

TALLER UNIDAD 2 BACKEND



Presentado por:

LUIS FELIPE SANTACRUZ CHINCHAJOA (220034097)

Docente:

VICENTE AUX

SEPTIEMBRE 2024

UNIVERSIDAD DE NARIÑO

**DIPLOMADO DE ACTUALIZACIÓN EN NUEVAS TECNOLOGÍAS
PARA EL DESARROLLO DE SOFTWARE**

PASTO – NARIÑO

1. Introducción

En el presente informe se detalla el desarrollo de una aplicación backend destinada a gestionar una empresa de adopción de mascotas. Esta aplicación permite llevar el registro de las mascotas disponibles para adopción, así como gestionar las solicitudes de adopción de los usuarios. Se implementó utilizando tecnologías modernas como NodeJS y ExpressJS, además de una base de datos en MySQL/MariaDB, la cual se gestiona localmente mediante el servidor XAMPP.

Para facilitar la administración y visualización de la base de datos, se utilizó DBeaver, una herramienta robusta que permite conectar y administrar bases de datos de manera eficiente. Asimismo, se realizaron pruebas exhaustivas con Postman para asegurar el correcto funcionamiento del sistema y validar las diferentes operaciones disponibles.

El objetivo principal de este proyecto es crear una solución eficiente para el manejo de los registros de mascotas y las solicitudes de adopción, garantizando la integridad y consistencia de los datos a lo largo de todo el proceso.

2. Desarrollo de la base de datos

2.1 Diseño de la base de datos

La base de datos fue creada utilizando MySQL/MariaDB a través de XAMPP, que proporcionó un servidor local para gestionar los datos de la empresa de adopción de mascotas. La base de datos, llamada **adopcion_mascotas**, incluye tres tablas principales: una para el registro de las mascotas, otra para los adoptantes y una última para las solicitudes de adopción.

Tablas creadas:

- ❖ **Tabla Mascotas:** Contiene información sobre cada mascota registrada en la empresa.
 - **Estructura:**
 - **id_mascota:** Identificador único para cada mascota (clave primaria).
 - **nombre:** Nombre de la mascota.
 - **especie:** Especie a la que pertenece la mascota (perro, gato, etc.).
 - **raza:** Raza de la mascota.
 - **edad:** Edad de la mascota.
 - **estado_adopcion:** Indica si la mascota está disponible para adopción (0 para disponible, 1 para adoptada).
- ❖ **Tabla Adoptantes:** Registra la información de los adoptantes interesados en las mascotas.
 - **Estructura:**
 - **id_adoptante:** Identificador único para cada adoptante (clave primaria).
 - **nombre:** Nombre del adoptante.
 - **contacto:** Información de contacto del adoptante (teléfono, correo electrónico, etc.).
 - **direccion:** Dirección del adoptante.

- ❖ **Tabla Solicitudes:** Almacena los detalles de las solicitudes de adopción realizadas por los usuarios interesados.

➤ **Estructura:**

- **id_solicitud:** Identificador único para cada solicitud (clave primaria).
- **id_adoptante:** Referencia al identificador del adoptante que realizó la solicitud (clave foránea).
- **id_mascota:** Referencia al identificador de la mascota a la que corresponde la solicitud (clave foránea).
- **fecha_solicitud:** Fecha en que se realizó la solicitud.
- **estado_solicitud:** Estado actual de la solicitud (pendiente, aprobada, rechazada).

Script SQL para crear la base de datos y las tablas:

```
<localhost> Script-2 X
CREATE DATABASE IF NOT EXISTS `adopcion_mascotas`;
USE adopcion_mascotas;

CREATE TABLE adoptantes (
  id_adoptante INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50),
  telefono VARCHAR(20),
  direccion VARCHAR(100),
  correo VARCHAR(50)
);

CREATE TABLE mascotas (
  id_mascota INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50),
  especie VARCHAR(50),
  raza VARCHAR(50),
  edad INT,
  estado_adopcion BOOLEAN DEFAULT 0
);




CREATE TABLE solicitudes (
  id_solicitud INT AUTO_INCREMENT PRIMARY KEY,
  id_adoptante INT,
  id_mascota INT,
  fecha_solicitud DATE,
  estado_solicitud VARCHAR(20),
  FOREIGN KEY (id_adoptante) REFERENCES adoptantes(id_adoptante),
  FOREIGN KEY (id_mascota) REFERENCES mascotas(id_mascota)
);
```

2.2 Población de tablas de la base de datos

Una vez creadas las tablas, se procedió a la inserción de datos iniciales en la base de datos para simular el entorno real de la empresa de adopción de mascotas. Se agregaron cuatro registros en la tabla de **Mascotas**, cada uno representando a un animal disponible para adopción, se agregó un adoptante en la tabla **Adoptantes** y se realizó una solicitud de adopción en la tabla **Solicitudes**. Los datos agregados fueron los siguientes:

<div>←T→</div>				id_mascota	nombre	especie	raza	edad	estado_adopcion
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1	Mateo	Perro	Bulldog	1	0
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	2	Kitty	Gato	Persa	2	0
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	3	Mono	Gato	Angora	1	0
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	4	Doky	Perro	Husky	3	0

←T→		id_adoptante	nombre	telefono	direccion	correo
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1	Matias Rosales	3224823651 Aranda matias10@gmail.com

←T→		id_solicitud	id_adoptante	id_mascota	fecha_solicitud	estado_solicitud
<input type="checkbox"/>	 Editar	 Copiar	 Borrar	1	1	4 2024-09-16 Pendiente

Con estos datos iniciales, se garantizó el correcto funcionamiento de las operaciones de inserción y consulta en las tablas correspondientes de la base de datos.

2.3 Conexión de la base de datos con el backend

Se utilizó el paquete **mysql2** para establecer la conexión entre la base de datos y la aplicación backend. La configuración de la conexión se realizó dentro del archivo principal **app.js** del proyecto, donde se especificaron los parámetros de acceso (host, usuario, contraseña, y nombre de la base de datos).

3. Desarrollo del Backend

3.1 Configuración del entorno de desarrollo

El backend fue desarrollado utilizando **NodeJS** y **ExpressJS**. Se configuró el entorno de desarrollo mediante la inicialización de un proyecto NodeJS en **Visual Studio Code**, seguido de la instalación de las dependencias necesarias, tales como **ExpressJS** y **mysql2**. A continuación, se presenta el archivo **app.js**, que define el servidor y las rutas principales.

Comando para inicializar el proyecto:

```
npm init -y
```

```
PROBLEMS OUTPUT TERMINAL ... powershell + v [icon] [icon] ... ^ x

PS D:\10semestre\TallerBackend-FelipeSantacruz> npm init -y
Wrote to D:\10semestre\TallerBackend-FelipeSantacruz\package.json:

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

Comando para la instalación de las dependencias:

```
npm install express mysql2 body-parser
npm install express mysql2 sequelize dotenv
```

```
PROBLEMS OUTPUT TERMINAL ... powershell + v [icon] [icon] ... ^ x

PS D:\10semestre\TallerBackend-FelipeSantacruz>
PS D:\10semestre\TallerBackend-FelipeSantacruz> npm install express mysql2 body-parser

added 77 packages, and audited 78 packages in 15s

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

found 0 vulnerabilities
PS D:\10semestre\TallerBackend-FelipeSantacruz> █
```

Podemos ver todas las dependencias en package.json:

```

{} package.json > ...
1  {
2    "name": "tallerbackend-felipesantacruz",
3    "version": "1.0.0",
4    "main": "app.js",
5    "scripts": {
6      "start": "node src/app.js",
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "description": "",
13   "dependencies": {
14     "body-parser": "^1.20.3",
15     "dotenv": "^16.4.5",
16     "express": "^4.21.0",
17     "mysql2": "^3.11.3",
18     "sequelize": "^6.37.3"
19   }
20 }

```

3.2 Organización de carpetas

La organización de carpetas del aplicativo se estructuró dentro de la carpeta principal, con el fin de facilitar el mantenimiento y la escalabilidad del código. En la subcarpeta **src**, se distribuyeron los componentes del proyecto en las siguientes carpetas:

- **controladores:** Contiene los archivos `adoptanteController.js`, `mascotaController.js` y `solicitudController.js`, que gestionan la lógica de la aplicación y las solicitudes HTTP.
- **database:** Incluye el archivo `conexion.js`, que establece la conexión con la base de datos.
- **modelos:** Almacena los modelos `adoptanteModelo.js`, `mascotaModelo.js` y `solicitudModelo.js`, que definen la estructura de las tablas en la base de datos.
- **rutas:** Contiene los archivos `adopcionRouter.js`, `adoptanteRouter.js` y `mascotaRouter.js`, que gestionan las rutas y solicitudes asociadas a cada recurso.

```

TALLERBACKEND-FELIPE...
├── node_modules
├── src
│   ├── controladores
│   │   ├── adoptanteController.js
│   │   ├── mascotaController.js
│   │   └── solicitudController.js
│   ├── database
│   │   └── conexion.js
│   ├── modelos
│   │   ├── adoptanteModelo.js
│   │   ├── mascotaModelo.js
│   │   └── solicitudModelo.js
│   ├── rutas
│   │   ├── adopcionRouter.js
│   │   ├── adoptanteRouter.js
│   │   └── mascotaRouter.js
│   └── app.js
├── .env
├── informe.pdf
├── package-lock.json
└── package.json

```

3.3 Desarrollo de las rutas para mascotas y solicitudes

El archivo **app.js** contiene la configuración básica para las rutas que gestionan las mascotas y las solicitudes de adopción en la aplicación. Estas rutas se estructuran de la siguiente manera:

- **Rutas de Mascotas:** Se definen las operaciones para listar todas las mascotas registradas, agregar nuevas mascotas y obtener detalles de una mascota específica.
- **Rutas de Solicitudes:** Se implementan las operaciones para registrar nuevas solicitudes de adopción y consultar el estado de las solicitudes existentes.

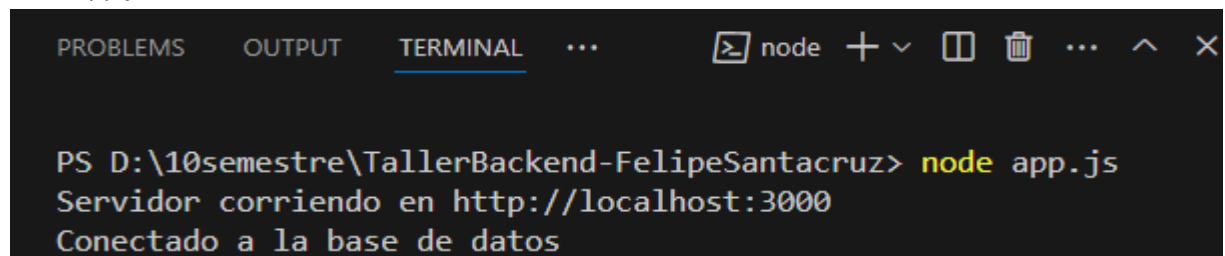
Código principal del servidor:

```
src > JS app.js > ...
1  // src/app.js
2  const express = require('express');
3  const sequelize = require('./database/conexion');
4  const adoptanteRouter = require('./rutas/adoptanteRouter');
5  const mascotaRouter = require('./rutas/mascotaRouter');
6  const adopcionRouter = require('./rutas/adopcionRouter');
7
8  const app = express();
9  app.use(express.json());
10
11 // Rutas
12 app.use('/api/adoptantes', adoptanteRouter);
13 app.use('/api/mascotas', mascotaRouter);
14 app.use('/api/solicitudes', adopcionRouter);
15
16 // Sincronizar modelos y luego iniciar el servidor
17 sequelize.sync()
18   .then(() => {
19     app.listen(3000, () => {
20       console.log('Servidor corriendo en http://localhost:3000');
21     });
22   })
23   .catch((error) => {
24     console.error('Error al sincronizar con la base de datos:', error);
25   });
26
```

3.4 Ejecución del servidor

El servidor se ejecuta en el puerto 3000 y responde a las solicitudes HTTP enviadas a través de **Postman**. Para iniciar el servidor, se utilizó el siguiente comando en la terminal:

node app.js



```
PROBLEMS  OUTPUT  TERMINAL  ...  node  +  -  [ ]  [ ]  ...  ^  X

PS D:\10semestre\TallerBackend-FelipeSantacruz> node app.js
Servidor corriendo en http://localhost:3000
Conectado a la base de datos
```

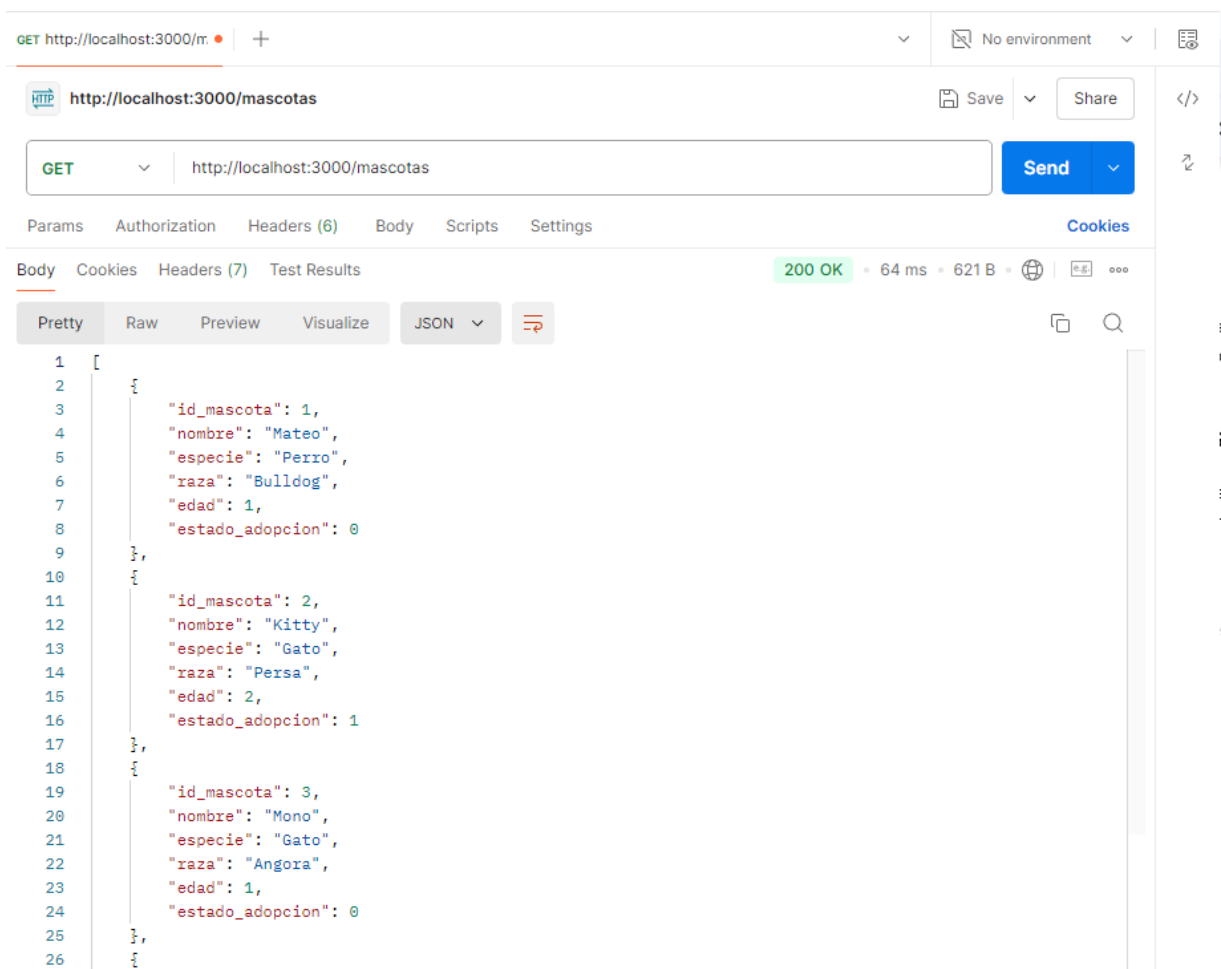
4. Pruebas realizadas con Postman

Se realizaron diversas pruebas para validar el correcto funcionamiento de las rutas implementadas. Estas pruebas se ejecutaron en **Postman**, enviando solicitudes HTTP a las rutas **GET** y **POST** desarrolladas en el backend.

4.1 Solicitud para listar todas las mascotas (GET /mascotas)

Se probó la ruta **GET /mascotas** para obtener el listado de todas las mascotas registradas en la base de datos. Se seleccionó el método **GET** y en el campo de URL se escribió la siguiente dirección:

<http://localhost:3000/mascotas>

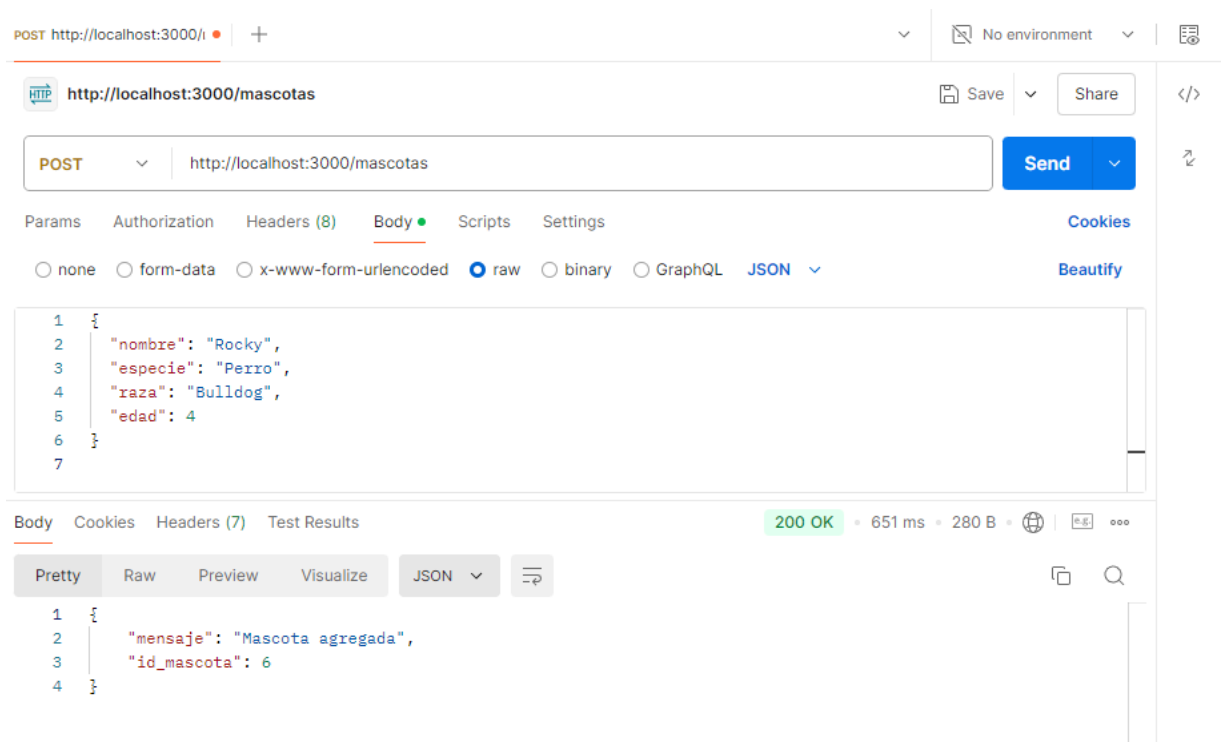


La solicitud devolvió los registros correctamente en formato JSON.

4.2 Solicitud para agregar una nueva mascota (POST /mascotas)

Se probó la ruta **POST /mascotas** para agregar una nueva mascota a la base de datos, enviando la información en formato JSON en el cuerpo de la solicitud. Se seleccionó el método **POST** y en el campo de URL se escribió la siguiente dirección:

<http://localhost:3000/mascotas>



La operación fue exitosa y la nueva mascota se agregó a la base de datos.

			id_mascota	nombre	especie	raza	edad	estado_adopcion
<input type="checkbox"/>	Editar	Copiar	Borrar	1	Mateo	Perro	Bulldog	1
<input type="checkbox"/>	Editar	Copiar	Borrar	2	Kitty	Gato	Persa	2
<input type="checkbox"/>	Editar	Copiar	Borrar	3	Mono	Gato	Angora	1
<input type="checkbox"/>	Editar	Copiar	Borrar	4	Doky	Perro	Husky	3
<input type="checkbox"/>	Editar	Copiar	Borrar	6	Rocky	Perro	Bulldog	4

5. Conclusión

El desarrollo de este backend para la gestión de una empresa de adopción de mascotas permitió aplicar los conocimientos adquiridos en NodeJS y ExpressJS, así como la integración con una base de datos MySQL/MariaDB. La creación de las tablas, la configuración del servidor y la implementación de las rutas fueron pasos clave para alcanzar los objetivos del proyecto. Las pruebas realizadas a través de Postman evidencian que el sistema es capaz de manejar correctamente los registros de mascotas y solicitudes de adopción, garantizando un flujo eficiente de información.

Este proyecto no solo permitió mejorar las habilidades técnicas, sino también el entendimiento del ciclo de vida de una aplicación backend, desde la creación de la base de datos hasta la interacción con los usuarios mediante rutas HTTP.