*Àlex Macià Fiteni*

# Práctica 3 Opcional – Login y Rutas

Añadiendo login de usuario a la aplicación ToDo

*NodeJS, Angular, API RESTFul*

## Objetivo de la práctica

El objetivo de esta práctica es extender la funcionalidad de nuestra **ToDo App** desarrollada en la práctica 3 para que soporte un sistema de usuarios. En esta segunda versión, se podrá registrarse en la página y luego iniciar sesión, de manera que cada usuario verá sólo sus tareas.

Para conseguirlo habrá que modificar el servidor API RESTFul, pero sobre todo la propia aplicación ToDo.

Si quiere saber más sobre el desarrollo de la aplicación distribuida y todos sus detalles véase la memoria de la práctica 3.

## Herramientas

Utilizaremos las mismas herramientas que utilizamos en el desarrollo de ToDo App:

- Un **editor de código:** El elegido en este caso es VSCode, pues conoce NodeJS de manera nativa y es más fácil leer el código.
- Un **navegador convencional** para ver la aplicación. Hemos elegido Firefox.
- Un **cliente web** más versátil que un navegador, capaz de realizar las pruebas que hagan falta. Al igual que en la práctica anterior utilizaremos Postman

## Desarrollo de las funcionalidades

El primer cambio que vamos a realizar es en el **API RESTful** para que nos permita iniciar sesión. Lo único que necesitamos añadir son nuevas rutas específicas para el inicio de sesión y la recuperación de tareas de un usuario específico. La creación de usuarios será una simple entrada de datos en una colección **users**, así que el **POST** que ya tenemos nos vale.

El inicio de sesión es con **POST** y enviando el nombre de usuario y la contraseña en el **body** al recurso **api/sessions**. Como tenemos una ruta genérica para cualquier **api/:coleccion** tendremos que poner esta nueva ruta por encima del resto de **POSTs** para que salte antes (de otra manera, **api/sessions** sería interpretado como **api/:coleccion**, siendo colección = sessions). Devolverá el objeto del usuario de la base de datos, y en la aplicación de Angular recuperaremos su ID. Sería algo así:

**Fragmento 1.1:** Ruta para inicio de sesión

```
app.post('/api/sessions', (req, res, next) => {
        const userData = req.body;
        console.log(userData);

        req.collection = db.collection("users");
        req.collection.findOne({username: userData.username, password: userData.password}, (err, elemento) => {
                if(err) return next(err);
                res.json(elemento);
        });
});
```

Para recuperar tareas de un usuario específico necesitamos otra excepción. Antes de nada tengamos en cuenta que las tareas ahora van a ser creadas con un campo **user** que indicará la ID de Mongo del usuario propietario. Esta vez queremos un **GET** al recurso **api/tasks/:userID**, en la que userID es la id del

usuario a filtrar en la colección **tasks**. De nuevo hay que añadirla sobre otros GET para que las llamadas más genéricas no intervengan en su lugar. Sería algo así:

**Fragmento 1.2:** Tareas de un solo usuario

```
app.get('/api/tasks/:user', (req, res, next) => {
		req.collection = db.collection("tasks");

		req.collection.find({user: req.params.user}).toArray((err, coleccion) => {
			if(err) return next(err);
			res.json(coleccion);
		});
});
```

Es muy importante que estas cosas se calculen aquí, en el lado del servidor. Si para mostrar las tareas de un usuario las bajáramos todas en la app y las filtráramos ahí, habríamos cargado las tareas de todo el mundo, lo que supone una falta de confidencialidad (cualquiera podría ver las tareas de los demás) a parte de un malgasto de memoria del cliente. Con todo, el archivo final queda así:

**Archivo 1:** index.js

```
'use strict'

// Asigna el puerto a la variable de entorno. Si no existe coge el 3000
const port = process.env.PORT || 3000;

const express = require('express');
const logger = require('morgan');
const mongojs = require('mongojs');
const cors = require('cors');

const app = express();

var db = mongojs("SD");
var id = mongojs.ObjectID;

var allowCrossTokenHeaders = (req, res, next) => {
		res.header("Access-Control-Allow-Headers", "*");
		return next();
};

var allowCrossTokenOrigin = (req, res, next) => {
		res.header("Access-Control-Allow-Origin", "*");
		return next();
};

// Middlewares
app.use(logger('dev'));
app.use(express.urlencoded({extended:false}));
app.use(express.json());

app.use(cors());
app.use(allowCrossTokenHeaders);
app.use(allowCrossTokenOrigin);

// Anyadimos un trigger para cambiar de coleccion dinamicamente
app.param("coleccion", (req, res, next, coleccion) => {
		req.collection = db.collection(coleccion);
		return next();
});

// Routes
app.get('/api', (req, res, next) => {
		db.getCollectionNames((err, colecciones) => {
			if(err) return next(err);
			res.json(colecciones);
		});
});

app.get('/api/tasks/:user', (req, res, next) => {
		req.collection = db.collection("tasks");
```

```
		req.collection.find({user: req.params.user}).toArray((err, coleccion) => {
			if(err) return next(err);
			res.json(coleccion);
		});
});

app.get('/api/:coleccion', (req, res, next) => {
		req.collection.find((err, coleccion) => {
			if(err) return next(err);
			res.json(coleccion);
		});
});

app.get('/api/:coleccion/:id', (req, res, next) => {
		req.collection.findOne({_id: id(req.params.id)}, (err, elemento) => {
			if(err) return next(err);
			res.json(elemento);
			});
});

app.post('/api/sessions', (req, res, next) => {
		const userData = req.body;
		console.log(userData);

		req.collection = db.collection("users");
		req.collection.findOne({username: userData.username, password: userData.password}, (err, elemento) => {
			if(err) return next(err);
			res.json(elemento);
		});
});

app.post('/api/:coleccion', (req, res, next) => {
		const elemento = req.body;

		// Save es capaz de crear o actualizar una entrada. Es como si
		// automaticamente hiciese insert o update en funcion de lo que necesite
		req.collection.save(elemento, (err, elementoGuardado) => {
			if(err) return next(err);
			res.json(elementoGuardado);
		});
});

app.put('/api/:coleccion/:id', (req, res, next) => {
		const elemento = req.body;
		console.log(elemento);

		// update nos pide una query json para actualizar la entrada especificada
		// En nuestro caso esta query sera la id que nos digan en los parametros
		// Tambien pasamos los nuevos datos
		req.collection.update({_id: id(req.params.id)}, elemento, (err, resultado) => {
			if(err) return next(err);
			res.json(resultado);
		});
});

app.delete('/api/:coleccion/:id', (req, res, next) => {
		// remove nos pide una query json para borrar la entrada especificada
		// En nuestro caso esta query sera la id que nos digan en los parametros
		req.collection.remove({_id: id(req.params.id)}, (err, resultado) => {
			if(err) return next(err);
			res.json(resultado);
		});
});

// Lanzamos nuestro servicio
app.listen(port, ()=>{
		console.log(`API REST ejecutándose en http://localhost:${port}/api`);
});
```

Ahora podemos desarrollar la funcionalidad de la app en **Angular**. En la aplicación vamos a añadir varias cosas nuevas: una clase **usuario**, un componente **login**, un comoponente **registro**, un servicio de **usuarios** (para registrarse e iniciar sesión) y los cambios pertinentes en **app.component**.

Vamos a empezar por la clase usuario. Nos dirigimos a la carpeta **src/app** y creamos el archivo **User.ts**.

Luego escribimos lo siguiente:

**Archivo 2:** User.ts

```
// Creamos la clase User
export class User {
        _id?: string;
        username: string;
        password: string;
        email?: string;
}
```

Ahora vamos a crear el servicio users que realice las llamadas al API. Vamos desde la terminal al directorio raíz de la aplicación y ejecutamos:

```
$ ng g s services/user
```

Ahora vamos en el editor de código al archivo **user.service.ts** en **src/app/services/** y nos disponemos a terminarlo. El procedimiento es similar al que seguimos para crear el servicio task: Primero importamos el módulo de cliente Http, el de map, y esta vez la clase usuario

**Fragmento 3.1:** Imports

```
import { HttpClient } from '@angular/common/http';
import { map } from 'rxjs/operators';
import { User } from '../User';
```

Ahora en la clase **UserService** declaramos la propiedad **apiREST** que contiene la url al api:

**Fragmento 3.2:** Ruta al API REST

```
apiREST = "http://localhost:3000/api";
```

Luego en el constructor declaramos un nuevo **HttpClient**.

**Fragmento 3.3:** Constructor

```
constructor(private http: HttpClient) { }
```

Ahora declaramos los métodos para registrarse y para iniciar sesión:

**Fragmento 3.4:** Métodos

```
userRegister(newUser: User) {
        return this.http.post<User>(`${this.apiREST}/users`, newUser)
        .pipe(map(res=>res));
}

userLogin(userData: User) {
        return this.http.post<User>(`${this.apiREST}/sessions`, userData)
        .pipe(map(res=>res));
}
```

Fíjate que hemos usado para el login un **POST** a la ruta que hemos creado antes de **api/sessions**, pero para registrar al usuario usamos un **POST** a la colección **users** con los datos del usuario.

Por último vamos a añadir una serie de métodos estáticos que gestionen el estado de sesión usuario en **localStorage**. Los métodos sirven para saber si el usuario ha iniciado sesión, para iniciarla (guardando la id en **localStorage**) y para comprobar esa ID.

**Fragmento 3.5:** Métodos estáticos

```
static isLoggedIn() {
		if(!localStorage.getItem("userID")) return false;
		else return true;
}

static getCurrentUserID() {
		if(!localStorage.getItem("userID")) return '';
		return localStorage.getItem("userID");
}

static getCurrentUserName() {
		if(!localStorage.getItem("username")) return '';
		return localStorage.getItem("username");
}

static setCurrentUserID(id: string) {
		localStorage.setItem("userID", id);
}

static setCurrentUserName(name: string) {
		localStorage.setItem("username", name);
}
```

Con todo, el archivo queda así:

**Archivo 3:** user.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { map } from 'rxjs/operators';
import { User } from '../User';

@Injectable({
		providedIn: 'root'
})
export class UserService {
		apiREST = "http://localhost:3000/api";

		constructor(private http: HttpClient) { }

		userRegister(newUser: User) {
			return this.http.post<User>(`${this.apiREST}/users`, newUser)
				.pipe(map(res=>res));
		}
		userLogin(userData: User) {
			return this.http.post<User>(`${this.apiREST}/sessions`, userData)
				.pipe(map(res=>res));
		}

		static isLoggedIn() {
			if(!localStorage.getItem("userID")) return false;
			else return true;
		}

		static getCurrentUserID() {
			if(!localStorage.getItem("userID")) return '';
			return localStorage.getItem("userID");
		}

		static getCurrentUserName() {
			if(!localStorage.getItem("username")) return '';
			return localStorage.getItem("username");
		}

		static setCurrentUserID(id: string) {
			localStorage.setItem("userID", id);
		}

		static setCurrentUserName(name: string) {
			localStorage.setItem("username", name);
```

```
        }
}
```

Ahora procederíamos a desarrollar los componentes login y registro, pero antes vamos a configurar las **rutas** para luego poder cambiar entre los componentes login, registro y tareas. Recordemos que las apps de **Angular** son de una sola página que carga distintos componentes.

Vamos **app.module.ts** y añadimos los import

```
import { RouterModule, Routes } from '@angular/router';
```

Luego creamos un array de rutas. Las rutas las define como objetos **JSON** con un formato específico

```
const appRoutes: Routes = [
        {path: 'tasks', component: TasksComponent},
        {path: 'login', component: LoginComponent},
        {path: 'register', component: RegisterComponent},
        {path: '', redirectTo: '/tasks', pathMatch: 'full'}
];
```

Todas las rutas tienen el parámetro **path** que indica la ruta que definen. Luego la mayoría tienen un **componente** asignado que es el que cargan si su ruta coincide. La última ruta, la ruta vacía, es la que se activa si no hay recurso en la URL. En este caso le hemos dicho que nos redirija a la ruta tasks. Además tiene el parámetro **pathMach** a full, ya que por defecto podría activarse por que la ruta en la url contenga parcialmente esa ruta, y la ruta vacía es una subcadena de cualquier ruta. En cualquier caso no se activaría, pues el orden importa y el router, cuando carguemos una ruta, buscará la primera que coincida en esta lista en orden.

Por último añadimos el módulo al array import:

```
RouterModule.forRoot(appRoutes, { enableTracing: true })
```

El parámetro **enableTracing** está por cuestiones de **debug**. Te va a llenar la consola de datos cada vez que algún evento del **router** se dispare. Ahora sí podemos desarrollar los componentes login y registro.

Desde la terminal, en el directorio raíz de la aplicación, ejecutamos

```
$ ng g c component/login
$ ng g c component/register
```

Ahora vamos al archivo **login.component.ts** en **src/app/component/login/** y lo abrimos con el editor de código.

Primero importamos los módulos que hagan falta. Necesitaremos las clases **User** y **UserService** y además **Router** para poder redirigirnos entre rutas.

**Fragmento 4.1:** Imports

```
import { User } from '../../User';
import { UserService } from '../../services/user.service';
import { Router } from '@angular/router';
```

Luego, en la clase, declaramos las propiedades necesarias. Dos son flags para ocultar o mostrar cierto contenido, y las otras dos son para vincularlas con el formulario.

**Fragmento 4.2:** Propiedades

```
 isLoggedIn: boolean = false;
lastTimeFailed: boolean = false;
username: string = '';
```

```
password: string = '';
```

Ahora en el constructor añadimos un **UserService** y un **Router**. En el cuerpo comprobamos si el usuario ha iniciado sesión para actualizar el valor de **isLoggedIn**

**Fragmento 4.3:** Constructor

```
constructor(private userService: UserService, private router: Router) {
	this.isLoggedIn = UserService.isLoggedIn();
}
```

Por último creamos el método login que luego invocaremos desde el formulario. Igual que en el componente **tasks**, recibimos por parámetro el **evento** y anulamos su acción por defecto (actualizar la página).

Luego creamos un **User** con las únicas propiedades obligatorias: username y password, que obtenemos del formulario. Por último usamos el método **userLogin** del servicio **UserService**.

Una vez dentro comprobamos el resultado. Si ningún usuario coincide con ese nombre y contraseña marcamos la flag **lastTimeFailed** (que luego mostrará mensaje de error), pero si ha funcionado guardamos su nombre de usuario e ID y redirigimos a la ruta vacía (que a su vez lleva a **/tasks**)

**Fragmento 4.4:** Método login()

```
login(event) {
	event.preventDefault();

	// Creamos los datos del usuario a enviar
	const userData:User = {
		username: this.username,
		password: this.password
	};

	this.userService.userLogin(userData).subscribe(result => {
		if(result) {
			console.log(result);

			// Reiniciamos el formulario
			this.username = '';
			this.password = '';

			// Guarda el usuario
			UserService.setCurrentUserID(result._id);
			UserService.setCurrentUserName(result.username);

			// Actualizamos los datos que hagan falta
			this.lastTimeFailed = false;
			this.isLoggedIn = UserService.isLoggedIn();

			// Redirigimos a tareas
			this.router.navigate(['']);
		}
		else {
			console.log("El usuario o la contraseña están mal");
			this.lastTimeFailed = true;
		}
	});
}
```

Con todo el archivo queda como sigue

**Archivo 4:** login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from '../../User';
import { UserService } from '../../services/user.service';
import { Router } from '@angular/router';

@Component({
	selector: 'app-login',
```

```
	templateUrl: './login.component.html',
	styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

	isLoggedIn: boolean = false;
	lastTimeFailed: boolean = false;
	username: string = '';
	password: string = '';

	constructor(private userService: UserService, private router: Router) {
		this.isLoggedIn = UserService.isLoggedIn();
	}

	ngOnInit() {
	}

	login(event) {
		event.preventDefault();

		// Creamos los datos del usuario a enviar
		const userData:User = {
			username: this.username,
			password: this.password
		};

		this.userService.userLogin(userData).subscribe(result => {
			if(result) {
				console.log(result);

				// Reiniciamos el formulario
				this.username = '';
				this.password = '';

				// Guarda el usuario
				UserService.setCurrentUserID(result._id);
				UserService.setCurrentUserName(result.username);

				// Actualizamos los datos que hagan falta
				this.lastTimeFailed = false;
				this.isLoggedIn = UserService.isLoggedIn();

				// Redirigimos a tareas
				this.router.navigate(['']);
			}
			else {
				console.log("El usuario o la contraseña están mal");
				this.lastTimeFailed = true;
			}
		});
	}

}
```

Ahora pasamos a **login.component.html** y escribimos todo esto:

**Archivo 5:** login.component.html

```
<label><h2>Iniciar Sesión</h2></label>
<div class="card">
	<!-- Esto aparece si todavía no has iniciado sesión -->
	<div class="card-body" *ngIf="!isLoggedIn">
		<form (submit)="login($event)">
			<div class="form-group">
				<label>Nombre de usuario:</label>
				<input type="text" [(ngModel)]="username" name="username" class="form-control"
placeholder="Introduce tu nombre de usuario">
			</div>
			<div class="form-group">
				<label>Contraseña:</label>
				<input type="password" [(ngModel)]="password" name="password" class="form-control"
placeholder="Introduce tu contraseña">
```

```
                                </div>
                                <div class="form-group">
                                        <button type="submit" class="btn btn-primary">
                                                Iniciar sesión
                                        </button>
                                </div>
                        </form>
                </div>

                <!-- Mensaje que aparece si fallas el login -->
                <div *ngIf="lastTimeFailed">
                        <label class="btn btn-danger">El usuario o la contraseña son incorrectos</label>
                </div>

                <!-- Esto aparece si ya has iniciado sesión -->
                <div class="card-body" *ngIf="isLoggedIn">
                        No hay necesidad de iniciar sesión, ya lo has hecho
                </div>
</div>
```

Aunque parece largo son tres bloques bien diferenciados:

El primero es el formulario de login. Mira en su div. Tiene un parámetro ***ngIf**. Lo que hace es evaluar la expresión indicada para ver si muestra su contenido o no. Pone ***ngIf="!isloggedIn"**, o sea que sólo aparece si NO has iniciado sesión.

El segundo bloque tiene ***ngIf="lastTimeFailed"**, o sea que aparece cuando fallas y esta propiedad se pone a **true**. Es un mensaje de error indicando que has puesto mal las credenciales.

El tercer bloque indica que ya has iniciado sesión y sólo aparece si ya has iniciado sesión.

Pasamos al componente registro que funciona de manera muy similar. Miremos el archivo completo directamente:

**Archivo 6:** register.component.ts

```
import { Component, OnInit } from '@angular/core';
import { User } from '../../User';
import { UserService } from '../../services/user.service';
import { Router } from '@angular/router';

@Component({
        selector: 'app-register',
        templateUrl: './register.component.html',
        styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {
        isLoggedIn: boolean = false;
        pwdNotEqual: boolean = false;
        somethingWentWrongOnServer: boolean = false;

        username: string = '';
        password: string = '';
        passwordRepeat: string = '';
        email: string = '';

        constructor(private userService: UserService, private router: Router) { this.isLoggedIn = UserService.isLoggedIn(); }

        ngOnInit() {
        }

        register(event) {
                event.preventDefault();

                // Si la contraseña repetida está mal lo decimos
                if(this.password != this.passwordRepeat) {
                        console.log("Las contraseñas deben ser iguales!");
                        this.pwdNotEqual = true;
                        return false;
```

```
                }
                this.pwdNotEqual = false;

                // Creamos los datos del usuario a enviar
                const userData:User = {
                        username: this.username,
                        password: this.password,
                        email: this.email
                };

                this.userService.userRegister(userData).subscribe(result => {
                        if(result) {
                                console.log(result);
                                this.username = '';
                                this.email = '';
                                this.password = '';
                                this.passwordRepeat = '';

                                this.somethingWentWrongOnServer = false;
                                this.router.navigate(['/login']);
                        }
                        else {
                                console.log("Algo ha ocurrido al intentar registrarte");
                                this.somethingWentWrongOnServer = true;
                        }
                });
        }
}
```

Todo es muy parecido. En este caso hay más propiedades de formulario y más flags. El método register también es similar:

Si las constraseñas no coinciden muestra un mensaje de error (a través de la flag **pwdnotEqual**) y termina.

El User creado tiene también el email.

Se usa el método **userRegister** del servicio **UserService**.

Cuando termina el registro te redirige a **/login**.

El archivo html también es similar:

**Archivo 7:** register.component.html

```
<label><h2>Registrarse</h2></label>
<div class="card">
        <!-- Esto aparece si no hay una sesión ya iniciada -->
        <div class="card-body" *ngIf="!isLoggedIn">
                <form (submit)="register($event)">
                        <div class="form-group">
                                <label>Nombre de usuario:</label>
                                <input type="text" [(ngModel)]="username" name="username" class="form-control"
placeholder="Introduce tu nombre de usuario">
                        </div>
                        <div class="form-group">
                                <label>E-Mail:</label>
                                <input type="email" [(ngModel)]="email" name="email" class="form-control"
placeholder="Introduce tu email">
                        </div>
                        <div class="form-group">
                                <label>Contraseña:</label>
                                <input type="password" [(ngModel)]="password" name="password" class="form-control"
placeholder="Introduce tu contraseña">
                        </div>
                        <div class="form-group">
                                <label>Confirmar contraseña:</label>
                                <input type="password" [(ngModel)]="passwordRepeat" name="password2" class="form-control"
placeholder="Repite tu contraseña">
                        </div>
                        <div class="form-group">
                                <button type="submit" class="btn btn-primary">
```

```
                                                Iniciar sesión
                                        </button>
                                </div>
                        </form>
                </div>

                <!-- Mensaje que aparece si las contrasenyas no coinciden -->
                <div *ngIf="pwdNotEqual">
                        <label class="btn btn-danger">Las contraseñas introducidas deben ser iguales</label>
                </div>

                <!-- Mensaje que aparece si algo hay un error desconocido proveniente del servidor -->
                <div *ngIf="somethingWentWrongOnServer">
                        <label class="btn btn-danger">Algo ha ido mal al intentar registrarte! Prueba de nuevo.<br>Si el error sigue ocurriendo
prueba más tarde</label>
                </div>

                <!-- Esto aparece si ya has iniciado sesión -->
                <div class="card-body" *ngIf="isLoggedIn">
                        Por favor, sal de tu cuenta antes de registrar otra
                </div>
</div>
```

De nuevo los bloques solo aparecen bajo ciertas circunstancias: El formulario si no hay sesión, el mensaje de que ya hay seión si ya la hay, y los mensajes de error si se activan sus flags.

Ahora tenemos que retocar lo que ya teníamos: La clase **Task** ahora debe incluir el campo **user**.

**Archivo 8:** Task.ts

```
// Creamos la clase Task
export class Task {
        _id?: string;
        title: string;
        isDone: boolean;
        user: string;
}
```

Vamos al servicio tasks y creamos un nuevo método que pida las tareas de un solo usuario.

**Fragmento 9.1:** task.service.ts (Sólo método nuevo)

```
getUserTasks(userId: string) {
        return this.http.get<Task[]>(`${this.apiREST}/${userId}`)
                .pipe(map(res=>res));
}
```

En este método usamos el recurso **api/tasks/:user** que creamos en el API RESTFul antes.

Vamos al archivo **tasks.component.ts** y en el constructor, cuando llamábamos a todas las tareas, lo cambiamos por esto:

**Fragmento 10.1:** tasks.component.ts (Sólo constructor)

```
constructor(private taskService: TaskService) {
        this.isLoggedIn = UserService.isLoggedIn();
        let user = UserService.getCurrentUserID();

        // Solo recuperamos las tarjetas si ha iniciado sesión
        if(this.isLoggedIn) {
                this.taskService.getUserTasks(user).subscribe(tasks => {
                        console.log(tasks);
                        this.tasks = tasks;
                });
        }
}
```

Ahora solo pide las tareas del usuario que ha iniciado sesión, y solo si hay sesión iniciada. Además al tasks.component.html le añadimos un ***ngIf** de manera que si no has iniciado sesión te salga un mensaje diciéndote que inicies sesión:

**Archivo 11:** tasks.component.html

```
<div *ngIf="isLoggedIn">
        <!-- Formulario para crear tareas -->
        <div class="card">
                <div class="card-body">
                        <form (submit)="addTask($event)">
                                <div class="input-group">
                                        <input type="text" [(ngModel)]="titulo" name="titulo" class="form-control"
placeholder="Añade tu tarea">
                                        <span class="input-group-addon">
                                                <button type="submit" class="btn btn-primary">
                                                        Añadir Tarea
                                                </button>
                                        </span>
                                </div>
                        </form>
                </div>
        </div>

        <!-- Lista con todas las tareas existentes para marcar y eliminar -->
        <table class="table text-center table-hover">
                <thead>
                        <tr>
                                <th>Estado</th>
                                <th>Descripción</th>
                                <th>Acciones</th>
                        </tr>
                </thead>
                <tbody>
                        <tr *ngFor="let task of tasks">
                                <td>
                                        <input type="checkbox" [checked]="task.isDone" (click)="updateStatus(task)">
                                </td>
                                <td>
                                        {{task.title}}
                                </td>
                                <td>
                                        <button class="btn btn-danger" (click)="deleteTask(task._id)">
                                                <i class="fa fa-trash"></i>
                                        </button>
                                </td>
                        </tr>
                </tbody>
        </table>
</div>

<div class="card-body" *ngIf="!isLoggedIn">
        Parece que no te has identificado! Inicia sesión para acceder a tus tareas
</div>
```

Bien, ya todos los componentes son funcionales, pero ahora mismo sólo mostramos el componente **tasks** de siempre. Para cambiar esto vamos al archivo **app.component.html**.

En el lugar en el que invocamos la etiqueta del componente **tasks** (**<app-tasks></app-tasks>**) vamos a cambiar la etiqueta por **<router-outlet></router-outlet>**. Aquí irá el componente que digamos en cada momento. Hay que cambiar la barra de navegación. Para cambiar de ruta usamos una etiqueta **<a>** con la propiedad **routerLink="ruta"** y **routerLinkActive="active"**. Nuestra nueva barra de navegación es así:

**Fragmento 12.1:** Barra de navegación

```
<!-- Incorporamos una barra de navegación -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container">
                <a class="navbar-brand" routerLink="/" routerLinkActive="active">{{title}}</a>
                <a class="navbar-brand" routerLink="/tasks" routerLinkActive="active">Mis tareas</a>
                <a class="navbar-brand" routerLink="/login" routerLinkActive="active">Login</a>
                <a class="navbar-brand" routerLink="/register" routerLinkActive="active">Register</a>
                <a class="navbar-brand" href='#' (click)="logout()">Logout</a>
        </div>
```

```
</nav>
```

Con todo el archivo queda así:

**Archivo 12:** app.component.html

```
<!--Punto inicial de entrada al módulo de mi app toDo.-->

<!-- Incorporamos una barra de navegación -->
<nav class="navbar navbar-expand-lg navbar-light bg-light">
	<div class="container">
		<a class="navbar-brand" routerLink="/" routerLinkActive="active">{{title}}</a>
		<a class="navbar-brand" routerLink="/tasks" routerLinkActive="active">Mis tareas</a>
		<a class="navbar-brand" routerLink="/login" routerLinkActive="active">Login</a>
		<a class="navbar-brand" routerLink="/register" routerLinkActive="active">Register</a>
		<a class="navbar-brand" href='#' (click)="logout()">Logout</a>
	</div>
</nav>

<!-- Creamos un contenedor para nuestro módulo Tasks -->
<div class="container">
	<div class="row">
		<img width="75" alt="Angular Logo"
src="">
		<div class="col-md-10">
			<router-outlet></router-outlet>
		</div>
	</div>
</div>
```

Fíjate que una de las etiquetas **<a>**, logout, lleva a una función **logout**. Hay que implementarla en **app.component.ts**. Importamos el módulo **UserService** y escribimos

**Fragmento 13.1:** Logout

```
logout() {
	console.log("Se ha cerrado la sesión");
	UserService.userLogout();
}
```

El archivo queda así:

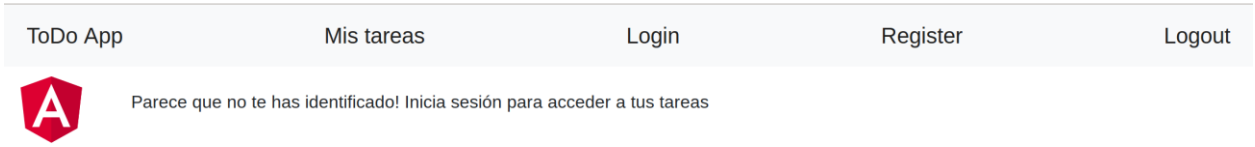
**Archivo 13:** app.component.ts

```
import { Component } from '@angular/core';
import { UserService } from './services/user.service';

@Component({
	selector: 'app-root',
	templateUrl: './app.component.html',
	styleUrls: ['./app.component.css']
})
export class AppComponent {
	title = 'ToDo App';

	logout() {
		console.log("Se ha cerrado la sesión");
		UserService.userLogout();
	}
}
```

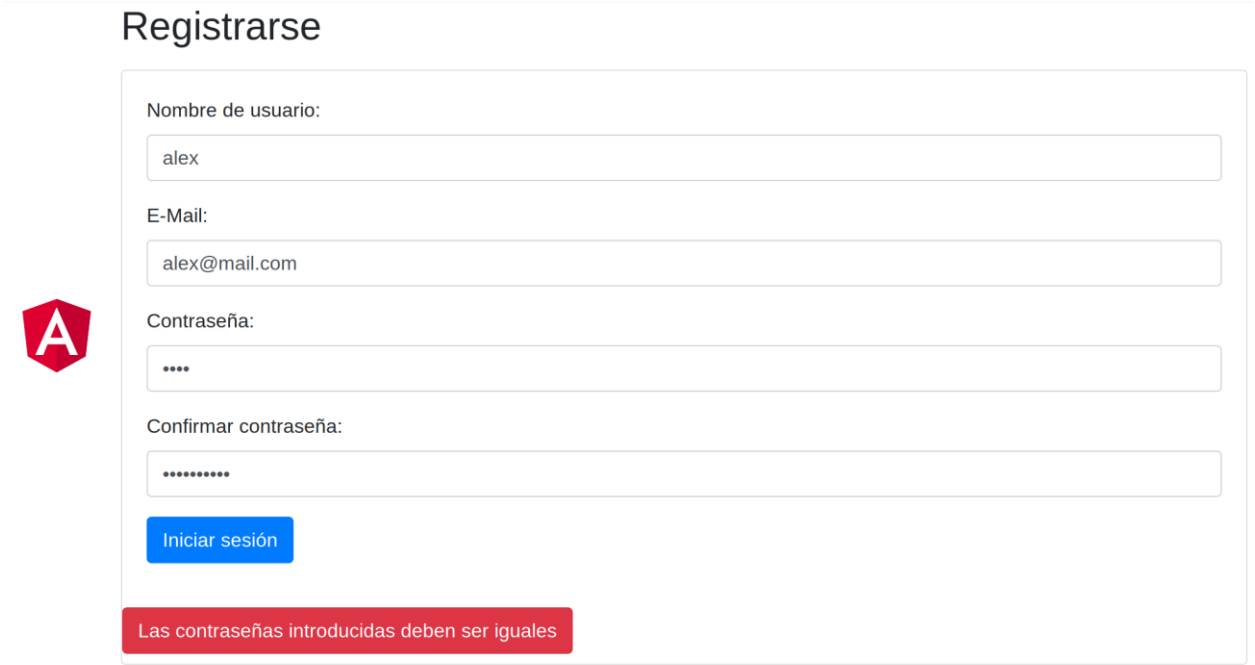
Con todo esto, la aplicación está terminada y procedemos a probarla.

## Pruebas de funcionamiento

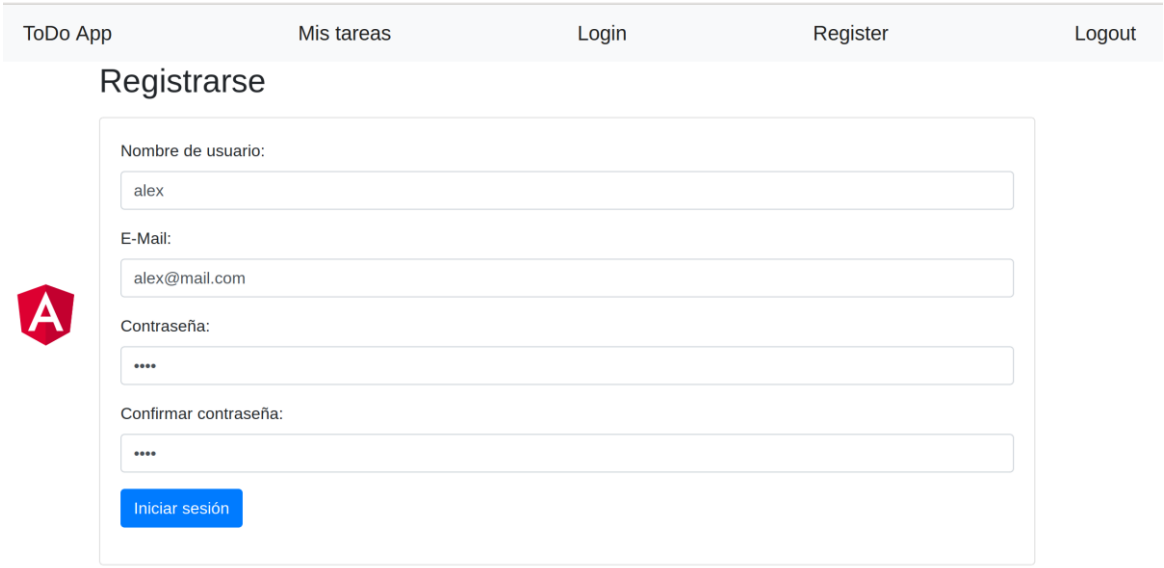
Lanzamos la aplicación y entramos en **localhost:4200**. Esto es lo que vemos



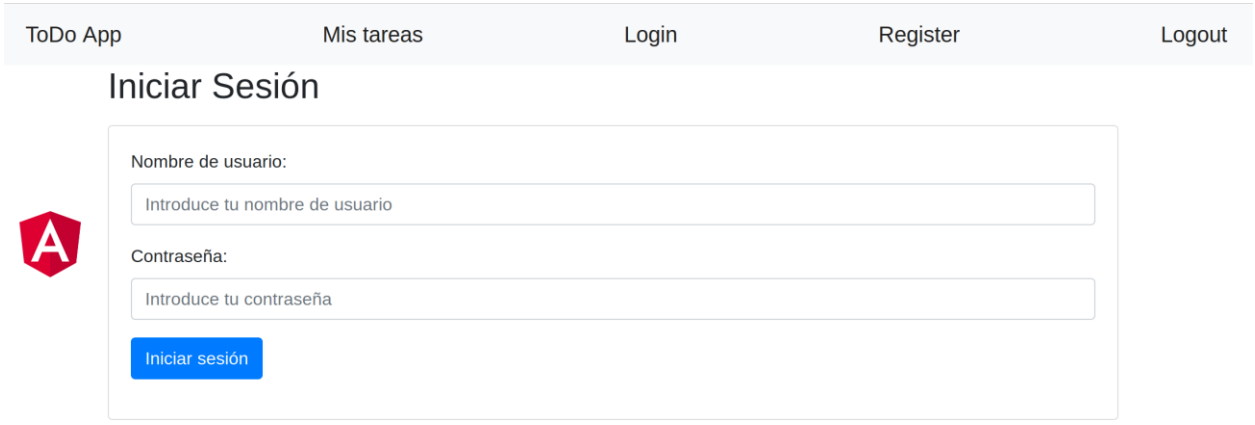

Ahora le damos a registrarnos y creamos un nuevo usuario, pero nos equivocamos al repetir la contraseña



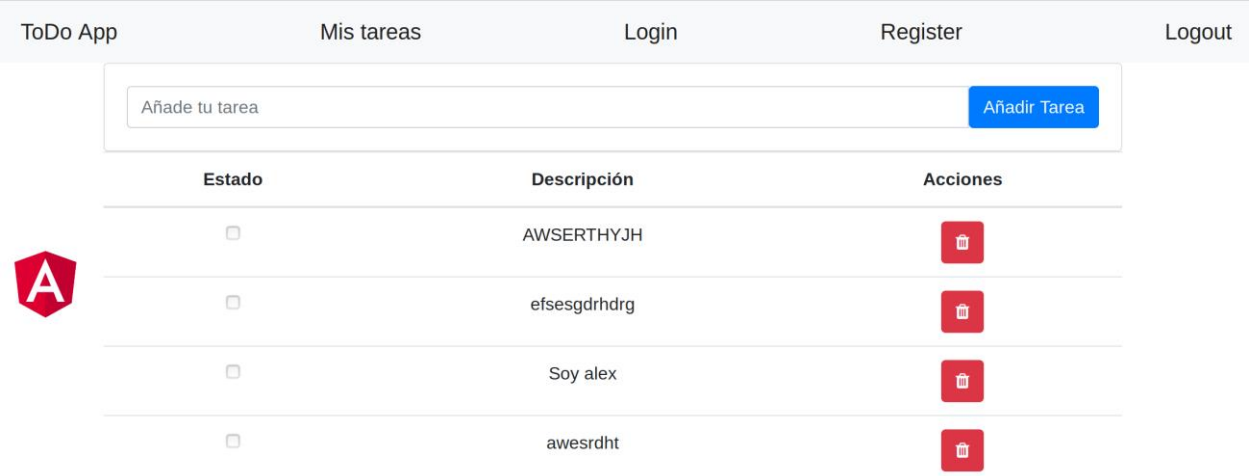
Las corregimos y rellenamos bien el formulario:




Ahora iniciamos sesión dándole a login



Y creamos un puñado de tareas en nuestra nueva cuenta




Luego le damos a Logout y creamos una nueva cuenta. Las tareas de la primera cuenta ya no están. Creemos algun par más.


Por último salimos de la sesión otra vez y volvemos a iniciarla con la primera cuenta. Las tareas siguen ahí.

## Usando GIT

Vamos a subir este proyecto a GIT. Creamos una cuenta en GitHUB y creamos un repositorio nuevo. Creamos un .gitignore con la ruta de la carpeta node_modules (no queremos que la copie). Luego buscamos la dirección para clonar el repositorio. En una carpeta de nuestra elección ejecutamos

```
$ git clone https://github.com/AlexMFGIT/SD_P3_Opcional.git
$ git init
```

Ahora nos identificamos con

```
$ git config --global user.name AlexMFGIT
$ git config --global user.email alex.macia.fiteni2@gmail.com
```

Luego arrastramos aquí el estado actual del proyecto. Después añadimos los cambios, en este caso:

```
$ git add MemoriaOpcional/
$ git add node/
```

Y hacemos commit.

```
$ git commit MemoriaOpcional/ node/
```

Nos pedirá una descripción.

```
$ git push origin master
```

Y nos pedirá el usuario y constraseña de GitHub. Ya tienes tu proyecto en GIT.

# Bibliografía

Las referencias que se han utilizado son las siguientes:

**Rutas en Angular**
https://angular.io/guide/router

**Uso de Angular**
https://angular.io/api/common/NgIf

**Querys de find a mongo en Node**
https://flaviocopes.com/node-mongodb/