

Lab - Uso de Express para acceso a Base de datos via API

Creación de base de datos con Node.js y PostgreSQL

Express es un framework de aplicaciones web Node.js mínimo y flexible que proporciona un conjunto robusto de características para desarrollar aplicaciones web y móviles. Facilita el rápido desarrollo de aplicaciones web basadas en Node. Las siguientes son algunas de las características principales del framework Express

Permite configurar middlewares para responder a peticiones HTTP.

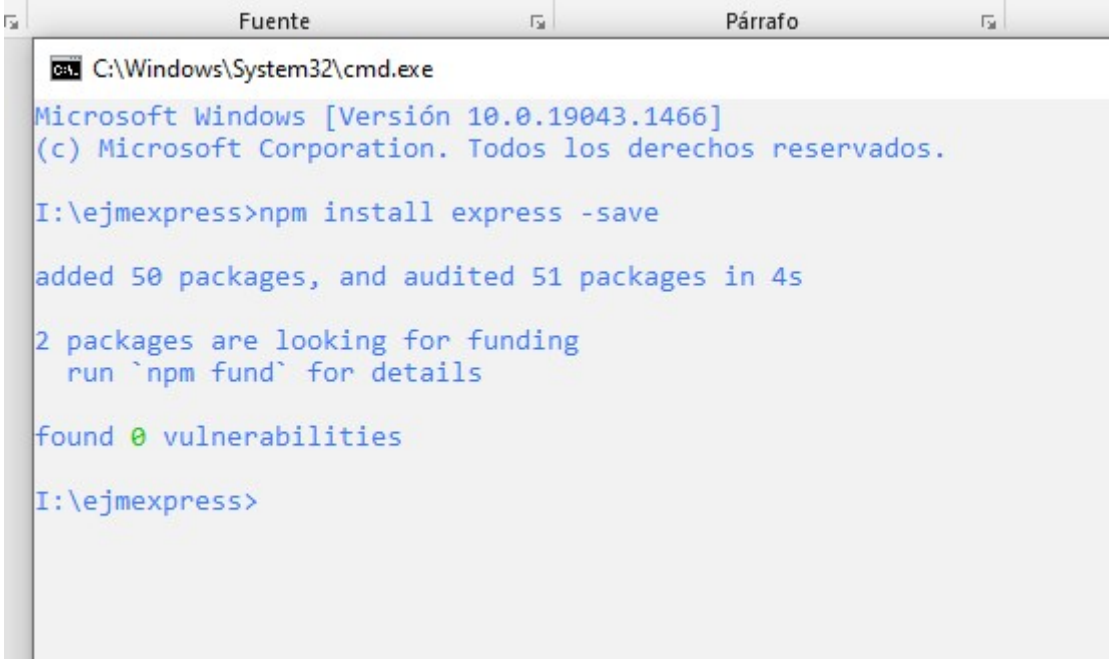
Define una tabla de enrutamiento que se utiliza para realizar diferentes acciones basadas en el método HTTP y la URL.

Permite renderizar dinámicamente páginas HTML basadas en el paso de argumentos a las plantillas.

Instalación de Express

En primer lugar, instala el framework Express de forma global usando NPM para que pueda ser usado para crear una aplicación web usando node terminal.

\$ npm install express --save



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19043.1466]
(c) Microsoft Corporation. Todos los derechos reservados.

I:\ejmexpress>npm install express -save

added 50 packages, and audited 51 packages in 4s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

I:\ejmexpress>
```

lo que provoco en la carpeta usada la siguiente organización:

Este equipo > Disco local (I:) > ejmexpress >				
	Nombre	Fecha de modificación	Tipo	Tamaño
	node_modules	1/02/2022 12:47	Carpeta de archivos	
	package.json	1/02/2022 12:47	Archivo de origen ...	1 KB
	package-lock.json	1/02/2022 12:47	Archivo de origen ...	32 KB

El comando anterior guarda la instalación localmente en el directorio `node_modules` y crea un directorio `express` dentro de `node_modules`.

A continuación, se debe instalar los siguientes módulos importantes junto con `express` —

body-parser - Este es un middleware de `node.js` para manejar datos de formularios codificados en JSON, Raw, Texto y URL.

cookie-parser - Analiza la cabecera de las cookies y rellena `req.cookies` con un objeto con los nombres de las cookies.

multer - Este es un middleware de `node.js` para manejar datos multipart/form-data

```
$ npm install body-parser --save
```

```
$ npm install cookie-parser --save
```

```
$ npm install multer --save
```

```
I:\ejmexpress>npm install body-parser --save

up to date, audited 51 packages in 905ms

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

I:\ejmexpress>npm install cookie-parser --save

added 1 package, and audited 52 packages in 2s

2 packages are looking for funding
  run `npm fund` for details

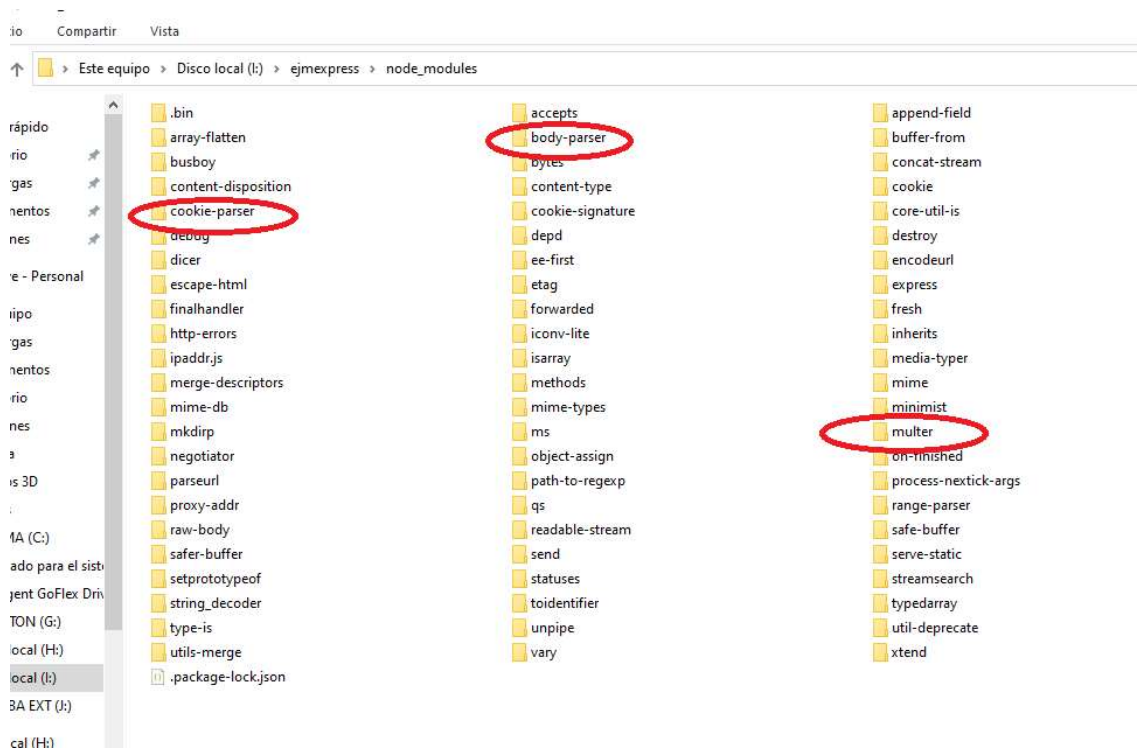
found 0 vulnerabilities

I:\ejmexpress>npm install multer --save

added 22 packages, and audited 74 packages in 4s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```



Ejemplo Básico de Express

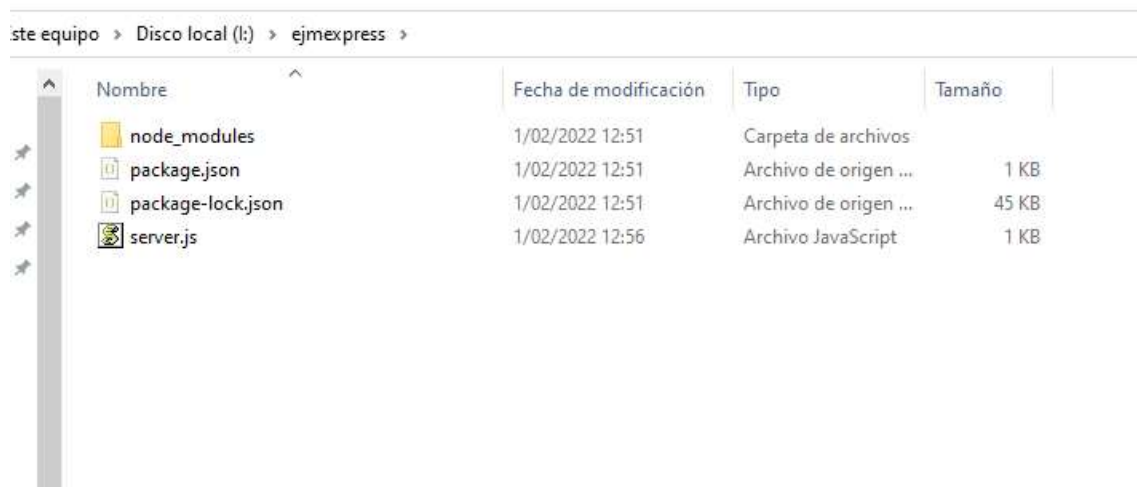
A continuación se muestra una aplicación Express muy básica que inicia un servidor y escucha en el puerto 8081 para la conexión. Esta aplicación responde con "¡Hola

Mundo!" para las peticiones a la página de inicio. Para cualquier otra ruta, responderá con un 404 Not Found.

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Uso basico de Express');
})
var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port
  console.log ("Ejemplo de app listening en http://%s:%s",
host, port)
})
```

Guarde el código anterior en un archivo llamado server.js y ejecútelo con el siguiente comando:

```
$ node server.js
```

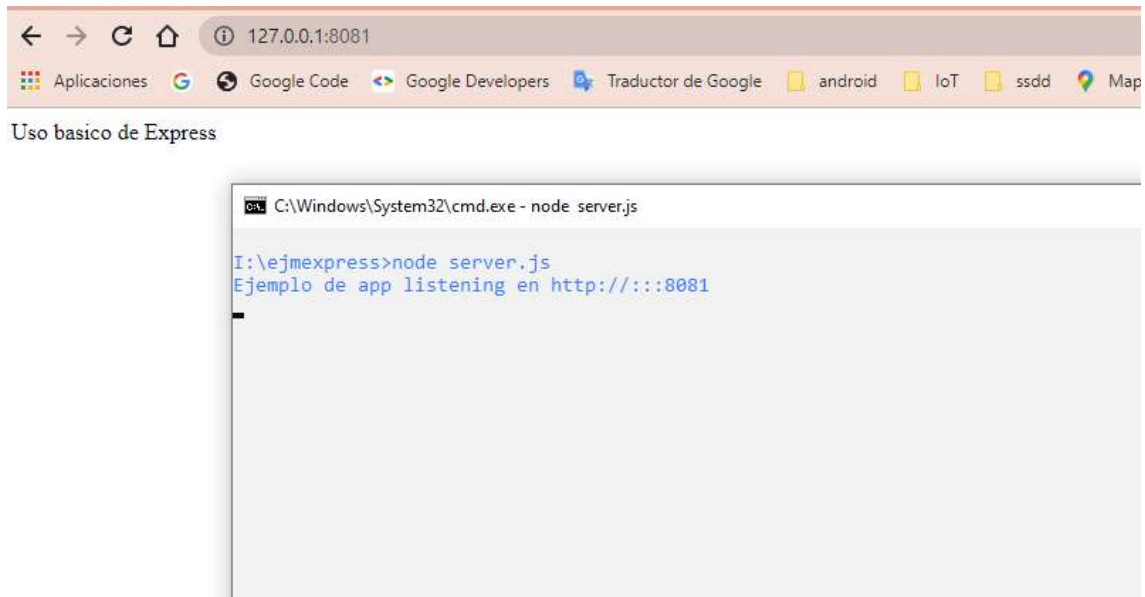


Nombre	Fecha de modificación	Tipo	Tamaño
node_modules	1/02/2022 12:51	Carpeta de archivos	
package.json	1/02/2022 12:51	Archivo de origen ...	1 KB
package-lock.json	1/02/2022 12:51	Archivo de origen ...	45 KB
server.js	1/02/2022 12:56	Archivo JavaScript	1 KB

Se observa la siguiente salida

Ejemplo de aplicación escuchando en `http://0.0.0.0:8081`

Se abre `http://127.0.0.1:8081/` en cualquier navegador para ver el siguiente resultado.



Solicitud y respuesta

La aplicación Express utiliza una función callback cuyos parámetros son los objetos request y response.

```
app.get('/', function (req, res) {  
    // --  
})
```

Objeto Request - El objeto request representa la petición HTTP y tiene propiedades para la cadena de consulta de la petición, los parámetros, el cuerpo, las cabeceras HTTP, etc.

Objeto Response - El objeto response representa la respuesta HTTP que una aplicación Express envía cuando recibe una petición HTTP.

Se puede imprimir los objetos req y res que proporcionan gran cantidad de información relacionada con la solicitud y la respuesta HTTP, incluyendo cookies, sesiones, URL, etc.

Routing básico

Se ha visto una aplicación básica que sirve peticiones HTTP para la página de inicio. El enrutamiento se refiere a la determinación de cómo una aplicación responde a una solicitud del cliente a un punto final particular, que es un URI (o ruta) y un método de solicitud HTTP específico (GET, POST, etc.).

Se va a extender el ejemplo anterior para manejar más tipos de peticiones HTTP.

```
var express = require('express');
var app = express();
// Esto responde con "mensaje" en la página de inicio
app.get('/', function (req, res) {
  console.log("Tengo una petición GET para la página de inicio");
  res.send('Hola GET');
})

// Esto responde a una petición POST para la página de inicio
app.post('/', function (req, res) {
  console.log("Recibí una solicitud POST para la página de inicio");
  res.send('Hola POST');
})

// Esto responde a una petición DELETE para la página /del_user.
app.delete('/del_user', function (req, res) {
  console.log("Tengo una petición DELETE para /del_user");
  res.send('Hola DELETE');
})

// Esto responde a una petición GET para la página /list_user.
app.get('/list_user', function (req, res) {
  console.log("Tengo una petición GET para /list_user");
  res.send("Listado de páginas");
})

// Esto responde a una petición GET para abcd, abxcd, ab123cd, etc.
app.get('/ab*cd', function(req, res) {
  console.log("Tengo una petición GET para /ab*cd");
  res.send("Coincidencia de patrón de página");
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port
```

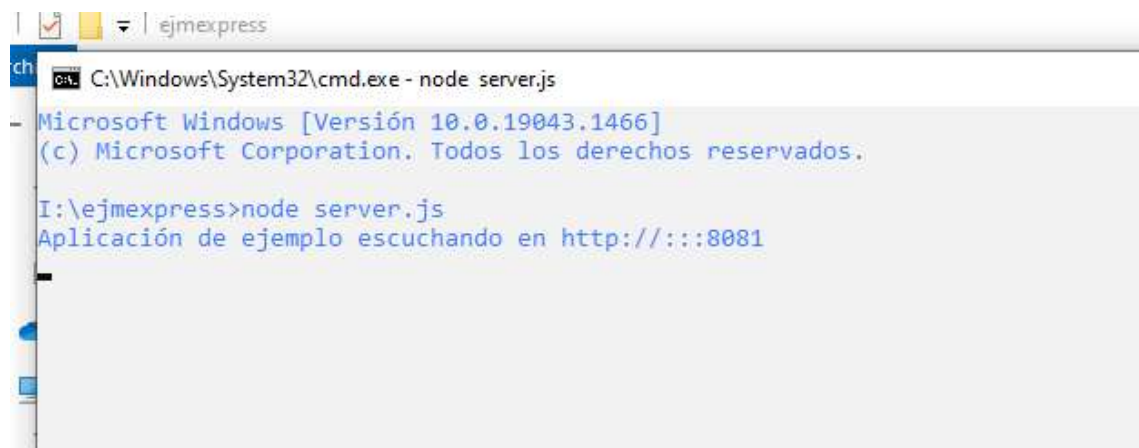
```
    console.log("Aplicación de ejemplo escuchando en  
http://%s:%s", host, port)  
})
```

Guarda el código anterior en un archivo llamado server.js y ejecútalo con el siguiente comando

```
$ node server.js
```

Se observará la siguiente salida

Ejemplo de aplicación escuchando en http://0.0.0.0:8081



Ahora puedes probar diferentes peticiones en http://127.0.0.1:8081 para ver la salida generada por server.js. A continuación, hay algunas capturas de pantalla que muestran diferentes respuestas para diferentes URLs.

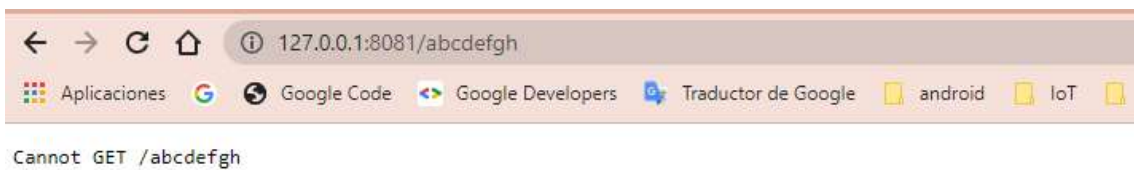
Pantalla mostrando de nuevo http://127.0.0.1:8081/list_user



http://127.0.0.1:8081/abcd



`http://127.0.0.1:8081/abcdefgh`



Mas información sobre Express, existe en varios tutoriales sobre el tema, como el siguiente:

https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

Este será el servidor que creará el acceso api a la base de datos, que en este caso, será PostgreSQL.

Archivos de Configuración

Son 2 los archivos de configuración que serán usados para este proyecto.

- a. postgresql-express-batch: carpeta que contiene un modulo node que se encargara de crear la base de datos y las tablas usadas por la app Angular estudiada.

- b. postgresql-express-crud: carpeta que contiene el modulo node.js que levanta el servidor express y carga las operaciones CRUD con PostgreSQL

Cambio de clave PostgreSQL

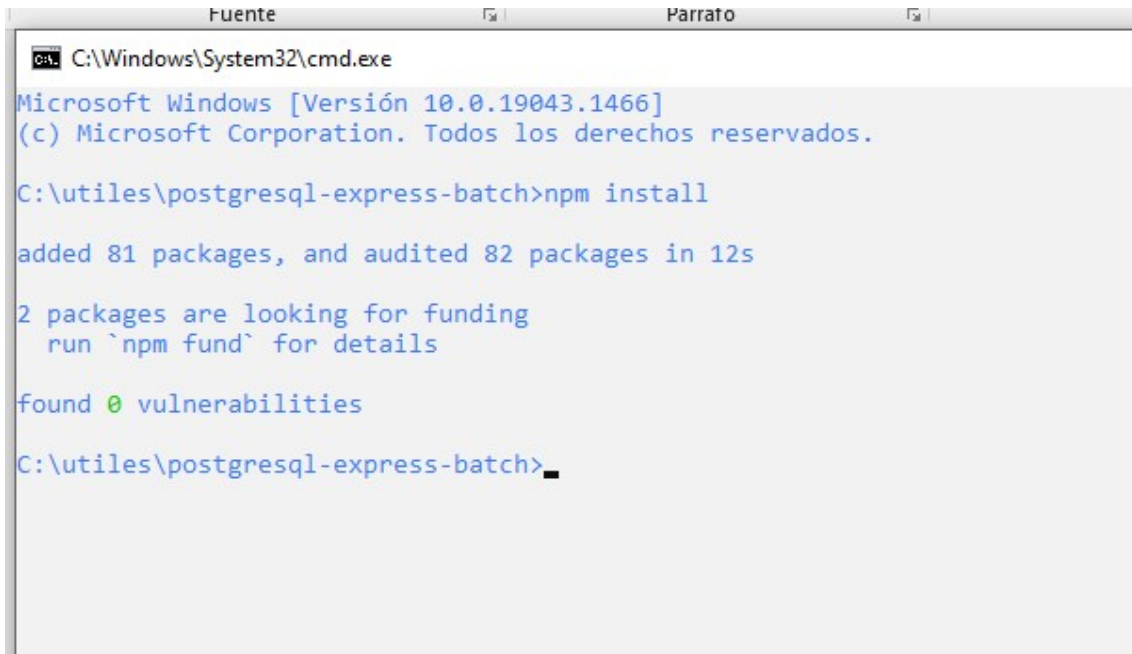
- Cambiar configuración en postgresql-express-batch/app/config
- Archivo config/config.json
- dbUser: "postgres"
- dbPassword: "sistemas" cambien la clave por la que el alumno haya definido

select the repo

```
cd postgresql-express-batch
```

install the repo with npm

```
npm install
```



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.19043.1466]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\utils\postgresql-express-batch>npm install

added 81 packages, and audited 82 packages in 12s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\utils\postgresql-express-batch>
```

create database and import JSON data

```
npm run create
```

```

C:\utils\postgresql-express-batch>npm run create

> postgresql-express-batch@1.0.0 create
> npm run database && npm run import

> postgresql-express-batch@1.0.0 database
> npm run create-database && npm run create-domains && npm run create-tables

> postgresql-express-batch@1.0.0 create-database
> node app/scripts/create/database

- Database Creation Started -> DatabaseName - { mock }
- PostgreSQL Server Connection -> { pg://postgres:*****@localhost:5432 }
- Connection Started
- DROP DATABASE : failed -> { error: no existe la base de datos «mock» }

```

```

C:\Windows\System32\cmd.exe

- Execute Promise 22 : Insert -> { Avengers: Endgame }
- Execute Promise 19 : Insert -> { Avengers: Infinity War }
- Execute Promise 20 : Insert -> { Ant-Man and the Wasp }
- Execute Promise 17 : Insert -> { Thor: Ragnarok }
- Execute Promise 23 : Insert -> { Spider-Man: Far From Home }
- Import finished -> { Movies }

> postgresql-express-batch@1.0.0 import-persons
> node app/scripts/import/execute --endpoint=persons

- Reading JSON Config File -> { ./data/config/persons.json }
- Reading JSON Data File -> { ./data/import/persons.json }
- Import started -> { Persons }
- Add Promise 1 : Insert => { Robert Downey Jr }
- Add Promise 2 : Insert => { Jeremy Renner }
- Add Promise 3 : Insert => { Tom Cruise }
- Add Promise 4 : Insert => { Henri Cavill }
- Add Promise 5 : Insert => { Rebecca Ferguson }
- Add Promise 6 : Insert => { Richard Kiel }
- PostgreSQL Database Connection -> { pg://postgres:*****@localhost:5432/mock }
- Connection Started
- Execute Promise 2 : Insert -> { Jeremy Renner }
- Execute Promise 6 : Insert -> { Richard Kiel }
- Execute Promise 5 : Insert -> { Rebecca Ferguson }
- Execute Promise 1 : Insert -> { Robert Downey Jr }
- Execute Promise 4 : Insert -> { Henri Cavill }
- Execute Promise 3 : Insert -> { Tom Cruise }
- Import finished -> { Persons }

C:\utils\postgresql-express-batch>

```

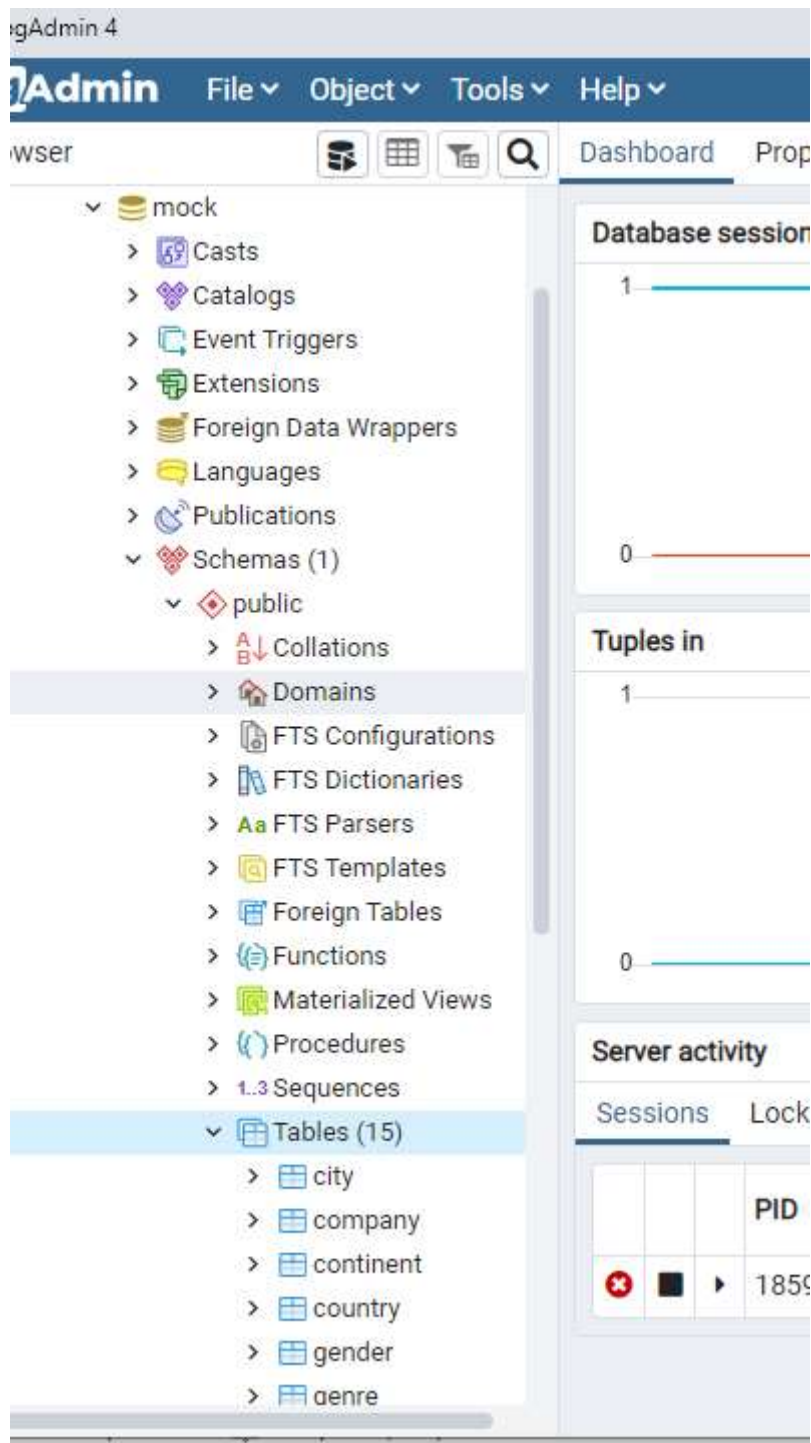
For the fun delete All data/export Files

Export JSON data in data/export

npm run export

Check the new files (for example movies.json)

se verifica que la operación ha sido exitosa..



Implementando CRUD REST API con Node.js & Express & PostgreSQL

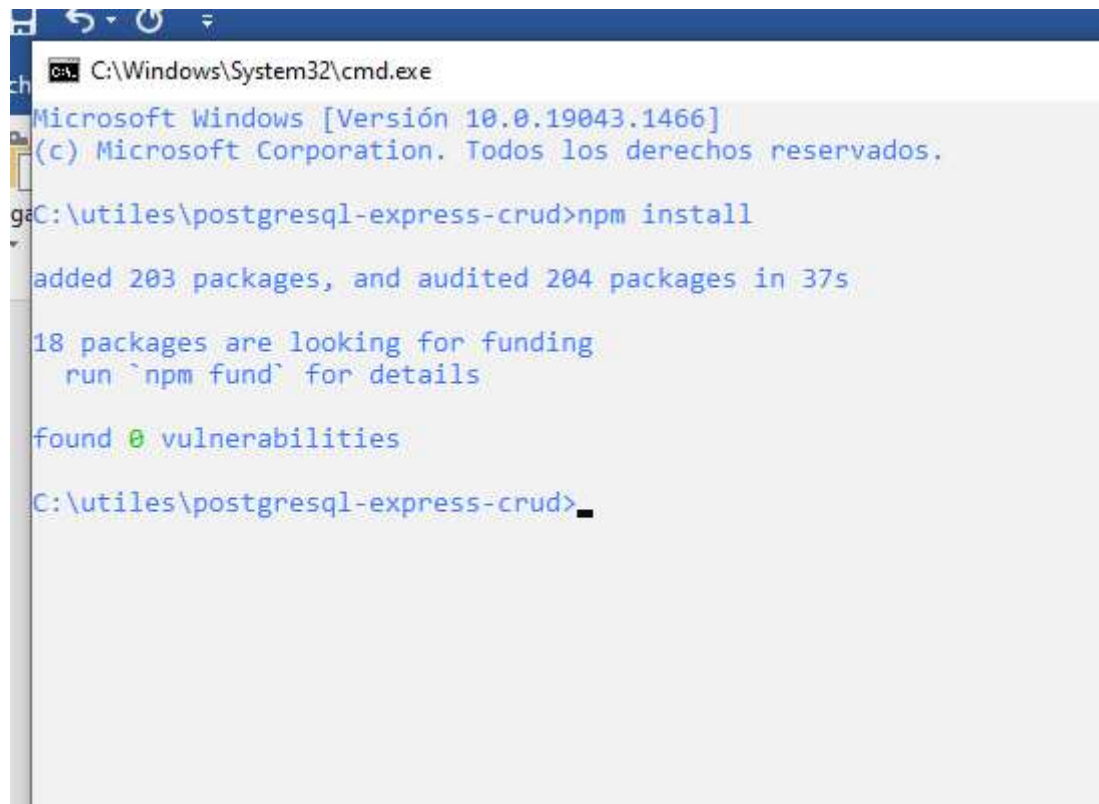
PostgreSQL Password

- Cambiar la configuración en postgresql-express-crud/app/config
- Archivo config/config.json
- dbUser: "postgres"

- dbPassword: "sistemas" ! cambiar el password por defecto

```
# select the repo  
cd postgresql-express-crud
```

```
# install the repo with npm  
npm install
```



```
C:\Windows\System32\cmd.exe  
Microsoft Windows [Versión 10.0.19043.1466]  
(c) Microsoft Corporation. Todos los derechos reservados.  
C:\utils\postgresql-express-crud>npm install  
  
added 203 packages, and audited 204 packages in 37s  
  
18 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities  
  
C:\utils\postgresql-express-crud>
```

```
# Serve CRUD REST API : development mode with nodemon  
npm run dev
```

```
# Serve CRUD REST API : local mode  
npm run start
```



```
C:\utils\postgresql-express-crud>npm run start  
  
> postgresql-express-crud@1.0.0 start  
> node server  
  
- PostgreSQL - Express - API RestFul CRUD  
- Listening on port 5004 !  
- PostgreSQL Database Connection -> { pg://postgres:*****@localhost:5004/mock }  
- Connection Started
```

```
# Serve CRUD REST API : production mode  
npm run prod
```

Serve CRUD REST API : production mode with pm2 (process manager)

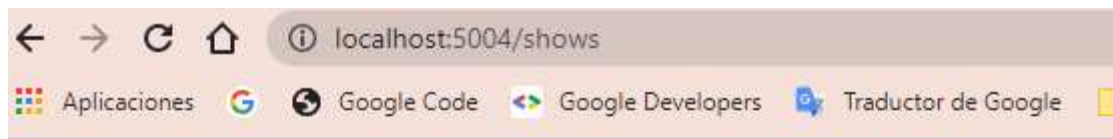
pm2 start process.config.js --env prod

Probar API & PostgreSQL

- En su navegador <http://localhost:5004/movies>



- en su navegador <http://localhost:5004/shows>



```
- {
  id: 1023,
  name: "Game of Thrones",
  releaseDate: "2011-04-17",
  frName: null,
  frWikipediaLink: null,
  wikipediaLink: "Game_of_Thrones",
  image: "http://localhost:5004/shows/img/Game of Thrones.jpg",
  links: [
    - {
      rel: "self",
      href: "http://localhost:5004/shows/1023"
    }
  ]
}
```

- en su navegador <http://localhost:5004/continents>
- en su navegador <http://localhost:5004/countries>
- en su navegador <http://localhost:5004/cities>

Con estas modificaciones, debe regresarse a environment.ts y cambiar los datos de la siguiente manera:

```

15 urlMovies: './assets/params/json/mock/movies.json',
16
17 /* urlNews: 'http://localhost:5004/trailers', */
18 // url: 'https://api.ganatan.com/tutorials',
19
20 config: {
21     /* SELECT ONE OF THOSE CONFIGURATIONS */
22
23     /* LOCAL JSON (NO CRUD) */
24     //api: false,
25     // url: './assets/params/json/crud/',
26
27     /* LOCAL REST API CRUD WITH POSTGRESQL */
28     api: true,
29     url: 'http://localhost:5004/',
30 },
31 };
32
33 /*
34  * For easier debugging in development mode, you can import
35  * to ignore zone related error stack frames such as `zone.js` or
36  *
37  * This import should be commented out in production mode

```

de esa forma, ya accedera, via API, a la base de datos.