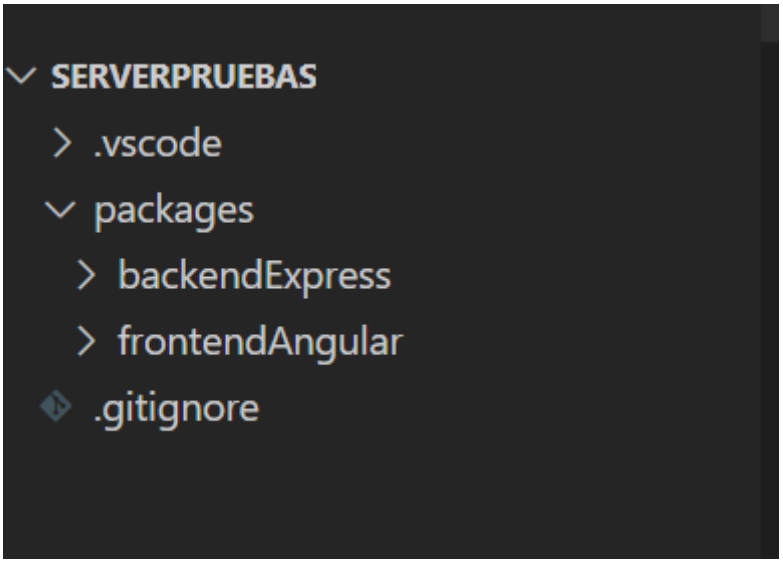
Documentación funcionamiento Backend y Frontend del proyecto de prueba

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



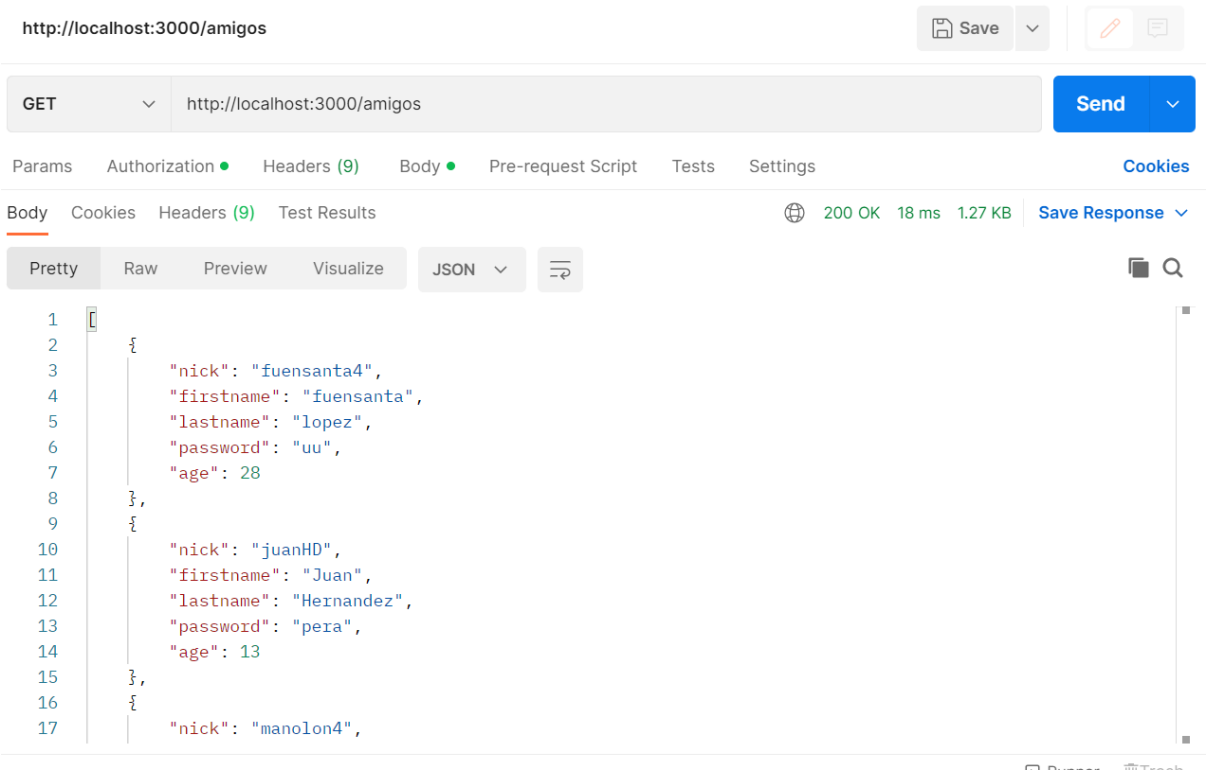
Contiene una carpeta *packages* la cual contiene una subcarpeta con la parte back y otra con la parte frontal del proyecto. He eliminado las carpetas node-modules para que pese mucho menos el proyecto, por lo que, habrá que hacer *npm i* dentro de ambas carpetas.

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.

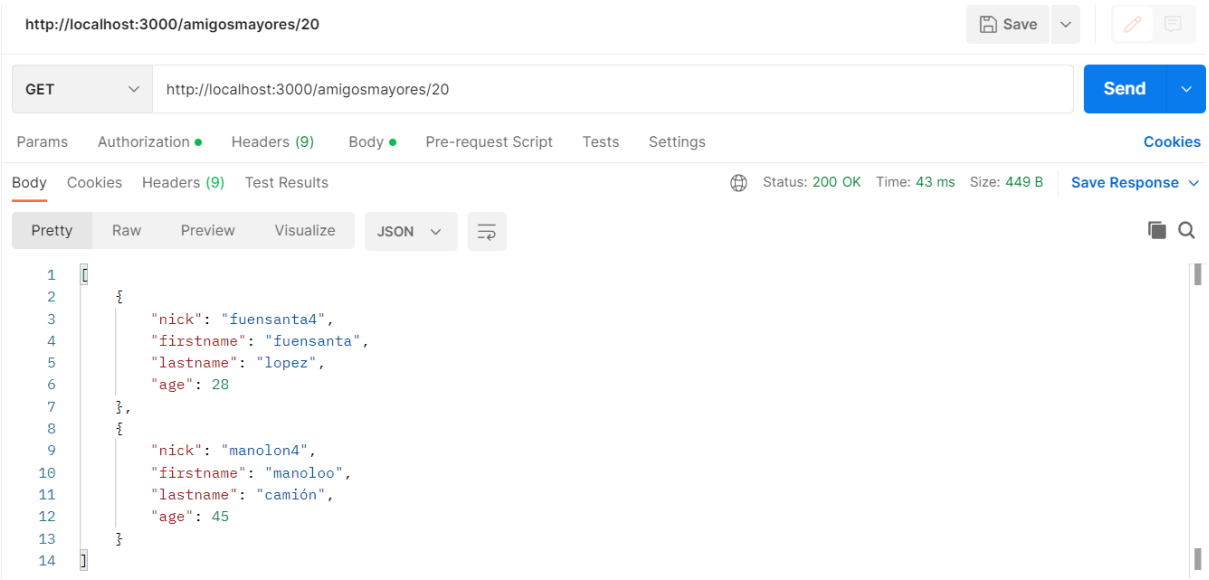
En mi caso he decidido crear instancias de personas, concretamente, de amigos. Haciendo la siguiente petición obtenemos todos los que se han dado de alta en la BD:



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

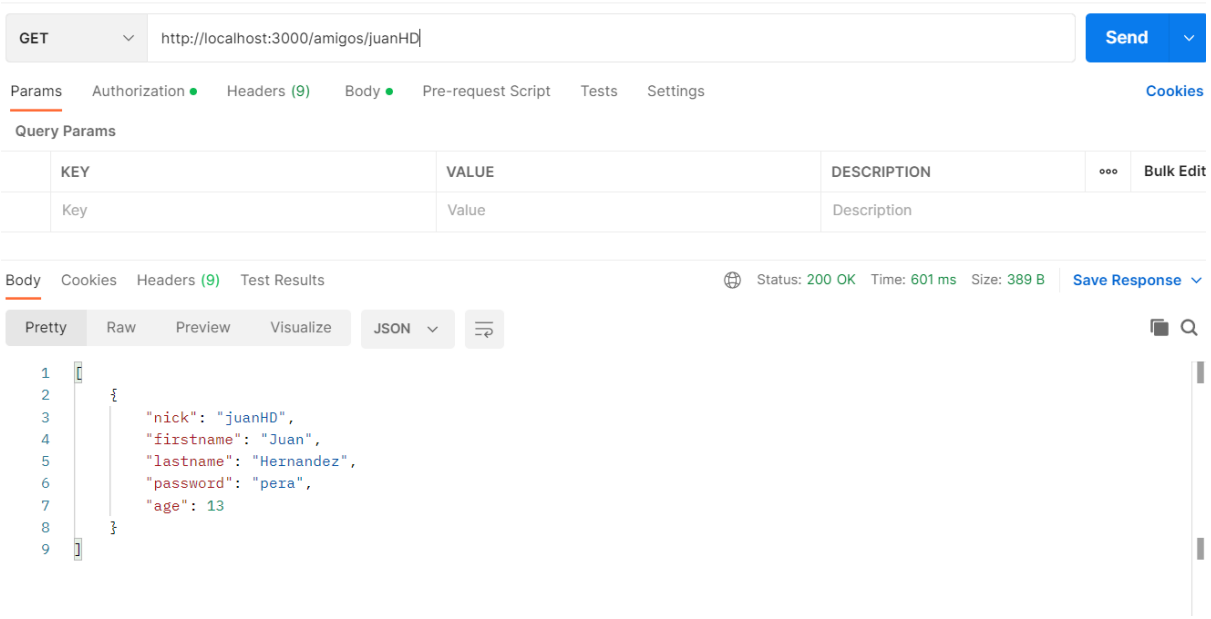
1.2 Filtrar usuarios por algún criterio

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

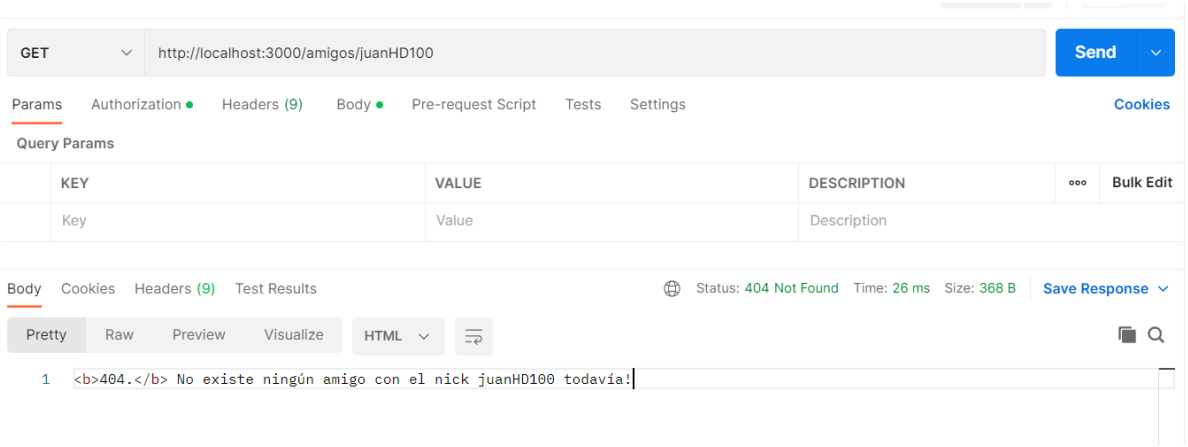


1.3 Filtrar por nombre

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



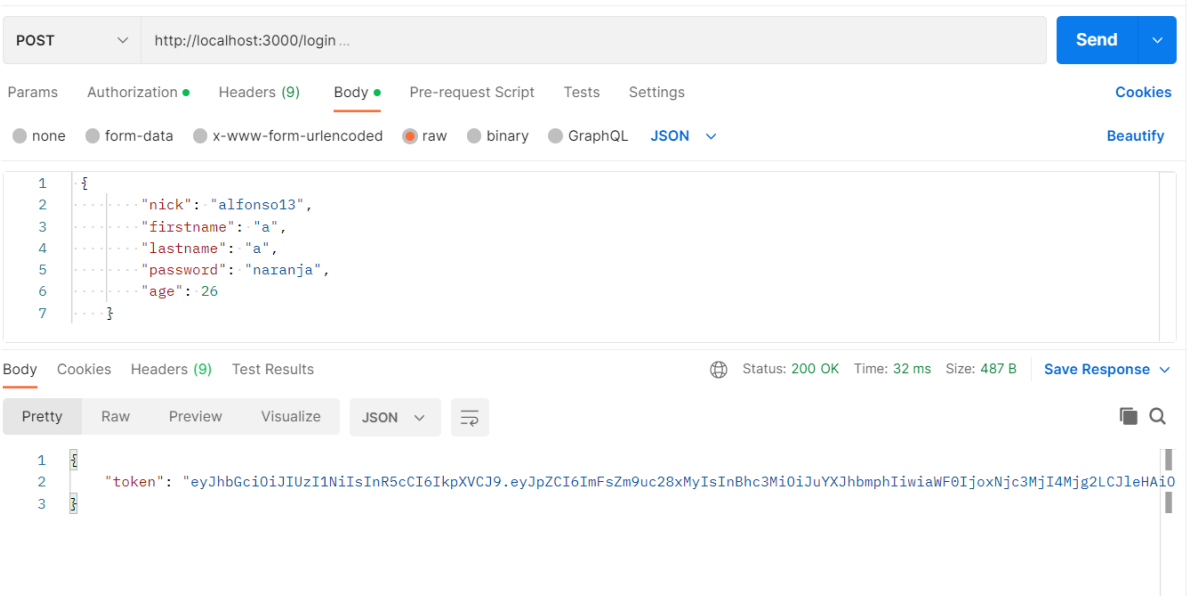
En caso de realizar la búsqueda de un usuario inexistente, se devolverá el error 404.



2. POST

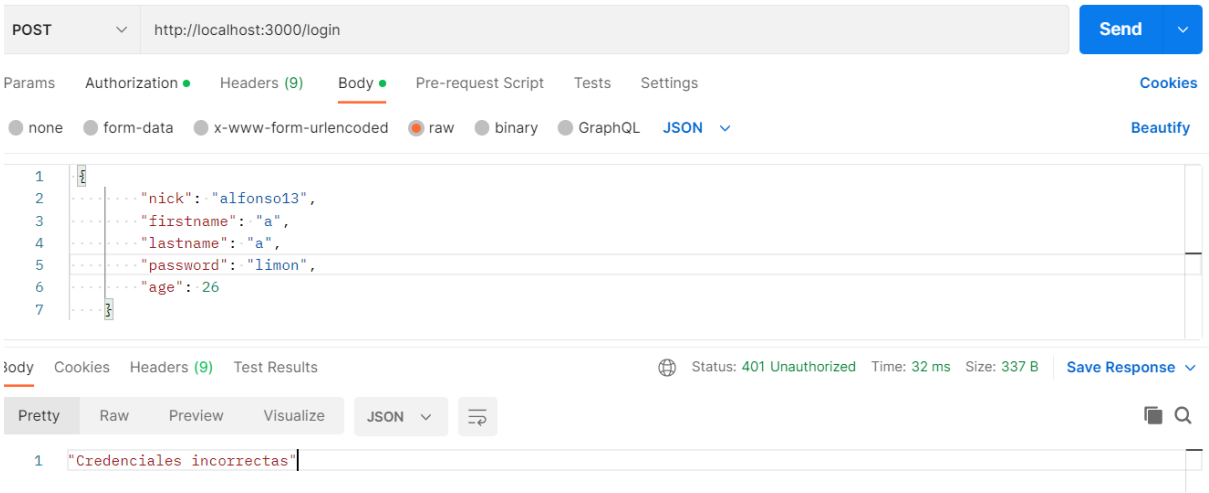
2.1 Login usuarios

Para realizar el login de un usuario y autenticarlo se usará una petición post a la ruta /login. Las credenciales del usuario se pasarán en el cuerpo de la petición en formato JSON.



Si son correctas, se nos devolverá un token JWT de autenticación de dicho usuario. Este lo podremos copiar y añadir a la cabecera de autenticación para poder realizar peticiones que precisen autenticación.

Si el usuario no existe o la contraseña no es la correcta se devolverá el siguiente error y no se generará ningún token:

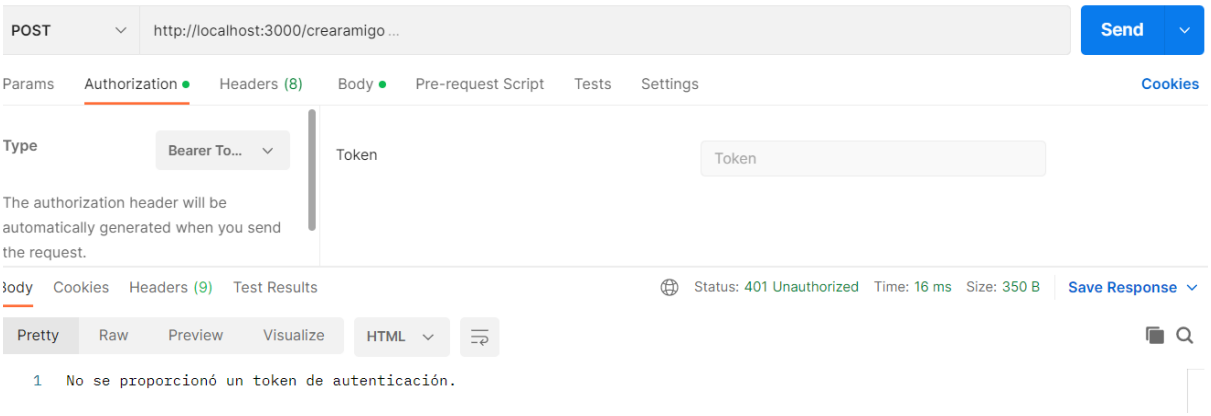


2.2 Crear un nuevo amigo

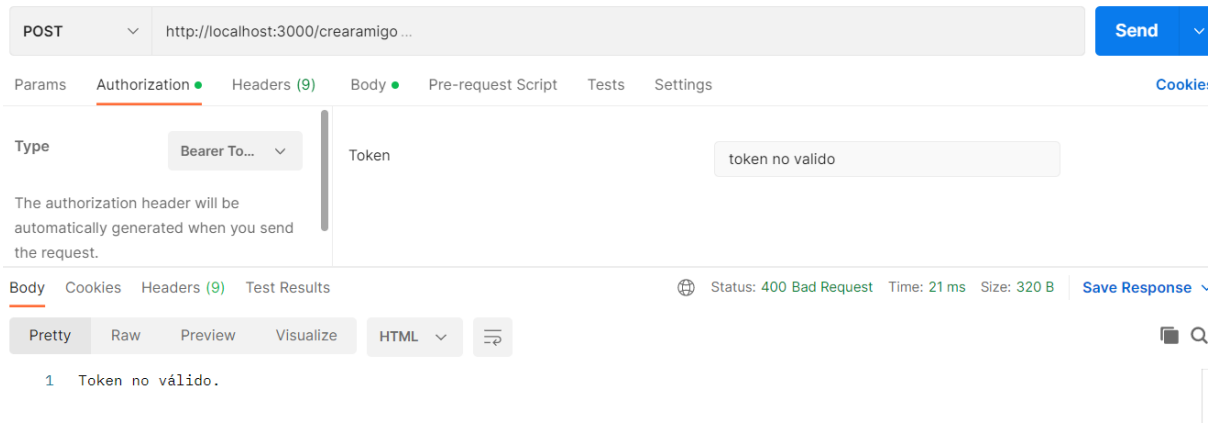
Para poder realizar esta petición a la dirección /crearamigo previamente pasa por la función *verificarToken*.

```
/** 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,
```

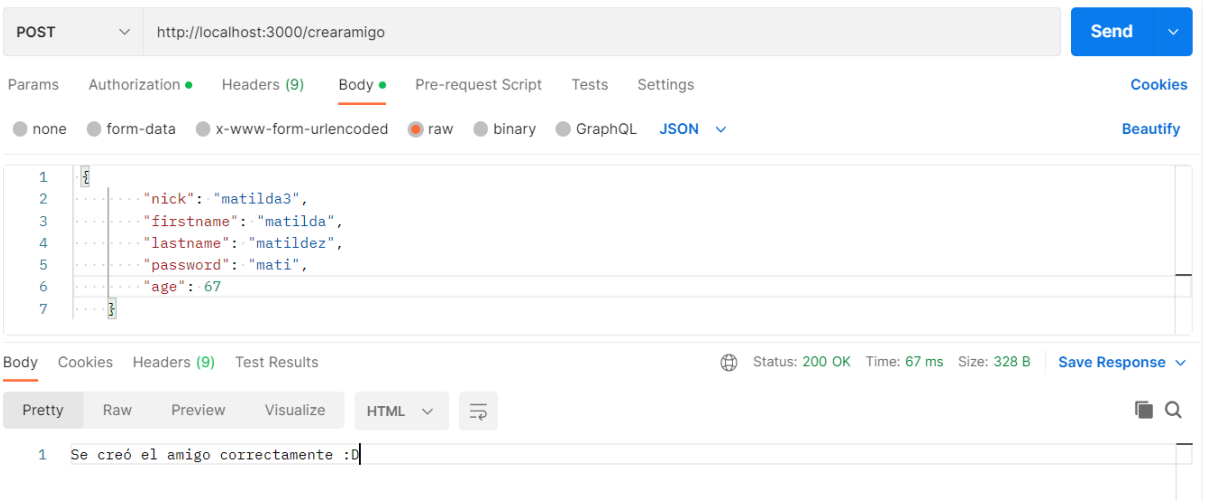
Esta función comprueba la cabecera “authorization”. En primer lugar se comprueba si la petición ha proporcionado algún token de autenticación:



A continuación si el token es válido:



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



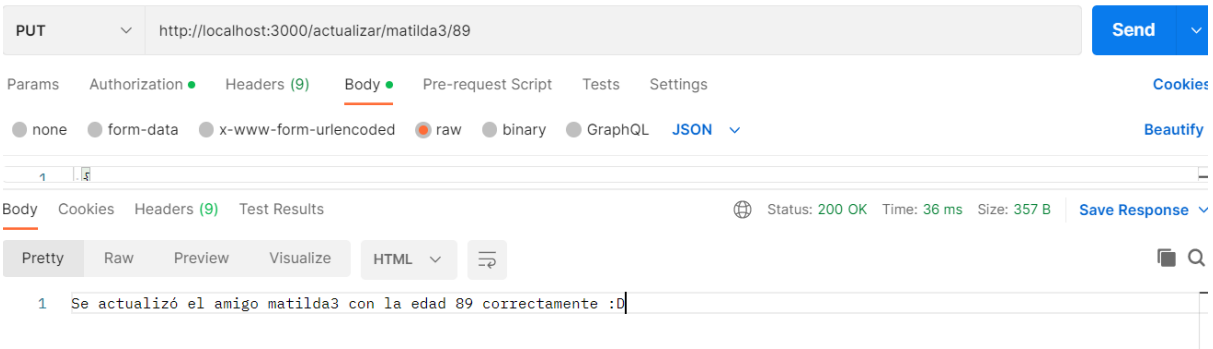
El amigo se crea según los parámetros pasados en el body de la petición en formato JSON.

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

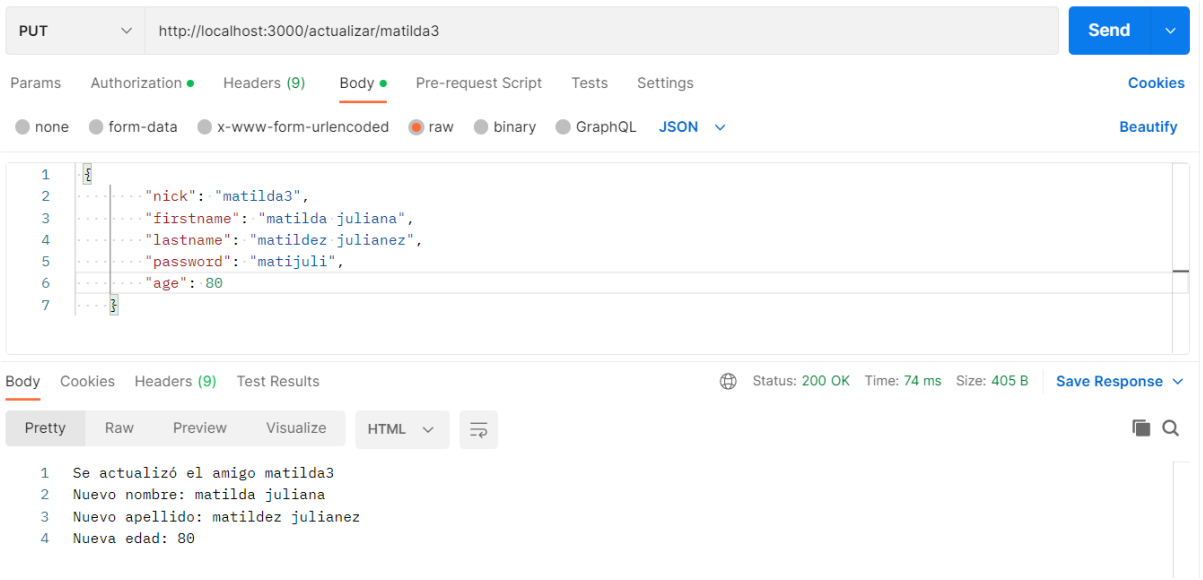
Realizando una petición put a /actualizar/nick/nuevaedad podemos actualizar las edades de los distintos usuarios de la BD.



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.

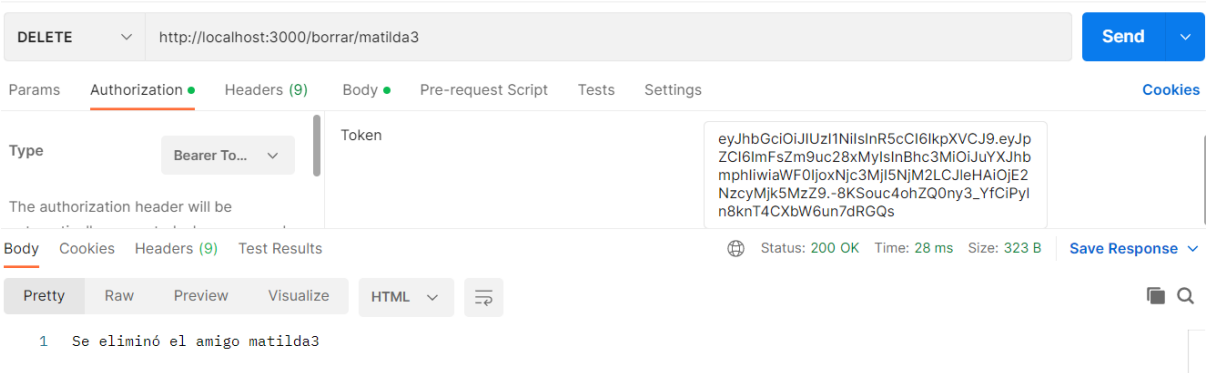


Esto nos permite actualizar varios campos en una única petición. Se requiere autenticación para esta petición.

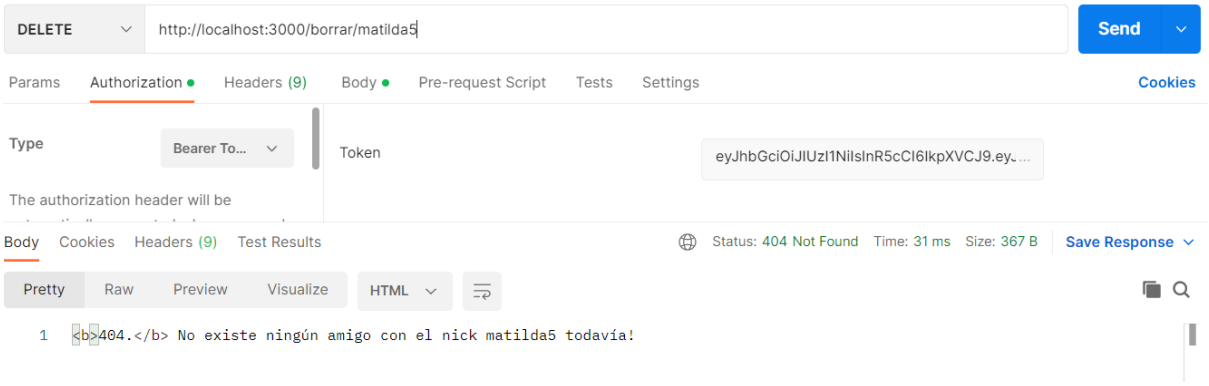
4. DELETE

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:



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



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**.

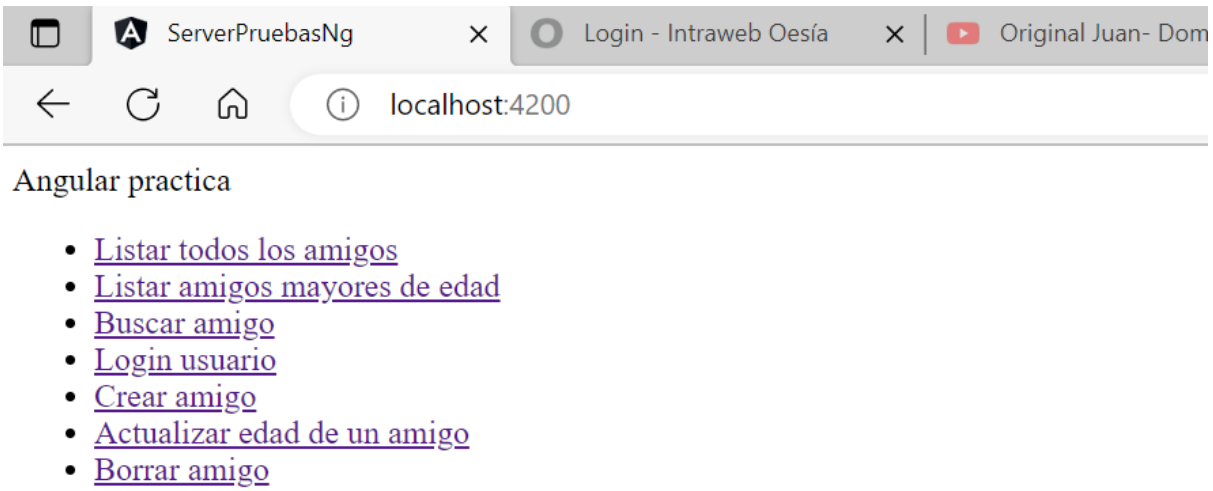
```
// Parametros de conexion a la base de datos pasamos por variables de entorno

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
```

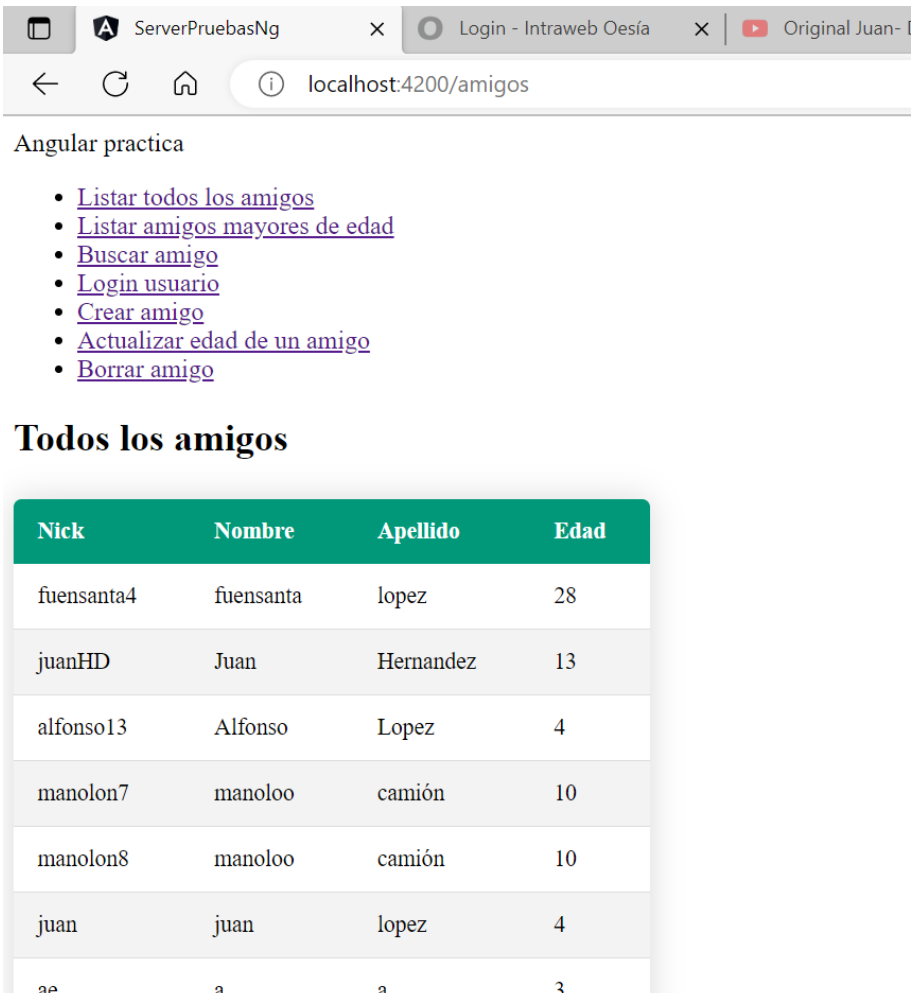
El frontend es un servidor el cual incorpora **Angular**. Para que ambos se comuniquen y el back pueda aceptar peticiones desde el front he usado **Cors**.

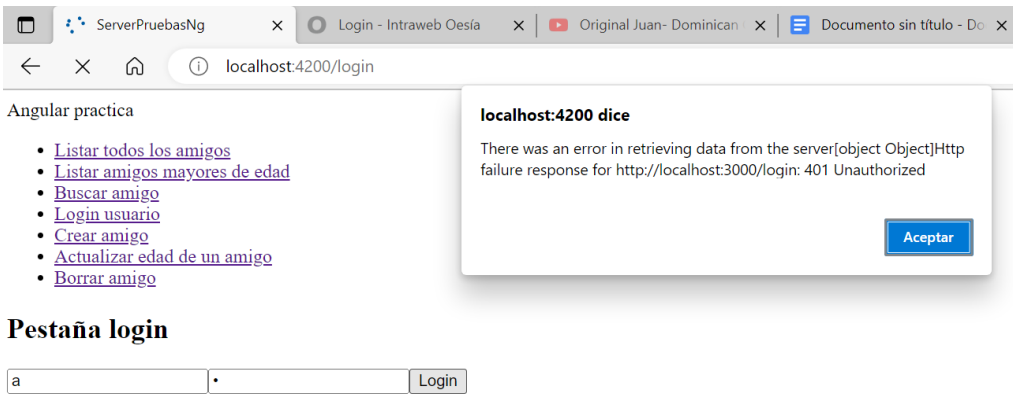
```
// CORS para permitir peticiones desde el front
const corsOptions = {
  origin: "http://localhost:4200",
  optionsSuccessStatus: 200,
};

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

El front es el siguiente:

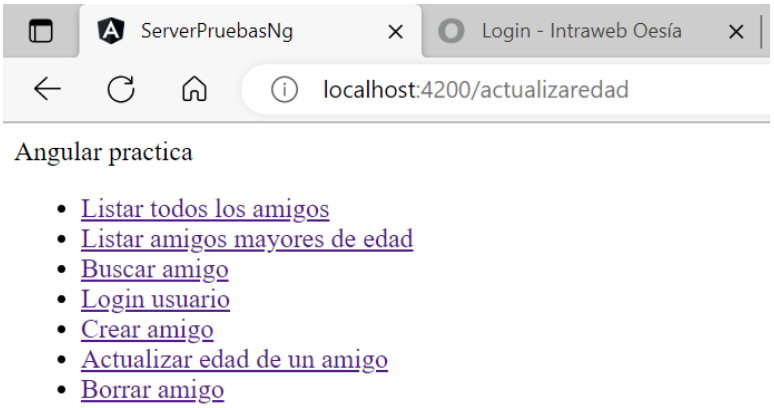


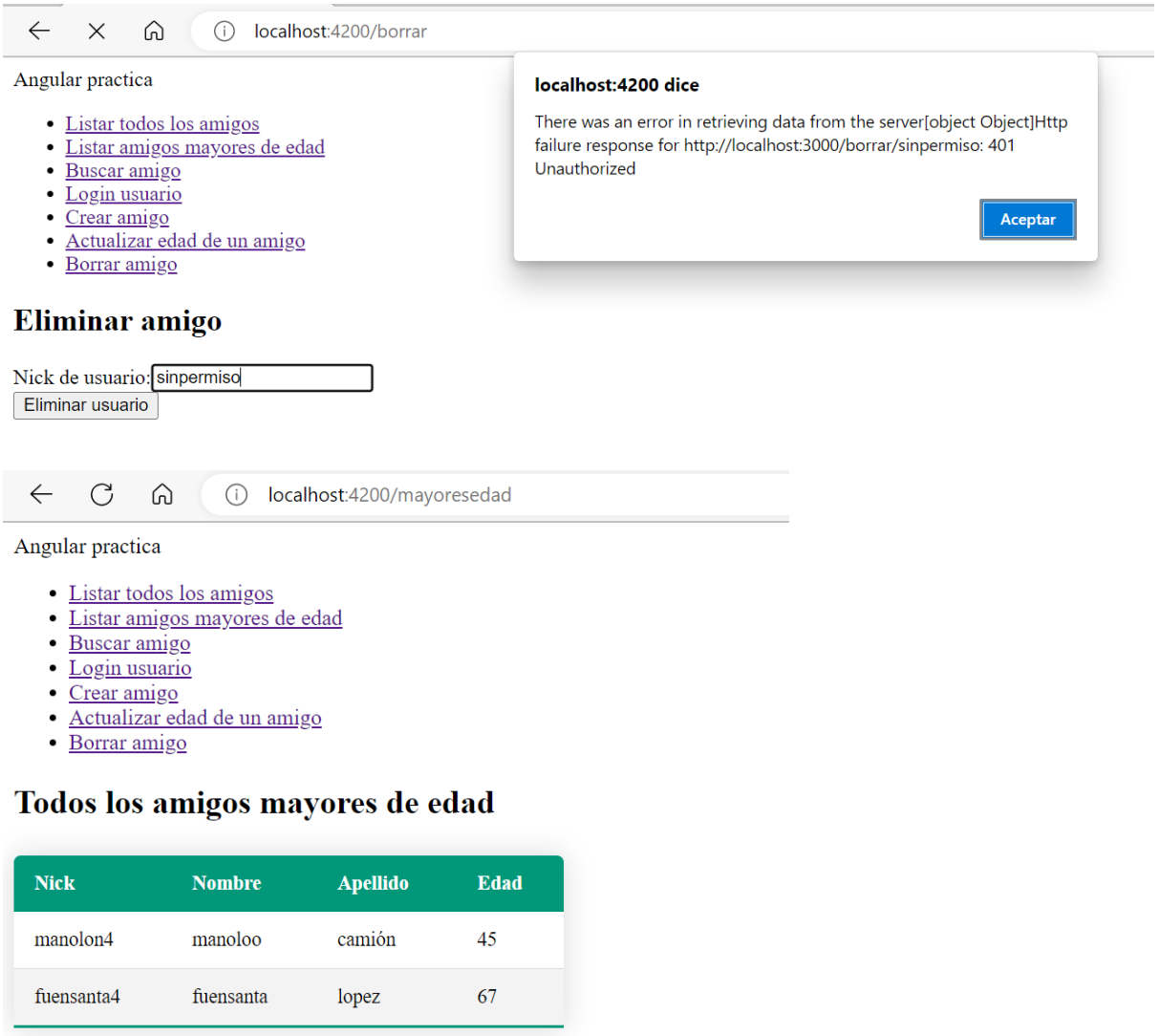
Se muestran las distintas operaciones posibles. Por ejemplo, la de listar todos los amigos:



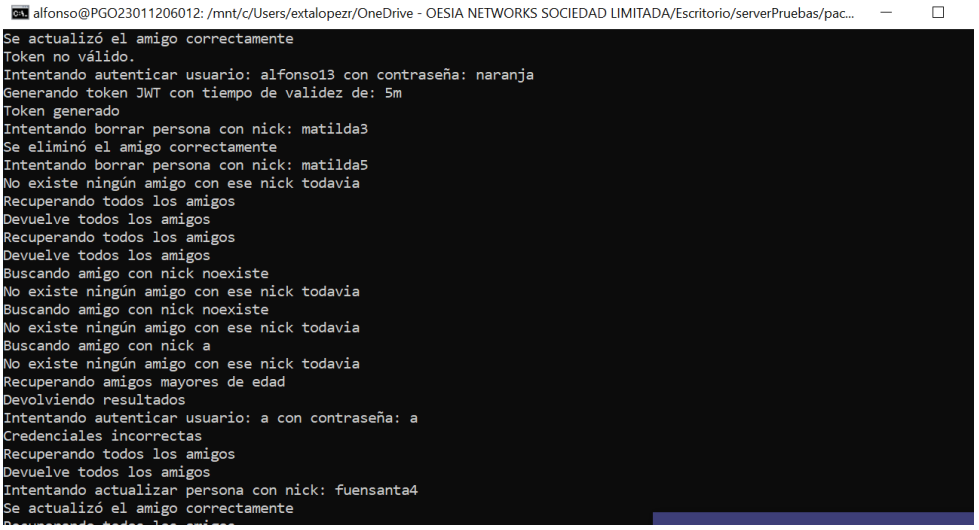


Se muestran también los errores cuando las peticiones no son correctas. La apariencia del front es la siguiente:





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



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.