

Esta clase va a ser

- grabada
- a

Clase 19. PROGRAMACIÓN BACKEND

Cookies, Sessions & Storage II

Temario

18

Cookies, Sessions & Storage I

- ✓ Cookies
- ✓ Dando identidad al cliente: sessions

19

Cookies, Sessions & Storage II

- ✓ [Storage](#)
- ✓ [Session Storage con Mongo Atlas](#)
- ✓ [Acercamiento a nuestro primer login](#)

20

Autorización y autenticación

- ✓ Autenticación y Autorización
- ✓ Estrategias de autenticación: Passport

Objetivos de la clase

- Comprender los métodos de almacenamiento de sesiones
- Utilizar el almacenamiento de sesiones en un sistema de login

Glosario

Cookie: Pequeña parte de información que se almacena en el navegador al momento de visitar un sitio web, utilizado para conservar detalles de información útiles para el sitio web.

CookieParser: Módulo de Node que permite gestionar cookies desde el servidor, con el fin de poder obtenerlas, leerlas, escribirlas o eliminarlas.

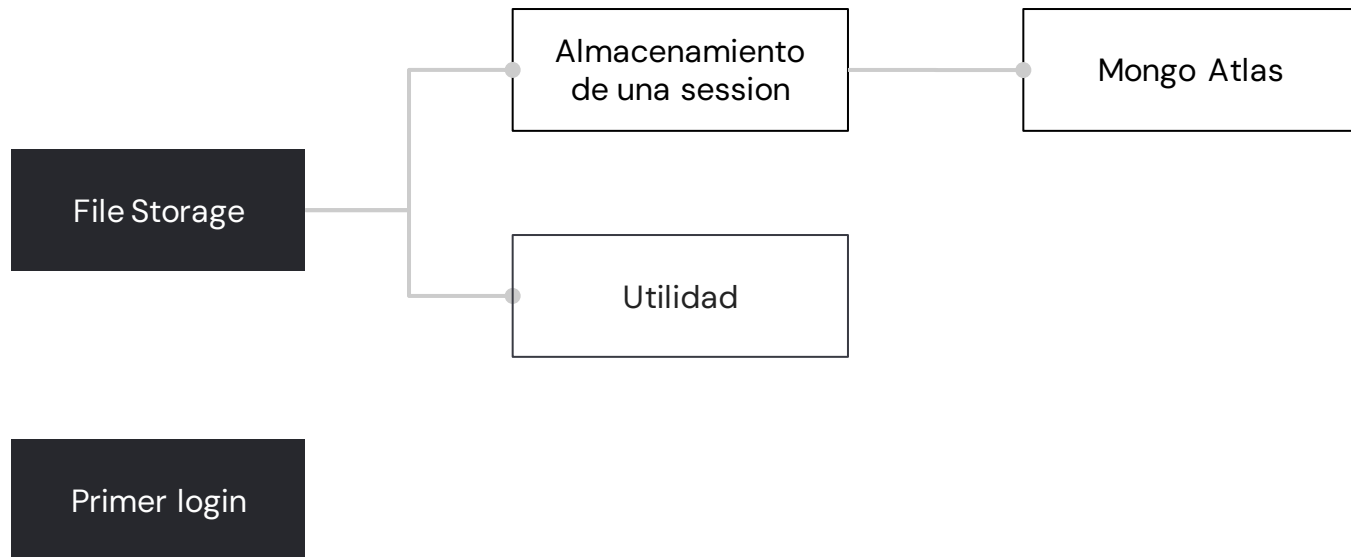
Cookie.maxAge: Indica en milisegundos el tiempo de vida de la cookie. Si no se especifica, la cookie persistirá hasta que se elimine

Signed Cookie: Cookie firmada que permite agregar seguridad, la cual escucha si la cookie fue alterada, invalidándose en dado caso.

Session: Ciclo de interacción entre un sitio y un cliente, donde el cliente cuenta con un identificador de sesión para reconocimiento desde el servidor.

express-session: Módulo de Node que permite gestionar sesiones y almacenarlas en el servidor.

MAPA DE CONCEPTOS



Storage



PARA RECORDAR

Session

Una sesión es un vínculo que se genera cuando el cliente conecta con un servidor, este vínculo se representa por una **sessionId**, la cual se guarda en el navegador como identificador de la sesión. Sin embargo... **¿Dónde se guarda la sesión en el servidor?**

Memory storage

El almacenamiento de una sesión por memory storage es exactamente igual a la persistencia en memoria que trabajamos la clase pasada. Es un almacenamiento muy peligroso, ya que **si el servidor muere o se reinicia, la sesión morirá con él y no habrá forma de recuperarla.**

Servidor

Servidor muere por un momento

Servidor vuelve a funcionar



Sessions

Sessions desaparecen con él

Sessions no vuelven y deben generarse desde 0



¿Cómo solucionar el problema de memory storage?

Una vez entendido el carácter de riesgo de almacenar una sesión en memoria, nos surge la duda: ¿cómo podríamos hacer persistir estas sesiones para evitar cualquier tema en el reinicio o caída repentina de un servidor?

Seguramente tienes clara la idea, ya que lo hiciste con tus productos y tu carrito: Vamos a utilizar **File Storage**, es decir, guardar nuestra sesión en un archivo, para poder consultar las sessions de un lado diferente a la memoria del servidor.

File Storage

File storage

Al igual que las primeras clases del curso, el File storage permitirá dar una persistencia de archivos a las sesiones que trabajamos, esto hará que el servidor pueda tomarlas de algún lado en caso de que algo inesperado llegase a pasar (un reinicio, o una caída).

Así, los usuarios pueden seguir haciendo consultas con sus sessionid, y el servidor podrá consultarlas del archivo que persiste en éste.

Es un recurso simple y que tiene sus defectos, sin embargo, es una solución sencilla para resolver el asunto de sesiones, sin tener que ocupar recursos de terceros.

File storage

Servidor vuelve a funcionar, toma las sessions de los archivos y no afecta a los usuarios

Servidor

Servidor guarda las sessions en archivos

Servidor muere por un momento

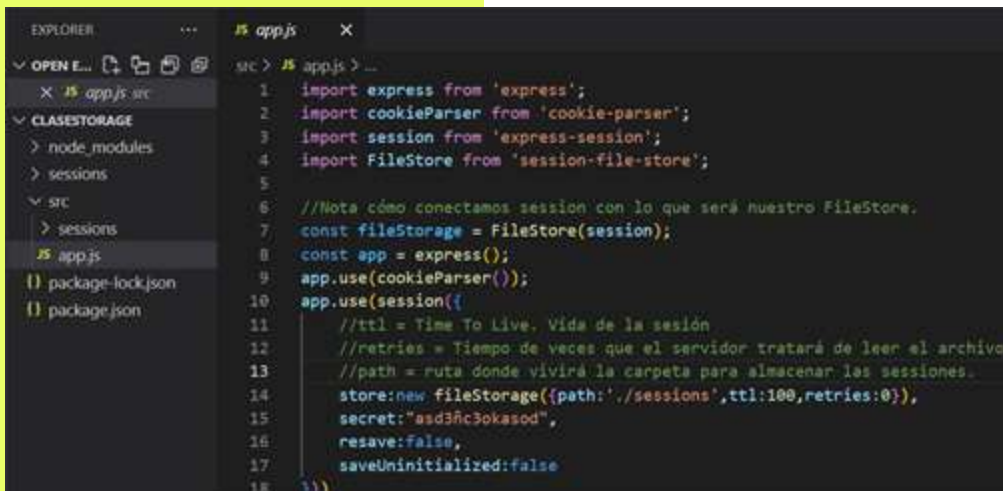


Sessions

Sessions persisten debido a que ahora están en un archivo



Utilizando File Storage en nuestro servidor.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with 'src' and 'sessions' folders. The code editor shows the following code in 'app.js':

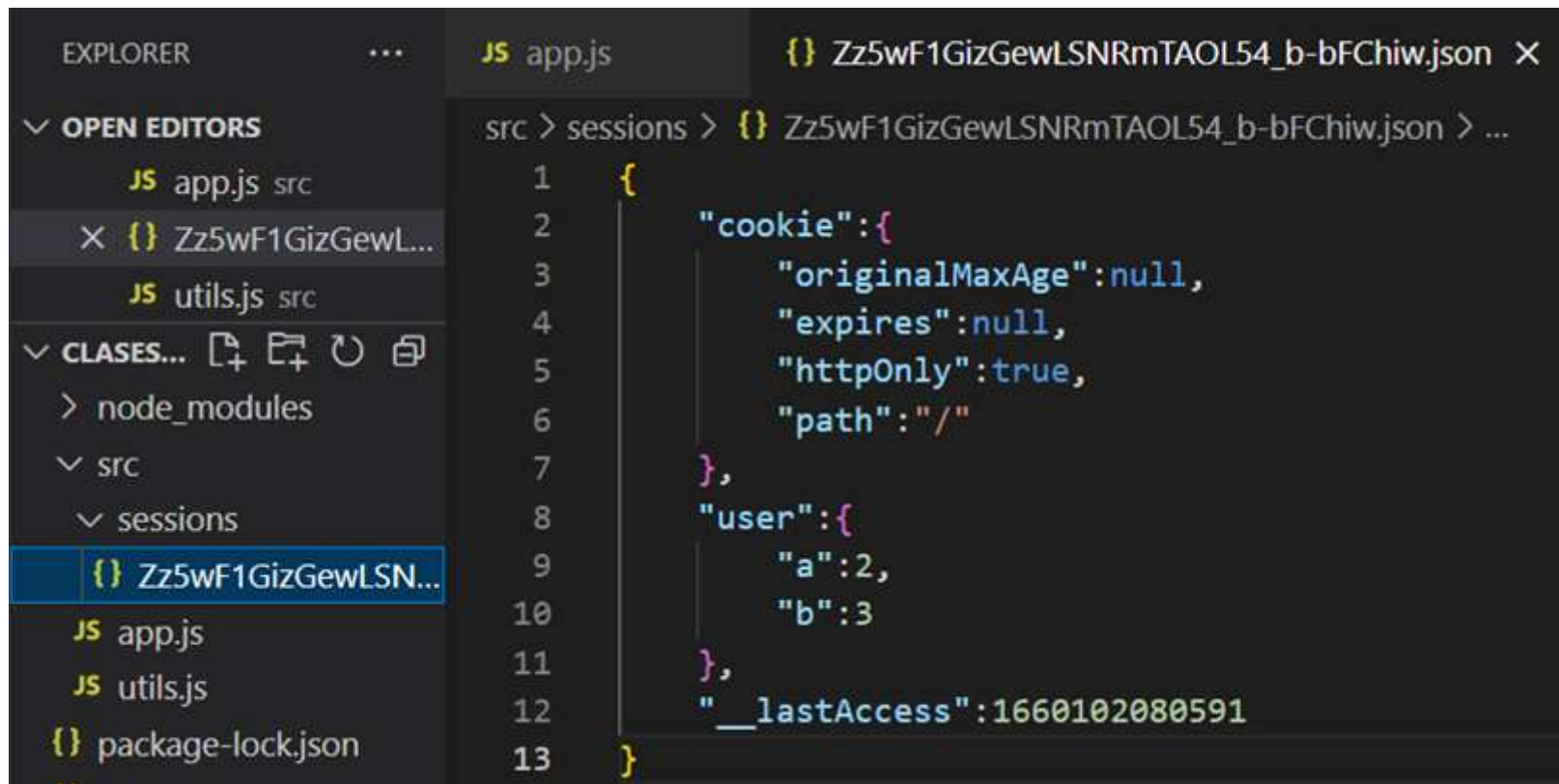
```
1 import express from 'express';
2 import cookieParser from 'cookie-parser';
3 import session from 'express-session';
4 import FileStore from 'session-file-store';
5
6 //Nota cómo conectamos session con lo que será nuestro FileStore.
7 const fileStorage = FileStore(session);
8 const app = express();
9 app.use(cookieParser());
10 app.use(session({
11   //ttl = Time To Live. Vida de la sesión
12   //retries = Tiempo de veces que el servidor tratará de leer el archivo
13   //path = ruta donde vivirá la carpeta para almacenar las sesiones.
14   store: new fileStorage({path: './sessions',ttl:100,retries:0}),
15   secret: "asd3ñc3kasod",
16   resave:false,
17   saveUninitialized:false
18 }));
```

Partiremos de hacer la instalación habitual con npm con el comando:

```
npm install session-file-store
```

Posteriormente, inicializamos el fileStore como lo indica la imagen, esto utilizando los tres argumentos principales:
path, ttl, retries.

Al final, se generará un archivo con la información indicada



```
EXPLORER  ...  JS app.js  {} Zz5wF1GizGewLSNRmTAOL54_b-bFChiw.json X

v OPEN EDITORS
  JS app.js src
  X {} Zz5wF1GizGewL...
  JS utils.js src
v CLASES... [+ [- ↺ [-
  > node_modules
  v src
    v sessions
      {} Zz5wF1GizGewLSN...
  JS app.js
  JS utils.js
  {} package-lock.json

src > sessions > {} Zz5wF1GizGewLSNRmTAOL54_b-bFChiw.json > ...
1  {
2      "cookie":{
3          "originalMaxAge":null,
4          "expires":null,
5          "httpOnly":true,
6          "path":"/"
7      },
8      "user":{
9          "a":2,
10         "b":3
11     },
12     "__lastAccess":1660102080591
13 }
```

A considerar

Utilizando File Storage podemos notar que:

- ✓ Aun cuando reiniciemos el servidor, la sesión persistirá en el tiempo indicado
- ✓ Cuando llegue a expirar una sesión, el servidor generará un nuevo archivo con la consulta, indicando el nuevo sessionId.
- ✓ Los archivos viejos no se limpian automáticamente, quedan en un cementerio de registros, lo cual puede generar complicaciones futuras.



Session Storage con Mongo Atlas

El siguiente paso para guardar sesiones

¡Seguro ya lo habías pensado! Si FileSystem terminará generando conflictos, ¿podría solucionarse con bases de datos, como hicimos con nuestros carritos y productos? La respuesta es: sí, ¡y en qué forma!

Session puede trabajar de la mano con MongoDB y MongoAtlas para poder guardar una sesión en una base de datos, esto permitirá que las sesiones tengan una gestión más limpia, además de poder contar con autoeliminación de sesiones expiradas.



Database storage



Lo primero será contar con nuestra connection string

Recordamos que, en Mongo Atlas, dentro del cluster que habíamos levantado anteriormente, tenemos un botón llamado



Mismo que nos llevará al connection string que utilizaremos en nuestro servidor:

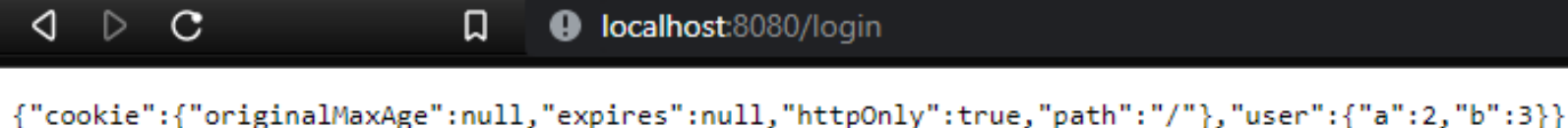
```
mongodb+srv://CoderUser:<password>@codercluster.w5adegs.mongodb.net/?  
retryWrites=true&w=majority
```

Nuestra configuración de session requerirá un nuevo módulo

```
npm install connect-mongo
```

```
import MongoStore from 'connect-mongo';
import __dirname from './utils.js';
const app = express();
app.use(cookieParser());
app.use(session({
  store: MongoStore.create({
    mongoUrl: "mongodb+srv://CoderUser:123@codercluster.w5adegs.mongodb.net",
    mongoOptions: {useNewUrlParser: true, useUnifiedTopology: true},
    ttl: 15,
  }),
  secret: "asd3ñc3okasod",
  resave: false,
  saveUninitialized: false
}));
```

La petición de login se hace normal:



A screenshot of a web browser's developer console showing a JSON response from a login endpoint. The response is: `{"cookie":{"originalMaxAge":null,"expires":null,"httpOnly":true,"path":"/"},"user":{"a":2,"b":3}}`. The browser's address bar shows `localhost:8080/login`.

Y la sesión se almacenará en la base de datos



La maravilla del expires

El objeto sesión almacenado en la base de datos cuenta con un "expires". Éste está basado en el ttl definido y se encargará de limpiarse automáticamente una vez que pase el tiempo de expiración. Esto

```
_id: "f-1mtas-16gmSwnSkxirQBzf9SP_VOZ"  
expires: 2022-08-10T10:49:24.473+00:00  
session: {"cookie":{"originalMaxAge":null,"expires":null,"httponly":true,"path"...}}
```





Pruebas de sesiones

Duración: 10 min



ACTIVIDAD EN CLASE

Pruebas de sesiones

Con base en un servidor de express

- ✓ Desarrollar un sistema de storage para Mongo en Atlas y realizar pruebas siguientes:
 - Coloca el ttl = 15. Analiza el comportamiento de la sesión si hago peticiones constantes al endpoint de login. Revisa el tiempo de expiración.
 - Retira el ttl de la configuración y genera una nueva sesión. Analiza el tiempo de expiración que se genera por default



Break

¡10 minutos y volvemos!

Acercamiento a nuestro primer login



Hands on lab

En esta instancia de la clase **repasaremos** algunos de los conceptos vistos en clase con una aplicación

¿De qué manera?

El profesor demostrará cómo hacerlo y tú lo puedes ir replicando en tu computadora. Si surgen dudas las puedes compartir para resolverlas en conjunto de la mano de los tutores.

Tiempo estimado: **35-45min**



Primer login por formulario

¿Cómo lo hacemos? **Se levantará un sistema de login completo utilizando router + motor de plantillas Handlebars + base de datos para usuarios y sesiones +**

- ✓ Se deberá contar con una estructura de router para sessions en `/api/sessions/` el cual contará con métodos para registrar a un usuario y para su respectivo login
- ✓ Se deberá contar además con un router de vistas en la ruta base `/` para llevar al formulario de login, de registro y de perfil.



HANDS ON LAB

- ✓ El formulario de registro insertará en la base de datos el usuario. El cual deberá contar con:
 - first_name
 - last_name
 - email
 - age
 - password
- ✓ Se debe contar con el formulario de login el cual corroborará que el usuario exista en la base, y además genere un objeto user en req.session, indicando que puede utilizar la página.
- ✓ Agregar validaciones a las rutas de vistas para que, si aún no estoy logueado, no pueda entrar a ver mi perfil, y si ya estoy logueado, no pueda volver a loguearme o registrarme.
- ✓ En la vista de perfil, se deben arrojar los datos **no sensibles** del usuario que se haya logueado.



Reajustando autorización

Duración: 15min



Reajustando autorización

Con base en el proyecto realizado en el Hands on lab:

- ✓ Cambiar la validación de rutas por middlewares de rutas públicas o privadas.
 - Las rutas públicas deben regresar siempre a la pantalla de login en caso de que no se reconozca una session activa.
 - Las rutas privadas deben regresar siempre a la pantalla de profile en caso de que haya una sesión activa en session.
- ✓ Realizar un botón "logout" en la vista de perfil, que permita destruir la sesión y redireccionar a la vista de login.



Implementación de login.



Implementación de login

Consigna

- ✓ Ajustar nuestro servidor principal para trabajar con un sistema de login.

Aspectos a incluir

- ✓ Deberá contar con todas las vistas realizadas en el hands on lab, así también como las rutas de router para procesar el registro y el login.
- ✓ Una vez completado el login, realizar la redirección directamente a la vista de productos.
- ✓ Agregar a la vista de productos un mensaje de bienvenida con los datos del usuario
- ✓ Agregar un sistema de roles, de manera que si colocamos en el login como correo adminCoder@coder.com, y la contraseña adminCod3r123, el usuario de la sesión además tenga un campo
- ✓ Todos los usuarios que no sean admin deberán contar con un rol "usuario".
- ✓ Implementar botón de "logout" para destruir la sesión y redirigir a la vista de login



Implementación de login

Formato

- ✓ Link al repositorio de Github sin `node_modules`

Tu código se testeará a partir [de este documento](#)

Sugerencias

- ✓ Recuerda que las vistas son importantes, más no el diseño, concéntrate en la funcionalidad de las sesiones antes que en la presentación.
- ✓ Cuida las redirecciones a las múltiples vistas.

¿Preguntas?

Opina y valora
esta clase

Resumen de la clase hoy

- ✓ Concepto de Websocket
- ✓ Ventajas en el uso de websocket por encima de HTTP
- ✓ Implementación de Websocket en Servidor Express

Muchas gracias.

#DemocratizandoLaEducación