MEAN Stack Guide

A diagram of a software program

Description automatically generated with medium confidence

A diagram of a course

Description automatically generated

A screenshot of a web page

Description automatically generated

Event binding:

(click)=”onAddPost()”

String Interpolation:

Use double brackets to show on html {{enteredValue}}

Property binding :

Bind property to value property and access it using # refrence and sendit to component like below

<textarea rows=”6” [value]=”newPost” #newPost>

<button (click)=”onAddPost(newPost)”>

Two way binding:

Or above can be avoided by using two way binding using

[(ngModel)] = “enteredValue”

Lets use Angular Material for designing components in Angular , run the command

Npm install –save @angular/material

TO iterate posts in loop and show them in UI , Angular ships with Structural Directives, Drectives are instructions yu put on elemtn. Some apply ot sigle elemtn , some change structure of rendered html code . We use \*ngFor, \*ngIf

Adding Form:

<form (submit)="onAddPost()" #postForm="ngForm">

Here Angular takes care of default submitting to server. Add reference to the form as #postForm and assign it to ngForm , which is a directive which angular attaches to form element and gives access to form object created by angular and managed by angular .

And on component file , we have form and in that we have value which has name as title, you can use it

const post = {

title: form.value.title,

content: form.value.content

}

And we can add default HTML 5 validators on the form to validate input data

<textarea matInput rows="4" name ="content" ngModel required #anyname = "ngModel"></textarea>

Here to get access to form element within html , use reference like #anyname and assign ngModel so you get access to the input element

For addin service you can ad it in app module ts in providers: [PostSerivce ] or

Add in service with @Injectable({providedIn:’root’})

@Injectable({providedIn:'root'})

Two ways of connecting Node+Angular:

Ng serve behind the scene uses node and is only development servers and doesn’t provide any features like app entry to add server logic etc.

Node App Server Angualr SPA o rHAve Two Seperated Servers with static host,

In first case node express hadles incoming erequests .we have endpointsf for this , we have special endpoint sto request targeting / path return angular SPA.

In second case we have node express handling incoming requewts with Angulra SPA served from separate static host.

In both cases we have Logically seperateed apps .

RSETFUL lapis are Stateless Backends. They are server side solution . You can request from mobile app , clinet app or someother app using REST API to same server. We use CRUD .

We can send AJAX request thorugh angular http client for exmpale to send post users to backend and get response you can use in client using JSON data format used fo rexchangong data. You can have XML , FORMdata , URLendoded etc.

Project ,

You can use separate folder or can use same folder for backend logic so running can be easy .

Create backend folder in same folder and create server.js file for server logic in root folder like below

const http = require('http');

const app = require('./backend/app')

const server = http.createServer((req,res)=>{

res.end('THis is my first response')

});

server.listen(process.env.PORT | 3000);

Now we add app.js file tfor express middleware in backend folder

const express = require('express');

const app = express();//Express is like bigchain of miidlewear , or a funnel for middleware

app.use((req, res, next)=>{

console.log('First middleware ')

next();

});

app.use((req, res, next)=>{

console.log('Second ')

res.send('Hello from express');

});

module.exports = app//exposrt all middlweawre by using app, we have epxress here and node there so we export it so node can listen and

Now we join both by below

const http = require('http');

const app = require('./backend/app')

// const server = http.createServer((req,res)=>{

// res.end('THis is my first response')

// });

const port = process.env.PORT | 3000;

app.set('port', port);//tell express which port you are working

const server = http.createServer(app);//we need epxress middlweare to listen to the requetws so insteae of above we use here like this

server.listen(port);

Copy paste server.js with fie in reosurces added in liceture

To automatilcayy restart server on changes ni backedn ,we install nodemon by

Npm install –save-dev nodemon

In package.json , add start:server

"scripts": {

"ng": "ng",

"start": "ng serve",

"build": "ng build",

"watch": "ng build --watch --configuration development",

"test": "ng test",

"start:server":"nodemon server.js"

},

And to run server use

Npm run start:server

Now use httpCLinet in angular to fetch the posts array to UI in angular posts.service.ts

If you get cors Cross Origin Resource Sharing error in console when running angular This comes when there are two servers on two hosts on frontend and one backend.

TO let other server access requests onto this backend server add this in express or app.js file

app.use((req,res,next)=>{

res.setHeader('Access-Control-Allow-Origin', "\*");

res.setHeader('Access-Control-Allow-Header', "Origin, X-Requested-With , Content-Type , Accept");

res.setHeader('Access-Control-Allow-Methods', "GET, POST , PUT , DELETE, OPTIONS, PATCH")

next();

})

Only first one is needed in the above , rest of headers are optional but good to have.

Now to get request body from posts request in backend , we install a package

Npm install –savw body-parser

In app.js file we use

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({extended:false}));

Connect backend to MongoDB

2 options –

1)by downloading mongodb

2)Free sandbox

In this course he is using Free sandbox Free tier cluster to connect to mongbdb by whitelist ing localhost IP. But I will use exisint mongodb

TO access DB there MondoDB node js driver, but we use Mongoose third party package with official mongoose driver , which makes easy . It uses schemas, this is not requirement if your project has unstructured data but mongoose still works well with unstructured data.

Install monggose package by using comnad

Npm install –save mongoose.

Now Add schema using mongoose in backend. And create post.js in models folder for your model object in js

(Also note that tyosecpript string with lower case s and in nodejs its javascript so String capital s in model objects)

Post.js

const mongoose = require('mongoose')

const postSchema = mongoose.Schema({

title: {type:String, required: true},

content:{type:string , required: true}

});

mongoose.model('Post' , postSchema);

Here P when creating model should be capital leter

The collection you get once you save is plural form of your model name so Posts will be collection name in DB.

He cannot see data in db , he is using mongo shell form within the cluster .

Create async validator to validate file using mime-type.validator.ts

Uplaoding image files on server:

Install multer package to extract incoming files.

Npm install –save multer

Add image to frotne end and bakend model objects

I backend post.js we add below for multer logic

const Post = require("../models/post");

const MIME\_TYPE\_MAP ={

'image/png':'png',

'image/jpeg':'jpg',

'image/jpg': 'jpg'

}

const storage = multer.diskStorage({//let multer whcih folde rmulter needs to save files

destination:(req , file , cb) =>{

const isValid = MIME\_TYPE\_MAP[file.mimetype];

let error = new Error("Invalid mime type");

if(isValid){

error = null;

}

cb(error , "backend/images");//here realtive path to serve.js

},

filename: (req, file, cb) =>{//tell multer what filename shoudl be

const name = file.originalname.toLowerCase().split(' ').join('-');

const ext = MIME\_TYPE\_MAP[file.mimetype];

cb(null, name + '-' + Date.now() + '.' + ext);

}

});

router.post("", multer({storage: storage}).single("image")),(req, res, next) => {//multer finds single file of type image

const url = req.protocol + '://' + req.get("host");//consturcts url to host of server

const post = new Post({

title: req.body.title,

content: req.body.content,

imagePath: url+ "/images/" + req.file.filename;

});

post.save().then(createdPost => {

res.status(201).json({

message: "Post added successfully",

post:{id:createdPost.\_id,title: createdPost.title , content: createdPost.content, imagePath: createdPost.imagePath}

//or

//post:{...createdPost , id:createdPost.\_id} //means extract createePost , overwrtie id with createdPost.\_id

});

});

};

In front end we add FomrData js object to upload files

addPost(title: string, content: string , image: File) {

//const post: Post = { id: null, title: title, content: content };//remvoing bcoz of file upload we need to send form data

const postData = new FormData();//FormData js object lets you cobone text values and blob/file values

postData.append("title", title);//we append data to Formdata

postData.append("content", content);

postData.append("image", image , title);

this.http

.post<{ message: string; postId: string }>(

"http://localhost:3000/api/posts",

postData

)

.subscribe(responseData => {

const id = responseData.postId;

post.id = id;

this.posts.push(post);

this.postsUpdated.next([...this.posts]);

this.router.navigate(["/"])

});

}

Mamke images folder statically accessible in app.js by importing package using path

Cosnt path= require(“path”)

app.use("/images",express.static());

Add pagination and send object and totalcount to UI so matpaginator can handle ffrom there

Authentication :

Add two new routes sing up and login .

For user model for login email and password , use mongoose-unique-validator for password d field by using

Npm install –-save mongoose-unique-validator

And using

userSchema.plugin(uniqueValidator)

No to save the password in raw form in db is bad idea so we use another package called bcrypt

Npm install – save bcrypt.

Understanding SPA Authentication :

Here in SPA signle page application backends API are stateless and decoupled from frontend so you cannot use session to store user details and let them access pages ,or a session ID in a cookie to store and automatically send cookie and session id in future request to server and validate cookie id match session id , so we use json web token (cookie/localstorage) generated on server upon login and store it in angular app in a cookie or local storage , this token will be attached to request and this token cannot be faked.

Now to create json web token use

Npm install –save jsonwebtoken

//now create json web token

const token = jwt.sign({email:user.email , userId: user.\_id }, 'secret\_this\_should\_be\_longer' , {expiresIn:"1h"})//you can use any to geenraete signhere we user email userid

You use bcrypt to compare hash of user entered password and stored db password

return bcrypt.compare( req.body.password, user.password)

if no error then you sign the jwt token to be sent to frontend

//now create json web token

const token = jwt.sign({email:fetchedUser.email , userId: fetchedUser.\_id }, 'secret\_this\_should\_be\_longer' , {expiresIn:"1h"})//you can use any to geenraete signhere we user email userid

res.status(200).json({

token:token

})

Now create a middleware folder in backend to parse the request and see if its authenticated , just a function to parse the request and check if tis allowed to continue or not .Create check-atuh.js file in that folder.

Now attach this token on frontend and store it during login and attach it to requests you send to backend. In login method of auth.service.ts

login(email:string , password:string){

const authData: AuthData = {email :email , password : password}

this.http.post<{token:string}>("http://localhost:3000/api/user/login" , authData)

.subscribe(response =>{

const token = response.token;

this.token = token;

});

}

getToken(){

return this.token;

}

Inject this in post.service.ts to add token to requests you send to backend by adding it to headers to http requests . You can use interceptors or function that run on any outgoing http requests. Interceptor are provided by angular already

Create a auth-interceptor.ts file

@Injectable()

export class AuthInterceptor implements HttpInterceptor{

constructor(private authService:AuthService){}

intercept(req:HttpRequest<any> , next:HttpHandler){

const authToken = this.authService.getToken();

const authRequest = req.clone({//add header to exisitng headers

headers: req.headers.set('Authorization',"Bearer " + authToken)//its authorization in middleware caps doesnt matter

})

return next.handle(authRequest);

}

}

Now in header component for header , if user is authenticated then show logout button or show longin and singup links.

Now even you are using token , you can edit posts by visiting and typing url like localhost:4200/posts or /create . TO avoid this we use route guards. Create auth.guard.ts

@Injectable()

export class AuthGuard implements CanActivate{

constructor(private authService:AuthService , private router :Router){}

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean | UrlTree | Observable<boolean | UrlTree> | Promise<boolean | UrlTree> {

//true then guard allwos it or flase then guard blocks it

const isAuth = this.authService.getIsAuth();

if(!isAuth){

this.router.navigate(['/login'])

}

return isAuth;

}

}

And in app routing module ts

providers:[AuthGuard]

{ path: 'create', component: PostCreateComponent , canActivate:[AuthGuard]},

{ path: 'edit/:postId', component: PostCreateComponent , canActivate:[AuthGuard]},

Saving the token in local storage of browser

So you don’t loose token whenever page is refreshed. Accessible through js and vulnerable to CSS stack but Angular handles it and we cant output script tags using angular

private saveAuthData(token:string , expirationDate:Date){

localStorage.setItem('token', token);

localStorage.setItem('expiration', expirationDate.toISOString())

}

private clearAuthData(){

localStorage.removeItem("token");

localStorage.removeItem("expiration")

}

Call these when logging in with new Date in date format since time we cannot say what and when time if date is there then it will be much easier

Auhtorization:Users can only edit or delete posts that they created.

Add new field creator in post model in backend.

We do not enter id on from but inferred by token reaching our backend.

Hanlding Errors:

Add handler to observables to handle errors.

In auth.serviec.ts you don’t see error even if you log there but you need to expect it in sugnup component, you don’t need to subscirne in service so return it , and subscribe in signup component ts .But removed it because we need to do other things in service itseld , ignore this approach now .

Use interceptor to habdle error to show http div errors

Optimization:

User contorllers for rqeusts methods instead of user.js in routes folder.

Create contorllers folder in backend ans keep routes in routes folder

Create file.js for mulert logic , create a new file.js in middleware folder

Create angular-material.ts to include mat related stud and import them into app module ts , this is heklpful in case of big apps

Create post module ts to include decleration post list and post create compoentnts and import reactivreformsmodule

Create auth module ts.

For post module we are importing but not exporting because Router takes care of it and is gloablaly managed so no need to write exports in that module even post create and post list componenet is used everywhere else.

Lazy laoding :

In router we configure lazy loading.

For this creat auth-routing.module.ts. WE get those above routes of post and put it there and add it in imports for authmodule ts and import this in app-routing.module.ts file.

Use url from environment file for frontend and for backend use node env variables for constants by using nodemon.json, all providers for node support it

Deploying:

Two options

Deploy two separated apps

Or

Deploy one combined app, has a route to render angular app , but not In this project

Angular needs static host only serve html , css js. , like AWS S3 , Firebase hosting and CORS Headers are absolutely required .

Node needs node.js host like AWS EC2 /EB, Heroku.

In combined App approach Nodejs host like AWS EC2/EB , Heroku , CORS headers are not required.

For ease moved server.js from root to backend folder and updated path in package.json for start:server in scripts.

Now copy package.json and make a new copy into backend folder since backend needs packages, remove scripts as its not needed as providers check for packages dependencies when deploying.Remove angular related dependencies in backend packages json.

To deploy lets go to AWS Elastic Beanstalk

Create application with presets choose node js as platform , got to abckend folder and zip the contents inside and upload it to EBS .

You need to switch to cloud solution if you are using local db .

Bcrypt may not run in EBS so install bcryptjs

Npm install –save bcryptjs

Replace bcrypt with bcyrptjs in node backend folder from packages json in root folder and zip the backend oflder and upload it again in EBS.

Sepcify EBS to run node server.js as command instead of defafutl in configuration

For frontend , we need to compile angular forntend end files into one htm,l flie and bunchof javasciprt files using angular cli using

ng build --prod

use env.prod file to load beanstalk dashboarrd url

use above command to create dist folder

After build you will see dist folder, it contiants files for static host

We use AWS S3 for static cloud /static host

Create new bucket.

Drag and drop dist folder files in Upload section of s3.

Give permissions in Bucket policy for user and give read only access

And enable static website hosting and type index.html as Index document and Error document, i Error document index.htnk forwards to proper component and angular router understand it .

You can visit app by clicking on s3 endopoint.

You canot upldaod images go to file.js and in multer diskcsotrrage function just give path as “images” and in app.js remove backend/images to just images.

Recomporess backend and deploy on EBS console.

For combined single app deploy approach:

Go to angular.json and change outputPath as “backend/angular”, and run ng build –prod command , this puts angular as sub folder in backend folder.

Angualr folder contains dist folder.

In app.js of backend foldr configure routes, it should be unique and let all other requests to reach angular folder other than post and user routes

App.use((req, res, next)=>{

Res.sendFile(path.join(\_\_dirname , “angular” , “index.html”))

}

You can test this by going to localhost:3000 in local

If any issue allow static accesss to angular folder

app.use("/", express.static(path.join(\_\_dirname, "angular ")));

Now if you reload page you can see th app

Now you can see the app

No build this app , and go to EBs and upload version and deploy