TRABAJANDO CON EL BACKEND – ESQUEMA USUARIO

Trabajando con el Esquema Usuario

Creamos el Esquema Usuario en el backend siguiendo los tipos de datos soportados por mongoose, luego se exporta el modelo para que sea referenciado desde otro lugar.

Archivo models/usuario.js

```
const mongoose = require("mongoose");
const {Schema} = mongoose;

const UsuarioSchema = new Schema({
username: {type: String, required: true},
password: {type:String, required:true},
nombres: {type:String, required:true}, // opcional: puede almacenarse en otra clase
apellido: {type:String, required:true}, // opcional: puede almacenarse en otra clase
perfil: {type:String, required: true} //administrador - visitante - administrativo
});
//exporto objeto para que pueda ser usado en otros lugares
module.exports = mongoose.model('Usuario', UsuarioSchema);
```

Trabajando con el controlador y rutas de Usuario

Mediante la librería EXPRESS definimos las rutas que escucharán llamadas en la API, en la definición de estas rutas se usan métodos del controller “usuarioCtrl” que son los que terminarán ejecutando código contra el repositorio Mongo DB.

Archivo routes/usuario.route.js

```
const express = require("express");
const router = express.Router();

//defino controlador para el manejo de CRUD
const usuarioCtrl = require('./../controllers/usuario.controller');

// definiendo rutas
router.post('/', usuarioCtrl.createUsuario);
router.post('/login', usuarioCtrl.loginUsuario);

//exportacion del modulo de rutas
module.exports = router;
```

Archivo controllers/usuario.controller.js

```
const Usuario = require ('./../models/usuario')


const usuarioCtrl = {}
```

```
usuarioCtrl.createUsuario = async (req, res)=>{
    //en req.body se espera que vengan los datos de usuario a crear
    const usuario = new Usuario (req.body);
    try {
        await usuario.save();
        res.status(200).json({
            'status': '1',
            'msg': 'Usuario guardado.'
        })
    } catch (error) {
        res. status(400).json({
            'status': '0',
            'msg': 'Error procesando operacion.'
        })
    }
}

usuarioCtrl.loginUsuario = async (req, res) => {
    //en req.body se espera que vengan las credenciales de login
    //defino los criterios de busqueda en base al username y password recibidos
    const criteria = {
        username: req.body.username,
        password: req.body.password
    }

    try {
        //el método findOne retorna un objeto que cumpla con los criterios de busqueda
        const user = await Usuario.findOne(criteria);
        if (!user) {
            res.json({
                status: 0,
                msg: "not found"
            })
        } else {
            res.json({
                status: 1,
                msg: "success",
                username: user.username, //retorno información útil para el frontend
                perfil: user.perfil, //retorno información útil para el frontend
```

```
                userid: user._id //retorno información útil para el frontend
            })
        }
    } catch (error) {
        res.json({
            status: 0,
            msg: 'error'
        })
    }

}
//exportacion del modulo controlador
module.exports = usuarioCtrl;
```

Agregamos el archivo de rutas al archivo index.js

```
….
app.use('/api/usuario', require('./routes/usuario.route'));
….
```

Creamos un usuario de prueba y lo ingresamos via POSTMAN ya que no tenemos la interface desarrollada

```
{
        "username": "jperez",
        "password": "test123",
        "nombres": "Juan",
        "apellido": "Perez",
        "perfil": "administrador"
}
```

TRABAJANDO CON EL FRONTEND - ANGULAR

1. Se genera una clase Usuario en la carpeta models.

```
> ng generate class models/usuario
CREATE src/app/models/usuario.spec.ts (158 bytes)
CREATE src/app/models/usuario.ts (25 bytes)
```

```
export class Usuario {
    _id!: string;
    username!: string;
    password!: string;
    nombres!: string;
    apellido!: string;
    perfil!: string;
```

```
    constructor(id:string="", username:string="", password:string="", nombres:string="",
apellido:string="", perfil:string=""){
      this._id = id;
      this.username = username;
      this.password = password;
      this.nombres = nombres;
      this.apellido = apellido;
      this.perfil = perfil;
    }
}
```

2. Se genera un servicio en angular para que haga la llamada al webservice del backend

```
> ng g service services/login
```

3. Se modifica el servicio para que soporte los métodos de login y logout.

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class LoginService {

  hostBase: string;

  constructor(private _http:HttpClient) {
    this.hostBase = "http://localhost:3000/api/usuario/";
  }

  public login(username: string, password: string):Observable<any> {
    const httpOption = {
      headers: new HttpHeaders({
        'Content-Type':  'application/json'
      })
    }
    let body = JSON.stringify({ username: username, password: password });
    console.log(body);
    return this._http.post(this.hostBase + 'login', body, httpOption);
  }

  public logout() {
    //borro el vble almacenado mediante el storage
    sessionStorage.removeItem("user");
    sessionStorage.removeItem("perfil");
    sessionStorage.removeItem("userid");

  }
```

```
  public userLoggedIn(){
    var resultado = false;
    var usuario = sessionStorage.getItem("user");
    if(usuario!=null){
      resultado = true;
    }
    return resultado;
  }

  public userLogged(){
    var usuario = sessionStorage.getItem("user");
    return usuario;
  }

  public idLogged(){
    var id = sessionStorage.getItem("userid");
    return id;
  }

}
```

4. SOLO VALIDO PARA Angular < 17

El servicio se cargará en el **module.ts** utilizando el array de PROVIDERS para que se instancie, luego este ÚNICO objeto se irá inyectando en los componentes que lo requieran (más adelante en headerComponent, loginComponent) para determinar si alguien está o no logueado.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

……
……

import { LoginService } from './services/login.service';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
     ……
     ……
      ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    AppRoutingModule
  ],
  providers: [LoginService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5. Se genera el componente de login para que soporte el diseño del formulario de login y el comportamiento del método de login como se muestra a continuación.

```
> ng generate component  components/login
```

Luego se agrega el componente al array de rutas en el archivo app-routing.module.ts para poderlo usar en las referencias de los enlaces.

```
    {path: 'login', component: LoginComponent},
```

6. Se modifica la vista y el controlador typescript del componente de login.

```
<div class="col-md-6 col-md-offset-3">
    <h2>Login</h2>
    <form name="form">
        <div class="form-group">
            <label for="username">Username</label>
            <input type="text" class="form-
control" name="username" [(ngModel)]="userform.username" />
        </div>
        <div class="form-group">
            <label for="password">Password</label>
            <input type="password" class="form-
control" name="password" [(ngModel)]="userform.password"  />
        </div>
        <div class="form-group">
            <button class="btn btn-primary" (click)="login()">Login</button>
        </div>
        <div class="alert-danger">{{msglogin}}</div>
    </form>
</div>
```

Se modifica el controlador del componente login. Se observa que se inyectan varios objetos al constructor.

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { Usuario } from 'src/app/models/usuario';
import { LoginService } from 'src/app/services/login.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})

export class LoginComponent implements OnInit {
```

```
  userform: Usuario = new Usuario(); //usuario mapeado al formulario
  returnUrl!: string;
  msglogin!: string; // mensaje que indica si no paso el loguin

  constructor(
    private route: ActivatedRoute,
    private router: Router,
    private loginService:LoginService){
    }

  ngOnInit() {
    this.returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/home';
  }

  login() {
    this.loginService.login(this.userform.username, this.userform.password)
        .subscribe(
            (result) => {
              var user = result;
              if (user.status == 1){
                //guardamos el user en cookies en el cliente
                sessionStorage.setItem("user", user.username);
                sessionStorage.setItem("userid", user.userid);
                sessionStorage.setItem("perfil", user.perfil);

                //redirigimos a home o a pagina que llamo
                this.router.navigateByUrl(this.returnUrl);
              } else {
                //usuario no encontrado  muestro mensaje en la vista
                this.msglogin="Credenciales incorrectas..";
              }
            },
            error => {
                alert("Error de conexion");
                console.log("error en conexion");
                console.log(error);
            });
  }
}
```

7. Se modifica el header component para que desde ahí se haga el logout del usuario, se usa la directiva *ngIf de angular para mostrar u ocultar partes de la vista, se usa el servicio de angular como punto de comunicación entre loginComponent y headerComponent que en este caso comparte el método userLoggedIn que es un boleano que si tiene true es porque hay un usuario logueado.

   Se agrega la opción login/logout en el navbar.

```
…
…
     <li class="active"><a [routerLink]="['home']">Home</a></li>
     <li><a [routerLink]="['about']">About</a></li>
…
…
  <div>
    <a *ngIf="!loginService.userLoggedIn()" [routerLink]="['login']">(Login)</a>
    <a *ngIf="loginService.userLoggedIn()" [routerLink]="['login']" (click)="logout()">
      {{loginService.userLogged()}} (logout)
    </a>
  </div>
…
…
```

Dentro del controller del header el método de logout() tiene el siguiente comportamiento en el typescript del controlador del componente header.

```
import { Component, OnInit } from '@angular/core';
import { LoginService } from 'src/app/services/login.service';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent implements OnInit {

  constructor(public loginService: LoginService) { }

  ngOnInit() {
  }

  logout(){
    this.loginService.logout();
  }
}
```

8. Luego para evitar el acceso a ciertas vistas de los componentes sin haber pasado por la autenticación se puede entre otras opciones. Ej. Ocultamos en caso de que no haya nadie autenticado el contenedor más abarcativo del componente. En el código de arriba `userLoggedIn` es una propiedad del service loginService.

```
<div *ngIf="loginService.userLoggedIn()">
…
…
…
</div>
```

Tomamos como ejemplo cualquier componente de nuestro proyecto y en su controlador será necesario nuevamente inyectar el servicio LoginService para poder utilizar sus propiedades y corroborar si existe existe un usuario logueado y en todo caso cual es.

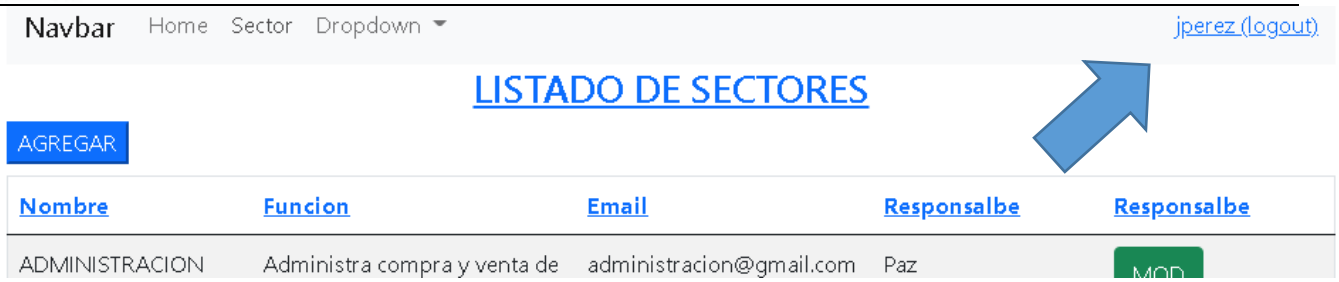
```
import { Component, OnInit } from '@angular/core';
import {MensajesService} from './../../services/mensajes.service';
import {Empresa} from './../../models/empresa';
import {Mensaje} from './../../models/mensaje';
import {LoginService} from '../../services/login.service';

@Component({
  selector: 'app-punto4',
  templateUrl: './punto4.component.html',
  styleUrls: ['./punto4.component.css']
})
export class Punto4Component implements OnInit {

.....
.....


  constructor(private servicio:MensajesService,
              public loginService: LoginService) {
      .....
      .....
  }
```

Se observa en la siguiente figura que si hay alguien logueado podremos acceder al componente caso contrario, no se verá nada.



9. También podemos en el constructor de los componentes del sistema preguntar por el usuario logueado y redirigir al componente de Loguin si el usuario aún no lo está:

```
…….
…….
  constructor(public loginService: LoginService) {
                    if(this.loginService.userLoggedIn()){
                          //controlo si alguien esta logueado, ejecuto acciones normales
                          //controlo si alguien esta logueado, ejecuto acciones normales
                    } else {
                          alert("Debe validarse e ingresar su usuario y clave");
                          this.router.navigate(['login']);
                    }
…….
…….
   }
```

Nota: las vistas o paginas públicas que no requieren autenticación y brindan autorización a todo el público no sufren ninguna modificación al respecto.