



PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS

Sistema Distribuido de Préstamo de Libros

Primera Entrega

Proyecto de Introducción a Sistemas Distribuidos

Período Académico 2025-30

Integrantes:

Valeria Catalina Caycedo Ramírez - v_caycedo@javeriana.edu.co

Jorge Esteban Gomez Zuluaga - gomezjo2@javeriana.edu.co

Karen Daniela Medina Naranjo - kdaniela.medina@javeriana.edu.co

Jeisson Camilo Ruiz Cristancho - jeissonc_ruizc@javeriana.edu.co

10 de octubre de 2025

Resumen

Este documento presenta el diseño y planificación inicial del Sistema Distribuido de Préstamo de Libros para la Universidad Ada Lovelace. Se describe la arquitectura del sistema, los modelos de interacción, fallos y seguridad, así como el protocolo de pruebas y métricas de desempeño. El sistema utiliza patrones de comunicación síncronos y asíncronos mediante ZeroMQ, garantizando tolerancia a fallos y persistencia de datos en múltiples sedes.

I. INTRODUCCIÓN

El presente proyecto tiene como objetivo desarrollar un sistema distribuido para la gestión de préstamos de libros en la Universidad Ada Lovelace. El sistema operará en al menos dos sedes bibliotecarias, permitiendo a usuarios realizar operaciones de préstamo, renovación y devolución de libros. La arquitectura distribuida garantiza disponibilidad y tolerancia a fallos mediante réplicas de bases de datos y mecanismos de actualización asíncrona.

II. MODELOS DEL SISTEMA

II-A. Modelo Arquitectónico

El sistema adopta una arquitectura *en capas y puertos & adaptadores* (Hexagonal/Clean). Las capas principales son:

- **Presentación (Procesos Solicitantes, PS):** punto de entrada; orquesta el envío de solicitudes y la recepción de respuestas.
- **Aplicación (Gestor de Carga y Actores):** expresa los casos de uso: préstamo (síncrono), devolución y renovación (asíncronos).
- **Datos (Gestor de Almacenamiento y repositorios):** fachada transaccional sobre Postgres (primaria/réplica) con posible caché Redis.
- **Infraestructura:** transporte ZeroMQ (REQ/REP y PUB/SUB), HAProxy como VIP de BD, Prometheus/Grafana y mecanismos de *failover*.

Patrones clave: *Observer* (PUB/SUB), *Strategy* (actores por operación), *Facade* (GA), *Repository + Unit of Work* (persistencia), *Circuit Breaker + Retry* (resiliencia), *Factory* (creación de sockets/repos), *Idempotency Key* (`op_id`) para evitar duplicados.

II-B. Modelo de Interacción

- **Préstamo (síncrono):** PS → GC (REQ) → ActorPréstamo (RPC REQ/REP) → GA/BD. GC responde al PS con resultado final (aprobado/denegado).
- **Devolución y Renovación (asíncronos):** PS → GC (ACK inmediato); GC publica evento por tópico (`Devolucion/Renovacion`). Actor correspondiente consume y actualiza BD a través de GA.
- **Replicación:** Postgres primaria ↔ réplica por *streaming* (consistencia eventual).

II-C. Modelo de Fallos

- **Detección:** *health checks* (ping/SQL), exportadores de Prometheus y umbrales de error.
- **Aislamiento:** *Circuit Breaker* en GC/Actores abre el circuito ante fallos repetidos; *Retry* con *backoff* y *jitter*.
- **Conmutación (Failover):** promoción automática de réplica a primaria (p.ej., `pg_auto_failover/Patroni`); HAProxy actualiza el VIP. Transparente para GA mediante *DataSource*.
- **Idempotencia:** todas las peticiones incluyen `op_id` para garantizar efectos exactamente-una-vez en presencia de reintentos.

II-D. Modelo de Seguridad

- **Transporte:** ZeroMQ *CURVE* (opcional) entre procesos; TLS hacia BD (HAProxy/Postgres).
- **Integridad:** transacciones ACID; restricciones e índices para evitar inconsistencias (por ejemplo, préstamo activo único por usuario/libro).
- **Autorización:** reglas por rol (alumno/docente/staff) embebidas en los casos de uso.
- **Observabilidad y auditoría:** trazas con `op_id` de extremo a extremo y métricas por tipo de operación/estado.

III. DISEÑO DEL SISTEMA

A continuación se documenta el diseño mediante los diagramas solicitados e incluye explícitamente los componentes que *enmascaran fallas*: **Circuit Breaker**, **Health Checker**, **HAProxy (VIP BD)** y la **promoción de réplica**.

III-A. Diagrama de Despliegue

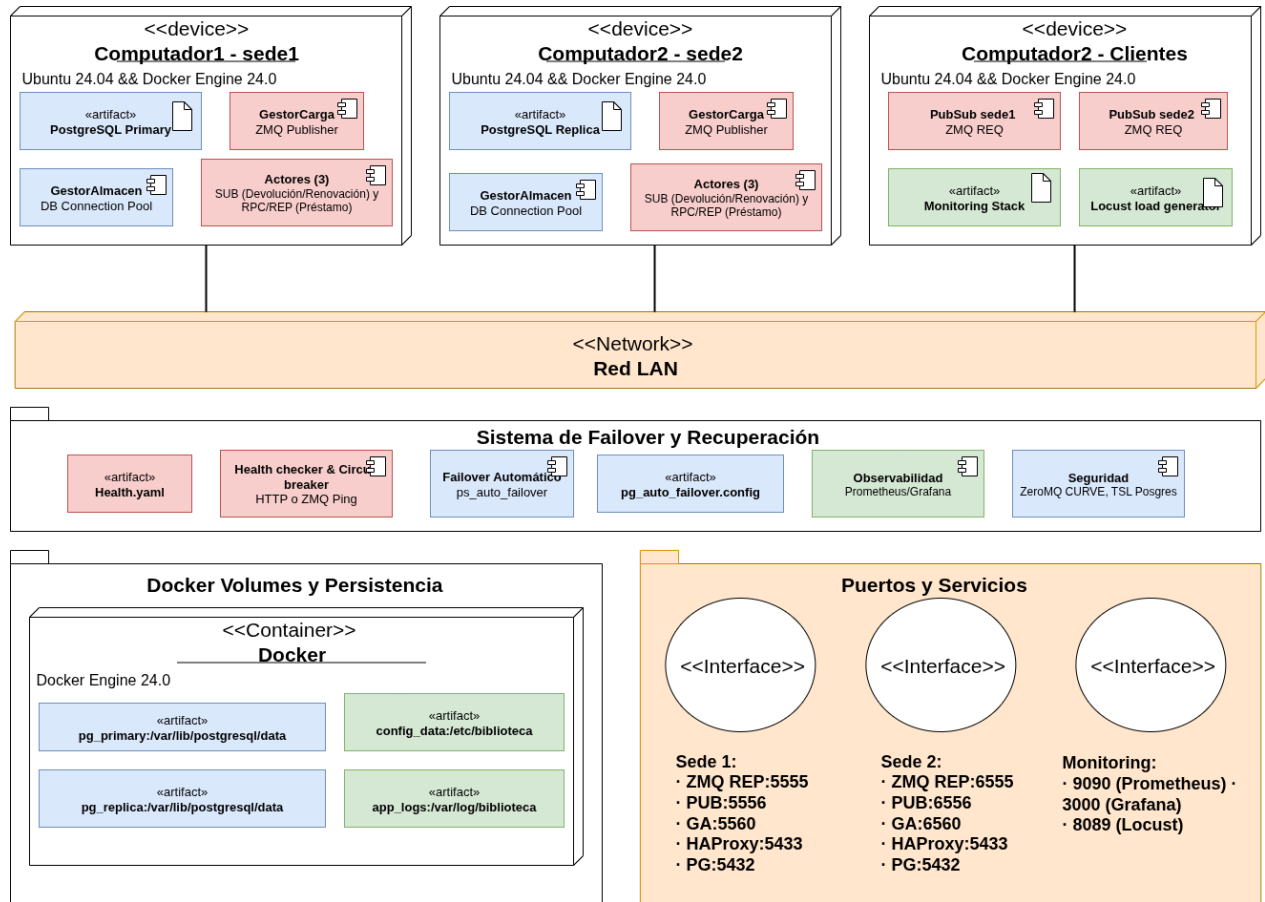


Figura 1. Diagrama de despliegue. Comunicación PS↔GC por ZeroMQ REQ/REP (puertos 5555/6555). GC→Actores por ZeroMQ PUB/SUB (5556/6556). Acceso a BD vía HAProxy (5433) con TLS. Replicación *streaming* entre Postgres primaria y réplica (5432).

Catálogo de componentes (qué es y para qué sirve)

Cuadro I
CATÁLOGO RESUMIDO DE COMPONENTES DEL DESPLIEGUE

| Componente | Rol en la arquitectura | Responsabilidad principal | Patrones |
|-------------------------------|---------------------------------|--|-----------------------------------|
| Proceso Solicitante (PS) | Punto de entrada de solicitudes | Construye DTOs y envía a GC | DTO, Idempotency Key |
| Gestor de Carga (GC) | Orquestador por sede | REQ/REP con PS; PUB eventos; RPC a ActorPréstamo | Facade, Observer, Circuit Breaker |
| Actores (3) | Workers por caso | Ejecutan devolución/renovación/préstamo | Strategy, Command |
| Gestor de Almacenamiento (GA) | Fachada transaccional | Reglas, idempotencia y persistencia vía repos | Facade, Repository, Unit of Work |
| HAProxy (VIP BD) | End-point de BD | Redirige a primaria vigente; TLS | Proxy |
| PostgreSQL Primary/Replica | Persistencia | ACID y replicación <i>streaming</i> | Leader/Replica |
| Health Checker | Observabilidad | Heartbeats y alertas | Observer, Scheduler |
| Prometheus/Grafana | Monitoreo | Recolección y visualización de métricas | — |
| Locust/JMeter | Pruebas de carga | Generación de usuarios virtuales | — |
| RedisCache (opcional) | Rendimiento | Cache de disponibilidad con TTL bajo | Cache-Aside |

Matriz de comunicaciones

Cuadro II
CANALES, PROTOCOLO Y GARANTÍAS

| Origen→Destino | Patrón | Protocolo | Puertos | Seguridad | Semántica/QoS |
|----------------------|-------------|-----------------|-----------------------|------------------|---|
| PS→GC | REQ/REP | ZMQ/TCP | 5555 (S1) / 6555 (S2) | CURVE (opcional) | ACK inmediato (dev/renov) o respuesta final (préstamo); op_id |
| GC→Actores Dev/Ren | PUB/SUB | ZMQ/TCP | 5556 (S1) / 6556 (S2) | CURVE (opcional) | Al menos una vez; idempotencia en GA |
| GCActorPréstamo | RPC (sync) | ZMQ REQ/REP | 5561 (S1) / 6561 (S2) | CURVE (opcional) | Resultado final préstamo |
| Actores/GA→BD | SQL/TX | TLS vía HAProxy | 5433 | TLS | ACID; failover transparente |
| PrimaryReplica | Replicación | Streaming | 5432 | (TLS opcional) | Consistencia eventual |
| Prometheus→Exporters | Pull | HTTP | 9090/varios | — | Observabilidad |

III-B. Diagrama de Componentes

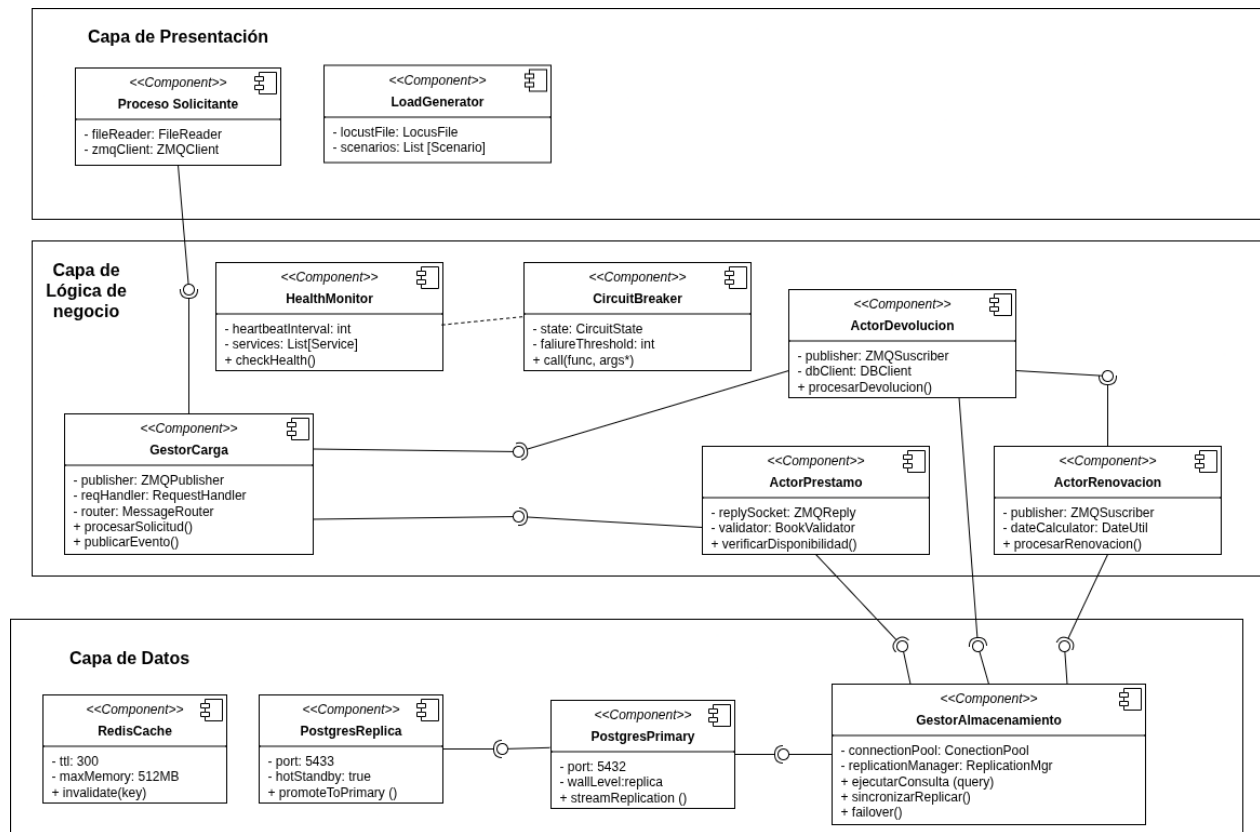


Figura 2. Diagrama de componentes. Se aprecia el *Circuit Breaker* protegiendo llamadas peligrosas, el *Health Monitor*, y la separación PS/GC/Actores/GA.

Sedes (servidor)

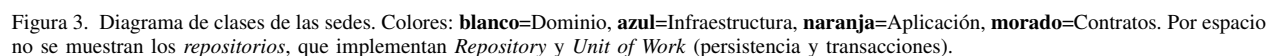


Figura 4. Diagrama de clases del cliente (PS). Colores: **azul**=Infraestructura (ZMQ, codec, claves, métricas), **naranja**=Aplicación (controlador, reintentos, ruteo/failover), **rosado**=Presentación (arranque/entrada), **verde**=Contratos (DTOs y enums compartidos).

III-D. Diagramas de Secuencia

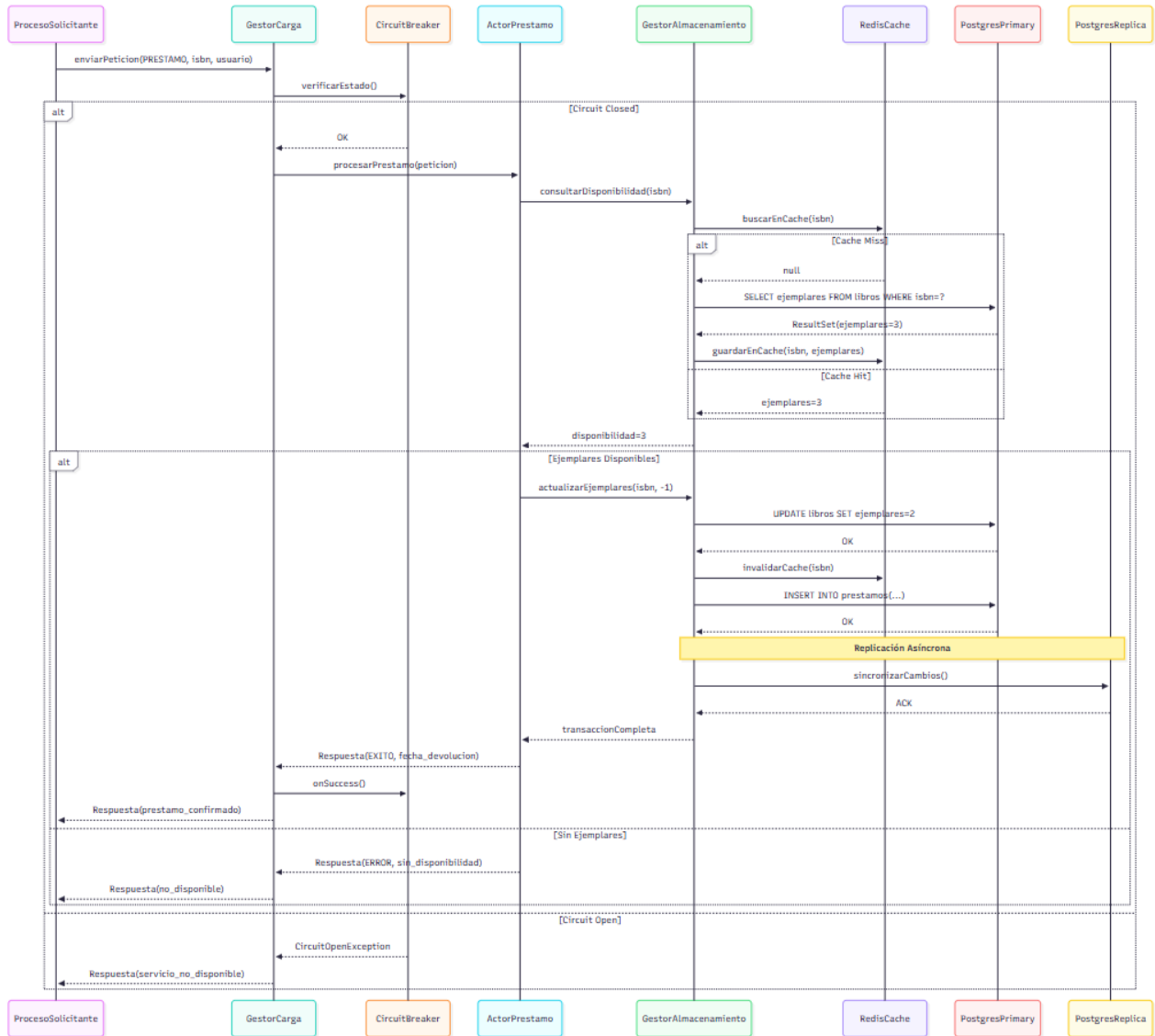


Figura 5. Secuencia de **Préstamo** (síncrono): respuesta final tras validar y escribir en BD.

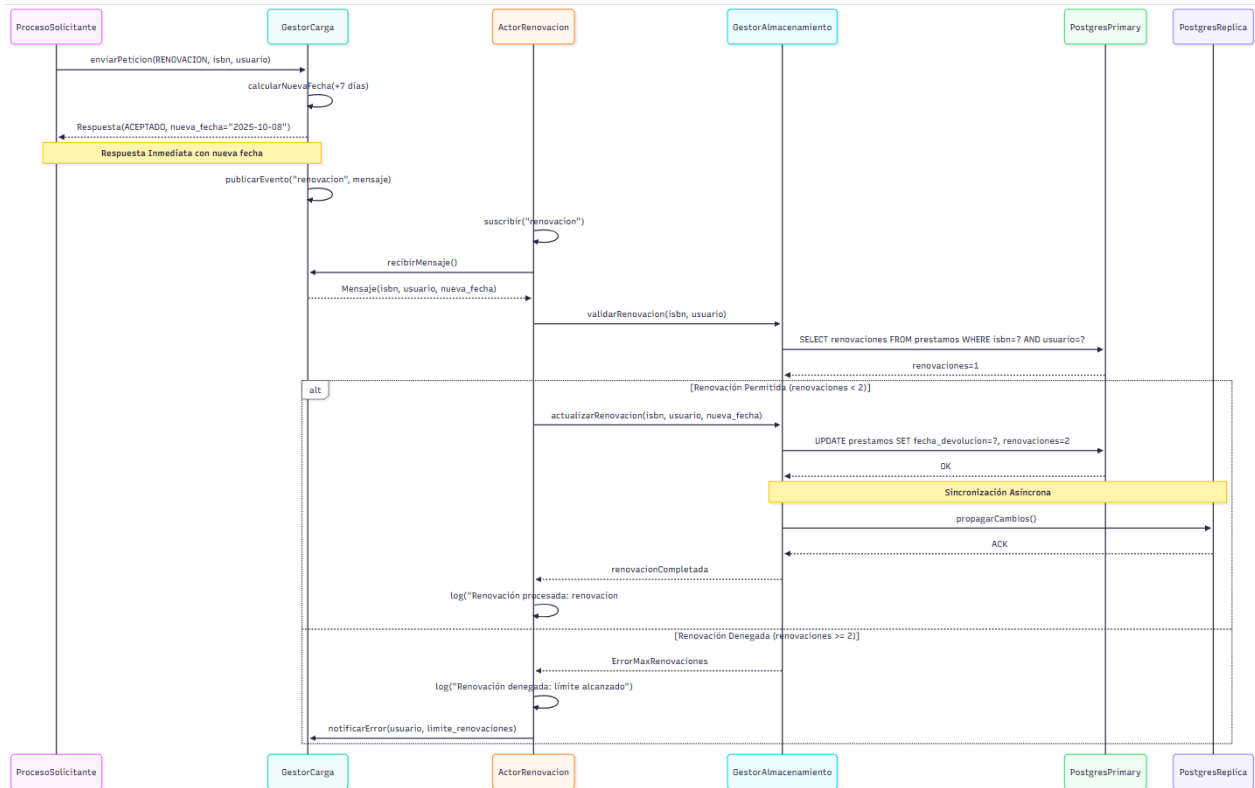


Figura 6. Secuencia de **Renovación** (asíncrona): ACK inmediato + actualización por actor.

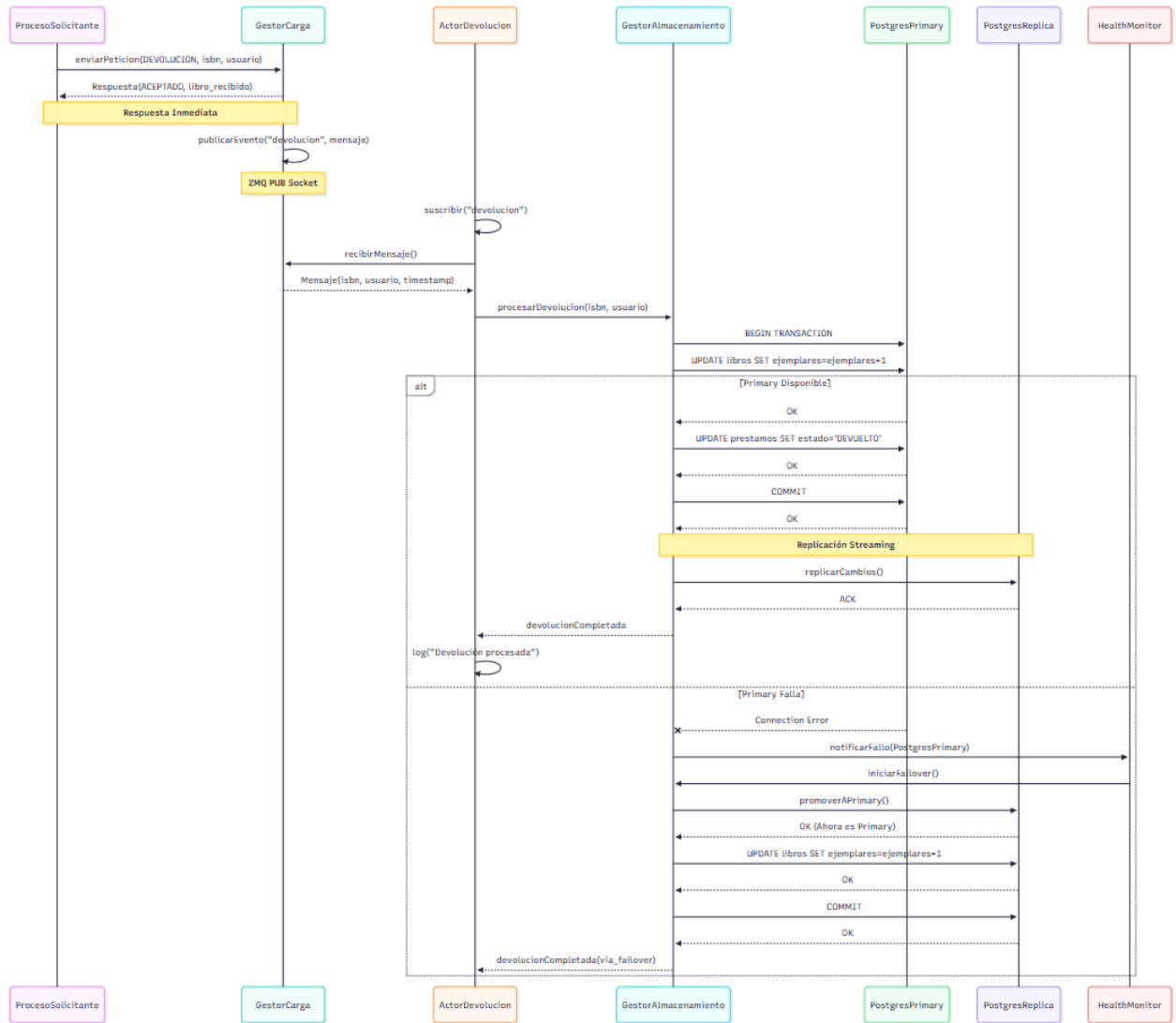


Figura 7. Secuencia de **Devolución** (asíncrona): ACK inmediato + actualización por actor.

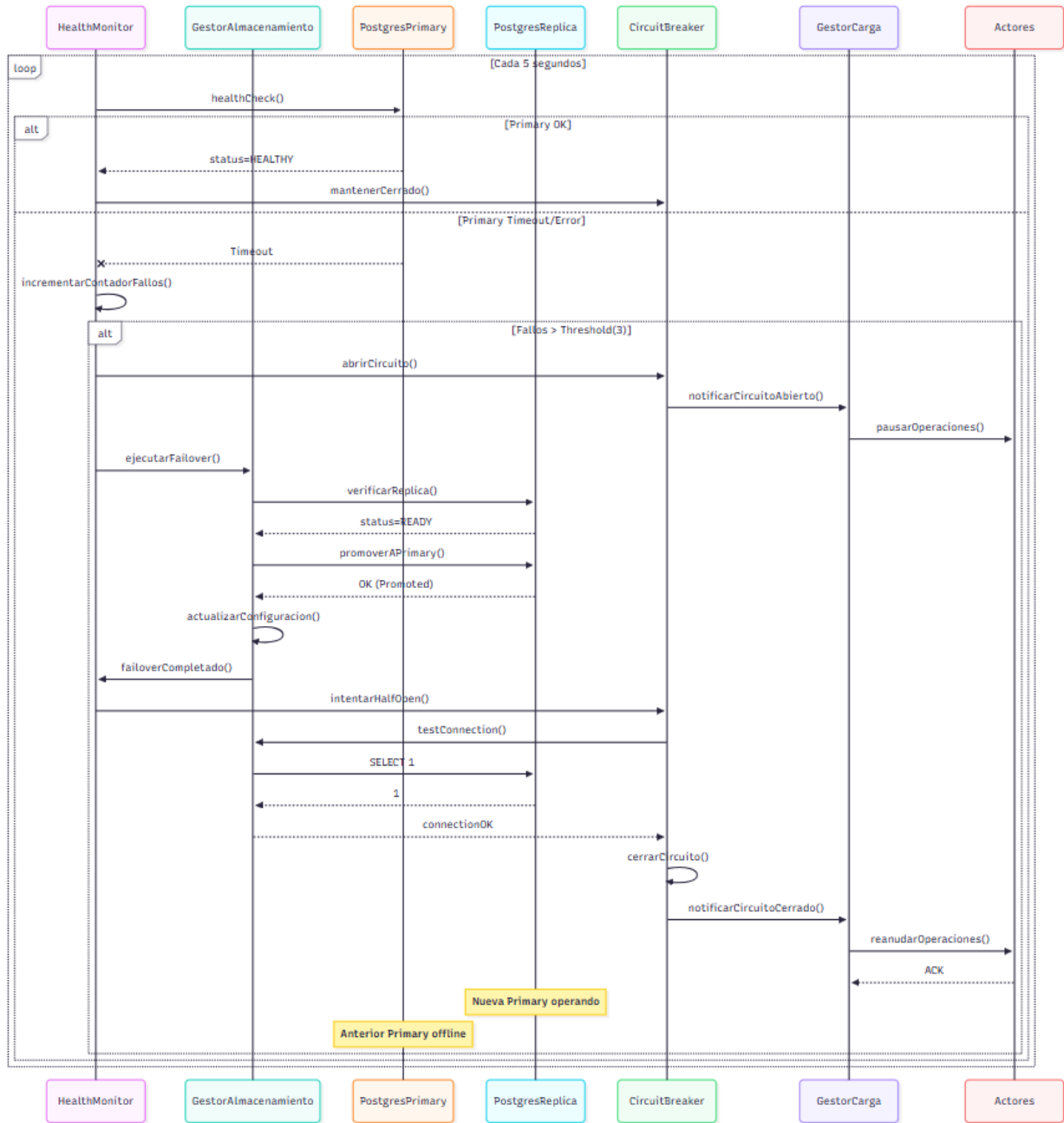


Figura 8. Secuencia de **Fallo y Recuperación**: detección, apertura de circuito, promoción de réplica y cierre del circuito.

IV. PROTOCOLO DE PRUEBAS

IV-A. Pruebas Funcionales

Las pruebas funcionales validan que el sistema opere correctamente según los requisitos establecidos, verificando cada servicio ofrecido: préstamos, devoluciones y renovaciones de libros. Estas pruebas garantizan que las reglas de negocio sean correctas, precisas y fiables desde la perspectiva del usuario.

IV-A1. Pruebas Unitarias

Son pruebas a nivel de código que se enfocan en la unidad más pequeña de software que puede ser aislada y probada, como una función, un método o una clase. Su propósito es verificar el correcto funcionamiento individual

de cada componente del sistema de forma aislada, asegurando que cada módulo opere según su especificación.

Cuadro III
CASOS DE PRUEBA UNITARIOS

| Prueba | Caso | Entrada | Acción | Resultado Esperado |
|--------|--|----------------------------------|--------------------------|--|
| UT-1 | Validación de lógica de renovación | Libro con 1 renovación previa | Intentar renovar | Renovación permitida, fecha de entrega aumentada 1 semana |
| UT-2 | Límite máximo de renovaciones | Libro con 2 renovaciones previas | Intentar renovar | Renovación denegada, mensaje de error indicando límite alcanzado |
| UT-3 | Disponibilidad de libro con ejemplares | Libro con 3 ejemplares | Consultar disponibilidad | Retorna <code>true</code> |
| UT-4 | Disponibilidad de libro vacío | Libro con 0 ejemplares | Consultar disponibilidad | Retorna <code>false</code> |
| UT-5 | Validación de devolución de libro | Libro prestado | Ejecutar devolución | Número de ejemplares incrementa en 1 |
| UT-6 | Préstamo de libro disponible | Libro disponible | Ejecutar solicitud | Retorna éxito y decrementa número de ejemplares en BD |

■ Pasos

1. Seleccionar la función o método a probar.
2. Aislar la función de sus dependencias externas. Si necesita cierto dato, hacer un mock.
3. Utilizar un framework de pruebas, para escribir un script de prueba.
4. Dentro del script, se invoca la función con los datos de entrada definidos y se verifica automáticamente que el resultado devuelto coincida con el resultado esperado mediante una aserción.

■ Criterios de Aceptación

1. El sistema debe permitir correctamente la renovación de un libro que tiene exactamente una renovación previa, extendiendo la fecha de entrega exactamente siete días naturales a partir de la fecha de vencimiento actual.
2. El sistema debe rechazar explícitamente cualquier solicitud de renovación que supere el límite de dos renovaciones por libro.
3. Cuando se consulta la disponibilidad de un libro que tiene tres o más ejemplares en inventario, el sistema debe retornar un valor booleano `true` junto con un objeto de respuesta que incluya el número exacto de ejemplares disponibles, el identificador del libro, y un estado “disponible”.
4. Al verificar la disponibilidad de un libro con cero ejemplares en inventario, el sistema debe retornar consistentemente el valor booleano `false` en todas las réplicas de la base de datos, acompañado de un estado “agotado” y un mensaje que indique “No hay ejemplares disponibles de este libro en este momento”.
5. Al procesar la devolución de un libro previamente prestado, el sistema debe incrementar exactamente en una unidad el contador de ejemplares disponibles en la base de datos, actualizar el estado del préstamo a “devuelto” con marca de tiempo precisa, y liberar el ejemplar para que esté inmediatamente disponible para nuevos préstamos.
6. Cuando un usuario solicita el préstamo de un libro disponible, el sistema debe ejecutar una transacción que: disminuya en uno el número de ejemplares disponibles, registre el nuevo préstamo con todos los datos requeridos, y retorne un mensaje de éxito “prestamo_aprobado”.

IV-A2. Pruebas de Funcionalidad Básica

Son pruebas de extremo a extremo (end-to-end) que simulan interacciones reales del usuario para validar los flujos completos del sistema. Su propósito es verificar que todos los componentes (PS, GC, Actores, GA, BD) colaboran correctamente para completar una operación de negocio.

Cuadro IV
CASOS DE PRUEBA FUNCIONALES

| Prueba | Caso | Entrada | Acción | Resultado Esperado |
|--------|------------------|--|------------------------|--|
| FB-1 | Préstamo exitoso | Libro "Cien Años de Soledad", 2 ejemplares disponibles | PS solicita préstamo | Préstamo aprobado, número de ejemplares decremента a 1, PS recibe confirmación |
| FB-2 | Préstamo fallido | Libro "El Aleph", 0 ejemplares | PS solicita préstamo | Préstamo rechazado, PS recibe mensaje "no disponible" |
| FB-3 | Renovación 1 | Libro "Rayuela", 0 renovaciones previas | PS solicita renovación | Fecha de entrega incrementa 1 semana, mensaje positivo al PS |
| FB-4 | Renovación 2 | Libro "Rayuela", 1 renovación previa | PS solicita renovación | Fecha de entrega incrementa otra semana, PS recibe confirmación |
| FB-5 | Renovación 3 | Libro "Rayuela", 2 renovaciones previas | PS solicita renovación | Renovación denegada, mensaje de error al PS |
| FB-6 | Devolución | Libro "Fahrenheit 451" prestado | PS solicita devolución | Número de ejemplares incrementa en BD, PS recibe confirmación |

■ Pasos

1. Desplegar el sistema completo en el entorno de pruebas, con todas las máquinas y contenedores en funcionamiento.
2. Asegurar que la base de datos se encuentra en un estado inicial conocido.
3. Ejecutar un Proceso Solicitante (PS) para enviar una petición específica al Gestor de Carga (GC).
4. Monitorear los logs de los diferentes componentes para observar el flujo de la petición a través del sistema.
5. Registrar la respuesta final recibida por el PS.
6. Observar la base de datos para verificar que el estado de los datos se ha actualizado correctamente.

■ Criterios de Aceptación

1. El préstamo se considerará exitoso si el sistema permite la operación únicamente cuando existen ejemplares disponibles en la base de datos. El número de ejemplares del libro debe decremента en uno. Además, el Proceso Solicitante debe recibir una confirmación de que la operación se ha realizado, garantizando que el usuario tiene información precisa sobre su préstamo.
2. Se considera aceptable que el préstamo sea rechazado cuando el libro no tiene ejemplares disponibles. En este caso, la BD no debe sufrir ningún cambio en el número de ejemplares. El PS debe recibir un mensaje que indique que la solicitud no puede ser atendida debido a la falta de ejemplares.
3. La renovación de un libro se considerará exitosa cuando el sistema permita la operación solo si el libro no ha alcanzado el límite máximo de dos renovaciones. La fecha de entrega debe incrementarse exactamente en siete días a partir de la fecha actual, y el PS debe recibir un mensaje de confirmación con la nueva fecha.
4. La renovación debe ser denegada si el libro ya ha sido renovado dos veces. En este caso, la fecha de entrega no debe cambiar y el PS debe recibir un mensaje claro que indique que se ha alcanzado el límite de renovaciones.
5. La operación de devolución se considera correcta si el número de ejemplares del libro en la BD incrementa en uno inmediatamente después de la operación. El PS debe recibir confirmación inmediata de la devolución, garantizando que el usuario conoce que el libro ha sido recibido por la biblioteca.
6. Todas las operaciones de préstamo, renovación y devolución deben reflejarse correctamente en la base de datos primaria y en sus réplicas. No debe haber inconsistencias, pérdidas de información ni errores al ejecutar múltiples solicitudes, ya sean consecutivas o concurrentes.

IV-A3. Pruebas de Persistencia e Integridad de Datos

Estas pruebas se centran en la capa de datos del sistema. Su propósito es garantizar que la información se almacena de forma correcta y consistente en la base de datos, y que no se corrompe, especialmente en situaciones de fallos.

También validan que la replicación de datos entre la base de datos primaria y la secundaria funcione como se espera.

Cuadro V
CASOS DE PRUEBA FUNCIONALES ADICIONALES

| Prueba | Caso | Entrada | Acción | Resultado Esperado |
|--------|---------------------------------------|---|---|--|
| PI-1 | Secuencia de operaciones mixtas | Ejecutar 5 préstamos, 3 devoluciones y 2 renovaciones desde distintos PS | Ejecutar las operaciones de forma concurrente | BD primaria y réplica reflejan todas las operaciones correctamente; número de ejemplares y fechas de entrega actualizadas |
| PI-2 | Reinicio del GA | GA se apaga y se reinicia mientras se realizan operaciones | Reiniciar el GA durante la ejecución | BD mantiene consistencia; operaciones previas no se pierden; PS recibe confirmaciones correctas |
| PI-3 | Fallo concurrente en libro único | Varios PS solicitan simultáneamente un libro con un único ejemplar disponible | Solicitudes concurrentes de préstamo | Solo un préstamo es aprobado; otros reciben rechazo; BD refleja correctamente 0 ejemplares disponibles |
| PI-4 | Verificación de réplica secundaria | Ejecutar operaciones mientras GA cae temporalmente | Realizar operaciones durante caída temporal | La réplica secundaria refleja los cambios realizados; no hay pérdida de datos; consistencia entre primario y réplica garantizada |
| PI-5 | Persistencia tras reinicio de PS y GC | Reiniciar procesos PS y GC durante operaciones | Reiniciar PS y GC durante la ejecución | Estado de BD se mantiene; las operaciones continúan correctamente tras reconexión; mensajes de confirmación llegan al PS |

■ Pasos

1. Preparar la base de datos en un estado predefinido.
2. Ejecutar los scripts que generan la condición de prueba.
3. Tras la ejecución, observar la base de datos primaria y la réplica.
4. Comparar el estado final de los datos en las tablas con el estado que teóricamente deberían tener.
5. Para el caso 4, se verifica que la operación fallida no haya dejado registros inconsistentes o basura en la base de datos.

■ Criterios de Aceptación

1. Todas las operaciones de préstamo, devolución y renovación ejecutadas de forma concurrente se reflejan correctamente en la base de datos primaria y en la réplica secundaria. Cada número de ejemplares debe actualizarse con precisión y las fechas de entrega deben reflejar correctamente las renovaciones realizadas. Además, los usuarios deben recibir confirmaciones correctas de sus operaciones, garantizando que el sistema mantiene coherencia de los datos bajo cargas mixtas y simultáneas.
2. Al apagar y reiniciar el Gestor de Almacenamiento (GA) durante la ejecución de operaciones, la base de datos mantiene su consistencia y no se pierden las operaciones que se estaban procesando. Todos los procesos solicitantes deben recibir confirmaciones correctas de sus transacciones, demostrando que el sistema puede recuperar automáticamente su estado y continuar funcionando sin afectar la integridad de los datos ni la experiencia del usuario.
3. Cuando varios procesos solicitantes intentan tomar simultáneamente un libro con un único ejemplar disponible, únicamente se aprueba un préstamo y los demás reciben un rechazo claro. La base de datos refleja correctamente que ahora hay 0 ejemplares disponibles.
4. Durante la ejecución de operaciones mientras el GA cae temporalmente, la réplica secundaria refleja todos los cambios realizados y no se produce pérdida de datos. La consistencia entre la base de datos primaria y la réplica secundaria se mantiene.
5. Al reiniciar los procesos solicitantes (PS) y el Gestor de Carga (GC) durante operaciones en curso, el estado de la base de datos se mantiene intacto y todas las transacciones pendientes se completan correctamente tras la reconexión. Los PS reciben confirmaciones de las operaciones.

IV-B. Pruebas de Desempeño

Las pruebas de desempeño evalúan la capacidad del sistema para manejar carga bajo diferentes escenarios, su foco está en medir aspectos de calidad no funcionales como la velocidad de respuesta (latencia), la cantidad de operaciones que puede manejar por segundo (throughput) y la estabilidad general bajo estrés.

IV-B1. Pruebas de Carga

Son pruebas que evalúan el comportamiento y la capacidad del sistema cuando se somete a un número significativo y concurrente de solicitudes, simulando el uso real por parte de múltiples usuarios a la vez. Su propósito no es encontrar errores funcionales, sino medir atributos de calidad como el tiempo de respuesta, el throughput (solicitudes procesadas por segundo) y el consumo de recursos (CPU, memoria).

Cuadro VI
CASOS DE PRUEBA FUNCIONALES DE CARGA

| Prueba | Caso | Entrada | Acción | Resultado Esperado |
|--------|------------------|--|---|--|
| PCA-1 | Carga progresiva | Incrementar gradualmente el número de PS generando solicitudes de préstamo, renovación y devolución (4, 6, 10 PS por sede) | Ejecutar las solicitudes de manera secuencial y concurrente | Tiempos de respuesta medidos; BD mantiene consistencia; todas las operaciones procesadas correctamente |
| PCA-2 | Carga máxima | Ejecutar simultáneamente 10 PS por sede con operaciones mixtas | Procesar todas las solicitudes de forma concurrente | Sistema no colapsa; GC y Actores procesan todas las solicitudes; tiempos de respuesta dentro de un rango aceptable |
| PCA-3 | Carga sostenida | Mantener 6 PS por sede generando solicitudes durante 2 minutos continuos | Ejecutar solicitudes continuamente durante el tiempo definido | Sistema mantiene desempeño estable; no se pierden solicitudes; PS reciben confirmaciones correctas |

■ Pasos

1. Desplegar el sistema en el entorno de pruebas. Asegurar que la base de datos está en su estado inicial y que las herramientas de monitoreo (Prometheus, Grafana) están activas y recolectando métricas.
2. Configurar la herramienta Locust en la máquina de Clientes, definiendo el número de usuarios virtuales (PS) para el caso de prueba (4, 6 o 10) y los endpoints de los Gestores de Carga de cada sede.
3. Iniciar la prueba de carga desde Locust. La prueba se ejecutará de forma continua durante un periodo exacto de 2 minutos.
4. Durante la ejecución, observar en tiempo real los dashboards de Grafana para monitorizar el tiempo de respuesta, el número de solicitudes por segundo y el estado de los recursos de los contenedores.
5. Al finalizar los 2 minutos, detener la prueba y exportar los datos agregados desde Locust (tiempo de respuesta promedio, desviación estándar) y el número total de solicitudes procesadas desde Grafana.
6. Repetir los pasos 2 a 5 para cada nivel de carga y para cada una de las dos arquitecturas a comparar (original vs. modificada).

■ Criterios de Aceptación

1. El sistema debe ser capaz de manejar un aumento gradual en el número de procesos solicitantes (PS) generando solicitudes de préstamo, devolución y renovación de forma secuencial y concurrente, sin que se pierda ninguna operación. La base de datos primaria y la réplica deben mantenerse consistentes, reflejando correctamente los cambios en el número de ejemplares y fechas de entrega. Los tiempos de respuesta medidos deben ser razonables y se le debe dar a los usuarios confirmaciones precisas de sus operaciones, asegurando que el sistema pueda escalar de manera controlada bajo cargas crecientes.
2. El sistema debe procesar simultáneamente varios PS por sede realizando operaciones mixtas sin colapsar ni generar errores en la base de datos. El Gestor de Carga y los Actores deben ser capaces de atender todas las solicitudes concurrentes, garantizando que cada operación se complete correctamente. Los tiempos de respuesta deben mantenerse dentro de un rango aceptable, asegurando que puede soportar picos de carga sin afectar la experiencia del usuario ni la consistencia de la información.

3. El sistema debe mantener un desempeño estable cuando 6 PS por sede generan solicitudes de manera continua durante 2 minutos. Ninguna operación debe perderse y todos los procesos solicitantes deben recibir confirmaciones correctas de sus transacciones.

IV-B2. Pruebas de Concurrencia

Estas pruebas se centran en verificar la robustez del sistema cuando múltiples procesos intentan acceder y modificar al mismo tiempo un mismo recurso compartido. A diferencia de las pruebas de carga, que distribuyen las solicitudes de manera uniforme, estas pruebas buscan provocar intencionadamente colisiones y condiciones de carrera. Su propósito es asegurar que los mecanismos de control de concurrencia como bloqueos de base de datos o transacciones, funcionen correctamente y mantengan la integridad y consistencia de la información.

Cuadro VII
CASOS DE PRUEBA DE CONCURRENCIA

| Prueba | Caso | Entrada | Acción | Resultado Esperado |
|--------|---------------------------------------|---|--|---|
| PCN-1 | Préstamo simultáneo de libro único | Varios PS solicitan un mismo libro con un ejemplar disponible | Solicitudes concurrentes de préstamo | Solo un PS obtiene confirmación de préstamo; otros reciben rechazo; BD refleja 0 ejemplares disponibles |
| PCN-2 | Devolución concurrente | Dos PS devuelven el mismo libro al mismo tiempo | Ejecutar devoluciones concurrentes | Número de ejemplares incrementa en 2; BD refleja correctamente el total; PS reciben confirmación inmediata |
| PCN-3 | Préstamo durante devolución asíncrona | PS1 envía una solicitud para devolver el último ejemplar de un libro (stock pasa de 0 a 1). Inmediatamente después de que el GC responde a PS1 (pero antes de que el Actor actualice la BD), PS2 envía una solicitud para prestar ese mismo libro | Ejecutar la devolución de PS1 y, antes de que el Actor actualice la BD, enviar la solicitud de préstamo de PS2 | PS1 recibe confirmación inmediata de devolución. PS2 recibe rechazo de préstamo porque la BD aún no refleja el ejemplar devuelto. Una vez que el Actor procesa la actualización, la BD refleja correctamente 1 ejemplar disponible. |

■ Pasos

1. Modificar la base de datos manualmente para que un libro específico tenga exactamente 1 ejemplar disponible.
2. Asegurar que un libro tenga 0 ejemplares disponibles.
3. Crear un script que lance dos clientes (PS) de forma concurrente . Este script se encargará de que ambas solicitudes se envíen a la red con la menor diferencia de tiempo posible.
4. Registrar las respuestas recibidas por ambos PS.
5. Inmediatamente después de la prueba, realizar observar la base de datos para verificar el estado final del recurso disputado (el número de ejemplares del libro).

■ Criterios de Aceptación

1. El sistema debe garantizar que, cuando varios procesos solicitantes (PS) intentan tomar simultáneamente un libro con un único ejemplar disponible, únicamente uno de ellos reciba confirmación de préstamo y los demás reciban un rechazo claro. La base de datos debe reflejar correctamente que ahora hay 0 ejemplares disponibles.
2. El sistema debe procesar correctamente la devolución de un libro cuando dos PS realizan la operación al mismo tiempo, incrementando el número de ejemplares en 2 y reflejando correctamente el total en la base de datos. Los procesos solicitantes deben recibir confirmación inmediata de la operación.
3. Si un proceso solicitante devuelve un libro y otro intenta tomarlo inmediatamente después, antes de que la BD se actualice, el sistema debe permitir que la devolución se registre correctamente y que la solicitud de préstamo posterior reciba un rechazo temporal hasta que la BD refleje el ejemplar disponible. Una vez que el Actor procesa la actualización, la BD debe reflejar correctamente el nuevo estado del ejemplar.

IV-C. Pruebas de Fallos

Las pruebas de fallos se encargan de comprobar qué tan capaz es el sistema de resistir y recuperarse cuando ocurre algún imprevisto. El objetivo no es solo ver si el sistema falla, sino cómo falla y si sus mecanismos de recuperación automática funcionan como se espera.

IV-C1. Prueba de Fallo del GA / Réplica Primaria

Esta prueba consiste en simular de manera intencional la caída completa del Gestor de Almacenamiento o, en especial, de la base de datos primaria. Su objetivo es comprobar que el sistema puede tolerar fallos y seguir funcionando según los requisitos establecidos. Se busca verificar que los mecanismos de detección de fallos (Health Checker) y de conmutación (Failover Automático) se activan y operan correctamente para mantener el sistema disponible, garantizando la continuidad del servicio con una interrupción mínima o nula para el usuario final.

Cuadro VIII
CASOS DE PRUEBA DE FALLOS

| Prueba | Caso | Entrada | Acción | Resultado Esperado |
|--------|-----------------------|---|---|---|
| PF-1 | Falla del GA primario | Operaciones de préstamo, devolución y renovación en curso | Apagar el GA primario durante las operaciones | El sistema detecta la falla; todas las operaciones se redirigen automáticamente a la réplica secundaria; PS recibe confirmaciones correctas |

■ Pasos

1. Iniciar el despliegue del sistema en las tres máquinas. Iniciar una prueba de carga moderada (ej. 4 o 6 PS) desde Locust para que haya un flujo constante de transacciones.
2. Conectar a la máquina Computador1 - sede1 y ejecutar el comando `docker stop` sobre el contenedor de la base de datos primaria.
3. Monitorear activamente:
 - Los logs del Health checker & Circuit breaker: observar cómo detecta que la conexión con la base de datos primaria ha fallado.
 - Los logs del script de Failover Automático: verificar cómo se activa tras la notificación del Health Checker y cómo inicia el proceso de promoción de la réplica.
 - El dashboard de Locust: observar si hay un pico de peticiones fallidas y medir cuánto tiempo dura la interrupción del servicio, si la hay.
4. Una vez que el sistema se estabilice, observar la base de datos en Computador2 - sede2 para confirmar que ha sido promovida a primaria y que está aceptando nuevas operaciones de escritura.

■ Criterios de Aceptación

1. El sistema debe ser capaz de detectar de manera automática la falla del Gestor de Almacenamiento o de la réplica primaria de la base de datos durante operaciones en curso de préstamo, devolución y renovación. El Health Checker debe identificar la falla en un tiempo predefinido (por ejemplo, menos de 15 segundos) y activar el proceso de conmutación. El script de Failover Automático debe promover correctamente la réplica secundaria ubicada en la otra sede a base de datos primaria, garantizando que las operaciones pendientes se procesen sin intervención manual.
2. Durante la conmutación, se permite una breve ventana de errores transitorios, pero el sistema no debe quedar permanentemente inoperativo. Una vez completado el failover, todas las solicitudes de los procesos solicitantes (PS) deben continuar procesándose de forma automática, y los usuarios deben recibir confirmaciones correctas de sus operaciones.
3. No debe existir pérdida de datos ni corrupción en la base de datos; todas las operaciones previas deben reflejarse correctamente, y la consistencia debe mantenerse entre la nueva primaria y la réplica secundaria.

IV-C2. Prueba Recuperación tras Reinicio

Esta prueba evalúa la capacidad del sistema para reintegrar un componente que ha fallado y ha sido restaurado. Dado que los componentes se ejecutan en contenedores, un reinicio es una estrategia de recuperación común. El propósito es asegurar que la arquitectura distribuida es robusta y que el fallo de un componente no causa un colapso en cascada del sistema. Se busca validar que los componentes pueden reiniciarse, reintegrarse y continuar sus funciones, manejando adecuadamente las operaciones que estaban en curso.

Cuadro IX
CASOS DE PRUEBA DE RECUPERACIÓN TRAS REINICIO

| Prueba | Caso | Entrada | Acción | Resultado Esperado |
|--------|--------------------------|--|---------------------------------------|---|
| PRI-1 | Reinicio de GA o PS | Reiniciar GA, GC o PS mientras se procesan operaciones | Ejecutar reinicio durante operaciones | El sistema recupera el estado previo; todas las operaciones pendientes se completan; PS reciben confirmaciones correctas |
| PPI-2 | Reinicio del GC en sede1 | Locust genera solicitudes de préstamo, devolución y renovación hacia sedes 1 y 2 | Ejecutar reinicio durante operaciones | El GC se reinicia correctamente; PS de sede1 y sede2 continúan enviando solicitudes; las operaciones pendientes se procesan correctamente después de la reconexión; la BD mantiene consistencia |

■ Pasos

1. Iniciar el despliegue completo y una prueba de carga constante desde Locust.
2. Conectar por terminal a la máquina Computador1 - sede1 y ejecutar el comando `docker restart` sobre el contenedor del Gestor de Almacenamiento/Carga.
3. Monitorear los logs de los Actores de la sede1. Deben registrar errores de conexión al intentar comunicarse con el GA/GC y, idealmente, activar una lógica de reintentos.
4. En Locust, observar si los tiempos de respuesta para operaciones síncronas (préstamos) en la sede1 aumentan drásticamente o resultan en `timeouts`, ya que los Actores no pueden contactar al GA.
5. Verificar que las operaciones en la sede2 no se vean afectadas.
6. Una vez que el GA/GC se ha reiniciado, observar en los logs de los Actores si logran restablecer la conexión y si las operaciones que estaban pendientes (o siendo reintentadas) ahora se completan exitosamente.

■ Criterios de Aceptación

1. El sistema debe ser capaz de recuperarse automáticamente de un reinicio de cualquiera de sus componentes críticos, como el Gestor de Almacenamiento (GA), el Gestor de Carga (GC) o los procesos solicitantes (PS), mientras se procesan operaciones de préstamo, devolución y renovación. El contenedor del GA debe reiniciarse correctamente y restablecer su pool de conexiones a la base de datos, de manera que pueda atender nuevamente las solicitudes sin intervención manual.
2. Durante el reinicio, los Actores no deben `crash`ear ni generar errores graves; deben manejar la indisponibilidad del GA de forma controlada, esperando a reconectarse automáticamente. Una vez que el GA vuelve a estar en línea, los Actores deben reconectarse automáticamente y continuar procesando todas las operaciones pendientes.
3. El sistema se considera exitoso si el GC se reinicia correctamente sin interrumpir permanentemente la operación de los PS ni de los Actores. Los procesos solicitantes deben poder seguir enviando solicitudes durante el reinicio, y las operaciones pendientes deben procesarse correctamente una vez que el GC esté nuevamente en línea. No debe producirse pérdida de datos ni corrupción en la base de datos; todas las transacciones deben reflejarse correctamente tanto en la réplica primaria como en la secundaria.

V. MÉTRICAS DE DESEMPEÑO

Para la ejecución de esta sección, se realizó la elección de la opción de mejora *B*: Comunicaciones asíncronas y síncronas. Esta decisión se basa en el interés de analizar una de las cuestiones más importantes en el diseño

de sistemas distribuidos: cómo equilibrar la consistencia de los datos, la rapidez con la que el usuario recibe la respuesta y el rendimiento general del sistema.

V-A. Herramientas de Monitoreo

V-A1. Locust

Se trata de un framework de pruebas de carga y rendimiento de código abierto, desarrollado en Python que permite definir el comportamiento de múltiples usuarios escribiendo su interacción en forma de código. Dentro de este proyecto, su rol será ser la herramienta responsable de simular los Procesos Solicitantes (PS). Se configurará para generar la carga de trabajo especificada (4,6,10 usuarios concurrentes) con una mezcla de solicitudes de préstamo, devolución y renovación. De esta forma se podrá medir el tiempo que tarda el sistema en procesar cada solicitud bajo diferentes niveles de concurrencia.

Locust es la herramienta encargada de obtener la Métrica 1 (Tiempo de Respuesta), ya que actúa como el cliente en las pruebas. Por esta razón, es el único componente capaz de medir el tiempo total de extremo a extremo, es decir, desde el momento en que se envía una solicitud hasta que se recibe la respuesta completa del Gestor de Carga (GC).

V-A2. Prometheus

Es un sistema de monitoreo y alerta de código abierto diseñado para confiabilidad y escalabilidad. Recoge y almacena métricas de series temporales. Dentro de este proyecto, actuará como la base de datos central encargada de almacenar las series temporales. No realiza mediciones de forma directa, sino que recopila información de manera periódica al consultar un endpoint HTTP (/metrics) que cada componente expone, para recolectar las métricas que esta genera. Será el backend para la Métrica 2 (Cantidad de Solicitudes Procesadas), ya que almacenará el conteo de solicitudes a lo largo del tiempo.

V-A3. Grafana

Es una plataforma de visualización y análisis de código abierto que permite crear dashboards interactivos donde se puedan observar los resultados de las pruebas de carga y métricas de series temporales. Se conectará a Prometheus como fuente de datos para crear dashboards. Se diseñará un dashboard específico para las pruebas de rendimiento, el cual mostrará, mediante un gráfico o un panel tipo Stat, el valor final de la Métrica 2 una vez concluya el experimento de dos minutos.

V-A4. Instrumentación de Código

Además del monitoreo externo, se incluirán métricas dentro del código de los componentes críticos (GA y GC), con el fin de capturar información sobre los tiempos de procesamiento interno. Para que Prometheus pueda recolectar métricas, el código debe exponerlas, así que, se añadirá una librería cliente de Prometheus (como prometheus-client) al código del Gestor de Carga (GC).

V-B. Métricas a Medir

1. Tiempo de respuesta promedio para préstamos: Es un indicador directo de la latencia percibida por el usuario y la eficiencia del flujo de comunicación, siendo una métrica importante para evaluar la experiencia del cliente.
2. Desviación estándar del tiempo de respuesta: Un valor bajo indicaría que el rendimiento del sistema es consistente. Un valor alto, podría sugerir inestabilidad o que algunos usuarios experimentan demoras mayores que otros, aun cuando el promedio sea aceptable.
3. Throughput (solicitudes procesadas en 2 minutos): El número de operaciones exitosamente completadas por el sistema durante el intervalo de medición definido. Esta métrica refleja la capacidad del sistema para manejar carga concurrente y es un indicador directo de su rendimiento y escalabilidad.
4. Tasa de fallos y recuperación: Porcentaje de solicitudes que no pudieron ser completadas exitosamente durante las pruebas, ya sea por timeouts, errores de conexión o fallos inducidos. Por otro lado, la recuperación, evalúa el tiempo que tarda el sistema en volver a un estado operativo estable tras un fallo (Time to Recover - TTR) y si dicha recuperación fue automática.

VI. IMPLEMENTACIÓN INICIAL

VI-A. *Avances Desarrollados*

- Arquitectura inicial del proyecto.
- Mecanismo de generación de requerimientos.
- Solicitud de operaciones de devolución y renovación desde los PS hasta los Actores.

VII. CONCLUSIONES

Esta entrega reúne los resultados de la fase de modelado y diseño arquitectónico del sistema de biblioteca, planteado una estructura robusta a través de los modelos, enfocado en crear un proyecto completo incorporando varias tecnologías. Se desarrolló un protocolo de pruebas completo que incluye casos funcionales, de resiliencia y de rendimiento, junto con una metodología clara para su evaluación. Se definió además la estrategia de obtención de métricas clave, utilizando Locust para la generación controlada de carga y Prometheus como parte del stack de monitoreo para la recolección de datos.