TALLER 04 – EXPRESS JS

Express js es un framework para desarrollar aplicaciones web basadas en nodejs, express es un paquete con módulos de nodejs que se administra sus dependencias a través de npm. Este framework provee funcionalidades preconstruidas para construir aplicaciones de forma mas rápida.

1. Requerimientos

A continuación encuentras los requerimientos para ejecutar este taller tutorial sobre express js

- Conocimientos de programación
- Conocimientos de javascript
- Conocimientos de html
- Conocimientos de css
- Conocimientos de node.js
- Conocimientos de npm
- Editor de código o IDE
- node.js instalado

2. Instalación

La instalación de express se hace a dentro de un proyecto de node.js, para lo cual es necesario primero crear un proyecto con node

- PASO 1: Crear una carpeta para el proyecto y crear un archivo llamado app.js
- PASO 2: Inicializar el proyecto de node.js para esto usar el comando "***npm init***"
    - Llenar la información solicitada
    - Para el *entry point* puede cambiarlo por app.js
- PASO 3: Para instalar ahora el framework haz uso de npm
    - Ejecuta el comando ***"npm install express –save"***
- PASO 4: Ya se adiciono express al proyecto, esto lo podemos verificar en el archivo package.json

3. Crear un servidor HTTP

- PASO 1: En el archivo app.js
- PASO 2: Importar el módulo de express en el archivo

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

- PASO 3: Inicializar el servidor con express

```
const appServer = express();
```

- PASO 4: Poner el servido a escuchar las peticiones en un puerto

```
appServer.listen (3000, ()=>{
    console.log('SERVER IS RUNNING ON PORT 3000');
});
```

- PASO 5: Crear una respuesta a una petición get para que responda con un mensaje, se puede evidenciar que se usa una función callback para que responda con el mensaje.

```
appServer.get ('/',
    (req, res) => {
        res.send ('HELLO WORLD WITH EXPRESS!!!');
    }
);
```

- PASO 6: Ejecutar el proyecto, abrir el navegador y acceder a localhost:3000 y ver el resultado

4. Enrutamiento

A continuación, se va a configurar las rutas para responder a las diferentes peticiones que puedan tenerse desde el cliente.

- PASO 1: Crear una nueva ruta que responde cuando se haga aun apetición GET responda con mi información básica cuando en el navegador se ingrese a la ruta *mybasicinfo* (para este caso vamos a retornar un texto).

```
appServer.get ('/mybasicinfo',
    (req, res) => {
        res.send ('THIS IS MY BASIC INFORMATION - My Name Is Carlos Iv
an!!!');
    }
);
```

- PASO 2: Crear una nueva ruta que responde cuando se haga aun apetición GET responda con mi información de experiencia laboral cuando en el navegador se ingrese a la ruta *myexperince* (para este caso vamos a retornar un texto).

```
appServer.get ('/myexperience',
    (req, res) => {
        res.send ('THIS IS MY EXPERIENCE');
    }
);
```

- PASO 3: Correr el proyecto, ingresar al navegador y acceder a cada una de las rutas para verificar su funcionamiento

5. Uso de métodos HTTP con el enrutamiento

Cuando desarrollamos proyectos web hacemos uso del protocolo HTTP, esto permite hacer peticiones de clientes a servidores, estas peticiones se hacen con los diversos métodos que provee el protocolo, para este caso vamos a usar los métodos POST, GET, PUT, DELETE.

- PASO 1: Adicionar una ruta que maneje peticiones a través del método GET que de una respuesta con un texto

```
appServer.get ('/getrequest',
    (req, res) => {
        res.send ('THIS IS A GET REQUEST');
    }
);
```

- PASO 2: Adicionar una ruta que maneje peticiones a través del método POST que de una respuesta con un texto

```
appServer.post ('/postrequest',
    (req, res) => {
        res.send ('THIS IS A POST REQUEST');
    }
);
```

- PASO 3: Adicionar una ruta que maneje peticiones a través del método DELETE que de una respuesta con un texto

```
appServer.delete ('/deleterequest',
    (req, res) => {
        res.send ('THIS IS A DELETE REQUEST');
    }
);
```

- PASO 4: Adicionar una ruta que maneje peticiones a través del método PUT que de una respuesta con un texto

```
appServer.put ('/putrequest',
    (req, res) => {
        res.send ('THIS IS A PUT REQUEST');
    }
);
```

- PASO 5: Ejecuta el proyecto, el servidor debe haberse iniciado y ahora en el navegador vamos a probar las rutas que acabamos de crear. En la siguiente figura (Figura 1) vamos a ver el retorno desde el navegador cuando hacemos una petición con el método GET.

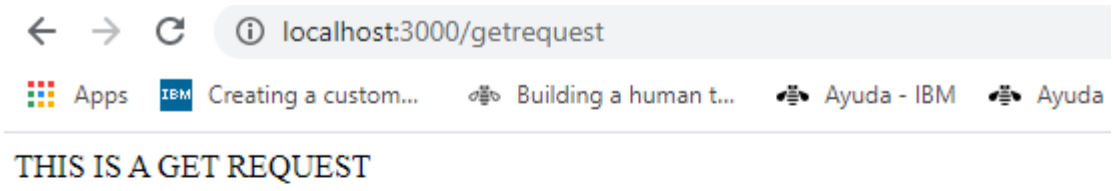

Figura 1 - Prueba y respuesta de la peticion con el metodo GET realizada

- PASO 6: Ahora se deben probar las peticiones con otros métodos, cuando lo hacemos a través del navegador vamos a encontrar con que el navegador nos muestra una respuesta diciendo que no puede obtener la ruta especificada (Figura 2). Esto sucede por que el navegador por defecto hace peticiones de tipo GET, y como en el código determinamos que las otras rutas recibían peticiones por métodos diferentes a este, entonces la ruta no puede se accedida directamente con el navegador.

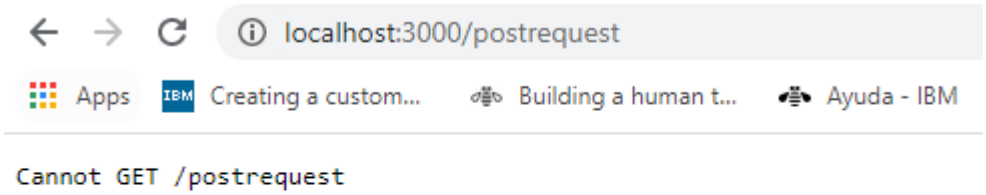

Figura 2 - Petición fallida a una ruta que recibe peticiones a través de post

Para poder probar estas rutas configurando su método, se debe usar alguna aplicación que nos permita hacer esto, para eso podemos usar un software conocido como *postman* o usar Talend API Tester, también existen muchas otras herramientas que puedes usar para probar los servicios que generes. En estas aplicaciones puedes elegir el método que vas a usar para hacer las peticiones y adicionar parámetros y otras configuraciones adicionales.

En la Figura 3 se muestra como a través del software Talend Api tester se realiza la petición POST a nuestra ruta y en la Figura 4 se puede ver el resultado en el mismo software.

Figura 3 - Ejemplo de petcicion post a traves del software TalendAPI Tester

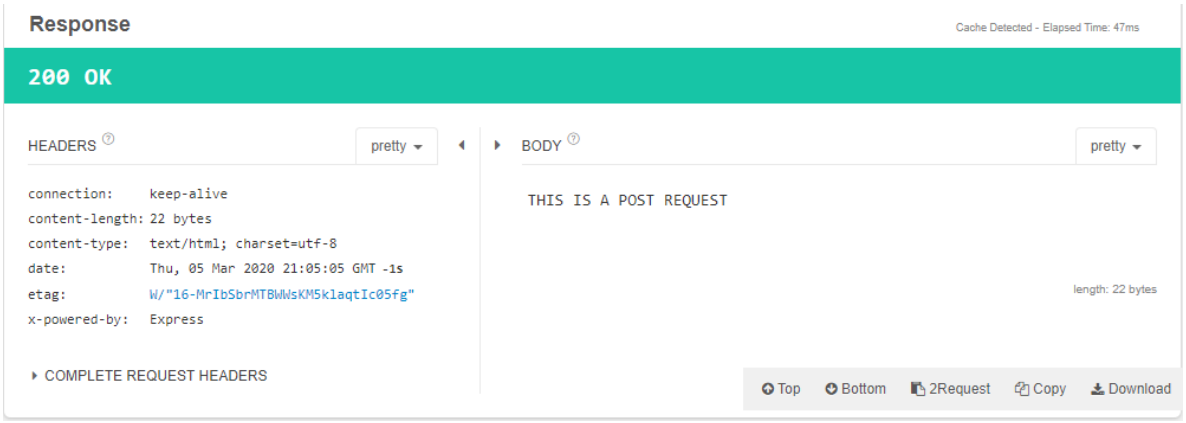

Figura 4 - Ejemplo de respuesta de la peticion post, visualizada a traves del software API Talend Tester

- PASO 7: Probar con el software elegido todas las rutas creadas previamente y revisar su respuesta.

6. Envío de parámetros y respuesta de objetos a través de los servicios

En las secciones anteriores lo que hemos creado con cada una de las rutas son servicios web que responden a través del protocolo HTTP, estos servicios también tienen la posibilidad de recibir parámetros e información para que esta sea procesada según la funcionalidad. Para esto vamos a hacer el registro de un usuario y la actualización de la información de este.

- PASO 1: vamos a crear un archivo user.js en el cual vamos a crear un objeto JSON con algunos atributos (nombre, apellido, edad, carrera, etc…) y valores por defecto, este objeto lo exportamos como modulo.

```
const user = {
    nombre : "",
    apellido: "",
    edad: "",
    carrera: ""
}

module.exports.user = user;
```

- PASO 2: En el archivo app.js importamos el módulo user que acabamos de crear y lo almacenamos en una variable.

```
const myUser = require ('./models/user');
```

- PASO 3: Creamos un servicio a la ruta user que retorne con el response el objeto json user. Para retornar un json usamos la función res.json.

```
appServer.get ('/user', (req, res)=>{

    res.json (myUser);

});
```

- PASO 4: Subimos el servidor y probamos desde el navegador o desde el software que usemos para probar servicios.

- PASO 5: Creamos un nuevo servicio con el método post y dentro de esto vamos a recibir en el body un json con la información de un usuario

```
appServer.post ('/adduser' , (req, res)=>{
    console.log(req.body);
    res.send ('POST USER ADDED');

});
```

- PASO 6: Adicionamos un middleware para que maneje la petición JSON

```
//Middleware, este debe estar antes de todas las rutas
appServer.use(express.json());
```

- PASO 7: Ahora desde el software para probar servicios probamos nuestro servicio post, para esto vamos a adicionar los siguiente a la petición y ejecutar
    - En el header el tipo de contenido que se envía
        - Content-Type: application/json
    - En el body un objeto JSON

```
{
"nombre": "Carlos Ivan",
"apellido": "Rivera P",
"edad": "29",
"carrera": "Ing Sistemas"
}
```

- PASO 8: Ya se ha recibido anteriormente el contenido a través del body de la petición, pero ese contenido puede estar asociado a un parámetro, para esto vamos a crear un

nuevo servicio en el cual vamos a recibir un parámetro idUser y este lo vamos a imprimir cuando se reciba en el request

```
appServer.post ('/updateuser/:idUser' , (req, res)=>{
    console.log(req.body);
    console.log ( req.params.idUser);
    res.send ('USER UPDATED');

});
```

- PASO 9: Hace runa petición a este servicio creado desde el software para probar servicios web y enviar el parámetro, este parámetro se envia en la ruta separado con un carácter /

7. EJERCICIO

Haciendo uso de node haga una aplicación que tenga los siguientes servicios :

- Crear un nuevo usuario
- Eliminar un usuario por id enviado como parámetro
- Mostrar todos los usuarios
- Traer un usuario por id enviado como parámetro
- Traer un usuario por nombre enviado como parámetro
- Traer todos los usuarios menores a una edad enviada como parámetro