

TALLER N°2 BACKEND

PRESENTADO POR:

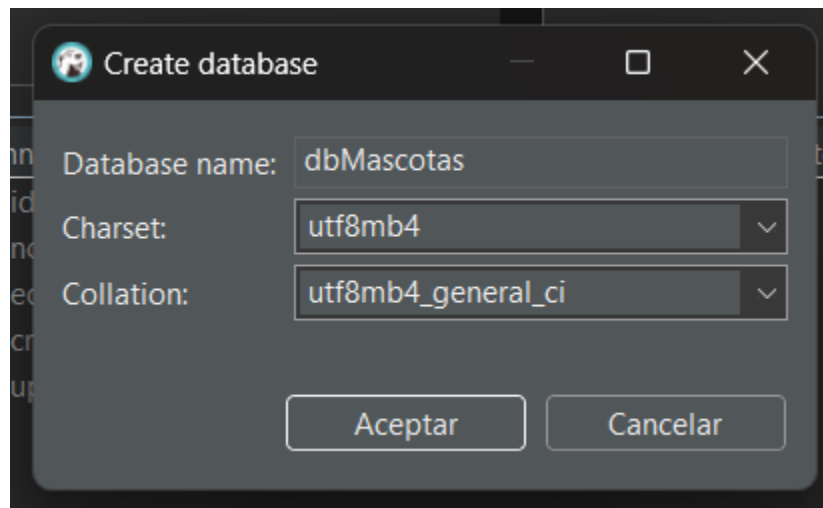
ERNEY SEBASTIAN BASTIDAS ROSALES

UNIVERSIDAD DE NARIÑO

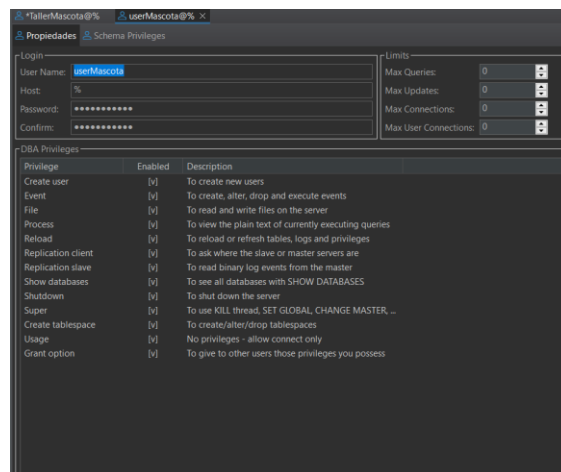
SEDE IPILAES

2024

1. **Crear una base de datos en MySQL/MariaDB** para registrar la información de una empresa de adopción de mascotas. Debes incluir la administración de las mascotas y la capacidad de registrar solicitudes de adopción.



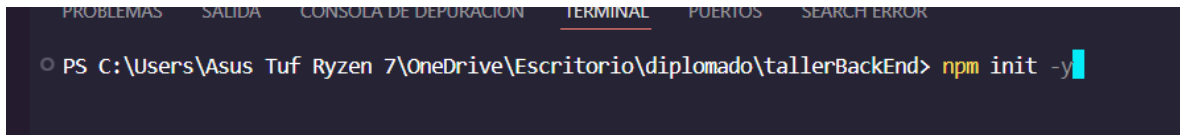
Después de que hayamos creado la base de datos creamos un usuario y le damos los privilegios.



Y hacemos la conexión a la base de datos

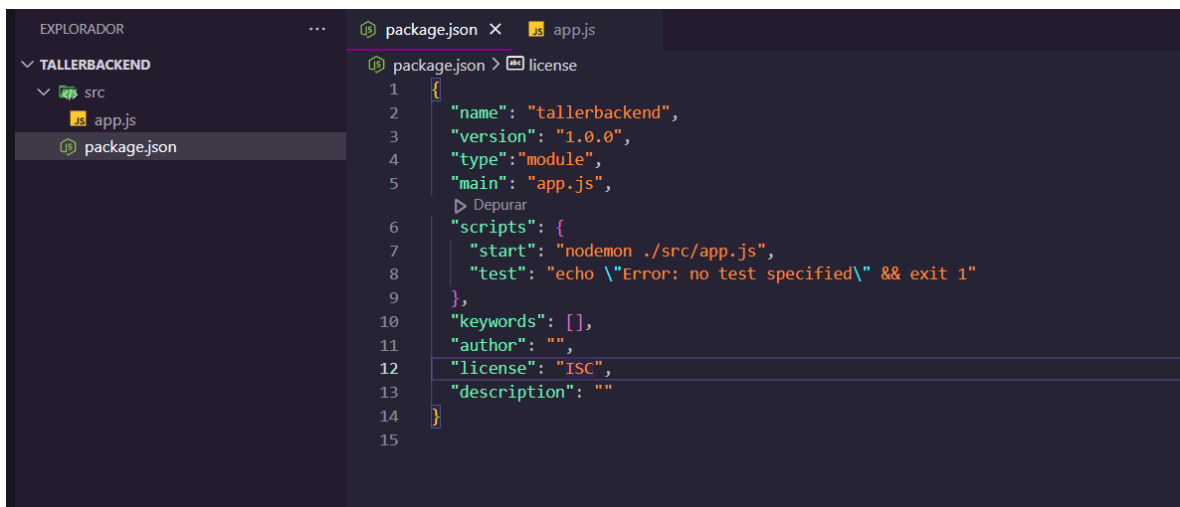


2. **Desarrollar una aplicación Backend en NodeJS y ExpressJS** que interactúe con la base de datos creada en el primer punto. Esta aplicación debe permitir todas las tareas relacionadas con el registro y administración de mascotas, así como las solicitudes de adopción. Asegúrate de utilizar los verbos HTTP correctamente para cada acción.



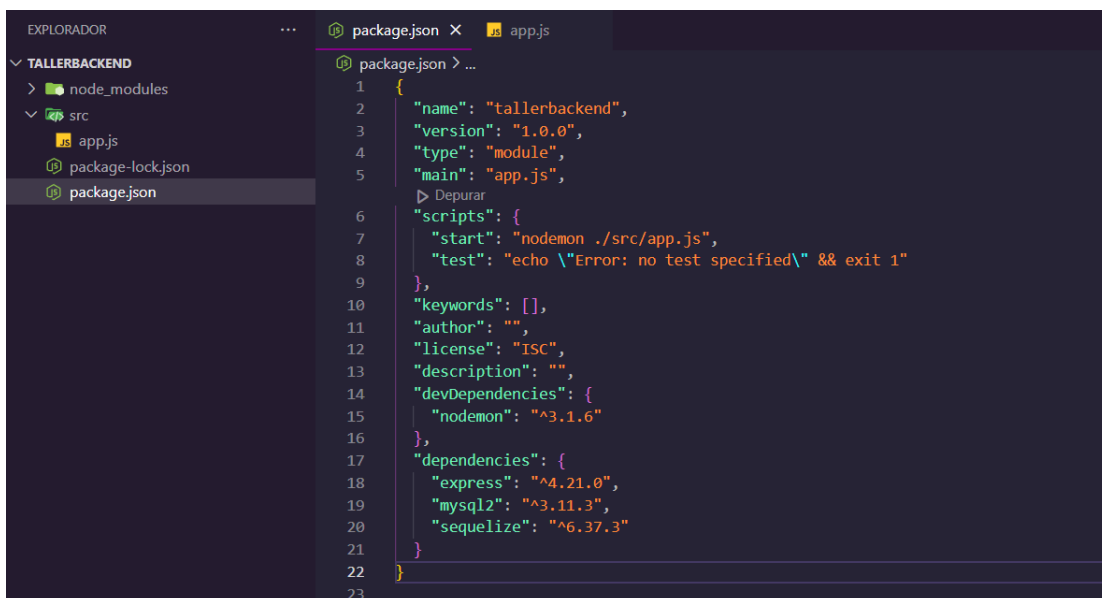
```
PS C:\Users\Asus Tuf Ryzen 7\OneDrive\Escritorio\diplomado\tallerBackEnd> npm init -y
```

Después de iniciar esta instrucción nos aparece esta pantalla y realizamos algunos cambios, como lo son el type a module y el main para que inicie en app.js, y también el nodemon para que se ejecuten los cambios.



```
package.json > license
1 {
2   "name": "tallerbackend",
3   "version": "1.0.0",
4   "type": "module",
5   "main": "app.js",
6   "scripts": {
7     "start": "nodemon ./src/app.js",
8     "test": "echo \\"Error: no test specified\\" && exit 1"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "description": ""
14 }
15
```

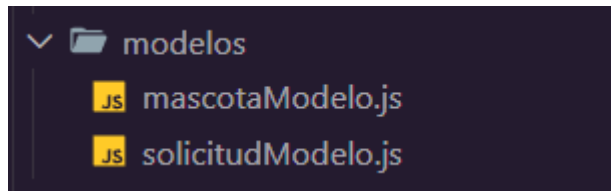
Después de ello instalamos las dependencias de nodemon, express, mysql2 y sequelize con npm



```
package.json > ...
1 {
2   "name": "tallerbackend",
3   "version": "1.0.0",
4   "type": "module",
5   "main": "app.js",
6   "scripts": {
7     "start": "nodemon ./src/app.js",
8     "test": "echo \\"Error: no test specified\\" && exit 1"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "description": "",
14  "devDependencies": {
15    "nodemon": "^3.1.6"
16  },
17  "dependencies": {
18    "express": "^4.21.0",
19    "mysql2": "^3.11.3",
20    "sequelize": "^6.37.3"
21  }
22 }
23
```

Después de ello formamos toda la estructura para hacer la lógica del proyecto, de la siguiente manera.

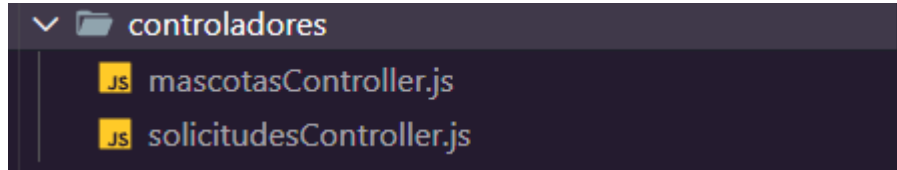
1. Creamos los modelos



```
src > modelos > mascotaModelo.js > Mascotas > edad
1  import { Sequelize, DataTypes } from "sequelize";
2  import { db } from "../database/conexion.js";
3
4  const Mascotas = db.define("mascotas", {
5    id: {
6      type: DataTypes.INTEGER,
7      allowNull: false,
8      autoIncrement: true,
9      primaryKey: true
10   },
11   nombre: {
12     type: DataTypes.STRING,
13     allowNull: false
14   },
15   descripcion: {
16     type: DataTypes.TEXT,
17     allowNull: true
18   },
19   edad: {
20     type: DataTypes.INTEGER,
21     allowNull: false
22   },
23   genero: {
24     type: DataTypes.ENUM('Macho', 'Hembra'),
25     allowNull: false
26   },
27   imagen: {
28     type: DataTypes.STRING, // Ruta de la imagen o URL
29     allowNull: true
30   }
31 }, {
32   tableName: 'mascotas', // Nombre de la tabla en la base de datos
33   timestamps: false
34 });
35
36 export { Mascotas }
```

```
src > modelos > solicitudModelo.js > ...
1  import { Sequelize, DataTypes } from "sequelize";
2  import { db } from "../database/conexion.js";
3
4  const Solicitudes = db.define("solicitudes", {
5    id: {
6      type: DataTypes.INTEGER,
7      allowNull: false,
8      autoIncrement: true,
9      primaryKey: true
10   },
11   nombreAdoptante: {
12     type: DataTypes.STRING,
13     allowNull: false
14   },
15   fechaSolicitud: {
16     type: DataTypes.DATE,
17     allowNull: false,
18     defaultValue: Sequelize.NOW
19   },
20   estado: {
21     type: DataTypes.ENUM('Pendiente', 'Aprobada', 'Rechazada'),
22     allowNull: false,
23     defaultValue: 'Pendiente'
24   }
25 }, {
26   tableName: 'solicitudes',
27   timestamps: false
28 });
29
30 export { Solicitudes }
```

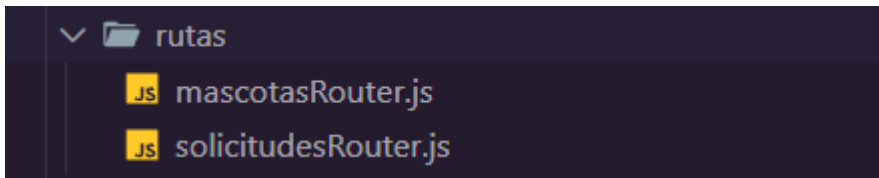
2. Creamos los controladores para cada objeto y aplicamos toda la programación en todas las rutas.



```
src > controladores > mascotasController.js > eliminar
1  import { Mascotas } from "../modelos/mascotaModelo.js";
2
3  //crear recurso mascota
4  const crear = (req, res) => {
5    if (!req.body.nombre) {
6      res.status(400).send({
7        mensaje: "El nombre de la mascota es requerido"
8      });
9      return;
10   }
11
12   const dataset = {
13     nombre: req.body.nombre,
14     descripcion: req.body.descripcion,
15     edad: req.body.edad,
16     genero: req.body.genero,
17     peso: req.body.peso,
18     imagen: req.body.imagen
19   }
20
21   //usar sequelize base de datos
22   Mascotas.create(dataset).then((resultado) => {
23     res.status(200).json({
24       mensaje: "Registro de mascota Exitoso"
25     });
26   }).catch((err) => {
27     res.status(500).json({
28       mensaje: "Error al registrar la mascota ::: ${err}"
29     });
30   });
31 }
32
```

```
src > controladores > solicitudesController.js > crearSolicitud > dataset
1  import { Solicitudes } from "../modelos/solicitudModelo.js";
2
3  const crearSolicitud = (req, res) => {
4    if (!req.body.nombreAdoptante) {
5      res.status(400).send({
6        mensaje: "El nombre del adoptante es requerido"
7      });
8      return;
9    }
10
11    const dataset = {
12      nombreAdoptante: req.body.nombreAdoptante,
13      fechaSolicitud: req.body.fechaSolicitud,
14      estado: req.body.estado
15    };
16
17    Solicitudes.create(dataset)
18      .then((resultado) => {
19      res.status(200).json({
20        mensaje: "Registro de solicitud exitoso",
21        solicitud: resultado
22      });
23      })
24      .catch((err) => {
25      res.status(500).json({
26        mensaje: "Error al registrar la solicitud: ${err}"
27      });
28      });
29  };
30
```

3. Después de ello empezamos a maquetar las rutas



```
src > rutas > mascotasRouter.js > routerMascotas.get('/buscarPorId/:id') callback
1 import express from "express";
2 import { buscarPorId, buscarTodos, crear, editar, eliminar } from "../controladores/mascotasController.js";
3
4 const routerMascotas = express.Router();
5
6 routerMascotas.get('/', (req, res) => {
7   res.send('Bienvenido al Sitio Principal.');
```

```
src > rutas > solicitudesRouter.js > ...
1 import express from "express";
2 import { buscarSolicitudPorId, buscarTodasSolicitudes, crearSolicitud, editarSolicitud, eliminarSolicitud } from "../contr
3
4 const routerSolicitudes = express.Router();
5
6 routerSolicitudes.get('/hola', (req, res) => {
7   res.send('Bienvenido al Sitio Principal.');
```

4 y en por último todas las configuraciones necesarias en app

```
src > app.js > ...
1 import express from "express";
2 import { routerMascotas } from "../rutas/mascotasRouter.js";
3 import { routerSolicitudes } from "../rutas/solicitudesRouter.js";
4 import { db } from "../database/conexion.js";
5
6 //crear instancia express
7 const app = express();
8
9
10 //json
11 app.use(express.json());
12
13 //verificar conexion base de datos
14 db.authenticate().then (()=>{
15   console.log(`conexion a la base de datos correcta`)
16 }).catch(err=>{
17   console.log(`conexion a la base de datos incorrecta ${err}`)
18 });
19
20 //llamando rutas de mascotas
21 app.use("/mascotas", routerMascotas);
22
23 //llamado rutas de solicitudes
24 app.use("/solicitudes", routerSolicitudes);
25
26 //puerto del servidor
27 const PORT = 4000;
28
29 db.sync().then(()=>{
30   //Abri servicio e iniciar el Servidor
31   app.listen(PORT,()=>{
32     console.log(`Servidor Inicializado en el puerto ${PORT}`);
33   })
34 }).catch(err=>{
35   console.log(`Error al sincronizar la base de datos ${err}`);
36 });
37
```

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

1. Operación crear mascotas

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:4000/mascotas/crear
- Status:** 200 OK
- Size:** 41 Bytes
- Time:** 27 ms
- Body:** A JSON object with the following content:

```
1 {
2   "nombre": "pepe",
3   "descripcion": "perrito de dos años",
4   "edad": 2,
5   "genero": "M",
6   "peso": 20,
7   "imagen": "jolal"
8 }
```
- Response:** A JSON object with the following content:

```
1 {
2   "mensaje": "Registro de mascota Exitoso"
3 }
```

2. Operación Buscar todas las Mascotas

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:4000/mascotas/buscarTodos
- Status:** 200 OK
- Size:** 191 Bytes
- Time:** 12 ms
- Response:** A JSON array containing two pet objects:

```
1 [
2   {
3     "id": 1,
4     "nombre": "pepe",
5     "descripcion": "perrito de dos años",
6     "edad": 2,
7     "genero": "",
8     "imagen": "jolal"
9   },
10  {
11    "id": 2,
12    "nombre": "Max",
13    "descripcion": "gatito",
14    "edad": 3,
15    "genero": "",
16    "imagen": "gatito.jpg"
17  }
18 ]
```


3. Operación editar mascotas

The screenshot shows a REST client interface with a PUT request to `http://localhost:4000/mascotas/editar/2`. The request body is a JSON object: `{ "nombre": "Maximiliano", "descripcion": "gato", "edad": 6, "genero": "F", "peso": 6, "imagen": "gatitos.jpg" }`. The response is a 200 OK status with a JSON body: `{ "mensaje": "Mascota actualizada correctamente" }`.

```
PUT http://localhost:4000/mascotas/editar/2
{
  "nombre": "Maximiliano",
  "descripcion": "gato",
  "edad": 6,
  "genero": "F",
  "peso": 6,
  "imagen": "gatitos.jpg"
}
```

```
{
  "mensaje": "Mascota actualizada correctamente"
}
```

4. Operación buscar solo uno

The screenshot shows a REST client interface with a GET request to `http://localhost:4000/mascotas/buscarPorId/1`. The response is a 200 OK status with a JSON body: `{ "id": 1, "nombre": "pepe", "descripcion": "perrito de dos años", "edad": 2, "genero": "", "imagen": "jolal" }`.

```
GET http://localhost:4000/mascotas/buscarPorId/1
```

```
{
  "id": 1,
  "nombre": "pepe",
  "descripcion": "perrito de dos años",
  "edad": 2,
  "genero": "",
  "imagen": "jolal"
}
```

5. Operación eliminar mascota

The screenshot shows a REST client interface with a DELETE request to `http://localhost:4000/mascotas/eliminar/1`. The response is a 200 OK status with a JSON body: `{ "mensaje": "Mascota eliminada correctamente" }`. A 'Copy' button is visible next to the response.

```
DELETE http://localhost:4000/mascotas/eliminar/1
```

```
{
  "mensaje": "Mascota eliminada correctamente"
}
```

6. Operación crear Solicitud

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** `http://localhost:4000/solicitudes/crearSolicitud`
- Status:** 200 OK
- Size:** 153 Bytes
- Time:** 42 ms
- Body:** The request body is a JSON object:

```
{  "nombreAdoptante": "Juan",  "fechaSolicitud": "10/12/2022",  "estado": "Aprobada"}
```
- Response:** The response is a JSON object:

```
{  "mensaje": "Registro de solicitud exitoso",  "solicitud": {    "id": 1,    "nombreAdoptante": "Juan",    "fechaSolicitud": "2022-10-12T05:00:00.000Z",    "estado": "Aprobada"  }}
```

7. Operación Editar Solicitud

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** `http://localhost:4000/solicitudes/editarSolicitud/4`
- Status:** 200 OK
- Size:** 49 Bytes
- Time:** 17 ms
- Body:** The request body is a JSON object:

```
{  "nombreAdoptante": "Sofia Vergara",  "fechaSolicitud": "2/12/2022",  "estado": "Rechazada"}
```
- Response:** The response is a JSON object:

```
{  "mensaje": "Solicitud actualizada correctamente"}
```

8. Operación buscar solicitud por id

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** `http://localhost:4000/solicitudes/buscarSolicitudPorId/4`
- Status:** 200 OK
- Size:** 107 Bytes
- Time:** 14 ms
- Body:** The request body is empty. The "Content-Type" header is set to `application/json`.
- Response:** The response is a JSON object:

```
{  "id": 4,  "nombreAdoptante": "Sofia Vergara",  "fechaSolicitud": "2022-02-12T05:00:00.000Z",  "estado": "Rechazada"}
```

9. Operación para buscar todas las solicitudes

The screenshot shows a REST client interface with a GET request to `http://localhost:4000/solicitudes/buscarTodasSolicitudes`. The response status is 200 OK, with a size of 370 Bytes and a time of 9 ms. The response body is a JSON array of four objects, each representing a request with fields like id, nombreAdoptante, fechaSolicitud, and estado.

Request:

```
GET http://localhost:4000/solicitudes/buscarTodasSolicitudes
```

Response:

```
[
  {
    "id": 1,
    "nombreAdoptante": "Juan",
    "fechaSolicitud": "2022-10-12T05:00:00.000Z",
    "estado": "Aprobada"
  },
  {
    "id": 2,
    "nombreAdoptante": "Andrea",
    "fechaSolicitud": null,
    "estado": "Aprobada"
  },
  {
    "id": 3,
    "nombreAdoptante": "Sofia Vergara",
    "fechaSolicitud": null,
    "estado": "Aprobada"
  },
  {
    "id": 4,
    "nombreAdoptante": "Sofia Vergara",
    "fechaSolicitud": "2022-02-12T05:00:00.000Z",
    "estado": "Rechazada"
  }
]
```

10. Operación para eliminar solicitud

The screenshot shows a REST client interface with a DELETE request to `http://localhost:4000/solicitudes/eliminarSolicitud/4`. The response status is 200 OK, with a size of 47 Bytes and a time of 9 ms. The response body is a JSON object with a single field, mensaje, indicating the request was successfully deleted.

Request:

```
DELETE http://localhost:4000/solicitudes/eliminarSolicitud/4
```

Response:

```
{
  "mensaje": "Solicitud eliminada correctamente"
}
```