Instituto Tecnológico de Buenos Aires

TRABAJO PRÁCTICO OBLIGATORIO

72.41 - Bases de Datos II - 2023 2Q

Autor:

Tomás Marengo - 61587

tmarengo@itba.edu.ar

Índice

1. Introducción

Este informe documenta el proceso de desarrollo de un sistema de facturación en el contexto del trabajo práctico de Bases de Datos 2. El objetivo principal del proyecto es abordar las consignas proporcionadas, que incluyen la implementación de consultas SQL y NoSQL, vistas específicas, migración de SQL a una base de datos NoSQL y la creación de una API para la gestión de clientes y productos. El proyecto se centra en estas tareas particulares, sin abordar la implementación completa de un sistema de facturación.

2. Estructura del Proyecto. Carpetas y Archivos Importantes

El proyecto se organizó en una estructura de carpetas y archivos que facilita la gestión y el desarrollo de las diferentes partes del sistema. A continuación, se presenta una descripción general de la estructura:

1. **api:** Esta carpeta contiene el código fuente de la API desarrollada utilizando Node.js y Express. Los archivos y carpetas importantes dentro de esta estructura son:

   a) **app.js:** El archivo principal de la aplicación Node.js donde se configuran las rutas y se inicia el servidor.

   b) **routers/mongo y routers/postgres:** Carpetas que contienen los archivos de enrutamiento para las rutas relacionadas con las bases de datos MongoDB y PostgreSQL, respectivamente. Estos archivos definen los endpoints para las diferentes secciones de la página web en cada base de datos.

   c) **models/mongoModels.js:** Un archivo que define los esquemas (schemas) de datos de MongoDB para representar clientes, productos y otros objetos en el sistema. Estos modelos son utilizados para interactuar con la base de datos MongoDB.

   d) **connections/mongoConfig.js y connections/postgresConfig.js:** Archivos de configuración que contienen información sobre la conexión a las bases de datos MongoDB y PostgreSQL. Es importante mencionar que las contraseñas y los usuarios están hardcodeados en estos archivos, los mismos que se utilizan en el docker-compose.yml.

   e) **migrateData.js:** Un script utilizado para realizar la migración de datos desde la base de datos SQL a la base de datos NoSQL MongoDB. Este script se encarga de extraer y transformar los datos de SQL antes de insertarlos en MongoDB. Cuando la migración se ha realizado con éxito, se mostrará un mensaje de registro (console.log) indicando que la migración se completó. Si se vuelve a generar el container, se intenta migrar pero se manejan los errores.

2. **ui:** Esta carpeta contiene la interfaz de usuario (UI) desarrollada en React. Los archivos y carpetas importantes son:

   a) **components/:** Una carpeta que contiene componentes reutilizables de React. Estos componentes pueden ser utilizados en diferentes partes de la aplicación para mantener un código limpio y modular.

   b) **pages/:** Una carpeta que contiene componentes de React que representan páginas específicas de la aplicación. Cada página puede estar compuesta por varios componentes y se utiliza para mostrar contenido específico al usuario.

c) **utils/:** Una carpeta que contiene archivos de utilidad o funciones que pueden ser utilizadas en diferentes partes de la aplicación. Estos archivos pueden incluir funciones de ayuda, utilidades de formato de datos u otras funcionalidades compartidas.

d) **app.jsx:** El archivo principal de la aplicación React. En este archivo, se configuran las rutas y se define la estructura principal de la aplicación React. Incluye la navegación entre páginas, la administración del estado de la aplicación y la renderización de componentes principales.

3. Migración a NoSQL

Migración a NoSQL en MongoDB:

1. Creación del Esquema en MongoDB: Se definieron esquemas de documentos en MongoDB para reflejar las tablas SQL del sistema. Algunas entidades se embebieron, como los teléfonos dentro de los clientes (se entiende que no puede haber teléfonos sueltos ya que lo que queremos es la información de los clientes).

2. Conexión a MongoDB: Se configuró la conexión utilizando Mongoose.

3. Migración de Datos: Se implementó un script para transferir datos desde PostgreSQL a MongoDB, utilizando la API de PostgreSQL.

4. Actualización de la API de MongoDB: Los endpoints de la API se adaptaron para utilizar los esquemas de MongoDB.

Creación de Vistas en MongoDB luego de la migración:

1. Vista de Facturas Ordenadas por Fecha: Se creó una vista para acceder a las facturas ordenadas cronológicamente.

2. Vista de Productos no Facturados: Se implementó una vista que lista los productos disponibles en el inventario y no vendidos.

4. Implementación y uso de la API

A continuación, se detallan los endpoints disponibles y cómo utilizarlos:

4.1. Endpoints de la API

Los endpoints de la API se pueden acceder a través de la URL base http://localhost:3000/postgres/... ó http://localhost:3000/mongo/..., según la base de datos seleccionada. A continuación, se describen los endpoints específicos:

4.2. Consultas

Ejecutar consulta: GET /queries/{número}

Se reemplaza número por el número de la consulta que se desea ejecutar (del 1 al 10 según la consigna). Los resultados de la consulta se devolverán como respuesta.

Para lograr que las consultas den igual en Mongo, se utilizaron distintas funciones de agregación que se pueden ver */api/routers/mongo/queries.js*.

4.3. Vistas

Facturas ordenadas por fecha: GET /views/ordered-invoices

Al acceder a este endpoint, obtendrás un conjunto de datos que muestra las facturas ordenadas por fecha.

Productos no facturados: GET /views/uninvoiced-products

Este endpoint proporciona una lista de productos que aún no han sido facturados.

4.4. CRUD para Clientes

Crear cliente: POST /clients

Utilizá este endpoint para crear un nuevo cliente en la base de datos. Se deben proveer los datos de los clientes en formato JSON en el body {nro_cliente ó _id, activo, apellido, direccion, nombre}. *(_id para Mongo).*

Obtener clientes: GET /clients

Accedé a este endpoint para obtener la lista de clientes.

Actualizar cliente: PUT /clients/{id}

Podés utilizar este endpoint para actualizar la información de un cliente existente. Proporciona los datos actualizados en el cuerpo de la solicitud en formato JSON y especifica el ID del cliente a actualizar.

Eliminar cliente: DELETE /clients/{id}

Este endpoint te permite eliminar un cliente de la base de datos según su ID. Reemplaza id por el ID del cliente que deseas eliminar.

4.5. CRUD para Productos

Análogo al CRUD de clientes, pero con body {codigo_producto ó _id, descripcion, nombre, marca, precio, stock}. *(_id para Mongo)*.

5. Uso de la Página

Cómo sacar el máximo provecho de esta plataforma:

1. **Home:** La página de inicio te dará la bienvenida con información general del sistema.
2. **Clients:** En esta sección, podés ver, crear, modificar o eliminar clientes (Extra: también se puede en Mongo! y probablemente te deje hacer más cosas al no tener las restricciones de SQL).
3. **Products:** Podrás administrar tu inventario de productos. Agregá nuevos productos o actualizá los existentes (Extra: también se puede en Mongo!).
4. **Views:** Explorá vistas personalizadas de tus datos. Acceso rápido a las facturas ordenadas por fecha o los productos que aún no han sido facturados.
5. **Queries:** Realizá consultas para obtener información detallada sobre clientes, facturas, etc.

Notarás un botón en la barra de navegación que te permite alternar entre Postgres y MongoDB. Después de navegar a una sección, el sistema vuelve automáticamente a la base de datos Postgres. ¡No te preocupes! Podés hacer clic en el botón nuevamente para volver a MongoDB.

Lamentablemente, no llegué a implementar una función que mantenga tu elección de base de datos entre páginas (¡lo siento, aún no aprendí UseContext de React!). Así que, para cambiar, simplemente hacé clic en el botón y estarás listo para explorar tus datos en la base de datos que elijas.

6. Dockerización del Proyecto

El proyecto se dockerizó utilizando Docker Compose para gestionar los contenedores de las bases de datos SQL (PostgreSQL), NoSQL (MongoDB), la API Node.js y la interfaz de usuario React. La dockerización se realizó para simplificar la gestión de dependencias, el despliegue y la configuración del entorno de desarrollo y producción.

En el archivo docker-compose.yml, se definieron los servicios de contenedor necesarios para cada componente del proyecto. Se configuraron los volúmenes, las redes y las variables de entorno para conectar y orquestar estos servicios de contenedor de manera eficiente.

Así que, con *docker-compose build* y luego *docker-compose up*, el sistema ya estará listo y se podrá visitar en *localhost:8080/*.

Para más información, leer el README del **repositorio**.