FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico - Bases de datos SQL y no SQL

[75.15] Base de datos
Cátedra Merlino
Segundo cuatrimestre de 2024

Alumno:	CAMPILLAY, Edgar Matías
Número de padrón:	106691
Email:	ecampillay@fi.uba.ar

Índice

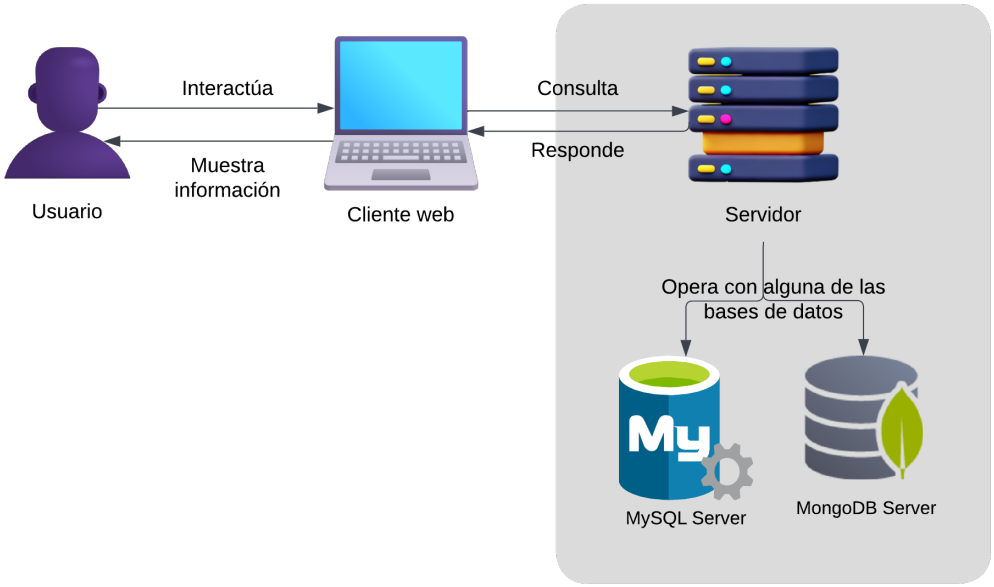

Figura 1: Diagrama de la arq. general

1. Introducción

El presente informe reune la documentación de la solución del trabajo práctico de la materia bases de datos de la facultad de ingeniería de la Universidad de Buenos Aires. El objetivo fue el desarrollo de una aplicación web para gestionar reseñas de restaurantes usando una base de datos SQL y otra no SQL.

2. Elección de las tecnologías

Para el desarrollo de la aplicación web se usó React con SweetAlert2, y TailwindCSS junto con los compenentes open-source de HyperUI para diseñar el *frontend*. React se eligió por el dinámismo que les da a las páginas, y tanto SweetAlert2 como TailwindCSS por la rapidez que ofrecen al momento de dar estilos a la página web. Y para el *backend* se usó express (elegida también por la facilidad de uso) para correr el servidor junto con librerías habladas más adelante.

Para trabajar con la base de datos SQL se eligió MySQL, más precisamente la versión 8.0, principalmente por tratarse de un proyecto que tiene una única entidad muy estructurada y definida. También se usó la librería mysql2 para hacer consultas a la base de datos, elegida por ser la más ampliamente usada siendo compatible con MySQL.

Respecto a la base de datos no SQL, se elegió MongoDB principalmente por la posibilidad de usar datos no estructurados, ademas de dejar posibilidad de escalado a la app en el futuro. Para interactuar con la base de datos se uso mongoose, elegida por ser también la más usada y dar la facilidad de trabajar con los elementos de los registros como si fueran objetos de JavaScript.

3. Estructura de la aplicación

Para llevar a cabo su propósito, partimos desde el cliente, el cual envía a través de solicitudes *fetch*, los datos o consultas al servidor, el cual a través de las rutas y las librerías *performa* las

operaciones *crud* previamente configuradas, como lo son el agregado de restaurantes y/o *reviews*, el cálculo de promedio en *rating*, la eliminación de restaurantes y reseñas (ya sea por reseña específica o según un restaurante).

Una vez que la operación se llevo a cabo el cliente vuelve a enviar solicitudes *get* al servidor para actualizar la información disponible en la página, el servidor responde con los resultados de las *queries* y el cliente trabaja con estos datos para *renderizarlos*.

4. Implementación de bases de datos

4.1. SQL

Se elegió la base de datos SQL para modelar la lista de restaurantes (restaurante, entidad fuerte) donde cada registro lleva los siguientes campos:

- id, atributo numérico único y entero e identificador para cada registro.
- name, texto único.
- comida, texto que representa el tipo de restaurante.
- precio, el rango de precios que maneja el restaurante.
- dirección, texto de la forma "xx nombre.º "nombre xx".
- ciudad, texto.
- provincia, texto.

De esa manera cada atributo y registro cumplen con las formas normales. Los IDs hacen que cada registro sea único, así como la división entre direccion, ciudad y provincia evitan que una misma columna tenga varios datos dentro. A su vez hay varías verificaciones que se hacen dentro del servidor para evitar la violación de las formas normales, ningún campo puede estar vacío, telefono solo acepta un único *string* numérico, el id solo es modificado por la base de datos en sí y el nombre del restaurante nunca se puede repetir, ni tampoco la combinación dirección, ciudad, provincia.

4.2. No SQL

Para el modelado de reseñas, se elegió la base de datos no SQL. Donde cada registro lleva los siguientes atributos:

- título, *string* no mayor a 50 caracteres.
- review, *string* no mayor a 500 caracteres.
- puntaje, entero del 1 al 5.
- restaurante, *string* donde si bien puede tomar cualquier valor la app lo limita a los restaurantes registrados.

Estos también a su vez tienen funciones dedicadas a la verificación de datos. Ningún registro puede tener atributos en blanco durante su creación ni su posterior modificación (dichas verficaciones se encuentran en el archivo mondgoDbValidation.js).

Figura 2: Agregado de registros a la tabla

Figura 3: Edición de registros

Figura 4: Eliminación de registros

5. Detalles de implementación y UI

5.1. Restaurantes

Los restaurantes se muestran en la tabla lateral al primer form, en la tabla misma se pueden modificar los registros o borrarlos en caso de así desearlo.

5.2. Ratings

Para el modelado de los *ratings* se eligió *renderizarlos* dinámicamente a medida que se agregan las reviews para no violar ninguna forma normal haciendo que una base dependa de la otra ni acoplar los componentes más de lo necesario.

5.3. Reseñas

Para agregar reseñas se uso en el *form* los nombres de la columna restaurante de la tabla SQL, y cuando este agrega una reseña se *triggerean* los cálculos de promedios de *ratings* (estos también son *triggereados* cuando se edita una *review* o se la borra).

Adicionalmente (sin referencias propias dentro de la tabla SQL, sino a través del propio del cliente) cuando se elimina un restaurante, el proceso también elimina las *reviews* asociadas a este mismo.

6. Retos

Durante el desarrollo del proyecto hubo varios dilemas, como:

Agregando a lo antes mencionado, los *ratings*, en la primera implementación el *rating* era un atributo más de cada restaurante y se *triggereaba* su cálculo y *post request* al servidor SQL al hacer una inserción y/o modificación en la base de datos MongoDB, esto agregaba un valor redundante. Por esa razón se lo eliminó de la entidad y se lo calcula al momento de *renderizado*.

Figura 5: Vista de las reseñas

Figura 6: Edición de reseñas

Definir las formas de verificación correctas para insertar y/o editar los registros ya sea dentro de la misma base de datos como desde las *api requests*. Cuáles combinaciones de atributos definen un elemento único y cualés pueden repetirse.

Conexión entre las bases de datos, definir cuando usar el cliente como intermediario para realizar cambios en la otra a partir de modicaciones en alguna de ellas. En mi caso el cliente solo consulta a la otra base de datos, cuando se elimina un restaurante.

Elección de la base de datos no SQL, se eligió MongoDB sobre los otras alternativas, por la flexibilidad que ofrece al momento de crear los registros (lo cual deja la posibilidad a futuro de implementar más campos para las *reviews*) y la posibilidad de uso de funciones de agregación, lo cual resultó muy útil para el cálculo de *ratings*.

6.1. Conclusiones

Respecto a las diferencias entre las bases de datos,

- SQL fue perfecto para modelar los restaurantes ya que estos son una entidad muy estructurada y definida.
- No SQL, MongoDB, sirvió muy bien para manejar con fácilidad los ratings. Si bien las reseñas actualmente tienen una estructura definida, me da posibilidad a expandir la implementación, p.ej. como añadir autor/es para una misma reseña, imagen/es o campos custom dinámicamente sin preocuparme por estructuras ni formas normales.