

Manual de uso del proyecto API REST

Requerimientos:

- Tener instalado NodeJS

Descarga, instalación y inicio del servicio de API

1- Descargar el repositorio del proyecto de GitHub del siguiente link y luego descomprimirlo:
[jdgnpy/testedge: API REST con NodeJS, Express, JWT y SQLite \(github.com\)](https://github.com/jdgnp/testedge)

2- Abrir el proyecto en su IDE preferido o abrir el cmd en la ruta del proyecto y ejecutamos el siguiente comando para instalar las dependencias del proyecto:

> npm i

```
PS D:\testedge-master> npm i
[.....] \ idealTree:testedge-master: sill idealTree buildDeps
```

3- Una vez terminado de instalar todas las dependencias, ejecutar el siguiente comando para iniciar el servicio de API:

> npm run start

o también pueden usar nodemon para iniciar el servicio con el siguiente comando:

> npm run server

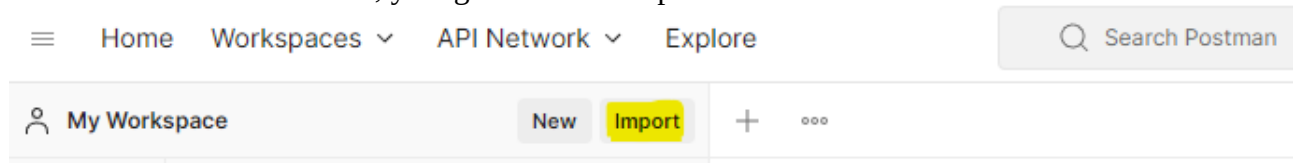
```
> testedge@1.0.0 server
> nodemon server.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
API Server corriendo en el puerto 5000
Conectado a la base de datos
```

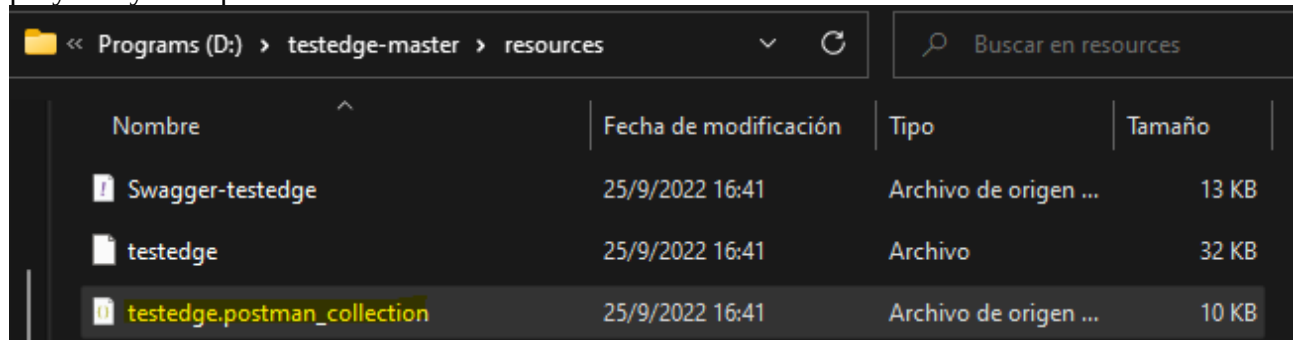
Si todo salio bien les saldrá un mensaje similar, haciendo referencia que se inicio el servicio

Importación del collection y testeo de API con Postman

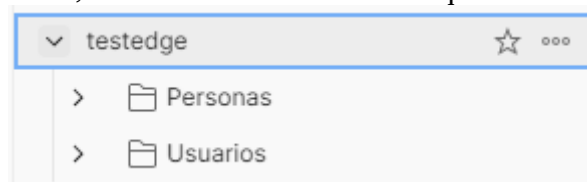
1- Abrimos nuestro Postman, y luego damos en importar:



2- Buscamos el archivo “testedge.postman_collection” que se encuentra en la carpeta resources del proyecto y lo importamos:

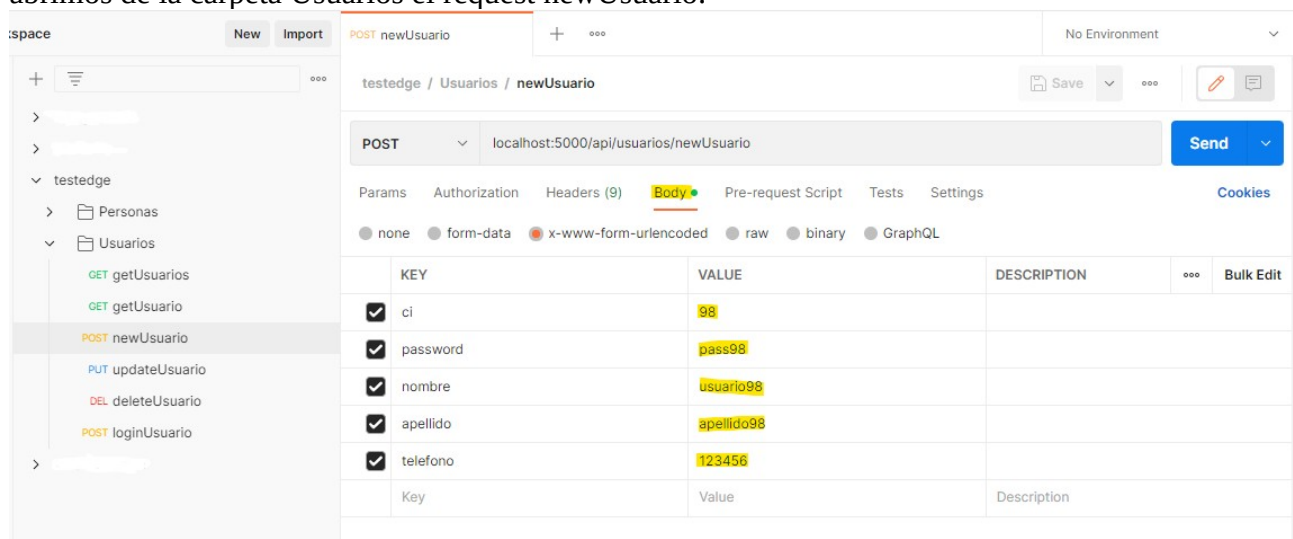


3- Una vez importado el archivo, se tendrá la colección de request de la API:



Se tiene 2 carpetas, uno para los request de Personas y otro para Usuarios

4- Se abre uno de los request, primeramente es necesario registrar un nuevo usuario para poder obtener en el login el token a utilizar para poder realizar las demas request de la colección, para ello abrimos de la carpeta Usuarios el request newUsuario:



Editamos los valores que serán enviados en el Body, en este caso se puede enviar los siguientes campos para crear un nuevo usuario:

- ci (campo numérico y requerido)
- password (campo texto y requerido)
- nombre (campo texto y requerido)
- apellido (campo texto)
- telefono (campo numérico)

5- Una vez terminado de cargar los datos para crear el nuevo usuario, realizamos el envío dando click en Send:

The screenshot shows a REST client interface with a POST request to `localhost:5000/api/usuarios/newUsuario`. The request body is configured as `x-www-form-urlencoded` with the following data:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> ci	98	
<input checked="" type="checkbox"/> password	pass98	
<input checked="" type="checkbox"/> nombre	usuario98	
<input checked="" type="checkbox"/> apellido	apellido98	
<input checked="" type="checkbox"/> telefono	123456	

Esto retornara una respuesta proveniente de la API:

The screenshot shows the response body of the API call, which is a JSON object:

```
1 {
2   "success": true,
3   "msg": "El usuario usuario98 fue registrado exitosamente",
4   "ci": "98",
5   "nombre": "usuario98",
6   "apellido": "apellido98",
7   "telefono": "123456"
8 }
```

En el cual se observa que dio una respuesta 201 de creado y retorno una respuesta el cual muestra el campo de success como true, un mensaje y luego los datos con el cual fue creado el usuario

6- Con el usuario creado procedemos a realizar el request de loginUsuario y verificamos la respuesta:

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

- Request Method:** POST
- Request URL:** localhost:5000/api/usuarios/loginUsuario
- Request Body (x-www-form-urlencoded):**

KEY	VALUE	DESCRIPTION
ci	98	
password	pass98	
Key	Value	Description
- Response Status:** 200 OK, 12 ms, 418 B
- Response Body (JSON):**

```
{  "success": true,  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjaSI6OTgsIm1hdCI6MTY2NDE0MTE3MiwiZXhwIjoxNjY0MTY5OTcyfQ.ils1QWgzPxiwNEApY30ltyQ6-mxQ6aRc8CIwgodt46I",  "duracion": "8h"}
```

Acá la respuesta da un token el cual tiene una duración de 8 horas, este token es el que servirá para realizar los demás request a la API, entonces lo copiamos.

7- Realizamos el request de getUsersarios agregando el token en el Header de la siguiente manera:
authorization: Bearer + token

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

- Request:** GET localhost:5000/api/usuarios/getUsuarios
- Headers:** authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjaSI6OT...
- Response:** 200 OK, 8 ms, 767 B
- Response Body (JSON):**

```
{
  "ci": 98,
  "nombre": "usuario98",
  "apellido": "apellido98",
  "telefono": 123456
},
{
  "ci": 99,
  "nombre": "administrador",
  "apellido": null,
  "telefono": null
}
```

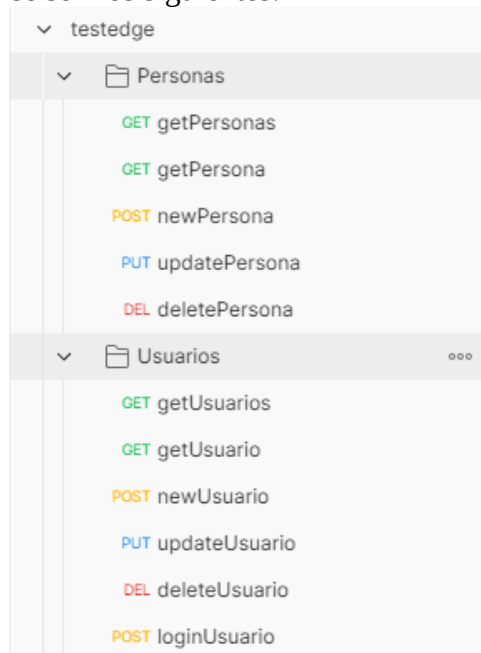
En caso de haber puesto mal el token o el token sea invalido saldrá el siguiente mensaje como respuesta:

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

- Request:** GET localhost:5000/api/usuarios/getUsuarios
- Headers:** authorization: Bearer qqeqyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjaSI6OT...
- Response:** 401 Unauthorized, 6 ms, 307 B
- Response Body (JSON):**

```
{
  "sucess": false,
  "msg": "Error, No autorizado - Token Invalido"
}
```

8- Los Request a poder realizarse son los siguientes:



Para mas información se puede revisar el archivo Swagger en la carpeta resources del proyecto

API testedge 1.0.0 OAS3

API realizada con NodeJS, Express y SQLite

Servers

http://localhost:5000/api/

usuarios CRUD de la tabla usuarios de la base de datos

GET	/usuarios/getUsuarios	Obtiene los datos de todos los usuarios	▼
GET	/usuarios/getUsuario/{ci}	Obtiene los datos de un usuario específico mediante el ci	▼
POST	/usuarios/newUsuario	Crea un nuevo usuario en la bd	▼
PUT	/usuarios/updateUsuario/{ci}	Actualiza los datos de un usuario	▼
DELETE	/usuarios/deleteUsuario/{ci}	Elimina un usuario de la bd	▼






personas CRUD de la tabla personas de la base de datos

GET	/personas/getPersonas	Obtiene los datos de todas las personas	▼
GET	/personas/getPersona/{ci}	Obtiene los datos de una persona específica mediante el ci	▼
POST	/personas/newPersona	Crea una nueva persona en la bd	▼
PUT	/personas/updatePersona/{ci}	Actualiza los datos de una persona	▼
DELETE	/personas/deletePersona/{ci}	Elimina una persona de la bd	▼


Schemas

Estructura de la base de datos(SQLite) utilizada

1- Tabla usuarios

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL
1	ci	INTEGER					
2	password	TEXT					
3	nombre	TEXT					
4	apellido	TEXT					
5	telefono	INTEGER					

2- Tabla personas

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL
1	id	INTEGER					
2	nombre	TEXT					
3	apellido	TEXT					
4	direccion	TEXT					
5	telefono	INTEGER					
6	ci	INTEGER			