

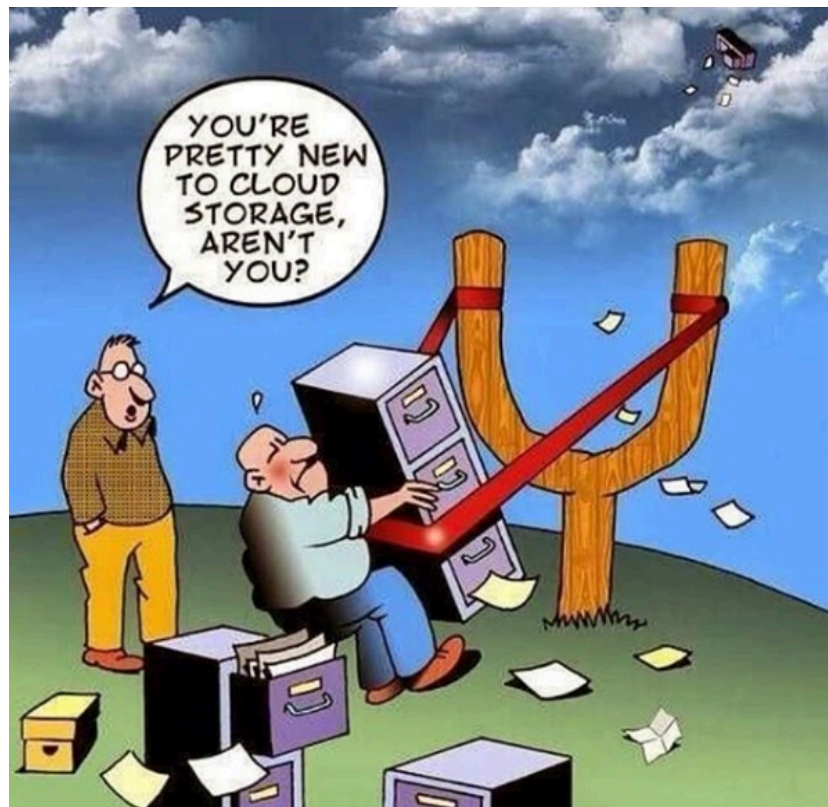


Project in Android Development

236272 // Spring 2024/25

Assignment 2

Please don't print me!



Introduction

Story time: In the previous assignment, we've created a very cool startup name generator app! Unfortunately, most of it doesn't work, and our users are unable to back up their favorite names to the cloud so that they can keep using the app, even between devices. For this assignment, we will extend this functionality, by adding authentication and a cloud database to our application.

Consider the following features, which are described as user stories:

- As a user, I want my favorite startup names backed up to the cloud so that I can always access my favorite startup names (even if I log in on a different device).
- As a user, I want to be able to login into the app, so that I can access the backed up favorites.

We'll begin by enabling **Firebase** for our project.

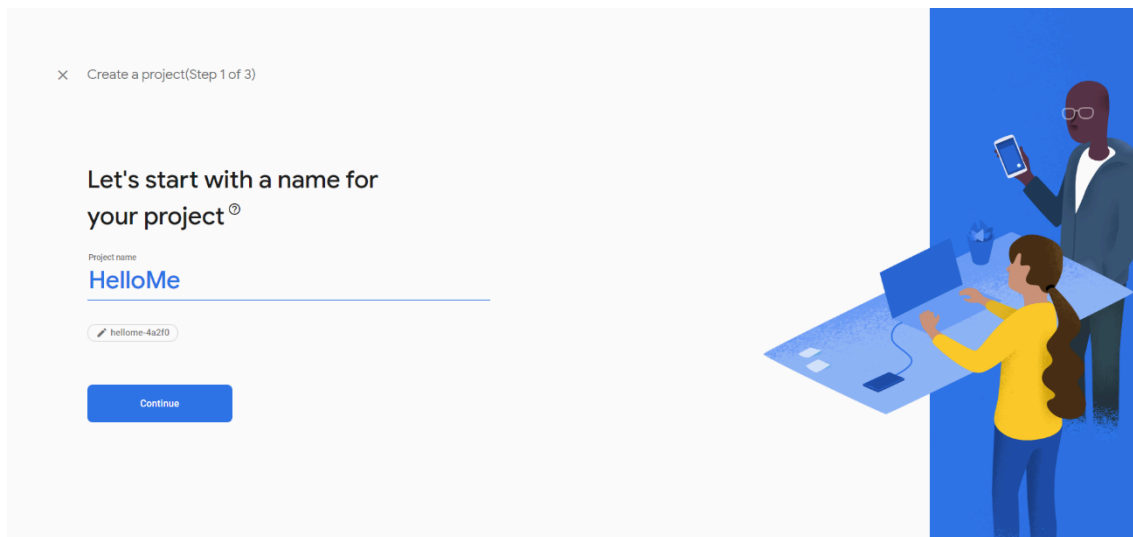
FAQ

This document might be updated from time to time, to reflect common questions and problems that students are tackled with, so be sure to keep an eye for updates, which will be highlighted.

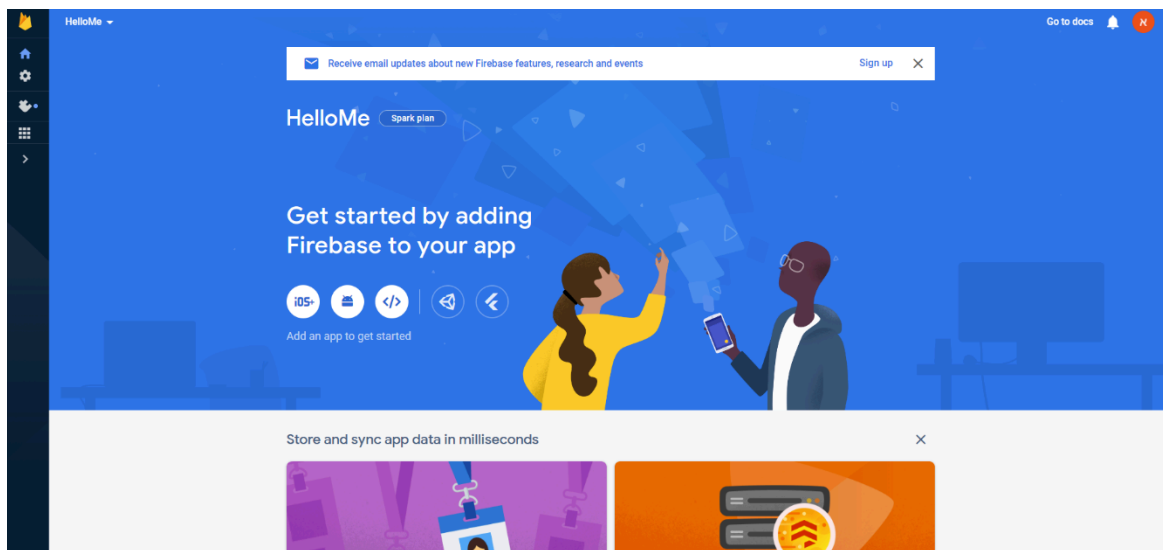
Setting up Firebase

Creating a Firebase project

1. Navigate to <https://console.firebase.google.com/>, log in with your Google account, and click **Add project**.
2. Name your project **HelloMe** and continue.



3. Disable **Google Analytics** for this project and click continue.



- Click on the Android logo to open the “**Add Firebase to your Android app**” screen.
- Fill in the project configuration:

Android package name	com.technion.android.hello_me
App nickname	HelloMe

Notice: The package name must always match the application, or Firebase will not be able to connect to it!

You can verify your package name by looking at `applicationId` inside `android/app/build.gradle`

- Click **Register**.
- Download `google-services.json` and place it in your project directory, under **<project_root>/android/app/google-services.json**. This file contains the configuration information required for our app to communicate with the Firebase cloud
- In your **<project_root>/android/build.gradle**, add the following line to the dependencies segment (inside `buildscript`):

```
classpath 'com.google.gms:google-services:4.3.13'
```

```
dependencies {  
    classpath 'com.android.tools.build:gradle:7.1.2'  
    classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"  
    classpath 'com.google.gms:google-services:4.3.13'  
}
```

- In your **<project_root>/android/app/build.gradle**, add the following line after the other "apply plugin" lines:

```
id : 'com.google.gms.google-services'
```

```
plugins {  
    id "com.android.application"  
    id "kotlin-android"  
    id "dev.flutter.flutter-gradle-plugin"  
    id "com.google.gms.google-services"  
}
```

and finally, replace the "defaultConfig" section in that file with the following lines:

```
defaultConfig {  
    // TODO: Specify your own unique Application ID (https://developer.android.com/studio/build/application-id.html).  
    applicationId "com.technion.android.hello_me"  
    // You can update the following values to match your application needs.  
    // For more information, see: https://docs.flutter.dev/deployment/android#reviewing-the-build-configuration.  
    minSdkVersion 19  
    targetSdkVersion flutter.targetSdkVersion  
    versionCode flutterVersionCode.toInteger()  
    versionName flutterVersionName  
    multiDexEnabled true  
}
```

Note: Google's firebase site might ask you to add different lines than shown here, But you don't need them for this project.

Connect Firebase to your app

First, open the `pubspec.yaml` file. This is your app configuration file, where configurations are stored in a tree form, by indenting different parts of the app (You've seen it in the lecture and assignment 1). To add Firebase support, modify the **dependencies** section to look like this:

```
dependencies:  
  flutter:  
    sdk: flutter  
  firebase_core: ^2.1.1
```

Then, click on **Pub get** to retrieve the dependencies, or run `flutter pub get` from the command line. Running **pub get** automatically downloads the necessary libraries for us.

Next, we modify `main.dart` with a widget that loads our Firebase configuration file when we start the app, and shows a progress bar until it is loaded. We saw exactly how this is done in **Workshop 2**.

open `main.dart` and add the following code snippet under `main()`:

```
class App extends StatelessWidget {
  final Future<FirebaseApp> _initialization = Firebase.initializeApp();

  @override
  Widget build(BuildContext context) {
    return FutureBuilder(
      future: _initialization,
      builder: (context, snapshot) {
        if (snapshot.hasError) {
          return Scaffold(
            body: Center(
              child: Text(snapshot.error.toString(),
                textDirection: TextDirection.ltr)));
        }

        if (snapshot.connectionState == ConnectionState.done) {
          return MyApp();
        }

        return Center(child: CircularProgressIndicator());
      },
    );
  }
}
```


Make sure that you import the new package with:

```
import 'package:firebase_core/firebase_core.dart';
```

Finally, replace `main()` with our implementation:

```
void main() {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(App());
}
```

Important: as a last step, return to the Firebase console

(<https://console.firebase.google.com/>). Navigate to  → Project settings → Users and permissions → Add member and add course236503@gmail.com as an **Editor** for your project.

Exercise: Authentication

Important: When writing your code, make sure you correctly apply **state management**, following the concepts we've seen in the workshop.

It is recommended that you use the **Provider** package for managing user state, like we've seen in Workshop 2.

Design specification

Most of the design specification was implemented in assignment 1. You are requested to add one little thing:

1. In the Login screen, add a "Sign up" button below the "login" button.

Operational specification

The operational specification describes how the components should behave:

1. When the user clicks the app bar's login action, the app navigates to the login screen.
2. **Login screen:** When the user clicks "*Login*", the user credentials (email, password) are used to sign him into the app.
 - a. When the user is **in the process of logging in** (i.e. pressed the login button, but Firebase did not yet return a successful answer), the **Login** button should either be disabled, or be replaced with an *indeterminate progress bar*.
 - b. When successful, the user is **automatically** navigated **back** to the main screen (suggestions).
 - c. If the authentication is unsuccessful, show a **Snackbar** with the following error message: "*There was an error logging into the app*" (and stay in the login screen).

Note: There is no need to check the validity of the email/password fields.

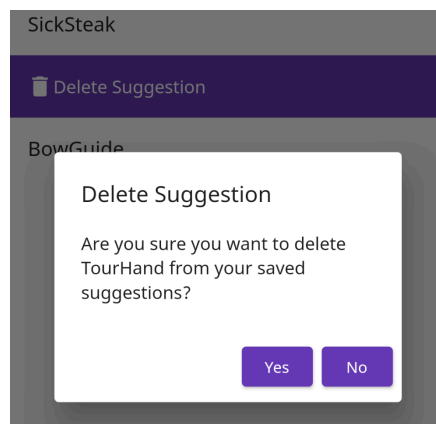
The **exact same requirements** apply for the "sign up" button (though, you might be required to use different Firebase functions for the signing up).

3. Main screen:

- a. When the user is **logged in**
 - i. The top bar icon is "exit_to_app".
 - ii. Clicking the icon immediately logs the user out (how can we do this immediately even though the logging out process might take some time?) and a **Snackbar** is shown with the following message: "Successfully logged out"
- b. When the user is **logged out**: No change from before.

4. Saved Suggestions Screen:

- a. Swiping a saved suggestion to the right/left will prompt an **AlertDialog** asking to confirm the deletion of the swiped suggestion (The dialog and the user's pick doesn't do anything functional **yet**, it's only UI). The deletion of a suggestion should look something like this:

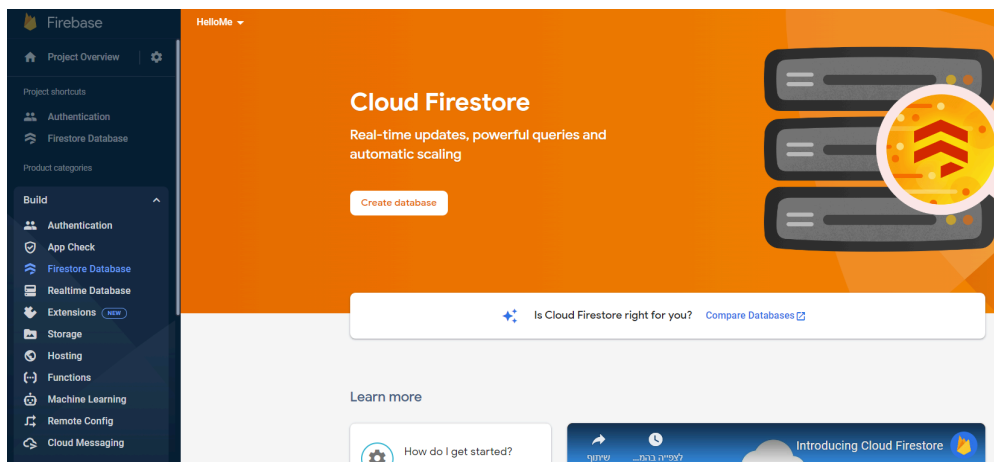


Exercise: Cloud Firestore

Now that our users can sign in to the application, we can help them back up their previous suggestions! First, we have to setup our Cloud Firestore.

Creating a Firestore database

1. In your Firebase project, navigate to the "Cloud Firestore" tab:



2. Click **Create database**.
3. Choose **Start in Test mode**, and then click **Next** and **Enable**.
4. Navigate to the "Rules" tab, change the code to:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if true;
    }
  }
}
```

And click **Publish**. This allows anyone connected to your firestore to read and write. Generally, this is an undesirable behavior; for now, this is fine.

5. Now, add the "cloud_firestore" package to your application (you should already now how to do such a thing).

Design specification

No change.

Operational specification

1. **Main Screen:** Clicking the “Star” icon for a suggestion:
 - a. If it was already favorited, it is removed from the user’s favorites list
 - b. Otherwise, it is added to the list
2. **Saved Suggestions Screen:** Clicking “No” *or the surroundings of the AlertDialog* will cancel the deletion, Clicking “Yes” removes the corresponding suggestion from the user’s favorite list

Application state

When the user is **logged out**, the following holds:

1. All suggestions are saved and deleted locally, as before. When the app is closed and reopened, the suggestions are no longer saved (i.e., there's no need to keep data between different sessions of the app if the data is saved locally since the user is logged out).

When the user is **logged in**, the following holds:

1. All suggestions are saved and deleted to and from the database. When the user logs out and back in, he should see his saved suggestions.
2. If the user previously had saved suggestions locally, and then logs into his account, then the updated list of favorites is the **Union** between the local set and the one held in the cloud database. That is, whenever possible, locally saved suggestions are moved to the cloud database.
3. After the user leaves the app, it isn't necessary to keep him logged in.

Throughout the entire application’s lifetime, the UI must always remain valid with respect to the **state**. For example, when a user deletes a favorite from the list, that is immediately reflected in the UI as well.

Implementation Tips

- In order to keep the implementation easy, it is suggested that you separate your logic into 2 components: one for user state management, and one for the suggestions themselves.
- If using Provider, you might find **ChangeNotifierProxyProvider** to be useful. Use Firebase's **authStateChanges()** to easily manage the user state.
- Think about how to structure your database documents easily, so that querying is efficient and the code implementation and state management is easy. There is no single solution that works, but try to keep it **as simple as possible**.

Example use-cases

In order to clarify some of the specifications, below are some example use-cases of the app:

1. John is a new user. He installs the application and opens it without logging in. He then marks "MindHead", "FogWolf", "MainScript" as 3 favorite names. When John exits the app and re-enters it, his favorites are gone!
2. John is a returning user. Previously, he had 1 favorite name backed up to the cloud: "ShellTree". He installs the application and opens it without logging in. He then marks "MindHead", "FogWolf", "MainScript" as 3 favorite names. John then navigates to the login screen and logs into his account. When he returns and checks his saved suggestions, he sees 4: "ShellTree", "MindHead", "FogWolf", "MainScript".

Submission

To prepare your project to submission, do the following:

1. In the project root folder, add a **submission.json** file, which contains **student_name** and **student_id** keys with your values. Download the template file from here:
<https://gist.github.com/mikimn/f6dfa802fa3a8128c499a22434c34a38>
2. Run `flutter clean`.
3. Go to **File** → **Export to Zip File** and name your file **submission.zip**. If you don't see this option, you can simply create a zip from all files in the root directory.
4. If successful, the exported zip should contain the following structure:
 - `android/`
 - `ios/`
 - `lib/`
 - `test/`
 - `pubspec.lock`
 - `pubspec.yaml`
 - `README.md`
 - `submission.json`
5. You might also see a `web/` folder, and you can include it as well if that's the case. (Everything should be set for submission after running `flutter clean`.)
6. Make sure to not include any other unnecessary files, except the ones that Flutter generated for you, and your code. Specifically, **don't** include the build folder (You shouldn't see it if *flutter clean* succeeded).
7. A submission that doesn't compile out of the box (after running `flutter pub get`) will not be checked, **make sure you unzip and run your submission before submitting.**

8. **This is a meme-friendly course! Make memes out of this assignment and your coding experience, add them to your dry.pdf and you might get featured!**
9. **Just as in Assignment 1, An exceptionally improved UI (no new functional features but looks prettier, aligned, centered, etc. without contradicting any specification) might get featured as well.**

Good Luck!