UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Diplomatura en programación web full stack con React JS

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 4: NodeJS Intermedio

Unidad 2: Middleware, sesiones y jwt

Presentación:

En esta Unidad vamos a aprender sobre los potentes middleware, una característica del modo de trabajo de NodeJS. Además, vemos más en profundidad sobre la tecnología cliente-servidor y la necesidad de implementar sesiones. Es una Unidad muy interesante y útil para comenzar a pensar en desarrollar aplicaciones comerciales.

Objetivos:

Que los participantes:

Comprendan el concepto de sesiones y token y sean capaces de implementarlos en su aplicación NodeJS

Bloques temáticos:

1. Concepto de middleware
2. Creación de un middleware
3. Autenticación
4. Concepto de sesiones
5. JWT
6. Ejemplo
7. Trabajo Práctico

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

** El MEC es el modelo de E-learning colaborativo de nuestro Centro.*

Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Concepto de middleware

El término middleware es ampliamente utilizado en diferentes tecnologías, aunque teniendo diferentes definiciones para cada una de ellas. Cuando hablamos de middleware estamos hablando de una capa intermedia (algo entre 2 cosas, que generalmente hace de nexo entre ellas).

Cuando hablemos de middleware en express, estaremos hablando de un componente que está diseñado para modificar una petición o respuesta HTTP, pero que generalmente no da la respuesta. Está diseñado para ser encadenado (ejecución uno después del otro), para formar una tubería de cambios de comportamiento durante el procesamiento de una petición.

2. Creación de un middleware

Son funciones que tienen acceso al objeto de petición (request), el de respuesta (response), y a la función que le sucede (next) en el ciclo de ejecución de la aplicación. La función siguiente (next), es una función que, cuando es llamada, ejecuta el siguiente middleware.

Estas funciones son ejecutadas previo a las funciones específicas para una ruta.
Las funciones middleware pueden realizar las siguientes tareas:

- Ejecutar cualquier código
- Realizar cambios en el objeto petición (request) y en objeto respuesta (response)
- Finalizar el ciclo de una petición
- Llamar a la siguiente función middleware

Al finalizar cada función middleware se debe llamar a next() para que continúe la ejecución de la siguiente función middleware o ruta. En caso de no llamarse a next(), no se ejecutará ninguna otra función.

Veamos un ejemplo

```
var express = require('express');
var app = express();

var myLogger = function (req, res, next) {
  console.log('-Ejecutando Logger-');
  next();
}

app.use(myLogger);

app.get('/', function (req, res) {
  res.send('Bienvenido al Curso!');
})

app.listen(3000);
```

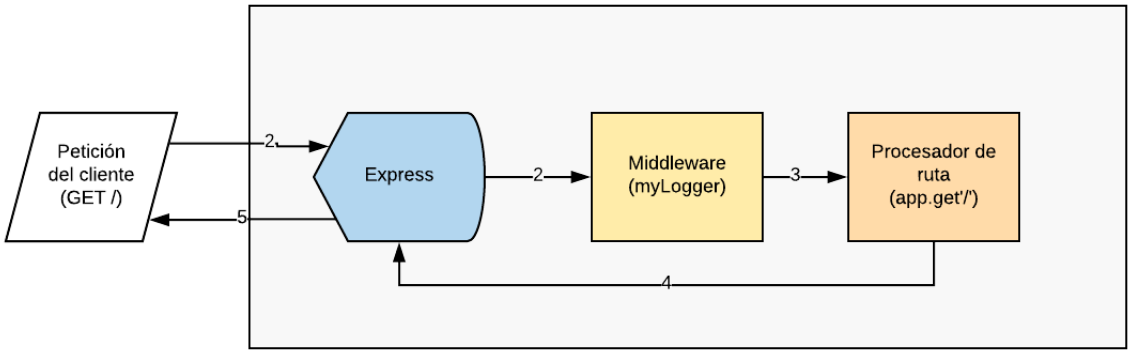
La salida del código es la siguiente:

Salida por consola	Respuesta al cliente (HTTP)
01 -Ejecutando Logger- 02	01 02 Bienvenido al Curso!

Primero mostrando por consola el mensaje -Ejecutando Logger- y luego respondiendo al cliente con “Bienvenido al Curso!”

Como podemos ver en el ejemplo, la única ruta que concuerda con la petición de un GET / es la de la línea 11 (app.get….), pero igualmente se ejecuta el código de la función guardada en la variable myLogger. Esto es porque myLogger es un middleware, un código que se ejecuta antes de ejecutarse el código de la ruta, y luego se ejecuta el código de la ruta específica.

Secuencia de ejecución



Analicemos el código del middleware

```
var myLogger = function (request, response, next) {
  console.log('-Ejecutando Logger-');
  next();
}
```

```
app.use(myLogger);
```

Las funciones middleware reciben 3 parámetros:

- request: petición que realiza el cliente
- response: respuesta que se le entrega al cliente
- next: próxima función de la cadena de ejecución (en este caso, como hay un único middleware, la próxima función es la que procesa la ruta app.get('/', …

```
var myLogger = function (request, response, next) {
...
}
```

La función middleware, es guardada en una variable (myLogger), para luego poder utilizarla más adelante.

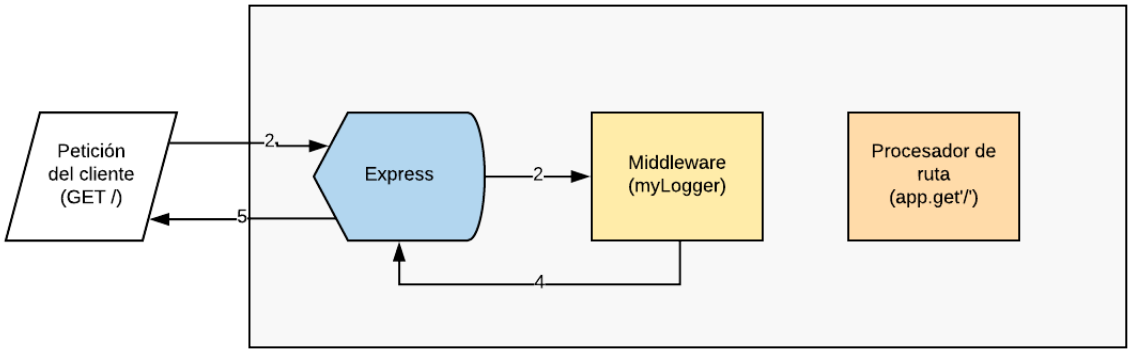
```
  console.log('-Ejecutando Logger-');
  next();
```

En la primer línea solo muestra por consola el mensaje -Ejecutando Logger-

En la línea siguiente, le indica a Express, que luego de ejecutar esta función middleware, ejecute la siguiente función. En caso de no incluir el next(), no se ejecutará ninguna otra función de la cadena de ejecución, siendo esta la última función en ejecutarse. La próxima función en la cadena, puede ser otro middleware, o una ruta.

Es de suma importancia incluir el next() en caso que deseemos que siga la cadena de ejecución.

En caso de no ejecutar el next() en el ejemplo anterior, la secuencia de ejecución sería la del diagrama siguiente.



En el cual podemos apreciar que la cadena de ejecución se termina en el middleware y nunca es llamado el código que procesa la petición a la ruta específica.

Un ejemplo de caso en el cual NO deseamos que continúe la ejecución, puede ser cuando realizamos una validación si el usuario ha ingresado en el sistema, y en caso que no haya ingresado, lo podemos (desde el mismo middleware) redireccionar a la página de login. En este caso, no deseamos pasar la ejecución a la próxima función (que puede ser la ruta que responda

```
app.use(myLogger);
```

El middleware no será ejecutado, a menos que le indiquemos a Express que incluya dicho middleware en la cadena de ejecución. Para incluirlo en la cadena de ejecución, utilizamos el método use del objeto Express.

3. Autenticación

Procedimiento informático que permite asegurar que un usuario de un sitio web u otro servicio similar es auténtico o quien dice ser.

Hay diferentes formas de autenticar, una de ellas es la autorización por tokens. Un token es una cadena de caracteres enviadas desde el cliente hacia el servidor, el servidor detecta el token y puede revisar a qué usuario pertenece, si es válido o no, mantener un registro de peticiones, etc.

4. Concepto de sesiones

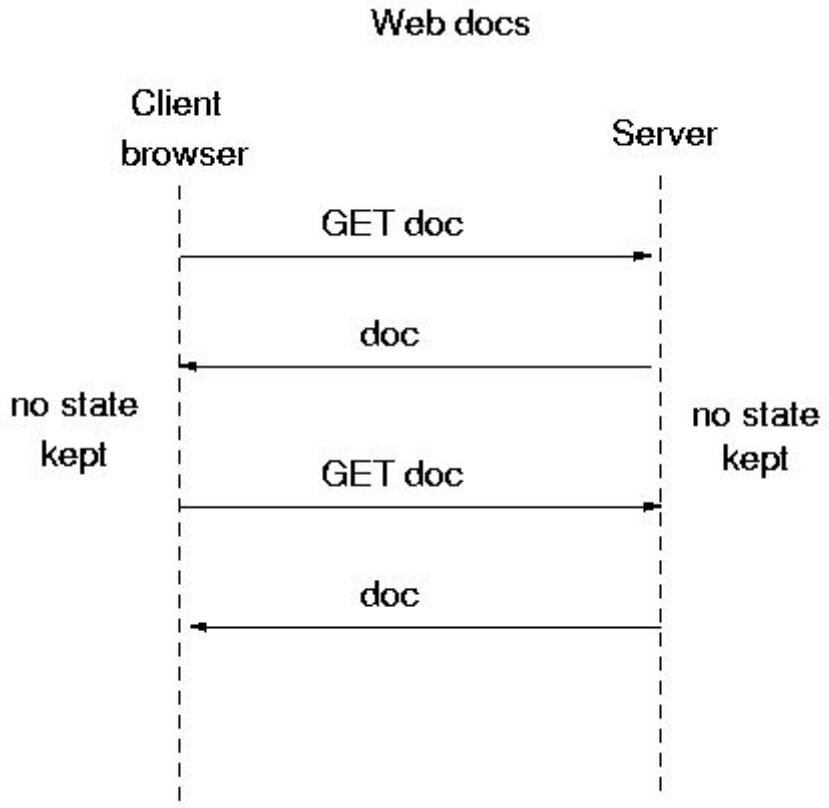
Una sesión, del latín sessio, es un período temporal ocupado por una cierta actividad.

En el ámbito de la informática, se conoce como sesión a la duración de una conexión a un determinado sistema o red. La sesión suele incluir el intercambio de paquetes de información entre un cliente y un servidor. La sesión suele iniciarse luego de que el usuario se loguea al sistema.

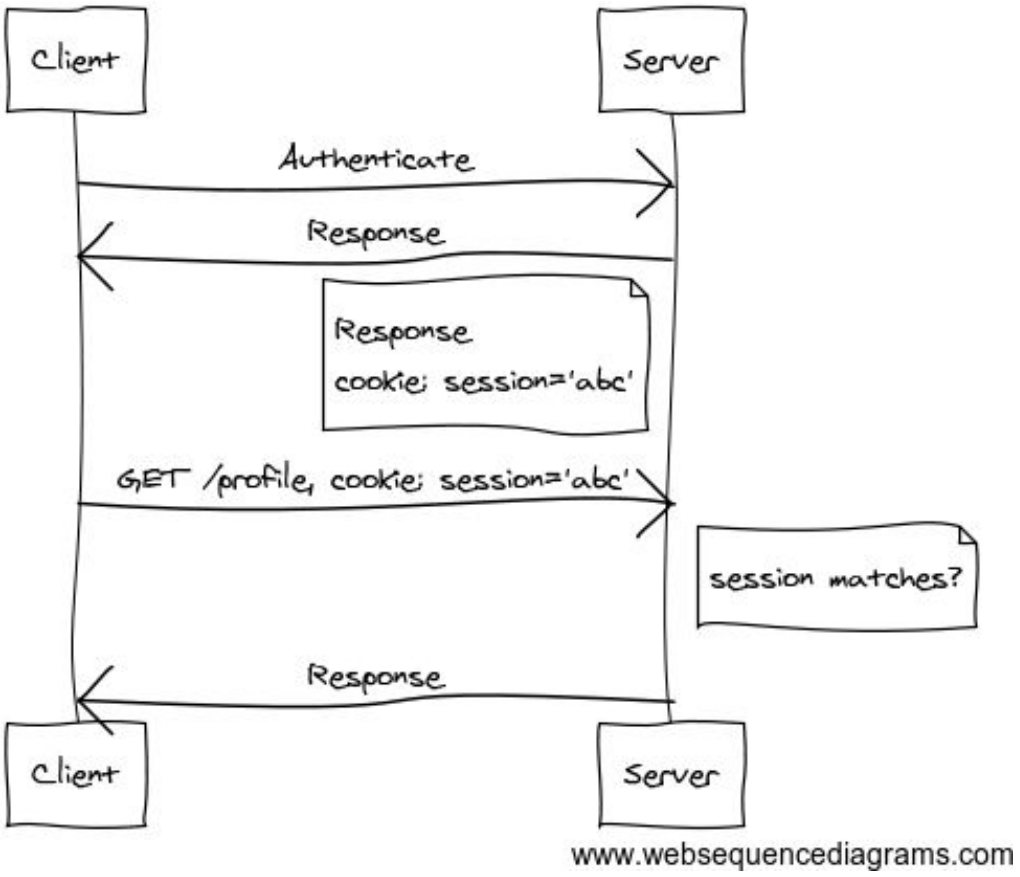
Un token de sesión es un identificador único que está generado y enviado desde un servidor a un cliente para identificar la sesión de interacción actual. El cliente envía una cookie HTTP y/o lo envía como parámetro en GET o POST.

HTTP es un protocolo sin estado. Cada petición es completamente independiente de otra y no hay forma de compartir información entre las peticiones. El objetivo de definir una sesión es poder almacenar información que sobrevive las peticiones HTTP de un visitante.

La información de la sesión es un objeto de JS que se puede almacenar en el cliente (dentro de una cookie) o en el servidor en una base de datos o la memoria.



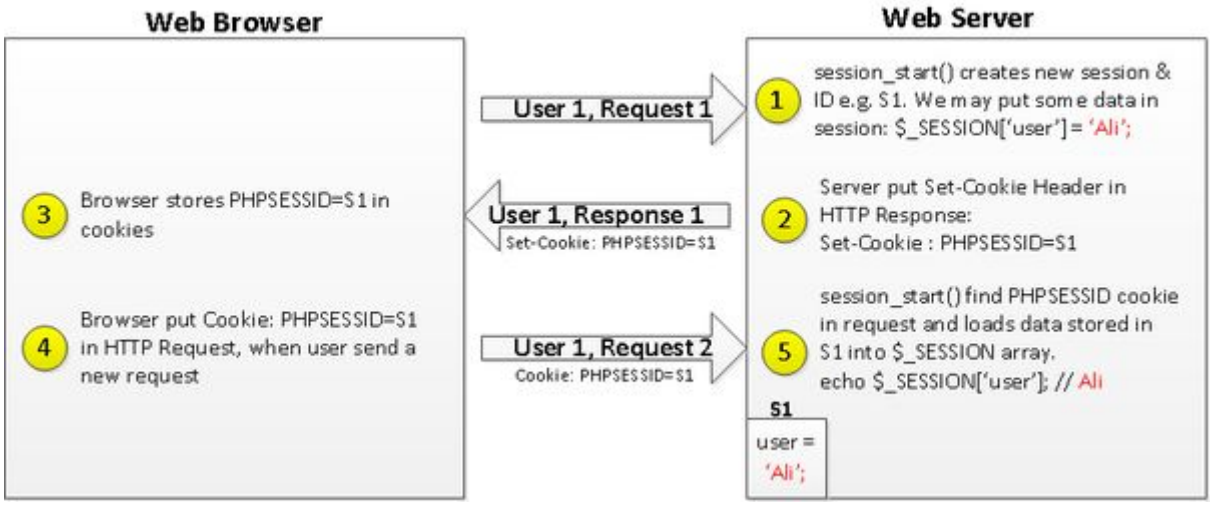
Cómo trabajan las sesiones



Consideraciones

- Todos los datos se guardan en el servidor (esto se llama session)
- Podemos agregar información a una session, borrar, modificar, o eliminar la session (logout)
- Podemos asignar información al usuario y consultarla o modificarla de cualquier página de nuestro sitio

Circuito de varias llamadas



Sesiones en NodeJS con Express

Requiere la instalación de un módulo

```
npm install --save express-session
```

Incorporación en Express

```
var session = require('express-session')
app.use(session({ secret: '<algún texto secreto>', cookie: { maxAge: 60000 }}))
```

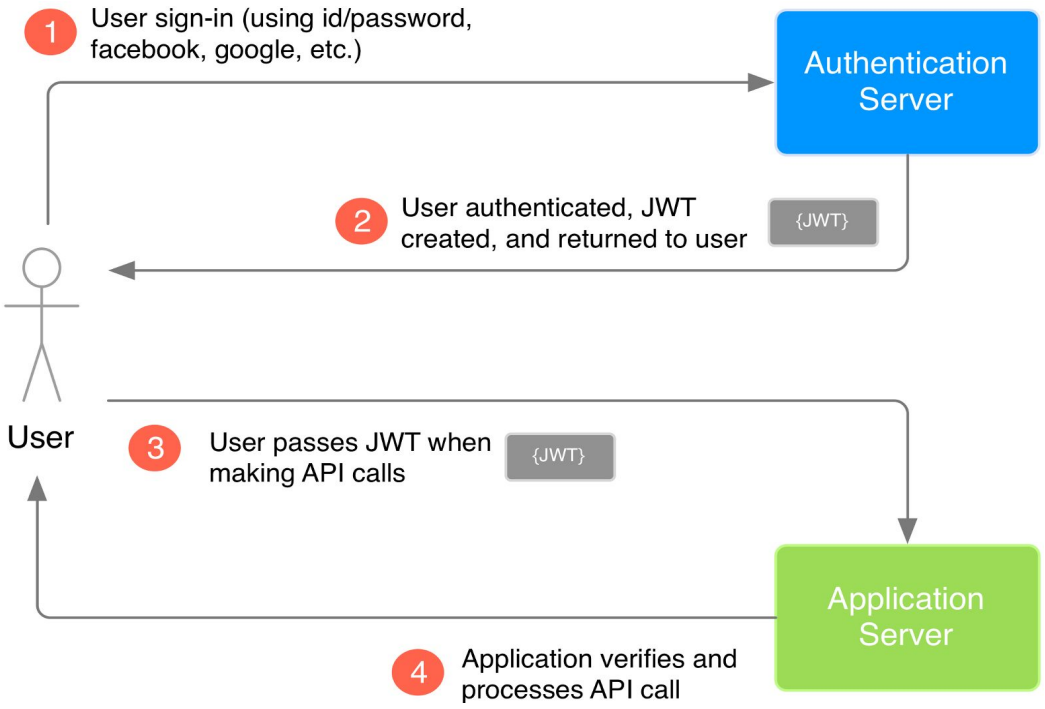
Se crea la variable de sesion

```
app.get('/crear', function(req, res) {
   req.session.contador = 0;
   res.send(req.session.contador);
})

Uso de req.session.contador
app.get('/incrementar', function(req, res) {
   res.session.contador++;
   res.send(req.session.contador);
});
```

5. JWT

Es el método de autenticación más popular para las API REST



JWT TOKEN

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5c
CI6IkpXVCJ9.eyJzdWIiOiIxM
jM0NTY3ODkwIiwibmFtZSI6Ik
pvaG4gRG9lIiwiYWRtaW4iOnR
ydWV9.TJVA95OrM7E2cBab30R
MHrHDcEfxjoYZgeFONFh7HgQ
```

Decoded

```
{
  "alg": "HS256",
  "typ": "JWT"
}
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
)
```

Header

Payload

Signature

Para utilizar JWT en NodeJS con Express

Se requiere instalar la siguiente biblioteca

```
npm install --save express jsonwebtoken
```

6. Ejemplo

En el foro encontrará un enlace al video con el ejemplo de implementación de JWT en NodeJS con Express

7. Trabajo Práctico

Agregar autenticación por jwt al desarrollo realizado en la unidad anterior.

Bibliografía utilizada y sugerida

Express (n. d.) Recuperado de: https://expressjs.com/es/

MDN - JavaScript (n.d.) Recuperado de: https://developer.mozilla.org/es/docs/Web/JavaScript

NodeJs (n. d.) Recuperado de: https://nodejs.org/es/

NodeJS Documentacion (n. d.) Recuperado de: https://nodejs.org/es/docs/

NPM (n. d.) Recuperado de: https://www.npmjs.com/package/page

Wikipedia - NodeJS (n. d.) Recuperado de: https://es.wikipedia.org/wiki/Node.js

Lo que vimos:

Middleware, sesiones y jwt.

Lo que viene:

Patrón MVC + Service.