**1.Protocolo HTTP: F12 y Network**

**2. get**

**1. query string**

**3. post**

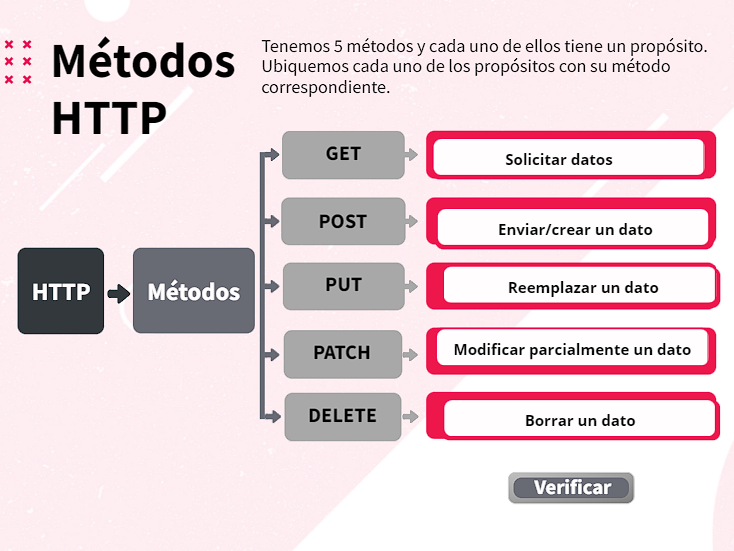
**4. put y delete**

**5. error 404**

**1.Protocolo HTTP:**

F12 y Network: vemos las peticiones que estan hacienda las paginas

protocolos de comunicación entre el cliente al servicior, método son los verbos los mas usados son:



**Get**: solcita cosas al servidor, solo recupera datos (enlaces por ejemplo) la info se ve en el url

Todo lo que se puda por el url va a ser por get

**Pos**t: envía datos al serv, crean registros en la app crea un producto, deja un comentario, etc. La info no se ve en el url

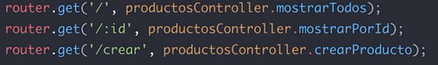
**Put y delet:**

**put** remplaza info ya presente el el serv (modifica info del usuario). **Delete** borra algo ya existente en el serv (borrar una foto).

**2.**

**Peticiones por get:**

Proces peticiones echar por el usuario, retorna algo.

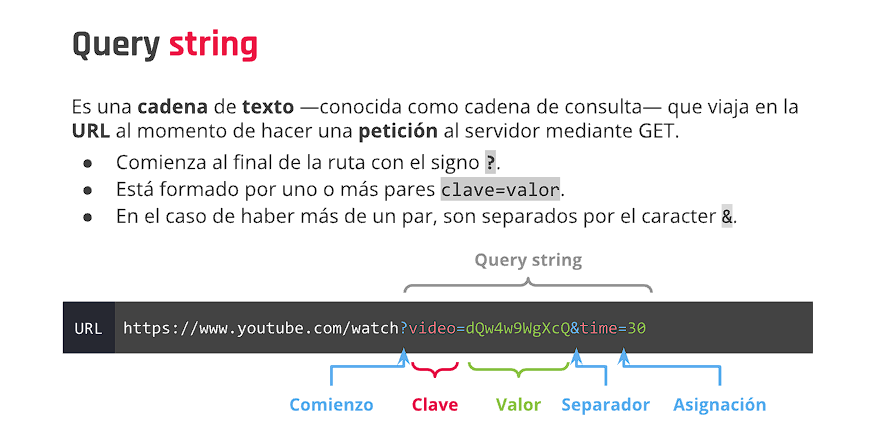


Estas rutas solo respoden a get, primer parámetro: el string es el que especifique el recurso a visitar, segundo parámetro: callback que maneja esa peticion, el método esta el en controlador (dice lo que va a hacer caundo entra en esa ruta).

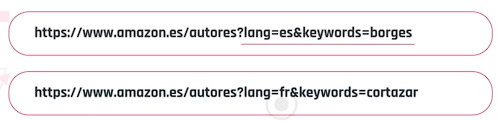
Mas comunes. Dercto de la url, un enlace.

1. **Query string**

Query string: al recibir un pedido por get, puede ser que recivamos info relevante en la URL.

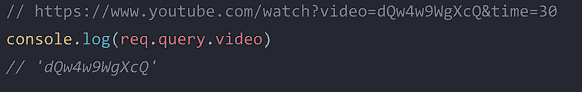
El query string contiene info dinámica que poder ser usada por el controlador para consultar ciertos datos o cosas por el estilo

Otro ejemplo:



Esta url puede ser agregada a los favoritos o marcadores del navegador, porque son **permalink,** la url guarda la configuración necesaria para mostrar info relacionada.

Para **acceder a estos datos** dentor el **callback** que aenje la petición usaremos la propiedad **query** del **objeto request**. Esta propiedad es un objeto literal con claves y valores iguales que las que viajan en la url. Siempre serán strings.



**En que casos usamos query string??** En formularios que piden info: por ejemplo un buscador.

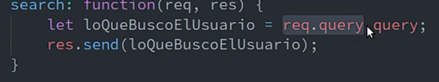
**IMPORTANTE METHOD GET, PORQUE LE ESTAMOS PIDIENDO AL SERVIDOR DATOS.**



 “user/search” es porque lo pusimos en el action, “query” porque lo pusimos en el name, y “d” porque es lo que buscó en usuario.

Todo eso aparece en la URL.

Hacemos la ruta el el router, y hacemos el search en el controles. Usamos el req.query para obtener la info que nos mando el usuriario , en este caso:



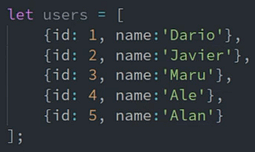
Curiosamente el name que le pusimos es query, por eso hay que ponerlo dos veces, pero si el name es pepito el código seria: req.query.pepito.

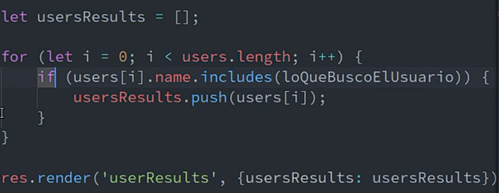
El objeto anterior(req.query) se vería así:

(si fuera req.query.”name que le hayamos puesto en el html” seria solo la “d” en este caso, no mostraría el objeto, solo el valor)

U8n objeto literal con la info, importante saber el name que pusimos porque es el que va a parecer en el objeto.

Esta info del req.query.query se puede usar para recorrer usuarios y preguntrar si el usuaro en su nombre contiene lo que busco el usuario y asi agregarlo en una array, y usar esa array en una vista:





Si busca la “e” solo le va a aparcerer los usuarios que en su nombre tenga la letra “e”

**3.post:**

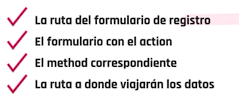
tendrán la responsabilidad de almacenar la información, sino que también garantizan que la información enviada viaje de una manera segura.

Las rutas por post al almacenar datos, es nombal que **no retornen una vista, sino que haga un redireccionamenito** a otra ruta por get

A través de un formulario enviaremos la info:

en action le indicaremos a donde enviará los datos cuando el usuario lo quiera enviar, en method usamos POST, si no se declara predeterminadamente usará get dando error.

Ya tenemos todo esto:

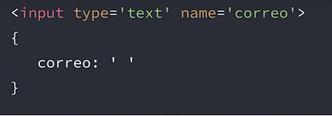


En app.js sumamos estas líneas de código: (si usamos express-generator estas líneas ya se crean)

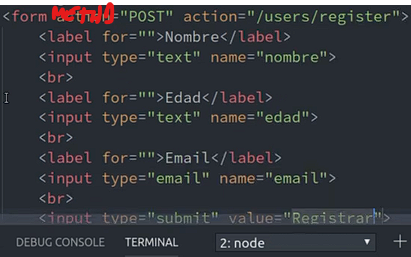


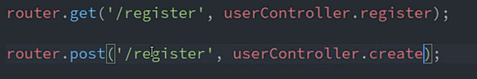
Estas líneas sirven para definir unos pasos previos de configuración. En este caso le estamos diciendo que queremos capturar lo que venga de un formulario en forma de un objeto literal y que podamos modificarlo en formato json si quisiéramos.

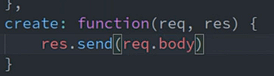
Como obteneoms esa info y la trabajamos? Con el método **req.body**: es un objetoi literl. El nombre que contenga la clave es el mismo que usamos en name.

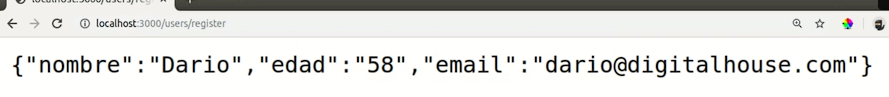
el valor será lo que el usuario puso.

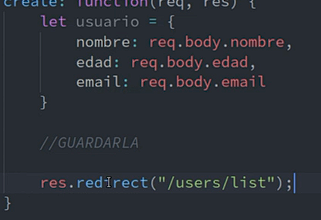
Cerrando el ciclo de controlador usamos **res.redirect(‘/’)** po ejemplo. Porque las rutas de post redireccionan a vistas, no muestran nada nuevo.

HTML

router

User controller

muestra esto

trabajmos la info asi por ejemplo

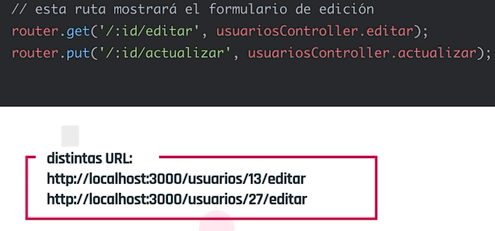
**4. put y delete:**

Parecidos al post, viene de un formulario.

Put envía info de manera segura al sevidor. rutas igual pero **dos puntos importantes**:

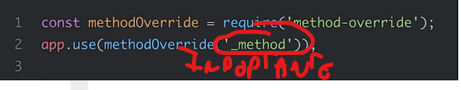
Primero: necesita un get para ver el formulario

Seguindo: pasar un parámetro en la ruta que reprecente un indicador único para ese registro que se quiere modificar, generalmente se usa :id

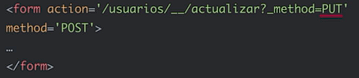


Method solo se puede poner get y post, asi que necesitamos un pasito mas para poder usar put:

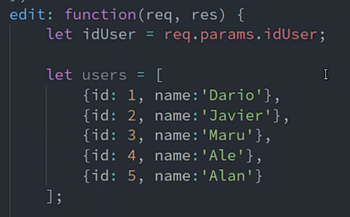
**Isntalamos** una dependecia de node **method.override** (npm install method-override –save). Luego lo requeirmos en app.js

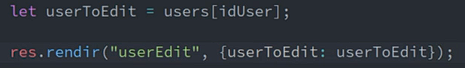


Luego en la estiquete del form en el html le agregaremos el siguiente **query string** luego de la ruta:

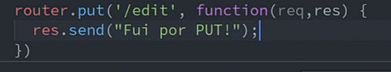


Le indicamos que usaremos put.

en el controller

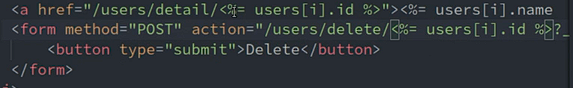


De esta forma vemos al usuario por el index de la array no por el id pero es para el ejemplo.

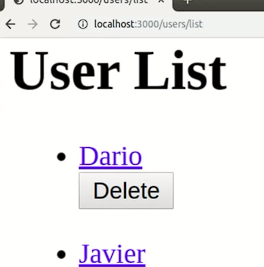
En app.js (ejemplo muy simple).

**Delete:**

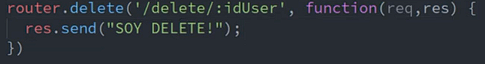
Ejemplo: (similar a put).



No se ve el query string: **\_method=DELETE”**



al hacer click en los bototes



El controller quedaría asi, simple para el ejemplo. Falta la parte de trabajar con eso. **Luego, al termianr el proceso, necesitaremos un res.redirect en los casos de put y delet** al igual que en post

**5. error 404:**

Una url que no se encuestra. Express tiene un respuesta automatizada.

app.use((req, res, next)=>{

res.status(404).render(“not-found”);

});

Hay que crear la vista not-found donde diga error 404 etc etc.

**6. Material extra: path y FS**

**Path.:**  tan usado que ya esta incoporado en express, solo hayq eu requerirlo.

.join: Une strings para hacer rutas de archivos:





.dirname: nos da el directorio del archivo:





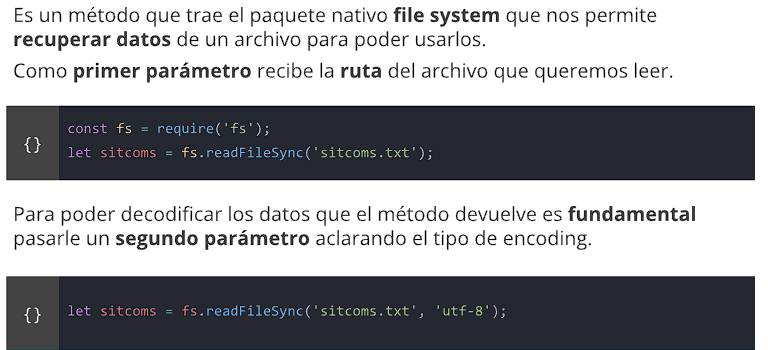
.extname: le dice que tipo de archivo es:





**FS (FileSync):** también incormporado en extres, solo se requier.

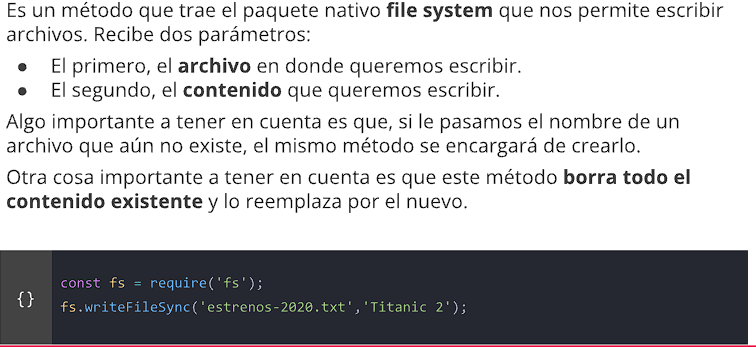
.readFileSync():

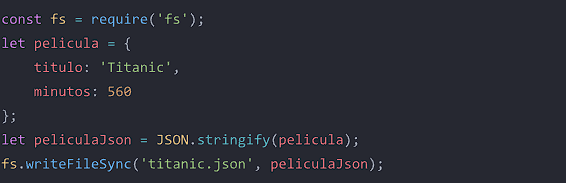


Si estamos usando un .JSON hayq eu convertirlo de un string a un objeti literal usando JSON.parce() :



.writeFileSync():

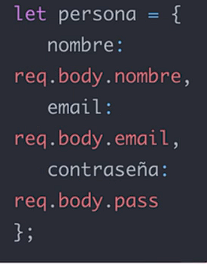


JSON.stringify(): convierte en un archivo .JSON: 

Para escribierr mucha info por ejemplo un objeto literl con los dats del ususario, primero debo combertilo a un json y ahí usar el .writeFileSync():



Si la info vieniera de un form completado por un usario el objeto literl seria algo asi, este deberías convertlo en JSON como vimos antes



.appendFileSync():

