CODER HOUSE

Clase 23. Programación Backend

Cookies, Session y Storage: Parte I

OBJETIVOS DE LA CLASE

- Conocer y comprender los conceptos de Cookies, Session y Storage.
- Incorporar de sus usos y aplicaciones.

CODER HOUSE

CRONOGRAMA DEL CURSO

Clase 22	Clase 23	Clase 24
Trabajo con datos: Normalización	**Cookies, Sesiones, storages: Parte 1**	**Cookies, Sesiones, storages: Parte 2**

CODER HOUSE

COOKIES

¿Qué son las Cookies?

Las Cookies son archivos que podemos guardar del lado del **cliente**, en el navegador del usuario.

Características

- A las cookies se les puede configurar un **tiempo de vida**. Una vez finalizado el mismo, la cookie se elimina del navegador.
- Al almacenarse del lado del cliente, el espacio con el que se cuenta es limitado, por lo que se recomienda elegir de forma adecuada lo que se vaya a guardar como cookie.
- Hay que recordar que no se deben almacenar datos sensibles en las cookies.

Empezando a usar cookies

Primero hay que instalar el paquete de cookie parser para poder utilizarlas:

```
$ npm i cookie-parser --save
```

```
const express = require('express')
const cookieParser = require('cookie-parser')

const app = express()

app.use(cookieParser())
```

Hay que requerirlo e incluirlo en la aplicación en la que se lo va a utilizar. Es un middleware que se requiere a nivel de aplicación.

Ejemplo de Cookies

CODER HOUSE

Crear una cookie

- En la ruta /set se crea una cookie de nombre “server” y valor “express”. La misma no tiene un tiempo de vida límite.
- En la ruta /setEX se crea una cookie de nombre “server2” y valor “express”. En esta, se le seteó un tiempo de vida máximo de 30 segundos.

```
1   const express = require('express')
2   const cookieParser = require('cookie-parser')
3
4   const app = express()
5   app.use(cookieParser())
6
7   app.get('/set', (req,res) => {
8       res.cookie('server','express').send('Cookie Set')
9   })
10
11  app.get('/setEX', (req,res) => {
12      res.cookie('server2','express2',{ maxAge: 30000}).send('Cookie SetEX')
13  })
```

Leer una cookie

```
app.get('/get', (req,res) => {
    res.send(req.cookies.server)
```

Este es el código para leer las cookies del ejemplo anterior.
Se utiliza el parámetro de **request**, y el nombre asignado a la cookie que se quiere leer.

Borrar una cookie

```
app.get('/clr', (req,res) => {
    res.clearCookie('server').send('Cookie Clear')
})
```

Para eliminar una cookie, se utiliza el parámetro **response** y el método **clearCookie**. El parámetro que se le pasa al método es el nombre de la cookie que se desea borrar.

SIGNED COOKIES

CODER HOUSE

Características

- A las cookies se les puede agregar un mecanismo de validación que consiste en adjuntar a cada cookie una versión encriptada de su contenido.
- Dicha encriptación se realiza mediante una palabra clave o “secreto” definido del lado del servidor, y desconocido por los clientes.
- El servidor es capaz de verificar si la cookie que se recibe desde el cliente ha sido adulterada o no, chequeando contra la versión encriptada.

cookieParser(secret)

- **Secret:** string o array de strings que se utiliza para firmar las cookies enviadas, y para analizar las recibidas.
  - Es opcional y, si no se especifica, no firmará ni analizará las cookies recibidas.
  - Si es un string, se utiliza como secret. Si es un array de strings, se firmará la cookie con cada string en el orden provisto (y lo mismo al analizar).

Crear una cookie firmada

- Para firmar una cookie antes de enviarla al cliente, solo basta con agregar a los dos argumentos usuales (nombre y valor), un tercer argumento de tipo objeto (como se hizo para setear la expiración) con la propiedad “signed” en *true*. `{ signed: true }`
- Las cookies firmadas recibidas, que hayan pasado la verificación de su firma, ya no se encontrarán en *req.cookies*, sino que aparecerán en **req.signedCookies**. Aquellas que no hayan pasado la verificación, no aparecerán, como si no existieran.

INYECTAR COOKIES EN FRONTEND

Tiempo: 10 minutos

CODER HOUSE

Desafío
generico

Inyectar cookies en frontend

Realizar un programa de backend que permita gestionar cookies desde el frontend. Para ello:

Definir una ruta “cookies”.

Definir un método POST que reciba un objeto con el nombre de la cookie, su valor y el tiempo de duración en segundos, y que genere y guarde dicha cookie.

Definir un método GET que devuelva todas las cookies presentes.

Definir un método DELETE que reciba el nombre de una cookie por parámetro de ruta, y la elimine.

Desafío
generico

Inyectar cookies en frontend

NOTA 1: Utilizar la librería express como estructura de servidor.

NOTA 2: Si algún parámetro recibido es inválido, o directamente inexistente, el servidor devolverá un objeto de error.
Ej: { error: 'falta nombre ó valor' } o { error: 'nombre no encontrado' }. Si todo sale bien, devolver el objeto { proceso: 'ok'}.

NOTA 3: Si el tiempo no está presente, generar una cookie sin tiempo de expiración.

NOTA 4: Generar los request con varios navegadores (Chrome, edge, Firefox) para simular los distintos clientes en forma local.

BREAK

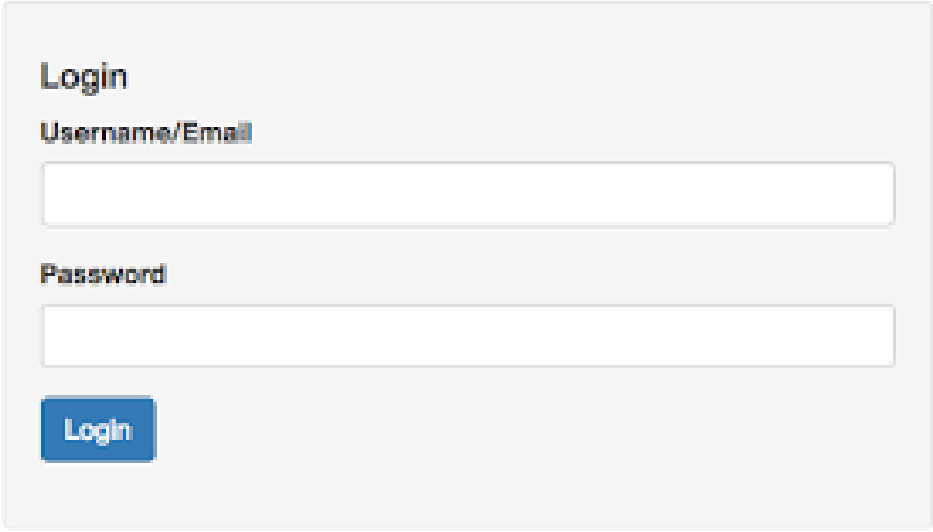
¡5/10 MINUTOS Y VOLVEMOS!

SESSION MEMORY

CODER HOUSE

¿Qué es Session?

Session es un paquete de Node, el cual permite que una variable sea accesible desde cualquier lugar del sitio. Se almacena del lado del **servidor**.



CODER HOUSE

Características

- La información que se quiera guardar en **session** se almacena del lado del servidor.
- Del lado del cliente, se crea un identificador único para poder acceder a esa información desde el navegador.
- Los datos almacenados en **session** se borran al cerrar la ventana del navegador.
- Se utiliza principalmente para guardar los datos de usuario al iniciar sesión.

Empezando a usar session

Se debe instalar el módulo de express-session para empezar a utilizar session:

```
$ npm i express-session --save
```

```
1   const express = require("express")
2   const session = require('express-session')
3   const app = express()
4
5   // Port Number Setup
6   var PORT = process.env.port || 3000
7
8   // Session Setup
9   app.use(session({
10      secret: 'secreto',
11      resave: true,
12      saveUninitialized: true
13  }))
14
15
```

Tiene que ser requerido e incluido en la aplicación en la que se lo va a utilizar. Es un middleware que se requiere a nivel de aplicación.

Guardar datos en session

```
4
5   app.get('/con-session', (req,res) => {
6       if(req.session.contador) {
7           req.session.contador++
8           res.send(`Ud ha visitado el sitio ${req.session.contador} veces.`)
9       }
10      else {
11          req.session.contador = 1
12          res.send('Bienvenido!')
13      }
14  })
```

- En el else se crea la variable en session llamada “contador” la cual tiene inicialmente un valor de 1.
- En el if, si ya existe esta variable en session, se aumenta su valor en 1.
- Tener en cuenta que tanto para inicializar una nueva variable en session como para leer los datos de la misma se utiliza el parámetro de **request**.

Eliminar datos de session

```
app.get('/logout', (req,res) => {
    req.session.destroy( err => {
        if(!err) res.send('Logout ok!')
        else res.send({status: 'Logout ERROR', body: err})
    })
})
```

Para eliminar datos de una variable de session, se utiliza el parámetro de **request** y el método **destroy**. Como parámetro se pasa un callback.

Login con session

```
app.get('/login', (req, res) => {
 const { username, password } = req.query
 if (username !== 'pepe' || password !== 'pepepass') {
   return res.send('login failed')
 }
 req.session.user = username
 req.session.admin = true
 res.send('login success!')
})
```

Para ***iniciar sesión*** se verifica que los datos ingresados por el usuario sean los correctos. Si lo es, se guarda en session los datos de este usuario. Además, se puede crear la variable admin, también en session, con el valor de true, lo que indica que el usuario logueado es un administrador

Middleware de autenticación

```
function auth(req, res, next) {
 if (req.session?.user === 'pepe' && req.session?.admin) {
   return next()
 }
 return res.status(401).send('error de autorización!')
}
```

Mediante estos middleware se puede **limitar el acceso** a determinadas rutas a aquellos usuarios que sean administradores (o, por ejemplo, otras a cualquier usuario logueado).

Si coincide el usuario guardado en session y además es admin, entonces sigue a la ruta, sino devuelve un error.

Aplicación del middleware

```
app.get('/privado', auth, (req, res) => {
 res.send('si estas viendo esto es porque ya te logueaste!')
})
```

Al aplicar el auth middleware en la ruta /content, estará accesible únicamente luego de que el usuario haya iniciado sesión.
Además, según el código del middleware, se puede especificar a cierto usuario o cierto tipo de usuario (admin o usuario común, por ejemplo)

Logout con session

```
app.get('/logout', (req, res) => {
 req.session.destroy(err => {
   if (err) {
     return res.json({ status: 'Logout ERROR', body: err })
   }
   res.send('Logout ok!')
 })
})
```

Para cerrar sesión, solo basta con aplicar el método **destroy** de session.

Al borrar los datos almacenados, ya no queda registro de que el usuario haya iniciado sesión. Y en este caso, ya no van a ser accesibles las rutas que tengan el auth middleware.

SESIONES DE USUARIO EN SERVER

Tiempo: 15 minutos

CODER HOUSE

Sesiones de usuario en server

Desafío generico

Realizar un programa de backend que establezca sesiones de usuarios en el servidor.

Cuando un cliente visita el sitio por primera vez en la ruta 'root', se presentará el mensaje de “Te damos la bienvenida”.

Con los siguientes request de ese mismo usuario, deberá aparecer el número de visitas efectuadas. El cliente podrá ingresar por query params el nombre, en cuyo caso se añadirá a los mensajes devuelto.

Por ejemplo: “Bienvenido Juan” o “Juan visitaste la página 3 veces”. Ese nombre sólo se almacenará la primera vez que el cliente visite el sitio.

Sesiones de usuario en server

Desafío generico

Se dispondrá de una ruta 'olvidar' que permita reiniciar el proceso de visitas para el usuario.

En caso de que no haya error, se retornará el mensaje “Hasta luego” más el nombre del cliente (de existir); caso contrario un objeto con el siguiente formato: { error : descripción }.

Luego de requerir esa ruta, el efecto será como el de visitar el sitio por primera vez.

NOTA1: Utilizar el middleware express como estructura de servidor.

NOTA2: Generar los request con varios navegadores (Chrome, edge, Firefox) para simular los distintos clientes en forma local.

¿PREGUNTAS? 

¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Cookies
- Session Memory

OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN

CODER HOUSE