REPORTE DE PROYECTO

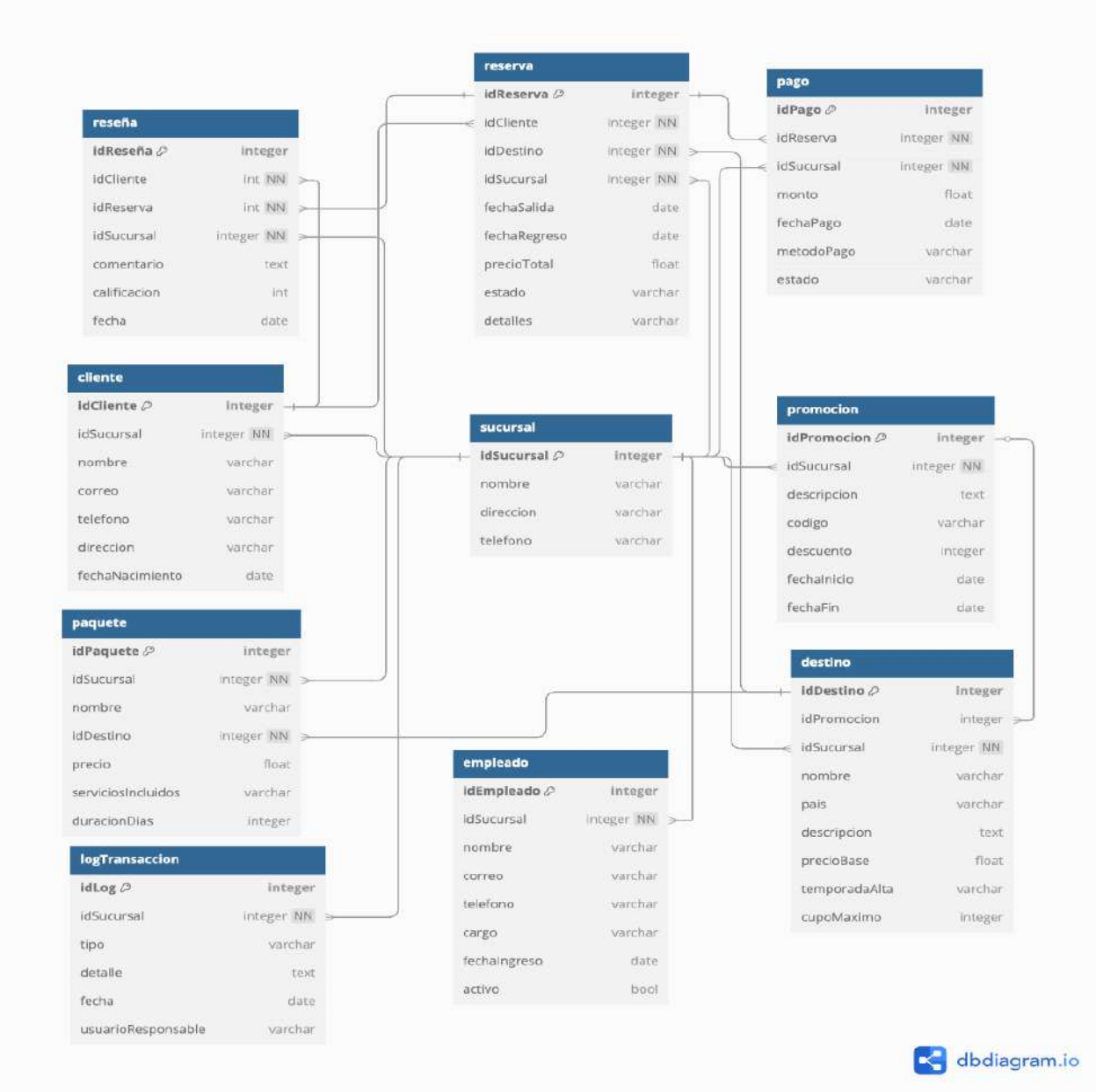
AGENCIA DE VIAJES

- Israel Campos Vázquez
- Luis Erick Esperilla Mendoza
- Axel Damian Ortiz Simon
- José Eduardo Valles Aguilera

Dr. Eduardo Cornejo-Velázquez

1. Descripción

Como proyecto para la materia de Base de Datos Distribuidas haremos una base de datos para una agencia de viajes. La haremos con el propósito de poder almacenar, gestionar y distribuir información relacionada con reservas, clientes, empleados, reseñas, paquetes, sucursales, promociones, destinos y pagos de manera eficiente. Elegiremos MongoDB para implementar nuestra base de datos para explorar y descubrir una base de datos no relacional. Además, usaremos mongoDB debido a su capacidad de escalabilidad horizontal mediante sharding (fragmentación), lo que permite manejar grandes volúmenes de datos y múltiples sucursales sin afectar el rendimiento.
A continuación mostramos un diagrama con la información que manejaremos en nuestra base de datos. En este caso, al tratarse de mongoDb, las tablas representarán colecciones, los atributos representarán llaves o claves, y las filas representarán documentos:



2. Objetivo General

Diseñar e implementar una base de datos distribuida con fragmentación horizontal utilizando sharding en MongoDB para analizar un caso práctico de una base de datos distribuida, evaluando sus beneficion en la eficiencia de consultas desde cualquiera de los nodos.

3. Objetivos Específicos

1.- Analizar cómo es el modelo de negocio de una agencia de viajes para determinar la estructura del sharding.
2.- Diseñar el esquema de la base de datos en MongoDB, considerando que la fragmentación de datos será por el id de las sucursales.
3.- Implementar el sharding de MongoDB en un modo de prueba o desarrollo, configurando los config servers, shards y el router mongos.
4.- Optimizar el funcionamiento de la base de datos creando índices para las consultas más recurrentes.
5.- Desarrollar una documentación a manera de bitácora o instructivo para posteriores desarrollos.

4. Alcances y limitaciones

Alcances

-Distribución eficiente de los datos: Mejoraremos la disponibilidad y escalabilidad de una base de datos para una agencia de viajes utilizando Sharding de MongoDB.
-Optimización del rendimiento: La latencia en consultas debe ser mejor en comparación a un sistema centralizado gracias al particionamiento de datos.
-Tolerancia a fallos: Cada servidor lo configuraremos como un Replica Set(replicación), para que en el caso de que un nodo del servidor falle, tenga mínimo otros dos nodos de respaldo.
-Escalabilidad horizontal: Se diseñará el sistema de manera que permita a la agencia de viajes añadir más sucursales sin afectar la operación del sistema.
-Documentación técnica: Haremos una documentación detallada sobre la configuración e implementación de bases de datos distribuidas con MongoDB.

Limitaciones

-Dificultad en la implementación: La implementación de una base de datos distribuida puede ser más compleja que una centralizada, y requiere de mayores conocimientos técnicos.
-Costos de infraestructura: Se requiere más equipo para una base de datos distribuida, ya que necesitas conectar tus nodos a una misma red.
-Dependencia de los shards: Es posible que si un shard no está activo o en funcionamiento, no puedas manipular la base de datos.

5. Marco teórico

Metodología de análisis: UML

Para analizar cómo interactuará un cliente y un administrador con la base de datos distribuida, utilizaremos UML (Unified Modeling Language). UML es un lenguaje de modelado estándar que nos permite representar visualmente los procesos, actores y estructuras del sistema.

Dentro de esta metodología, se empleará el siguiente diagrama:

1. Diagrama de Casos de Uso Este diagrama mostrará las interacciones entre los clientes y los administradores con el sistema de agencia de viajes. Permitirá identificar las funcionalidades clave, como la gestión de reservas y consulta de disponibilidad.

Metodología de diseño: Microservicios

Para el diseño de la base de datos y su implementación usaremos la arquitectura basada en microservicios, lo cual facilita la escalabilidad y modularidad.
Definiremos sucursalId como la clave de sharding para garantizar un balance adecuado en la distribución de datos.
Modelado del sistema con diagrama Relacional.

Metodología de desarrollo: Scrum

Scrum nos permite:
-Dividir el desarrollo en sprints.
-Trabajar con entregables en cada iteración.
-Facilitar la adaptación del sistema a nuevos requerimientos conforme avanza el proyecto.

El desarrollo incluiría:
-Configuración del clúster de MongoDB con shards y réplicas.
-Implementación del backend con Flask para la gestión de la base de datos.
-Desarrollo de endpoints para la consulta y manipulación de datos distribuidos.

6. Análisis de requerimientos

Como ya se mencionó anteriormente, la base de datos de agencia de viajes debe permitir gestionar información relacionada con clientes, destinos, reservas, reseñas, empleados, paquetes, sucursales, promociones y pagos de manera eficiente y escalable. El sistema debe ser rápido, confiable y accesible desde múltiples sucursales.

Requerimientos Funcionales

Clientes:
-Registrar nuevos clientes con datos personales.
-Consultar clientes.
-Almacenar el histoial de reservas de cada cliente.

Destinos:
-Registrar y modificar información sobre destinos turísticos.
-Consultar destinos según país, precio y temporada alta.

Reservas:
-Permitir a los clientes realizar reservas de viajes.
-Verificar disponibilidad de fechas.

-Registrar cambios o cancelaciones de reservas.

Pagos:
-Registrar pagos asociados a reservas.
-Permitir diferentes métodos de pago.
-Consultar pagos por cliente o fecha.

Reseñas:
-Permitir a los clientes realizar reseñas de sus viajes.
-Permitir a los clientes calificar la experiencia de sus viajes.

Empleados:
-Permitir a la agaencia llevan un control de sus empleados.

Sucursales:
-Permitir a la agencia de viajes agregar sucursales.
-Permitir a la agencia de viajes saber qué empleados se encuentran en cada sucursal.

Paquetes:
-Permitir a los clientes comprar paquetes de viajes.

Promociones:
-Permitir a la agencia de viajes agregar promociones a ciertos destino.
-Permitir a los clientes reservar con promoción para determinados destinos.

Requerimientos No Funcionales

Escalabilidad y Distribución:
-Implementar sharding(fragmentación) para soportar alta carga de datos.
-Permitir la operación desde diferentes sucursales sin pérdida de rendimiento.

Casos de Uso

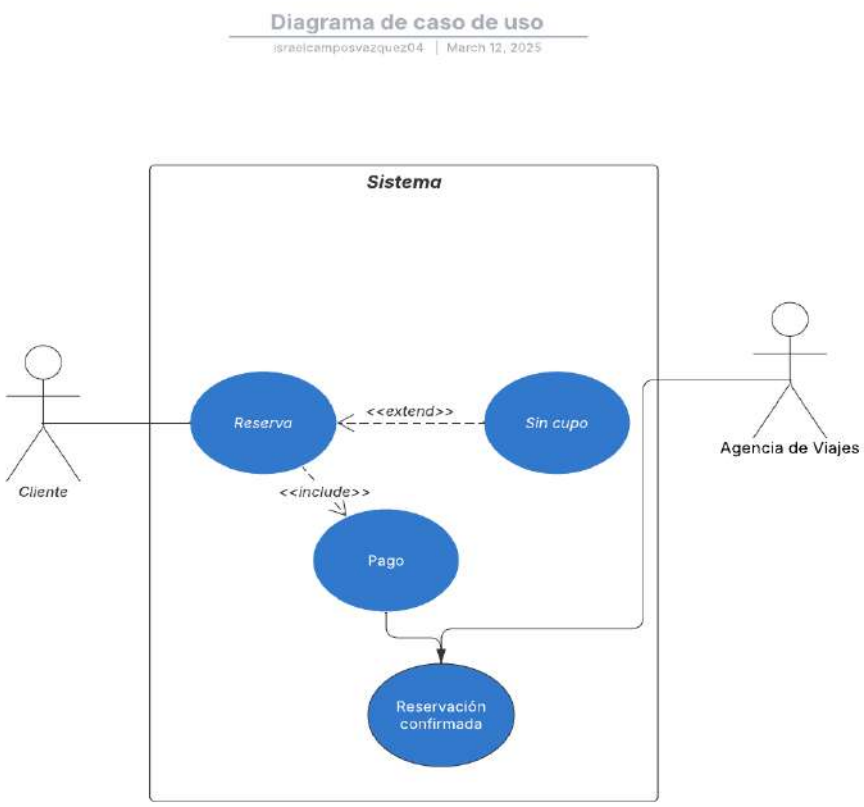


Diagrama de caso de uso
israelcamposvazquez04 | March 12, 2025
Sistema
Cliente
Reserva
<<extend>>
Sin cupo
Agencia de Viajes
<<include>>
Pago
Reservación confirmada

7. Herramientas empleadas

1. DataGrip: Es un entorno de desarrollo integrado (IDE) para bases de datos creado por JetBrains. Está diseñado para ayudar a los desarrolladores y administradores de bases de datos a gestionar, consultar y optimizar sus bases de datos de manera eficiente.

2. MongoDB: Es una base de datos NoSQL orientada a documentos, lo que significa que almacena la información en formato JSON o BSON en lugar de tablas y filas como en bases de datos relacionales (SQL).

8. Desarrollo

8.1 Teoría

Informándonos con la documentación oficial de MongoDB, descubrimos que para hacer sharding (fragmentación) necesitamos un Replica Set que actúe como config server, un Replica set que actúe como shard server y otra instancia o nodo que actúe como router mongos. Puede que no conozcamos el significado de ciertos términos, pero a continuación los explicaremos para comprender cómo funciona MongoDB para hacer fragmentación.

El servidor de base de datos MongoDB tiene un programa **daemon** o demonio (proceso en segundo plano) que es el encargado principal de gestionar las bases de datos, manejar las conexiones de clientes (mongo, MongoDB, Compass, Flask, Node, etc) y ejecutar operaciones de lectura y escritura.

Un **Replica Set** (conjunto de réplicas) en MongoDB es un grupo de procesos mongod (se recomienda que sean mínimo 3) que mantienen el mismo conjunto de datos. Los Replica Sets nos brindan redundancia y una gran disponibilidad de los datos, y son la base de cualquier desarrollo en producción.

Usar Replica Sets nos permite tener replicación en nuestra base datos, ya que el tener la misma información en cada uno de los servidores o nodos del Replica Set, si alguno falla, los demás cuentan con los mismos datos del servidor que falló.

Uno, y sólo uno de los nodos del Replica Set debe ser denominado como nodo primario, y el resto serán nodos secundarios. Cada uno de los nodos de un Replica Set debe pertenecer a uno, y sólo un Replica Set.

El nodo **primario** es el único servidor que puede confirmar operaciones de escritura a través del write concern w: "majority" , aunque en algunas circunstancias se puede cambiar el tipo de write concern. Este mismo nodo guarda todos los cambios en sus datos en su **oplog** (registro de operaciones).

Los nodos **secundarios** replican el oplog del nodo primario y aplican las operaciones a su conjunto de datos de tal manera que la información de los nodos secundarios refleje la misma información que tiene el nodo primario. Ahora que ya sabemos que es un Replica Set, podemos entrar más en detalle en el Sharding

El **sharding** es un método para distribuir datos a través de múltiples máquinas. MongoDB recomienda el uso del sharding para bases de datos con una gran cantidad de información y para sistemas que requieran un excelente rendimiento en consultas.

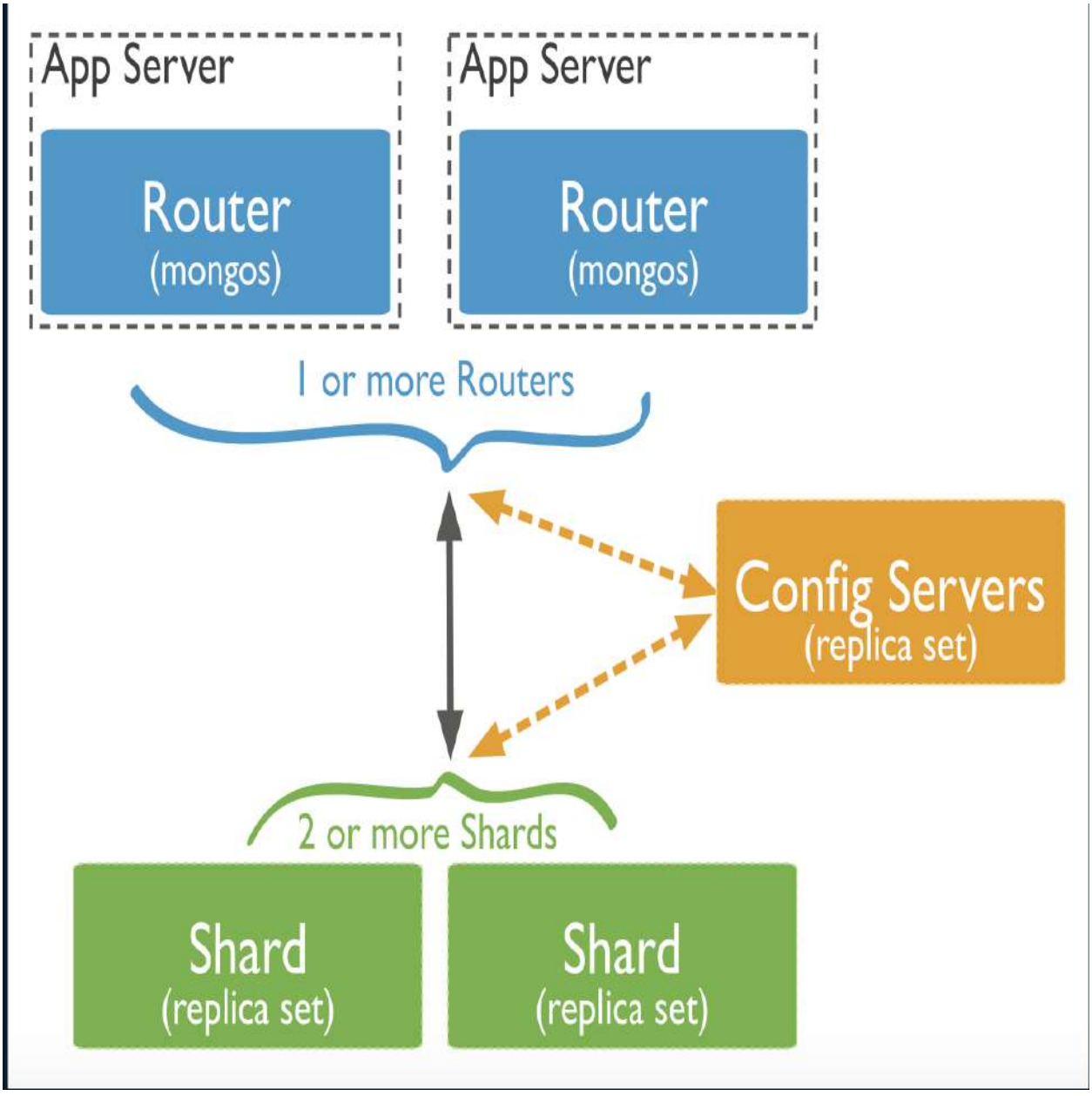
Si nuestro sistema está creciendo velozmente, podemos usar escalado vertical u horizontal. El **escalado vertical** consiste en aumentar las capacidades de hardware de nuestro servidor (agregando más memoria RAM por ejemplo). El **escalado horizontal** implica dividir los datos del sistema en múltiples servidores, e ir añadiendo servidores según sea necesario. Esto repartirá la carga de trabajo de nuestro primer servidor entre múltiples servidores, esto brindará un excelente rendimiento en consultas, para lo que está hecho el sharding.

Como mencionamos al principio, los componentes del sharding son: fragmentos (shards), servidores de configuración (config servers) y un router (mongos). MongoDB fragmenta los datos a nivel de colección.

Cada **shard** contiene un subconjunto de los datos fragmentados. Cada fragmento debe desplegarse como un Replica Set.

Mongos actúa como un enrutador de consultas, brindandonos una interfaz entre aplicaciones cliente y el sharded cluster (grupo fragmentado). Debemos conectarnos a un router mongos para interactuar con cualquier colección en el sharded cluster. Esto incluye colecciones fragmentadas y no fragmentadas. Los clientes nunca deben conectarse a un solo shard para realizar operaciones de lectura y escritura.

Los **Config Servers** almacenan metadatos y ajustes de configuración para el clúster. Los servidores de configuración deben implementarse como un Replica Set.



MongoDB utiliza **shard key** (clave de fragmento) para distribuir documentos de las colecciones a través de los shards. Un shard key es un campo o varios campos dentro de los documentos. El campo o los campos shard key que falten se tratarán como si tuvieran valores nulos al distribuir los documentos a través de los shards, pero no cuando se enruten consultas.

Para fragmentar una colección poblada, la colección debe tener un **index** (índice) que comience con la shard key. Si fragmentamos una colección vacía, automáticamente MongoDB creará un índice de soporte si la colección aún no tiene un índice apropiado para la shard key especificada.

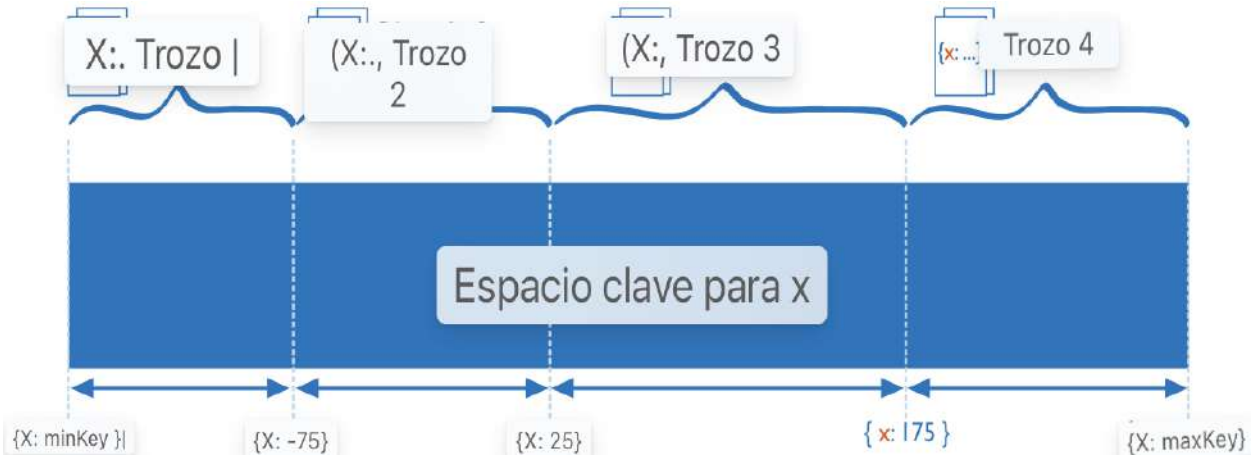
Podemos tener una base de datos que combine colecciones sin fragmentar con colecciones fragmentadas. Las colecciones fragmentadas se dividen y se distribuyen a través de los shards del cluster. Las colecciones no fragmentadas se pueden ubicar en cualquier shard, pero no en varios a la vez.

Las particiones en MongoDB fragmentarán los datos en **chunks** (trozos). Cada shard tiene un rango inferior inclusivo y superior exclusivo basado en la shard key.

MongoDB tiene dos estrategias de sharding para distribuir los datos a traves del sharded cluster:

-**Hashed Sharding** (Fragmentación en picado) implica calcular un hash del valor del campo shard key. A cada shard se le asigna un rango basado en los valores de la hashed shard key. MongoDB calcula automáticamente los hashes al resolver consultas utilizando índices hash.

-**Ranged Sharding** (Fragmentación en rangos) implica dividir los datos en rangos basados en los valores del shard key. A cada shard se le asigna un rango.



En sharded clusters podemos crear **zonas** de datos fragmentados basados en la shard key. Podemos asociar cada zona a uno o más shards en el cluster. Un shard puede asociarse con cualquier número de zonas.

Cada zona cubre uno o más rangos de valores shard key.Cada rango que cubre una zona siempre incluye su límite inferior y excluye su límite superior. Debemos usar campos contenidos en la shard key al definir un nuevo rango para una zona a cubrir.

Configurar zonas y rangos de zonas antes de fragmentar una colección vacía o inexistente permite una configuración más rápida de fragmentación en zonas.

Una vez que ya conocemos los fundamentos de la fragmentación en MongoDB, podemos pasar a la práctica.

8.2 Despliegue de Config Servers

Para este proyecto utilizamos 4 laptops, las conectamos a un switch para que tengan comunicación entre sí mediante cable ethernet, configuramos a cada una una IPv4 estática dentro de una misma subred, y por último probamos la conexión haciendo ping entre ellas. Cada laptop ya tiene MongoDB instalado.

Listing 1: Paso 1

En el directorio home de una laptop creamos las siguientes carpetas:

```
mkdir -p shard-demo/configsrv shard-demo/configsrv1 shard-demo/configsrv2
```

Listing 2: Paso 2

En esa misma laptop, iniciamos un Replica Set para los servidores de configuración:

```
nohup mongod --configsvr --port 28041 --bind_ip 192.168.1.10 --replSet config_repl
--dbpath shard-demo/configsrv &

nohup mongod --configsvr --port 28042 --bind_ip 192.168.1.10 --replSet config_repl
--dbpath shard-demo/configsrv1 &

nohup mongod --configsvr --port 28043 --bind_ip 192.168.1.10 --replSet config_repl
--dbpath shard-demo/configsrv2 &
```

Para comprobar que nuestros Config Servers están en funcionamiento, usamos el siguiente comando: `ps aux | grep mongo`

Nos conectamos al primer Config Server con el siguiente comando:

```
mongosh --host 192.168.1.10 --port 28041
```

```
rm: /tmp/mongo.sock: No such file or directory
israelcamposvazquez@MacBook-Air-de-Israel ~ % nohup mongod --configsvr  --port 28041 --bind_ip 192.
168.1.10 --replSet config_repl --dbpath shard-demo/configsrv &
[1] 77859
israelcamposvazquez@MacBook-Air-de-Israel ~ % appending output to nohup.out
israelcamposvazquez@MacBook-Air-de-Israel ~ % ps aux | grep mongo
israelcamposvazquez 77859   0.2  0.7 412012944 120160 s000  SN    2:35PM   0:00.43 mongod --configs
vr --port 28041 --bind_ip 192.168.1.10 --replSet config_repl --dbpath shard-demo/configsrv
israelcamposvazquez 77999   0.0  0.0 410724096   1424 s000  S+    2:35PM   0:00.00 grep mongo
israelcamposvazquez@MacBook-Air-de-Israel ~ % nohup mongod --configsvr  --port 28042 --bind_ip 192.
168.1.10 --replSet config_repl --dbpath shard-demo/configsrv1 &
[2] 78388
```

```
israelcamposvazquez@MacBook-Air-de-Israel ~ % nohup mongod --configsvr  --port 2
8043 --bind_ip 192.168.1.10 --replSet config_repl --dbpath shard-demo/configsrv2
 &
[1] 79442
israelcamposvazquez@MacBook-Air-de-Israel ~ % appending output to nohup.out
```

Listing 3: Paso 3

```
#Definimos la configuraci n del Replica Set y lo inicializamos:
rsconf = {
    _id: "config_repl",
    members: [
                {
                    _id: 0,
                    host: "192.168.1.10:28041"
                },
                {
                    _id: 1,
                    host: "192.168.1.10:28042"
                },
                {
                    _id: 2,
                    host: "192.168.1.10:28043"
                }
            ]
        }
rs.initiate(rsconf)

#Para checar el status actual de nuestro Replica Set usamos el comando:
rs.status()
```

8.3 Despliegue de Shard Servers

Listing 4: Paso 1

En el directorio home de la misma laptop creamos las siguientes carpetas:
mkdir –p shard–demo/shardrep1 shard–demo/shardrep2 shard–demo/shardrep3

Listing 5: Paso 2

Iniciamos un Replica Set para los shard servers:
nohup mongod —shardsvr —port 28081 —bind_ip 192.168.1.10 —replSet shard_repl —dbpath shard–demo/shardrep1 &

nohup mongod —shardsvr —port 28082 —bind_ip 192.168.1.10 —replSet shard_repl —dbpath shard–demo/shardrep2 &

nohup mongod —shardsvr —port 28083 —bind_ip 192.168.1.10 —replSet shard_repl —dbpath shard–demo/shardrep3 &

```
israelcamposvazquez@MacBook-Air-de-Israel ~ % nohup mongod --shardsvr --port 280
81 --bind_ip 192.168.1.10 --replSet shard_repl --dbpath shard-demo/shardrep1 &
[1] 81808
israelcamposvazquez@MacBook-Air-de-Israel ~ % appending output to nohup.out

[1]  + abort      nohup mongod --shardsvr --port 28081 --bind_ip 192.168.1.10 --
replSet
israelcamposvazquez@MacBook-Air-de-Israel ~ %
```

```
israelcamposvazquez 81808   0.8  0.9 412032352 151472 s002  SN    2:40PM   0:29.45 mongod --shardsv
r --port 28081 --bind_ip 192.168.1.10 --replSet shard_repl --dbpath shard-demo/shardrep1
israelcamposvazquez 82094   0.7  0.9 412031792 151440 s003  SN    2:41PM   0:28.81 mongod --shardsv
r --port 28082 --bind_ip 192.168.1.10 --replSet shard_repl --dbpath shard-demo/shardrep2
israelcamposvazquez 82504   0.7  0.9 412037152 156464 s004  SN    2:41PM   0:29.25 mongod --shardsv
r --port 28083 --bind_ip 192.168.1.10 --replSet shard_repl --dbpath shard-demo/shardrep3
```

Listing 6: Paso 3

Nos conectamos al primer Shard Server con el siguiente comando:
mongosh —host 192.168.1.10 —port 28081

```
israelcamposvazquez@MacBook-Air-de-Israel ~ % mongosh --host 192.168.1.10 --port 28081
Current Mongosh Log ID: 67e31526586f28eb9061d57d
Connecting to:          mongodb://192.168.1.10:28081/?directConnection=true&appName=mongosh+2.4.2
Using MongoDB:          8.0.6
Using Mongosh:          2.4.2

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-03-25T14:40:39.393-06:00: Access control is not enabled for the database. Read and write acc
ess to data and configuration is unrestricted
   2025-03-25T14:40:39.393-06:00: Soft rlimits for open file descriptors too low
------
```

Listing 7: Paso 4

```
#Definimos la configuraci n del Replica Set y lo inicializamos:
rsconf = {
    _id: "shard_repl",
    members: [
                {
                    _id: 0,
                    host: "192.168.1.10:28081"
                },
                {
                    _id: 1,
                    host: "192.168.1.10:28082"
                },
                {
                    _id: 2,
                    host: "192.168.1.10:28083"
                }
            ]
        }
rs.initiate(rsconf)
```

```
test> rsconf = {
...     _id: "shard_repl",
...     members: [
...                 {
...                     _id: 0,
...                     host: "192.168.1.10:28081"
...                 },
...                 {
...                     _id: 1,
...                     host: "192.168.1.10:28082"
...                 },
...                 {
...                     _id: 2,
...                     host: "192.168.1.10:28083"
...                 }
...             ]
...         }
{
  _id: 'shard_repl',
  members: [
    { _id: 0, host: '192.168.1.10:28081' },
    { _id: 1, host: '192.168.1.10:28082' },
    { _id: 2, host: '192.168.1.10:28083' }
  ]
}
test> rs.initiate(rsconf)
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742935351, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1742935351, i: 1 })
}
```

Listing 8: Paso 5

En el directorio home de la segunda laptop creamos las siguientes carpetas:
mkdir -p shard-demo/shard2rep1 shard-demo/shard2rep2 shard-demo/shard2rep3

Listing 9: Paso 6

Iniciamos un Replica Set para los shard servers:
nohup mongod --shardsvr --port 28081 --bind_ip 192.168.1.11 --replSet shard2_repl --dbpath shard-demo/shard2rep1 &

nohup mongod --shardsvr --port 28082 --bind_ip 192.168.1.11 --replSet shard2_repl --dbpath shard-demo/shard2rep2 &

nohup mongod --shardsvr --port 28083 --bind_ip 192.168.1.11 --replSet shard2_repl --dbpath shard-demo/shard2rep3 &

Nos conectamos al primer Shard Server con el siguiente comando:
mongosh --host 192.168.1.11 --port 28081

Listing 10: Paso 7

```
#Definimos la configuraci n del Replica Set y lo inicializamos:
rsconf = {
    _id: "shard2_repl",
    members: [
                {
                    _id: 0,
                    host: "192.168.1.11:28081"
                },
                {
                    _id: 1,
                    host: "192.168.1.11:28082"
                },
                {
                    _id: 2,
                    host: "192.168.1.11:28083"
                }
            ]
        }
rs.initiate(rsconf)
```

Listing 11: Paso 8

En el directorio home de la tercer laptop creamos las siguientes carpetas:
mkdir -p shard-demo/shard3rep1 shard-demo/shard3rep2 shard-demo/shard3rep3

Listing 12: Paso 9

Iniciamos un Replica Set para los shard servers:
nohup mongod --shardsvr --port 28081 --bind_ip 192.168.1.12 --replSet shard3_repl --dbpath shard-demo/shard3rep1 &

nohup mongod --shardsvr --port 28082 --bind_ip 192.168.1.12 --replSet shard3_repl --dbpath shard-demo/shard3rep2 &

nohup mongod --shardsvr --port 28083 --bind_ip 192.168.1.12 --replSet shard3_repl --dbpath shard-demo/shard3rep3 &

Nos conectamos al primer Shard Server con el siguiente comando:
mongosh —host 192.168.1.12 —port 28081

Listing 13: Paso 10

```
#Definimos la configuraci n del Replica Set y lo inicializamos:
rsconf = {
    _id: "shard3_repl",
    members: [
                {
                    _id: 0,
                    host: "192.168.1.12:28081"
                },
                {
                    _id: 1,
                    host: "192.168.1.12:28082"
                },
                {
                    _id: 2,
                    host: "192.168.1.12:28083"
                }
            ]
        }
rs.initiate(rsconf)
```

Listing 14: Paso 11

En el directorio home de la cuarta laptop creamos las siguientes carpetas:
mkdir –p shard–demo/shard4rep1 shard–demo/shard4rep2 shard–demo/shard4rep3

Iniciamos un Replica Set para los shard servers:
nohup mongod —shardsvr —port 28081 —bind_ip 192.168.1.13 —replSet shard4_repl —dbpath shard–demo/shard4rep1 &

nohup mongod —shardsvr —port 28082 —bind_ip 192.168.1.13 —replSet shard4_repl —dbpath shard–demo/shard4rep2 &

nohup mongod —shardsvr —port 28083 —bind_ip 192.168.1.13 —replSet shard4_repl —dbpath shard–demo/shard4rep3 &

Nos conectamos al primer Shard Server con el siguiente comando:
mongosh —host 192.168.1.13 —port 28081

Listing 15: Paso 13

```
#Definimos la configuraci n del Replica Set y lo inicializamos:
rsconf = {
    _id: "shard4_repl",
    members: [
                {
                    _id: 0,
                    host: "192.168.1.13:28081"
                },
                {
                    _id: 1,
                    host: "192.168.1.13:28082"
                },
```

```
                {
                    _id: 2,
                    host: "192.168.1.13:28083"
                }
            ]
        }
rs.initiate(rsconf)
```

8.4 Despliegue de Router Mongos

Listing 16: Paso 1

Inicializamos mongos con el siguiente comando:
nohup mongos —configdb config_repl/192.168.1.10:28041,192.168.1.10:28042,
192.168.1.10:28043 —bind_ip 192.168.1.10 &

```
israelcamposvazquez@MacBook-Air-de-Israel ~ % nohup mongos --configdb config_repl/192.168.1.10:2804
1,192.168.1.10:28042,192.168.1.10:28043 --bind_ip 192.168.1.10 &

[3] 12649
israelcamposvazquez@MacBook-Air-de-Israel ~ % appending output to nohup.out
```

Listing 17: Paso 2

Nos conectamos al Sharded Cluster a trav s de mongos con el siguiente comando:
mongosh —host 192.168.1.10 —port 27017

```
israelcamposvazquez@MacBook-Air-de-Israel ~ % mongosh --host 192.168.1.10 --port 27017
Current Mongosh Log ID: 67e321bddae36e992eaf1592
Connecting to:          mongodb://192.168.1.10:27017/?directConnection=true&appName=mongosh+2.4.2
```

Listing 18: Paso 3

```
#Si quisieramos modificar el tamanio de los chunks (por defecto es de 64MB)
    podemos hacer lo siguiente:
use config
db.settings.updateOne(
  { _id: "chunksize" },
  { $set: { _id: "chunksize", value: 2 } },
  { upsert: true }
)

#Aniadimos los shards:
sh.addShard("shard_repl/192.168.1.10:28081,192.168.1.10:28082,192.168.1.10:28083")
sh.addShard("shard2_repl/192.168.1.11:28081,192.168.1.11:28082,192.168.1.11:28083"
    )
sh.addShard("shard3_repl/192.168.1.12:28081,192.168.1.12:28082,192.168.1.12:28083"
    )
sh.addShard("shard4_repl/192.168.1.13:28081,192.168.1.13:28082,192.168.1.13:28083"
    )
```

```
[direct: mongos] test> sh.addShard("shard_repl/192.168.1.10:28081,192.168.1.10:28082,192.168.1.10:28
083")
{
  shardAdded: 'shard_repl',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742939025, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1742939025, i: 1 })
}
[direct: mongos] test> sh.addShard("shard2_repl/192.168.1.11:28081,192.168.1.11:28082,192.168.1.11:2
8083")
{
  shardAdded: 'shard2_repl',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742939034, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1742939034, i: 1 })
}
[direct: mongos] test> sh.addShard("shard3_repl/192.168.1.12:28081,192.168.1.12:28082,192.168.1.12:2
8083")
{
  shardAdded: 'shard3_repl',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742939045, i: 49 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1742939045, i: 49 })
}
```

Listing 19: Paso 4

```
#Creamos la Base de Datos:
use agenciaViajes

#Habilitamos el sharding en la base de datos:
sh.enableSharding("agenciaNegocios")

#Creamos colecciones con  ndices  en la shard key (sucursal_id)
db.createCollection("sucursal")
db.sucursal.createIndex({ sucursal_id: 1 })

db.createCollection("empleado")
db.empleado.createIndex({ sucursal_id: 1 })

db.createCollection("reserva")
db.reserva.createIndex({ sucursal_id: 1 })

db.createCollection("resenia")
```

```
db.resenia.createIndex({ sucursal_id: 1 })

db.createCollection("pago")
db.pago.createIndex({ sucursal_id: 1 })

db.createCollection("cliente")
db.cliente.createIndex({ sucursal_id: 1 })

db.createCollection("destino")
db.destino.createIndex({ sucursal_id: 1 })

db.createCollection("promocion")
db.promocion.createIndex({ sucursal_id: 1 })

db.createCollection("paquete")
db.paquete.createIndex({ sucursal_id: 1 })

db.createCollection("logTransaccion")
db.logTransaccion.createIndex({ sucursal_id: 1 })

#Habilitamos el sharding en las colecciones usando como shard key sucursal_id:
sh.shardCollection("agenciaViajes.sucursal", { sucursal_id: 1 })
sh.shardCollection("agenciaViajes.empleado", { sucursal_id: 1 })
sh.shardCollection("agenciaViajes.reserva", { sucursal_id: 1 })
sh.shardCollection("agenciaViajes.resenia", { sucursal_id: 1 })
sh.shardCollection("agenciaViajes.pago", { sucursal_id: 1 })
sh.shardCollection("agenciaViajes.cliente", { sucursal_id: 1 })
sh.shardCollection("agenciaViajes.destino", { sucursal_id: 1 })
sh.shardCollection("agenciaViajes.promocion", { sucursal_id: 1 })
sh.shardCollection("agenciaViajes.paquete", { sucursal_id: 1 })
sh.shardCollection("agenciaViajes.logTransaccion", { sucursal_id: 1 })

#Definimos zonas de shards para organizar por sucursales
sh.addShardTag("shard_repl", "zona1")
sh.addShardTag("shard2_repl", "zona2")
sh.addShardTag("shard3_repl", "zona3")
sh.addShardTag("shard4_repl", "zona4")

#Definimos rangos para que cada sucursal est  en una zona o en un shard diferente
sh.updateZoneKeyRange("agenciaViajes.sucursal", { sucursal_id: 1 }, { sucursal_id:
    2 }, "zona1")
sh.updateZoneKeyRange("agenciaViajes.empleado", { sucursal_id: 1 }, { sucursal_id:
    2 }, "zona1")
sh.updateZoneKeyRange("agenciaViajes.reserva", { sucursal_id: 1 }, { sucursal_id:
    2 }, "zona1")
sh.updateZoneKeyRange("agenciaViajes.resenia", { sucursal_id: 1 }, { sucursal_id:
    2 }, "zona1")
sh.updateZoneKeyRange("agenciaViajes.pago", { sucursal_id: 1 }, { sucursal_id: 2
    }, "zona1")
sh.updateZoneKeyRange("agenciaViajes.cliente", { sucursal_id: 1 }, { sucursal_id:
    2 }, "zona1")
sh.updateZoneKeyRange("agenciaViajes.destino", { sucursal_id: 1 }, { sucursal_id:
    2 }, "zona1")
sh.updateZoneKeyRange("agenciaViajes.promocion", { sucursal_id: 1 }, { sucursal_id
    : 2 }, "zona1")
sh.updateZoneKeyRange("agenciaViajes.paquete", { sucursal_id: 1 }, { sucursal_id:
    2 }, "zona1")
sh.updateZoneKeyRange("agenciaViajes.logTransaccion", { sucursal_id: 1 }, {
    sucursal_id: 2 }, "zona1")
```

```
67
68 sh.updateZoneKeyRange("agenciaViajes.sucursal", { sucursal_id: 2 }, { sucursal_id:
       3 }, "zona2")
69 sh.updateZoneKeyRange("agenciaViajes.empleado", { sucursal_id: 2 }, { sucursal_id:
       3 }, "zona2")
70 sh.updateZoneKeyRange("agenciaViajes.reserva", { sucursal_id: 2 }, { sucursal_id:
      3 }, "zona2")
71 sh.updateZoneKeyRange("agenciaViajes.resenia", { sucursal_id: 2 }, { sucursal_id:
      3 }, "zona2")
72 sh.updateZoneKeyRange("agenciaViajes.pago", { sucursal_id: 2 }, { sucursal_id: 3
      }, "zona2")
73 sh.updateZoneKeyRange("agenciaViajes.cliente", { sucursal_id: 2 }, { sucursal_id:
      3 }, "zona2")
74 sh.updateZoneKeyRange("agenciaViajes.destino", { sucursal_id: 2 }, { sucursal_id:
      3 }, "zona2")
75 sh.updateZoneKeyRange("agenciaViajes.promocion", { sucursal_id: 2 }, { sucursal_id
      : 3 }, "zona2")
76 sh.updateZoneKeyRange("agenciaViajes.paquete", { sucursal_id: 2 }, { sucursal_id:
      3 }, "zona2")
77 sh.updateZoneKeyRange("agenciaViajes.logTransaccion", { sucursal_id: 2 }, {
      sucursal_id: 3 }, "zona2")
78
79 sh.updateZoneKeyRange("agenciaViajes.sucursal", { sucursal_id: 3 }, { sucursal_id:
       4 }, "zona3")
80 sh.updateZoneKeyRange("agenciaViajes.empleado", { sucursal_id: 3 }, { sucursal_id:
       4 }, "zona3")
81 sh.updateZoneKeyRange("agenciaViajes.reserva", { sucursal_id: 3 }, { sucursal_id:
      4 }, "zona3")
82 sh.updateZoneKeyRange("agenciaViajes.resenia", { sucursal_id: 3 }, { sucursal_id:
      4 }, "zona3")
83 sh.updateZoneKeyRange("agenciaViajes.pago", { sucursal_id: 3 }, { sucursal_id: 4
      }, "zona3")
84 sh.updateZoneKeyRange("agenciaViajes.cliente", { sucursal_id: 3 }, { sucursal_id:
      4 }, "zona3")
85 sh.updateZoneKeyRange("agenciaViajes.destino", { sucursal_id: 3 }, { sucursal_id:
      4 }, "zona3")
86 sh.updateZoneKeyRange("agenciaViajes.promocion", { sucursal_id: 3 }, { sucursal_id
      : 4 }, "zona3")
87 sh.updateZoneKeyRange("agenciaViajes.paquete", { sucursal_id: 3 }, { sucursal_id:
      4 }, "zona3")
88 sh.updateZoneKeyRange("agenciaViajes.logTransaccion", { sucursal_id: 3 }, {
      sucursal_id: 4 }, "zona3")
89
90 sh.updateZoneKeyRange("agenciaViajes.sucursal", { sucursal_id: 4 }, { sucursal_id:
       5 }, "zona4")
91 sh.updateZoneKeyRange("agenciaViajes.empleado", { sucursal_id: 4 }, { sucursal_id:
       5 }, "zona4")
92 sh.updateZoneKeyRange("agenciaViajes.reserva", { sucursal_id: 4 }, { sucursal_id:
      5 }, "zona4")
93 sh.updateZoneKeyRange("agenciaViajes.resenia", { sucursal_id: 4 }, { sucursal_id:
      5 }, "zona4")
94 sh.updateZoneKeyRange("agenciaViajes.pago", { sucursal_id: 4 }, { sucursal_id: 5
      }, "zona4")
95 sh.updateZoneKeyRange("agenciaViajes.cliente", { sucursal_id: 4 }, { sucursal_id:
      5 }, "zona4")
96 sh.updateZoneKeyRange("agenciaViajes.destino", { sucursal_id: 4 }, { sucursal_id:
      5 }, "zona4")
97 sh.updateZoneKeyRange("agenciaViajes.promocion", { sucursal_id: 4 }, { sucursal_id
      : 5 }, "zona4")
```

```
98 sh.updateZoneKeyRange("agenciaViajes.paquete", { sucursal_id: 4 }, { sucursal_id:
       5 }, "zona4")
99 sh.updateZoneKeyRange("agenciaViajes.logTransaccion", { sucursal_id: 4 }, {
       sucursal_id: 5 }, "zona4")
100
101 #Para revisar el estado del sharding usamos el sisguiente comando: sh.status()
```

```
[direct: mongos] test> use agenciaViajes
switched to db agenciaViajes
[direct: mongos] agenciaViajes> sh.enableSharding("agenciaNegocios")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742939078, i: 7 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1742939078, i: 4 })
}
```

```
[direct: mongos] agenciaViajes> db.createCollection("sucursal")
... db.sucursal.createIndex({ sucursal_id: 1 })
...
... db.createCollection("empleado")
... db.empleado.createIndex({ sucursal_id: 1 })
...
... db.createCollection("reserva")
... db.reserva.createIndex({ sucursal_id: 1 })
...
... db.createCollection("resenia")
... db.resenia.createIndex({ sucursal_id: 1 })
...
... db.createCollection("pago")
... db.pago.createIndex({ sucursal_id: 1 })
...
... db.createCollection("cliente")
... db.cliente.createIndex({ sucursal_id: 1 })
...
... db.createCollection("destino")
... db.destino.createIndex({ sucursal_id: 1 })
...
... db.createCollection("promocion")
... db.promocion.createIndex({ sucursal_id: 1 })
...
... db.createCollection("paquete")
... db.paquete.createIndex({ sucursal_id: 1 })
...
... db.createCollection("logTransaccion")
... db.logTransaccion.createIndex({ sucursal_id: 1 })
sucursal_id_1
```

```
[direct: mongos] agenciaViajes> sh.shardCollection("agenciaViajes.sucursal", { sucursal_id: 1 })
... sh.shardCollection("agenciaViajes.empleado", { sucursal_id: 1 })
... sh.shardCollection("agenciaViajes.reserva", { sucursal_id: 1 })
... sh.shardCollection("agenciaViajes.resenia", { sucursal_id: 1 })
... sh.shardCollection("agenciaViajes.pago", { sucursal_id: 1 })
... sh.shardCollection("agenciaViajes.cliente", { sucursal_id: 1 })
... sh.shardCollection("agenciaViajes.destino", { sucursal_id: 1 })
... sh.shardCollection("agenciaViajes.promocion", { sucursal_id: 1 })
... sh.shardCollection("agenciaViajes.paquete", { sucursal_id: 1 })
... sh.shardCollection("agenciaViajes.logTransaccion", { sucursal_id: 1 })
{
  collectionsharded: 'agenciaViajes.logTransaccion',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742939231, i: 233 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1742939231, i: 233 })
}
[direct: mongos] agenciaViajes> sh.addShardTag("shard_repl", "zona1")
... sh.addShardTag("shard2_repl", "zona2")
... sh.addShardTag("shard3_repl", "zona3")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742939257, i: 3 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1742939257, i: 3 })
}
[direct: mongos] agenciaViajes> sh.updateZoneKeyRange("agenciaViajes.sucursal", { sucursal_id: 1 },
{ sucursal_id: 2 }, "zona1")
... sh.updateZoneKeyRange("agenciaViajes.empleado", { sucursal_id: 1 }, { sucursal_id: 2 }, "zona1")
... sh.updateZoneKeyRange("agenciaViajes.reserva", { sucursal_id: 1 }, { sucursal_id: 2 }, "zona1")
... sh.updateZoneKeyRange("agenciaViajes.resenia", { sucursal_id: 1 }, { sucursal_id: 2 }, "zona1")
... sh.updateZoneKeyRange("agenciaViajes.pago", { sucursal_id: 1 }, { sucursal_id: 2 }, "zona1")
... sh.updateZoneKeyRange("agenciaViajes.cliente", { sucursal_id: 1 }, { sucursal_id: 2 }, "zona1")
... sh.updateZoneKeyRange("agenciaViajes.destino", { sucursal_id: 1 }, { sucursal_id: 2 }, "zona1")
... sh.updateZoneKeyRange("agenciaViajes.promocion", { sucursal_id: 1 }, { sucursal_id: 2 }, "zona1"
)
... sh.updateZoneKeyRange("agenciaViajes.paquete", { sucursal_id: 1 }, { sucursal_id: 2 }, "zona1")
... sh.updateZoneKeyRange("agenciaViajes.logTransaccion", { sucursal_id: 1 }, { sucursal_id: 2 }, "z
ona1")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742939269, i: 11 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
```

```
[direct: mongos] agenciaViajes> sh.updateZoneKeyRange("agenciaViajes.sucursal", { sucursal_id: 2 },
{ sucursal_id: 3 }, "zona2")
... sh.updateZoneKeyRange("agenciaViajes.empleado", { sucursal_id: 2 }, { sucursal_id: 3 }, "zona2")
... sh.updateZoneKeyRange("agenciaViajes.reserva", { sucursal_id: 2 }, { sucursal_id: 3 }, "zona2")
... sh.updateZoneKeyRange("agenciaViajes.resenia", { sucursal_id: 2 }, { sucursal_id: 3 }, "zona2")
... sh.updateZoneKeyRange("agenciaViajes.pago", { sucursal_id: 2 }, { sucursal_id: 3 }, "zona2")
... sh.updateZoneKeyRange("agenciaViajes.cliente", { sucursal_id: 2 }, { sucursal_id: 3 }, "zona2")
... sh.updateZoneKeyRange("agenciaViajes.destino", { sucursal_id: 2 }, { sucursal_id: 3 }, "zona2")
... sh.updateZoneKeyRange("agenciaViajes.promocion", { sucursal_id: 2 }, { sucursal_id: 3 }, "zona2"
)
... sh.updateZoneKeyRange("agenciaViajes.paquete", { sucursal_id: 2 }, { sucursal_id: 3 }, "zona2")
... sh.updateZoneKeyRange("agenciaViajes.logTransaccion", { sucursal_id: 2 }, { sucursal_id: 3 }, "z
ona2")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742939283, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1742939283, i: 1 })
}
[direct: mongos] agenciaViajes> sh.updateZoneKeyRange("agenciaViajes.sucursal", { sucursal_id: 3 },
{ sucursal_id: 4 }, "zona3")
... sh.updateZoneKeyRange("agenciaViajes.empleado", { sucursal_id: 3 }, { sucursal_id: 4 }, "zona3")
... sh.updateZoneKeyRange("agenciaViajes.reserva", { sucursal_id: 3 }, { sucursal_id: 4 }, "zona3")
... sh.updateZoneKeyRange("agenciaViajes.resenia", { sucursal_id: 3 }, { sucursal_id: 4 }, "zona3")
... sh.updateZoneKeyRange("agenciaViajes.pago", { sucursal_id: 3 }, { sucursal_id: 4 }, "zona3")
... sh.updateZoneKeyRange("agenciaViajes.cliente", { sucursal_id: 3 }, { sucursal_id: 4 }, "zona3")
... sh.updateZoneKeyRange("agenciaViajes.destino", { sucursal_id: 3 }, { sucursal_id: 4 }, "zona3")
... sh.updateZoneKeyRange("agenciaViajes.promocion", { sucursal_id: 3 }, { sucursal_id: 4 }, "zona3"
)
... sh.updateZoneKeyRange("agenciaViajes.paquete", { sucursal_id: 3 }, { sucursal_id: 4 }, "zona3")
... sh.updateZoneKeyRange("agenciaViajes.logTransaccion", { sucursal_id: 3 }, { sucursal_id: 4 }, "z
ona3")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1742939300, i: 42 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1742939300, i: 42 })
}
```

```
}
[direct: mongos] agenciaViajes> sh.status()
shardingVersion
{ _id: 1, clusterId: ObjectId('67e3146a31c7a8100daa44fa') }
---
shards
[
  {
    _id: 'shard2_repl',
    host: 'shard2_repl/192.168.1.11:28081,192.168.1.11:28082,192.168.1.11:28083',
    state: 1,
    topologyTime: Timestamp({ t: 1742938995, i: 9 }),
    replSetConfigVersion: Long('-1'),
    tags: [ 'zona2' ]
  },
  {
    _id: 'shard3_repl',
    host: 'shard3_repl/192.168.1.12:28081,192.168.1.12:28082,192.168.1.12:28083',
    state: 1,
    topologyTime: Timestamp({ t: 1742938995, i: 32 }),
    replSetConfigVersion: Long('-1'),
    tags: [ 'zona3' ]
  },
  {
    _id: 'shard_repl',
    host: 'shard_repl/192.168.1.10:28081,192.168.1.10:28082,192.168.1.10:28083',
    state: 1,
    topologyTime: Timestamp({ t: 1742938991, i: 10 }),
    replSetConfigVersion: Long('-1'),
    tags: [ 'zona1' ]
  }
]
---
active mongoses
[ { '8.0.6': 1 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently enabled': 'yes',
  'Currently running': 'yes',
  'Failed balancer rounds in last 5 attempts': 0,
  'Migration Results for the last 24 hours': { '6': 'Success' }
}
```

8.5 Prueba

Listing 20: Paso 1

```
#Insertamos documentos en coleccion sucursal:
db.sucursal.insertMany([{ sucursal:id: 1, nombre: "Sucursal Pachuca", direccion: "
    Pachuca Centro", telefono: "7712563451"}, { sucursal_id: 2, nombre: "Sucursal
    CDMX", direccion: "CDMX Centro", telefono: "5546738912"}, {sucursal_id: 3,
    nombre: "Sucursal Cancun", direccion: "Cancun Centro", telefono: "5546721811"
    }])
```

```
[direct: mongos] agenciaViajes> db.sucursal.insertMany([{ sucursal_id: 1, nombre: "Sucursal Pahuca",
 direccion: "Pachuca Centro", telefono: "7713676541"}, {sucursal_id: 2, nombre: "Sucursal CDMX", dir
eccion: "CDMX Centro", telefono: "5546788731"}, { sucursal_id: 3, nombre: "Sucursal Cancun", direcci
on: "Cancun Centro", telefono: "5546782312"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67e32a98114abc2d6f819c9f'),
    '1': ObjectId('67e32a98114abc2d6f819ca0'),
    '2': ObjectId('67e32a98114abc2d6f819ca1')
  }
}
```

Listing 21: Paso 2

```
#Buscamos una sucursal y usamos explain() para ver mas informacion:
db.sucursal.find({ sucursal_id: 1}).explain()
```

```
[direct: mongos] agenciaViajes> db.sucursal.find({ sucursal_id: 1}).explain()
{
  queryPlanner: {
    winningPlan: {
      stage: 'SINGLE_SHARD',
      shards: [
        {
          explainVersion: '1',
          shardName: 'shard_repl',
          connectionString: 'shard_repl/192.168.1.10:28081,192.168.1.10:28082,192.168.1.10:28083',
          serverInfo: {
            host: 'MacBook-Air-de-Israel.local',
            port: 28082,
            version: '8.0.6',
            gitVersion: '80f21521ad4a3dfd5613f5d649d7058c6d46277f'
          },
          namespace: 'agenciaViajes.sucursal',
          parsedQuery: { sucursal_id: { '$eq': 1 } },
          indexFilterSet: false,
          queryHash: 'C383C6F4',
          planCacheShapeHash: 'C383C6F4',
          planCacheKey: 'B4ED006F',
          optimizationTimeMillis: 0,
          maxIndexedOrSolutionsReached: false,
          maxIndexedAndSolutionsReached: false,
          maxScansToExplodeReached: false,
          prunedSimilarIndexes: false,
          winningPlan: {
            isCached: false,
            stage: 'FETCH',
            inputStage: {
              stage: 'IXSCAN',
              keyPattern: { sucursal_id: 1 },
              indexName: 'sucursal_id_1',
              isMultiKey: false,
              multiKeyPaths: { sucursal_id: [] },
              isUnique: false,
              isSparse: false,
              isPartial: false,
              indexVersion: 2,
              direction: 'forward',
              indexBounds: { sucursal_id: [ '[1, 1]' ] }
            }
          },
          rejectedPlans: []
        }
      ]
    }
  },
  queryShapeHash: '6798E3FA018C7EF52D8A0A47826D186541F8F146CDE6C23A13FB24592AD124AD',
  serverInfo: {
    host: 'MacBook-Air-de-Israel.local',
    port: 27017,
    version: '8.0.6',
    gitVersion: '80f21521ad4a3dfd5613f5d649d7058c6d46277f'
  },
```

8.6 Apagar los Servidores

Para apagar los servidores debemos hacerlo en el siguiente orden: Mongos, Shards y al final el config server. Recordemos que tanto los shards como el config server están en Replica Set, por lo que primero debemos terminar los procesos secundarios y al final el primario.

Listing 22: Paso 1

```
#Nos conectamos al Sharded Cluster con Mongos:
mongosh --host 192.168.1.10 --port 27017
```

Listing 23: Paso 2

```
#Finalizamos el servidor:
use admin
db.shutdownServer()
```

Listing 24: Paso 3

```
#Nos conectamos a un servidor Replica Set de la primer PC Shard:
mongosh --host 192.168.1.10 --port 28081
```

Listing 25: Paso 4

```
#Revisamos que servidor es el primario y cu les son los secundarios:
rs.status()

#Nos conectamos a los secundarios primero y los finalizamos:
use admin
db.shutdownServer()

#Por  ltimo , nos conectamos al servidor primario y lo finalizamos:
use admin
db.shutdownServer()

#Hacemos lo mismo con el resto de las PC Shard
```

Listing 26: Paso 5

```
#Nos conectamos a un servidor Replica Set de la PC que tiene el Config Server:
mongosh --host 192.168.1.10 --port 28041
```

Listing 27: Paso 6

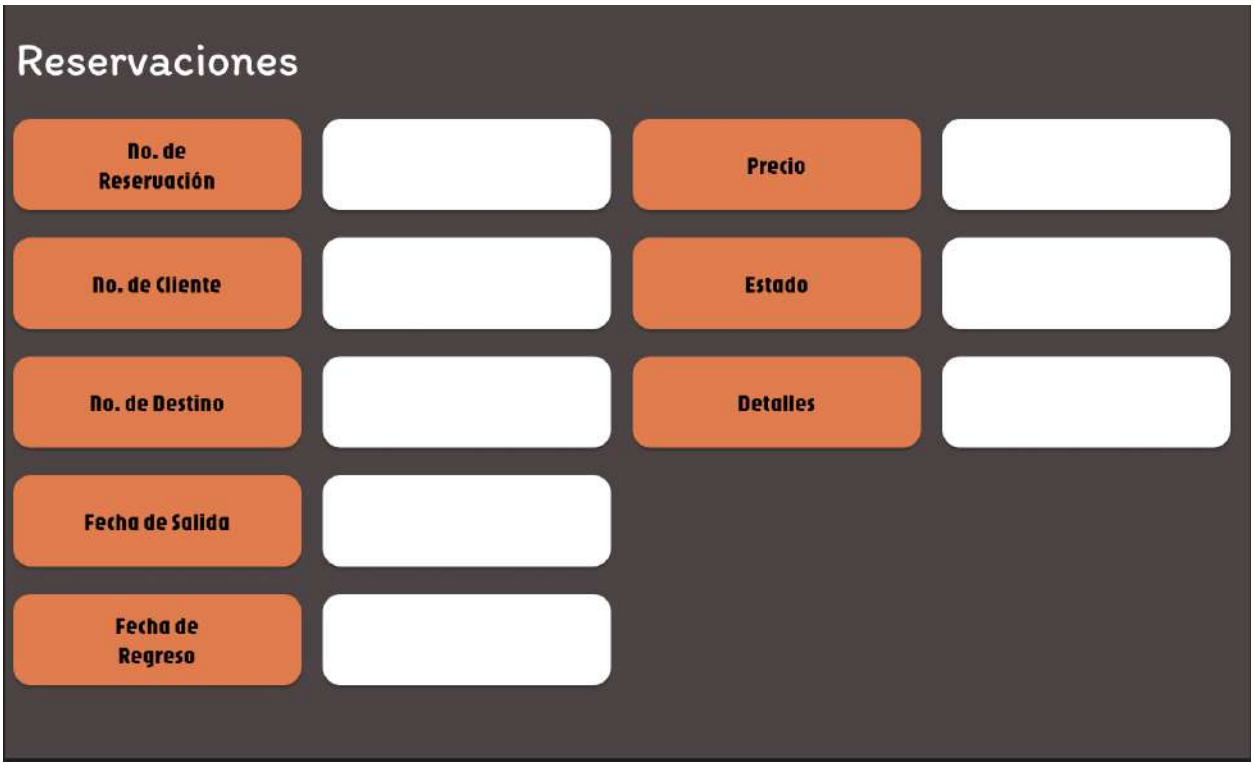
```
#Revisamos que servidor es el primario y cu les son los secundarios:
rs.status()

#Nos conectamos a los secundarios primero y los finalizamos:
use admin
db.shutdownServer()

#Por  ltimo , nos conectamos al servidor primario y lo finalizamos:
use admin
db.shutdownServer()
```

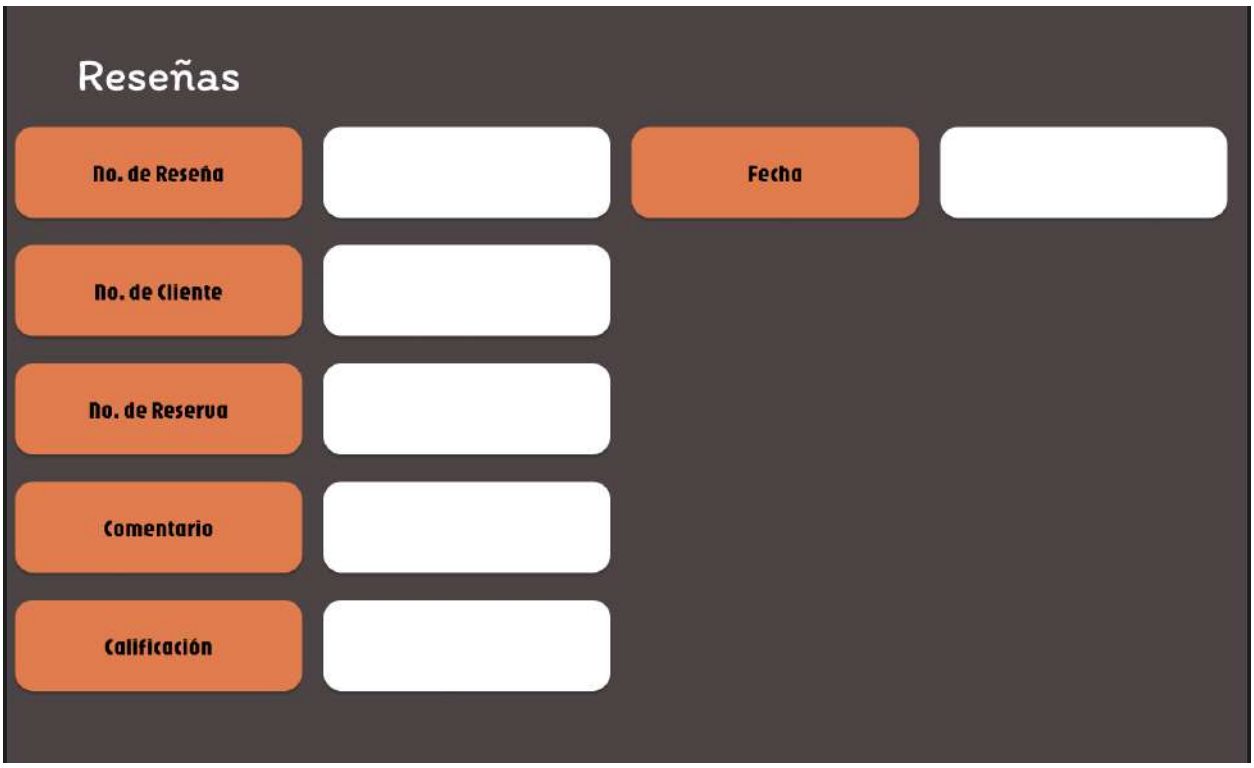
8.7 Red Local

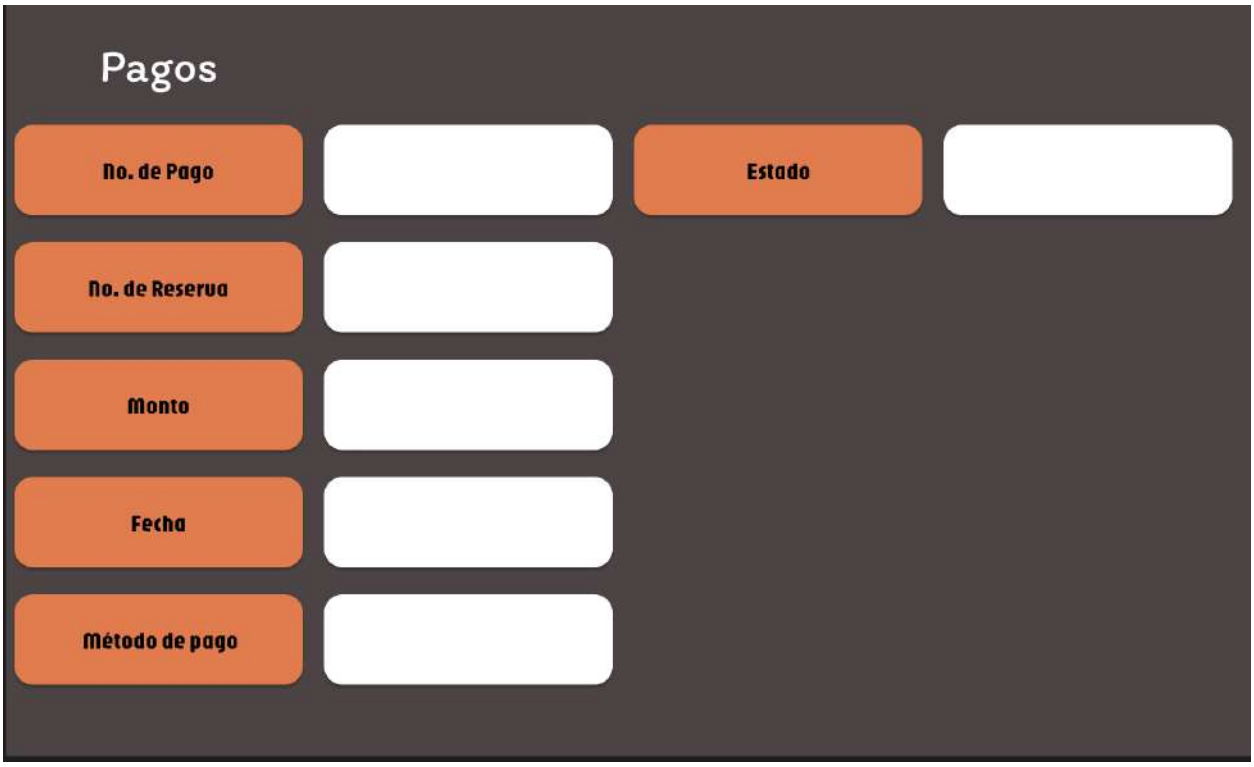
Para tener comunicación entre las laptops donde alojamos los Shards Servers tenemos que estar conectados a una subred local, ya sea Wi-Fi o Ethernet. Decidimos que en este caso sería por Ethernet, así que utilizamos un switch y conectamos las laptops a dicho switch.

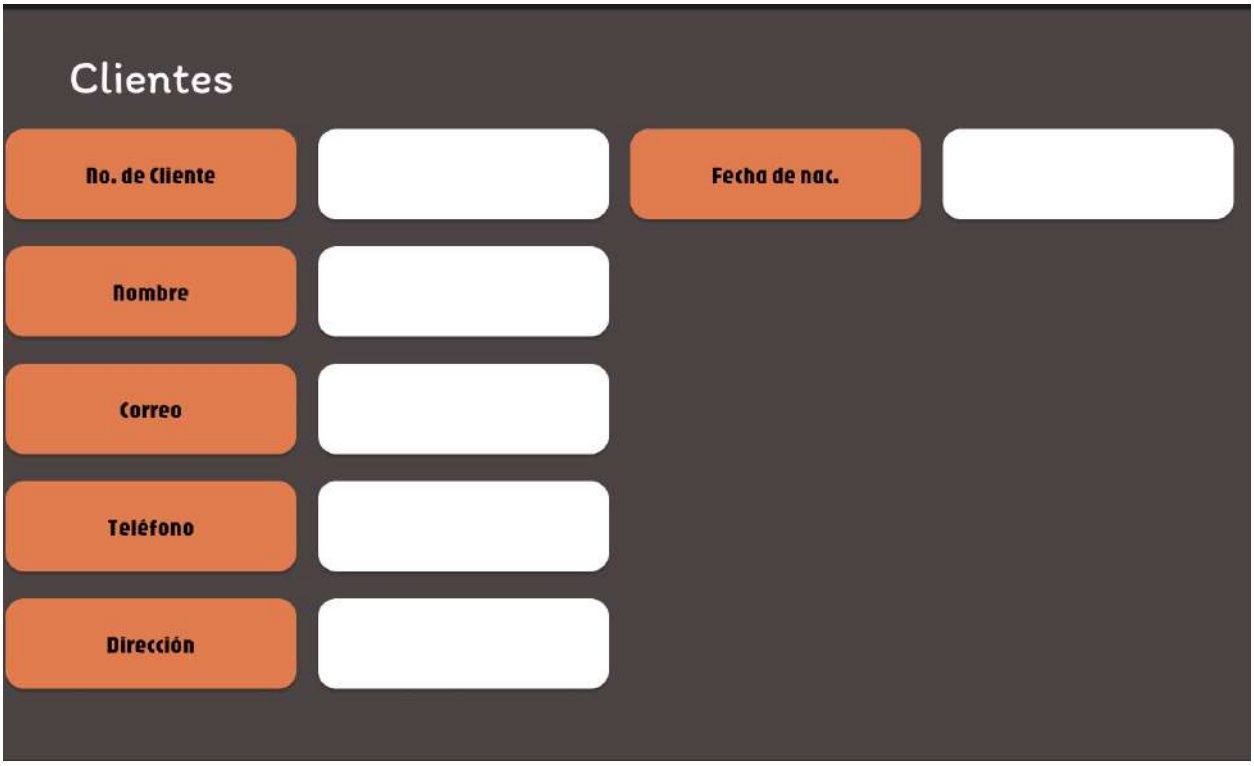
A cada laptop le asignamos una IPv4 estática dentro de nuestra subred. Las IPs usadas fueron: 192.168.1.10, 192.168.1.11 y 192.168.1.12. Probamos la conexión entre nuestras laptops haciendo ping a cada una y obtuvimos un resultado exitoso.

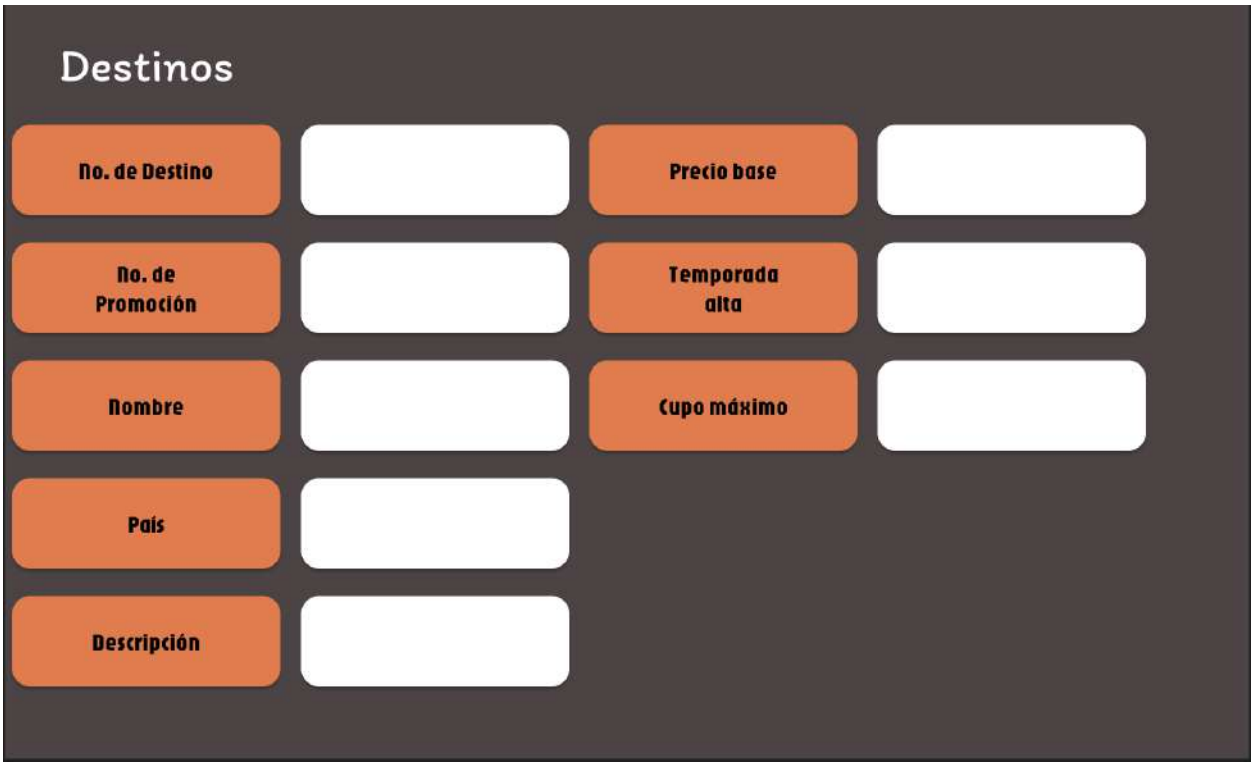
4.8 Diseño de vistas en Figma

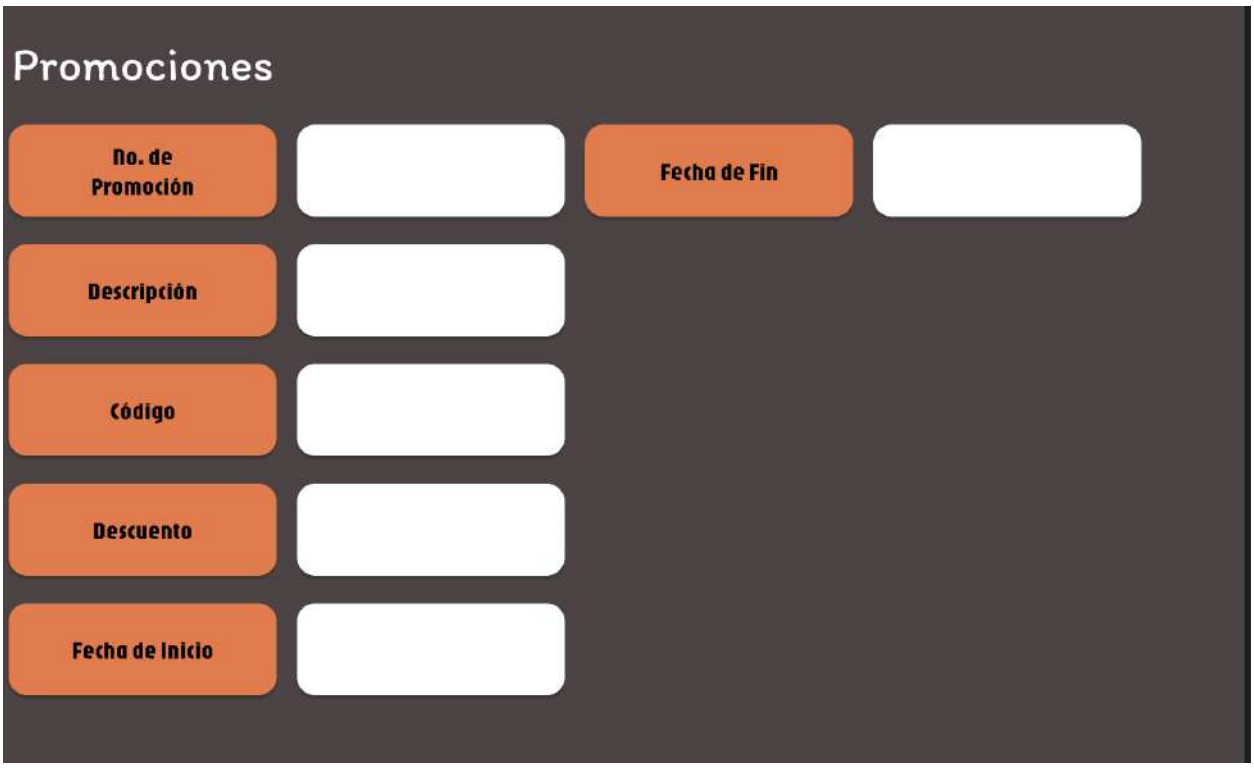


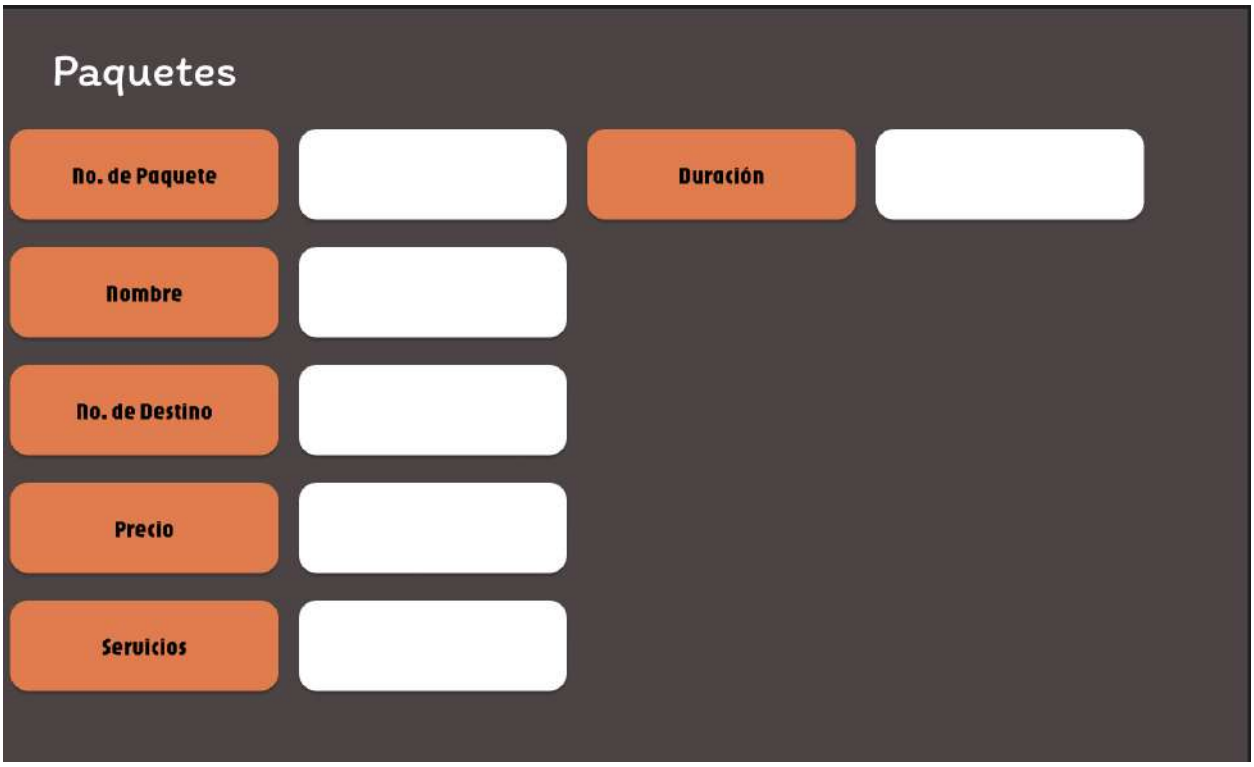

Reseñas
No. de Reseña
Fecha
No. de Cliente
No. de Reserva
Comentario
Calificación

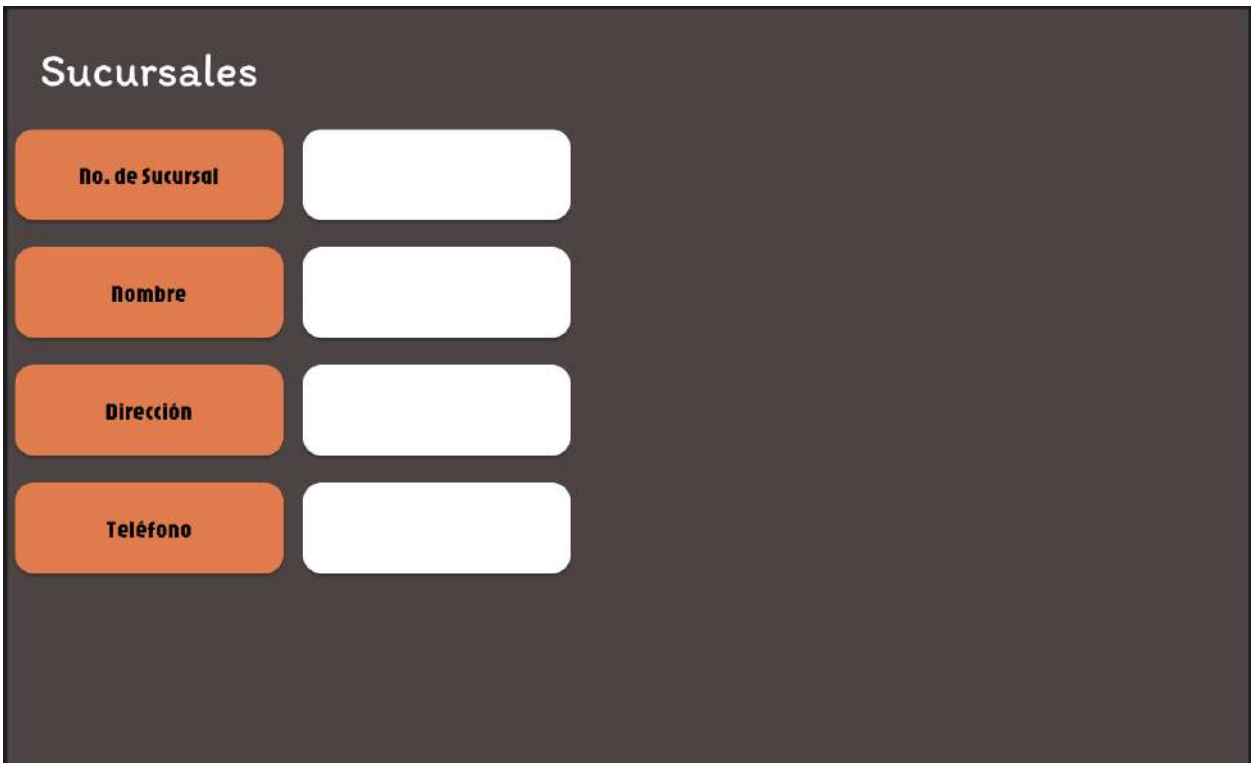

Pagos
No. de Pago
Estado
No. de Reserva
Monto
Fecha
Método de pago

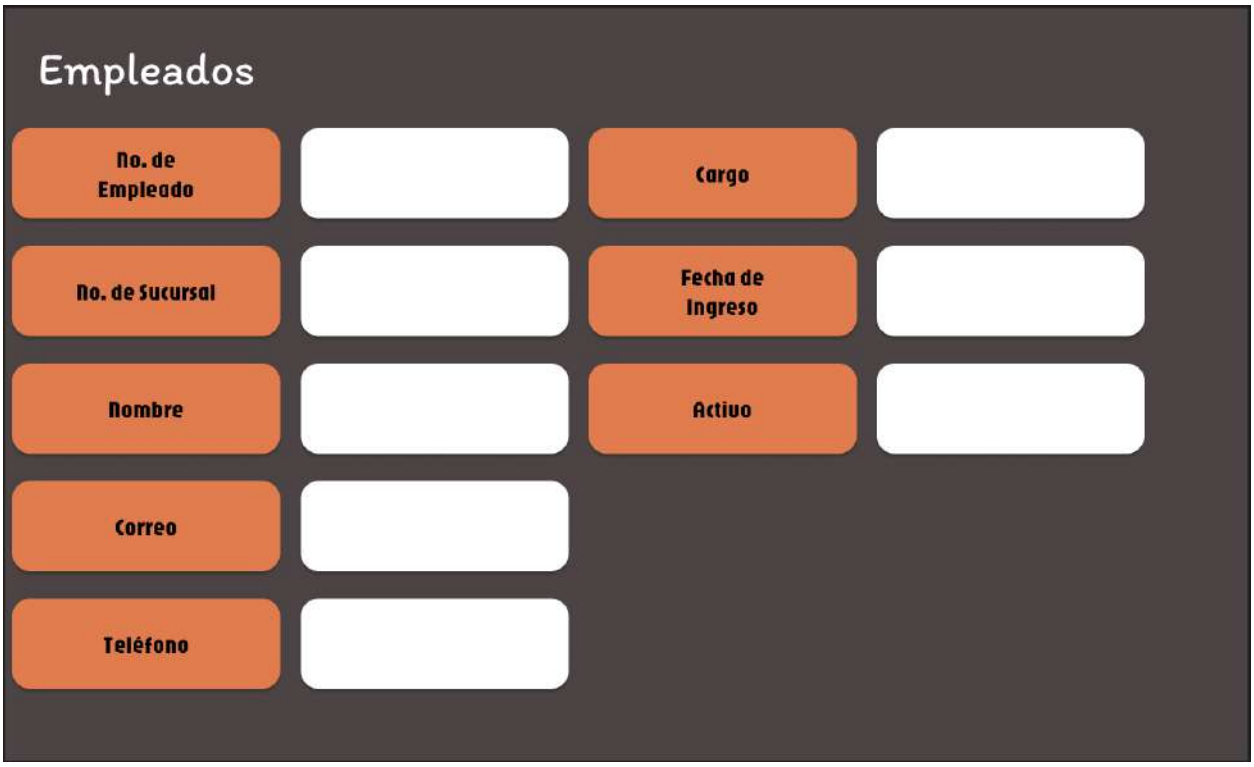

Clientes
No. de Cliente
Fecha de nac.
Nombre
Correo
Teléfono
Dirección


Destinos
No. de Destino
Precio base
No. de Promoción
Temporada alta
Nombre
Cupo máximo
País
Descripción


Promociones
No. de Promoción
Fecha de Fin
Descripción
Código
Descuento
Fecha de Inicio


Paquetes
No. de Paquete
Duración
Nombre
No. de Destino
Precio
Servicios


Sucursales
No. de Sucursal
Nombre
Dirección
Teléfono


Empleados
No. de Empleado
No. de Sucursal
Nombre
Correo
Teléfono
Cargo
Fecha de Ingreso
Activo

9. Conclusiones

Este proyecto ha sido muy enriquecedor, ya que nos ha permitido aprender cómo funciona y cómo usar MongoDB, desde operaciones CRUD hasta usar el Sharding (Fragmentación). Hemos tenido que aprender desde cero un nuevo motor de bases de datos, y considero que es muy buena opción para proyectos futuros. MongoDB nos brinda una muy buena flexibilidad, ya que las colecciones no exigen (a menos que lo desees) que sigas una estructura rígida en tus documentos. Además de que, si planeas y diseñas bien tu esquema de base de datos desde un inicio, MongoDB te puede dar muy buena escalabilidad y disponibilidad de la información.

Como vimos en este proyecto, si en algún momento tu negocio necesita escalar por ejemplo a tener más sucursales, puedes ir agregando más servidores por cada sucursal, o asignar a un servidor un rango de sucursales. Esto hará que las operaciones de escritura y lectura sean más eficientes, ya que tus consultas no tendrán que ser buscadas en todo el sistema, sino sólo en el servidor que corresponda a tu sucursal.

Ha sido un proyecto con complicaciones, con mucho por investigar, y con mucho por aprender, pero ha sido gratificante ver el resultado final.

Israel Campos Vázquez

Referencias Bibliográficas

References

[1] Team, M. D. (s. f.). Sharding. MongoDB Manual v8.0 - MongoDB Docs. https://www.mongodb.com/docs/manual/sharding/std-label-sharding-strategy

[2] Neerajg. (s. f.). mongodb-tutorial/mongodb-sharding-AWS-git-video-version.txt at main · neerajg5/mongodb-tutorial. GitHub. https://github.com/neerajg5/mongodb-tutorial/blob/main/mongodb-sharding-AWS-git-video-version.txt