1. Sesión
2. Cookies
3. Hashing (encriptación de datos)
4. Notas clases

**1.**

Session: es una variable “global”/”magica” que esta accesible siempre para la app y nos permite guardar y compartir info y usarla en todas las vistas.

Un usuario X quiere pone la pagina en español, entoence en el servidor al usuario lo identifica con un número, por ejemplo, y recuerda esa info durante toda la paginas. Cuando el usuario cierra la ventana se elimina toda se borra.

Como implementamos session en express:

**npm install express.session**

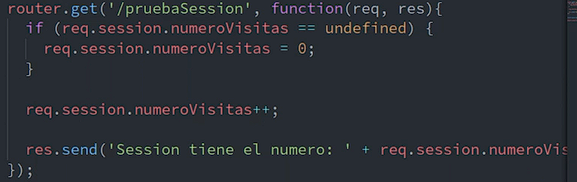
**La requerimos: ** en app.js

Lo hacemos middelware “global” usamos sesión y le pasamos un objeto literal con la propiedad secret, el texto indicara que es infomacion impostante y necesita seguridad.

En un controlador se usaría asi en el caso del idioma



Ejemplod e un contador de visitas:



**Crear un login:**

Se crea ruta de login



En el controlador hacemos el método login que retorna una vista llamada login:



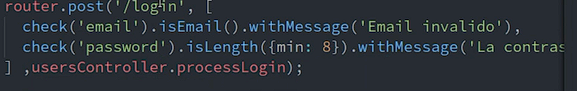
Hacemos una vista simple:



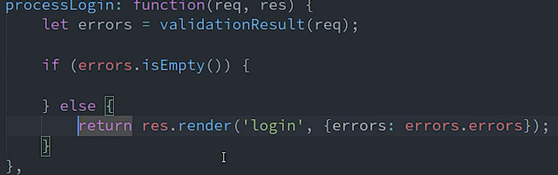
Hacemos la rutoa del login por post:



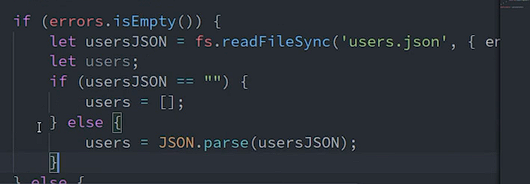
Aquí podemos hacer unas validaciones:



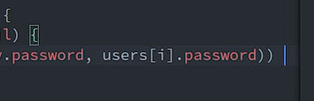
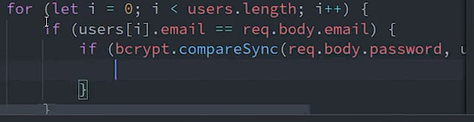
Hacemos el método processLogin con su varaible por si hay errores:



Agregamos código para buscar en la base de datos y a EL usuario en cuestión:

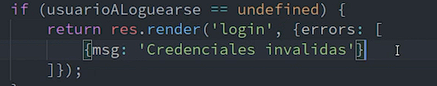


If preguntando si el email de cada uno es igual al que entro por el formulario, además si la contraceña tambine es la misma



 y abajo break

Abajo otro if por si el usuario no se encontró:



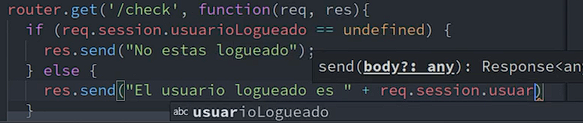
Después de todo esto si llegue al final y tengo un usuario a logearce lo voy a queres guardar en session:



Abajo un res.render a la vista deseada:



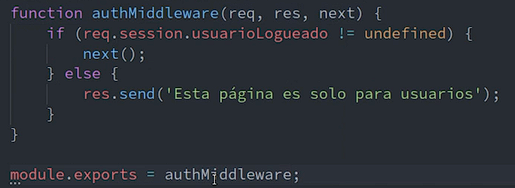
**Ahora creemos una ruta para saber si el usuario está logueado**:

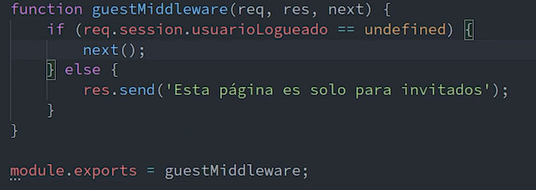


\*ususarioLogueado.email

Ahora que ya están logeado, no podes registrarte otravez o capas que tenes mas cosas para ver en la pagina.

Creamos mideelwares para chequear si el usuario esta logueado (authMiddleware) y otro para ver si no esta logueado (guestMiddleware)





**2.**

**Cookies:** es un archivo que se guarda del lado del cliente, es decir, en el navegador. Y cuyo objetivo principal es guardar información NO SENSIBLE que podamos recuperar fácilmente desde nuestra aplicación.

Como dijimos la session se borran cuando cierran el navegador. Las cookies podemos decirles cuando queremos que se borren.

¿Cómo se implementa?

Primero lo instalamos

npm install cookie-parser

Si estamos en un proyecto creado por express generator ya va a estar requerido.

Requerimos cookieParser

var cookieParser = require('cookie-parser')

Y lo pasamos de forma global a la app



**las cookies viajan por RES Y CORRER EL MÉTODO .COOKIE y debemos aclaramos el nombre de lo que estamos guardando y el valor:**

****  en este ejemplo estamos guardadno una cookie llamada colorPreferido con el valor “rojo”

**PARA LEER LA COOKIE EN OTRO PEDIDO:** usamos el **req** usado el método **cookies** y ”nombre de la cookie”.



En el login, por ejemplo, para que no se deslogee cuando se cierra el navegador, marcamos el check box de recordame, pasando esa info a las cookies

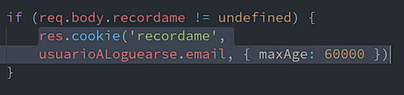




En el controlador hacemos un if verificando si se tildo ese lugar:

si no está tildado viene como undefined\*

Con el res creamos la cookie le damos un nombre y un valor. Po ultimo un ultimo parámetro que es un objeto literal con configuraciones: maxAge es para decirle cuando tiempo va a duara la cookie. (60000 milisegundos).



Hacemos un middelware para avisarle a todas las vistas si el usuario esta logueado o no

recordameMiddleware.js, donde mirara si hay cookies y si hay una llamada recordame. Si es asi le pasara la información al session avisándole de que hay alguien logeado.

**3.**

**Hashing (encriptación de datos)**

Encripta las contraseñas, tales como la siguiente:

Contraseña: "holaMundo123"

Encriptación: "4c35a8ddbc285e0aa079041b30c5944e96edf36a"

La encriptacion no sirve para nada más que para compararla con la contraceña original y saber si son la misma.

Primero hay que instalar: 

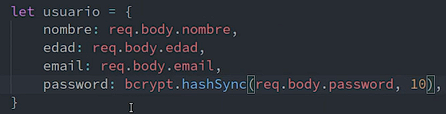
Y luego lo requerimos en el archive que lo vayamos a usar 

Para encriptar un string usamos la funsion hashSync. Y le pasamos dos parámetros. El primero el string a encriptar, y el segundo la cantidad de sal o de rondas que queremos que use (que tan complicado va a ser el encriptado, por lo generema se usa 10 a 12):

.

Ejemplo de un formulario:

En el controlador que recibe los datos del registro del usuario hacemos esto:



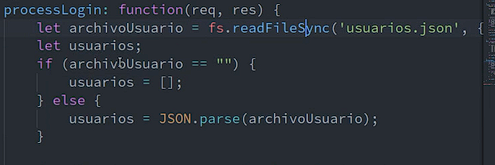
Luego viene la función que lo agrega en la DB

Ahora, con la función compareSync() podremos comparar el encriptado con la contraseña original y si son iguales nos dará true, de no ser así false

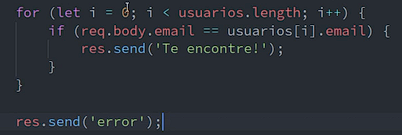
Primero el string de la contraseña original y luego el hash almacenado

Implementación real:

El controller que se encarga del login primero obtiene la DB de ususarios. Si no hay obtiene un array vacío y si hay usuarios registrados nos descomprime el JSON, optenieno una array con todos los usuarios registrados.



Con un for recorremos todos los usuarios, y con if verificamos si el usuario es correcto. Verificando el email que nos llega con el email del JSON



Y si la contraseña es válida, comparando la que nos llega con la del JSON:

&& bccript.compareSync(re.body.password, usuario[i].password)

El login