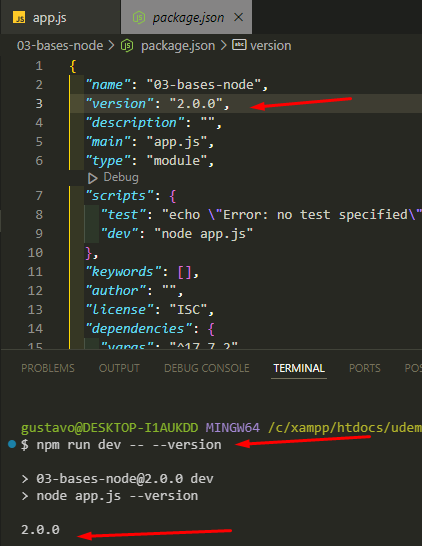
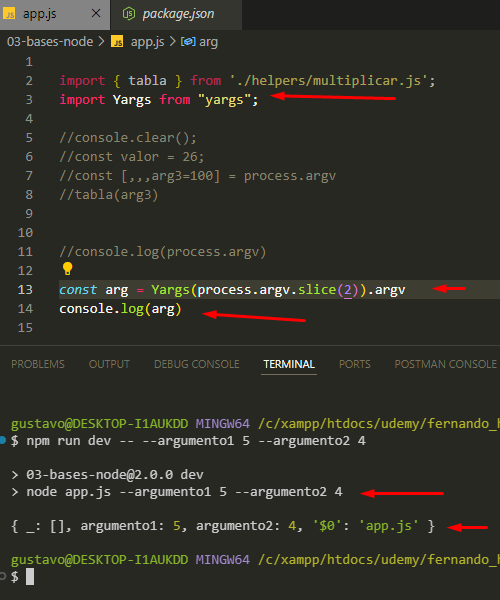
Como saber la vercion de nuestro archivo package.json

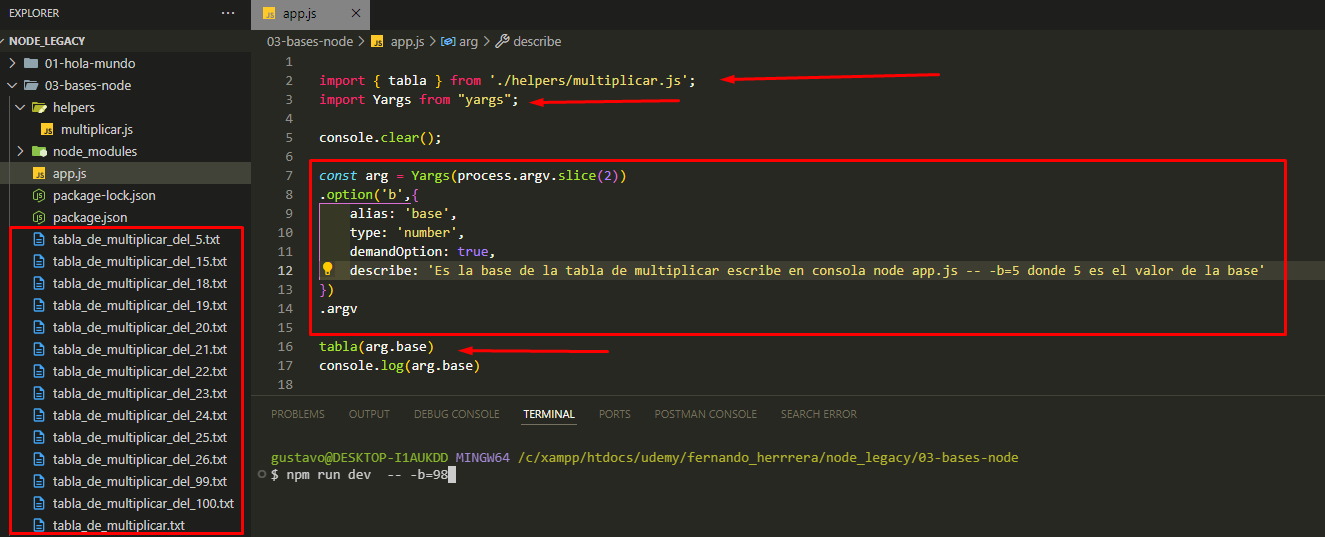


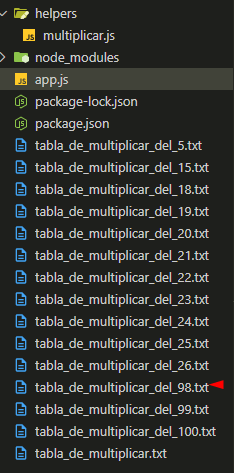
Utilizando yargs y pasandole parámetros, le pasamos los comandos

npm run dev -- --argumento1 5 --argumento2 4 y los recuperamos con yargs

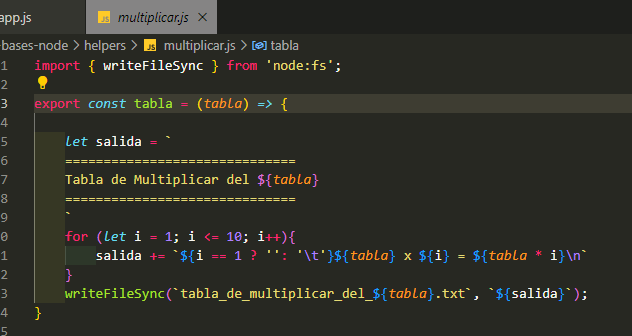


Generando una tabla de multiplicar desde la consola, con nuestro script, y con yargs



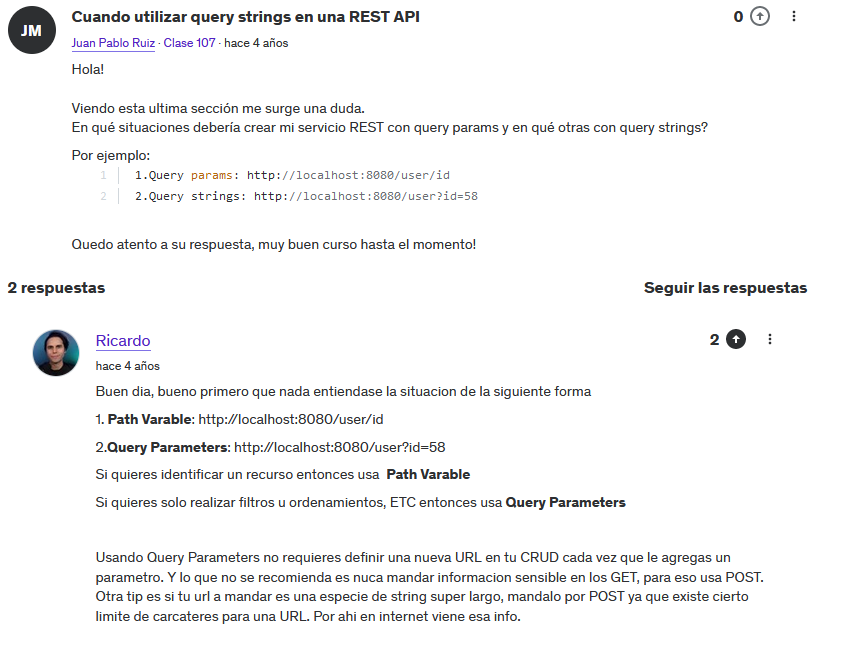


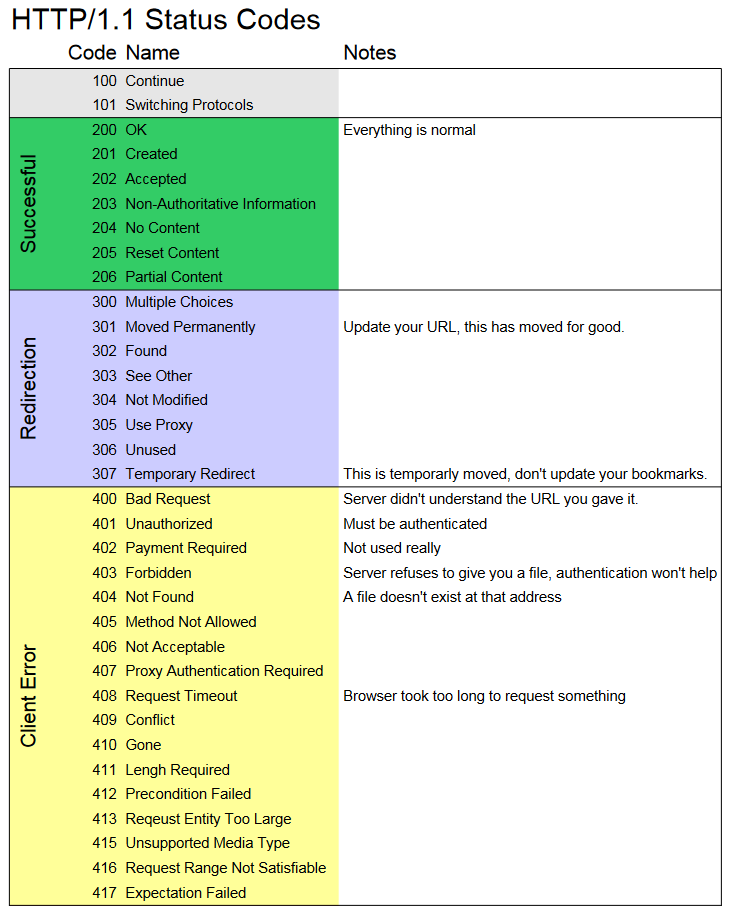
Este es el código de la tabla de multiplicar

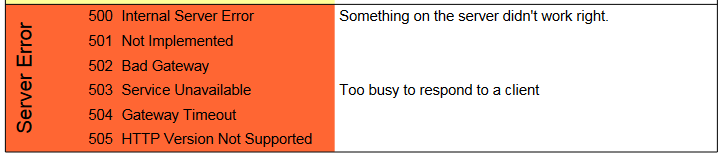


Separando nuestros archivos



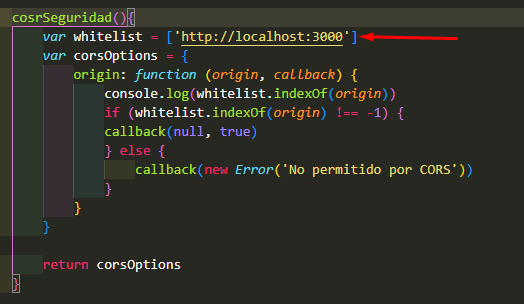




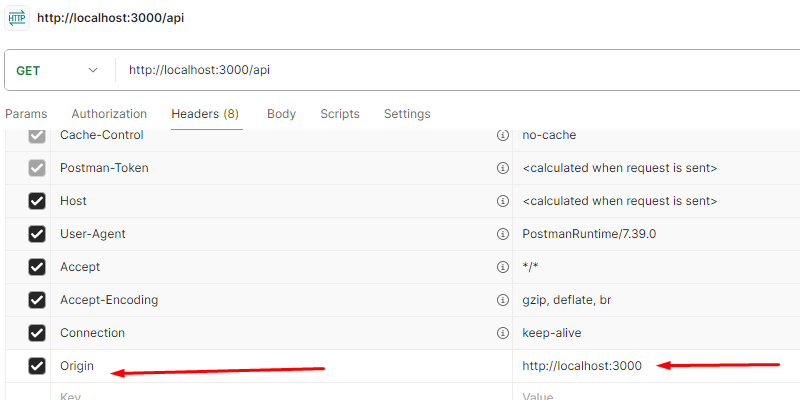


Mi configuración de cors para este ejemplo

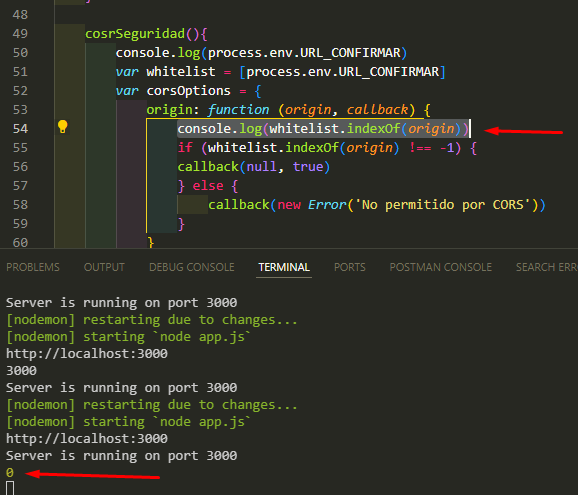




Se le coloco un origin a postman para que me permitiera enviar los datos desde el mismo server de lo contrario postman manda null y por lo tanto las pruebas de cors siempre caen en el error por que siempre muestra el valor de -1 en el origin, para lo cual tuve que ponerle un origin a postman y des esta forma ya me funciono.



Sabemos que esta funcionando correctamente cuando marca error y el origen no esta en la lista blanca y cuando si esta en la lista blanca y nos permite continuar su valor es 0



Los verbos Http involucrados en un sistema REST son GET, POST, PUT, PATCH y DELETE

GET es el que usamos para consultar un recurso. Una de las principales características de una petición GET es que no debe causar efectos secundarios en el servidor, no deben producir nuevos registros, ni modificar los ya existentes. A esta cualidad la llamamos idempotencia, cuando una acción ejecutada un número indefinido de veces, produce siempre el mismo resultado.

Esto quiere decir, que no importa cuántas veces hagamos una petición GET, los resultados obtenidos serán los mismos.

Cuando ingresamos a la dirección usando GET <https://cf.com/c/backend-profesional/> estamos solicitando que se nos entregue el recurso identificado por /cursos/backend-profesional, este es un buen ejemplo de uso con GET.

Esta otra ruta: <https://codigofacilito.com/c/recomendar?selected_level=0&category_options=28> aunque más compleja, también es correcta, estamos solicitando los recursos identificados por /cursos con las opciones de filtrado ahí indicadas. Sin importar cuantas veces hagamos esta solicitud, no modificará los resultados por sí misma.

Las peticiones con POST son sólo para crear recursos nuevos. Cada llamada con POST debería producir un nuevo recurso.

Normalmente, la acción POST se dirige a una recurso que representa una colección, para indicar que el nuevo recurso debe agregarse a dicha colección, por ejemplo POST /cursos para agregar un nuevo recurso a la colección cursos.

Si queremos crear un nuevo artículo, pudiéramos tener una URI /articulos. Lo que es importante en estos casos, es recordar que la URI no debe decir qué acción estamos ejecutando, nos olvidamos de /articulos/crear o de /cursos/agregar, etc. El verbo dice qué haremos, y la URI sobre qué recurso se harán las modificaciones.

Algunos escenarios más complejos para el uso de POST son los inicios de sesión, agregar a un carrito de compras, procesar un pago nuevo, etc. Estos ejemplos nos dejan en claro que el recurso creado por POST no es precisamente una fila en la base de datos, el recurso puede ser una sesión, un pago en una API externa, etc.

Los verbos PUT/PATCH son muy similares ya que ambos se usan para modificar un recurso existente. En la teoría, PUT se diferencía de PATCH, en que el primero indica que vamos a sustituir por completo un recurso, mientras que PATCH habla de actualizar algunos elementos del recurso mismo, sin sustituirlo por completo.

Un escenario común para el uso de PUT sería una llamada para actualizar la información de un curso, por ejemplo:

PUT /cursos/backend-profesional

O también:

PATCH /cursos/backend-profesional

En la práctica, y particularmente en Rails, ambos verbos se usan para actualizar un recurso, sin importar si lo sustituimos parcial o totalmente.

Por último, DELETE es el verbo que usamos para eliminar registros, bien pudiera ser para eliminar un recurso individual como en:

DELETE /cursos/backend-profesional

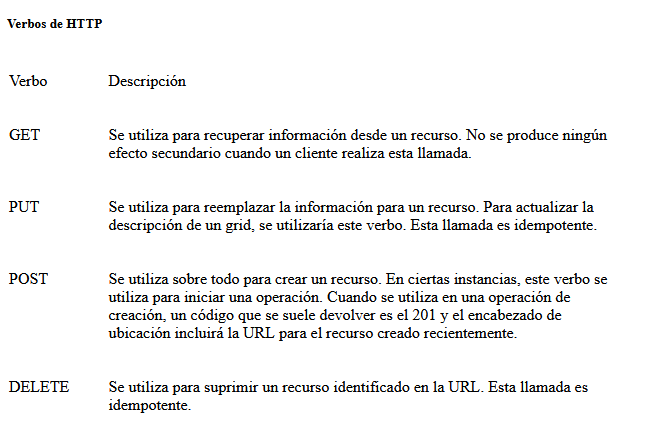
O para eliminar una colección completa:

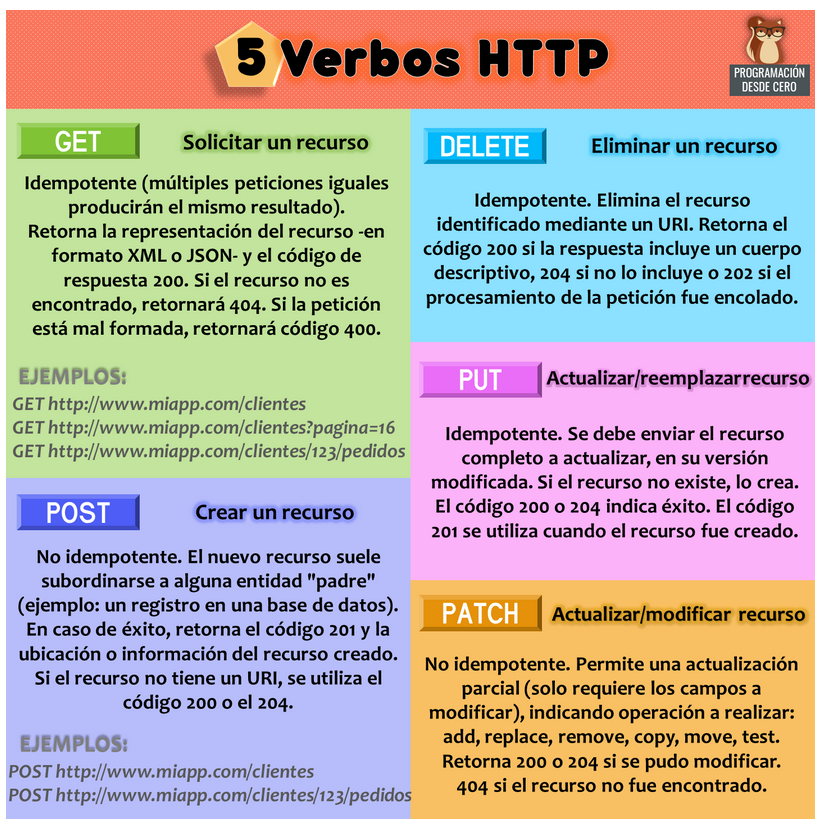
DELETE /cursos

Esta es la manera a través de la que usamos los verbos Http en una aplicación web. Estos en combinación con las URIs proveen la interfaz uniforme de la que hablamos cuando discutimos las características de un sistema REST.

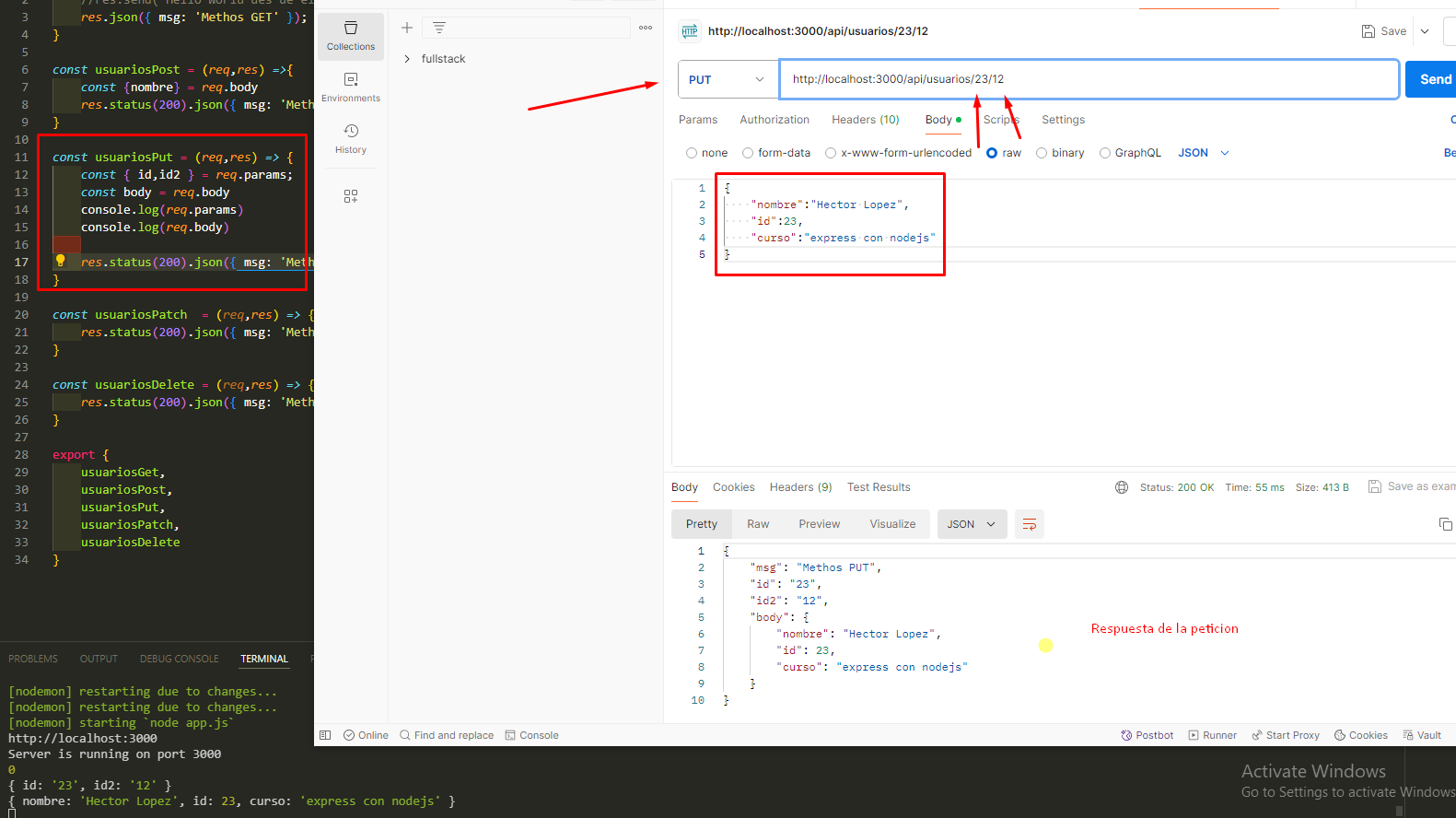
Como podrás notar, el beneficio de estas es que se establece una guía para la construcción de la aplicación, las rutas siempre representan recursos, las acciones se representan con Http.

Veamos cómo Rails usa REST en el ejemplo del scaffold, continuemos.

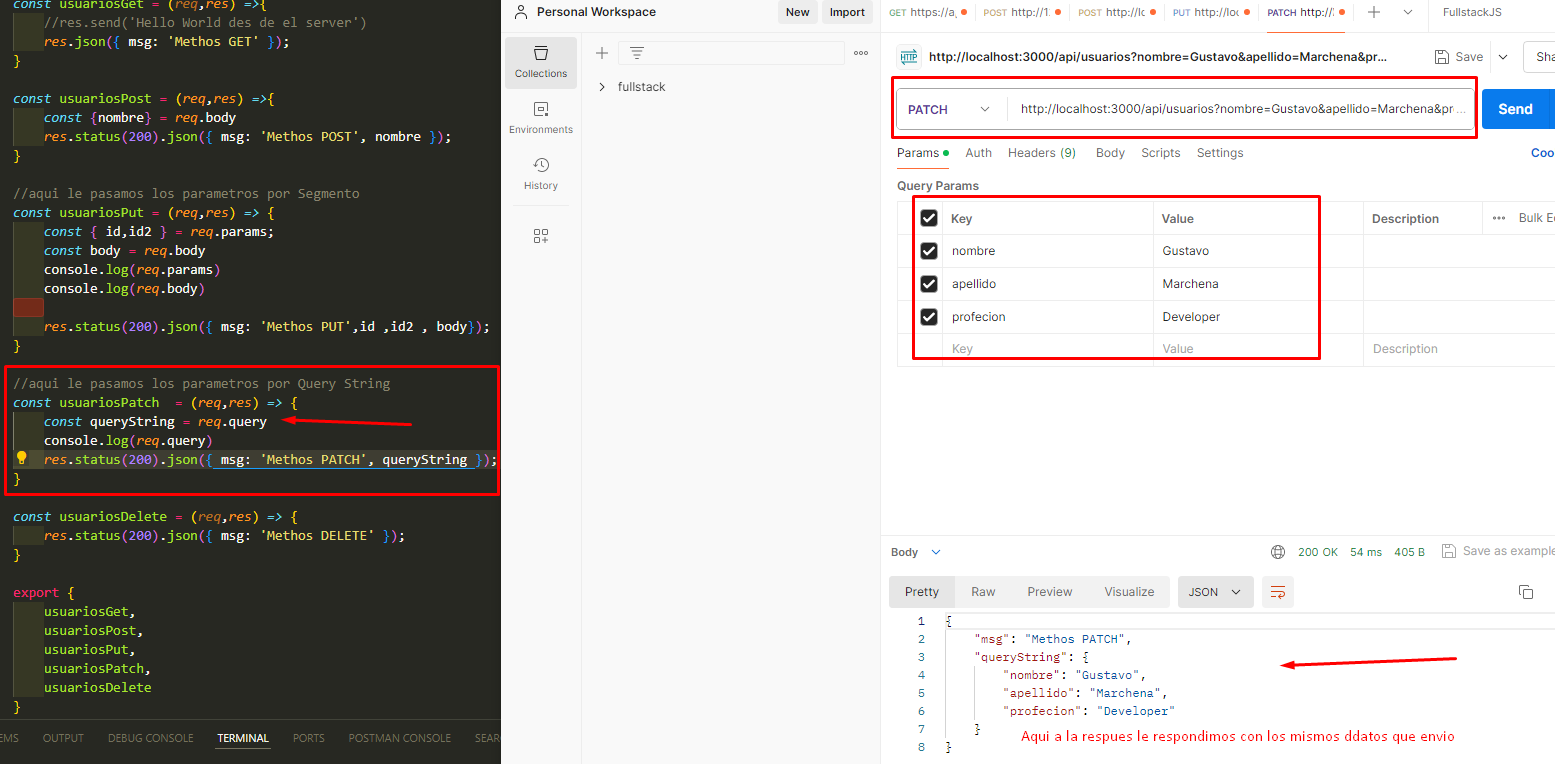




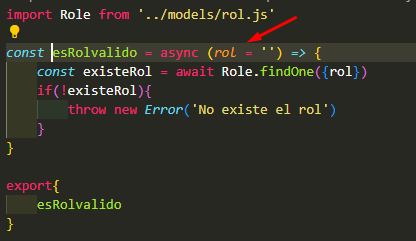
Peticion Put con parámetros via json para actualizar y con los parámetros de segmento



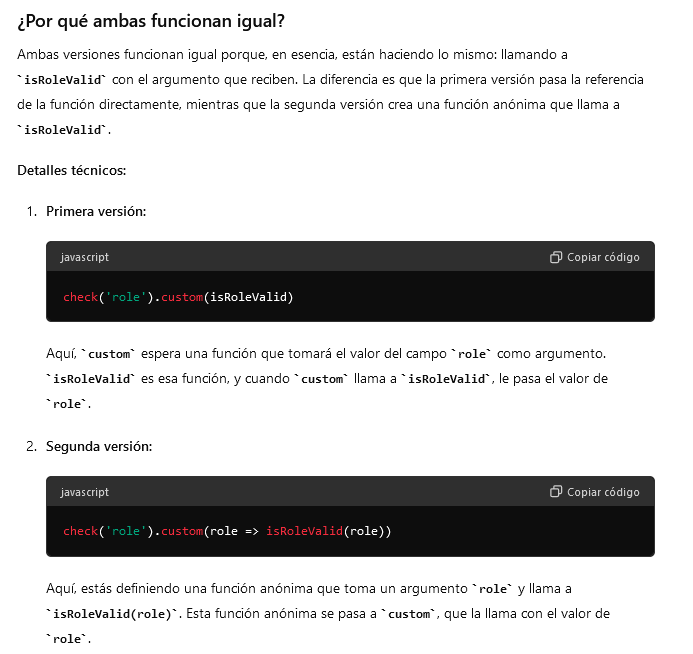
Peticion PAtch con Query String



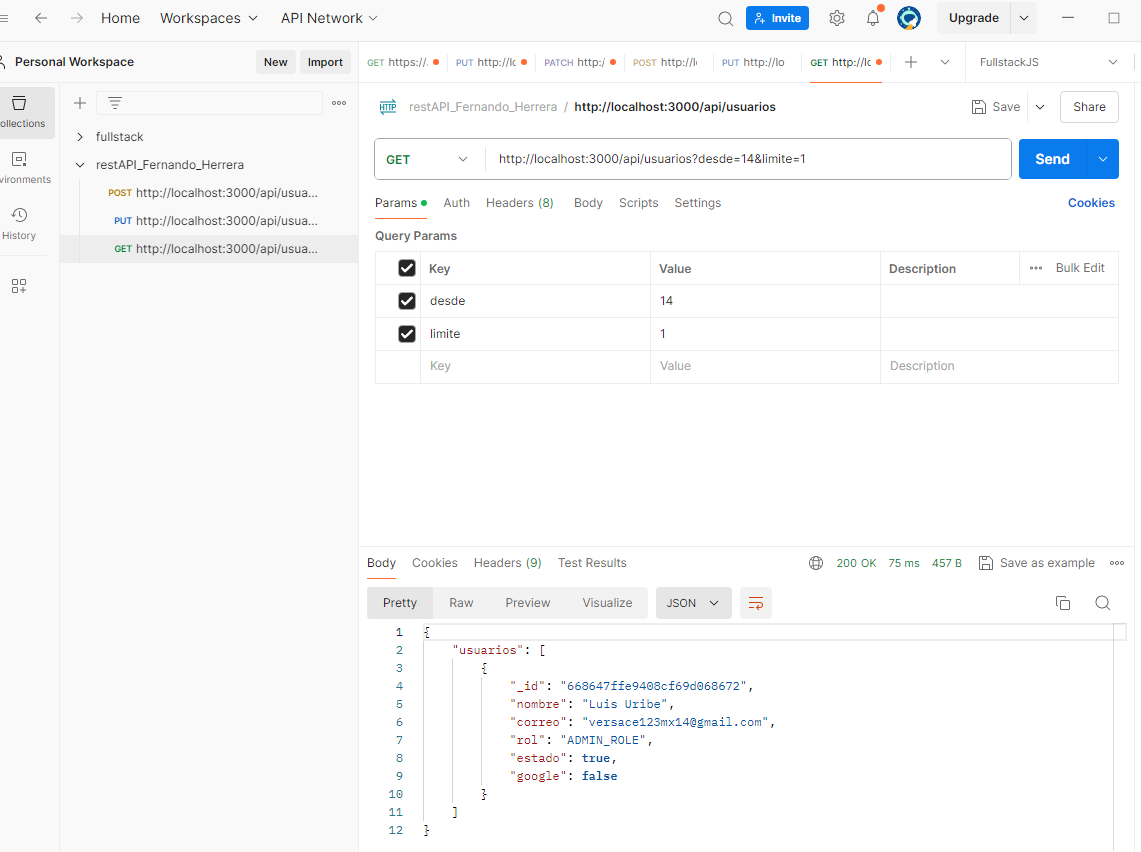




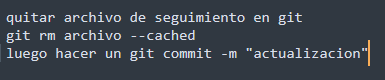




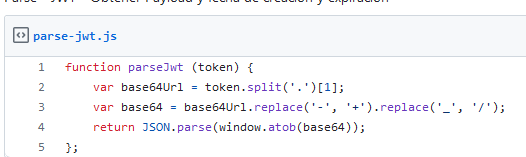


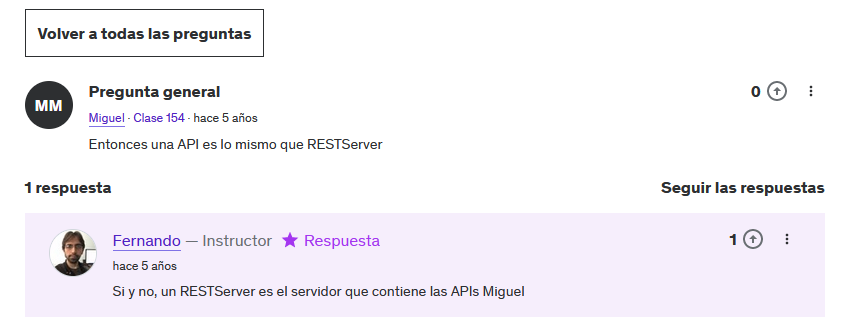


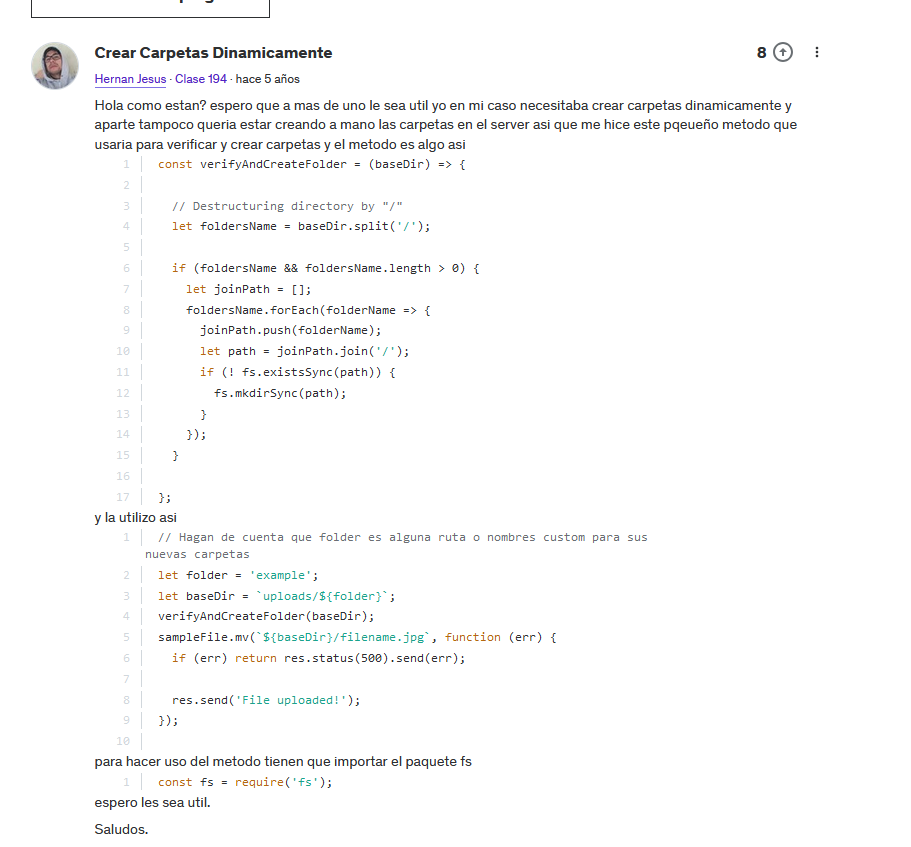
Quitar un archivo stageado



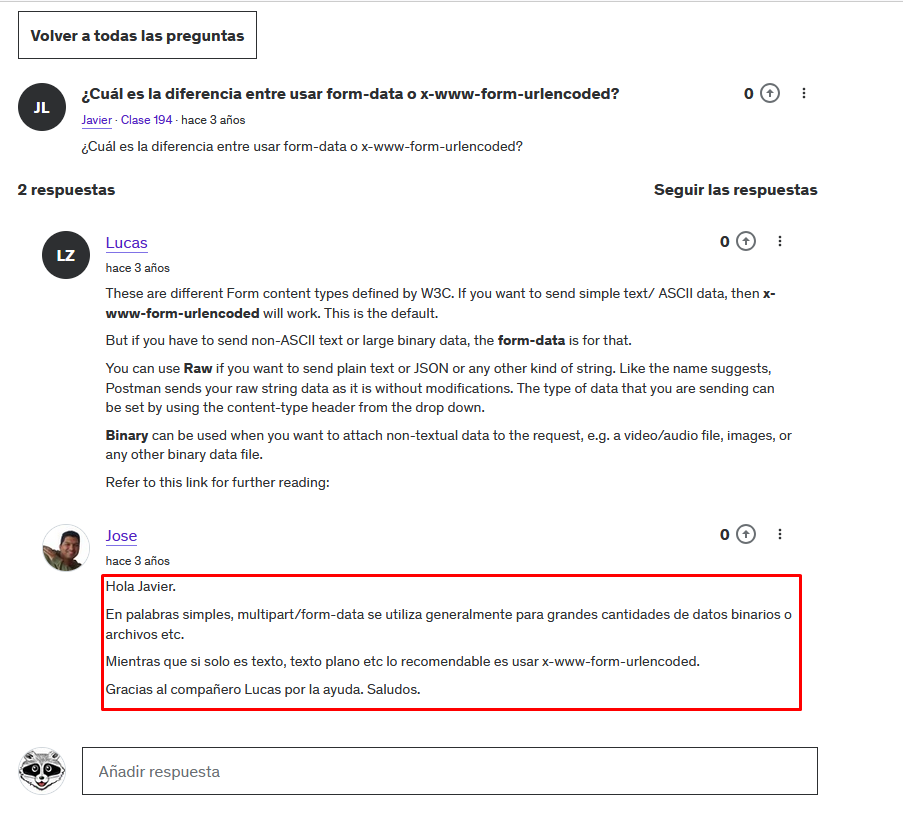
Saber el contenido de un JWT



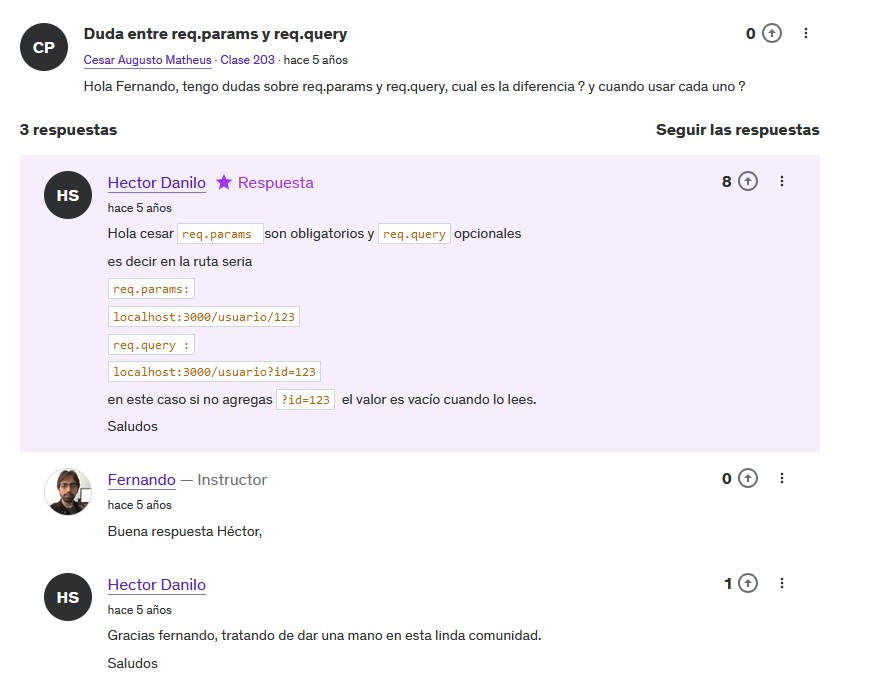












socket.io es el servidor

socket.io client es el cliente