GETTING STARTED

WITH FIREBASE

PLATFORM : ▷

STARTED : 7th JAN 2023

TUTOR : Netninja

FINISHED : 8th JAN 2023

DURATION : 14 Videos in Playlist

INTRO & WHAT'S NEW #1

Alrighty, Welcome guys to this wonderful little journey into being a Firebase Ninja.

Recently firebase introduced version 9 of the library and the way we use version 9 is a bit different as to how we've been using firebase in the past (previous versions)

WHAT IS FIREBASE ?

Firebase is a mobile and web application development platform developed by Google. It provides a number of services that can be used to build, test, and deploy applications.

Features/Services of FIREBASE

- Real-time database
- Authentication
- Hosting
- Cloud Functions
- Cloud Messaging
- Cloud Storage

Firebase is widely used by developers and it is particularly well-suited for real-time, data-driven applications and has a large and active developer community

We can plug this services provided to us by firebase directly into our Frontend Applications or Websites to make use of them and then firebase helps us with the Heavy-Lifting.

In Summary, It's an alternative to setting up our own back-end Infra-structure using tools like Mongo Db and a Node Js server.

Now:

It's really simple to get started with and most of it's features are completely free for small-personal websites.

STEPS to GETTING STARTED

- Visit 'firebase.google.com'
- Sign up or Login
- Go to Home and click on 'Go to Console'

↳ This is where all your firebase projects are listed

Now the major change in version 9 of firebase is that it now adopts a more modular & functional approach.

MEANING

▷ We only import the firebase functions that we need from the libraries and not those we don't need, which ~~improves~~ facilitates tree-shaking

(removal of unused code) making our web apps as small in size and as fast as possible

Same Code, Different Versions

- Firebase 9

```
import { getAuth, createUserWithEmailAndPassword } from "firebase/auth";

const auth = getAuth();
createUserWithEmailAndPassword(auth, email, password)
  .then((userCredential) => {
    //signed in
    const user = userCredential.user;
    //...
  })
  .catch(error) => {
    const errorCode = error.code;
    const errorMessage = error.message;
    //...
  })
```

- Firebase 8

```
firebase.auth().createUserWithEmailAndPassword(email, password)
  .then(userCredential) => {
    ...
  })
  .catch(error) => {
    ...
  })
```

What We Will Be Covering

- How to use Version 9, using the Firestore database
- Firebase Auth Service and so on.

Setting Up Webpack #2

In order to take advantage of tree shaking in Firebase 9 (to remove any unused code), we need to use a module bundler like "webpack".

Because the module bundler is what bundles all our different pieces of imported source code together into a final Javascript bundle file and it's that Javascript bundle that we would link to from our HTML page.

(e.g. npx create-react-app)

NOTE: If you're creating your website with CLI tool, then most of the time - webpack or most of the module bundler is automatically setup for us

And In such a case we could Immediately start using Firebase 9 from the "getgo".

However, if you aren't using a CLI tool to build ur website, then u'll need to setup a module bundler yourself.

Regardless, Let's Setup

- FIREBASE-9-DOSO — # Create New Working Directory
  - src — # Create src folder
    - index.js — # Create our Javascript entry file
  - dist — # This is where our final bundled JS code is going to live
    - index.html → This will be the webpage we serve to the browser and will hook up later to our bundled JS file

Boilerplate Code Setup

index.html

snippet: doc

```
<html lang="en">
<head>
  ...
  <title> Firebase 9 </title>
</head>
<body>
  <h1> Greeting Started With Firebase 9 </h1>
</body>
</html>
```

Install Webpack and Configure It

Terminal # Open CLI

```
> npm init -y
```

This creates new package.Json file to help keep track of all our different dependencies

```
> npm i webpack webpack-cli -D
```

Installing 2 packages

-D → saves to our dev dependencies

webpack → webpack (module bundler)

webpack-cli → This helps us run webpack

FIREBASE-9-DOJO

- webpack.config.Js # Create File in CWD

  # Inside this file is where we configure what we want webpack to do. Which is to look at our source index.Js file and any imports we have their as well - and then bundle all of that code together into a single bundle file

≡ Add Webpack Configurations and export it

☐ webpack.config.js

```
+ const path = require('path')

+ module.exports = {
    mode: 'development',          # Can also be 'production
    entry: './src/index.js',      # our JS source file relative Path
    output: {
      path: path.resolve(__dirname, 'dist'),   # A path to where we want our output to be put
      filename: 'bundle.js'
    }
    watch: true                   # watches our entry file for any change and then automatically updates our bundle.js
}
```

≡ Add Custom Script that allows us run webpack

☐ package.json

```
{
  ...
  "scripts": {
    "test": "...",
    "build": "webpack",           # When ran, this will now autom.. find our webpack.config.js and RUN it.
  }
  ...
  "devDependencies": {
    "webpack": "^5.60.0",
    "webpack-cli": "^4.9.1"
  }
}
```

≡ Before Running Webpack - Add some code into our source index.js file

○ index.js

```
console.log('hello from index.js');
```

≡ Run Webpack Finally

⊙ Terminal

```
> npm run build
```

When ran, webpack goes and takes our source code - bundle it - and places the file in dist

≡ Check dist folder

○ dist

bundle.js # Pops Newly Added

...

NOTE: Thanks to watch: true, webpack now watches the source file and anytime we make a change to it and save - Webpack automatically rebundles it.

○ index.html ≡ Add bundle.js file to our index.html

...

+ <script src="bundle.js"></script>
</body>

≡ Lastly Run Live Server Vs Code Extension on our index.html

Browser Pops

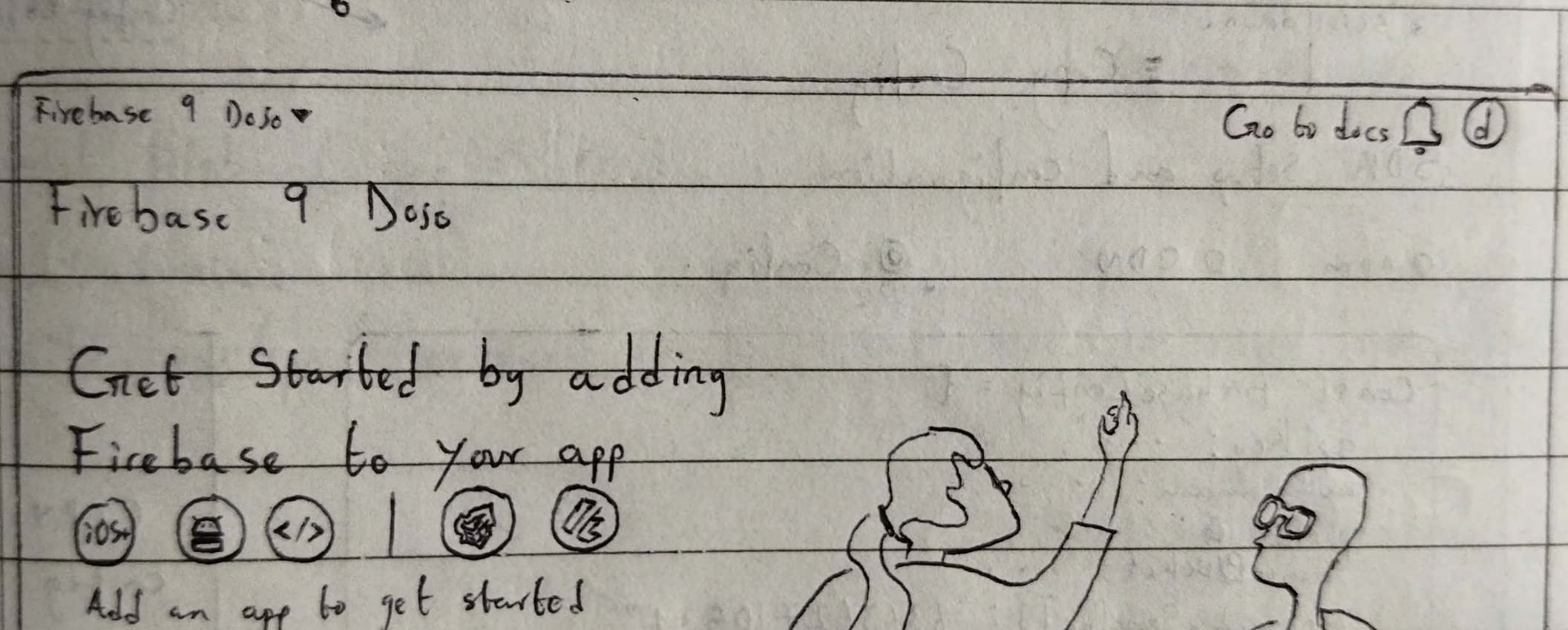
Getting Started With Firebase 9

> hello from index.js

Creating a Firebase Project #3

Now we've webpack setup let's create a new firebase project

- Go to your firebase console - [console.firebase.google.com]
- Click [Create a Project]
- Name it Firebase 9 Dojo
- Disable Enable Google Analytics for this Project
- Click [CREATE]



This Is the [DASHBOARD] where we can Control the backend after registering a frontend (app)

When we create a new firebase project we have to make a new frontend application for the project as well

- Click on web
- Enter App name firebase 9 dojo website
- Click [Register] skip npm... - Click [Continue to Console]

Now we should see the new app we just created

Firebase 9 Dojo

1 app

1 To see all apps

1 apps in project

Firebase 9 dojo website
Web App

2 This represents our frontend web application

Let's grab the Config Object

3 scroll down

≡ Copy Config

SDK Setup and configuration

○ npm ○ CDN ◉ Config

4

```
const firebaseConfig = {
  apiKey: "...",
  authDomain: "...",
  projectId: "...",
  storageBucket: "...",
  messagingSenderId: "631362710531",
  appId: "1:..."
};
```

5 Copy all

COPY config

To be used in our frontend to connect to our current Firebase project

≡ Go Back To Our Frontend and Add/Paste Config Object

index.js

```
const firebaseConfig = {
  ...
}
```

PASTED!

≡ Install Firebase

Terminal
>npm install firebase

Now we've installed the Firebase package, we can go ahead and use it to connect to the firebase backend along with the firebase Config object.

≡ Now Initiallize our Firebase Backend / Connect to It

○ index.Js

+ import { initializeApp } from 'firebase/App'; # Tree shaking in firebase 9
- Importing only the functions to be used

const firebaseConfig = {...};

//init firebase app

+ initializeApp(firebaseConfig); # Initializes / Connects to our backend

Firestore Setup & Fetching Data #4

Now we've successfully connected to our firebase backend, Next we want to setup a database and then connect to that db from our frontend

Firestore Database
1. → newer
Realtime Database
→ older

Create Database
2.

Create database
① Secure rules for Cloud Firestore
○ start in production mode
◉ Start in **test** mode
3.
↳ Important to start in test mode, if not you won't be able to access your data
Fire Store Rules.

Next
4.

Cloud Firestor location
eur3 (europe-west)
5. select location

Enable
6. Complet

Cloud Firestore

Data Rules Indexes Usage

fir-9-dojo

+ Start collection

Database Created Successfully!

This Database is split up into Collections & Documents. E.g
- We would have a collection of a certain data type (e.g books)

≡ Create a Books Collection

+ Start Collection
1.

Collection ID
books
2. Enter

Next
3.

Start a Collection

① Give the collection ID ——— ② Add it's first document

Document parent path

/books

Document ID

... Auto-ID # each document in a collection should have a unique document id

Field | Type | value

... title = String "the name of the wind" ⊖

Field | Type | value

... author String patrick rothfuss ⊖

... ⊕ Add Field

Save

With all that done - We should now have a new collection with a single document inside of it

+ Start collection	+ Add document	+ Start collection
books >	1PqXjrRk1ybvs ... >	+ Add field
		author: "patrick rothfuss"
		title: "the name of the wind"

ASSIGNMENT : Add 2 more documents to the books collection

NEXT UP:

≡ Initialize / Connect to our newly created firestore database

☐ index.js

```
1 ...
+ 2 import { getFirestore } from 'firebase/firestore'
```

// init services

+ const db = getFirestore() # This now holds our firestore db Connection

// collection ref

+ const colRef = collection(db, 'books') # This grabs a particular collection in our firestore db the colRef

#Add Import

// get collection data

+ getDocs(colRef) # Retrieves all the document in collection

#Add Import

.then((snapshot) => {

console.log(snapshot.docs) # snapshot.docs gives us all the documents

})

≡ Add Imports

```
import {
  getFirestore, collection, getDocs
} from 'firebase/firestore'
```

≡ Save and Preview the results in a browser

Crafting Started with Firebase 9

>(3) [Nu, Nu, Nu]
▼0: Nu
▷ metadata: ...
- converter: ...
▷ document: ...
▷ id: ...
▽ Prototype
▷ data: f data(t={})

→ We see our documents printed out, but we aren't interested in all the gibberish. What we want is the actual data in the data() function for each of our docs item

≡ Re-Write Code to print list of books document data

```
v getDocs(colRef)
  .then((snapshot) => {
    let books = []
    snapshot.docs.forEach((doc) => {
      books.push({ ...doc.data(), id: doc.id })
    })

    console.log(books)
  })
+ .catch(err => {                    # Add catch just incase any errors
                                      occur
    console.log(err.message)
  })
```

≡ Save and Preview Once More

```
▽ (3) [{...}, {...}, {...}]
  ▽ 0:
      author: "patrick rothfuss"
      id: "..."
      title: "the name of the wind"
  ...
```

Adding & Deleting Documents #5

Now we want to see how we can add documents and delete documents.

index.html ≡ Add 2 forms for adding and deleting

```
...
<body>
...
  <h2> Firebase Firestore </h2>


  <form class="add">                                  # Add Form
    <label for="title"> Title: </label>
    <input type="text" name="title" required>         - title
    <label for="author"> Author: </label>
    <input type="text" name="author" required>        - author


    <button> add a new book </button>
  </form>

  <form class="delete">                               # Delete Form
    <label for="id"> Document id: </label>
    <input type="text" name="id" required>            - id


    <button> delete a book </button>
  </form>


  <script src="bundle.js"> </script>
</body>
```

D Index.Js

```
import {
    ..., addDoc, deleteDoc, doc
} from 'firebase/firestore';
```

→ This gets us a reference to a doc which helps us in deleting.

... ↓ (# To Bottom)

```
// Adding documents
const addBookForm = document.querySelector('.add'); # Grabs Add Form
addBookForm.addEventListener('submit', (e) => { # Add Submit Listener
    e.preventDefault();

addDoc(colRef, {
    title: addBookForm.title.value,      # Grabs value of title input
    author: addBookForm.author.value,    # Grabs value of author input
})
.then(() => { addBookForm.reset(); });
})
```

→ Reset Forms Input Fields

```
// Deleting documents
const deleteBookForm = document.querySelector('.delete'); # Grabs Delete Form
deleteBookForm.addEventListener('submit', (e) => { #Add Submit Listener
    e.preventDefault();

    const docRef = doc(db, 'books', deleteBookForm.id.value);
    deleteDoc(docRef)
        .then(() => { deleteBookForm.reset() })
})
```

→ Grabs doc ref

≡ Lastly, Test out by adding 2 new doc and deleting 1

@

Getting Started With Firebase 9

Firebase Firestore

Title: [ ] Author: [ ] [add a new book]

Documentation id: [ ] [delete a book]

Expected Result

▾ (4) [{...}, {...}, {...}, {...}]

NOTE: Manually refresh browser after every submit to see latest state of ~~data~~ our books collection

Real Time Collection Data #6

We've learnt how to add data and remove data which is great, but in the moment in order to see the updated collection data - we would have to refresh the page :(.

To See updates in REAL-TIME, We can setup a real-time listener or a subscription to the firestore collection which actively listens for changes to that collection.

And How Do We Setup This Subscription?

We do so by using another firebase provided function called onSnapshot - And this would actively listen to our collection for any changes (unlike getDocs())

index.js ≡ Replace getDocs() with onSnapshot()

```
// real time collection data
onSnapshot(colRef, (snapshot) => {        # This Callback function runs anytime our collection changes
  let books = [];
  snapshot.docs.forEach(doc => {
    books.push({ ...doc.data(), id: doc.id })
  });
  console.log(books);
})
```

Assignment: Test our subscription out by Adding 3 new books and removing 1

FIRESTORE QUERIES #7

Sometimes we might want to grab a specific document with a matching property/field - We do this using FIRESTORE QUERY.

index.js ≡ Search or Get a Particular Document

```
import {..., query, where} from firebase/firestore;

// collection ref
...

// queries
+ const q = query(colRef, where("author", "==", "patrick rothfuss"))

// real time collection data
v onSnapshot(q, (snapshot) => {...});
```

Check Browsers Console to see that this filters our docs and only gives us a snapshot of books where the author is "patrick rothfuss"

```
▼(1) [{...}]
  ▶0: {author: 'patrick rothfuss', title: '...', id: '...'}
```

NOTE: Once done revert this lecture changes

ORDERING DATA & TIMESTAMPS #8

We've seen how to use the getDocs() function and the onSnapshot() function. But now the data we fetch are not in order.

So now we want to learn how to ORDER our records when fetched

index.js

```
import { ..., orderBy } from 'firebase/firestore';   # New function
...
//queries
const q = query(colRef, where(...), orderBy('title', 'desc'))
```

- remove this

↳ This now orders each document by the title in desc... order

But if we preview this in our browser console we see an Error

CONSOLE

⊗ Uncaught Error in snapshot Listener: index.esm2017.js:13906
FirebaseError: The query requires an index. You can create it here: https://console.firebase.google.com/v1/r/project

1. ↳ Click on link to create Index

'Next Page', Pop up

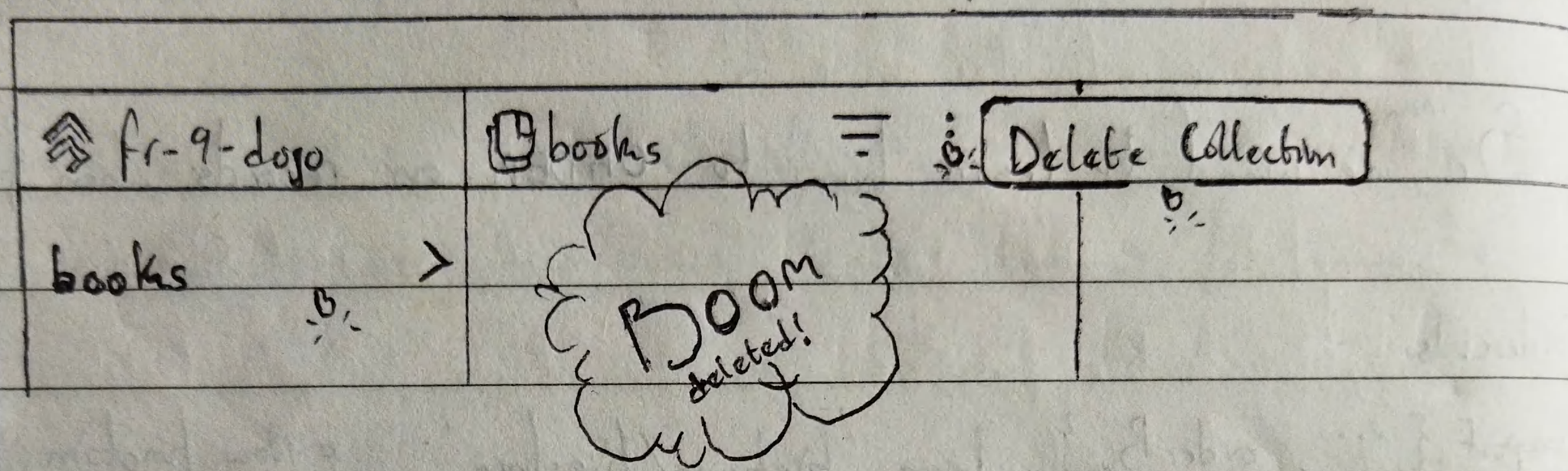
Create Index

2. → NOW wait for Index to be created and once done go ahead and refresh browser to see or documents in a descending order

(Timestamp)

≡ Next Up we want to add a new property to our books.

- Go to Firestore Database
- Delete all the documents in the books collection



index.js

```
import { ..., serverTimestamp } from 'firebase/firestore';

// adding documents
const addBookForm = document.querySelector('.add');
addBookForm.addEventListener('submit', (e) => {
  ...
  addDoc(colRef, {
    ... : ...,  x2
    createdAt: serverTimestamp()
  })
  ....
})
```

This creates a timestamp for our firestore DB

Assignment: Order our data fetched by our createdAt property

Assignment: Add 4 books using our frontend and then confirm in our console

NOTE: Notice whenever we add a book, our console prints the new snapshot twice. This is because serverTimestamp() takes some time to finish and then when it does it updates CreatedAt which then sends the 2nd snapshot

Fetching Single Documents #9

In order to fetch a single document from ~~the~~ our firestore db, we will be using a new function - getDoc()

index.js

```
import {..., getDoc} from 'firebase/firestore';
...↓

// get a single document
const docRef = doc(db, 'books', ' Paste Id Here ');


getDoc(docRef)
.then((doc) => { console.log(doc.data(), doc.id); });
```

```
▷ (4) [...]
▽ {title: "the wise", createdAt: it, author: "..."}
'LHhPBvdwn4k6THW8Uypb'
```

≡ Setup a subscription on ~~[illegible]~~ a Single Document docRef

```
onSnapshot(docRef, (doc) => { console.log(doc.data(), doc.id); })
```

Open 2 Browser Tabs to test Subscription

Console

```
▷ ...
...
▷ (4) [{...}, {...}, {...}, {...}]
{author: 'patrick rothfuss', title:
"The wise man"}
'LHhPBvdwn4k6THW8Uypb'
```

BOOM! new snapshot sent

Cloud Firestore

Data | Rules | Indexes | Usage

Field		Type	Value
title	=	String	the wise man

Update the field of the document with a subscription and if you save, ~~also~~ the console should print update

UPDATING DOCUMENTS #10

Next, we want to now learn how to update properties on a document using - updateDoc()

index.html ≡ Add Update form

```
...
<form class = "update">                                  # Update Form
  <label for = "id">Document id: </label>
  <input type = "text" name = "id" required>             - id

  <button> Update a book </button>
</form>
...
```

index.js ≡ Implement doc Update

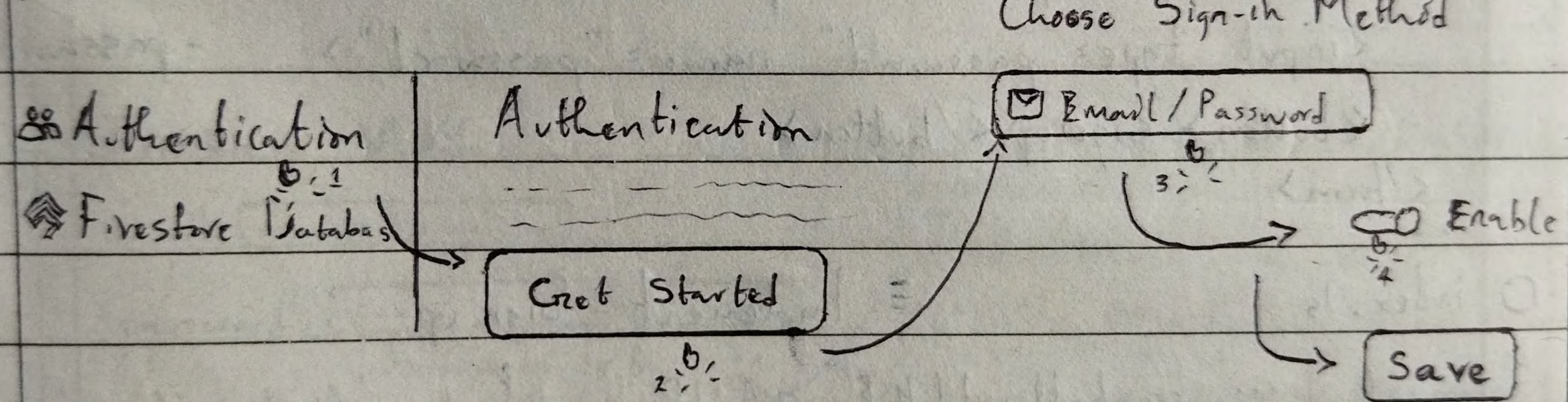
```
import {..., updateDoc} from 'firebase/firestore';

// updating a document
const updateForm = document.querySelector('.update');
updateForm.addEventListener('submit', (e) => {
    e.preventDefault();

const docRef = doc(db, 'books', updateForm.id.value);

updateDoc(docRef, {
    title: 'updated title'
})
.then(() => { updateForm.reset(); });
})
```

Assignment: Update Any 2 docs of your choice

Firebase Auth Setup #11

We want to now learn how to use firebase auth in version 9. Firebase Auth uses a JSON web token to authenticate users who might signup, login & logout of our applications.

When a user is logged in, this web token is sent to the firebase server on every request, - so that firebase can authenticate every request and only have ~~some~~ authenticated users have access to certain features of our site.



SPIN-OFF
NOTE: To Add more Sign-in providers click [Add New Provider]

Now we've our Email/Password provider enabled, we need to initiallize this authentication service on the frontend (Just like we did with firebase app, & firestore db)

≡ Initiallize Auth Service

0 index.js

```
+3 import { getAuth } from 'firebase/auth';


   // init services
   const db = ...;
+  const auth = getAuth();
```

// Authentication Service Initiallised

Signing Users Up #12

Let's now learn how to sign a new user up

index.html ≡ Add Sign up Boilerplate Form

```
...
<h2> Firebase Auth </h2>
<form class="signup">                                  # Sign up Form
  <label for="email"> Email: </label>
  <input type="email" name="email">                    -email
  <label for="password"> Password: </label>
  <input type="password" name="password">              -password
  <button> signup </button>
</form>
...
```

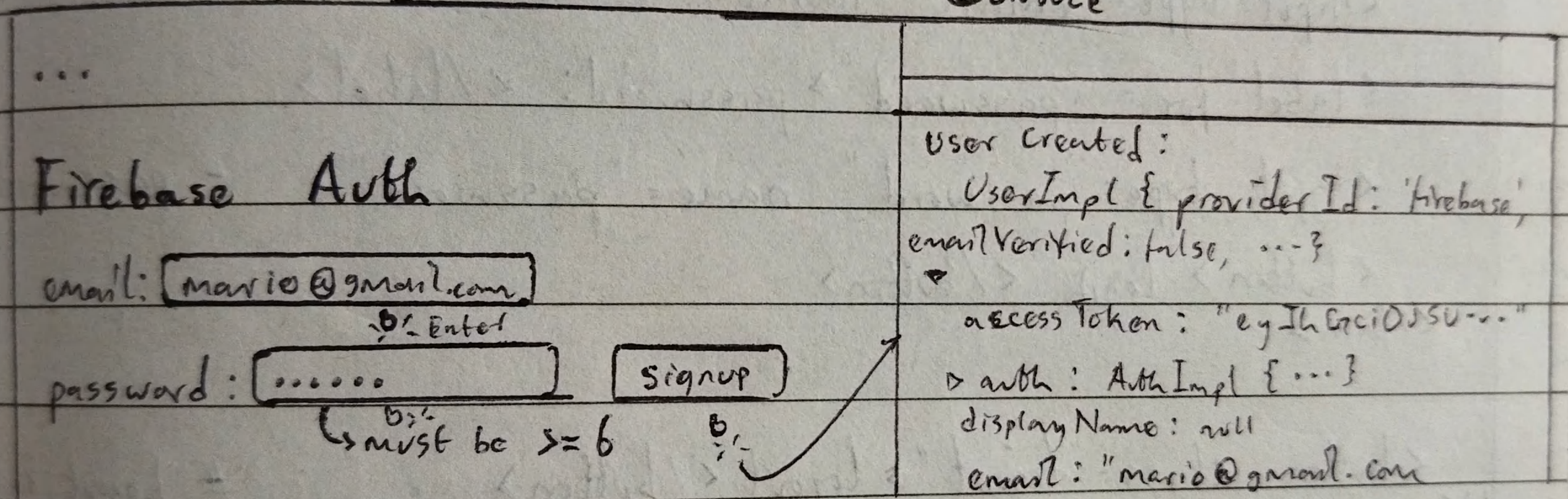
index.js ≡ Implement Sign up

```
3 import {..., createUserWithEmailAndPassword} from 'firebase/auth';


// signing users up
+ const signupForm = document.querySelector('.signup');
+ signupForm.addEventListener('submit', (e) => {
    e.preventDefault();

    const email = signupForm.email.value;
    const password = signupForm.password.value;
```

```
createUserWithEmailAndPassword(auth, email, password)
  .then((cred) => {                    # Returns user credential
    console.log('user created:', cred.user);
    signupForm.reset();
  })
  .catch((err) => { console.log(err.message); });
})
```

~~Console~~

④ Browser



Logging In & Logging Out #13

We want to now see how users can be logged out & logged in.
We will first look at logging out because the createUserWithEmailAndPassword() function already logged us in

☐ index.html ≡ Add Boilerplate Login Form & Logout

```
...
<form class="login">                                    # Login Form
    <label for="email">email:</label>
    <input type="email" name="email">                   - email
    <label for="password">password:</label>
    <input type="password" name="password">             - password
    <button>login</button>
</form>

<button class="logout">logout</button>                  # Logout Button
...
```

☐ index.js ≡ Implement Logout

```
import {..., signOut} from 'firebase/auth';

// logging out
const logoutButton = document.querySelector('.logout');
logoutButton.addEventListener('click', () => {
    signOut(auth)
        .then(() => { console.log('the user signed out'); })
        .catch(err => { console.log(err.message); });
});
```

≡ Implement Loggin.

```
import {..., signInWithEmailAndPassword} from 'firebase/auth';

// loggin In
const loginForm = document.querySelector('.login');
loginForm.addEventListener('submit', (e) => {
  e.preventDefault();

  const email = loginForm.email.value;
  const password = loginForm.password.value;

  signInWithEmailAndPassword(auth, email, password)
    .then((cred) => {
      console.log('user logged in:', cred.user);
      loginForm.reset();
    })
    .catch ((err) => { console.log(err.message); });

    . console.

});
```

Assigment: Try Logging out
Log in with fake credentials (not signed up)
Login with proper credentials (signed up

Listening To Auth Changes # 14

We want to now setup a subscription that listens to Auth changes - When a user is logged In and when they logout

index.js

```
import {..., onAuthStateChanged } from 'firebase/auth';

// subscribing to auth changes
onAuthStateChanged(auth, (user) => {
    console.log('user status changed:', user);
});
```

This refers to the logged In user or logged out user

Now we

Paying attention to the console

Assignment: Login with proper user Credentials
Logout and pay attention to the Console

Course Completed

Thank You

Thariath