API REST

Mi primera API REST

{desafío} latam_

Describir las características principales de una arquitectura REST distinguiendo buenas prácticas para el diseño de una API REST para la interoperación de sistemas.

Implementar un servidor REST utilizando el framework Express para la disponibilización de recursos acorde a las buenas prácticas.

- Unidad 1:
  API REST
- Unidad 2:
  Subida de archivos al servidor
- Unidad 3:
  JWT

{desafío}
latam_

¿Qué aprenderás en esta sesión?

- *Implementa rutas o endpoints para reflejar el estado representacional de un recurso utilizando las buenas prácticas de diseño acorde al framework Express.*

- *Implementa el cuerpo de una respuesta a una petición utilizando las buenas prácticas de diseño acorde al framework express.*

{desafío} latam_

Recordemos:

¿Cuales son los verbos HTTP y qué acciones realiza cada uno?

/* Consideraciones previas */

Consideraciones previas

Una consideración importante que debes tomar al momento de construir una API REST con Node, es el consumo de datos que fueron enviados como payload en una consulta POST o PUT, ya que estos no se manejarían igual que una data enviada en una consulta GET, recordando que esta última por defecto envía la información como parámetros query Strings.

Consideraciones previas

La forma de capturar la data de una consulta con payload es a través de un middleware que parsea la data recibida en el cuerpo de todas las consultas.

En el siguiente código, se muestra un ejemplo de una ruta que recibe en formato JSON un objeto con una propiedad **name**:

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

app.post("/producto", (req,res) => {
    console.log(req.body)
    res.send()
})
```

/* Construcción de mi primera API REST */

{desafío}
latam_

Construcción de mi primera API REST

Ahora que conoces las consideraciones previas, procedemos con la construcción de una API REST, que tendrá como objetivo generar la gestión de los datos de una tienda de bicicletas, cuyas propiedades serán id, marca, modelo y precio. Los ID se generarán con el paquete UUID, por lo que deberás instalarlo con el comando:

npm i uuid

Además, crea con el siguiente código un documento llamado Bicicletas.json que contenga una bicicleta registrada:

```
{
  "bicicletas": [
    {
       "id": "a36658",
       "marca": "oxford",
       "modelo": "nova2",
       "precio": "300000"
    }
  ]
}
```

GET

Ahora sí, iniciemos con la creación de la ruta GET, para esto prosigue con los siguientes pasos:

- **Paso 1**: Crea la ruta GET /bicicletas
- **Paso 2**: Almacena en una variable el JSON del archivo Bicicletas.json.
- **Paso 3**: Devuelve al cliente el JSON de bicicletas.

```
// Paso 1
app.get("/bicicletas", (req, res) => {
    // Paso 2
    const bicicletasJSON = JSON.parse(fs.readFileSync("Bicicletas.json",
"utf8"));
    // Paso 3
    res.send(bicicletasJSON);
})
```

GET

Para probar este servidor y la ruta creada, levanta el servidor y en otra terminal pega el siguiente comando:

```
curl http://localhost:3000/bicicletas
```

Y obtendrás lo que se muestra en la siguiente imagen:

```
$ curl http://localhost:3000/bicicletas
{"bicicletas":[{"id":"a36658","marca":"oxford","modelo":"nova2","precio":"300000"}]}
```

Bien, ahí lo tenemos, ruta GET creada con éxito!

POST

Ahora que podemos obtener las bicicletas registradas con la ruta GET, debemos crear una ruta que permita registrar una nueva bicicleta, para eso usaremos un condicional que evalúe que la consulta se está haciendo con el método POST.

Sigue estos pasos para disponibilizar en nuestro servidor el uso del verbo POST:

- **Paso 1**: Crear la ruta POST /bicicletas
- **Paso 2**: Crea una variable bicicleta con las propiedades recibidas en el cuerpo de la consulta
- **Paso 3**: Almacena en una variable la data del archivo Bicicletas.json y de su arreglo bicicletas.

POST

- **Paso 4:** Agrega la bicicleta creada en el arreglo del JSON.
- **Paso 5:** Sobrescribe el archivo Bicicletas.json con el arreglo modificado.

```
// Paso 1
app.post("/bicicletas", (req, res) => {
    // Paso 2
    const { marca, modelo, precio } = req.body
    const bicicleta = { id: uuidv4().slice(30), marca, modelo, precio ;
    // Paso 3
    const bicicletasJSON = JSON.parse(fs.readFileSync("Bicicletas.json",
"utf8"));
    const bicicletas = bicicletasJSON.bicicletas;
    // Paso 4
    bicicletas.push(bicicleta);
    // Paso 5
    fs.writeFileSync("Bicicletas.json", JSON.stringify(bicicletasJSON));
    res.send("Bicicleta agregada con éxito");
})
```

POST

Ahora consulta el servidor con la siguiente línea de comando desde la terminal:

```
curl -d '{"marca":"BIANCHI","modelo":"MTB PRO 26","precio":"300000"}' -H
"Content-Type: application/json" -X POST
http://localhost:3000/bicicletas
```

POST

Y si revisas el documento **Bicicletas.json** verás lo que se muestra en la siguiente imagen:

PUT

En nuestro caso consideraremos la segunda alternativa, por lo que habrá que iterar el arreglo de bicicletas para modificar la bicicleta que tenga el id recibido. Sigue estos pasos para lograr la construcción de esta ruta en nuestro servidor:

- **Paso 1**: Crear la ruta PUT **/bicicletas**.
- **Paso 2**: Crea una variable bicicleta con las propiedades recibidas en el cuerpo de la consulta.
- **Paso 3**: Almacena la data del archivo ***Bicicletas.json*** en una variable y utiliza el método map en el arreglo de bicicletas para sobrescribir el objeto que tenga el mismo id que los recibidos en la consulta.

PUT

- **Paso 4**: Sobrescribe el archivo ***Bicicletas.json*** y culmina la consulta indicando que la bicicleta fue modificada con éxito.

```
// Paso 1
app.put("/bicicletas", (req, res) => {
    // Paso 2
    const { id, marca, modelo, precio } = req.body;
    const bicicleta = { id, marca, modelo, precio, };
    // Paso 3
    const bicicletasJSON = JSON.parse(fs.readFileSync("Bicicletas.json",
"utf8"));
    const bicicletas = bicicletasJSON.bicicletas;
    bicicletasJSON.bicicletas = bicicletas.map((b) =>
        b.id === id ? bicicleta : b
    );
    // Paso 4
    fs.writeFileSync("Bicicletas.json", JSON.stringify(bicicletasJSON));
    res.send("Bicicleta modificada con éxito");
});
```

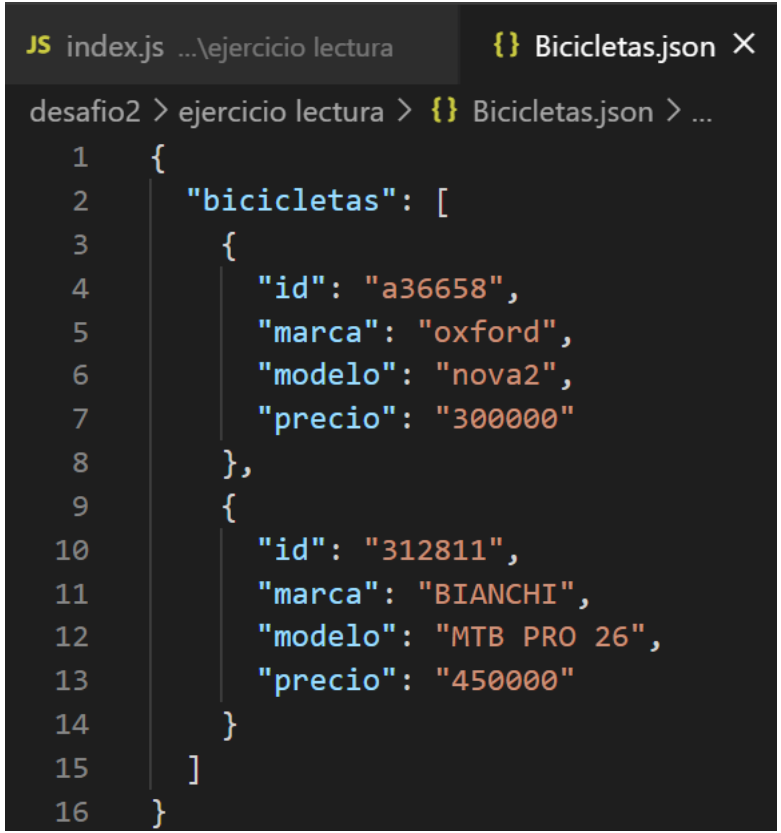
PUT

Ahora si consultas al servidor con curl usando la siguiente línea de comando:

```
curl -d '{"id":"312811","marca":"BIANCHI","modelo":"MTB PRO
26","precio":"450000"}' -H "Content-Type: application/json" -X PUT
http://localhost:3000/bicicletas
```

PUT

Nota que incluí la propiedad id con el valor “312811”, porque a mi se me generó ese id con el paquete uuid en esa bicicleta, en tu caso deberás usar un id propio, además de modificar algún valor sea de marca, modelo o precio, en mi caso le cambié el precio por 450000.



DELETE

Solo falta un cuarto verbo para tener nuestra API REST lista y este es el método DELETE. Para la creación de esta ruta hay que considerar que no es necesario que el cliente envíe ningún payload, puesto que solo debe indicar el id de la bicicleta que quiere que sea eliminada.

Sigue estos pasos para terminar la construcción de tu primera API REST con la creación de una ruta que atienda las peticiones DELETE:

- **Paso 1**: Crea la ruta **DELETE /bicicletas**.
- **Paso 2**: Almacena en una variable el ID recibido en la query string de la consulta.
- **Paso 3**: Utiliza el método *filter* para crear un nuevo arreglo que excluya la bicicleta con el mismo id recibido.

DELETE

- **Paso 4**: Sobrescribe el archivo Bicicletas.json y culmina la consulta indicando que la bicicleta fue eliminada con éxito.

```
// Paso 1
app.delete("/bicicletas", (req, res) => {
    // Paso 2
    const { id } = req.query;
    // Paso 3
    const bicicletasJSON = JSON.parse(fs.readFileSync("Bicicletas.json",
"utf8"));
    const bicicletas = bicicletasJSON.bicicletas;
    bicicletasJSON.bicicletas = bicicletas.filter((b) => b.id !== id);
    // Paso 4
    fs.writeFileSync("Bicicletas.json", JSON.stringify(bicicletasJSON));
    res.send("Bicicleta eliminada con éxito");
});
```

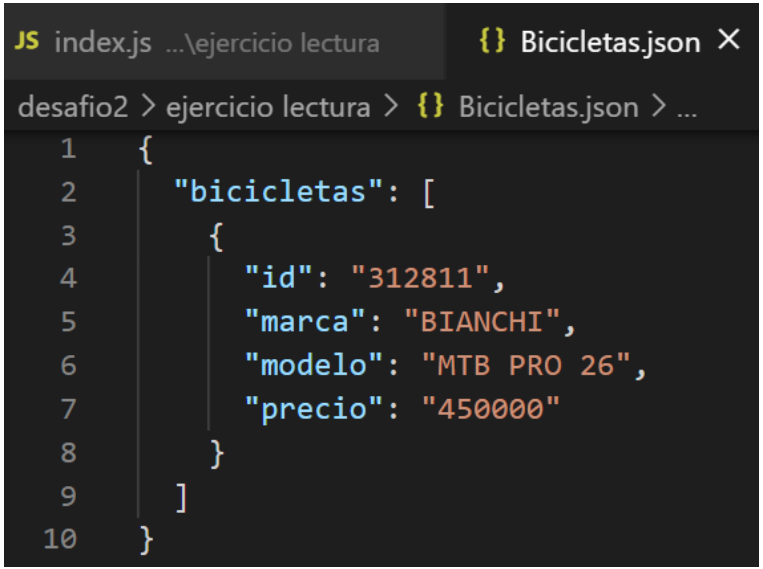
DELETE

Ahora si consultas al servidor pasando como parámetro algun id existente en el arreglo de bicicletas con la siguiente línea de comando:

```
curl -X DELETE http://localhost:3000/bicicletas?id=a36658
```

DELETE

Y revisas el documento Bicicletas.json verás que ya no está esa bicicleta, tal y como se muestra en la siguiente imagen:



Listo, ¡Felicidades! Con esto has logrado construir tu primera API REST. Te invito a que uses tus conocimientos en frontend para generar un sitio web con formularios que hagan consultas a tu servidor en los 4 diferentes verbos.

{desafío} latam_

Próxima sesión...

- *Desafío evaluado*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales