

DIPLOMADO DE ACTUALIZACION EN NUEVAS TECNOLOGIAS PARA
DESARROLLO DE SOFTWARE

PREPARADO POR:
MILER ANDRES ESPAÑA
COD: 2130341131

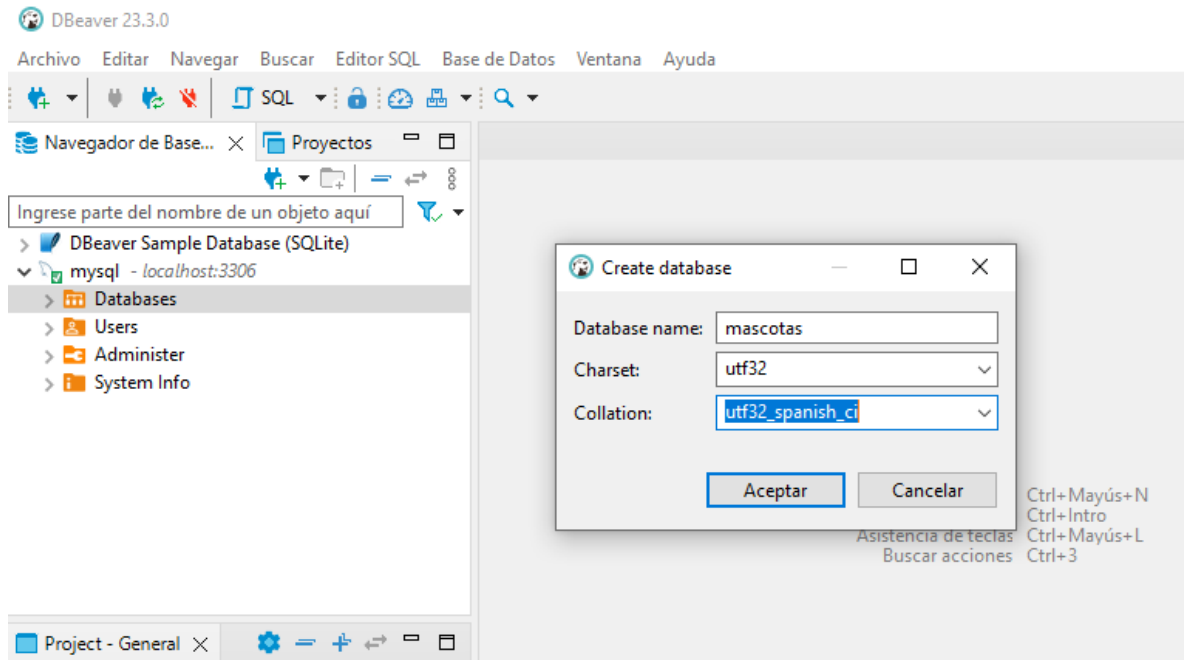
PREPARADO PARA:
VICENTE AUX

UNIVERSIDAD DE NARIÑO
INGENIERIA DE SISTEMAS
DICIEMBRE 2023

1. Crear una base de datos MYSQL que permita llevar el registro de mascotas (perros y gatos), así como también el proceso de solicitud de adopción de estas.

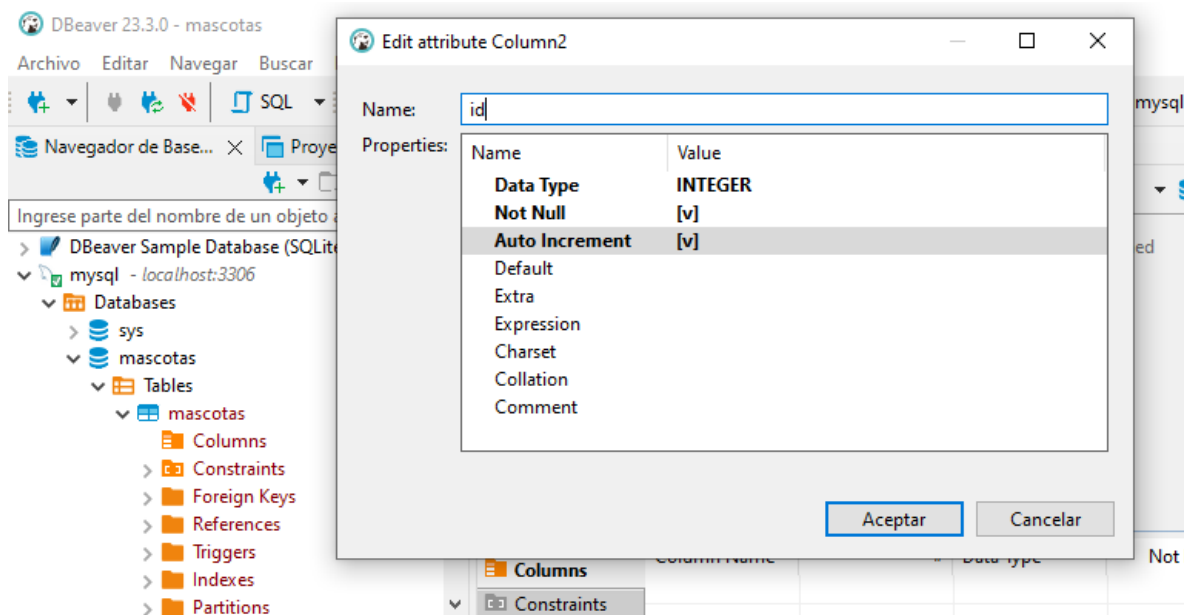
- Instalamos **WampServer** y lanzamos el servicio
- Instalamos **DBeaver** y abrimos el software

- a. Primero creamos la base de datos llamada “mascotas”, seleccionamos utf32 y el idioma español.

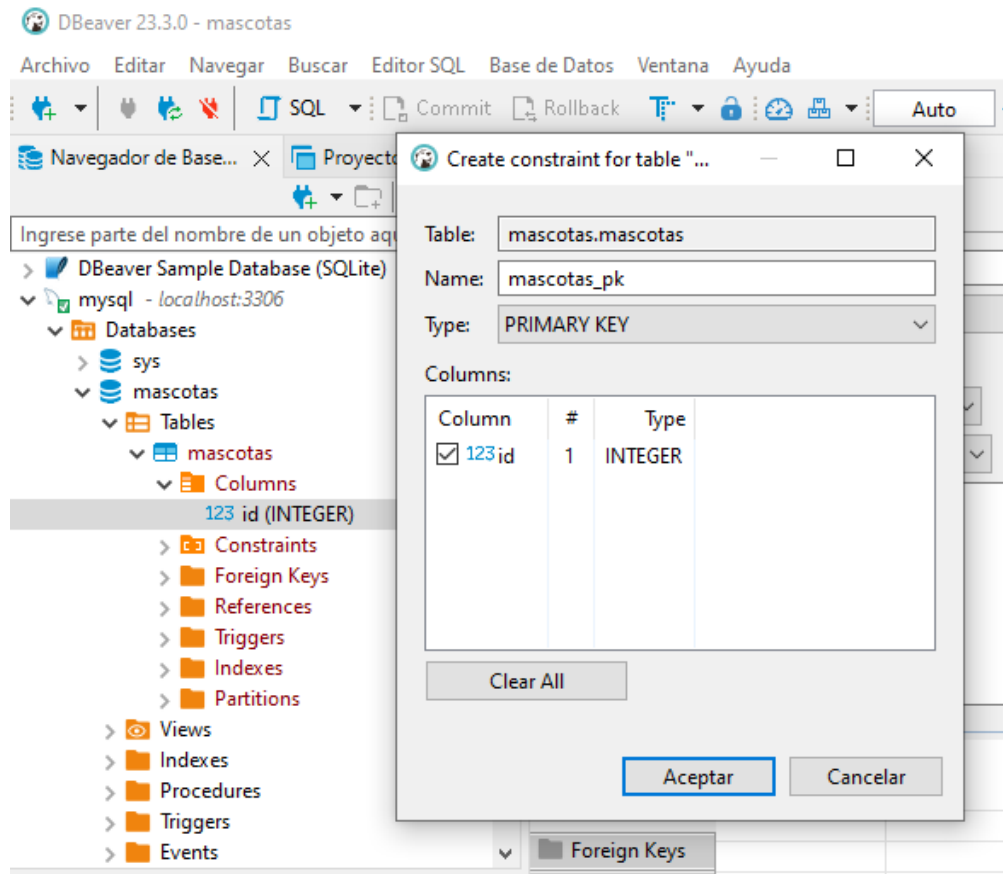


- b. Dentro de nuestra base de datos llamada “mascotas” creamos las diferentes tablas que necesitaremos.

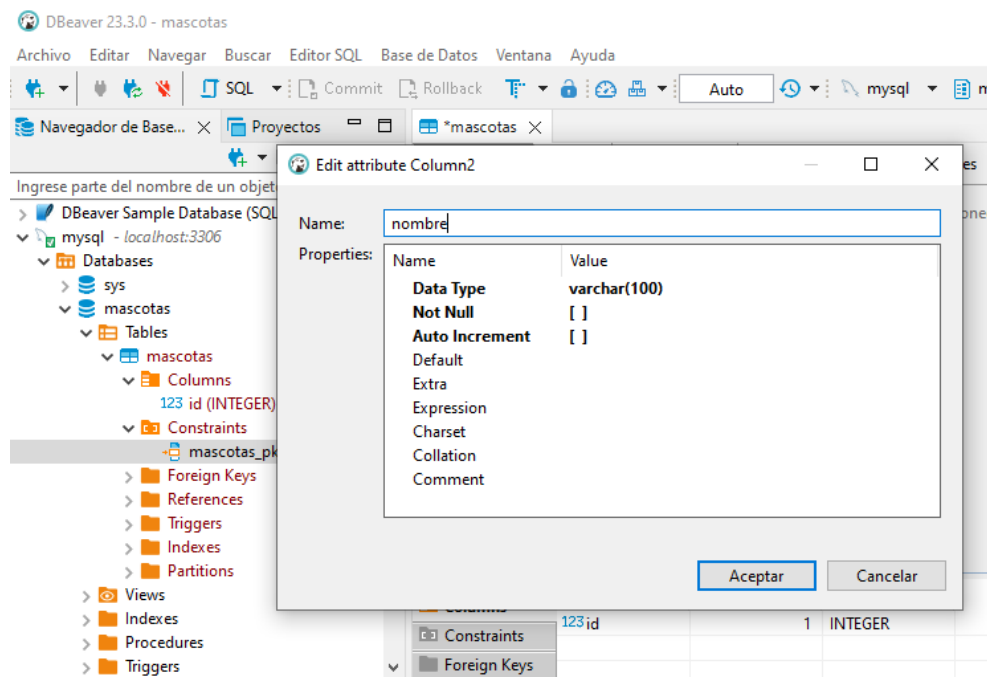
- Creamos la tabla id, de tipo entero, que no permita campos vacíos y se autoincrementa.



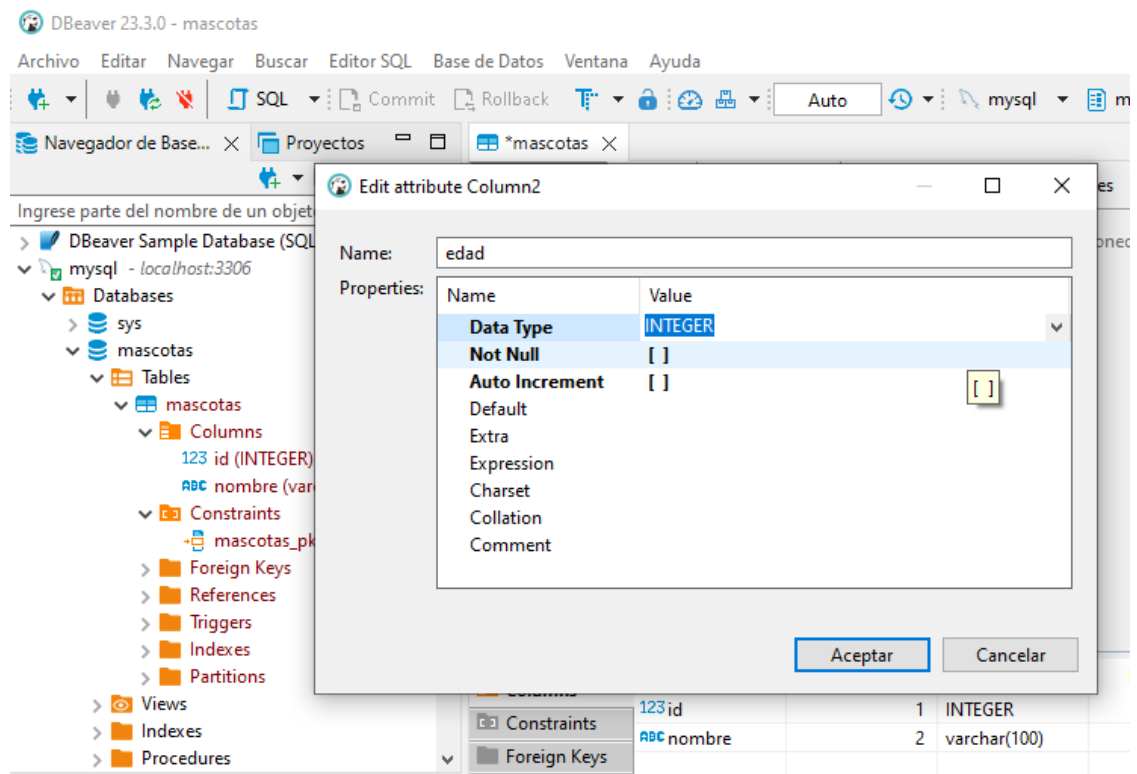
- Ahora en el apartado “Constraints” creamos la llave primaria de nuestra base de datos, la cual será la tabla id.



- Creamos la tabla nombre.



- Creamos la tabla edad.



- Guardamos los cambios realizados y verificamos que la base de datos “mascotas” se haya creado correctamente con sus tablas.

```
mysql> use mascotas;
Database changed
mysql> show tables;
+-----+
| Tables_in_mascotas |
+-----+
| mascotas            |
+-----+
1 row in set (0.01 sec)

mysql> show mascotas;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near '1'
mysql> describe mascotas;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id    | int           | NO   | PRI | NULL    | auto_increment |
| nombre | varchar(100) | YES  |     | NULL    |                |
| edad  | int           | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

2. Desarrollar una aplicación Backend implementada en NodeJS y ExpressJS que haga uso de la base de datos del primer punto y que permita el desarrollo de todas las tareas asociadas al registro y administración de las mascotas dadas en adopción por la empresa (La empresa debe contar con un nombre).

Se debe hacer uso correcto de los verbos HTTP dependiendo de la tarea a realizar.

- a. Creamos una carpeta con el nombre "Taller002backend" y la abrimos con VsCode, ahora instalaremos las dependencias que vamos a usar para nuestro proyecto.

Inicializamos el node.js con el comando "npm init -y"

```
PS D:\diplomado\modulo 1\modulo back\Taller002backend> npm init -y
```

Se nos creara un archivo package.json, en el cual cambiamos algunos parámetros para que se pueda ejecutar con nodemon, además, creamos una carpeta "src" y dentro de ella creamos un archivo "app.js" el cual será el archivo principal.

```
package.json > ...
1  {
2    "name": "taller002backend",
3    "version": "1.0.0",
4    "description": "",
5    "main": "app.js",
6    "scripts": {
7      "start": "nodemon ./src/app.js"
8    },
9  }
```

Ahora instalamos nodemon con el comando "npm install nodemon -D"

```
PS D:\diplomado\modulo 1\modulo back\Taller002backend> npm i nodemon --D
```

Ahora instalamos las demás dependencias que necesitaremos para arrancar nuestro proyecto de mascotas.

```
found 0 vulnerabilities
PS D:\diplomado\modulo 1\modulo back\Taller002backend> npm i express
```

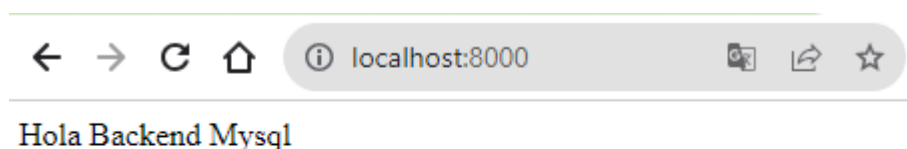
```
found 0 vulnerabilities
PS D:\diplomado\modulo 1\modulo back\Taller002backend> npm i mysql2
```

```
found 0 vulnerabilities
PS D:\diplomado\modulo 1\modulo back\Taller002backend> npm i sequelize
```

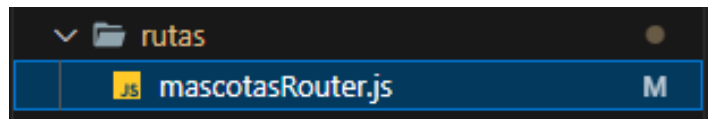
Así quedara configurado nuestro archivo "package.json".

```
package.json > ...
3  "version": "1.0.0",
4  "description": "",
5  "main": "app.js",
6  "scripts": {
7    "start": "nodemon ./src/app.js"
8  },
9  "repository": {
10   "type": "git",
11   "url": "git+https://github.com/Miler98/Taller002backend.git"
12 },
13 "keywords": [],
14 "author": "",
15 "license": "ISC",
16 "bugs": {
17   "url": "https://github.com/Miler98/Taller002backend/issues"
18 },
19 "homepage": "https://github.com/Miler98/Taller002backend#readme"
20 "devDependencies": {
21   "nodemon": "^3.0.2"
22 },
23 "dependencies": {
24   "express": "^4.18.2",
25   "mysql2": "^3.6.5",
26   "sequelize": "^6.35.2"
27 }
28 }
29 }
```

- b. Ahora empezaremos con la programación de nuestro software, lo primero que haremos es la conexión de Express mediante el puerto 8000.



- c. Creamos las rutas en las cuales vamos a manejar el software “mascotas”.



```

rutas > JS mascotasRouter.js > routerMascotas.delete("/eliminar/id") callback
1  import express from "express";
2  //import {crear, buscarId, buscar, actualizar, eliminar} from "../controladores/mascotasController.js";
3  const routerMascotas = express.Router();
4
5  routerMascotas.get("/", (req, res) => {
6    res.send("Bienvenido a Mascotas");
7  });
8
9  routerMascotas.post("/crear", (req, res) => {
10    //crear(req, res);
11    res.send("crearMascotas");
12  });
13
14  //routerMascotas.get("/buscar/:id", (req, res) => {
15    // buscarId(req, res);
16    // res.send("Buscar Mascotas");
17  //});
18
19  routerMascotas.get("/buscar", (req, res) => {
20    // buscar(req, res);
21    res.send("Buscar Mascotas");
22  });

```

- d. Establecemos la conexión de la base de datos que habíamos creado previamente.



```

database > JS conexion.js > ...
1  import Sequelize from "sequelize";
2
3  const db = new Sequelize("mascotas", "mascotas", "mascotas2023", {
4    dialect: "mysql",
5    host: "localhost"
6  });
7
8  export {db}

```

- Validamos que la conexión se haya establecido correctamente desde “app.js”

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

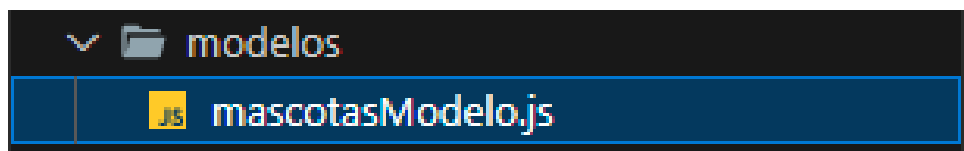
//Crear Instancia de Express
const app = express();

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

//Verificar Conexion a Base de Datos
db.authenticate().then(()=>{
  console.log(`Base de Datos conectada de manera exitosa`);
}).catch(err=>{
  console.log(`Error al conectarse a la Base de Datos ::: ${err}`);
})
```

```
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node ./src/app.js`
Executing (default): SELECT 1+1 AS result
Executing (default): SELECT 1+1 AS result
Base de Datos conectada de manera exitosa
Servidor Inicializado en puerto 8000
```

- e. Ahora pasamos a programar el modelo que tendrá nuestro software para el registro y control de las mascotas.

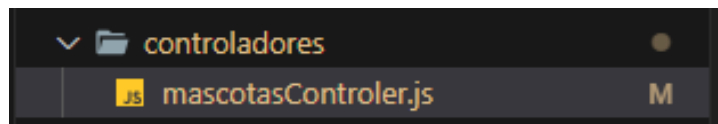



```


modelos >  mascotasModelo.js > ...
1  import Sequelize from "sequelize";
2  import {db} from "../database/conexion.js";
3
4  const mascotas = db.define("mascotas",{
5      id:{
6          type:Sequelize.INTEGER,
7          allowNull: false,
8          autoIncrement: true,
9          primaryKey: true
10     },
11     nombre:{
12         type: Sequelize.STRING,
13         allowNull: true
14     },
15     edad:{
16         type: Sequelize.INTEGER,
17         allowNull:true
18     }
19 });
20
21 export {mascotas}

```

- f. A continuación, creamos los controladores de nuestro software, aquí va toda la lógica funcional de la aplicación, (Crear, Buscar, Modificar, Eliminar).



```

controladores >  mascotasControler.js > ...
1  import {mascotas} from "../modelos/mascotasModelo.js";
2
3  //Crear un recurso
4  const crear = (req,res)=>{
5      if(!req.body.nombre){
6          res.status(400).json({
7              mensaje: "El nombre no puede estar vacio."
8          });
9          return;
10     }
11     const dataset={
12         nombre: req.body.nombre,
13         edad: req.body.edad
14     };
15
16     //Usar Sequelize para crear el recurso
17     mascotas.create(dataset).then((resultado)=>{
18         res.status(200).json({
19             mensaje: "Registro creado correctamente"
20         })
21     }).catch((err)=>{
22         res.status(500).json({

```

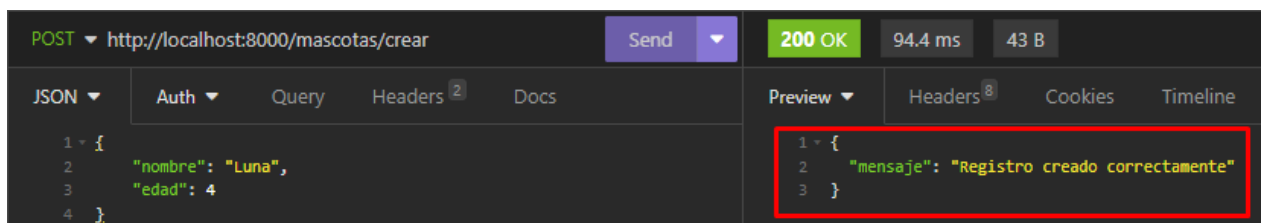
g. Finalmente creamos los recursos HTTP con los verbos que usaremos para las pruebas.

```
requests.http > GET /mascotas/buscarId/3
Send Request
1 GET http://localhost:8000/mascotas/buscar HTTP/1.1
2
3 ###
Send Request
4 POST http://localhost:8000/mascotas/crear HTTP/1.1
5 Content-Type: application/json
6
7 {
8     "nombre": "Troya",
9     "edad": 2
10 }
11
12 ###
Send Request
13 GET http://localhost:8000/mascotas/buscarId/3 HTTP/1.1
14
15 ###
Send Request
16 PUT http://localhost:8000/mascotas/actualizar/2 HTTP/1.1
17 Content-Type: application/json
18
19 {
20     "nombre": "Maximiliano",
21     "edad": 10
22 }
23
24 ###
Send Request
25 DELETE http://localhost:8000/mascotas/eliminar/2 HTTP/1.1
26
```

3. Realizar verificación de las diferentes operaciones a través de un cliente grafico (Postman, Imnsomnia, etc.), tomar capturas de pantalla que evidencien el resultado de las solicitudes realizadas.

- En este caso usaremos el software Insomnia para realizar las pruebas HTTP

a. Crear: ingresamos la Url del método crear por medio del método Post.



- b. Buscar: ingresamos la Url del método Buscar por medio del método GET.

```
GET http://localhost:8000/mascotas/buscar 200 OK 19.4 ms 175 B

JSON Auth Query Headers Docs Preview Headers Co

1 ...

1 [
2 {
3   "id": 1,
4   "nombre": "Lola",
5   "edad": 3
6 },
7 {
8   "id": 4,
9   "nombre": "Alaska",
10  "edad": 3
11 },
12 {
13   "id": 6,
14   "nombre": "rene",
15   "edad": 8
16 },
17 {
18   "id": 8,
19   "nombre": "Laica",
20   "edad": 3
21 },
22 {
23   "id": 10,
24   "nombre": "Luna",
25   "edad": 4
26 }
27 ]
```

Evidenciamos que el método Crear funciona correctamente porque podemos evidenciar que se creo con el Id 10 como muestra la imagen.

Modificar: ingresamos la Url del método Actualizar por medio del método PUT.

```
PUT http://localhost:8000/mascotas/actualizar/1 200 OK 29.3 ms 34 B

JSON Auth Query Headers Docs Preview Headers Cookies Timeline

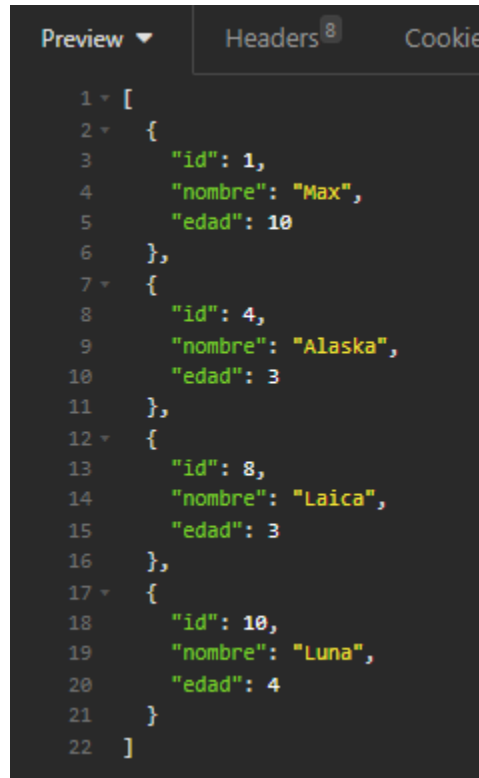
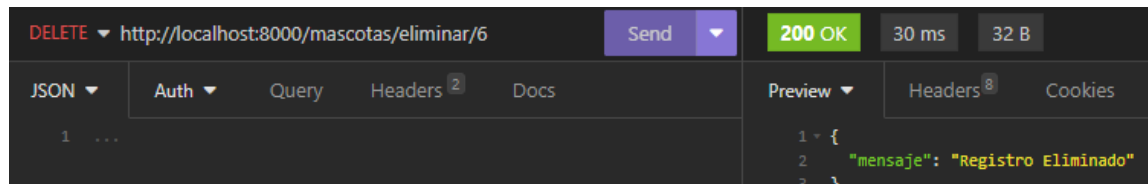
1 {
2   "nombre": "Max",
3   "edad": 10
4 }

1 {
2   "mensaje": "Registro Actualizado"
3 }
```

```
1 [
2 {
3   "id": 1,
4   "nombre": "Max",
5   "edad": 10
6 },
7 {
8   "id": 4,
9   "nombre": "Alaska",
10  "edad": 3
11 },
12 ]
```

Verificamos que el nombre del Id 1 cambio a los nuevos parámetros que ingresamos.

Eliminar: ingresamos la Url del método Eliminar por medio del método DELETE.



Verificamos que el Id 6 ya no se encuentra en la lista de mascotas, se eliminó correctamente.