API REST

¡Comencemos!

Habilidades a evaluar

- Construir una API REST básica utilizando el framework Express para resolver un problema acorde al entorno Node.

Introducción

Las API REST no son un tema nuevo para nosotros, las conocimos previamente cuando aprendimos a desarrollar con Node puro. Ahora que nos encontramos aprendiendo Express, debemos adaptar los conceptos que ya conocemos a las sintaxis de nuestro framework.

En este capítulo aprenderás a crear y utilizar una API REST en un servidor desarrollado con Express, para servir un canal de comunicación que permita conectar 2 aplicaciones creadas bajo la arquitectura cliente-servidor. Esta arquitectura está en todos lados, por ello conocerla y manejarla te permitirá modularizar aún más tus aplicaciones, separando completamente el modelo de la vista, manteniendo su comunicación e incluso creando enlaces de servidor a servidor en una petición reenviada de una API REST a otra.

API REST

Las API REST son arquitecturas basadas en el protocolo HTTP, creadas para ser usadas como un puente de comunicación entre aplicaciones independientemente de la tecnología en la que sean construidas. Son desarrolladas en el lado del servidor, a través de los métodos o verbos HTTP que disponibilizan diferentes funciones, las que comúnmente conllevan a la gestión de recursos alojados en algún motor de bases de datos, sin embargo, también sirven para emitir una consulta que represente una situación por parte de quien emite la petición. Es importante destacar que es totalmente posible que un servidor se comunique con otro a través de una API REST.

Una de las ideas principales de su creación es la construcción de estructuras lógicas reutilizables, dentro de estas se pueden encontrar los middlewares, que no son más que funciones que se ejecutan en cada consulta de la ruta a la que pertenecen.

Esta arquitectura es considerada un estándar mundial y destaca por ser consumida desde cualquier tecnología o lenguaje que realice comunicaciones por HTTP y maneje datos JSON o XML. Aunque desde hace varios años se ha establecido por excelencia que el formato de datos para estas comunicaciones idealmente es JSON, por su cómoda lectura y peso ligero en comparación con XML.

Endpoints

En pocas palabras, los endpoints son una ruta creada para el consumo de un recurso, para la activación de una función o middleware definida en la configuración de nuestro servidor. También es conocido como la suma de la URL base, el path declarado en la ruta y el método HTTP.

Un ejemplo claro puede ser la PokeApi que siendo una API pública, tiene definido diferentes endpoints que devuelven diversas respuestas. En este link encontrarás todos los endpoints disponibles en esta API.

¿Cómo se puede formar un endpoint? Su estructura es la siguiente:

```
<url base>/<path>
```

Como puedes ver un endpoint es la unión entre la URL base y el path de una ruta. El path comúnmente se crea con parámetros para la definición de un endpoints de recurso dinámico, es decir, que un endpoint puede ser variable y su respuesta dependerá de los parámetros que se declaren en la ruta.

Parámetros de los endpoints

Al momento de definir un path en una ruta, podemos establecer un espacio para el libre tipeo del usuario o la definición esperada por el servidor. Este espacio variable se considera un parámetro.

Si retomamos el ejemplo de la pokeapi, podemos obtener los datos de un solo pokémon declarando al final del path el número identificador o el nombre directamente, por ejemplo:

https://pokeapi.co/api/v2/**pokemon/pikachu**

Siendo la palabra "pokémon" una definición en la ruta y "pikachu" un valor variable, nos encontramos con el parámetro del endpoint.

En Express manipulamos los parámetros de las rutas entre las propiedades del objeto request, específicamente la propiedad "params". Por ejemplo, si suponemos que pokeapi fue construida con Express, una ruta hipotética para este endpoint sería algo parecido al siguiente código:

```
app.get('/pokemon/:id', (req, res) => {
 const { id } = req.params
 ...
 res.send(pokemon)
})
```

En donde se tiene creada una ruta **GET /pokemon/:id** que devuelve como respuesta el JSON correspondiente al pokémon solicitado.

El uso más típico que encontramos en la construcción de una API REST, es servir como apoyo de un sistema tipo CRUD (Create Read Update Delete) a través de los métodos HTTP POST, READ, UPDATE y DELETE. En esta lectura construiremos progresivamente una API REST para el manejo de datos alojados en PostgreSQL, por supuesto, con el paquete pg, pero antes debemos tener lista nuestra arquitectura base.

Ejercicio guiado: Una API REST con Express

Construir una API REST que disponibilice los 4 métodos básicos HTTP para servir el backend de una aplicación cliente. La temática de esta API REST tratará de la gestión de canales de televisión que inicialmente estarán alojados como un arreglo de objetos en el servidor, pero posteriormente serán almacenados en PostgreSQL y a través del paquete pg haremos las consultas correspondientes para cumplir con las funcionalidades CRUD, este será un ejercicio progresivo que estaremos realizando en los próximos capítulos.

Sigue los pasos para la creación de esta API REST:

- ***Paso 1:*** Crear un servidor con express, incluir la importación e integración del paquete body-parser.

- ***Paso 2:*** Crear un arreglo con 2 objetos llamados “canales”, teniendo cada canal una propiedad “nombre”. Por motivos del ejercicio agregaremos los canales TNT y ESP.

- ***Paso 3:*** Crear una ruta **GET /canales** que devuelva el arreglo de objetos creado en el paso 2.

- ***Paso 4:*** Crear una ruta **POST /canal** que ingrese (push) el objeto recibido en el cuerpo de la consulta al arreglo de canales y devuelva como respuesta al cliente el arreglo con el nuevo canal agregado.

  Cabe destacar que podremos manipular el JSON enviado desde POSTMAN gracias a la integración del paquete body-parser en nuestro servidor.

- ***Paso 5:*** Crear una ruta **PUT /canal/:canal** que reciba como parámetro el nombre del canal que se desea cambiar y un payload en formato JSON con el nuevo nombre. Se devuelve el arreglo modificado como respuesta de esta ruta al cliente.

- ***Paso 6:*** Crear una ruta **DELETE /canal/:canal** que reciba como parámetro el nombre de un canal que deberás filtrar del arreglo de objetos. Se devuelve el arreglo modificado como respuesta de esta ruta al cliente.

```
// Paso 1
const express = require("express");
const app = express();
const bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.listen(3000);

// Paso 2
let canales = [{ nombre: "TNT" }, { nombre: "ESP" }];

// Paso 3
app.get("/canales", (req, res) => {
  res.send(canales);
});

// Paso 4
app.post("/canal", async (req, res) => {
  const nuevo_Canal = req.body;
  canales.push(nuevo_Canal);
  res.send(canales);
});

// Paso 5
app.put("/canal/:canal", async (req, res) => {
  const { canal } = req.params;
  const { nombre } = req.body;
  canales = canales.map((c) => (c.nombre == canal ? { nombre } : c));
  res.send(canales);
});

// Paso 6
app.delete("/canal/:canal", async (req, res) => {
  const { canal } = req.params;
  canales = canales.filter((c) => c.nombre !== canal);
  res.send(canales);
});
```

Con la API REST creada procedemos con las pruebas, así que abre POSTMAN y realiza los siguientes test:

1. Realiza una consulta **GET** con la siguiente dirección http://localhost:3000/canales, deberás recibir lo que te muestro en la siguiente imagen:

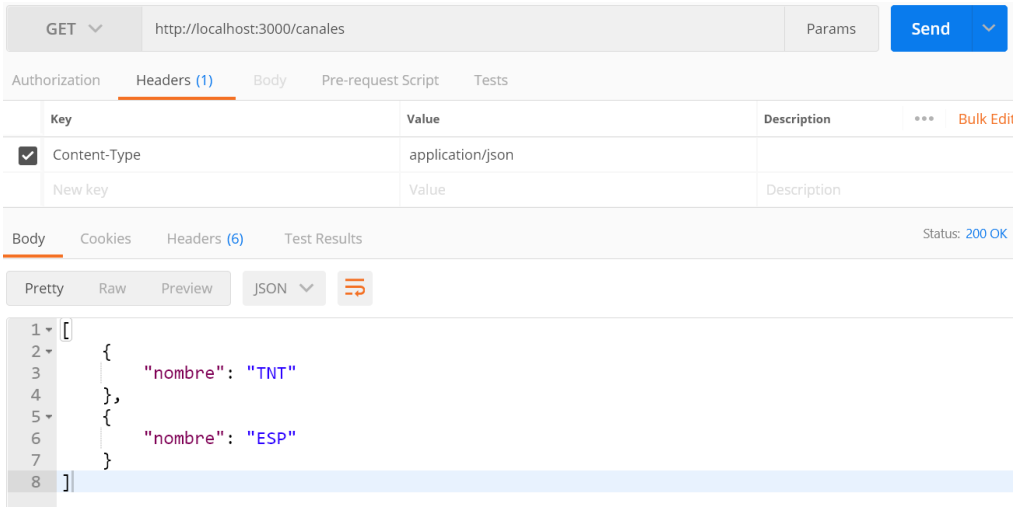

Imagen 1. Canales guardados en el servidor.
Fuente: Desafío Latam

Como ves, estamos recibiendo el arreglo de canales devuelto por nuestra API REST.

2. Realiza una consulta **POST** con la siguiente dirección http://localhost:3000/canal. Envía como cuerpo un JSON con el nombre de un canal nuevo. Deberás recibir lo que te muestro en la siguiente imagen:

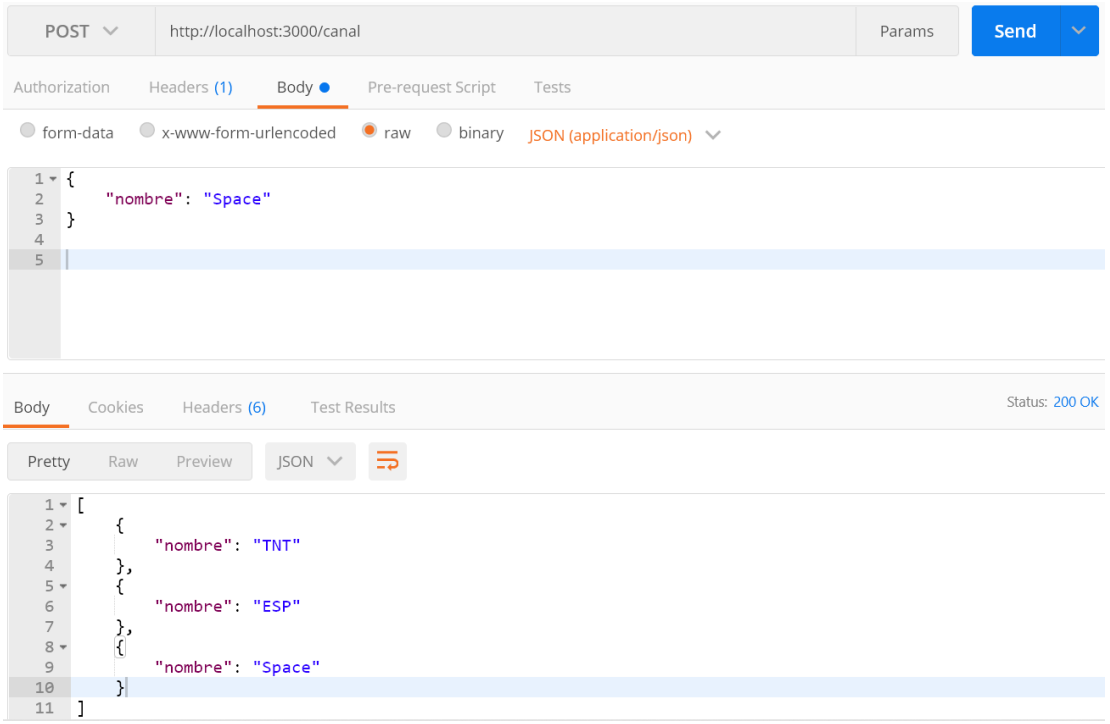

Imagen 2. Nuevo canal agregado al arreglo de canales.
Fuente: Desafío Latam

Ahora podemos observar que fue agregado al arreglo el canal que enviamos como cuerpo de la consulta **POST.**

3. Realiza una consulta **PUT** con la siguiente dirección http://localhost:3000/canal/TNT. Envía como cuerpo un JSON con el nombre de un canal que se sobrescribirá con el canal TNT que estamos declarando como parámetro. Deberás recibir lo que te muestro en la siguiente imagen:

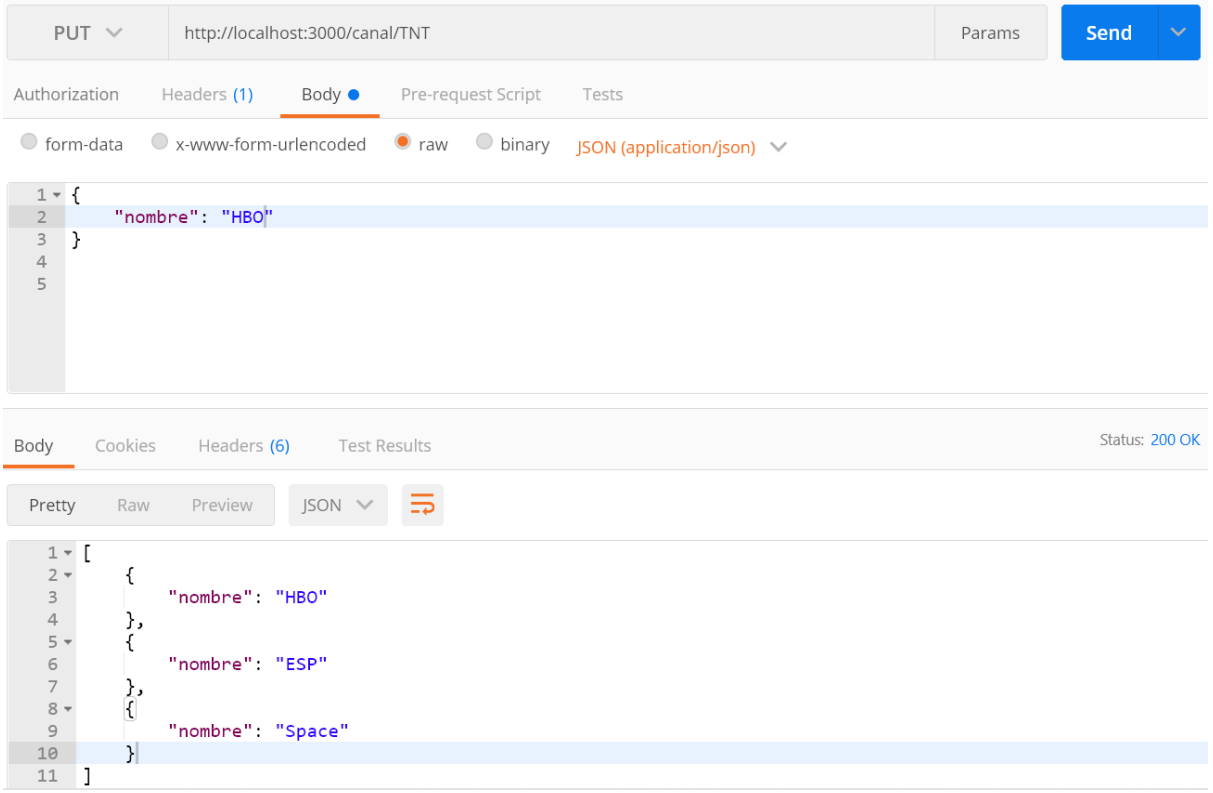

Imagen 3. Canal actualizado en el arreglo de canales.
Fuente: Desafío Latam

Con esto confirmamos que fue modificado el nombre del canal TNT por el canal HBO, ahora solo falta probar el **DELETE**.

4. Realiza una consulta **DELETE** con la siguiente dirección http://localhost:3000/canal/Space, deberás recibir lo que te muestro en la siguiente imagen:

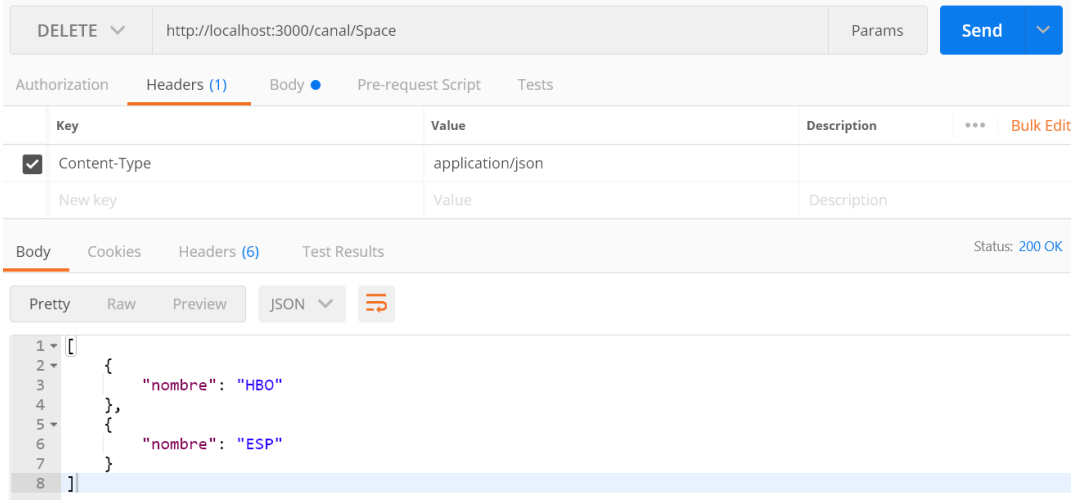

Imagen 4. Canal eliminado del arreglo de canales
Fuente: Desafío Latam

¡Excelente!, el canal fue eliminado del arreglo y nosotros hemos comprobado los 4 métodos HTTP definidos como rutas en nuestra API REST.

Ejercicio propuesto (5)

Debate con tus compañeros la siguiente pregunta:

En comparación con el desarrollo de una API REST con node puro, **¿Cuánto tiempo me he ahorrado por usar Express? ¿Cómo puedo mezclar esto con los conocimientos que he adquirido a lo largo de la carrera?**

Ejercicio propuesto (6)

Basado en el ejercicio guiado “Una API REST con Express”, desarrollar una API REST que gestione el siguiente arreglo:

```
let comidas = [{ nombre: "Pizza" }, { nombre: "Hamburguesa" }];
```