

# API

Start point in scripts is made to get rid of typing in the main entry point every time we start

Mongoose : object document mapper to connect with mongodb

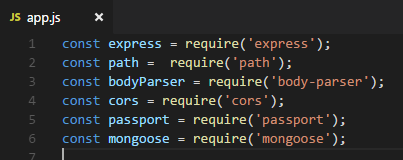
Cors : to make request from frontend even if its on a different port

Passport-jwt : allow us to create a login strategy that uses web tokens

## Main Server entry point

Bring in all the modules:

Path – we didn’t install using dependencies because it is a core dependency

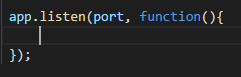


## Initialise variables

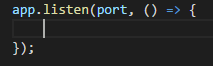


## Listening and Callback function

Can do function like

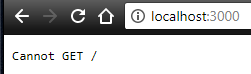


But also can use arrow function like



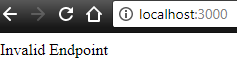
Use **npm** **start** for starting server or use **nodemon** which is easier

Now if you go into localhost:3000



* That is because there is no route to the home page.

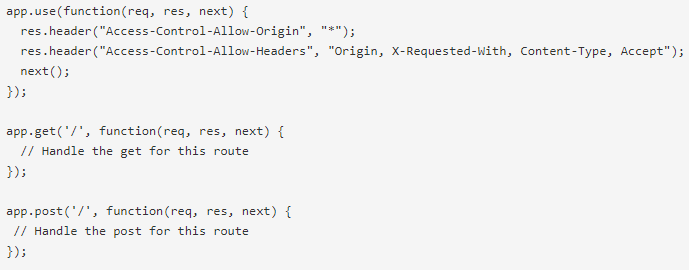
## Put a route

 🡪

## Cors middleware

Allow us to make a request to out API from a different domain name.

If you don’t wanna use cors, then make our own middleware by:



Or use cors which is much easier 🡪 <https://www.npmjs.com/package/cors>

 makes it accessible from any domain.

## Body Parser

Parses incoming request bodies



## Router

Gonna use express router. Gonna put it in a separate file. Include that file too:

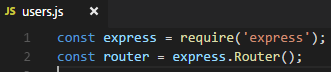


And then pass in that users variable 🡪 

It’ll make every calls like <http://localhost:3000/users/whatever> go to that users file

### Router file : Users

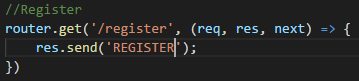
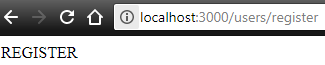
import express, bring in the router from express



Export it too 🡪 

#### Requests

Instead of **app.get** , now its **router.get**. Here, requests works after /users/

🡪 

The following route is to be protected with token

## Connect with Client Side app – Angular 2

Set static folder

Pass in express.static and in there, we use path module so that we can join current directory and alias

Here, the **current** **directory** is joined to the alias **public**.



After that,

an index.html is created inside public folder. We can see that <http://localhost:3000/> is loading that index.html – I’m deleting it as it is not needed right now

## Mongoose

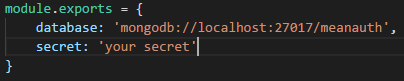
### Connect Mongoose to DB

gotta call the connect function from mongoose to connect to database: .

Instead of putting the location of db right there, let’s create a config file and put the db in that:

Inside config folder, database.js is created

Set that to an object – give in dba and token and require it in app:

 🡪 

And then give it to the mongoose connect 🡪 

This will let us know if we are connected : 

### Models and Register request

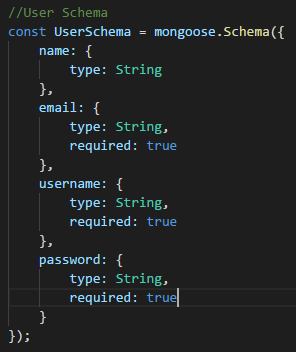
Created a folder called models and added a file user.js

Require mongoose, bcrypt for encryption, and the config file



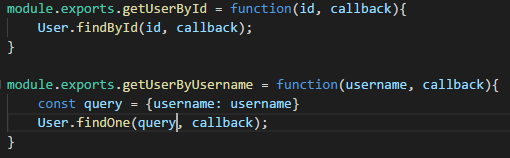
Bcrypt is used to encrypt the password

Create the schema in it :



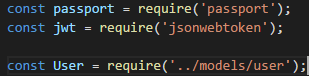
And export the model 

Make sure the created functions are exported as well because they are to be used outside:



Now, in the route user:

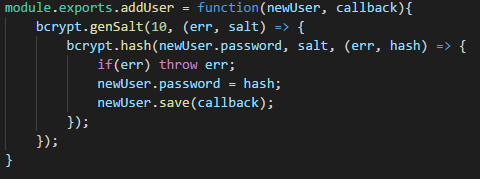
Bring in the model, passport, the web token module jwt:

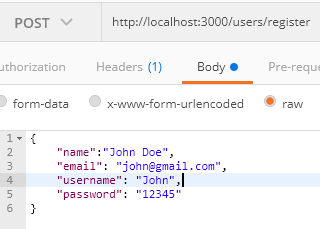


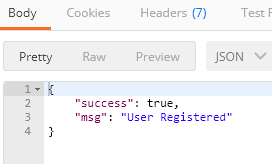
Change register call such that:



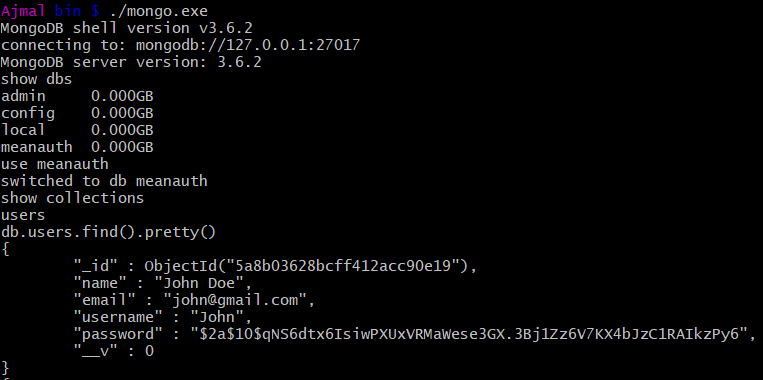
Also, add a function to add user in model:



Now a request   
  
will return



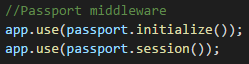
Check your database now :



## Token system : passport and authentication

Ref : <https://www.npmjs.com/package/passport-jwt>

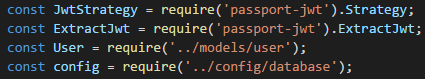
Add middleware for passport in app.js:



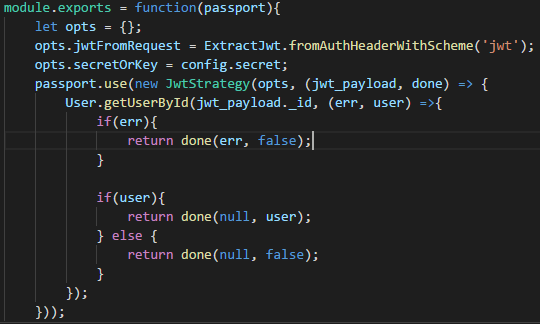
### Token generation

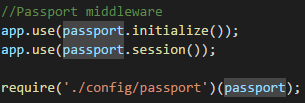
Add a config file for passport in config folder :

Add the required imports :

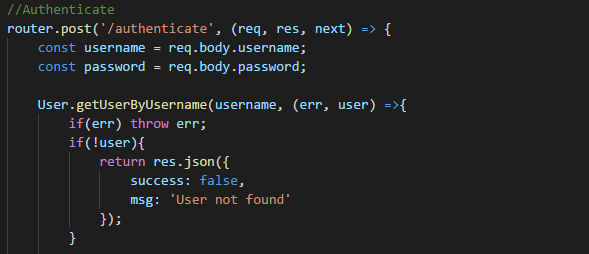


Create the function for jwt

  
We can see that passport is passed in as a parameter above. So, we’re gonna do the following in app.js : notice the require:

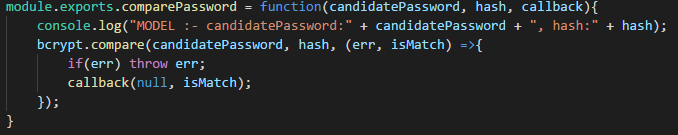


Now, in our routes, edit the authenticate :

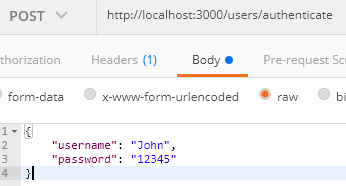


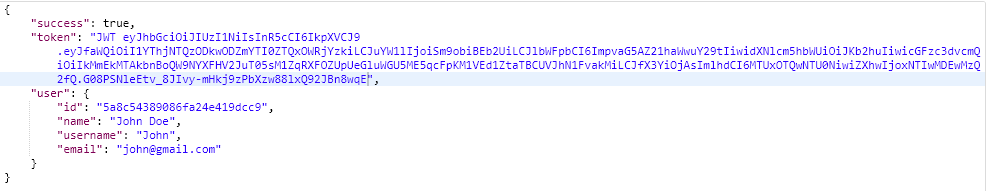
The “JWT ” with the whitespace in token is mandatory because passport needs that thing to recognise the token.

Now, we need to create the comparePassword in the models:



Now, try out the request :

  
and we’ll get



Voila !

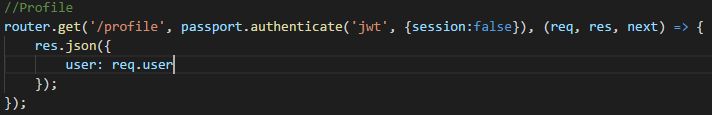
### Using the token in a route

As we now have the token , let’s protect a route :

We’re gonna do it on profile route

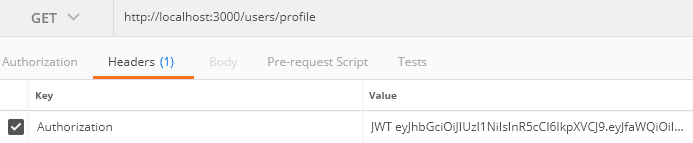
Add passport.authenticate function as a parameter to the request

We’re not using session, so, session:false …..then, return the user

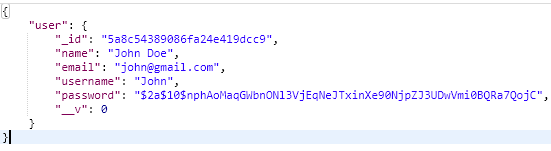


Now, a normal request   
   
will get a response

Let’s add the token generated from authenticate to the profile request’s header



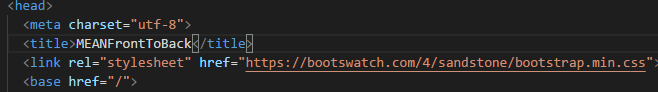
Now, the response is :



# Client - Angular 2

## Bootswatch

We can get custom bootstrap themes from <https://bootswatch.com/>



Added the theme css

Note that out base url is “/”

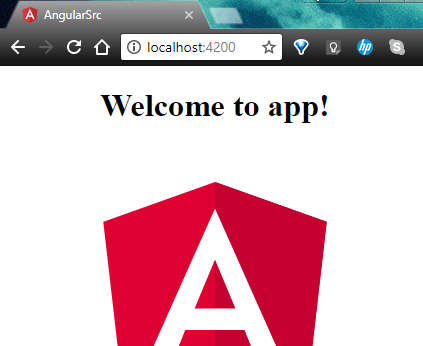
## Generate the angular app

If you want to install the angular cli, use **npm install –g angular-cli**.   
We are creating the angular app in the root folder 

Now, to change the output directory : open **.angular-cli.json** file and change the **outDir** : I’m setting it as 

Run the server for the backend – keep it running – it’s on port 3000

Now cd into the **angular-src** folder and start the dev server using **ng serve**. It’ll run on port 4200 by default :



## App.module.ts

We gotta import all the components and services into this file

**Decalrations** : custom components

**Imports**: modules

**Providers** : service

## Components

We gotta break up our app into components. We create new components in a folder called **components** in the **app** folder.

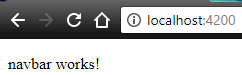
Cd into the **components** folder and generate a new component:

* Navbar
* Login
* Register
* Home
* Dashboard
* profile

### Navbar component

**ng g component navbar –** the relevant files are created inside a new folder navbar

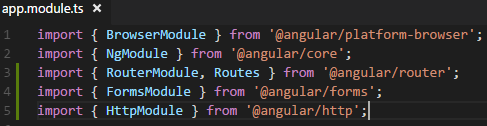
open **app.component.html** and insert the navbar’s selector into it

 🡪 

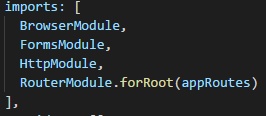
Router

App.module.ts --> import :

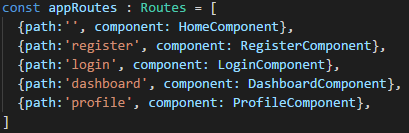
* router module and routes
* FormsModule
* HttpModule



Now, include it in imports:

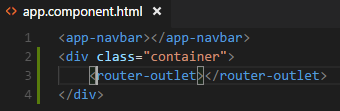


For Routermodule – pass in appRoutes to forRoot – and create that appRoutes:

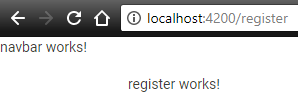


We have to add router-outlets

Open app.component.html and add router-outlet to it :



Now, the routing works :

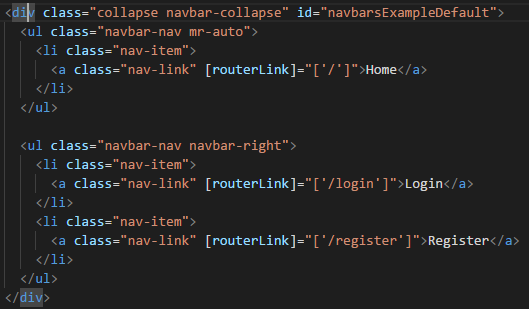


#### Creating the Navbar - HTML

<https://getbootstrap.com/> : I’m taking the starter template from it – Ctr + u to see the source code. And put it in the navbar html file.

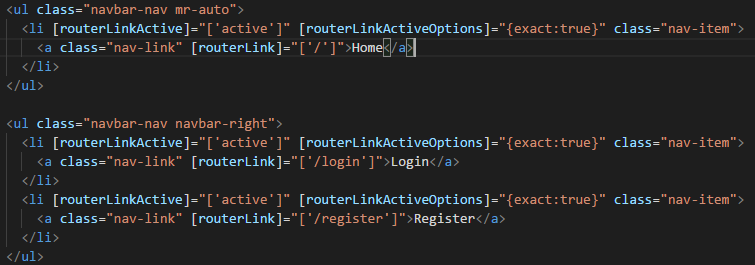
Now, the navbar shows up everywhere

We don’t use hrefs for routing in the html – instead use 

* 
* 

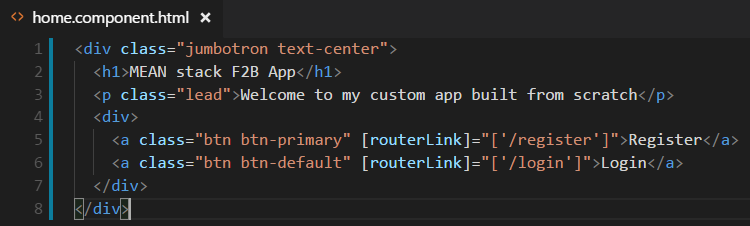
Use [routerLinkActive]="['active']" To toggle the active –use it in the li

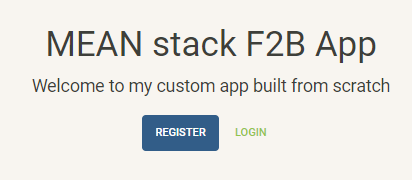
Also use [routerLinkActiveOptions]="{exact:true}"



### Home Component

#### Home HTML

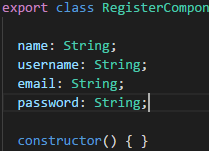




### Register Component

#### register.component.ts

Added 4 properties to registercomponent



onRegisterSubmit function creates a user object and populate the values from the form

#### Html

Each form elements have ng model to bind it to the properties [(ngModel)]=”bla” and name=”bla”

Also, add an event to the form . Type of event is a submit which calls a function – create that function to be called in the component file

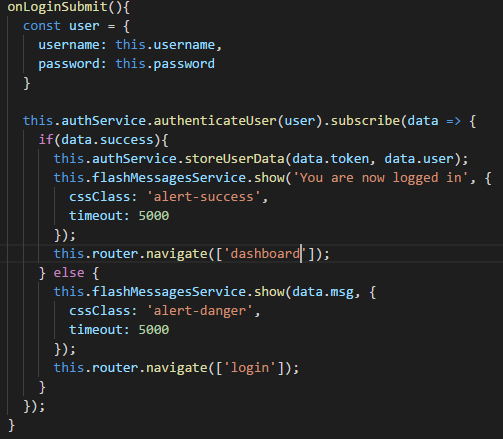
### Login Component

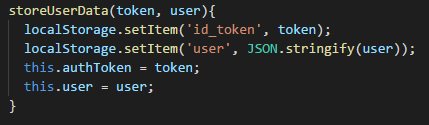
Make the form in the html file and add submit event in the form.

In the Component file, add the Router, FlashMessageService and AuthService.

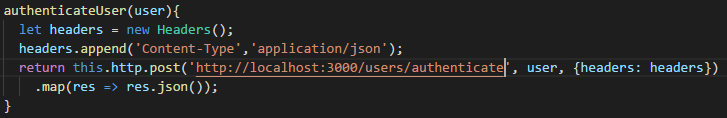
Create a function called onLoginSubmit() 🡪 inside it, create an object with the username and password, pass it to the authenticateUser function called from the AuthService.

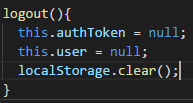
It will give back an observable, So, subscribe to it. ---- inside --- login success will need a store user data

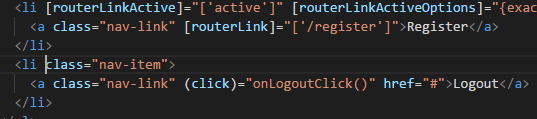
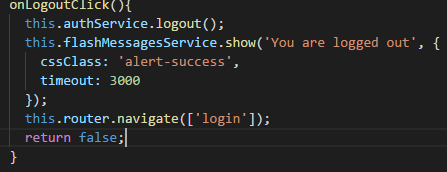


And the storeuserdata function is  


Localstorage only stores string

Then, create the authenticateUser finction inside the AuthService   


Now, creating a logout in the AuthService  


Create the logout html piece – get rid of the the routerlink and use the click event to do a function  
  
  
the onLogoutClick function in NavbarComponent would be  


## Services

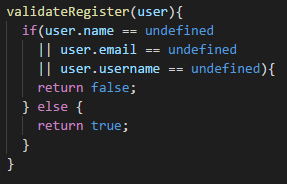
Create a folder called services in the app folder

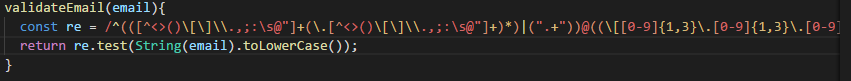
### Validate Service

**Ng g service validate** to create validate service

Import ValidateService to app.module.ts, also put that in the providers

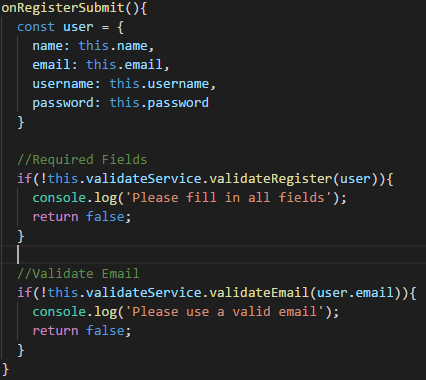
 

Goto ValidateService and create a validateRegister function 

Then a validateEmail 

Import the validate service in the register component 

Inject it into the constructor anytime you use a service in a component 

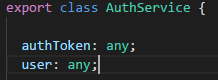
Create a function in the component …. onRegister Submit 

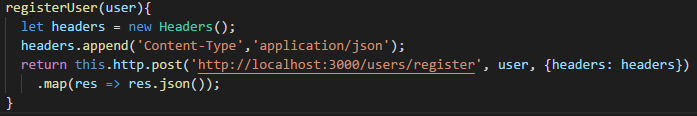
### Auth Service

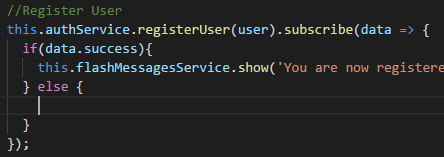
Generate the service  
import it to app.module.ts  
add it to providers

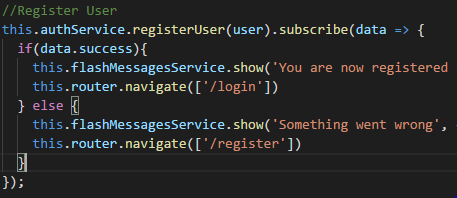
Bring in the http Module into the authservice, http as well as headers package  
Also bring in the map operator   


Inject the Http into the constructor

Declared two variables  


Creating a function to register the user (make a post request to register)  


Import this service to RegisterComponent and use it there as :  
[As it is an observable, subscribe to it 

Now, import Router to reroute -- inject to constructor and use 

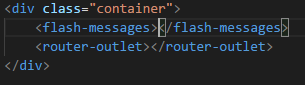
## Angular2 flash messages

In angular-src, install angular flash messages

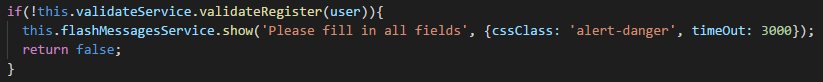
npm install angular2-flash-messages –save

import it to app.module.ts 

add that to imports   


Need an outlet for this. So, go to the main app html file and 

Import FlashMessagesService in the RegisterComponent and inject it into the constructor 

Use it like 

## AuthGuard

Npm install angular2-jwt –save