Introducción a API REST básica con Node y Express

Desarrollo de Sistemas Web - Back End

Versión 1.0 F. Act 05/09/2024

API

El término API es una abreviatura de Application Programming Interfaces, que en español significa interfaz de programación de aplicaciones. Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.

De manera general podemos concluir que una API es un software que tiene la capacidad de conectarse e interactuar con otros para cumplir una o varias tareas.

¿Por qué REST? REST se refiere a cualquier interfaz entre sistemas que emplean el protocolo de comunicación HTTP para solicitar datos o realizar operaciones sobre estos usando algún formato como JSON.

API REST

Cuando hablamos de REST api nos referimos a un estándar para desarrollo de WEB api que incluye una arquitectura y protocolos http, REST es un acrónimo para sus siglas en inglés Representational State Transfer.

Con rest podemos realizar las 4 operaciones básicas conocidas como CRUD

- Create -> Creación de datos
- Read -> Lectura de datos
- Update -> Actualización de datos
- Delete -> Eliminado de datos

Las API REST pueden ser consumidas por aplicaciones web, aplicaciones móviles o cualquier otra cosa que sea capaz de entender el protocolo HTTP.

Los métodos básicos de HTTP

REST Crea una petición HTTP que contiene toda la información necesaria ya sea por medio del Body o parámetros en la llamada, un REQUEST a un servidor tiene toda la información necesaria y solo espera una respuesta o RESPONSE.

Los métodos básicos de HTTP que se utilizan son:

- Post: para generar o crear un registro nuevo.
- Get: para obtener uno o todos los elementos de un recurso.
- Put: realiza actualizaciones en los registros.
- Patch: realiza actualizaciones parciales de un registro.
- Delete: Para borrar un registro.

Todos los objetos se manipulan mediante URI, por ejemplo, si tenemos un recurso usuario y queremos acceder a un usuario en concreto nuestra URI seria /user/identificadordelobjeto, con eso ya tendríamos un servicio USER preparado para obtener la información de un usuario, dado un ID.

¿Qué es Express.js?

- Es un framework rápido, minimalista y flexible de Node.js. Permite crear APIs y aplicaciones web fácilmente, provee un conjunto de características como manejo de rutas (direccionamiento), archivos estáticos, uso de motor de plantillas, integración con bases de datos, manejo de errores, middlewares entre otras.

¿Qué es un framework de desarrollo web?

- Es un framework rápido, minimalista y flexible de Node.js. Permite crear APIs y aplicaciones web fácilmente, provee un conjunto de características como manejo de rutas (direccionamiento), archivos estáticos, uso de motor de plantillas, integración con bases de datos, manejo de errores, middlewares entre otras.

Proporciona mecanismos para:

- Escritura de manejadores de peticiones con diferentes verbos HTTP en diferentes caminos URL (rutas).
- Integración con motores de renderización de "vistas" para generar respuestas mediante la introducción de datos en plantillas.
- Establecer ajustes de aplicaciones web como qué puerto usar para conectar, y la localización de las plantillas que se utilizan para renderizar la respuesta.
- Añadir procesamiento de peticiones Middleware adicional en cualquier punto dentro de la tubería de manejo de la petición.

¿Cómo funciona Express?

En sitios web o aplicaciones web dinámicas, que accedan a bases de datos, el servidor espera a recibir peticiones HTTP del navegador (o cliente). Cuando se recibe una petición, la aplicación determina cuál es la acción adecuada correspondiente, de acuerdo a la estructura de la URL y a la información (opcional) indicada en la petición con los métodos POST o GET .

Dependiendo de la acción a realizar, puede que se necesite leer o escribir en la base de datos, o realizar otras acciones necesarias para atender la petición correctamente. La aplicación ha de responder al navegador, normalmente, creando una página HTML dinámicamente para él, en la que se muestre la información pedida, usualmente dentro de un elemento específico para este fin, en una plantilla HTML.

Instalación

Express se instala vía "npm". Ya vimos cómo es la operativa para instalar paquetes de Node, iniciamos nuestro proyecto con: **npm init** en la carpeta que vamos a trabajar. Ponemos el nombre del proyecto y nos va a generar el package.JSON

Algunas cosas para recordar hasta ahora

Crear carpeta: Luego desde la barra de navegación poner cmd y escribir en la consola code .
Iniciar Package.JSON: npm ini –y en terminal
Instalar Thunder Client: Es una extensión para Visual Studio Code que se utiliza para probar y hacer solicitudes
Instalar variables de entorno: npm install dotenv
Instalar Nodemon: npm install --save-dev nodemon
Para instalar Express: npm install express
Instalar Pug: npm install pug
Instalar App-Gen: npm install express-generator -g
Para generar una nueva aplicación Express: express nombre-de-tu-aplicacion

Express Requests (El objeto request)

El objeto request tiene más propiedades que la petición del http request del cual extiende. Estas propiedades simplifican el desarrollo y proveen funcionalidad adicional.

Propiedades de objeto request

request.params
Parámetros del url.

request.query
Parámetros del query string..

request.route
Ruta actual como string.

request.cookies
Cookies (requiere de cookie parser).

request.signedCookies
Cookies firmadas (requiere de cookie parser).

request.body
Contenido del cuerpo de la petición (requiere de cookie parser).

request.headers
Cabeceras de la petición.

Express Responses (El objeto response)

El response puede ser utilizado y modificado antes de ser enviado de regreso.
Métodos del objeto response

`response.redirect(url)`
Redirecciona la respuesta.

`response.send(data)`
Envía la respuesta.

`response.json(data)`
Envía un json de regreso

y agrega las cabeceras apropiadas.

`response.sendfile(path, options, callback)`
Envía un archivo para descarga.

`response.render(template, locals, callback)`
Procesa un template.

`response.locals`

Nodemon

Cada vez que hacemos cambios en el código de la aplicación, tenemos que reiniciarla para ver esos cambios en nuestro servidor. Reiniciamos la aplicación cerrándola primero escribiendo Ctrl+C en la consola y luego volviendo a levantarla.

Sin embargo, esto es un poco engorroso. La solución a este problema es nodemon: nodemon observará los archivos en el directorio en el que se inició nodemon, y si algún archivo cambia, nodemon reiniciará automáticamente su aplicación de node.

Obteniendo un solo recurso

Se puede acceder al parámetro id en la ruta de una solicitud a través del objeto request:

```
router.get('/:id', (req, res) => {

        const id = req.params.id;

        const note = notes.find((note)
      => note.id === id);

        res.json(note);

});
```

Ejemplo de rutas en Express

El enrutamiento se refiere a determinar la forma en que una aplicación responde a una solicitud del cliente a un punto final en particular. Por ejemplo, un cliente puede hacer una solicitud HTTP (GET, POST, PATCH o DELETE) para varias URL, como las que se muestran a continuación:

http://localhost:3000/cuenta/1

```
appExpress.get("/cuenta/:id", (req, res) => {

    res.send('Hola mundo!! Express!!');

});
```

Manejo de rutas (Routes)

Rutas dinámicas: Definir rutas como en el ejemplo anterior es muy tedioso de mantener. Para separar las rutas de nuestro archivo index.js principal, usaremos Express.Router. Imaginemos que tenemos la aplicación de una biblioteca. Posiblemente necesitaríamos una ruta /books que nos permita obtener una lista de libros.

Rutas coincidentes con patrón: También podemos Usar expresiones regulares para restringir la coincidencia de parámetros de URL. Supongamos que necesitamos que el ID sea un número largo de 10 dígitos. Debemos tener en cuenta que esto sólo coincidirá con las solicitudes que tienen un ID de 10 dígitos. Podemos usar expresiones regulares más complejas para hacer coincidir/validar las rutas.

Rutas no válidas: Si ninguna de las rutas coincide con la solicitud, recibiremos el mensaje "Cannot GET /Hola" como respuesta.

Middlewares en Express

Un Middleware es un mecanismo de enlace de software que se encuentra entre la aplicación y los servicios subyacentes. Los middlewares en Express se montan por múltiples razones, una de ellas por ejemplo es validar la información antes de que llegue a la rutina que enviará respuesta hacia el cliente, también pueden usarse para hacer una consulta y guardar información antes de que pase a las funciones que responderán. El patrón middleware implementa continuidad. La petición proviene de un cliente y una respuesta es enviada de vuelta a este.

> El patrón middleware implementa continuidad
>
> La petición proviene de un cliente y una respuesta es enviada de vuelta a este.
>
> petición => middleware1 => middleware2 => ruta => respuesta

Existen 2 tipos de middleware en express:

- Middleware de npm: como por ejemplo body-parser: app.use(bodyParser.json()).
- Middleware personalizados: que se utilizan como app.use((req, res, next)).

Middleware de terceros

La mayoría de las aplicaciones usan middlewares desarrollados por terceras personas para simplificar funciones habituales en el desarrollo web, como puede ser: gestión de cookies, sesiones, autenticación de usuarios, peticiones `POST` y datos en JSON, registros de eventos, etc.

Formato de Respuestas a Solicitudes

En la mayoría de los sitios web que visitamos o las APIs que usamos, las respuestas del servidor son raramente texto sin formato. Obtenemos páginas HTML y datos JSON como formatos de respuesta comunes. La respuesta que devolvemos de un servidor web puede tener varios formatos. Hasta ahora hemos utilizado texto plano, pero Node permite devolver otros formatos como JSON, HTML , XML y CSV. Además, los servidores web pueden devolver datos que no son texto, como PDFs, archivos comprimidos, audio y vídeo (archivos estáticos, como vimos anteriormente).

A continuación, aprenderemos cómo devolver los siguientes tipos de datos:

JSON: Formato para el intercambio de datos basado en texto.

CSV: Formato de archivo que permite almacenar datos en forma de texto separados por comas.

HTML: Lenguaje de marcado utilizado para la creación de páginas web.

Plantillas: "motor de renderización de vistas" por Express

El Motor de plantilla (referido como "motor de renderización de vistas" por Express) nos permite definir la estructura de documento de salida en una plantilla, usando marcadores de posición para datos que serán llenados cuando una página es generada. Las plantillas son utilizadas generalmente para crear HTML, pero también pueden crear otros tipos de documentos.

Los motores de plantillas: Se utilizan para eliminar el desorden de nuestro código de servidor con HTML, concatenando cadenas de forma salvaje a las plantillas HTML existentes.

Pug

Pug es un motor de plantillas muy poderoso que tiene una variedad de características que incluyen filtros, inclusiones, herencia, interpolación, etc. Para usar Pug con Express, necesitamos instalarlo.

Para que Express pueda representar archivos de plantilla, deben establecerse los siguientes valores de aplicación:

views: El directorio donde se encuentran los archivos de plantilla. Ejemplo: app.set('views', './views')

view engine: El motor de plantilla que se utiliza. Ejemplo: app.set('view engine', 'pug')

Ahora creamos un nuevo directorio llamado *views* que contendrá nuestras vistas. Dentro de eso, creamos un archivo llamado first_view.pug (.pug es la extensión de las plantillas) e ingresamos los siguientes datos en él:

Gestión de errores

El manejo de errores en Node.js es uno de los aspectos más importantes en cualquier aplicación. A continuación, aprenderemos cómo crear objetos de error y cómo lanzar y manejar errores en Node.js.

Primero, tenemos que entender que no todos los errores son iguales. Veamos cuántos tipos de errores pueden ocurrir en una aplicación:

Error generado por el usuario: El todo es más que la suma de las partes.

Fallo de hardware: El todo determina la naturaleza de las partes.

Error de tiempo de ejecución: Las partes no pueden comprenderse si se consideran en forma aislada del todo.

Error de la base de datos: Las partes están dinámicamente interrelacionadas o son interdependientes.

¿Cómo está organizada la estructura de express?

Organización de la estructura de Express:

app.js: Es el archivo principal en el cual se encuentra la lógica del servidor y de la aplicación.

/public: Contiene archivos estáticos que son desplegados por el servidor.

/routes: Contiene rutas personalizadas para API’s basadas en REST.

/views: Contiene plantillas que pueden ser procesadas por un motor de plantillas.

/package.json: Archivo manifest del proyecto.

/www: Contiene los scripts de arranque del servidor.

Testear con Postman

Para poder probar nuestras API de forma más productiva vamos a tener que usar algún cliente de APIs que nos permita hacer las pruebas de funcionamiento de lo que estamos construyendo, los dos clientes más famosos son Insomnia y Postman y vamos a necesitar de alguno de ellos para ir probando cada característica que vamos a ir construyendo.

Postman en sus inicios nace como una extensión que podía ser utilizada en el navegador Chrome de Google y básicamente nos permite realizar peticiones de una manera simple para testear APIs de tipo REST propias o de terceros.

Gracias a los avances tecnológicos, Postman ha evolucionado y ha pasado de ser de una extensión a una aplicación que dispone de herramientas nativas para diversos sistemas operativos como lo son Windows, Mac y Linux.

Para qué sirve Postman

Postman sirve para múltiples tareas dentro de las cuales destacaremos en esta oportunidad las siguientes:

Testear colecciones o catálogos de APIs tanto para Frontend como para Backend.

Organizar en carpetas, funcionalidades y módulos los servicios web.

Permite gestionar el ciclo de vida (conceptualización y definición, desarrollo, monitoreo y mantenimiento) de nuestra API.

Generar documentación de nuestras APIs.

Trabajar con entornos (calidad, desarrollo, producción) y de este modo es posible compartir a través de un entorno cloud la información con el resto del equipo involucrado en el desarrollo.