



Sistemas Avanzados de Bases de Datos

9768

Julio 14, 2023

Génesis Calapaqui
Diego Portilla
Camila Rivera

Proyecto Segundo Parcial

Descripción del proyecto

El aplicativo de gestión de solicitud de libros de la biblioteca de la Universidad de las Fuerzas Armadas ESPE es una plataforma diseñada para facilitar el proceso de préstamo de libros a los estudiantes. El sistema utiliza la base de datos de los estudiantes de la universidad para autenticar a los usuarios y proporcionarles acceso a la aplicación. Los administradores tienen la capacidad de administrar la información de los libros disponibles, mientras que los estudiantes pueden visualizarlos y solicitar préstamos. Los administradores revisarán las solicitudes y las aprobarán o rechazarán según la disponibilidad del libro y el historial de préstamos del estudiante.

Alcance

Este proyecto tiene como objetivo, desarrollar una plataforma de gestión de solicitud de libros para la biblioteca de la Universidad de las Fuerzas Armadas ESPE, que permita a los estudiantes visualizar y solicitar préstamos de libros, y a los administradores gestionar la información de los libros, revisar y aprobar/rechazar las solicitudes, con un enfoque en la autenticación de usuarios, seguridad de datos y utilización de tecnologías como sharding en MongoDB, PostgreSQL para la replicación, y React para el frontend y el API backend.

Requerimientos

Funcionales

- Autenticación: Los usuarios deben poder registrar sus credenciales de la base de datos de estudiantes de la Universidad.
- Gestión de libros: Los administradores deben poder agregar, editar y eliminar información sobre los libros disponibles en la biblioteca.
- Visualización de libros: Los estudiantes deben poder ver la lista de libros disponibles junto con su información relevante, como título, autor, categoría, etc.
- Solicitud de préstamo: Los estudiantes deben poder solicitar el préstamo de libros específicos a través de la aplicación.
- Aprobación/rechazo de solicitudes: Los administradores deben poder revisar las solicitudes de préstamo y aprobarlas o rechazarlas según la disponibilidad del libro y el historial de préstamos del estudiante.
- Gestión de préstamos: Los administradores deben poder llevar un registro de los préstamos realizados, incluyendo la fecha de inicio, fecha de vencimiento y estado del préstamo.
- Notificaciones: Los usuarios (estudiantes y administradores) deben recibir notificaciones sobre el estado de sus solicitudes de préstamo y recordatorios de devolución de libros.

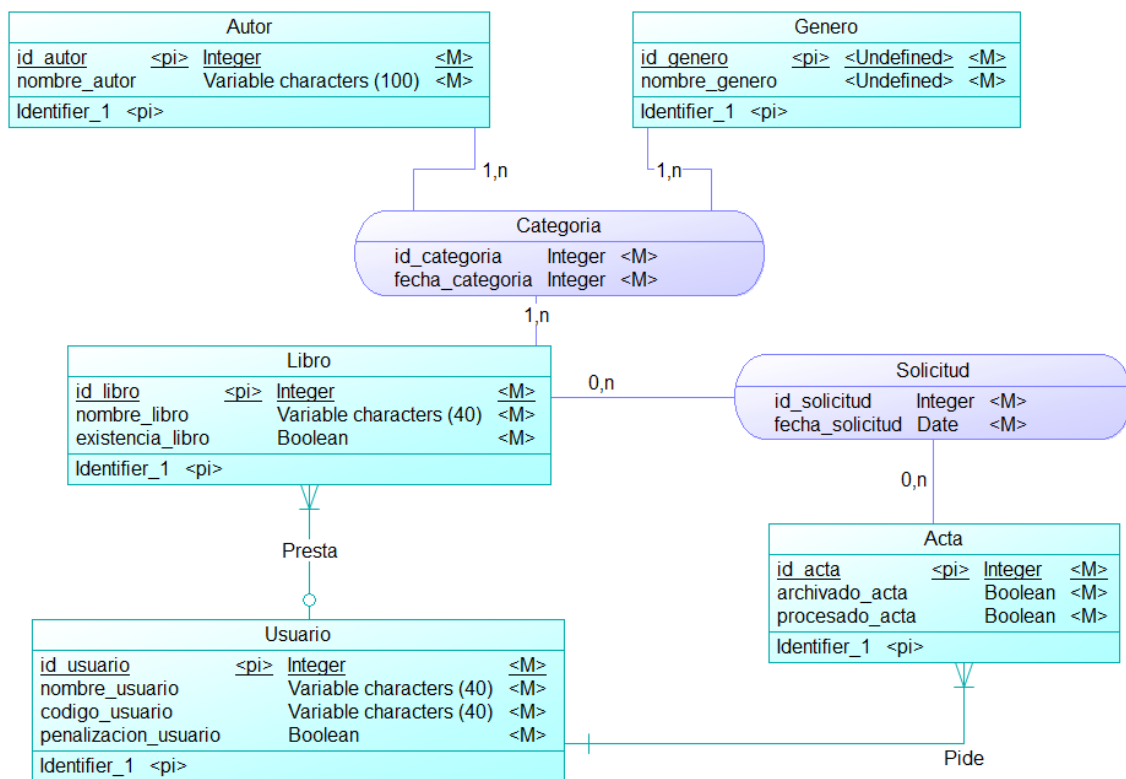
No funcionales

- Escalabilidad: El sistema debe estar preparado para manejar un alto volumen de usuarios y libros, utilizando la técnica de sharding en MongoDB para distribuir la carga de la base de datos.
- Fiabilidad: El sistema debe ser confiable y asegurar la integridad de los datos, utilizando replicas bidireccionales de PostgreSQL para garantizar la disponibilidad y redundancia de la base de datos.
- Interfaz de usuario intuitiva: La interfaz de usuario debe ser fácil de usar y navegar, con una experiencia de usuario fluida tanto en el frontend desarrollado en React como en el backend basado en React Express.

- Rendimiento: El sistema debe ser eficiente en términos de rendimiento, minimizando los tiempos de respuesta y maximizando la velocidad de carga de los datos.
- Seguridad: El sistema debe contar con medidas de seguridad para proteger la información confidencial de los usuarios, como las credenciales de inicio de sesión y los datos personales.
- Mantenibilidad: El código fuente y la infraestructura del sistema deben ser fáciles de mantener y actualizar, utilizando contenedores Docker para facilitar la implementación y el despliegue en diferentes entornos.

Modelado

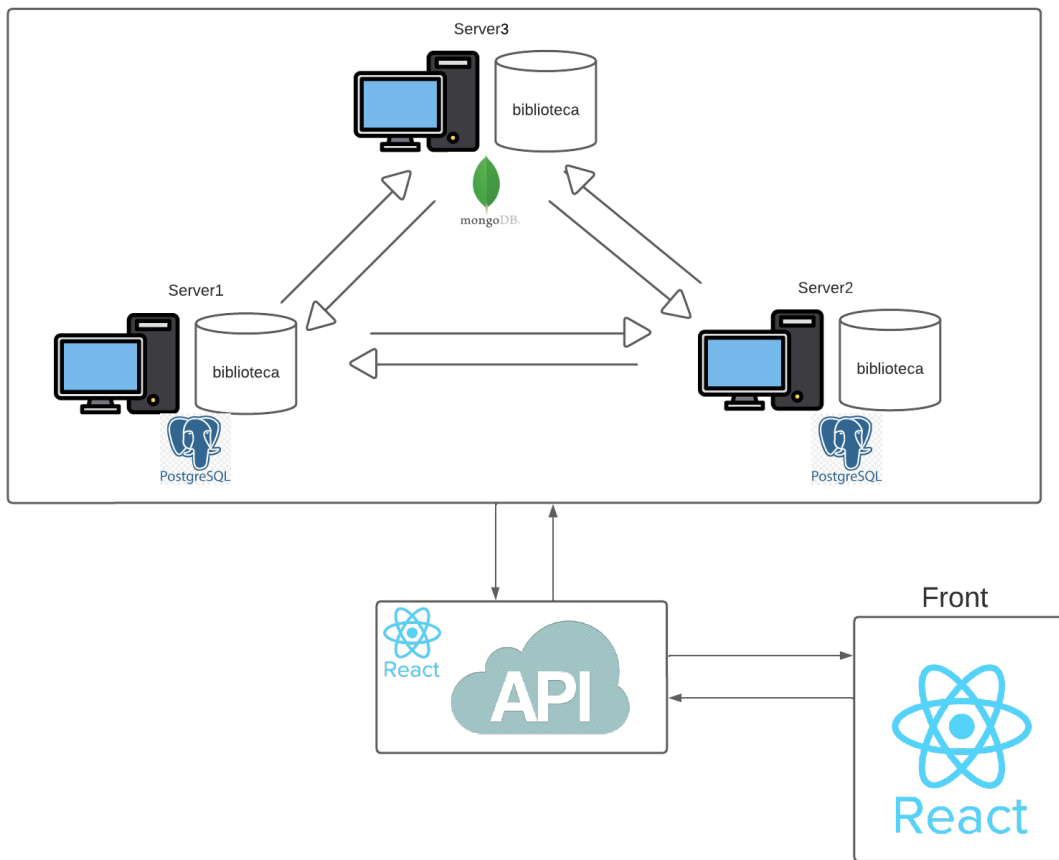
Diagrama entidad-relación



Diagrama

de

infraestructura



Desarrollo

Docker

Levantamiento de los contenedores

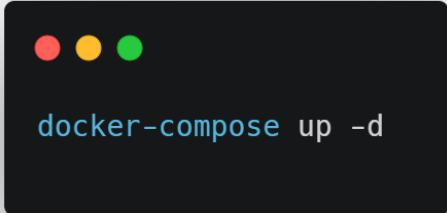
```
version: '3'

services:
  server1:
    image: postgres
    container_name: server1
    environment:
      - POSTGRES_PASSWORD=admin
      - '5435:5432'

  server2:
    image: postgres
    container_name: server2
    environment:
      - POSTGRES_PASSWORD=admin
      - '5433:5432'

  server3:
    image: postgres
    container_name: server3
    environment:
      - POSTGRES_PASSWORD=admin
      - '5434:5432'
```

En este paso, usamos un docker-compose.yaml para construir nuestros contenedores en base a la imagen en postgres ya existente, así, para tenerlo, corremos:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text 'docker-compose up -d' is displayed in a light blue monospace font.

```
docker-compose up -d
```

PostgreSQL

Creación de la Base de Datos Distribuida con nodos maestros.

Para crear una Base de Datos Distribuida con nodos maestros y replicación bidireccional se debe seguir los siguientes pasos:

1. Creamos una red docker:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text 'docker create network red1' is displayed in a light blue monospace font.

```
docker create network red1
```

2. Creamos los contenedores Docker asignándole una red, nombre, contraseña, puertos y la imagen que se va a utilizar, en este caso Postgres, de la siguiente manera:



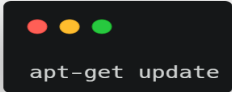
```
docker run --name server1 --net red1 -e POSTGRES_PASSWORD=***** -p 5435:5432 -d postgres
```

- Entramos al bash del servidor que vamos a configurar e instalamos los paquetes que se van a necesitar para la configuración del contenedor:



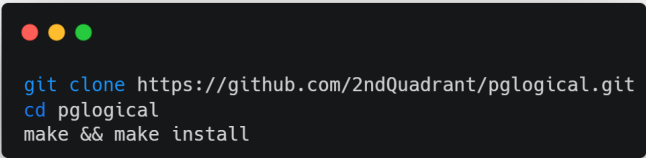
```
apt-get install -y build-essential postgresql-server-dev-all git postgresql-server-dev-15 libgssapi3-heimdal libgssapi-krb5-2 libkrb5-dev libgssglue-dev libselinux1-dev zlib1g-dev liblz4-dev libxslt1-dev libxml2-dev libpam0g-dev libzstd-dev nano
```

- Actualizamos la lista de paquetes.




```
apt-get update
```

- Clonamos un repositorio de github que nos ayudará a instalar la extensión de replicación lógica para Postgres llamada *pglogical*.



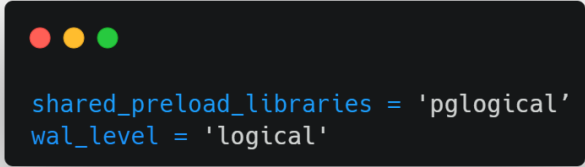
```
git clone https://github.com/2ndQuadrant/pglogical.git  
cd pglogical  
make && make install
```

- Ingresamos al archivo de configuración *postgresql.conf*



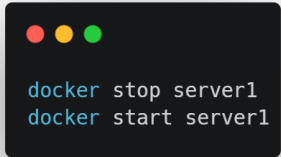
```
nano /var/lib/postgresql/data/postgresql.conf
```

7. Modificamos la configuración del archivo:



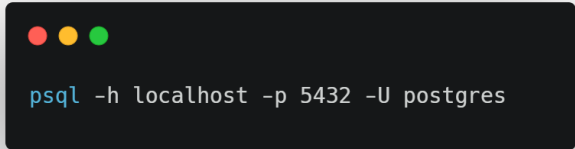
```
shared_preload_libraries = 'pglogical'  
wal_level = 'logical'
```

8. Guardamos los cambios y reiniciamos el contenedor con los siguientes comandos:



```
docker stop server1  
docker start server1
```

9. Ingresamos nuevamente al bash del contenedor y nos conectamos a las bases de datos usando:




```
psql -h localhost -p 5432 -U postgres
```

10. Creamos la Base de Datos que vamos a replicar con sus respectivas tablas.


11. Ingresamos a la Base de Datos que se va a replicar con el comando '`\c`'.

12. Creamos la extensión a *pglogical*:




```
CREATE EXTENSION pglogical;
```

13. Creamos un slot que permita la replicación de los datos:



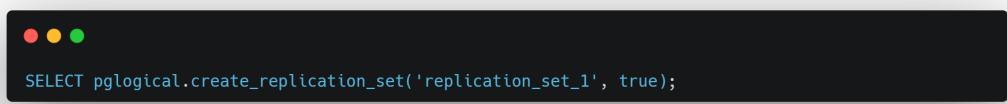
```
SELECT pg_create_logical_replication_slot('slot_1', 'pgoutput');
```

14. Creamos un nodo, en la parte de host pondremos la dirección ip del contenedor::



```
SELECT pglogical.create_node(node_name := 'node1', dsn := 'host=172.21.0.2 port=5432 dbname=biblioteca user=postgres password=admin');
```

15. Creamos el set de replicación:



```
SELECT pglogical.create_replication_set('replication_set_1', true);
```

16. Añadimos las tablas que queremos replicar a nuestro set de replicación.


```

SELECT pglogical.replication_set_add_table('replication_set_1', 'public.acta', true);
SELECT pglogical.replication_set_add_table('replication_set_1', 'public.autor', true);
.
.
SELECT pglogical.replication_set_add_table('replication_set_1', 'public.categoria', true);

```

17. Creamos una suscripción de nodo de origen al de destino, especificando la dirección IP, la base de datos, usuario y contraseña.

```

SELECT pglogical.create_subscription( subscription_name := 'subscription_1', provider_dsn :=
'host=172.21.0.3 port=5432 dbname=biblioteca user=postgres password=admin');

```

18. Añadimos nuestro set de réplica a la suscripción.









```

SELECT pglogical.alter_subscription_add_replication_set('subscription_1', 'replication_set_1');

```

Se realizan los mismos pasos con los demás contenedores, solo se debe tener en cuenta en especificar correctamente los puertos al realizar la suscripción y los nombres que se vayan a asignar.

19. Como resultados se tendrán los dos servidores levantados.

<input type="checkbox"/>	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	 server2 036810af80b5	postgres	Running	5433:5432	3 seconds ago	  
<input type="checkbox"/>	 server1 f23ca51943ab	postgres	Running	5435:5432	0 seconds ago	  

MongoDB

Para realizar la el sharding y replicación con MongoDB se creó un docker- compose el cual contiene lo siguiente:

1. Router (mongos):

El enrutador, también conocido como mongos, actúa como un punto de entrada para las aplicaciones y clientes que desean acceder a los datos almacenados en la base de datos MongoDB sharded. Su función principal es enrutar las operaciones y consultas a los servidores de shard correspondientes. El contenedor "router1" representa un enrutador configurado para escuchar en el puerto 27117.

```
router1:
  container_name: router1
  image: mongo
  volumes:
    - ./router-init.js:/scripts/router-init.js
  networks:
    - mongo-network-sharded
  ports:
    - 27117:27017
  entrypoint: [ "/usr/bin/mongos", "--port", "27017", "--configdb", "rs-config-server/configsvr1:27017", "--bind_ip_all" ]
```

2. Servidores de configuración (config servers):

Los servidores de configuración almacenan los metadatos y la configuración necesaria para el funcionamiento del clúster sharded. Almacenan información sobre los rangos de shard y las ubicaciones de los datos en el clúster. En el archivo de configuración, el contenedor "configsvr1" representa un servidor de configuración configurado para escuchar en el puerto 27118.

```
configsvr1:
  container_name: configsvr1
  image: mongo
  volumes:
    - ./configserver-init.js:/scripts/configserver-init.js
  networks:
    - mongo-network-sharded
  ports:
    - 27118:27017
  entrypoint: [ "/usr/bin/mongod", "--port", "27017", "--configsvr", "--replSet", "rs-config-server", "--bind_ip_all" ]
  links:
    - mongo-shard1a
    - mongo-shard2a
```

3. Servidores de shard:

Los servidores de shard son los nodos de almacenamiento reales en los que se almacenan los datos de la base de datos sharded. Cada servidor de shard es responsable de almacenar un subconjunto de datos. En el archivo de configuración, se crearon los contenedores "mongo-shard1a", "mongo-shard1b", "mongo-shard2a" y "mongo-shard2b" los cuales representan los servidores de shard configurados para escuchar en diferentes puertos.

```
mongo-shard1a:
  container_name: mongo-shard1a
  image: mongo
  volumes:
    - ./shard1-init.js:/scripts/shard1-init.js
  networks:
    - mongo-network-sharded
  ports:
    - 27119:27017
  entrypoint: [ "/usr/bin/mongod", "--port", "27017", "--shardsvr", "--bind_ip_all", "--replSet", "iabds shard1" ]
```

Para realizar el sharding y replicación de este proyecto se deberá clonar el archivo docker-compose.yml y ejecutar el siguiente comando:

```
docker-compose up -d
```

React Express

Front-End

Para la interfaz de usuario, hemos usado la librería de JavaScript React, en el entorno de NodeJS. Así, para crear nuestro proyecto ejecutamos:

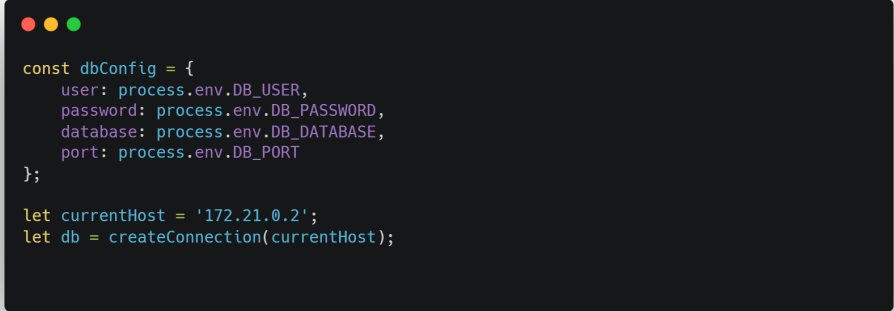
```
npx create-react-app Proyecto
```

Y proseguimos con el desarrollo del aplicativo. En caso de querer recrear el sistema diseñado en el informe, debemos acceder al [repositorio del proyecto](#), y una vez clonado, realizar.

```
npm install
npm start
```

Back-End

Para la creación del API se utilizó NodeJs y JavaScript, para ello primero se realizó la conexión con la base de datos de la siguiente manera:




```
const dbConfig = {  
  user: process.env.DB_USER,  
  password: process.env.DB_PASSWORD,  
  database: process.env.DB_DATABASE,  
  port: process.env.DB_PORT  
};  
  
let currentHost = '172.21.0.2';  
let db = createConnection(currentHost);
```

El host variará dependiendo del servidor que se encuentre disponible y de la dirección IP que se encuentre asignada a nuestros contenedores, las otras variables de la conexión se obtendrán del entorno que se especifica en el docker-compose.yml.

Posteriormente se deben realizar las funciones de los controladores para poder crear, obtener, editar o eliminar los datos desde nuestra API y por último se exportan los módulos.

Para utilizar el API del sistema de este informe se deberá clonar el repositorio del API y ejecutar los siguientes comandos:



```
npm install  
npm run dev
```

Una vez corriendo nuestra aplicativo, podemos consumir de la API tanto para lectura como con escritura con los endpoints definidos.

Anexos

- Repositorio del proyecto:
<https://github.com/Genesis-Calapaqui/BDDProyecto1/tree/main>
- Documentación de la API en
Swagger:<https://app.swaggerhub.com/apis-docs/CARIVERA14/Biblioteca/1.0.0>