FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico

Base de datos
Segundo cuatrimestre de 2024

Alumno:	Padrón	Mail
Nicolas Funes Cabanelas	109830	nfunes@fi.uba.ar
Matias Morales	106793	matmorales@fi.uba.ar

Índice

1. Introducción

Este trabajo práctico busca integrar bases de datos relacionales y NoSQL mediante el desarrollo de una aplicación web interactiva. Utilizamos **SQLite** para manejar datos estructurados, como información de estudiantes, y **MongoDB** para almacenar datos flexibles, como exámenes académicos. La interfaz web permite a los usuarios realizar operaciones CRUD en ambas bases de datos, brindando una experiencia práctica en la gestión de datos.

2. Objetivos del trabajo

- Comprender las diferencias entre bases de datos relacionales y NoSQL.
- Desarrollar una aplicación que integre ambas tecnologías en un entorno web.
- Implementar operaciones CRUD en ambas bases de datos.
- Diseñar una interfaz web funcional y responsiva para gestionar los datos.
- Evaluar las ventajas y desventajas de cada tipo de base de datos en el contexto de este proyecto.

3. Elección de las Tecnologías

3.1. Frontend

Para el desarrollo del frontend utilizamos:

- **HTML5**: Para estructurar las vistas.
- **CSS**: Para estilizar la interfaz y garantizar una experiencia visual atractiva.
- **JavaScript**: Para la interacción dinámica, incluyendo las operaciones CRUD mediante `fetch`.
- **SweetAlert2**: Biblioteca para mostrar mensajes emergentes personalizados, como alertas de éxito o error.

3.2. Backend

El backend fue desarrollado en **Node.js** utilizando el framework **Express**, que permite construir aplicaciones web y APIs de forma sencilla y escalable. Se usaron las siguientes bibliotecas:

- **sqlite3**: Para interactuar con SQLite.
- **mongodb** y **GridFSBucket**: Para manejar la base de datos NoSQL y gestionar archivos grandes como imágenes o PDFs.
- **express-fileupload**: Para permitir la carga de archivos en el servidor.

3.3. Bases de Datos

3.3.1. Elección de SQLite

La elección de **SQLite** para manejar la base de datos de estudiantes fue motivada por su simplicidad y eficiencia en el manejo de datos estructurados. SQLite es ideal para operaciones que requieren una estructura bien definida y relaciones entre los datos, como el registro de estudiantes con campos como `nombre`, `mail` y `padron`. Su implementación es rápida y fácil de mantener, lo que permite gestionar registros con integridad referencial y sin complicaciones.

3.3.2. Elección de MongoDB

Por otro lado, **MongoDB** fue la elección para almacenar la base de datos de exámenes debido a su flexibilidad para manejar datos no estructurados y grandes volúmenes de información, como archivos PDF o imágenes. MongoDB, en combinación con GridFS, es perfecto para almacenar y servir archivos de gran tamaño de forma eficiente. Esta base de datos se adapta bien a la naturaleza variable de los datos de los exámenes, como el `id_materia` y la `fecha`, permitiendo una rápida adaptación a cambios en el esquema de los datos.

SQLite	padron: TEXT (PK)
	nombre: TEXT
usuarios	mail: TEXT

MongoDB	id_materia: String
	fecha: Date
examenes	examen: File (GridFS)

Figura 1: Estructura y elementos de las bases de datos utilizadas.

4. Diagrama de Arquitectura

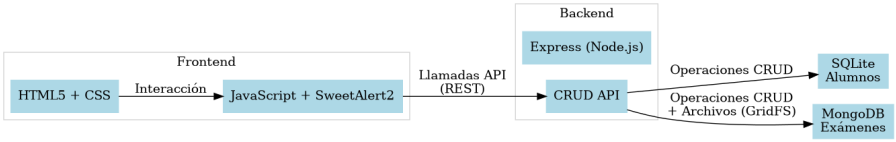

Figura 2: Diagrama de arquitectura de la aplicación.

5. Configuración y Conexión a Bases de Datos

5.1. SQLite

Para SQLite, se configuró una tabla llamada `usuarios` con los campos `padron` (clave primaria), `nombre` y `mail`. Las operaciones CRUD se implementaron mediante consultas SQL directas.

5.2. MongoDB

En MongoDB, se configuró una colección llamada `examenes`, que almacena:

- `id_materia`: Identificador de la materia.
- `fecha`: Fecha del examen.
- `examen`: Archivo subido y almacenado en `GridFS`.

6. Funciones CRUD

6.1. SQLite (Estudiantes)

En la base de datos SQLite, las operaciones CRUD se implementaron con las siguientes características:

- **Crear**: Los usuarios pueden agregar nuevos estudiantes a través de un formulario en la interfaz web. Los datos ingresados se envían al servidor, que los procesa y los inserta en la base de datos con una consulta SQL `INSERT INTO`.

- **Leer**: La aplicación permite consultar todos los estudiantes almacenados. Esto se realiza con una consulta SQL `SELECT * FROM`, y los resultados se muestran en una tabla en la interfaz web.
- **Actualizar**: La modificación de los datos de un estudiante se realiza seleccionando el registro específico. Los datos actualizados se envían al servidor, donde se ejecuta una consulta SQL `UPDATE` para reflejar los cambios en la base de datos.
- **Eliminar**: Los estudiantes se pueden borrar por su `padron` mediante una solicitud de eliminación. La consulta SQL `DELETE FROM` se utiliza para remover el registro de la base de datos.

6.2. MongoDB (Exámenes)

En la base de datos MongoDB, se implementaron las siguientes funciones CRUD:

- **Crear**: Los usuarios pueden agregar un examen mediante un formulario en la interfaz web, ingresando el `id_materia`, la `fecha` y el archivo asociado. Este archivo se almacena en GridFS, y la información del examen se guarda en la colección `examenes` con una operación `insertOne()`.
- **Leer**: Se implementó la capacidad de listar y filtrar exámenes almacenados por `id_materia` y `fecha`. La función utiliza `find()` para recuperar los documentos de la colección y mostrarlos en la interfaz.
- **Actualizar**: La actualización de un examen se realiza mediante una solicitud PUT, que envía los datos actualizados y, si es necesario, un nuevo archivo. Se usa `updateOne()` para modificar los detalles en la colección y se reemplaza el archivo en GridFS si se sube uno nuevo.
- **Eliminar**: Para eliminar un examen, se utiliza `deleteOne()` en la colección `examenes` y la función de eliminación de archivos en GridFS para remover el archivo correspondiente.

7. Interfaz de Usuario

Figura 3: Interfaz del sector de la base de datos de alumnos.

Observamos que también hemos añadido la funcionalidad para buscar alumnos por medio de su padrón, lo cual funciona por medio de un campo de búsqueda que filtra dinámicamente los resultados en la tabla. Esta funcionalidad permite una experiencia de usuario más eficiente y agiliza la búsqueda de registros específicos sin necesidad de recargar la página, mejorando así la interactividad y usabilidad de la aplicación.

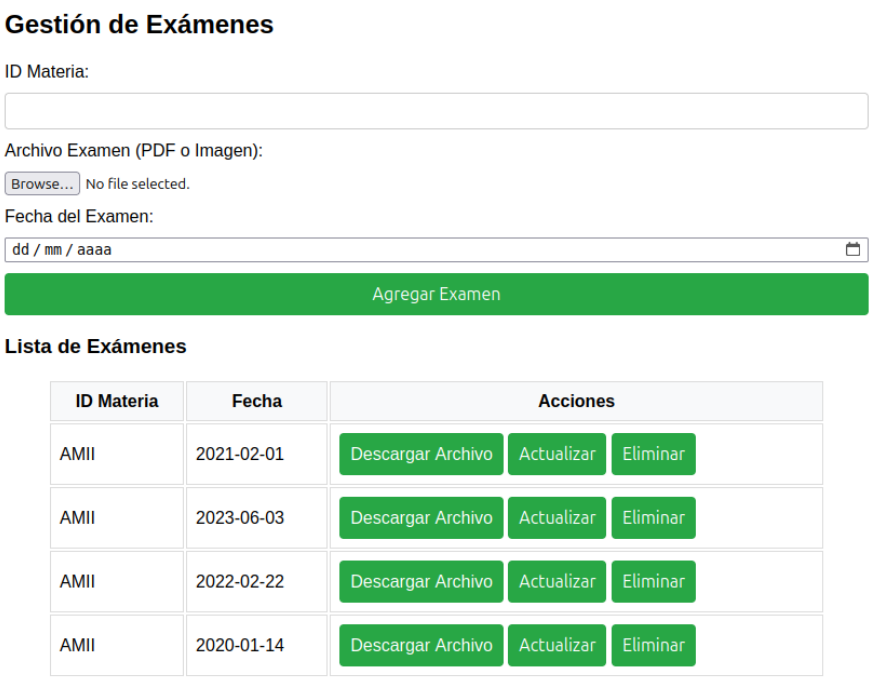

Figura 4: Interfaz de la base de datos de exámenes.

Además, para que sea aún mejor la experiencia del usuario, se implementaron una serie de alertas que se lanzan al momento en que el mismo ejecuta las distintas acciones. A continuación, solo algunas de ellas.

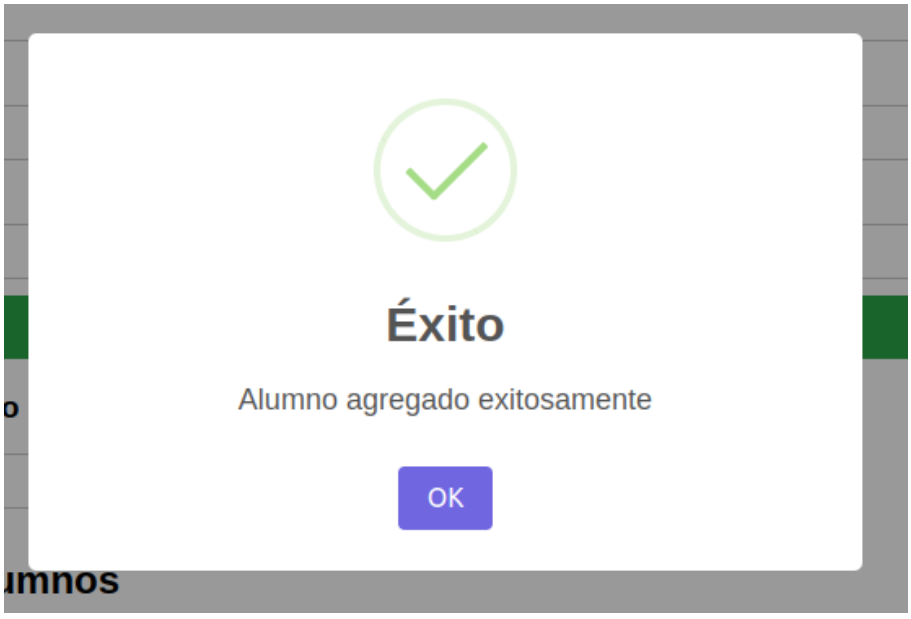

Figura 5: Alerta que aparece al agregar con éxito un elemento a la base de datos.

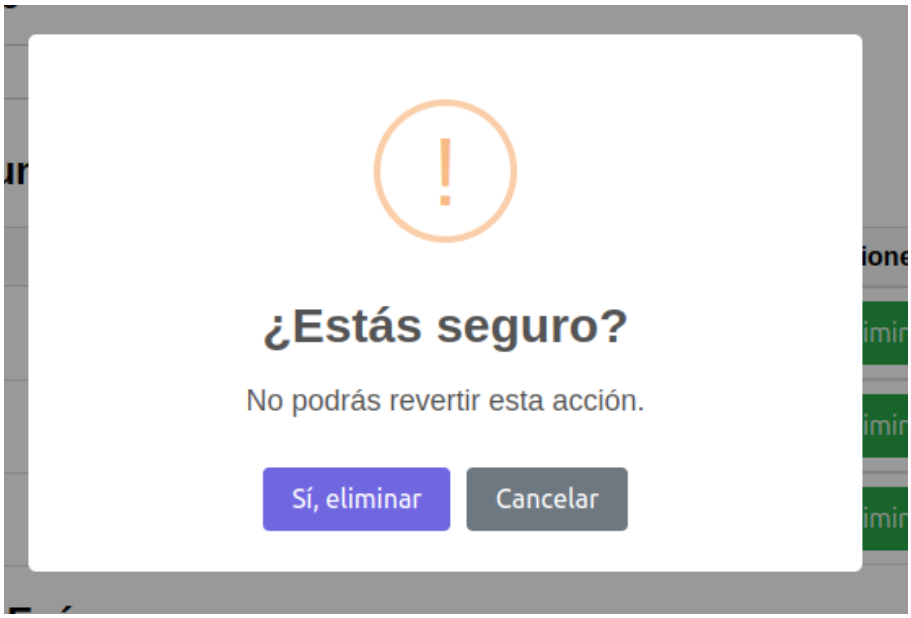

Figura 6: Alerta que aparece al eliminar un elemento de la base de datos.

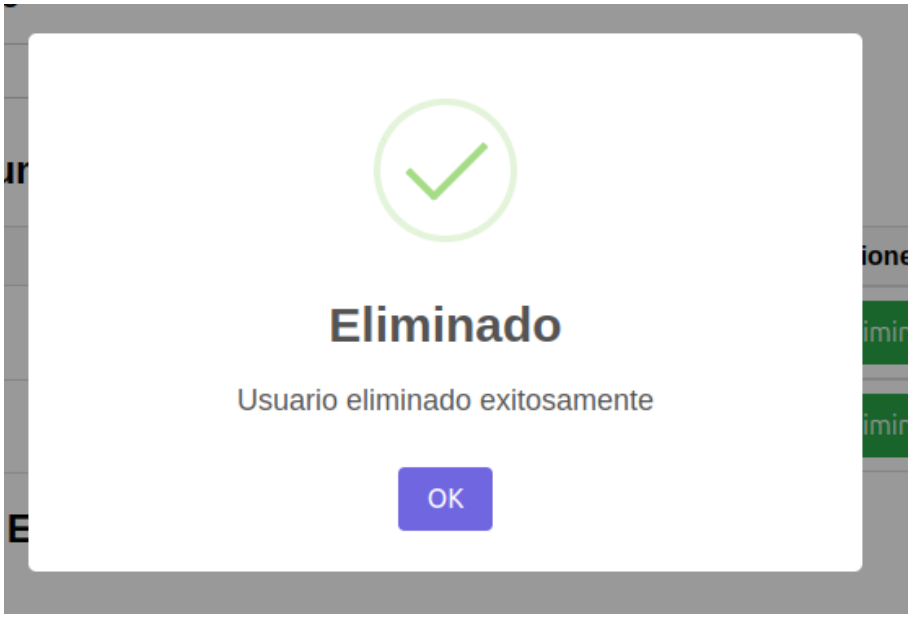

Figura 7: Alerta que aparece al eliminar un elemento de la base de datos.

8. Comparación entre Bases de Datos Relacionales y NoSQL

8.1. Ventajas y Desventajas Observadas

- **SQLite**:
  - Ventajas: Fácil de configurar y manejar para datos estructurados.
  - Desventajas: Menos flexible para datos no normalizados.

- **MongoDB**:
  - Ventajas: Excelente para datos flexibles y no estructurados.
  - Desventajas: Requiere mayor configuración inicial.

9. Dificultades y Aprendizajes

9.1. Dificultades

- Manejo de GridFS para almacenar y servir archivos grandes.
- Validación de datos y manejo de errores en ambas bases.

9.2. Aprendizajes

- Uso práctico de bases de datos relacionales y NoSQL.
- Integración de frontend y backend para operaciones dinámicas.
- Ventajas de usar herramientas modernas como SweetAlert2 para mejorar la experiencia del usuario.

10. Conclusiones

El trabajo permitió entender las fortalezas y limitaciones de SQLite y MongoDB en un entorno real. La integración de ambas bases en una misma aplicación demostró ser útil para manejar diferentes tipos de datos y requerimientos. SQLite, con su simplicidad y estructura definida, resultó ideal para gestionar información de estudiantes, donde se requería integridad referencial y operaciones rápidas. Por otro lado, MongoDB mostró ser una solución robusta para almacenar datos flexibles y grandes volúmenes de información, como los exámenes, gracias a su capacidad para gestionar archivos de manera eficiente a través de GridFS.

A lo largo del desarrollo del proyecto, se pudo observar cómo cada base de datos se complementa con las necesidades específicas del sistema. Esto permitió aprovechar lo mejor de ambos enfoques, maximizando la eficiencia y la flexibilidad de la aplicación. Este trabajo práctico también subrayó la importancia de un diseño bien planificado y la integración de tecnologías adecuadas para construir aplicaciones web robustas y escalables. La experiencia adquirida fue valiosa para consolidar conocimientos sobre la implementación de bases de datos y la interacción entre el frontend y el backend en un entorno de desarrollo moderno.