



Universidad de **Nariño**

TALLER BACKEND.

ELABORADO POR:

CRISTIAN FERNANDO JACOME RECALDE

UNIVERSIDAD DE NARIÑO - SEDE IPIALES

FACULTAD DE INGENIERIA DE SISTEMAS

X SEMESTRE

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

IPIALES – NARIÑO 2024



Universidad de **Nariño**

TALLER BACKEND.

ELABORADO POR:

CRISTIAN FERNANDO JACOME RECALDE

PRESENTADO A:

MG. VICENTE AUX REVELO

UNIVERSIDAD DE NARIÑO - SEDE IPIALES

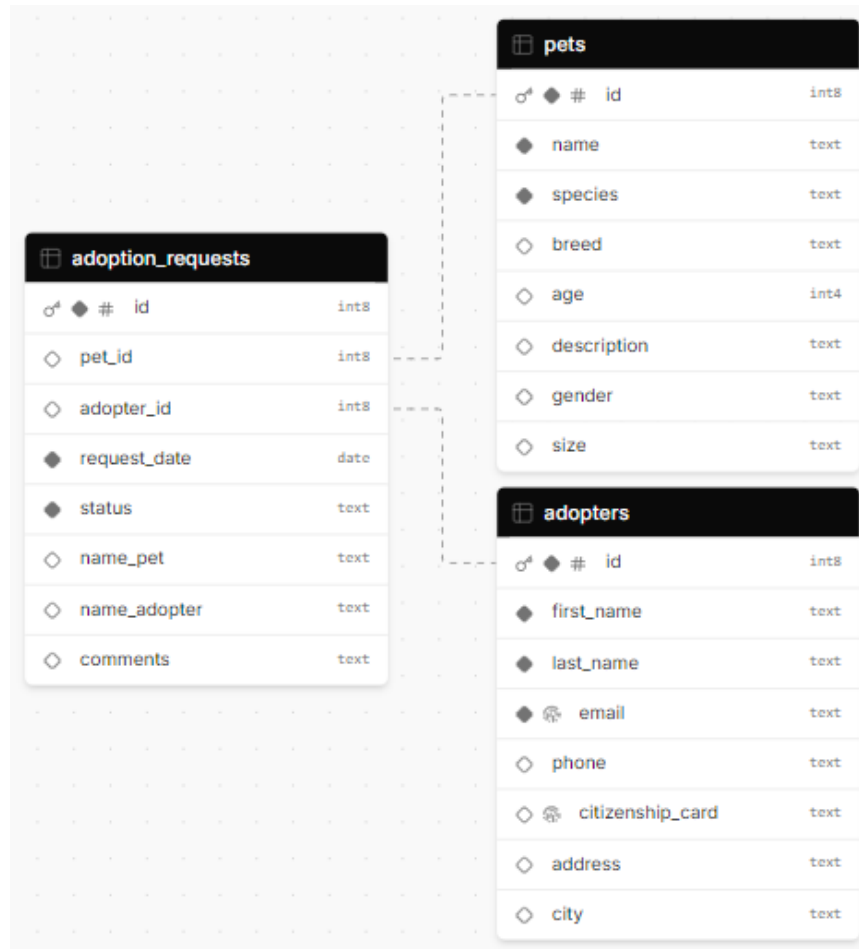
FACULTAD DE INGENIERIA DE SISTEMAS

X SEMESTRE

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

IPIALES – NARIÑO 2024

1. **Base de datos que permita llevar el registro de una empresa de adopción de mascotas:** Incluye tres tablas principales: **pets**, **adopters**, y **adoption_requests**. Cada tabla tiene su propósito y relaciones para cubrir las necesidades de este sistema.



A. Tabla pets (Mascotas): Esta tabla almacena la información de las mascotas que están disponibles para adopción.

- **id:** Es la clave primaria única de la mascota. Este valor es generado automáticamente como un número incremental.
- **name:** Nombre de la mascota. Es obligatorio.
- **species:** Especie de la mascota (perro, gato, etc.). Es obligatorio.
- **breed:** Raza de la mascota. Es opcional.
- **age:** Edad de la mascota en años. Es opcional.
- **description:** Descripción general o notas adicionales sobre la mascota. Es opcional.
- **gender:** El género de la mascota (macho, hembra). Este campo es opcional.

- **size:** El tamaño de la mascota (pequeño, mediano, grande). Este campo es opcional.

B. Tabla adopters (Adoptantes): Esta tabla almacena la información de las personas interesadas en adoptar mascotas.

- **id:** Clave primaria única del adoptante, generada automáticamente.
- **first_name:** Nombre del adoptante. Es obligatorio.
- **last_name:** Apellido del adoptante. Es obligatorio.
- **email:** Correo electrónico único del adoptante, obligatorio. No pueden existir dos registros con el mismo email.
- **phone:** Número de teléfono del adoptante. Es opcional.
- **citizenship_card:** Número de cédula o documento de identidad. Es opcional, pero debe ser único si se proporciona.
- **address:** Dirección del adoptante. Este campo es opcional.
- **city:** La ciudad donde reside el adoptante. Este campo es opcional.

C. Tabla adoption_requests (Solicitudes de adopción): Esta tabla registra las solicitudes de adopción, relacionando a los adoptantes con las mascotas que desean adoptar.

- **id:** Clave primaria única de la solicitud, generada automáticamente.
- **pet_id:** Clave foránea que hace referencia al campo id de la tabla pets, es decir, a la mascota que se desea adoptar.
- **adopter_id:** Clave foránea que hace referencia al campo id de la tabla adopters, es decir, a la persona que está realizando la solicitud de adopción.
- **request_date:** Fecha en la que se realizó la solicitud de adopción. Se genera automáticamente con la fecha actual por defecto.
- **status:** Estado de la solicitud de adopción. Solo puede ser uno de los tres valores: Pending, Approved, o Rejected (Pendiente, Aprobada, o Rechazada).
- **name_pet:** El nombre de la mascota relacionada con la solicitud de adopción. Es un campo opcional.
- **name_adopter:** El nombre del adoptante que realiza la solicitud. Es un campo opcional.
- **comments:** Comentarios adicionales sobre la solicitud de adopción. Es un campo opcional.
- **FOREIGN KEY (pet_id):** Establece una relación con la tabla pets, eliminando la solicitud de adopción si la mascota es eliminada (ON DELETE CASCADE).

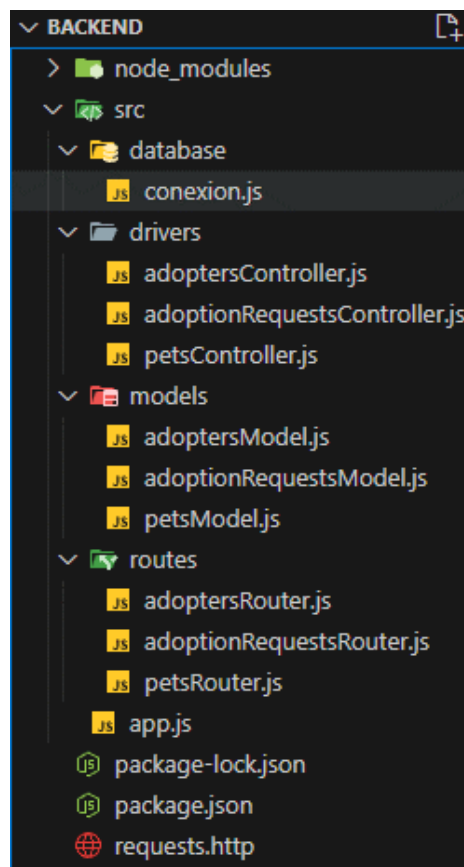
- **FOREIGN KEY (adopter_id):** Establece una relación con la tabla adopters, eliminando la solicitud de adopción si el adoptante es eliminado (ON DELETE CASCADE).

Relaciones entre tablas:

- **Relación entre pets y adoption_requests:** Una mascota puede tener múltiples solicitudes de adopción (relación uno a muchos). Cada solicitud está vinculada a una mascota específica.
- **Relación entre adopters y adoption_requests:** Un adoptante puede hacer múltiples solicitudes de adopción (relación uno a muchos). Cada solicitud está vinculada a un adoptante específico.

2. Desarrollar una aplicación Backend implementada en NodeJS y ExpressJS:

I. Estructura del proyecto:



- **Database - Conexion.js:** Este código establece una conexión a una base de datos MySQL usando Sequelize. Se conecta a la base de datos llamada **"pets"**. El dialecto usado es MySQL, y la conexión se exporta como db para ser utilizada en otras partes del proyecto, facilitando la interacción con la base de datos mediante modelos y consultas.

```
src > database > JS conexion.js > ...
1  import Sequelize from "sequelize";
2
3  const db = new Sequelize("pets","mascotasaux","mascotas2024",{
4      dialect: "mysql",
5      host: "localhost"
6  });
7
8  export {db}
9
```

App.js: Se crea una instancia de Express y se habilita el middleware para procesar datos en formato JSON. Se establece la conexión a la base de datos, verificando su autenticidad y manejando posibles errores. Además, se define una ruta principal que responde con un mensaje de saludo y se integran las rutas específicas para las mascotas a través del enrutador importado. El servidor se ejecuta en el puerto 4000, y, al sincronizar correctamente la base de datos, se inicia el servicio, permitiendo que la aplicación comience a recibir solicitudes.

```
import express from "express";
import { routerPets } from "../routes/petsRouter.js";
import { routerAdopters } from "../routes/adoptersRouter.js";
import { routerAdoptionRequests } from "../routes/adoptionRequestsRouter.js";
import { db } from "../database/conexion.js";

//Crear instancia de Express
const app = express();

//Middleware JSON
app.use(express.json());

//Verificar Conexion Base Datos
db.authenticate().then(()=>{
  console.log(`Conexion a Base de datos correcta`);
}).catch(err=>{
  console.log(`Conexion a Base de datos incorrecta ${err}`);
});

//Definir Rutas
app.get('/', (req, res) => {
  res.send('Hola Sitio Principal');
});

//Llamar rutas de Mascotas
app.use("/mascotas",routerPets);

//Llamar rutas de Adoptantes
app.use("/adoptantes",routerAdopters);

//Llamar rutas de Solicitud de Adopcion
app.use("/solicitudes",routerAdoptionRequests);

//Puerto de Servidor
const PORT=4000;

db.sync().then(()=>{
  //Abri servicio e iniciar el Servidor
  app.listen(PORT,()=>{
    console.log(`Servidor Inicializado en el puerto ${PORT}`);
  })
}).catch(err=>{
  console.log(`Error al Sincronizar base de datos ${err}`);
});
```

- II. **Models:** Donde se implementan las rutas y la lógica relacionada con la gestión de los modelos (crear, buscar, actualizar y eliminar información).

Ejemplo tabla pets:

- **Drivers - petsController.js:** Contiene las funciones que gestionan las operaciones relacionadas con el modelo. En este archivo se definen métodos para crear nuevos registros, buscar todos los registros, buscar registros por su ID específico, así como funciones para eliminar y actualizar la información de los registros existentes.
- **Función Crear Mascotas:** Primero, valida que el nombre de la mascota no esté vacío; si es así, responde con un mensaje de error y un estado 400. Luego, recoge la información de la mascota desde el cuerpo de la solicitud y la organiza en un objeto "dataset". Se utiliza Sequelize para intentar crear el registro en la base de datos, si la creación es exitosa, envía una respuesta con un mensaje de éxito y un estado 200, En caso de un error, captura la excepción y responde con un mensaje de error y un estado 500.

```
import { pets } from "../models/petsModel.js";

//Funcion Crear Mascotas

const crear = (req, res) => {

  //validar
  if (!req.body.name) {
    res.status(400).send({
      mensaje: "El nombre no puede estar vacío."
    });
    return;
  }

  const dataset = {
    name: req.body.name,
    species: req.body.species,
    breed: req.body.breed,
    age: req.body.age,
    gender: req.body.gender,
    size: req.body.size,
    description: req.body.description
  }

  //usar sequelize para crear el recurso en la base de datos

  pets.create(dataset).then((resultado) => {
    res.status(200).json({
      mensaje: "Registro de Mascota Creado con Exito"
    });
  }).catch((err) => {
    res.status(500).json({
      mensaje: `Registro de Mascota No Creado ::: ${err}`
    });
  });
}
```


- **Función Buscar Mascotas:** Utiliza el método `findAll` de Sequelize para realizar la consulta y, si se encuentran registros, envía una respuesta con un estado 200 y los datos en formato JSON. En caso de que ocurra un error durante la búsqueda, captura la excepción y responde con un mensaje de error junto con un estado 500, indicando que no se pudieron encontrar los registros.

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

- **Función BuscarId Mascotas:** Primero, obtiene el ID de los parámetros de la solicitud y verifica que no esté vacío; si es nulo, responde con un estado 400 y un mensaje de error correspondiente. Si el ID es válido, utiliza el método `findByPk` de Sequelize para buscar la mascota correspondiente. Si se encuentra el registro, devuelve una respuesta con un estado 200 y los datos en formato JSON. En caso de error durante la búsqueda, captura la excepción y responde con un estado 500, junto con un mensaje que indica que no se encontraron registros.

```
//Funcion BuscarId Mascotas
const buscarID = (req, res) => {
  const id = req.params.id;
  if (id == null) {
    res.status(400).json({
      mensaje: "El id no puede estar vacio"
    });
    return;
  } else {
    pets.findByPk(id).then((resultado) => {
      res.status(200).json(resultado);
    }).catch((err) => {
      res.status(500).json({
        mensaje: `No se encontraron registros ::: ${err}`
      });
    });
  }
};
```

- **Función Actualizar Mascotas:** Se obtiene el ID de la mascota a partir de los parámetros de la solicitud y verifica si al menos uno de los campos relevantes (nombre o especie) está presente en el cuerpo de la solicitud; si no se encuentran datos para actualizar, responde con un estado 400 y un mensaje de error. Si hay datos válidos, utiliza el método update de Sequelize para realizar la actualización, especificando los nuevos valores y el ID correspondiente. Si la actualización es exitosa, devuelve una respuesta con un estado 200 y un mensaje de éxito. En caso de que ocurra un error durante el proceso, captura la excepción y responde con un estado 500, junto con un mensaje de error que indica el problema.

```
//Funcion Actualizar Mascotas

const actualizar = (req, res) => {
  const id = req.params.id
  if (!req.body.name && !req.body.species) {
    res.status(400).json({
      mensaje: "No se encontraron Datos para Actualizar."
    });
    return;
  } else {
    const name = req.body.name;
    const species = req.body.species;
    const breed = req.body.breed;
    const age = req.body.age;
    const gender = req.body.gender;
    const size = req.body.size;
    const description = req.body.description;
    pets.update({ name, species, breed, age, gender, size, description },
      { where: { id } }).then((resultado) => {
        res.status(200).json({
          tipo: 'success',
          mensaje: "Registro actualizado."
        });
      }).catch((err) => {
        res.status(500).json({
          tipo: 'error',
          mensaje: `Error al actualizar registro. ::: ${err}`
        });
      });
  }
}
```

- **Función Borrar Mascota:** Se obtiene el ID de los parámetros de la solicitud y verifica si está presente; si no, responde con un estado 400 y un mensaje de error indicando que el ID no puede estar vacío. Si el ID es válido, utiliza el método destroy de Sequelize para intentar eliminar el registro correspondiente. Si la eliminación es exitosa y se elimina un registro, devuelve una respuesta con un estado 200 y un mensaje de éxito. En caso de que no se encuentre la mascota, responde con un estado 404 y un mensaje de error. Si ocurre un error durante el proceso de eliminación, captura la excepción y responde con un estado 500, junto con un mensaje que indica el problema.

```
// Función Borrar mascota
const borrar = (req, res) => {
  const id = req.params.id;

  if (!id) {
    res.status(400).json({
      mensaje: "El id no puede estar vacío."
    });
    return;
  }

  pets.destroy({ where: { id } })
    .then((resultado) => {
      if (resultado === 1) {
        res.status(200).json({
          tipo: 'success',
          mensaje: "Registro de mascota eliminado con éxito."
        });
      } else {
        res.status(404).json({
          tipo: 'error',
          mensaje: "Mascota no encontrada."
        });
      }
    })
    .catch((err) => {
      res.status(500).json({
        tipo: 'error',
        mensaje: `Error al eliminar registro. ::: ${err}`
      });
    });
};
```

- **export {crear, buscar, buscarID, actualizar, borrar}:** Al exportar estas funciones, el archivo se convierte en un módulo que proporciona las operaciones básicas para gestionar mascotas en el backend, como crear, buscar, actualizar y eliminar registros.
- **Models – petModel.js:** Contiene la definición del modelo pets para gestionar la tabla de mascotas en la base de datos utilizando Sequelize. En este archivo, se importan las bibliotecas necesarias, incluida la conexión a la base de datos. Finalmente, el modelo se exporta para ser utilizado en otras partes de la aplicación, facilitando las operaciones CRUD sobre los registros de mascotas.

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

const pets = db.define("pets", {
  id: {
    type: Sequelize.INTEGER,
    allowNull: false,
    autoIncrement: true,
    primaryKey: true
  },
  name: {
    type: Sequelize.STRING,
    allowNull: true
  },
  species: {
    type: Sequelize.STRING,
    allowNull: true
  },
  breed: {
    type: Sequelize.STRING,
    allowNull: true
  },
  age: {
    type: Sequelize.INTEGER,
    allowNull: true
  },
  gender: {
    type: Sequelize.STRING,
    allowNull: true
  },
  size: {
    type: Sequelize.STRING,
    allowNull: true
  },
  description: {
    type: Sequelize.TEXT,
    allowNull: true
  }
}, {
  timestamps: false // Deshabilita 'createdAt' y 'updatedAt'
});

export {pets}
```

- **Routes – petsRouter.js:** Se definen las rutas relacionadas con las operaciones de gestión de mascotas en la aplicación. Utilizando un enrutador de Express, se establece diferentes endpoints para interactuar con el modelo de mascotas, permitiendo realizar acciones como crear una nueva mascota, buscar todas las mascotas, buscar una mascota específica por su ID, actualizar los detalles de una mascota y eliminar un registro de mascota.

Cada ruta está asociada a una función de controlador correspondiente (como las definidas en `petsController.js`), lo que facilita la separación de la lógica de la aplicación y promueve una estructura organizada. Al exportar el enrutador, permite que otras partes de la aplicación, como el archivo principal del servidor, lo integren y lo utilicen para manejar las solicitudes HTTP relacionadas con las mascotas.

```
import express from "express";
import {crear,buscar,buscarID,actualizar,borrar} from "../drivers/petsController.js";

const routerPets = express.Router();

routerPets.get('/', (req,res) =>{
  res.send('Hola sitio de mascotas');
})

routerPets.post('/crear', (req,res) =>{
  crear(req,res);
})

routerPets.get('/buscar', (req,res) =>{
  buscar(req,res);
})

routerPets.get('/buscarId', (req,res) =>{
  buscarID(req,res);
})

routerPets.put('/actualizar/:id', (req,res) =>{
  actualizar(req,res);
})

routerPets.delete('/borrar/:id', (req,res) =>{
  borrar(req,res);
})

export {
  routerPets
}
```

- **Requests:** Facilita la prueba permitiendo enviar solicitudes HTTP directamente desde un entorno de desarrollo, se incluye solicitudes para crear, buscar, actualizar y eliminar registros de mascotas. Este enfoque es útil para probar rápidamente los endpoints de la API sin necesidad de crear una interfaz de usuario o utilizar herramientas externas como Postman.

```
//Requests Find
Send Request
GET http://127.0.0.1:4000/mascotas/buscar HTTP/1.1

//Requests FindId
###
Send Request
GET http://127.0.0.1:4000/mascotas/buscarid/1 HTTP/1.1

//Requests Create
###
Send Request
POST http://127.0.0.1:4000/mascotas/crear HTTP/1.1
Content-Type: application/json

{
  "name": "Bongo",
  "species": "Perro",
  "breed": "Mestizo",
  "age": 7,
  "gender": "Macho",
  "size": "Grande",
  "description": "Color Blanco"
}

//Requests Update
###
Send Request
PUT http://127.0.0.1:4000/mascotas/actualizar/2 HTTP/1.1
Content-Type: application/json

{
  "name": "Tini",
  "species": "Gato",
  "breed": "Ruso",
  "age": 5,
  "gender": "Hembra",
  "size": "Pequeño",
  "description": "Color Gris"
}

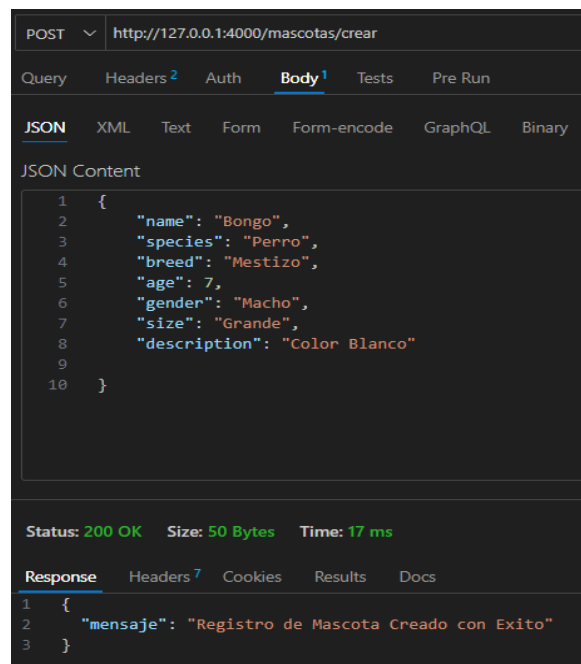
//Requests Delete
###
Send Request
DELETE http://127.0.0.1:4000/mascotas/borrar/2 HTTP/1.1
```

De manera similar a la tabla “**pets**”, los mismos pasos se repetirán para las tablas “**adopters**” y “**adoption_requests**”. Esto incluye la creación de controladores, modelos y rutas específicos para cada tabla, donde se implementarán las funciones para crear, buscar, actualizar y eliminar registros. Los controladores se encargarán de gestionar la lógica de negocio, los modelos definirán la estructura de los datos y sus relaciones, y las rutas conectarán las solicitudes HTTP con las funciones correspondientes.

- 3. Realizar verificación de las diferentes operaciones a través de un cliente gráfico:** Para asegurar el correcto funcionamiento de las diferentes operaciones (crear, buscar, actualizar y eliminar registros), es necesario realizar una verificación a través de un cliente gráfico como “Thunder Client”. Esta herramienta, integrada en Visual Studio Code, permite enviar solicitudes HTTP a las rutas definidas en el servidor y observar las respuestas que devuelve la API. De esta manera, se puede comprobar que las rutas están correctamente configuradas y que los controladores y modelos gestionan los datos adecuadamente.

➤ Modulo Mascotas

- **Crear Mascota:** Se hace uso del verbo HTTP POST.



The screenshot shows the Thunder Client interface. The top bar indicates a POST request to `http://127.0.0.1:4000/mascotas/crear`. The 'Body' tab is selected, showing a JSON payload:

```
{  "name": "Bongo",  "species": "Perro",  "breed": "Mestizo",  "age": 7,  "gender": "Macho",  "size": "Grande",  "description": "Color Blanco"}
```

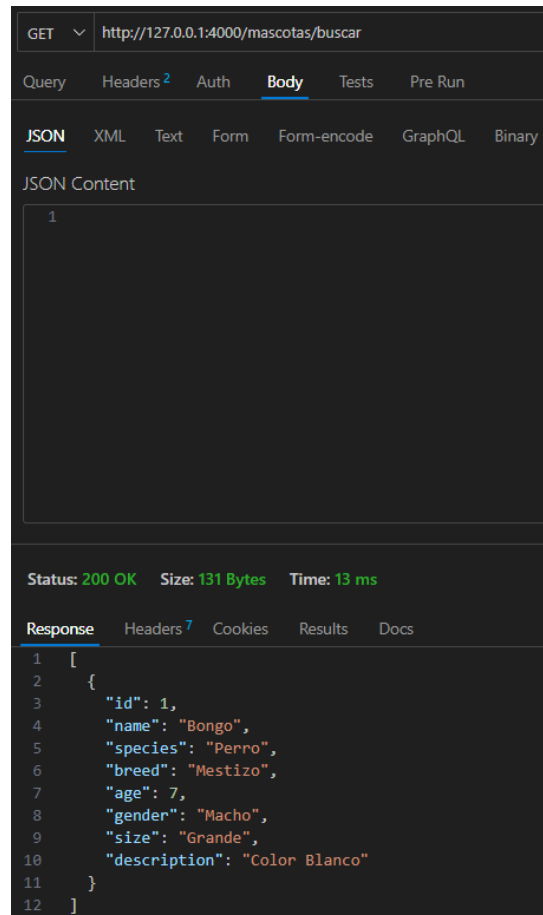
. Below the body, the status is `200 OK`, size is `50 Bytes`, and time is `17 ms`. The 'Response' tab is selected, showing a JSON response:

```
{  "mensaje": "Registro de Mascota Creado con Exito"}
```

.

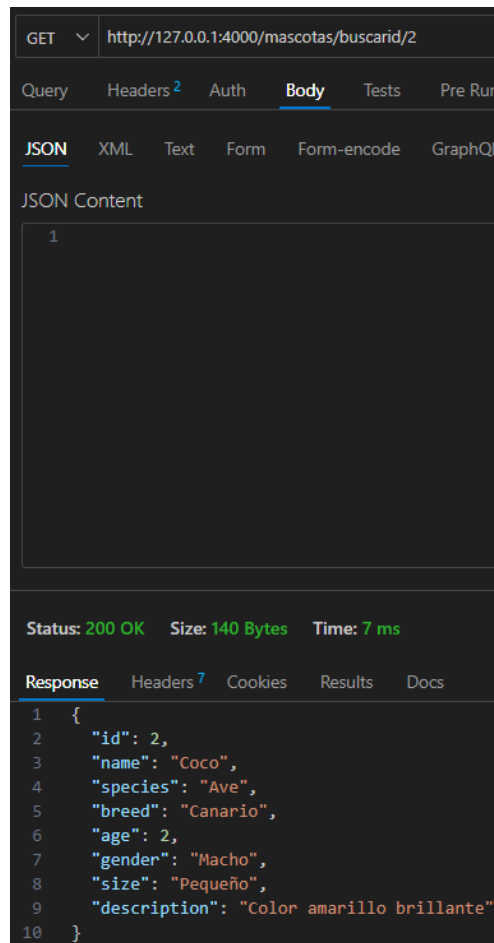
id	name	species	breed	age	description	gender	size
1	Bongo	Perro	Mestizo	7	Color Blanco	Macho	Grande

- **Buscar mascota: Se hace uso del verbo HTTP GET.**



123 id	A-z name	A-z species	A-z breed	123 age	A-z description	A-z gender	A-z size
1	Bongo	Perro	Mestizo	7	Color Blanco	Macho	Grande

- **Buscar mascota por ID: Se hace uso del verbo HTTP GET.**



123 id	A-z name	A-z species	A-z breed	123 age	A-z description	A-z gender	A-z size
1	Bongo	Perro	Mestizo	7	Color Blanco	Macho	Grande
2	Coco	Ave	Canario	2	Color amarillo brillante	Macho	Pequeño

- **Actualizar Mascota:** Se hace uso del verbo HTTP PUT.

PUT ▼ http://127.0.0.1:4000/mascotas/actualizar/1

Query Headers ² Auth Body¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content

```

1  {
2      "name": "Luna",
3      "species": "Gato",
4      "breed": "Siames",
5      "age": 3,
6      "gender": "Hembra",
7      "size": "Pequeño",
8      "description": "Color gris con manchas blancas"
9  }
10

```

Status: 200 OK Size: 52 Bytes Time: 15 ms

Response Headers ⁷ Cookies Results Docs

```

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

```

123 id ▼	A-z name ▼	A-z species ▼	A-z breed ▼	123 age ▼	A-z description ▼	A-z gender ▼	A-z size ▼
1	Luna	Gato	Siames	3	Color gris con manchas blancas	Hembra	Pequeño
2	Coco	Ave	Canario	2	Color amarillo brillante	Macho	Pequeño

- **Eliminar Mascota:** Se hace uso del verbo HTTP DELETE.

DELETE

▼

http://127.0.0.1:4000/mascotas/borrar/2

Query

Headers²

Auth

Body

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

1

Status: 200 OK Size: 72 Bytes Time: 11 ms

Response

Headers⁷

Cookies

Results

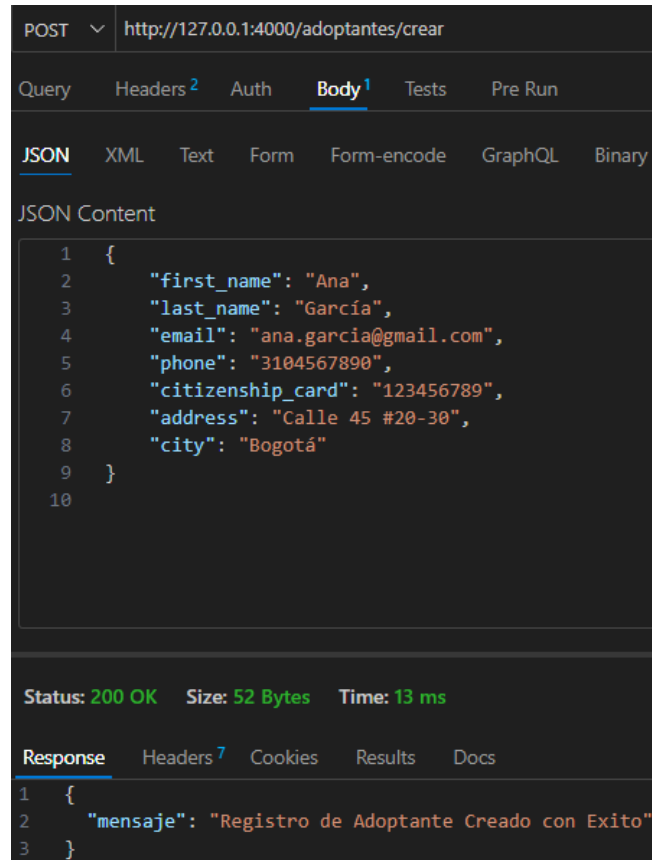
Docs

1 {
2 "tipo": "success",
3 "mensaje": "Registro de mascota eliminado con éxito."
4 }

123 id ▼	A-z name ▼	A-z species ▼	A-z breed ▼	123 age ▼	A-z description ▼	A-z gender ▼	A-z size ▼
1	Luna	Gato	Siames	3	Color gris con manchas	Hembra	Pequeño

➤ **Modulo Adoptantes**

- **Crear Adoptante: Se hace uso del verbo HTTP POST.**



id	first_name	last_name	email	phone	citizenship_card	address	city
1	Ana	García	ana.garcia@gmail.com	3104567890	123456789	Calle 45 #20-30	Bogotá

- **Buscar Adoptante: Se hace uso del verbo HTTP GET.**

GET http://127.0.0.1:4000/adoptantes/buscar

Query Headers 2 Auth **Body** Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content

```
1
```

Status: 200 OK Size: 179 Bytes Time: 6 ms

Response Headers 7 Cookies Results Docs

```
1 [
2   {
3     "id": 1,
4     "first_name": "Ana",
5     "last_name": "García",
6     "email": "ana.garcia@gmail.com",
7     "phone": "3104567890",
8     "citizenship_card": "123456789",
9     "address": "Calle 45 #20-30",
10    "city": "Bogotá"
11  }
12 ]
```

id	first_name	last_name	email	phone	citizenship_card	address	city
1	Ana	García	ana.garcia@gmail.com	3104567890	123456789	Calle 45 #20-30	Bogotá

- **Buscar Adoptante por ID:** Se hace uso del verbo HTTP GET.

The screenshot shows a web browser's developer tools interface. The top bar indicates a GET request to `http://127.0.0.1:4000/adoptantes/buscarid/2`. The 'Body' tab is selected, showing the JSON response content. Below the JSON content, the status is '200 OK', size is '179 Bytes', and time is '8 ms'. The 'Response' tab is also selected, showing the raw JSON response.


```

1
{
  "id": 2,
  "first_name": "Carlos",
  "last_name": "Pérez",
  "email": "carlos.perez@gmail.com",
  "phone": "3117894561",
  "citizenship_card": "987654321",
  "address": "Carrera 7 #15-67",
  "city": "Cali"
}

```

id	first_name	last_name	email	phone	citizenship_card	address	city
1	Ana	García	ana.garcia@gmail.com	3104567890	123456789	Calle 45 #20-30	Bogotá
2	Carlos	Pérez	carlos.perez@gmail.com	3117894561	987654321	Carrera 7 #15-67	Cali

- **Actualizar Adoptante:** Se hace uso del verbo HTTP PUT.

PUT  http://127.0.0.1:4000/adoptantes/actualizar/1

Query Headers ² Auth Body ¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL

JSON Content

```

1  {
2    "first_name": "Andrés",
3    "last_name": "Ríos",
4    "email": "andres.rios@gmail.com",
5    "phone": "3137894560",
6    "citizenship_card": "192837465",
7    "address": "Calle 123 #9-10",
8    "city": "Cartagena"
9  }
10

```









Status: 200 OK Size: 52 Bytes Time: 10 ms

Response Headers ⁷ Cookies Results Docs

```

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

```

 id	 first_name	 last_name	 email	 phone	 citizenship_card	 address	 city
1	Andrés	Ríos	andres.rios@gmail.com	3137894560	192837465	Calle 123 #9-10	Cartagena
2	Carlos	Pérez	carlos.perez@gmail.com	3117894561	987654321	Carrera 7 #15-67	Cali

- **Eliminar Adoptante:** Se hace uso del verbo HTTP DELETE.

DELETE

▼

http://127.0.0.1:4000/adoptantes/borrar/2

Query

Headers 2

Auth

Body

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

1

Status: 200 OK Size: 74 Bytes Time: 8 ms

Response

Headers 7

Cookies

Results

Docs

1 {
2 "tipo": "success",
3 "mensaje": "Registro de Adoptante eliminado con éxito."
4 }

123 id	A-Z first_name	A-Z last_name	A-Z email	A-Z phone	A-Z citizenship_card	A-Z address	A-Z city
1	Andrés	Ríos	andres.rios@gmail.com	3137894560	192837465	Calle 123 #9-10	Cartagena

➤ **Modulo Solicitudes de Adopción**

- **Crear Solicitud de Adopción: Se hace uso del verbo HTTP POST.**

POST

http://127.0.0.1:4000/solicitudes/crear

Query

Headers²

Auth

Body¹

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

1

{

2

"pet_id": 3,

3

"adopter_id": 5,

4

"request_date": "2024-09-26T10:30:00Z",

5

"status": "Pending",

6

"comments": "El adoptante parece apto"

7

}

8

Status: 201 Created

Size: 258 Bytes

Time: 15 ms

Response

Headers⁷

Cookies

Results

Docs

1

{

2

"mensaje": "Registro de Solicitud de Adopción creado con éxito",

3

"data": {

4

"id": 1,

5

"pet_id": 3,

6

"adopter_id": 5,

7

"request_date": "2024-09-26T10:30:00.000Z",

8

"status": "Pending",

9

"name_pet": "Max",

10

"name_adopter": "Lucía Martínez",

11

"comments": "El adoptante parece apto"

12

}

13

}

123 id	123 pet_id	123 adopter_id	🕒 request_date	A-Z status	A-Z name_pet	A-Z name_adopter	A-Z comments
1	3 ↗	5 ↗	2024-09-26	Pending	Max	Lucía Martínez	El adoptante parece apto

- **Buscar Solicitud de Adopción:** Se hace uso del verbo HTTP GET.

GET

http://127.0.0.1:4000/solicitudes/buscar

Query

Headers 2

Auth

Body

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

JSON Content

1

Status: 200 OK Size: 172 Bytes Time: 7 ms

Response

Headers 7

Cookies

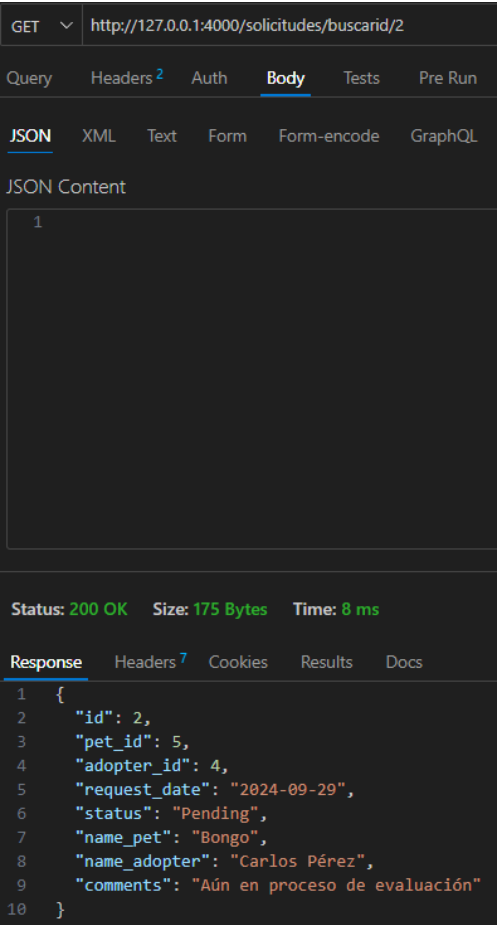
Results

Docs

1 [
2 {
3 "id": 1,
4 "pet_id": 3,
5 "adopter_id": 5,
6 "request_date": "2024-09-26",
7 "status": "Pending",
8 "name_pet": "Max",
9 "name_adopter": "Lucía Martínez",
10 "comments": "El adoptante parece apto"
11 }
12]

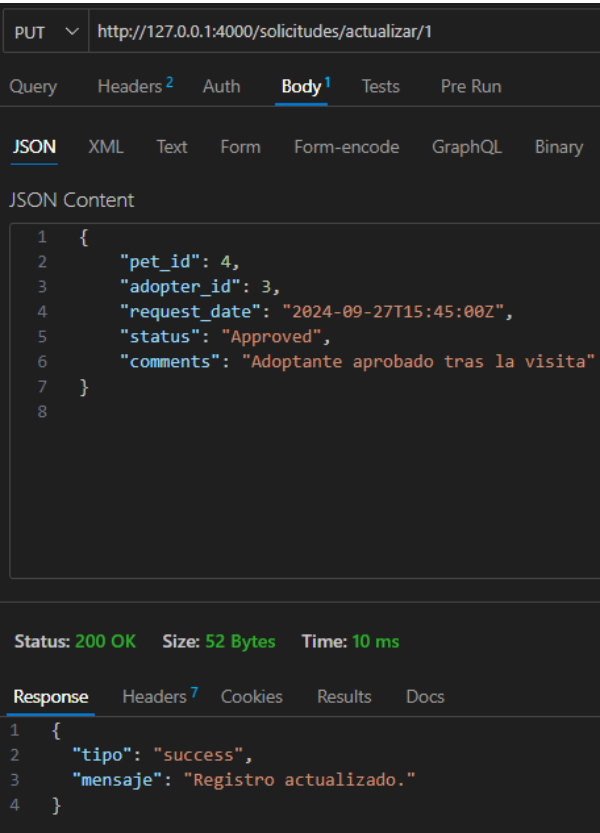
id	pet_id	adopter_id	request_date	status	name_pet	name_adopter	comments
1	3	5	2024-09-26	Pending	Max	Lucía Martínez	El adoptante parece apto

- **Buscar Solicitud de Adopción por ID: Se hace uso del verbo HTTP GET.**



123 id	123 pet_id	123 adopter_id	🕒 request_date	A-Z status	A-Z name_pet	A-Z name_adopter	A-Z comments
1	3 ↗	5 ↗	2024-09-26	Pending	Max	Lucía Martínez	El adoptante parece apto
2	5 ↗	4 ↗	2024-09-29	Pending	Bongo	Carlos Pérez	Aún en proceso de evaluac

- **Actualizar Solicitud de Adopción:** Se hace uso del verbo HTTP PUT.



id	pet_id	adopter_id	request_date	status	name_pet	name_adopter	comments
1	4	3	2024-09-27	Approved	Max	Lucía Martínez	Adoptante aprobado tras la visita
2	5	4	2024-09-29	Pending	Bongo	Carlos Pérez	Aún en proceso de evaluación

- **Eliminar Solicitud de Adopción:** Se hace uso del verbo HTTP DELETE.

DELETE

▼

http://127.0.0.1:4000/solicitudes/borrar/2

Query

Headers²

Auth

Body

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

1

Status: 200 OK

Size: 86 Bytes

Time: 7 ms

Response

Headers⁷

Cookies

Results

Docs

1

{

2

"tipo": "success",

3

"mensaje": "Registro de Solicitud de Adopcion eliminado con éxito."

4

}

<div>123 id</div>	<div>123 pet_id</div>	<div>123 adopter_id</div>	<div>🕒 request_date</div>	<div>A-Z status</div>	<div>A-Z name_pet</div>	<div>A-Z name_adopter</div>	<div>A-Z comments</div>
1	4	3	2024-09-27	Approved	Max	Lucía Martínez	Adoptante aprobado tras la visita