TRABAJO PRÁCTICO BDD

Cátedra Merlino 2do Cuatrimestre 2024

Navarro Cafferata Marcos - Padrón 106974

Facultad de Ingeniería
Universidad de Buenos Aires

Introducción

El presente trabajo tiene como objetivo comprender las diferencias prácticas entre bases de datos relacionales y NoSQL mediante el desarrollo de una aplicación web que interactúa con ambos tipos de sistemas de almacenamiento.

La aplicación desarrollada permite gestionar información de alumnos y sus actividades asociadas. Para ello, se utilizaron dos bases de datos diferentes: una base de datos relacional y una base de datos NoSQL

El sistema incluye una interfaz web que permite realizar operaciones CRUD (crear, leer, actualizar y eliminar) tanto en SQLite como en MongoDB. Esta interfaz brinda al usuario una experiencia clara y funcional, mostrando los datos provenientes de ambas bases de datos en una misma página, pero separando visualmente su origen.

Link al repositorio: https://github.com/NVRar/bdd

Elección de las Tecnologías

Lenguajes y Frameworks:

- **Backend (Python con Flask):** Elegí Python con Flask porque es un framework ligero y flexible, ideal para desarrollar APIs RESTful de manera eficiente. Además, ya tenía experiencia previa con Python, lo que facilitó la implementación del backend. Flask me permitió trabajar con rutas personalizadas y manejar solicitudes HTTP de manera clara.
- **Bases de Datos:**
    - **SQLite:** Usé SQLite porque es una base de datos relacional ligera y de fácil configuración, ideal para almacenar datos estructurados como la información de los alumnos. Utilicé SQLAlchemy como ORM, lo que simplificó la interacción con la base de datos al permitir trabajar con objetos Python.
    - **MongoDB:** Decidí usar MongoDB para manejar datos más flexibles, como las actividades. MongoDB permite almacenar documentos con esquemas no estrictos, lo cual es ideal para manejar datos donde algunos campos, como la fecha, pueden ser opcionales. Utilicé pymongo para interactuar con la base de datos de manera eficiente.

- **Frontend (HTML5, Bootstrap y JavaScript):** Decidí trabajar con HTML5, Bootstrap y JavaScript porque son tecnologías simples y efectivas para crear interfaces web responsive y funcionales. Bootstrap me permitió diseñar una interfaz moderna y responsive sin necesidad de escribir estilos desde cero.

Diagrama de Arquitectura

Descripción:

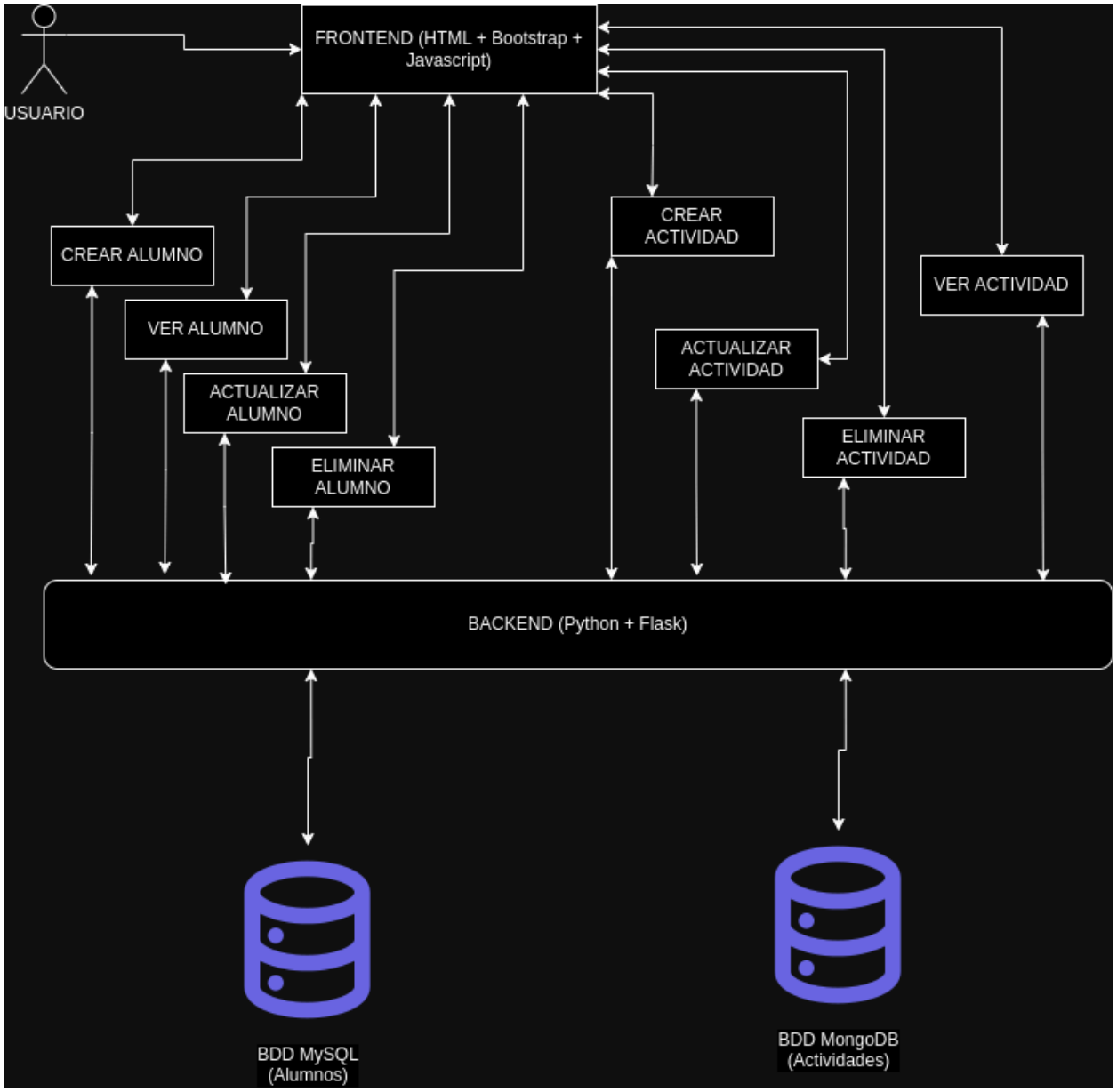
El sistema tiene tres componentes principales:

1. **Frontend:** Construido con HTML5, Bootstrap y JavaScript, maneja la interacción con el usuario y realiza solicitudes HTTP al backend para ejecutar las operaciones CRUD.
2. **Backend:** Implementado con Flask en Python, recibe las solicitudes del frontend, las procesa y realiza las operaciones en las bases de datos. Actúa como intermediario entre el frontend y las bases de datos.
3. **Bases de Datos:**
    - **SQLite:** Almacena información estructurada como datos de los alumnos.
    - **MongoDB:** Se emplea para manejar datos más flexibles, específicamente las actividades de los alumnos.

Flujo de Comunicación:

1. El usuario interactúa con la interfaz del frontend.
2. El frontend realiza solicitudes HTTP al backend para crear, leer, actualizar o eliminar datos.
3. El backend procesa la solicitud y realiza las operaciones necesarias en la base de datos correspondiente (SQLite o MongoDB).
4. Los resultados se devuelven al frontend para ser mostrados.

Aquí se puede observar un diagrama que ilustra un poco mejor esta situación.



Configuración y Conexión a Bases de Datos

SQLite:

- Configuré una base de datos llamada alumnos.db para almacenar información estructurada como padrón, nombre, apellido, edad y email de los alumnos.
- Utilicé SQLAlchemy como ORM para simplificar la interacción con la base de datos y definir el modelo Alumno como una clase en Python.

Ejemplo de conexión:

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///alumnos.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db.init_app(app)
```

MongoDB:

- Configuré una base de datos llamada alumnos_no_sql en MongoDB, que permite almacenar documentos flexibles. Por ejemplo, algunas actividades pueden no tener fecha asignada.
- Usé pymongo para manejar la conexión con MongoDB desde el backend.

Ejemplo de conexión:

```
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017/")
db = client["alumnos_no_sql"]
```

Descripción de Funcionalidades CRUD

SQLite:

1. **Crear:** Implementé un formulario en el frontend para agregar nuevos alumnos. El backend recibe los datos y los inserta en la tabla Alumno utilizando SQLAlchemy.
2. **Leer:** Los datos de los alumnos se obtienen con consultas al modelo Alumno y se muestran en una tabla del frontend.
3. **Actualizar:** Los datos de un alumno pueden modificarse a través de un modal. El backend utiliza SQLAlchemy para ejecutar la operación UPDATE.
4. **Eliminar:** Un botón en la interfaz permite eliminar un alumno. El backend realiza

la operación DELETE en la base de datos relacional.

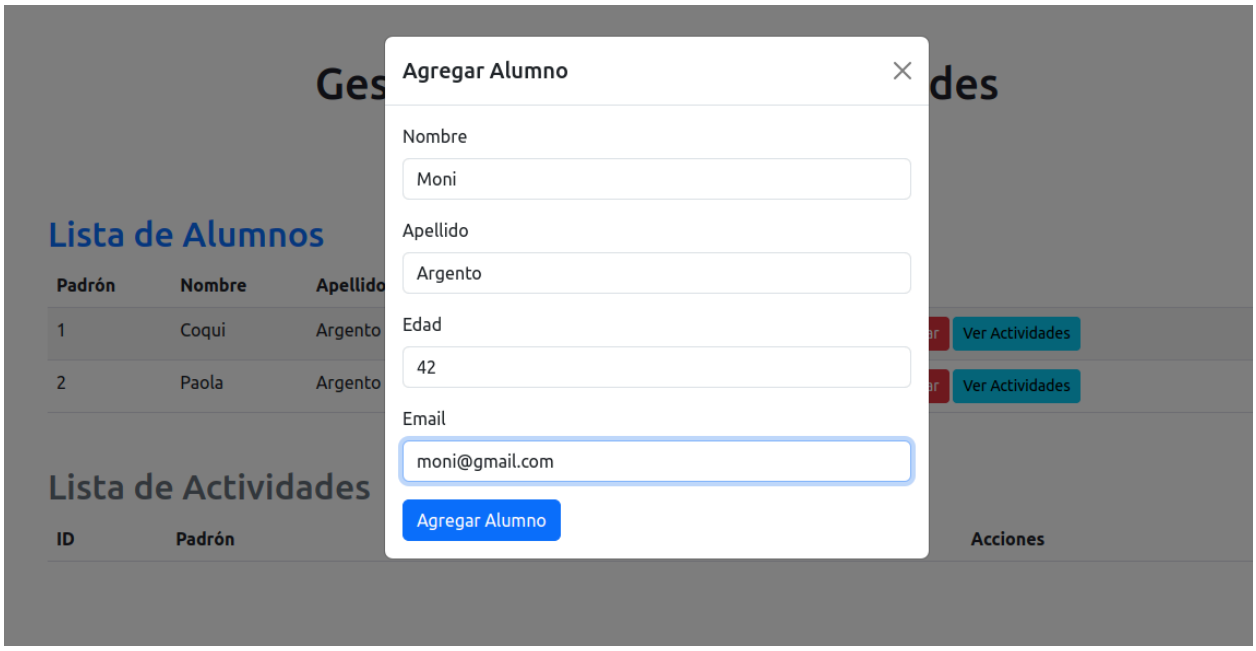
MongoDB:

- **Crear:** Las actividades de los alumnos se agregan con un formulario que permite datos flexibles. Se usa insert_one para almacenar cada actividad como un documento en MongoDB.
- **Leer:** Implementé una tabla en el frontend que muestra las actividades obtenidas con find en MongoDB.
- **Actualizar:** A través de un modal, las actividades pueden editarse. El backend utiliza update_one para modificar el documento correspondiente.
- **Eliminar:** Un botón en la interfaz permite borrar actividades específicas, y el backend utiliza delete_one para eliminar el documento.

Captura de pantalla de las acciones realizadas

Acá se muestra la secuencia de tener una lista de alumnos, hacer click en el botón Agregar Alumno, rellenar el formulario y por último como ya queda agregado a la lista.

Acá es donde se debe rellenar para agregar un alumno:

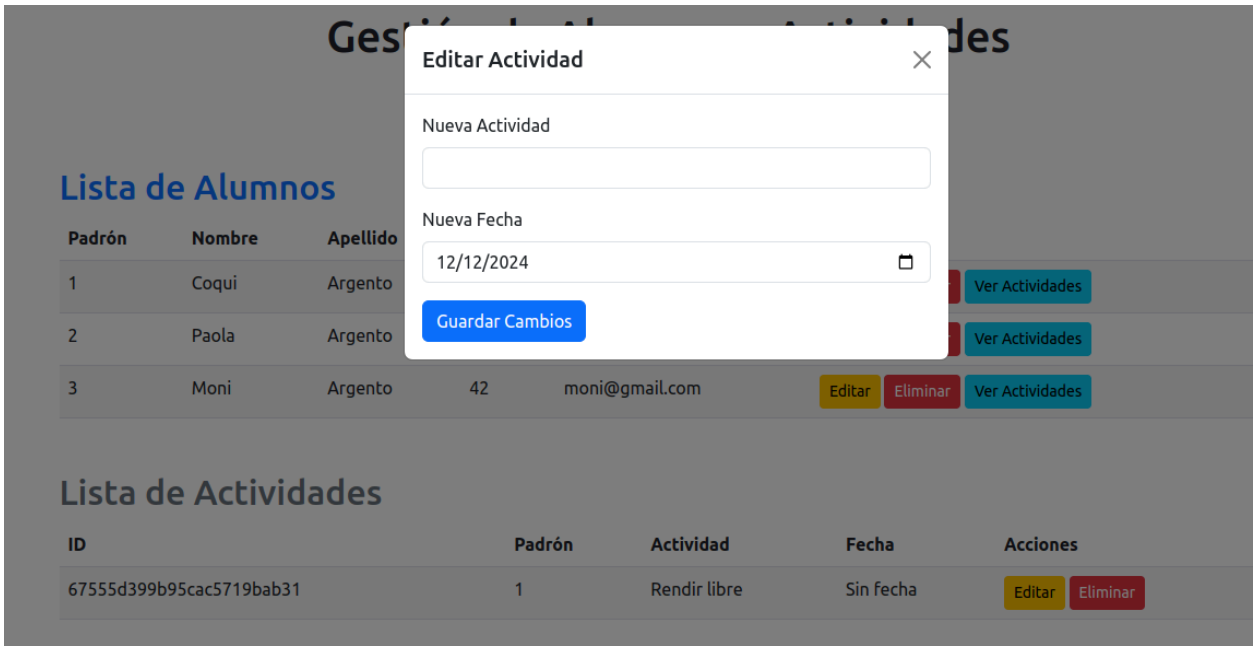


Y este es el resultado luego de agregarlo:

Acá se muestra como Eliminar a un alumno, que esto es haciendo click en el botón Eliminar y luego tenes una reconfirmación de que queres eliminarlo.

Así se ve si se hace click en el botón Ver Actividades de Coqui, como se puede observar se despliegan las Actividades que tiene.

En esta captura se intenta editar la actividad de “Rendir libre”, que si ve que esta sin fecha asignada.



Luego de le elegir la fecha y hacer click en el botón de Guardar Cambios, podemos observar que la fecha de la actividad cambia a 2024-12-12

En este caso se prueba Eliminar una actividad y nuevamente tiene un mensaje para reconfirmar que realmente se desea eliminarla.

Comparación entre Bases de Datos Relacionales y NoSQL

El uso de SQLite y MongoDB en este proyecto fue clave para abordar las diferentes necesidades de almacenamiento de datos que surgieron al implementar las funcionalidades. Ambas bases de datos tienen ventajas específicas que se complementaron de manera efectiva en el desarrollo.

SQLite:

SQLite demostró ser una excelente elección para gestionar los datos estructurados y normalizados, como la información de los alumnos. Los alumnos tienen campos fijos como nombre, apellido,padrón, edad y email, lo que encaja perfectamente en un modelo relacional. Además, el uso de SQLite fue particularmente útil debido a su simplicidad de configuración: no requiere la instalación de un servidor independiente, lo que simplifica el proceso de desarrollo y despliegue.

En este contexto, SQLite permitió garantizar la integridad de los datos mediante restricciones como claves primarias y unicidad en el email. Estas características son

esenciales en una base de datos relacional, ya que aseguran que los datos sean consistentes y fáciles de consultar mediante SQL. Por ejemplo, fue posible realizar operaciones como buscar alumnos por padrón o verificar si un email ya estaba registrado antes de crear un nuevo alumno.

MongoDB:

Por otro lado, MongoDB fue ideal para gestionar las actividades de los alumnos, que tienen una estructura más flexible y dinámica. En este caso, cada actividad tiene campos como el nombre, la fecha, y el padrón del alumno al que está asociada. MongoDB permite manejar situaciones donde ciertos campos, como la fecha, pueden ser opcionales.

La flexibilidad de MongoDB es particularmente útil para adaptarse rápidamente a cambios en los requisitos. Por ejemplo, si se hubiese requerido agregar nuevos campos a las actividades, MongoDB lo habría permitido sin necesidad de alterar esquemas predefinidos, como ocurre en las bases de datos relacionales. Además, su capacidad para manejar documentos JSON-like simplificó la integración con el backend, ya que los datos enviados y recibidos en las solicitudes HTTP encajaban perfectamente con el formato esperado por MongoDB.

Dificultades y Aprendizajes

Dificultades:

Uno de los principales desafíos fue manejar los dos tipos de bases de datos desde el backend. Cada una tiene su propia lógica, métodos y estructuras, lo que requería un entendimiento claro de cómo interactuar con SQLite y MongoDB de manera eficiente. Esto implicó ajustar el código para que las operaciones CRUD fueran consistentes y funcionales para ambas bases de datos, considerando sus particularidades.

Otra dificultad importante fue pensar y manejar los casos bordes. Por ejemplo, asegurar que el sistema manejara adecuadamente situaciones como la eliminación de un alumno que tuviera actividades asociadas o la actualización de un dato inexistente. Este trabajo requirió una planificación cuidadosa para garantizar que la aplicación respondiera correctamente en este tipo de escenarios.

Aprendizajes:

Hacer este proyecto me permitió comprender mejor para qué situaciones es más adecuado usar cada tipo de base de datos. Aprendí que SQLite es excelente para gestionar datos estructurados con relaciones claras, mientras que MongoDB ofrece flexibilidad para datos dinámicos o menos estructurados, como en el caso de las actividades.

Además, profundicé mis conocimientos en SQLite y MongoDB, tanto en su configuración como en el manejo de sus operaciones CRUD. Por ejemplo, aprendí a integrar SQLite utilizando SQLAlchemy en Flask, lo que facilitó la gestión de modelos y consultas, y a aprovechar las características de MongoDB para almacenar y consultar datos no estructurados de manera eficiente.

También desarrollé una mayor habilidad para diseñar e implementar aplicaciones que integran múltiples tecnologías, lo que incluyó el aprendizaje sobre cómo estructurar un backend que actúe como intermediario entre el frontend y dos bases de datos distintas.