NoSQL & Firebase

SWE 432, Fall 2018

Web Application Development

Review: Nouns vs. Verbs

- URIs should hierarchically identify **nouns** describing **resources** that exist
- Verbs describing actions that can be taken with resources should be described with an HTTP **action**

- PUT /cities/:cityID (nouns: cities, :cityID)(verb: PUT)
- GET /cities/:cityID (nouns: cities, :cityID)(verb: GET)

- Want to offer **expressive** abstraction that can be reused for many scenarios

Review: HTTP Actions

- How do intermediaries know what they can and cannot do with a request?
- Solution: HTTP Actions
  - Describes what will be done with resource
  - GET: retrieve the current state of the resource
  - PUT: modify the state of a resource
  - DELETE: clear a resource
  - POST: initialize the state of a new resource

Review: URI Design

- URIs represent a contract about what resources your server exposes and what can be done with them
- Leave out **anything that might change**
  - Content author names, status of content, other keys that might change
  - File name extensions: response describes content type through MIME header not extension (e.g., .jpg, .mp3, .pdf)
  - Server technology: should not reference technology (e.g., .cfm, .jsp)
- Endeavor to make all changes backwards compatible
  - Add new resources and actions rather than remove old
- If you must change URI structure, support old URI structure **and** new URI structure

Today

- Design considerations in identifying resources
- REST
  - What is it?
  - Why use it?
- HW2 Discussion (Posted now, due Oct 9)

Storing state in a global variable

- **Global variables**

```
var express = require('express');
var app = express();
var port = process.env.port || 3000;

var counter = 0;
app.get('/', function (req, res) {
    res.send('Hello World has been said ' + counter + ' times!');
    counter++;
});

app.listen(port, function () {
    console.log('Example app listening on port' + port);
});
```

- Pros/cons?
  - Keep data between requests
  - **Goes away** when your server stops
    - Should use for transient state or as cache

What forms of data might you have

- Key / value pairs
- JSON objects
- Tabular arrays of data
- Files

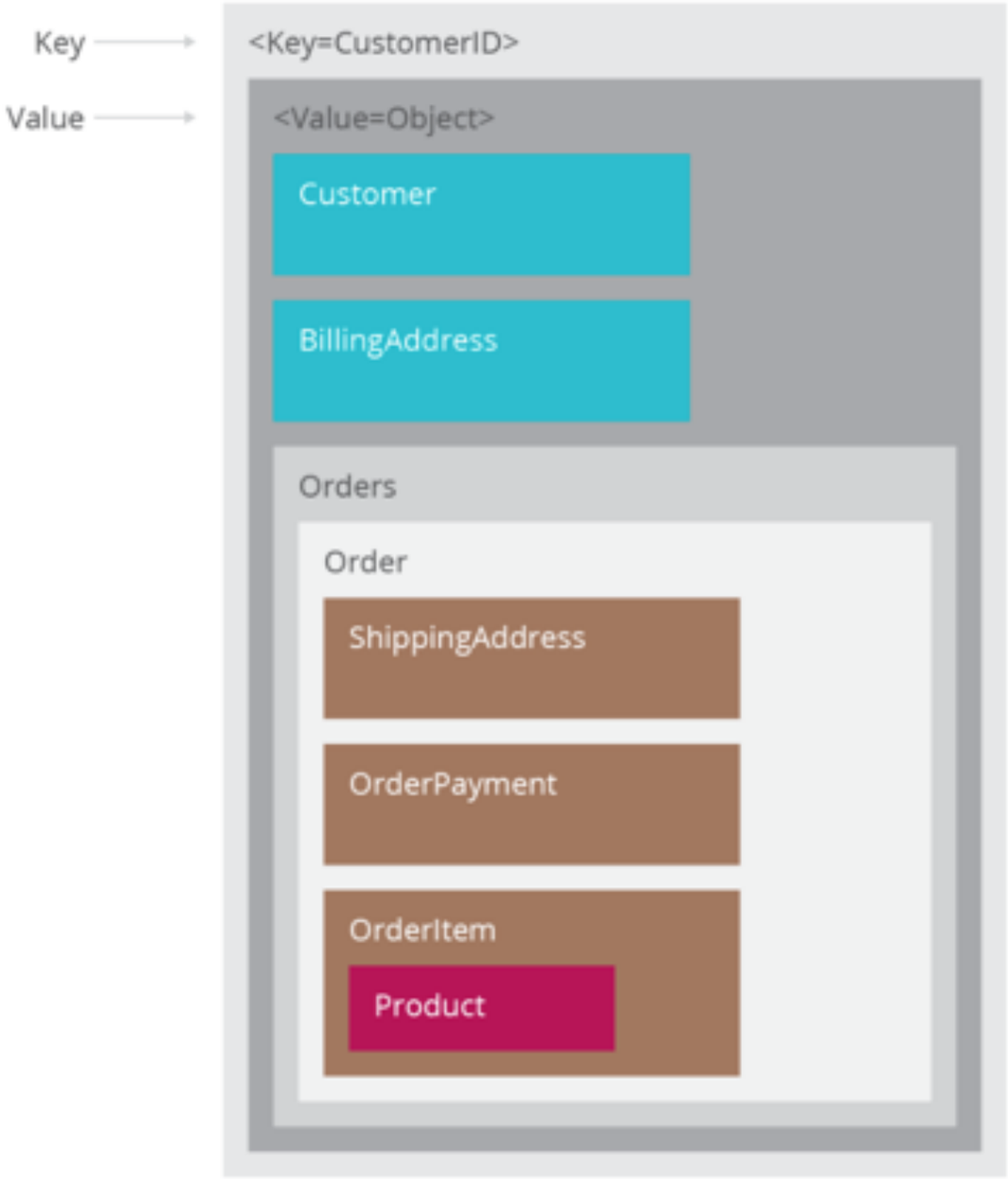
Options for backend persistence

- Where it is stored
  - On your server or another server you own
    - SQL databases, NoSQL databases
    - File system
  - Storage provider (not on a server you own)
    - NoSQL databases
    - BLOB store

NoSQL

- non SQL, non-relational, "not only" SQL databases
- Emphasizes simplicity & scalability over support for relational queries
- Important characteristics
    - Schema-less: each row in dataset can have different fields (just like JSON!)
    - Non-relational: no structure linking tables together or queries to "join" tables
    - (Often) weaker consistency: after a field is updated, all clients *eventually* see the update but may see older data in the meantime
- Advantages: greater scalability, faster, simplicity, easier integration with code
- Several types. We'll look only at key-value.

Key-Value NoSQL



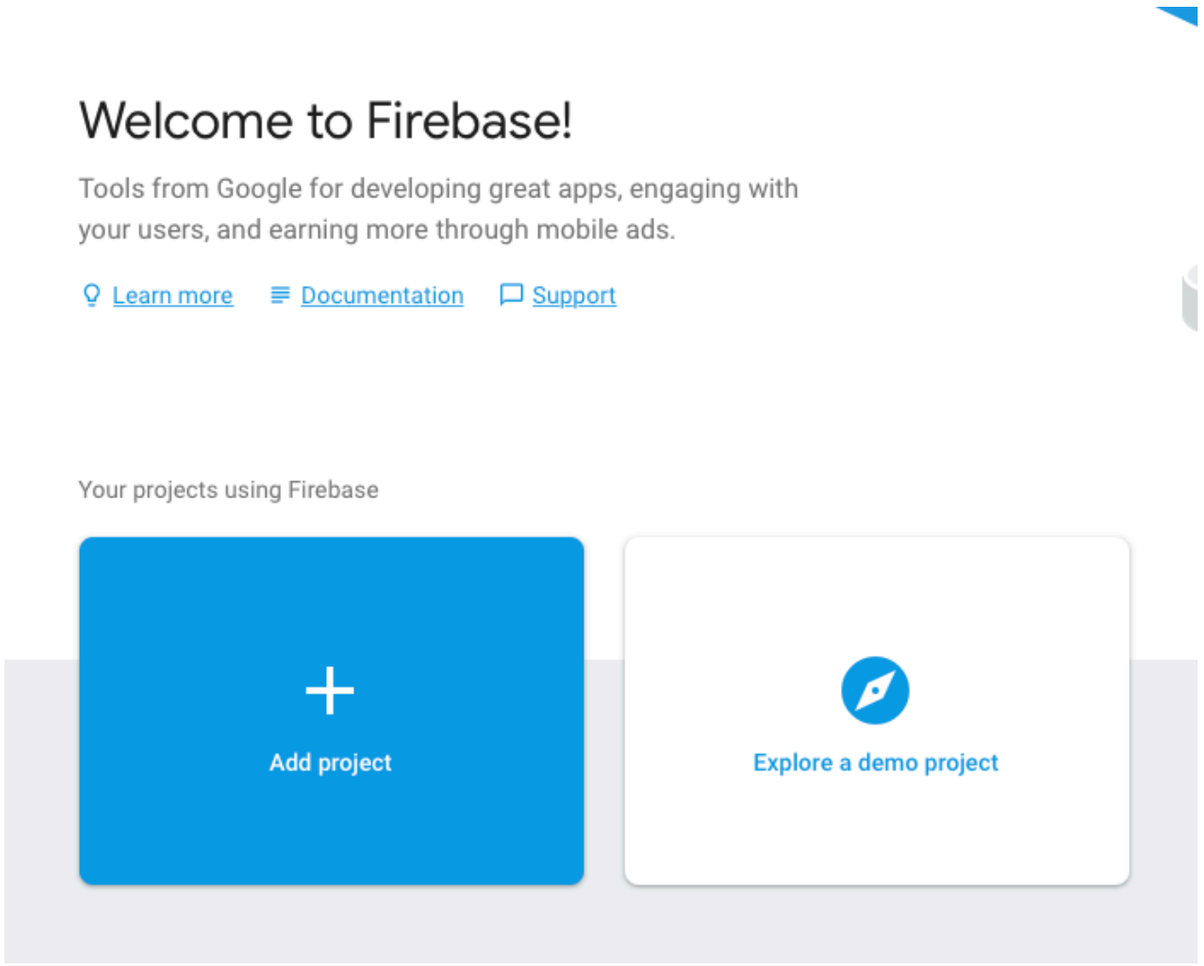
https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Firebase Cloud Firestore

- Example of a NoSQL datastore
- Google web service
  - https://firebase.google.com/docs/firestore/
- “Realtime” database
  - Data stored to remote web service
  - Data synchronized to clients in real time
- Simple API
  - Offers library wrapping HTTP requests & responses
  - Handles synchronization of data
- Can also be used on frontend to build web apps with persistence without backend

Setting up Firebase

- Detailed instructions to create project, get API key
  - https://firebase.google.com/docs/firestore/quickstart



Setting up Firebase Realtime Database

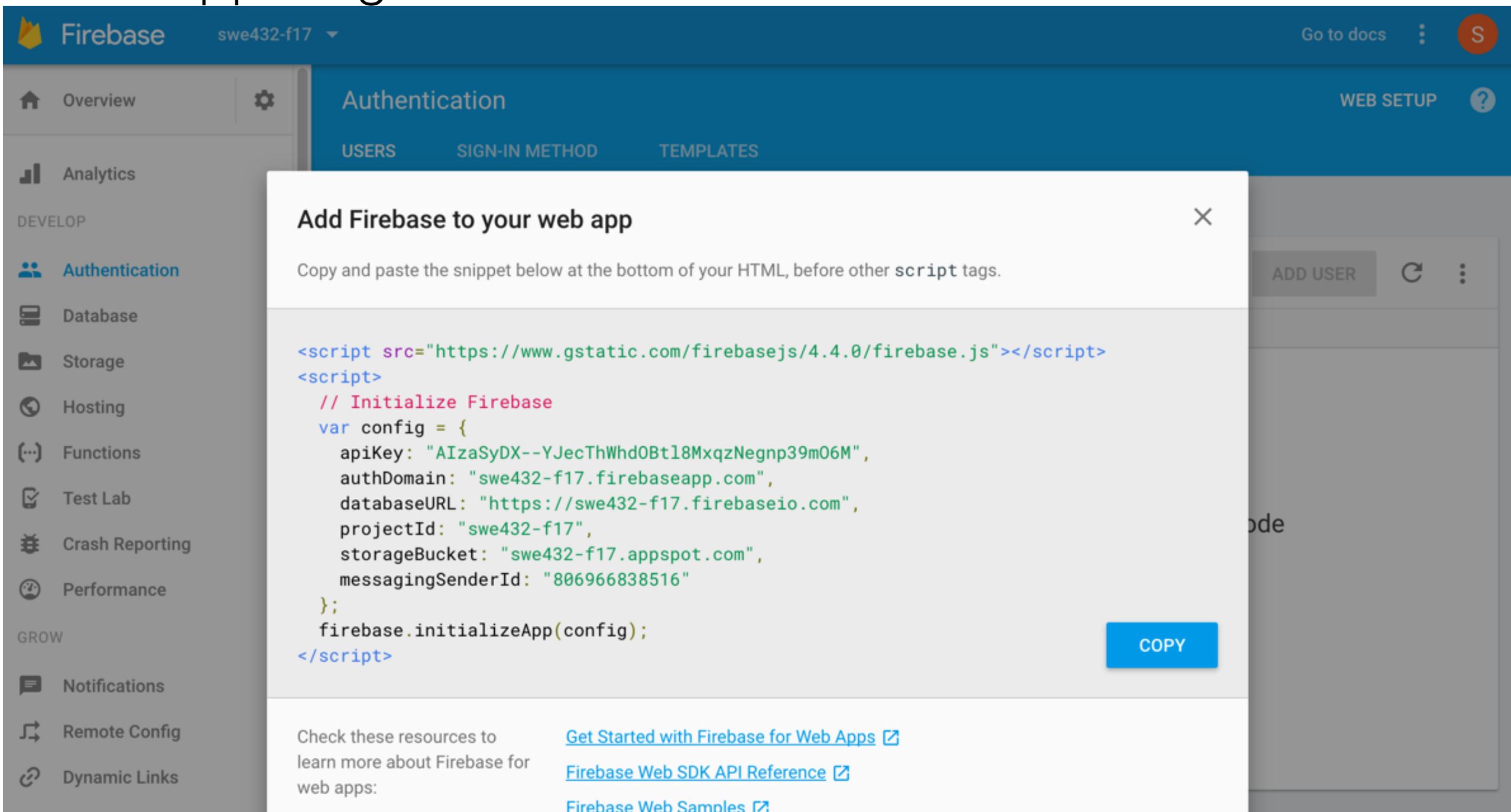
- Go to https://console.firebase.google.com/, create a new project
- Install firebase module `npm install firebase`
- Include Firebase in your web app

```
const firebase = require("firebase");

// Initialize Firebase
// TODO: Replace with your project's customized code snippet
const config = {
  apiKey: "<API_KEY>",
  authDomain: "<PROJECT_ID>.firebaseapp.com",
  databaseURL: "https://<DATABASE_NAME>.firebaseio.com",
  storageBucket: "<BUCKET>.appspot.com",
};
firebase.initializeApp(config);
```

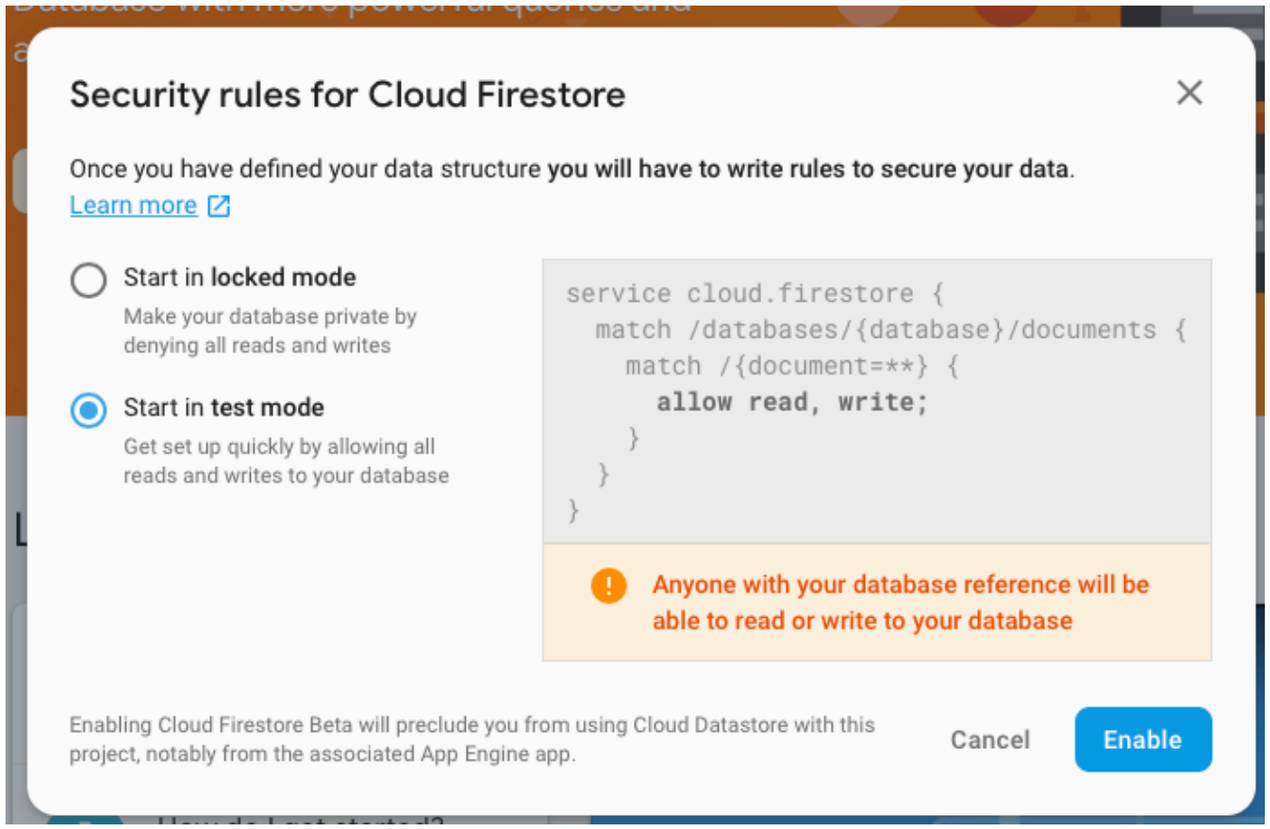
Get Config Object for your Project

- Go to Authentication, Click "Web Setup" in the upper right corner



Permissions

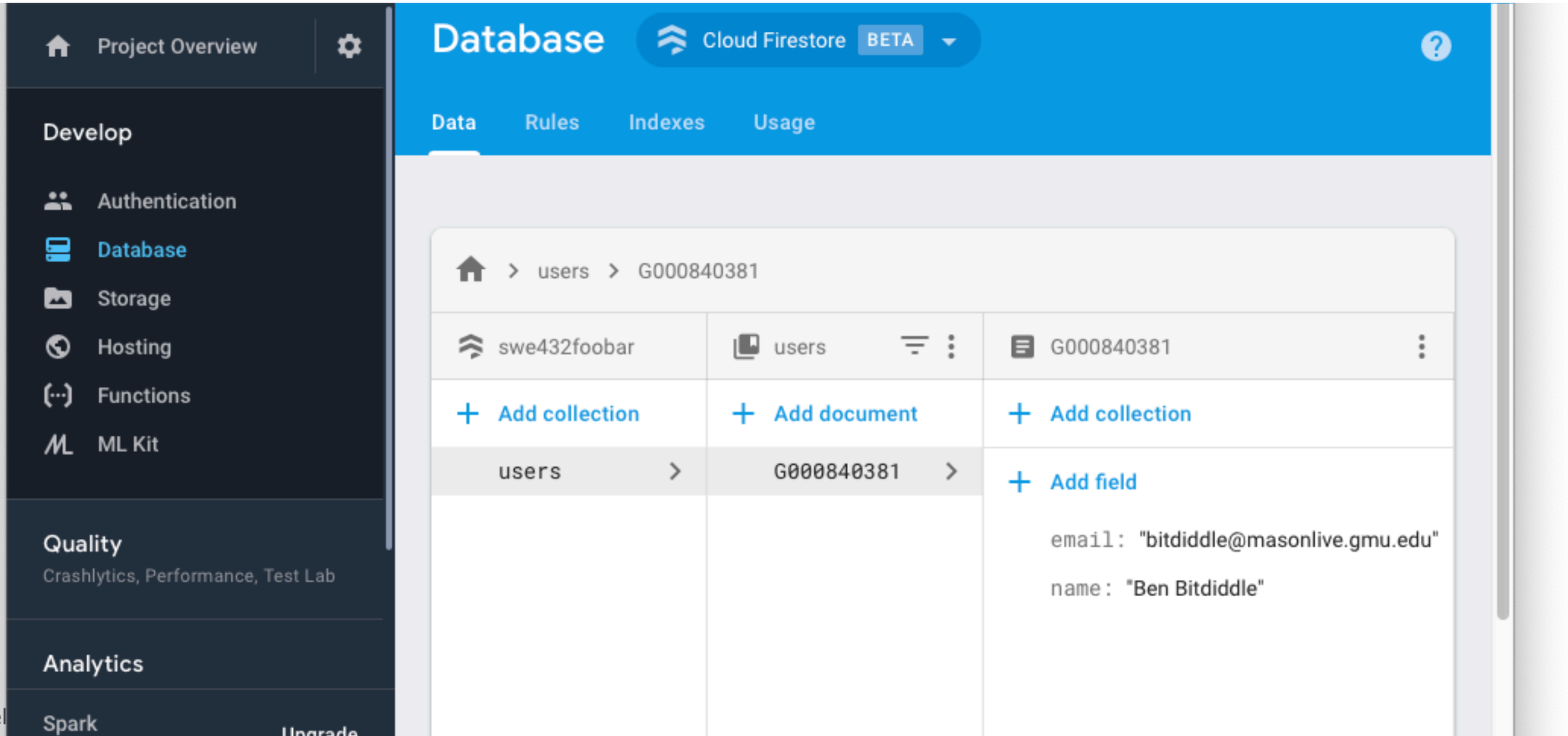
- "Test mode" - anyone who has your app can read/write all data in your database
  - Good for development, bad for real world
- "Locked mode" - do not allow everyone to read/write data
  - Best solution, but requires learning how to configure security (we'll cover in a few weeks)



Firebase Console

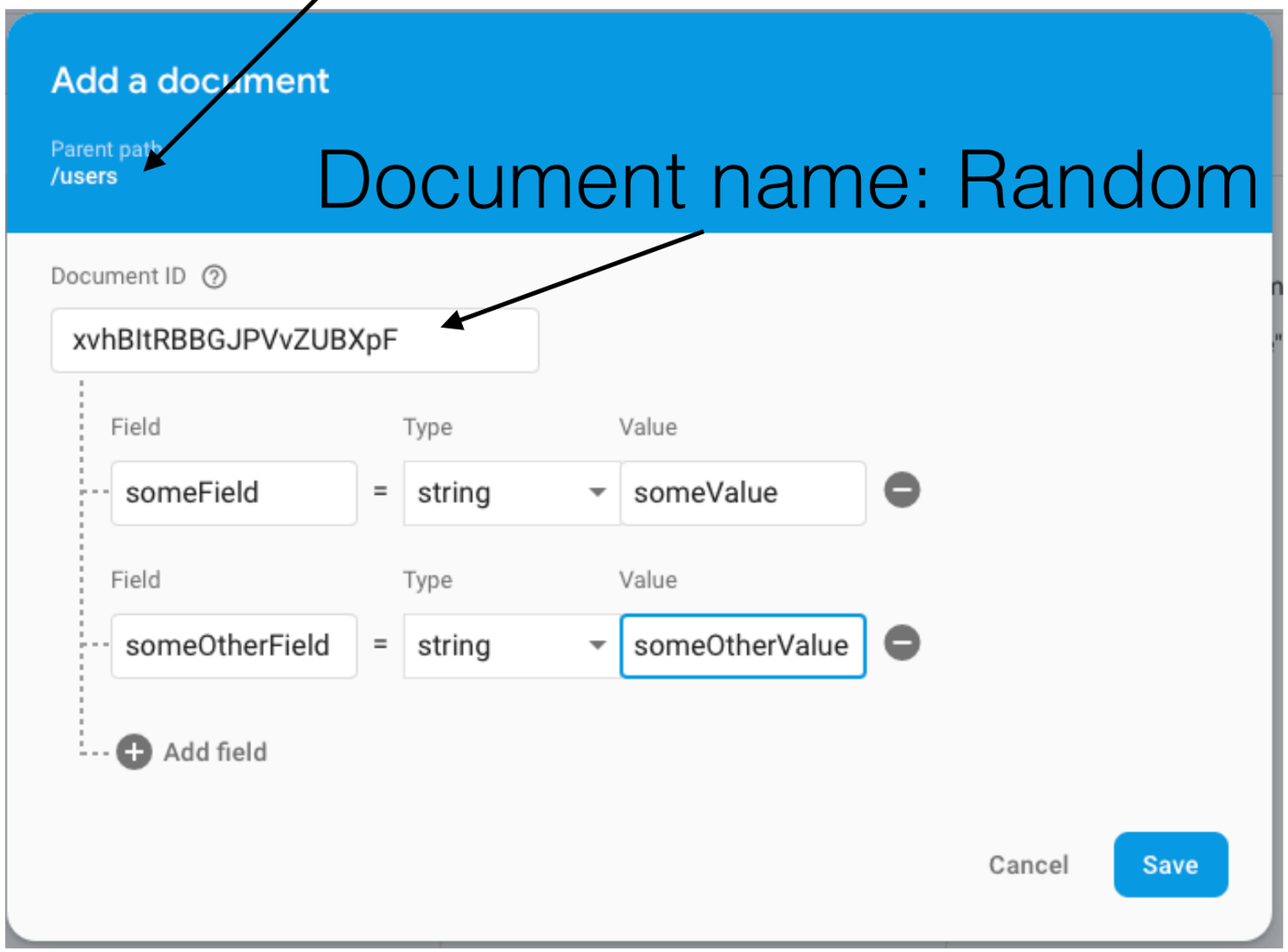
- See data values, updated in realtime
- Can edit data values

https://console.firebase.google.com

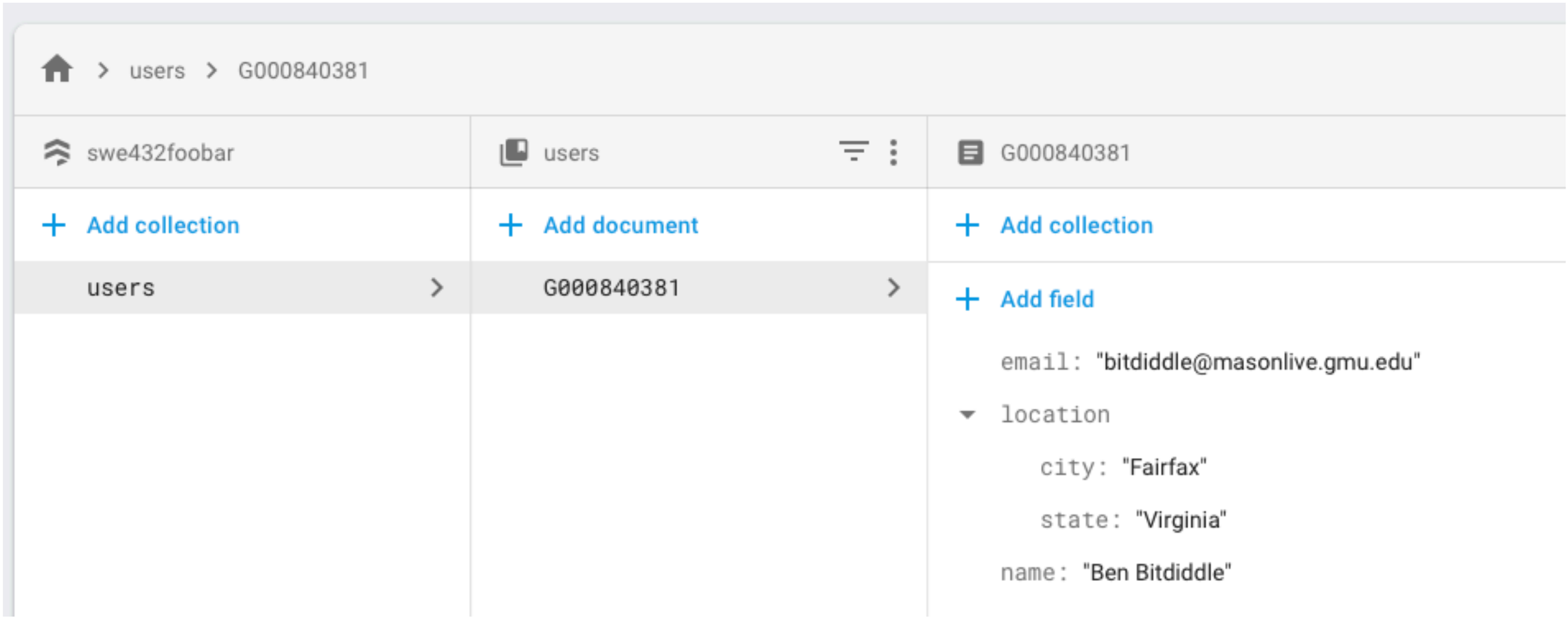


Firebase data model: JSON

- **Collections** of JSON documents
  - Hierarchic tree of key/value pairs
  - Can view as one big object
  - Or describe path to descendent and view descendent as object



JSON is JSON...


users > G000840381
swe432foobar
users
G000840381
+ Add collection
+ Add document
+ Add collection
users
G000840381
+ Add field
email: "bitdiddle@masonlive.gmu.edu"
location
city: "Fairfax"
state: "Virginia"
name: "Ben Bitdiddle"

Simple test program

- After successfully completing previous steps, should be able to replace config and run this script. Can test by viewing data on console.

```
const firebase = require("firebase");

// Initialize Firebase
var config = {
    apiKey: “...”,
    authDomain: “...”,
    databaseURL: “https://...”,
    projectId: “...”,
    storageBucket: “...”,
    messagingSenderId: “...”
};
firebase.initializeApp(config);


let database = firebase.firestore();
database.settings({timestampsInSnapshots: true})
database.collection("users").doc("G123456789").set({
    name: "Alyssa P Hacker",
    email: "aph@gmu.edu"
}).then((r) => {
    if(!r)
        console.log("Success!")
})
```

Structuring Data

- I want to build a chat app with a database
- App has chat rooms: each room has some users in it, and messages
- How should I store this data in Firebase? What are the collections and documents?

Structuring data

- Should be considering what types of records clients will be requesting.
    - Do not want to force client to download data that do not need.
- Better to think of structure as **lists** of data that clients will retrieve

Storing Data: Set

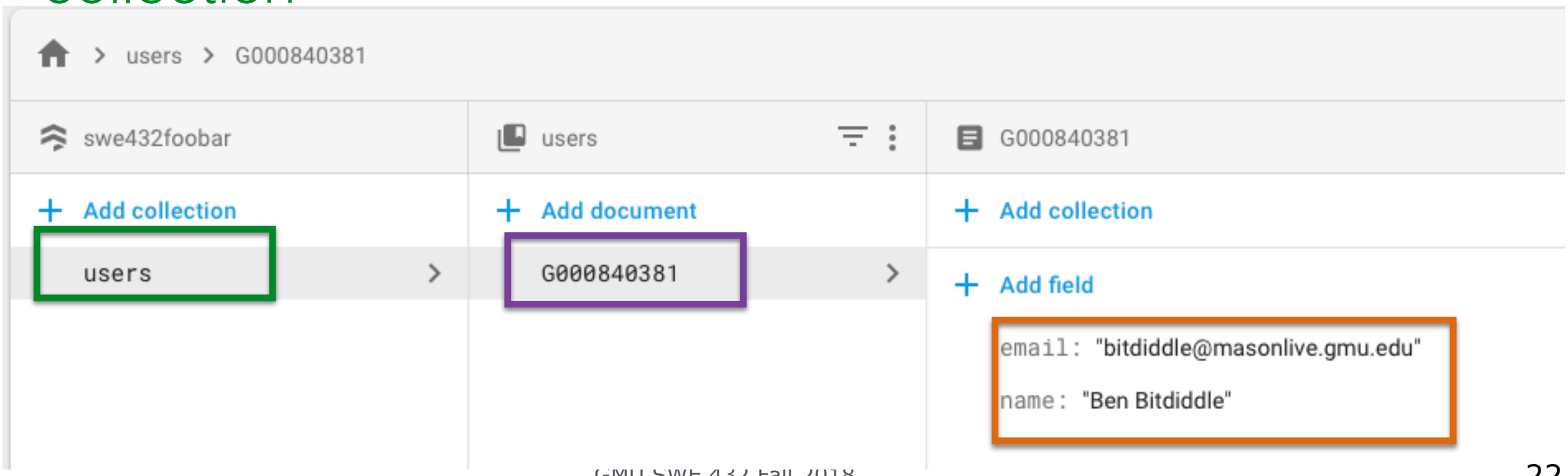
(because firebase is asynchronous)

```
async function writeUserData(userID, newName, newEmail) {
    return database.collection("users").doc(userID).set({
        name: newName,
        email: newEmail
    });
}
```

Create this one user by ID

Set the val

Get the users collection



Storing Data: Add

- Where does this ID come from?
  - It MUST be unique to the document
- Sometimes easier to let Firebase manage the IDs for you - it will create a new one uniquely automatically

```
async function addNewUser(newName, newEmail) {
    return database.collection("users").add({
        name: newName,
        email: newEmail
    });
}
async function demo(){
   let ref = await addNewUser("Foo Bar","fbar@gmu.edu")
    console.log("Added user ID " + ref.id)
}
```

Storing Data: Update

- Can either use “set” (with {merge:true}) or “update” to update an existing document (set will possibly create the document if it doesn’t exist)

```
database.collection("users").doc(userID).update({
    name: newName
});
```

Storing Data: Delete

```
database.collection("users").doc("ojtp4HrEeGB4Y9jErz0T").delete();
```

Removes a document

```
database.collection("users").doc(userID).update({
    name: firebase.firestore.FieldValue.delete()
});
```

Removes a field

- Can delete a key by setting value to null
  - If you want to store null, first need to convert value to something else (e.g., 0, '')

Fetching Data (One Time)

```
async function getUser(userId){
    return database.collection("users").doc(userId).get();
}
async function demo(){
   let user = await getUser("G000840381");
    console.log(user.data());
}
```

Can also call get directly on the collection

Putting it all together

```
var express = require('express');
var app = express();
var port = process.env.port || 3000;
const firebase = require("firebase");

// Initialize Firebase
var config = {
    apiKey: "AIzaSyBgPHiUhiub1pq_anz2IVTWlC_Y19AXhSQ",
    authDomain: "swe432foobar.firebaseapp.com",
    databaseURL: "https://swe432foobar.firebaseio.com",
    projectId: "swe432foobar",
    storageBucket: "swe432foobar.appspot.com",
    messagingSenderId: "275251107"
};
firebase.initializeApp(config);


let database = firebase.firestore();
database.settings({timestampsInSnapshots: true})

app.get('/users/:userID', function (req, res) {
    database.collection("users").doc(req.params.userID).get().then(valFromFirebase => {
        if (!valFromFirebase.data())
            res.status(404).send("Error 404");
        res.send(valFromFirebase.data());
    });
});

app.listen(port, function () {
    console.log('Example app listening on port' + port);
});
```

Listening to data changes

```
async function getAndListen(userId,callback){
    database.collection("users").doc(userId).onSnapshot(callback);
}
async function demo(){
    getAndListen("G000840381",val => {
       console.log("Got some data:" );
       console.log(val.data());
    });
}
```

"When values changes, invoke function"

Specify a subtree by creating a reference to a path. This listener will be called until you cancel it

- Read data by *listening* to changes to specific subtrees
- Events will be generated for initial values and then for each subsequent update

Ordering data

- Data is by, default, ordered by document ID in ascending order
  - e.g., numeric index IDs are ordered from 0…n
  - e.g., alphanumeric IDs are ordered in alphanumeric order
- Can get only first (or last) n elements

```
database.collection(“users”).orderBy(“users”).limit(3).get();
```

  - e.g., get n most recent news items

```
database.collection(“users”).orderBy(“users”).limit(3)
.onSnapshot(callback);
```

Blobs: Storing uploaded files

- Example: User uploads picture
  - … and then?
  - … somehow process the file?

How do we store our files?

- Dealing with text is easy - we already figured out firebase
  - Could use other databases too… but that's another class!
- But
  - What about pictures?
  - What about movies?
  - What about big huge text files?
- Aka…Binary Large OBject (BLOB)
  - Collection of binary data stored as a single entity
  - Generic terms for an entity that is array of bytes

Working with Blobs

- Module: multer

- Simplest case: take a file, save it on the server

```
app.post('/upload',upload.single("upload"), function(req, res) {
        var sampleFile = req.file.filename;
      //sampleFile is the name of the file that now is living on our server
        res.send('File uploaded!');
    });
});
```

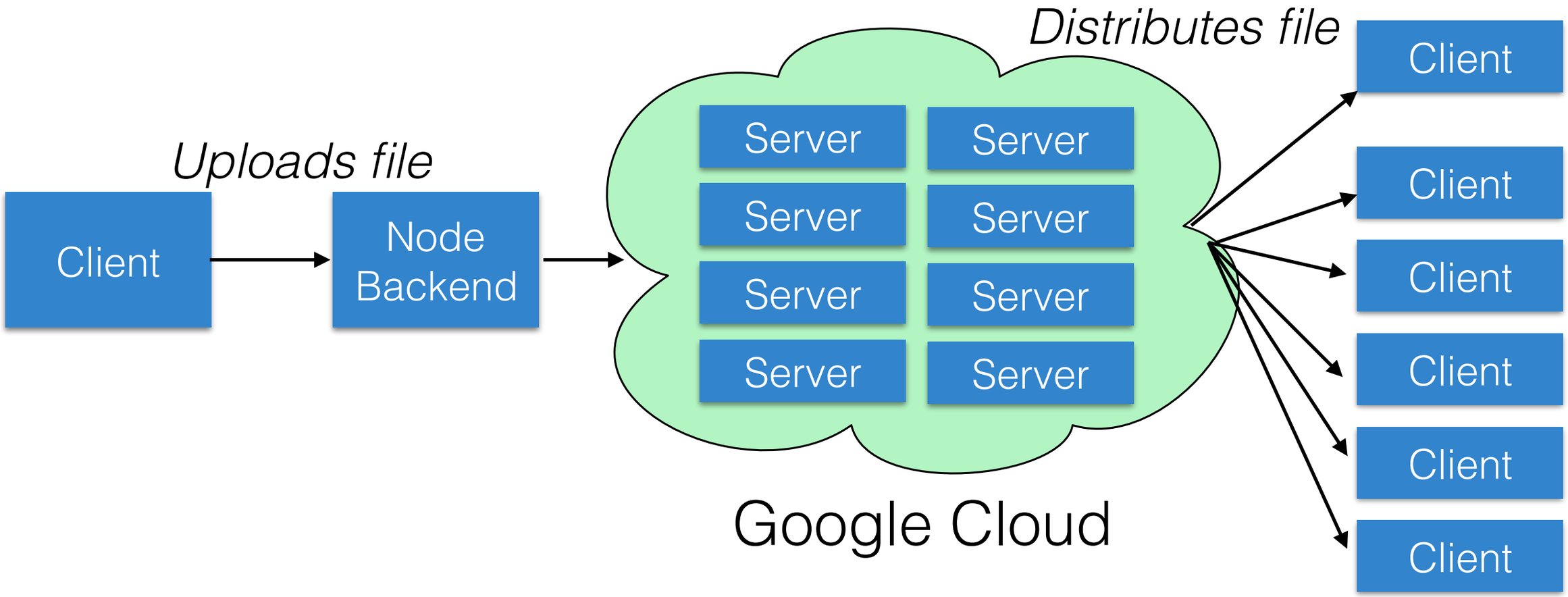
- Long story... can't easily have file uploads and JSON requests at the same time

Where to store blobs

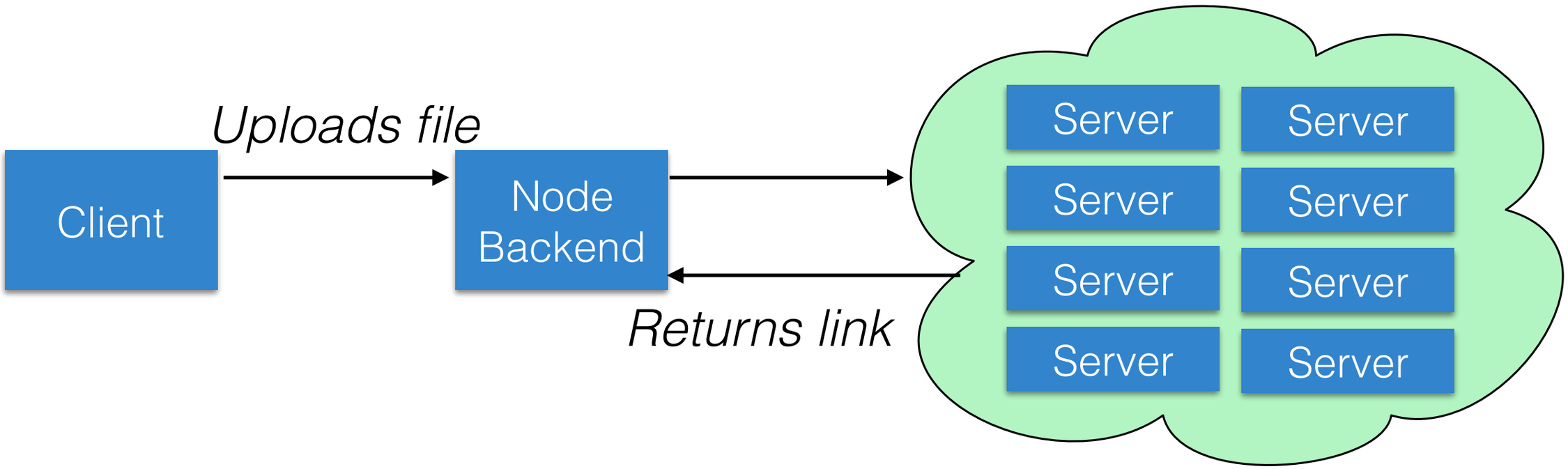
- Saving them on our server is fine, but…
  - What if we don't want to deal with making sure we have enough storage
  - What if we don't want to deal with backing up those files
  - What if our app has too many requests for one server and state needs to be shared between load-balanced servers
  - What if we want someone else to deal with administering a server

Blob stores

- Amazon, Google, and others want to let you use their platform to solve this!



Blob Stores



Typical workflow:

Client uploads file to your backend

Backend persists file to blob store

Backend saves link to file, e.g. in Firebase

Google Cloud Storage

- You get to store 5GB for free (but not used in this class)

- Setup `npm install --save @google-cloud/storage`

```
var storage = require('@google-cloud/storage');

var fs = require('fs');

// Authenticating on a per-API-basis. You don't need to do this if you auth on a
// global basis (see Authentication section above).

var gcs = storage({
    projectId: 'grape-spaceship-123',
    keyFilename: '/path/to/keyfile.json'
});

// Create a new bucket.
gcs.createBucket('my-new-bucket', function(err, bucket) {
    if (!err) {
        // "my-new-bucket" was successfully created.
    }
});
```

- https://www.npmjs.com/package/google-cloud

Google Cloud Storage

```
// Reference an existing bucket.
var bucket = gcs.bucket('my-existing-bucket');

// Upload a local file to a new file to be created in your bucket.
bucket.upload('/photos/zoo/zebra.jpg', function(err, file) {
    if (!err) {
        // "zebra.jpg" is now in your bucket.
    }
});

// Download a file from your bucket.
bucket.file('giraffe.jpg').download({
    destination: '/photos/zoo/giraffe.jpg'
}, function(err) {});
```