RESTful API y Autenticación

¿Qué es una API?

Una interfaz de programación de aplicaciones (API) es un conjunto de reglas, definiciones y protocoles que permiten la comunicación entre distintos sistemas. A veces son interpretadas como un ‘contrato’ entre el proveedor de información, el proveedor, y el consumidor de dicha información, el cliente. En este ‘contrato’ se establece el contenido que se requiere de ambos el cliente y el proveedor. Por ejemplo, un API para una librería puede requerir que el usuario especifique el titulo del libro que esta buscando y requerir que el proveedor responda con una lista de todos los libros con títulos similares. En otras palabras, una API ayuda a comunicar que es lo que quiere el cliente para que el sistema logre entendí y cumplir su solicitud.

¿Qué es REST?

La transferencia de estado representacional (REST) es una arquitectura de software que impones ciertas condiciones sobre cómo debería funcionar una API. Para que una API se considere RESTful debe cumplir los siguientes criterios:

- Una arquitectura cliente-servidor con peticiones gestionadas a través de HTTP.
- Comunicación cliente-servidor sin estado, lo que significa que no almacena información del cliente entre solicitudes y cada solicitud es independiente de otras.
- Datos almacenables en cache
- Interfaz uniforme entre componentes Esto requiere lo siguiente:
    - Los recursos solicitados son identificables y están separados en las representaciones enviadas al cliente.
    - El cliente puede manipular los recursos a través de la representación que recibe.
    - Los mensajes devueltos al cliente tienen suficiente información para describir como lo debe procesar el cliente.
    - El cliente puede utilizar hipervínculos para encontrar las demás acciones disponibles que puede realizar.
- Un sistema de capas que organiza cada tipo de servidor que interviene en la recuperación de la información.
- Tener la opción de ampliar o personalizar temporalmente la funcionalidad del cliente transfiriéndole código de programación.

A pesar de que debe cumplir todos estos criterios, esta arquitectura de API sigue considerándose más fácil de usar que varias de sus contrapartes.

¿Cómo funcionan las API RESTful?

Los pasos generales para cualquier solicitud por medio de una API RESTful son los siguientes:

1. El cliente envía una solicitud al servidor. Para que el servidor entienda dicha solicitud se formatea según la documentación de la API.
2. El servidor autentica al cliente y confirma que tiene permiso para realizar esa solicitud.
3. El servidor recibe la solicitud y la procesa.
4. El servidor devuelve una respuesta al cliente que contiene una indicación de si la solicitud se realizo correctamente. Si se logro realizar correctamente también incluirá la información que fue solicitada.

Los detalles particulares de cada solicitud y su respuesta correspondiente pueden varían dependiendo del diseño de los desarrolladores.

¿Qué contiene la solicitud de cliente de la API RESTful?

Las API RESTful requieren que tengan ciertos componentes principales. El primero de ellos es el endpoint, este componente es la URL especifica que se utiliza para identificar la ruta del recurso que se esta solicitando. Los endpoint están compuestos por una URL base concatenada con la ruta específica del recurso. Ejemplo:

```
https://api.example.com/ruta
```

Después del endpoint tenemos el método HTTP, esto le indica al servidor lo que debe hacer con el recurso. Los métodos mas comunes son el GET, POST, PUT, DELETE que corresponden a las operaciones CRUD. Una vez definido el método pasamos a las cabeceras, estas son metadatos intercambiados entre el cliente y el servidor. Comúnmente los metadatos que se incluyen en las cabeceras son el tipo de contenido ya sea json, xml, etc. y la autenticación. Finalmente se incluyen los datos que el cliente mandara al servidor junto con la solicitud para que el método especificado funcione correctamente.

En JavaScript podemos hacer solicitudes al servidor con la función fetch, utilizando la siguiente estructura:

```
fetch('https://api.example.com/ruta', {
       method: 'METODO HTTP',
       headers: {
            'Content-Type': 'application/json',
            'Authorization': 'token',
       },
       body: JSON.stringify(data) // en casos donde no se necesita datos adicionales esto se puede remover
       })
        .then(response => response.json())
        .then(data => console.log(data))
        .catch((error) => console.error('Error: ', error));
```

Autenticación con JWT

La autenticación de API es el proceso de gestión de usuarios que verifica la identidad de un usuario o aplicación que solicita acceso a una API. Esto se puede realizar utilizando varios métodos tales como proporcionar un usuario junto con contraseña o utilizar un sistema basado en tokens. En los sistemas basados en tokens usualmente se utilizan Json Web Tokens (JWT) las cuales consisten de tres partes. La cabecera donde se especifica el tipo de token y el algoritmo de firma, la carga con información relevante sobre una entidad (usualmente el usuario) e información adicional y la firma. La firma es lo que nos asegura la integridad y autenticidad del token, usualmente se crea a base de la cabecera encriptada, la carga encriptada y un secreto. Todo esto termina formando una cadena que contiene toda la información necesaria para verificar la identidad del usuario.

Para implementar un sistema basado en tokens con JWT en Node.js hacemos lo siguiente:

1. Instalar el paquete jsonwebtoken

```
npm install jsonwebtoken
```

2. Configurar el secreto dentro del .env

```
//.env
PORT = 3030
JWT_SECRET = jwt_secret_key
```

3. Generar Token

En el server.js podemos introducir el siguiente bloque que se encargara de generar los tokens. Para este caso en especifico el token contiene información sobre cuando ingreso por primera vez el usuario en lugar de información del usuario en sí.

```
app.post("/user/generateToken", (req, res) => {

    let jwtSecret = process.env.JWT_SECRET;
    let data = {
        time: Date(),
    }

    const token = jwt.sign(data, jwtSecret); //const token = jwt.sign(data, jwtSecret {expiresIn:'tiempo'});
                                                para tokens temporales
    res.send(token);
});
```

4. Validar Token

Cada vez que el usuario haga una solicitud al servidor se necesita verificar su token, para esto utilizaremos la siguiente función de middleware.

```
const verify = async(req, res, next) => {
    let jwtSecret = process.env.JWT_SECRET;
    try {
        const token = req.header('Authorization');
        const verified = jwt.verify(token, jwtSecret);
        if(verified){
            console.log("Verificado Exitosamente");
            next();
        }else{
            //usuario no validado
            return res.status(401).send(error);
        }
    } catch (error) {
        return res.status(401).send(error);
    }
};
```

Ahora para toda ruta/solicitud que requiera verificación utilizaremos la siguiente estructura dentro del servidor:

```
app.metodo("/ruta", verify, (req, res) => {
    //codigo
});
```

Esta función llama al middleware para verificar antes de procesar la solicitud. Si el usuario fue validado la procesara sin problema, en caso contrario el servidor responderá con un error 401, solicitud no autorizada.