Google Developer Student Clubs

Introduction to Firebase

Firebase as a backend

Stelios Papamichail
@MikePapamichail

What is Firebase?

Purpose & product

“Firebase is an app development platform that helps you build and grow apps and games users love. Backed by Google and trusted by millions of businesses around the world.”

What services does it provide?

Products & solutions overview

- **Build Services:** Accelerate and scale app development without managing infrastructure (we're going to work with this one)
- **Release & Monitoring:** Release with confidence and monitor performance and stability
- **Engagement Tools:** Boost engagement with rich analytics, A/B testing, and messaging campaigns

Cloud Firestore

`Store and sync app data at global scale`

- NoSQL **document based** database built for global apps
- Allows us to query & structure our data using **collections** & **documents** to form hierarchies
- Offers mobile & web SDKs with multiple security features, allowing us to be **serverless**
- Easily sync data between clients & offer offline functionality
- Scalable thanks to Google's storage infrastructure

Cloud Storage

Store and serve content with ease

- Designed for serving & storing user generated content (i.e., videos, photos)
- **Robust uploads & downloads:** Automatically pauses & resumes downloads/uploads based on the user's connectivity
- Easy integration with **FirebaseAuth** for simple & intuitive access control
- "Effortlessly" scalable (same tech that powers **Spotify** & **Google Photos**)

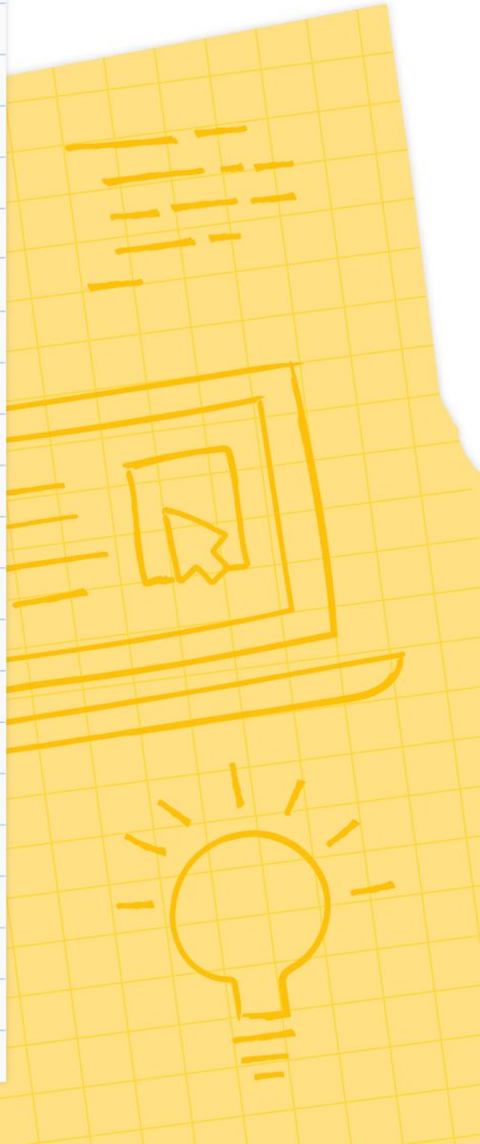
Let’s put our knowledge to practice

Database Example

Using Firestore & Storage for our sample app

We're now going to see how to use **Firestore** in order to build our database. We will also look at leveraging **Cloud Storage** for storing & retrieving artwork images!

Due to time constraints, we're going to keep the database scheme very simple. We will explore relationships **very** briefly!



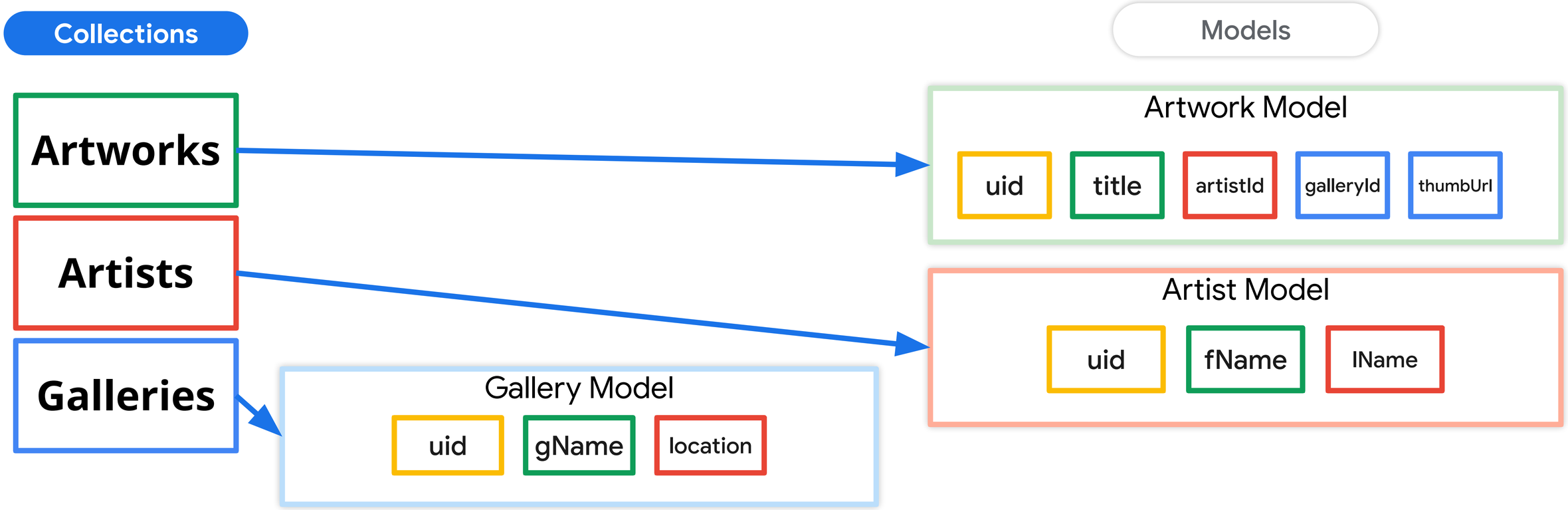
Database Design Tips

Things to keep in mind when designing your database

- Determine the purpose of your system
- **Identify the entities** that your system will require
- Identify what **information** your entities will need
- Find existing **relationships between your entities** & consider possible new ones
- Think about "*What would the user want to know and do with our data?*"
- Determine **Keys** & possible **Foreign Keys** for your entities
- Select the **appropriate data types** for your entity's fields

Database Outline

Diagram of our sample database scheme



Communicating with Firebase

HTTP requests

In order to get, add, remove or update data to and from our database, we will need to make appropriate HTTP calls to it.

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Some of these actions are:

- The **GET** method requests a representation of the specified resource. Requests using GET should only retrieve data.
- The **POST** method submits an entity to the specified resource, often causing a change in state or side effects on the server.
- The **PUT** method replaces all current representations of the target resource with the request payload.
- The **DELETE** method deletes the specified resource.

More about HTTP methods

Communicating with Firebase

The JSON format

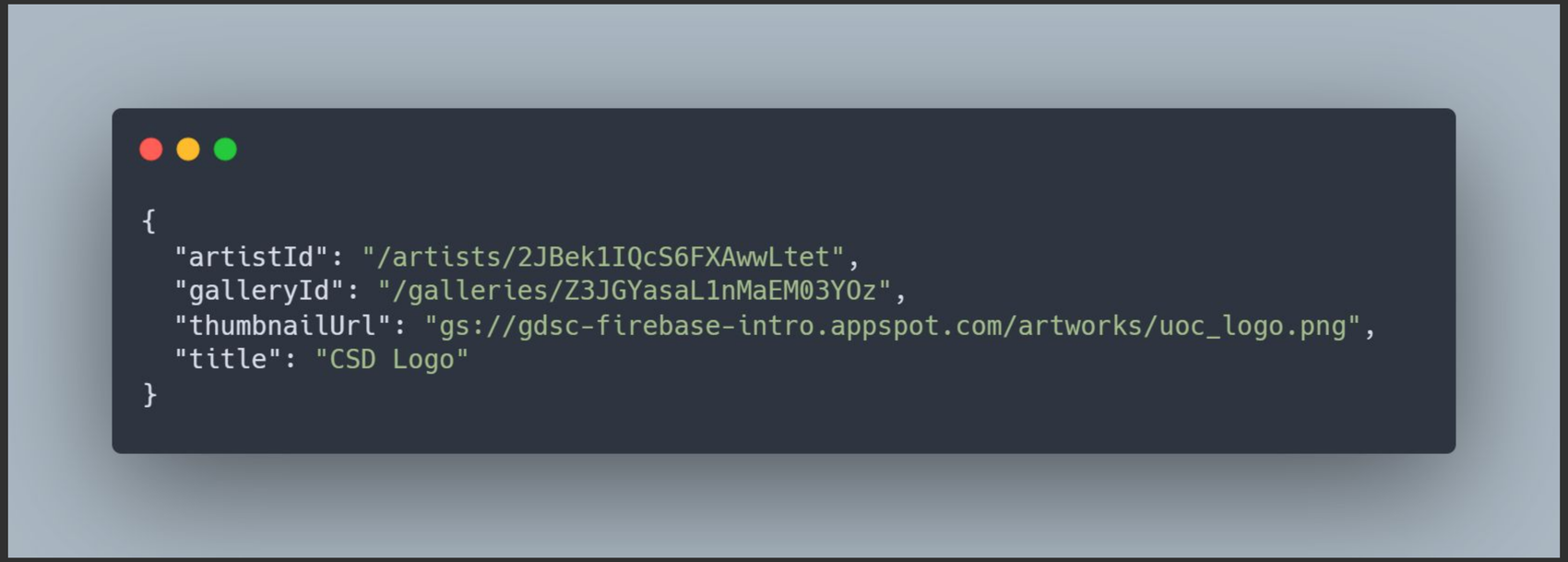
JSON stands for **JavaScript Object Notation**. It's a lightweight format for storing and transporting data and is often used when data is sent from a server to some client (i.e., mobile app, web page, etc.). It's "self-describing" and easy to understand.

JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

More about JSON

Example of a JSON response from our artworks collection



```
{
  "artistId": "/artists/2JBek1IQcS6FXAwwLtet",
  "galleryId": "/galleries/Z3JGYasaL1nMaEM03YOz",
  "thumbnailUrl": "gs://gdsc-firebase-intro.appspot.com/artworks/uoc_logo.png",
  "title": "CSD Logo"
}
```

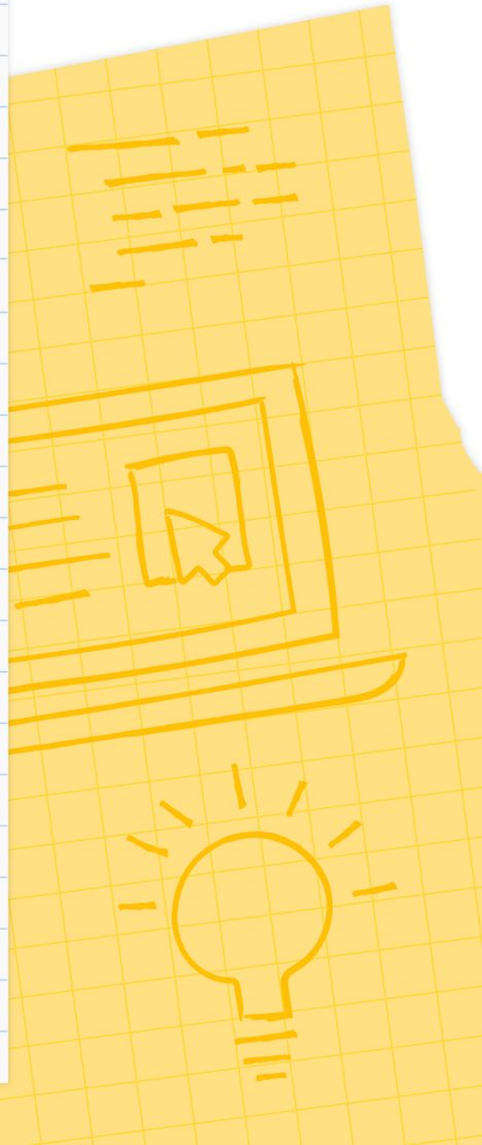
Firebase SDK

Development environment setup

Before we start making requests to our Firestore database, we first need to **add the Firebase SDK to our project**.

The process may differ based on your selected programming language & platform, so **visit the official documentation** using the link on the bottom right. For this presentation, we'll follow the steps for Java!

Official docs



Adding Firebase Admin SDK

Development environment setup

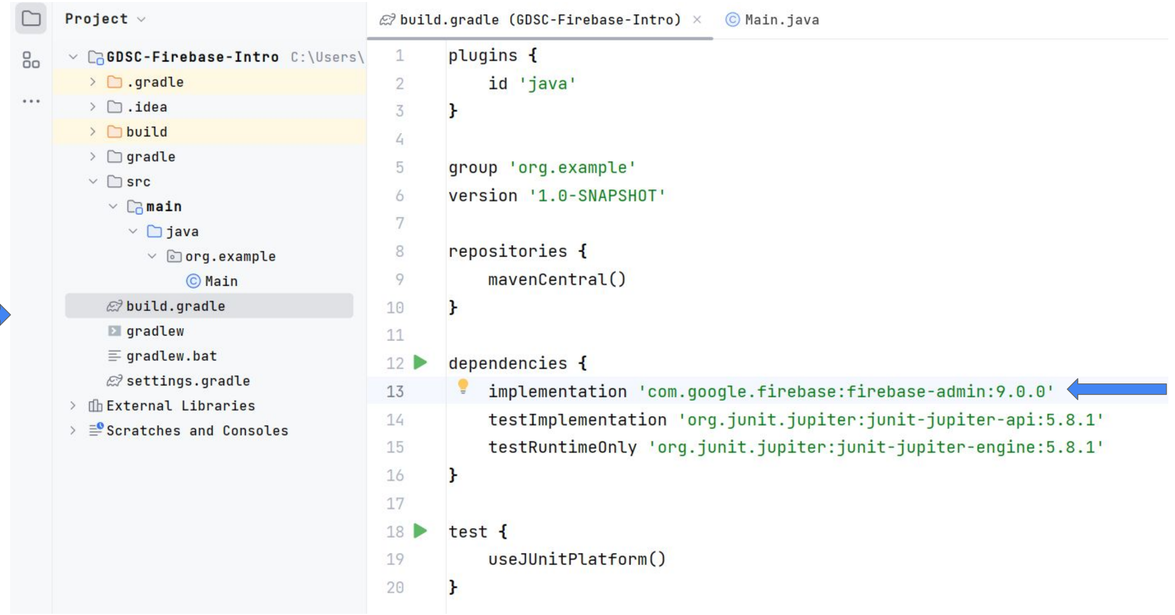
Node.js | Java | Python | Go | C#

The Firebase Admin Java SDK is published to the Maven central repository. To install the library, declare it as a dependency in your `build.gradle` file:

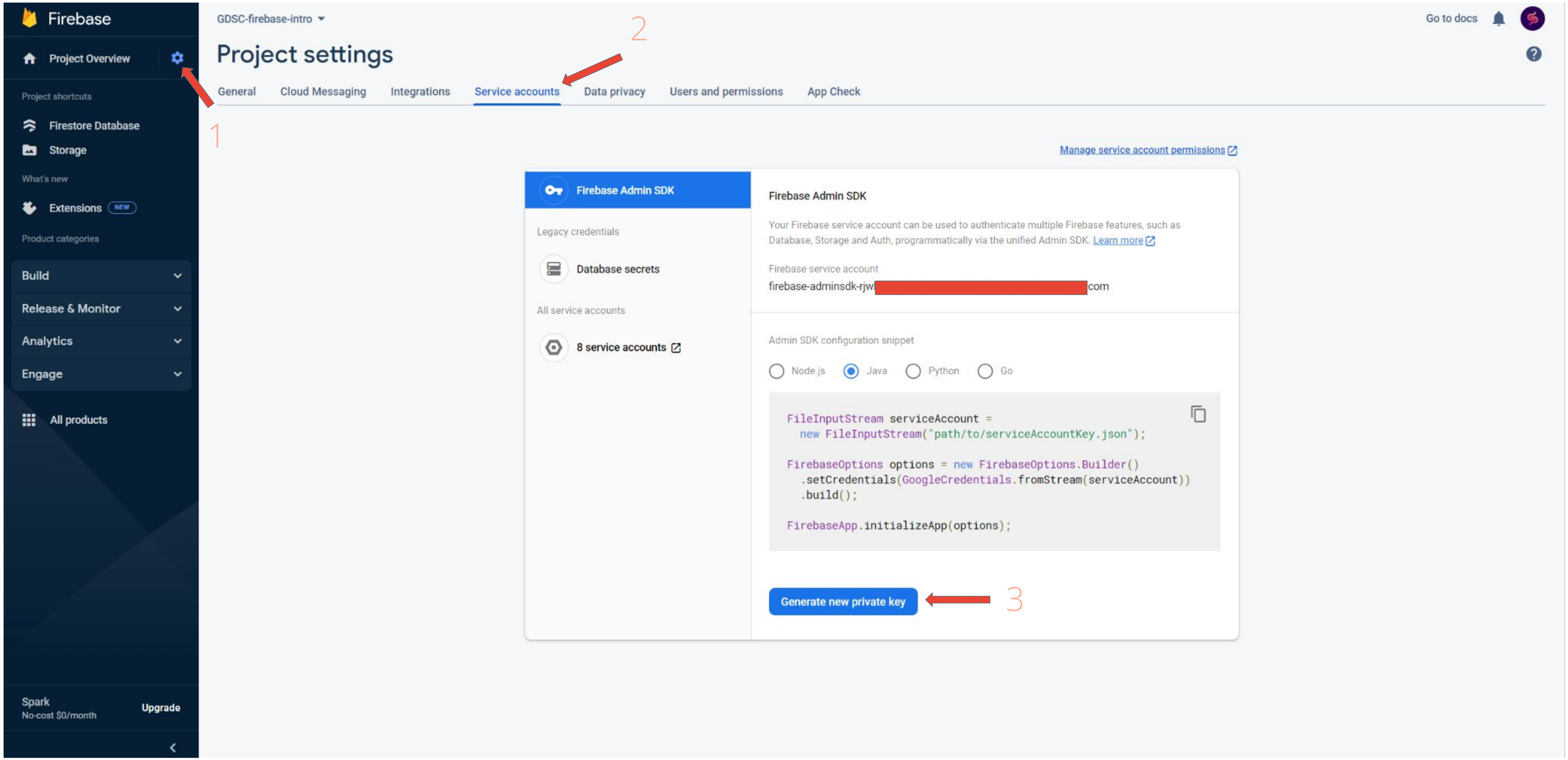
```
dependencies {
  implementation 'com.google.firebase:firebase-admin:9.0.0'
}
```

If you use Maven to build your application, you can add the following dependency to your `pom.xml`:

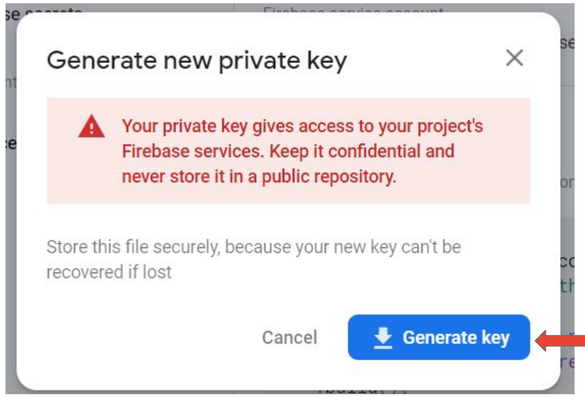
```
<dependency>
  <groupId>com.google.firebase</groupId>
  <artifactId>firebase-admin</artifactId>
  <version>9.0.0</version>
</dependency>
```

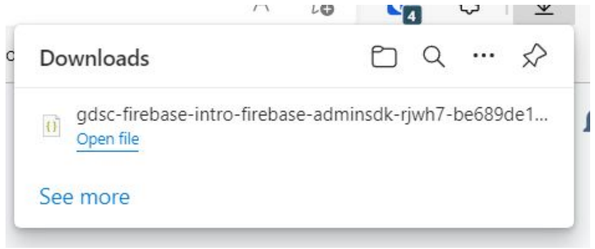


Official docs



Google Developer Student Clubs





build.gradle | Main.java

Module JDK is not defined

```java
import java.io.FileNotFoundException;
import java.io.IOException;

no usages
public class Main {
    no usages
    public static void main(String[] args) {

        FileInputStream serviceAccount =
                null;
        try {
            serviceAccount = new FileInputStream( name: "C:\\Users\\Steli\\Desktop\\GDSC-Firebase-Intro\\gdsc-firebase-intro-firebase-adminsdk-key.json");
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        }

        FirebaseOptions options = null;
        try {
            options = new FirebaseOptions.Builder()
                    .setProjectId("gdsc-firebase-intro")
                    .setCredentials(GoogleCredentials.fromStream(serviceAccount))
                    .build();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        FirebaseApp.initializeApp(options);

    }
}
```

Accessing our database

Using the Firebase SDK

Now that we have initialized our Firebase project, we can start using our Firestore & Cloud Storage instances! First and foremost, let's get a **reference to our database**:

```
Firestore db = FirestoreClient.getFirestore();
```

With our Firestore reference at hand, we can now begin communicating with our database!

Let's see an example of how we can **request all the documents/entries of our artworks collection**, and print each document's fields to the console using Java!

More about JSON

Fetching all documents in our ‘artworks’ collection

```
// asynchronously retrieve all users
ApiFuture<QuerySnapshot> query = db.collection("artworks").get();

// query.get() blocks on response
QuerySnapshot querySnapshot = null;
try {
    querySnapshot = query.get();
    List<QueryDocumentSnapshot> documents = querySnapshot.getDocuments();
    for (QueryDocumentSnapshot document : documents) {
        System.out.println("ArtworkId: " + document.getId());
        DocumentReference artistRef = (DocumentReference) document.get("artistId");
        System.out.println("Related artist path: " + artistRef.getPath());
        DocumentReference galleryRef = (DocumentReference) document.get("galleryId");
        System.out.println("Related gallery path: " + galleryRef.getPath());
        System.out.println("Thumbnail url: " + document.getString("thumbnailUrl"));
        System.out.println("Title: " + document.getString("title"));
    }
} catch (InterruptedException | ExecutionException e) {
    throw new RuntimeException(e);
}
```

Data manipulation

Making CRUD* requests

To **create or overwrite** a single document, use the **set()** method:

```
        HashMap<String, Object> galleryInfo = new HashMap<>();
        galleryInfo.put("gName", "Dio Horia");
        galleryInfo.put("location", new GeoPoint(10.051, 110.000));
        // Add a new document (asynchronously) in collection "galleries" with id "dio_horia"
        ApiFuture<WriteResult> future =
db.collection("galleries").document("dio_horia").set(galleryInfo);
        try {
            // future.get() blocks on response
            System.out.println("New entry created successfully!\nUpdate time : " +
future.get().getUpdateTime());
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
```

If the document does not exist, it will be created. If the document does exist, its contents will be overwritten with the newly provided data, unless you specify that the data should be merged into the existing document.

*CRUD == Create Read Update Delete

Data manipulation

Making CRUD requests

Let's see how we can specify that the data that we are sending to the preexisting document, should be **merged into the existing document**.

```
        HashMap<String, Object> galleryInfo = new HashMap<>();
        galleryInfo.put("gName", "Dio Horia");
        galleryInfo.put("location", new GeoPoint(10.051, 110.000));
        galleryInfo.put("ownerName", "Manolis K.");
        // Add a new document (asynchronously) in collection "galleries" with id "dio_horia"
        ApiFuture<WriteResult> future =
db.collection("galleries").document("dio_horia").set(galleryInfo, SetOptions.merge());
        try {
            // future.get() blocks on response
            System.out.println("New entry created successfully!\nUpdate time : " +
future.get().getUpdateTime());
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
```

What changed?

Here, we are specifying the **update strategy** to the set() method via the **SetOptions** class!

Data manipulation

Adding documents using custom & auto-generated IDs

When you use **set()** to create a document, you must specify an ID for the document to create. For example:

```
db.collection("galleries").document("dio_horia").set(galleryInfo);
```

But sometimes there isn't a meaningful ID for the document, and it's more convenient to **let Cloud Firestore auto-generate an ID** for you. You can do this by calling add():

```
db.collection("galleries").add(galleryInfo);
```

Adding/Updating a document in the “galleries” collection (with manual IDs or auto-generated ones)

```
        HashMap<String, Object> galleryInfo = new HashMap<>();
        if (!autoGenId) {
            galleryInfo.put("gName", "Dio Horia");
            galleryInfo.put("location", new GeoPoint(10.051, 110.000));
            // Add a new document (asynchronously) in collection "galleries" with id "dio_horia"
            ApiFuture<WriteResult> future =
db.collection("galleries").document("dio_horia").set(galleryInfo/*, SetOptions.merge()*/);
            try {
                // future.get() blocks on response
                System.out.println("New entry created successfully!\nUpdate time : " +
future.get().getUpdateTime());
            } catch (InterruptedException | ExecutionException e) {
                throw new RuntimeException(e);
            }
        } else {
            galleryInfo.put("gName", "AutoGen Gallery");
            galleryInfo.put("location", new GeoPoint(0.00, 170.00));
            // Add a new document (asynchronously) in collection "galleries" with auto-generated id
            ApiFuture<DocumentReference> docRef = db.collection("galleries").add(galleryInfo);
            try {
                // future.get() blocks on response
                System.out.println("Added document with ID: " + docRef.get().getId());
            } catch (InterruptedException | ExecutionException e) {
                throw new RuntimeException(e);
            }
        }
```

Data manipulation

Removing a document from a collection

- To delete a document, use the **delete()** method! When you delete a document, Cloud Firestore does not automatically delete the documents within its subcollections. You can still access the subcollection documents by reference.
- For example, you can access the document at path */mycoll/mydoc/mysubcoll/mysubdoc* even if you delete the ancestor document at */mycoll/mydoc*.
- Non-existent ancestor documents appear in the console, but they do not appear in query results and snapshots.
- If you want to delete a document and all the documents within its subcollections, you must do so manually. For more information, see Delete Collections.

Removing a document from the ‘artists’ collection

```
// asynchronously delete a document
ApiFuture<WriteResult> writeResult = db.collection("artists").document("pls-dont-do-
this").delete();

try {
    System.out.println("Entry deleted successfully!\nUpdate time : " +
    writeResult.get().getUpdateTime());
} catch (InterruptedException | ExecutionException e) {
    throw new RuntimeException(e);
}
```

Cloud Storage images

Fetching images stored in cloud storage

Images are stored in **Blobs**! Firebase automatically generates a default blob for us, so we're going to work with that. If you would like to create custom blobs & learn more about them, visit the docs.

The files in our cloud storage instance are accessible through various ways. One of them, is using a URL that follows the following format: *"https://firebasestorage.googleapis.com/v0/b/{$blob_name}/o/"*, where {$blob_name} should be replaced with the blob's path that we want to use.

The default blob's path can be found on the "Storage" tab, above our files! For our sample project, the path would be *"gdsc-firebase-intro.appspot.com"*. So, the resulting base URL for our storage would be:
https://firebasestorage.googleapis.com/v0/b/gdsc-firebase-intro.appspot.com/o/

Storage docs

Cloud Storage images

Fetching images stored in cloud storage

Now that we have our base URL, we can access specific files by appending their path to the end of the base URL. **Important to note** is that file paths that are appended to the base URL, should be **URL encoded**. This means that, for example, the "/" character will have to be replaced with "%2F"!

For example, our default storage blob, has a folder named "artworks" with an image inside called uoc_logo.png. In order to access it, we would encode the path and append it like so:
https://firebasestorage.googleapis.com/v0/b/gdsc-firebase-intro.appspot.com/o/artworks%2Fuoc_logo.png
But this isn't enough yet! We also need to specify the ?alt=media query parameter so that our clients can view the image as intended. By adding the query parameter, we get our final URL:
https://firebasestorage.googleapis.com/v0/b/gdsc-firebase-intro.appspot.com/o/artworks%2Fuoc_logo.png?alt=media

Storage docs

Cloud Storage images

Fetching images stored in cloud storage

So how are we going to do all that programmatically? Here's a sample function:

```
    private static String fetchArtworkUrl(Firestore db) {
        String baseUrl = "https://firebasestorage.googleapis.com/v0/b/gdsc-firebase-
intro.appspot.com/o/";
        // asynchronously retrieve a specific artwork
        DocumentReference documentReference =
db.collection("artworks").document("oegT8sMEZf60rooWgZfE");
        // asynchronously retrieve the document
        ApiFuture<DocumentSnapshot> future = documentReference.get();
        // future.get() blocks on response
        DocumentSnapshot document = null;
        try {
            document = future.get();
        } catch (InterruptedException | ExecutionException e) {
            throw new RuntimeException(e);
        }
        if (document.exists()) {
            return baseUrl + document.getString("thumbnailUrl").replace("/","%2F") + "?alt=media";
        } else {
            return "";
        }
    }
```

Once we have the image's URL, we can render on our UI (which is out of scope)!

Storage docs

Thank you for your time!

`Hope you learned something new today`

There's a **TON** of awesome things that we couldn't cover today, so if you're interested in diving a bit deeper, then **visit the docs** by clicking on the various links at the bottom-right side of the slides (or using your *google-fu*)!

Special thanks to the @GoogleDevs that are making initiatives such as this possible!

Don't forget to follow the GDSC UoC on our social media & join our **discord server** to stay up to date with upcoming events!

Instagram Twitter LinkedIn Discord Server

Google Developer Student Clubs