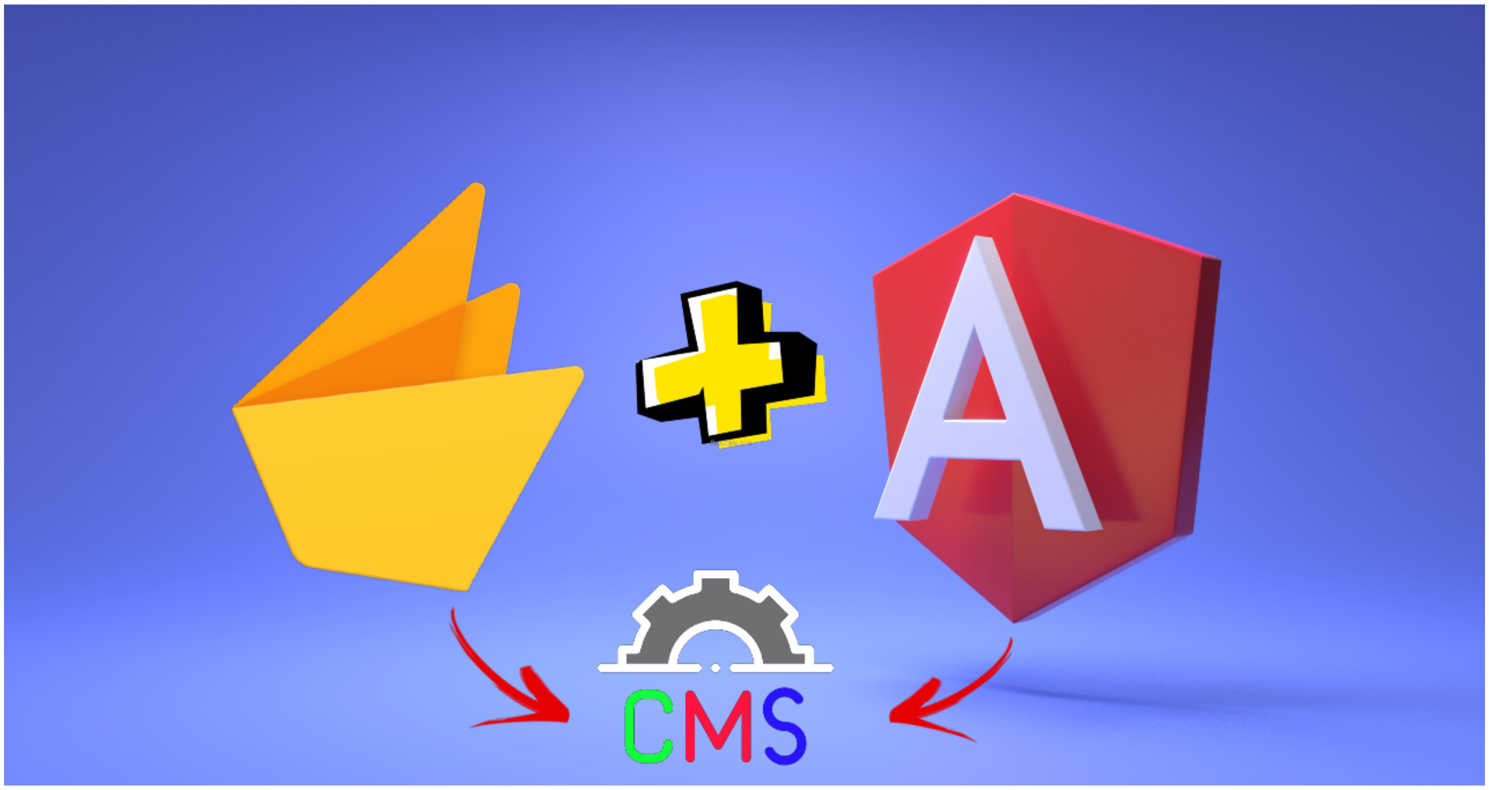
ANGULAR — FIREBASE CMS SERIES — PART 2

Create a CMS with Angular + Firebase — Part 2

Build Angular 11 CMS with Firebase Series

Liraz Shaka Amir Follow

Jan 25, 2021 · 7 min read



Angular + Firebase = CMS

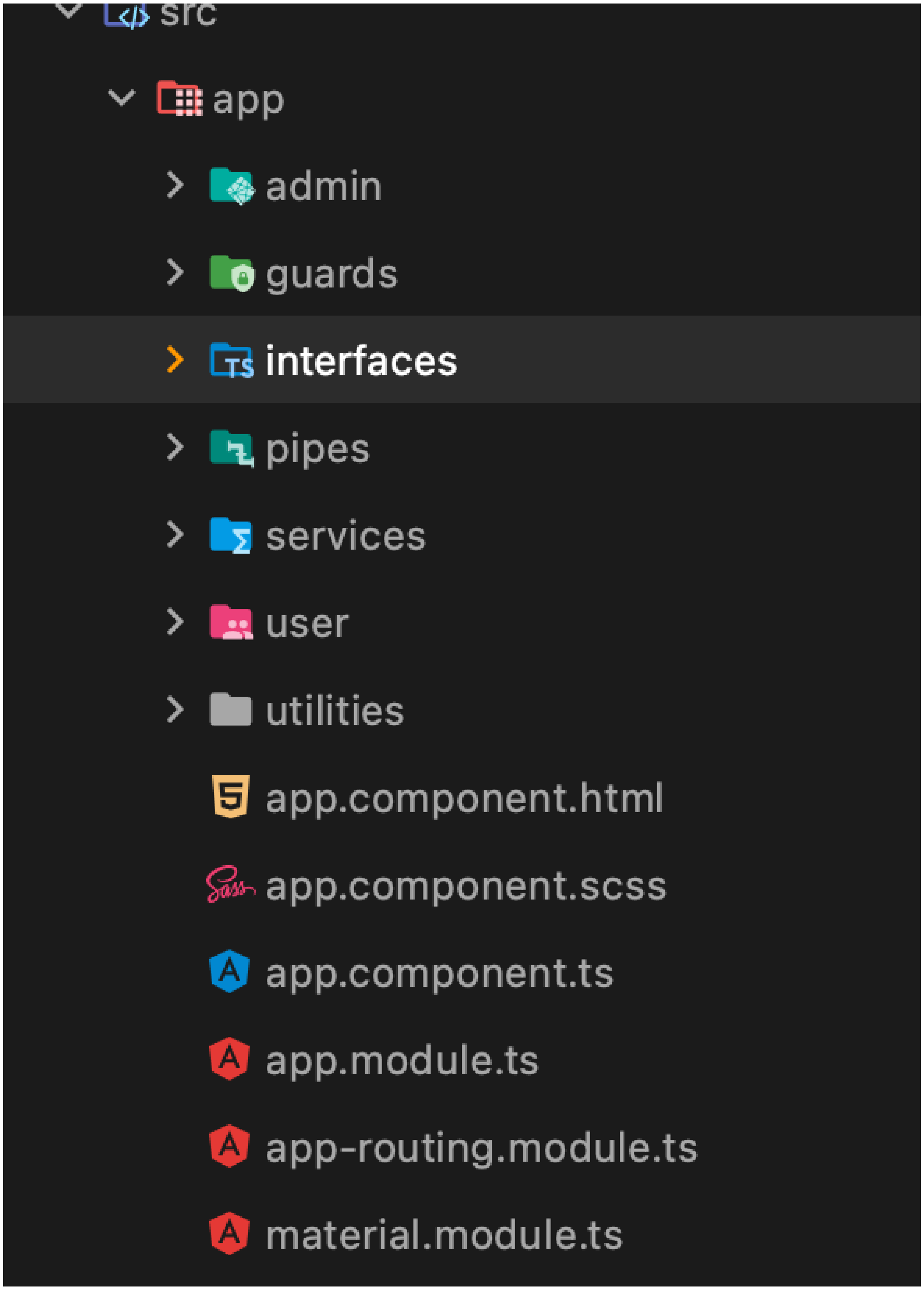
Welcome to the second part of our Angular Cms Tutorial. Here are the links to the other parts of the tutorial (I will add the links after I publish them):

- Part 1

- Part 2 — no need for a link, right? You're already in it!
- Part 3 — will be added soon :)
- Part 4 — will be added soon :)

So, we've set everything up (firebase, database, and project structure). Now it’s time to fill in all of our routing, services, guards, pipes, and utilities.

Our app tree looks like now:


src
app
admin
guards
interfaces
pipes
services
user
utilities
app.component.html
app.component.scss
app.component.ts
app.module.ts
app-routing.module.ts
material.module.ts

app-tree

Let’s start with the services directory.

services directory

The most common part for any application I use is the logging part. Every app I wrote has a ton of console.log commands. The problem is, we don't want the end-user to see logs! So how do we do that? Well, there are two ways of doing that. The short way and the long way.

> *The Short Way*

go the main.ts file and add this:

```
if (environment.production) {
      window.console.log = () => { }
}

platformBrowserDynamic().bootstrapModule(AppModule).then(ref => {
  ...
}).catch(err => console.error(err));
```

this function will overwrite all of the console.log commands and tell them to do nothing if you're in production mode.

I've used it on more than one occasion. But then I decided to do it the long way, a better way, a service that will control all of my logging issues.

> *The Long Way*

Inside the services directory, we create a new service. console-logger.service.ts:

https://gist.github.com/blakazulu/7bf73355bc9c1db281a2aa91d6af4855

Every time we will need to log something, we will use this service. Every function is the same inside. The difference is what we will log, error/warn/log/info. The actual logging will be made if it’s not in production mode, and afterward, a Snackbar (a simple message) from Material will be called. We pass a value and a class name for the snackbar. Remember that main styles.scss file from Part 1?

Every logging function has two parameters passed to it, a value of type any and rest, which is of type any[]. What are those three dots, you say? That’s the spread operator.

What is it, and what does it do?

> The spread syntax “spreads” the array into separate arguments

You can read more about it here. It will take whatever array we are passed inside and spread it out for us.

> *Bonus. you can add a nice function to the class if you want to display objects differently — in a table form. it’s easier to view objects like that. all you have to do is use console.table(object) instead of console.log. try it :). please keep in mind though that* `console.table()` *is not supported in IE — but who uses IE now days.*

You can see it implements an interface called Logger. We’ll get to all of the interfaces later.

Now we will create our second service. When you start making calls to an outside source, you can either create a single service that will make all the actual calls(which can end with a pretty big file) or separate them. When I started, I wanted to use both methods, as you will see.

The first service we will create is the angular-firebase.service.ts:

https://gist.github.com/blakazulu/a03b0957391fe2efa10abb4ec7277d1b

The first thing you need to understand about firebase is that everything is streamed. This means that whenever something changes, we are aware of it. Why? Observables, that’s why.

When the service is first created, it checks if the user exists; if not, we call the authentication method to get the user. We then take the user info from the user's collection using the user id and use the valueChanges() function, which listens to updates from the document (see Part 1 about how firebase stores its data). We pass that data to our user Observable so we can use it anywhere in our app.

The first function is the Login with google function. After we log in with a google account, we call the updateUser() function, which updates our user data inside the database with data from the google account.

Next is the logout() function, which is pretty simple.

The finale function is getCollection(). We passא the collection name we are looking for(pages or posts), and we call snapshotChanges() function, which creates a stream of synchronized changes. Like I said earlier, in Firebase — everything is streamed. Firebase returns all of the firebase documents, which we then map to a single collection and return an Observable object with the id and the spread data.

Now I know it's not easy to understand, but it will be a lot easier after we finish implementing it all.

We will create our third service pages.service.ts — this service will be in charge of our firebase page functions. This is a part of my combined approach to outgoing calls:

https://gist.github.com/blakazulu/754d8fe8336688292aacb621949bf8f5

When the service is created for the first time(singleton service :)) we call the getPages() function, which calls the angular Firebase Service and calls the getCollection() method and saves all pages to our pages Observable.

The first function is the getPage() function which takes 3 parameters. a field path, a condition, and the value. This function is responsible for getting a specific page from Firebase. To get the page, we pass a Query Function to the store, which uses the where() function from the AngularFirestore we imported in the constructor. The where() function takes in three parameters, the path to search, what condition, and what value. We then return an Observable with the data.

The Next functions are the addPage(), deletePage(), and updatePage() functions. They are pretty simple. inside the addPage() we create a new page. We call the AngularFirestore collection() and add/delete/update the page. We then log everything. Notice that we keep a current Timestamp on everything.

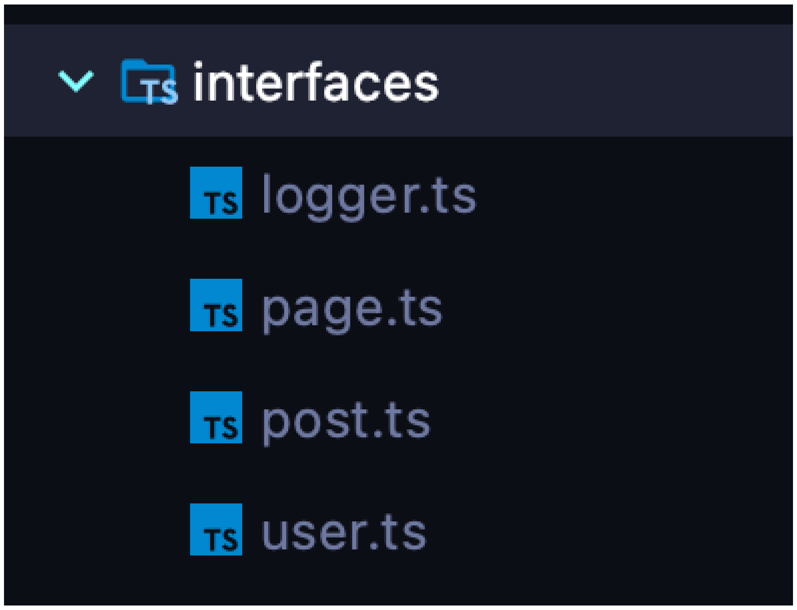
You can find all of the documentation to the Angular Firestore here.

Now we will create our fourth service posts.service.ts — this service will be in charge of our firebase post functions:

https://gist.github.com/blakazulu/cbf9270dd95ad479f0b01e35c0b59dcc

All of the methods here are identical to the pages.service.ts except for the data types. You can combine them into one service or keep that like this.

That’s it; we're finished with the services. Let’s continue with the interfaces directory.


interfaces
logger.ts
page.ts
post.ts
user.ts

Interfaces directory.

The first one is our logget.ts, which is used by our logger service:

https://gist.github.com/blakazulu/277fcf7572a44f037a527da4ab15680b

Next up is our page.ts, which is our page model:

https://gist.github.com/blakazulu/1a17126c88ae0b26825f3bc390c434f1

The third one is our post.ts model:

https://gist.github.com/blakazulu/c1fafc5d85200b0d8b87ba27ab03ade7

And the last one is our user.ts, which has both the user model and the roles model because they are connected logically:

https://gist.github.com/blakazulu/7fa49b91d7b9fcd4f1773d128843c078

Now we are moving to our custom pipes. If you already know how to build custom pipes, you can copy the code and move to the next section. if not — keep reading :).

So what are pipes? Pipes are a useful feature in Angular. They are a simple way to transform values in an Angular template. There are some built-in pipes (like the async pipe), but you can also build your pipes. You can read more about it here.

Our first pipe will be the safe.pipe.ts:

https://gist.github.com/blakazulu/7d75721b3859a30aeb90db2c66a87abb

What is this safe pipe? Well, this will help sanitize HTML code. This means that you can write HTML syntax inside an input; for example, it will display the resulting HTML and not the code itself.

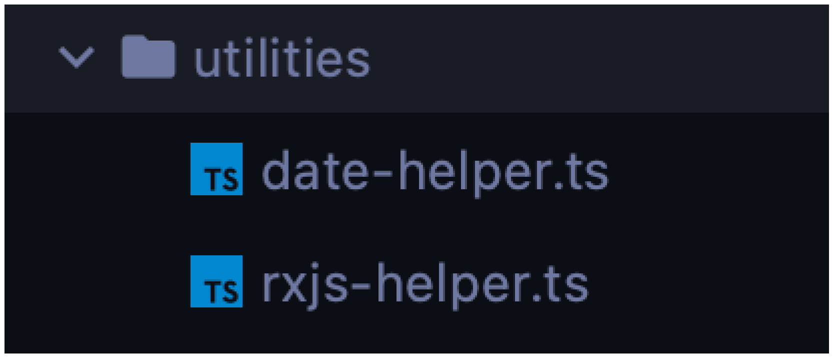
Our second pipe is the date-format.pipe.ts:

https://gist.github.com/blakazulu/e981d1cee9283398db296a2b584b52e8

Now I know some of you are saying — Shaka, what are you doing? a date pipe already exists in angular. I know, your right. But I need a specific format that doesn't exist.

This pipe takes in a date in the form of a number and converts it to a UTC date string.

That’s it; we’re finished with the pipes. Let’s continue with our utility directory.



utilities directory

The first file is our date-helper.ts:

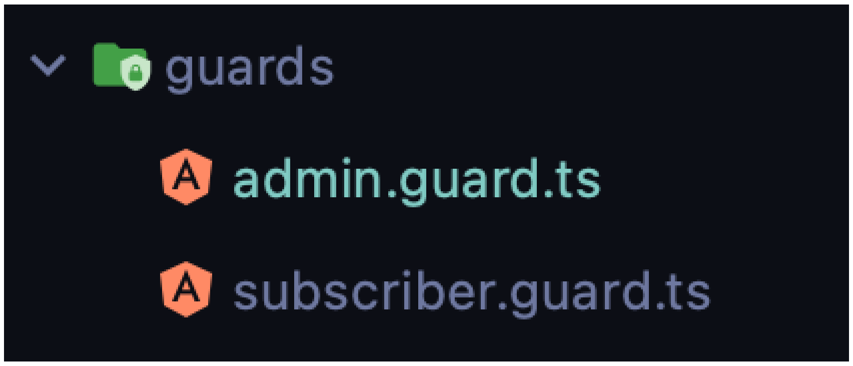
https://gist.github.com/blakazulu/e2d2875c68162dc8aa8e0e3b05df70fd

The second file is the rxjs-helper.ts:

https://gist.github.com/blakazulu/3a0159cda09bfc44ffe464883229089d

This file has one purpose, to destroy our subjects and make sure we don’t have a memory leak.

Now for the last section of the Part 2 tutorial, the guard's directory.



guards directory

Why do we need guards? Like I said in Part 1 of the tutorial, we need to ensure that anyone who “**accidentally**” goes somewhere he shouldn’t; the guard checks if that person should be there.

The first guard is admin.guard.ts:

https://gist.github.com/blakazulu/046211e144443b2f30f10ce7b9d7fcd7

This is how this works. If the user goes to the admin path, it activates the AdminGuard. Why? because that’s how we configured the app-routing.module.ts file:

app-routing.module.ts

Now the admin.guard.ts kicks in and checks if the user has an admin role. If it does not have it, it sends him home.

The second file is the subscriber.guard.ts:

https://gist.github.com/blakazulu/19ac485118678502496e0b8218273bde

This is the same. The only difference is it checks if the user is a subscriber.

And that’s it. We are done with the second part of our Angular CMS tutorial.

Be sure to check out the next part (which will be available next week), where we will work on the admin side of things.

Here are the links to the other parts of the tutorial (I will add the links after I publish them):

- Part 1
- Part 2 — no need for a link, right? You’re already in it!
- Part 3 — will be added soon :)
- Part 4 — will be added soon :)

Until next time!

Photo by Reuben Juarez on Unsplash

Angular CMS Firebase Firestore Programming

Download on the App Store

GET IT ON Google Play