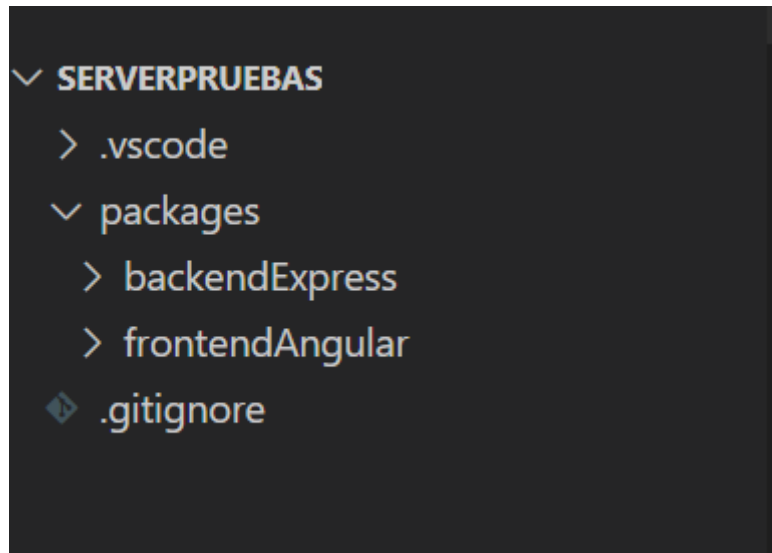


Documentación funcionamiento Backend y Frontend del proyecto de prueba

Documentación breve con imágenes del servidor en funcionamiento. El directorio del proyecto es el siguiente:



Contiene una carpeta *packages* la cual contiene una subcarpeta con la parte back y otra con la parte frontal del proyecto. He eliminado las carpetas node-modules para que pese mucho menos el proyecto, por lo que, habrá que hacer *npm i* dentro de ambas carpetas.

El backend es un servidor **Express** el cual tiene asociado una base de datos **PostgreSQL**. He definido las siguientes peticiones:

1. Get

1.1 Obtener todos los usuarios de la BD.

En mi caso he decidido crear instancias de personas, concretamente, de amigos. Haciendo la siguiente petición obtenemos todos los que se han dado de alta en la BD:

http://localhost:3000/amigos

GET http://localhost:3000/amigos Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (9) Test Results 200 OK 18 ms 1.27 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "nick": "fuensanta4",
4     "firstname": "fuensanta",
5     "lastname": "lopez",
6     "password": "uu",
7     "age": 28
8   },
9   {
10    "nick": "juanHD",
11    "firstname": "Juan",
12    "lastname": "Hernandez",
13    "password": "pera",
14    "age": 13
15  },
16  {
17    "nick": "manolon4",
```

Obtenemos en formato JSON el listado de todos los usuarios, con sus respectivos atributos en la BD.

1.2 Filtrar usuarios por algún criterio

Con la dirección /amigosmayores/X podemos filtrar y mostrar exclusivamente los amigos que tengan una edad superior a X.

http://localhost:3000/amigosmayores/20

GET http://localhost:3000/amigosmayores/20 Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (9) Test Results Status: 200 OK Time: 43 ms Size: 449 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "nick": "fuensanta4",
4     "firstname": "fuensanta",
5     "lastname": "lopez",
6     "age": 28
7   },
8   {
9     "nick": "manolon4",
10    "firstname": "manoloo",
11    "lastname": "camión",
12    "age": 45
13  }
14 ]
```

1.3 Filtrar por nombre

Si en la url /amigos/ especificamos el nick de un usuario se realizará una búsqueda de dicho amigo.

GET [Send](#)

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (9) Test Results Status: 200 OK Time: 601 ms Size: 389 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2     
3   "nick": "juanHD",  
4   "firstname": "Juan",  
5   "lastname": "Hernandez",  
6   "password": "pera",  
7   "age": 13  
8 }  
9 }
```

En caso de realizar la búsqueda de un usuario inexistente, se devolverá el error 404.

GET [Send](#)

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

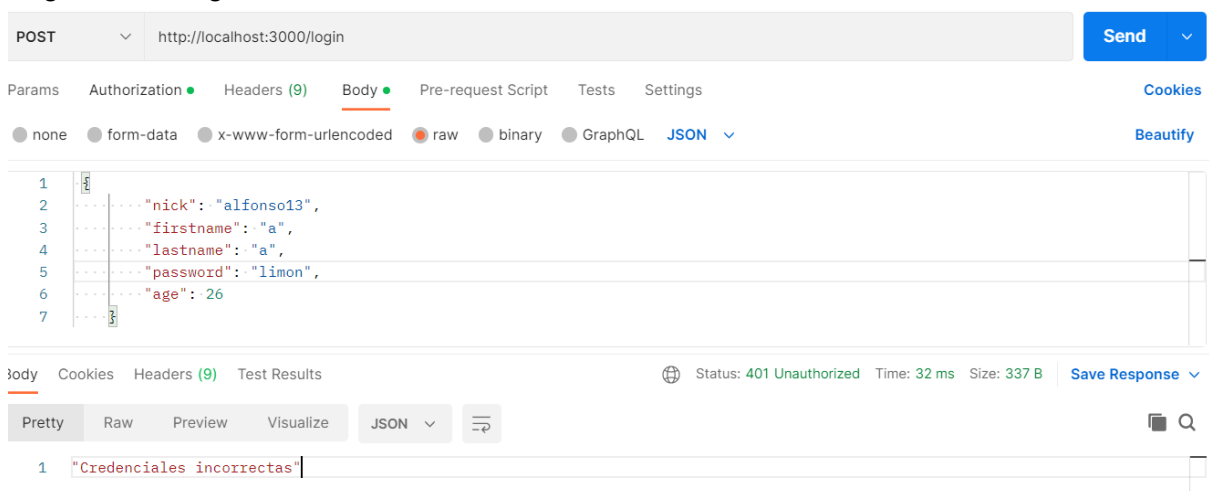
Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (9) Test Results Status: 404 Not Found Time: 26 ms Size: 368 B Save Response

Pretty Raw Preview Visualize HTML

```
1 <b>404.</b> No existe ningún amigo con el nick juanHD100 todavía!
```



2.2 Crear un nuevo amigo

Para poder realizar esta petición a la dirección /crearamigo previamente pasa por la función `verificarToken`.

```
/** 6. Insertando un nuevo usuario con JSON, deberemos de habernos autenticado previamente */
app.post("/crearamigo", verificarToken, (req, res) => {
  // Obtenemos el objeto persona del JSON
  let persona = new Persona(
    req.body.nick,
    req.body.firstname,
    req.body.lastname,
    req.body.password,
    req.body.age
  );

  console.log("Intentando crear nueva persona: " + persona.nick);

  // Definir la consulta SQL preparada
  let consulta = {
    text: "INSERT INTO amigos (nick, firstName, lastName, password, age) VALUES ($1, $2, $3, $4, $5)",
    values: [
      persona.nick,
      persona.nombre,
```

Esta función comprueba la cabecera “authorization”. En primer lugar se comprueba si la petición ha proporcionado algún token de autenticación:

The screenshot shows a REST client interface with the following details:

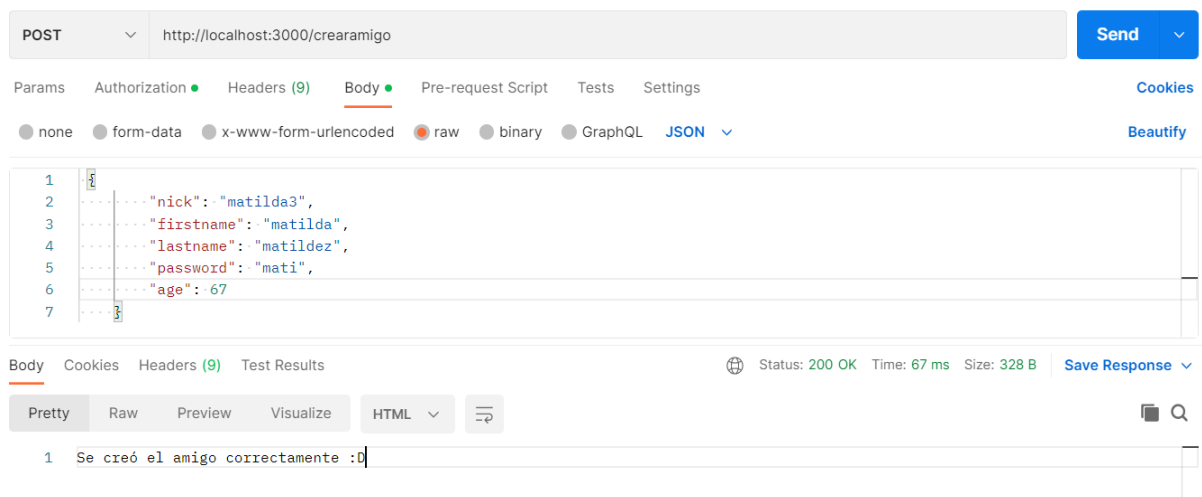
- Method:** POST
- URL:** http://localhost:3000/crearamigo...
- Authorization:** Bearer To... (selected)
- Token:** Token
- Status:** 401 Unauthorized
- Time:** 16 ms
- Size:** 350 B
- Response:** 1 No se proporcionó un token de autenticación.

A continuación si el token es válido:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:3000/crearamigo...
- Authorization:** Bearer To... (selected)
- Token:** token no valido
- Status:** 400 Bad Request
- Time:** 21 ms
- Size:** 320 B
- Response:** 1 Token no válido.

Si supera todas estas comprobaciones, entonces se crea el amigo:



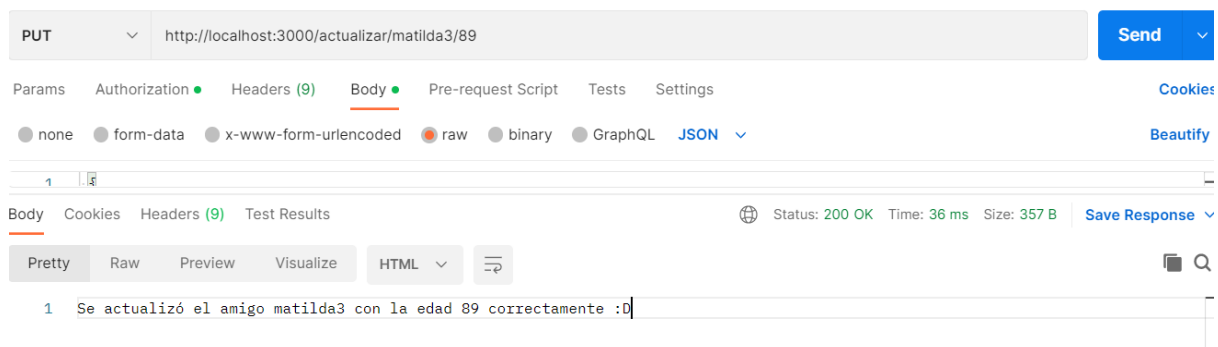
El amigo se crea según los parámetros pasados en el body de la petición en formato JSON.

3. PUT

He definido dos rutas para la actualización de valores en la BD. La primera permite modificar la edad de los usuarios y la segunda cualquier campo, a excepción del nick.

3.1 Actualizar la edad de un amigo

Realizando una petición put a `/actualizar/nick/nuevaedad` podemos actualizar las edades de los distintos usuarios de la BD.



Si se intenta actualizar un usuario que no existe en la BD devolverá el error 404.

3.2 Actualizar varios campos

Si realizamos la petición a `/actualizar/nick` se comprobarán los campos en el cuerpo de la petición los cuales deberán estar especificados en JSON.

The screenshot shows a REST client interface with a PUT request to `http://localhost:3000/actualizar/matilda3`. The request body is a JSON object: `{ "nick": "matilda3", "firstname": "matilda juliana", "lastname": "matildez julianez", "password": "matijuli", "age": 80 }`. The response status is 200 OK, with a time of 74 ms and a size of 405 B. The response body is: `1 Se actualizó el amigo matilda3`, `2 Nuevo nombre: matilda juliana`, `3 Nuevo apellido: matildez julianez`, `4 Nueva edad: 80`.

Esto nos permite actualizar varios campos en una única petición. Se requiere autenticación para esta petición.

4. DELETE

4.1 Borrar un usuario de la BD

Para eliminar un amigo de la tabla de la BD habrá que realizar una petición DELETE a la dirección `/borrar/nick`. Donde `nick` será el `nick` del usuario que queremos eliminar. Esta petición requerirá autenticación de usuario:

The screenshot shows a REST client interface with a DELETE request to `http://localhost:3000/borrar/matilda3`. The request is authenticated with a Bearer token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImFsZm9uc28xMyIsInBhc3MiOiJuYXJhbGphbiwifWF0IjoxNjc3MjI5NjM2LCJleHAiOiE2NzcyMjk5MzZ9.-8KSouc4ohZQ0ny3_YfCiPyl n8knT4CXbW6un7dRGQs`. The response status is 200 OK, with a time of 28 ms and a size of 323 B. The response body is: `1 Se eliminó el amigo matilda3`.

Si se especifica un `nick` de un amigo que no existe se devuelve error.

The screenshot shows a REST client interface with a DELETE request to `http://localhost:3000/borrar/matilda5`. The request is authenticated with the same Bearer token. The response status is 404 Not Found, with a time of 31 ms and a size of 367 B. The response body is: `1 404. No existe ningún amigo con el nick matilda5 todavía!`.

Cualquier petición a una ruta de las no definidas aquí, devolverá un error 404.

Mencionar, que algunos parámetros son pasados como variables de entorno gracias al paquete **dotenv**.

```
// Parametros de conexion a la base de datos pasamos por variables de entorno

let client = new Client({
  user: process.env.NOMBRE,
  host: process.env.HOST,
  database: process.env.DATABASE,
  password: process.env.PASS,
  port: process.env.PORT,
});
```

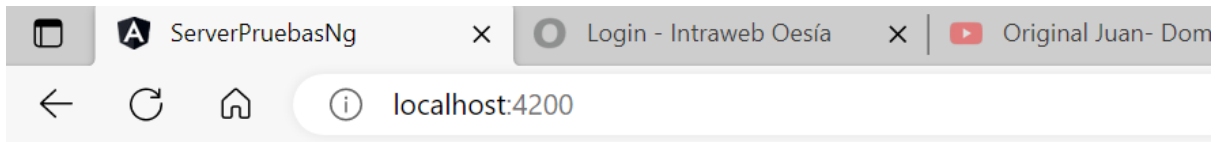
```
SAMPLE_DATA=dotenv is working!
NOMBRE=postgres
HOST=localhost
DATABASE=bdserver
PASS=alfonso
PORT=5432
JWT_EXPIRATION=5m
SECRET_KEY=alfonso
```

El frontend es un servidor el cual incorpora **Angular**. Para que ambos se comuniquen y el back pueda aceptar peticiones desde el front he usado **Cors**.

```
// CORS para permitir peticiones desde el front
const corsOptions = {
  origin: "http://localhost:4200",
  optionsSuccessStatus: 200,
};

app.use(cors(corsOptions));
```

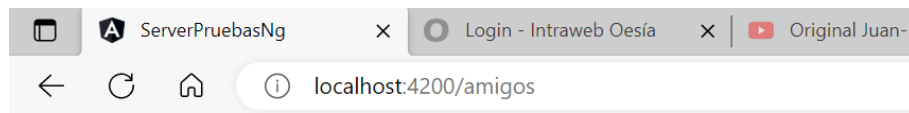
El front es el siguiente:



Angular practica

- [Listar todos los amigos](#)
- [Listar amigos mayores de edad](#)
- [Buscar amigo](#)
- [Login usuario](#)
- [Crear amigo](#)
- [Actualizar edad de un amigo](#)
- [Borrar amigo](#)

Se muestran las distintas operaciones posibles. Por ejemplo, la de listar todos los amigos:

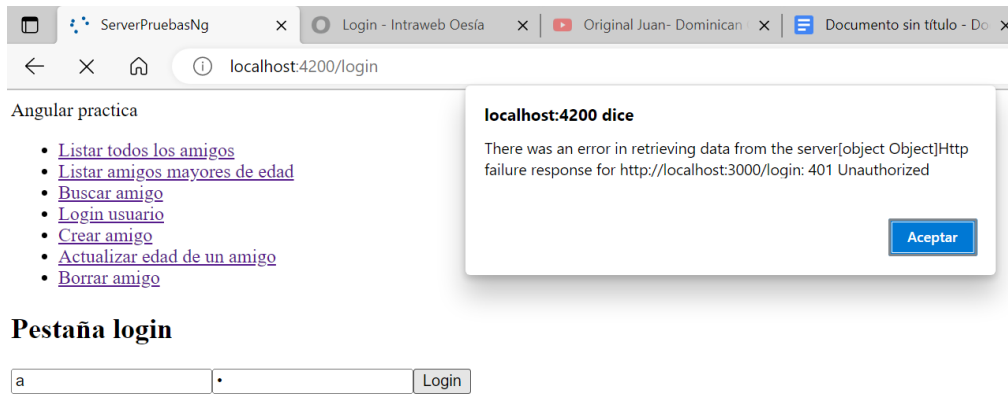


Angular practica

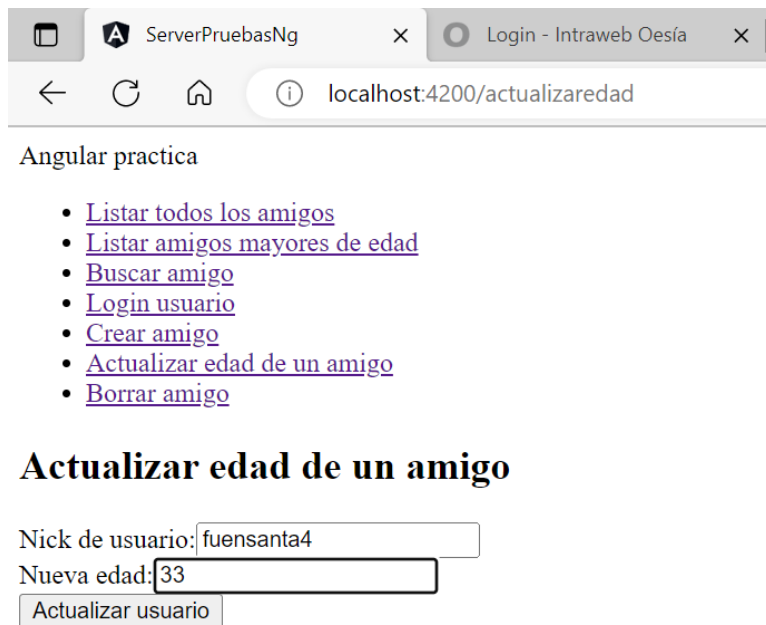
- [Listar todos los amigos](#)
- [Listar amigos mayores de edad](#)
- [Buscar amigo](#)
- [Login usuario](#)
- [Crear amigo](#)
- [Actualizar edad de un amigo](#)
- [Borrar amigo](#)

Todos los amigos

Nick	Nombre	Apellido	Edad
fuensanta4	fuensanta	lopez	28
juanHD	Juan	Hernandez	13
alfonso13	Alfonso	Lopez	4
manolon7	manoloo	camión	10
manolon8	manoloo	camión	10
juan	juan	lopez	4
22	2	2	2



Se muestran también los errores cuando las peticiones no son correctas. La apariencia del front es la siguiente:



Actualizar edad de un amigo

Nick de usuario:

Nueva edad:



Crea un nuevo amigo

Nick de usuario:

Nombre:

Apellido:

Contraseña:

Edad:

← × 🏠 ⓘ localhost:4200/borrar

Angular practica

- [Listar todos los amigos](#)
- [Listar amigos mayores de edad](#)
- [Buscar amigo](#)
- [Login usuario](#)
- [Crear amigo](#)
- [Actualizar edad de un amigo](#)
- [Borrar amigo](#)

localhost:4200 dice

There was an error in retrieving data from the server[object Object]Http failure response for http://localhost:3000/borrar/sinpermiso: 401 Unauthorized

Aceptar

Eliminar amigo

Nick de usuario:

← ↻ 🏠 ⓘ localhost:4200/mayoresedad

Angular practica

- [Listar todos los amigos](#)
- [Listar amigos mayores de edad](#)
- [Buscar amigo](#)
- [Login usuario](#)
- [Crear amigo](#)
- [Actualizar edad de un amigo](#)
- [Borrar amigo](#)

Todos los amigos mayores de edad

Nick	Nombre	Apellido	Edad
manolon4	manoloo	camión	45
fuensanta4	fuensanta	lopez	67

Todos los logs de las distintas peticiones se muestran en la consola del servidor Backend:

```
alfonso@PGO23011206012: /mnt/c/Users/extalopezr/OneDrive - OESIA NETWORKS SOCIEDAD LIMITADA/Escritorio/serverPruebas/pac...
Se actualizó el amigo correctamente
Token no válido.
Intentando autenticar usuario: alfonso13 con contraseña: naranja
Generando token JWT con tiempo de validez de: 5m
Token generado
Intentando borrar persona con nick: matilda3
Se eliminó el amigo correctamente
Intentando borrar persona con nick: matilda5
No existe ningún amigo con ese nick todavía
Recuperando todos los amigos
Devuelve todos los amigos
Recuperando todos los amigos
Devuelve todos los amigos
Buscando amigo con nick noexiste
No existe ningún amigo con ese nick todavía
Buscando amigo con nick noexiste
No existe ningún amigo con ese nick todavía
Buscando amigo con nick a
No existe ningún amigo con ese nick todavía
Recuperando amigos mayores de edad
Devolviendo resultados
Intentando autenticar usuario: a con contraseña: a
Credenciales incorrectas
Recuperando todos los amigos
Devuelve todos los amigos
Intentando actualizar persona con nick: fuensanta4
Se actualizó el amigo correctamente
Recuperando todos los amigos
```

Aclaraciones: la aplicación contiene graves fallos de seguridad como revelar las contraseñas de todos los usuarios cuando se realiza el listado de todos estos. Sin embargo, en este entorno de prueba no ha sido lo que se ha tenido en cuenta principalmente durante el desarrollo, sino que más bien sea funcional.