NodeJS Express

Fuente: https://www.youtube.com/watch?v=794Q71KVw1k&t=951s

1 – Qué es Express

Express es un framework de NodeJS orientado al desarrollo de aplicaciones en el servidor, es decir facilita el proceso de recibir peticiones del navegador, conectarse a bases de datos, responder con documentos HTML y CSS, etc.

En si no es más que un módulo de NPM que trabaja en conjunto de forma estandarizada con otros módulos para la creación de páginas web. Por ejemplo para el manejo de imágenes, conexión a base de datos y otras funciones se apoya en módulos de terceros, la función principal de Express es crear el servidor y gestionar peticiones.

La principal ventaja de usar este framework es que permite compartir el mismo lenguaje de programación tanto en el frontend como en el backend. Esta facilidad genera conceptos como los stack MEAN (MongoDB, Express, Angular, NodeJS) y MERN (…, …, React, ….) que integran todo el desarrollo de una página web con javascript.

Otra característica es que simplifica mucho la utilización de tecnologías de comunicación en tiempo real como WebSockets y WEBRTC. También permite implementar la arquitectura MVC.

2 – Primer servidor con Express

Para crear un servidor básico con Express hay que seguir una serie de pasos. El primero, si bien no es obligatorio es una buena práctica muy usual, y consiste en crear el **package.json** por medio del comando

```
npm init
```

A continuación instalamos el módulo de Express y lo guardamos en el archivo package.json, usando el comando

```
npm i express --save
```

Vemos como aparece en la sección de dependencias

```
"dependencies": {
  "express": "^4.17.1"
}
```

Ahora escribimos el código básico de un servidor con express

```
const express = require('express');

const app = express();

app.get('/', (req, res)=>{
    res.send('Hola mundo');
})

app.listen(3000, ()=>{
    console.log('server on port 3000');
});
```

Primero se importa el módulo con la función **require**. Luego del módulo express se llama al método **express()**, que crea el servidor y se lo guarda en la constante **app**. A diferencia de crear un servidor tradicional con nodejs, express no necesita un función que devuelva algo a la ruta por defecto del servidor (‘/’), pero al ejecutarlo y acceder al puerto donde está escuchando va a mostrar un error, para eso implementamos una función **get()**, que gestiona una petición http y envía una respuesta. Por último activamos el servidor con la función **listen()** que recibe como parámetros el numero de puerto donde va a escuchar y una función de callback.

Nodemon

Nodemon es un módulo que agiliza el proceso de desarrollo. Al desarrollar una aplicación de servidor es necesario cancelarlo y volverlo a ejecutar cada vez que se realiza un cambio, lo cual se vuelve muy repetitivo. Para solucionar esto se puede intalar el módulo nodemon.

```
npm i nodemon -D
```

-D es una forma abreviada de escribir **–save -dev**. Esto significa que es una dependencia de desarrollador, no de la aplicación. En el package.json se va a registrar en la sección **devDependecies**.

Para ejecutar el archivo con nodemon se usa el comando

```
nodemon index.js
```

3 – Routing

El routing o enrutamiento consiste en definir funciones mediante peticiones http para cada url que puede solicitar el cliente y debe ser gestionada por el navegador. En el caso inicial vemos que la siguiente función responde a la ruta **localhost:3000/**

```
app.get('/', (req, res)=>{
    res.send('Hola mundo');
})
```

Implementemos otra a modo de ejemplo. Suponiendo que queremos disponer de otra ruta para mostrar información de la aplicación sería **localhost:3000/informacion**. En el servidor la función sería.

```
app.get('/informacion', (req, res)=>{
    res.send('<h3> Esta es una página hecha con express </h3>')
});
```

3.1 – Tipos de rutas

Todas las peticiones usadas hasta ahora son de tipo GET, pero http soporta muchos tipos más de peticiones según la acción que desee realizar el navegador. Las mas comunes son

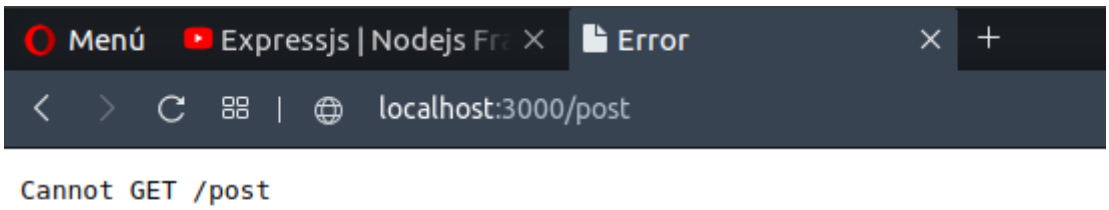
- GET. Usada cuando se solicita información.
- POST. Cuando se envía información al servidor.
- PUT. Usada para actualizar algún dato en la página o una base de datos.
- DELETE. Usada para eliminar información.

Express tiene funciones para menejar cada tipo de petición al igual que get().

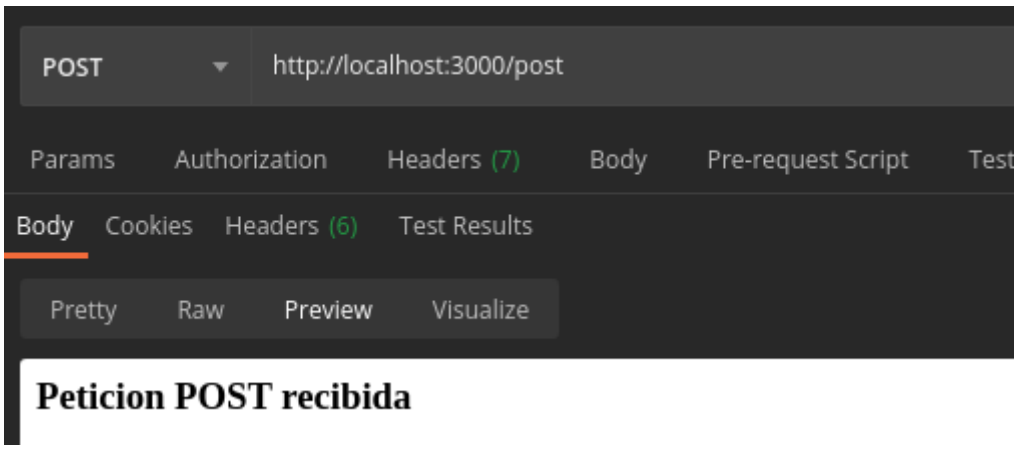
```
app.post('/post', (req, res)=>{
    res.send('<h3> Peticion POST recibida </h3>');
});

app.put('/put', (req, res)=>{
    res.send('<h3> Peticion PUT recibida </h3>');
});

app.delete('/delete', (req, res)=>{
    res.send('<h3> Peticion DELETE recibida </h3>');
});
```

3.2 - Postman

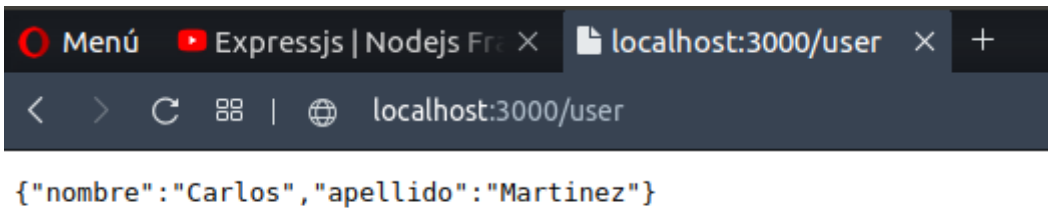
El navegador por defecto realiza peticiones get, así que si intentamos acceder a estas rutas directamente con la URL no va a funcionar, deberíamos hacerlo desde un formulario html o mediante un archivo js.



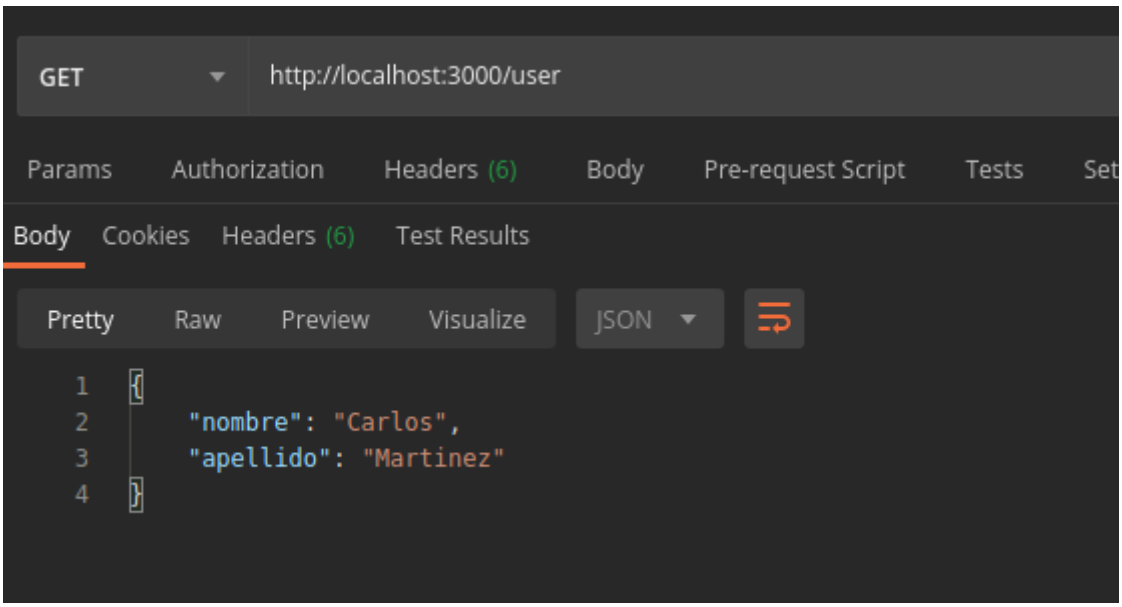
Para solucionar este inconveniente se pueden probar los distintos tipos de peticiones usando una aplicación llamada **POSTMAN**.



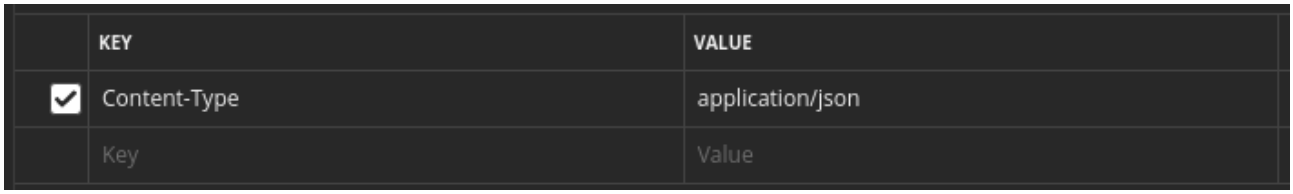
Desde postman también se puede recibir otro tipo de datos, no solo texto plano y html. Por ejemplo si queremos retornar un objeto json, desde el servidor sería usando el método **json()** del objeto res. Si se accede a la peticion desde el navegador muestra el json en forma de texto plano.



Y desde postman lo muestra ya formateado



Otra cosa que se puede hacer con postman es enviar información al servidor usando el método **POST.** Para hacer esto vamos a la pestaña **headers** y agregamos el siguiente par clave/valor.

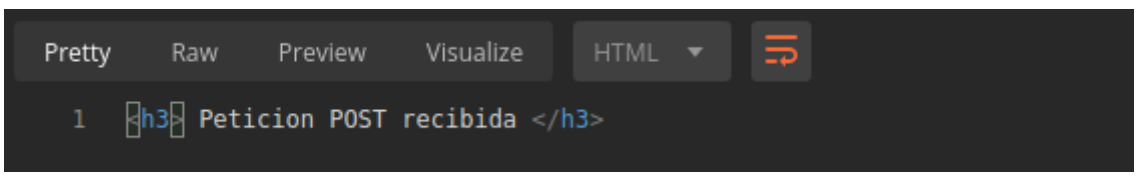


Y en **body** → **raw** especificamos el json que queremos enviar al servidor.

Para recibirlo desde el servidor se utiliza un objeto de express que permite tomar información del request (**req.body**). Por último, para que express pueda entender el formato json es necesario usar un middleware (mas adelante los vemos en detalle), estos se declaran generalmente antes de las rutas.

```
app.use(express.json());

app.post('/user', (req, res)=>{
    console.log(req.body);
    res.send('<h3> Peticion POST recibida </h3>');
});
```

Entonces, al hacer ahora la peticion POST se va a ver en postman el mensaje



y en la consola del servidor

```
server on port 3000
{ nombre: 'Carlos', apellido: 'Martinez' }
```

Estos datos recibidos pueden no solo mostrarse por consola, pueden extraerse, cambiarse, guardarse en un BD, etc.

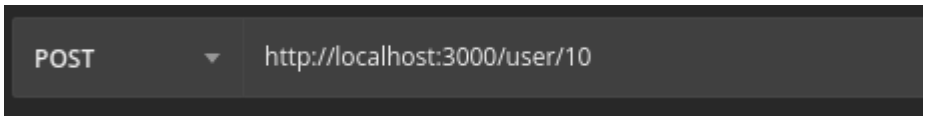
3.3 – Parámetros

Utilizando las rutas también pueden enviarse datos al servidor mediante parámetros. Para esto se agrega a la ruta ***/*:<param>**, y al realizar la peticion ahora se pueden agregar valores. Veamos un ejemplo, agregando un id a la petición POST /user

En el servidor el código es

```
app.post('/user/:id', (req, res)=>{
    console.log(req.body);
    console.log(req.params);
    res.send('<h3> Peticion POST recibida </h3>');
});
```

Al hacer una peticion POST a la ruta



En la consola del servidor ahora también se recibió el valor 10.

```
server on port 3000
{ nombre: 'Carlos', apellido: 'Martinez' }
{ id: '10' }
```

Para finalizar, hay otro tipo de ruta que no corresponde a una petición http, sino que actúa como intermediario ante todos los tipos de peticiones. Para hacer esto se debe llamar al método **all()** de express. Este método puede usarse de igual manera que todos los anteriores, respondiendo las peticiones del navegador, pero su función principal es hacer “algo” antes de responder con el método y ruta solicitados.

```
app.all('/user', (req, res)=>{
    res.send('respuesta con método ALL');
});

app.get('/user', (req, res)=>{
    res.send('respuesta con método GET');
});
```

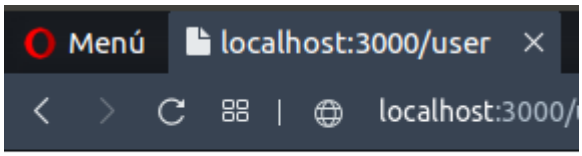
Como vemos, en este caso la respuesta enviada al navegador es la implementada en all

respuesta con método ALL

Si all() no tiene implementado un **res.send()** el navegador se queda esperando una respuesta y no se pasa al método get. Para solucionarlo es necesario un tercer parámetro llamado next, que permite pasar el control al método **get()**.

```
app.all('/user', (req, res, next)=>{
    //res.send('respuesta con método ALL');
    console.log('respuesta ALL');
    next();

});

app.get('/user', (req, res)=>{
    res.send('respuesta con método GET');
});
```

En
este caso sí se envía una respuesta al navegador desde el método **get()**.



respuesta con método GET

En la documentación oficial de Express se encuentran todos los métodos con los que se puede manejar una petición: https://expressjs.com/es/4x/api.html#app

4 – Middlewares

Un middleware es un manejador de peticiones que puede ejecutarse antes que esta llegue a su ruta original. En general son funciones que requieren tres parámetros **(req, res, next)**, los dos primeros para manejar la información que se comunica con el cliente, y el tercero para indicar cuándo se pasa el control a la siguiente función, ya que los métodos que manejan peticiones http pueden recibir como función de callback

- Un middleware.
- Un conjunto de middlewares separados por comas.
- Un array de middlewares.
- Cualquier combinación de los anteriores

Una forma mas común de usarlos es mediante el método **use()** que provee express. Este hace que un middleware se ejecute antes de cualquier peticion, sin necesidad de especificar la ruta como en el método **all()**. Dentro de use se puede usar de dos maneras, declarando la función y pasándosela como parámetro, o declarando una función anónima en el mismo método.

El ejemplo mas común de middleware es una funcion logger, que registra las peticiones recibidas.

```
function logger(req, res, next){
    let fecha = new Date();
    fs.appendFile('./log', `\n${fecha} - ${fecha.getTime()} - peticion recibida:`, (err)=>{
        if (err) console.log('error al escribir archivo');
    });
    fs.appendFile('./log', ` ${req.protocol}://${req.get('host')}${req.originalUrl}`, (err)=>{
        if (err) console.log('error al escribir archivo');
    });
    next();
}

app.use(logger);

app.get('/user/:nombre', (req, res)=>{
    res.send(`Bienvenido usuario ${req.params.nombre}`);
});
```

Esto genera un log con la siguiente entrada

```
Thu Dec 17 2020 08:25:03 GMT-0300 (-03) - 1608204303011 - peticion recibida: http://localhost:3000/user/Carlos
```

4.1 – Morgan

Un ejemplo de middleware que facilita los logs por consola es Morgan:

https://www.npmjs.com/package/morgan

Al ser un modulo de terceros es necesario instalarlo con

```
npm i morgan - - save
```

y luego es requerido desde el código

```
const morgan = require('morgan');
```

Este posee varios formatos predefinidos, y también da la opción de crear formatos propios. Los formatos mas comunes son:

- Combined.
- Common.
- Dev.
- Short.
- Tiny.

Por ejemplo, el formato dev imprime por consola el tipo de peticion, la url, el codigo de servidor (en ver si la peticion se procesó exitosamente, amarillo para errores de cliente y rojo para errores de servidor), el tiempo de respuesta y el peso en bytes de la misma.

```
app.use(morgan('dev'));
```

```
GET / 404 0.314 ms - 139
GET /user 404 0.338 ms - 143
GET /user/Carlos 304 3.735 ms - -
```

4.2 – Archivos estáticos

Otro Middleware muy usual que ya viene incluido con Express es **static()**. Este se declara luego de todas las rutas ya que es un middleware que se ejecuta luego de devolver información al navegador. El método static recibe como parámetro un string con el nombre de una carpeta, convencionalmente llamada **public** donde se almacenan todos los archivos html, css y javascript que deben ser procesados de forma tradicional por el navegador. En esta carpeta se incluye un archivo **index.html** que se carga en el navegador luego de retornar la petición.

https://expressjs.com/es/starter/static-files.html

Ejemplo

Creamos una página html con un mensaje, fondo gris y tipo de letra blanca, y que pasado 3 segundo cambie el mensaje que se muestra en pantalla.

Para esto dentro de la capeta public es necesario crear los siguientes archivos

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Curso Express</title>
    <link rel="stylesheet" href="./estilos.css">
</head>
<body>
    <h1 id="titulo">Aplicación Express</h1>
    <script src="./app.js"></script>
</body>
</html>
```

Estilos.css

```
body{
    background: gray;
    color: white;
}
```

app.js

```
setTimeout(()=>{
    document.getElementById('titulo')
        .innerHTML = 'Javascript y Express';
}, 3000);
```

Si se invoca una ruta no especificada en el servidor, este carga por defecto el archivo index.html indicado en la carpeta public, ya que esta es invocada en el middleware static. También pueden declararse varios middlewares con referencia a distintas carpetas, y estos son enviados al navegador en el orden que fueron escritos.

Este middleware también permite acceder a los archivos de la carpeta public desde el navegador. Si se escribe como URL la dirección de host, seguido del path del archivo el navegador carga el código del mismo.


```
setTimeout(()=>{
    document.getElementById('titulo')
        .innerHTML = 'Javascript y Express';
}, 3000);
```

Ej.

5 – Configuraciones de Express

Cuando creamos un servidor es necesario configurar muchas cosas, como el puerto, el motor de plantillas, el nombre de la aplicación, etc. Para esto se usa un método de express llamado **set()**. Este se declara antes de los middlewares, para este punto es necesario comenzar a ordenar el código del servidor con comentarios

```
const express = require('express');

const app = express();

//Settings

//Middelwares

//Routes

app.listen(3000, ()=>{
    console.log('server on port 3000');
});
```

Las **configuraciones** de express pueden ser vistas como declaración de variables. La función set() recibe dos parámetros de tipo string, uno es el nombre de la variable y otro el valor de la misma, y luego se puede acceder a las mismas con una función **get()**.

Por ejemplo, se puede poner un nombre a la aplicación y luego informarlo al iniciar el servidor. Otro tipo de configuraciones son variables predefinidas o de uso convencional, como el número de puerto

```
//Settings
app.set('appName', 'Tutorial Express');
app.set('port', 3000)

//Middelwares

//Routes

app.listen(app.get('port'), ()=>{
    console.log(app.get('appName'));
    console.log(`server on port ${app.get('port')}`);
});
```

5.1 – Motores de plantillas

Una característica muy importante de las configuraciones de Express es que permite asignar un motor de plantillas. Un motor de plantillas es un módulo que permite crear archivos que extienden la funcionalidad de HTML. Algunos motores de plantillas muy usados son:

- Ejs. (embedded javascript template)
- Handlebars.
- Pug

EJS

Vamos a usar el motor ejs. Este es un módulo tan integrado con Express que de hecho no hace falta importarlo, alcanza con instalarlo

```
npm i ejs - - save
```

y asignarlo en la sección de configuraciones.

```
//Settings
app.set('appName', 'Tutorial Express');
app.set('port', 3000);
app.set('view engine', 'ejs');
```

Para utilizar **vistas** mediante el motor de plantillas en Express hay que crear dentro del proyecto una nueva carpeta llamada views donde vamos a crear todos los archivos ejs, y en el callback de las rutas ya no usamos el método res.send() sino **res.render('<archivo.ejs>')**.

*La carpeta que contiene las vistas se llama views por defecto, por lo tanto al renderizarlas no hace falta especificarla en el path, con el nombre del archivo alcanza.

Ruta inicial

```
//Routes
app.get('/', (req, res)=>{
    res.render('index.ejs');
});
```

Ejs permite embeber en html código javascript. Por lo tanto se puede enviar para renderizar datos junto con la vista y estos son manipulados dentro de la misma. Supongamos que hicimos una consulta a una base de datos y se devuelve un json con un conjunto de usuarios, entonces en la ruta **/users** se responde de la siguiente manera.

```
app.get('/users', (req, res)=>{
    const users = [{name: 'Carlos'}, {name:'Sebastian'}, {name: 'martinez'}];
    res.render('users.ejs', {users: users});
});
```

Y en users.ejs se utiliza esta información con la siguiente sintaxis

```
<body>
    <h3>Usuarios</h3>
    <ul>
        <% users.forEach(function(user){ %>
            <li> <%= user.name %></li>
        <% }) %>
    </ul>
</body>
```

HANDLEBARS

Si queremos hacer lo mismo pero en con handlebars el proceso es muy similar, solo cambia la sintaxis.

Primero ordenamos un poco el array de usuarios.

```
//Users
let users = [
    {
        id: 1,
        name: 'Carlos'
    },
    {
        id: 2,
        name:'Sebastian'
    },
    {
        id: 3,
        name: 'martinez'
    }
    ];
```

Instalamos y asignamos Handlebars como motor de plantillas

```
npm i hbs –save
```

```
app.set('view engine', 'handlebars');
```

Y creamos en la carpeta views un documento **users.hbs**. Para este ejemplo agregamos el cdn de bootstrap para poder usar los estilos y que la página se vea mejor.

Para recibir parámetros desde las rutas del servidor en handlebars es necesario declararlos entre dos pares de llaves. Así, si queremos recibir un título del servidor, la etiqueta **<title> </title>** encierra la variable **{{titulo}}**.

Y para iterar sobre elementos de un arreglo el bloque de código se escribe

{{#each array}}

//codigo html

{{/each}}

En este caso vamos a mostrar los usuarios en una tabla con estilos de bootstrap

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <title>{{titulo}}</title>
</head>
<body>
    <div class="container">
        <h1>{{mensaje}}</h1>

        <table class="table table-striped">
            {{#each users}}
                <tr>
                    <td>{{this.id}}</td>
                    <td>{{this.name}}</td>
                </tr>
            {{/each}}
        </table>

    </div>
</body>
</html>
```

En el navegador vemos el seguiente resultado

usuarios

1	Carlos
2	Sebastian
3	Martinez

PUG

Por último, el motor de plantillas pug posee una sintaxis mucho mas sencilla y acotada, pero es el mas distinto a html tradicional. El codigo en el servidor es practicamente el mismo, agregando las siguientes diferencias.

```
Npm i pug –save
```

```
app.set('view engine', 'pug');
```

La plantilla **users.pug** es

```
html
head
    link(rel="stylesheet", href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css")
    title=titulo

body
    div.container
        h1=mensaje

        table.table.table-striped
            for user in users
                tr
                    td= user.id
                    td= user.name
```

y el navegador muestra el siguiente resultado