CIS 371 Web Application Programming

Cloud DataBase

Firebase Cloud Firestore

GRAND VALLEY STATE UNIVERSITY®

Lecturer: Dr. Yong Zhuang

Based on the original version by Professor Hans Dulimarta.

Why Cloud Data Stores?

- Highly scalable
- Usually built using No SQL technology
- Accessible to both web and mobile clients

No SQL = No DB Schema

SQL vs. noSQL

- Relational model
- Schema: relationship between tables and fields
- Popular examples
  - Oracle
  - DB2
  - MySQL
  - PostGreSQL

- Non-relational
- Schemaless Datastore
- Cloud Computing and Cloud Storage
- Rapid Development
- Popular examples
  - MongoDB
  - CouchDB
  - BigTable
  - Firebase Realtime DB
  - Firebase Cloud Firestore

Schema or Schemaless?

First	Last	G#	Major
Alice	Smith	12345678	Statistics
Brad	Jordan	23456789	History

Must redefine the SCHEMA to add a new column.

First	Last	G#	Major	Color	SocMedia	?
Alice	Smith	12345678	Statistics	Green		
Brad	Jordan	23456789	History		IG, FB	
Gary	deGroot	72551834	Biology		TW	
Ann	Hunt	78921631	Physics	Blue		
Fay	Ross	72631235	English		LinkedIn	

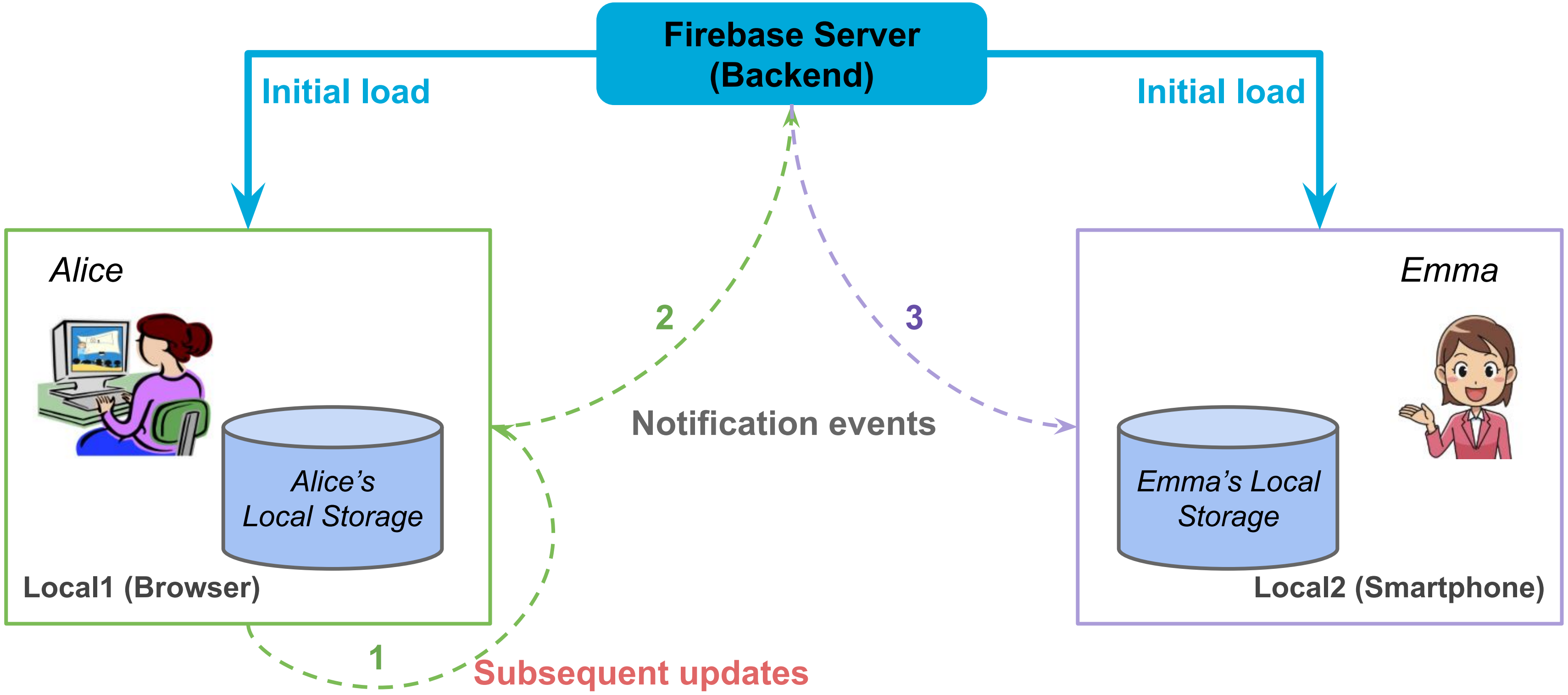
Firebase

- A collection of many products
- Cloud Firestore (beta since 2017, GA since 2019)
- Authentication
- Cloud Storage
- Realtime DB (beta since 2012, GA since 2014?)
- Cloud Messaging
- ML Kit
- Cloud Functions

Firebase

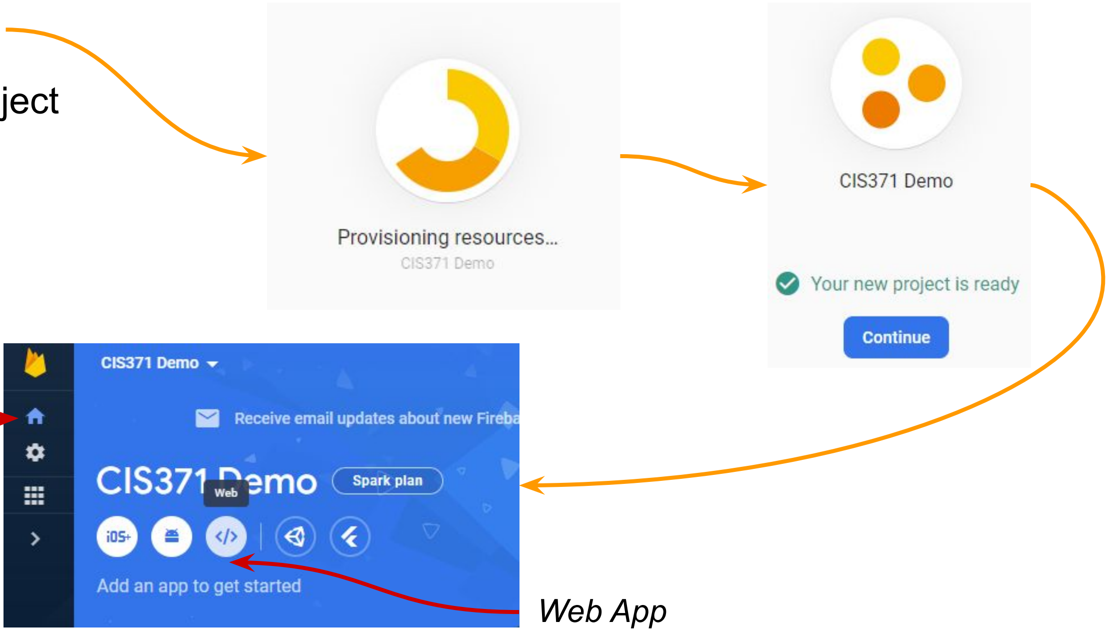
	Realtime DB	Cloud Firestore
Auto-generated Key	Time-based	Not time-based
Write operations	Max 1000 writes/second	Max 10,000 writes/second
Offline support	iOS and Android clients	iOS, Android, and Web clients
Concurrent Connections	Max 200,000	Max 1,000,000
Data Model	Giant JSON tree	Hierarchy of Collections (“Tables”) and Documents (“Records”)
Queries	Deep (**slower performance**), fetching a node will return the entire subtree of the node	Shallow (**better performance**), it is possible to fetch a document without its “children”
	Queries can use sorting or filtering (but not both)	Queries can use sorting and filtering

Local Storage, Local Events, & Global Events



Creating a new WebApp

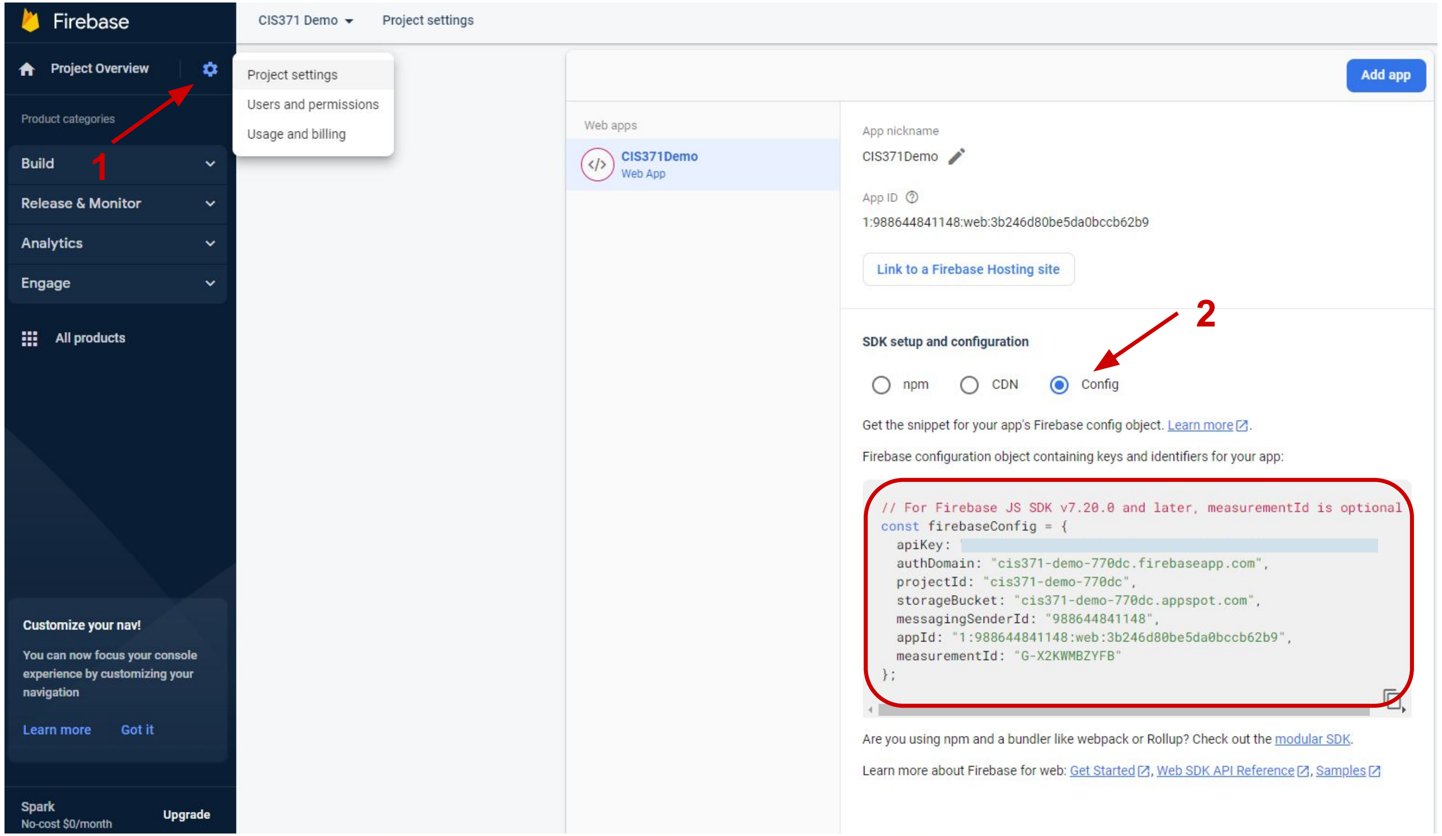
1. Use a personal Google account to login to Firebase Console
   a. GVSU Google Mail account may not work
2. Create a new project
3. Add an app to the project


Provisioning resources...
CIS371 Demo
CIS371 Demo
Your new project is ready
Continue
CIS371 Demo
Receive email updates about new Fireba
CIS371 Demo
Spark plan
Web
iOS+
Add an app to get started

Project Overview

Web App

Creating a new WebApp



App nickname
CIS371Demo

App ID
1:988644841148:web:3b246d80be5da0bccb62b9

Link to a Firebase Hosting site

SDK setup and configuration

npm CDN [✓ Config]

Get the snippet for your app's Firebase config object. Learn more.

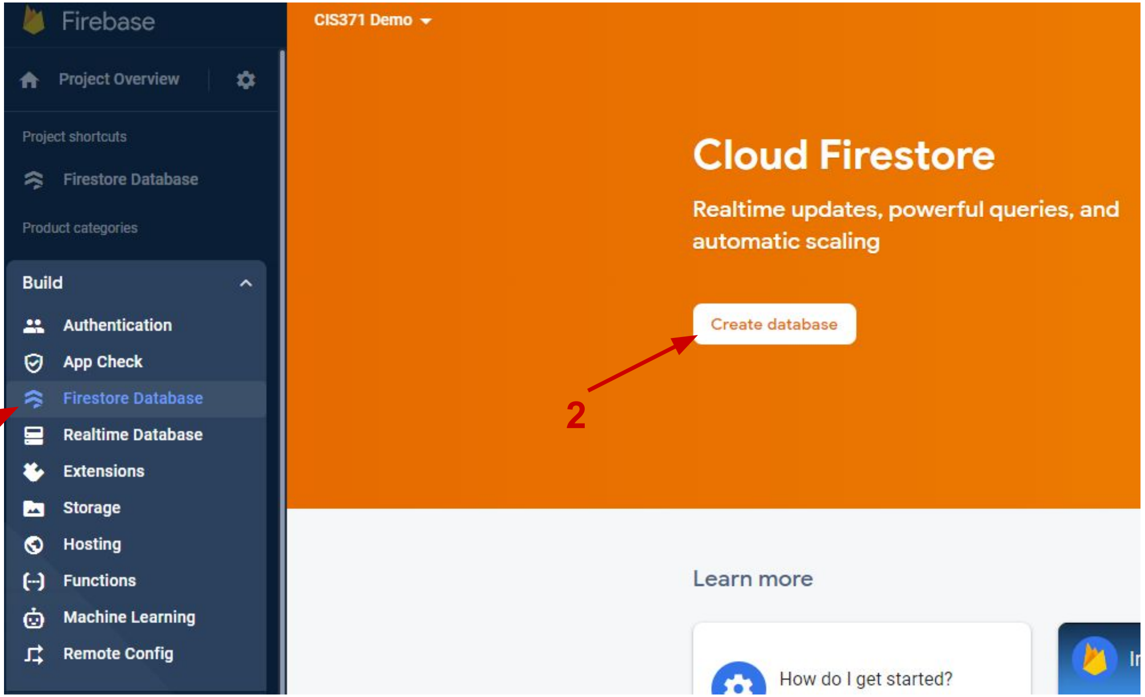
Firebase configuration object containing keys and identifiers for your app:

```
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: [redacted]
  authDomain: "cis371-demo-770dc.firebaseapp.com",
  projectId: "cis371-demo-770dc",
  storageBucket: "cis371-demo-770dc.appspot.com",
  messagingSenderId: "988644841148",
  appId: "1:988644841148:web:3b246d80be5da0bccb62b9",
  measurementId: "G-X2KWMBZYFB"
};
```

Are you using npm and a bundler like webpack or Rollup? Check out the modular SDK.

Learn more about Firebase for web: Get Started, Web SDK API Reference, Samples

Initialize Firestore



Local Project Setup
(On Your Computer)

Project Setup & Initialization

```
yarn init -y
yarn add firebase
```

OR

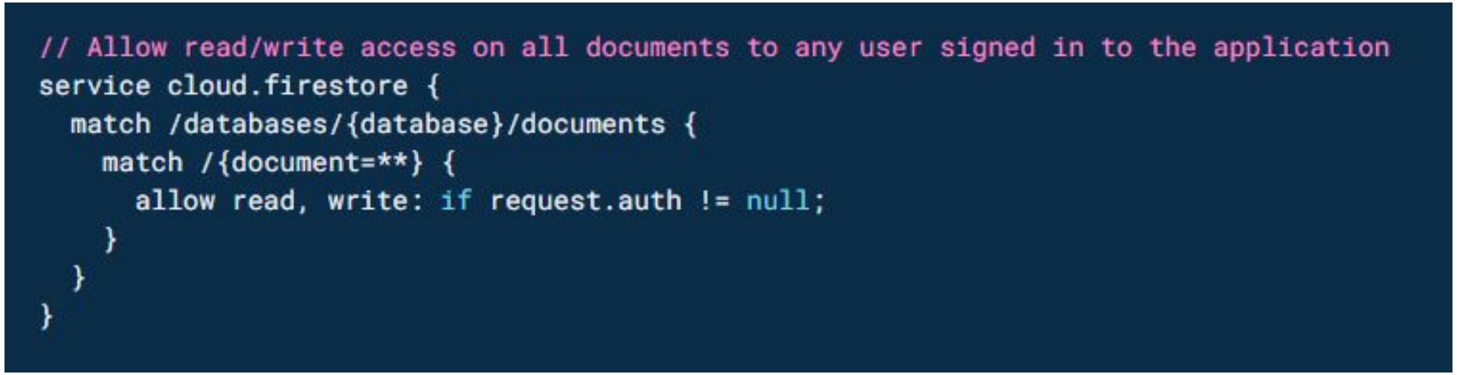
```
npm init -y
npm install firebase
```

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

const firebaseConfig = {
  apiKey: "your-api-key-goes-here",
  authDomain: "your-project-name-here.firebaseapp.com",
  databaseURL: "https://your-project-name-here.firebaseio.com",
  projectId: "your-project-name-here",
  storageBucket: "your-project-name.appspot.com",
  messagingSenderId: "xxxxxxxx"
};

// Initialize Firebase
const myapp: FirebaseApp = initializeApp(firebaseConfig);
const db: Firestore = getFirestore(myapp);
```

// COPY this from your Firebase Console

Database Dashboard

- Browse and Modify Data
- Security Rules (default settings: user authentication required)



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

[Cloud Firestore Security Rules]

Data Model: Hierarchy of Collections-Documents

- Hierarchical structure
  - The “root” holds one or more collections
  - A collection consists of one or more documents
  - A document is one or more key-value pairs
  - A value in a document may refer to a subcollection (1-to-many relationships)
- Data Types in a document
  - string, number, boolean, array, timestamp, map (kv-pairs), geolocation
  - Reference to a subcollection

SQL	Cloud Firestore
Tables	Collections
Rows	Documents
Primary Key	Document ID
Fields	key-value pairs

Data Model: Hierarchy of Collections-Documents

State (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

States (Collection of 3 Documents)

AK

Name: Alaska
Capital: Juneau

AL

Name: Alabama
Capital: Montgomery

FL

Name: Florida
Capital: Tallahassee

NatlPark (SQL table)

Code (PK)	Name	Location
U6123	Arches	Utah
C1632	Black Canyon	Colorado

Parks (Collection of 2 Documents)

U6123

Name: Arches
Location: Utah

C1632

Name: Black Canyon
Location: Colorado

SQL Table ⇒ Firestore Collection
SQL Primary Key ⇒ Firestore Document ID/"name"
SQL Table Row ⇒ Firestore Document

All Firestore Collection/Doc Manipulation Functions return a Promise

Firestore Functions (version 9.x)

- Functions for creating references
  - collection(refToFirestore, “path/to/collection”)
  - doc(refToFirestoreOrCollection, “path/to/your/document”)
  - query(refToCollection, ____)
- **Retrieval functions**
  - getDoc(refToDoc)
  - getDocs(refToCollection)
- **Manipulation functions**
  - addDoc(refToColl, { new_content_object })
  - setDoc(refToDoc, { new_content_object })
  - updateDoc(refToDoc, { new_content_object })
  - deleteDoc(refToDoc)
- Update listener `onSnapShot()` **(specific to Firebase)**

C
R
U
D

CRUD Operations (Summary)

	Collection	Document

Create	Implied when a doc is created	`// Option #1`
		`const collPar = collection(db, “cName”);`
		`addDoc(collPar, { /* new content here */ });`
		`// Option #2`
		`const myDoc = doc(db, “cName”, “docName”);`
		`setDoc(myDoc, { /* new content here */ })`

Read	`const myC =`
	`collection(db,“cName”);`
	`getDocs(myC).then(___);`
		`const myDoc = doc(____, ____, ___);`
		`getDoc(myDoc).then(___);`

Update	N/A	`const myDoc = doc(____, ____, ___);`
		`updateDoc(myDoc, {/* content */}).then(___);`

Delete	N/A	`const myDoc = doc(____, ____, ___);`
		`deleteDoc(myDoc).then(___);`

SQL vs Firebase Firestore

SQL	Firestore 8.x	Firestore 9.x

	`const myColl = db.collection(“xyz”)`
	`const myDoc = db.doc(“xyz/def”)`
	`const myDoc = db.collection(“xyz”).doc(“def”)`
		`const myColl = collection(db, “xyz”)`
		`const myDoc = doc(db, “xyz”, “def”)`
		`const myDoc = doc(db, “xyz/def”)`

`SELECT * FROM myTable`	`myColl.getDocs().then(___)`	`getDocs(myCollection).then(___)`

`INSERT INTO myTable`	`const myData = { /* TS object */ }`
	`myColl.addDoc(myData).then(___)`
	`myDoc.setDoc(myData).then(___)`
		`const myData = { /* TS object */ }`
		`addDoc(myColl, myData).then(___)`
		`setDoc(myDoc, myData).then(___)`

`UPDATE myTable WHERE`	`myDoc.updateDoc(newData).then(___)`	`updateDoc(myDoc, newData).then(___)`
`DELETE from WHERE`	`myDoc.deleteDoc().then(___)`	`deleteDoc(myDoc).then(___)`

CRUD Operations: Create Doc (own Doc ID)

SQL

```
// Use "AK" as the primary key for the tuple
INSERT INTO states (abbrev, name, capital) VALUES("AK", "Alaska", "Juneau")
```

states (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

```
import { DocumentReference, setDoc, doc } from "firebase/firestore";
// Option #1: Use file name syntax for doc path
// Primary key "AK" becomes doc id
const doc1: DocumentReference = doc(db, "states", "AK");
setDoc(doc1, { name: "Alaska", capital: "Juneau" })
  .then(() => {
    console.log("New doc added");
  })
  .catch((err: any) => {
    /* your code here */
  });
```

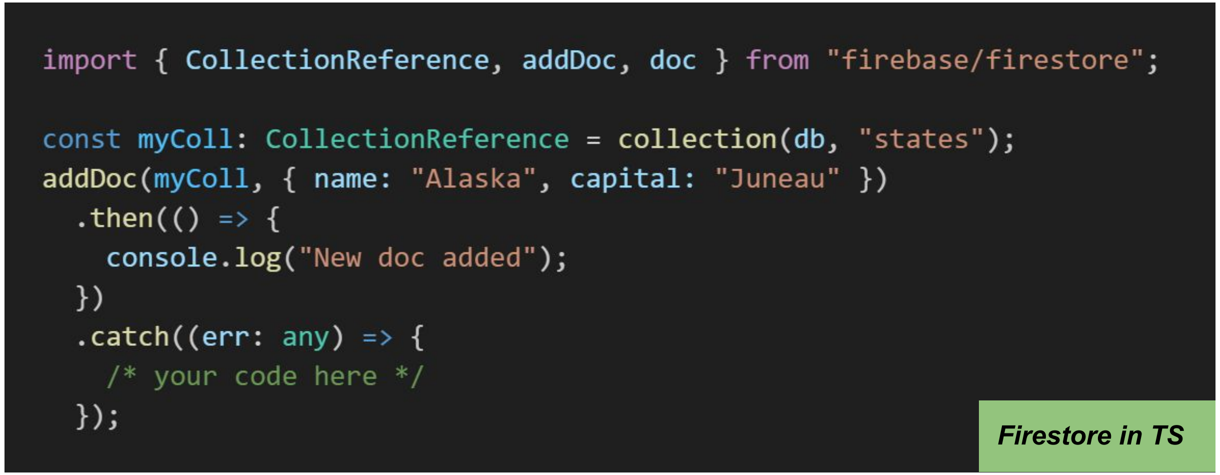
Firestore in TS

CRUD Operations: Create Doc (automatic Doc ID)

INSERT INTO states (name, capital) VALUES(“Alaska”, “Juneau”) **SQL**

states (SQL table)

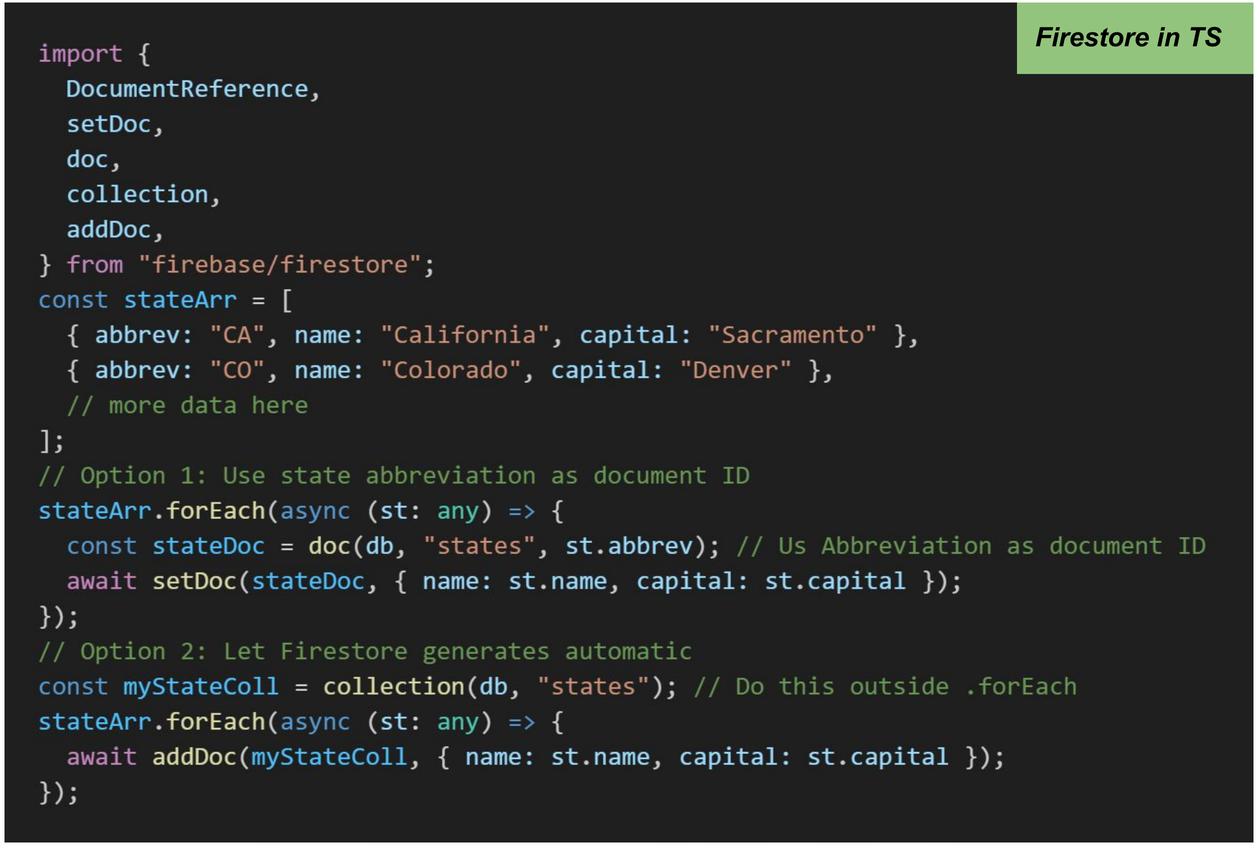
Name	Capital
Alaska	Juneau
Alabama	Montgomery
Florida	Tallahassee



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

const myColl: CollectionReference = collection(db, "states");
addDoc(myColl, { name: "Alaska", capital: "Juneau" })
  .then(() => {
    console.log("New doc added");
  })
  .catch((err: any) => {
    /* your code here */
  });
```

Firestore in TS

CRUD Operations: Create Docs from Array



Firestore in TS

```ts
import {
  DocumentReference,
  setDoc,
  doc,
  collection,
  addDoc,
} from "firebase/firestore";
const stateArr = [
  { abbrev: "CA", name: "California", capital: "Sacramento" },
  { abbrev: "CO", name: "Colorado", capital: "Denver" },
  // more data here
];
// Option 1: Use state abbreviation as document ID
stateArr.forEach(async (st: any) => {
  const stateDoc = doc(db, "states", st.abbrev); // Us Abbreviation as document ID
  await setDoc(stateDoc, { name: st.name, capital: st.capital });
});
// Option 2: Let Firestore generates automatic
const myStateColl = collection(db, "states"); // Do this outside .forEach
stateArr.forEach(async (st: any) => {
  await addDoc(myStateColl, { name: st.name, capital: st.capital });
});
```

await vs. .then()

GRAND VALLEY STATE UNIVERSITY

CRUD Operations: Read All Documents

```
SELECT * FROM states
```
SQL

states (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

```
// Assume saved data has the
// following structure
type StateType = {
  abbrev: string;
  name: string;
  capital: string;
};
```

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

const myStateColl: CollectionReference = collection(db, "states");

getDocs(myStateColl).then((qs: QuerySnapshot) => {
  qs.forEach((qd: QueryDocumentSnapshot) => {
    const stateData = qd.data() as StateType;
    const docId = qd.id; // Fixed 'cost' to 'const'
    // More code here to manipulate stateData
  });
});
```

Firestore in TS

CRUD Operations: Read A Specific Document

SQL

```
// Select a tuple with a known primary key
SELECT * FROM states WHERE abbrev = “FL”
```

states (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

```
// Assume saved data has the
// following structure
type StateType = {
  abbrev: string;
  name: string;
  capital: string;
};
```

```
import {
  DocumentReference,
  doc,
  DocumentSnapshot,
  getDoc,
} from "firebase/firestore";
// FL is a document ID
const myDoc: DocumentReference = doc(db, "states/FL");
getDoc(myDoc).then((qd: DocumentSnapshot) => {
  if (qd.exists()) {
    const stateData = qd.data() as StateType;
    // More code here to manipulate stateData
  }
});
```

Firestore in TS

CRUD Operations: Fetch Document(s) Where...

SQL

// Select tuples satisfying some conditions
SELECT * FROM states WHERE name = “Florida”

states (SQL table)

Abbrev (PK)	Name	Capital
AK	Alaska	Juneau
AL	Alabama	Montgomery
FL	Florida	Tallahassee

```ts
// Assume saved data has the
// following structure
type StateType = {
  abbrev: string;
  name: string;
  capital: string;
};
```

```ts
import {
  Query,
  getDocs,
  collection,
  where,
  query,
  QuerySnapshot,
  QueryDocumentSnapshot,
} from "firebase/firestore";
const getFL: Query = query(
  collection(db, "states"),
  where("name", "==", "Florida")
);
getDocs(getFL).then((qs: QuerySnapshot) => {
  qs.forEach((qd: QueryDocumentSnapshot) => {
    const stateData = qd.data() as StateType;
    // More code here to manipulate stateData
  });
});
```

Firestore in TS

CRUD Operations: Fetch Document(s) Where...

SQL

```
// Select tuples satisfying some conditions
SELECT * FROM states WHERE population > 10_000_000
```

states (SQL table)

Name	Capital	Population
California	Sacramento	39_123_612
Michigan	Lansing	8_432_911
Florida	Tallahassee	26_222_943

```
// Assume saved data has the
// following structure
type StateType = {
  name: string;
  capital: string;
  population: number;
};
```

```
import {
  Query,
  getDocs,
  collection,
  query,
  where,
  QuerySnapshot,
  QueryDocumentSnapshot,
} from "firebase/firestore";
const aboveTenMil: Query = query(
  collection(db, "states"),
  where("population", ">", 10_000_000)
);
getDocs(aboveTenMil).then((qs: QuerySnapshot) => {
  qs.forEach((qd: QueryDocumentSnapshot) => {
    const stateData = qd.data() as StateType;
    // More code here to manipulate stateData
  });
});
```

Firestore in TS

GRAND VALLEY STATE UNIVERSITY

CRUD Operations: Fetch Document(s) Where…

SQL

```
// Select tuples satisfying some conditions
SELECT * FROM states WHERE population > 10_000_000
AND population < 15_000_000
```

states (SQL table)

Name	Capital	Population
California	Sacramento	39_123_612
Michigan	Lansing	8_432_911
Florida	Tallahassee	26_222_943

```
// Assume saved data has the
// following structure
type StateType = {
  name: string;
  capital: string;
  population: number;
};
```

```
import {
  Query,
  getDocs,
  collection,
  query,
  where,
  QuerySnapshot,
  QueryDocumentSnapshot,
} from "firebase/firestore";
const aboveTenMil: Query = query(
  collection(db, "states"),
  where("population", ">", 10_000_000),
  where("population", "<", 15_000_000)
);
getDocs(aboveTenMil).then((qs: QuerySnapshot) => {
  qs.forEach((qd: QueryDocumentSnapshot) => {
    const stateData = qd.data() as StateType;
    // More code here to manipulate stateData
  });
});
```

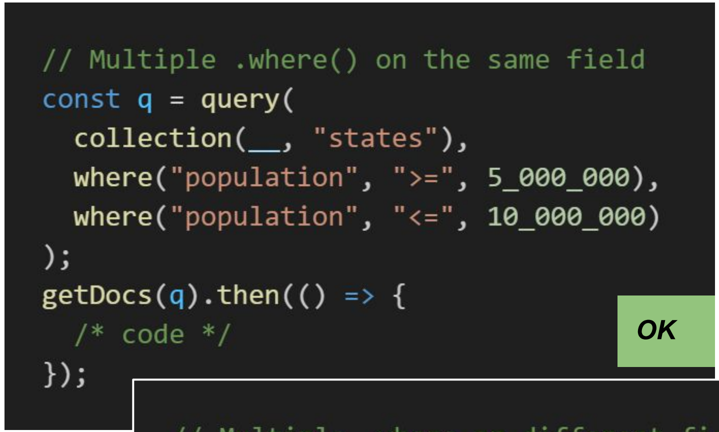
Firestore in TS

Available Query Where Operators

Operator	Example	SQL Equivalent
`<, <=, ==, >=, >`	`where("population", ">", 20_000_000)`	`WHERE population > 20000000`
`!=`	`where("name", "!=", "Andy")`	`WHERE name != "Andy"`
`in`	`where("city", "in", ["Ada", "Flint"])`	`WHERE city == "Ada" OR city == "Flint"`
`not-in`	`where("city", "not-in", ["Ada", "Flint"])`	`WHERE city != "Ada" AND city != "Flint"`

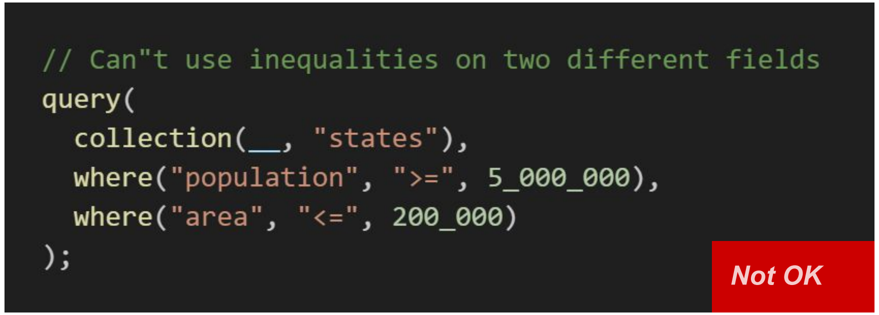
Operator	Example (courses must be an ARRAY)
`array-contains`	`// Has this student taken MTH200?` `where("courses", "array-contains", "MTH200")`
`array-contains-any`	`// Has this student taken either MTH200 or STA215? where("courses", "array-contains-any", ["MTH200", "STA215"])`

Query Limitations



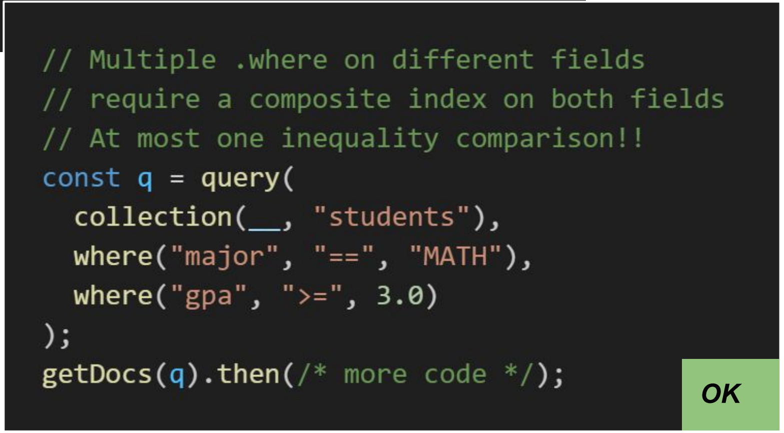
```
// Multiple .where() on the same field
const q = query(
  collection(__, "states"),
  where("population", ">=", 5_000_000),
  where("population", "<=", 10_000_000)
);
getDocs(q).then(() => {
  /* code */
});
```

OK



```
// Can"t use inequalities on two different fields
query(
  collection(__, "states"),
  where("population", ">=", 5_000_000),
  where("area", "<=", 200_000)
);
```

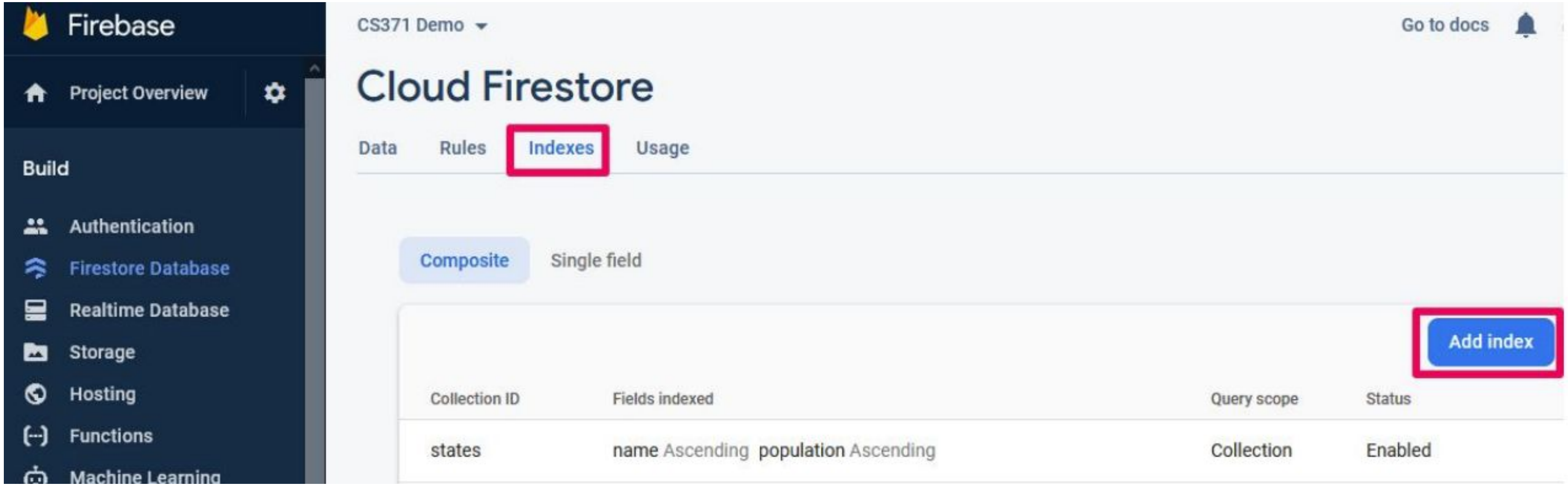
Not OK



```
// Multiple .where on different fields
// require a composite index on both fields
// At most one inequality comparison!!
const q = query(
  collection(__, "students"),
  where("major", "==", "MATH"),
  where("gpa", ">=", 3.0)
);
getDocs(q).then(/* more code */);
```

OK

Building Composite Index



Collection ID	Fields indexed	Query scope	Status
states	name Ascending population Ascending	Collection	Enabled

Order of index build does matter!!!