



Clase 24. Programación Backend

# ***Cookies, Session y Storage: Parte II***



## ***OBJETIVOS DE LA CLASE***

- Comprender el concepto de persistencia de Session en memoria.
- Conocer y comprender los conceptos de persistencia de datos de session.
- Aprender acerca de los usos y aplicaciones de los mismos.

# ***CRONOGRAMA DEL CURSO***

Clase 23



**Cookies, Sesiones,  
storages:  
Parte I**

Clase 24



**Cookies, Sesiones,  
storages:  
Parte II**

Clase 25



**Autorización y  
Autenticación**

# ***SESSION MEMORYSTORE Y SESSION FILESTORE***

# ***SESSION MEMORYSTORE***

# *¿Qué es y cómo se utiliza el **memoryStore**?*

- Cuando nos manejamos con session-memory, de forma predeterminada estaremos utilizando el almacenamiento en memoria: el memoryStore.
- Al reiniciar el servidor, estos datos se borran, de modo que no tienen persistencia. Por eso, memoryStore solo está disponible en desarrollo (nunca en producción).

>> Para superar esta limitación utilizaremos Session FileStore.

# ***SESSION FILESTORE***

# *¿Qué es y cómo se utiliza el **fileStore**?*

- Se utiliza igual que memoryStore, con la diferencia de que se crea una carpeta de archivos en donde se almacenan los datos de session.
- Estos tendrán persistencia, ya que quedarán guardados en el servidor.



# *Empezando a usar fileStore*



Además de tener instalado el express-session habrá que instalar session-file-store:

```
$ npm install session-file-store --save
```

Por otro lado, requerimos el session-file-store de la forma que se muestra en la imagen.



```
const express = require('express')
const cookieParser = require('cookie-parser')
const session = require('express-session')
/* ----- */
/*           Persistencia por file store           */
/* ----- */
const FileStore = require('session-file-store')(session)
/* ----- */
```

# Usando fileStore



```
const app = express()
app.use(cookieParser())
app.use(session({
  /* ----- */
  /*      Persistencia por file store      */
  /* ----- */
  store: new FileStore({path: '../sesiones', ttl:300, retries: 0}),
  /* ----- */

  secret: 'shhhhhhhhhhhhhhhhhhhhh',
  resave: false,
  saveUninitialized: false/* ,
  cookie: {
    |   maxAge: 40000
  } */
}))
```

Se incluye session como middleware a nivel aplicación, como lo vimos la clase pasada.

Pero se agrega la clave store en el objeto, de la forma que se muestra en la imagen. El path especificado es la ubicación y nombre de la carpeta que se crea.

👉 Se aplica req.session en las rutas deseadas, de la misma forma ya vista anteriormente.

# ***Carpeta de archivos***

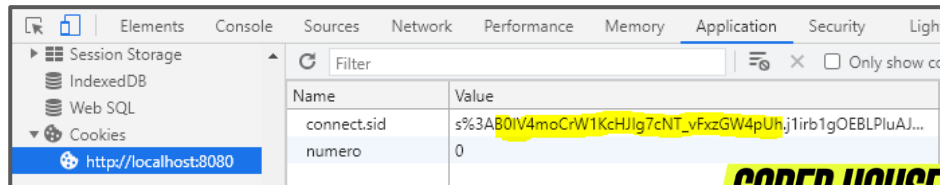
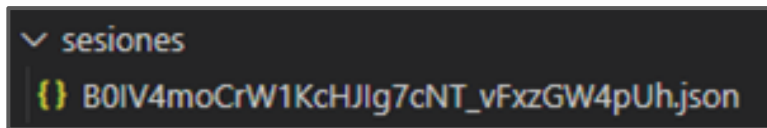


```
{  
  "cookie": {  
    "originalMaxAge": null,  
    "expires": null,  
    "httpOnly": true,  
    "path": "/"  
  },  
  "contador": 5,  
  "__lastAccess": 1617753734522  
}
```



Una vez que se ejecuta el código y se guardan datos en req.session, se crea la carpeta con un archivo .json, con el mismo contenido mostrado en la imagen.

El nombre de este archivo corresponderá a las cookies de session, como se muestra en las siguientes imágenes:





# ***GUARDAR DATOS EN FILE SYSTEM***

*Tiempo: 10 minutos*

# ***Guardar Datos en File System***

Desafío  
generico



Modificar el resultado del [desafío de session](#) de la clase anterior para que almacene las sesiones de usuario en el file system; en vez de que su persistencia sea en la memoria del servidor.

- La carpeta destino será 'sesiones' y estará creada en el directorio anterior al proyecto.
- Verificar que con las distintas sesiones de usuario se crean archivos dentro de esa carpeta, cuyos nombres corresponden a las cookies de sesión activas.
- Fijar la duración del tiempo de vida de la sesión y de su cookie de 1 minuto.
- Analizar los resultados.

# ***SESSION REDIS Y REDISLAB***

***REDIS***

# ***¿Qué es Redis?***

- Es un *servidor de diccionarios remoto (Remote Dictionary Server)*.
- Almacén de datos clave-valor en memoria de código abierto que se puede utilizar como base de datos, caché y agente de mensajes.



# Contar con Redis



- Se debe descargar el archivo comprimido y luego agregar la ruta de la carpeta al PATH del sistema.
- Para iniciar el servidor de Redis, en consola: *redis-server*

```
Coder
C:\coder_mini
A cd C:\Cursos\Coderhouse\CursoBackend\Clase25\codigoEjemplo\Redis-x64-3.0.504
C:\Cursos\Coderhouse\CursoBackend\Clase25\codigoEjemplo\Redis-x64-3.0.504
A redis-server
[21460] 04 Apr 11:44:45.804 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf

Redis 3.0.504 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 21460

http://redis.io

[21460] 04 Apr 11:44:45.821 # Server started, Redis version 3.0.504
[21460] 04 Apr 11:44:45.823 * DB loaded from disk: 0.002 seconds
[21460] 04 Apr 11:44:45.824 * The server is now ready to accept connections on port 6379

C:\Cursos\Coderhouse\CursoBackend\Clase25\codigoEjemplo\Redis-x64-3.0.504
A redis-cli
127.0.0.1:6379>
```

# ***Características***



- Los datos de Redis se almacenan en **memoria** del servidor, por lo que el acceso a los mismos es muy rápido.
- Tiene mucha flexibilidad en cuanto a las estructuras de datos que admite (strings, listas, hashes, sets, entre otros). De esta forma, el código queda mucho más simple y con menos líneas.
- Por persistencia, Redis admite copias de seguridad puntuales (guarda el conjunto de datos en el disco).
- Crea soluciones con un alto nivel de disponibilidad, lo que ofrece fiabilidad y rendimiento estables.

# ***Comando Keys***



- Las Redis Keys son binarias y seguras. Esto significa que puede usar cualquier secuencia binaria como clave, ya sea un string o un archivo de imagen.
- El tipo más usado y recomendado por su mayor simpleza es un string como Redis Keys.
- Con el uso de los comandos SET y GET configuramos y recuperamos un valor de un string.



# ***SET key value***

- Es el comando con el que se pueden setear nuevos **key value**.
- Se le puede especificar un tiempo de expiración en segundos o milisegundos.
- Da como respuesta “OK” si el comando SET se ejecutó correctamente y, si hubo algún problema, devuelve “Null”.

```
redis> SET mykey "Hello"  
"OK"
```

```
redis> SET anotherkey "will expire in a minute" EX 60  
"OK"
```



# ***GET key value***

- Es el comando con el que se puede leer el valor de la key.
- Devuelve un error si el valor de la key es distinto de un string.
- Si se ejecuta correctamente devuelve el valor de la key. Si esta no existe, devuelve la palabra reservada *nil*.

```
redis> GET nonexistent  
(nil)
```

```
redis> GET mykey  
"Hello"
```



# ***TTL key***

- Devuelve el tiempo de vida que le queda a la key, si es que tiene seteado un timeout.
- Permite al cliente chequear por cuánto tiempo más esa key va a ser parte del conjunto de datos.
- Devuelve -1 si la key no existe o no tiene un tiempo de expiración.

```
redis> SET mykey "Hello"  
"OK"  
redis> EXPIRE mykey 10  
(integer) 1  
redis> TTL mykey  
(integer) 10
```

# Empezando a usar Redis



Además de instalar express-session, se deben instalar las dependencias redis y connect-redis:

```
$ npm install redis connect-redis --save
```

Se requiere redis y connect-redis de la forma que se muestra en la imagen.

Además, se crea un Client de redis.



🔗 Desde el [link de github](#) en pantalla es posible descargar la carpeta con los datos necesarios para crear una Redis local como persistencia de los datos.

```
const express = require('express')
const cookieParser = require('cookie-parser')
const session = require('express-session')
/* ----- */
/*           Persistencia por redis database           */
/* https://github.com/microsoftarchive/redis/releases */
/* ----- */
const redis = require('redis')
const client = redis.createClient()
const RedisStore = require('connect-redis')(session)
/* ----- */
```

# Usando Redis



```
const app = express()
app.use(cookieParser())
app.use(session({
  /* ----- */
  /*      Persistencia por redis database      */
  /* ----- */
  store: new RedisStore({
    host: 'localhost',
    port: 6379,
    client: client,
    ttl: 300
  }),
  /* ----- */

  secret: 'shhhhhhhhhhhhhhhhhhh',
  resave: false,
  saveUninitialized: false/* ,
  cookie: {
    maxAge: 40000
  } */
}))
```



Se agrega en el app.use de session otra clave al objeto llamada store, similar a sessionFile.

Luego, se utiliza en las rutas y controladores de la misma forma que lo ya visto en sessionMemory.



***REDISLAB***



# *¿De qué se trata?*

- RedisLab es lo mismo que Redis, pero los datos se guardan en la nube.
- Entrando a su página oficial, se crea una cuenta para poder empezar a utilizarlo: <https://redislabs.com/>



redislabs  
HOME OF REDIS

# Crear una cuenta



[Home](#) [Sessions](#) [Training](#) [FAQs](#)

[Register now](#) →

Fields marked with "\*" are mandatory.

\* First Name

\* Last Name

\* Company Name

\* Business Email

\* Phone

\* Job Function

\* Country

\* Industry

What Redis are you using?

\* Register me in the \$100,000 Hackathon ☐ Yes ☐ No

\* Training or Conference?

**CODER HOUSE**



# *redis-cli*

- Redis-cli es la interfaz de línea de comandos de Redis, un programa simple que permite enviar comandos a Redis y leer las respuestas enviadas por el servidor, directamente desde la terminal.
- Para empezar a usarlo seguir los siguientes pasos de comandos en consola:
  1. `redis-cli` para conectar el servidor local.
  2. `redis-cli -h host -p port -a password` para conectar con el servidor remoto.

# ***PARTE 1: Conectar con Redis***

Desafío  
generico



*Tiempo estimado: 10 minutos*

Poner en marcha el servidor de base de datos Redis y conectar su cliente CLI.

Realizar las siguientes tareas:

1. Listar la información total en la base.
2. Crear 5 claves sin tiempo de expiración que contengan nombres de productos.
3. Listar nuevamente toda la información.
4. Mostrar el contenido de cada una de las claves de productos.
5. Agregar un producto más, fijando un tiempo de vida de 30 segundos.
6. Listar el nuevo producto y su tiempo de expiración.
7. Verificar que al transcurrir ese tiempo, el producto desaparezca del listado general.

# ***PARTE 2: Guardar con Redis***

Desafío  
generico



*Tiempo estimado: 15 minutos*

Realizar nuevamente el desafío “*Guardar datos en File System*” pero esta vez persistiendo las sesiones de usuario en Redis.

- Fijar un tiempo de vida de la sesión de 1 minuto que será recargada en cada visita del cliente al sitio.
- Acceder con dos clientes distintos y verificar que las sesiones respectivas hayan sido creadas en la base.
- Comprobar los datos y el tiempo de vida de las sesiones en la base verificando que cuando se extingan desaparezcan de la misma y que el usuario quede automáticamente deslogueado de su sesión.



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**

# ***SESSION MONGO Y MONGO ATLAS***



# ***SESSION MONGO***



# *¿Qué es?*

Mediante el paquete de Node llamado **connect-mongo** se puede utilizar la base de datos de MongoDB para persistir los datos almacenados en Session.



# *Empezando con connect-mongo*



Instalación del  
módulo:

```
$ npm install connect-mongo --save
```

```
const express = require('express')
const cookieParser = require('cookie-parser')
const session = require('express-session')
/* ----- */
/*      Persistencia por MongoDB      */
/* ----- */
const MongoStore = require('connect-mongo')
/* ----- */
```

Se requieren los módulos como se muestra en la imagen. Se incluye como lo mencionamos el de connect-mongo

# *Usando connect-mongo*



```
const app = express()
app.use(cookieParser())
app.use(session({
  /* ----- */
  /*      Persistencia por redis database      */
  /* ----- */
  store: MongoStore.create({ mongoUrl: 'mongodb://localhost/sesiones' }),
  /* ----- */

  secret: 'shhhhhhhhhhhhhhhhhhhhh',
  resave: false,
  saveUninitialized: false/* ,
  cookie: {
    |   maxAge: 40000
  } */
}))
```

Se agrega en el app.use de session una clave en el objeto, especificando la url de Mongo local donde se van a guardar los datos almacenados en session.

El código queda como se muestra en la imagen.

# ***SESSION MONGO ATLAS***

# ***Empezando a usar Mongo Atlas***



- Es lo mismo que session con Mongo pero la diferencia es que Atlas es la base de datos en la nube, por lo que allí se van a almacenar los datos de session.
- Se necesitan los mismos módulos que para mongo session y se requieren como se muestra a continuación:

```
const express = require('express')
const cookieParser = require('cookie-parser')
const session = require('express-session')
/* ----- */
/*      Persistencia por MongoDB      */
/* ----- */
const MongoStore = require('connect-mongo')
const advancedOptions = { useUrlParser: true, useUnifiedTopology: true }
/* ----- */
```

*(AdvancedOptions se explica en la siguiente diapositiva)*

# USando Mongo Atlas



```
const app = express()
app.use(cookieParser())
app.use(session({
  /* ----- */
  /*      Persistencia por redis database      */
  /* ----- */
  store: MongoStore.create({
    //En Atlas connect App : Make sure to change the node version to 2.2.12:
    mongoUrl: 'mongodb://daniel:daniel123@cluster0-shard-00-00.nfdif.mongodb.net:27017',
    mongoOptions: advancedOptions
  }),
  /* ----- */

  secret: 'shhhhhhhhhhhhhhhhhhh',
  resave: false,
  saveUninitialized: false/* ,
  cookie: {
    |   maxAge: 40000
  } */
}))
```

Para utilizarlo, debemos conectar con la URL de la base de datos en Atlas, es decir, en la nube.

La constante de `advancedOptions` definida anteriormente se utiliza para las opciones avanzadas de la conexión con la BD.



## ***LOG-IN POR FORMULARIO***

Incorporaremos un mecanismo sencillo que permite loguear un cliente por su nombre mediante un formulario de ingreso.

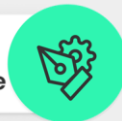


# LOG-IN POR FORMULARIO

**Formato:** link a un repositorio en Github con el proyecto cargado.

**Sugerencia:** no incluir los node\_modules

Desafío  
entregable



## >> Consigna:

Continuando con el desafío de la clase anterior, vamos a incorporar un mecanismo sencillo que permite loguear un cliente por su nombre, mediante un formulario de ingreso.

Luego de que el usuario esté logueado, se mostrará sobre el contenido del sitio un cartel con el mensaje “Bienvenido” y el nombre de usuario. Este cartel tendrá un botón de deslogueo a su derecha.

Verificar que el cliente permanezca logueado en los reinicios de la página, mientras no expire el tiempo de inactividad de un minuto, que se recargará con cada request. En caso de alcanzarse ese tiempo, el próximo request de usuario nos llevará al formulario de login.

Al desloguearse, se mostrará una vista con el mensaje de 'Hasta luego' más el nombre y se retornará automáticamente, luego de dos segundos, a la vista de login de usuario.

**>> Ejemplos:** Se adjuntan tres *screenshoot* con las vistas anteriormente mencionadas. **CODER HOUSE**

## >> Ejemplo 1

Vista de Productos x +

localhost:8080/login

Bienvenido Daniel [Desloguear](#)

### Ingrese Producto

*Nombre*

*Precio*

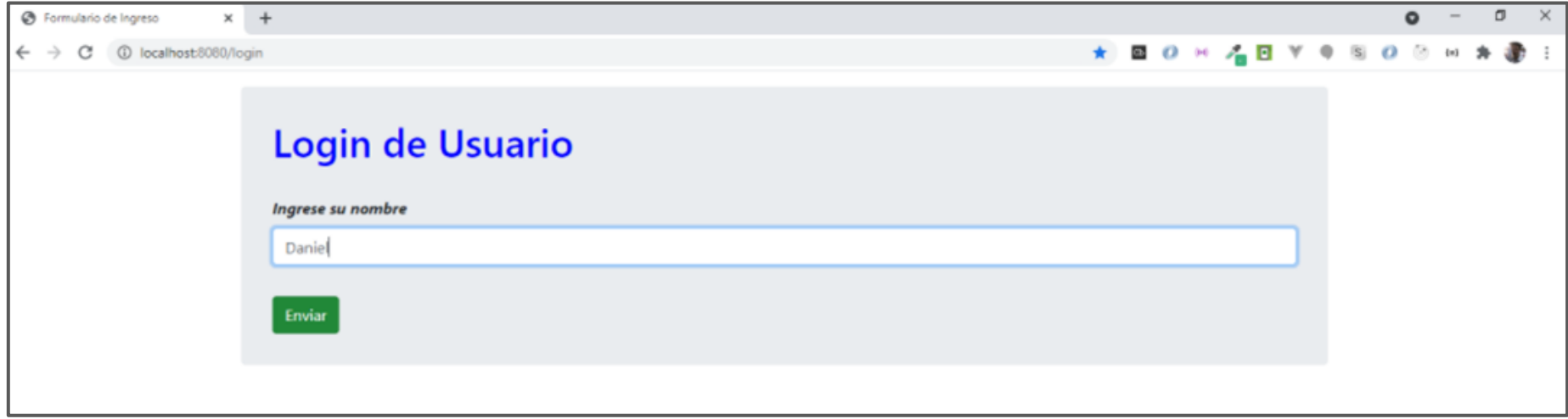
*Foto URL*

[Enviar](#)

### Lista de Productos

Nombre	Precio	Foto
--------	--------	------

## >> Ejemplo 2



The screenshot shows a web browser window with the title 'Formulario de Ingreso'. The address bar displays 'localhost:8080/login'. The main content area features a light gray box with the heading 'Login de Usuario' in blue. Below the heading is the instruction 'Ingrese su nombre' in italics. A text input field contains the text 'Danie'. A green button labeled 'Enviar' is positioned below the input field.

Formulario de Ingreso

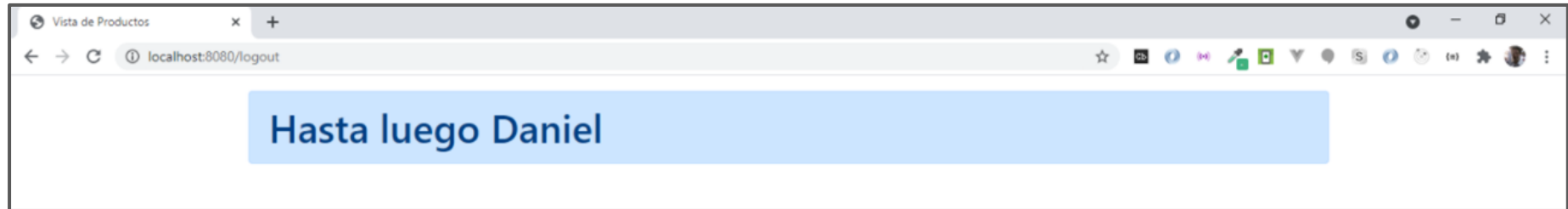
localhost:8080/login

### Login de Usuario

*Ingrese su nombre*

Enviar

## >> Ejemplo 3



The screenshot shows a web browser window with the title 'Vista de Productos'. The address bar displays 'localhost:8080/logout'. The main content area features a light blue box with the text 'Hasta luego Daniel' in dark blue.

Vista de Productos

localhost:8080/logout

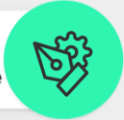
Hasta luego Daniel

# ***PERSISTIR DATOS DE SESSION EN MONGO ATLAS***

**Formato:** link a un repositorio en Github con el proyecto cargado.

**Sugerencia:** no incluir los node\_modules

Desafío  
entregable



## **>> Detalles del entregable:**

La solución entregada deberá persistir las sesiones de usuario en Mongo Atlas.

- Verificar que en los reinicios del servidor, no se pierdan las sesiones activas de los clientes.
- Mediante el cliente web de Mongo Atlas, revisar los id de sesión correspondientes a cada cliente y sus datos.
- Borrar una sesión de cliente en la base y comprobar que en el próximo request al usuario se le presente la vista de login.
- Fijar un tiempo de expiración de sesión de 10 minutos recargable con cada visita del cliente al sitio y verificar que si pasa ese tiempo de inactividad el cliente quede deslogueado.

***¿PREGUNTAS?***





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Session fileStore
  - Session Redis / RedisLab
  - Session Mongo / Atlas
- 



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***