

TALLER UNIDAD 2 BACKEND



Presentado por:

JOSE LUIS DELGADO (220034116)

Docente:

VICENTE AUX REVELO

UNIVERSIDAD DE NARIÑO

16 DE SEPTIEMBRE DE 2024

PASTO – NARIÑO

1. Lo primero que haremos será crear una carpeta para el BackEnd. Luego, accedemos a MySQL para crear la base de datos destinada a la empresa de adopción de mascotas, **HogarPeludo**, tal como se muestra en la imagen siguiente.

```
C:\DIPLOMADO\HogarPeludo\HogarPeludoBackend>mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.4.27-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database Hogarpeludo;
Query OK, 1 row affected (0.004 sec)

MariaDB [(none)]> S
```

2. A continuación, abrimos Visual Studio Code y, desde la terminal, iniciamos un nuevo proyecto con **Node.js** y **Express.js** utilizando el siguiente comando: **npm init -y**.

```
PS C:\DIPLOMADO\HogarPeludo\HogarPeludoBackend> npm init -y
Wrote to C:\DIPLOMADO\HogarPeludo\HogarPeludoBackend\package.json:

{
  "name": "hogarpeludobackend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Luego, realizamos las configuraciones necesarias para el proyecto. Primero, instalamos **Nodemon** para que el servidor se reinicie automáticamente cada vez que se realicen cambios en el código. Para ello, ejecutamos: **npm install nodemon -D**, Esta dependencia solo será utilizada durante la etapa de desarrollo. También instalamos **Express**, el framework que nos permitirá gestionar las rutas y solicitudes del servidor con el comando: **npm install express**. A continuación, instalamos el módulo **mysql2** para establecer la conexión con la base de datos MySQL con el comando: **npm install mysql2**. Finalmente, instalamos **Sequelize**, que nos facilitará la interacción con la base de datos a través de un ORM para esto ejecutamos: **npm install sequelize**.

```

● PS C:\DIPLOMADO\HogarPeludo\HogarPeludoBackend> npm install nodemon -D

added 29 packages, and audited 30 packages in 4s

4 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
● PS C:\DIPLOMADO\HogarPeludo\HogarPeludoBackend> npm install express

added 70 packages, and audited 100 packages in 6s

17 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
● PS C:\DIPLOMADO\HogarPeludo\HogarPeludoBackend> npm install mysql2

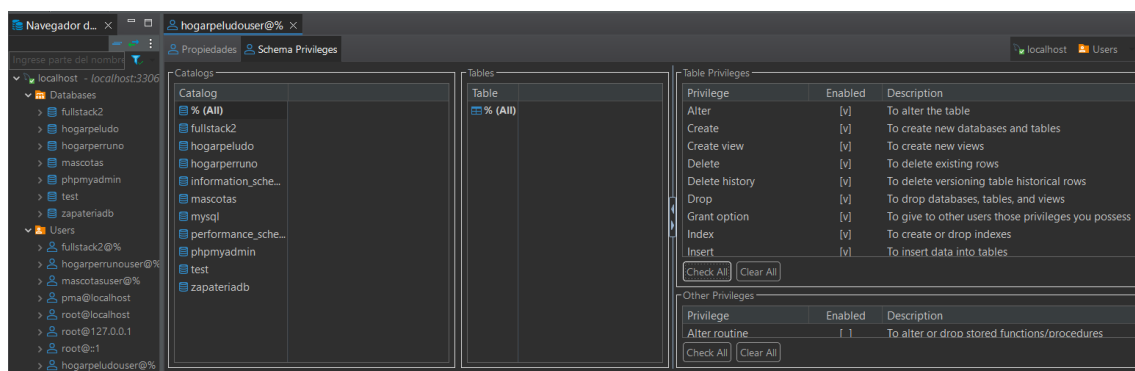
added 12 packages, and audited 112 packages in 8s

18 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
● PS C:\DIPLOMADO\HogarPeludo\HogarPeludoBackend> npm install sequelize

```

3. Después de configurar el entorno, procedemos a crear un usuario y establecer una contraseña en **DBeaver** para administrar la base de datos. Asignamos los privilegios necesarios al nuevo usuario para garantizar que tenga acceso adecuado a la base de datos y pueda realizar las operaciones requeridas. Una vez que el usuario está configurado, comenzamos a trabajar en el desarrollo del backend en **VSC**.



4. Ya en **Visual Studio Code (VSC)**, lo primero que hacemos es crear un archivo llamado `app.js`, donde inicializamos una instancia de **Express**. En este archivo configuramos el puerto en el que correrá el servidor (en este caso, el puerto 3000) y otros aspectos necesarios para el funcionamiento del backend. Además, modificamos el archivo `package.json` para integrar **Nodemon**.

```
"scripts": {  
  "start": "nodemon ./app.js",  
}
```

5. A continuación, organizamos todo el código dentro de una carpeta llamada `src`, donde estructuraremos las diferentes partes del proyecto. Dentro de `src`, creamos las siguientes cuatro carpetas: `database`, `modelos`, `controladores` y `rutas`.

a. Carpeta `database`:

En esta carpeta configuramos la conexión a la base de datos utilizando **Sequelize**. Aquí se define el archivo principal de conexión, donde se establecen los parámetros de la base de datos como el `host`, `usuario`, `contraseña`, y `nombre de la base de datos`.

b. Carpeta `modelos`:

En la carpeta `modelos`, también utilizando **Sequelize**, se crean las tablas correspondientes a los datos. En este caso, tenemos dos modelos:

- **Mascotas**: Representa las mascotas disponibles para adopción.
- **Solicitudes**: Registra las solicitudes de adopción de las mascotas.

Estos modelos definen las propiedades de cada tabla, como los atributos, tipos de datos y relaciones entre las tablas.

c. Carpeta `controladores`:

Aquí se crean los controladores que gestionan la lógica de las operaciones sobre las tablas. En concreto:

- **mascotasController.js**: Se encarga de implementar las operaciones **CRUD** para el modelo de **Mascotas**.
- **solicitudesController.js**: Se implementa el **CRUD** para las **Solicitudes** de adopción.

d. Carpeta `rutas`:

En esta carpeta se crean dos archivos:

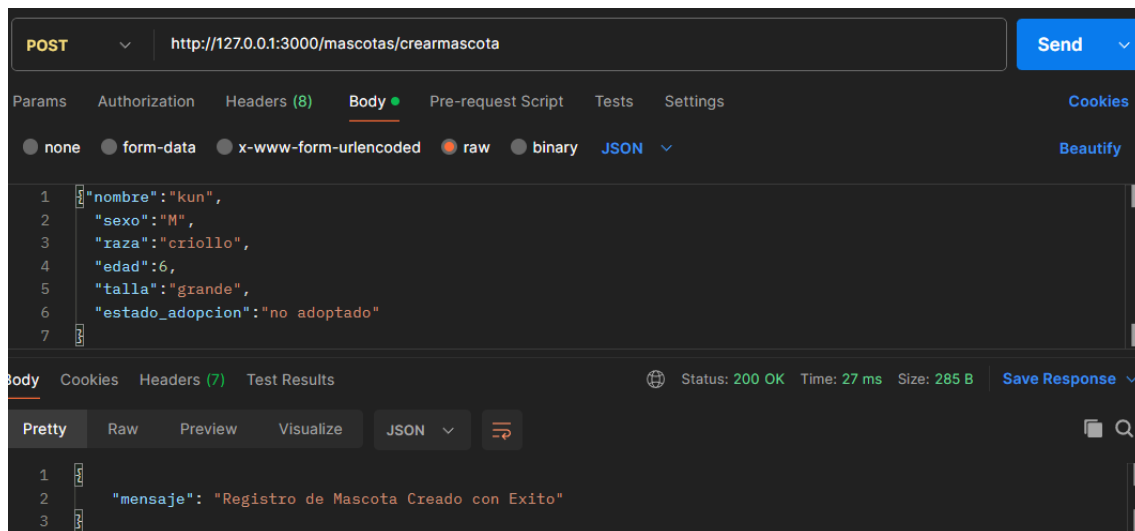
- **mascotasRouter.js**: Aquí se definen las rutas para manejar las operaciones **CRUD** relacionadas con las mascotas.
- **solicitudesRouter.js**: Se definen las rutas para las operaciones **CRUD** de solicitudes de adopción.

Las rutas corresponden a los diferentes verbos HTTP (GET, POST, PUT, DELETE), y cada una está vinculada a las funciones definidas en los controladores.

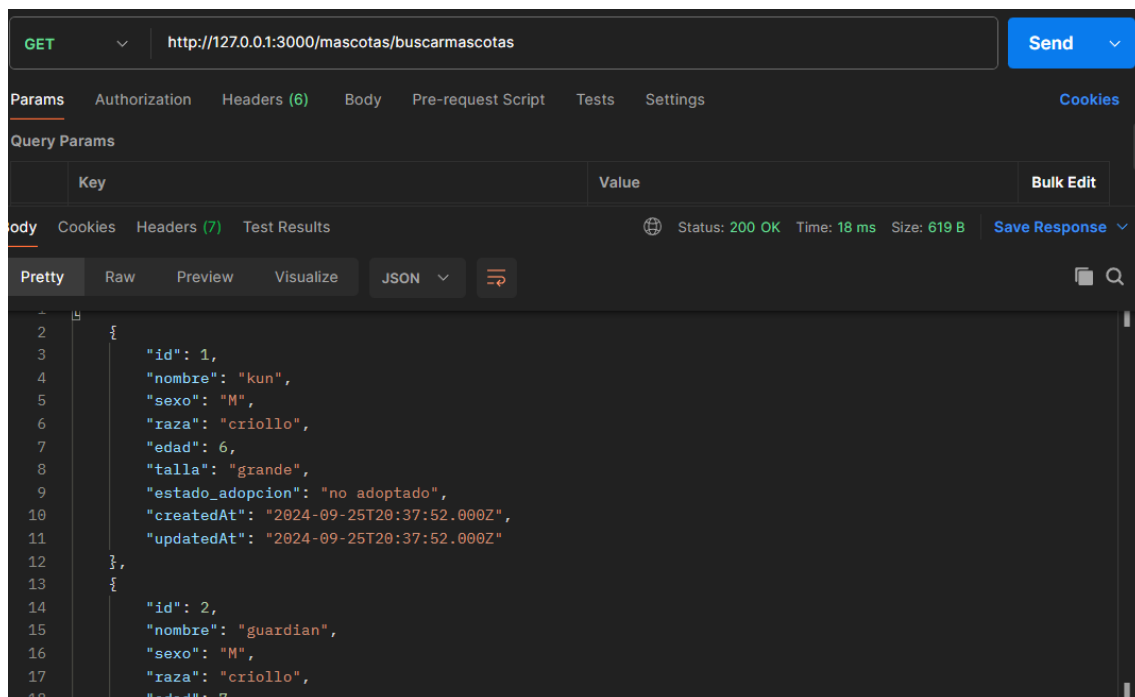
5. Prueba de funcionamiento:

Para asegurarnos de que todo está configurado correctamente, utilizamos **Postman** para probar cada una de las rutas creadas. Se verificaron las operaciones de creación, lectura, actualización y eliminación tanto para **Mascotas** como para **Solicitudes**, confirmando que todo funciona correctamente como se muestra en las siguientes imágenes.

Para crear una mascota.



Buscar mascotas



Buscar mascotas por id

GET

http://127.0.0.1:3000/mascotas/buscarmascota/1

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Bulk Edit
--	-----	-------	-----------

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 41 ms

Size: 423 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "id": 1,
3   "nombre": "kun",
4   "sexo": "M",
5   "raza": "criollo",
6   "edad": 6,
7   "talla": "grande",
8   "estado_adopcion": "no adoptado",
9   "createdAt": "2024-09-25T20:37:52.000Z",
10  "updatedAt": "2024-09-25T20:37:52.000Z"
11 }
```

Actualizar mascota

PUT

http://127.0.0.1:3000/mascotas/actualizarmascota/1

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

JSON

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 39 ms

Size: 286 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "nombre": "firulaia",
3   "sexo": "M",
4   "raza": "criollo",
5   "edad": 5,
6   "talla": "grande",
7   "estado_adopcion": "no adoptado"
8 }
```

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 39 ms

Size: 286 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "tipo": "success",
3   "mensaje": "Registro Actualizado"
4 }
```

Eliminar mascota

DELETE

http://127.0.0.1:3000/mascotas/eliminarmascota/1

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Bulk Edit
--	-----	-------	-----------

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 18 ms

Size: 307 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   "tipo": "success",
3   "mensaje": "Registro con id 1 eliminado correctamente"
4 }
```

CRUD PARA SOLICITUDES

Crear solicitud

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:3000/solicitud/crearsolicitud/1`. The request body is a JSON object with the following fields: `nombre_solicitante`, `identificacion`, `telefono`, `direccion`, and `fecha`. The response is a JSON object with the field `mensaje`.

```
POST http://127.0.0.1:3000/solicitud/crearsolicitud/1

{
  "nombre_solicitante": "luis",
  "identificacion": "2345",
  "telefono": "312456789",
  "direccion": "cra 44 #12b",
  "fecha": "2024-09-22"
}
```

```
{
  "mensaje": "Registro de Solicitud Creado Con Exito"
}
```

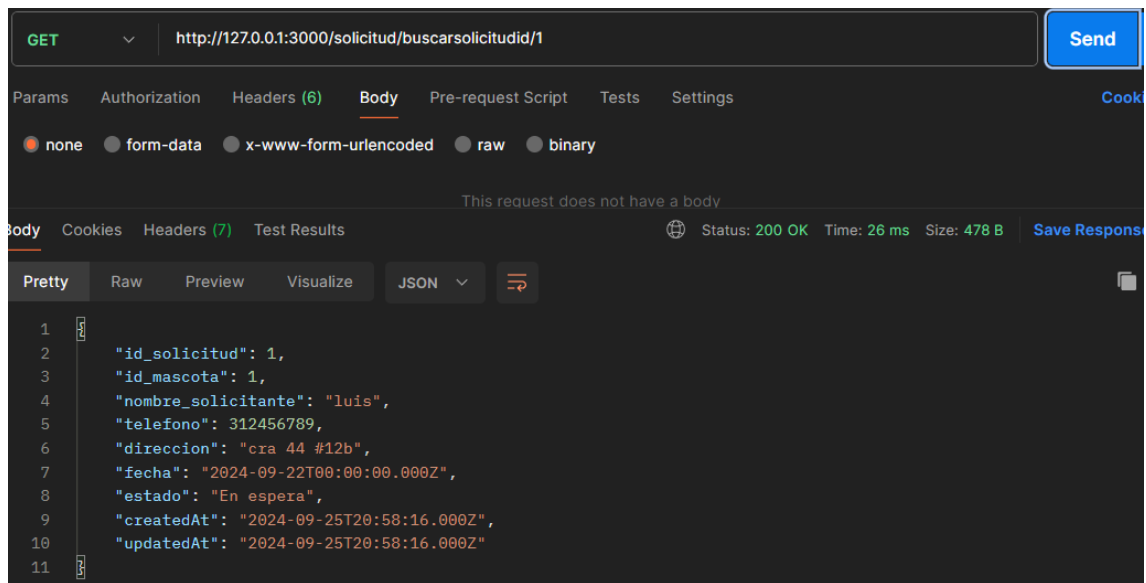
Buscar solicitudes

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:3000/solicitud/buscarsolicitudes`. The response is a JSON object with the following fields: `id_solicitud`, `id_mascota`, `nombre_solicitante`, `telefono`, `direccion`, `fecha`, `estado`, `createdAt`, and `updatedAt`.

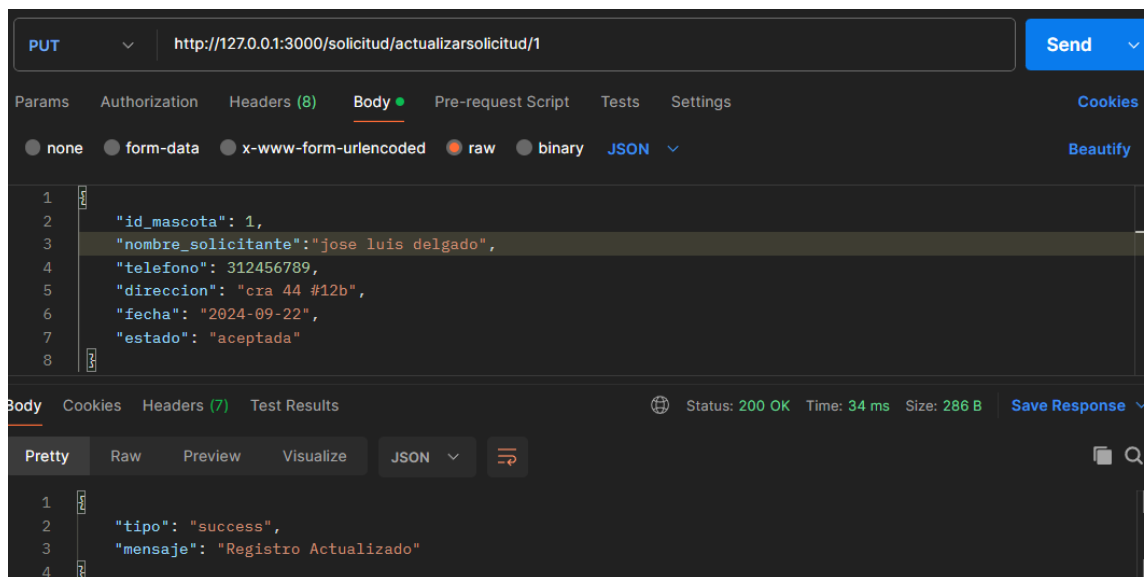
```
GET http://127.0.0.1:3000/solicitud/buscarsolicitudes

{
  "id_solicitud": 1,
  "id_mascota": 1,
  "nombre_solicitante": "luis",
  "telefono": "312456789",
  "direccion": "cra 44 #12b",
  "fecha": "2024-09-22T00:00:00.000Z",
  "estado": "En espera",
  "createdAt": "2024-09-25T20:58:16.000Z",
  "updatedAt": "2024-09-25T20:58:16.000Z"
}
```

Buscar solicitud por id



Actualizar solicitud



Eliminar solicitud

