CUI Abbottabad

Department of Computer Science

Mobile Application Development

Lecture Topic:

React Native

Agenda

- Introduction to Firestore Database
- Performing CRUD operations using firestore database
- Retrieving data using firestore database

Cloud Firestore

- Use our flexible, scalable NoSQL cloud database to store and sync data for client- and server-side development.

- Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. Like Firebase Realtime Database, it keeps your data in sync across client apps through realtime listeners and offers offline support for mobile and web so you can build responsive apps that work regardless of network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions..

Key capabilities

- *Flexibility* The Cloud Firestore data model supports flexible, hierarchical data structures. Store your data in documents, organized into collections. Documents can contain complex nested objects in addition to subcollections.
- *Expressive querying* In Cloud Firestore, you can use queries to retrieve individual, specific documents or to retrieve all the documents in a collection that match your query parameters. Your queries can include multiple, chained filters and combine filtering and sorting. They're also indexed by default, so query performance is proportional to the size of your result set, not your data set.

Key capabilities

- *Realtime updates* Like Realtime Database, Cloud Firestore uses data synchronization to update data on any connected device. However, it's also designed to make simple, one-time fetch queries efficiently.
- *Offline support* Cloud Firestore caches data that your app is actively using, so the app can write, read, listen to, and query data even if the device is offline. When the device comes back online, Cloud Firestore synchronizes any local changes back to Cloud Firestore.

- **Key capabilities**
- *Designed to scale* Cloud Firestore brings you the best of Google Cloud's powerful infrastructure: automatic multi-region data replication, strong consistency guarantees, atomic batch operations, and real transaction support. We've designed Cloud Firestore to handle the toughest database workloads from the world's biggest apps

How firestore works

- Designed Cloud Firestore is a cloud-hosted, NoSQL database that your Apple, Android, and web apps can access directly via native SDKs. Cloud Firestore is also available in native Node.js, Java, Python, Unity, C++ and Go SDKs, in addition to REST and RPC APIs.

- Following Cloud Firestore's NoSQL data model, you store data in documents that contain fields mapping to values. These documents are stored in collections, which are containers for your documents that you can use to organize your data and build queries. Documents support many different data types, from simple strings and numbers, to complex, nested objects.

❖ How firestore works

- ❖ You can also create subcollections within documents and build hierarchical data structures that scale as your database grows. The Cloud Firestore data model supports whatever data structure works best for your app.
- ❖ Additionally, querying in Cloud Firestore is expressive, efficient, and flexible. Create shallow queries to retrieve data at the document level without needing to retrieve the entire collection, or any nested subcollections. Add sorting, filtering, and limits to your queries or cursors to paginate your results. To keep data in your apps current, without retrieving your entire database each time an update happens, add realtime listeners.

How firestore works

- Adding realtime listeners to your app notifies you with a data snapshot whenever the data your client apps are listening to changes, retrieving only the new changes.

- Protect access to your data in Cloud Firestore with Firebase Authentication and Cloud Firestore Security Rules for Android, Apple platforms, and JavaScript, or Identity and Access Management (IAM) for server-side languages.

Get started with Cloud Firestore

- This quickstart shows you how to set up Cloud Firestore, add data, then view the data you just added in the Firebase console.
- Create a Cloud Firestore database
- If you haven't already, create a Firebase project: In the Firebase console, click Add project, then follow the on-screen instructions to create a Firebase project or to add Firebase services to an existing GCP project.
- Navigate to the Cloud Firestore section of the Firebase console. You'll be prompted to select an existing Firebase project. Follow the database creation workflow.

Get started with Cloud Firestore

- Follow the database creation workflow.
- Select a starting mode for your Cloud Firestore Security Rules:

- Test mode
- Good for getting started with the mobile and web client libraries, but allows anyone to read and overwrite your data. After testing, make sure to review the Secure your data section.

- To get started with the web, Apple platforms, or Android SDK, select test mode.

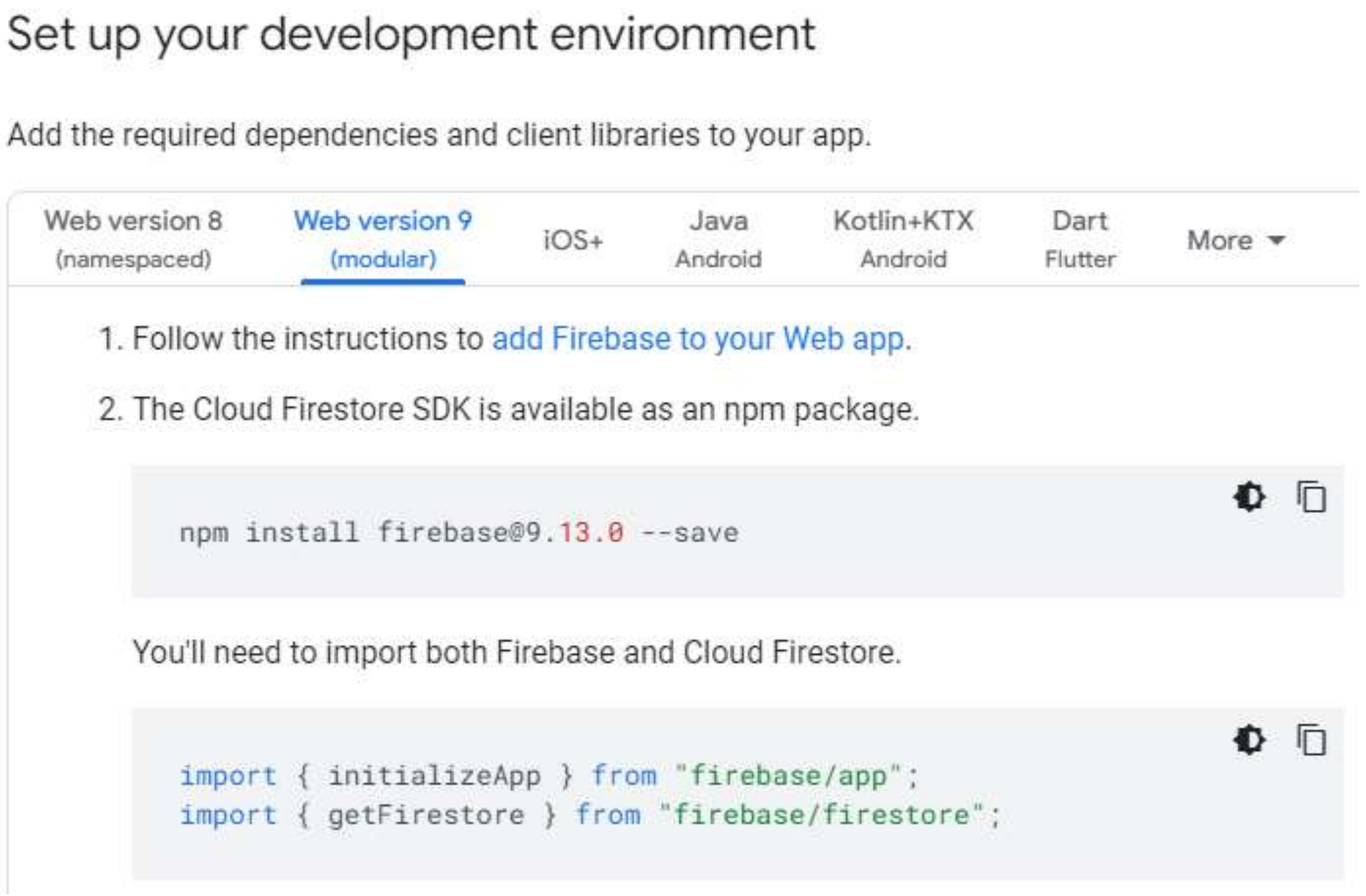
Get started with Cloud Firestore

- Locked mode
- Denies all reads and writes from mobile and web clients. Your authenticated application servers (C#, Go, Java, Node.js, PHP, Python, or Ruby) can still access your database.

- To get started with the C#, Go, Java, Node.js, PHP, Python, or Ruby server client library, select locked mode

Get started with Cloud Firestore

- Select a location for your database.
- This location setting is your project's default Google Cloud Platform (GCP) resource location. Note that this location will be used for GCP services in your project that require a location setting, specifically, your default Cloud Storage bucket and your App Engine app (which is required if you use Cloud Scheduler).
- If you aren't able to select a location, then your project already has a default GCP resource location. It was set either during project creation or when setting up another service that requires a location setting.
- Click Done



Set up your development environment

Add the required dependencies and client libraries to your app.

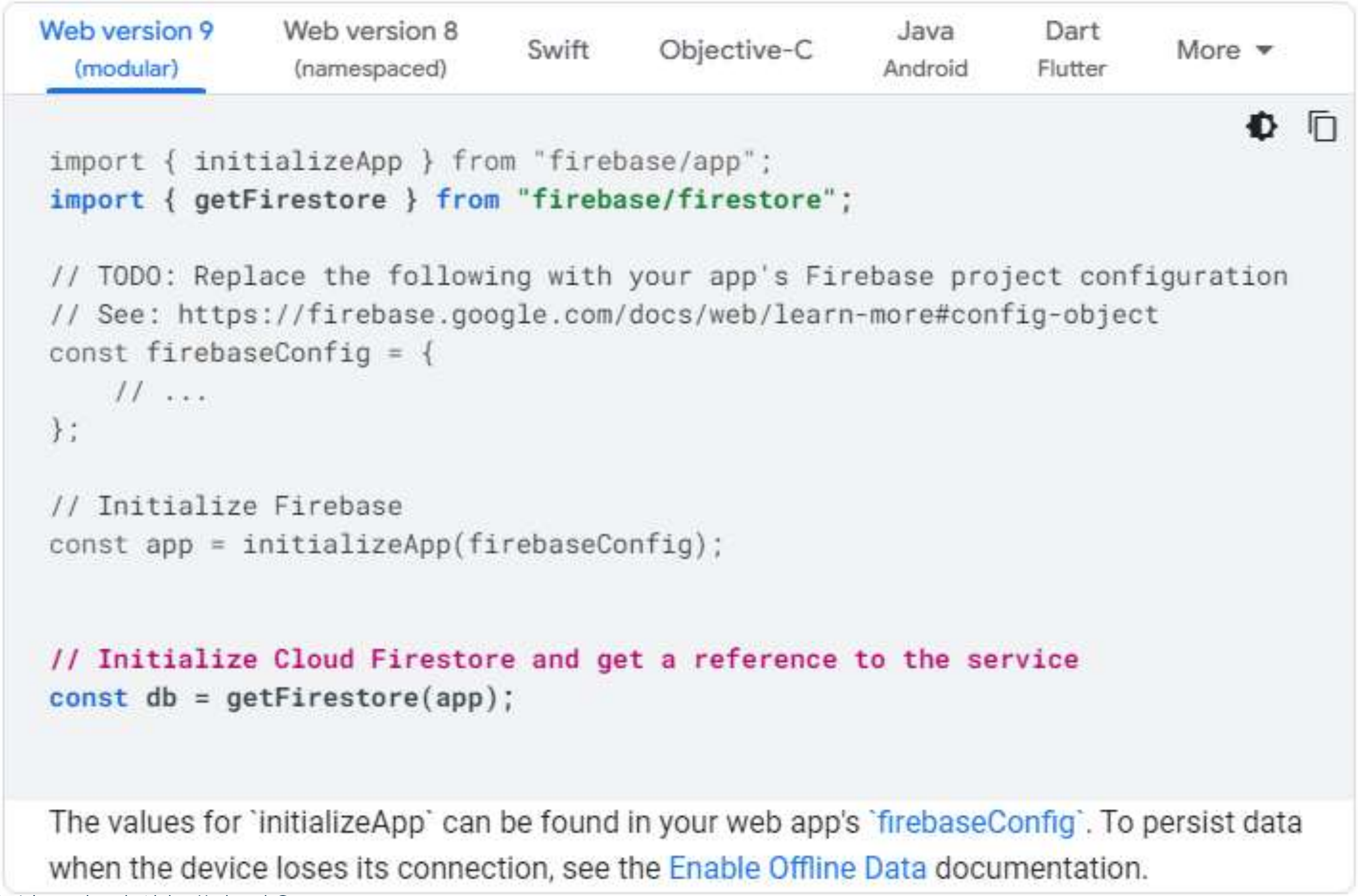
1. Follow the instructions to add Firebase to your Web app.
2. The Cloud Firestore SDK is available as an npm package.

```
npm install firebase@9.13.0 --save
```

You'll need to import both Firebase and Cloud Firestore.

```
import { initializeApp } from "firebase/app";
import { getFirestore } from "firebase/firestore";
```

Initialize Cloud Firestore



Initialize an instance of Cloud Firestore:

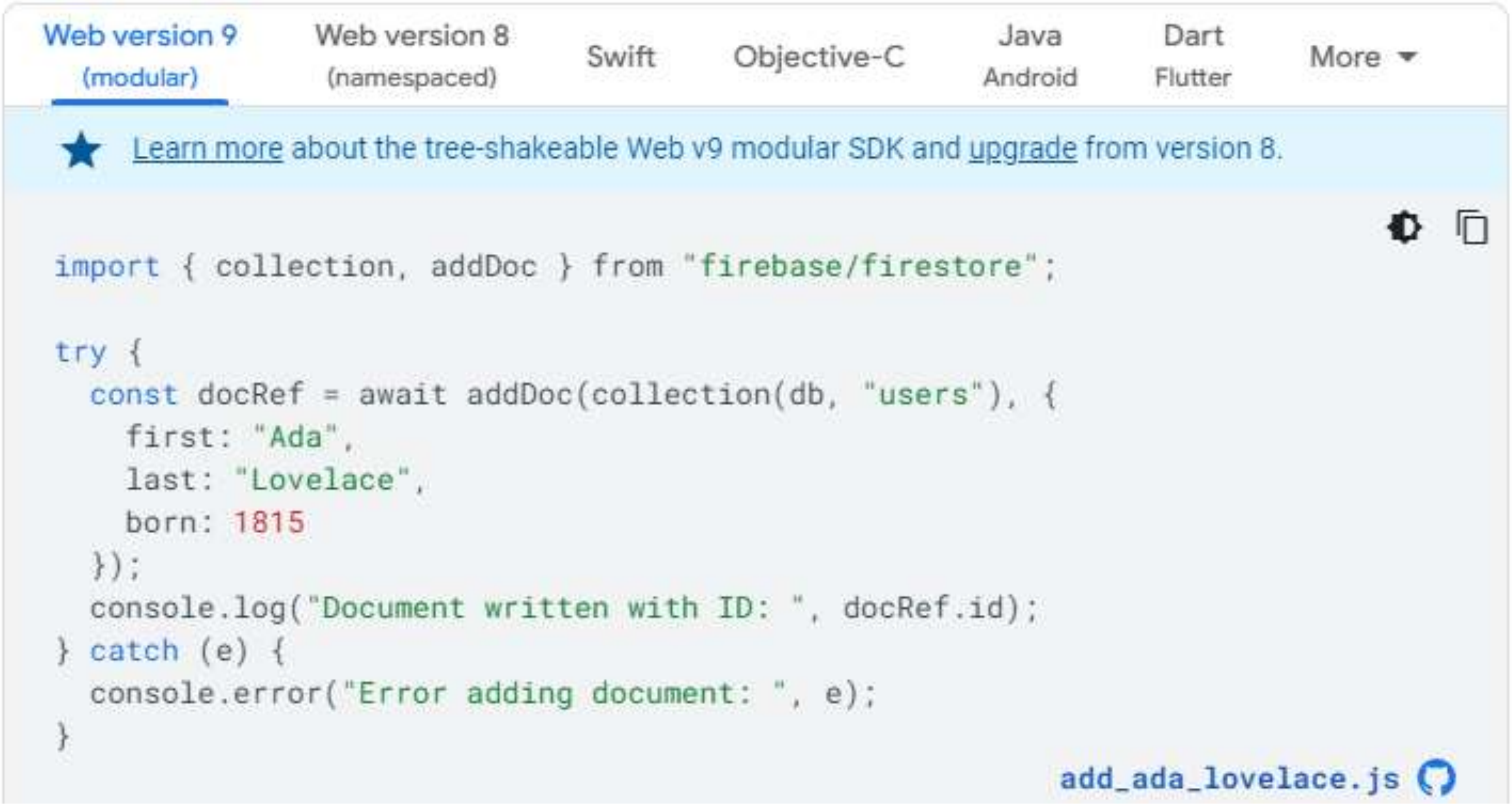
```
import { initializeApp } from "firebase/app";
import { getFirestore } from "firebase/firestore";

// TODO: Replace the following with your app's Firebase project configuration
// See: https://firebase.google.com/docs/web/learn-more#config-object
const firebaseConfig = {
    // ...
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);


// Initialize Cloud Firestore and get a reference to the service
const db = getFirestore(app);
```

The values for `initializeApp` can be found in your web app's `firebaseConfig`. To persist data when the device loses its connection, see the Enable Offline Data documentation.

❖ Add data

- ❖ Cloud Firestore stores data in Documents, which are stored in Collections. Cloud Firestore creates collections and documents implicitly the first time you add data to the document. You do not need to explicitly create collections or documents.

- ❖ Create a new collection and a document using the following example code.

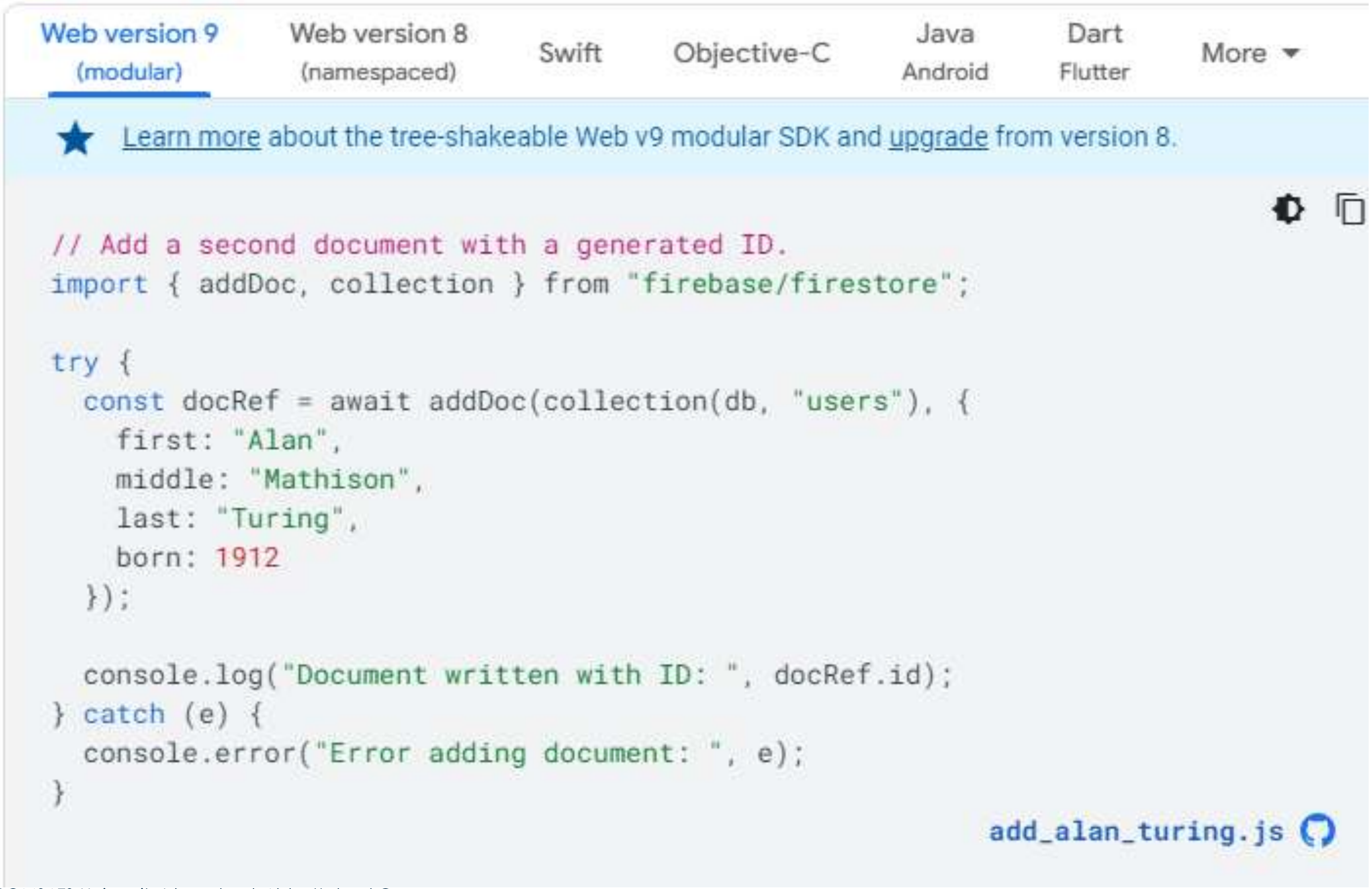


Learn more about the tree-shakeable Web v9 modular SDK and upgrade from version 8.

```
import { collection, addDoc } from "firebase/firestore";

try {
  const docRef = await addDoc(collection(db, "users"), {
    first: "Ada",
    last: "Lovelace",
    born: 1815
  });
  console.log("Document written with ID: ", docRef.id);
} catch (e) {
  console.error("Error adding document: ", e);
}
```

add_ada_lovelace.js

- Now add another document to the users collection. Notice that this document includes a key-value pair (middle name) that does not appear in the first document. Documents in a collection can contain different sets of information.



Learn more about the tree-shakeable Web v9 modular SDK and upgrade from version 8.

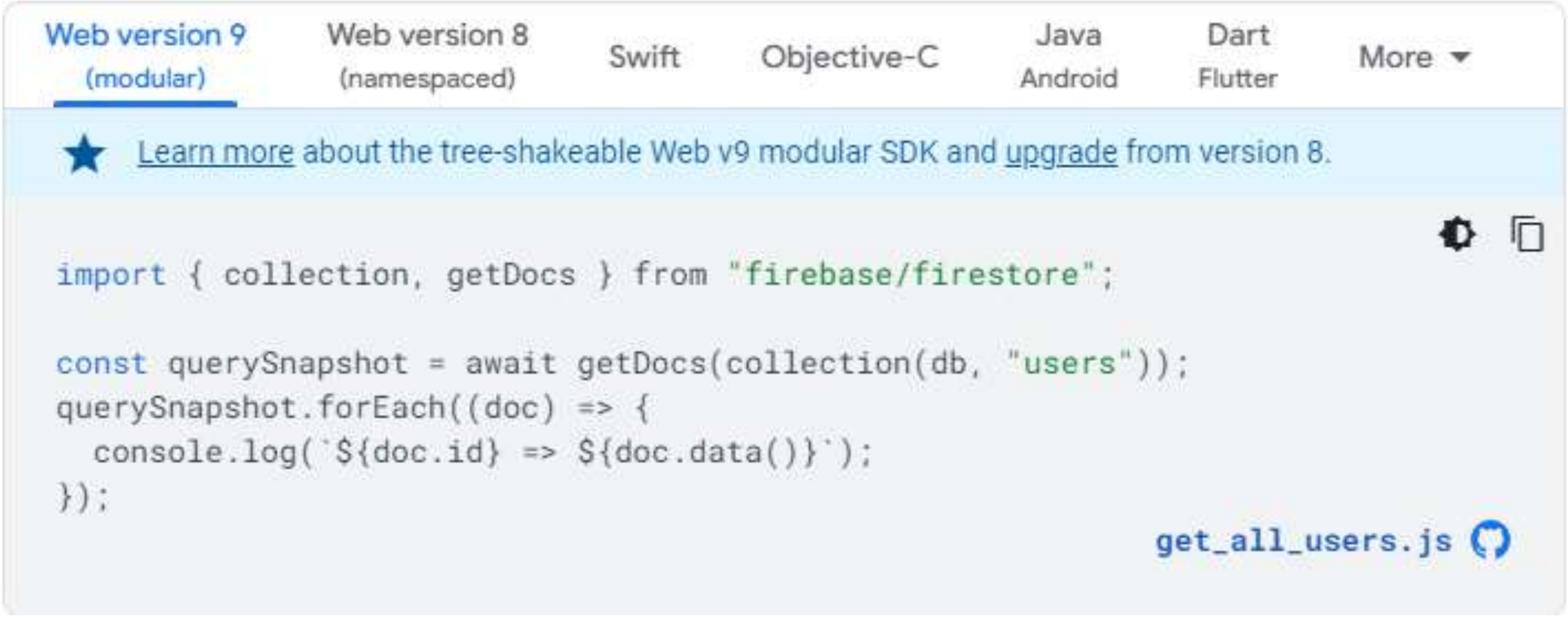
```
// Add a second document with a generated ID.
import { addDoc, collection } from "firebase/firestore";

try {
  const docRef = await addDoc(collection(db, "users"), {
    first: "Alan",
    middle: "Mathison",
    last: "Turing",
    born: 1912
  });

  console.log("Document written with ID: ", docRef.id);
} catch (e) {
  console.error("Error adding document: ", e);
}
```

add_alan_turing.js

Read data

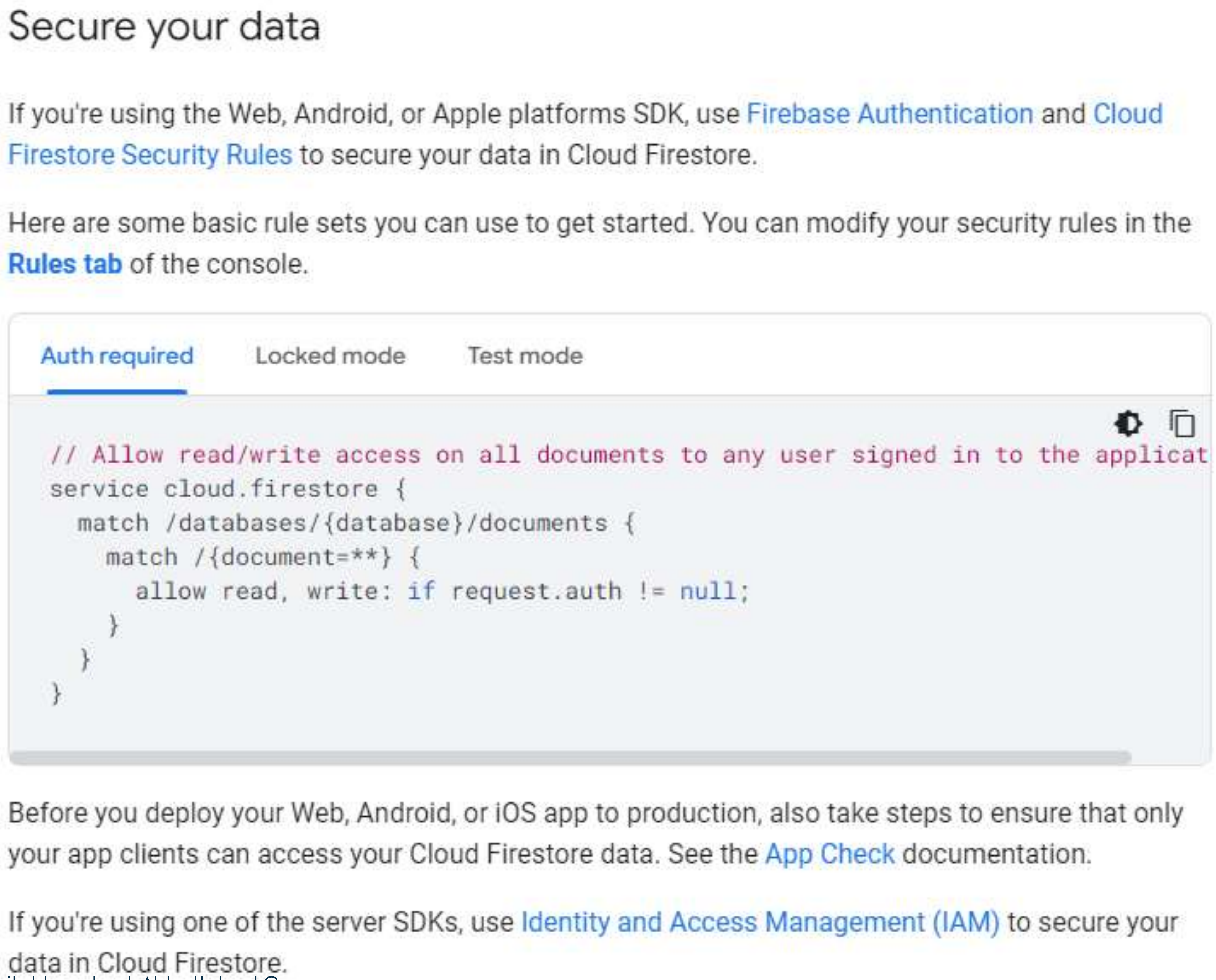
To quickly verify that you've added data to Cloud Firestore, use the data viewer in the Firebase console.

You can also use the "get" method to retrieve the entire collection.



Web version 9 (modular)
Web version 8 (namespaced)
Swift
Objective-C
Java Android
Dart Flutter
More
Learn more about the tree-shakeable Web v9 modular SDK and upgrade from version 8.
get_all_users.js

```
import { collection, getDocs } from "firebase/firestore";

const querySnapshot = await getDocs(collection(db, "users"));
querySnapshot.forEach((doc) => {
  console.log(`${doc.id} => ${doc.data()}`);
});
```



Secure your data

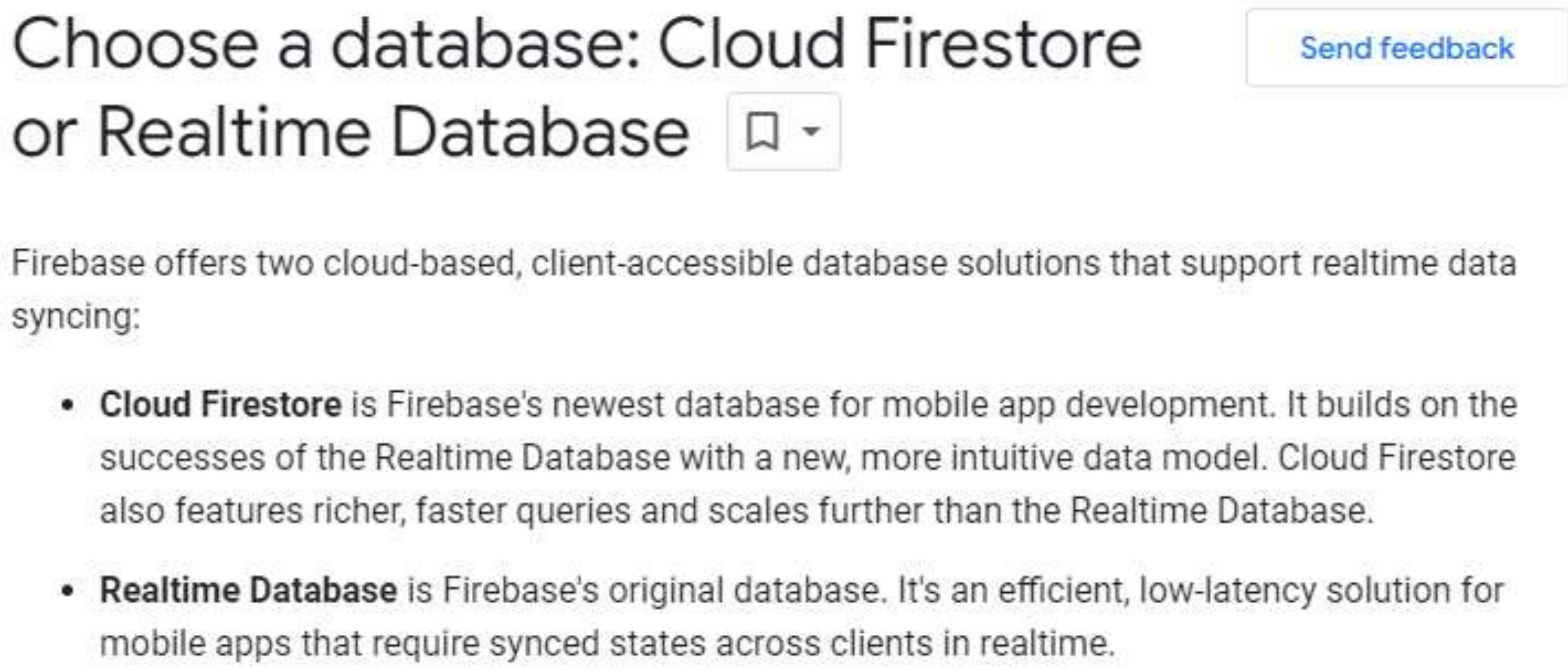
If you're using the Web, Android, or Apple platforms SDK, use Firebase Authentication and Cloud Firestore Security Rules to secure your data in Cloud Firestore.

Here are some basic rule sets you can use to get started. You can modify your security rules in the **Rules tab** of the console.

```
// Allow read/write access on all documents to any user signed in to the applicat
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

Before you deploy your Web, Android, or iOS app to production, also take steps to ensure that only your app clients can access your Cloud Firestore data. See the App Check documentation.

If you're using one of the server SDKs, use Identity and Access Management (IAM) to secure your data in Cloud Firestore.



Choose a database: Cloud Firestore or Realtime Database

Firebase offers two cloud-based, client-accessible database solutions that support realtime data syncing:

- **Cloud Firestore** is Firebase's newest database for mobile app development. It builds on the successes of the Realtime Database with a new, more intuitive data model. Cloud Firestore also features richer, faster queries and scales further than the Realtime Database.
- **Realtime Database** is Firebase's original database. It's an efficient, low-latency solution for mobile apps that require synced states across clients in realtime.

Cloud Firestore Data model

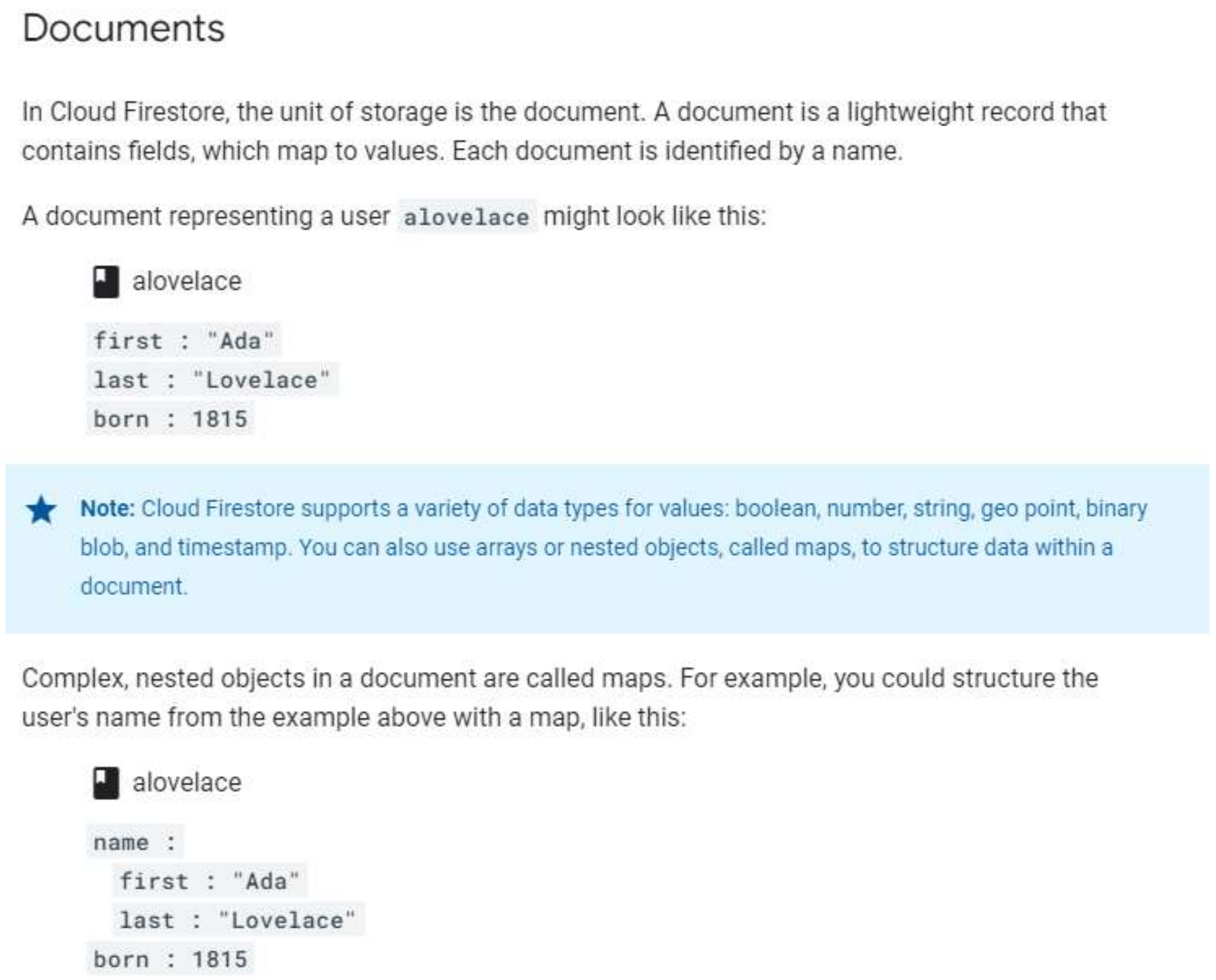
Send feedback

Cloud Firestore is a NoSQL, document-oriented database. Unlike a SQL database, there are no tables or rows. Instead, you store data in *documents*, which are organized into *collections*.

Each *document* contains a set of key-value pairs. Cloud Firestore is optimized for storing large collections of small documents.

All documents must be stored in collections.
Documents can contain *subcollections* and nested objects, both of which can include primitive fields like strings or complex objects like lists.

Collections and documents are created implicitly in Cloud Firestore. Simply assign data to a document within a collection. If either the collection or document does not exist, Cloud Firestore creates it.



Documents

In Cloud Firestore, the unit of storage is the document. A document is a lightweight record that contains fields, which map to values. Each document is identified by a name.

A document representing a user `alovelace` might look like this:

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

★ **Note:** Cloud Firestore supports a variety of data types for values: boolean, number, string, geo point, binary blob, and timestamp. You can also use arrays or nested objects, called maps, to structure data within a document.

Complex, nested objects in a document are called maps. For example, you could structure the user's name from the example above with a map, like this:

alovelace

```
name :
  first : "Ada"
  last : "Lovelace"
born : 1815
```

You may notice that documents look a lot like JSON. In fact, they basically are. There are some differences (for example, documents support extra data types and are limited in size to 1 MB), but in general, you can treat documents as lightweight JSON records.

Collections

Documents live in collections, which are simply containers for documents. For example, you could have a `users` collection to contain your various users, each represented by a document:

users

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

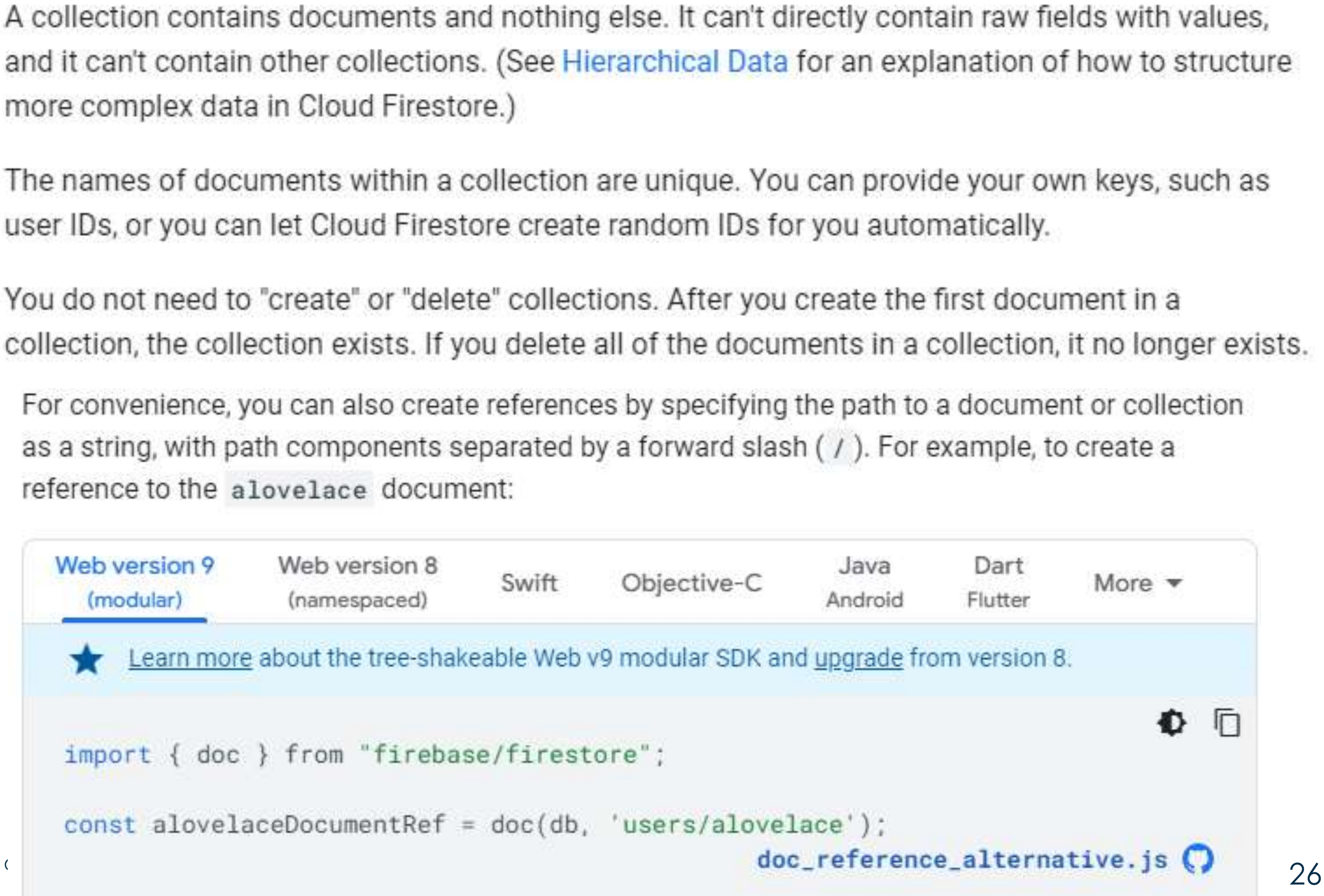
aturing

```
first : "Alan"
last : "Turing"
born : 1912
```

data
document
collection

Cloud Firestore is schemaless, so you have complete freedom over what fields you put in each document and what data types you store in those fields. Documents within the same collection can all contain different fields or store different types of data in those fields. However, it's a good idea to use the same fields and data types across multiple documents, so that you can query the documents more easily.

A collection contains documents and nothing else. It can't directly contain raw fields with values, and it can't contain other collections. (See Hierarchical Data for an explanation of how to structure more complex data in Cloud Firestore.)



A collection contains documents and nothing else. It can't directly contain raw fields with values, and it can't contain other collections. (See Hierarchical Data for an explanation of how to structure more complex data in Cloud Firestore.)

The names of documents within a collection are unique. You can provide your own keys, such as user IDs, or you can let Cloud Firestore create random IDs for you automatically.

You do not need to "create" or "delete" collections. After you create the first document in a collection, the collection exists. If you delete all of the documents in a collection, it no longer exists.

For convenience, you can also create references by specifying the path to a document or collection as a string, with path components separated by a forward slash (`/`). For example, to create a reference to the `alovelace` document:

Web version 9 (modular) | Web version 8 (namespaced) | Swift | Objective-C | Java Android | Dart Flutter | More

★ Learn more about the tree-shakeable Web v9 modular SDK and upgrade from version 8.

```
import { doc } from "firebase/firestore";

const alovelaceDocumentRef = doc(db, 'users/alovelace');
```

doc_reference_alternative.js

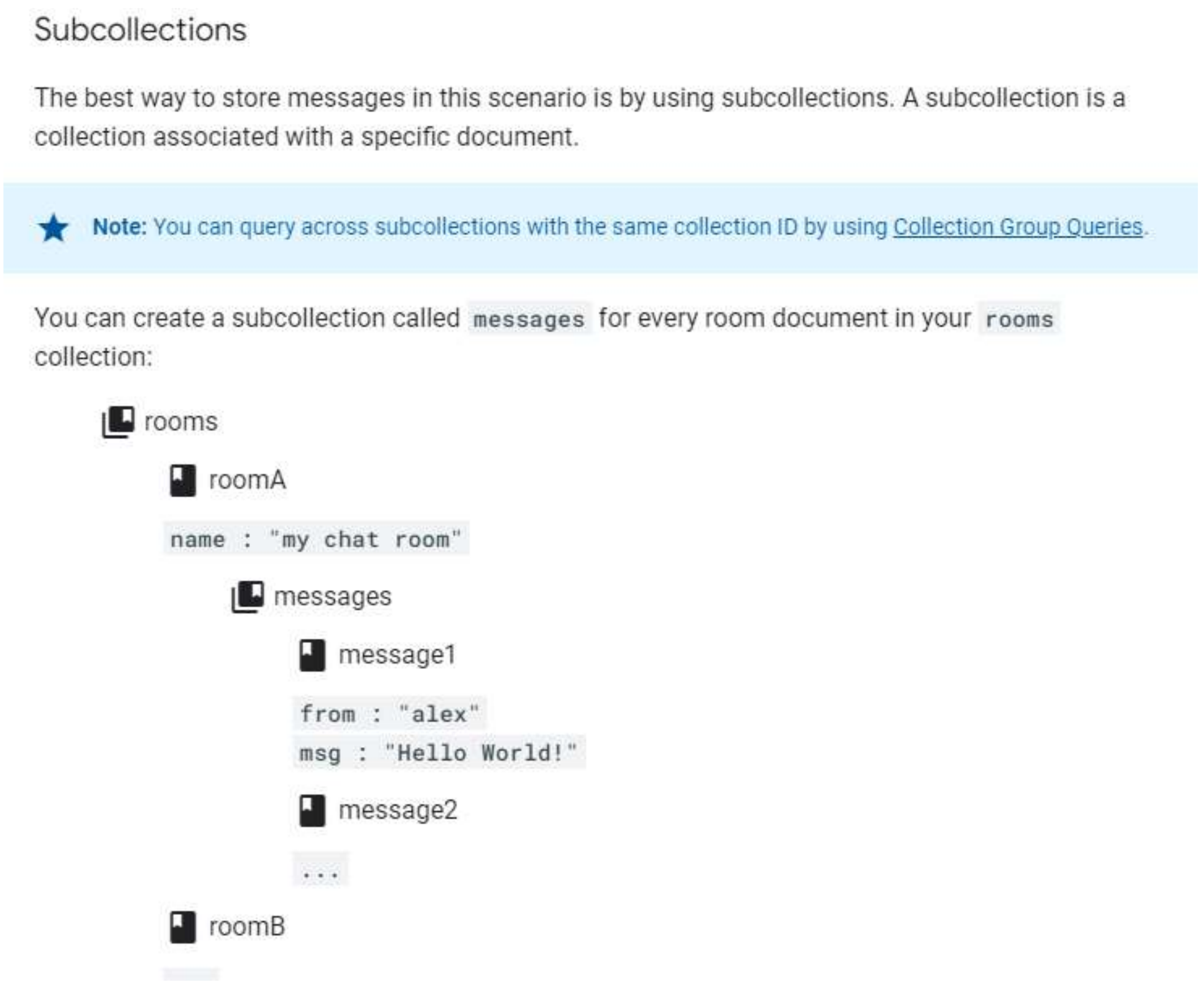
Hierarchical Data

To understand how hierarchical data structures work in Cloud Firestore, consider an example chat app with messages and chat rooms.

You can create a collection called `rooms` to store different chat rooms:

- rooms
    - roomA
      `name : "my chat room"`
    - roomB
      `...`

Now that you have chat rooms, decide how to store your messages. You might not want to store them in the chat room's document. Documents in Cloud Firestore should be lightweight, and a chat room could contain a large number of messages. However, you can create additional collections within your chat room's document, as subcollections.



Subcollections

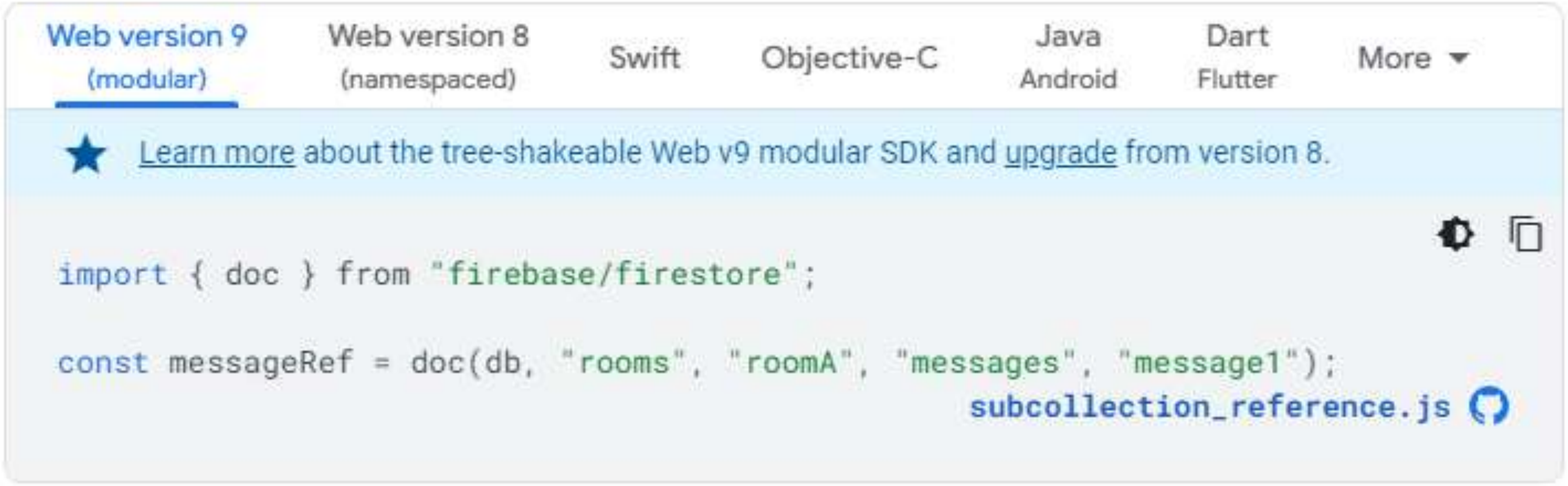
The best way to store messages in this scenario is by using subcollections. A subcollection is a collection associated with a specific document.

★ **Note:** You can query across subcollections with the same collection ID by using Collection Group Queries.

You can create a subcollection called `messages` for every room document in your `rooms` collection:

- rooms
  - roomA
    `name : "my chat room"`
    - messages
      - message1
        `from : "alex"`
        `msg : "Hello World!"`
      - message2
        `...`
  - roomB

In this example, you would create a reference to a message in the subcollection with the following code:



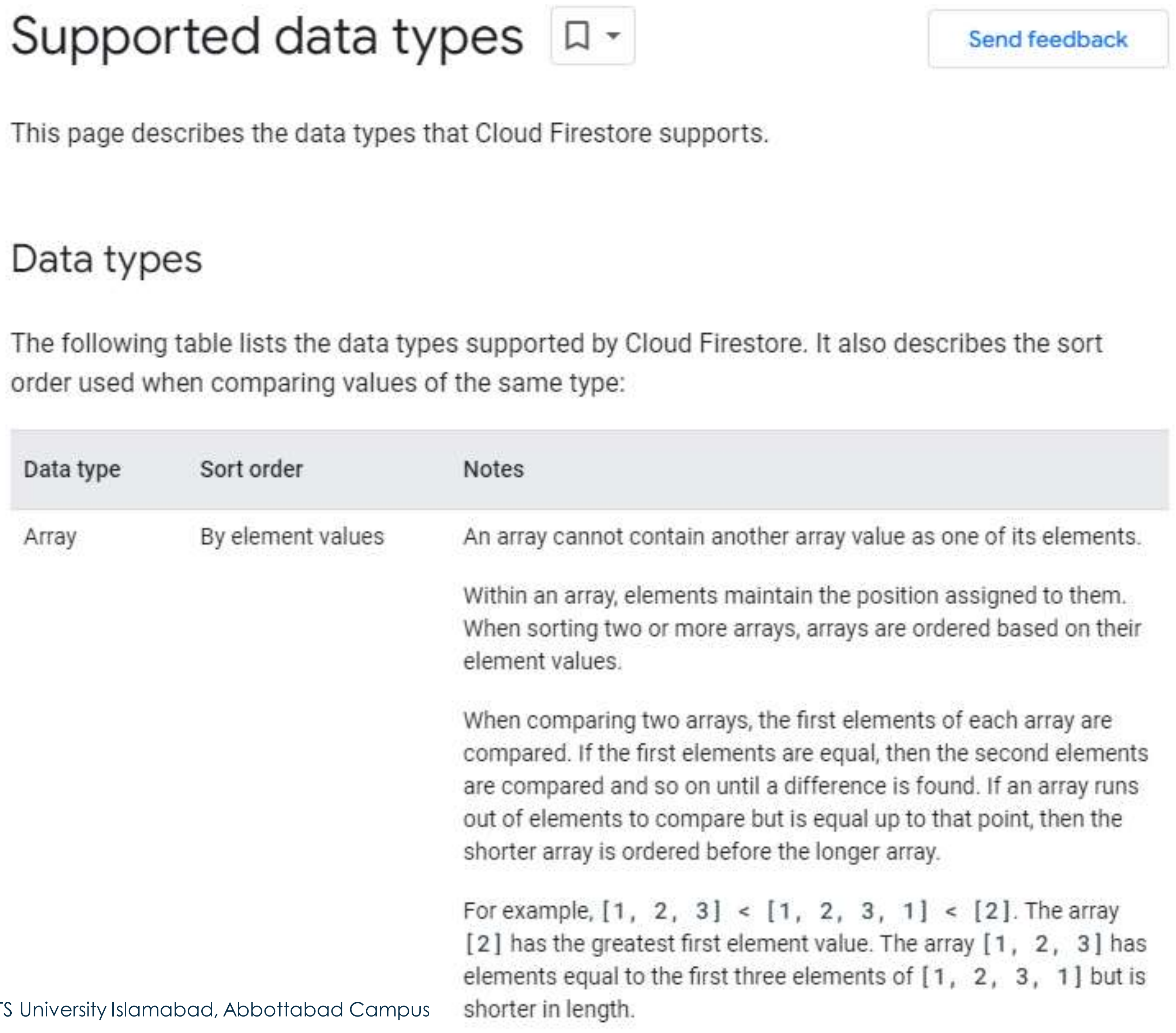
★ Learn more about the tree-shakeable Web v9 modular SDK and upgrade from version 8.

```
import { doc } from "firebase/firestore";

const messageRef = doc(db, "rooms", "roomA", "messages", "message1");
```
subcollection_reference.js

Notice the alternating pattern of collections and documents. Your collections and documents must always follow this pattern. You cannot reference a collection in a collection or a document in a document.

Subcollections allow you to structure data hierarchically, making data easier to access. To get all messages in `roomA`, you can create a collection reference to the subcollection `messages` and interact with it like you would any other collection reference.

Documents in subcollections can contain subcollections as well, allowing you to further nest data. You can nest data up to 100 levels deep.



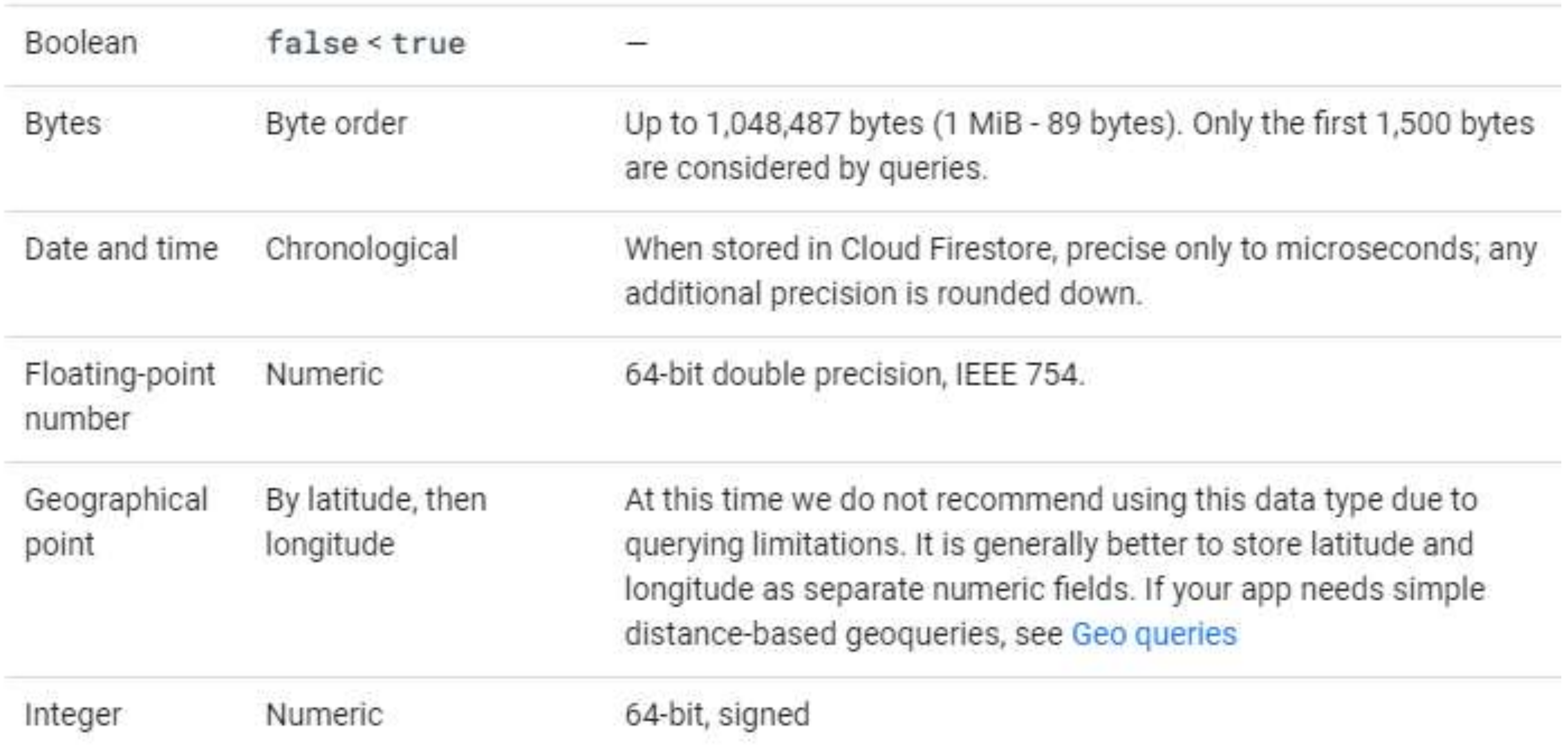
Supported data types

This page describes the data types that Cloud Firestore supports.

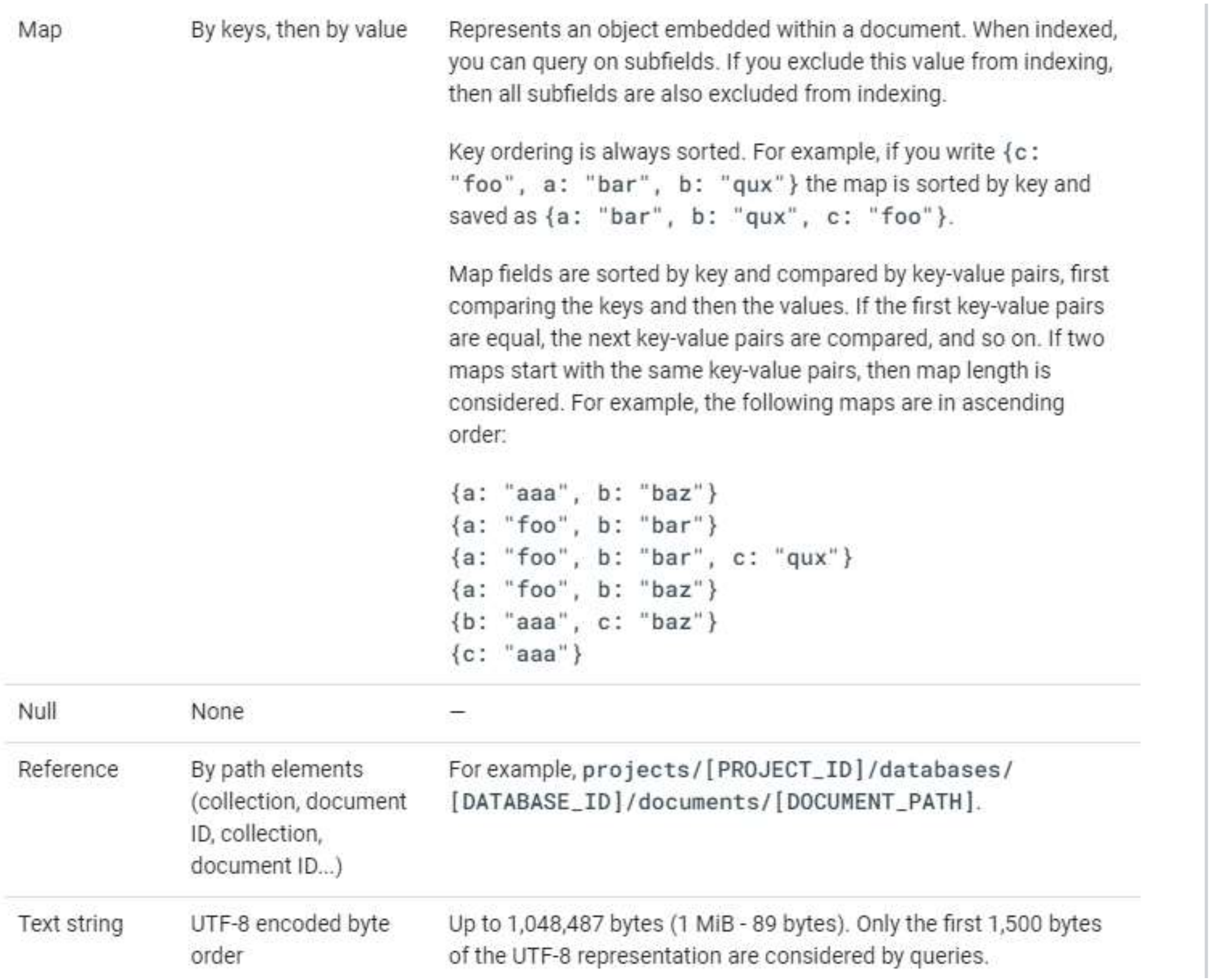
Data types

The following table lists the data types supported by Cloud Firestore. It also describes the sort order used when comparing values of the same type:

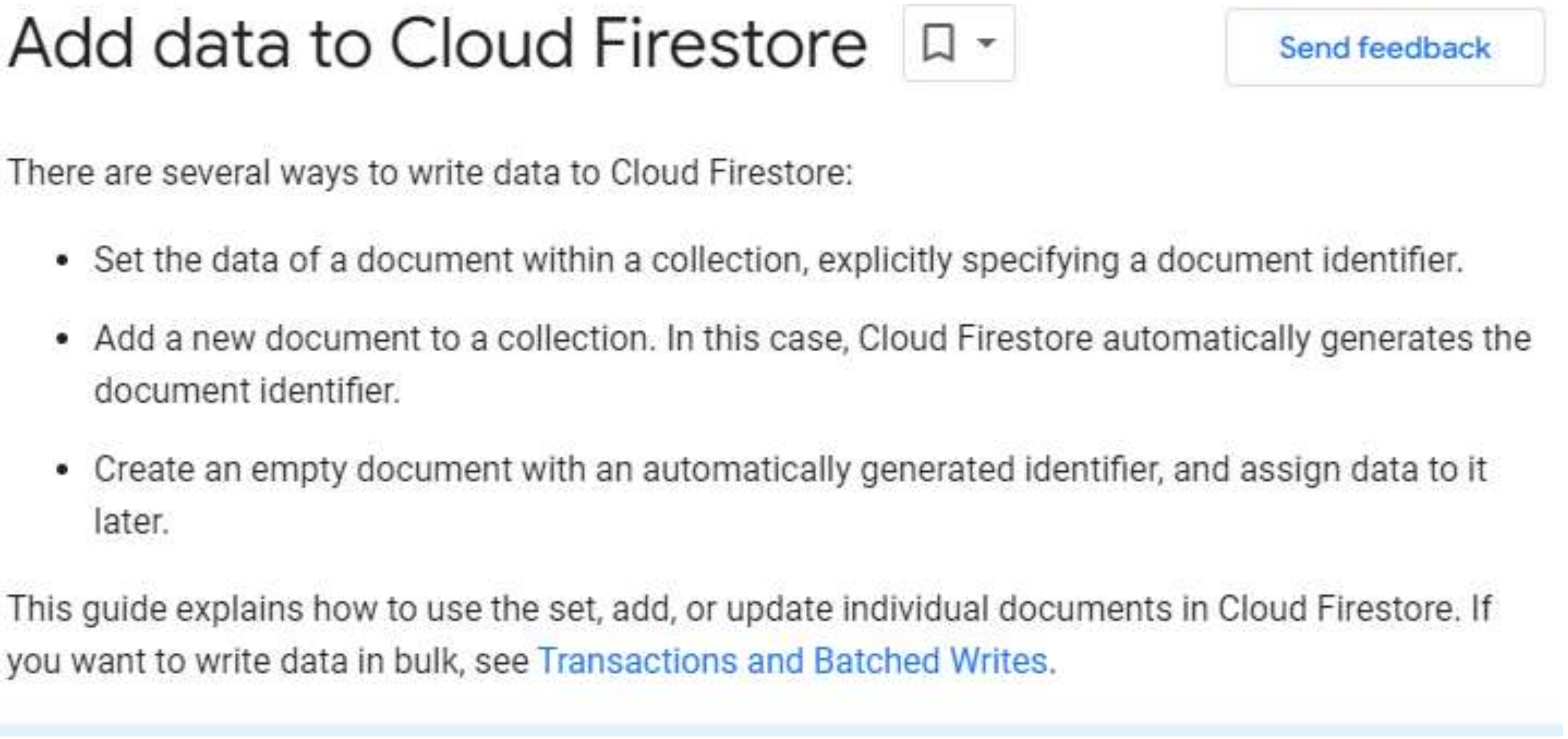
Data type	Sort order	Notes
Array	By element values	An array cannot contain another array value as one of its elements. Within an array, elements maintain the position assigned to them. When sorting two or more arrays, arrays are ordered based on their element values. When comparing two arrays, the first elements of each array are compared. If the first elements are equal, then the second elements are compared and so on until a difference is found. If an array runs out of elements to compare but is equal up to that point, then the shorter array is ordered before the longer array. For example, `[1, 2, 3] < [1, 2, 3, 1] < [2]`. The array `[2]` has the greatest first element value. The array `[1, 2, 3]` has elements equal to the first three elements of `[1, 2, 3, 1]` but is shorter in length.



Boolean	`false` < `true`	—
Bytes	Byte order	Up to 1,048,487 bytes (1 MiB - 89 bytes). Only the first 1,500 bytes are considered by queries.
Date and time	Chronological	When stored in Cloud Firestore, precise only to microseconds; any additional precision is rounded down.
Floating-point number	Numeric	64-bit double precision, IEEE 754.
Geographical point	By latitude, then longitude	At this time we do not recommend using this data type due to querying limitations. It is generally better to store latitude and longitude as separate numeric fields. If your app needs simple distance-based geoqueries, see Geo queries
Integer	Numeric	64-bit, signed



Map	By keys, then by value	Represents an object embedded within a document. When indexed, you can query on subfields. If you exclude this value from indexing, then all subfields are also excluded from indexing. Key ordering is always sorted. For example, if you write `{c: "foo", a: "bar", b: "qux"}` the map is sorted by key and saved as `{a: "bar", b: "qux", c: "foo"}`. Map fields are sorted by key and compared by key-value pairs, first comparing the keys and then the values. If the first key-value pairs are equal, the next key-value pairs are compared, and so on. If two maps start with the same key-value pairs, then map length is considered. For example, the following maps are in ascending order: `{a: "aaa", b: "baz"}` `{a: "foo", b: "bar"}` `{a: "foo", b: "bar", c: "qux"}` `{a: "foo", b: "baz"}` `{b: "aaa", c: "baz"}` `{c: "aaa"}`
Null	None	—
Reference	By path elements (collection, document ID, collection, document ID...)	For example, `projects/[PROJECT_ID]/databases/[DATABASE_ID]/documents/[DOCUMENT_PATH]`.
Text string	UTF-8 encoded byte order	Up to 1,048,487 bytes (1 MiB - 89 bytes). Only the first 1,500 bytes of the UTF-8 representation are considered by queries.



Add data to Cloud Firestore

There are several ways to write data to Cloud Firestore:

- Set the data of a document within a collection, explicitly specifying a document identifier.
- Add a new document to a collection. In this case, Cloud Firestore automatically generates the document identifier.
- Create an empty document with an automatically generated identifier, and assign data to it later.

This guide explains how to use the set, add, or update individual documents in Cloud Firestore. If you want to write data in bulk, see Transactions and Batched Writes.