Escuela Técnica Superior de
Ingeniería Informática

Lab. 08: Registro y autenticación

Introducción a la Ingeniería del Software y los Sistemas de Información II

Curso 2021/22

Daniel Ayala, Carlos Arévalo, Agustín Borrego, Margarita Cruz, Inma Hernández, David Ruiz

Índice

1. Objetivo

El propósito de esta práctica es aprender a realizar operaciones de registro, login y cierre de sesión en el cliente, haciendo uso de los endpoints provistos por el servidor a tal efecto. Además, se mostrarán conceptos básicos sobre gestión de la sesión en el cliente usando JavaScript. El alumno aprenderá a:

- Realizar el registro de un nuevo usuario en la aplicación.
- Almacenar en el cliente, mediante JavaScript, los datos del usuario con la sesión iniciada.
- Implementar un elemento para cerrar la sesión.
- Mantener y gestionar tokens de sesión para poder usar endpoints protegidos.
- Mostrar y usar datos del usuario actual en las diferentes vistas.
- Adaptar las vistas en función de si el usuario está autenticado o no.

2. Introducción

La gestión de sesiones de usuarios es, sin duda, un aspecto fundamental en cualquier aplicación Web. Este aspecto varía en gran medida según el framework Web usado, aunque algunos conceptos básicos son de aplicación general.

En esta práctica dotaremos de funcionalidad al formulario de registro implementado anteriormente, y aprenderemos a realizar una gestión básica de las sesiones de usuario en cliente.

3. Configuración de Silence

El framework Silence provee por defecto dos endpoints para realizar estas operaciones:

- `/login` recibe los datos de inicio de sesión de un usuario (identificador y contraseña).
- `/register` recibe todos los datos de un usuario y lo da de alta en la apliación, almacenando su contraseña de manera segura en la BD.

Estos dos endpoints devuelven **la misma respuesta** JSON: un token de sesión en un atributo llamado "sessionToken" y los datos completos del usuario logueado o registrado en un atributo llamado "user". **En esta práctica implementaremos únicamente el registro, dado que la gestión del inicio de sesión es idéntica**. El cierre de sesión, por otro lado, se efectúa únicamente por parte del cliente (navegador), eliminando de la memoria local el token de sesión almacenado.

Para que funcionen correctamente los dos endpoints anteriormente descritos es fundamental configurar adecuadamente el parámetro `USER_AUTH_DATA` del proyecto, especificando el nombre de la tabla que contiene los usuarios, así como las columnas que se usan como identificador y contraseña respectivamente.

4. Módulo API de autenticación

Al igual que para todas las interacciones que hemos hecho hasta el momento con la API del proyecto, las operaciones de login y registro las efectuaremos haciendo uso de un módulo JavaScript destinado a interactuar con estos endpoints. Por ejemplo, podemos implementar las operaciones POST de login y registro en un módulo `js/api/auth.js`:

```
"use_strict";

import { BASE_URL, requestOptions } from "./common.js";

const authAPI = {

    login: async function(formData) {
       let response = await axios.post(`${BASE_URL}/login`, formData, requestOptions);
       return response.data;
    },

    register: async function(formData) {
       let response = await axios.post(`${BASE_URL}/register`, formData,
           requestOptions);
       return response.data;
    },

};

export { authAPI };
```


5. Módulo de gestión de sesiones

El proyecto descargado incluye un módulo encargado de ofrecer métodos para facilitar la gestión de las sesiones en el navegador, que se puede encontrar en `js/utils/session.js`. Los métodos que ofrece el objeto `sessionManager` exportado por el módulo son:

- `login(token, user)` inicia sesión con el token de sesión y los datos de usuario proporcionados.
- `logout()` cierra la sesión activa.
- `getToken()` devuelve el token de sesión actual, o `null` si la sesión no está activa o ha caducado.
- `isLogged()` devuelve un boolean indicando si hay una sesión activa o no.
- `getLoggedUser()` devuelve los datos del usuario actual, o `null` si la sesión no está activa.
- `getLoggedId()` devuelve el `userId` del usuario actual, o `null` si la sesión no está activa.

Internamente, este módulo hace uso del `localStorage` del navegador para almacenar localmente datos recibidos del servidor, tales como el token de sesión que identifica a una sesión activa, y que es necesario para acceder a endpoints protegidos. En las siguientes secciones, veremos cómo usar este módulo de manera prática.

6. Registro de usuario

En prácticas anteriores, implementamos un formulario de registro en HTML en la vista `register.html` y lo dotamos de validación en `register.js`. Ahora, es hora de modificar este último fichero para realizar una petición a la API de registro y usar el resultado para iniciar una sesión.

Comenzaremos por importar en `register.js` tanto el módulo de API creado anteriormente como el módulo de gestión de sesiones:

```
import { sessionManager } from "/js/utils/session.js";
import { authAPI } from "/js/api/auth.js";
```


La función que realiza la validación del formulario es `handleSubmitRegister`, que se ejecuta cada vez que el usuario envía el formulario. Modificaremos esta función para que, si la validación es exitosa (no hay errores), se envíe el formulario haciendo uso del módulo de API correspondiente, usando una función aparte. Si no, se le mostrarán al usuario los errores:

```
function handleSubmitRegister(event) {
    event.preventDefault();
    let form = event.target;
    let formData = new FormData(form);

    let errors = userValidator.validateRegister(formData);

    if (errors.length > 0) {
        let errorsDiv = document.getElementById("errors");
        errorsDiv.innerHTML = "";

        for (let error of errors) {
            messageRenderer.showErrorAsAlert(error);
        }
    } else {
        sendRegister(formData);
    }
}
```

En la función `sendRegister`, que hemos de definir, recibiremos los datos del formulario ya validados y los enviaremos al endpoint de registro. Esta función será **asíncrona**, ya que haremos en ella una petición a la API. Inicialmente, mostraremos por consola los datos recibidos del servidor, para poder inspeccionar su contenido:

```
async function sendRegister(formData) {
    try {
        let loginData = await authAPI.register(formData);
        console.log(loginData);
    } catch (err) {
        messageRenderer.showErrorAsAlert("Error registering a new user", err);
    }
}
```

Si creamos un nuevo usuario mediante el formulario, aparecerá la respuesta del servidor en la consola:

Como se explicó en la Sección 3, esta respuesta tiene dos atributos: “sessionToken”, que contiene el token de sesión, y “user”, que contiene los datos del usuario. Estos dos parámetros son los requeridos por el método `login()` del módulo de gestión de sesiones, explicado en la Sección 5. Así, podemos guardarlos como variables y usarlos para iniciar sesión:

```
async function sendRegister(formData) {
    try {
        let loginData = await authAPI.register(formData);
        let sessionToken = loginData.sessionToken;
        let loggedUser = loginData.user;

        sessionManager.login(sessionToken, loggedUser);
        window.location.href = "index.html";
    } catch (err) {
        messageRenderer.showErrorAsAlert("Error registering a new user", err);
    }
}
```

Si el registro es exitoso, iniciaremos la sesión del usuario creado y redirigiremos al usuario a la página principal. Si no, mostraremos el error correspondiente.

Recuerde que los nombres de los atributos de los usuarios se deben corresponder con los valores de los atributos “name” de los `<input>` del formulario.

7. Actualizando la barra de navegación

Tras implementar el inicio de sesión, existen varias modificaciones que podemos hacer en la barra de navegación:

7.1. Mostrar el usuario actual

Las modificaciones anteriores son suficientes para registrar un usuario e iniciar sesión con él, pero no hay ningún tipo de feedback visual que indique al usuario si éste tiene una sesión iniciada o no. Para lograr este propósito, dotaremos de dinamismo al texto que aparece a la izquierda en la barra de navegación:

- Si hay una sesión iniciada, mostraremos un saludo y el nombre de usuario.
- Si no, mostraremos “Anónimo”.

El elemento en cuestión es el enlace `<a>` con `class="navbar-brand"` que se encuentra en el archivo HTML de la cabecera (`header.html`). Para poder acceder a él mediante JS, le daremos un ID:

```
<a class="navbar-brand" id="navbar-title" href="#"></a>
```

A continuación, crearemos un archivo `header.js` para realizar operaciones en la cabecera, con la estructura habitual:

```
"use strict";

function main() {
    ...
}

document.addEventListener("DOMContentLoaded", main);
```

Deberá enlazar `header.js` en todas las vistas de su aplicación, junto con el resto de archivos JavaScript:

```
<script src="js/header.js" type="module"></script>
```

En la función `main()`, comprobaremos si hay una sesión iniciada y, en ese caso, accederemos al nombre de usuario actual, usando el módulo `session.js`:

```
import { sessionManager } from "/js/utils/session.js";

function main() {
    showUser();
}

function showUser() {
    let title = document.getElementById("navbar-title");
    let text;

    if (sessionManager.isLogged()) {
        let username = sessionManager.getLoggedUser().username;
        text = "Hi, @" + username;
    } else {
        text = "Guest";
    }

    title.textContent = text;
}
```

7.2. Cierre de sesión

Es habitual incluir en la barra de navegación un botón para cerrar la sesión rápidamente. En nuestro caso, podemos hacerlo añadiendo un nuevo elemento a la barra, con un ID determinado para poder darle funcionalidad mediante JavaScript:

```
<li class="nav-item"><a class="nav-link" href="#" id="navbar-logout">Logout</a></li>
```

Podemos implementar una función que se lance cuando el usuario pulse en él, y cierre su sesión enviándolo a la página de inicio:

```
function main() {
    showUser();
    addLogoutHandler();
}

function addLogoutHandler() {
    let logoutButton = document.getElementById("navbar-logout");

    logoutButton.onclick = function () {
        sessionManager.logout();
        window.location.href = "index.html";
    };
}
```


7.3. Ocultar opciones no disponibles

Actualmente, nuestra aplicación muestra al usuario todas las opciones disponibles con independencia de si está logueado o no. Mediante JavaScript podemos ocultar aquellos elementos de la barra de navegación que no deben mostrarse a un usuario invitado, como por ejemplo, el cierre de sesión o el enlace al formulario de crear foto. Asimismo, le ocultaremos a un usuario logueado opciones como el login y el registro.

Como es habitual, debemos darle IDs a los elementos de la barra de navegación para poder acceder a ellos mediante JS y ocultarlos si es necesario:

```
<ul class="navbar-nav me-auto mb-2 mb-lg-0">
    <li class="nav-item">
        <a class="nav-link" href="index.html" id="navbar-recent">Recent photos</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="edit_photo.html" id="navbar-create">Upload photo</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="register.html" id="navbar-register">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="login.html" id="navbar-login">Login</a>
    </li>
    <li class="nav-item dropdown" id="navbar-trending">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button"
             data-bs-toggle="dropdown"
            aria-expanded="false">
            Trending photos
        </a>
        <ul class="dropdown-menu bg-dark" aria-labelledby="navbarDropdown">
            <li><a class="dropdown-item text-light" href="trending_users.html">
                Trending users</a></li>
            <li><a class="dropdown-item text-light" href="trending_photos.html">
                Trending photos</a></li>
        </ul>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="#" id="navbar-logout">Logout</a>
    </li>
</ul>
```

En la función `main()`, llamaremos por último a una función auxiliar que ocultará los elementos adecuados según el estado de la sesión del usuario:

```
function main() {
    showUser();
    addLogoutHandler();
    hideHeaderOptions();
}

function hideHeaderOptions() {
    let headerRegister = document.getElementById("navbar-register");
    let headerLogin = document.getElementById("navbar-login");
    let headerLogout = document.getElementById("navbar-logout");
    let headerRecent = document.getElementById("navbar-recent");
    let headerCreate = document.getElementById("navbar-create");
    let headerTrending = document.getElementById("navbar-trending");

    if (sessionManager.isLogged()) {
        headerRegister.style.display = "none";
        headerLogin.style.display = "none";
    } else {
        headerRecent.style.display = "none";
        headerCreate.style.display = "none";
        headerLogout.style.display = "none";
        headerTrending.style.display = "none";
    }
}
```

Nota: Ocultar enlaces y otros elementos no es, en sí mismo, una medida de seguridad, ya que el usuario avanzado puede acceder de todos modos introduciendo la URL a mano o usando herramientas como REST Client. La seguridad radica en configurar adecuadamente el back-end para que el servidor no permita a un usuario realizar operaciones que no le corresponden, por ejemplo, limitándolas a usuarios autenticados o que tengan un determinado rol.

8. Ocultando elementos en otras vistas

Al igual que hemos hecho con las opciones de la barra de navegación, existen elementos en otras vistas que debemos ocultar a usuarios no autenticados, ya que si intentaran interactuar con ellos se les mostraría un error por falta de permisos. Uno de esos casos son los botones de edición y modificación de fotos en la vista de detalle de foto, así como el formulario para emitir una valoración, dado que esas operaciones están reservadas a usuarios autenticados.

La forma de proceder es similar a la anterior: se le debe dar a esos elementos una ID, y posteriormente, en el código JS asociado a la vista, ocultarlos si el usuario no tiene la sesión

iniciada. Por ejemplo, en `photo_detail.html` le daremos un ID a la columna que contiene estos elementos:

```
<div class="col-md-3" id="actions-col">
    <p>
        <button id="button-edit" class="btn btn-primary">Edit this photo</button>
        <button id="button-delete" class="btn btn-danger">Delete this photo</button>
    </p>
    ...
</div>
```

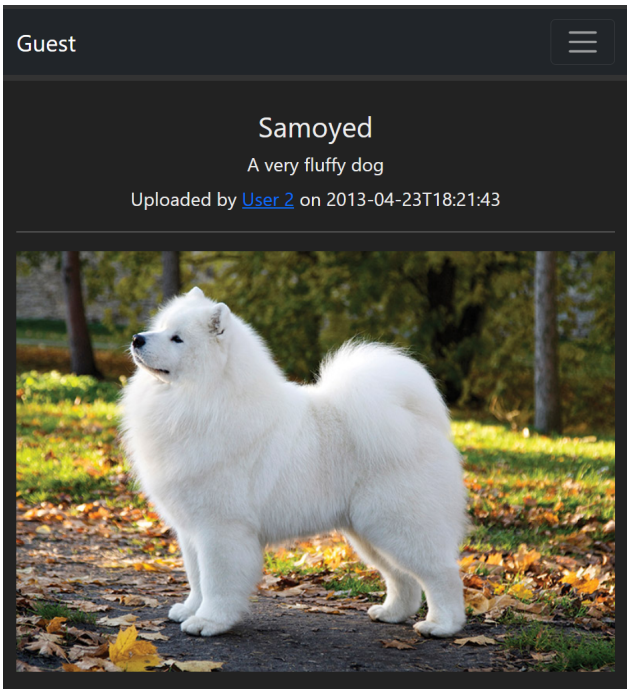
En el archivo JS asociado, `photo_details.js`, eliminaremos este elemento si no hay una sesión iniciada:

```
import { sessionManager } from "/js/utils/session.js";

async function main() {
    ...
    hideActionsColumn();
}

function hideActionsColumn() {
    let actions_col = document.getElementById("actions-col");
    if (!sessionManager.isLogged()) {
        actions_col.style.display = "none";
    }
}
```

Observe que, si se elimina esta columna, la anterior pasa a tener todo el ancho de la página, optimizando así el uso de la vista:



9. Subida de fotos

En la práctica anterior, dejamos un elemento pendiente en la gestión del formulario de subida de fotos: las nuevas fotos se asignaban al ID de un usuario prestablecido, en lugar de aquel que tenía la sesión iniciada. Ahora que tenemos control sobre las sesiones de los usuarios, podemos usar el módulo de sesiones para acceder al ID del usuario actual.

Realizaremos entonces la siguiente modificación en `edit_photo.js`, reemplazando el ID del usuario que introdujimos manualmente por la función que obtiene el ID del usuario logueado:

```
import { sessionManager } from "/js/utils/session.js";

async function handleSubmitPhoto(event) {
    ...

    if (currentPhoto === null) { // Creating a new photo
        // Add the current user's ID
        formData.append("userId", sessionManager.getLoggedId());
    ...
}
```

Recuerde que la creación de fotos es una operación reservada a usuarios logueados, por

lo que debe configurar su endpoint adecuadamente.

Como sabe, los endpoints configurados con `auth_required: true` requieren el envío del token de sesión para su uso. Sin embargo, los módulos API que hemos creado en las sesiones anteriores envían siempre el token de sesión almacenado en caso de que esté disponible, gracias al objeto común `requestOptions`. Por ello, comprobará que no es necesario que realice ningún cambio adicional para que la creación de fotos se comporte como se espera: si está autenticado podrá crear, editar y borrar fotos; mientras que si no tiene la sesión iniciada se mostrará un mensaje de error.

10. Actualización en GitHub

Actualice su proyecto en GitHub con los cambios hechos durante esta sesión. Recuerde los comandos relevantes:

- `git add .` añade los cambios efectuados en todos los archivos al próximo *commit* a efectuar.
- `git commit -m ‘‘mensaje’’` crea un nuevo *commit* con los cambios efectuados a los archivos añadidos con el comando anterior, y con el mensaje indicado.
- `git push` actualiza el repositorio remoto con los cambios efectuados.