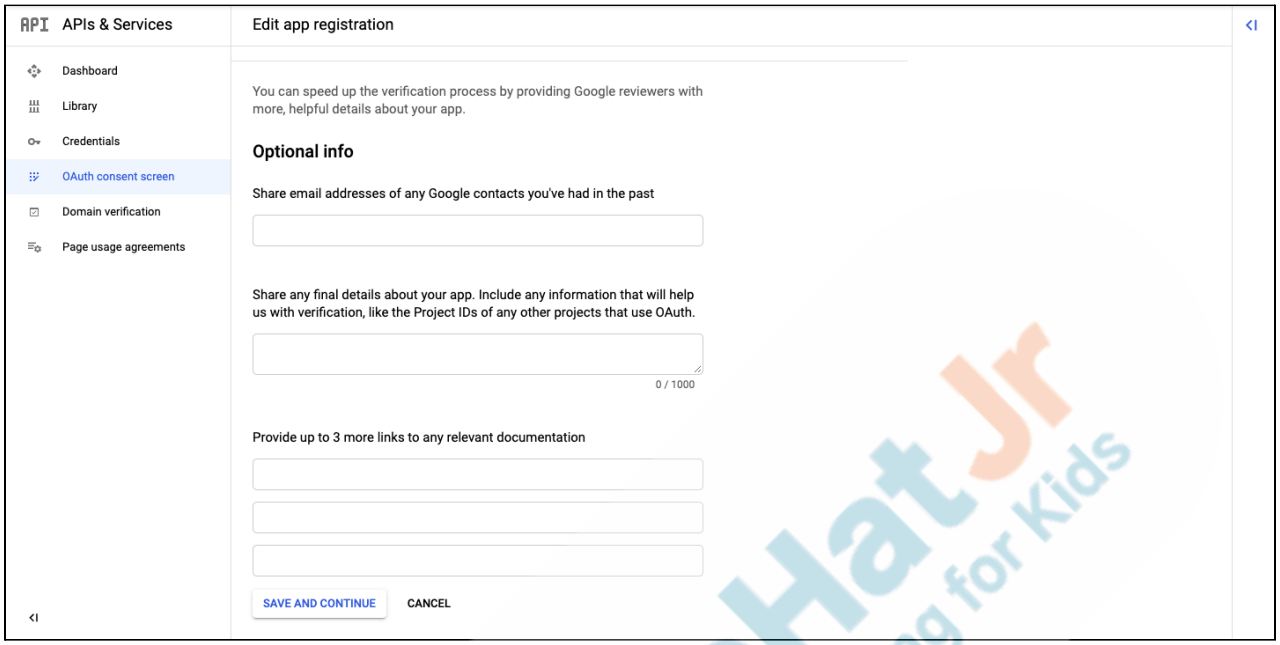
8. Next we need to enter our email ID for user support.



9. And also, our developer contact information.

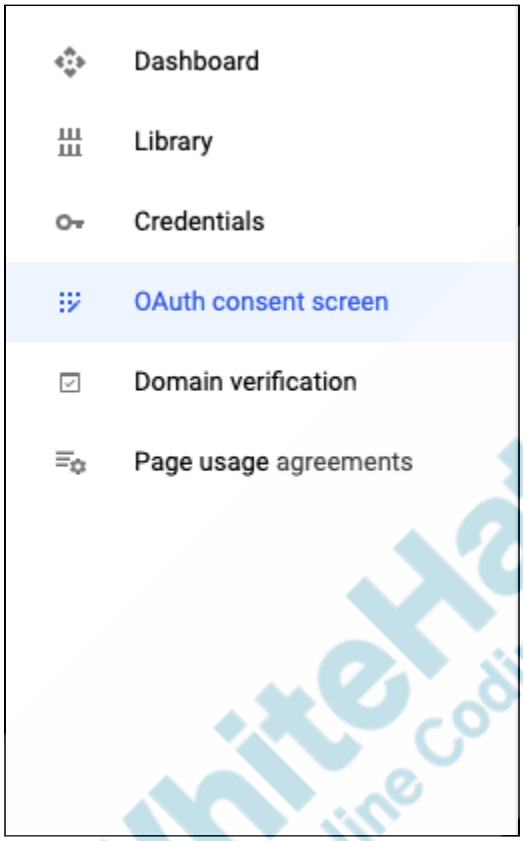
10. We'll click on save and continue to proceed. In the next few screens, we just have to click on save and continue without doing anything -

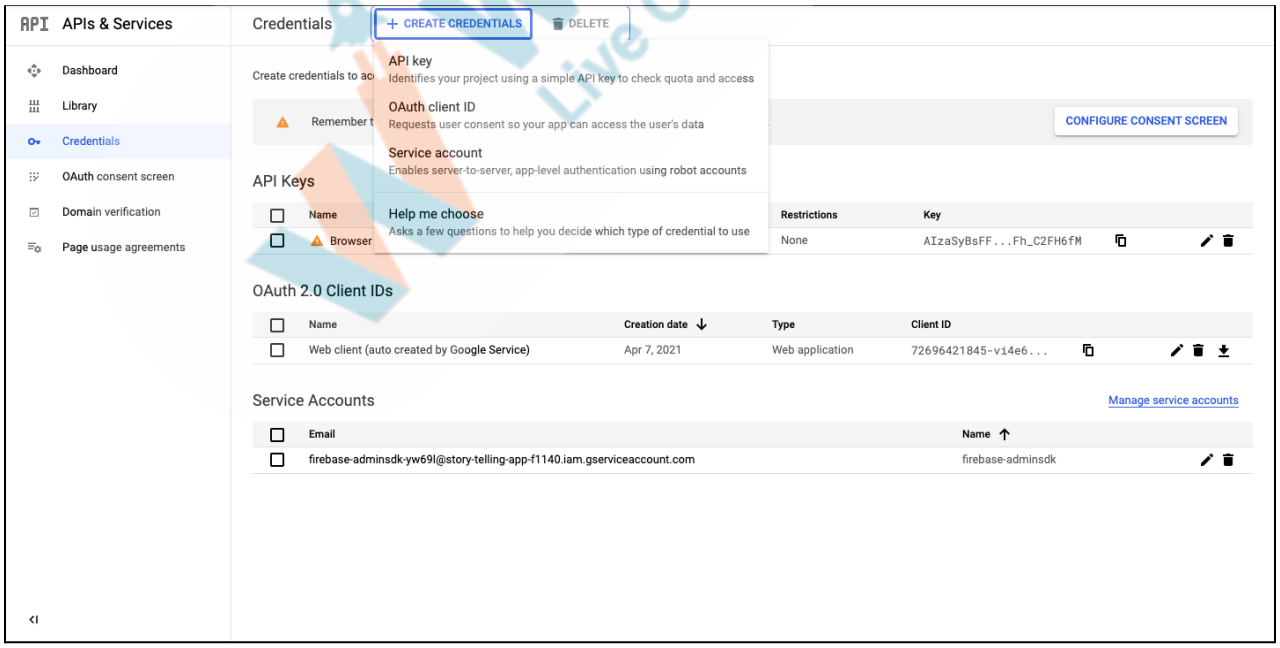




11. Finally, we go back to the dashboard.

12. From here, you can click on credentials in the side menu to go back to the credentials page -





13. Then finally click on OAuth Client ID from Create Credentials; we will now see the following screen -

Here, we will need to set up the credentials separately for Android and iOS. If we take a look at the **Expo's documentation** (From Teacher and Student Activity 2), we will see the following sections -

- **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).

There are instructions to set it up for both.

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

One issue that you might face is that ***openssl is not*** being installed in your device to run the command mentioned in the steps for Android.

To tackle this, here are the steps -

1. **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 -

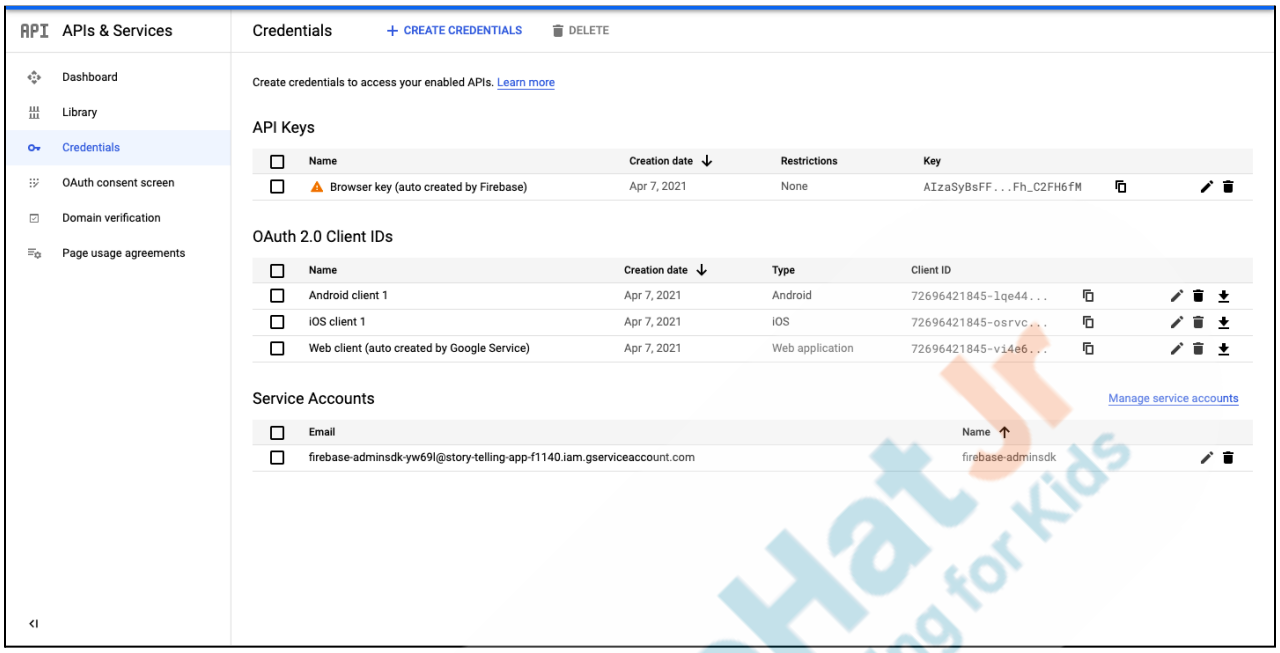
Run the following commands in a new Terminal -

sudo apt-get install libssl-dev

Windows -

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

2. Run the command - ***openssl rand -base64 32 | openssl sha1 -c***
3. This will generate the device's fingerprint that can be used to generate OAuth ID for android.
4. Once done, you will be able to see the OAuth ID's for both Android and iOS on your credentials page -



5. Under the OAuth 2.0 Client IDs, you will be able to see ID's for Android and iOS.

Now that we have set up our credentials, these can be used in the ***LoginScreen***!

Let's now go over the code in ***LoginScreen.js*** to understand what we have done there -

```
import React, { Component } from "react";
import { StyleSheet, View, Button } from "react-native";
import * as Google from "expo-google-app-auth";
import firebase from "firebase";
```

Teacher explains the code to the student

We first have our import statements, before having our Class component for LoginScreen created.

Ignore the functions ***isUserEqual*** and ***onSignIn*** for now. We will discuss them later.

Next -

```
signInWithGoogleAsync = async () => {
  try {
    const result = await Google.logInAsync({
      behaviour: "web",
      androidClientId:
        "72696421845-lqe44rrjuiggsegp1uv4gklv34tvl3gc.apps.googleusercontent.com",
      iosClientId:
        "72696421845-osrvc36bjie4264j4c0812sp5a2egqhj.apps.googleusercontent.com",
      scopes: ["profile", "email"]
    });

    if (result.type === "success") {
      this.onSignIn(result);
      return result.accessToken;
    } else {
      return { cancelled: true };
    }
  } catch (e) {
    console.log(e.message);
    return { error: true };
  }
};
```

We have a function called ***signInWithGoogleAsync***.

Note - The OAuth Client IDs that we have generated for Android and iOS needs to be updated in the **androidClientId and iosClientId** here**.**

signInWithGoogleAsync() function is available in the Expo Documentation ([Teacher and Student activity 2)](#) -

- **Add the Client IDs to your app**

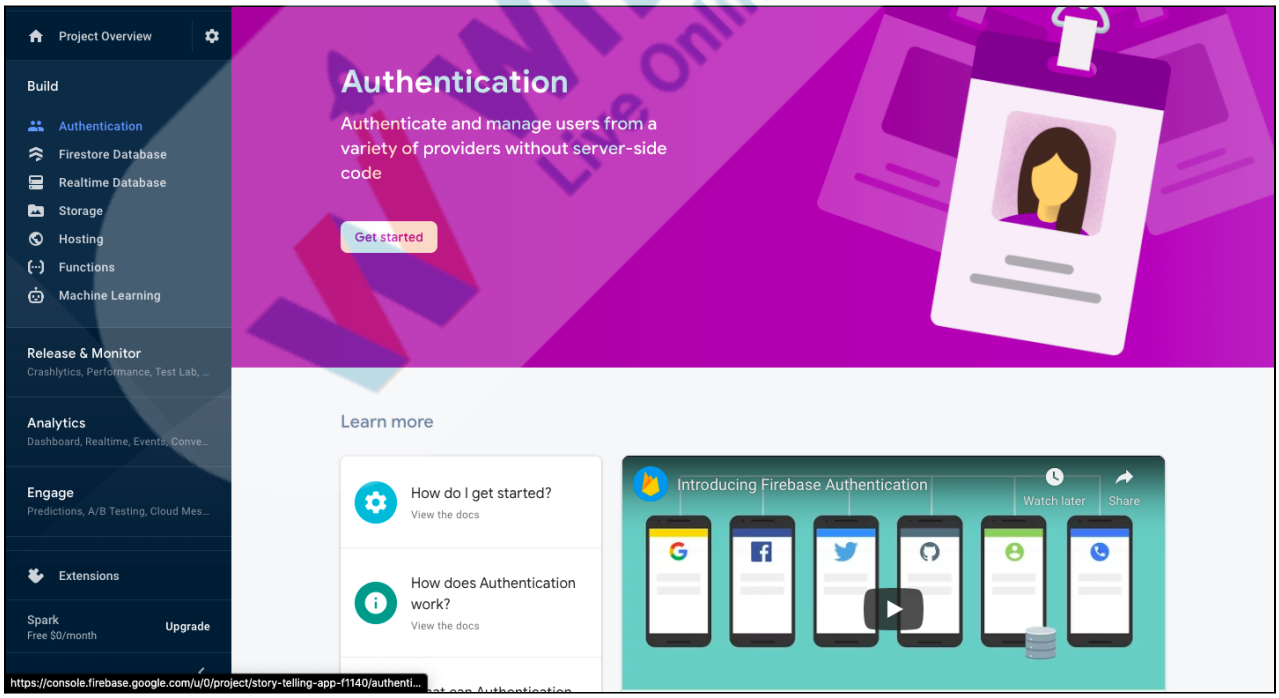
```
import * as Google from 'expo-google-app-auth';

async function signInWithGoogleAsync() {
  try {
    const result = await Google.logInAsync({
      androidClientId: YOUR_CLIENT_ID_HERE,
      iosClientId: YOUR_CLIENT_ID_HERE,
      scopes: ['profile', 'email'],
    });

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

We replaced our **androidClientID** and **iosClientId** from the ones we just created and we also added one more key value - **"behaviour"** set to **"web"**. This means that we want to work it as if we were using the web on react native.

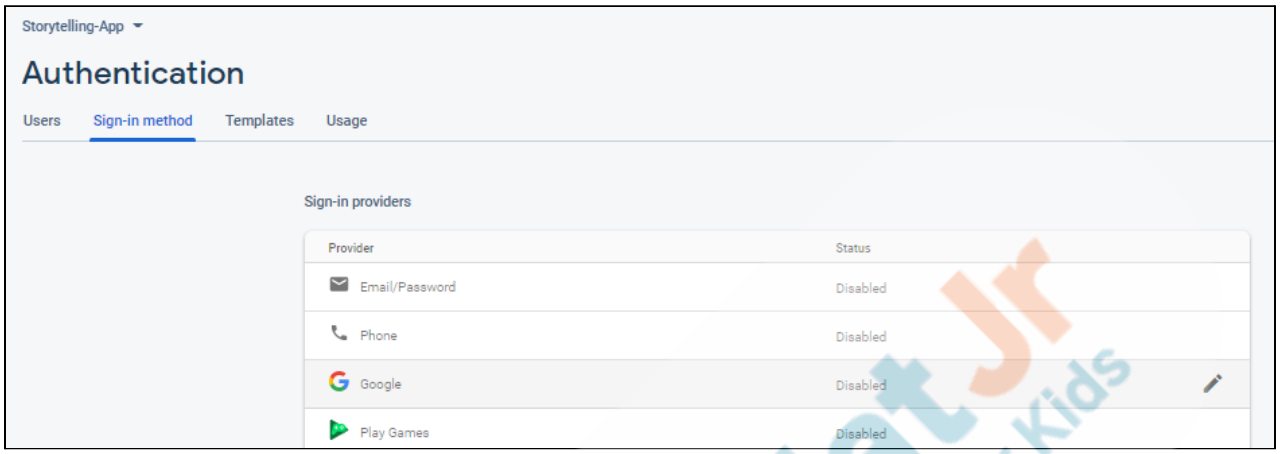
We also have a **<Button>** component in the ***render()*** function, which will execute this function on press -

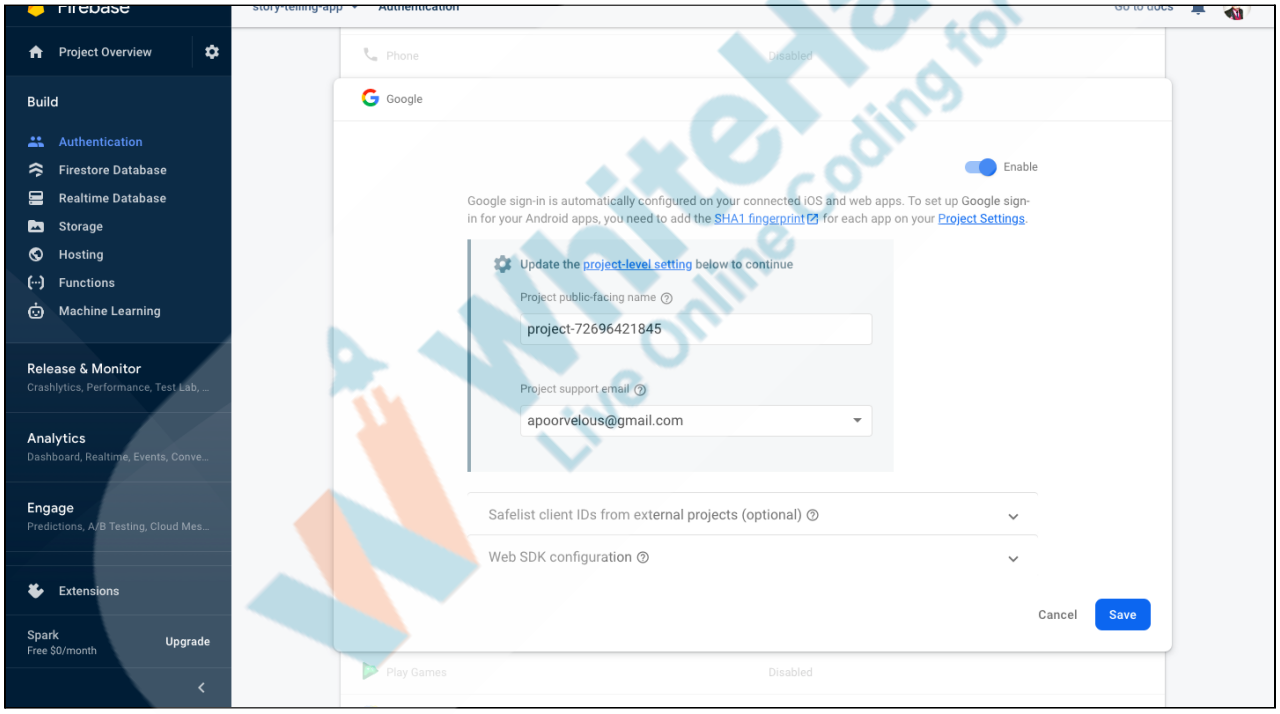
```
render() {
  return (
    <View style={styles.container}>
      <Button
        title="Sign in with Google"
        onPress={() => this.signInWithGoogleAsync()}
      ></Button>
    </View>
  );
}
```

Now since we want to save our user's data that logs in into Firebase, let's go back to the Firebase console, to enable Google Sign In method for our App!



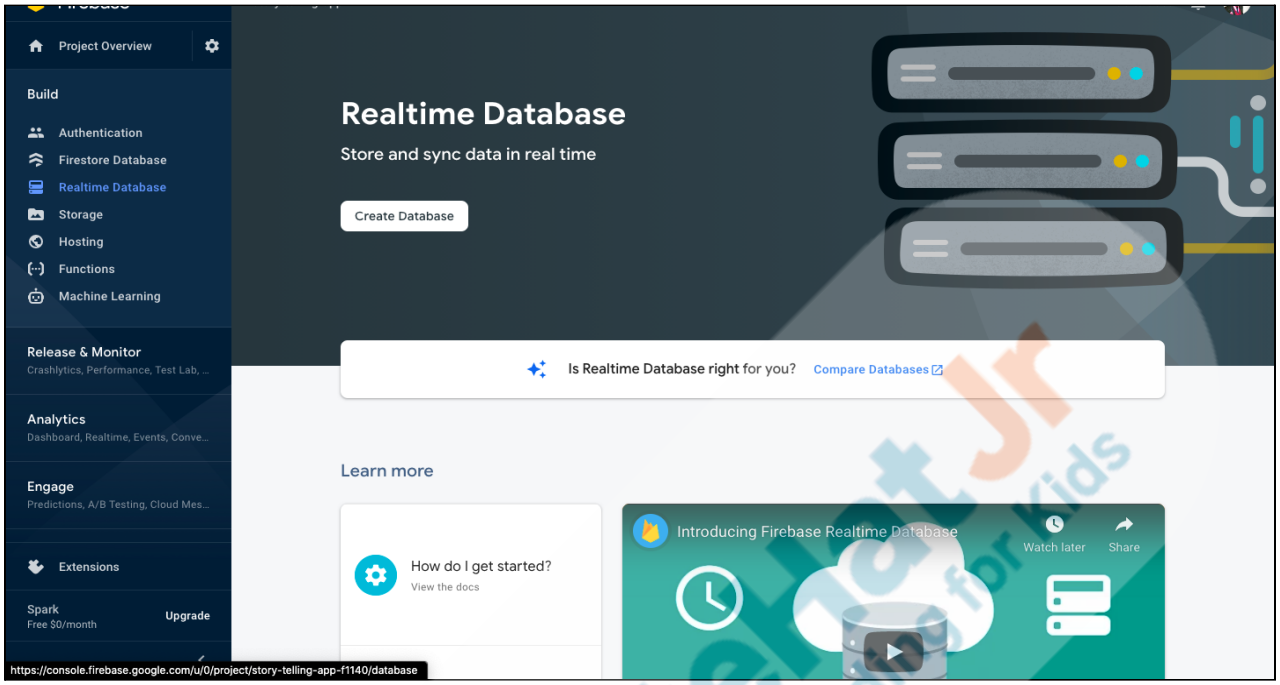
1. We'll first go to the authentication tab and click on **"Get Started".**



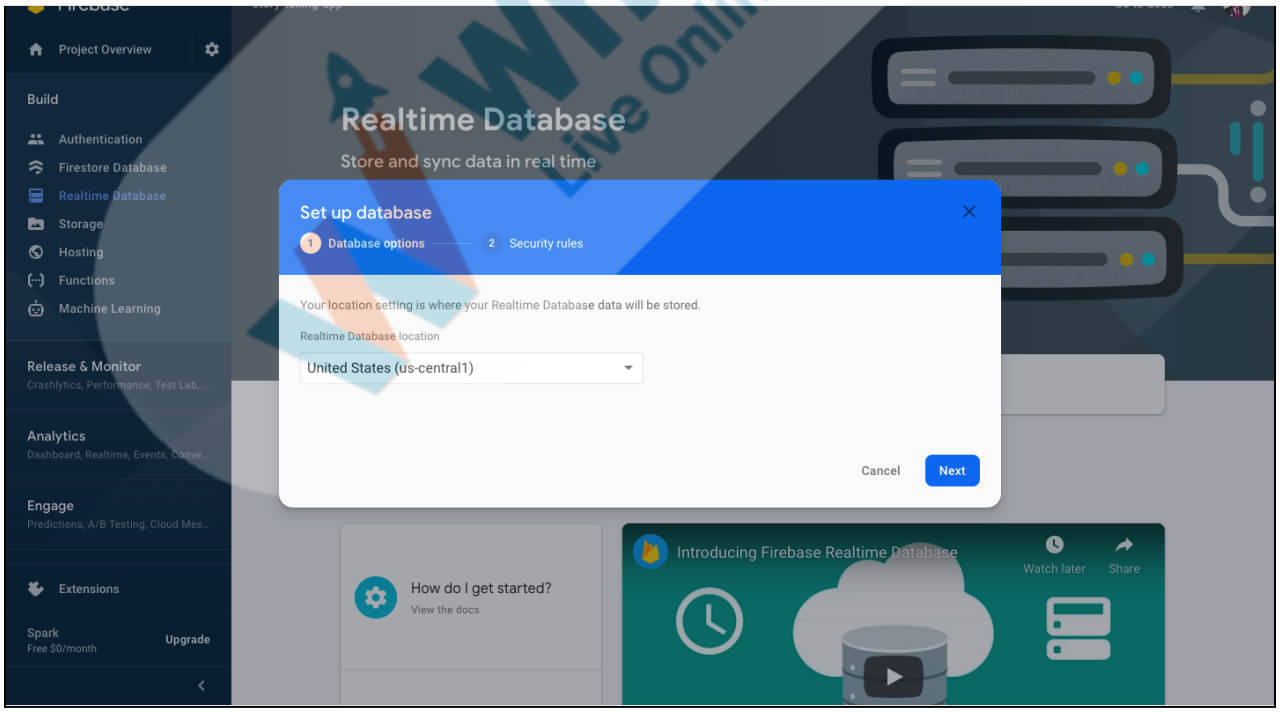


2. We'll then enable Google Sign In.

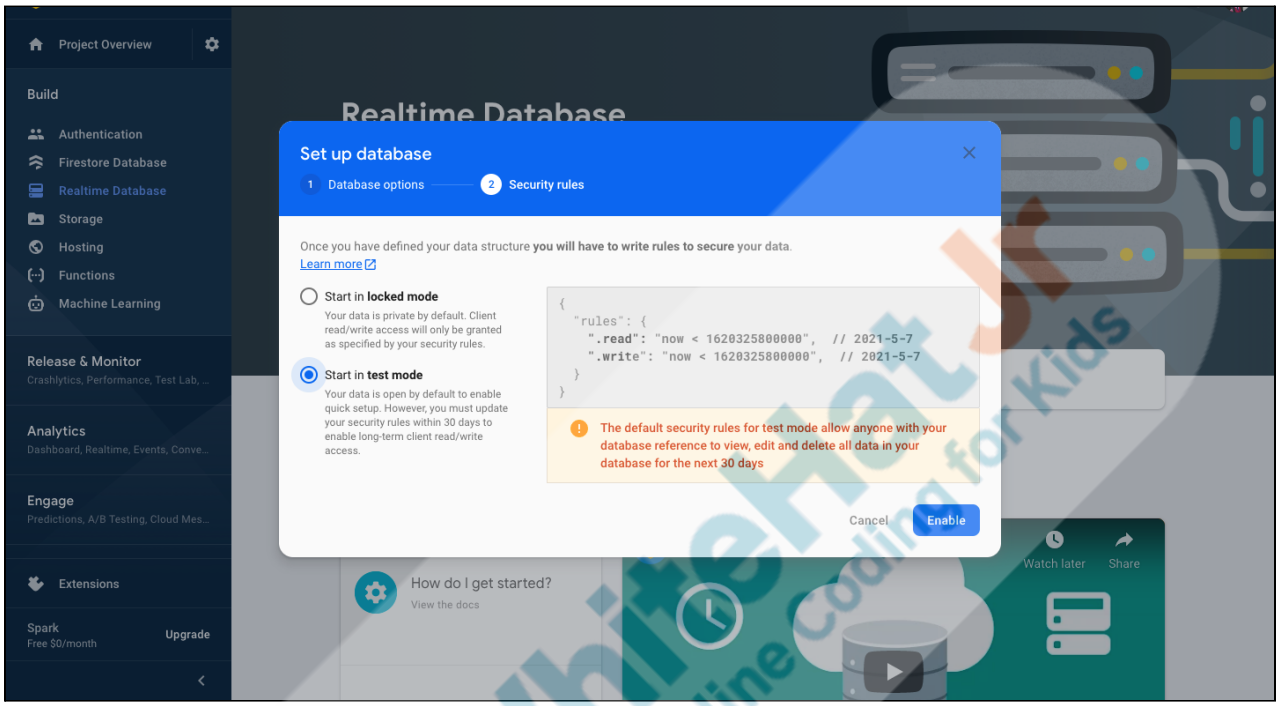
3. Next, let's go to Realtime Database (Remember, we are not using Firestore Database) -



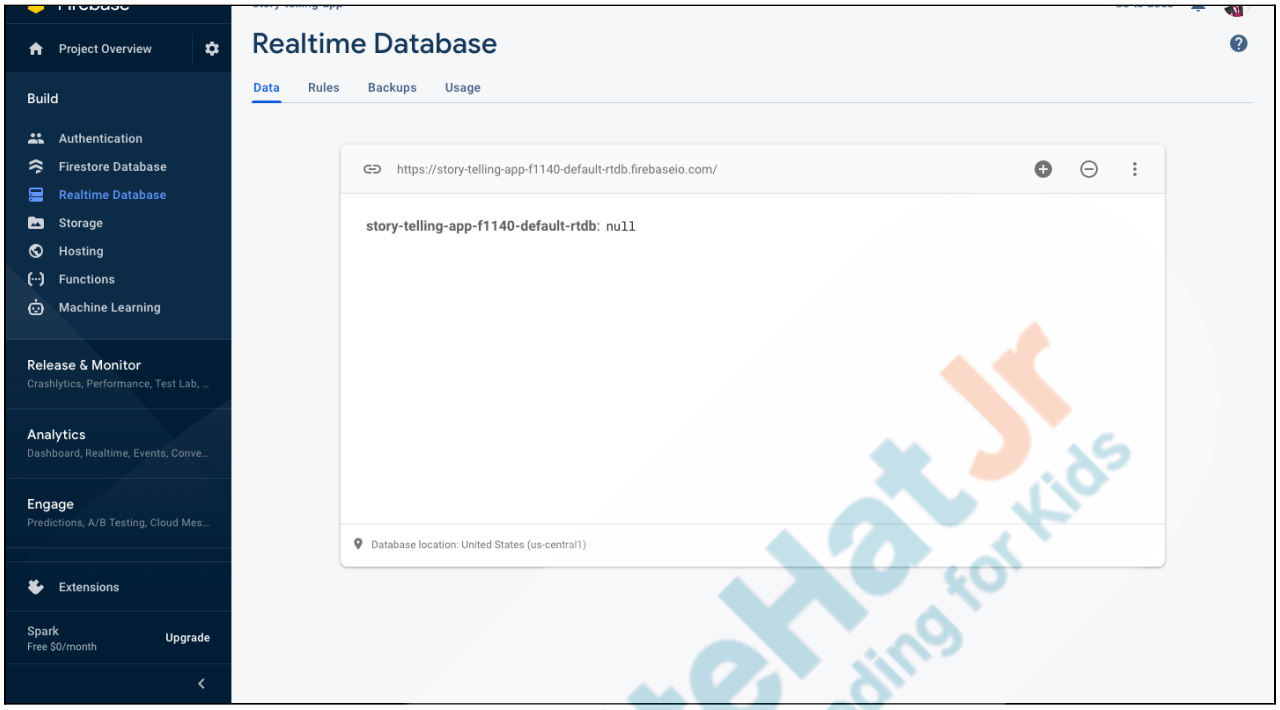
4. We'll click on Create Database -



5. Keep the country selection as the United States (since we don't have options with this one).



6. We will keep our database in Test Mode for our app so we can read and write to it.



Now the database is ready.

	Now, let's refer to the Google Sign In authentication with JavaScript documentation. *Teacher refers to Teacher Activity 3.* We will start by installing the following dependency - *expo install expo-google-auth-app* Now let's go over the remaining 2 functions ***isUserEqual*** and ***onSignIn*** to see what they do -	*Student refers to Student Activity 3.*

Here, we have added 2 more functions from the documentation -

For onSignIn() we picked up code from [Teacher Activity 3](#) Point .2
For isUserEqual() - picked up code from [Teacher Activity 3](#) Point .3

Now if we take a look at the **signInWithGoogleAsync()** function we created earlier, we were calling an **onSignIn()** function inside it on finding the user.

```
onSignIn = googleUser => {
  var unsubscribe = firebase.auth().onAuthStateChanged(firebaseUser => {
    unsubscribe();
    if (!this.isUserEqual(googleUser, firebaseUser)) {
      var credential = firebase.auth.GoogleAuthProvider.credential(
        googleUser.idToken,
        googleUser.accessToken
      );
      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 => {
          ;
        });
    } else {
      console.log("User already signed-in Firebase.");
    }
  });
};
```

Teacher explains the code to the student.

In the **onSignIn()** function, after we sign in the user with Expo, it first checks if the user is signed into the Firebase authentication system or not with **firebase.auth().onAuthStateChanged()** function.

The **isUserEqual()** function checks if the user who is signed in is the same as the user who is trying to sign in.

In our app, we already have a check in **LoadingScreen** for if the user is signed in or not. Therefore we don't need this **isUserEqual()** function here since the signed in users do not see the Login screen but go directly to the dashboard. But for the sake of simplicity, we will keep it as is in our code since we took this code from the documentation.

```
isUserEqual = (googleUser, firebaseUser) => {
  if (firebaseUser) {
    var providerData = firebaseUser.providerData;
    for (var i = 0; i < providerData.length; i++) {
      if (
        providerData[i].providerId ===
          firebase.auth.GoogleAuthProvider.PROVIDER_ID &&
        providerData[i].uid === googleUser.getBasicProfile().getId()
      ) {
        // We don't need to reauth the Firebase connection.
        return true;
      }
    }
  }
  return false;
};
```

Once we are done with checking using the **isUserEqual()** function and ensure that it's not the same user, we 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, we will use the **id_token** and the **access_token** of our **googleUser.**

Once this is done, we signin the user with the Firebase authentication system using

firebase.auth().signInWithCredential() function.

We have used a **.then()** function to check if the user is a new user or not with **result.additionalUserInfo.isNewUser**.
The result will be the user returned by Firebase Authentication, which contains a lot of other information as well.
If it is a new user, we are saving the user's info in our database under a **"user"** reference -

```
// 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) {

                })
        }
    })
```

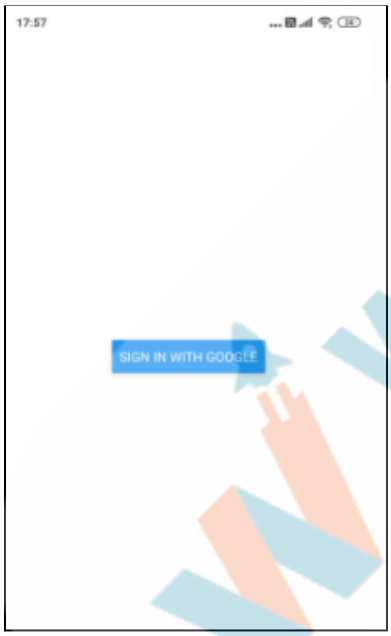
Here, we are saving the:

1. Email ID
2. Profile Picture
3. Local Language
4. First Name
5. Last Name
6. Current Theme

	We are getting the first 5 data points from Google itself, and we are setting the user's preferred theme as **"dark".** That's because we have built all our screens in dark mode until now. We'll be adding the light theme later. Now, there's only one thing left—our dashboard screen. In our *App.js,* before integrating Google Sign In, We were using our Drawer Navigator as the main screen. If we use it again on our Dashboard Screen, will it work?	**ESR:** Yes!
	That's what we have in the ***DashboardScreen.js*** -	

85t > screens > JS DashboardScreen.js > DashboardScreen

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

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

	We are ready. Let's test our app now by trying to Log In. (*Teacher and student try to Login and see the Feed screen.*) If you check your Database, you will see that your email ID has been registered.	
OUTPUT: 		
	Great! Now we have our Google Authentication all done and our app is integrated with Firebase. (***A note for teachers -*** *We know this class might be difficult to complete based on the machine speed and*	

	installation required. We have kept the next classes smaller to get time in the later classes so the process can be completed in next classes. You will get enough time in C-87 will things start falling behind in this class)	

Teacher Stops Screen Share

WRAP-UP SESSION - 5 mins

FEEDBACK

- **Appreciate the student for their attentiveness in the class.**
- **Get them to play around with different ideas**

Teacher can show slideshow from slides 12 to 22
Refer to speaker notes and follow the instructions on each slide.

For the 'Wrap-Up' section, there will be slides on the panel as a visual aid to summarize what has been done in the class.

Activity details	Solution/Guidelines
Run the presentation from slide 12 to slide 16 **Following are the warm up session deliverables:** • **Appreciate the student.** • **Revise the current class activities.** • **Discuss the quizzes.**	Discuss with the student the current class activities and Student will ask doubts related to the activities.

QnA Session

Question	Answer
Which function returns the current user that has logged in? A. firebase.auth().onAuthStateChanged()	**A**

B. firebase.auth().currentUser() C. firebase.auth().UserLoggedIn D. firebase.auth().onAuthChanged()	
Which function sign-ins the user with the Firebase authentication system? A. firebase.auth().signInWithCredential() B. signInWithGoogleAsync() C. firebase.auth().onAuthStateChanged() D. firebase.auth().GoogleAuthProvider.credential()	**A**
Which database we are using in our app? A. Cloud firestore B. Realtime database C. Friestore database D. None of the above	**B**
End the quiz panel	
Activity details	**Solution/Guidelines**
Run the presentation from slide 17 to slide 22 **Following are the warm 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 with us
Amazing work today! You get a "hats-off". 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.	*Make sure you have given at least 2 Hats Off during the class for:* Creatively Solved Activities +10

	Alright. See you in the next class.	 
Project Overview ***Teachers make sure to tell students to refer to documents used during class and Post class Summary to implement Authentication in the Project.*** **Spectagram Stage - 5** **Goal of the Project:** In Class 85, we learned to implement Google Authentication and integrate the app with Firebase. In this project, you will practice the concepts learned in the class and implement Google Authentication and integrate the Spectagram app with Firebase. *This is a continuation project of 81, 82, 83 & 84; 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		*The students engage with the teacher over the project.*

has decided to create a social media app. She has asked for your help to create an app. Guide Jenny to implement Google Authentication and integrate the app with Firebase.		
Teacher ends slideshow		
Teacher Clicks ✖ End Class		

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.

Activity	Activity Name	Links
Teacher Activity 1	Firebase Console	https://console.firebase.google.com/
Teacher Activity 2	Expo Documentation for Google	https://docs.expo.io/versions/latest/sdk/google/#using-it-inside-of-the-expo-app
Teacher Activity 3	Authenticate Using Google Sign In with Firebase	https://firebase.google.com/docs/auth/web/google-signin#expandable-2
Teacher Activity 4	Reference Code	https://github.com/pro-whitehatjr/ST-85-Solution
Teacher Activity 5	Teacher Aid	https://drive.google.com/file/d/1WA1BQff4dmgv5BInU3f_imk4vlpvAyMa/view?usp=sharing
Teacher Activity 6	Snack Support Document	https://docs.google.com/document/d/11vq49uJQCfdxaUUzOoY7A65aau0kZqNMFhObZH-e71Y/edit?usp=sharing
Student Activity 1	Firebase Console	https://console.firebase.google.com/
Student Activity 2	Expo Documentation for Google	https://docs.expo.io/versions/latest/sdk/google/#using-it-inside-of-the-expo-app
Student Activity 3	Authenticate Using Google Sign In with Firebase	https://firebase.google.com/docs/auth/web/google-signin#expandable-2
Student Activity 4	Reference Code	https://github.com/pro-whitehatjr/ST-85-Solution
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/PRO_V3_C85_LITE_withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/6e8b7abc-392c-48b5-ae9

		0-12414dc83826.pdf