Contenido

[Creación del proyecto 2](#_Toc42960511)

[FRONTEND ANGULAR 5](#_Toc42960512)

[Generación de servicio 8](#_Toc42960513)

[Token interceptor service: 12](#_Toc42960514)

# Creación del proyecto

Creamos el package con npm init -y

Instalamos las siguientes dependencias

Npm i express mongoose cors jsonwebtoken

Cors => compatibilidad servidor angular con nodejs-express

Jsonwebtoken => el servidor crea un token y se lo da al user si tiene permiso de entrada. El server valida cada vez que el user hace algo.

Inicializamos server en puerto 3000

const express = require('express');

require('./database');  //link a dataabase

const app = express();

app.set('port',process.env.PORT || 3000); //inicializa el puerto del pc o el 3000

const puerto = app.get('port');

app.listen(puerto);

console.log("server on:",puerto);

Creamos archivo database con la conexión y la requerimos en el index principal

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/angular-auth',

{

    useNewUrlParser:true,

    useUnifiedTopology:true,

}).then(db => console.log('database on'))

    .catch(err => console.log(err));

Creamos una carpeta para modelar los datos. Carpeta models y dentro una archivo llamado users.js

const { Schema , model } = require('mongoose'); //nos traemos de moongose las clases schema y model

//Schema definira el formato de datos, en este caso 2.

const userSchema = new Schema({

    email: String,

    password: String})

// Definimos el modelo user que usara este esquema y lo exportamos.

module.exports = model('User',userSchema);

Creamos carpeta routes y un archivo que contendrá las rutas llamado index.js

const express = require('express');

const router = express.Router();

router.get('/',(req,res)=>{

    res.send('ya no te pasha por el parke');

});

module.exports = router;

y lo linkamos a index.js

app.use(require('./routes/index'));

ahora linkamos en index.js de routes el modelo usuario que creamos en la carpeta models

const User = require('../models/users');

y lo usamos en una nueva ruta llamada registro:

router.post('/registro', async (req,res) =>{

    const { email , password } = req.body; //captura la petición y la guarda en dos variables

    const newUser = new User({email, password}); //crea una clase usuario y la almacena con las variables en newUser

    await newUser.save(); //guarda la clase en la db

    const token = jwt.sign({\_id: newUser.\_id},'secret'); //con el modulo jwt creo un token con la id del usuario

    res.status(200).json({token}); // devuelvo el token al cliente

    //IMPORTANTE: ENCRIPTAR CON BCRIPT

})

Ahora creamos una ruta llamada login

//IMPORTANTE: validacion simple, hay que cifrarlo

router.post('/login', async (req,res)=>{

    const { email , password } = req.body;

    const user = await User.findOne({email}); //busca el correo en la bd para ver si coincide con el introducido.

    if (!user) return res.status(401).send('El correo no existe');

    if (user.password !== password) return res.status(401).send('Password no existe');

    const token = jwt.sign({\_id: user.\_id},'secret'); //le asignamos el token, debe ser el mismo que el especificado antes.

    res.status(200).json({token});

})

Ahora necesitamos información privada y publica para que logear sirva de algo.

Hacemos unas rutas con información privada y otra con publica

router.get('/private', validatetoken, (req,res)=>{

    res.json([

        {

            \_id:1,

            name:"task1PRIVATE",

            desc: "PRIVATE"

        },

        {

            \_id:2,

            name:"task2PRIVATE",

            desc: "PRIVATE"

….

Ambas serán similares. Pero la privada tendrá la función de validar token.

function validatetoken(req,res,next)

{

    if (!req.headers.authorization)

    return res.status(401).send('no autorizado');

    const token = req.headers.authorization.split(' ')[1]; //saca el token (quita bearer)

    if (token === null)

    return res.status(401).send('no autorizado');

    const payload = jwt.verify(token,'secret',(err, decoded) => {

        if (err) {

            console.log(jwt.decode(token));

            console.log(err.name,":-:", err.message);

            return res.status(401).send('no autorizado');

        } else {

            return decoded;

        }

    });

    // const payload = jwt.verify(token,'secret');

    req.userID = payload.\_id;; //obtiene el content del token

    next();

}

Esta función mal hecha es para validar el token.

Podemos emplearla y usar lo que devuelve (req.userID) en cualquier función

router.get('/profile',validatetoken,(req,res)=>{

    res.send(req.userID);

})

# FRONTEND ANGULAR

Instalamos el angular cli en la consola : npm install -g @angular/cli (global)

Arreglamos el path del sistema con C:\Users\Usuario\AppData\Roaming\npm en las variables de entorno path.

Desactivamos el bloqueo de ejecución de script con Set-ExecutionPolicy Unrestricted en el terminal. Ejecutamos en el terminal ng -new frontend –routing Para crear el proyecto, esto crea la carpeta con todas la estructura. Entramos en la terminal en la carpeta y escribimos ng-serve para iniciar el servidor de angular jo

Vamos a la ruta frontend>src>app>app.component.html y borramos todo lo que está entre comentarios. Exceptuando

<router-outlet></router-outlet>

Con esto quitamos la plantilla de angular que viene por defecto, router-outlet es el enrutador. Debemos dejarlo.

Creamos un componente typando en la terminal ng generate component (n g c)

ng g c components/login … creamos 4 login,registro,publico,privado

En la ruta frontend>src>app>app.routing.modules.ts encontramos un array vacío

const routes: Routes = [];

Aquí se ponen las rutas. Para ello vamos a importar los componentes creados

import { LoginComponent } from "././components/login/login.component";

import { TaskComponent } from "././components/task/task.component";

import { RegistroComponent } from "././components/registro/registro.component";

import { PrivateTaskComponent } from "././components/private-task/private-task.component";

Ahora modificamos el array vacío de las rutas.

Para la ruta inicial es el primer objeto, el segundo es para cualquier ruta component.

const routes: Routes = [

  { //ruta inicial

    path:'', //dejamos el path vacio

    redirectTo:"/task", //redirecionamos a lo que queramos

    pathMatch:"full" // para que redirecione

  },

  {path:"task",component: TaskComponent},

  {path:"private",component:PrivateTaskComponent},

  {path:"login",component:LoginComponent},

  {path:"registro",component:RegistroComponent}

];

La única ruta con validación de entrada es la de private.

Agregamos boostrap para que quede bonito el css Podemos instalarlo con npm o usar el cdn.

Para pegar el cdn nos vamos a src>index.html y pegamos en un stylesheet el cdn.

Ahora en src>app>app.component.html pegamos una barra de navegación de boostrap

La ajustamos como queramos, y cambiamos los href por routerlink=”/login”

        <a class="nav-link" routerLink="/task" routerLinkActive="active">PUBLIC <span class="sr-only"></span></a>

Arreglamos el contenido poniéndole un container a la outlet para que se vea mas centrado y no pegado a los bordes.

<div class="container p-3">

  <router-outlet></router-outlet>

</div>

Nos vamos al componente de registro. En la ruta src>app>components>registro>registro.component.html

Y creamos la plantilla del registro

<div class="row">

    <div class="col-md4-mx-auto">

        <div class="card">

            <div class="card-header">

                Registro

                <div class="card-body">

                    <form (submit)="signUp()">

                        <div class="form-group">

                            <input type="text" name="email" class="form-control" placeholder="email" required autofocus>

                        </div>

                        <div class="form-group">

                            <input type="text" name="password" class="form-control" placeholder="password" required>

                        </div>

                        <button type="submit" class="btn-primary btn-block">Registrar</button>

                    </form>

                </div>

            </div>

        </div>

    </div>

</div>

Esta plantilla html nos pondrá dos campos email y password para rellenar y un botón que envía el formulario.

Para importar el modulo de angular de forms vamos a la ruta src>app>app.module.ts

import { FormsModule} from '@angular/forms';

@NgModule({

  declarations: [

    AppComponent,

    RegistroComponent,

    LoginComponent,

    TaskComponent,

    PrivateTaskComponent

  ],

  imports: [

    BrowserModule,

    AppRoutingModule,

    FormsModule //agregamos el modulo aqui tambien

  ],

  providers: [],

  bootstrap: [AppComponent]

})

La función del formulario signUp la definiremos en src>app>components>registro>registro.component.ts

export class RegistroComponent implements OnInit {

  user ={

    email:'',

    password: ''

  }

  constructor() { }

…

  signUp()

  {

    console.log(this.user);

  }

Aquí definimos la función. Para pasarle datos a user nos vamos al a plantilla del componente y añadimos los ngmodel

<input type="text" [(ngModel)]="user.email" name="email" class="form-control" placeholder="email" required autofocus>

<input type="text" [(ngModel)]="user.password"  name="password" class="form-control" placeholder="password" required>

# Generación de servicio

Para generar un servicio typamos en la consola ng g s services/auth

Generamos 3 el auth, task y token-interceptor.

El primero que crearemos será auth service

En él importamos el modulo http

import { HttpClient } from "@angular/common/http";

NOTA: Para usar este módulo, debemos importar otro modulo general en app.module.ts

import { HttpClientModule } from "@angular/common/http";

y instanciarlo en los imports

y lo instanciamos

  constructor(private http:HttpClient) {

creamos una variable con la URL del server

private URL = "http://localhost:3000"

la función del servicio que se comunicará con esto

  singUp(user) {

    return this.http.post(this.URL + '/registro', user);

  }

Ahora agregamos nuestro servicio auth al modulo principal app.module.ts si lo queremos usar desde cualquier parte. O lo agregamos al componente que usará el servicio (local). Nosotros lo agregamos al componente registro.

En src>app>component>registro>registro.component.ts

import { AuthService } from "../../services/auth.service";

y instanciamos la clase AuthService en el constructor

  constructor(private AuthService:AuthService) { }

Modificamos la función de abajo (signup) que solo tenía un console log

signUp()

  {

    console.log(this.user);

  }

Para comunicar el servidor de angular con el del backend de express tenemos que configurar CORS.

Vamos al backend. A index.js y importamos el modulo cors

const cors =require('cors');

app.use(cors());

Este modulo se encarga de escribir unas cabeceras (headers) para poder realizar la comunicación entre servidores. Se debe de configurar para producción.

Una vez hecho esto no nos dará error de CORS y vemos que nos devuelve por console.log del registro.component.ts el res es el token de identificación. Este token lo tenemos que guardar en cookies o algo asi, pj en localstorage.

  signUp()

  {

    this.AuthService.singUp(this.user).subscribe(

      res => {console.log(res),localStorage.setItem('token',res['token'])}, err => console.log(err)

    ); //suscribe maneja la respuesta del servidor

  }

En console > application > LocalStorage podemos visualizar el token almacenado.

Para redireccionar al usuario una vez registrado desde el componente importamos el siguiente modulo

import { Router } from "@angular/router";

Esto es router de angular. Lo instanciamos en el constructor y lo usamos en la función

 res => {

        localStorage.setItem('token',res['token']);

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

      }, err => console.log(err)

Una vez esto tenemos una herramienta básica de registro. Vamos ahora a hacer la del login

Copiamos la misma template de registro para login y le cambiamos los nombres correspondientes

En el componente de login agregamos el objeto user con los parámetros y también declaramos la función signIn de la template en el componente. Y en el servicio de authentificacion (auth) le escribimos la función para que envíe un post a la url actual cuando el formulario haga submit.

Escribimos lo siguiente en src>app>services>auth.services

  singIn(user) {

    return this.http.post(this.URL + '/login', user);

  }

Ahora volvemos al componente e importamos el servicio

import { AuthService } from "../../services/auth.service";

y lo instanciamos en el constructor

  constructor(private AuthService:AuthService) { }

La función signIn tendrá la misma estructura que la signup

 signIn()

  {

    this.AuthService.singIn(this.user).subscribe(

      res => {

        console.log(res);

        localStorage.setItem('token', res['token']);

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

      }, err=> console.log(err)

    );

  }

Ahora bloquearemos la vista privada para que solo se pueda acceder authentificado.

Para ello usaremos un Ward, para crearlo typamos en consola : ng g g auth donde auth es el nombre del guard. Lo instalamos con la interfaz canactivate y vemos que crea 2 archivos.

Vamos a src>app>auth.guard.ts , aquí importamos el auth service.

import { AuthService } from './services/auth.service';

En el servicio añadimos una función para ver si el usuario está logeado o no, básicamente checekea el token

  loggedIn(){

    if (localStorage.getItem('token'))

    return true;

    else

    return false;

    // Podemos poner en su lugar return !! localStorage.getItem('token)

    //!! significa si existe return true si no return false.

  }

Volvemos de nuevo al Guard. Aquí escribimos el constructor para instanciar el servicio importado

constructor(

  private AuthService:AuthService,

  private router:Router)

  {}

y modificamos la función canActivate que es retornara true o false en función de si puede o no acceder a la ruta.

  canActivate()

  {

    if (this.AuthService.loggedIn())

    return true //si la funcion loggedIn retorna true canactivate tambien.

    this.router.navigate(['/signin']); // sino redirect signin y retornara false.

    return false;

  }

Para usar el guard. Tenemos que declararlo en app.modules.ts . lo importamos

import { AuthGuard } from "./auth.guard";

y lo declaramos en providers

  providers: [AuthGuard],

con esto podemos usarlo en cualquier clase de la app.

Vamos ahora a app.routing.modules.ts donde están los enrutados de angular y importamos el guard.

Para usarlo, vamos a la ruta que queremos proteger y añadimos lo siguiente

  {path:"private",component:PrivateTaskComponent,canActivate:[AuthGuard]}

Hemos añadido el canActivate con el authguard.

# Token interceptor service:

Importamos el http interceptor

import { HttpInterceptor } from "@angular/common/http";

lo instanciamos

export class TokenInterceptorService implements HttpInterceptor {

Vamos a auth service y creamos una función para obtener el token.

  getToken()

  {

    return localStorage.getItem('token');

  }

Esta función nos devuelve el token.

Volvemos al servicio token interceptor, importamos el auth service y lo instanciamos en el constructor, y usamos la función intercept de http interception

  intercept(req,next)

  {

    const tokenizeReq = req.clone({

      setHeaders:{autorizacion:`Bearer ${this.AuthService.getToken()}`}

    })

    return next.handle(tokenizeReq);

  }

Esta función añade en la cabecera de la petición http un texto bearer seguido del token y la devuelve.

Vamos a app.module.ts para importarlo todo.

import { HttpClientModule , HTTP\_INTERCEPTORS } from "@angular/common/http";

import { TokenInterceptorService } from "./services/token-interceptor.service";

y en el provider lo declaramos asi

  providers: [

    AuthGuard,

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: TokenInterceptorService,

      multi:true

    }

  ],

Ahora nuestras peticiones van a tener una cabecera extra.(bearer + token)

NOTA: realiza el interceptor antes de obtener el token.. no funciona bien.

Realizamos ahora el logout, en auth escribimos la función

  logout()

  {

    localStorage.removeItem('token');

    this.router.navigate(['/login']);

  }

Y ahora nos vamos a la plantilla de navegación para modificarla

    <ul class="navbar-nav ml-auto">

      <ng-container \*ngIf="!AuthService.loggedIn(); else LoggedIn">

        <li class="nav-item">

          <a class="nav-link" routerLink="/login" routerLinkActive="active">LOGIN</a>

        </li>

        <li class="nav-item">

          <a class="nav-link "routerLink="/registro" routerLinkActive="active">REGISTER</a>

        </li>

      </ng-container>

      <ng-template #LoggedIn>

        <li class="nav-item">

         <a class="nav-link" style="cursor:pointer;" (click)="AuthService.logout()">L0G0UT</a>

        </li>

      </ng-template>

    </ul>

Para poder usar authservice en la plantilla tenemos que importarlo en el componente.

En src>app>app.component.ts importamos

import { AuthService } from './services/auth.service';

y creamos el constructor (que no estaba creado)

  constructor(public AuthService:AuthService){}

la ponemos public para poder usarla desde el template.

Ahora ponemos funcionalidad a las tareas.

En task service

import { Injectable } from '@angular/core';

import { HttpClient } from "@angular/common/http";

@Injectable({

  providedIn: 'root'

})

export class TaskService {

  private URL = 'http://localhost:3000'

  constructor(private http:HttpClient) { }

  getTask()

  {

    return this.http.get(this.URL + '/task');

  }

  getPrivateTask()

  {

    return this.http.get(this.URL + '/private');

  }

En task component

import { Component, OnInit } from '@angular/core';

import { TaskService } from "../../services/task.service";

@Component({

  selector: 'app-task',

  templateUrl: './task.component.html',

  styleUrls: ['./task.component.css']

})

export class TaskComponent implements OnInit {

  tasks = [];

  constructor(private taskservice:TaskService) { }

  ngOnInit(): void {

    this.taskservice.getTask().subscribe(

      res =>this.tasks = res

    )

  }

}

Y en la template los iteramos para mostrarlos:

<ul class="list-group">

    <li class="list-group-item" \*ngFor="let task of tasks">

        {{task.\_id}} - {{task.name}}

    </li>

</ul>

Haríamos lo mismo para las tasks privadas.