Lab7 - Peticiones AJAX POST, PUT y DELETE

Pepe Calderón, Fernando Sola, Daniel Ayala, Inma Hernández, Margarita Cruz, Carlos Arévalo, David Ruiz

Escuela Técnica Superior de
Ingeniería Informática

Índice

1 Objetivo

El objetivo de esta práctica es profundizar en la comunicación del back-end con el front--end mediante peticiones AJAX que modifiquen el estado de la base de datos. El alumno aprenderá a:

- Usar la librería JavaScript "Axios" para realizar peticiones AJAX de tipo POST, PUT y DELETE a los endpoints del proyecto, mediante el enfoque orientado a módulos JavaScript aprendido en la práctica anterior.
- Realizar envíos de formulario por la vía indicada en el punto anterior.

2

Introducción

En la práctica anterior, realizamos una comunicación básica con el back-end de nuestra aplicación, realizando consultas GET para obtener datos almacenados en la base de datos y poder mostrarlos en las diferentes pantallas. En esta sesión profundizaremos en este aspecto realizando otro tipo de operaciones, tales como la creación, modificado y borrado de recursos usando métodos POST, PUT y DELETE respectivamente.

Para ello, implementaremos una nueva vista con un formulario que servirá tanto para subir nuevas fotos como para modificar una foto existente. Además, dotaremos de funcionalidad a los botones de "Editar foto" y "Borrar foto" ubicados en la vista de detalle de foto.

3

Formulario de edición de foto

Para poder crear y modificar fotos, necesitaremos una vista que contenga un formulario para poder introducir los datos propios de cada foto, a saber:

- URL de la imagen
- Título
- Descripción
- Visibilidad (pública / privada)

Para ello, crearemos un archivo `edit_photo.html` en el que, dentro de la plantilla usual para documentos HTML5, crearemos el formulario correspondiente:

```
<form id="form-photo-upload">
    <div class="form-group">
        <label for="input-url">URL:</label>
        <input required type="text" class="form-control" id="input-url"
name="url"
            placeholder="Photo URL">
    </div>

    <div class="form-group">
        <label for="input-title">Title:</label>
        <input required type="text" class="form-control" id="input-title"
name="title"
            placeholder="Title">
    </div>

    <div class="form-group">
        <label for="input-description">Description:</label>
```

```
        <textarea class="form-control" id="input-description"
name="description"
            placeholder="Describe your image..." rows="4"></textarea>
    </div>

    <div class="form-group">
        <label for="input-visibility">Visibility:</label>
        <select required class="form-control" id="input-visibility"
name="visibility">
            <option value="Public">Public</option>
            <option value="Private">Private</option>
        </select>
    </div>

    <div class="row">
        <div class="col-md text-center">
            <button type="submit" class="btn btn-primary">Send</button>
        </div>
    </div>
</form>
```

{: .warning }

> **Info**
>
> **Nota:** Se recomienda añadir un enlace a `edit_photo.html` en la barra de navegación para su fácil acceso.

4

Creación de fotos con POST

Tras estos cambios en el proyecto, estamos listos para implementar la creación de nuevos fotos en el formulario creado. Como es habitual, crearemos un archivo JavaScript con el mismo nombre de la vista en cuestión, en este caso, `edit_photo.js`:

```
"use strict";

import { photosAPI } from "/js/api/photos.js";
import { messageRenderer } from "/js/renderers/messages.js";

async function main() {
    ...
}

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

Crearemos una función destinada a gestionar el envío del formulario de creación de foto, que tiene ID `form-photo-upload`:

```
async function main() {
    let registerForm = document.getElementById("form-photo-upload");
    registerForm.onsubmit = handleSubmitPhoto;
}

function handleSubmitPhoto(event) {
    ...
}
```

En esta función, crearemos un objeto FormData a partir del formulario que está siendo enviado, y usaremos el módulo de API para enviarlo mediante una petición POST. Si la petición tiene éxito, redirigiremos al usuario de nuevo a la página principal para que pueda ver la nueva foto creada. Si hay algún fallo, mostraremos el mensaje:

```
async function handleSubmitPhoto(event) {
    event.preventDefault();

    let form = event.target;
    let formData = new FormData(form);

    // Add the current user ID
    formData.append("userId", 1);

    try {
        let resp = await photosAPI_auto.create(formData);
        let newId = resp.lastId;
        window.location.href = `photo_detail.html?photoId=${newId}`;
    } catch (err) {
        messageRenderer.showErrorMessage(err.response.data.message);
    }
}
```

Son importantes las siguientes consideraciones adicionales:

- La primera línea, `event.preventDefault()`, evita que el navegador envíe por su cuenta el formulario, ya que lo estamos haciendo nosotros mediante Axios.
- Un atributo de la foto es el ID del usuario que la sube, información que no se puede obtener del formulario. En el siguiente laboratorio aprenderemos a gestionar todo lo relacionado con sesiones, por ahora proporcionaremos un valor por defecto.
- El método `.append()` de un objeto FormData permite añadir un par clave/valor a los datos a enviar.
- La página actual se guarda en `window.location.href`. Si cambiamos su valor, hacemos que el navegador vaya a otra página.
- En la implementación proporcionada no se realiza ninguna validación del formulario. Deberá implementar la validación que considere necesaria, y únicamente realizar la petición POST si no se ha producido ningun error.
- Podemos acceder a los datos devueltos por el servidor tras crear la foto, y usar la ID devuelta para dirigir al usuario a la vista de detalle de la foto que acaba de crear.

{: .warning }

> **Info**
>
> Si ha establecido `auth_required` a `true` en el endpoint de creación de foto se mostrará un error al enviar el formulario, ya que no se ha iniciado sesión. En la siguiente sesión de laboratorio resolveremos este problema, por el momento, puede establecerlo a `false`.

5 Edición y borrado de fotos

En la vista de detalle de fotos, `photo_detail.html`, existen dos botones para editar y modificar la foto en cuestión, pero hasta el momento no tenían ninguna funcionalidad.

Para asignarles funciones manejadoras de eventos al pulsar en ellos, primero debemos darles un ID a cada uno:

```
<button id="button-edit" class="btn btn-primary">Edit this photo</button>
<button id="button-delete" class="btn btn-danger">Delete this photo</
button>
```

Entonces, editaremos el archivo `photo_detail.js` para añadirle funciones manejadoras a los eventos `click` de ambos botones. En el caso del botón de borrado, crearemos una función que pregunte al usuario si realmente desea eliminar la foto mediante la función `confirm()`. En el caso de una respuesta afirmativa, usaremos el módulo de API para emitir una petición DELETE y devolveremos al usuario a la página principal:

```
async function main() {
    // Assign the handler function to the delete button
    let deleteBtn = document.querySelector("#button-delete");
    deleteBtn.onclick = handleDelete;
}

async function handleDelete(event) {
    let answer = confirm("Do you really want to delete this photo?");

    if (answer) {
        try {
```

```
            await photosAPI_auto.delete(photoId);
            window.location = "/index.html";
        } catch (err) {
            messageRenderer.showErrorMessage(err.response.data.message);
        }
    }
}
```

Como observará, dado que en la práctica anterior definimos `photoId` en el ámbito global, podemos usarlo en la función `handleDelete` para saber el ID de la foto que estamos mostrando.

Para editar la foto, redirigiremos al usuario a un formulario de edición de foto. En principio, podríamos estar tentados a crear una nueva vista para editar una foto. Sin embargo, ya contamos con un formulario que contiene todo lo necesario para modificar los campos de una foto: el formulario de creación de foto. Podemos aprovechar esta vista para darle dos propósitos y evitar tener vistas duplicadas:

- Si `edit_photo.html` no recibe ningún parámetro de URL, lo usaremos para crear una nueva foto.
- Si recibe un `photoId` como parámetro de URL (p.ej. `edit_photo.html?photoId=6`), lo usaremos para modificar la foto en cuestión.

En ese caso, el botón para editar foto de `photo_detail.js` se limitará a redireccionar a esta vista, proporcionando el ID de foto correspondiente a la foto actual:

```
async function main() {
    ...
    let editBtn = document.querySelector("#button-edit");
    editBtn.onclick = handleEdit;
}

function handleEdit(event) {
    window.location.href = "edit_photo.html?photoId=" + photoId;
}
```

En la siguiente sección realizaremos las modificaciones necesarias en el formulario de creación de foto para permitir editar una foto.

5.1. Actualizando la vista de creación de fotos

Nos centraremos ahora en `edit_photo.html` y su archivo JS asociado, `edit_photo.js`. Como explicamos anteriormente, debemos determinar si hemos recibido un `photoId` (en ese caso, editaremos una foto existente) o si no lo hemos recibido (en ese caso, crearemos una foto nueva). Podemos usar el mismo código usado en `photo_details.js` para capturar los parámetros de URL recibidos y obtener uno de ellos:

```
let urlParams = new URLSearchParams(window.location.search);
let photoId = urlParams.get("photoId");
let currentPhoto = null;
```

Además, usaremos una variable `currentPhoto` para almacenar los atributos de la foto que estamos editando, si es el caso. Si no, esta variable permanecerá como `null`.

En la función `main()` comprobaremos si hemos recibido algún parámetro de URL. Si es el caso, realizaremos las siguientes operaciones:

- Modificar el título de la página para que sea "Editando una foto" en lugar de "Creando una nueva foto".
- Consultar la foto con el ID recibido a la API.
- Almacenar la foto en la variable `currentPhoto`.
- Editar los campos del formulario para establecer sus valores iniciales a aquellos que tenga la foto que estamos editando.

```
async function main() {
    if (photoId !== null) {
        loadCurrentPhoto();
    }

    ...
}

async function loadCurrentPhoto() {
    let pageTitle = document.getElementById("page-title");
    let urlInput = document.getElementById("input-url");
    let titleInput = document.getElementById("input-title");
    let descriptionInput = document.getElementById("input-description");
    let visibilityInput = document.getElementById("input-visibility");

    pageTitle.textContent = "Editing a photo";

    try {
        currentPhoto = await photosAPI_auto.getById(photoId);
        urlInput.value = currentPhoto.url;
        titleInput.value = currentPhoto.title;
        descriptionInput.value = currentPhoto.description;
        visibilityInput.value = currentPhoto.visibility;
    } catch (err) {
        messageRenderer.showErrorMessage(err.response.data.message);
    }
}
```

Esto provocará que, si estamos editando una foto, ya aparezcan cargados todos los atributos de la foto en los inputs correspondientes. Finalmente, debemos modificar la función encargada del envío del formulario, `handleSubmitPhoto`, para realizar una operación u otra dependiendo de si estamos editando una foto o creando una nueva:

```
async function handleSubmitPhoto(event) {
        event.preventDefault();

        let form = event.target;
        let formData = new FormData(form);

        if (currentPhoto === null) { // Creating a new photo
            // Add the current user ID
            formData.append("userId", 1);

            try {
                let resp = await photosAPI_auto.create(formData);
                let newId = resp.lastId;
                window.location.href = `photo_detail.html?photoId=${newId}
`;
            } catch (err) {
messageRenderer.showErrorAsAlert(err.response.data.message);
            }
        } else { // Updating an existing photo
            formData.append("userId", currentPhoto.userId);
            formData.append("date", currentPhoto.date);

            try {
                await photosAPI_auto.update(formData, photoId);
                window.location.href = `photo_detail.html?
photoId=${photoId}`;
            } catch (err) {
messageRenderer.showErrorAsAlert(err.response.data.message);
            }
        }
    }
```

Si estamos editando una foto, `currentPhoto` no será `null`, y podemos acceder a atributos como el usuario que subió la foto y la fecha de subida para añadirlos al formulario.

6

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.

> **Info**
>
> Versión PDF disponible