

INFORME TALLER 2 BACKEND

Presentado por:

BRANDON CAMILO CUATAPI (220034171)

Docente:

VICENTE AUX REVELO

UNIVERSIDAD DE NARIÑO

OCTUBRE - 2024

PASTO – NARIÑO

Introducción

El presente informe detalla el desarrollo de una aplicación Backend como parte del Taller de la Unidad 2 en el Diplomado de actualización en nuevas tecnologías para el desarrollo de software de la Universidad de Nariño. El objetivo principal de este taller fue crear una solución que permita gestionar una empresa de adopción de mascotas llamada “Patitas Felices”, abarcando desde el registro de las mismas hasta la administración de las solicitudes de adopción.

Para cumplir con estos objetivos, se utilizó una base de datos MARIADB, que facilita el manejo de la información tanto de las mascotas como de las solicitudes de adopción. La implementación del backend se realizó utilizando NodeJS y el framework ExpressJS, aplicando correctamente los verbos HTTP para las distintas operaciones de registro, actualización y eliminación de datos.

Finalmente, se realizaron pruebas del sistema utilizando la herramienta grafica Thunder Client para verificar la correcta ejecución de las distintas solicitudes. Este informe documenta el proceso de desarrollo, las tecnologías utilizadas, así como los resultados obtenidos a lo largo del taller.

Proceso de construcción.

1. Instalación de NodeJS y NPM

Primero, nos aseguramos de tener Node.js y npm (Node Package Manager) instalados en nuestro sistema.

Node.js es un entorno de ejecución de JavaScript del lado del servidor que permite ejecutar código JavaScript fuera del navegador. Se utiliza para crear aplicaciones web escalables y rápidas, ya que está basado en un modelo asíncrono y orientado a eventos, lo que lo hace eficiente para manejar múltiples conexiones simultáneamente.

npm (Node Package Manager) es el gestor de paquetes de Node.js. Sirve para instalar, compartir y gestionar librerías y módulos que puedes utilizar en tus proyectos de Node.js, facilitando la reutilización de código y el manejo de dependencias.

2. Inicialización del Proyecto

Creamos un nuevo directorio para nuestro proyecto y navegamos a él.

Inicializamos un proyecto NodeJS con el siguiente comando, que crea un archivo package.json donde se almacenará toda la configuración del proyecto:

```
npm init -y
```

3. Instalación de Dependencias

A continuación, instalamos ExpressJS, que es el framework que utilizaremos para manejar las rutas HTTP y la lógica del backend. Seguido necesitaremos instalar Sequelize que es un **ORM** (Object-Relational Mapper) que facilita la interacción entre aplicaciones Node.js y bases de datos SQL como MySQL, PostgreSQL, MariaDB, SQLite, etc. Con la que podemos usar modelos de JavaScript para interactuar con nuestra base de datos, lo que hace que el código sea más limpio y fácil de mantener. También necesitaremos mariadb para la conexión con la base de datos y un entorno grafico que en este caso sería Dbeaver, y nodemon para facilitar el desarrollo (reinicia automáticamente el servidor cuando hay cambios en el código).

Ejecutamos los siguientes comandos:

```
npm install express mysql2
```

```
npm install sequelize
```

```
npm install --save-dev nodemon
```

El paquete mysql2 nos permite interactuar con bases de datos MySQL o MariaDB.

Luego de esto configuramos correctamente nuestro archivo package.json con las dependencias correspondientes.

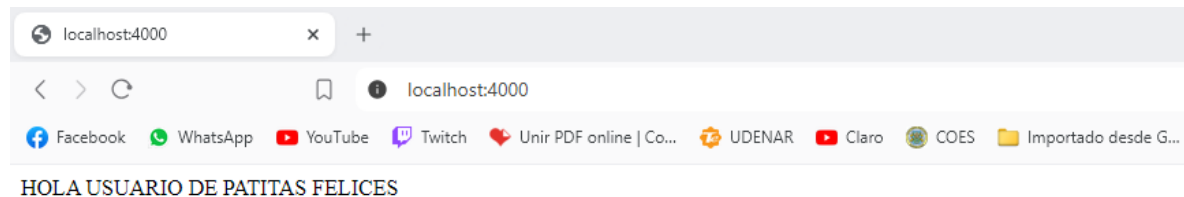
4. Creación del Servidor con Express

Creamos un archivo llamado app.js, que será el punto de entrada de nuestra aplicación.

Para ejecutar el servidor usamos el comando:

```
npm run start
```

De esta manera el servidor corre en <http://localhost:4000>.



5. Creación y conexión a la Base de Datos MariaDB

Para crear y conectar nuestra base de datos utilizamos XAMPP y DBeaver

XAMPP para el servicio de MySQL/MariaDB:

XAMPP es un paquete de software que incluye varios servicios, como Apache y MySQL, entre otros, y permite tener un entorno de servidor local en tu máquina.

En este caso, utilizamos XAMPP para levantar el servicio de MySQL, que en realidad usa MariaDB como motor de base de datos.

A través del panel de control de XAMPP, activé el servicio de MySQL, lo que me permitió tener una instancia de MariaDB corriendo localmente en mi máquina.

DBeaver para la administración de la base de datos:

DBeaver es una herramienta gráfica (GUI) que facilita la administración y manipulación de bases de datos.

Con DBeaver, conecté mi base de datos MariaDB que se estaba ejecutando en XAMPP.

Para esto, configuramos una nueva conexión con un nuevo archivo en nuestro proyecto llamado conexion, especificando:

El nombre de nuestra base de datos, nuestro usuario y nuestra contraseña creadas en DBeaver.

```
src > database > conexion.js > ...
1 import Sequelize from "sequelize";
2
3 const db = new Sequelize("adoptmascotas","adminmascotas","220034171",{
4   dialect: "mysql",
5   host: "localhost"
6 });
7
8 export {db}
9
```

Para verificar la conexión añadimos un código en nuestro app.js:

```
14 //Verificar Conexion Base de Datos
15 db.authenticate().then(()=>{
16   console.log('Conexion correcta');
17 }).catch(err=>{
18   console.log('Conexion incorrecta');
19 })
```

6. Creación de modelos

Para la creación de las tablas Mascotas y AdopcionesMascotas, es necesario crear una carpeta llamada modelos en nuestro proyecto. Dentro de esta carpeta, deben incluirse dos archivos: uno para la tabla Mascotas, que contendrá los atributos correspondientes a las mascotas, y otro para la tabla Adopciones, donde se definirán los atributos relacionados con las adopciones.

7. Definición de Rutas y Funciones

Una vez conectada la base de datos, podemos proceder a definir las rutas para las operaciones de Crear, Buscar, Actualizar y Eliminar, tanto para la administración de mascotas como para las solicitudes de adopción.

Para ello, creamos una carpeta llamada rutas en nuestro proyecto, que contendrá dos archivos: uno para las rutas y funciones relacionadas con la gestión de mascotas, y otro para las rutas y funciones correspondientes a la gestión de las solicitudes de adopción.

Este ajuste mejora la claridad y el flujo de la información, manteniendo una estructura más organizada y formal.

8. Creación de controladores

Necesitamos crear dos controladores para permitir a la aplicación interactuar con la base de datos de mascotas y realizar todas las operaciones necesarias para gestionar estos datos y gestionar las operaciones relacionadas con las solicitudes de adopción, permitiendo crear nuevas solicitudes, visualizarlas, actualizarlas o eliminarlas, manteniendo un registro de las adopciones.

Ambos archivos permiten que la aplicación web interactúe con la base de datos a través de Sequelize.

Archivo 1: Controlador de Mascotas

Este archivo define las funciones del controlador relacionadas con la entidad mascotas. Su objetivo es gestionar las operaciones sobre las mascotas en la base de datos. Las funciones incluyen:

Crear una mascota (crear): Toma los datos de la mascota desde la solicitud HTTP (nombre, edad, tamaño, info) y los almacena en la base de datos utilizando Sequelize.

Buscar todas las mascotas (buscar): Recupera todas las mascotas almacenadas en la base de datos.

Buscar una mascota por ID (buscarId): Recupera una mascota específica por su ID.

Actualizar una mascota (actualizar): Modifica los detalles de una mascota identificada por su ID.

Eliminar una mascota (eliminar): Elimina una mascota de la base de datos basándose en su ID.

Archivo 2: Controlador de Solicitudes de Adopción

Este archivo define las funciones del controlador para gestionar las solicitudes de adopción. Estas funciones permiten crear, consultar, actualizar y eliminar las solicitudes de adopción. Las operaciones incluyen:

Crear una solicitud de adopción (crearSolicitud): Toma la información del solicitante y la mascota (nombre del solicitante, contacto y el ID de la mascota) y crea una nueva solicitud de adopción.

Buscar todas las solicitudes de adopción (buscarSolicitudes): Recupera todas las solicitudes de adopción almacenadas en la base de datos.

Buscar una solicitud por ID (buscarSolicitudId): Recupera una solicitud específica por su ID.

Actualizar una solicitud de adopción (actualizarSolicitud): Modifica los detalles de una solicitud de adopción basándose en su ID.

Eliminar una solicitud de adopción (eliminarSolicitud): Elimina una solicitud de la base de datos según su ID.

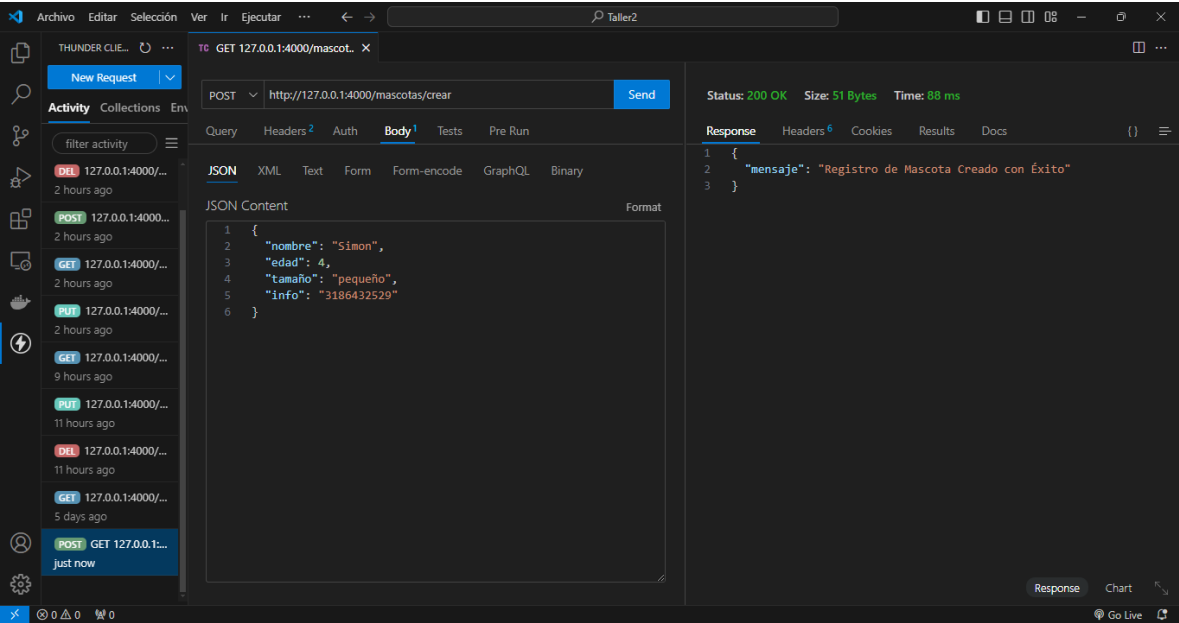
9. Verificación de operaciones con THUNDER CLIENT

Thunder Client es una extensión ligera para Visual Studio Code que permite realizar pruebas de APIs REST, ofreciendo funcionalidades como solicitudes HTTP, manejo de encabezados y parámetros, visualización de respuestas y soporte para autenticación, todo integrado en el editor de código.

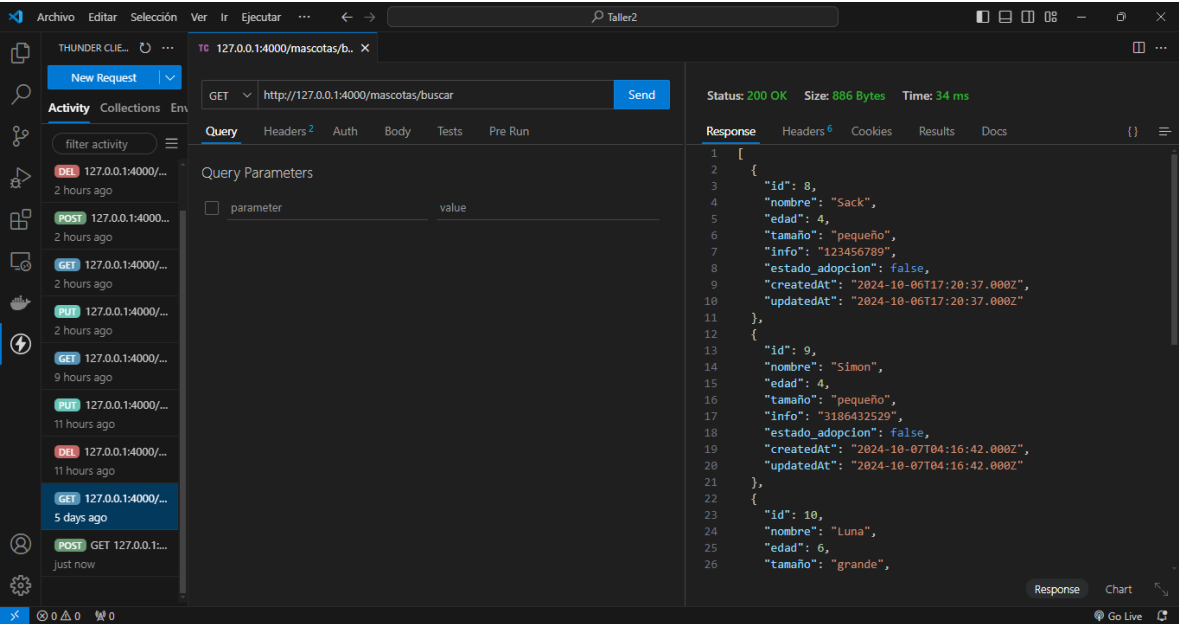
A continuación, tenemos las pruebas de las solicitudes realizadas:

OPERACIONES MASCOTAS.

Creación de mascotas:



Buscar todas las mascotas:



Buscar por ID:

The screenshot shows the Thunder Client interface with a GET request to `http://127.0.0.1:4000/mascotas/buscarid/9`. The response is a 200 OK status with 177 bytes and a response time of 30 ms. The response body is a JSON object representing a pet with ID 9.

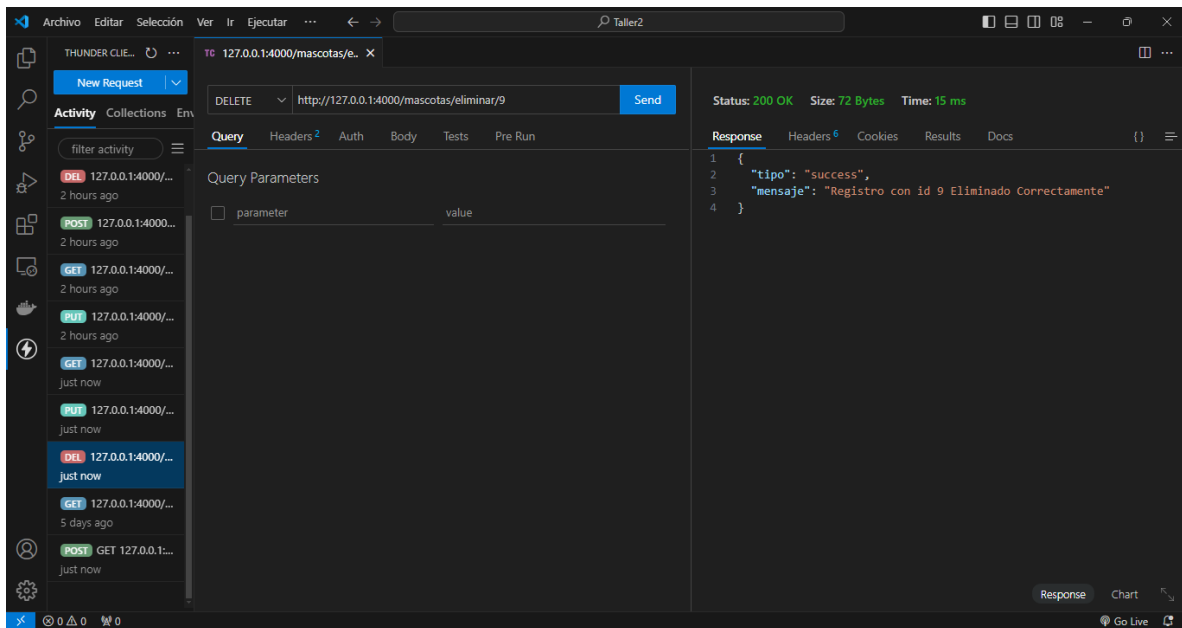
```
1 {
2   "id": 9,
3   "nombre": "Simon",
4   "edad": 4,
5   "tamaño": "pequeño",
6   "info": "3186432529",
7   "estado_adopcion": false,
8   "createdAt": "2024-10-07T04:16:42.000Z",
9   "updatedAt": "2024-10-07T04:16:42.000Z"
10 }
```

Actualizar mascota:

The screenshot shows the Thunder Client interface with a PUT request to `http://127.0.0.1:4000/mascotas/actualizar/9`. The request body is a JSON object with the updated pet information. The response is a 200 OK status with 51 bytes and a response time of 27 ms. The response body is a JSON object indicating a successful update.

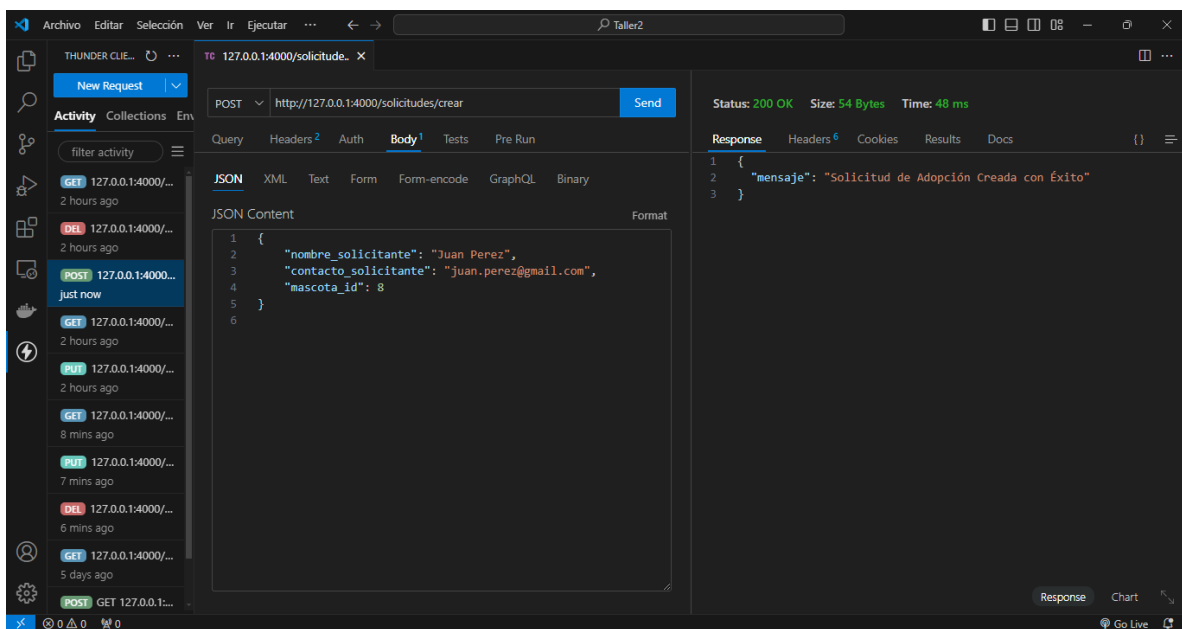
```
1 {
2   "tipo": "success",
3   "mensaje": "Registro Actualizado"
4 }
```


Eliminar mascota:

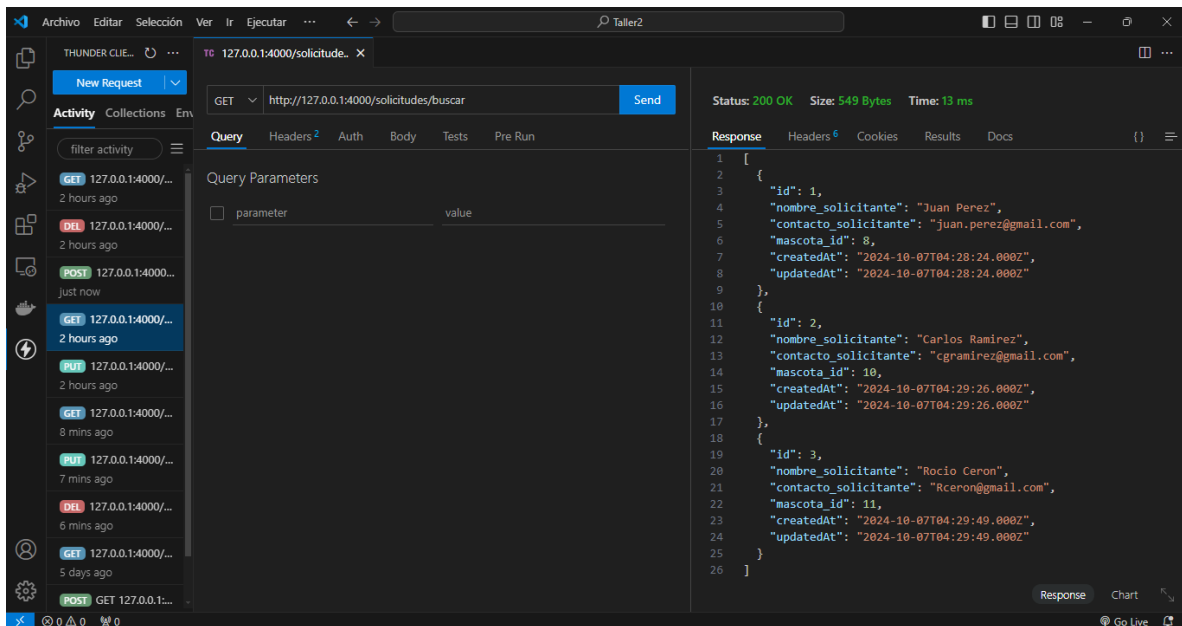


OPERACIONES SOLICITUDES.

Crear solicitud:



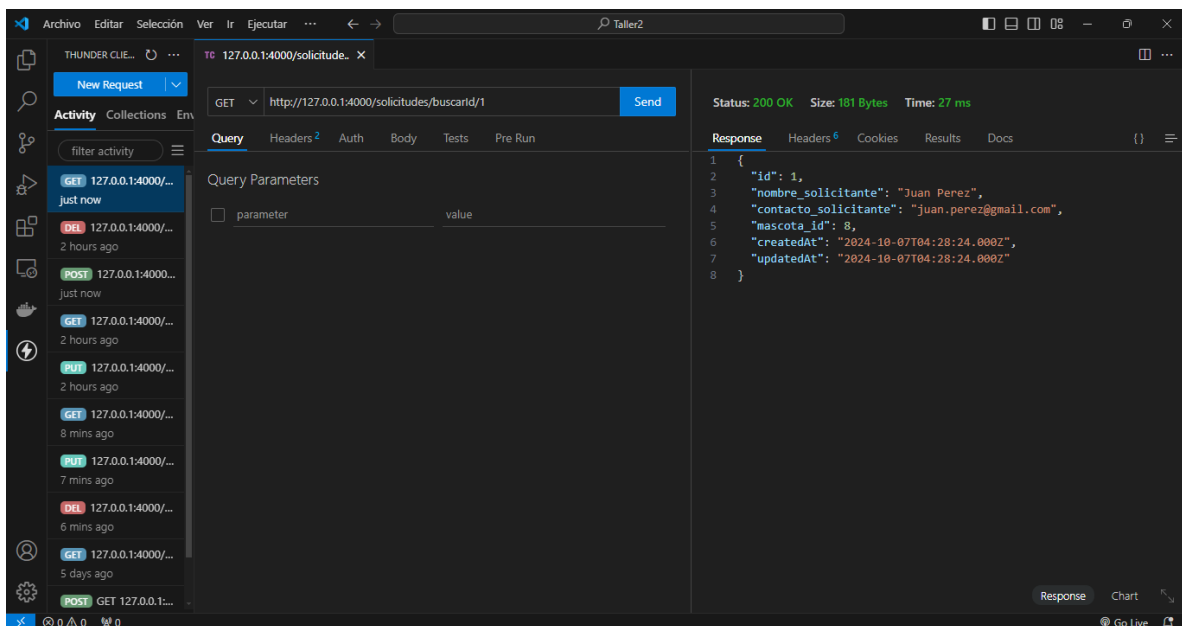
Buscar todas las solicitudes:



Thunder Client interface showing a GET request to `http://127.0.0.1:4000/solicitudes/buscar`. The response is a JSON array of 3 objects, each representing a request with fields like `id`, `nombre_solicitante`, `contacto_solicitante`, `mascota_id`, `createdAt`, and `updatedAt`.

```
1 [{
2   {
3     "id": 1,
4     "nombre_solicitante": "Juan Perez",
5     "contacto_solicitante": "juan.perez@gmail.com",
6     "mascota_id": 8,
7     "createdAt": "2024-10-07T04:28:24.000Z",
8     "updatedAt": "2024-10-07T04:28:24.000Z"
9   },
10  {
11    "id": 2,
12    "nombre_solicitante": "Carlos Ramirez",
13    "contacto_solicitante": "cgramirez@gmail.com",
14    "mascota_id": 10,
15    "createdAt": "2024-10-07T04:29:26.000Z",
16    "updatedAt": "2024-10-07T04:29:26.000Z"
17  },
18  {
19    "id": 3,
20    "nombre_solicitante": "Rocio Ceron",
21    "contacto_solicitante": "Rceron@gmail.com",
22    "mascota_id": 11,
23    "createdAt": "2024-10-07T04:29:49.000Z",
24    "updatedAt": "2024-10-07T04:29:49.000Z"
25  }
26  ]
```

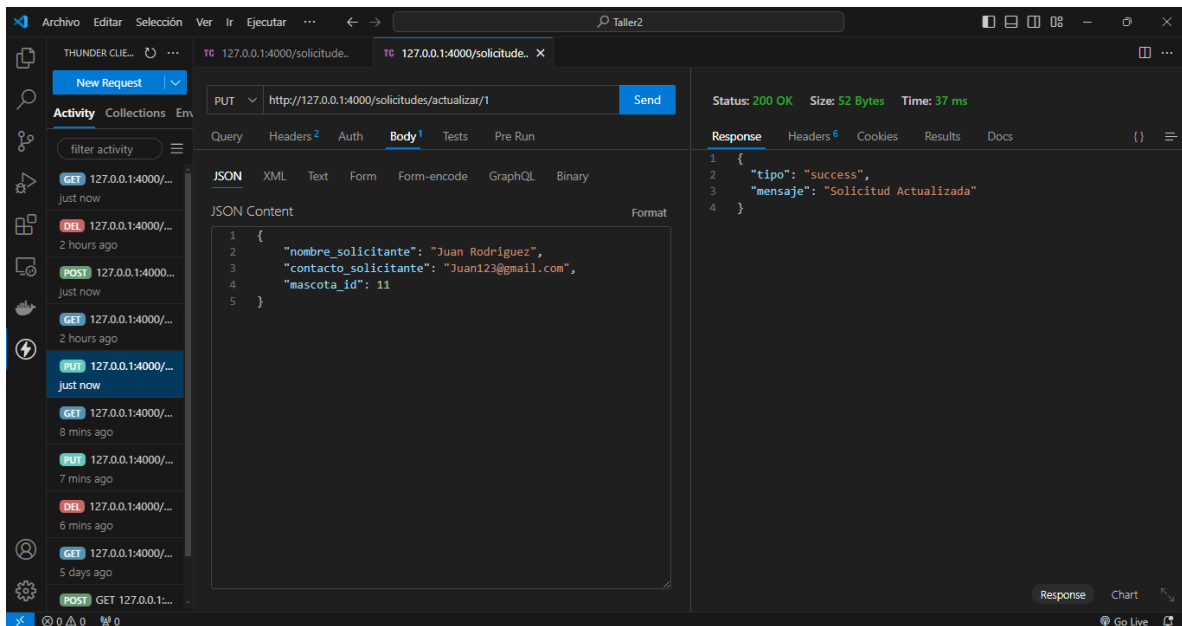
Buscar Solicitudes por ID:



Thunder Client interface showing a GET request to `http://127.0.0.1:4000/solicitudes/buscarid/1`. The response is a single JSON object representing the request with ID 1.

```
1 {
2   "id": 1,
3   "nombre_solicitante": "Juan Perez",
4   "contacto_solicitante": "juan.perez@gmail.com",
5   "mascota_id": 8,
6   "createdAt": "2024-10-07T04:28:24.000Z",
7   "updatedAt": "2024-10-07T04:28:24.000Z"
8 }
```

Actualizar Solicitudes:



Eliminar Solicitudes:

