HATEOAS y los subrecursos

¡Comencemos!

Habilidades a evaluar

- Reconocer qué es HATEOAS para la implementación de una navegación basada en subrecursos.-

- Implementar HATEOAS en un servidor con Express para disponibilizar la consulta de subrecursos.

Introducción

Tarde o temprano nos encontraremos con la tarea de desarrollar una API REST para gestionar una gran cantidad de datos, los cuales pueden dividirse en categorías, que a su vez contienen una subcategoría con otra gran cantidad de información, ¿Cuál es el problema con esto? Que al momento de querer devolver todos los datos en una sola consulta, quedaría un modelo de datos incómodo de entender y leer, pudiéramos crear muchas rutas que representan cada categoría y sus subcategorías, pero de igual manera podría no ser lo más indicado.

En este capítulo aprenderemos a implementar HATEOAS, que es una acrónimo de Hypermedia As The Engine Of Application State (Hipermedia como motor del estado de la aplicación), que se define como el enlace de recursos derivados, más conocido como una arquitectura de recursos enlazados. Aprender a implementar esta arquitectura les brindará a tus clientes una mejor experiencia para consumir los datos que quieras servir a través de tu servidor.

HATEOAS

Por sus siglas en inglés Hypermedia as the Engine of Application State (Hipermedia como motor del estado de la aplicación), es un modelo de datos que podemos adoptar en nuestras API REST basado en recursos y subrecursos relacionados por enlaces. Con HATEOAS logramos tener un mapa claro y simple de navegar para cualquier cliente que quiera consumir los recursos que ofrezca nuestra API REST.

¿En qué casos debería considerar implementar HATEOAS en mi API REST? Veamos el siguiente ejemplo:

Una tienda de teléfonos popular cuenta con una API REST para simplificar las consultas de sus aplicaciones que solo requieren datos superficiales de sus productos, es decir, el software en el lado del cliente desea solo un listado con el nombre de todos los modelos de teléfonos disponibles en stock. En caso de no tener implementado HATEOAS el cliente recibiría la información detallada de todos los modelos en una misma lista, sin embargo, recordemos que solo necesita saber sus nombres, es aquí donde HATEOAS entra en juego devolviendo un modelo de datos que ofrece información superficial de los recursos y en la misma presentación ofrece un enlace para consumir subrecursos en donde se encuentre la información completa correspondiente al recurso seleccionado.

¿Aún no se entiende? No te preocupes, vamos con otro ejemplo pero ahora más visual. La API pública https://pokeapi.co/ aplica HATEOAS para la programación de sus rutas, podemos notarlo al consultar el siguiente endpoint https://pokeapi.co/api/v2/pokemon/ y recibir lo que te muestro en la imagen.


```
{
  "count": 1118,
  "next": "https://pokeapi.co/api/v2/pokemon/?offset=20&limit=20",
  "previous": null,
  "results": [
    {
      "name": "bulbasaur",
      "url": "https://pokeapi.co/api/v2/pokemon/1/"
    },
    {
      "name": "ivysaur",
      "url": "https://pokeapi.co/api/v2/pokemon/2/"
    },
    {
      "name": "venusaur",
      "url": "https://pokeapi.co/api/v2/pokemon/3/"
    },
    {
      "name": "charmander",
      "url": "https://pokeapi.co/api/v2/pokemon/4/"
```


Imagen 2. Ejemplo de HATEOAS con la pokeapi.
Fuente: PokeApi

Como puedes ver, la consulta a la url nos devuelve en la propiedad “results” un arreglo de objetos, donde cada objeto contiene solo dos propiedades: name y url. La URL representa el subrecurso de cada pokémon, es decir, que si la consultamos obtendremos la información detallada y específica de cada uno de ellos. Para ver un ejemplo ingresa a la url de “bulbasaur” y deberás recibir la data que se observa en la siguiente imagen:


```
{
  "abilities": [↔],
  "base_experience": 64,
  "forms": [↔],
  "game_indices": [↔],
  "height": 7,
  "held_items": [↔],
  "id": 1,
  "is_default": true,
  "location_area_encounters": "https://pokeapi.co/api/v2/pokemon/1/encounters",
  "moves": [↔],
  "name": "bulbasaur",
  "order": 1,
  "species": {↔},
  "sprites": {↔},
  "stats": [↔],
  "types": [↔],
  "weight": 69
}
```


Imagen 3. Data específica de 1 pokémon.
Fuente: Desafío Latam

¿Ves? Al consultar el sub recurso recibimos la información detallada de este pokémon y de la misma manera puedes hacerlo con todos.

La ventaja e importancia de considerar usar HATEOAS en nuestras API REST, se da por la necesidad de conocer la navegación de los recursos que ofrecemos en nuestro servicio, ello implica, que nuestros clientes al consultar un recurso puedan tener de primera mano el enlace directo a sus datos derivados. Ahora que entendemos que es HATEOAS y cómo reconocerlo, realicemos un ejercicio donde lo implementemos en un servidor con Express.

Ejercicio guiado: Implementando HATEOAS

La tienda All You Need Is Music Spa hará su apertura en un par de meses, por lo que necesita un sistema de consultas de sus productos y ha decidido contratar a un programador full stack developer para esto. Luego de unos días el programador ya tiene la aplicación en el lado del cliente y ha creado la base de datos. Para hacer las prueba registró 4 guitarras y ahora necesita proceder con la creación de la API REST aplicando HATEOAS.
Si no sabes de guitarras no tienes porqué preocuparte, pues no profundizaremos demasiado en la materia. Esto te servirá como simulación a un caso real, donde tendrás que crear o mantener APIs sobre rubros o áreas desconocidas e inevitablemente te tocará aprender un poco sobre el tema en pro de la lógica que debes aplicar en tu servicio.

Para el ejercicio guiado se debe descargar el **Apoyo Lectura - Servidor base Parte I** de esta sesión en donde encontrarás un servidor base hecho con Express y una carpeta data con un archivo JavaScript que exporta un arreglo de objetos que contienen las 4 guitarras. Para este ejercicio usarás el archivo guitarras.js para simular lo que te devolvería una consulta a una base de datos real.

El siguiente código corresponde al servidor que encontrarás en el apoyo lectura:

```
const express = require('express')
const guitarras = require('./data/guitarras.js')
const app = express()
app.listen(3000)

app.get('/', async (req, res) => {
  console.log(guitarras)
  res.send(guitarras)
})
```

¿Qué tenemos aquí? Estamos devolviendo todas las guitarras en la ruta raíz del servidor y con esto mostramos la información de cada guitarra, sin embargo, nuestro objetivo es aplicar HATEOAS en nuestra API REST, es decir, ofrecerle al cliente una respuesta con datos superficiales y los enlaces de los subrecursos para profundizar el detalle de cada guitarra, por lo que crearemos una función para estructurar mejor esta data. Sigue los pasos para devolver un modelo de datos basado en HATEOAS:

- ***Paso 1:*** Crear una función de nombre HATEOAS.

- ***Paso 2:*** Usar el método "map" para crear un arreglo de objetos, utilizando los datos de la guitarra. El objetivo es devolver solo 2 propiedades: name y href, en donde "href" deberá ser la siguiente ruta dinámica:

  http://localhost:3000/guitarra/<id de la guitarra>

  La función debe retornar el arreglo creado.

- ***Paso 3:*** Crear una ruta **GET /guitarras** que devuelva un objeto con una propiedad "guitarras" cuyo valor sea la ejecución de la función HATEOAS creada en el paso 1.

```
const express = require("express");
const guitarras = require("./data/guitarras.js");
const app = express();
app.listen(3000);

app.use(express.static("public"));

// Paso 1
const HATEOAS = () =>
  // Paso 2
  guitarras.map((g) => {
    return {
      name: g.name,
      href: `http://localhost:3000/guitarra/${g.id}`,
    };
  });

// Paso 3
app.get("/guitarras", (req, res) => {
  res.send({
    guitarras: HATEOAS(),
  });
});
```

Observa que ya no tendremos una ruta raíz y en cambio tenemos una ruta **GET /guitarras**, esto lo hacemos para aplicar las buenas prácticas que aprendimos en el capítulo anterior.

Ahora consulta la ruta creada y deberás obtener lo que te muestro en la siguiente imagen:

```
{
  "guitarras": [
    {
      "name": "Dean 350f",
      "href": "http://localhost:3000/guitarra/1"
    },
    {
      "name": "Ibanez RG8570Z",
      "href": "http://localhost:3000/guitarra/2"
    },
    {
      "name": "Epiphone Les Paul LP-100",
      "href": "http://localhost:3000/guitarra/3"
    },
    {
      "name": "Fender American Performer",
      "href": "http://localhost:3000/guitarra/4"
    }
  ]
}
```


Imagen 4. Modelo de datos de las guitarras con HATEOAS.
Fuente: Desafío Latam

Muy bien, hemos logrado mostrar solamente el nombre de las guitarras y un atributo “href” que redireccionará a un endpoint correspondiente a cada guitarra diferenciandolas por el id, sin embargo, aún no tenemos esa ruta creada, por lo que al consultar cualquiera de esos enlaces obtendremos el “Cannot GET /guitarra/1”, y ya sabemos que es un mensaje por defecto que devuelve Express cuando se consulta una ruta que no está aún programada.

Ahora sigue los pasos para agregar una función y ruta que permita devolver el detalle de una guitarra consultando el siguiente endpoint: http://localhost:3000/guitarra/<id>

- ***Paso 1:*** Crear una función de nombre guitarra que reciba un parámetro “id” y utilice el método “find” para retornar el objeto de una guitarra según el id recibido.

- ***Paso 2:*** Crear una ruta **GET /guitarra/:id** que recibirá de forma dinámica el id de una guitarra que se desea consultar.

- ***Paso 3:*** Almacenar en una constante el id recibido como parámetro en la ruta.

- ***Paso 4:*** Devolver la ejecución de la función “guitarra” pasando como argumento el id recibido en la ruta.

```
// Paso 1
const guitarra = (id) => {
  return guitarras.find((g) => g.id == id);
};

// Paso 2
app.get("/guitarra/:id", (req, res) => {
  // Paso 3
  const id = req.params.id;
  // Paso 4
  res.send(guitarra(id));
});
```

Ahora intenta consultar la siguiente dirección http://localhost:3000/guitarra/1 y deberás recibir lo que te muestro en la imagen:


```
{
  "id": 1,
  "name": "Dean 350f",
  "brand": "DEAN",
  "model": "350f",
  "body": "Stratocaster",
  "color": "Dark blue",
  "pickups": "Sigle Coil",
  "strings": 6,
  "value": 350,
  "stock": 2
}
```


Imagen 5. Data específica de una guitarra.
Fuente: Desafío Latam

¡Excelente! Ahí lo tenemos, la data específica de solo una guitarra, la que indicamos como parámetro "id" en la ruta.

Ejercicio propuesto (2)

Utilizar el mismo código del ejercicio guiado "Implementando HATEOAS" pero se debe agregar al modelo de datos devuelto en la ruta **/guitarras** una propiedad "cantidad" que indique cuántas guitarras existen en el arreglo.