

Universidad de Nariño.

Ingeniería de Sistemas.

Diplomado de actualización en nuevas tecnologías para el
desarrollo de Software.

Taller Unidad 2 Backend

Nombre: Daniel Sebastian Mueses Rivera

Codigo: 220036027

Primero se inicializo el proyecto Node.js

```
PS C:\Users\acer\Documents\DIPLOMADO2024B\TRABAJO\backEnd> npm init -y
```

Se instalo las diferentes dependencias

```
PS C:\Users\acer\Documents\DIPLOMADO2024B\TRABAJO\backEnd> npm install nodemon -D
```

```
PS C:\Users\acer\Documents\DIPLOMADO2024B\TRABAJO\backEnd> npm install express
```

```
PS C:\Users\acer\Documents\DIPLOMADO2024B\TRABAJO\backEnd> npm install mysql2
```

```
PS C:\Users\acer\Documents\DIPLOMADO2024B\TRABAJO\backEnd> npm install sequelize
```

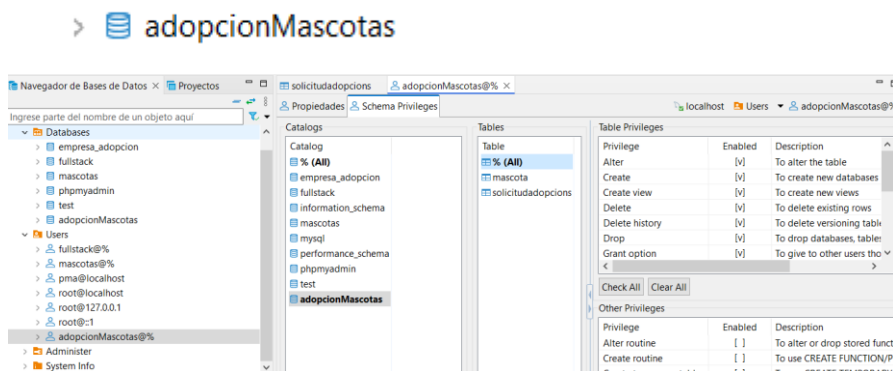
Se inicializo el proyecto y se procedió con su estructuración

```
PS C:\Users\acer\Documents\DIPLOMADO2024B\TRABAJO\backEnd> npm run start
```

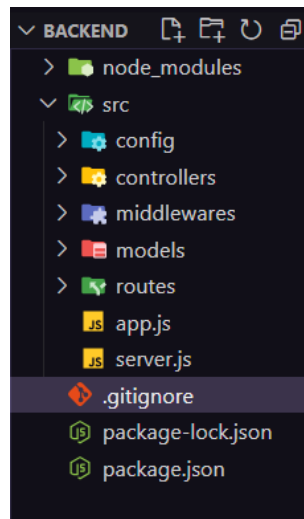
Luego de esto en DBeaver se creo la base de datos llamada adopcionMascotas

Usuario: adopcionMascotas

Contraseña: adopcion123



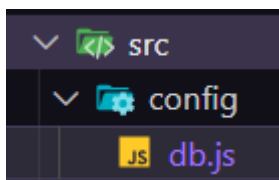
Esta es la estructura del proyecto



Se configuro el package.json

```
{
  "name": "backend",
  "version": "1.0.0",
  "type": "module",
  "main": "app.js",
  "scripts": {
    "start": "nodemon ./src/app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "nodemon": "^3.1.7"
  },
  "dependencies": {
    "express": "^4.21.0",
    "mysql2": "^3.11.3",
    "sequelize": "^6.37.3"
  }
}
```

Luego se creo la carpeta config y dentro de ella el archivo db.js para configurar la conexión a la base de datos



```
import {Sequelize} from "sequelize";

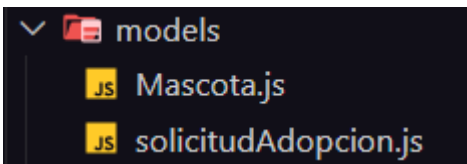
const sequelize = new Sequelize("adopcionMascotas","adopcionMascotas","adopcion123",{
  dialect: "mysql",
  host: "localhost"
});

// Verifica la conexión
try {
  await sequelize.authenticate();
  console.log('Conexión a la base de datos establecida correctamente.');
```

```
} catch (error) {
  console.error('No se pudo conectar a la base de datos:', error);
}

// Exporta el objeto sequelize
export default sequelize;
```

Luego se crearon los modelos de datos que se representaran en la base de datos, se creo una carpeta llamada models y dentro de ella dos archivos llamados Mascota.js y solicitudAdopcion.js



```
import { DataTypes } from 'sequelize';
import sequelize from '../config/db.js';

const Mascota = sequelize.define('Mascota', {
  nombre: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  raza: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  edad: {
    type: DataTypes.INTEGER,
    allowNull: false,
  }
}, {
  timestamps: false
});

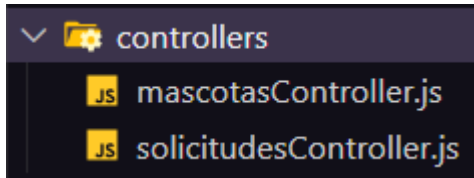
export default Mascota;
```

```
import { DataTypes } from 'sequelize';
import sequelize from '../config/db.js';

const SolicitudAdopcion = sequelize.define('SolicitudAdopcion', {
  nombre_interesado: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  direccion: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  telefono: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  email: {
    type: DataTypes.STRING,
    allowNull: false,
  }
}, {
  timestamps: false
});

export default SolicitudAdopcion;
```

Luego se creo los controladores para gestionar las operaciones de negocio, se creo la carpeta controllers y dentro de ella dos archivos mascotasController.js y solicitudesController.js uno para la getion de mascotas y la otra para la gestion de solicitudes de adopcion



```
import Mascota from '../models/Mascota.js';

// Obtener todas las mascotas
export const getMascotas = async (req, res) => {
  try {
    const mascotas = await Mascota.findAll();
    res.json(mascotas);
  } catch (error) {
    res.status(500).json({ message: 'Error al obtener mascotas', error });
  }
};

// Crear una nueva mascota
export const createMascota = async (req, res) => {
  try {
    const { nombre, raza, edad } = req.body;
    const nuevaMascota = await Mascota.create({ nombre, raza, edad });
    res.status(201).json(nuevaMascota);
  } catch (error) {
    res.status(500).json({ message: 'Error al crear mascota', error });
  }
};

// Actualizar una mascota
export const updateMascota = async (req, res) => {
  const { id } = req.params;
  const { nombre, raza, edad } = req.body;

  try {
    const mascota = await Mascota.findById(id);
    if (!mascota) return res.status(404).json({ message: 'Mascota no encontrada' });

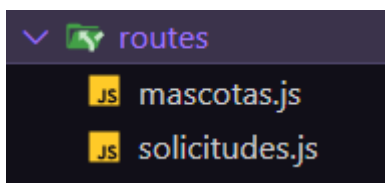
    await mascota.update({ nombre, raza, edad });
    res.json(mascota);
  } catch (error) {
    res.status(500).json({ message: 'Error al actualizar mascota', error });
  }
};
```

```
import SolicitudAdopcion from '../models/SolicitudAdopcion.js';

// Obtener todas las solicitudes de adopción
export const getSolicitudes = async (req, res) => {
  try {
    const solicitudes = await SolicitudAdopcion.findAll();
    res.json(solicitudes);
  } catch (error) {
    res.status(500).json({ message: 'Error al obtener solicitudes', error });
  }
};

// Crear una nueva solicitud de adopción
export const createSolicitud = async (req, res) => {
  try {
    const { nombre_interesado, direccion, telefono, email } = req.body;
    const nuevaSolicitud = await SolicitudAdopcion.create({ nombre_interesado, direccion, telefono, email });
    res.status(201).json(nuevaSolicitud);
  } catch (error) {
    res.status(500).json({ message: 'Error al crear solicitud', error });
  }
};
```

Luego se definió las rutas para gestionar las solicitudes y las mascotas, se creo una carpeta llamada routes y dentro de ella dos archivos llamados mascotas.js y solicitudes.js



```
import { Router } from 'express';
import {
  getMascotas,
  createMascota,
  updateMascota,
  deleteMascota,
} from '../controllers/mascotasController.js';

const router = Router();

// Rutas para mascotas
router.get('/visualizar', getMascotas); // GET visualizar todas las mascotas
router.post('/crear', createMascota); // POST crear una nueva mascota
router.put('/actualizar/:id', updateMascota); // PUT actualizar una mascota
router.delete('/borrar/:id', deleteMascota); // DELETE borrar una mascota

export default router;
```

```
import { Router } from 'express';
import {
  getSolicitudes,
  createSolicitud,
} from '../controllers/solicitudesController.js';

const router = Router();

// Rutas para solicitudes de adopción
router.get('/visualizar', getSolicitudes); // GET visualizar todas las solicitudes
router.post('/crear', createSolicitud); // POST crear una nueva solicitud

export default router;
```

Luego se configuro el archivo principal para usar las rutas y manejar la conexión a la base de datos para esto se creó el archivo app.js

 app.js

```
import express from 'express';
import bodyParser from 'body-parser';
import sequelize from './config/db.js';
import mascotasRoutes from './routes/mascotas.js';
import solicitudesRoutes from './routes/solicitudes.js';
import { errorHandler } from './middlewares/errorHandler.js';

const app = express();

// Middleware para parsear el cuerpo de las solicitudes
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

// Rutas
app.use('/api/mascotas', mascotasRoutes);
app.use('/api/solicitudes', solicitudesRoutes);



// Middleware de manejo de errores
app.use(errorHandler);

// Sincronizar la base de datos y arrancar el servidor
const startServer = async () => {
  try {
    await sequelize.sync(); // Sincroniza los modelos con la base de datos
    console.log('Base de datos sincronizada');

    const PORT = process.env.PORT || 3000;
    app.listen(PORT, () => {
      console.log(`Servidor corriendo en http://localhost:${PORT}`);
    });
  } catch (error) {
    console.error('Error al iniciar el servidor:', error);
  }
};

startServer();
```

También se creo una carpeta llamada middlewares y dentro de ella un archivo llamado errorHandler.js para poder manejar la parte de errores

▼  middlewares
 errorHandler.js

```
export const errorHandler = (err, req, res, next) => {
  console.error(err);
  res.status(500).json({ message: 'Ocurrió un error en el servidor' });
};
```

También se creó un archivo `.gitignore` para ignorar archivos y carpetas especificadas cuando se haga commits en el repositorio



A screenshot of a `.gitignore` file. At the top, there is a logo consisting of an orange diamond with a white 'G' inside, followed by the text `.gitignore`. Below this, the file contains several lines of text, each preceded by a hash symbol (`#`), indicating ignored files or directories. The ignored items include Node modules, logs, DBeaver directories, system configuration files, IDE configuration files, and other files like `.DS_Store` and `Thumbs.db`.

```
# Node modules
node_modules/

# Logs
logs
*.log
npm-debug.log*

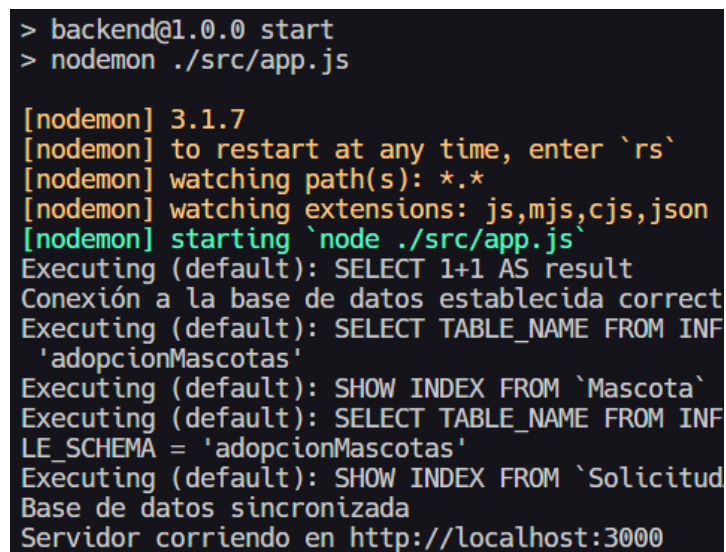
# Directorios de DBeaver
.dbeaver/

# Archivos de configuración del sistema
*.env

# Archivos de configuración de IDE
.idea/
.vscode/

# Otros archivos o directorios que quieras ignorar
.DS_Store
Thumbs.db
```

Una vez terminadas las configuraciones se verifica la conexión correcta con la base de datos



A screenshot of a terminal window showing the output of running `backend@1.0.0 start`. The terminal displays the Nodemon process starting, watching for file changes, and executing the application. It shows several SQL queries being executed, including a connection test, and finally reports that the database is synchronized and the server is running on `http://localhost:3000`.

```
> backend@1.0.0 start
> nodemon ./src/app.js

[nodemon] 3.1.7
[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
Conexión a la base de datos establecida correctamente
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'adopcionMascotas'
Executing (default): SHOW INDEX FROM `Mascota`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'adopcionMascotas'
Executing (default): SHOW INDEX FROM `Solicitud`
Base de datos sincronizada
Servidor corriendo en http://localhost:3000
```

Ahora se procede a realizar las diferentes pruebas en Thunder Client de VSC

Crear una nueva mascota

POST: <http://localhost:3000/api/mascotas/crear>

The screenshot shows the Thunder Client interface with a POST request to `http://localhost:3000/api/mascotas/crear`. The request body is a JSON object: `{ "nombre": "Firulais", "raza": "Labrador", "edad": 3 }`. The response status is `201 Created` with a size of `55 Bytes` and a time of `49 ms`. The response body is a JSON object: `{ "id": 1, "nombre": "Firulais", "raza": "Labrador", "edad": 3 }`.

The screenshot shows the Thunder Client interface with a POST request to `http://localhost:3000/api/mascotas/crear`. The request body is a JSON object: `{ "nombre": "Lulu", "raza": "Dalmata", "edad": 2 }`. The response status is `201 Created` with a size of `50 Bytes` and a time of `18 ms`. The response body is a JSON object: `{ "id": 2, "nombre": "Lulu", "raza": "Dalmata", "edad": 2 }`.

Visualizar las mascotas

GET: <http://localhost:3000/api/mascotas/visualizar>

The screenshot shows the Thunder Client interface with a GET request to `http://localhost:3000/api/mascotas/visualizar`. The response status is `200 OK` with a size of `108 Bytes` and a time of `14 ms`. The response body is a JSON array of two objects: `[{ "id": 1, "nombre": "Firulais", "raza": "Labrador", "edad": 3 }, { "id": 2, "nombre": "Lulu", "raza": "Dalmata", "edad": 2 }]`.

Actualizar una mascota

PUT: <http://localhost:3000/api/mascotas/actualizar/1>

The screenshot shows the Thunder Client interface with a PUT request to `http://localhost:3000/api/mascotas/actualizar/1`. The request body is a JSON object: `{ "nombre": "Firulais", "raza": "Pastor Aleman", "edad": 2 }`. The response status is `200 OK` with a size of `60 Bytes` and a time of `20 ms`. The response body is a JSON object: `{ "id": 1, "nombre": "Firulais", "raza": "Pastor Aleman", "edad": 2 }`.

Borrar una mascota

DELETE: <http://localhost:3000/api/mascotas/borrar/2>

DELETE <http://localhost:3000/api/mascotas/borrar/2> Send

Status: 204 No Content Size: 0 Bytes Time: 14 ms

Query Parameters

parameter	value
-----------	-------

Response

```
1
```

Verificamos los cambios realizados

GET <http://localhost:3000/api/mascotas/visualizar> Send

Status: 200 OK Size: 62 Bytes Time: 10 ms

Query Parameters

parameter	value
-----------	-------

Response

```
1 [
2   {
3     "id": 1,
4     "nombre": "Firulais",
5     "raza": "Pastor Aleman",
6     "edad": 2
7   }
8 ]
```

mascota X

Propiedades Datos Diagrama ER localhost

mascota Enter a SQL expression to filter results (use Ctrl+Space)

Grilla	id	nombre	raza	edad
1	1	Firulais	Pastor Aleman	2

Visualizar las solicitudes

GET: <http://localhost:3000/api/solicitudes/visualizar>

GET <http://localhost:3000/api/solicitudes/visualizar> Send

Status: 200 OK Size: 128 Bytes Time: 12 ms

Query Parameters

parameter	value
-----------	-------

Response

```
1 [
2   {
3     "id": 1,
4     "nombre_interesado": "Juan Perez",
5     "direccion": "Calle Falsa 123",
6     "telefono": "123456789",
7     "email": "juanperez@example.com"
8   }
9 ]
```


Crear una nueva solicitud

POST: <http://localhost:3000/api/solicitudes/crear>

The screenshot shows a REST client interface with a POST request to `http://localhost:3000/api/solicitudes/crear`. The request body is a JSON object: `{ "nombre_interesado": "Daniel Rivera", "direccion": "Cra 6 No123", "telefono": "123456789", "email": "daniel@example.com" }`. The response status is `201 Created` with a size of `122 Bytes` and a time of `15 ms`. The response body is a JSON object: `{ "id": 2, "nombre_interesado": "Daniel Rivera", "direccion": "Cra 6 No123", "telefono": "123456789", "email": "daniel@example.com" }`.

Verificamos los cambios

The screenshot shows a REST client interface with a GET request to `http://localhost:3000/api/solicitudes/visualizar`. The response status is `200 OK` with a size of `251 Bytes` and a time of `14 ms`. The response body is a JSON array of two objects: `[{ "id": 1, "nombre_interesado": "Juan Perez", "direccion": "Calle Falsa 123", "telefono": "123456789", "email": "juanperez@example.com" }, { "id": 2, "nombre_interesado": "Daniel Rivera", "direccion": "Cra 6 No123", "telefono": "123456789" }]`.

The screenshot shows a database management tool interface with a table named `solicitudadopciones`. The table has columns: `id`, `nombre_interesado`, `direccion`, `telefono`, and `email`. The table contains two records.

	id	nombre_interesado	direccion	telefono	email
1	1	Juan Perez	Calle Falsa 123	123456789	juanperez@example.com
2	2	Daniel Rivera	Cra 6 No123	123456789	daniel@example.com