



Instituto Tecnológico de Buenos Aires

72.41 - Base de datos II

Segundo Cuatrimestre 2024 - Grupo 3

Trabajo Práctico Obligatorio

Autores:

Nicolás Andres Rossi - 53225

Santos Galarraga - 62185

Thomás Busso - 62519

Desiree Melisa Limachi - 59463


1. Introducción

El presente trabajo práctico tiene como objetivo desarrollar un sistema de facturación utilizando diferentes tecnologías de bases de datos, específicamente MongoDB y Redis. Esta combinación es popular en la industria por su capacidad de ofrecer una capa de persistencia que es tanto altamente eficiente como flexible en sus esquemas. MongoDB destaca en este aspecto, permitiendo la evolución y adaptación a nuevos requerimientos conforme cambian las necesidades del negocio. Además, Redis se utiliza como capa de caché, ideal para almacenar y recuperar rápidamente información de acceso frecuente, como el stock de productos, sin necesidad de realizar consultas complejas.

En resumen, la combinación de MongoDB y Redis permite aprovechar las fortalezas de cada tecnología:

- MongoDB se encargará de la persistencia de datos estructurados y las consultas complejas que requieren flexibilidad en la estructura.
- Redis complementa con velocidad para datos temporales, conteos rápidos y caché de consultas frecuentes.

Este enfoque permite una arquitectura de persistencia eficiente y escalable que puede responder a los requerimientos funcionales del sistema de facturación de manera óptima.

 **Repositorio del proyecto:** [github/nicrossi/tp-bases-2](https://github.com/nicrossi/tp-bases-2)

La implementación del servidor y API se hizo utilizando NodeJS y Typescript.

Se recomienda leer el README.md para más detalles acerca de cómo probar los requerimientos.

2. Modelo de Datos

El sistema de facturación se organiza en torno a cinco entidades principales: telefono, cliente, factura, detalle_factura y producto. Estas entidades muestran la estructura necesaria para gestionar el sistema con sus detalles correspondientes.

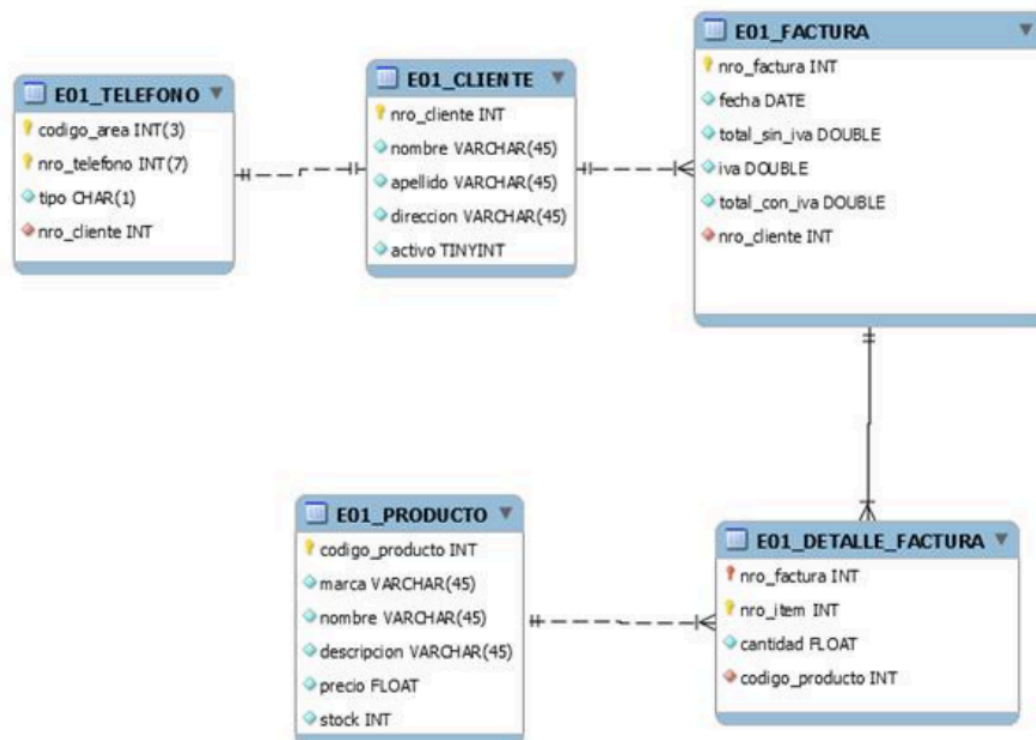


Figura 2.1 DER del sistema de facturación

A continuación, veremos cómo fueron estructuradas las entidades para acomodarse al diseño políglota mencionado anteriormente.

3. Estructura de Datos en las Bases NoSQL

Tal y como mencionamos en la introducción, se utilizó MongoDB para gestionar la información estructurada del sistema debido a que tiene la capacidad de trabajar con documentos y subdocumentos de una manera eficiente.

Los datos de clientes y teléfonos se almacenan juntos en la colección **Cientes** mediante documentos embebidos para los teléfonos, lo que permite obtener toda la información en una sola consulta.

Colección Cientes:

- **nro_cliente**: Es un campo de tipo Number que debe ser único para cada documento. Esto significa que no puede haber dos clientes con el mismo número de cliente.
- **nombre**: Es un campo de tipo String y es obligatorio (required: true). Esto significa que cada cliente debe tener un nombre. Refiere al nombre del cliente.
- **apellido**: Similar al campo nombre, es de tipo String y también es obligatorio. Refiere al apellido del cliente.
- **direccion**: Este campo es de tipo String y es obligatorio, indicando que cada cliente debe tener una dirección registrada. Refiere a la dirección del cliente.
- **activo**: Es un campo de tipo Boolean y es obligatorio. Este campo se utiliza para indicar si el cliente está activo o no.
- **telefonos**: Es un arreglo de objetos. Cada objeto dentro del arreglo representa un teléfono asociado con el cliente y tiene los siguientes campos:
 - **codigo_area**: De tipo String y obligatorio, representa el código de área del número de teléfono.
 - **nro_telefono**: También de tipo String y obligatorio, representa el número de teléfono.
 - **tipo**: De tipo String y obligatorio, indica el tipo de teléfono (por ejemplo, móvil, casa, trabajo, etc.).

En nuestra implementación, hacemos uso de la librería **mongoose-sequence** que nos ayuda a modelar datos auto-incrementales. Algo que utilizamos para generar el **nro_cliente** al momento de una inserción.

Colección Facturas:

- **nro_factura**: Es un campo de tipo String que debe ser único para cada documento. Esto significa que no puede haber dos facturas con el mismo número.
- **codigo_producto**: Es un campo de tipo String y obligatorio. Este campo representa el identificador único de cada producto.
- **marca**: Es un campo de tipo String y obligatorio. Indica la marca del producto.
- **nombre**: Es un campo de tipo String y obligatorio. Representa el nombre del producto.
- **descripcion**: Es un campo de tipo String y obligatorio. Describe las características del producto.
- **precio**: Es un campo de tipo Number y obligatorio. Representa el precio base del producto, sin incluir el IVA.

MongoDB permite consultas complejas sobre los productos y soporta cambios en los atributos de los productos en el futuro. También permite la recuperación de la factura completa en una única consulta.

Colección Productos:

- **codigo_producto**: Es un campo de tipo String que debe ser único para cada documento. Esto significa que no puede haber dos productos con el mismo código.
- **marca**: Es un campo de tipo String y obligatorio. Representa la marca del producto.
- **nombre**: Es un campo de tipo String y obligatorio. Indica el nombre del producto.
- **descripción**: Es un campo de tipo String y obligatorio. Proporciona una descripción detallada del producto.
- **precio**: Es un campo de tipo Number y obligatorio. Representa el precio base del producto, excluyendo el IVA.

A su vez se utilizó Redis para almacenar datos de acceso rápido, su manejo de datos en memoria permite que estos sean más rápidos entonces lo utilizamos para datos temporales y de acceso rápido. A continuación se encuentran los datos y contadores en los que utilizamos esta tecnología.

Detalles de implementación: [redisService.ts](#)

Stock de productos

Clave: `stock:<codigo_producto>`

Valor: Número entero que representa el stock disponible del producto.

Redis permite actualizaciones y lecturas rápidas del stock. Cada vez que se realiza una venta, se puede decrementar el stock en tiempo real sin requerir una consulta compleja. Redis es ideal si se espera un alto volumen de transacciones que afecten el stock de productos.

Totales de facturación por cliente

Clave: `total_gastado:<nro_cliente>`

Valor: Suma acumulada del gasto total del cliente con IVA incluido.

Redis permite manejar contadores que se actualizan rápidamente, por lo que cada vez que un cliente realiza una compra, se puede actualizar el total de gasto sin necesidad de realizar una consulta agregada en MongoDB. Esto puede ser útil para reportes rápidos de consumo por cliente.

Productos vendidos al menos una vez

Clave: `sold_at_least_once`

Valor: Es un set donde almacenamos los `codigo_producto`, y cada vez que se realiza una factura, se agrega el identificador correspondiente a este conjunto.

Haciendo uso de los conjuntos (no admiten repetidos) de redis, modelamos fácilmente la lista de productos que fueron vendidos al menos una vez. Por consiguiente, también podemos encontrar fácilmente los productos que nunca fueron vendidos.

4. Implementación de Operaciones CRUD

En esta sección se describen las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) que fueron implementadas en el sistema para las entidades principales: Clientes, Facturas y Productos.

Clientes:

Crear Cliente

- **Endpoint:** `/clientes`
- **Método:** POST
- **Descripción:** Permite crear un nuevo cliente en el sistema.
- **Detalles:** Incluye validaciones para campos requeridos y manejo de errores.

Listar Clientes

- **Endpoint:** `/clientes`
- **Método:** GET
- **Descripción:** Devuelve una lista con todos los clientes y sus datos.

Consultar Cliente por ID

- **Endpoint:** `/clientes/:nro_cliente`
- **Método:** GET
- **Descripción:** Devuelve los datos de un cliente específico identificado por su número.

Actualizar Cliente

- **Endpoint:** `/clientes/:nro_cliente`
- **Método:** PUT
- **Descripción:** Actualiza los datos de un cliente específico.

Eliminar Cliente

- **Endpoint:** `/clientes/:nro_cliente`
- **Método:** DELETE
- **Descripción:** Elimina un cliente del sistema.

Facturas:

Crear Factura

- **Endpoint:** `/facturas`
- **Método:** POST
- **Descripción:** Permite registrar una nueva factura asociada a un cliente.

Listar Facturas

- **Endpoint:** `/facturas`
- **Método:** GET
- **Descripción:** Devuelve todas las facturas registradas.

Actualizar Factura

- **Endpoint:** `/facturas/:nro_factura`
- **Método:** PUT
- **Descripción:** Actualiza los datos de una factura específica.

Eliminar Factura

- **Endpoint:** `/facturas/:nro_factura`
- **Método:** DELETE
- **Descripción:** Elimina una factura del sistema.

Productos:

Crear Producto

- **Endpoint:** `/productos`
- **Método:** POST
- **Descripción:** Permite registrar un nuevo producto en el sistema.

Listar Productos

- **Endpoint:** `/productos`
- **Método:** GET
- **Descripción:** Devuelve una lista de todos los productos registrados.

Consultar Producto por Código

- **Endpoint:** `/productos/:codigo_producto`
- **Método:** GET
- **Descripción:** Devuelve los datos de un producto identificado por su código.

Actualizar Producto

- **Endpoint:** `/productos/:codigo_producto`
- **Método:** PUT
- **Descripción:** Actualiza los datos de un producto específico.

Eliminar Producto

- **Endpoint:** `/productos/:codigo_producto`
- **Método:** DELETE
- **Descripción:** Elimina un producto del sistema.

Dentro de cada operación incluimos validaciones básicas para garantizar la integridad de los datos como por ejemplo la existencia de un cliente o producto antes de realizar una operación. Este proyecto también cuenta con manejo de errores utilizando try-catch. Gracias a estas medidas, el sistema de facturación garantiza confiabilidad y consistencia en la gestión de clientes, facturas y productos, reduciendo posibles conflictos o inconsistencias en la base de datos.

5. Requerimientos del sistema

Para facilitar las pruebas, a pesar de que no es lo correcto en el diseño de un API, incluimos dentro del servicio, algunos endpoints diseñados para probar los requerimientos del sistema que fueron pedidos. Y se detallan a continuación:

Más detalles: requerimientos_del_sistema.http

1. Obtener los datos de los clientes junto con sus teléfonos.

```
GET http://localhost:3000/clientes/query/1
```

2. Obtener el/los teléfono/s y el número de cliente del cliente con nombre “Jacob” y apellido “Cooper”.

```
GET http://localhost:3000/clientes/query/2
```

3. Mostrar cada teléfono junto con los datos del cliente.

```
GET http://localhost:3000/clientes/query/3
```

4. Obtener todos los clientes que tengan registrada al menos una factura.

```
GET http://localhost:3000/clientes/query/4
```

5. Identificar todos los clientes que no tengan registrada ninguna factura.

```
GET http://localhost:3000/clientes/query/5
```

6. Devolver todos los clientes, con la cantidad de facturas que tienen registradas (si no tienen considerar cantidad en 0)

```
GET http://localhost:3000/clientes/query/6
```

7. Listar los datos de todas las facturas que hayan sido compradas por el cliente de nombre "Kai" y apellido "Bullock".

```
GET http://localhost:3000/facturas/query/7
```

8. Seleccionar los productos que han sido facturados al menos 1 vez.

```
GET http://localhost:3000/productos/query/8
```

9. Listar los datos de todas las facturas que contengan productos de las marcas "Ipsum".

```
GET http://localhost:3000/facturas/query/9
```

10. Mostrar nombre y apellido de cada cliente junto con lo que gastó en total, con IVA incluido.

```
GET http://localhost:3000/clientes/query/10
```

11. Se necesita una vista que devuelva los datos de las facturas ordenadas por fecha.

order = [asc / desc]

```
GET http://localhost:3000/facturas?order=desc
```

12. Se necesita una vista que devuelva todos los productos que aún no han sido facturados.

```
GET http://localhost:3000/productos/query/12
```

- 13. Clientes:** Implementar la funcionalidad que permita crear nuevos clientes, eliminar y modificar los ya existentes.

```
GET/POST/PUT/DELETE http://localhost:3000/clientes
```

- 14. Productos:** Implementar la funcionalidad que permita crear nuevos productos y modificar los ya existentes. Tener en cuenta que el precio de un producto es sin IVA.

```
GET/POST/PUT/DELETE http://localhost:3000/productos
```

6. Conclusión

Utilizando las herramientas adquiridas durante la cursada se logró la implementación del sistema de facturación con MongoDB y Redis combinando lo mejor de ambas tecnologías. MongoDB asegura una gestión sólida y flexible de datos estructurados y relacionados, mientras que Redis optimiza las operaciones que requieren alta velocidad y baja latencia. Este enfoque garantiza un sistema eficiente, escalable y adaptado a las necesidades..

El sistema desarrollado permite una interacción eficiente con las entidades principales (clientes, productos y facturas), integrando funcionalidades CRUD y consultas específicas que aportan un alto valor informativo para la toma de decisiones empresariales. Además, el manejo adecuado de errores y las validaciones implementadas aseguran confiabilidad y robustez en la operación del sistema.

En resumen, este trabajo práctico demuestra cómo una arquitectura híbrida, basada en tecnologías de bases de datos complementarias, puede responder a los desafíos que puede presentar el procesamiento de datos con bases para futuras mejoras y escalabilidad.