

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

Taller Unidad 2 Backend.

Estudiante:

Yeison Oswaldo Ruano Ortega

Código:

220036075

Universidad de Nariño.

Ingeniería de Sistemas.

Semestre X

Ipiales – Nariño

2024

1 Crear una base de datos MYSQL/MARIADB que permita llevar el registro de una empresa de adopción de mascotas, debe soportar la administración de las mismas y la posibilidad de registrar solicitudes de adopción. (1 Pto).

Para la creación de la base de datos se utilizó los comandos

-- Crear la base de datos

```
CREATE DATABASE empresa_adopcion;
```

-- Tabla de mascotas

```
CREATE TABLE mascotas (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  nombre VARCHAR(50) NOT NULL,  
  especie VARCHAR(50) NOT NULL,  
  raza VARCHAR(50),  
  edad INT,  
  descripcion TEXT,  
  estado ENUM('disponible', 'adoptado') DEFAULT 'disponible',  
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
  fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

-- Tabla de solicitudes de adopción

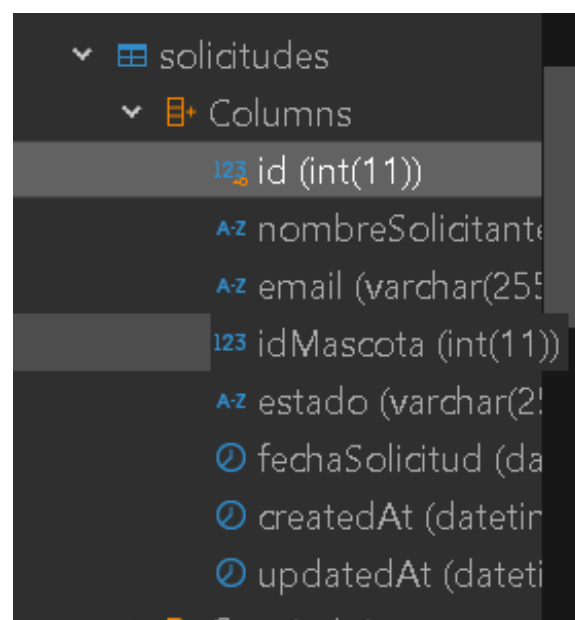
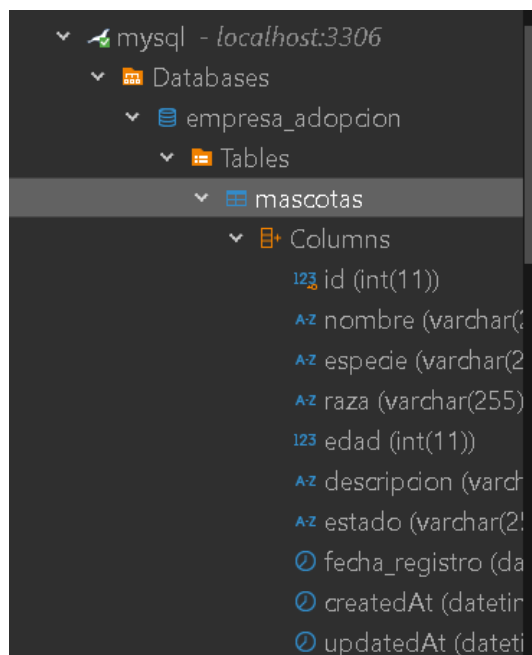
```
CREATE TABLE solicitudes_adopcion (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  idMascota INT,  
  nombreSolicitante VARCHAR(100) NOT NULL,  
  email VARCHAR(100) NOT NULL,
```

```
estado VARCHAR(20),  
createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
fecha_solicitud TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
estado ENUM('pendiente', 'aprobada', 'rechazada') DEFAULT 'pendiente',  
FOREIGN KEY (idMascotas) REFERENCES mascotas(id)
```

```
);
```

Tablas y base de datos



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 (La empresa debe contar con un nombre), así como también las solicitudes de adopción. Se debe hacer uso correcto de los verbos HTTP dependiendo de la tarea a realizar. (3 Ptos).

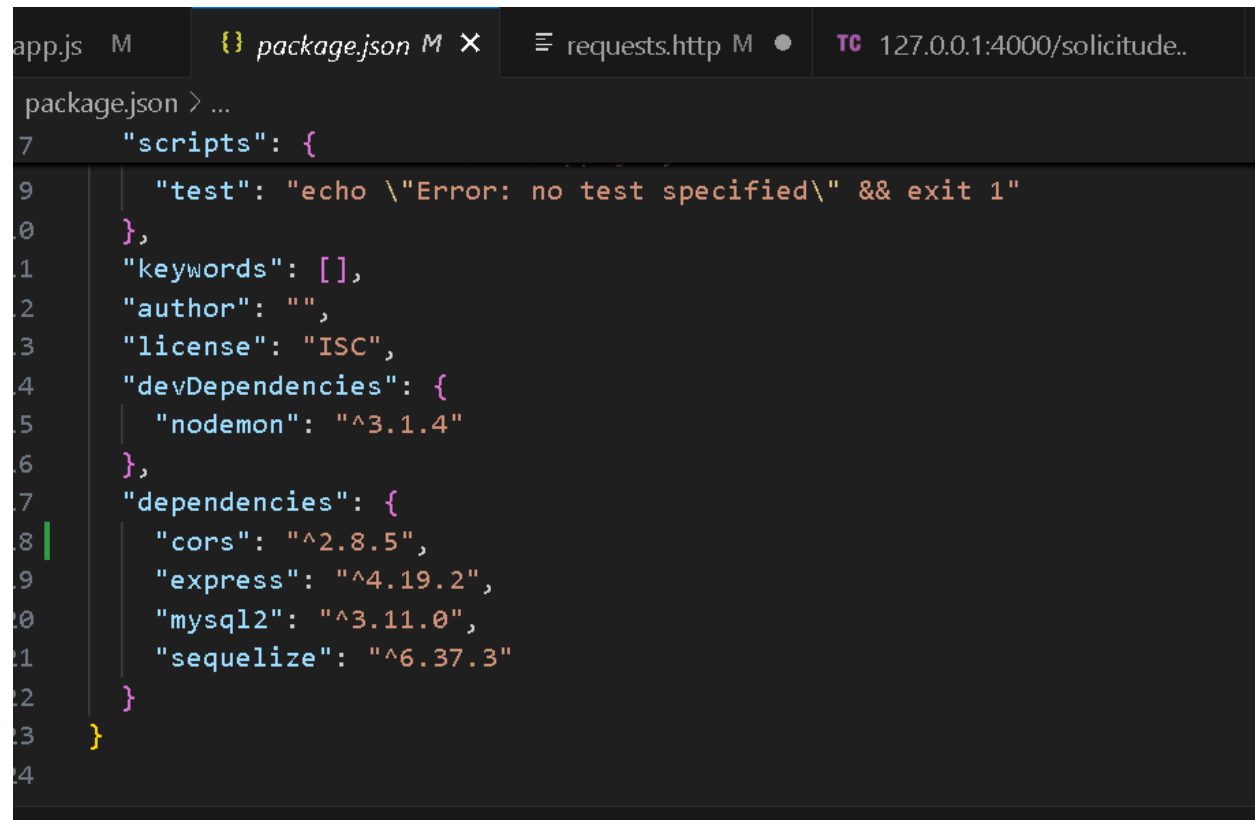
Para el desarrollo del backend se utilizao las siguientes Herramientas:

npm install nodemon -D

npm install express

npm install mysql2

npm install sequelize



```
app.js M    {} package.json M X    ≡ requests.http M ●    TC 127.0.0.1:4000/solicitud..  
package.json > ...  
7   "scripts": {  
9     "test": "echo \"Error: no test specified\" && exit 1"  
0   },  
1   "keywords": [],  
2   "author": "",  
3   "license": "ISC",  
4   "devDependencies": {  
5     "nodemon": "^3.1.4"  
6   },  
7   "dependencies": {  
8     "cors": "^2.8.5",  
9     "express": "^4.19.2",  
0     "mysql2": "^3.11.0",  
1     "sequelize": "^6.37.3"  
2   }  
3 }  
4
```

Después se realizó la implementación de los verbos http en mascotas y solicitudes:

POST para crear (/crear).

GET para leer (lista o detalle /y /:id).

PUT para actualizar (editar datos /:id).

DELETE para eliminar (/:id).

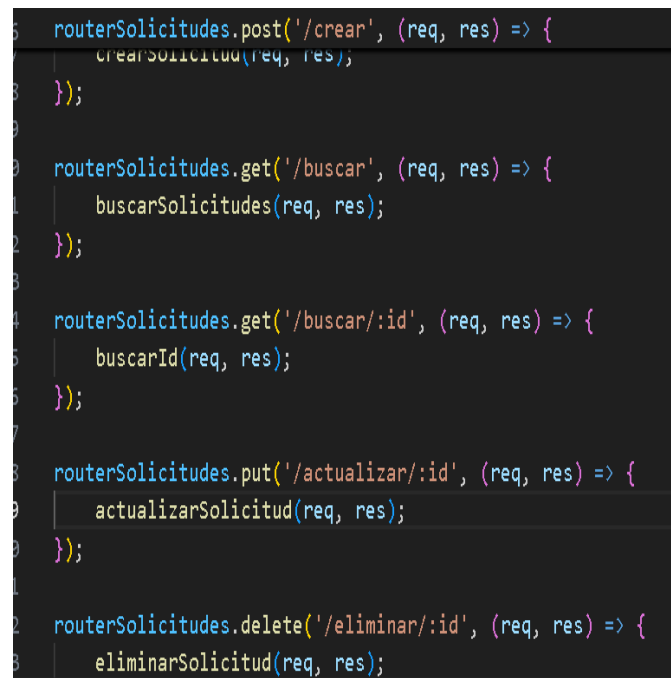
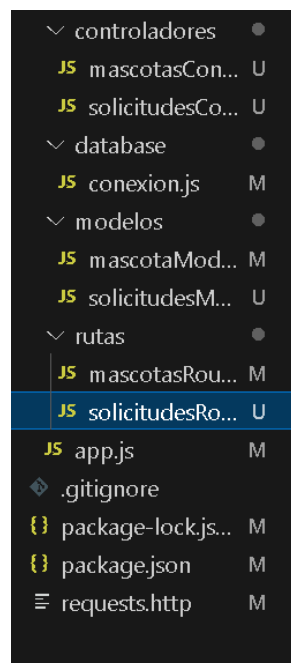
Mascotas

- POST /mascotas/crear: Crear una nueva mascota.
- GET /mascotas/:id: Obtener una mascota por ID.
- PUT /mascotas/:id: Actualizar una mascota.
- DELETE /mascotas/:id: Eliminar una mascota.

Solicitudes de Adopción

- POST /solicitudes/crear: Crear una nueva solicitud de adopción.
- GET /solicitudes/:id: Obtener una solicitud por ID.
- PUT /solicitudes/:id: Actualizar una solicitud de adopción.
- DELETE /solicitudes/:id: Eliminar una solicitud de adopción.

```
1 routerMascotas.post('/crear', (req, res) => {
2   //res.send('Crear Mascota');
3   crear(req,res);
4 });
5
6 routerMascotas.get('/buscar', (req, res) => {
7   //res.send('Buscar Mascota');
8   buscar(req,res);
9 });
10 routerMascotas.delete('/eliminar/:id', (req, res) => {
11   eliminar(req,res);
12 });
13
14 routerMascotas.put('/actualizar/:id', (req, res) => {
15   //res.send('Actualizar Mascota');
16   actualizar(req,res);
17 })
```



El código incluye controladores para cada acción que interactúan con la base de datos usando **Sequelize** para ejecutar las operaciones de manera eficiente, y las rutas están correctamente configuradas para cada operación con los verbos HTTP adecuados.

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 Pto).

Se empieza inicializando el servicio con:

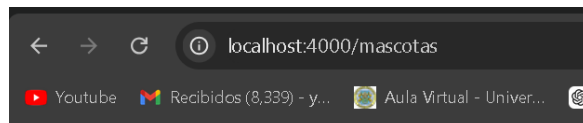
Npm run start

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Executing (default): SHOW INDEX FROM `mascotas`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'solicitudes' AND TABLE_SCHEMA = 'empresa_adopcion'
Executing (default): SHOW INDEX FROM `solicitudes`
Servidor Inicializado en el puerto 4000
```

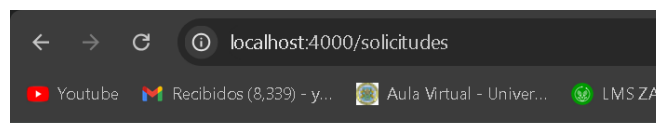
Revisamos que este corriendo el servicio



empresa_adopcion Sitio Principal

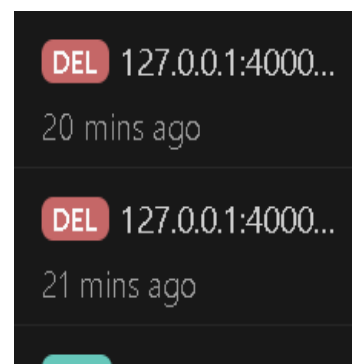
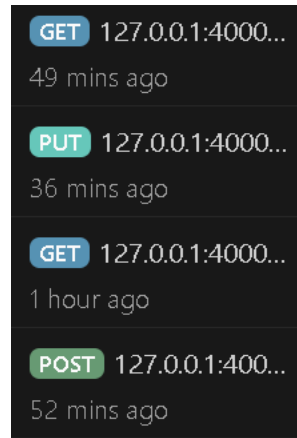
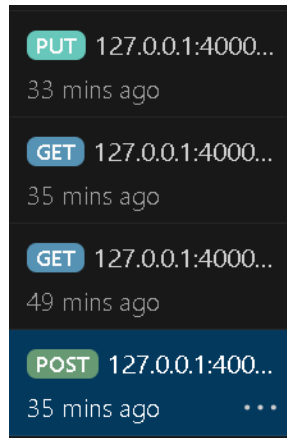


empresa_adopcion este es el Sitio de Mascotas

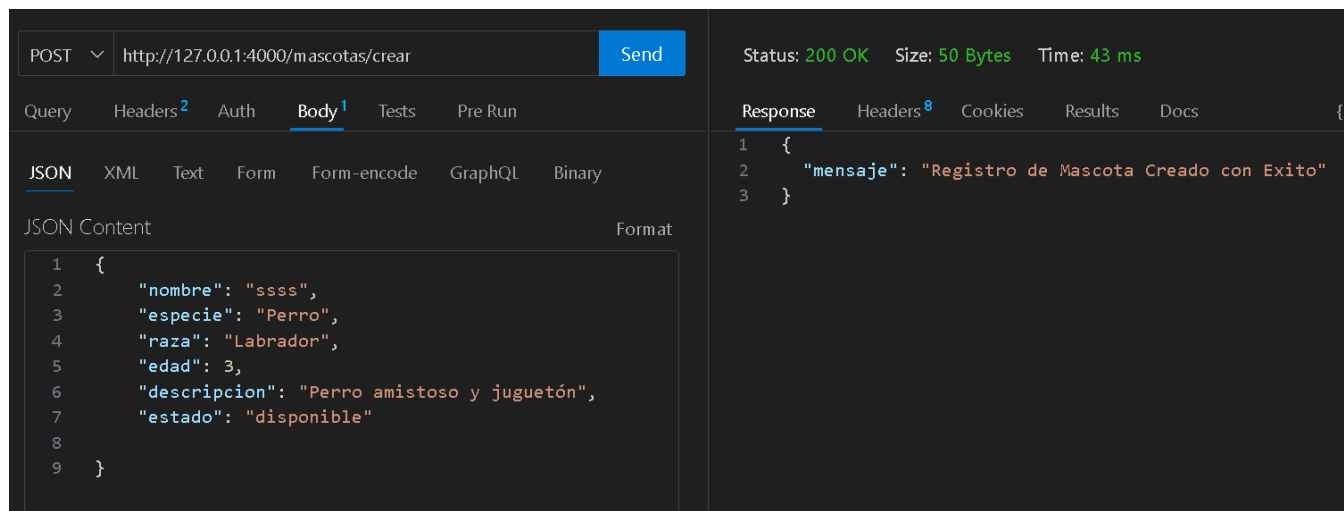


empresa_adopcion este es el Sitio de Solicitudes de Adopción

Ahora vamos a utilizar thunder client para hacer pruebas



Crear mascota



Buscar mascota

GET

http://127.0.0.1:4000/mascotas/buscar

Send

Query

Headers²

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

Status: 200 OK

Size: 512 Bytes

Time: 8 ms

Response

Headers⁸

Cookies

Results

Docs

{}

1

[

2

{

3

"id": 2,

4

"nombre": "s",

5

"especie": "Perro",

6

"raza": "Labrador",

7

"edad": 3,

8

"descripcion": "Perro amistoso y juguetón",

9

"estado": "disponible",

10

"fecha_registro": "2024-09-20T19:06:47.000Z",

11

"createdAt": "2024-09-20T19:06:47.000Z",

12

"updatedAt": "2024-09-20T19:06:47.000Z"

13

},

14

{

15

"id": 3,

16

"nombre": "ssss",

17

"especie": "Perro",

Response

Chart

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

node

+

⌵

⌵

⌵

⌵

⌵

⌵

Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'solic

des' AND TABLE_SCHEMA = 'lampaca_adopcion'

Actualizar mascota

PUT

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

Send

Query

Headers²

Auth

Body¹

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

1

{

2

"nombre": "sassa",

3

"especie": "Perro"

4

}

5

Status: 200 OK

Size: 51 Bytes

Time: 16 ms

Response

Headers⁸

Cookies

Results

Docs

1

{

2

"tipo": "success",

3

"mensaje": "Registro Actualizado"

4

}

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

node

+

⌵

⌵

⌵

⌵

⌵

⌵

Buscar id mascota

GET

▼

http://127.0.0.1:4000/mascotas/buscar/1

Send

Query

Headers²

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

Status: 200 OKSize: 258 BytesTime: 10 ms

Response

Headers⁸

Cookies

Results

Docs

1

{

2

"id": 1,

3

"nombre": "megane",

4

"especie": "Perro",

5

"raza": "Labrador",

6

"edad": 3,

7

"descripcion": "Perro amistoso y juguetón",

8

"estado": "disponible",

9

"fecha_registro": "2024-09-20T19:01:53.000Z",

10

"createdAt": "2024-09-20T19:01:53.000Z",

11

"updatedAt": "2024-09-20T19:06:12.000Z"

12

}

Eliminar por id mascota

DELETE

▼

http://127.0.0.1:4000/mascotas/eliminar/1

Send

Query

Headers²

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

Status: 200 OKSize: 72 BytesTime: 27 ms

Response

Headers⁸

Cookies

Results

Docs

{}

≡

1

{

2

"tipo": "success",

3

"mensaje": "Registro con id 1 Eliminado Correctamente"

4

}

Tabla de mascotas

[illegible]

SOLICITUDES

Crear solicitud

The screenshot displays the VS Code interface with a REST client extension. The top bar shows the method **POST** and the URL **http://127.0.0.1:4000/solicitudes/crear**. The **Send** button is highlighted in blue. Below the URL bar, tabs for **Query**, **Headers**, **Auth**, **Body**, **Tests**, and **Pre Run** are visible. The **Body** tab is active, showing a JSON payload:

```
1 {
2   "nombreSolicitante": "Juan aaaaaas",
3   "email": "juan.perez@example.com",
4   "idMascota": 1
5 }
```

The **JSON Content** and **Format** buttons are located below the editor. The right sidebar shows the **Response** tab, indicating a successful status of **201 Created** with a size of **299 Bytes** and a time of **12 ms**. The response body is a JSON object:

```
1 {
2   "mensaje": "Solicitud de adopción creada con éxito",
3   "resultado": {
4     "fechaSolicitud": "2024-09-20T19:58:12.703Z",
5     "id": 4,
6     "nombreSolicitante": "Juan aaaaaas",
7     "email": "juan.perez@example.com",
8     "idMascota": 1,
9     "estado": "pendiente",
10    "updatedAt": "2024-09-20T19:58:12.704Z",
11    "createdAt": "2024-09-20T19:58:12.704Z"
12  }
13 }
```

A **Copy** button is visible next to the response text. The bottom status bar shows **node** and various icons for file management and debugging.

Buscar solicitud

GET

http://127.0.0.1:4000/solicitudes/buscar

Send

Query

Headers²

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

Status: 200 OK

Size: 689 Bytes

Time: 4 ms

Response

Headers⁸

Cookies

Results

Docs

{}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

[

{

"id": 2,

"nombreSolicitante": "Juan s",

"email": "juan.perez@example.com",

"idMascota": 1,

"estado": "cerrado",

"fechaSolicitud": "2024-09-20T19:37:03.000Z",

"createdAt": "2024-09-20T19:37:03.000Z",

"updatedAt": "2024-09-20T19:57:56.000Z"

},

{

"id": 3,

"nombreSolicitante": "Juan as",

"email": "juan.perez@example.com",

"idMascota": 1,

"estado": "pendiente",

Copy

Response

Chart

Buscar por id solicitudes

GET

▼

http://127.0.0.1:4000/solicitudes/buscar/4

Send

Query

Headers2

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

Status: 200 OK

Size: 232 Bytes

Time: 4 ms

Response

Headers8

Cookies

Results

Docs

1

{

2

"id": 4,

3

"nombreSolicitante": "Juan aaaaaas",

4

"email": "juan.perez@example.com",

5

"idMascota": 1,

6

"estado": "pendiente",

7

"fechaSolicitud": "2024-09-20T19:58:12.000Z",

8

"createdAt": "2024-09-20T19:58:12.000Z",

9

"updatedAt": "2024-09-20T19:59:42.000Z"

10

}

Response

Char

Actualizar por id solicitud

PUT

▼

http://127.0.0.1:4000/solicitudes/actualizar/4

Send

Query

Headers2

Auth

Body1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

1

{

2

"estado": "pendiente"

3

}

Status: 200 OK

Size: 47 Bytes

Time: 19 ms

Response

Headers8

Cookies

Results

Docs

1

{

2

"mensaje": "Solicitud actualizada con éxito."

3

}

Response

Char

Eliminar por id solicitud

DELETE

▼

http://127.0.0.1:4000/solicitudes/eliminar/1

Send

Query

Headers²

Auth

Body

Tests

Pre Run

Query Parameters

☐

parameter

value

Status: 200 OKSize: 45 BytesTime: 21 ms

Response

Headers⁸

Cookies

Results

Docs

1

{

2

"mensaje": "Solicitud eliminada con éxito."

3

}

Response

Char

Tabla solicitudes

[illegible]