**Instituto Tecnológico de Costa Rica**

**Escuela de Ingeniería en Computación**

**IC-4302 – Bases de Datos 2**

# Proyecto #1

Documentación de la arquitectura del sistema para API de restaurante

**Estudiantes:**

Daniel Alemán Ruiz – 2023051957

Luis Meza Chavarría – 2023800023

**Profesor:**

Kenneth Obando Rodríguez

**17 de mayo del 2025**

**I Semestre 2025**

**Documentación de la Arquitectura del Sistema**

## 1. Balanceador de Carga (NGINX)

El balanceador de carga NGINX actúa como punto de entrada único para todas las solicitudes al sistema. Está configurado mediante proxy para que los usuarios puedan acceder a los distintos servicios a través de rutas simplificadas:

- /api - Redirige al servicio de API
- /auth - Redirige al servicio de Autenticación
- /search - Redirige al servicio de Búsqueda

Esta configuración ahorra a los usuarios la necesidad de conocer los puertos específicos de cada servicio. Además, el balanceador distribuye la carga entre las réplicas de cada servicio, mejorando la disponibilidad y el rendimiento del sistema.

## 2. Capa de Servicios

La capa de servicios está compuesta por tres servicios principales, cada uno con dos réplicas para alta disponibilidad y un contenedor adicional para pruebas.

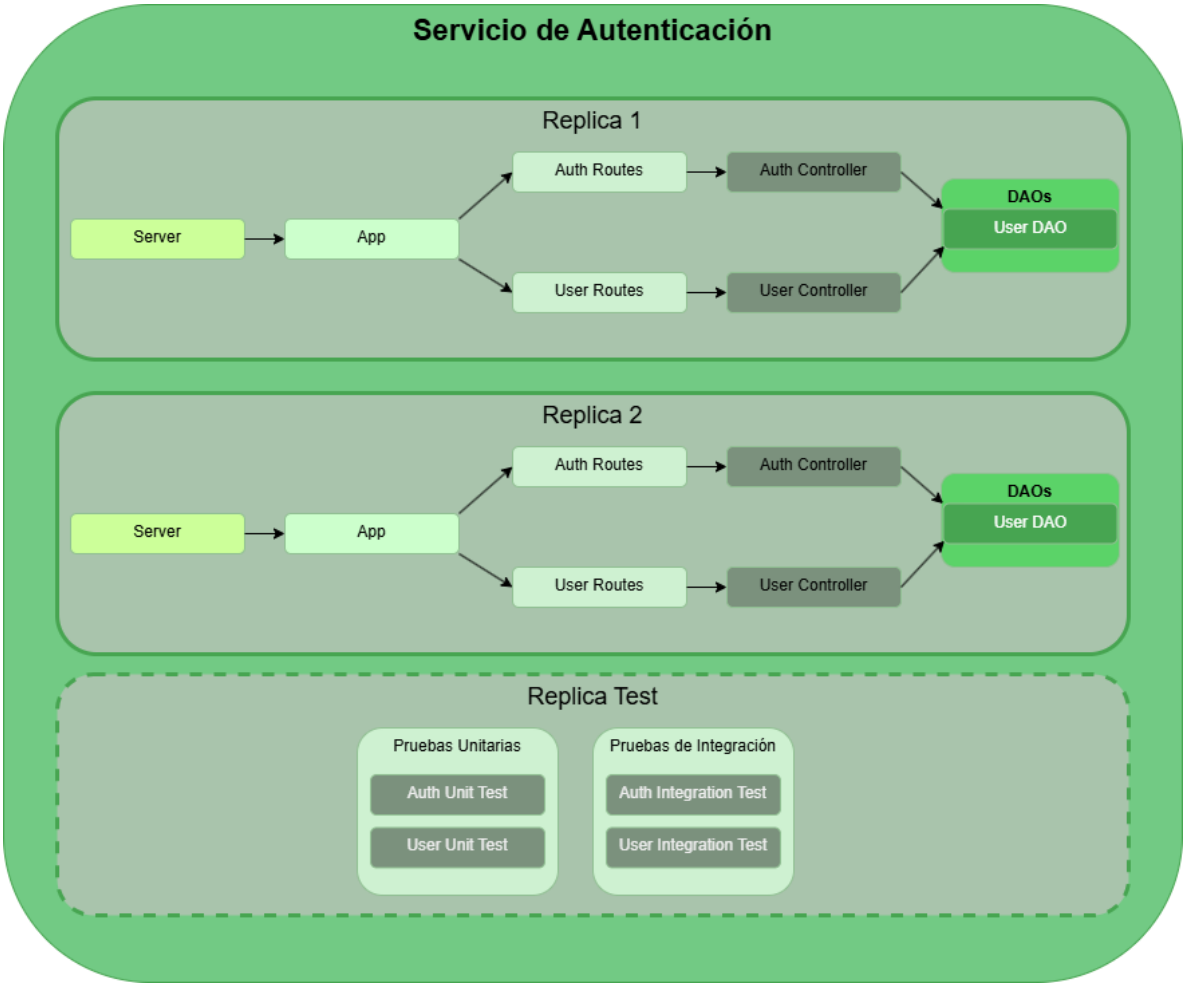
### 2.1 Servicio de Autenticación

El servicio de autenticación gestiona la identidad de los usuarios y controla el acceso a los recursos del sistema. Su arquitectura interna está organizada de la siguiente manera:

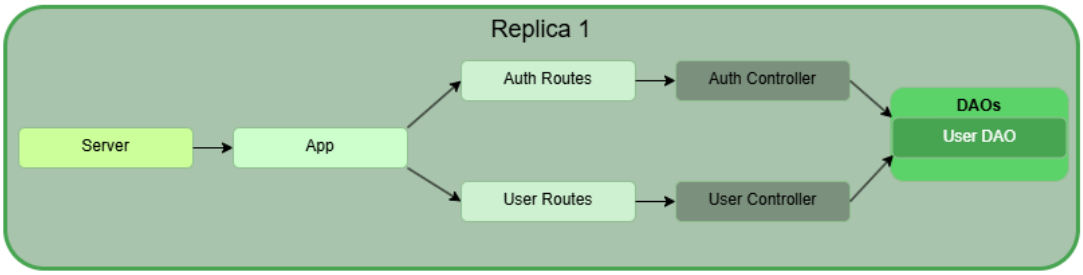
- El componente Server inicia la aplicación y configura el puerto de escucha
- App contiene la configuración de Express y middleware general
- Las rutas (Auth Routes y User Routes) definen los endpoints disponibles
- Los controladores (Auth Controller y User Controller) contienen la lógica de negocio
- Los DAOs (User DAO y Auth DAO) abstraen el acceso a la base de datos

El flujo de una solicitud típica comienza en el servidor, pasa a través de la aplicación principal que dirige la solicitud a la ruta correspondiente. El controlador procesa la lógica de negocio

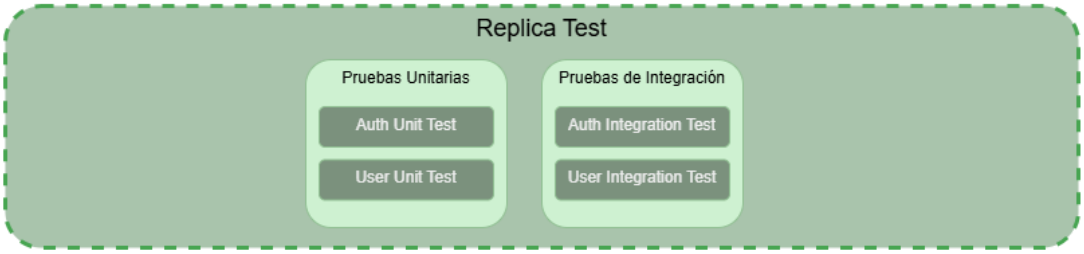
y utiliza el DAO apropiado para interactuar con la base de datos seleccionada según la variable DB_TYPE.




Cada réplica del servicio contiene componentes idénticos, lo que garantiza que cualquiera pueda atender las solicitudes en caso de fallo de otra réplica.



El contenedor de pruebas contiene pruebas unitarias y de integración para verificar el funcionamiento correcto de los componentes Auth y User.

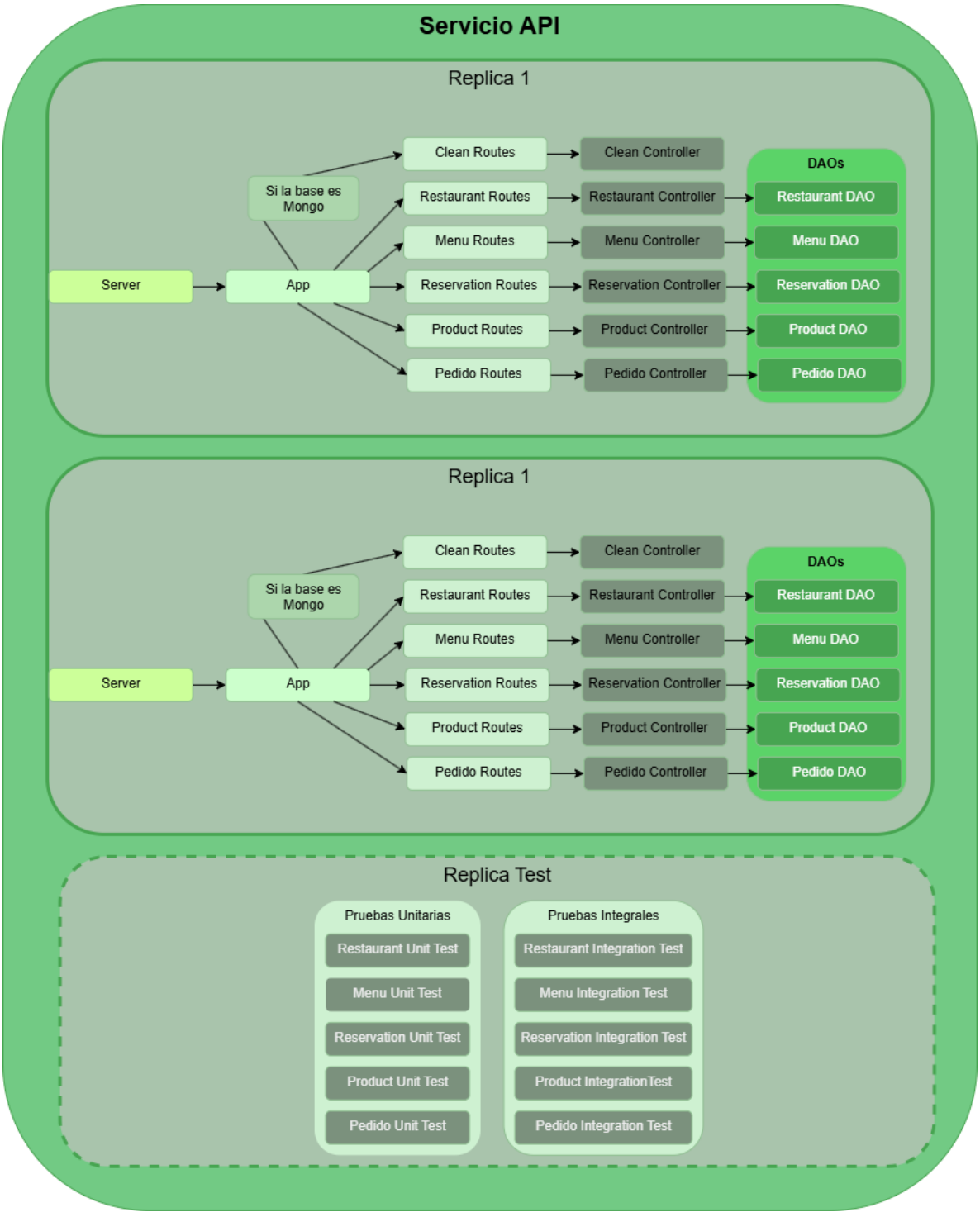



### 2.2 Servicio de API

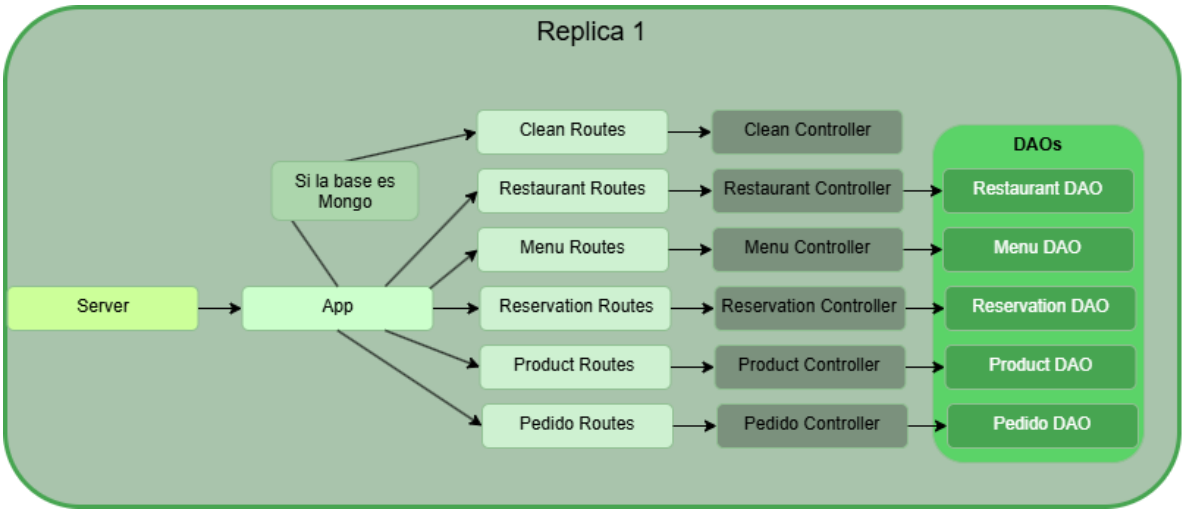
El servicio de API es el núcleo del sistema y maneja todas las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para las entidades principales del negocio. Su estructura interna es la siguiente:

- Server y App funcionan de manera similar al servicio de autenticación
- Incluye rutas y controladores para cada entidad principal: Restaurant, Menu, Product, Reservation y Pedido
- Cada entidad tiene su propio DAO para interactuar con la base de datos
- Adicionalmente, cuando se usa MongoDB (DB_TYPE=mongo), se activan las rutas y controlador de Clean para manejar operaciones de limpieza en cascada

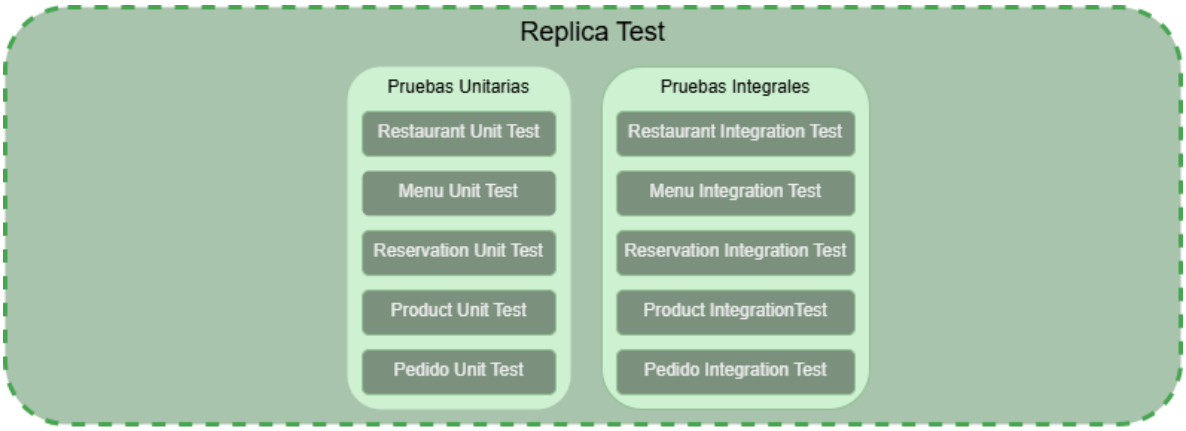
Una característica importante del servicio API es que cuando se utiliza MongoDB como base de datos, se activa una ruta especial de "Clean" que implementa operaciones de limpieza en cascada, necesarias debido a las diferencias en el manejo de relaciones entre PostgreSQL y MongoDB.




Al igual que el servicio de autenticación, la API cuenta con dos réplicas idénticas para garantizar alta disponibilidad.




El contenedor de pruebas incluye pruebas unitarias e integración para cada entidad: Restaurant, Menu, Reservation, Product y Pedido.

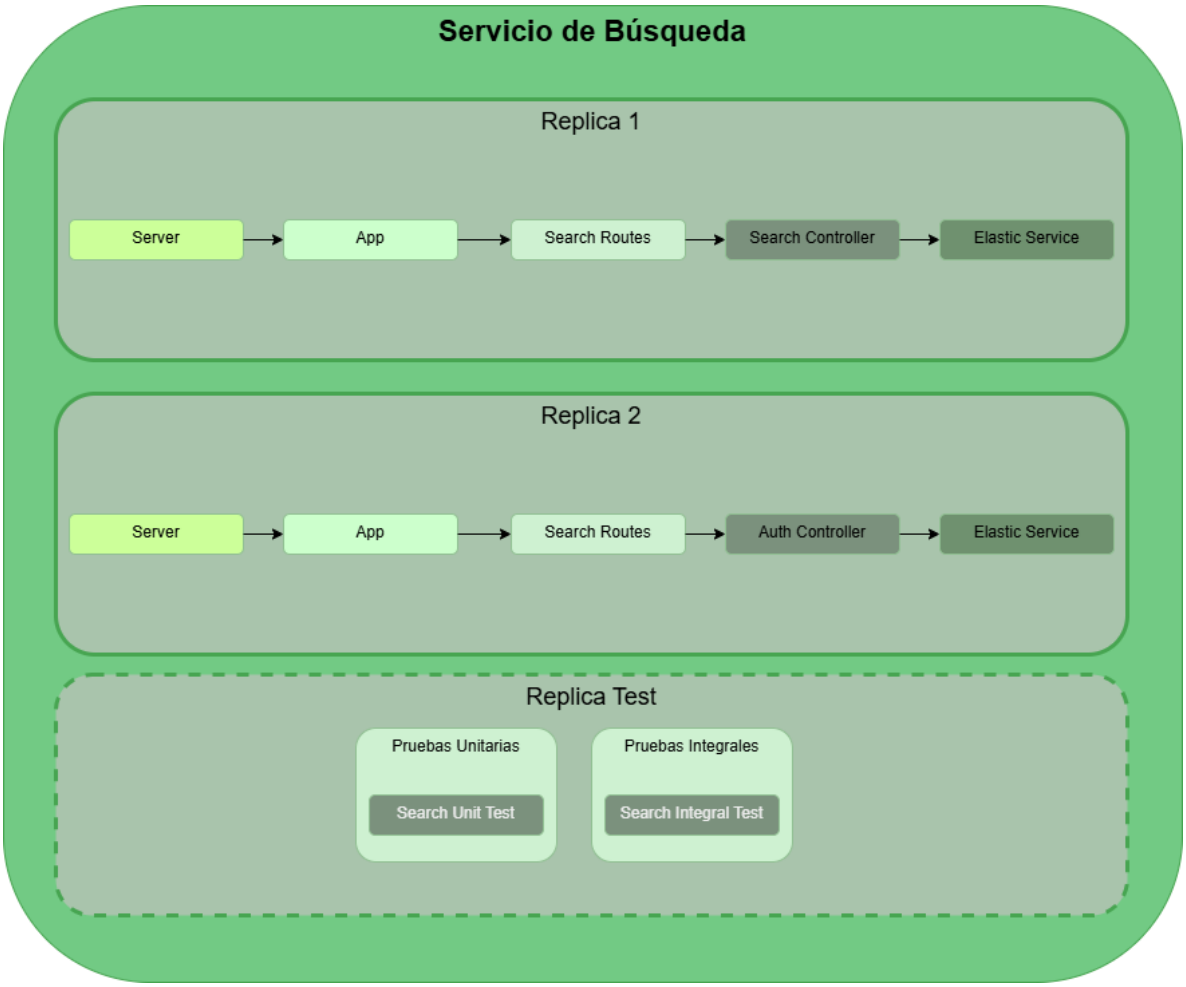



### 2.3 Servicio de Búsqueda

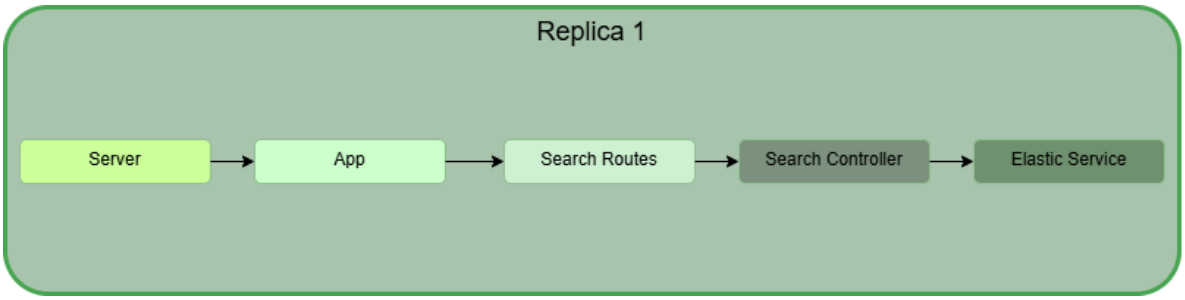
El servicio de búsqueda proporciona funcionalidades avanzadas de indexación y búsqueda de productos utilizando Elasticsearch. Su estructura es más sencilla que los otros servicios:

- Server y App gestionan la inicialización y configuración
- Search Routes define los endpoints de búsqueda disponibles
- Search Controller implementa la lógica de búsqueda
- Elastic Service gestiona la comunicación con Elasticsearch

A diferencia de los otros servicios, el servicio de búsqueda no implementa sus propios DAOs, sino que se comunica con el servicio de API para obtener datos actualizados de productos. Una vez obtenidos los datos, los indexa en Elasticsearch para ofrecer búsquedas rápidas y eficientes.




El servicio de búsqueda también cuenta con dos réplicas idénticas para garantizar disponibilidad.



El contenedor de pruebas incluye pruebas unitarias e integración para los componentes de búsqueda.


## 3. Capa de Caché y Búsqueda

### 3.1 Redis (Caché)

Redis funciona como un sistema de caché en memoria para todo el sistema. Todos los servicios (Autenticación, API y Búsqueda) utilizan Redis para almacenar temporalmente datos frecuentemente accedidos, reduciendo la carga en las bases de datos y mejorando los tiempos de respuesta.

Usos principales de Redis:

- Almacenamiento de sesiones de usuario
- Caché de resultados de consultas frecuentes
- Almacenamiento temporal de datos de productos populares

### 3.2 Elasticsearch

Elasticsearch es utilizado exclusivamente por el servicio de búsqueda para proporcionar capacidades avanzadas de indexación y búsqueda de productos. Permite realizar búsquedas textuales, filtrado por categorías, ordenamiento y otras operaciones complejas de manera eficiente.


### 4. Capa de Base de Datos

El sistema está diseñado para funcionar con dos tipos diferentes de bases de datos, determinados por la variable de entorno DB_TYPE.

#### 4.1 PostgreSQL

Cuando DB_TYPE está configurado como "postgres", el sistema utiliza PostgreSQL como base de datos principal. La estructura de tablas incluye:

- **Usuario**: Almacena información de usuarios y credenciales
- **Restaurante**: Datos de los restaurantes
- **Menu**: Menús asociados a cada restaurante
- **Producto**: Productos individuales dentro de cada menú
- **Reserva**: Reservas de clientes en restaurantes
- **Pedido**: Pedidos realizados por los clientes
- **Detalle_Pedido**: Detalles específicos de cada pedido (productos, cantidades, etc.)


### 4.2 MongoDB

Cuando DB_TYPE está configurado como "mongo", el sistema utiliza MongoDB como base de datos principal. Las colecciones en MongoDB corresponden con las tablas en PostgreSQL:

- **usuarios**
- **restaurantes**
- **menus**
- **productos**
- **reservas**
- **pedidos**
- **counters**: Colección especial para gestionar IDs secuenciales similares a PostgreSQL

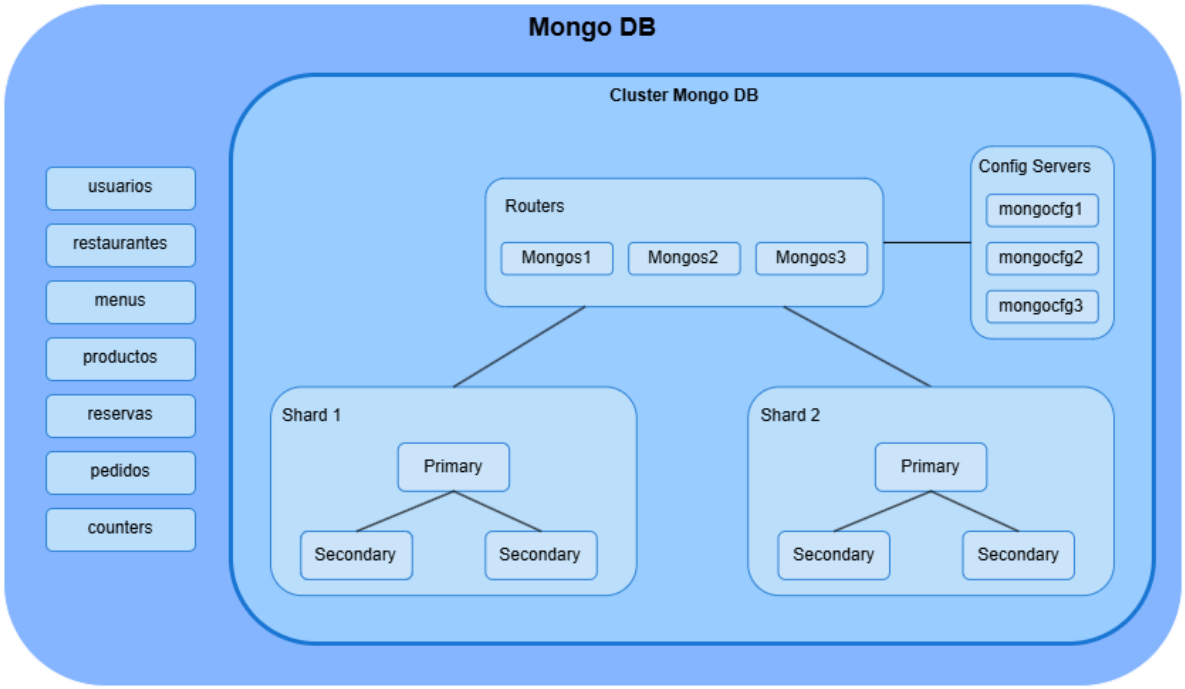
**Arquitectura del Cluster MongoDB**

El cluster de MongoDB está formado por:

- **3 Mongos Routers**: Dirigen las consultas a los shards apropiados

- **3 Config Servers**: Almacenan metadatos y configuración del cluster
- **Shard #1**: Consta de 1 nodo primario y 2 secundarios
- **Shard #2**: Consta de 1 nodo primario y 2 secundarios

Esta arquitectura proporciona alta disponibilidad, escalabilidad horizontal y balanceo de carga.

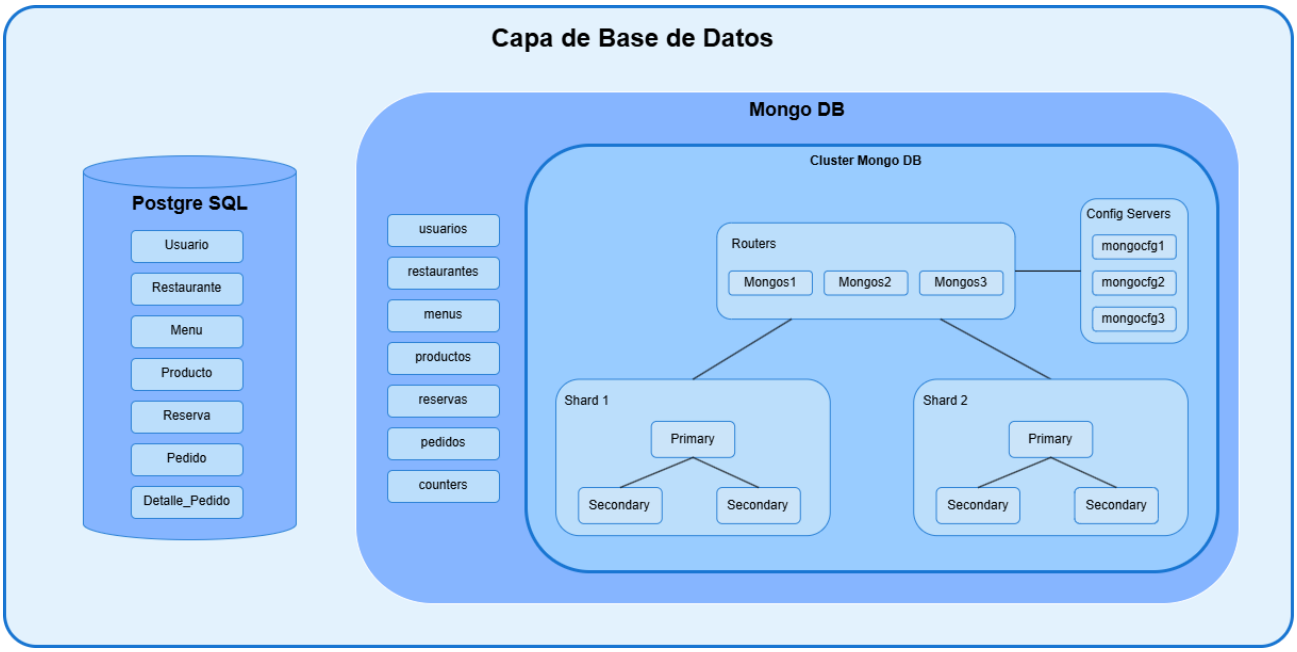



## 5. Patrón DAO y Selección de Base de Datos

Todos los servicios acceden a la base de datos mediante el patrón DAO (Data Access Object), que abstrae la lógica de acceso a datos del resto del sistema. Cada entidad tiene su propio DAO, y la implementación específica que se utiliza (PostgreSQL o MongoDB) se determina en tiempo de ejecución según el valor de la variable de entorno DB_TYPE.

Esta arquitectura permite:

1. Cambiar fácilmente entre bases de datos sin modificar el código de los servicios
2. Aislar la lógica de acceso a datos, facilitando el mantenimiento
3. Implementar estrategias específicas para cada base de datos aprovechando sus características particulares


Capa de Base de Datos
Postgre SQL
Usuario
Restaurante
Menu
Producto
Reserva
Pedido
Detalle_Pedido
Mongo DB
usuarios
restaurantes
menus
productos
reservas
pedidos
counters
Cluster Mongo DB
Routers
Mongos1
Mongos2
Mongos3
Config Servers
mongocfg1
mongocfg2
mongocfg3
Shard 1
Primary
Secondary
Secondary
Shard 2
Primary
Secondary
Secondary