

# Universidad de Nariño



Septiembre de 2024

Taller: unidad 2

**Diana Marcela Toro Ortiz**

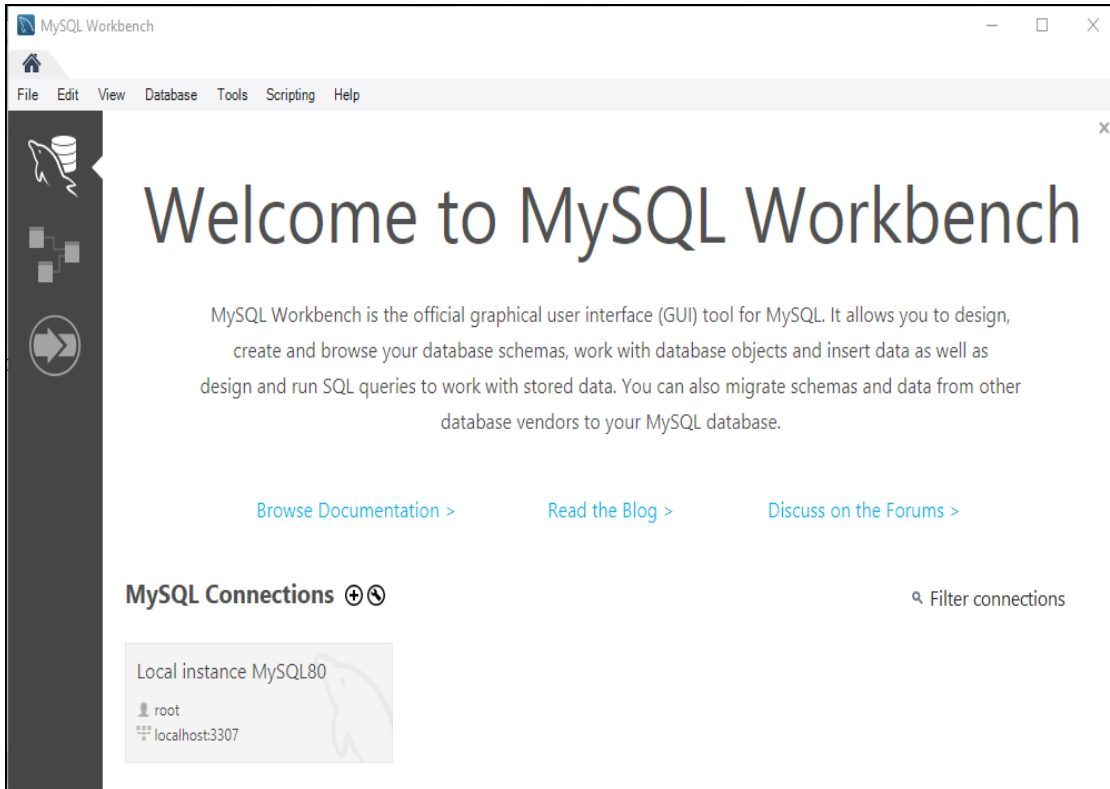
**Diplomado Nuevas Tecnologías De Desarrollo De Software**

**Mg. Vicente Aux Revelo**

## 1. Proceso de Construcción de la Base de Datos para la Empresa de Adopción de Mascotas

### Descargamos MySQL

Para empezar, accedemos al sitio oficial de MySQL: <https://mysql.com>. Desde allí, descargamos el instalador correspondiente a nuestro sistema operativo (Windows, macOS o Linux). A continuación, Seguimos los pasos correspondientes de instalación, asegurándonos de configurar una contraseña segura para el usuario root durante el proceso de configuración.



Una vez instalado MySQL iniciamos el servicio, nos aseguramos de que el servicio este en ejecución, para ello nos dirigimos a la terminal o consola de comandos y verificamos que el servicio este activo. Digitando el siguiente comando.

**net start MySQL80**, esto iniciará el servicio o confirmará que el servicio ya está en ejecución.

```
C:\> Seleccinar Administrador: Símbolo del sistema

Microsoft Windows [Versión 10.0.19045.4894]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Windows\system32>net start MySQL80
El servicio solicitado ya ha sido iniciado.

Puede obtener más ayuda con el comando NET HELPMSG 2182.

C:\Windows\system32>
```

Una vez que ya miramos que el servicio está funcionando correctamente accedemos a MySQL, Para ingresar al cliente de MySQL. En el terminal escribimos el siguiente comando

**"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql.exe" -u root -p --port=3307**

En este caso escribimos el comando de esta manera ya que usar la ruta completa nos asegura que estemos ejecutando en el ejecutable correcto (mysql.exe), puerto lo colocamos ya que MySQL no se está ejecutando en el puerto predeterminado (3306), también evitamos errores. Luego ingresamos la contraseña de MySQL.

```

C:\Windows\system32>net start MySQL80
El servicio solicitado ya ha sido iniciado.

Puede obtener más ayuda con el comando NET HELPMSG 2182.

C:\Windows\system32>"C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql.exe" -u root -p --port=3307
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 26
Server version: 8.0.39 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
  
```

A continuación, creamos la base de datos llamada emadopcionMascotas para ellos usamos el siguiente comando: **CREATE DATABASE EmAdopcion\_Mascotas;**

Verificamos que la base de datos ha sido creada con éxito con este comando: **SHOW DATABASES;**

```

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| emadopcion_mascotas |
| information_schema |
| mysql |
| performance_schema |
| sakila |
| sys |
| world |
+-----+
7 rows in set (0.66 sec)

mysql>
  
```

Para facilitar la administración de la base de datos usaremos una herramienta gráfica DBeaver. Entonces abrimos DBeaver seleccionamos nueva conexión, como el tipo de base de datos elegimos MySQL.

Especificamos los campos de conexión así:

**Servidor:** localhost

**Puerto:** 3307

**Base de datos:** emadopcion\_mascotas

**Usuario:**

**Contraseña:**

Luego hacemos click en probar conexión para hacer la verificación de que todo está correcto y luego finalizar.

Conectar a base de datos

**Connection Settings**  
MySQL ajustes de conexión

General | Driver properties | SSH | SSL | + Network configurations...

**Server**  
Connect by: ☒ Host ☐ URL  
URL: jdbc:mysql://localhost:3307/emadopcion\_mascotas  
Server Host: localhost Port: 3307  
Database: emadopcion\_mascotas

**Authentication (Database Native)**  
Nombre de usuario: root  
Contraseña: ..... ☒ Save password

**Advanced**  
Server Time Zone: Auto-detect  
Local Client: MySQL Binaries

[Connection variables information](#) [Database documentation](#) Connection details (name, type, ... )

Driver name: MySQL Driver Settings Licencia del driver

Probar conexión ... Anterior Siguiente Finalizar Cancelar

Ahora que ya tenemos la conexión, lo siguiente es crear las tablas para la base de datos y así poder gestionar las mascotas y las solicitudes de adopción.

En DBeaver seleccionamos la base de datos emadopcion\_mascotas y ejecutamos lo siguiente, para crear cada una de las tablas de nuestra base de datos.

vamos creando una a una cada tabla

```
CREATE TABLE mascotas (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  especie ENUM('perro', 'gato', 'conejo', 'hamster', 'ave', 'pez') NOT NULL,
  raza VARCHAR(100),
  edad INT NOT NULL,
  sexo ENUM('macho', 'hembra'),
  estado ENUM('disponible', 'adoptado') DEFAULT 'disponible',
  foto VARCHAR(255),
  fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

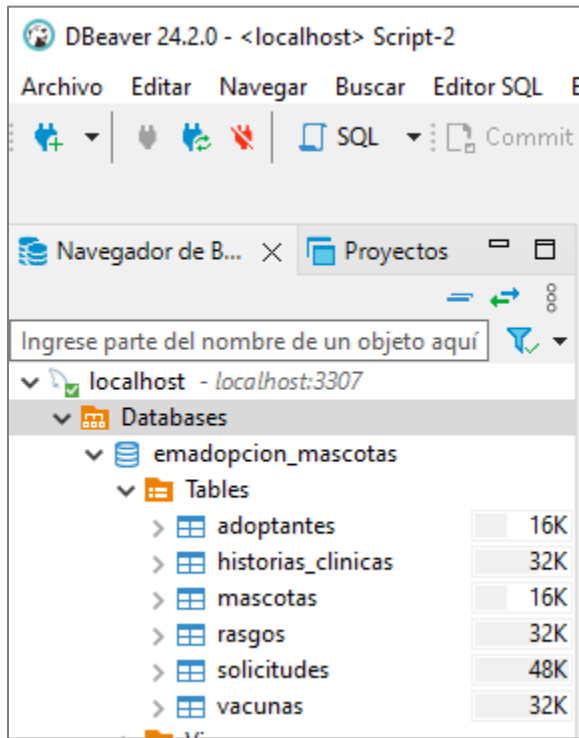
```
CREATE TABLE adoptantes (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  telefono VARCHAR(15),
  email VARCHAR(100),
  direccion VARCHAR(255),
  fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TABLE solicitudes (
  id INT AUTO_INCREMENT PRIMARY KEY,
  mascota_id INT,
  adoptante_id INT,
  fecha_solicitud TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  estado ENUM('pendiente', 'aprobada', 'rechazada') DEFAULT 'pendiente',
  FOREIGN KEY (mascota_id) REFERENCES mascotas(id),
  FOREIGN KEY (adoptante_id) REFERENCES adoptantes(id)
);
```

```
CREATE TABLE vacunas (
  id INT AUTO_INCREMENT PRIMARY KEY,
  mascota_id INT,
  nombre_vacuna VARCHAR(100),
  fecha_vacunacion DATE,
  FOREIGN KEY (mascota_id) REFERENCES mascotas(id)
);
```

```
CREATE TABLE rasgos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  mascota_id INT,
  rasgo VARCHAR(100),
  FOREIGN KEY (mascota_id) REFERENCES mascotas(id)
);
```

```
CREATE TABLE historias_clinicas (
  id INT AUTO_INCREMENT PRIMARY KEY,
  mascota_id INT,
  fecha DATE,
  observaciones TEXT,
  FOREIGN KEY (mascota_id) REFERENCES mascotas(id)
);
```



## 2. Proceso de construcción

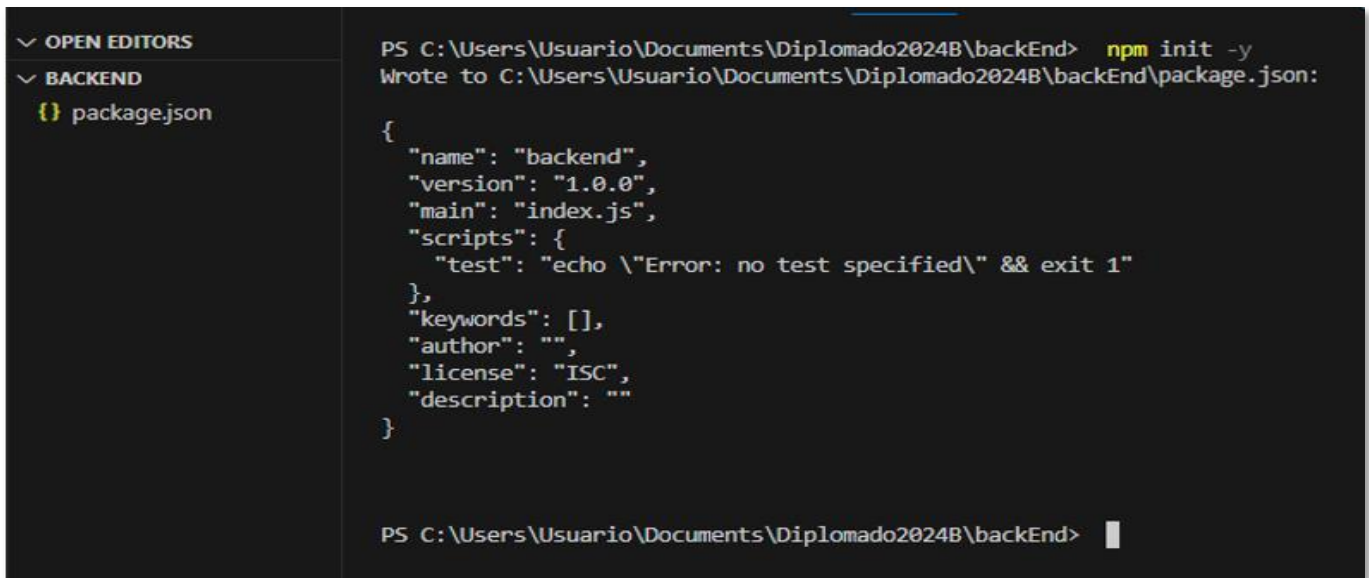
Para la inicialización del proyecto con Node.js, debemos tener instalado node.js en nuestro sistema. Para asegurarnos vamos a verificar usando el siguiente comando en la terminal. **node -v**

```

Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.19045.4894]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Windows\system32> node -v
v20.17.0
  
```

Una vez que ya hemos verificado que tenemos instalado **node.js**, creamos una carpeta para nuestro proyecto luego la abrimos desde nuestro editor Visual studio code, una vez que ya estemos en la carpeta de nuestro proyecto abrimos una nueva terminal ahí miramos como ya está seleccionada la carpeta donde está nuestro proyecto y ponemos el siguiente comando **npm init -y** esto crea el archivo **package.json**



```

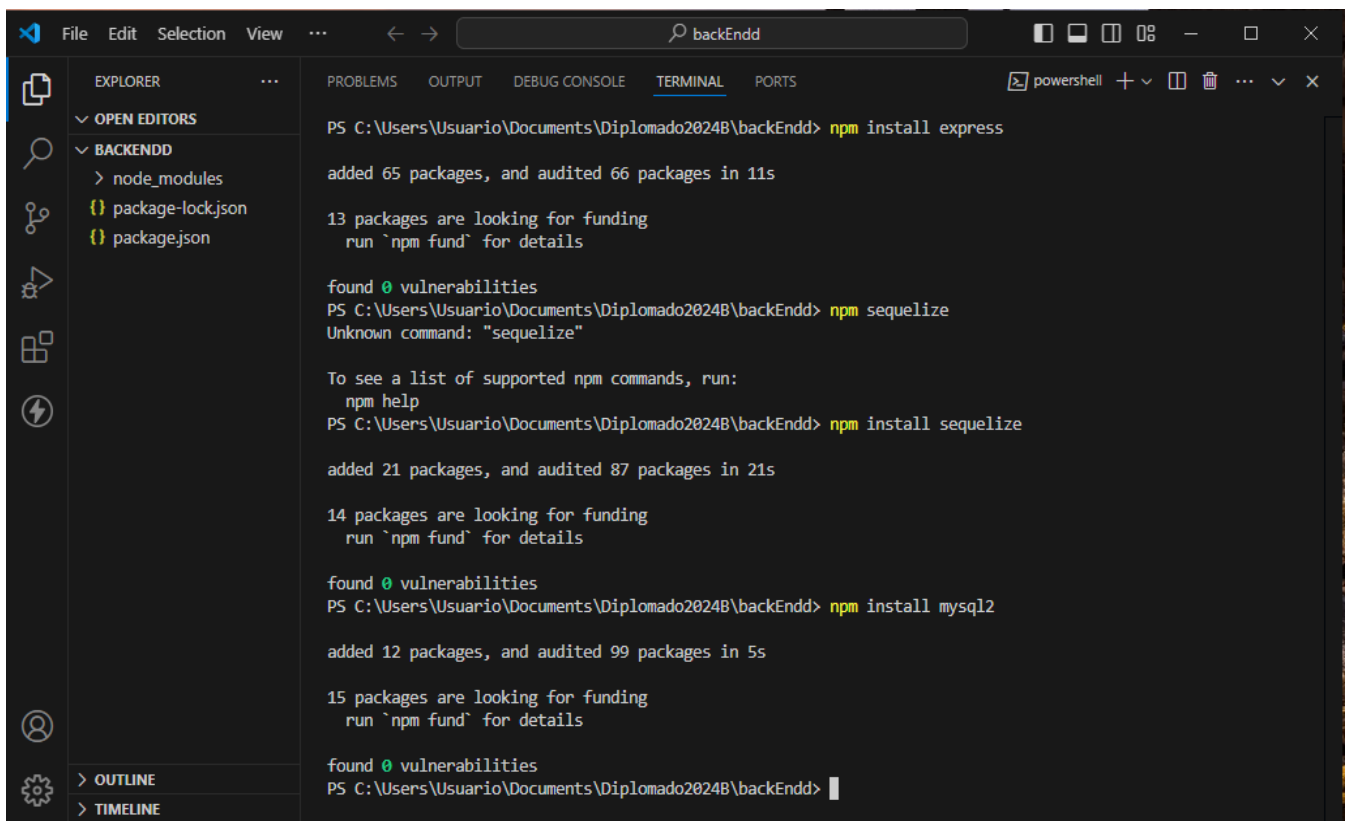
PS C:\Users\Usuario\Documents\Diplomado2024B\backEnd> npm init -y
Wrote to C:\Users\Usuario\Documents\Diplomado2024B\backEnd\package.json:

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

PS C:\Users\Usuario\Documents\Diplomado2024B\backEnd>

```

Ahora instalemos Express y Sequelize para la creación de la API y la interacción con la base de datos. Utilizaremos el siguiente comando:  
**npm install express sequelize mysql2**



```

PS C:\Users\Usuario\Documents\Diplomado2024B\backEnd> npm install express

added 65 packages, and audited 66 packages in 11s

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

found 0 vulnerabilities
PS C:\Users\Usuario\Documents\Diplomado2024B\backEnd> npm sequelize
Unknown command: "sequelize"

To see a list of supported npm commands, run:
  npm help
PS C:\Users\Usuario\Documents\Diplomado2024B\backEnd> npm install sequelize

added 21 packages, and audited 87 packages in 21s

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

found 0 vulnerabilities
PS C:\Users\Usuario\Documents\Diplomado2024B\backEnd> npm install mysql2

added 12 packages, and audited 99 packages in 5s

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

found 0 vulnerabilities
PS C:\Users\Usuario\Documents\Diplomado2024B\backEnd>

```

Ahora para que recarguemos automáticamente el servidor instalamos Nodemon con el siguiente comando:  
**npm install --save-dev nodemon**

```

found 0 vulnerabilities
PS C:\Users\Usuario\Documents\Diplomado2024B\backEndd> npm install --save-dev nodemon

added 28 packages, and audited 127 packages in 5s

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

found 0 vulnerabilities
PS C:\Users\Usuario\Documents\Diplomado2024B\backEndd>

```

Ahora hacemos la configuración de la conexión con la base de datos, creamos el **archivo.js**

```

import Sequelize from 'sequelize';

const db = new Sequelize("EmAdopcion_Mascotas","mascotasdi","mascotas2024", {
  dialect: "mysql", // motor BD
  host: "localhost", // dirección IP
  port: 3307
});

export { db };

```

Este archivo nos ayuda a establecer la conexión entre la aplicación backend y la base de datos MySQL que ya hemos creado previamente.

Para configurar la conexión a la base de datos tenemos que definir lo siguiente:

**host** (Dirección del servidor donde está alojada la base de datos), **Dialect** (tipo de base de datos MySQL), **port** (se especifica el puerto que este en uso de MySQL).



Ahora definiremos los **modelos** que representan las tablas de la base de datos dentro de nuestro proyecto usando Sequelize, estos nos permiten trabajar con las tablas de la base de datos de una forma más fácil.

Creamos un archivo llamado **mascotaModelo.js** en el cual definiremos la escritura de la tabla de mascotas en Sequelize y definimos los campos del modelo para la tabla mascotas.

```
import Sequelize from 'sequelize';
import {db} from "../../database/conexion.js";

const mascotas = db.define('mascotas', {
  id: {
    type: Sequelize.INTEGER, // tipo entero
    allowNull: false, // no permite nulos
    autoIncrement: true, // es auto incrementable
    primaryKey: true // es la llave primaria
  },
  nombre: {
    type: Sequelize.STRING,
    allowNull: false // Cambiado a false, ya que el nombre no puede estar vacío
  },
  especie: {
    type: Sequelize.ENUM('perro', 'gato', 'conejo', 'hamster', 'ave', 'pez' ), //
    allowNull: false // No puede estar vacío
  },
  raza: {
    type: Sequelize.STRING,
    allowNull: true // Puede estar vacío
  },
  edad: {
    type: Sequelize.INTEGER,
    allowNull: false // Cambiado a false, ya que la edad es obligatoria
  },
  sexo: {
    type: Sequelize.ENUM('macho', 'hembra'), // Género
    allowNull: true // Puede estar vacío
  },
  estado: {
    type: Sequelize.ENUM('disponible', 'adoptado'), // Estado de la mascota
    defaultValue: 'disponible' // Valor por defecto
  },
  foto: {
    type: Sequelize.STRING,
    allowNull: true // Puede estar vacío
  },
  fecha_registro: {
    type: Sequelize.DATE,
    defaultValue: Sequelize.NOW // Fecha de registro por defecto
  }
});

export { mascotas }; // Exportamos el modelo
```

Ahora implementaremos las operaciones **CRUD** (crear, buscar, actualizar, eliminar) para manejar las mascotas en la base de datos.

Creamos el archivo mascotasController.js este archivo tiene las funciones controladoras las cuales serán responsables de manejar las solicitudes **HTTP** para la administración de las mascotas. Cada función está vinculada a una ruta específica llevando a cabo una operación **CRUD** en la base de datos.

```

import { mascotas } from "../modelos/mascotaModelo.js";
// Crear un recurso Mascota
const crear = (req, res) => {
  // Validar que el nombre no esté vacío
  if (!req.body.nombre) {
    return res.status(400).json({
      mensaje: 'El nombre no puede estar vacío.'
    });
  }
  const database = {
    nombre: req.body.nombre,
    edad: req.body.edad,
    especie: req.body.especie,
    raza: req.body.raza,
    sexo: req.body.sexo,
    estado: req.body.estado,
    foto: req.body.foto
  };
  // Usar Sequelize para crear el recurso en la BD
  mascotas.create(database)
    .then((resultado) => {
      res.status(201).json({
        mensaje: 'Registro de Mascota Creado con Éxito',
        mascota: resultado
      });
    })
    .catch((err) => {
      res.status(500).json({
        mensaje: `Registro de Mascota No creado ::: ${err}`
      });
    });
};

// Buscar todas las Mascotas
const buscar = (req, res) => {
  // Buscar todas las mascotas registradas
  mascotas.findAll()
    .then((resultado) => {
      res.status(200).json(resultado);
    })
    .catch((err) => {
      res.status(500).json({
        mensaje: `No se encontraron registros ::: ${err}`
      });
    });
};

// Buscar Mascota por ID
const buscarId = (req, res) => {
  const id = req.params.id;

  // Validar que el ID no esté vacío
  if (!id) {
    return res.status(400).json({
      mensaje: 'El id no puede estar vacío'
    });
  }

  // Buscar la mascota por ID
  mascotas.findByPk(id)
    .then((resultado) => {
      if (!resultado) {
        return res.status(404).json({
          mensaje: 'Mascota no encontrada'
        });
      }
      res.status(200).json(resultado);
    })
    .catch((err) => {
      res.status(500).json({
        mensaje: `Error al buscar mascota ::: ${err}`
      });
    });
};

// Actualizar Mascota
const actualizar = (req, res) => {
  const id = req.params.id;

  // Validar que se hayan proporcionado datos para actualizar
  if (!req.body.nombre && !req.body.edad) {
    return res.status(400).json({
      mensaje: 'No se encontraron Datos para Actualizar'
    });
  }

  const updates = {
    nombre: req.body.nombre,
    edad: req.body.edad,
    especie: req.body.especie,
    raza: req.body.raza,
    sexo: req.body.sexo,
    estado: req.body.estado,
    foto: req.body.foto
  };
};

```

```

// Actualizar la mascota en la BD
mascotas.update(updates, { where: { id } })
.then((resultado) => {
  if (resultado[0] === 0) { // Verifica si se actualizó alguna fila
    return res.status(404).json({
      mensaje: 'Mascota no encontrada o no se realizaron cambios'
    });
  }
  res.status(200).json({
    tipo: 'success',
    mensaje: 'Registro Actualizado'
  });
})
.catch((err) => {
  res.status(500).json({
    tipo: 'error',
    mensaje: `Error al actualizar Registro ::: ${err}`
  });
});

// Eliminar Mascota
const eliminar = (req, res) => {
  const id = req.params.id;
  // Validar que el ID no esté vacío
  if (!id) {
    return res.status(400).json({
      mensaje: 'El id es requerido para eliminar una mascota'
    });
  }

  // Eliminar la mascota en la BD
  mascotas.destroy({ where: { id } })
  .then((resultado) => {
    if (resultado === 0) {
      return res.status(404).json({
        mensaje: 'Mascota no encontrada'
      });
    }
    res.status(200).json({
      tipo: 'success',
      mensaje: 'Mascota eliminada con éxito'
    });
  })
  .catch((err) => {
    res.status(500).json({
      tipo: 'error',
      mensaje: `Error al eliminar la mascota ::: ${err}`
    });
  });
});

// Exportar los métodos del controlador
export { crear, buscar, buscarId, actualizar, eliminar };

```

Ahora creamos el archivo **mascotasRouter.js** que es donde se define las rutas para interactuar con la API de nuestro proyecto de adopción de mascotas. Estas rutas permiten que el usuario pueda enviar solicitudes **HTTP** para realizar operaciones **CRUD** sobre las mascotas.

Creamos el archivo **mascotasRouter.js** generalmente en una carpeta llamada rutas donde se crea una instancia utilizando **express.Router()**, que nos permitirá definir las rutas de forma organizada, a continuación, definimos las rutas correspondientes a cada operación **CRUD**, cada ruta invoca un método según corresponda luego se exporta el router para que pueda ser utilizado en el archivo principal donde se integra con la aplicación express.

```

import express from "express";
import { crear, buscar, buscarId, actualizar, eliminar } from "../controladores/mascotasController.js";

const routerMascotas = express.Router();

routerMascotas.get('/', (req, res) => {
  res.send('Hola Sitio de Mascotas');
});

routerMascotas.post('/crear', (req, res) => {
  //res.send('Crear Mascota');
  crear(req, res);
});

routerMascotas.get('/buscar', (req, res) => {
  //res.send('Buscar Mascota');
  buscar(req, res);
});

routerMascotas.get('/buscarId/:id', (req, res) => {
  //res.send('Buscar Mascota');
  buscarId(req, res);
});

routerMascotas.put('/actualizar/:id', (req, res) => {
  //res.send('Actualizar Mascota');
  actualizar(req, res);
});

routerMascotas.delete('/eliminar/:id', (req, res) => {
  //res.send('Eliminar Mascota');
  eliminar(req, res);
});

export { routerMascotas };

```

Ahora creamos el archivo principal de nuestro proyecto **app.js** aquí configuramos el servidor, definimos rutas, se inicializa la conexión con la base de datos y se establece la configuración necesaria para la ejecución de la API de gestión de mascotas.

Para iniciar debemos importar el **módulo express**, crear una instancia y configurar el puerto donde se ejecutará el servidor y se puede configurar para que escuche solicitudes, configuramos un **Middleware** en formato json para manejar solicitudes y respuestas de la API, después de la confirmación inicial definimos las rutas utilizando el router importado para organizar mejor el código.

Es primordial asegurar que la conexión a la base de datos está activa y sincronizada antes de iniciar el servidor.

```
src > JS app.js > ...
1  import express from 'express';
2  import { routerMascotas } from "../rutas/mascotasRouter.js";
3  import { db } from "../database/conexion.js";
4
5  // Crear instancia de Express
6  const app = express();
7
8  // Middleware para parsear JSON
9  app.use(express.json());
10 |
11 // Verificar conexión con la base de datos
12 db.authenticate().then(() => {
13   console.log('Conexión a la base de datos correcta');
14 }).catch(err => {
15   console.log(`Conexión a la base de datos incorrecta: ${err}`);
16 });
17
18 // Definir la ruta principal
19 app.get('/', (req, res) => {
20   res.send('Hola, Bienvenido al Sitio Principal');
21 });
22
23 // Llamar rutas de mascotas
24 app.use("/mascotas", routerMascotas);
25
26 // --Middleware para manejo de errores global--
27 app.use((err, req, res, next) => {
28   console.error(err.stack);
29   res.status(500).send({
30     mensaje: 'Ocurrió un error en el servidor',
31     error: err.message
32   });
33 });
34
35 // Definir el puerto del servidor
36 const PORT = 4000;
37
38 // Sincronizar la base de datos y abrir el servidor
39 db.sync({ alter: true }).then(() => { // force: false para no borrar
40   app.listen(PORT, () => {
41     console.log(`Servidor inicializado en el puerto ${PORT}`);
42   });
43 }).catch(err => {
44   console.log(`Error al sincronizar base de datos: ${err}`);
45 });
```

Vamos a hacer una verificación de rutas, utilizamos las extensiones **Rest Client** que nos permite enviar solicitudes **HTTP** directamente desde archivos de texto, y **HTTP Requests Snippets** nos proporciona fragmentos de código que facilitan la creación de solicitudes **HTTP** en varios formatos. En Visual Studio Code, estas herramientas nos permiten realizar pruebas de API desde el editor de código, facilitando el proceso de desarrollo y prueba.

```

requests.http > ...
Send Request
1 GET http://127.0.0.1:4000/mascotas/buscar HTTP/1.1
2
3
4
5
6
7
8
9
10
11
http://127...
http://127...
http://127...

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: text/html; charset=utf-8
4 Content-Length: 14
5 ETag: W/"e-KPQ2krMj+tjsWv7F/u+WpoJHNZE"
6 Date: Sun, 29 Sep 2024 01:25:13 GMT
7 Connection: keep-alive
8 keep-alive: timeout=5
9
10 Buscar Mascota

```

```

requests.http > ...
Send Request
1 GET http://127.0.0.1:4000/mascotas/buscar HTTP/1.1
2
3 ###
4 Send Request
5 POST http://127.0.0.1:4000/mascotas/crear HTTP/1.1
6 Content-Type: application/json
7 {
8   "nombre": "Max",
9   "edad": 4
10 }
11
http://127...
http://127...
http://127...

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: text/html; charset=utf-8
4 Content-Length: 13
5 ETag: W/"d-lrM35vvQq8cpbWzhURtdCNsvkU8"
6 Date: Sun, 29 Sep 2024 01:34:10 GMT
7 Connection: keep-alive
8 keep-alive: timeout=5
9
10 Crear Mascota

```

```

requests.http > ...
Send Request
1 GET http://127.0.0.1:4000/mascotas/buscar HTTP/1.1
2
3 ###
4 Send Request
5 POST http://127.0.0.1:4000/mascotas/crear HTTP/1.1
6 Content-Type: application/json
7 {
8   "nombre": "Max",
9   "edad": 4
10 }
11
12 ###
13 Send Request
14 PUT http://127.0.0.1:4000/mascotas/actualizar/2 HTTP/1.1
http://127...
http://127...
http://127...

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: text/html; charset=utf-8
4 Content-Length: 18
5 ETag: W/"12-Dit0lakjJCahs8z5J60U3FnlwkW"
6 Date: Sun, 29 Sep 2024 01:36:26 GMT
7 Connection: keep-alive
8 keep-alive: timeout=5
9
10 Actualizar Mascota

```

```

requests.http > ...
Send Request
1 GET http://127.0.0.1:4000/mascotas/buscar HTTP/1.1
2
3 ###
4 Send Request
5 POST http://127.0.0.1:4000/mascotas/crear HTTP/1.1
6 Content-Type: application/json
7 {
8   "nombre": "Max",
9   "edad": 4
10 }
11
12 ###
13 Send Request
14 PUT http://127.0.0.1:4000/mascotas/actualizar/2 HTTP/1.1
15 ###
16 Send Request
17 DELETE http://127.0.0.1:4000/mascotas/eliminar/2 HTTP/1.1
http://127...
http://127...
http://127...

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: text/html; charset=utf-8
4 Content-Length: 16
5 ETag: W/"10-9ILVKSqGjki+HUH2E5Yk0sJP50Q"
6 Date: Sun, 29 Sep 2024 01:38:27 GMT
7 Connection: keep-alive
8 keep-alive: timeout=5
9
10 Eliminar Mascota

```

3. Para realizar la verificación de las diferentes operaciones en la API utilizaremos un cliente gráfico, en éste caso usaremos **Insomnia**.

Descargamos e instalamos **Insomnia** desde su página oficial, abrimos el cliente gráfico, y vamos a verificar que el servidor **NodeJs** este corriendo. Ejecutamos el comando: **node app.js**, y nos debe aparecer un mensaje de confirmado que el servicio está en funcionamiento.

The screenshot shows the Insomnia client interface. The top bar indicates a POST request to `http://127.0.0.1:4000/mascotas/crea` with a status of **201 Created**, a response time of 462 ms, and a body size of 345 B. The left pane shows the request body in JSON format:

```
1 {
2   "nombre": "Luna",
3   "edad": 3,
4   "especie": "Gato",
5   "raza": "Persa",
6   "sexo": "hembra",
7   "estado": "Disponible",
8   "foto":
9     "https://co.pinterest.com/pin/344525440225697661/"
10 }
```

The right pane shows the response body in JSON format:

```
1 {
2   "mensaje": "Registro de Mascota Creado con Éxito",
3   "mascota": {
4     "fecha_registro": "2024-09-29T03:47:40.969Z",
5     "id": 1,
6     "nombre": "Luna",
7     "edad": 3,
8     "especie": "Gato",
9     "raza": "Persa",
10    "sexo": "hembra",
11    "estado": "Disponible",
12    "foto":
13      "https://co.pinterest.com/pin/344525440225697661/",
14    "updatedAt": "2024-09-29T03:47:40.972Z",
15    "createdAt": "2024-09-29T03:47:40.972Z"
16  }
17 }
```

The screenshot shows the Insomnia client interface. The top bar indicates a POST request to `http://127.0.0.1:4000/mascotas/crea` with a status of **201 Created**, a response time of 338 ms, and a body size of 361 B. The left pane shows the request body in JSON format:

```
1 {
2   "nombre": "Bunny",
3   "edad": 2,
4   "especie": "conejo",
5   "raza": "Mini Rex",
6   "sexo": "Hembra",
7   "estado": "Disponible",
8   "foto":
9     "https://www.expertoanimal.com/conejos/conejo-rex-mini.html"
10 }
```

The right pane shows the response body in JSON format:

```
1 {
2   "mensaje": "Registro de Mascota Creado con Éxito",
3   "mascota": {
4     "fecha_registro": "2024-09-29T04:02:26.512Z",
5     "id": 3,
6     "nombre": "Bunny",
7     "edad": 2,
8     "especie": "conejo",
9     "raza": "Mini Rex",
10    "sexo": "Hembra",
11    "estado": "Disponible",
12    "foto":
13      "https://www.expertoanimal.com/conejos/conejo-rex-mini.html",
14    "updatedAt": "2024-09-29T04:02:26.514Z",
15    "createdAt": "2024-09-29T04:02:26.514Z"
16  }
17 }
```

GET ▼ http://127.0.0.1:4000/mascotas/buscar Send ▼ 200 OK 136 ms 872 B 2 Minutes Ago ▼

Params Body Auth Headers 4 Scripts Docs Preview Headers 7 Cookies Tests 0 / 0 → Mock Con

JSON ▼

```

1 ...

```

Preview ▼

```

15 {
16   "id": 2,
17   "nombre": "Luky",
18   "especie": "perro",
19   "raza": "Labrador",
20   "edad": 5,
21   "sexo": "macho",
22   "estado": "disponible",
23   "foto":
24     "https://es.wikipedia.org/wiki/Labrador_retriever",
25   "fecha_registro": "2024-09-29T03:56:10.000Z",
26   "createdAt": "2024-09-29T03:56:10.000Z",
27   "updatedAt": "2024-09-29T03:56:10.000Z"
28 },
29 {
30   "id": 3,
31   "nombre": "Bunny",
32   "especie": "conejo",
33   "raza": "Mini Rex",
34   "edad": 2,
35   "sexo": "hembra",
36   "estado": "disponible",
37   "foto":
38     "https://www.expertoanimal.com/conejos/conejo-rex-
39     mini.html",
40   "fecha_registro": "2024-09-29T04:02:26.000Z",
41   "createdAt": "2024-09-29T04:02:26.000Z",
42   "updatedAt": "2024-09-29T04:02:26.000Z"
43 }

```

GET ▼ http://127.0.0.1:4000/mascotas/buscar Send ▼ 200 OK 166 ms 286 B Just Now ▼

Params Body Auth Headers 4 Scripts Docs Preview Headers 7 Cookies Tests 0 / 0 → Mock Con

JSON ▼

```

1 ...

```

Preview ▼

```

1 {
2   "id": 2,
3   "nombre": "Luky",
4   "especie": "perro",
5   "raza": "Labrador",
6   "edad": 5,
7   "sexo": "macho",
8   "estado": "disponible",
9   "foto":
10     "https://es.wikipedia.org/wiki/Labrador_retriever",
11   "fecha_registro": "2024-09-29T03:56:10.000Z",
12   "createdAt": "2024-09-29T03:56:10.000Z",
13   "updatedAt": "2024-09-29T03:56:10.000Z"
14 }

```

PUT ▼ http://127.0.0.1:4000/mascotas/actuali: Send ▼ 200 OK 250 ms 51 B 5 Minutes Ago ▼

Params Body Auth Headers 4 Scripts Docs Preview Headers 7 Cookies Tests 0 / 0 → Mock Con

JSON ▼

```

1 {
2   "id": 2,
3   "nombre": "Luky",
4   "especie": "perro",
5   "raza": "Labrador",
6   "edad": 5,
7   "sexo": "macho",
8   "estado": "adoptado",
9   "foto":
10     "https://es.wikipedia.org/wiki/Labrador_retriever"
11   ,
12   "fecha_registro": "2024-09-29T03:56:10.000Z",
13   "createdAt": "2024-09-29T03:56:10.000Z",
14   "updatedAt": "2024-09-29T03:56:10.000Z"
15 }

```

Preview ▼

```

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

```

GET ▼ http://127.0.0.1:4000/mascotas/buscar Send ▼ **200 OK** 116 ms 284 B Just Now ▼

Params **Body** ● Auth Headers 4 Scripts Docs **Preview** Headers 7 Cookies Tests 0 / 0 → Mock Con

JSON ▼

1 ...

Preview ▼

```
1 {
2   "id": 2,
3   "nombre": "Luky",
4   "especie": "perro",
5   "raza": "Labrador",
6   "edad": 5,
7   "sexo": "macho",
8   "estado": "adoptado",
9   "foto":
10  "https://es.wikipedia.org/wiki/Labrador_retriever",
11  "fecha_registro": "2024-09-29T03:56:10.000Z",
12  "createdAt": "2024-09-29T03:56:10.000Z",
13  "updatedAt": "2024-09-29T04:16:15.000Z"
14 }
```

DELETE ▼ http://127.0.0.1:4000/mascotas/eliminar/ Send ▼ **200 OK** 399 ms 59 B Just Now ▼

Params **Body** ● Auth Headers 4 Scripts Docs **Preview** Headers 7 Cookies Tests 0 / 0 → Mock Console

JSON ▼

1 ...

Preview ▼

```
1 {
2   "tipo": "success",
3   "mensaje": "Mascota eliminada con éxito"
4 }
```

GET ▼ http://127.0.0.1:4000/mascotas/buscarId/3 Send ▼ **404 Not Found** 72 ms 35 B Just Now ▼

Params **Body** ● Auth Headers 4 Scripts Docs **Preview** Headers 7 Cookies Tests 0 / 0 → Mock Console

JSON ▼

1 ...

Preview ▼

```
1 {
2   "mensaje": "Mascota no encontrada"
3 }
```