Ejercicio Entregable 7

Gestión de usuarios mediante Express.js

Aplicaciones Web
Año 2017/2018
Grado en Ing. Software
Fdi - UCM

El ejercicio de esta hoja es entregable y puede realizarse en el laboratorio. Los ejercicios entregables realizados a lo largo del curso computan el 10 % de la nota final, pero su entrega no es obligatoria para aprobar la asignatura. La entrega se realiza por parejas.

Fecha límite de entrega: 7 de diciembre de 2017.

La entrega se realizará a través del Campus Virtual.

El objetivo de este ejercicio es poner en práctica los atributos de sesión en nuestra aplicación web de gestión de tareas. Para realizar el ejercicio es necesario haber realizado el anterior (aunque no se hubiese entregado finalmente), pues consiste en realizar modificaciones sobre el mismo.

Antes de comenzar, descarga los ficheros `login.html` y `dao_users.js` que encontrarás en el Campus Virtual. Guarda el primero de ellos en la carpeta `public` del proyecto Node que hiciste en el ejercicio entregable anterior, y el segundo en la carpeta raíz de dicho proyecto. Arranca el servidor y comprueba, accediendo a la dirección `http://localhost:3000/login.html`, que el formulario de *login* se visualiza correctamente (ver Figura 1).

1 Introducir el middleware `express-session`

En primer lugar necesitarás el middleware `express-session` para poder almacenar y obtener los atributos de sesión. Además, los datos de sesión se almacenarán en nuestra base de datos MySQL. Para ello:

1. Importa los módulos `express-session` y `express-mysql-session`.
2. Obtén la clase `MySQLStore`. Esta clase se obtiene llamando a la función exportada por el módulo `express-mysql-session`, pasándole como parámetro el objeto exportado por el módulo `express-session`[1].
3. Crea una instancia de `MySQLStore`, pasando a la constructora los datos de conexión a la BD. Estos datos los habrás obtenido a partir del módulo `config.js`.
4. Crea un middleware de sesión y añádelo a la cadena de middlewares de la aplicación. Al especificar las opciones de este middleware, no olvides incluir la opción `store` con la instancia de `MySQLStore` obtenida en el paso anterior, para que los datos de sesión se guarden en la BD.

Tras hacer esto, reinicia el servidor y comprueba que se haya creado una tabla vacía llamada `session` en tu base de datos. En esta tabla se almacenarán los atributos de sesión de cada usuario.

2 Entrada y salida en el sistema.

[1] Más información: `https://manuelmontenegro.github.io/AW-2017-18/05.html#/34/8`

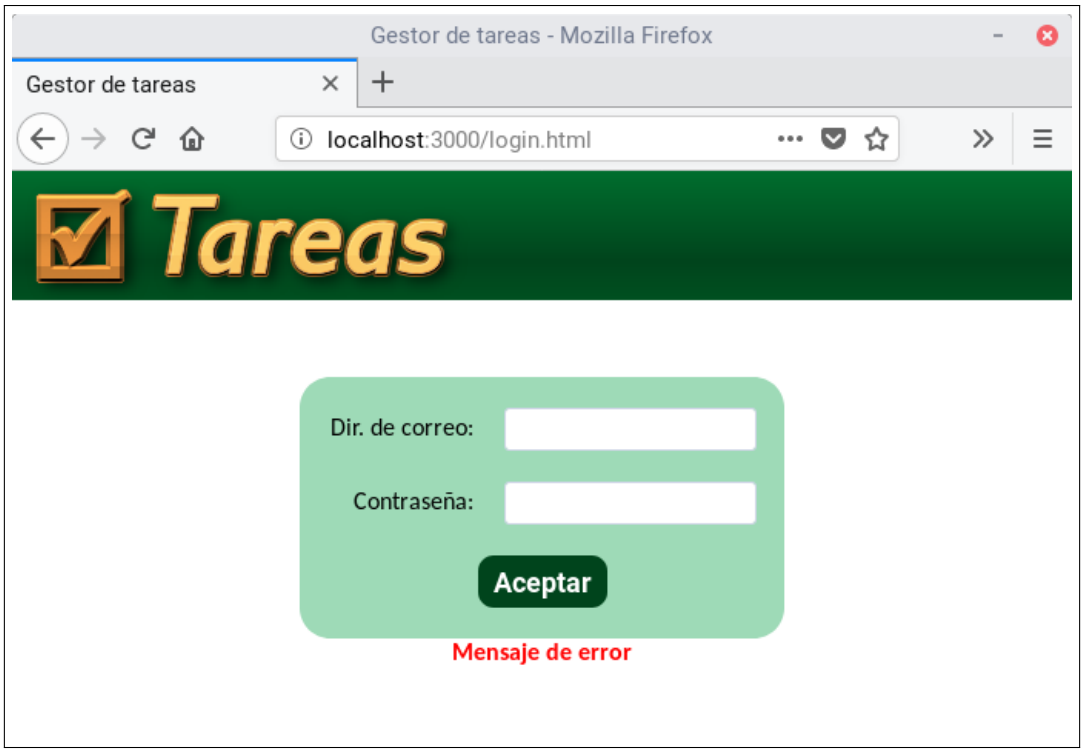

Figura 1: Página de identificación de usuario.

La página de entrada al sistema (`login.html`) ha de ser una página dinámica, ya que contiene un mensaje de error que deberá mostrarse o no en función de los datos de acceso que haya introducido el usuario. Por tanto, mueve el fichero `login.html` al directorio de plantillas `views`, y renómbralo a `login.ejs`. Modifica esta plantilla para que el mensaje de error dependa de una variable llamada `errorMsg`, que tomará el valor `null` en el caso en que no se quiera mostrar ningún mensaje de error.

Añade un manejador de ruta GET `/login.html` a tu aplicación, para que se muestre la vista `login` pasando el valor `null` a la variable `errorMsg`.

Cuando el usuario hace clic en el botón `[Aceptar]` del formulario de identificación, se realiza una petición POST a la ruta `/login` con la información introducida. Escribe el manejador de esta ruta para que se compruebe si la dirección de correo y contraseña introducidos se encuentran en la base de datos. Para realizar esta comprobación utiliza el método `DAOUsers.isUserCorrect()` (ver Apéndice).

- Si la dirección y contraseña son correctos, se debe establecer un atributo nuevo de sesión llamado `currentUser`, cuyo valor es la dirección de correo introducida. Después debe redirigirse a la ruta `/tasks`.
- En caso contrario, se debe volver a la vista `login` con el mensaje de error `Dirección de correo y/o contraseña no válidos`.

Dentro de la vista de tareas existe un enlace `[Desconectar]` que salta a la dirección `/logout`. Implementa el manejador de esta ruta para que se eliminen los atributos de sesión (utiliza, para ello, `request.session.destroy()`), y se redirija a la ruta `/login.html`.

Si quieres comprobar si los atributos de sesión se establecen correctamente, haz *login* con un nombre de usuario y contaaseña válidos. Debería añadirse una fila nueva a la tabla `sessions` de la base de datos. Tras hacer *logout*, esta fila debería eliminarse.

3 Middleware de control de acceso

Implementa un *middleware* que compruebe la existencia de un atributo llamado `currentUser` dentro de los atributos de sesión.

- Si se encuentra, el middleware debe añadir un nuevo atributo al objeto `response.locals` que contenga la dirección de correo del usuario actual. Por ejemplo, puedes llamar a este nuevo atributo `userEmail`[2]. Tras hacer esto, se debe saltar al siguiente middleware de la cadena.
- Si no se encuentra, se debe redirigir a la página `/login.html`, sin saltar al siguiente middleware.

Una vez hayas implementado este middleware, identifica todas aquellas rutas de tu aplicación que requieran identificación: `/tasks`, `/finish`, `/addTask`, `/deleteCompleted`. Añade este middleware a todas estas rutas, de modo que sólo se permita su acceso a aquellos usuarios que estén identificados en el sistema[3].

4 Información específica del usuario

En la entrega anterior, todas las llamadas a `getAllTasks()`, `insertTask()` y `deleteCompleted()` se realizaban sobre las tareas del usuario `usuario@ucm.es`, introducido "a pelo" dentro del código. Modifica estas llamadas para que el usuario sobre el cual se apliquen sea el que esté actualmente identificado en el sistema.

Además de esto, vamos a modificar la vista de tareas para que muestre el nombre del usuario actualmente identificado. Localiza, dentro de `tareas.ejs` el elemento `<div>` que contiene la cadena `usuario@ucm.es`. Sustituye esta dirección de correo estática por el valor de la variable `userEmail`.

5 Imagen de perfil

Por último, vamos a hacer que se muestre la imagen de perfil del usuario actualmente identificado. Hasta ahora teníamos la imagen de `usuario@ucm.es` almacenada dentro de la carpeta `/public/img`. En general, no es buena idea guardar las imágenes de perfil dentro de la carpeta `/public` de recursos estáticos. Es mejor tenerlas almacenadas en una carpeta separada. Por ello, crea una carpeta `profile_imgs` dentro del proyecto y mueve el fichero `img2315.png` (que es la imagen de perfil de `usuario@ucm.es`) a esta nueva carpeta.

A continuación, busca dentro de la vista `tareas.ejs` el elemento `<img>` que contiene la imagen de perfil. Sustituye el atributo `src="img/img2315.png"` por `src="/imagenUsuario"`. Ahora añade un nuevo manejador de ruta `/imagenUsuario` a tu aplicación. Este manejador debe:

[2] El objetivo de este atributo es poder hacer uso de la variable `userEmail` en todas las plantillas sin necesidad de pasarla explícitamente como modelo en cada llamada a la función `response.render()`. Recuerda que todos los atributos que insertemos en el objeto `response.locals` estarán automáticamente disponibles en todas las vistas EJS: `https://manuelmontenegro.github.io/AW-2017-18/05.html#/39`

[3] Recuerda que es posible añadir un middleware que afecte solamente a una determinada ruta: `https://manuelmontenegro.github.io/AW-2017-18/05.html#/23`

- Buscar en la BD el nombre de la imagen de perfil correspondiente al usuario actualmente identificado. Utiliza, para ello, la función `getUserImageName` del DAO de usuarios.
- Si el usuario no tiene imagen de perfil, la operación anterior devolverá `null`. En este caso llama al método `response.sendFile()` pasando como argumento la ruta al fichero `NoPerfil.jpg` que se encuentra en la carpeta `img` de los recursos estáticos.
- Si el usuario tiene imagen de perfil, utiliza el nombre de fichero devuelto para llamar al método `response.sendFile()`.

6 Apéndice: módulo auxiliar

Fichero `dao_users.js`:

```
/**
 * Proporciona operaciones para la gestión de usuarios
 * en la base de datos.
 */
class DAOUsers {
    /**
     * Inicializa el DAO de usuarios.
     *
     * @param {Pool} pool Pool de conexiones MySQL. Todas las operaciones
     *                    sobre la BD se realizarán sobre este pool.
     */
    constructor(pool) { ... }

    /**
     * Determina si un determinado usuario aparece en la BD con la contraseña
     * pasada como parámetro.
     *
     * Es una operación asíncrona, de modo que se llamará a la función callback
     * pasando, por un lado, el objeto Error (si se produce, o null en caso contrario)
     * y, por otro lado, un booleano indicando el resultado de la operación
    * (true => el usuario existe, false => el usuario no existe o la contraseña es incorrecta)
    * En caso de error error, el segundo parámetro de la función callback será indefinido.
     *
     * @param {string} email Identificador del usuario a buscar
     * @param {string} password Contraseña a comprobar
     * @param {function} callback Función que recibirá el objeto error y el resultado
     */
    isUserCorrect(email, password, callback) { ... }

    /**
     * Obtiene el nombre de fichero que contiene la imagen de perfil de un usuario.
     *
```

```
     * Es una operación asíncrona, de modo que se llamará a la función callback
     * pasando, por un lado, el objeto Error (si se produce, o null en caso contrario)
     * y, por otro lado, una cadena con el nombre de la imagen de perfil (o undefined
     * en caso de producirse un error). Si el usuario no tiene imagen de perfil, dicha
     * cadena tomará el valor null.
     *
     * @param {string} email Identificador del usuario cuya imagen se quiere obtener
     * @param {function} callback Función que recibirá el objeto error y el resultado
     */
    getUserImageName(email, callback) { ... }
}

module.exports = {
    DAOUsers: DAOUsers
}
```