TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE

- Implementando una API REST
- Iniciar un proyecto REST Server con Express
- Crear Rutas con retorno JSON para peticiones HTTP del tipo: GET, PUT, POST, DELETE
- En qué caso se usa cada tipo de Petición
- Cómo es la estructura de los Endpoints según el tipo de petición
- Recepción de parámetros en una petición HTTP
- Procesando el retorno JSON
- Uso del paquete body-parser
- Códigos de respuesta HTTP
- Inspeccionar los distintos códigos de respuesta del protocolo HTTP: 1XX, 2XX, 3XX, 4XX, 5XX
- Respondiendo códigos personalizados en una petición HTTP

INICIAR UN PROYECTO REST SERVER CON EXPRESS

Para inicializar una primera API REST se debe verificar que se tiene instalado tanto Node.js como npm, seguidamente se puede realizar la creación del proyecto de tres formas, la primera es creando y construyendo manualmente el archivo package.json (no recomendado), haciendo uso del comando npm init (de cierta forma el más recomendado) o usando la dependencia de express-generator.

En primera instancia se debe crear una carpeta en alguna ubicación dentro de nuestro sistema, seguidamente accedemos a la misma por medio de línea de comandos, y estando dentro la carpeta ejecutaremos el comando `npm init`, el cual nos solicitará algunos datos referentes al proyecto, al finalizar, nos habrá creado el archivo package.json y una carpeta llamada node_modules, que contiene todas las librerías estándar de NodeJS.

Seguidamente procedemos a instalar los módulos de NPM que vamos a requerir, el express y el body-parser. El Express es el módulo más popular para desarrollos web en NodeJS y además, es especialmente útil para la construcción de API's, por otra parte, el Middleware body-parse, que anteriormente era parte de Express, ayuda a procesar el payload (o body) del request, el cual lo procesa y nos lo deja listo y en el formato que lo necesitamos, en este caso, en JSON.

```
npm install -save express
npm install -save body-parser
```

CREAR RUTAS CON RETORNO JSON PARA PETICIONES HTTP DEL TIPO: GET, PUT, POST, DELETE

Las API REST se utilizan para acceder y manipular datos mediante un conjunto común de operaciones sin estado. Estas operaciones son parte integral del protocolo HTTP y representan la funcionalidad esencial de creación, lectura, actualización y eliminación (CRUD).

EN QUÉ CASO SE USA CADA TIPO DE PETICIÓN

A continuación se detalla cuales son las peticiones disponibles al igual que información sobre cuando se usa cada tipo de petición:

- `POST` (crear un recurso o proporcionar datos en general, se utiliza para enviar datos, normalmente se utiliza para crear nuevas entidades o editar entidades ya existentes.).
- `GET` (recuperar un índice de recursos o un recurso individual. se usa para solicitar datos del servidor, generalmente se usa para leer datos).
- `PUT` (crear o reemplazar un recurso. se usa para reemplazar completamente el recurso con el recurso enviado, generalmente se usa para actualizar datos).
- `PATCH` (actualizar/modificar un recurso).
- `DELETE` (quitar un recurso. Se utiliza para eliminar una entidad del servidor).

Usando estas operaciones HTTP y un nombre de recurso como dirección, podemos construir una API REST creando un endpoint para cada operación. Y al implementar el patrón, tendremos una base estable y fácilmente comprensible que nos permitirá desarrollar el código rápidamente y mantenerlo después. Como se mencionó anteriormente, se usará la misma base para integrar funciones de terceros, la mayoría de las cuales también usan API REST, lo que hace que dicha integración sea más rápida.

CÓMO ES LA ESTRUCTURA DE LOS ENDPOINTS SEGÚN EL TIPO DE PETICIÓN

Se debe utilizar sustantivos en lugar de verbos en las rutas de los endpoint. Los sustantivos representan la entidad que el endpoint estamos recuperando o manipulando como el nombre de la ruta.

Esto se debe a que nuestro método de solicitud HTTP ya tiene el verbo. Tener verbos en nuestras rutas de puntos finales de API no es útil y lo hace innecesariamente largo ya que no transmite ninguna información nueva. Los verbos elegidos pueden variar según el desarrollador. Por ejemplo, a algunos les gusta 'obtener'

y otros les gusta 'recuperar', por lo que es mejor dejar que el verbo HTTP `GET` nos diga qué hace un punto final.

Con los dos principios mencionados anteriormente, podemos construir los siguientes endpoint de un conjunto de artículos de noticias:

Ruta	Verbo HTTP	Descripción
/api/articulos	GET	Listado de todos los artículos
/api/articulos/search?=palabra	GET	Listado de artículo según palabra
/api/articulos/:id	GET	Datos de un artículo
/api/articulos	POST	Ingreso de un artículo
/api/articulos/:id	PUT	Actualización de un artículo
/api/articulos/:id	DELETE	Eliminación de un artículo

El siguiente ejemplo refleja el código en Express del API con los endpoint:

```
const express = require('express');
const bodyParser = require('body-parser');

const app = express();

app.use(bodyParser.json());

app.get('/articulos', (req, res) => {
  const articulos = [];
  // código necesario para la obtención
  res.json(articulos);
});

app.post('/articulos', (req, res) => {
  // código para agregar un artículo
  res.json(req.body);
});

app.put('/articulos/:id', (req, res) => {
  const { id } = req.params;
  // código para la actualización de un artículo
  res.json(req.body);
```

```
});

app.delete('/articulos/:id', (req, res) => {
  const { id } = req.params;
  // código para la eliminación de un  artículos
  res.json({ deleted: id });
});

app.listen(3000, () => console.log('Servidor Iniciado'));
```

RECEPCIÓN DE PARÁMETROS EN UNA PETICIÓN HTTP

Cuando el router recibe una petición, podemos observar que ejecuta una función de callback:

```
function (req, res){}
```

- El parámetro `req` representa la petición (request)
- El parámetro `res` representa la respuesta (response)
- Una respuesta en json puede ser:
  `res.json({ mensaje: '¡Bienvenido a API RESTful!' })`

Por lo general cuando se envía algún parámetro adicional se puede obtener por:

La url: se recogerán mediante `req.param.nombreVariable`. Por ejemplo:

Enviando el parámetro nombre a la API, y que refleje un mensaje personalizado.

```
router.get('/:nombre', function(req, res) {
            res.json({
                mensaje: '¡Hola' + req.params.nombre
            })
        }
```

Mediante post en http hay dos posibilidades:

- Parámetros mediante `POST` y `application/x-www-form-urlencoded`:
  Necesitaremos `body-parser`: extrae los datos del body y los convierte en json

- Parámetros mediante POST y multipart/form-data
  Usaremos https://www.npmjs.com/package/busboy o https://www.npmjs.com/package/multer

En peticiones post, se utiliza por lo general x-www-form-urlencoded si no se envía una gran cantidad de datos en archivos

PROCESANDO EL RETORNO Y ACEPTACIÓN DE JSON

Las API REST deben aceptar JSON para la carga útil de la solicitud y también enviar respuestas a JSON. JSON es el estándar para la transferencia de datos. Casi todas las tecnologías en red pueden usarla: JavaScript tiene métodos integrados para codificar y decodificar JSON, ya sea a través de Fetch API u otro cliente HTTP. Las tecnologías del lado del servidor tienen bibliotecas que pueden decodificar JSON sin hacer mucho trabajo.

Hay otras formas de transferir datos. XML no es ampliamente compatible con los frameworks sin transformar los datos por nosotros mismos, es por ello que se suele usar JSON. No podemos manipular estos datos tan fácilmente en el lado del cliente, especialmente en los navegadores. Termina siendo mucho trabajo extra solo para hacer una transferencia de datos normal, en el caso del XML.

Los datos de formulario son buenos para enviar datos, especialmente si queremos enviar archivos. Pero para texto y números, no necesitamos datos de formulario para transferirlos ya que, con la mayoría de los frameworks, podemos transferir JSON simplemente obteniendo los datos directamente en el lado del cliente.

Para asegurarnos de que cuando nuestra aplicación REST API responda con JSON, los clientes lo interpreten como tal, debemos configurar el encabezado `Content-Type` de respuesta `application/json` después de realizar la solicitud. Muchos frameworks de aplicaciones del lado del servidor configuran el encabezado de respuesta automáticamente. Algunos clientes HTTP miran el encabezado `Content-Type` de respuesta y analizan los datos de acuerdo con ese formato.

La única excepción es si estamos tratando de enviar y recibir archivos entre el cliente y el servidor. Luego, debemos manejar las respuestas de los archivos y enviar los datos del formulario del cliente al servidor.

También debemos asegurarnos de que nuestros endpoint devuelvan JSON como respuesta. Muchos marcos del lado del servidor tienen esto como una función integrada.

En una API REST podemos realizar cargas con formato JSON. Utilizaremos un frameworks de back-end con Node.js y Express. Podemos usar el middleware `body-parser` para analizar el cuerpo de la solicitud JSON y luego podemos llamar al método res.json con el objeto que queremos devolver como la respuesta JSON.