Universidad Siglo 21

Licenciatura en Informática

Seminario de Práctica de Informática

Sistema de Gestión de Préstamos para COFINSUR S.A. (SiGeP)

Alumno:	FERNÁNDEZ NAVARRO, Cristian José
Legajo:	VINF02891
Fecha:	27/04/2025
Docente/Tutor:	VIRGOLINI, Pablo Alejandro

Tabla de contenido

Introducción

El presente documento tiene como objetivo presentar el análisis y diseño de un sistema de información orientado a la gestión de préstamos para la empresa **COFINSUR S.A.**.
Esta organización, dedicada al otorgamiento de préstamos personales y comerciales, requiere una solución tecnológica que permita optimizar sus procesos administrativos y operativos. Entre estos se incluyen la gestión de prestamos, el control de cuotas, la gestión del historial de clientes y la generación de reportes financieros.
Con este desarrollo se busca mejorar la eficiencia operativa, garantizar la trazabilidad de las operaciones y fortalecer la confiabilidad en la administración de los prestamos otorgados.
Asimismo, se pretende acompañar el crecimiento de la empresa y facilitar la toma de decisiones mediante el acceso a información precisa, centralizada y actualizada.

Antecedentes

COFINSUR S.A. fue fundada en el año 2010 en la ciudad de Ushuaia, provincia de Tierra del Fuego, con el propósito de brindar asistencia financiera a trabajadores de la zona, enfocándose especialmente en empleados del sector público y pequeños comerciantes. En sus primeros años, la empresa operaba como una oficina local, prestando servicios limitados y gestionando sus operaciones de manera manual.
Con el tiempo, y en respuesta a una creciente demanda, la organización expandió su ámbito de acción, abriendo nuevas sucursales en las ciudades de Río Grande y Tolhuin, consolidando así su presencia en toda la provincia.
En una etapa posterior, y con el objetivo de mejorar la eficiencia administrativa, se incorporó una aplicación informática que permitió informatizar parcialmente la gestión de créditos. No obstante, esta herramienta, si bien fue funcional en su momento, actualmente se encuentra tecnológicamente desactualizada y presenta importantes limitaciones tanto técnicas como funcionales, lo que afecta la agilidad y confiabilidad de los procesos operativos.

Frente al avance sostenido de las tecnologías de la información y las comunicaciones, y considerando el crecimiento experimentado por la empresa, se torna imprescindible modernizar el sistema actual. En este contexto, se propone el desarrollo de un nuevo sistema integral que

permita consolidar la información institucional, interconectar las distintas sucursales, optimizar los procesos internos y acompañar el crecimiento estratégico de la empresa.

Descripción del área problemática

El sistema actualmente en uso presenta serias limitaciones técnicas y funcionales que afectan de manera directa su operatividad y dificultan el normal desenvolvimiento de las actividades administrativas, operativas y gerenciales. A continuación, se detallan los principales problemas identificados:

- **Bases de datos desconectadas:** Cada sucursal opera con su propia base de datos local, sin ningún tipo de conexión con el resto de la organización. Esta situación impide la centralización de la información y genera importantes dificultades para obtener una visión consolidada de los clientes, los préstamos otorgados y los pagos registrados en tiempo real.
- **Dificultad para trabajo en red:** El sistema no permite la operación simultánea por parte de múltiples usuarios, lo cual limita considerablemente la capacidad de atención al público. Esta restricción genera bloqueos en el uso del sistema, ocasionando demoras en la atención al publico retrasando la ejecución de tareas cotidiana.
- **Dependencia tecnológica crítica:** El sistema carece de mantenimiento, ya que depende exclusivamente de personal con conocimientos en una tecnología desactualizada (CA-Clipper 5.2)., lo que incrementa el riesgo ante eventuales fallas o necesidades de actualización.
- **Gestión de respaldos ineficiente:** No existen mecanismos automatizados ni confiables para el respaldo y recuperación de datos. Las copias de seguridad se realizan manualmente, mediante la duplicación de archivos individuales de la base de datos, lo que representa un riesgo significativo para la integridad de la información.
- **Falta de seguridad y control de accesos**: El sistema carece de mecanismos para gestionar distintos niveles de acceso según el rol del usuario, lo que implica un riesgo en términos de confidencialidad, integridad y trazabilidad de la información sensible.

- **Ausencia de auditoría de operaciones**: No existe un registro o historial de actividades realizadas por los usuarios dentro del sistema, lo cual impide detectar errores, identificar malas prácticas o realizar seguimientos ante incidentes.

Las áreas más afectadas por esta situación son la administración, la atención al cliente y la gerencia, ya que requieren un acceso ágil, preciso y centralizado a la información institucional para poder responder a consultas, efectuar gestiones, supervisar las operaciones y tomar decisiones estratégicas. La falta de integración entre sucursales, la debilidad en los mecanismos de seguridad y la persistencia de procesos manuales representan una barrera crítica para la eficiencia operativa. Ante el crecimiento sostenido de la empresa y la complejidad creciente de sus operaciones intensifican las deficiencias del sistema actual, comprometiendo gravemente su escalabilidad y sustentabilidad a futuro.

Formulación de la problemática

¿Cómo puede **COFINSUR S.A.** reemplazar su sistema de gestión actual, basado en bases de datos locales no integradas y una tecnología obsoleta, por una solución moderna que centralice la información institucional, permita el trabajo en red entre sucursales, garantice la seguridad y trazabilidad de las operaciones, y optimice los procesos administrativos y operativos?

Justificación

La necesidad de modernizar el sistema de gestión surge de las profundas limitaciones técnicas y funcionales del sistema actual, el cual fue desarrollado en una tecnología hoy en día obsoleta (CA-Clipper 5.2). Esta herramienta, que alguna vez resultó funcional, ya no responde a las exigencias actuales de la organización ni a las dinámicas del entorno competitivo.
Disponer de un sistema de información moderno, integrado y centralizado es fundamental para garantizar la eficiencia operativa, la trazabilidad de las operaciones en tiempo real y el acceso seguro a información actualizada y confiable. Asimismo, permite automatizar numerosos procesos que actualmente se llevan a cabo de manera manual, con el consiguiente ahorro de tiempo y reducción de errores.
La implementación de una nueva solución fortalecerá la capacidad de análisis, auditoría y control

interno, disminuyendo significativamente la probabilidad de errores operativos o fraudes. Además, se prevé un impacto tecnológico positivo, dado que el proyecto contempla la adopción de herramientas actuales de desarrollo, el uso de bases de datos relacionales, mecanismos de respaldo automatizado y opciones de acceso remoto seguro, lo que representa un salto cualitativo respecto al sistema vigente.

Desde una perspectiva organizacional, el nuevo sistema contribuirá a mejorar los tiempos de atención al cliente, agilizará la gestión administrativa y operativa.

Por todo lo expuesto, la implementación de este proyecto no solo es técnicamente viable, sino que se presenta como una iniciativa estratégica y necesaria. Contar con una plataforma tecnológica adecuada no solo permitirá sostener el crecimiento de la empresa, sino que también mejorará la calidad de vida laboral del personal, al reducir la carga operativa y los riesgos asociados a la utilización de sistemas desactualizados y desentralizados.

Objetivo general del proyecto

Analizar, diseñar e implementar un sistema de información para COFINSUR S.A. que reemplace la solución tecnológica obsoleta actual, con el fin de resolver las limitaciones de descentralización de datos, falta de seguridad y procesos manuales, mediante el desarrollo de una plataforma integrada, escalable y acorde a las necesidades operativas de la organización.

Objetivos específicos del proyecto

- Analizar el funcionamiento del sistema actual y relevar los procesos administrativos, operativos y de atención al cliente en cada una de las sucursales de COFINSUR S.A.
- Identificar y documentar los requerimientos funcionales y no funcionales necesarios para el nuevo sistema, en base a entrevistas con usuarios clave y análisis de procesos.
- Diseñar una arquitectura tecnológica que centralice la información, garantice la disponibilidad y seguridad de los datos, y permita la escalabilidad futura del sistema.
- Diseñar un sistema de gestión que permita administrar clientes, préstamos y pagos de forma centralizada y segura.

- Desarrollar un prototipo funcional validado mediante pruebas de usuario y evaluación de su desempeño, asegurando que responda a los requerimientos definidos.
- Capacitar al personal en el uso del nuevo sistema, promoviendo la adopción tecnológica y mejorando la eficiencia operativa.
- Proponer adecuaciones y mejoras complementarias que optimicen la implementación del sistema, incluyendo recomendaciones sobre capacitación e infraestructura tecnológica necesaria.
- Evaluar la viabilidad técnica y operativa del sistema desarrollado, demostrando su impacto positivo en la mejora de los procesos clave de la organización.

Objetivo general del sistema

Gestionar de forma centralizada la información de clientes, préstamos y pagos, mediante un sistema, que integre las operaciones de todas las sucursales, mejore la eficiencia operativa y asegure el acceso confiable a los datos actualizados para la toma de decisiones.

Limite

El sistema abarca todo el proceso, desde el momento en que una persona o comercio es registrado como cliente, permitiendo así la gestión de operaciones vinculadas a su actividad crediticia. El sistema opera a lo largo del ciclo de vida del préstamo, incluyendo la creación, seguimiento, registro de pagos, y emisión de reportes.
El sistema finaliza cuando todos los préstamos activos de un cliente son cancelados y no existen más obligaciones pendientes, o cuando se da de baja su registro por cese de la relación comercial.

Alcance

El alcance del proyecto se establece para los siguientes procesos:

- Proceso de registro y actualización de los datos personales de clientes, incluyendo alta, modificación y mantenimiento de la información relevante para su identificación y contacto.

- Proceso de gestión de préstamos, que comprende la creación de nuevas solicitudes, la modificación de condiciones, el seguimiento del estado de cada préstamo, la simulación de escenarios de financiamiento y la cancelación anticipada o regular de los mismos.
- Proceso de administración de parámetros financieros, incluyendo la definición, actualización y control de tasas de interés, convenios con terceros, esquemas de amortización y otras variables necesarias para el cálculo y gestión de los préstamos.
- Proceso de registro de pagos y actualización de estados de cuenta, contemplando la imputación de pagos realizados, la generación y mantenimiento del historial de pagos de cada cliente, y el control de saldos pendientes.
- Proceso de gestión de acceso de usuarios, que involucra el registro de usuarios del sistema, la asignación de niveles de permisos, y el control de acceso a los diferentes procesos según el perfil correspondiente.

No Contempla

El sistema no contempla:

- Proceso de gestión de copias de seguridad automáticas y recuperación de datos ante fallos del sistema o pérdidas de información.
- Proceso de generación de reportes financieros y operativos exportables a distintos formatos para análisis externo.
- Proceso de seguimiento de morosidad avanzada y gestión de pase a ejecuciones judiciales, incluyendo la preparación de documentación y acciones legales.
- Proceso de análisis crediticio a través de consultas a bases de datos externas, como sistemas de información crediticia tipo VERAZ o servicios Web del BCRA.

Elicitación

Las técnicas de recolección de datos utilizadas en este proyecto se aplicaron en la sucursal Ushuaia de COFINSUR S.A., y tuvieron como objetivo relevar información clave sobre el funcionamiento actual del sistema y los procesos asociados.

Entre las técnicas empleadas se incluyeron:

- Entrevistas personales con miembros de diferentes áreas de la organización (administración, atención al cliente y gerencia).
- Observación directa de las tareas diarias realizadas por el personal.
- Recolección de material digital, que incluyó capturas de pantalla del sistema actual y copias de bases de datos en formato .dbf.

Activad del cliente

Para comprender el contexto en el cual opera COFINSUR S.A., resulta fundamental revisar algunos conceptos clave que caracterizan a las entidades financieras dedicadas a la gestión de préstamos personales. Esta comprensión permitirá un mejor análisis de los procesos actuales y el diseño de un sistema que responda a las necesidades específicas del sector.

¿Qué es un préstamo?

Un préstamo es un mecanismo de financiación mediante el cual una entidad financiera, denominada **prestamista**, entrega al cliente, denominado **prestatario**, una suma de dinero bajo condiciones pactadas, como plazo de devolución e intereses. (*¿Qué es un préstamo?,* Recuperado el 26 de abril de 2025 de https://www.bbva.com.ar/economia-para-tu-dia-a-dia/ef/prestamos/diferencias-entre-credito-y-prestamo.html*).*

Otorgamiento de préstamos

El otorgamiento de préstamos comprende el proceso de solicitud, evaluación, aprobación y desembolso de fondos al cliente. Durante la evaluación se consideran aspectos como los antecedentes crediticios, la capacidad de pago y el cumplimiento de requisitos formales.

Solicitud de préstamo

El proceso se inicia cuando el solicitante presenta una solicitud formal, que usualmente incluye:

- Información personal (nombre, dirección y datos de contacto).
- Información sobre ingresos y activos.

Evaluación crediticia

La evaluación crediticia es el proceso mediante el cual la entidad financiera analiza la capacidad y disposición del solicitante para asumir y cumplir con una obligación de pago. Su principal objetivo es minimizar el riesgo de incobrabilidad, asegurando que los préstamos se otorguen a clientes con una probabilidad razonable de cumplimiento.

Durante este proceso se consideran, entre otros aspectos:

- Análisis de ingresos y egresos para determinar la capacidad de pago.
- Identificación y verificación de datos personales.
- Historial crediticio en bases de datos de informes comerciales o internos.
- Relación deuda/ingreso que mide el porcentaje de ingresos comprometidos en otras deudas.
- Antigüedad laboral o comercial, estabilidad económica.
- Existencia de garantías o avales, si fueran requeridos.

Aprobación o denegación

Finalizada la evaluación crediticia, la entidad toma la decisión de aprobar o rechazar la solicitud de préstamo, en función de los resultados obtenidos.

En caso de aprobación, se realiza una oferta basada en la capacidad de pago del solicitante, especificando términos y condiciones como el monto, el número de cuotas y el tipo de interés. El prestatario podrá aceptar o rechazar esta oferta.

Si la evaluación indica un alto riesgo de incumplimiento, la solicitud será denegada.

Mora

La mora se refiere al incumplimiento en el pago de las cuotas en los plazos pactados. Ante esta situación, las entidades financieras suelen aplicar recargos o intereses punitorios, y eventualmente

iniciar procesos de recuperación de créditos.

Gestión de cobranzas

La gestión de cobranzas implica el seguimiento de los pagos realizados, la identificación de deudores morosos y la ejecución de acciones destinadas a recuperar los montos adeudados.

T.I.C (Tecnología de la Información y Comunicación)

En función de los objetivos establecidos para el desarrollo del nuevo sistema de gestión de préstamos (SIGeP) para **COFINSUR S.A.**, resulta fundamental analizar y definir los recursos tecnológicos necesarios para garantizar una solución robusta, escalable y eficiente.

Arquitectura general

El sistema SIGeP estará compuesto por los siguientes elementos tecnológicos:

- **Aplicación cliente-servidor:** desarrollada en Java, la aplicación se instalará en las terminales de cada sucursal. Permitirá a los usuarios interactuar con el sistema mediante una interfaz gráfica moderna, accediendo a la información en tiempo real.
- **Base de datos:** se utilizará **MySQL**, instalado en un servidor central ubicado en la sede principal, que es la sucursal Ushuaia. Todas las instancias de la aplicación se conectarán a esta base de datos a través de la red, utilizando una configuración segura.
- **Conectividad entre sucursales**: se implementará una VPN (Red Privada Virtual) para permitir que las terminales de cada sucursal accedan al servidor de base de datos de forma segura. La VPN garantizará la confidencialidad e integridad de la información transmitida entre sedes.

Descripción de red

El sistema operará en un entorno cliente-servidor distribuido. Las terminales de cada sucursal se conectarán mediante una red local (LAN) a la infraestructura de red interna.

A continuación, se describen las tecnologías, conceptos y metodologías relevantes para el desarrollo del proyecto, incluyendo aquellas que presentan un alto potencial para ser utilizadas o implementadas.

UML: El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelados visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimientos sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener y controlar la información sobre tales sistemas. (Rumbaugh, Jacobson, & Booch, 2000, pág. 4).

Scrum: Metodología ágil usada frecuentemente para desarrollar, mantener y entregar software en situaciones de complejidad e incertidumbre (Schwaver et al., 2006).

Java: Java es un lenguaje de programación orientado a objetos creado en 1991 y publicado en 1995 por Sun Microsystem (adquirida por Oracle en 2010), con la intención de que los programadores escribieran el código solo una vez y lo ejecutarán en cualquier dispositivo. Esto es posible gracias a que Java cuenta con una JVM o Java Virtual Machine que brinda portabilidad al lenguaje, ya que hoy existen JVMs para diferentes arquitecturas para todas las plataformas. (Guevara Benites, 2017).

JavaFX: JavaFX es una familia de productos y tecnologías de Oracle Corporation (inicialmente Sun Microsystems), para la creación de Rich Internet Applications (RIAs), esto es, aplicaciones web que tienen las características y capacidades de aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas. (JavaFX . (2025, abril 24). En Wikipedia. Recuperado de https://es.wikipedia.org/wiki/JavaFX)

API REST: Una API de transferencia de estado representacional (REST), o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) creada por el informático Roy Fielding, la cual se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful. (Red Hat Inc. 2021,¿Qué es una API de REST?, Recuperado de https://www.redhat.com/es/topics/api/what-is-a-rest-api)

Base de datos: Se llama base de datos, o también banco de datos, a un conjunto de información perteneciente a un mismo contexto, ordenada de modo sistemático para su posterior recuperación, análisis y/o transmisión. Existen actualmente muchas formas de bases de datos, que van desde una biblioteca hasta los vastos conjuntos de datos de usuarios de una empresa de telecomunicaciones. (Raffino M. , 2019).

Modelo Relacional: El modelo relacional es un modelo de datos basado en la lógica de predicados

y en la teoría de conjuntos el cual fue introducido en la década de 1970 por Edgar Frank Codd. (Muñoz Gomez, y otros, 2012).

MySQL: MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: licencia pública general/licencia comercial por Oracle Corporation. Está considerada la base de datos de código abierto más popular del mundo y una de las más populares en general, junto a Oracle y Microsoft SQL Server, todas para entornos de desarrollo web. (MySQL . (2025, abril 24). En Wikipedia. Recuperado de https://es.wikipedia.org/wiki/MySQL).

Git: Sistema distribuido de control de versiones creado por Linus Torvald en 2005. Está incluido por defecto en sistemas operativos basados en Unix (Linux y MacOs). Sus principales ventajas son la atomicidad de cambios, rendimiento y seguridad (Somasundaram, 2013).

Docker: Versión virtual de la infraestructura de una computadora que fue optimizada para obtener mejor rendimiento de memoria y procesamiento que la de una máquina virtual. Disminuyendo costos. (Krishan, 2016).

VPN: Una red privada virtual (RPV) (en inglés, virtual private network, VPN) es una tecnología de red de ordenadores que permite una extensión segura de la red de área local (LAN) sobre una red pública o no controlada como Internet. (*Red privada virtual .* (2025, abril 24). En Wikipedia. Recuperado de https://es.wikipedia.org/wiki/Red_privada_virtual).

Competencia

(Por completar)

Relevamiento

Relevamiento estructural y tecnológico

(Por completar)

Procesos de negocio

Derivado de los procesos definidos en el alcance del presente proyecto y como resultado de la aplicación de diversas técnicas de elicitación, se expone a continuación una descripción detallada del funcionamiento actual de los procesos de negocio, incluyendo su dinámica operativa y los

elementos involucrados.

Proceso: Proceso de registro y actualización de los datos personales de clientes		
Subproceso: Alta de cliente		
Roles: Operador Atención al Público, Sistema y Cliente.		
Rol	Orden	Descripción
Operador Atención al Público	1	Selecciona dar de alta un nuevo cliente, ingresa en la opción Alta de clientes. Ingresa el número de DNI.
Sistema	2	Valida si el DNI ingresado ya se encuentra registrado en la base de datos. Si el cliente existe, muestra un mensaje indicando que ya está registrado y regresa a la pantalla de créditos.
		Si el cliente no existe, muestra un formulario vacío solicitando la siguiente información: nombre y apellido, domicilio (calle y número), teléfono particular y laboral, fecha de nacimiento, estado civil, nacionalidad, código postal, sexo, CUIL, número de caja de ahorro y escalafón (en caso de empleados públicos).
Operador Atención al Público	3	Solicita verbalmente al cliente todos los datos requeridos, indicando que son de carácter obligatorio.
Cliente	4	Proporciona toda la información solicitada.
Operador Atención al Público	5	Carga los datos provistos en el formulario del sistema, verifica su correcta entrada, confirma y procede a grabar la información en la base de datos.
Sistema	6	Registra la información en la base de dato. (Clientes.dbf).

Proceso: Proceso de registro y actualización de los datos personales de clientes		
Subproceso: Actualización de datos de cliente		
Roles: Operador Atención al Público, Sistema y Cliente.		
Rol	Orden	Descripción
Operador Atención al Público	1	Selecciona la opción de modificar datos de cliente. Ingresa el número de DNI.
Sistema	2	El sistema realiza la búsqueda en la base de clientes según el DNI ingresado. Si no encuentra una coincidencia muestra un mensaje de sistema y sale del proceso. Caso contrario selecciona al registro del cliente que corresponde al DNI ingresado.
Operador Atención al Público	3	Selecciona el registro encontrado.
Sistema	4	Muestra los datos actuales del cliente en un formulario editable.
Operador Atención al Público	5	Consulta al cliente qué datos necesita actualizar.
Cliente	6	Proporciona la nueva información (por ejemplo, cambio de domicilio o teléfono).
Operador de Atención al Público	7	Modifica los datos en el sistema, asegurándose de completar correctamente los campos actualizados.
Operador de Atención al Público	8	Confirma la modificación y guarda los cambios en el sistema.
Sistema	9	Registra la actualización en la base de datos (Clientes.dbf).

Proceso: Proceso de gestión de préstamos		
Subproceso: Alta de un Nuevo Préstamo		
Roles: Operador Atención al Público, Sistema y Cliente.		
Rol	Orden	Descripción
Cliente	1	Solicita formalmente iniciar un trámite de préstamo en la sucursal.
Operador Atención al Público	2	Solicita el número de DNI para localizar al cliente en el sistema.
Sistema	3	Busca el registro del cliente y muestra sus datos personales, sino existe nuestra un mensaje.
Operador Atención al Público	4	Solicita al cliente el monto solicitado y el plazo de devolución pretendido.
Cliente	5	Informa el monto y plazo deseados.
Sistema	6	Calcula automáticamente la cuota estimada aplicando tasas y condiciones vigentes.
Operador Atención al Público	7	Informa al cliente la cuota estimada y condiciones del préstamo, y consulta si desea avanzar.
Cliente	8	Acepta continuar con el trámite del préstamo. Si no acepta informa otro monto y plazo.
Operador Atención al Público	9	Inicia el proceso de evaluación crediticia, verificando: antecedentes de pagos anteriores, situación de morosidad interna, cumplimiento de documentación requerida (DNI, recibos de sueldo, entre otros). Si no es apto se informa al cliente y cancela el proceso.
Operador de Atención al Público	10	Procede a aprobar el préstamo.
Sistema	11	Registra la aprobación formal del préstamo en el sistema y genera el contrato de préstamo.

Cliente	12	Firma contrato.
Operador de Atención al Público	13	Entrega al cliente la documentación correspondiente y libera los fondos a la cuenta del cliente.

Gráfica de procesos de negocios

Proceso: Alta de un Nuevo Préstamo

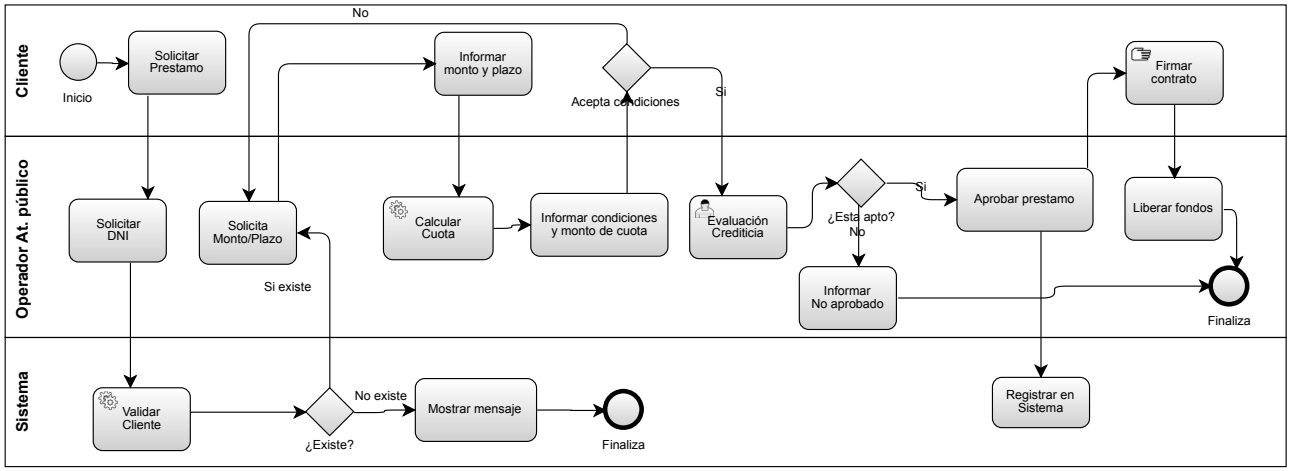


Ilustración 1: Proceso: Alta de un Nuevo Préstamo

Proceso: Alta de cliente

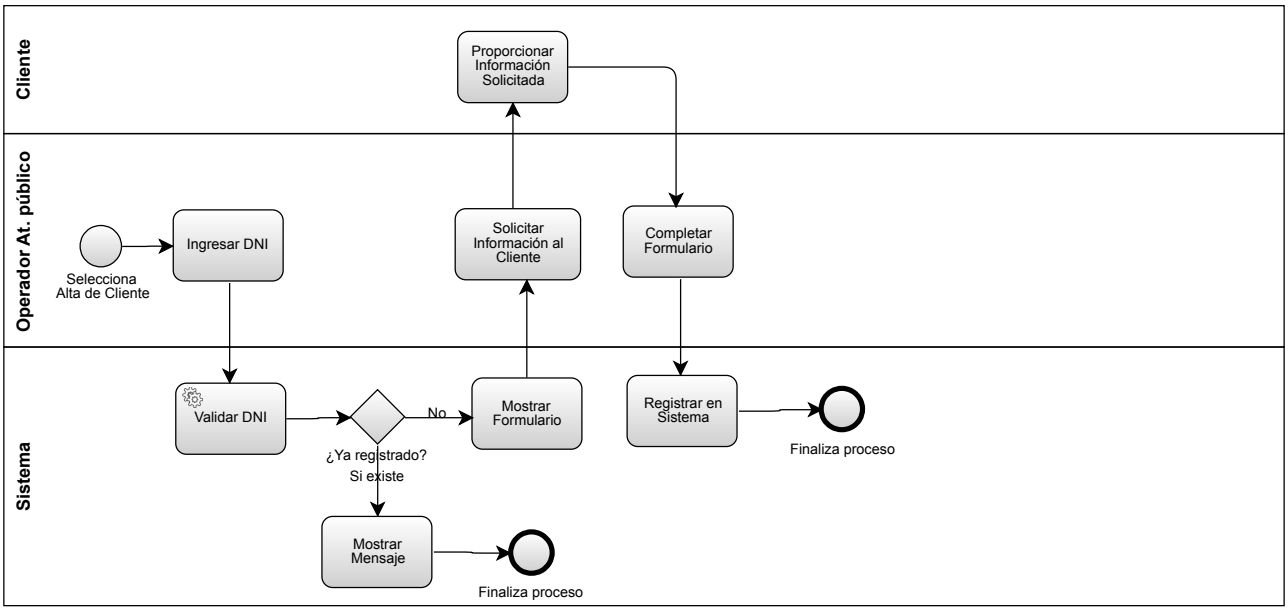


Ilustración 2: Proceso: Alta de cliente

Proceso : Actualización de datos de cliente

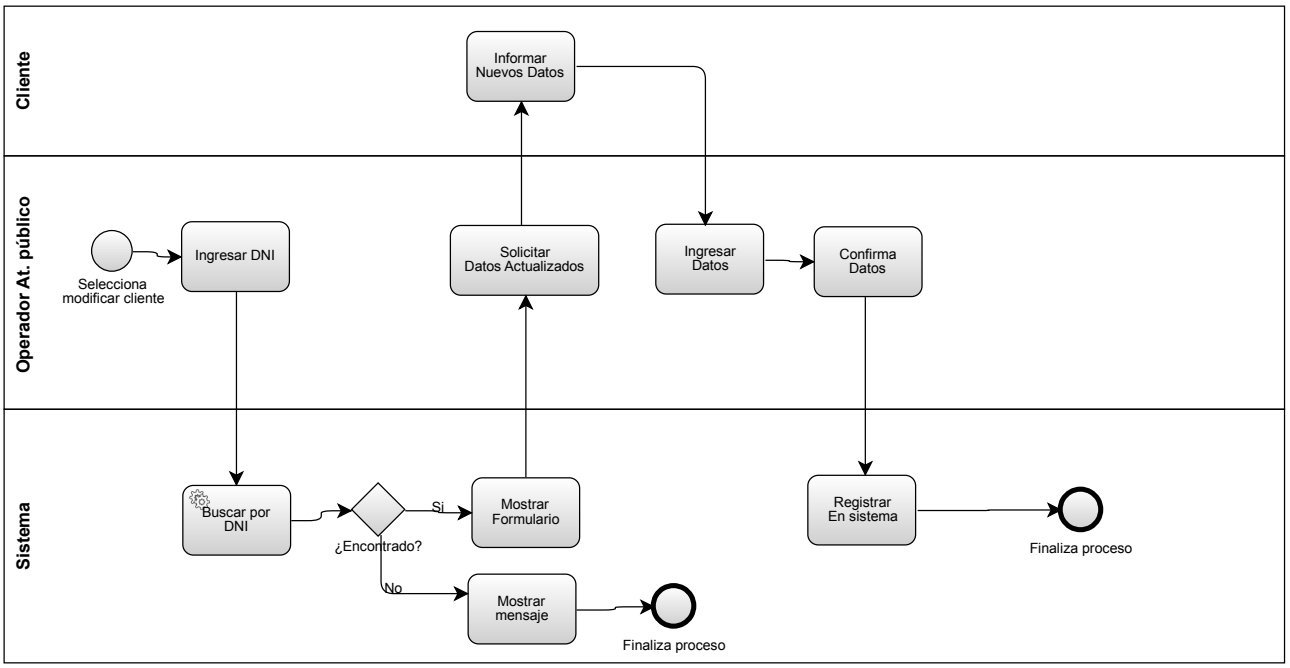


Ilustración 3: Proceso : Actualización de datos de cliente

Diagrama de Dominio

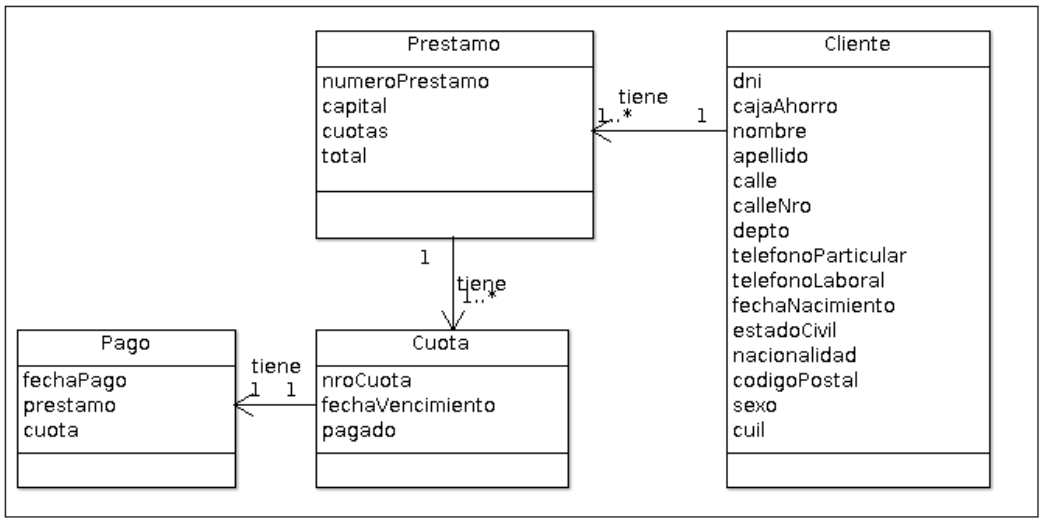


Ilustración 4: Diagrama de Dominio

Diagnostico

Problemas de alcance general

Al analizar los procesos de negocio mencionados anteriormente, es posible enumerar de manera general los problemas que afectan a cada uno de ellos.

Problemas:

Falta de Integración y Centralización de la Información

Impacto en el negocio: La falta de un sistema integrado y centralizado genera problemas de duplicación de datos, errores en la actualización de información y desconexión entre sucursales. Esto afecta la eficiencia operativa, incrementando el tiempo necesario para procesar solicitudes, y aumenta el riesgo de inconsistencias que pueden perjudicar tanto la relación con los clientes como la toma de decisiones estratégicas.

Causa: El sistema actual no está diseñado para operar en red ni para compartir información de manera centralizada, lo que provoca que cada sucursal gestione los datos de manera aislada. Esto también dificulta la generación de reportes consolidados y la planificación a nivel organizacional.

Retrasos en la Atención al Cliente y Gestión Administrativa

Impacto en el negocio: Los tiempos de atención al cliente y la gestión administrativa se ven

retrasados debido a los procesos manuales y la falta de integración del sistema. Esto afecta negativamente la satisfacción del cliente y eficiencia operativa.

Causa: La tecnología desactualizada no permite optimizar los tiempos de proceso ni facilita la gestión ágil de los préstamos. Además, los procesos manuales en la entrada de datos y generación de informes dificultan la agilidad operativa.

Falta de control eficiente sobre la evaluación crediticia.

Impacto en el negocio: Se otorgan préstamos a clientes con riesgo crediticio no detectado, aumentando la probabilidad de mora y afectando la rentabilidad de la cartera de créditos.

Causas identificadas:

El sistema no incorpora alertas automáticas sobre antecedentes de morosidad interna.

No existe integración con bases externas (como Veraz o BCRA) para verificar la situación crediticia del solicitante.

Falta de Herramientas de Auditoría y Control Interno

Impacto en el negocio: La ausencia de mecanismos eficaces de auditoría y control interno incrementa la probabilidad de fraude y errores operativos, afectando la confiabilidad y seguridad de las operaciones. Además, la falta de visibilidad sobre las transacciones diarias reduce la capacidad de respuesta ante incidentes o irregularidades.

Causa: El sistema actual no proporciona herramientas ni funcionalidades para el seguimiento en tiempo real de las actividades, lo que dificulta la detección de problemas o irregularidades. Esto genera vulnerabilidades en el sistema operativo y pone en riesgo la integridad de los procesos financieros.

Bajos Niveles de Seguridad en el Acceso y Respaldo de Datos

Impacto en el negocio: El sistema actual no tiene mecanismos adecuados de respaldo de datos ni de seguridad de acceso, lo que aumenta el riesgo de pérdida de datos y acceso no autorizado a información sensible. Esta falta de seguridad puede comprometer la confidencialidad de los datos, afectando la confianza de los clientes y exponiendo a la empresa a sanciones legales.

Causa: No existen sistemas de respaldo automatizados ni políticas claras de control de acceso a la información, lo que pone en riesgo la confidencialidad y disponibilidad de los datos.

Propuesta de solución

Propuesta funcional

Con el objetivo de resolver los problemas identificados en el diagnóstico, se propone el desarrollo de un sistema de gestión integral basado en una arquitectura cliente-servidor que permita:

1. **Centralización de la Información**

   El nuevo sistema centralizará toda la información relacionada con los clientes, los préstamos y las operaciones, permitiendo su acceso y actualización en tiempo real desde cualquier sucursal o terminal. La base de datos centralizada garantizará que todos los usuarios trabajen con la misma información actualizada y correcta, eliminando la desconexión entre sucursales.

2. **Automatización de Procesos**

   El sistema eliminará todos los procesos manuales existentes, como la introducción y validación de datos, la generación de informes y el seguimiento de pagos. Todos estos procesos serán automatizados, reduciendo el riesgo de errores humanos y aumentando la eficiencia operativa. Los usuarios podrán gestionar los préstamos, pagos y datos de los clientes de manera ágil y sin necesidad de intervención manual.

3. **Mejoras en la Seguridad y Control de Datos**

   Se implementarán mecanismos de respaldo automatizado de los datos y múltiples niveles de control de acceso para garantizar la seguridad de la información. Esto reducirá significativamente los riesgos de pérdida de datos o acceso no autorizado. Los datos estarán encriptados y el sistema contará con mecanismos de auditoría para un control más riguroso de las operaciones realizadas.

4. **Optimización en la Atención al Cliente**

   La centralización y automatización permitirán mejorar la respuesta y tiempos de atención al cliente. Los operadores podrán acceder a la información de los clientes de manera más rápida y precisa, lo que mejorará la experiencia del cliente y reducirá los tiempos de espera.

5. **Sistema Escalable y Flexible**

   El sistema estará diseñado para ser escalable, lo que permitirá su expansión conforme a las

necesidades futuras de la empresa. La tecnología utilizada (Java para la aplicación y MySQL como base de datos) permite la integración de nuevos módulos o funcionalidades sin afectar la operatividad del sistema.

6. **Herramientas de Auditoría y Control Interno**

   Se incorporarán herramientas avanzadas de auditoría y seguimiento de operaciones. Esto permitirá a los gerentes y supervisores monitorear las actividades del sistema, garantizando un alto nivel de control interno y reduciendo el riesgo de fraudes o errores operativos.

7. **Acceso Remoto Seguro**

   El sistema permitirá acceso remoto seguro para que los usuarios autorizados puedan conectarse desde diferentes ubicaciones de manera controlada y segura. Esto facilitará el trabajo remoto o en sucursales fuera de la sede central sin comprometer la seguridad de los datos.

Listado de Requerimientos funcionales

ID	Nombre
RF001	El sistema debe permitir el ingreso mediante usuario y contraseña
RF002	El sistema debe permitir el registrar clientes.
RF003	El sistema debe permitir buscar clientes.
RF004	El sistema debe permitir modificar datos de clientes.
RF005	El sistema debe permitir la creación de un nuevo préstamos.
RF006	El sistema debe permitir la modificación de las condiciones de un préstamo.
RF007	El sistema debe permitir el seguimiento del estado de un préstamo.
RF008	El sistema debe ofrecer una funcionalidad de simulación de escenarios de financiamiento.
RF009	El sistema debe permitir la cancelación de un préstamo.
RF010	El sistema debe permitir aprobar un préstamo.
RF011	El sistema debe permitir la creación de esquemas de financiación.

RF012	El sistema debe permitir la modificación de esquemas de financiación.
RF013	El sistema debe permitir el registro de pagos de cuotas.
RF014	El sistema debe actualizar el saldo pendiente de cada préstamo después de cada pago registrado.
RF015	El sistema debe permitir la consulta de los estados de cuenta por parte de los clientes.
RF016	El sistema debe permitir el registrar un usuario.
RF017	El sistema debe permitir modificar un usuario.
RF018	El sistema debe permitir la creación de roles.
RF019	El sistema debe permitir la asignación de roles a un usuario.
RF021	El sistema debe permitir consultar listado de usuarios.
RF022	El sistema debe permitir consultar roles disponibles.
RF023	El sistema debe mostrar mensajes claros para éxito o error en la autenticación.

Listado de Requerimientos no funcionales

ID	Nombre	Tipo
RNF01	La interfaz de usuario debe ser intuitiva y fácil de usar, permitiendo a los operadores completar tareas comunes (registro de cliente, creación de préstamos, registro de pagos) sin capacitación extensiva.	Usabilidad
RNF02	Los mensajes de error deberán ser lo suficientemente claros para identificar puntualmente el inconveniente.	Usabilidad
RNF03	El sistema debe funcionar a través de una red privada virtual	Rendimiento
RNF04	La aplicación debe utilizar los recursos del sistema de manera óptima, como el consumo de memoria y procesamiento.	Rendimiento
RNF05	El sistema debe realizar copias de seguridad automáticas diarias de la	Respaldo y

	base de datos, almacenando las copias en un lugar seguro y accesible.	Recuperación
RNF06	El sistema debe permitir la recuperación rápida de datos en caso de pérdida, con un tiempo máximo de recuperación de 4 horas.	Respaldo y Recuperación
RNF07	El sistema debe contar con una base de datos MySQL.	Implementación
RNF08	El sistema debe estar desarrollado en Java.	Implementación
RNF09	El sistema operativo debe ser Linux	Portabilidad

Listado de Requerimientos candidatos

ID	Nombre
RCF01	El sistema debe integrarse con los WebServices del BCRA para consultar el estado crediticio de los clientes.
RCF01	El sistema debe integrarse con API de Veraz para consultar el estado crediticio de los clientes.

Desarrollo del prototipo

Diagrama de casos de uso

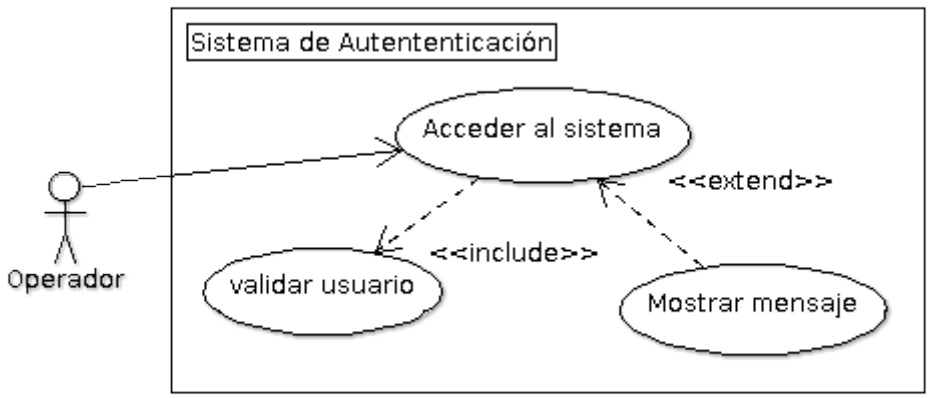


Ilustración 5: Diagrama de caso de uso: Acceder al sistema

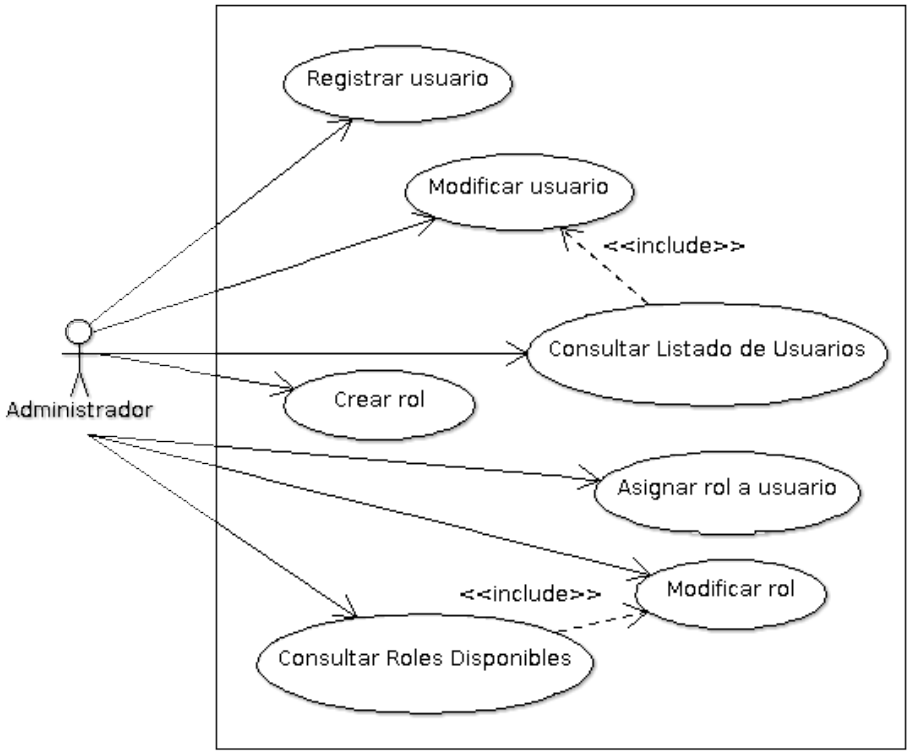


Ilustración 6: Diagrama de caso de uso: Gestionar usuarios

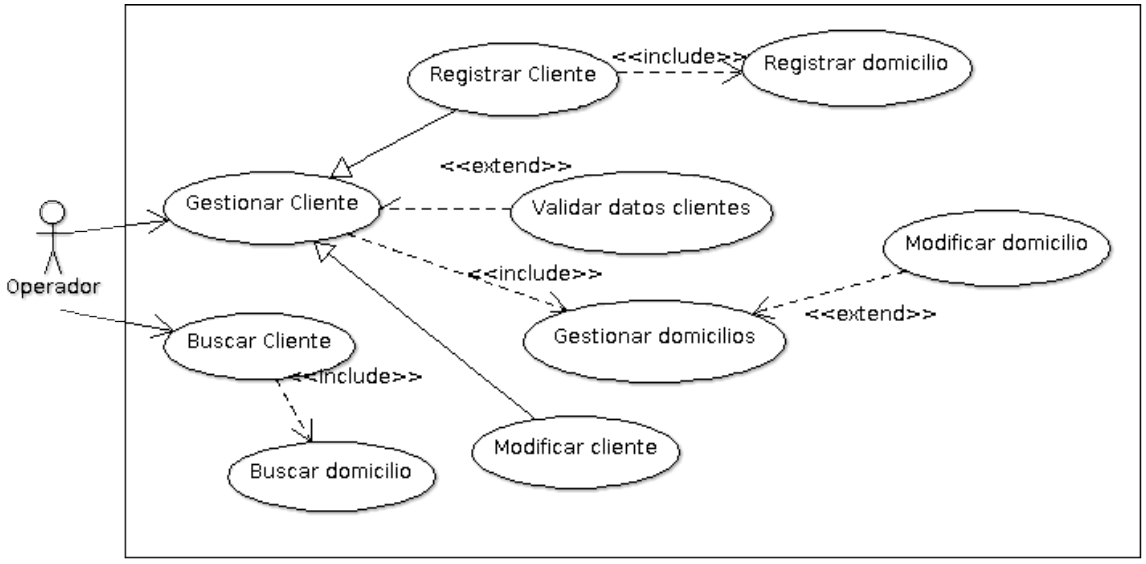


Ilustración 7: Diagrama de caso de uso: Gestión de clientes

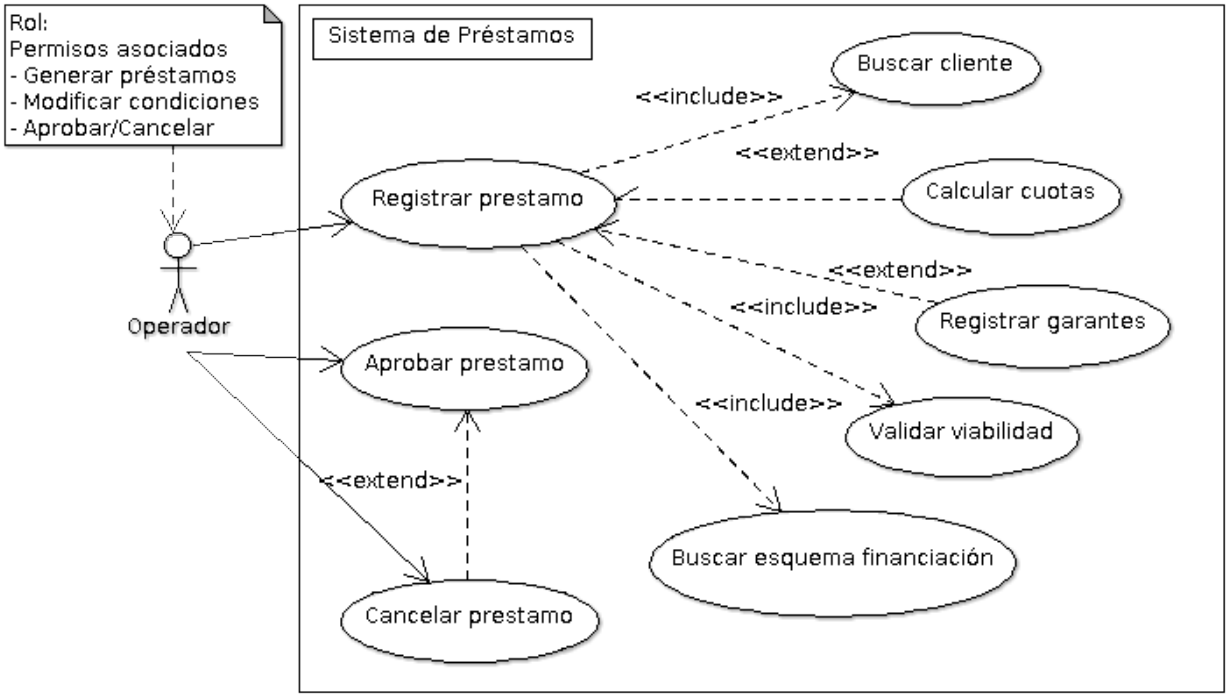


Ilustración 8: Diagrama de caso de uso: Gestionar préstamos

Identificación de autores

Operador de Atención al Público: Usuario del sistema que está autorizada a gestionar clientes y prestamos.

Administrador: Usuario del sistema que está autorizada a gestionar usuarios y parámetros del sistema.

Trazabilidad

CU001 Acceder al sistema

CU002 Registrar usuarios

CU003 Gestionar clientes

CU004 Registrar Préstamos

Requerimiento	Caso de uso	Actor principal	Paquete de análisis	Comentario
RF001,RF023	CU001	Operador	Seguridad	Acceder al sistema

RF016	CU002-01	Administrador	Gestión de Usuarios	Registrar usuario
RF017	CU002-02	Administrador	Gestión de Usuarios	Modificar usuario
RF018	CU002-03	Administrador	Gestión de Usuarios	Crear Rol
RF019	CU002-04	Administrador	Gestión de Usuarios	Asignar roles
RF021	CU002-05	Administrador	Gestión de Usuarios	Consultar listado de Usuarios
RF022	CU002-05	Administrador	Gestión de Usuarios	Consultar listado de Roles
RF002	CU003-01	Operador	Gestión de Clientes	Registrar Cliente
RF003	CU003-02	Operador	Gestión de Clientes	Buscar Clientes
RF004	CU003-03	Operador	Gestión de Clientes	Modificar datos de Clientes
RF005	CU004	Operador	Gestión de Préstamos	Registrar Préstamo
RF009	CU004	Operador	Gestión de Préstamos	Cancelar Préstamo
RF010	CU004	Operador	Gestión de Préstamos	Aprobar Préstamo

Descripción de casos de uso

Nombre del Caso de Uso:	**CU001 - Acceder al sistema**	
Actor Principal:	Operador	
Referencias:	RF001,RF023	
Descripción General:	Este caso de uso describe el proceso mediante el cual un operador accede al sistema mediante la validación de sus credenciales de usuario.	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema.	
Flujo Principal:	1	El operador solicita acceso al sistema.
	2	El sistema verifica las credenciales proporcionadas (nombre de usuario y contraseña).
	3	Si las credenciales son válidas, el sistema permite el acceso.
	4	El sistema muestra un mensaje de confirmación de acceso exitoso.
	5	El caso de uso termina.
Postcondición:	El usuario ha sido autenticado correctamente en el sistema.	

Flujo alternativo:		Fallo de validación de datos.
		El sistema muestra un mensaje de error
		El sistema regresa al paso 2 para que el operador corrija los errores.
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU002-01 – Registrar Usuario**	
Actor Principal:	Administrador	
Referencias:	RF016	
Descripción General:	Crear un nuevo usuario en el sistema con credenciales y roles asignados.	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para registrar un nuevo usuario.	
Flujo Principal:	1	El administrador selecciona la opción "Crear Usuario" en el menú de la aplicación.
	2	El sistema muestra un formulario con campos obligatorios: 1. Nombre de usuario (único). 2. Contraseña (cumple políticas de seguridad). 3. Rol asignado (selección de lista desplegable).
	3	El administrador completa los datos y confirma.
	4	El sistema valida que el nombre de usuario no exista y que la contraseña sea válida.
	5	Si es correcto, registra al usuario en la base de datos y muestra mensaje de éxito. La contraseñas se guarda encriptada (bcrypt).
Postcondición:	El nuevo usuario queda registrado y puede acceder al sistema según su rol.	
Flujo alternativo:		**Usuario ya existe:** El sistema notifica y solicita otro nombre de usuario.
		Contraseña inválida: El sistema indica los requisitos no cumplidos (ej.: longitud mínima).
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU002-02 – Modificar Usuario**	
Actor Principal:	Administrador	
Referencias:	RF017	
Descripción General:	Actualizar datos de un usuario existente (ej.: contraseña, estado, rol).	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para Modificar Usuario.	
Flujo Principal:	1	El administrador selecciona la opción "Modificar Usuario" y busca al usuario por nombre.
	2	El sistema muestra los datos actuales del usuario en un formulario editable.
	3	El administrador realiza los cambios (ej.: restablece contraseña, cambia rol).
	4	El sistema valida los datos y actualiza el registro en la base de datos.
	5	Muestra confirmación de la modificación.
Postcondición:	Los datos del usuario quedan actualizados en el sistema.	
Flujo alternativo:		**Usuario no encontrado**: El sistema muestra error y sugiere verificar el criterio de búsqueda.
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU002-03 – Crear Rol**	
Actor Principal:	Administrador	
Referencias:	RF018	
Descripción General:	Definir un nuevo rol con permisos específicos (ej.: "Operador", "Gerente").	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para Crear Rol.	
Flujo Principal:	1	El administrador selecciona "Crear Rol" en el módulo de gestión.
	2	El sistema solicita: • Nombre del rol (ej.: "Analista Crediticio"). • Permisos asociados (selección múltiple: "Aprobar préstamos", "Consultar reportes").
	3	El administrador confirma.
	4	El sistema valida que el nombre del rol no exista y lo registra.
Postcondición:	El nuevo rol está disponible para asignar a usuarios.	
Flujo alternativo:		**Rol duplicado:** El sistema notifica y solicita otro nombre.
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU002-04 – Asignar Roles a Usuario**	
Actor Principal:	Administrador	
Referencias:	RF019	
Descripción General:	Vincular un usuario existente con un rol definido.	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para Asignar Roles a Usuario.	
Flujo Principal:	1	El administrador selecciona "Asignar Roles" y elige un usuario de la lista.
	2	El sistema muestra los roles disponibles.
	3	El administrador selecciona uno o más roles y confirma.
	4	El sistema actualiza los permisos del usuario y notifica el éxito.
Postcondición:	El usuario hereda los permisos asociados al rol asignado.	
Flujo alternativo:		**Usuario sin roles asignables:** El sistema sugiere crear un rol primero.
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU002-05 – Consultar Listado de Usuarios**	
Actor Principal:	Administrador	
Referencias:	RF021	
Descripción General:	Visualizar todos los usuarios registrados con sus datos básicos y roles.	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para Consultar Listado de Usuarios.	
Flujo Principal:	1	El administrador selecciona "Consultar Usuarios".
	2	El sistema muestra una tabla con: Usuario, estado (activo/inactivo), rol asignado.
	3	Opciones de filtrado por rol o búsqueda por nombre.
Postcondición:	El administrador obtiene información actualizada para tomar decisiones (ej.: desactivar usuarios).	
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU002-06 – Consultar Listado de Roles**	
Actor Principal:	Administrador	
Referencias:	RF022	
Descripción General:	Revisar los roles existentes y sus permisos asociados.	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para Consultar Listado de Roles.	
Flujo Principal:	1	El administrador selecciona "Consultar Roles".
	2	El sistema muestra una lista con: • Nombre del rol. • Permisos asociados (ej.: "Gestionar préstamos", "Auditar operaciones").
	3	Opciones de filtrado por rol o búsqueda por nombre.
Postcondición:	El administrador identifica posibles ajustes en la configuración de permisos.	
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU003-01 - Registrar Cliente**	
Actor Principal:	Operador de Atención al Público	
Referencias:	RF002	
Descripción General:	Este caso de uso describe el proceso mediante el cual un operador accede al sistema mediante la validación de sus credenciales de usuario. El usuario debe tener permisos necesarios para Registrar Cliente.	
Precondición	Dar de alta a un nuevo cliente en el sistema con sus datos personales y financieros.	
Flujo Principal:	1	El operador selecciona "Alta de Cliente" en el sistema.
	2	El sistema solicita el número de DNI del cliente.
	3	**Validación:** Si el DNI ya existe, el sistema muestra un mensaje: "El cliente ya está registrado" y vuelve al menú principal. Si el DNI **no existe**, el sistema despliega un formulario con campos obligatorios: • Nombre y apellido.

		• Domicilio Particular (calle, número, código postal). • Domicilio Laboral (calle, número, código postal). • Teléfonos particular y laboral. • Fecha de nacimiento, genero. • E-mail
	4	El operador solicita verbalmente los datos al cliente y los ingresa en el sistema.
	5	El operador verifica que los datos estén completos y correctos.
	6	Confirma el registro.
Postcondición:	El cliente queda registrado en el sistema y puede solicitar préstamos.	
Flujo alternativo:		**Campos incompletos:** El sistema no permite guardar hasta que todos los campos obligatorios estén completos.
		Error en datos: El cliente corrige la información si el operador detecta inconsistencias (ej.: teléfono inválido).
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU003-02 – Buscar Cliente**	
Actor Principal:	Operador de Atención al Público	
Referencias:	RF003	
Descripción General:	Localizar un cliente registrado para consultar o modificar sus datos, o iniciar un préstamo.	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para Buscar Cliente.	
Flujo Principal:	1	El operador selecciona "Buscar Cliente"
	2	El sistema permite buscar por: • DNI (opción principal). • Nombre y apellido (búsqueda parcial).
	3	Si el cliente **existe**, el sistema muestra sus datos completos.
	4	Si **no existe**, muestra un mensaje: *"No se encontraron coincidencias"*.
	5	El operador puede seleccionar entre: • Modificar datos (CU003-03). • Iniciar un préstamo (CU004).
Flujo alternativo:	**Múltiples coincidencias:** El sistema lista posibles clientes para selección manual.	
Postcondición:	El operador accede a la ficha del cliente para gestionar trámites.	
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU003-03 – Modificar Cliente**	
Actor Principal:	Operador de Atención al Público	
Referencias:	RF004	
Descripción General:	Actualizar datos personales o financieros de un cliente existente (ej.: cambio de domicilio, teléfono).	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para Modificar Cliente.	
Flujo Principal:	1	El operador selecciona "Modificar Cliente"
	2	El sistema permite buscar por: • DNI (opción principal). • Nombre y apellido (búsqueda parcial).
	3	El sistema valida que el cliente exista. Si no, muestra error.
	4	Muestra el formulario con los datos actuales en modo editable.
	5	El operador consulta al cliente qué datos actualizar (ej.: nuevo teléfono).
	6	El operador guarda los cambios.
	7	El sistema valida y actualiza la base de datos
	8	Muestra mensaje: *"Datos actualizados correctamente"*.
Flujo alternativo:	DNI / CUIT no encontrado: El sistema sugiere verificar el número o registrar al cliente (CU003-01).	
Postcondición:	Los datos del cliente quedan actualizados en el sistema.	
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU004-01 – Registrar Préstamo**	
Actor Principal:	Operador de Atención al Público	
Referencias:	RF005	
Descripción General:	Iniciar el trámite de un nuevo préstamo para un cliente registrado, ingresando los datos básicos del préstamo (monto, plazo, condiciones).	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para Registrar Préstamo.	
Flujo Principal:	1	El operador selecciona "Registrar Préstamo" en el sistema.
	2	El sistema solicita el DNI del cliente.
	3	Si el cliente no existe, muestra un mensaje de error y sugiere registrarlo primero (CU003-01).
	4	Si el cliente **existe**, el sistema muestra sus datos y préstamos activos (si los tiene).
	5	El operador solicita al cliente: • **Monto solicitado.** • **Plazo de devolución** (en meses). • **Selecciona un esquema de financiación.**
	6	El sistema calcula automáticamente: • **Cuota estimada** (según esquema). • **Total a pagar** (capital + intereses).
	7	El operador realiza la validación de viabilidad del préstamo: • El operador verifica: ◦ Historial crediticio del cliente. ◦ Documentación requerida (DNI, recibos de sueldo, etc.). • Si el cliente no es apto, se cancela el proceso y se informa al cliente.
	8	Generación del Préstamo: • Si el cliente acepta las condiciones, el operador confirma el préstamo. • El sistema genera un número de préstamo único y lo registra en estado "Pendiente de Aprobación".
	9	El operador confirma el préstamo y el sistema muestra un

		mensaje: "Préstamo registrado exitosamente. Esperando aprobación."
Postcondición:	-El préstamo queda registrado en estado "Pendiente de Aprobación". -El cliente recibe un comprobante de solicitud.	
Flujo alternativo:		**Cliente rechaza condiciones:** El operador puede ajustar monto/plazo y recalcular.
		Falta documentación: El sistema notifica qué documentos faltan y pausa el trámite.
Excepciones:	No contempla	

Nombre del Caso de Uso:	**CU004-02 – Aprobar Préstamo**	
Actor Principal:	Operador de Atención al Público	
Referencias:	RF010	
Descripción General:	Validar y autorizar un préstamo previamente registrado, permitiendo su desembolso.	
Precondición	El sistema debe estar operativo y el operador debe contar con credenciales de acceso (nombre de usuario y contraseña) previamente registradas en el sistema. El usuario debe tener permisos necesarios para Aprobar Préstamo.	
Flujo Principal:	1	El operador selecciona "Aprobar Préstamo" y busca el préstamo por: • Número de préstamo. • DNI del cliente.
	2	El sistema muestra: • Datos del cliente. • Monto, plazo, cuotas calculadas. • Historial crediticio (si tiene préstamos anteriores).
	3	El operador realiza una validación de la información préstamo. Si no cumple, rechaza el préstamo y registra el motivo.
	4	Si todo es correcto, se confirma la aprobación. El sistema: Cambia el estado a "Aprobado". Genera el contrato digital con condiciones legales.
	5	El sistema registra la firma y libera el desembolso a la cuenta del cliente.
Flujo alternativo:	Rechazo del préstamo: El sistema registra el motivo (ej.: "Historial de morosidad") y notifica al cliente.	
Postcondición:	El préstamo pasa a estado "Activo". Las cuotas se generan automáticamente en el sistema.	
Excepciones:	No contempla	

Diagramas de Secuencia

Acceder al Sistema (CU001)

Actores: Operador, Sistema, Base de Datos.

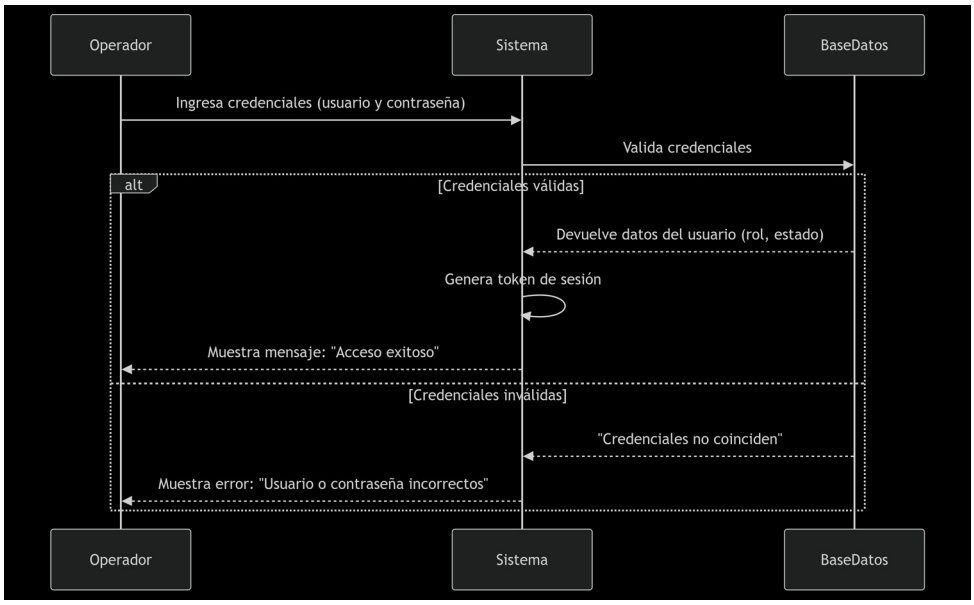


Ilustración 9: Diagrama se secuencia: Acceder al Sistema (CU001)

Registrar Usuario (CU002-01)

Actores: Administrador, Sistema, Base de Datos.

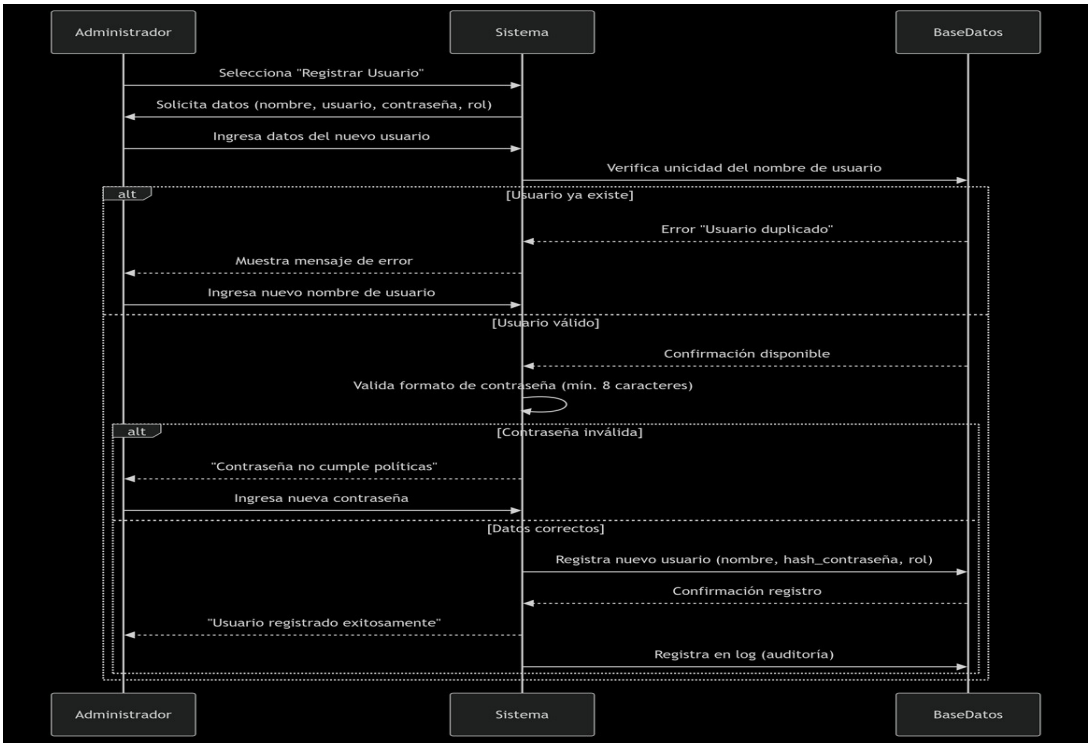


Ilustración 10: Diagrama de secuencia: Registrar Usuario (CU002-01)

Registrar Cliente (CU003-01)

Actores: Operador, Sistema, Base de Datos, Cliente.

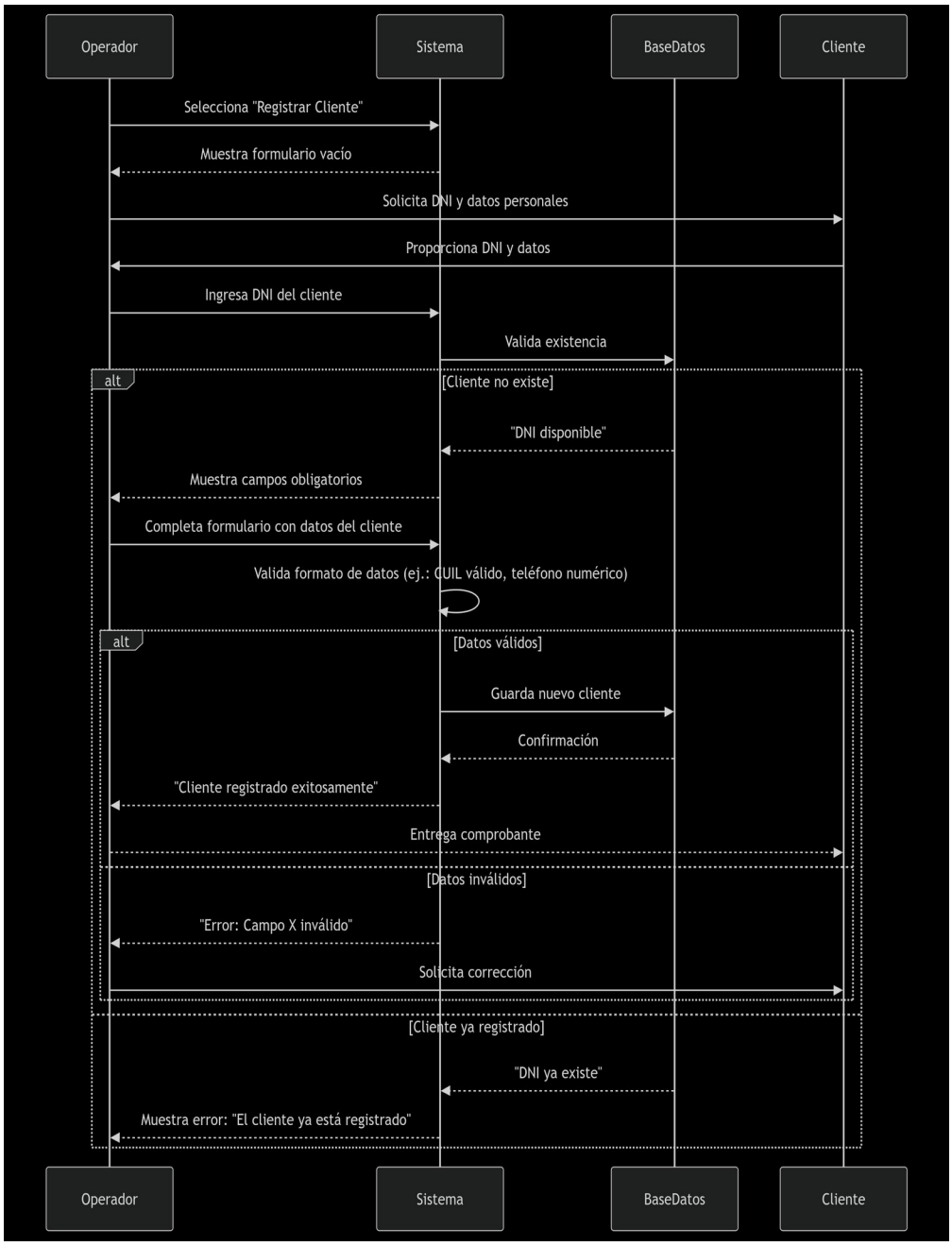


Ilustración 11: Diagrama de secuencia: Registrar Cliente (CU003-01)

Registrar Préstamo (CU004-01)

Actores: Cliente, Operador, Sistema, Base de Datos.

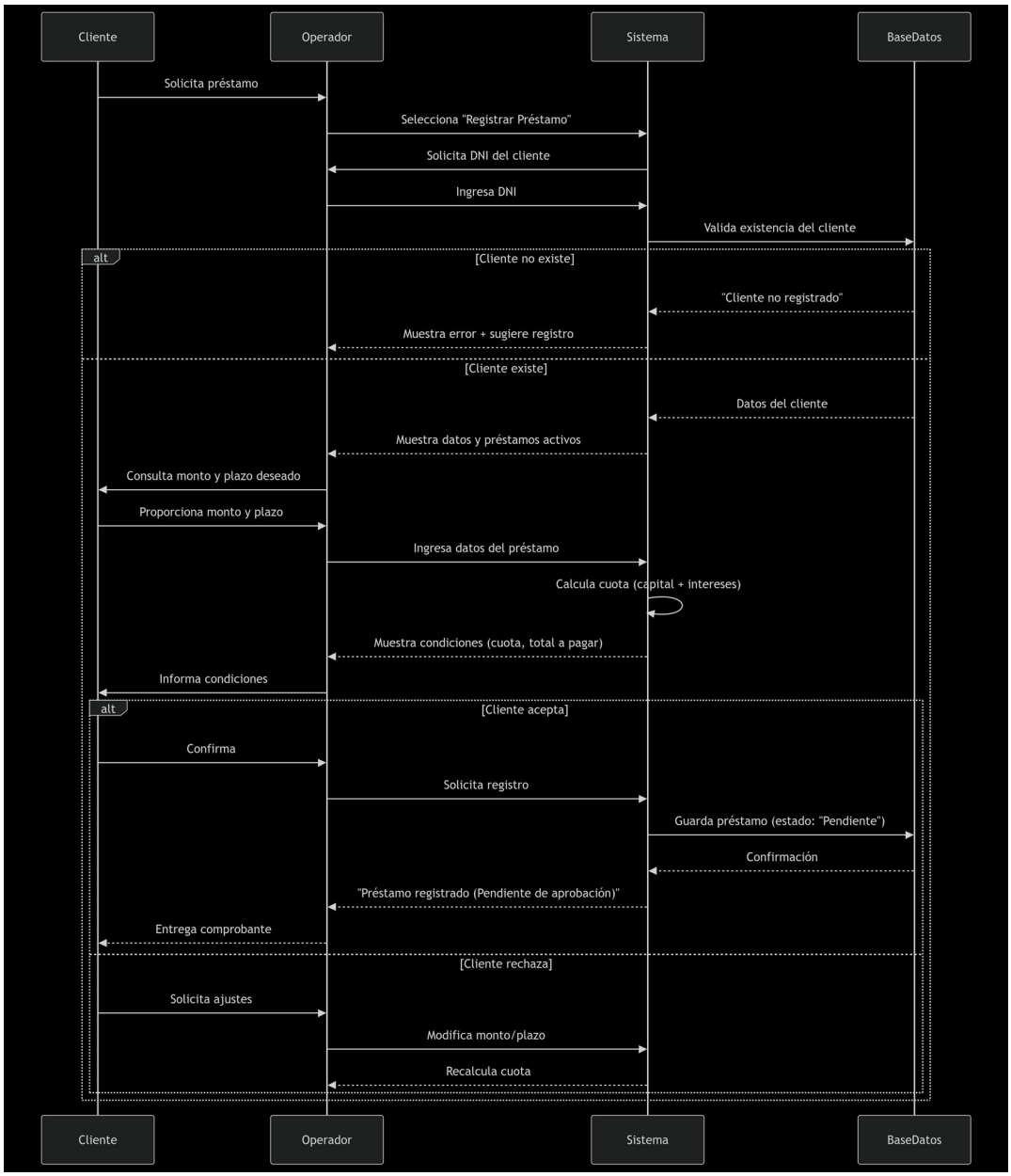


Ilustración 12: Diagrama de secuencia: Registrar Préstamo (CU004-01)

Diagrama de Clases

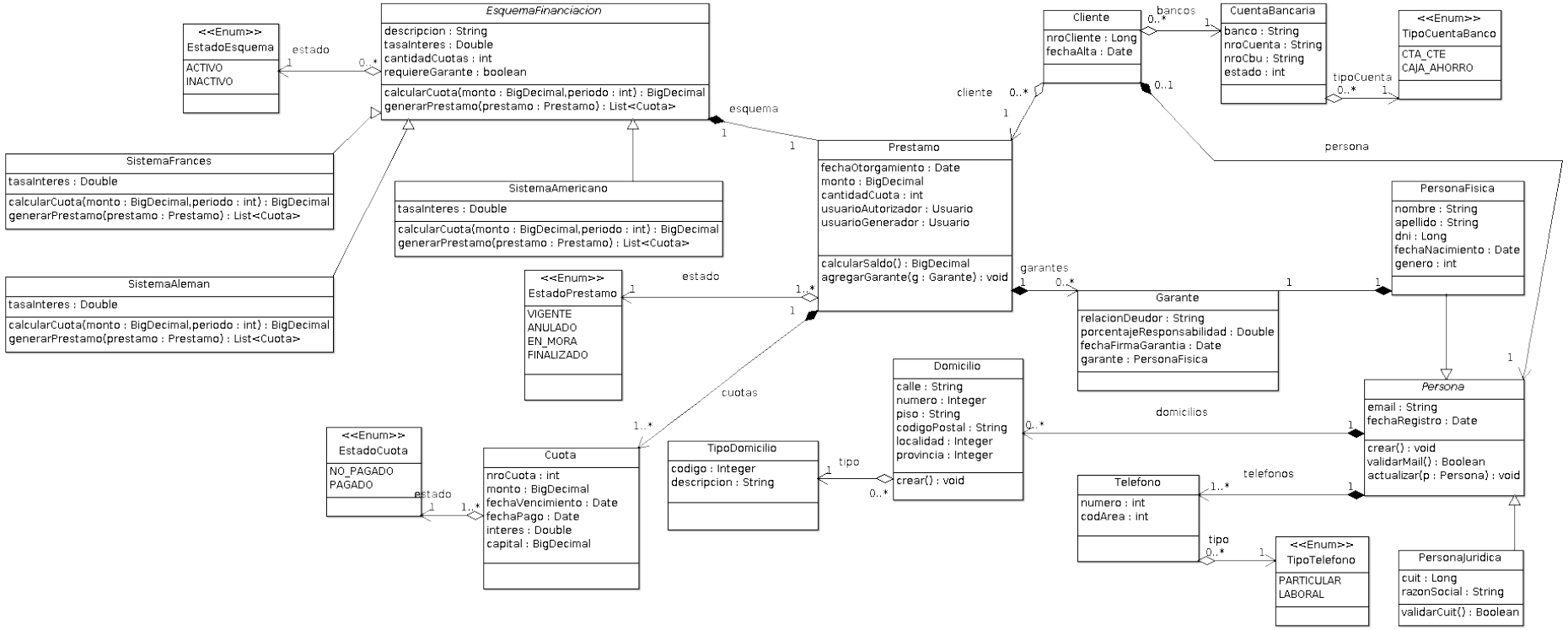


Ilustración 13: Diagrama de Clases - Package Aplicación

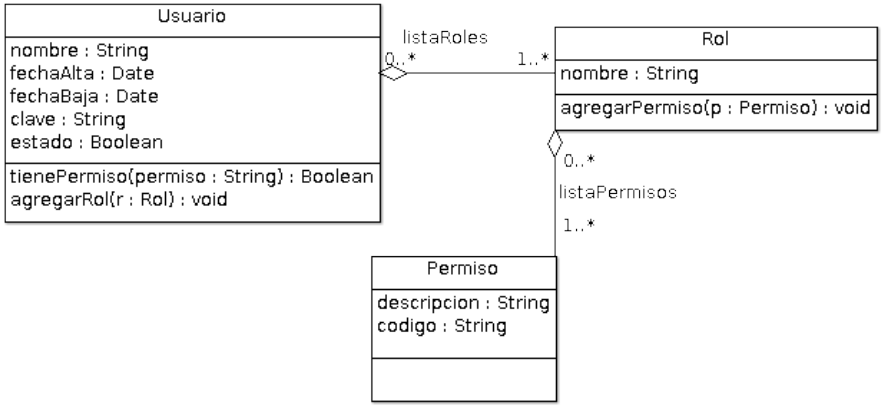


Ilustración 14: Diagrama de clases: Package Seguridad

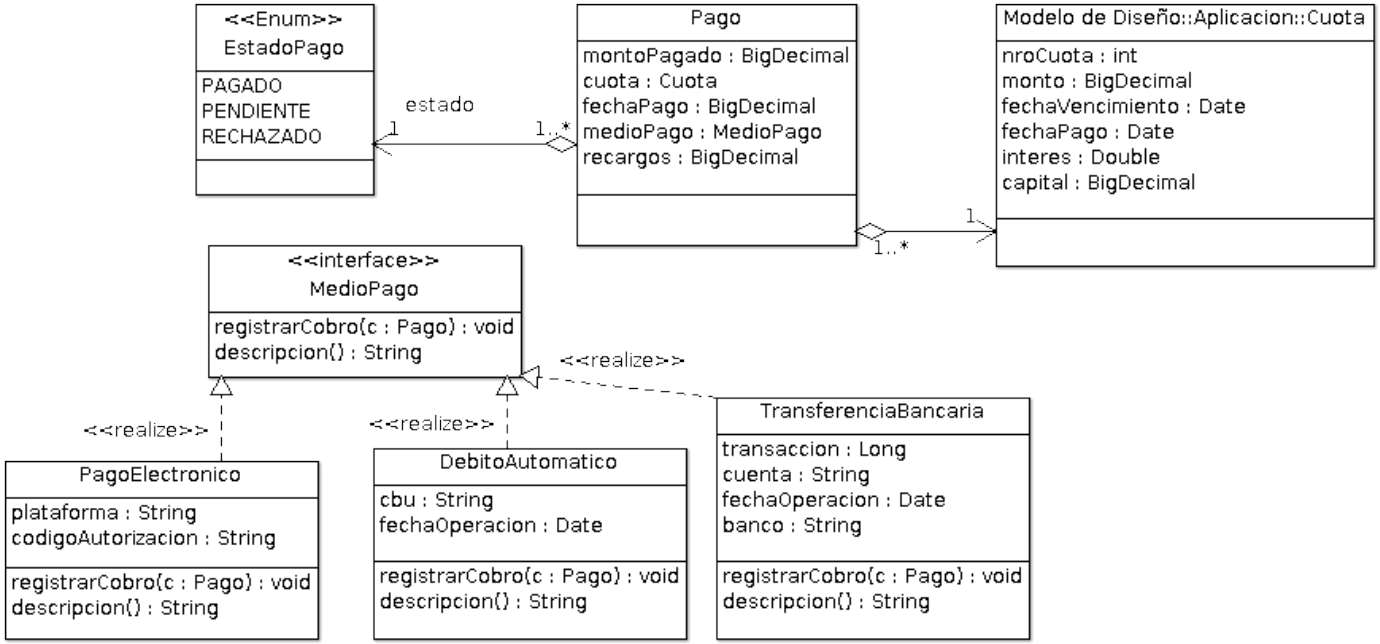


Ilustración 15: Diagrama de clases: Package Cobro

Modelo De Implementación

Requerimientos de Sistema

ID	Nombre
RNF03	El sistema debe funcionar a través de una red privada virtual
RNF07	El sistema debe contar con una base de datos MySQL.
RNF08	El sistema debe estar desarrollado en Java.
RNF09	El sistema operativo debe ser Linux

Diagrama de despliegue

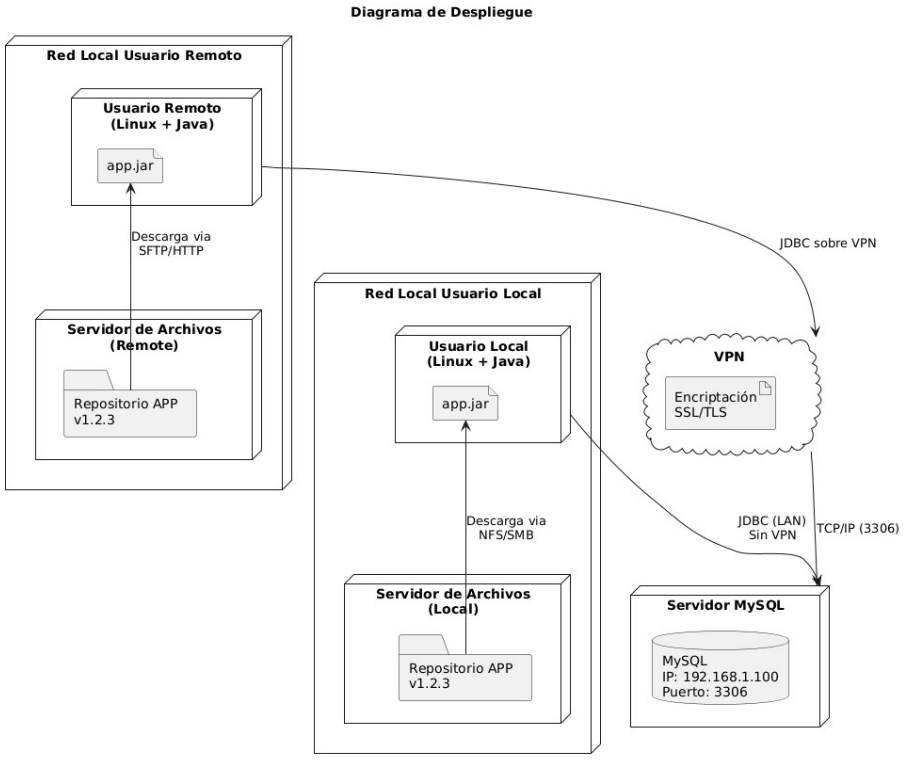


Ilustración 16: Diagrama de despliegue

Plan de Pruebas

Caso de uso	Código prueba	Tipo de prueba	Técnica propuesta	Observaciones
CU001	CP001	Funcional	Caja Negra	Verifica que el sistema permita el ingreso mediante

				usuario y contraseña.
CU001	CP002	Funcional	Caja Negra	Valida mensaje de error al ingresar credenciales inválidas
CU002-01	CP003	Funcional	Caja Negra	Verifica registro exitoso de nuevo usuario con datos válidos
CU002-01	CP004	Funcional	Caja Negra	Valida restricción de nombres de usuario duplicados
CU003-01	CP005	Funcional	Caja Negra	Verifica registro exitoso de nuevo cliente con datos completos
CU003-01	CP006	Funcional	Caja Negra	Valida restricción para DNI duplicados
CU004-01	CP007	Funcional	Caja Negra	Verifica registro exitoso de nuevo préstamo para cliente válido
CU004-01	CP008	Funcional	Caja Negra	Valida mensaje de error al intentar préstamo para cliente no registrado
CU004-02	CP009	Funcional	Caja Negra	Verifica aprobación exitosa de préstamo pendiente
CU004-02	CP010	Funcional	Caja Negra	Valida rechazo de préstamo con registro de motivo
CU003-02	CP011	Funcional	Caja Negra	Verifica búsqueda exacta de cliente por DNI
CU003-02	CP012	Funcional	Caja Negra	Valida búsqueda parcial de clientes por nombre/apellido
RF013	CP013	Funcional	Caja Negra	Verifica registro exitoso de pago de cuota con monto exacto
RF013	CP014	Funcional	Caja Negra	Valida restricción para pagos con monto insuficiente
Varios	CP015	Integración	Caja Negra	Valida flujo completo desde registro de cliente hasta pago de cuota
Seguridad	CP016	Seguridad	Caja Negra	Verifica control de acceso por roles de usuario

Rendimiento	CP017	Rendimiento	Caja Negra	Mide tiempo de respuesta con múltiples usuarios concurrentes

Caso de prueba

CU001 - Acceder al sistema

CP001 - Acceso exitoso con credenciales válidas

Precondición: Usuario registrado en el sistema con credenciales válidas
Pasos:

1. Ingresar nombre de usuario válido
2. Ingresar contraseña correcta
3. Hacer clic en "Acceder"
   Resultado esperado: Sistema muestra pantalla principal y mensaje "Acceso exitoso"
   Criterio de aceptación: RF001, RF023

CP002 - Acceso fallido con credenciales inválidas

Precondición: Usuario registrado en el sistema
Pasos:

1. Ingresar nombre de usuario válido
2. Ingresar contraseña incorrecta
3. Hacer clic en "Acceder"
   Resultado esperado: Sistema muestra mensaje "Credenciales inválidas. Intente nuevamente"
   Criterio de aceptación: RF023

CU002-01 - Registrar usuario

CP003 - Registro exitoso de nuevo usuario

Precondición: Administrador autenticado
Pasos:

1. Seleccionar "Crear Usuario"
2. Ingresar nombre de usuario único
3. Ingresar contraseña que cumpla políticas
4. Seleccionar rol válido

5. Confirmar registro
   Resultado esperado: Sistema muestra mensaje "Usuario registrado exitosamente" y aparece en listado
   Criterio de aceptación: RF016

CP004 - Intento de registro con nombre de usuario existente

Precondición: Usuario "jperez" ya registrado
Pasos:

1. Seleccionar "Crear Usuario"
2. Ingresar "jperez" como nombre de usuario
3. Completar resto de campos válidos
4. Confirmar registro
   Resultado esperado: Sistema muestra mensaje "Nombre de usuario ya existe. Por favor elija otro"
   Criterio de aceptación: RF016

CU003-01 - Registrar cliente

CP005 - Registro exitoso de nuevo cliente

Precondición: Operador autenticado
Pasos:

1. Seleccionar "Alta de Cliente"
2. Ingresar DNI no registrado (ej: 30123456)
3. Completar todos los campos obligatorios con datos válidos
4. Confirmar registro
   Resultado esperado: Sistema muestra mensaje "Cliente registrado exitosamente" y genera ficha
   Criterio de aceptación: RF002

CP006 - Intento de registro con DNI existente

Precondición: Cliente con DNI 30123456 ya registrado
Pasos:

1. Seleccionar "Alta de Cliente"
2. Ingresar DNI 30123456
3. Completar formulario
4. Confirmar registro
   Resultado esperado: Sistema muestra mensaje "El cliente ya está registrado" y no permite

continuar
Criterio de aceptación: RF002

CU004-01 - Registrar préstamo

CP007 - Registro exitoso de nuevo préstamo

Precondición: Cliente válido registrado sin préstamos activos
Pasos:

1. Seleccionar "Registrar Préstamo"
2. Ingresar DNI del cliente válido
3. Ingresar monto (ej: $50,000)
4. Seleccionar plazo (ej: 12 meses)
5. Seleccionar esquema de financiación
6. Confirmar préstamo
   Resultado esperado: Sistema muestra mensaje "Préstamo registrado exitosamente. Esperando aprobación" con número de préstamo
   Criterio de aceptación: RF005

CP008 - Intento de registro con cliente no existente

Precondición: Ninguna
Pasos:

1. Seleccionar "Registrar Préstamo"
2. Ingresar DNI no registrado (ej: 99999999)
3. Intentar continuar
   Resultado esperado: Sistema muestra mensaje "Cliente no encontrado. Registre al cliente primero"
   Criterio de aceptación: RF005

CU004-02 - Aprobar préstamo

CP009 - Aprobación exitosa de préstamo

Precondición: Préstamo en estado "Pendiente de Aprobación"
Pasos:

1. Seleccionar "Aprobar Préstamo"
2. Buscar préstamo por número o DNI
3. Verificar información

4. Confirmar aprobación
   Resultado esperado: Sistema cambia estado a "Aprobado", genera contrato y muestra mensaje "Préstamo aprobado exitosamente"
   Criterio de aceptación: RF010

CP010 - Rechazo de préstamo con motivo

Precondición: Préstamo en estado "Pendiente de Aprobación"
Pasos:

1. Seleccionar "Aprobar Préstamo"
2. Buscar préstamo por número o DNI
3. Seleccionar "Rechazar"
4. Ingresar motivo (ej: "Historial de morosidad")
5. Confirmar rechazo
   Resultado esperado: Sistema cambia estado a "Rechazado", registra motivo y muestra mensaje "Préstamo rechazado"
   Criterio de aceptación: RF010

CU003-02 - Buscar cliente

CP011 - Búsqueda exitosa por DNI exacto

Precondición: Cliente con DNI 30123456 registrado
Pasos:

1. Seleccionar "Buscar Cliente"
2. Ingresar "30123456" en campo DNI
3. Hacer clic en "Buscar"
   Resultado esperado: Sistema muestra ficha completa del cliente
   Criterio de aceptación: RF003

CP012 - Búsqueda por nombre parcial

Precondición: Cliente "Juan Pérez" registrado
Pasos:

1. Seleccionar "Buscar Cliente"
2. Ingresar "Pérez" en campo nombre
3. Hacer clic en "Buscar"
   Resultado esperado: Sistema muestra lista de clientes cuyo apellido contiene "Pérez"
   Criterio de aceptación: RF003

RF013 - Registrar pago de cuota

CP013 - Registro exitoso de pago

Precondición: Préstamo activo con cuotas pendientes
Pasos:

1. Seleccionar "Registrar Pago"
2. Ingresar número de préstamo válido
3. Seleccionar cuota a pagar
4. Ingresar monto exacto
5. Seleccionar medio de pago
6. Confirmar pago
   Resultado esperado: Sistema actualiza estado de la cuota a "PAGADO", muestra comprobante y actualiza saldo pendiente
   Criterio de aceptación: RF013, RF014

CP014 - Intento de pago con monto insuficiente

Precondición: Préstamo activo con cuota de $10,000 pendiente
Pasos:

1. Seleccionar "Registrar Pago"
2. Ingresar número de préstamo válido
3. Seleccionar cuota a pagar
4. Ingresar monto $9,000
5. Confirmar pago
   Resultado esperado: Sistema muestra mensaje "Monto insuficiente. El pago debe ser de $10,000" y no registra el pago
   Criterio de aceptación: RF013

Pruebas de integración clave

CP015 - Flujo completo préstamo

Precondición: Sistema limpio, sin datos
Pasos:

1. Registrar nuevo usuario (operador)
2. Iniciar sesión como operador
3. Registrar nuevo cliente
4. Registrar nuevo préstamo para el cliente

5. Aprobar préstamo (como administrador)
6. Registrar pago de primera cuota
   Resultado esperado: Todos los pasos se completan exitosamente, el sistema mantiene consistencia en los datos
   Criterio de aceptación: RF002, RF005, RF010, RF013

Pruebas de seguridad

CP016 - Acceso no autorizado

Precondición: Usuario con rol "Operador" registrado
Pasos:

1. Iniciar sesión como operador
2. Intentar acceder a pantalla "Gestión de Usuarios"
   Resultado esperado: Sistema muestra mensaje "No tiene permisos para acceder a esta funcionalidad"
   Criterio de aceptación: RNF07 (Control de acceso)

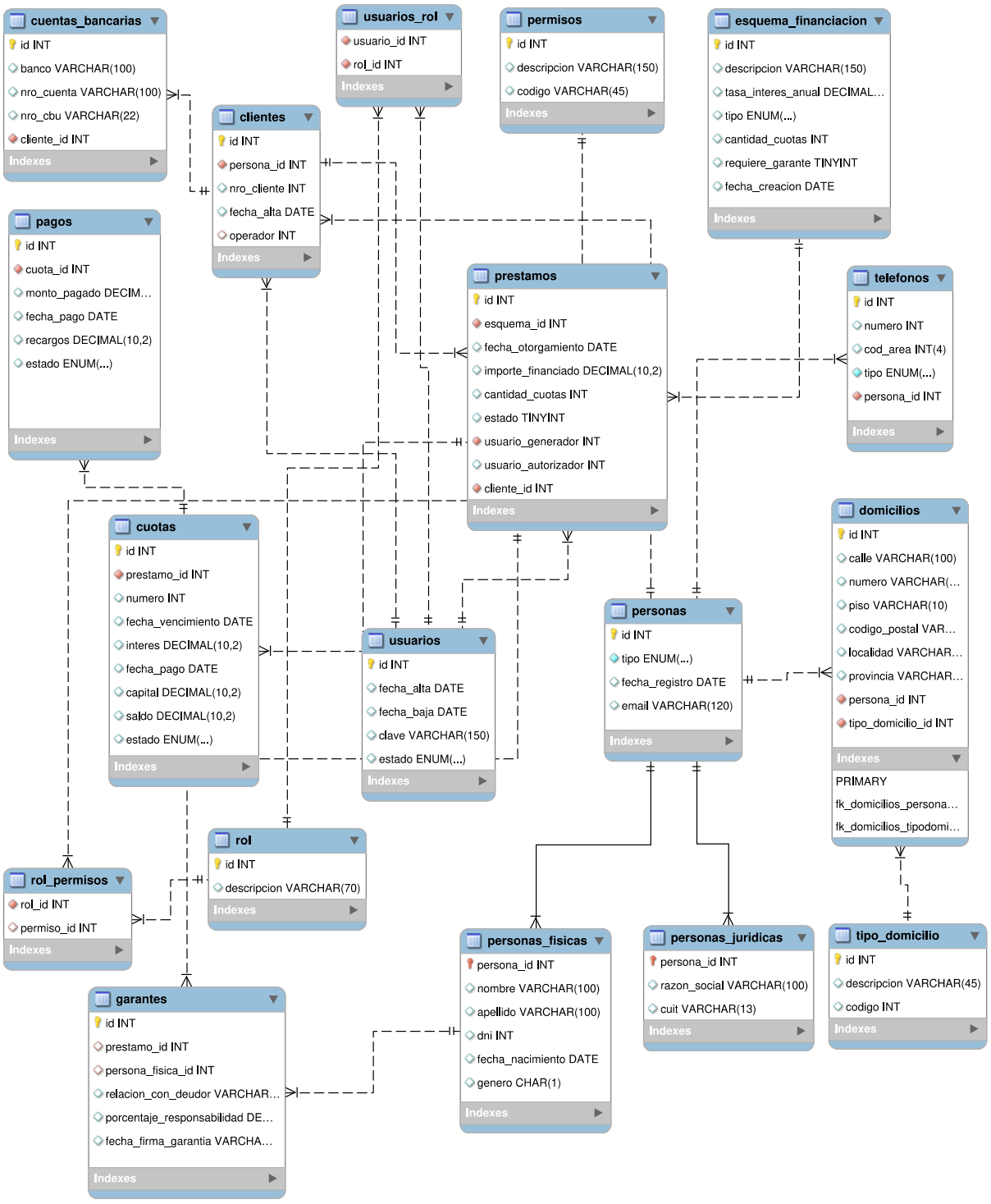
Pruebas de rendimiento

CP017 - Carga múltiples usuarios

Precondición: 100 clientes de prueba registrados
Pasos:

1. Simular 10 usuarios concurrentes buscando clientes
2. Medir tiempo de respuesta
   Resultado esperado: Tiempo promedio de respuesta < 2 segundos
   Criterio de aceptación: RNF04 (Optimización de recursos)

Diagrama Entidad Relación



Creación de base de datos

Creación de base de datos y usuario para uso del sistema.

```sql
CREATE DATABASE seminario_db;
CREATE USER 'seminario'@'%' IDENTIFIED BY 'password123';
GRANT SELECT, INSERT, UPDATE, DELETE ON seminario_db.* TO 'seminario'@'%';

FLUSH PRIVILEGES;
```

Linea 1: Crea la base de datos con el nombre “seminario_db”

Linea 2: Se crea un usuario “seminario” para que pueda conectarse desde cualquier host.

Linea 3: Asigna permisos basicos al usuario **seminario** sobre la base **seminario_db**.

Linea 5: Fuerza al motor de MySQL a recargar los permisos de todas las tablas.

Descripción de tablas del sistema

Tabla usuarios

Representa a usuarios que pueden utilizar el sistema.

```sql

CREATE TABLE IF NOT EXISTS `seminario_db`.`usuarios` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `nombre` VARCHAR(45) NULL DEFAULT NULL,
  `fecha_alta` DATE NULL DEFAULT NULL,
  `fecha_baja` DATE NULL DEFAULT NULL,
  `clave` VARCHAR(150) NULL DEFAULT NULL,
  `estado` ENUM('ACTIVO', 'INACTIVO') NULL DEFAULT 'INACTIVO',
  PRIMARY KEY (`id`),
  UNIQUE INDEX `nombre_UNIQUE` (`nombre` ASC) VISIBLE)
ENGINE = InnoDB;
```

Inserción inicial de tres usuarios del sistema.

```sql
INSERT INTO seminario_db.usuarios (fecha_alta,  fecha_baja, clave,  estado, nombre)
VALUES('2025-06-01', NULL, 'a242a85055fd0ba4221473647e21ae',
'ACTIVO', 'administrador');

INSERT INTO seminario_db.usuarios (fecha_alta,  fecha_baja, clave,  estado, nombre)
VALUES('2025-06-01', NULL,'a242a85055fd0ba4221473647e21ae',
'ACTIVO','operador');

INSERT INTO seminario_db.usuarios (fecha_alta,  fecha_baja, clave,  estado, nombre)
VALUES('2025-06-01', NULL, 'a242a85055fd0ba4221473647e21ae',
'ACTIVO','super');
```

Tablas rol, permisos, rol_permisos, usuarios_rol:

Representan el sistema de roles y permisos del sistema, utilizado para controlar el acceso y las funcionalidades disponibles.

```
CREATE TABLE `rol` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `descripcion` VARCHAR(70) NULL DEFAULT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;

CREATE TABLE `permisos` (
  `id` int NOT NULL AUTO_INCREMENT,
  `descripcion` varchar(150) DEFAULT NULL,
  `codigo` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

CREATE TABLE `rol_permisos` (
  `rol_id` int NOT NULL,
  `permiso_id` int DEFAULT NULL,
  UNIQUE KEY `rol_permisos_id_rol_IDX` (`rol_id`,`permiso_id`) USING BTREE,
  KEY `rol_permisos_FK` (`permiso_id`),
  CONSTRAINT `rol_permisos_FK` FOREIGN KEY (`permiso_id`) REFERENCES `permisos` (`id`),
  CONSTRAINT `rol_permisos_FK_1` FOREIGN KEY (`rol_id`) REFERENCES `rol` (`id`)
) ENGINE=InnoDB;

CREATE TABLE `usuarios_rol` (
  `usuario_id` int NOT NULL,
  `rol_id` int NOT NULL,
  UNIQUE KEY `usuarios_rol_id_usuario_IDX` (`usuario_id`,`rol_id`) USING BTREE,
  KEY `usuarios_rol_FK` (`rol_id`),
  CONSTRAINT `usuarios_rol_FK` FOREIGN KEY (`rol_id`) REFERENCES `rol` (`id`)
        ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `usuarios_rol_FK_1` FOREIGN KEY (`usuario_id`) REFERENCES `usuarios` (`id`)
        ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE=InnoDB;
```

Inicialmente se insertan los valores básicos necesarios para su funcionamiento.

```
-- Roles
INSERT INTO seminario_db.rol (id,descripcion) VALUES(1,'SUPER');
INSERT INTO seminario_db.rol (id,descripcion) VALUES(2,'ADMINISTRADOR');
INSERT INTO seminario_db.rol (id,descripcion) VALUES(3,'OPERADOR');
INSERT INTO seminario_db.rol (id,descripcion) VALUES(4,'TESORERIA');

-- Permisos
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(1, 'Permiter crear cliente', 'crear_cliente');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(2, 'Permiter crear cliente', 'modificar_cliente');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(3, 'Permitir eliminar cliente', 'eliminar_cliente');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
```

```
    VALUES(4, 'Permitir crear préstamo', 'crear_prestamo');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(5, 'Permitir aprobar préstamo', 'aprobar_prestamo');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(6, 'Permitir cancelar préstamo', 'cancelar_prestamo');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(7, 'Permitir registrar usuario', 'crear_usuario');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(8, 'Permitir modificar usuario', 'modificar_usuario');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(9, 'Permitir crear rol', 'crear_rol');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(10, 'Permitir modificar rol', 'modificar_rol');
INSERT INTO seminario_db.permisos (id, descripcion, codigo)
    VALUES(11, 'Permitir registrar cobro', 'registrar_cobro');

-- Relación Rol -> Permiso 1: muchos
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(2, 7);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(2, 8);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(2, 9);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(2, 10);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(2, 5);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(2, 6);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(3, 1);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(3, 2);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(3, 3);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(3, 4);
INSERT INTO seminario_db.rol_permisos (rol_id, permiso_id) VALUES(4, 11);

-- Relación Usuario -> Rol 1: muchos
INSERT INTO seminario_db.usuarios_rol (usuario_id, rol_id) VALUES(2, 3);
INSERT INTO seminario_db.usuarios_rol (usuario_id, rol_id) VALUES(1, 2);
```

Tabla personas

Actúa como entidad base y contiene los atributos comunes de cualquier persona registrada en el sistema. Cada registro representa una persona, que puede ser de tipo:
'FISICA': Persona física (individuo)
'JURIDICA': Persona jurídica (empresa u organización).

Tabla personas_fisicas: es subentidad de personas y contiene los datos específicos de las personas físicas (nombre, apellido, dni, fecha_nacimiento, genero). Su clave primaria persona_id es también una **clave foránea que apunta a personas.id**.

Tabla personas_juridicas: es subentidad de personas y contiene los datos específicos de las personas jurídicas (razon_social, cuit). Su clave primaria persona_id es también una **clave foránea que apunta a personas.id**.

```
CREATE TABLE `personas` (
  `id` int NOT NULL AUTO_INCREMENT,
```

```
  `tipo` enum('FISICA','JURIDICA') NOT NULL,
  `fecha_registro` date DEFAULT NULL,
  `email` varchar(120) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

CREATE TABLE `personas_fisicas` (
  `persona_id` int NOT NULL,
  `nombre` varchar(100) DEFAULT NULL,
  `apellido` varchar(100) DEFAULT NULL,
  `dni` int DEFAULT NULL,
  `fecha_nacimiento` date DEFAULT NULL,
  `genero` char(1) DEFAULT NULL,
  PRIMARY KEY (`persona_id`),
  CONSTRAINT `fk_personas_fisicas_1`
    FOREIGN KEY (`persona_id`) REFERENCES `personas` (`id`)
) ENGINE=InnoDB;


CREATE TABLE `personas_juridicas` (
  `persona_id` int NOT NULL,
  `razon_social` varchar(100) DEFAULT NULL,
  `cuit` varchar(13) DEFAULT NULL,
  PRIMARY KEY (`persona_id`),
  CONSTRAINT `fk_personas_juridicas_1`
    FOREIGN KEY (`persona_id`) REFERENCES `personas` (`id`)
) ENGINE=InnoDB;
```

Tabla clientes

Representa a las personas que son clientes del sistema.

Se vincula con:

- **personas** mediante persona_id (FK): Cada cliente debe estar asociado a una persona registrada (ya sea física o jurídica).
- **usuarios** mediante operador (FK): Referencia al operador que dio de alta al cliente (usuario del sistema).

Tabla tipo_domicilio

Define los tipos posibles de domicilios (ej. particular, laboral, legal). Se vincula exclusivamente con domicilios.

```
CREATE TABLE `tipo_domicilio` (
  `id` int NOT NULL AUTO_INCREMENT,
  `descripcion` varchar(45) DEFAULT NULL,
  `codigo` int DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;

```

```sql
INSERT INTO seminario_db.tipo_domicilio (id, descripcion, codigo) VALUES(1, 'Particular', 1);
INSERT INTO seminario_db.tipo_domicilio (id, descripcion, codigo) VALUES(2, 'Laboral', 2);
INSERT INTO seminario_db.tipo_domicilio (id, descripcion, codigo) VALUES(3, 'Otro', 3);
```

Tabla domicilios

Representa direcciones asociadas a personas.

Se vincula con:

- **personas** mediante persona_id (FK): Cada domicilio pertenece a una persona.
- **tipo_domicilio** mediante tipo_domicilio_id (FK): Indica si es un domicilio fiscal, legal, particular, etc.

```sql
CREATE TABLE `domicilios` (
  `id` int NOT NULL AUTO_INCREMENT,
  `calle` varchar(100) DEFAULT NULL,
  `numero` varchar(20) DEFAULT NULL,
  `piso` varchar(10) DEFAULT NULL,
  `codigo_postal` varchar(10) DEFAULT NULL,
  `localidad` varchar(100) DEFAULT NULL,
  `provincia` varchar(100) DEFAULT NULL,
  `persona_id` int NOT NULL,
  `tipo_domicilio_id` int NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_domicilios_personas_idx` (`persona_id`),
  KEY `fk_domicilios_tipodomicilio_idx` (`tipo_domicilio_id`),
  CONSTRAINT `domicilios_personas_FK`
       FOREIGN KEY (`persona_id`) REFERENCES `personas` (`id`),
  CONSTRAINT `domicilios_tipo_domicilio_FK`
        FOREIGN KEY (`tipo_domicilio_id`) REFERENCES `tipo_domicilio` (`id`)
) ENGINE=InnoDB;
```

Tabla telefonos

Contiene los números telefónicos asociados a una persona. Puede ser laboral, personal, etc.

Se vincula con:

- **personas** mediante persona_id (FK)

```sql
CREATE TABLE `telefonos` (
  `id` int NOT NULL AUTO_INCREMENT,
  `numero` int DEFAULT NULL,
  `cod_area` int DEFAULT NULL,
  `tipo` enum('PARTICULAR','LABORAL') NOT NULL,
  `persona_id` int NOT NULL,
```

```
  PRIMARY KEY (`id`),
  KEY `fk_telefonos_1_idx` (`persona_id`),
  CONSTRAINT `fk_telefonos_1`
    FOREIGN KEY (`persona_id`) REFERENCES `personas` (`id`)
) ENGINE=InnoDB;
```

Tabla garantes

Representa a las personas físicas que actúan como garantes de préstamos.

Se vincula con:

- **prestamos** mediante prestamo_id (FK)
- **personas_fisicas** mediante persona_fisica_id (FK)

```
CREATE TABLE `garantes` (
  `id` int NOT NULL AUTO_INCREMENT,
  `prestamo_id` int DEFAULT NULL,
  `persona_fisica_id` int DEFAULT NULL,
  `relacion_con_deudor` varchar(150) DEFAULT NULL,
  `porcentaje_responsabilidad` decimal(5,2) DEFAULT NULL,
  `fecha_firma_garantia` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_garantes_1_idx` (`prestamo_id`),
  KEY `fk_garantes_2_idx` (`persona_fisica_id`),
  CONSTRAINT `fk_garantes_1`
    FOREIGN KEY (`prestamo_id`) REFERENCES `prestamos` (`id`),
  CONSTRAINT `fk_garantes_2`
    FOREIGN KEY (`persona_fisica_id`) REFERENCES `personas_fisicas` (`persona_id`)
) ENGINE=InnoDB;
```

Tabla prestamos

La tabla prestamos representa los préstamos otorgados a los clientes en el sistema.

```
CREATE TABLE `prestamos` (
  `id` int NOT NULL AUTO_INCREMENT,
  `esquema_id` int NOT NULL,
  `fecha_otorgamiento` date DEFAULT NULL,
  `importe_financiado` decimal(10,2) DEFAULT NULL,
  `cantidad_cuotas` int DEFAULT NULL,
  `estado` tinyint DEFAULT NULL,
  `usuario_generador` int NOT NULL,
  `usuario_autorizador` int DEFAULT NULL,
  `cliente_id` int NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_prestamos_1_idx` (`esquema_id`),
  KEY `fk_prestamos_2_idx` (`usuario_generador`),
  KEY `fk_prestamos_4_idx` (`cliente_id`),
  CONSTRAINT `fk_prestamos_1`
    FOREIGN KEY (`esquema_id`) REFERENCES `esquema_financiacion` (`id`),
```

```
  CONSTRAINT `fk_prestamos_2`
    FOREIGN KEY (`usuario_generador`) REFERENCES `usuarios` (`id`),
  CONSTRAINT `fk_prestamos_4`
    FOREIGN KEY (`cliente_id`) REFERENCES `clientes` (`id`)
) ENGINE=InnoDB;
```

Un préstamo está asociado a un cliente (**cliente_id**), que a su vez está vinculado a una persona (**persona_id**).

Tabla esquema_financiacion

Define los distintos esquemas de financiación disponibles para el otorgamiento de préstamos. Cada esquema contiene información clave sobre el tipo de amortización, tasa de interés y condiciones generales.

```
CREATE TABLE `esquema_financiacion` (
  `id` int NOT NULL AUTO_INCREMENT,
  `descripcion` varchar(150) DEFAULT NULL,
  `tasa_interes_anual` decimal(5,2) DEFAULT NULL,
  `tipo` enum('FRANCES','ALEMAN','AMERICANO') DEFAULT NULL,
  `cantidad_cuotas` int DEFAULT NULL,
  `requiere_garante` tinyint DEFAULT NULL,
  `fecha_creacion` date DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

Tabla cuotas

Representa cada una de las cuotas asociadas a un préstamo.

```
CREATE TABLE `cuotas` (
  `id` int NOT NULL,
  `prestamo_id` int NOT NULL,
  `numero` int DEFAULT NULL,
  `fecha_vencimiento` date DEFAULT NULL,
  `interes` decimal(10,2) DEFAULT NULL,
  `fecha_pago` date DEFAULT NULL,
  `capital` decimal(10,2) DEFAULT NULL,
  `saldo` decimal(10,2) DEFAULT NULL,
  `estado` enum('PENDIENTE','PAGADA','VENCIDA') DEFAULT 'PENDIENTE',
  PRIMARY KEY (`id`),
  UNIQUE KEY `uc_prestamo_cuota` (`prestamo_id`,`numero`),
  KEY `fk_cuotas_prestamos1_idx` (`prestamo_id`),
  CONSTRAINT `fk_cuotas_prestamos1`
    FOREIGN KEY (`prestamo_id`) REFERENCES `prestamos` (`id`)
) ENGINE=InnoDB;
```

Tabla pagos

Registra los pagos realizados sobre cuotas.

```sql
CREATE TABLE `pagos` (
  `id` int NOT NULL AUTO_INCREMENT,
  `cuota_id` int NOT NULL,
  `monto_pagado` decimal(10,2) DEFAULT NULL,
  `fecha_pago` date DEFAULT NULL,
  `recargos` decimal(10,2) DEFAULT NULL,
  `estado` enum('PENDIENTE','COMPLETADO','RECHAZADO') DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_pagos_cuotas1_idx` (`cuota_id`),
  CONSTRAINT `fk_pagos_cuotas1`
  FOREIGN KEY (`cuota_id`) REFERENCES `cuotas` (`id`)
) ENGINE=InnoDB;
```

Implementación del sistema utilizando el lenguaje Java

El sistema será desarrollado utilizando el lenguaje de programación Java, aplicando principios de diseño orientado a objetos. A continuación, se describen las principales clases generadas, su responsabilidad y cómo interactúan entre sí dentro del proyecto:

Clase Usuario

Pertenece al paquete de seguridad y representa a un usuario registrado en la aplicación. Su diseño aplica principios de encapsulamiento y está orientado a facilitar los procesos de autenticación, autorización y gestión de roles.

```java
public class Usuario {
    private Long id;
    private String nombre;
    private Date fechaAlta;
    private Date fechaBaja;
    private String clave;
    private Boolean estado;
    private List<Rol> listaRoles = new ArrayList<>();
    public Usuario() { }
    public Usuario(Long id, String nombre, String clave) {
        this.id = id;
        this.nombre = nombre;
        this.clave = clave;
    }
    public Usuario(Long id, String nombre, Date fechaAlta, Date fechaBaja, String clave, Boolean estado) {
        this.id = id;
        this.fechaAlta = fechaAlta;
        this.fechaBaja = fechaBaja;
        this.clave = clave;
```

```
20          this.nombre = nombre;
21          this.estado = estado;
22      }
23
24      public Boolean tienePermiso(String pPermiso) {
25          for (Rol r : this.listaRoles) {
26              for (Permiso p : r.getListaPermisos()) {
27                  if (p.getCodigo().equals(pPermiso))
28                      return true;
29              }
30          }
31          return false;
32      }
33
34      // Getters Setters
35  }
```

Tiene 3 constructores: Constructor vacío, Constructor básico con id, nombre y clave y Constructor completo que incluye fechas y estado.

Metodo tienePermiso(String pPermiso):

Recorre la lista de roles del usuario y sus permisos para verificar si tiene asignado un permiso específico, identificado por su código. Devuelve true si el permiso está presente, false en caso contrario. Se utiliza para validar autorizaciones en tiempo de ejecución.

Relaciones:

- Uno a muchos con **Rol**: Un usuario puede tener múltiples roles.
- Relación indirecta con **Permiso** a través de **Rol**.

Clase Rol

Pertenece al paquete de seguridad y define los roles que pueden asignarse a los usuarios. Cada rol agrupa un conjunto de permisos, estableciendo una relación de uno a muchos con la clase **Permiso**.

```
1   public class Rol {
2       private Long idRol;
3       private String nombre;
4       private List<Permiso> listaPermisos = new ArrayList<>();
5       public Rol() {}
6       public Rol(Long idRol, String nombre) {
7           this.idRol = idRol;
8           this.nombre = nombre;
9           this.listaPermisos = new ArrayList<>();
10      }
11      // Métodos getters y setters
12  }
```

Clase Permiso

Pertenece al paquete de seguridad y representa una acción específica o capacidad del sistema que puede ser otorgada mediante un rol. Es utilizada por la clase **Rol** para definir los alcances y niveles de acceso permitidos dentro del sistema.

```java
public class Permiso {
    private String descripcion;
    private String codigo;
    public Permiso() {
    }
    public Permiso(String descripcion, String codigo) {
        this.descripcion = descripcion;
        this.codigo = codigo;
    }
    // Métodos getters y setters
}
```

Clase Domicilio

Pertenece al paquete base del modelo de dominio y representa una dirección física asociada a una entidad del sistema (por ejemplo, un usuario o cliente). Incluye información detallada como calle, número, piso, código postal, localidad y provincia. Además, está asociada a un **TipoDomicilio**, lo que permite clasificar la dirección (por ejemplo, fiscal, legal o particular).

```java
public class Domicilio {
    private Long idDomicilio;
    private String calle;
    private Integer numero;
    private String piso;
    private String codigoPostal;
    private Integer localidad;
    private Integer provincia;
    private TipoDomicilio tipoDomicilio;
    public Domicilio() {
    }
    public Domicilio(Long idDomicilio, String calle, Integer numero,
String piso, String codigoPostal, Integer localidad, Integer provincia, TipoDomicilio tipoDomicilio) {
        this.idDomicilio = idDomicilio;
        this.calle = calle;
        this.numero = numero;
        this.piso = piso;
        this.codigoPostal = codigoPostal;
        this.localidad = localidad;
        this.provincia = provincia;
        this.tipoDomicilio = tipoDomicilio;
    }
    // Métodos getters y setters
}
```

Clase TipoDomicilio

Pertenece al paquete base del modelo de dominio y representa la categoría o tipo asignado a un domicilio. Incluye un código identificador y una descripción textual. Esta clase permite distinguir entre diferentes usos o funciones de una dirección (como domicilio fiscal, comercial, legal, etc.).

```java
public class TipoDomicilio {
    private Long codigo;
    private String descripcion;
    public TipoDomicilio(){}
    public TipoDomicilio(Long codigo, String descripcion) {
        this.codigo = codigo;
        this.descripcion = descripcion;
    }
    // Métodos getters y setters
}
```

Clase Persona

Pertenece al paquete base del modelo de dominio y representa una entidad abstracta común a todas las personas registradas en el sistema, ya sean físicas o jurídicas. Incluye atributos compartidos como identificador (idPersona), correo electrónico (email), fecha de registro y listas de domicilios y teléfonos asociados.

Esta clase abstracta sirve como base para las subclases **PersonaFisica** y **PersonaJuridica**, promoviendo la reutilización de código.

```java
public abstract class Persona {
    private Long idPersona;
    private String email;
    private LocalDate fechaRegistro;
    List<Domicilio> domicilios = new ArrayList<>();
    List<Telefono> telefonos = new ArrayList<>();

    public Persona(Long idPersona, String email, LocalDate fechaRegistro,
            List<Domicilio> domicilios, List<Telefono> telefonos) {
        this.idPersona = idPersona;
        this.email = email;
        this.fechaRegistro = fechaRegistro;
        this.domicilios = domicilios;
        this.telefonos = telefonos;
    }
    // Métodos getters y setters
}
```

Clase PersonaFisica

Extiende la clase Persona y representa a una persona humana dentro del sistema. Añade atributos específicos como nombre, apellido, número de documento (DNI), fecha de nacimiento y género. Esta clase permite modelar de forma precisa los datos personales necesarios para la gestión de clientes u otros individuos registrados.

```java
public class PersonaFisica extends Persona {
    private String nombre;
    private String apellido;
    private Long dni;
    private LocalDate fechaNacimiento;
    private Character genero;
    public PersonaFisica(Long idPersona, String email,
        LocalDate fechaRegistro, List<Domicilio> domicilios, List<Telefono> telefonos) {
        super(idPersona, email, fechaRegistro, domicilios, telefonos);
    }
    // Métodos getters y setters
}
```

Clase PersonaJuridica

Extiende la clase Persona y representa a una entidad legal o empresa registrada en el sistema. Incorpora atributos específicos como razón social y CUIT.
Esta clase es útil para modelar instituciones, organizaciones o sociedades comerciales que interactúan con el sistema.

```java
public class PersonaJuridica extends Persona {
    private String razonSocial;
    private Long cuit;
    public PersonaJuridica(Long idPersona, String email, LocalDate fechaRegistro,
        List<Domicilio> domicilios, List<Telefono> telefonos) {
        super(idPersona, email, fechaRegistro, domicilios, telefonos);
    }
    // Métodos getters y setters
}
```

Clase Telefono

Pertenece al paquete base del modelo de dominio y representa un número telefónico asociado a una persona o entidad. Incluye atributos como identificador (idTelefono), número telefónico (numero), código de área (codArea) y tipo de teléfono (tipo), que está definido por la enumeración **TipoTelefono**.

Esta clase permite gestionar distintos números de contacto, distinguiendo entre teléfonos particulares y laborales, facilitando la organización y clasificación de los datos de contacto.

```java
public class Telefono {
    private Long idTelefono;
    private Integer numero;
    private Integer codArea;
    private TipoTelefono tipo;

    public Telefono() {
    }
    public Telefono(Integer numero, Integer codArea, TipoTelefono tipo) {
        this.numero = numero;
        this.codArea = codArea;
        this.tipo = tipo;
    }
    // Métodos getters y setters
}
```

Enumeración TipoTelefono

Pertenece al paquete base del modelo de dominio y define los tipos de teléfonos que pueden asociarse a una persona o entidad en el sistema. Actualmente contempla dos valores:

- **PARTICULAR**: Representa un teléfono personal o privado.
- **LABORAL**: Representa un teléfono relacionado con el ámbito laboral o profesional.

Cada valor incluye una descripción legible accesible mediante el método getTipo(), facilitando la presentación amigable en interfaces o reportes.

```java
public enum TipoTelefono {
    PARTICULAR("Particular"), LABORAL("Laboral");
    private final String tipo;
    TipoTelefono(String tipo) {
        this.tipo = tipo;
    }
    public String getTipo() {
        return tipo;
    }
}
```

Clase Cliente

Pertenece al paquete aplicación del modelo de dominio y representa a un cliente del sistema. Cada cliente posee un número único de cliente (nroCliente), una fecha de alta en el sistema (fechaAlta) y está asociado a una persona (persona), que puede ser física o jurídica. Además, mantiene una lista de préstamos (prestamos) vinculados a ese cliente.

Ademas la clase tiene dos responsabilidades principales:

- **obtenerSaldoConsolidado()**: devuelve el saldo total adeudado por el cliente, consolidando todos los préstamos registrados. Actualmente, retorna BigDecimal.ZERO como valor por defecto.
- **consultarSaldosDetallados()**: retorna un Map con información detallada del saldo de cada préstamo. Cada entrada del mapa puede tener como clave una descripción (número de préstamo) y como valor el monto pendiente.

```java
public class Cliente {
    private Long nroCliente;
    private LocalDate fechaAlta;
    private Persona persona;
    private List<Prestamo> prestamos = new ArrayList<>();

    public Cliente(){}
    public Cliente(Long nroCliente, LocalDate fechaAlta, Persona persona) {
        this.nroCliente = nroCliente;
        this.fechaAlta = fechaAlta;
        this.persona = persona;
    }

    public BigDecimal obtenerSaldoConsolidado() {
        return BigDecimal.ZERO;
    }
    public Map<String, BigDecimal> consultarSaldosDetallados() {
        Map<String, BigDecimal> saldos = new LinkedHashMap<>();
        for (Prestamo prestamo : prestamos) {
            // Recorro los prestamos para obtener saldos
        }
        return saldos;
    }

    // Métodos getters y setters
}
```

Clase EsquemaFinanciacion

Clase abstracta que pertenece al paquete aplicación del modelo que representa un esquema de financiación utilizado para estructurar préstamos.

Este esquema define las reglas bajo las cuales se otorgan y amortizan los créditos.

```java
public abstract class EsquemaFinanciacion {
    private Long idEsquema;
    private String descripcion;
    private Double tasaInteres;
    private Integer cantidadCuotas;
    private Boolean requiereGarante;
    private EstadoEsquema estado;
    public EsquemaFinanciacion() {
    }
    public EsquemaFinanciacion(String descripcion, Double tasaInteres,
Integer cantidadCuotas, Boolean requiereGarante, EstadoEsquema estado) {
        this.descripcion = descripcion;
```

```
13          this.tasaInteres = tasaInteres;
14          this.cantidadCuotas = cantidadCuotas;
15          this.requiereGarante = requiereGarante;
16          this.estado = estado;
17      }
18
19      public abstract BigDecimal calcularCuota(BigDecimal montoAFinanciar, Integer periodo);
20
21      public abstract List<Cuota> generarPrestamo(Prestamo prestamo);
22  }
```

Métodos abstractos:

- **public abstract BigDecimal calcularCuota(BigDecimal montoAFinanciar, Integer periodo);**
  Calcula el valor de la cuota correspondiente a un período determinado, según el monto a financiar. Debe ser implementado por las subclases según la lógica del tipo de esquema (por ejemplo, cuotas fijas, decrecientes, etc.).
- **public abstract List<Cuota> generarPrestamo(Prestamo prestamo);**
  Genera el plan de cuotas para un préstamo determinado según las reglas del esquema. Retorna una lista completa de objetos Cuota y la asocie al préstamo.

Clase SistemaFrances

Clase concreta que extiende a **EsquemaFinanciacion** e implementa la lógica de generación de cuotas utilizando el sistema francés de amortización. En este sistema, el valor total de la cuota es constante durante toda la vida del préstamo, aunque la proporción de interés y amortización cambia mes a mes.

```
1   public class SistemaFrances extends EsquemaFinanciacion {
2       public SistemaFrances(String descripcion, Double tasaInteres,
3   Integer cantidadCuotas, Boolean requiereGarante, EstadoEsquema estado) {
4           super(descripcion, tasaInteres, cantidadCuotas, requiereGarante, estado);
5       }
6       @Override
7       public BigDecimal calcularCuota(BigDecimal montoAFinanciar, Integer periodo) {
8           // Validaciones básicas
9           if (montoAFinanciar.compareTo(BigDecimal.ZERO) <= 0 || periodo <= 0) {
10              throw new IllegalArgumentException("El monto a financiar y el plazo deben ser positivos.");
11          }
12          // Convertir tasa anual a mensual y a decimal (ej: 12% -> 0.01)
13          BigDecimal tasaInteresMensual = BigDecimal.valueOf(getTasaInteres() / 100 / 12);
14          // Fórmula: C = (P * i * (1 + i)^n) / ((1 + i)^n - 1)
15          BigDecimal potencia = BigDecimal.ONE.add(tasaInteresMensual).pow(getCantidadCuotas());
16          BigDecimal numerador = montoAFinanciar.multiply(tasaInteresMensual).multiply(potencia);
17          BigDecimal denominador = potencia.subtract(BigDecimal.ONE);
18          BigDecimal cuota = numerador.divide(denominador, 2, RoundingMode.HALF_UP);
19          return cuota;
20      }
21      @Override
22      public List<Cuota> generarPrestamo(Prestamo prestamo) {
23          List<Cuota> cuotas = new ArrayList<>();
```

```
        BigDecimal monto = prestamo.getMonto();
        BigDecimal tasaMensual = BigDecimal.valueOf(getTasaInteres() / 100 / 12);
        BigDecimal cuotaFija = calcularCuota(monto, getCantidadCuotas());
        BigDecimal saldo = monto;

        for (int i = 1; i <= getCantidadCuotas(); i++) {
            LocalDate fechaVencimiento = prestamo.getFechaOtorgamiento().plusMonths(i);
            BigDecimal intereses = saldo.multiply(tasaMensual).setScale(2, RoundingMode.HALF_UP);
            BigDecimal amortizacion = cuotaFija.subtract(intereses);
            saldo = saldo.subtract(amortizacion);
            cuotas.add(new Cuota(
                    i,
                    fechaVencimiento,
                    cuotaFija,
                    intereses,
                    amortizacion,
                    saldo.compareTo(BigDecimal.ZERO) > 0 ? saldo : BigDecimal.ZERO
            ));
        }
        return cuotas;
    }
```

Clase Prestamo

La clase Prestamo representa un préstamo otorgado a un cliente, contiene toda la información relevante a un préstamo, incluyendo su número identificador, monto total, cantidad de cuotas, fecha de otorgamiento, estado actual, esquema de financiación, y los usuarios involucrados en su generación y autorización. Además, mantiene la relación con el cliente solicitante, los garantes y las cuotas generadas.

```
public class Prestamo {
    private Long idPrestamo;
    private Long nroPrestamo;
    private LocalDate fechaOtorgamiento;
    private BigDecimal monto;
    private Integer cantidadCuota;
    private Usuario usuarioAutorizador;
    private Usuario usuarioGenerador;
    private final List<Garante> garantes = new ArrayList<>();
    private EstadoPrestamo estado;
    private Cliente cliente;
    private EsquemaFinanciacion esquema;
    private List<Cuota> cuotas = new ArrayList<>();

    public Prestamo(Long nroPrestamo, BigDecimal monto, EsquemaFinanciacion esquema) {
        if (monto == null || monto.compareTo(BigDecimal.ZERO) <= 0) {
            throw new IllegalArgumentException("El monto debe ser mayor que cero");
        }
        this.nroPrestamo = Objects.requireNonNull(nroPrestamo, "El número de préstamo no puede ser nulo");
        this.monto = monto;
        this.esquema = Objects.requireNonNull(esquema, "El esquema de financiación no puede ser nulo");
        this.estado = EstadoPrestamo.PENDIENTE;
    }

    public BigDecimal totalPagado() {
        return cuotas.stream()
                .filter(c -> c.getEstado().equals(EstadoCobro.COBRADO)) // suponiendo que exista ese método
                .map(Cuota::getMontoCuota)
                .reduce(BigDecimal.ZERO, BigDecimal::add);
```

```
    }
    // Métodos getters y setters
}
```

Clase Cuota

Representa una cuota correspondiente a un préstamo.

Esta clase encapsula los datos necesarios para registrar y gestionar una cuota individual, incluyendo su número, monto total, intereses, amortización de capital, saldo pendiente, fechas de vencimiento y de pago, así como su estado de cobro.

```
public class Cuota {
    private Long idCuota;
    private Integer numeroCuota;
    private BigDecimal montoCuota;
    private LocalDate fechaVencimiento;
    private LocalDate fechaPago;
    private BigDecimal intereses;
    private BigDecimal amortizacionCapital;
    private BigDecimal saldoPendiente;
    private EstadoCobro estado;

    public Cuota() {
        this.estado = EstadoCobro.NO_COBRADO;
    }

    public Cuota(Integer numeroCuota, LocalDate fechaVencimiento, BigDecimal monto,
                 BigDecimal intereses, BigDecimal amortizacionCapital, BigDecimal saldoPendiente) {
        this.numeroCuota = numeroCuota;
        this.fechaVencimiento = fechaVencimiento;
        this.montoCuota = monto;
        this.intereses = intereses;
        this.amortizacionCapital = amortizacionCapital;
        this.saldoPendiente = saldoPendiente;
    }
    // Métodos getters y setters
}
```

Interfaz MedioPago

La interfaz MedioPago representa el contrato que deben cumplir todos los mecanismos utilizados para registrar un cobro en el sistema. Define el comportamiento básico que debe implementar cualquier medio de pago (por ejemplo, efectivo, transferencia bancaria, débito automático, etc.), asegurando una integración homogénea con el proceso de gestión de pagos.

Métodos definidos:

- **void registrarCobro(Pago pPago) throws LogicaException**

  Registra un cobro utilizando este medio de pago. Puede lanzar una excepción de tipo

LogicaException si se produce algún error lógico durante el proceso (por ejemplo, si el pago ya fue registrado, si los datos son inválidos o si el medio no está disponible).

- **String descripcion()**

  Devuelve una descripción textual del medio de pago. Esta información puede ser utilizada en interfaces gráficas, reportes o logs para identificar el tipo de medio utilizado.

```
1  public interface MedioPago {
2      void registrarCobro(Pago pPago) throws LogicaException;
3      String descripcion();
4  }
```

Clase DebitoAutomatico

La clase DebitoAutomatico implementa la interfaz MedioPago y representa un medio de cobro automatizado mediante el débito directo desde una cuenta bancaria. Su propósito es registrar pagos en el sistema utilizando información bancaria del titular.

```
 1  public class DebitoAutomatico implements MedioPago {
 2      private String banco;
 3      private String numeroCuenta;
 4      private String cbu;
 5      private String titular;
 6      private LocalDate fechaOperacion;
 7
 8      public DebitoAutomatico(String banco, String numeroCuenta,
 9  String cbu, String titular, LocalDate fechaOperacion) {
10          this.banco = banco;
11          this.numeroCuenta = numeroCuenta;
12          this.cbu = cbu;
13          this.titular = titular;
14          this.fechaOperacion = fechaOperacion;
15      }
16
17      @Override
18      public void registrarCobro(Pago pPago) throws LogicaException {
19          validarPago(pPago);
20          // Logica
21      }
22      @Override
23      public String descripcion() {
24          return String.format("Debito automatico en %s - número de cuenta: %s - fecha %s",
25                  banco, numeroCuenta, Util.formatearFecha(fechaOperacion));
26      }
27      private void validarPago(Pago pago) throws LogicaException {
28          if (pago == null) {
29              throw new LogicaException("El pago no puede ser nulo");
30          }
31          if (pago.getMontoPagado().compareTo(BigDecimal.ZERO) <= 0) {
32              throw new LogicaException("El monto pagado debe ser mayor a cero");
33          }
34      }
35      // Métodos getters y setters
36  }
```

Clase LogicaException

La clase LogicaException es una excepción personalizada que extiende RuntimeException y se utiliza para representar errores lógicos en la aplicación.

```
1  public class LogicaException extends RuntimeException {
2      public LogicaException(String mensaje) {
3          super(mensaje);
4      }
5  }
```

Funcionalidad principal:

Permite encapsular errores relacionados con las reglas de negocio (por ejemplo, validaciones fallidas o estados inválidos) de una manera clara y diferenciada de las excepciones técnicas o del sistema.

Clase AuthException

La clase AuthException es una excepción personalizada que extiende Exception y se utiliza para representar errores relacionados con la autenticación dentro de la aplicación.

```
1  public class AuthException extends Exception {
2      public AuthException(String message) {
3          super(message);
4      }
5  }
```