# UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**

## INGENIERÍA DE SOFTWARE



**Base de datos distribuida y heterogénea con Postgresql y MongoDB**

**Ing. Alexis Estevez**

**Integrantes:**

**Dayse Poma**

**Bryan Yanzapanta**

**Henry Tiamba**

# Mayo – Septiembre

# 

[**UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE 1**](#_heading=h.gjdgxs)

[INGENIERÍA DE SOFTWARE 1](#_heading=h.30j0zll)

[**Mayo – Septiembre 1**](#_heading=h.1fob9te)

[**1 Introducción 3**](#_heading=h.3znysh7)

[1.1 Objetivo general 3](#_heading=h.2et92p0)

[**2 Desarrollo 3**](#_heading=h.tyjcwt)

[**2.1 Caso de estudio 3**](#_heading=)

[**2.2 Arquitectura 4**](#_heading=)

[**2.3 Modelo conceptual,físico y lógico de la base de datos 5**](#_heading=h.oo7xeikvdm14)

[**2.4 Tecnologías 5**](#_heading=h.uh995l2o2zk8)

[**2.5 Frontend 5**](#_heading=h.bdcidt6anfgo)

[**2.6 Backend 6**](#_heading=h.a3aggscqqhdu)

[**3 Resultados 22**](#_heading=h.pj6y53wnbc83)

[**4 Conclusión 23**](#_heading=h.87q8sc2s5063)

[**5 Referencia 23**](#_heading=h.oaceycif76v0)

# 

# 

# 

# 

# 

# 

# Introducción

Una base de datos distribuida y heterogénea combina diferentes sistemas de gestión de bases de datos para almacenar y gestionar datos en un entorno distribuido. En este caso, se mencionan PostgreSQL y MongoDB como ejemplos de sistemas de gestión de bases de datos que se utilizan conjuntamente.

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto y ampliamente utilizado. Ofrece una gran cantidad de características avanzadas y soporte para consultas complejas y transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad). PostgreSQL utiliza el lenguaje SQL para realizar consultas y manipular datos.

MongoDB, por otro lado, es una base de datos NoSQL que se basa en el modelo de documentos. En lugar de utilizar tablas y filas como en las bases de datos relacionales, MongoDB almacena los datos en documentos JSON flexibles y dinámicos. Esto permite una mayor flexibilidad y escalabilidad en comparación con las bases de datos relacionales.

La combinación de PostgreSQL y MongoDB en una base de datos distribuida y heterogénea permite aprovechar las fortalezas de ambos sistemas de gestión de bases de datos. PostgreSQL puede ser utilizado para manejar datos estructurados y relaciones complejas, mientras que MongoDB puede ser utilizado para datos no estructurados y casos de uso donde la escalabilidad y la flexibilidad son prioritarios.

## Objetivo general

Implementar una arquitectura de base de datos distribuida y heterogénea utilizando PostgreSQL y MongoDB para lograr un mayor rendimiento, escalabilidad y flexibilidad en el manejo de datos, optimizando la utilización de recursos y mejorando la disponibilidad y confiabilidad del sistema de ventas de videojuegos.

# Desarrollo

# Caso de estudio

Este caso de estudio se enfoca en “MyTems” , una tienda en línea de videojuegos, que busca mejorar la experiencia del cliente y aumentar las ventas. A través de un análisis exhaustivo, se identifican los desafíos clave y se proponen soluciones estratégicas para impulsar el crecimiento y la lealtad del cliente.

Descripción del negocio:

“MyTems” es una tienda en línea especializada en la venta de videojuegos para diversas plataformas, como consolas de juegos, PC y dispositivos móviles. La empresa ofrece una amplia selección de títulos populares, ediciones especiales y accesorios relacionados con los videojuegos. Su objetivo es atraer a una amplia base de clientes, desde jugadores ocasionales hasta entusiastas y profesionales.

Problemas identificados:

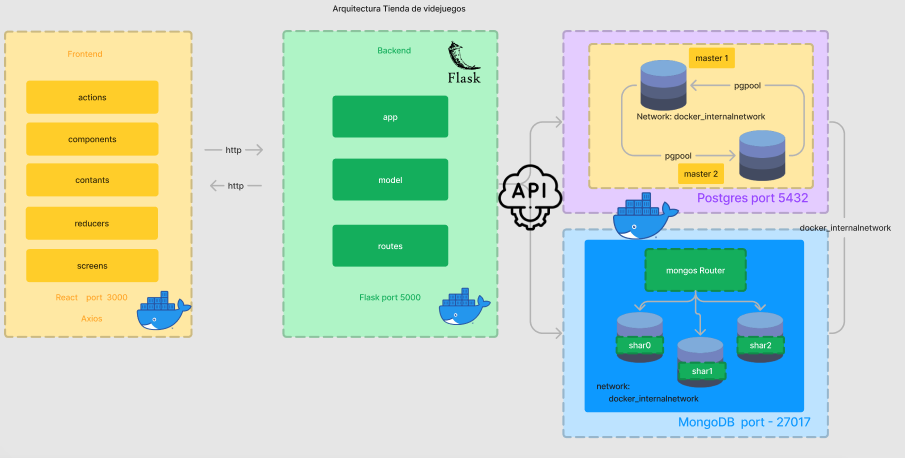
a) Experiencia de usuario deficiente: Los clientes han informado problemas relacionados con la navegación del sitio web, la búsqueda de productos, la falta de información detallada sobre los juegos y dificultades al realizar compras.

b) Competencia feroz: La industria de los videojuegos es altamente competitiva, con múltiples tiendas en línea que ofrecen productos similares. MyTems necesita diferenciarse y ofrecer un valor único a los clientes.

Soluciones propuestas:

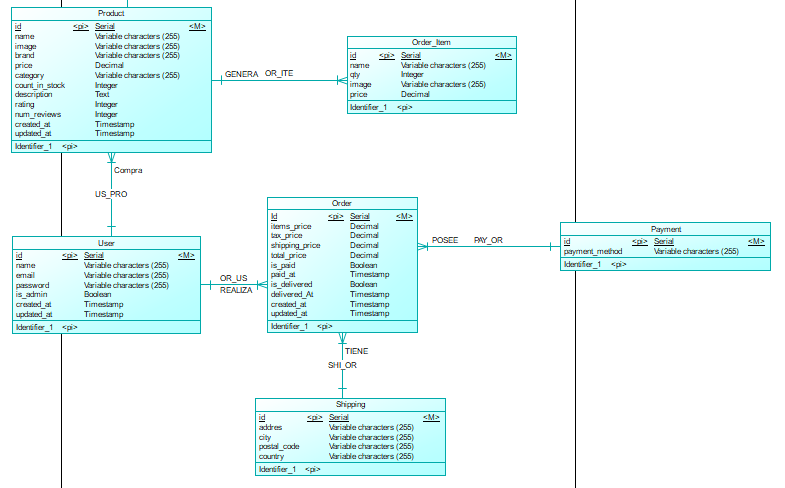
1. Diseño de interfaz de usuario intuitiva: Se creó una interfaz de usuario amigable que permite a los clientes buscar y filtrar videojuegos según sus preferencias, ver detalles de los productos, agregar elementos al carrito de compras y finalizar la compra de manera segura.
2. Sistema de gestión de inventario: Se desarrolló un sistema que rastrea automáticamente el inventario de videojuegos, actualizando las existencias en tiempo real y notificando al equipo de “MyTems” cuando es necesario reponer ciertos títulos.
3. Integración de sistemas de pago: Se integraron múltiples pasarelas de pago para permitir a los clientes pagar con tarjeta de crédito, PayPal y transferencia bancaria. Se implementaron medidas de seguridad adicionales, como el cifrado de datos sensibles.

# Arquitectura

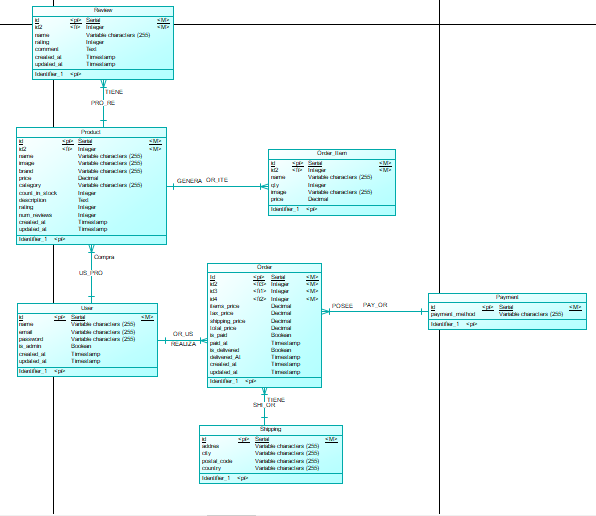


# Modelo conceptual,físico y lógico de la base de datos

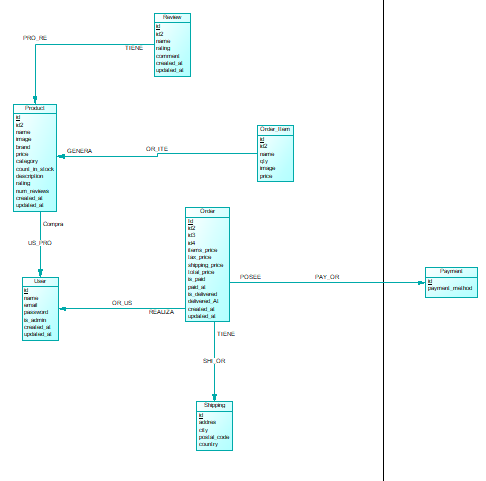
**Conceptual**



**Lógico**



**Fisico**



# Tecnologías

| Frontend | * React * Axios | version 18.2.0 y 1.4.0 |
| --- | --- | --- |
| Backend | * Python * Flask | version 3.11 y 2.0.3 |
| Base de datos | * Postgres * Mongodb | versión 15.2 y 6.0 |
| Administrador de contenedores | * Docker | versión 24.0 |

# Frontend

Como se pudo observar en la arquitectura el frontend se desarrolló con React y Axios la cual se ejecuta en el puerto 3000.

A Continuación se presenta los componentes necesarios para levantar el cliente :

* **docker-compose.yml**

version: "3.8"

services:

app:

build:

context: .

dockerfile: Dockerfile

volumes:

- .:/app

networks:

- tienda\_default

ports:

- "3000:3000"

stdin\_open: true

tty: true

networks:

tienda\_default:

external: true

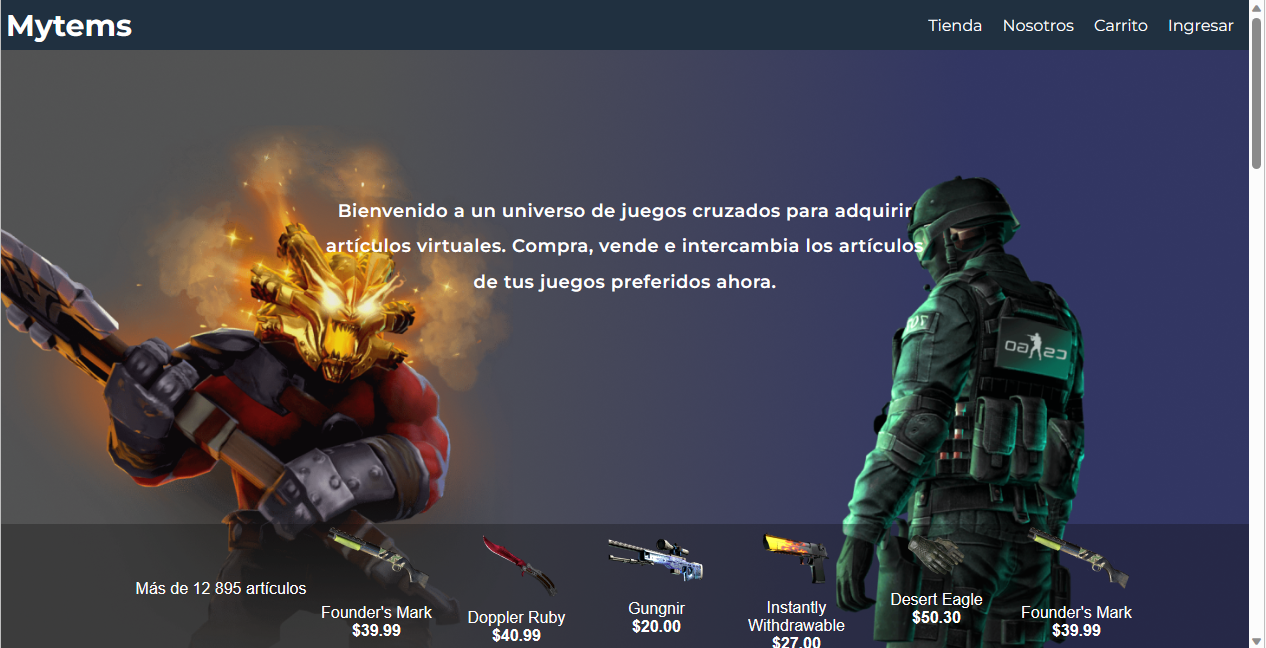
* **Pasos para ejecutar el aplicativo**

**## To start :3**

**docker-compose up**

**docker-compose up --build**

* **Interfaz de la aplicación**

****

# Backend

Como se pudo observar en la arquitectura el backend se desarrolló con python y flask como frameworks la cual se está ejecutando en el puerto 5000 la cual se encuentra dokerizada.

A continuación se presenta los componentes necesarios para levantar el servidor:

* **docker-compose.yml**

version: "3"

services:

app:

build:

context: .

dockerfile: Dockerfile

volumes:

- .:/app

ports:

- 5000:5000

restart: always

networks:

- tienda\_default

networks:

tienda\_default:

external: true

* **dockerfile**

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]

* pasos para levantar el servidor

**## Ejecuta el siguiente comando para construir la imagen de Docker:**

docker build -t flask-app .

Una vez que se haya construido la imagen, puedes ejecutar un contenedor basado en ella utilizando el siguiente comando:

docker run -p 5000:5000 flask-app

http://localhost:5000

docker-compose up -d

docker exec -it #### bash

**## In docker container terminal:**

pip install flask-cors

pip install pymongo

**## To postgrest**

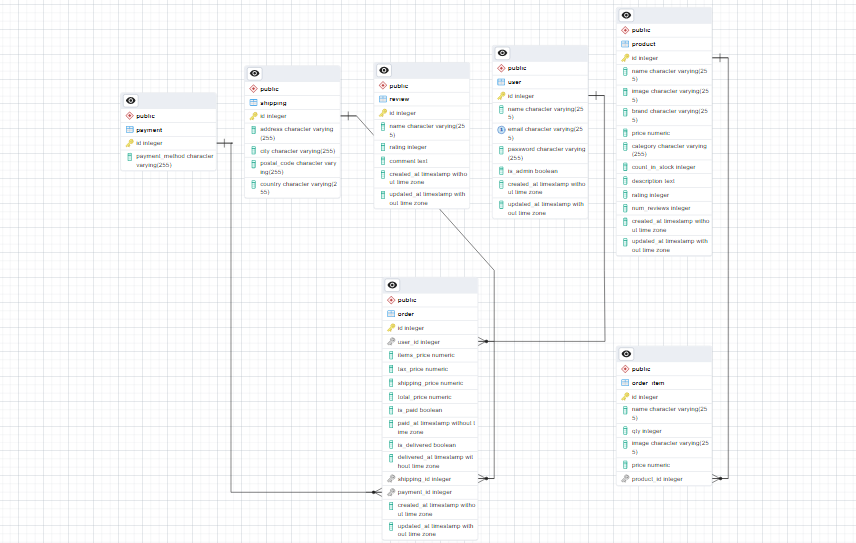
app.config['SQLALCHEMY\_DATABASE\_URI'] = 'postgresql://postgres:adminpassword@tienda-pgpool-1:5432/sa\_tienda'

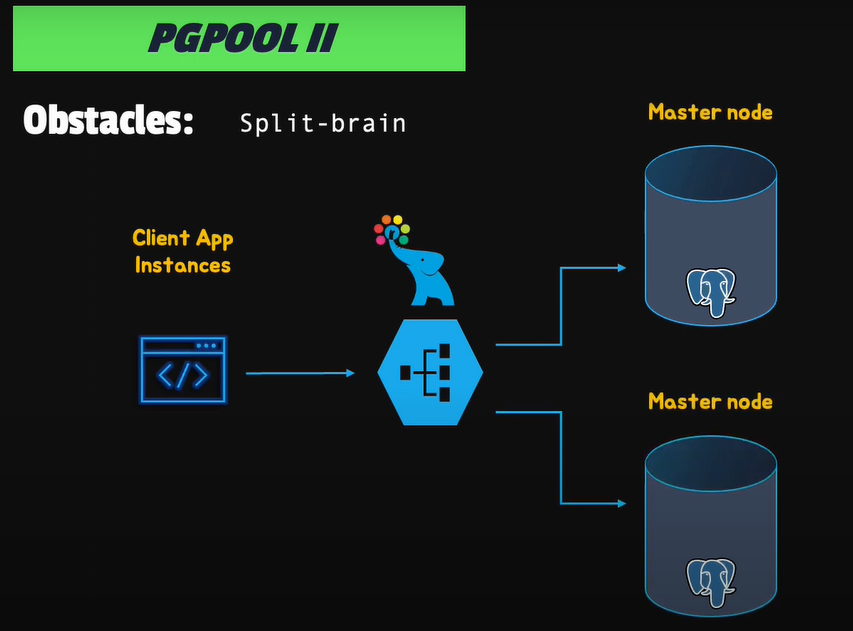
**## To mongo**

app.config['MONGO\_URI'] = 'mongodb://localhost:27018/amazona'

# 

* 1. **Base de datos distribuida con postgres master-master**

****

****

Para el desarrollo de la base de datos en postgres de manera relacional se ha realizado un modelo conceptual como base del proyecto.

para la implementación de esta base de datos se utilizó docker a continuación se detalla los pasos y elementos para construir la base de datos en un contenedor docker:

* **Docker-compose.yml**

version: "2.1"

services:

master-0:

image: docker.io/bitnami/postgresql-repmgr:14

ports:

- 5532:5432

volumes:

- master\_0\_data:/bitnami/postgresql

environment:

- POSTGRESQL\_POSTGRES\_PASSWORD=adminpassword

- POSTGRESQL\_USERNAME=grupo7

- POSTGRESQL\_PASSWORD=grupo7

- POSTGRESQL\_DATABASE=sa\_tienda

- POSTGRESQL\_NUM\_SYNCHRONOUS\_REPLICAS=1

- REPMGR\_PRIMARY\_HOST=master-0

- REPMGR\_PARTNER\_NODES=master-1,master-0

- REPMGR\_NODE\_NAME=master-0

- REPMGR\_NODE\_NETWORK\_NAME=master-0

- REPMGR\_USERNAME=repmgr

- REPMGR\_PASSWORD=repmgrpassword

master-1:

image: docker.io/bitnami/postgresql-repmgr:14

ports:

- 5533:5432

volumes:

- master\_1\_data:/bitnami/postgresql

environment:

- POSTGRESQL\_POSTGRES\_PASSWORD=adminpassword

- POSTGRESQL\_USERNAME=grupo7

- POSTGRESQL\_PASSWORD=grupo7

- POSTGRESQL\_DATABASE=sa\_tienda

- POSTGRESQL\_NUM\_SYNCHRONOUS\_REPLICAS=1

- REPMGR\_PRIMARY\_HOST=master-0

- REPMGR\_PARTNER\_NODES=master-0,master-1

- REPMGR\_NODE\_NAME=master-1

- REPMGR\_NODE\_NETWORK\_NAME=master-1

- REPMGR\_USERNAME=repmgr

- REPMGR\_PASSWORD=repmgrpassword

pgpool:

image: docker.io/bitnami/pgpool:4

ports:

- 5432:5432

environment:

- PGPOOL\_BACKEND\_NODES=0:master-0:5432,1:master-1:5432

- PGPOOL\_SR\_CHECK\_USER=repmgr

- PGPOOL\_SR\_CHECK\_PASSWORD=repmgrpassword

- PGPOOL\_ENABLE\_LDAP=no

- PGPOOL\_POSTGRES\_USERNAME=postgres

- PGPOOL\_POSTGRES\_PASSWORD=adminpassword

- PGPOOL\_ADMIN\_USERNAME=admin

- PGPOOL\_ADMIN\_PASSWORD=adminpassword

- PGPOOL\_ENABLE\_LOAD\_BALANCING=yes

- PGPOOL\_POSTGRES\_CUSTOM\_USERS=customuser

- PGPOOL\_POSTGRES\_CUSTOM\_PASSWORDS=custompassword

healthcheck:

test: ["CMD", "/opt/bitnami/scripts/pgpool/healthcheck.sh"]

interval: 10s

timeout: 5s

retries: 5

volumes:

master\_0\_data:

driver: local

master\_1\_data:

driver: local

* **create.sql**

CREATE DATABASE sa\_tienda;

-- Models

CREATE TABLE product (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

image VARCHAR(255) NOT NULL,

brand VARCHAR(255) NOT NULL,

price DECIMAL NOT NULL,

category VARCHAR(255) NOT NULL,

count\_in\_stock INTEGER DEFAULT 0,

description TEXT NOT NULL,

rating INTEGER DEFAULT 0,

num\_reviews INTEGER DEFAULT 0,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

CREATE TABLE "user" (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL,

is\_admin BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

CREATE TABLE shipping (

id SERIAL PRIMARY KEY,

address VARCHAR(255) NOT NULL,

city VARCHAR(255) NOT NULL,

postal\_code VARCHAR(255) NOT NULL,

country VARCHAR(255) NOT NULL

);

CREATE TABLE payment (

id SERIAL PRIMARY KEY,

payment\_method VARCHAR(255) NOT NULL

);

CREATE TABLE order\_item (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

qty INTEGER NOT NULL,

image VARCHAR(255) NOT NULL,

price DECIMAL NOT NULL,

product\_id INTEGER REFERENCES product(id) NOT NULL

);

CREATE TABLE "order" (

id SERIAL PRIMARY KEY,

user\_id INTEGER REFERENCES "user"(id) NOT NULL,

items\_price DECIMAL,

tax\_price DECIMAL,

shipping\_price DECIMAL,

total\_price DECIMAL,

is\_paid BOOLEAN DEFAULT FALSE,

paid\_at TIMESTAMP,

is\_delivered BOOLEAN DEFAULT FALSE,

delivered\_at TIMESTAMP,

shipping\_id INTEGER REFERENCES shipping(id),

payment\_id INTEGER REFERENCES payment(id),

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

CREATE TABLE review (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

rating INTEGER DEFAULT 0,

comment TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT NOW(),

updated\_at TIMESTAMP DEFAULT NOW()

);

-- Relations

CREATE TABLE product\_review (

product\_id INTEGER REFERENCES product(id),

review\_id INTEGER REFERENCES review(id),

PRIMARY KEY (product\_id, review\_id)

);

INSERT INTO product (name, image, brand, price, category, count\_in\_stock, description, rating, num\_reviews)

VALUES

('Product 1', 'https://cdn.glitch.global/934ab745-2e5a-4075-81a2-abc26b94ccc3/descarga%20(1).jpeg?v=1687540018218', 'Brand A', 9.99, 'Category 1', 10, 'Description 1', 4, 8),

('Product 2', 'https://cdn.glitch.global/934ab745-2e5a-4075-81a2-abc26b94ccc3/descarga%20(2).jpeg?v=1687540021636', 'Brand B', 19.99, 'Category 2', 5, 'Description 2', 3, 2),

('Product 3', 'image3.jphttps://cdn.glitch.global/934ab745-2e5a-4075-81a2-abc26b94ccc3/descarga.jpeg?v=1687540025774g', 'Brand C', 14.99, 'Category 1', 2, 'Description 3', 5, 12),

('Product 4', 'image4.https://cdn.glitch.global/934ab745-2e5a-4075-81a2-abc26b94ccc3/images.jpeg?v=1687540029398', 'Brand A', 29.99, 'Category 3', 8, 'Description 4', 2, 1),

('Product 5', 'https://cdn.glitch.global/934ab745-2e5a-4075-81a2-abc26b94ccc3/The-Last-of-Us-Box-Art.jpg?v=1687540035123', 'Brand B', 39.99, 'Category 2', 0, 'Description 5', 5, 15);

INSERT INTO "user" (name, email, password, is\_admin)

VALUES

('John Doe', 'john@example.com', 'password123', FALSE),

('Jane Smith', 'jane@example.com', 'securepass', TRUE),

('Alice Johnson', 'alice@example.com', 'mypassword', FALSE),

('Bob Brown', 'bob@example.com', 'pass123', FALSE),

('Eve Davis', 'eve@example.com', 'letmein', FALSE);

INSERT INTO shipping (address, city, postal\_code, country)

VALUES

('123 Main St', 'City A', '12345', 'Country X'),

('456 Elm St', 'City B', '54321', 'Country Y'),

('789 Oak St', 'City C', '67890', 'Country Z'),

('321 Pine St', 'City D', '09876', 'Country X'),

('654 Cedar St', 'City E', '56789', 'Country Y');

INSERT INTO payment (payment\_method)

VALUES

('Credit Card'),

('PayPal'),

('Bank Transfer'),

('Cash on Delivery'),

('Cryptocurrency');

INSERT INTO order\_item (name, qty, image, price, product\_id)

VALUES

('Product 1', 2, 'image1.jpg', 9.99, 1),

('Product 2', 1, 'image2.jpg', 19.99, 2),

('Product 3', 3, 'image3.jpg', 14.99, 3),

('Product 4', 1, 'image4.jpg', 29.99, 4),

('Product 5', 2, 'image5.jpg', 39.99, 5);

INSERT INTO "order" (user\_id, items\_price, tax\_price, shipping\_price, total\_price, is\_paid, paid\_at, is\_delivered, delivered\_at, shipping\_id, payment\_id)

VALUES

(1, 59.95, 4.50, 10.00, 74.45, TRUE, '2023-06-15 10:30:00', TRUE, '2023-06-17 14:20:00', 1, 1),

(2, 19.99, 1.50, 5.00, 26.49, TRUE, '2023-06-16 12:45:00', FALSE, NULL, 2, 3),

(3, 44.97, 3.37, 7.50, 55.84, FALSE, NULL, FALSE, NULL, 3, 4),

(4, 29.99, 2.25, 5.00, 37.24, TRUE, '2023-06-17 08:10:00', FALSE, NULL, 2, 2),

(5, 79.98, 6.00, 12.50, 98.48, TRUE, '2023-06-15 15:20:00', TRUE, '2023-06-18 11:05:00', 5, 1);

INSERT INTO review (name, rating, comment)

VALUES

('User A', 4, 'Great product!'),

('User B', 2, 'Not satisfied with the quality.'),

('User C', 5, 'Highly recommended.'),

('User D', 3, 'Average product.'),

('User E', 5, 'Love it! Will buy again.');

* 1. **Base de datos con sharding-mongosdb**

Para la realización del sharding se ha utilizado mongodb como base de datos no relacional.

A continuación se presenta los elementos que lo componen:

* **mongodb** 
  + **dockerfile**

**ROM mongo:latest**

**RUN apt-get update && apt-get -q install -y**

**COPY --chown=mongodb:mongodb mongod-start.sh /usr/local/bin/**

**COPY --chown=mongodb:mongodb mongod-runextra.sh /usr/local/bin/**

**RUN chmod u+x /usr/local/bin/mongod-start.sh /usr/local/bin/mongod-runextra.sh**

**ENTRYPOINT ["mongod-start.sh"]**

**CMD mongod -f /etc/mongod.conf**

* **mongos**
  + **dockerfile**

**FROM mongo:latest**

**RUN apt-get update && apt-get -q install -y**

**COPY --chown=mongodb:mongodb mongos-start.sh /usr/local/bin/**

**COPY --chown=mongodb:mongodb mongos-runextra.sh /usr/local/bin/**

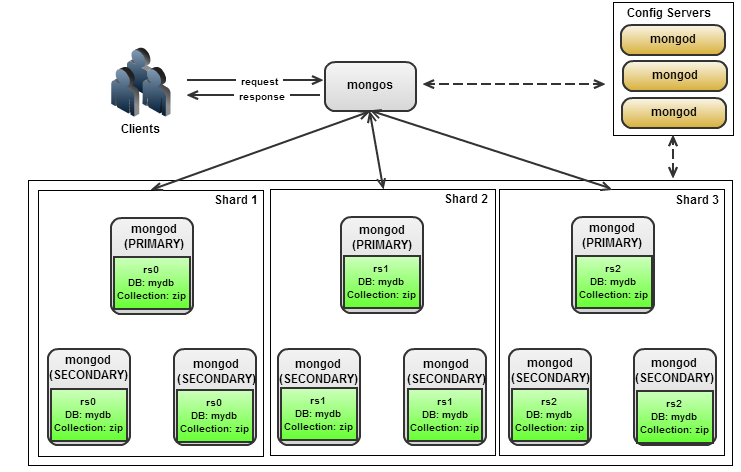
**RUN chmod u+x /usr/local/bin/mongos-start.sh /usr/local/bin/mongos-runextra.sh**

**ENTRYPOINT ["mongos-start.sh"]**

**CMD mongos --port 27017 --bind\_ip 0.0.0.0**

* **docker-compose.yml**

**El siguiente archivo está compuesto por 3 sharding**

****

**version: '3.8'**

**services:**

**######## MONGODB SHARD0 ########**

**shard0-replica0:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --shardsvr --replSet shard0**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- shard0-replica0**

**environment:**

**- REPSET\_NAME=shard0**

**- DO\_INIT\_REPSET=true**

**expose:**

**- "27017"**

**shard0-replica1:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --shardsvr --replSet shard0**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- shard0-replica1**

**environment:**

**- REPSET\_NAME=shard0**

**expose:**

**- "27017"**

**shard0-replica2:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --shardsvr --replSet shard0**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- shard0-replica2**

**environment:**

**- REPSET\_NAME=shard0**

**expose:**

**- "27017"**

**######## MONGODB SHARD1 ########**

**shard1-replica0:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --shardsvr --replSet shard1**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- shard1-replica0**

**environment:**

**- REPSET\_NAME=shard1**

**- DO\_INIT\_REPSET=true**

**expose:**

**- "27017"**

**shard1-replica1:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --shardsvr --replSet shard1**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- shard1-replica1**

**environment:**

**- REPSET\_NAME=shard1**

**expose:**

**- "27017"**

**shard1-replica2:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --shardsvr --replSet shard1**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- shard1-replica2**

**environment:**

**- REPSET\_NAME=shard1**

**expose:**

**- "27017"**

**######## MONGODB SHARD2 ########**

**shard2-replica0:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --shardsvr --replSet shard2**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- shard2-replica0**

**environment:**

**- REPSET\_NAME=shard2**

**- DO\_INIT\_REPSET=true**

**expose:**

**- "27017"**

**shard2-replica1:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --shardsvr --replSet shard2**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- shard2-replica1**

**environment:**

**- REPSET\_NAME=shard2**

**expose:**

**- "27017"**

**shard2-replica2:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --shardsvr --replSet shard2**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- shard2-replica2**

**environment:**

**- REPSET\_NAME=shard2**

**expose:**

**- "27017"**

**######## MONGODB CONFIGDB ########**

**configdb-replica0:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --configsvr --replSet configdb**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- configdb-replica0**

**environment:**

**- REPSET\_NAME=configdb**

**- DO\_INIT\_REPSET=true**

**expose:**

**- "27017"**

**configdb-replica1:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --configsvr --replSet configdb**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- configdb-replica1**

**environment:**

**- REPSET\_NAME=configdb**

**expose:**

**- "27017"**

**configdb-replica2:**

**build:**

**context: mongod**

**command: mongod -f /etc/mongod.conf --configsvr --replSet configdb**

**volumes:**

**- ./mongod/mongod.conf:/etc/mongod.conf**

**networks:**

**internalnetwork:**

**aliases:**

**- configdb-replica2**

**environment:**

**- REPSET\_NAME=configdb**

**expose:**

**- "27017"**

**######## MONGODB MONGOS ROUTERS ########**

**mongos-router0:**

**build:**

**context: mongos**

**command: mongos --port 27017 --bind\_ip 0.0.0.0 --configdb "configdb/configdb-replica0:27017,configdb-replica1:27017,configdb-replica2:27017"**

**depends\_on:**

**- shard0-replica0**

**- shard0-replica1**

**- shard0-replica2**

**- shard1-replica0**

**- shard1-replica1**

**- shard1-replica2**

**- shard2-replica0**

**- shard2-replica1**

**- shard2-replica2**

**- configdb-replica0**

**- configdb-replica1**

**- configdb-replica2**

**networks:**

**internalnetwork:**

**aliases:**

**- mongos-router0**

**environment:**

**- SHARD\_LIST=shard0/shard0-replica0:27017,shard0-replica1:27017,shard0-replica2:27017;shard1/shard1-replica0:27017,shard1-replica1:27017,shard1-replica2:27017;shard2/shard2-replica0:27017,shard2-replica1:27017,shard2-replica2:27017**

**expose:**

**- "27017"**

**ports:**

**- "27017:27017"**

**mongos-router1:**

**build:**

**context: mongos**

**command: mongos --port 27017 --bind\_ip 0.0.0.0 --configdb "configdb/configdb-replica0:27017,configdb-replica1:27017,configdb-replica2:27017"**

**networks:**

**internalnetwork:**

**aliases:**

**- mongos-router1**

**expose:**

**- "27017"**

**ports:**

**- "27018:27017"**

**networks:**

**internalnetwork: {}**

* **Json**

**[{'\_id': ObjectId('64af8e6dcb796f2732d6ca2a'), 'user': <Response 215 bytes [200 OK]>, 'orderItems': [{'name': 'Product 1', 'qty': 2, 'image': 'image\_url\_here', 'price': '19.99', 'product': 'product\_id\_here'}, {'name': 'Product 2', 'qty': 1, 'image': 'image\_url\_here', 'price': '29.99', 'product': 'product\_id\_here'}], 'shipping': {'address': '123 Main St', 'city': 'City', 'postalCode': '12345', 'country': 'Country'}, 'payment': {'paymentMethod': 'Credit Card'}, 'itemsPrice': 69.97, 'taxPrice': 5.6, 'shippingPrice': 10, 'totalPrice': 85.57, 'isPaid': True, 'paidAt': '2023-07-13T10:30:00Z', 'isDelivered': False, 'deliveredAt': None}]**

* **Pasos para activar el sharding**

**#comandos para shardear**

**sh.enableSharding("nombre\_base\_de\_datos")**

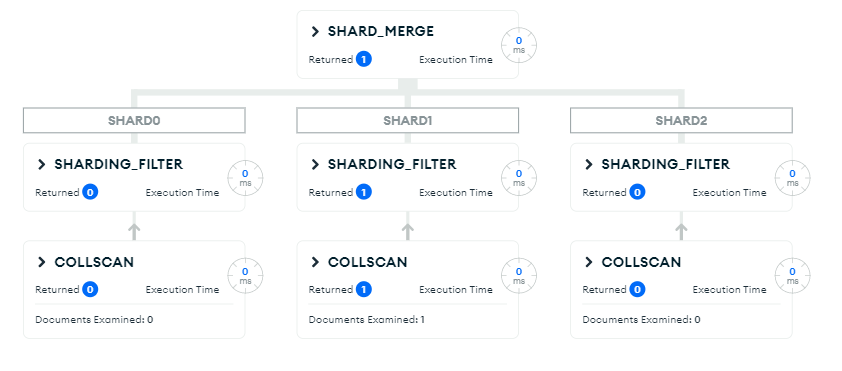
**use nombre\_base\_de\_datos**

**sh.shardCollection("nombre\_base\_de\_datos.coleccion”, {"dato\_compatible:"hashed"})**

**para probar si está funcionando**

**db.coleccion.getShardDistribution()**

* **Sharding**

****

# Resultados

La combinación de PostgreSQL y MongoDB en una base de datos distribuida y heterogénea proporciona una serie de beneficios y resultados positivos. Al aprovechar las fortalezas de ambos sistemas de gestión de bases de datos, se obtiene una solución más completa y versátil para almacenar y gestionar datos en un entorno distribuido.

Algunos de los resultados obtenidos incluyen:

Flexibilidad y escalabilidad mejoradas: MongoDB permite almacenar datos no estructurados en documentos JSON, lo que brinda una mayor flexibilidad para adaptarse a diferentes tipos de datos y esquemas cambiantes. Esto facilita la escalabilidad horizontal y el manejo eficiente de grandes volúmenes de datos.

Mayor agilidad en el desarrollo: La combinación de PostgreSQL y MongoDB permite a los desarrolladores elegir la mejor herramienta para cada caso de uso, lo que facilita el desarrollo de aplicaciones ágiles y adaptables. La capacidad de utilizar diferentes modelos de datos en la misma solución mejora la productividad y la eficiencia del equipo de desarrollo.manejo eficiente de relaciones complejas: PostgreSQL es especialmente adecuado para manejar datos estructurados y relaciones complejas entre entidades. Su soporte completo de SQL y características avanzadas de consulta permiten realizar consultas complejas y realizar operaciones sofisticadas en los datos relacionales.

Rendimiento optimizado: Al utilizar el sistema de gestión de bases de datos adecuado para cada tipo de dato, se puede optimizar el rendimiento general del sistema. PostgreSQL brinda un rendimiento sólido para consultas relacionales, mientras que MongoDB se destaca en escenarios con una alta carga de lectura y escritura de datos no estructurados.

# 

# Conclusión

La implementación de una arquitectura de base de datos distribuida y heterogénea con PostgreSQL y MongoDB ha mejorado significativamente el rendimiento, la escalabilidad y la flexibilidad en el manejo de datos del sistema de ventas de videojuegos. Esta solución ha optimizado la utilización de recursos, mejorado la disponibilidad y confiabilidad del sistema, y permitido adaptarse de manera ágil a las necesidades cambiantes. La combinación de PostgreSQL para datos estructurados y relaciones complejas, junto con MongoDB para datos no estructurados, ha sentado las bases para un sistema eficiente y adaptable, capaz de satisfacer las demandas actuales y futuras de la empresa.

# Referencia

***Sharding*. (s/f). Mongodb.com. Recuperado el 14 de julio de 2023, de** [**https://www.mongodb.com/docs/manual/sharding/**](https://www.mongodb.com/docs/manual/sharding/)

***Docker*. (s/f). Docker.com. Recuperado el 14 de julio de 2023, de** [**https://hub.docker.com/r/bitnami/pgpool/**](https://hub.docker.com/r/bitnami/pgpool/)

**Mamidwar, S. (2023, marzo 24). How to Install MongoDB on Ubuntu 22.04 LTS. *FOSS TechNix*.** [**https://www.fosstechnix.com/how-to-install-mongodb-on-ubuntu-22-04-lts/**](https://www.fosstechnix.com/how-to-install-mongodb-on-ubuntu-22-04-lts/)

***Postgresql and logical replication*. (s/f). Raveland.Tech. Recuperado el 14 de julio de 2023, de** [**https://blog.raveland.tech/post/postgresql\_lr\_en/**](https://blog.raveland.tech/post/postgresql_lr_en/)

***Sharded cluster components*. (s/f). Mongodb.com. Recuperado el 14 de julio de 2023, de https://www.mongodb.com/docs/manual/core/sharded-cluster-components**