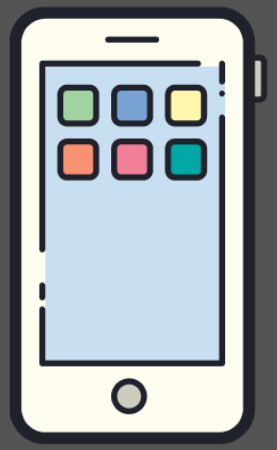


MOBILE DEVELOPMENT

Remote DB (Firebase)

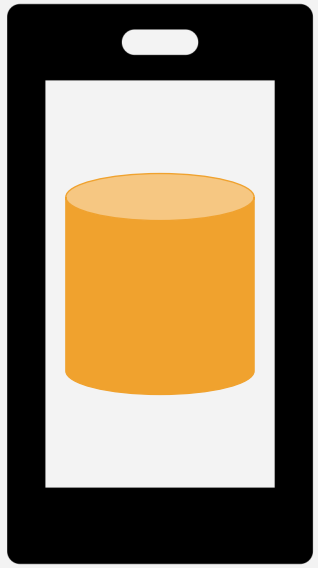
RECALL: Saving Data

- There are many ways one can save data in Android
  - Database (local / online)
  - Files
  - ~~SharedPreferences~~

- ~~For now, we'll focus on SharedPreferences~~

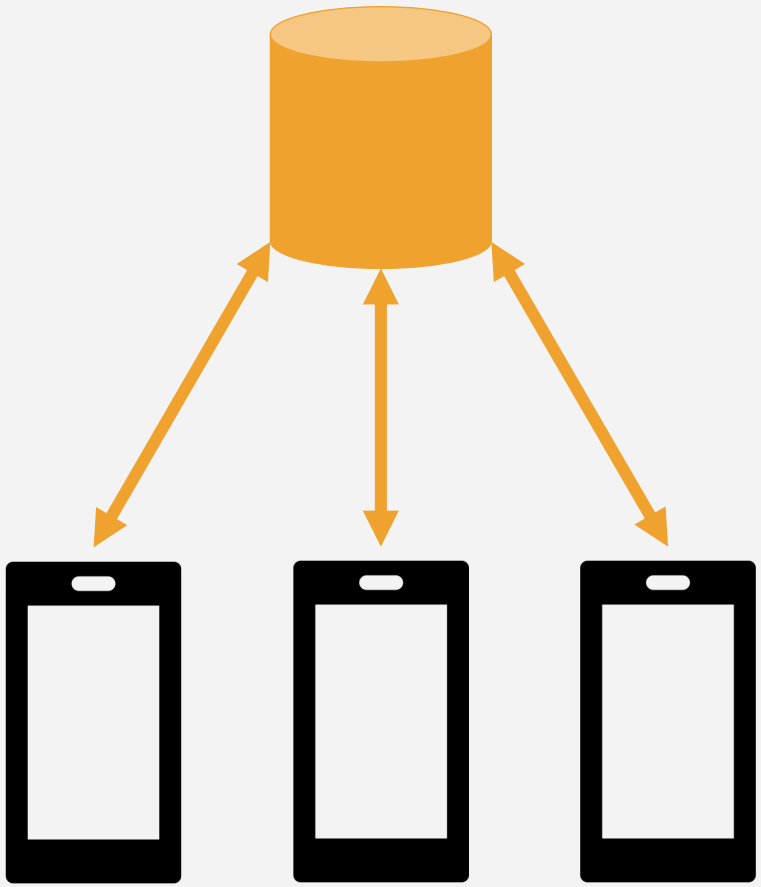
So far…

- We should be able to handle data stored locally



So far…

- We should be able to handle data stored locally
- However, there might be certain applications that…
  - Require that users interact with each other
  - Read / write live data



This is where remote DBs can help

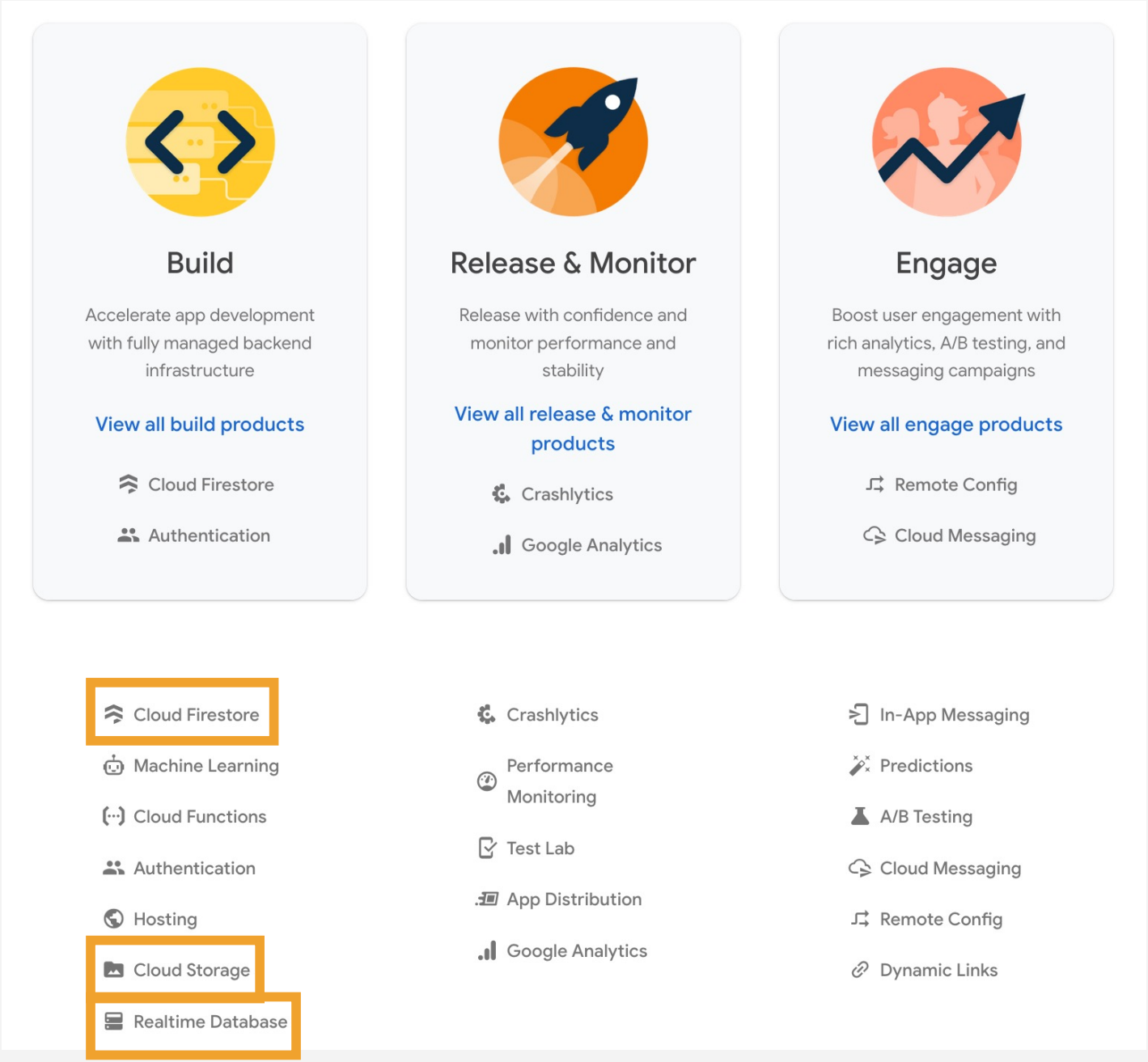
Remote Connections

- More generally, you can connect to a remote server which could handle any http request

- Available online DB services for Android:
  - Google Firebase ← Focus of our discussion today
  - MongoDB Realm / Atlas
  - Amazon DynamoDB
  - Or any remote server with a database

Firebase

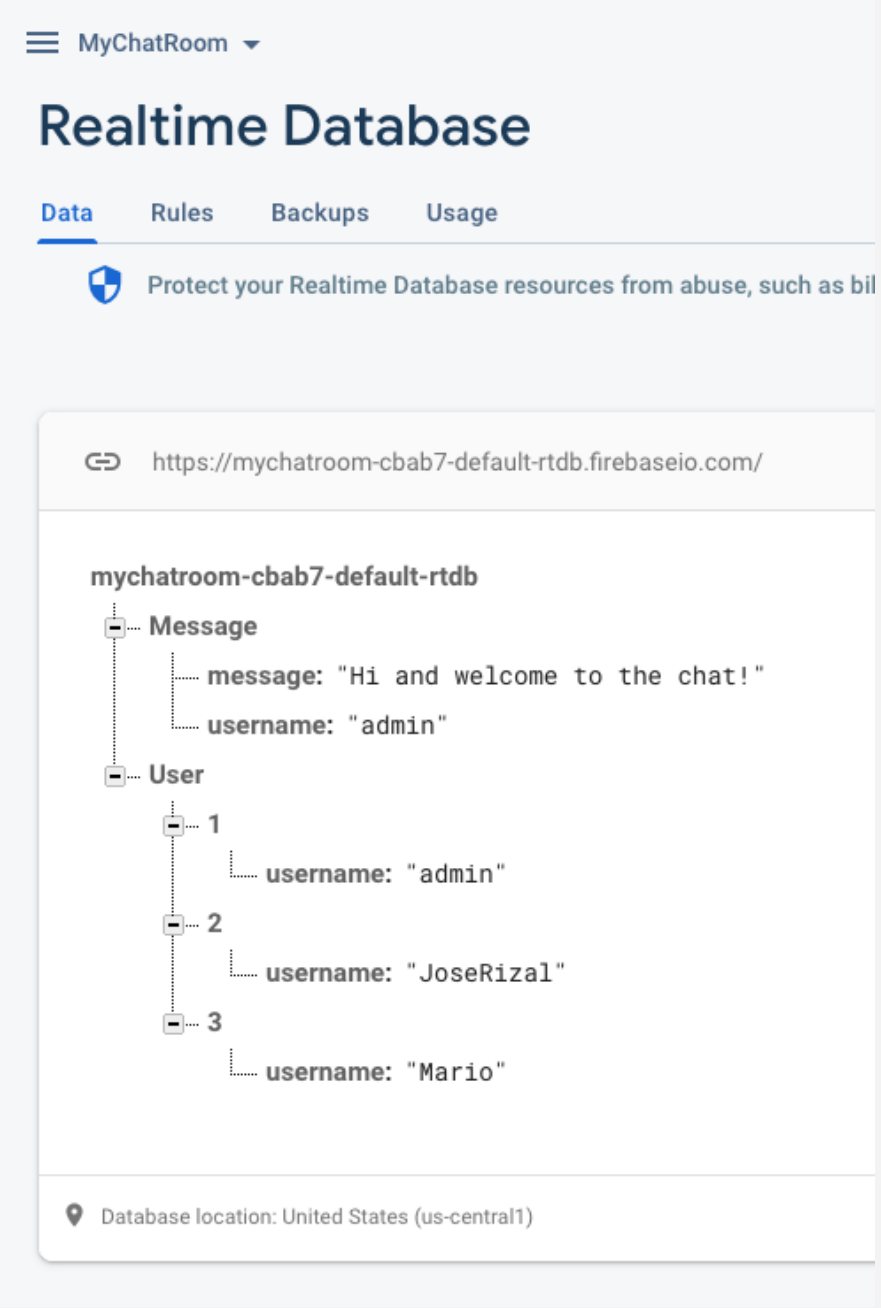
- Provides numerous tools for applications development
- Allows for a centralized location for your data (and other services) assessable across platforms



Sauce: https://www.gstatic.com/devrel-devsite/prod/vd7ff44c5bfb37fb3237a957a44096487cf6c040e737614b6a01efed4db62f3c8/firebase/images/touchicon-180.png

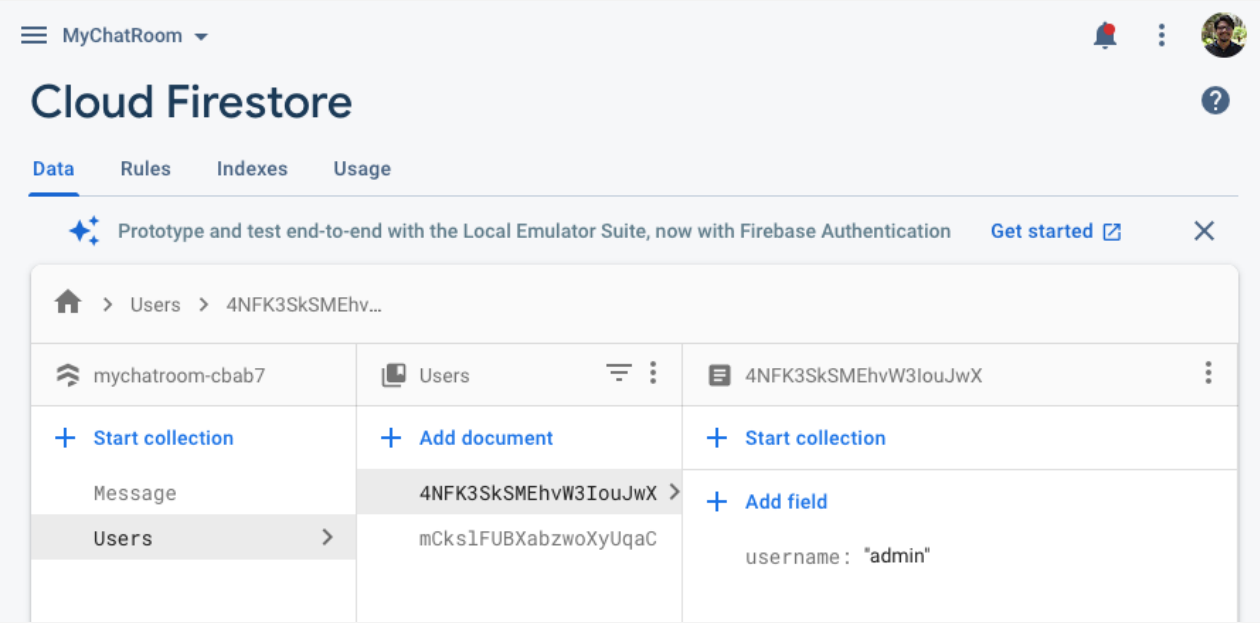
Realtime Database

- Was the original NoSQL DB (prior to Firestore)
- Entire database is stored as a single JSON tree
- Querying is limited as you're only able to sort based on one attribute



Firestore Database

- Is Firebase's improvement over Realtime DB
- Has a little more structure
  - Collections have documents
  - Documents have entries
- Has better querying capabilities

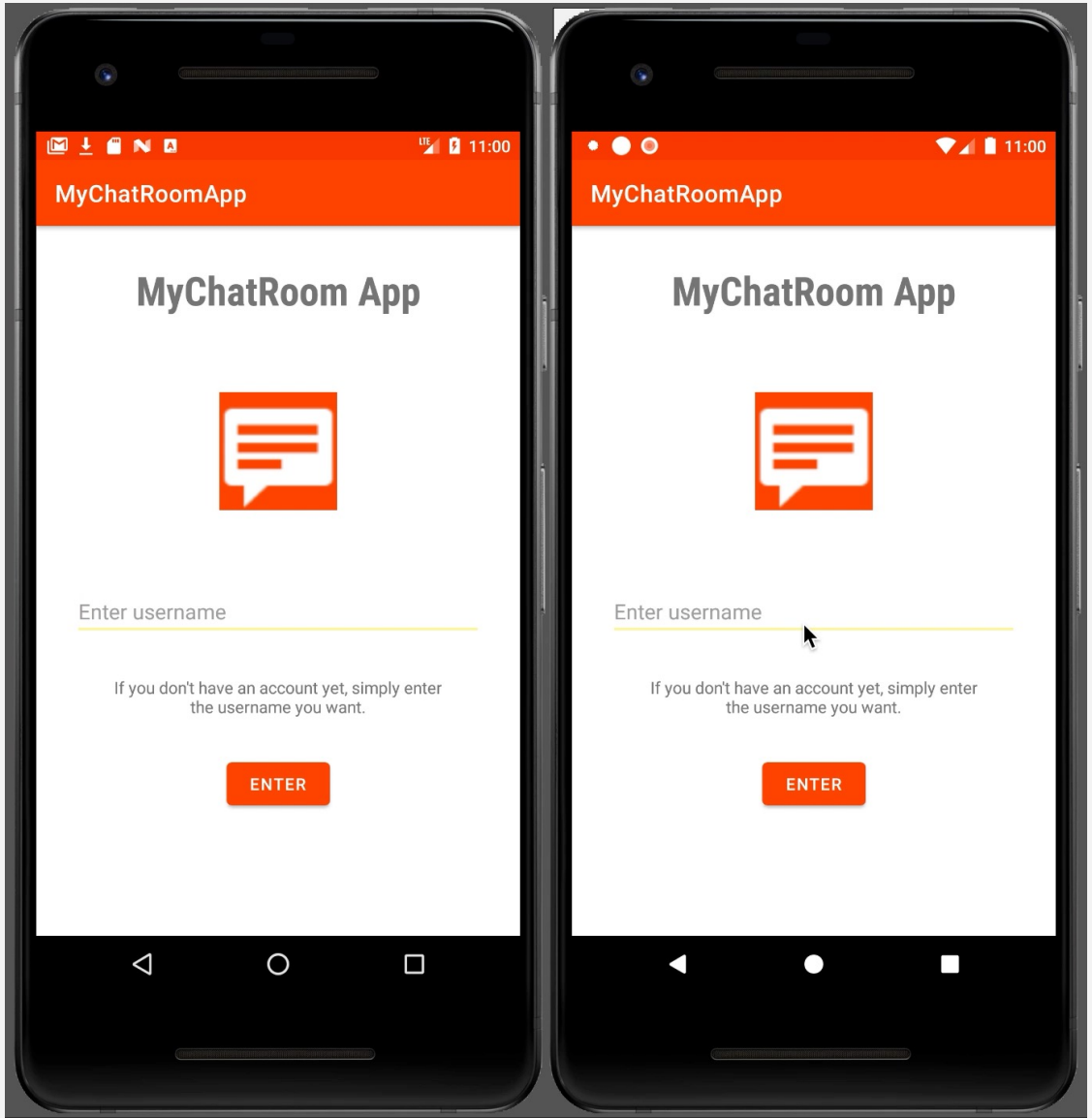


Which to pick – Firestore or Realtime?

- From a technical standpoint, Firestore seems the way to go because its an updated version of Realtime
  - Android doesn't make a recommendation and lists considerations to make when making a choice
  - https://firebase.google.com/docs/database/rtdb-vs-firestore
- However, general discussion points to pricing being the main deciding factor
  - Realtime -> bandwidth and storage
  - Firestore -> # of operations

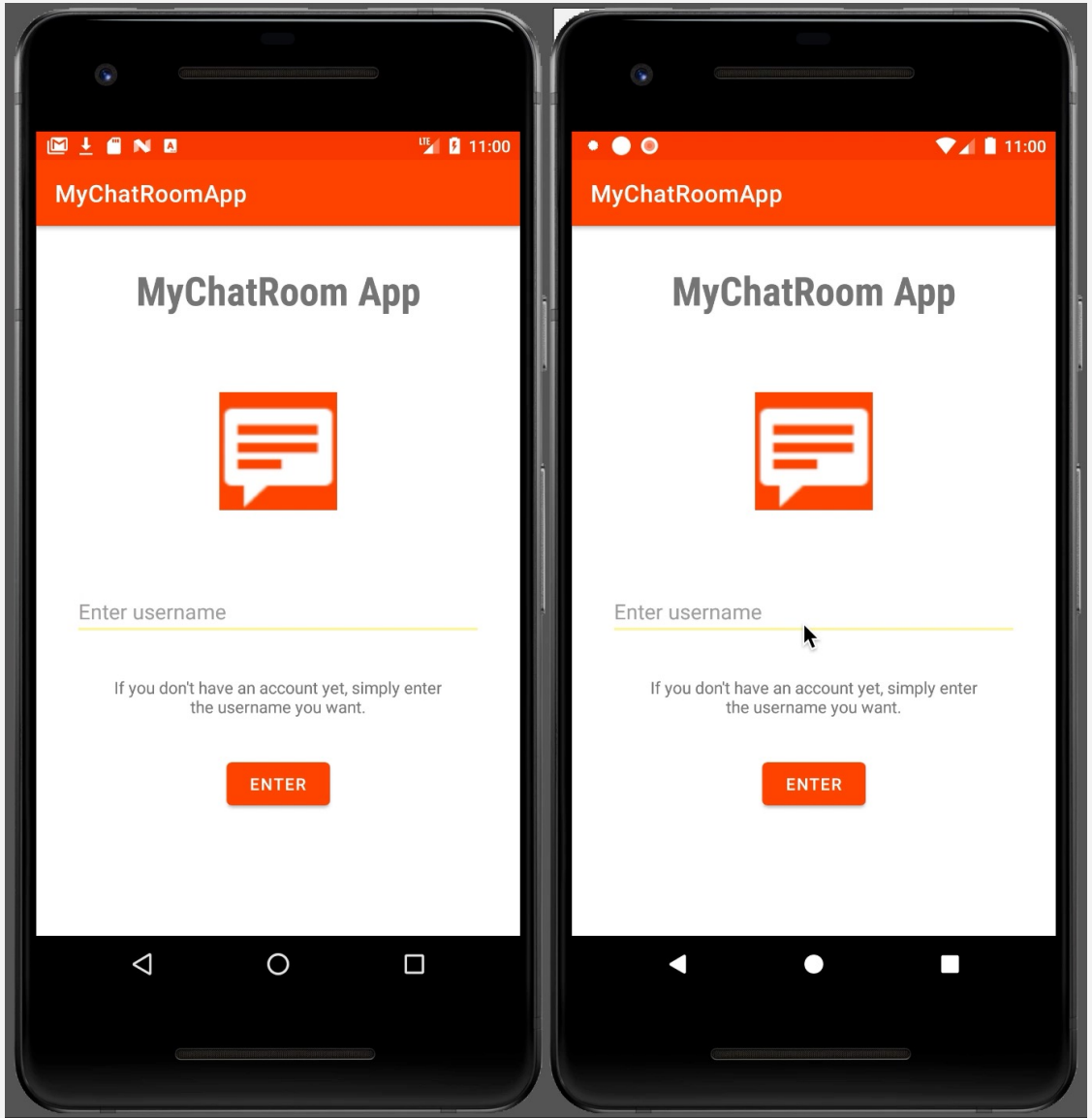
As a guide, for this lecture…

- We're going to create a very simply chat room application!
  - We'll use Firestore for the DB



As a guide, for this lecture…

- To make things simple…
  - Login is simply checking is a username has been used; there's no need for a password here
  - A message has a username, the message itself, and a timestamp



To set things up…

In Android Studio

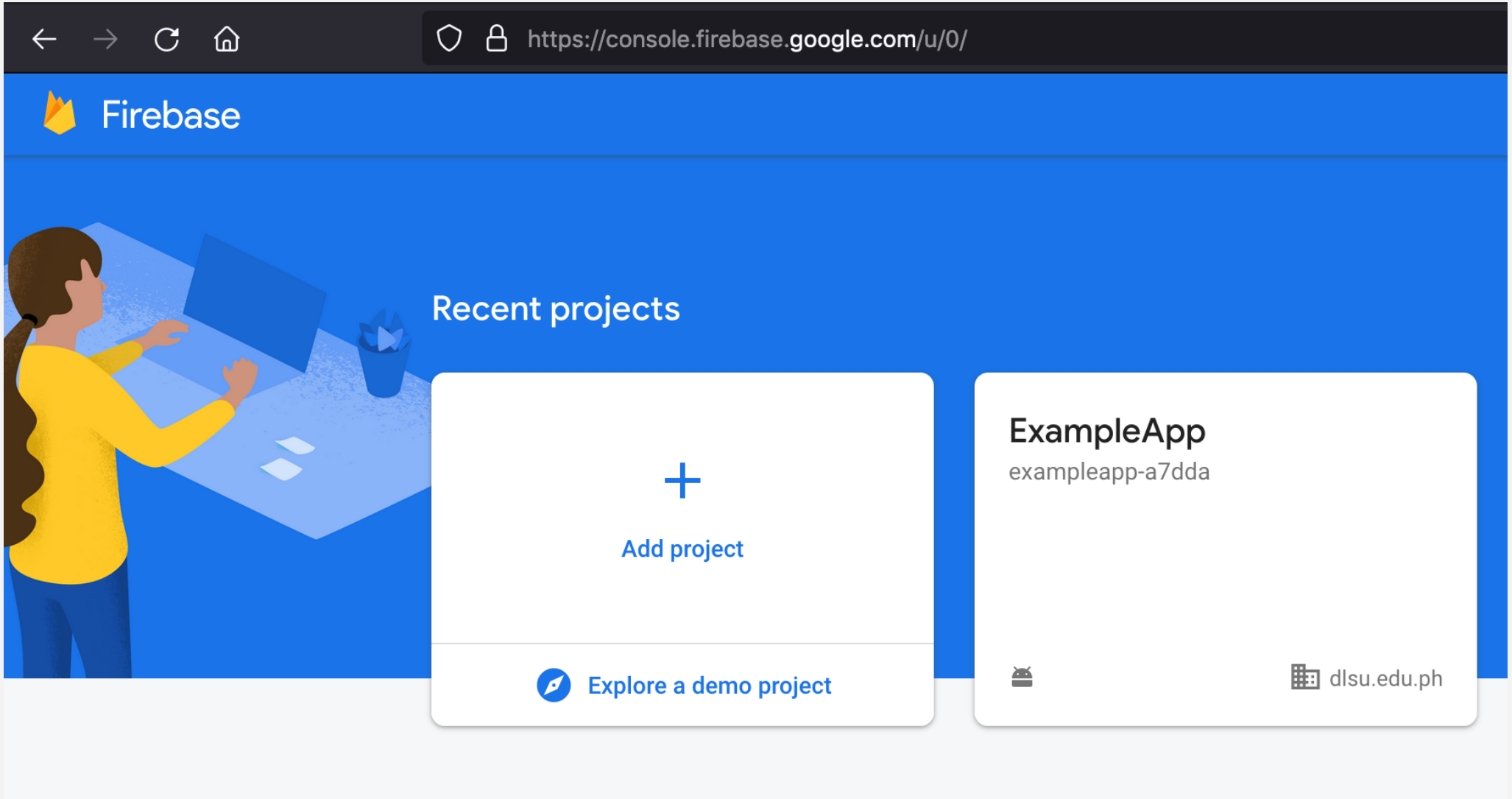
1. Create your app in Android Studio

In Firebase Console

2. Sign in to firebase.google.com
3. In the console, create a project
4. Within the project, add an application
5. Configure your app to support Firebase

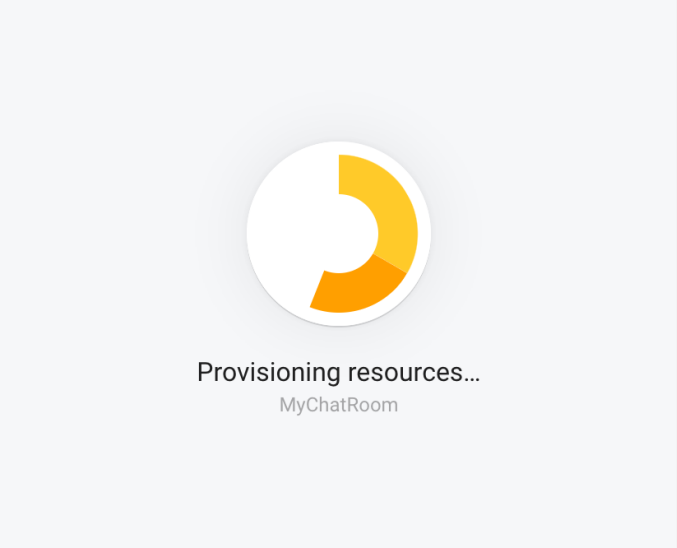
Sauce: https://firebase.google.com/docs/android/setup

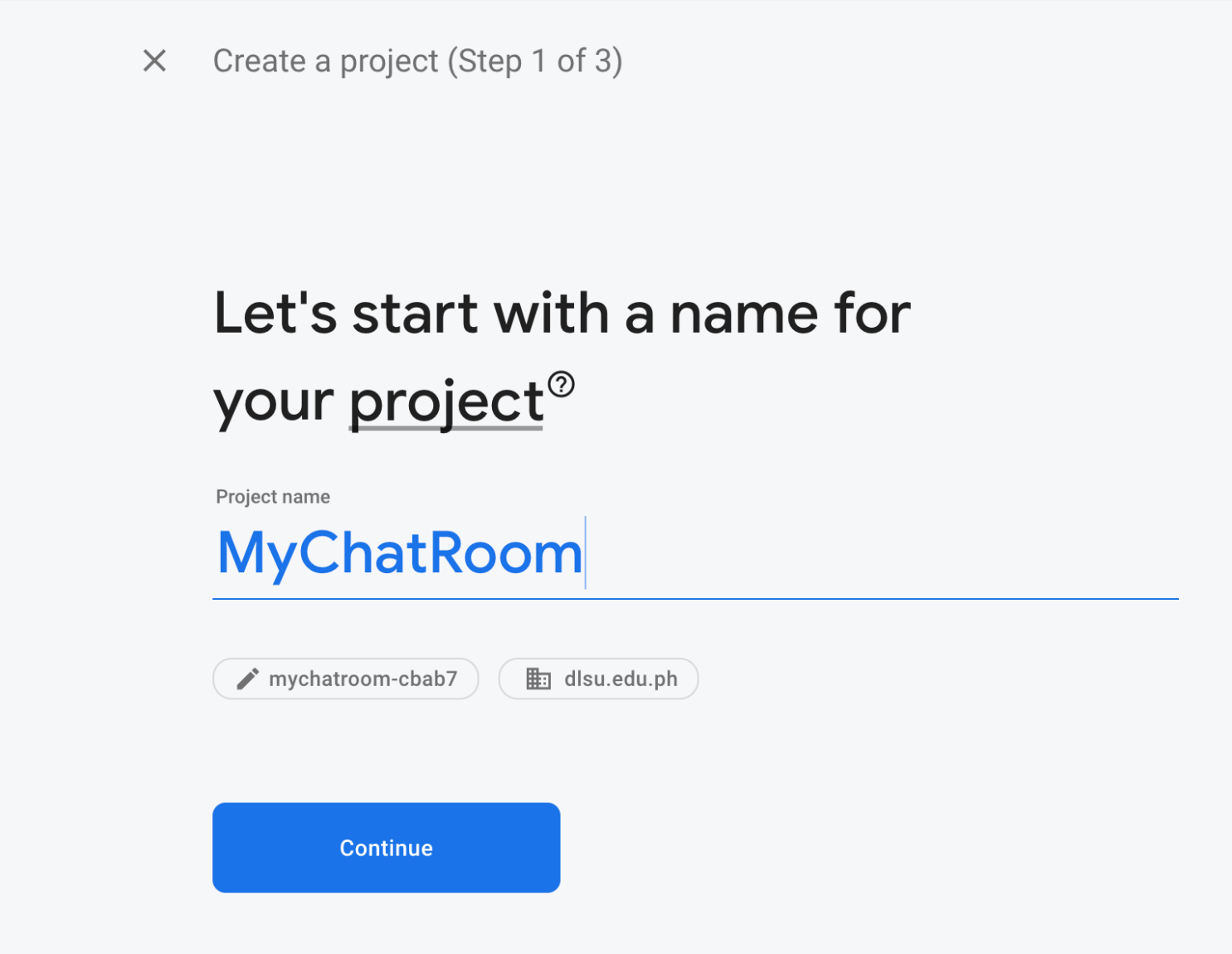
In the console, create a project…



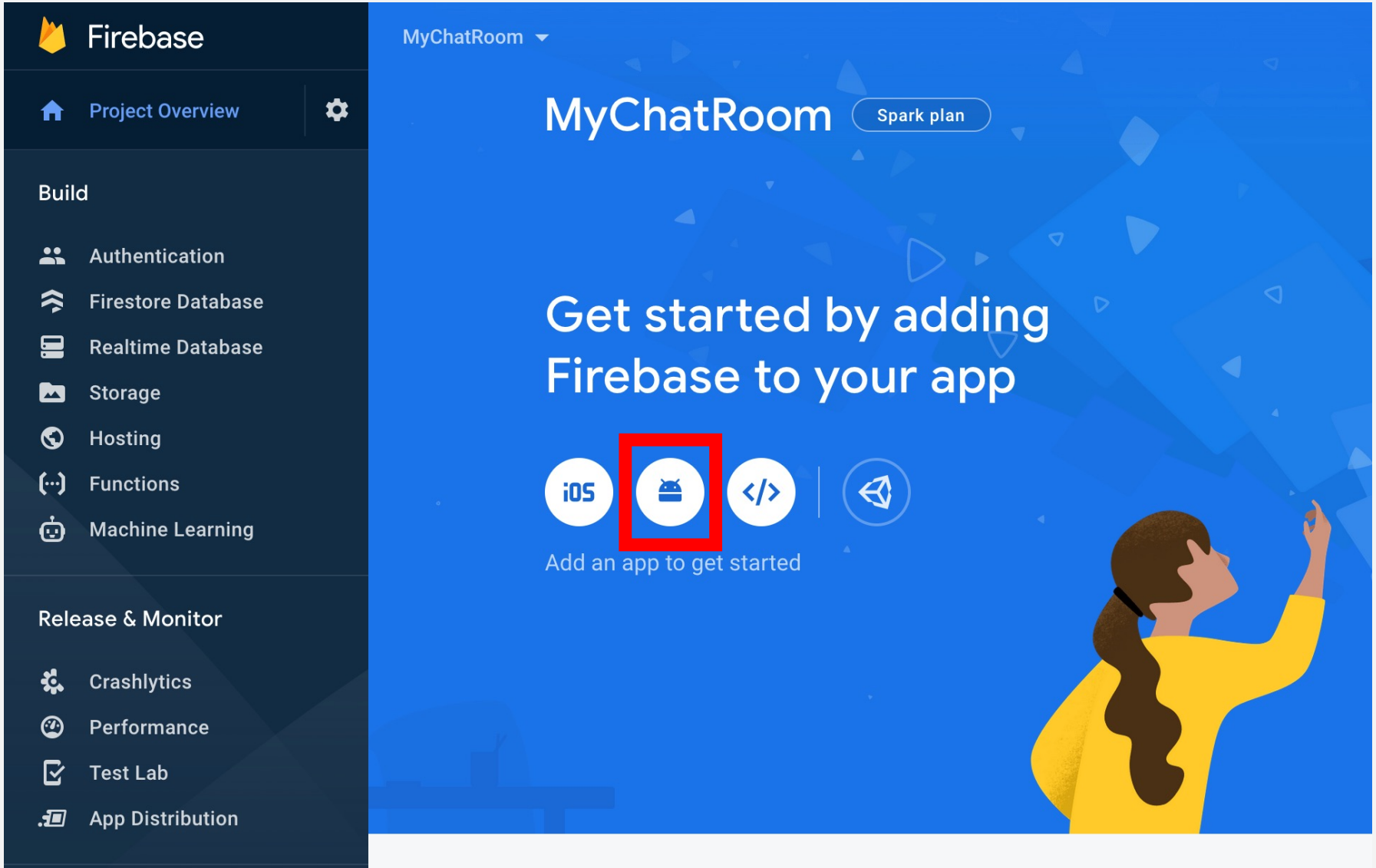
In the console, create a project…

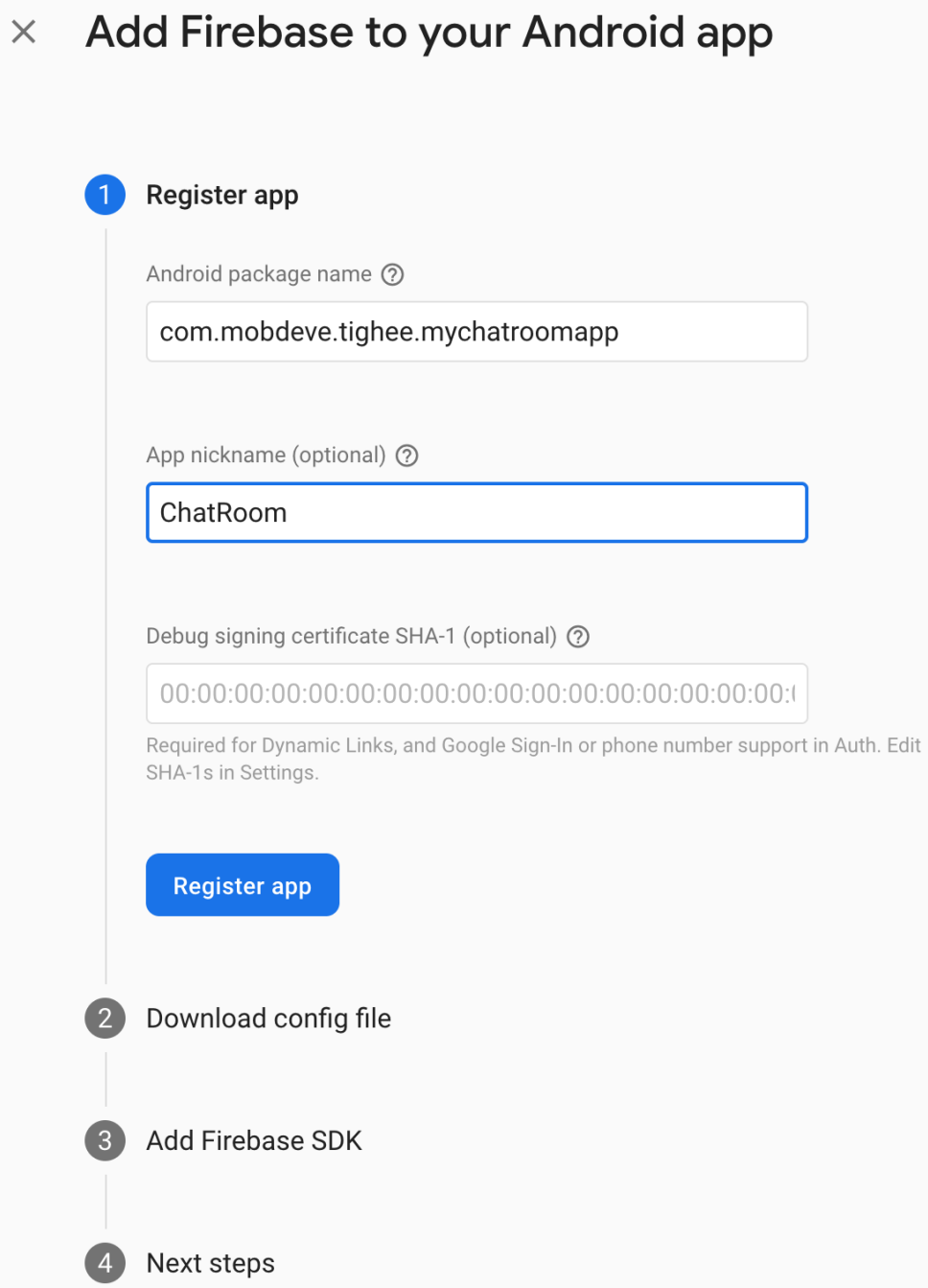
- You'll be prompted to
  - Name the project
  - Configure Google Analytics





Then add an Android app

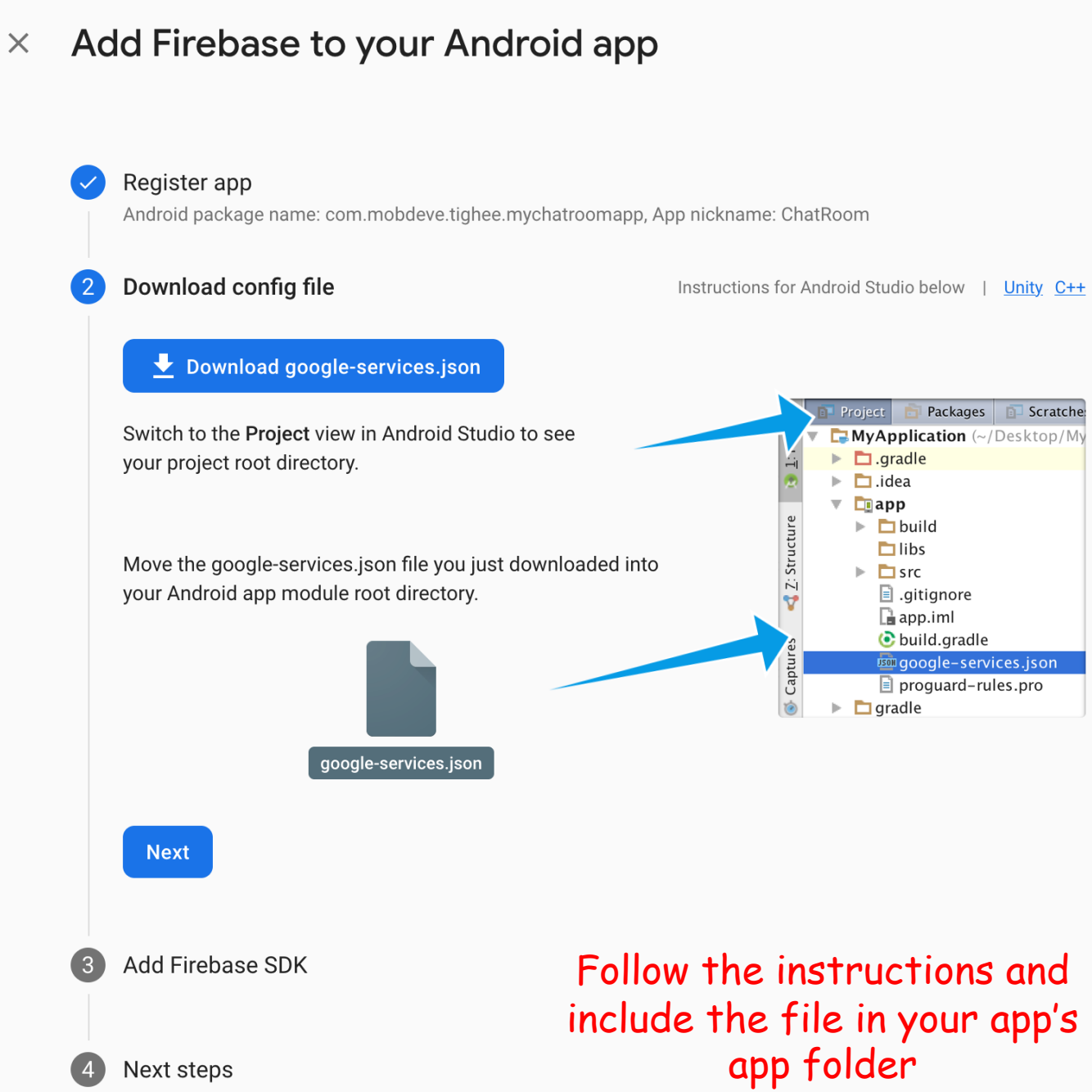




Supply the info asked

- You only have to supply the package name
- Make sure to enter the exact package name of your app
  - Package name value is case-sensitive
  - Cannot be changed after it's registered with your Firebase project

Download the config file



Follow the instructions and include the file in your app's app folder

- A Firebase config file associates an app with a specific Firebase project and its resources (databases, storage buckets, etc.)
  - Includes parameters required by Firebase and Google services for communication

You can always download this file from the app's page in the console of Firebase

Modify your Gradle files accordingly

After doing so, sync your gradle

3 Add Firebase SDK

Instructions for Gradle | Unity C++

The Google services plugin for Gradle loads the `google-services.json` file you just downloaded. Modify your build.gradle files to use the plugin.

Project-level build.gradle (`<project>/build.gradle`):

```
buildscript {
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }
  dependencies {
    ...
    // Add this line
    classpath 'com.google.gms:google-services:4.3.10'
  }
}

allprojects {
  ...
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    ...
  }
}
```

Make sure to remove jcenter() if it is there

Java Kotlin

App-level build.gradle (`<project>/<app-module>/build.gradle`):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
  // Import the Firebase BoM
  implementation platform('com.google.firebase:firebase-bom:28.3.1')

  // Add the dependency for the Firebase SDK for Google Analytics
  // When using the BoM, don't specify versions in Firebase dependencies
  implementation 'com.google.firebase:firebase-analytics'

  // Add the dependencies for any other desired Firebase products
  // https://firebase.google.com/docs/android/setup#available-libraries
}
```

By using the Firebase Android BoM, your app will always use compatible Firebase library versions. Learn more

Finally, press "Sync now" in the bar that appears in the IDE:

Gradle files have changed sin Sync now

Previous Next

Or… you could just use the Firebase Assistant

Android Studio File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

Tasks & Contexts
Save as Live Template...
Generate JavaDoc...
IDE Scripting Console
Create Command-line Launcher...
XML Actions
JShell Console...
Groovy Console...
Kotlin
AVD Manager
SDK Manager
Resource Manager
Troubleshoot Device Connections
Firebase
App Links Assistant
Layout Inspector

Assistant | Firebase

Firebase

Firebase gives you the tools and infrastructure from Google to help you develop, grow and earn money from your app. Learn more

Analytics
Measure user activity and engagement with free, easy, and unlimited analytics. More info
Get started with Google Analytics
Get started with Google Analytics [KOTLIN]

Authentication
Sign in and manage users with ease using popular login providers like Google Sign-In, Facebook, and others. You can even use a custom authentication system. More info

Realtime Database
Store and sync data with this cloud-hosted NoSQL database. Data is synced across all clients in realtime and remains available when your app goes offline. More info

Cloud Firestore
Store and sync your app data with this flexible, scalable NoSQL cloud-hosted database. More info.

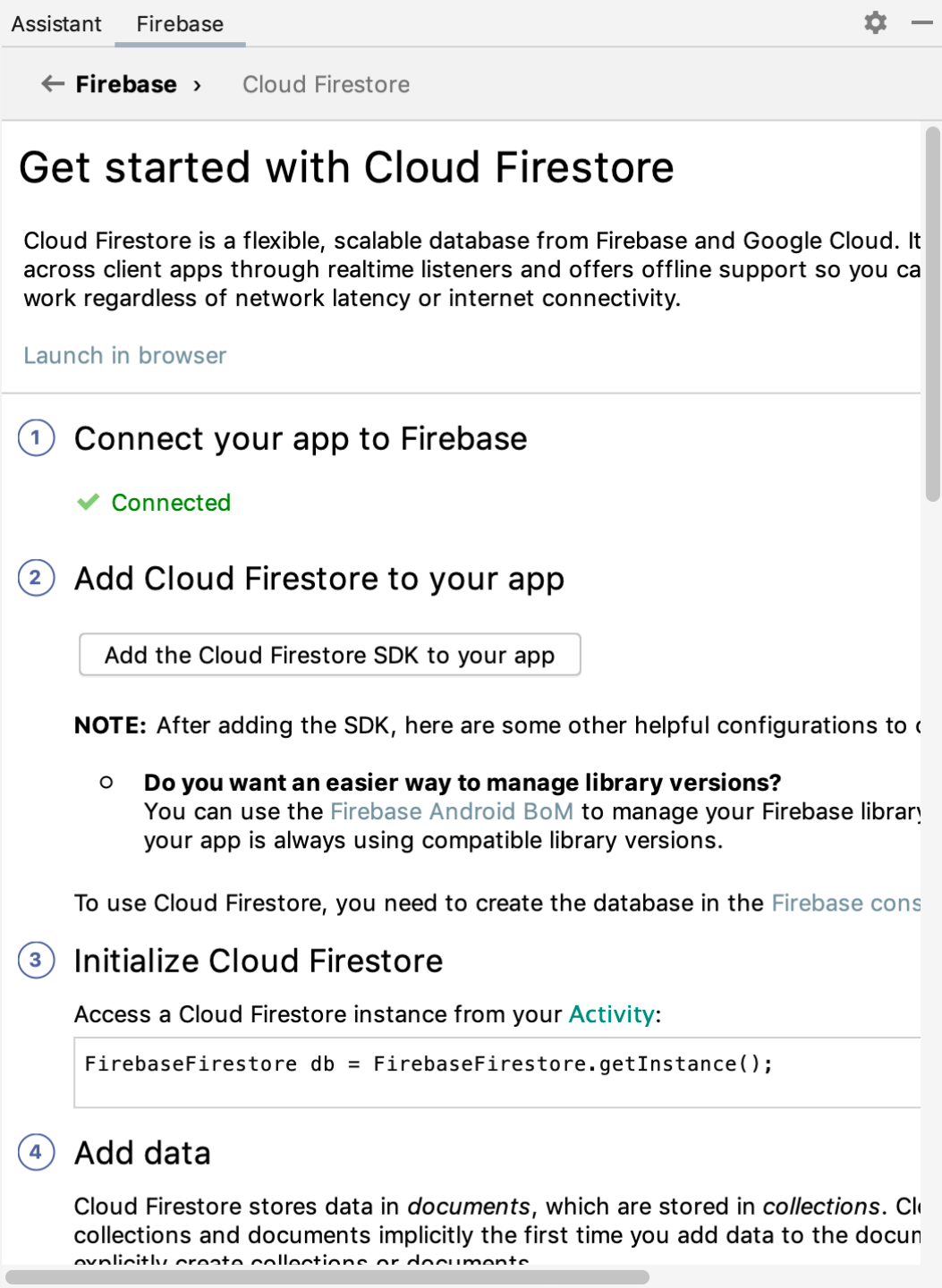
Cloud Storage for Firebase
Store and retrieve large files like images, audio, and video without writing server-side code. More info

Cloud Functions for Firebase

Just make sure you don't have any build errors. You might have jcenter() in your gradle. If ever, simply remove it and using the assistant should work.

Assistant in AS

- Makes life much easier
- Helps in setting up the appropriate dependencies needed
- Also provides code snippets to help you get started



Model + DB Structure

- First, let's define what our DB needs so implementation will be straight forward
- We want…
  - User
    - id
    - username
  - Message
    - id
    - username
    - message
    - timestamp

Given the simplicity of the User model, we won't create it our app, but we will create the Message model

We don't need to define this in our Firestore DB as adding to a non-existing collection creates the collection.

We're also going to auto-increment the ID -- something Realtime storage can't easily do

Model

- Our model follows a standard approach
  - Define attributes
  - Have a constructor
  - Declare getters and setters
- We also need to declare a blank default constructor

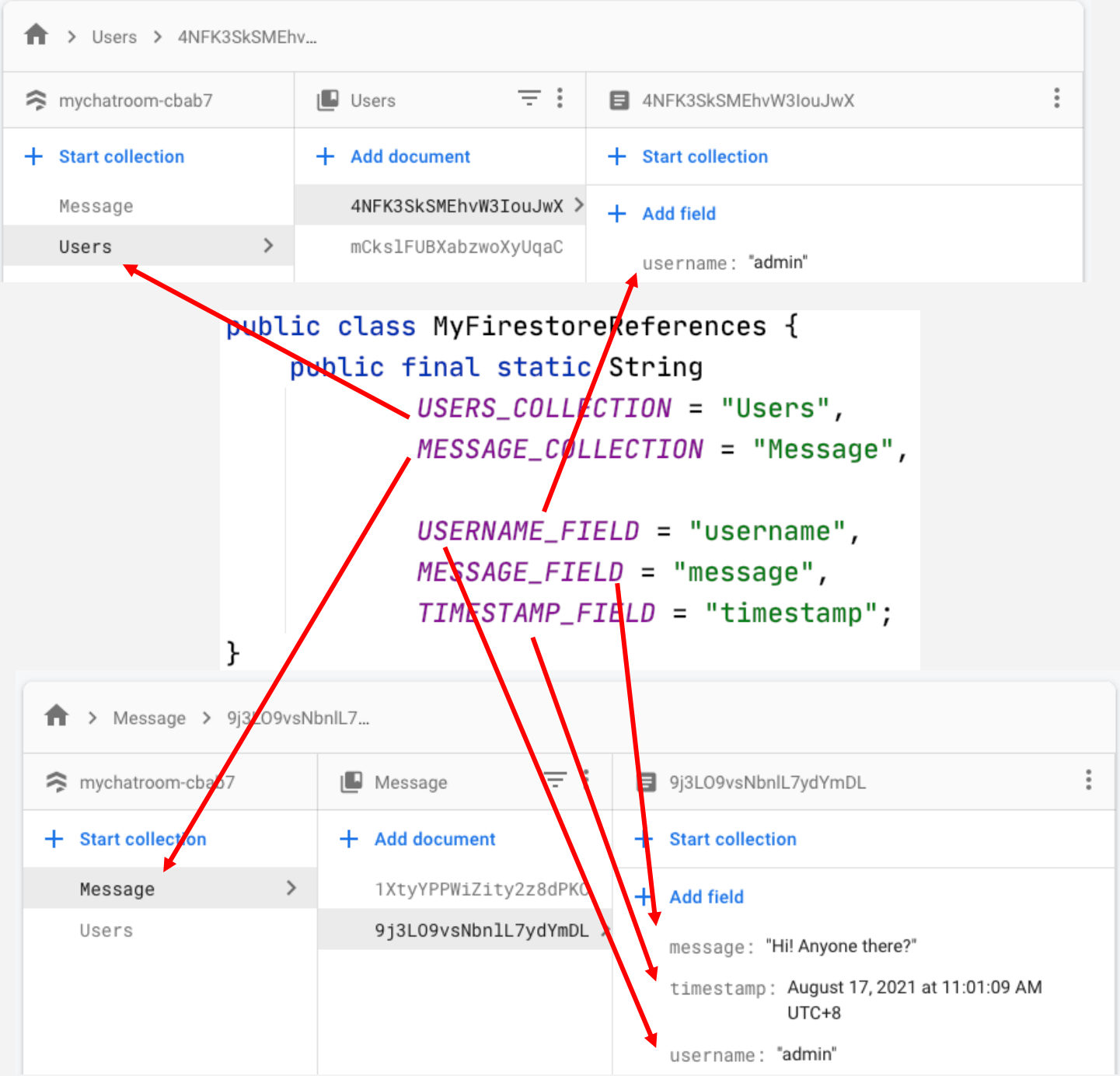
```
public class Message {
    private long id;
    private String username;
    private String message;
    private @ServerTimestamp Date timestamp;

    public Message() {

    }

    public Message(String username, String message, long id, Date timestamp) {
        this.username = username;
        this.message = message;
        this.id = id;
        this.timestamp = timestamp;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }
```

This is needed by Firebase

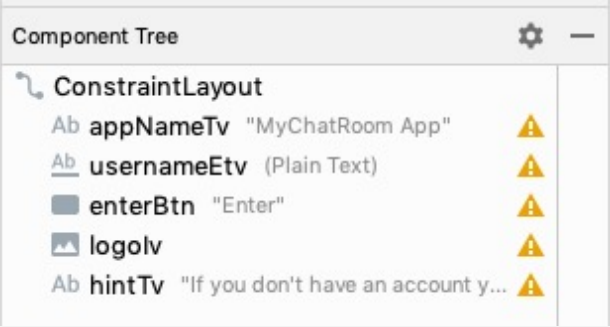
Getters and setters for each of our methods

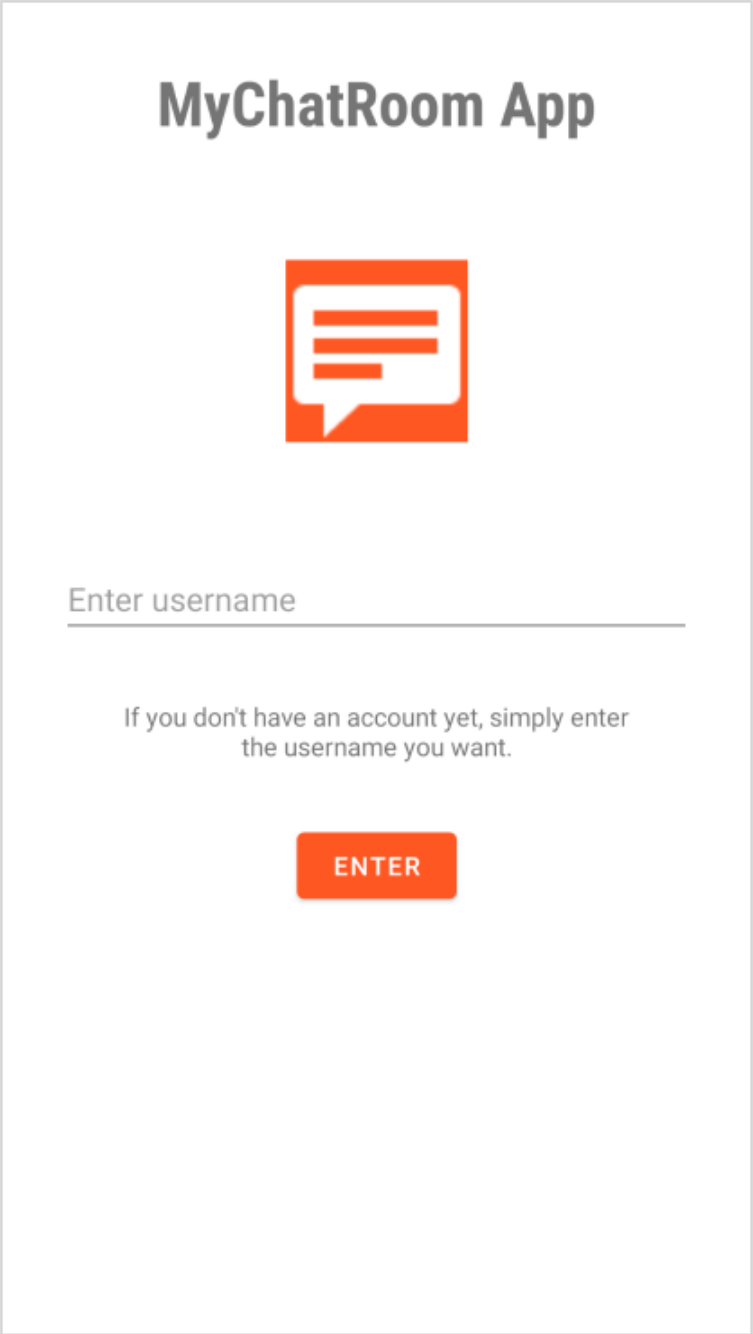
DB References

- Just like during the discussion of SQLite, we might want to create a class that holds all our DB references, so we don't make any spelling errors



Login UI + Logic





- For the first activity, we want the user to first input a username
- On press of the enter button, we'll want to check if the username is in the DB
  - If its not there, we'll want to create it
  - If it is there, then we proceed and start the Chatroom Activity by sending an intent with the username

RECALL: SQLite Utilization

- When we started with SQLite, we had the following steps for implementation:
  - Get a reference to the database (read/write)
  - Perform operations
  - Close the database

For Firestore, we only need to perform the first two

Firestore Referencing

- In Firestore, we'd want to create references to our DB and/or our collections
- We can do so as such:

```
FirebaseFirestore db = FirebaseFirestore.getInstance();
CollectionReference usersRef = db.collection(MyFirestoreReferences.USERS_COLLECTION);
```

getInstance() utilizes our google-services.json file we placed into our project awhile ago, so make sure everything is set up or else this won't result to anything

Firestore Querying

- For querying, we can utilize the Query object provided by Firebase
  - This helps in filtering and sorting
- We can do so as such:

```
Query query = usersRef.whereEqualTo(
        MyFirestoreReferences.USERNAME_FIELD,
        username);
```

With Firestore, we can also chain filters ☺

Firestore Querying

- To execute the query, we simply use the .get() method call and attach an OnCompleteListener to know when our query finishes

```
query.get().addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
    @Override
    public void onComplete(@NonNull Task<QuerySnapshot> task) {
        if (task.isSuccessful())
            if(task.getResult().isEmpty())
                showNewUserDialog(usersRef, username);
            else
                moveToChatRoomActivity(username);
        else
            Log.d(TAG, "Error getting documents: ", task.getException());
    }
});
```

Firestore Writing

- To add a document to a Firestore DB, we'd want to define a HashMap (key + value pairs) and send that off using a collection reference

```
Map<String, Object> data = new HashMap<>();
data.put(MyFirestoreReferences.USERNAME_FIELD, username);

usersRef.add(data)
    .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
        @Override
        public void onSuccess(DocumentReference documentReference) {
            Log.d(TAG, "DocumentSnapshot written with ID: " + documentReference.getId());
        }
    })
    .addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            Log.w(TAG, "Error adding document", e);
        }
    });
```

Like the ContentValues with SQLite

We write the data using the add() method call

We can also add the respective OnSuccess or OnFailure listeners to have our app adjust accordingly

Questions? ☺

Thanks everyone!

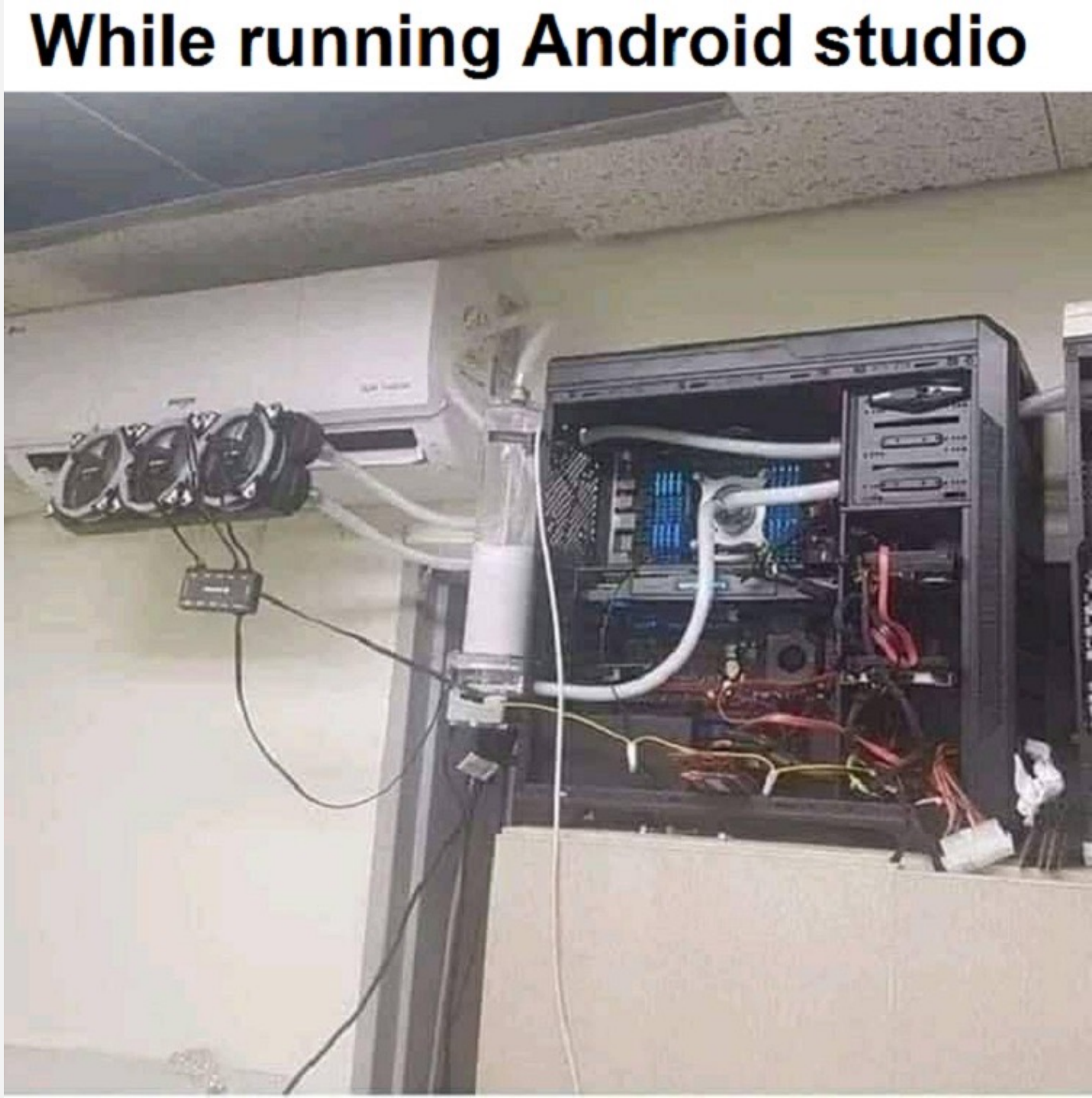


Image sauce:
https://www.facebook.com/ProgrammersCreateLife/photos/a.241809332534619/4043704579011723/