Unit2

* The “map” type is assigned to a nested object in firebase(there are strings, number types, and so on)
* A “document” is the minimal unit of storage in this db
* When querying an object, if it has a collection inside, it’s not shown in the object, because it’s not considered part of it. It’s better because otherwise the query would get tons of data if the document is what’s required.
* Random keys can also be generated in the client side synchronously and then sent to firestore when the connection is back. Many clients can do simultaneous operations online. Traditional sql database generates itself the id’s and it’s slower
* When returning something that has to match a type, use a cast: return <Users> {}
* Firebase SDK is promised based. AngularFire is observable based.

Firebase SDK:

import \* as firebase from 'firebase/app';

import 'firebase/firestore';

firebase.initializeApp(config);

const db = firebase.firestore();

db.doc('courses/1hegoY592OaK9gJwSSLV') //it gives a reference

      .get() //reads the document referenced above

      .then(snap => {

        console.log(snap.data());

        }

      )

db.collection('courses')

      .get()

        .then(snaps => {

          const courses: Course[] = snaps.docs.map(snap => {

            return <Course>{

              id: snap.id, ...snap.data()};

          });

          console.log(courses);

        })

  }

* stateChanges returns all the collection the first time, but then, just the type=’modified’ elements.
* snapshotChanges always return the whole collection, modified and unchaged elements.
* The live stream data connection from database is performed thanks to web sockets
* Use first() or take(1) operator to listen to the first observable value and then complete it

Queries:

return this.db.collection('courses', ref => {

      return ref.where("seqNo", ">", 0)

            .where("seqNo", "<=", 5)

    }).snapshotChanges()

    .pipe(

loadAllCourses(): Observable<Course[]> {

    return this.db.collection('courses', ref => {

      return ref.orderBy('seqNo')

            .startAfter(0) or startAt(1)

            .endAt(5)

    }).snapshotChanges()

    .pipe(

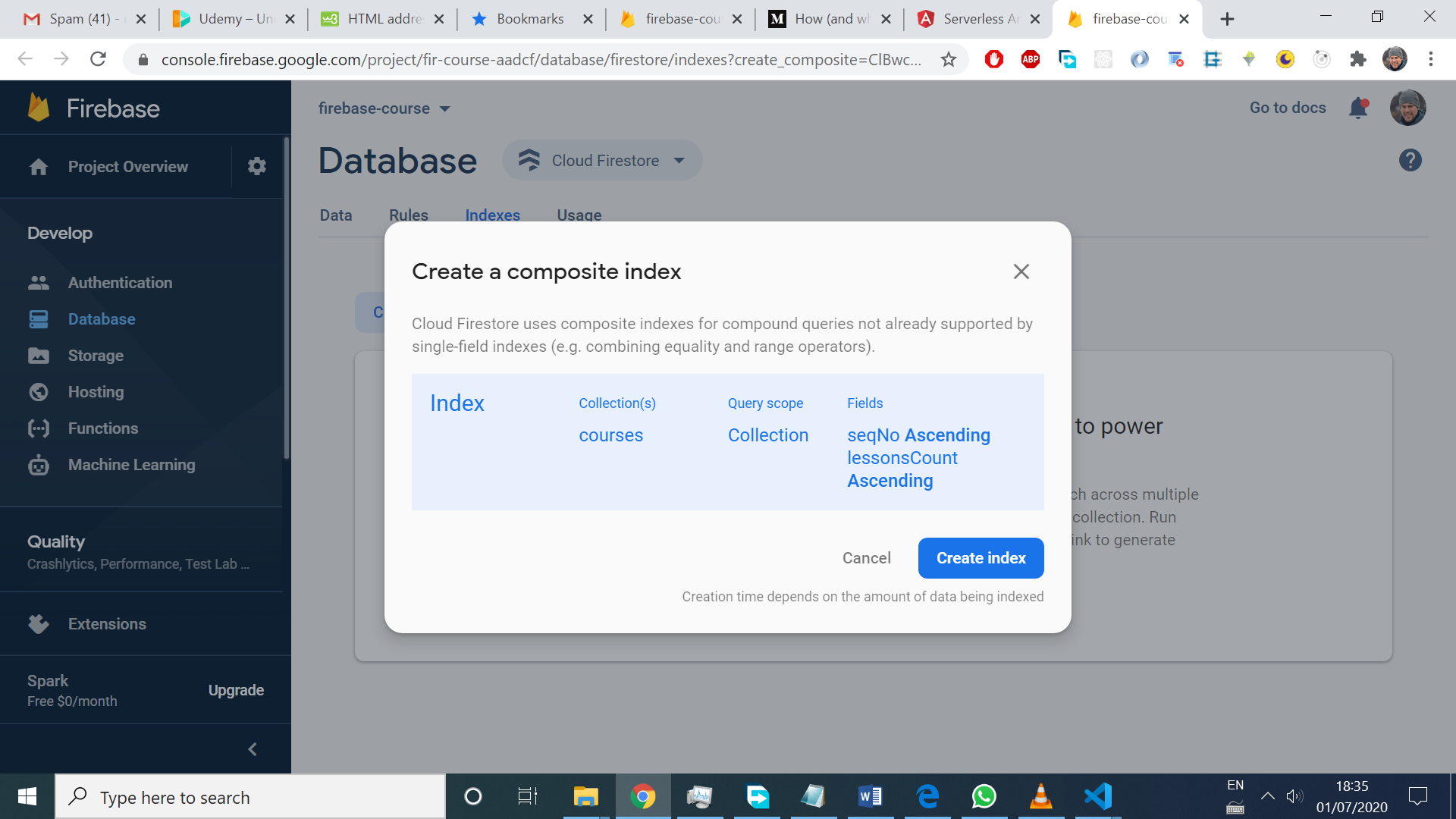
loadAllCourses(): Observable<Course[]> {

    return this.db.collection('courses', ref => {

      return ref.where('categories', 'array-contains', 'BEGINNER')

    }).snapshotChanges()

    .pipe(

* The time it takes firebase to resolve the queries depends on the amount of results, not the size of the database
* To perform the traditional table JOIN, two queries should be done and then glue the afterwards
* Firestore create indexes automatically for each field on documents, to make queries work (and not loop all the documents looking for the key ‘seqNo’, which is an expensive query.
* For compound queries, (two different fields) a creation of an index is necessary: 
* loadAllCourses(): Observable<Course[]> {
* return this.db.collection('courses', ref => {
* return ref.where('seqNo', '==', 5)
* .where('lessonsCount', '>', 5)
* }).snapshotChanges()
* .pipe(
* Every query we do on firebase needs an index (generated automatically for single fields) or created manually for compound queries.

Limitations of firestore queries

* Can’t do range queries (inequality checks) in more than one field:
* loadAllCourses(): Observable<Course[]> {
* return this.db.collection('courses', ref => {
* return ref.where('seqNo', '>=', 5)
* .where('lessonsCount', '>=', 5)
* }).snapshotChanges()
* .pipe(

Requires another index creation, because the equality and inequality where swapped with the first example.

loadAllCourses(): Observable<Course[]> {

    return this.db.collection('courses', ref => {

      return ref.where('seqNo', '>=', 5)

      .where('lessonsCount', '==', 10)

    }).snapshotChanges()

    .pipe(

* A Resolver doesn’t resolve if the observable expected doesn’t complete, so it needs to have first() or take(1). It’s tricky to resolve, because there aren’t any errors on the screen
* To run code on success or error!
* .pipe(
* finalize(() => )
* A path to a collection has an odd number of segments, and to a document, even.

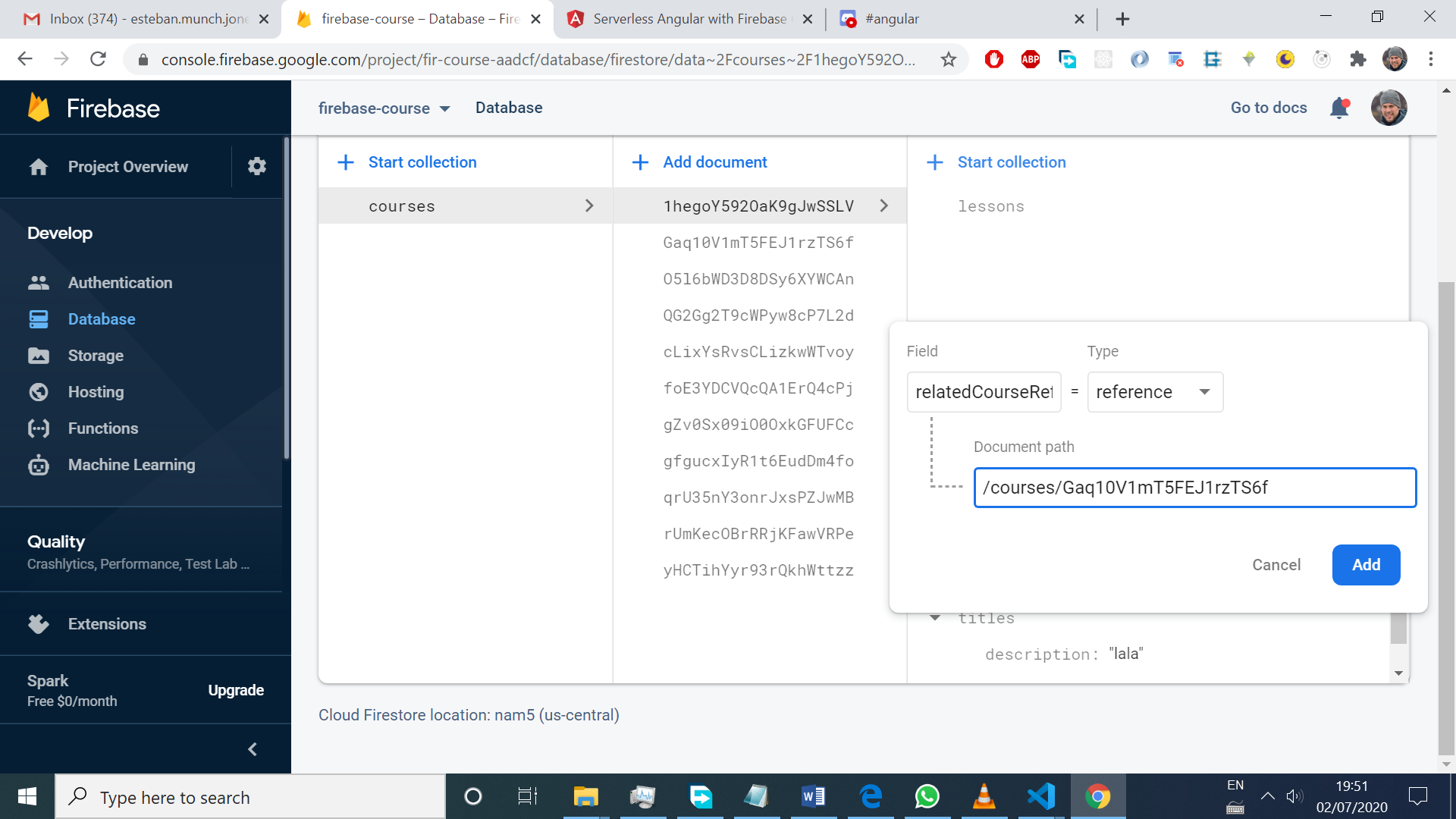
To wrap a promise with an observable, use from() operator:

It’s like an http call in angular, it’s done after returning the value once or failing

saveCourse(courseId: string, changes: Partial<Course>): Observable<any> {

    return from(this.db.doc(`courses/${courseId}`).update(changes))

  }

* To add offline capabilities, apply this method to the AngulaFirestoreModule:
* AngularFirestoreModule.enablePersistence(),
* Batch transaction done in an atomic way: all operations are done, or any of them.
* save() {
* const firebaseCourseRef = this.db.doc('/courses/1hegoY592OaK9gJwSSLV').ref;
* const deepDiveCourseRef = this.db.doc('/courses/Gaq10V1mT5FEJ1rzTS6f').ref;
* //create a batch instance
* const batch = this.db.firestore.batch();
* batch.update(firebaseCourseRef, {titles: {description: 'lala'}});
* batch.update(deepDiveCourseRef, {titles: {description: 'lalala'}});
* batch.commit()
* .then(() => {
* console.log('batch operation succesful');
* })
* }
* I couldn’t make write operation offline work. It didn’t update the modified data when still offline, and the dialog didn’t close when clicking save.
* When doing a transaction, a read lock operation (ensures that read value doesn’t change while the client side is performing the transaction) can be achieved via:
* async runTransaction(){
* const newLessonsCount = await this.db.firestore.runTransaction(async transaction => {
* console.log('running transaction');
* const courseRef = this.db.doc('/courses/1hegoY592OaK9gJwSSLV').ref;
* const snap = await transaction.get(courseRef);
* const course = <Course> snap.data();
* const lessonsCount = course.lessonsCount + 1;
* transaction.update(courseRef, {lessonsCount}) //ES6 feature, it's the same as {lessonsCount: lessonsCount}
* return lessonsCount;
* });
* console.log(`new lessonsCount now is: ${newLessonsCount}`);
* }
* If there was an external data modification of data through the transaction, it’s gonna be aborted and run again.
* To import in css files, and the location is node\_modules, use ~sign:
* @import "~firebase/dist/firebaseui.css";
* Instead of saving references to documents in the client side, I can be stored inside the database: 
* The referece may or not be valid. The reference can then be accessed from the client, and be generated as well in to different ways in the client:
* const relatedCourseRef = this.db.doc('/courses/1hegoY592OaK9gJwSSLV').snapshotChanges()
* .subscribe(snap => {
* const course:any = snap.payload.data();
* console.log(course.relatedCourseRef);
* });
* const ref = this.db.doc('courses/Gaq10V1mT5FEJ1rzTS6f').snapshotChanges()
* .subscribe(snap => {
* console.log(snap.payload.ref);
* })
* const firebaseCourseRef = this.db.doc('/courses/1hegoY592OaK9gJwSSLV').ref;

Authentication

* Error core.js:39997 Navigation triggered outside Angular zone, did you forget to call 'ngZone.run()'? After login successfully and redirected to /courses.

It happens because Angular is not aware of our method in the callback config:

onLoginSuccessful(result) {

    console.log(result);

    this.router.navigate(['/courses']);

  }

//this method is out of the Angular zone(checks for events and see if the ui needs to update)

* the same object is printed in the console for these 2 cases:
* //in app.component.ts
* this.afAuth.authState.subscribe(user => {
* console.log(user);
* })

//in login.component.ts

onLoginSuccessful(result) {

    console.log(result);

    this.ngZone.run(() => {

      this.router.navigate(['/courses']);

    })

  }

* Use dolar sign for variables that are observables myObservable$
* Variables can be created in the template an used inside ng-container !! super useful:
* <ng-container \*ngIf="(pictureUrl$ | async) as pictureUrl">
* <img [src]="pictureUrl" alt="" class="user-avatar">
* </ng-container>

Database security rules

service cloud.firestore {// the firestore service is targeted

match /databases/{database}/documents {// path to the root of the databse

match /{document=\*\*} { // targets all the documents in the parent match path The path should point to a document (odd number of segments)

allow read: if true;

allow write: if false;

}

}

}

Security rules don’t cascade to nested documents, here, the lessons collection can’t be read:

service cloud.firestore {

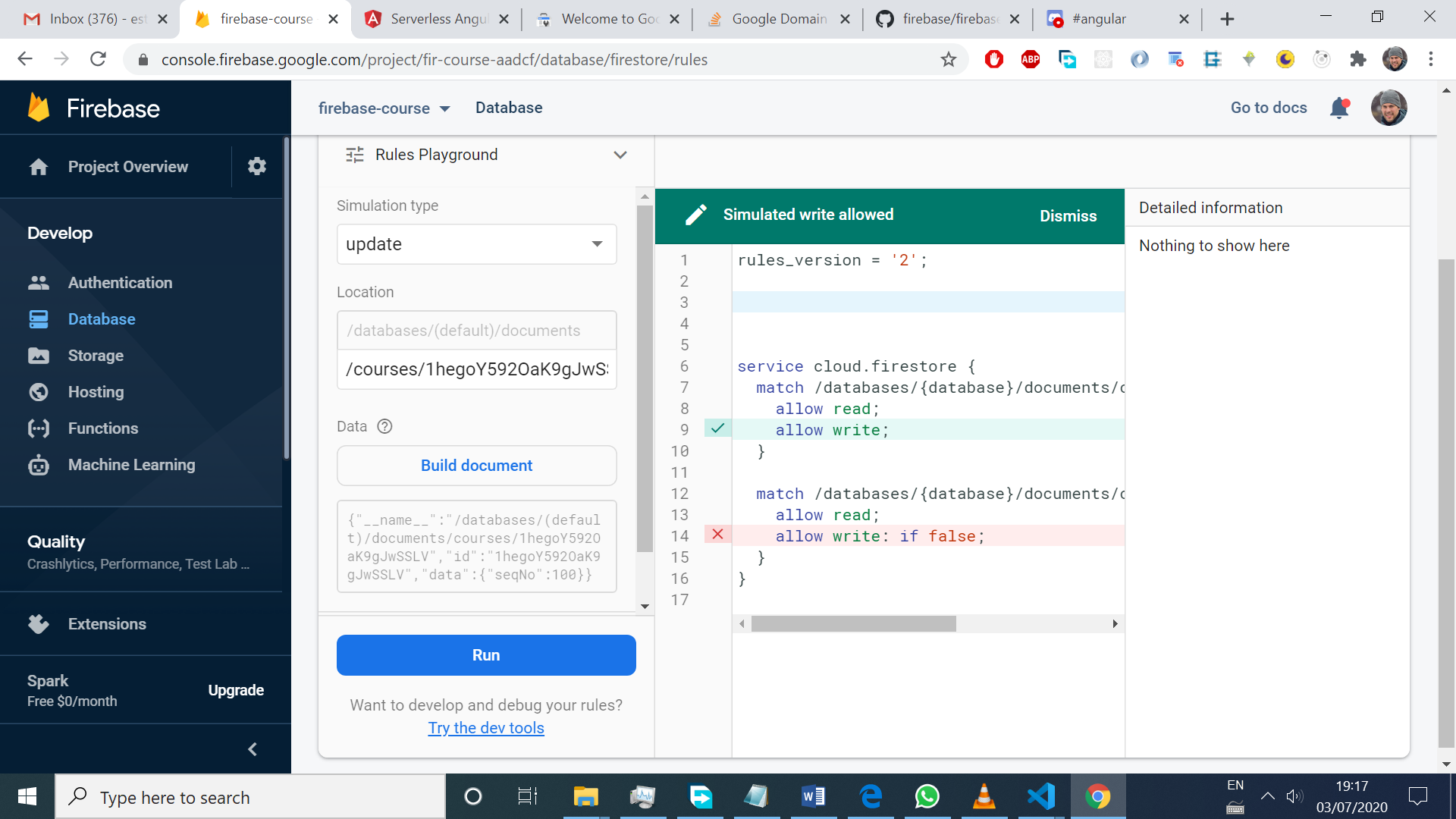
match /databases/{database}/documents/courses/{courseId} {

allow read;

allow write: if false;

}

}

* If one rule permitted a request, and other one doesn’t, the request succeeds:
* 
* The order doesn’t matter in the rules declarations

match /databases/{database}/documents/courses/{courseId} {

allow read; is the same as allow get list (it means, get the document, or the list of documents (collection) in the /courses collection, which is part of the path)

allow write: if courseId == '1hegoY592OaK9gJwSSLV';

}

match /databases/{database}/documents/courses/{courseId} {

allow read;

allow create; //write can be split into create delete and update

allow delete: if false;

allow update: if courseId == '1hegoY592OaK9gJwSSLV';

}

}

* Wildcard should be used with caution, in order to prevent permissions we don’t want
* Functions inside rules can only return a true or false value, and have only one return statement

function isValidCourse() {

return request.resource.data.seqNo is number &&

request.resource.data.lessonsCount is number &&

request.resource.data.lessonsCount >0 &&

isNonEmptyString("url");

}

function isNonEmptyString(fieldName) {

return request.resource.data[fieldName] is string &&

request.resource.data[fieldName].size() > 0;

}

* “Request.resource” access the new document, and just “resource” gives access to the old document (the one still written in the database before updating)

Storage

* A bucket is a folder where we can upload our files
* .subscribe(console.log) just logs what I get from the observable :0
* Set up rules to limit the size of the file to be uploaded

Cloud Functions

* Cloud functions use express to handle requests
* import \* as functions from 'firebase-functions';
* // // Start writing Firebase Functions
* // // https://firebase.google.com/docs/functions/typescript
* // this example only handles get requests
* export const helloWorld = functions.https.onRequest((request, response) => {
* response.status(200).json({message: 'Hello World'});
* });
* 1st step: build the cloud function locally, using cd functions, to get to that folder
* 2nd step: npm run build, that run tsc (typescript compiler, and the output is the /lib folder)
* 3rd Firebase deploy
* 4th Paste the node.js function url <https://us-central1-fir-course-aadcf.cloudfunctions.net/helloWorld> on the browser, to have a get request to that endpoint
* The above is an http get request cloud function handler
* Types of cloud functions: database triggers, storage triggers, http event (like the example) and so on.

Building custom REST endpoint with cloud functions

* 1st At the /functions folder level, npm I express

import \* as functions from 'firebase-functions';

import {db} from './init';

// const express = require('express');

// importing express on the foloowing way provides type completion for req and res

import \* as express from 'express';

const cors = require('cors');

const app = express();

app.use(cors({origin: true}));// to enable cross origin requests from 3rd party apps

app.get('/courses', async (req, res) => {

    const snaps = await db.collection('courses').get();

    const courses: any[] = [];

    snaps.forEach((snap: { data: () => any; }) => courses.push(snap.data()));

    res.status(200).json({courses});

});

export const getCourses = functions.https.onRequest(app);

const admin = require('firebase-admin');

admin.initializeApp(); // it uses the firebase config depending on the environment

export const db = admin.firestore();

* To test before deploying , an emulator can be used, with npm run serve
* To depoy functions only, firebase deploy –only functions
* To see the logs of the functions, npm run logs o on the dashboard. Using console.log on the function code, will be available on npm run logs or the dashboard

Database trigger function

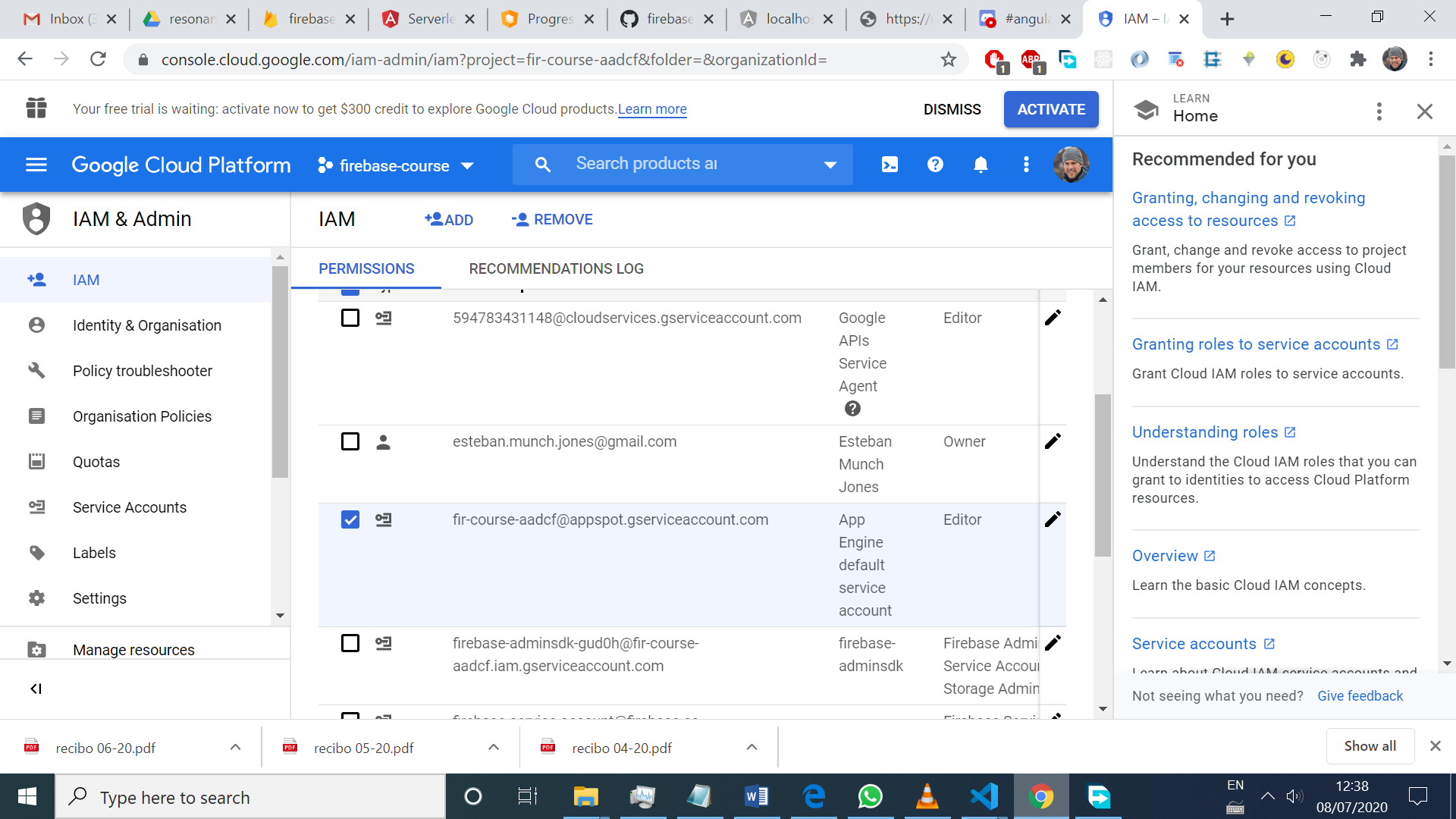
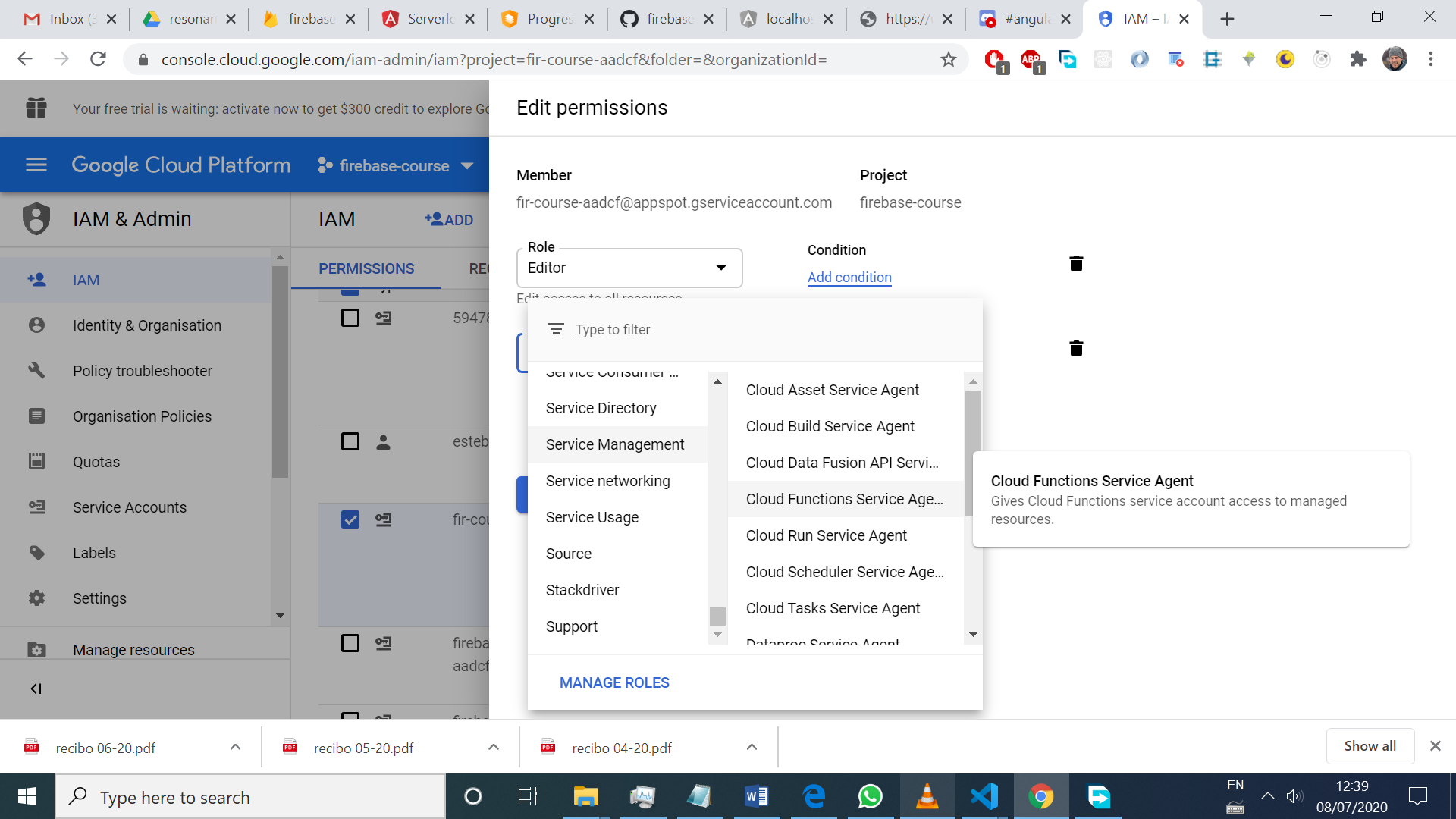
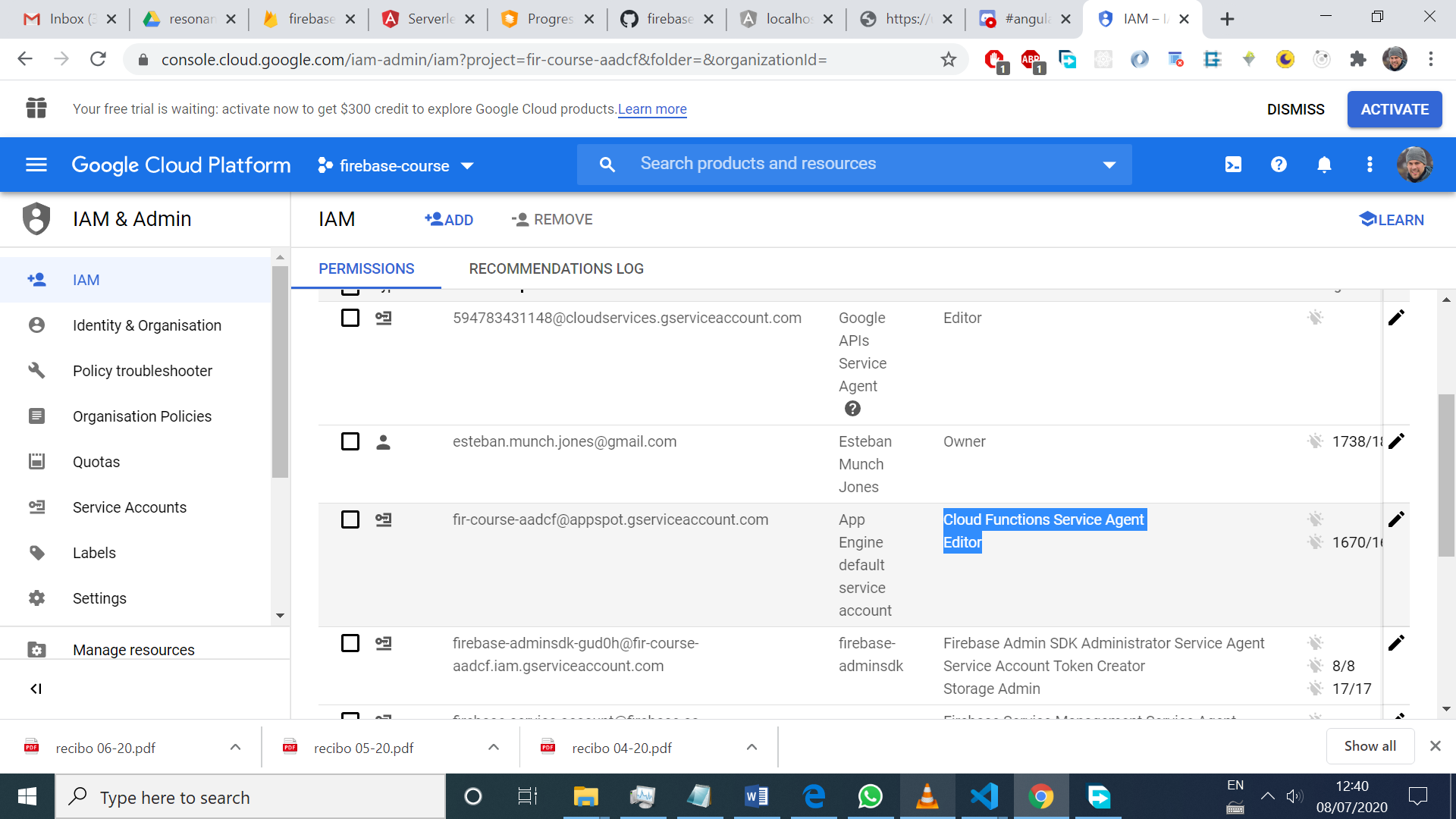
* Upon addition or deletion of a lesson, to update lessonsCount
* Using transaction.get() we are sure we read the data in a transactional way, and that value isn’t gonna be modified.
* To deploy only the functions I want, firebase deploy –only functions:<funcName>, functions:<funcName>

resizeThumbnail function

To apply

const url = thumbnail.getSignedUrl({action: 'read', expires: new Date(3000,1,1)});

we need to get permissions to do so through google cloud platform:

* 
* Edit
* 
* 

How to add multiple sites to a firebase project?

firebase target:apply hosting ***TARGET\_NAME*** ***RESOURCE\_NAME***

example:

firebase target:apply hosting **blog myapp-blog**

firebase.json

{  
  "hosting": {  
      **"target": "blog",**  
      "public": "dist",  
  
      // ...  
  
      "rewrites": [...]  
  }  
}

firebase deploy --only hosting:**TARGET\_NAME**