METODOS

GET => Solicitar datos

POST => Enviar/Crear un dato

PUT => Reemplazar un dato

PATCH => Modificar parcialmente un dato

DELETE => Borrar un dato

Para hacer la petición hay 2 formas depende si usamos o no un sistema de routes

app.put()

o

routes.put()

POST **=> IMPORTANTE**: Para poder trabajar con los datos que se envían desde el formulario es necesario configurar el entorno de nuestra aplicación, para que sea capaz de capturar esa información. Si estamos trabajando con express-generator, esta configuración se creara por defecto durante la instalación, en caso contrario tenemos que agregar estas 2 líneas en el archivo app.js

app.use(express.urlencoded({ extended: false }));

app.use(express.json());

**IMPORTANTE: estas 2 líneas deben ir en el archivo que se encuentren las rutas!!**

-formulario, queremos capturarlo en forma de un OBJETO LITERAL , y a su vez, tener la posibilidad de convertir esa información en un formato JSON, en caso de necesitarlo.

req.body => nos devuelve un objeto literal que contendrá toda la información del formulario:

* El nombre de cada clave de ese objeto, será el nombre del atributo name de cada input del formulario
* El valor será el dato que se haya ingresado en ese campo.

Ej:

(en HTML)

<form method= “POST” action= “/película/crear”>

Titulo: <input type= “text” name= “titulo” value= “Batman”>

</form>

(en routes o router { })

router.post(‘/película/crear’, (req, res) => {

console.log(req.body) }); //titulo: Batman

res.redirect() => Para cerrar el ciclo del req y res que hace el servidor es necesario hacer un redireccionamiento después de implementada la lógica, usando el método redirect() sobre el response. En el caso de la película anterior, luego de la acción de crear podríamos llevar al usuario al listado de películas completas, ej:

router.post(‘/película/crear’, (req, res) => {

//procesamos la información enviada por el usuario y si todo sale bien, luego lo redireccionamos al listado de películas.

res.redirect(‘/peliculas’)

})

PUT y DELETE:

para poder usar PUT y DELETE en el navegador, hay que ejecutar el comando: npm i method-override --save

después agregar estas 2 líneas en nuestro archivo app.js o de rutas:

cosnt methodOverride = require('method-override')

app.use(methodOverride('\_method'));

ó

router.use(methodOverride('\_method'));

CONFIGURACION DE FORMULARIO PARA PUT Y DELETE

Para terminar de configurar los métodos tenemos que agregarle el query string al formulario:

?\_method=PUT

?\_method=DELETE

De esta forma indicamos, sin importar el método original que tenga seteado el formulario, que queremos recibir la información usando PUT o DELETE, ej:

<form method=“**POST**” action=“/película/:id/editar?\_method=PUT”>

(detalle importante se envia por método POST y se cambia el método en el action)

<form>

ERROR 404

Se declara en el app.js o el router.js según como lo estemos usando y el código es el siguiente:

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

res.status(404).render(‘not-found’)

})

De esta manera si hay un error 404 (que no se encontró la página), podríamos redireccionarlos a una vista mas amigable que hayamos creado, en este ejemplo sería “not-found”

**FS**

fs.writeFileSync(‘bienvenida.txt’, ‘Hola Mundo’) => Para escribir un archivo en file system, el primer parámetro indica el nombre del archivo en el que se escribirán los datos, el segundo indica lo que queremos escribir.

También podríamos convertir un archivo de objetos en un archivo json y enviarlo como file Sync de esta manera, ej:

let persona = {

nombre: ‘Roberta’,

email: ‘roberta@gmail.com’,

contraseña: ‘minijuegos’

};

let personaJSON = JSON.stringify(persona);

fs.writeFileSync(‘persona.json’, personaJSON);

IMPORTANTE: combinando fs.writeFileSync y JSON, podremos guardar mucha información de manera dinámica, esto lo podría utilizar, por ejemplo en el carrito de compras, para intentar transformar el archivo pructos.js en un JSON y así cargar el archivo desde por ejemplo una vista en el navegador, en vez de hacerlo manual como lo tenía hecho, obviamente habría que utilizar los métodos GET POST PUT y DELETE vistos anteriormente.

En este caso el armado del objeto literal sería algo así:

let persona = {

nombre: req.body.nombre,

email: req.body.email

constraseña: req.body.contraseña

};

fs.writeFileSync() = pisa el archivo, bloquea el resto de la ejecución de tu archivo

fs.appendFileSync() = agrega contenido al final del archivo, importante la fierencia entre este y fs.writeFileSync().

fs.writeFile() = es como fs.writeFileSync(), pero, writeFile **es asincrónica**, no bloquea el resto de la ejecución de tu archivo

let texto = fs.readFileSync()= lee un archivo FS

fs.readFileSync('leerUnTxt.txt', {encoding: ‘utf-8’}

); 🡨 indicamos el archivo que queremos leer, {encoding: "utf-8"}

Transforma el código a lenguaje utf-8(texto), es importante guardar el resultado en una variable ej:

const contenido = fs.readFileSync(‘leerUnTxt.txt’); 🡨 de esta manera podemos reutilizar el contenido para una posible “BASE DE DATOS”.