Documento Arquitectura de Software

Fecha: 06/02/2024

Aprobadores de documentos

Nombre Aprovador	Rol
Armando Augusto Cabrera Silva	Docente Arquitectura de Software

Revisores de documentos:

Nombre Revisor	Rol
Tais Belén Balcázar Alban	Diseño UX/UI
José Miguel Regalado Valarezo	Líder del Grupo / Desarrollo Front-end
Miguel Alejandro Álvarez Lima	Desarrollo Back-end
Jeremy Fabricio Jaramillo Peña	Q/A Tester
Carlos Agustin Salas Churo	Analista de Negocio

Resumen de Cambios:

Versión	Fecha	Creado por	Descripción corta de los cambios
0.1	01.10.2024	Tais Belén Balcázar Alban	Borrador inicial, introducción al Modelo de Arquitectura 4+1 de Krutchen.
0.2	10.10.2024	José Miguel Regalado Valarezo	Cambios según comentarios, explicación detallada de la vista lógica (una de las vistas 4+1).
0.3	18.10.2024	Miguel Alejandro Álvarez Lima	Se añadieron capítulos sobre la vista de proceso (otra vista del modelo de Krutchen), enfocándose en las interacciones en tiempo de ejecución.
0.4	25.10.2024	Jeremy Fabricio Jaramillo Peña	Se añadió información sobre la vista de desarrollo, enfocándose en los componentes y sus relaciones dentro del sistema.
0.5	01.11.2024	Carlos Agustin Salas Churo	Información sobre la vista física, detallando la arquitectura de hardware del sistema y los modelos de despliegue.
1.0	08.11.2024	Tais Belén Balcázar Alban	Primera versión publicada, inclusión de la vista de escenarios e integración de todas las vistas 4+1 en el sistema.
1.1	15.11.2024	José Miguel Regalado Valarezo	Documentación de nuevas adaptaciones y refinamientos en las vistas lógica y física, añadiendo consideraciones de casos de uso.
1.2	22.11.2024	Miguel Alejandro Álvarez Lima	Actualización de las vistas de desarrollo y escenarios, añadiendo más detalles sobre el comportamiento del sistema y los flujos de casos de uso.
1.3	29.11.2024	Jeremy Fabricio Jaramillo Peña	Refinamiento de las descripciones de las vistas de proceso y física para mayor claridad.
1.4	06.12.2024	Carlos Agustin Salas Churo	Finalización de la integración de las 4+1 vistas y asegurando la alineación con los requisitos del sistema.
1.5	13.12.2024	Tais Belén Balcázar Alban	Actualizaciones reflejando cambios en el diseño arquitectónico, alineados con los comentarios de los usuarios sobre el

			rendimiento del sistema.
1.6	20.12.2024	José Miguel Regalado Valarezo	Ajustes menores en las vistas física y de proceso, asegurando mejor alineación con el despliegue.
1.7	03.01.2025	Miguel Alejandro Álvarez Lima	Se añadieron aclaraciones adicionales a la vista de desarrollo, reforzando los aspectos de escalabilidad del sistema.
1.8	10.01.2025	Jeremy Fabricio Jaramillo Peña	Actualización de la vista lógica para reflejar nuevos requisitos del sistema, incluyendo flujos de datos e interacciones.
1.9	17.01.2025	Carlos Agustin Salas Churo	Últimas actualizaciones en todas las vistas, asegurando consistencia y completitud del modelo 4+1 de Krutchen en la documentación del proyecto.
1.10	24.01.2025	Tais Belén Balcázar Alban	Refinamiento de las descripciones finales, incorporación de retroalimentación de usuarios para mejorar la comprensión de la arquitectura del sistema.
1.11	31.01.2025	José Miguel Regalado Valarezo	Ajustes finales menores, reflejando nuevos conocimientos provenientes de pruebas e integración del sistema.
1.12	06.02.2025	Miguel Alejandro Álvarez Lima	Revisión final y actualizaciones para asegurar la claridad y relevancia del modelo 4+1 de vistas para el diseño global del sistema.

Tabla de Contenidos

1. INTRODUCCIÓN

1.1. Objetivo

Este documento proporciona una descripción general arquitectónica completa del sistema, utilizando varias vistas arquitectónicas diferentes para representar aspectos individuales del sistema. Su objetivo es capturar y transmitir las decisiones arquitectónicas importantes que se han tomado en el sistema.

1.2. Alcance

Este documento registra la gestión de versiones del sistema **Car-Loan**, detallando las actualizaciones realizadas desde su fase inicial hasta la presentación final. Incluye los cambios implementados, las fechas de actualización y los responsables de cada modificación, garantizando un control estructurado del desarrollo y asegurando la calidad de la Arquitectura utilizada en el sistema.

1.3. Referencias

#	Documento	Esquema de Contenidos
[REF1]	Software Architecture in Practice	Bass, L., Clements, P., & Kazman, R. (2012). "Software Architecture in Practice". Addison-Wesley.
[REF2]	The Unified Modeling Language User Guid	Booch, G., Rumbaugh, J., & Jacobson, I. (2005). "The Unified Modeling Language User Guide". Addison-Wesley.
[REF3]	ebMS3 Core	OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features. OASIS Standard. 1 October 2007.
[REF4]	Domibus plugin cookbook	Technical manual on Domibus plugin development.

[REF5]	WS-Policy Specification	The Web Services Framework provides a general-purpose model and corresponding syntax to describe the policies of entities in a Web services-based system.
[REF6]	Spring Security	Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. It can easily be extended to meet custom requirements.
[REF7]	Bcrypt	*Provos, Niels; Mazières, David; Talan Jason Sutton 2012 (1999). "A Future-Adaptable Password Scheme". Proceedings of 1999 USENIX Annual Technical Conference: 81–92.*
[REF8]	JMS	The Java Message Service (JMS) API is a Java Message Oriented Middleware API for sending messages between two or more clients.
[REF9]	e-CODEX	The e-CODEX project improves the cross-border access of citizens and businesses to legal means in Europe and furthermore creates the interoperability between legal authorities within the EU.
[REF10]	Java Servlet 3.0	A Java servlet is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers. Such Web servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET.
[REF11]	Xtext	Xtext is a framework for development of programming languages and domain-specific languages.
[REF12]	SOAP	Simple Object Access Protocol
[REF13]	HTTP Chunking	A mechanism by which data is broken up into a number of chunks when sent over an HTTP

1.4. Descripción general del contenido del documento

Después de resumir la representación arquitectónica, los objetivos y las limitaciones, este documento describe el sistema utilizando varias vistas arquitectónicas (caso de uso, lógica, proceso, implementación y datos) y luego concluye con consideraciones de tamaño, rendimiento y calidad.

2. Representación Arquitectónica

Las siguientes secciones del documento describen los objetivos y restricciones arquitectónicas del sistema Car-Loan utilizando el Modelo 4+1 de Vistas Arquitectónicas de Philippe Kruchten. Este enfoque permite representar la arquitectura desde diferentes perspectivas para satisfacer las necesidades de los distintos interesados (stakeholders).

- Casos de Uso: Proporcionan ejemplos concretos que demuestran cómo las diferentes vistas trabajan juntas para satisfacer los requisitos del sistema. Los casos de uso ayudan a validar y comunicar el diseño al ilustrar comportamientos clave del sistema.

- Vista Lógica: Describe la funcionalidad del sistema desde el punto de vista de los usuarios y desarrolladores. Muestra cómo se organiza el software en módulos o componentes que colaboran para cumplir con los requisitos funcionales.

- Vista de Procesos: Explica cómo interactúan los procesos del sistema para cumplir con los requisitos funcionales y no funcionales, como el rendimiento, la concurrencia y la comunicación entre procesos.

- Vista de Desarrollo: Representa la organización del software desde la perspectiva del equipo de desarrollo. Detalla la estructura del código, los módulos, bibliotecas y cómo se empaqueta el sistema.

- Vista de Despliegue: Describe cómo se implementa el sistema en la infraestructura física, mostrando servidores, redes y cómo se distribuyen los componentes en el hardware.

3. Objetivos y Limitaciones Arquitectónicas

Se han identificado los siguientes requisitos no funcionales que afectan a la solución arquitectónica del sistema Car-Loan:

Requisitos No Funcionales	Descripción
Adaptabilidad	El sistema debe integrarse fácilmente con flujos de trabajo y protocolos existentes.
Escalabilidad	La arquitectura debe soportar el aumento en la carga de usuarios y transacciones sin pérdida significativa de rendimiento.
Rendimiento	El tiempo de respuesta debe ser rápido para todas las operaciones críticas, manteniendo una experiencia de usuario fluida.
Confiabilidad	El sistema debe ser resistente a fallos y garantizar la disponibilidad continua de los servicios.
Mantenibilidad	La arquitectura debe facilitar la identificación y corrección de errores, además de permitir actualizaciones futuras con facilidad.

4. SEGURIDAD

4.1. Introducción

El punto de acceso Krutchken proporciona seguridad integral conforme a las especificaciones y mejores prácticas de la industria implementadas en su arquitectura basada en microservicios. Además, Krutchken se integra de forma fluida en dominios de seguridad existentes, permitiendo un enfoque flexible y escalable para la gestión de la seguridad.

4.2. Combinación de Servicios en Microarquitectura

Dado que no se pueden hacer suposiciones sobre la arquitectura de seguridad entre los

servicios dentro de Krutchken, la integración en la infraestructura existente debe realizarse mediante los módulos de seguridad específicos. Los módulos predeterminados proporcionan una base sólida de autenticación y autorización, aunque pueden ser extendidos o personalizados para cumplir con los requisitos de seguridad adicionales según las necesidades del cliente.

4.3. Comunicación entre Servicios Distribuidos

La comunicación entre los distintos servicios de la arquitectura de Krutchken puede cumplir con todos los requisitos de seguridad especificados en los perfiles de integración. Esto se maneja mediante archivos de configuración de políticas de seguridad (por ejemplo, WS-Policy y PMode). Los marcos de seguridad que se aplican en las comunicaciones entre microservicios incluyen tecnologías avanzadas como Apache CXF, WSS4J y Santuario para garantizar la protección de los datos y las transacciones.

4.3.1. Configuración de Certificado

La ubicación y las credenciales de los certificados públicos y privados utilizados en las comunicaciones seguras entre microservicios se configuran en los archivos de configuración de Krutchken

4.3.2. Autenticación de Cliente

La autenticación de cliente (SSL bidireccional) se gestiona mediante certificados de cliente configurados en el archivo de configuración de Spring, clientauthentication.xml. Las conexiones TLS seguras entrantes son terminadas en un servidor proxy, como Nginx o Apache, y deben configurarse de acuerdo con las directrices del servidor proxy utilizado en la infraestructura de Krutchken.

4.4. Seguridad entre Servicios Externos

La seguridad entre los servicios externos e internos de Krutchken se maneja mediante los mismos mecanismos de seguridad implementados en la comunicación entre servicios internos, utilizando tecnologías avanzadas de cifrado y autenticación robusta para asegurar la integridad y confidencialidad de las transacciones y datos.

4.5. Sitios Administrativos

El acceso a la página de administración de Krutchken está protegido por un sistema de autenticación multifactor (MFA) que utiliza un nombre de usuario y contraseña como primer nivel de seguridad. Las credenciales son gestionadas por un servicio central de autenticación basado en microservicios, lo que permite conectar múltiples proveedores de autenticación, como OAuth2 o SSO, de manera flexible y escalable. Las credenciales de los usuarios se almacenan de forma segura en un servicio de gestión de identidad, utilizando algoritmos de cifrado avanzados como AES-256 para proteger la base de datos.

La integración con un sistema de autenticación externo, como LDAP o Active Directory, es completamente compatible y puede realizarse mediante configuraciones detalladas de Spring Security en la arquitectura de Krutchken.

Descargo de Responsabilidad de Seguridad

A pesar de las medidas de seguridad implementadas por Krutchken, el usuario es responsable de adoptar prácticas adicionales de seguridad siguiendo las normativas y mejores prácticas recomendadas. Esto incluye la implementación de firewalls, controles de acceso a redes, cifrado de bases de datos y sistemas de archivos, así como la gestión rigurosa de contraseñas y autenticación de usuarios. DIGIT no se hace responsable por incidentes de seguridad que ocurran como resultado de que el usuario no adopte estas recomendaciones adicionales.

5. VISTA DE CASO DE USO

Esta sección proporciona una representación de los casos de uso relevantes para la arquitectura.

5.1. ¿Qué es un diagrama de casos de uso?

Un diagrama de casos de uso es una representación gráfica empleada en el modelado de sistemas para describir las interacciones entre los actores (usuarios, sistemas externos u otros componentes) y las funcionalidades principales que ofrece un sistema. Este tipo de diagrama, basado en el estándar UML (Lenguaje Unificado de Modelado), se utiliza para documentar *qué hace el sistema* desde la perspectiva de los usuarios, sin entrar en detalles sobre *cómo se implementa*.

5.2. ¿Para qué sirve un Diagrama de Casos de Uso?

El diagrama de casos de uso es una herramienta clave para el desarrollo y documentación de sistemas, ya que permite:

1. *Especificar requisitos funcionales*: Identificar y detallar las funcionalidades principales que debe cumplir el sistema.
2. *Facilitar la comunicación*: Servir como un medio común entre los interesados, desarrolladores y analistas para garantizar la comprensión de los requisitos.
3. *Definir el alcance del sistema*: Determinar los límites del sistema, especificando qué incluye y qué excluye.
4. *Identificar actores clave*: Describir quiénes interactuarán con el sistema, ya sean usuarios humanos o sistemas externos.
5. *Priorizar funcionalidades*: Ayudar a evaluar y clasificar las funcionalidades en función de su importancia o impacto para los usuarios.

5.3. Estructura de un Diagrama de Casos de Uso

La estructura de un diagrama de casos de uso incluye los siguientes componentes principales:

1. Actores:

- *Representan las entidades externas que interactúan con el sistema.*
- *Pueden ser personas, organizaciones o sistemas.*
- *Se representan gráficamente como una figura humana o una etiqueta que los identifica.*
- *Ejemplo: Usuario, Administrador, Sistema Externo.*

2. Casos de Uso:

- *Representan las funcionalidades o servicios específicos que el sistema proporciona a los actores.*
- *Se representan como óvalos con un nombre que describe la acción.*

- *Ejemplo: Registrar Usuario, Generar Informe, Procesar Pago.*

3. Relaciones:

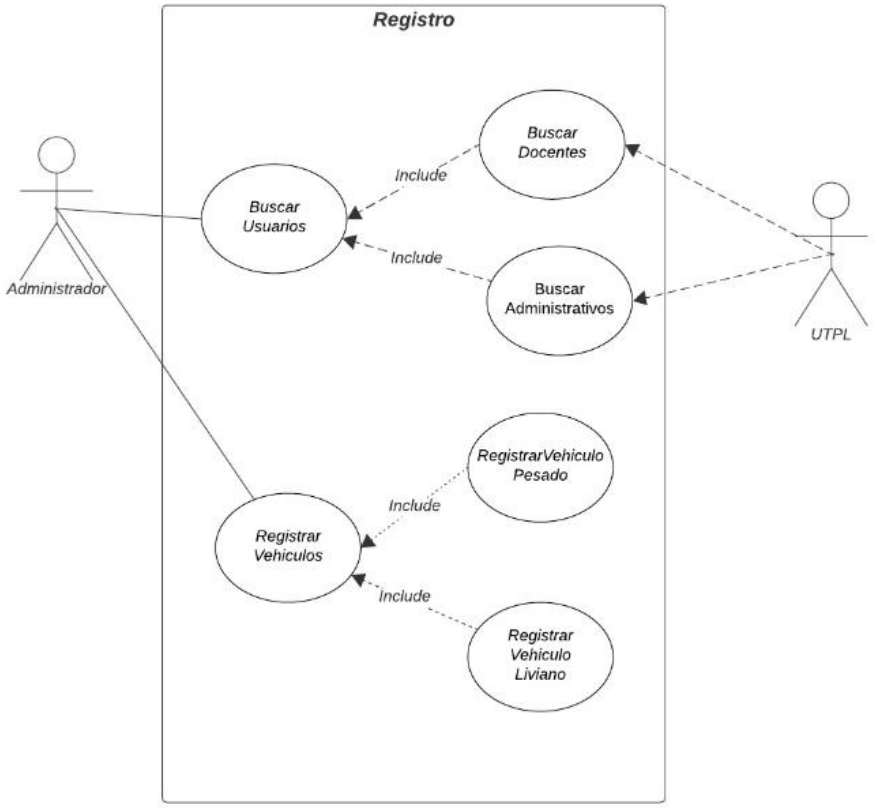
- *Asociación: Representa la interacción entre un actor y un caso de uso (línea sólida).*
- *Inclusión (<>): Indica que un caso de uso incluye la funcionalidad de otro, generalmente para evitar redundancia.*
- *Extensión (<>): Señala un caso de uso opcional que extiende la funcionalidad de otro principal, dependiendo de ciertas condiciones.*
- *Generalización: Define relaciones de herencia entre actores o entre casos de uso.*

4. Sistema:

- *Representa los límites del sistema modelado.*
- *Se ilustra como un rectángulo que contiene los casos de uso.*

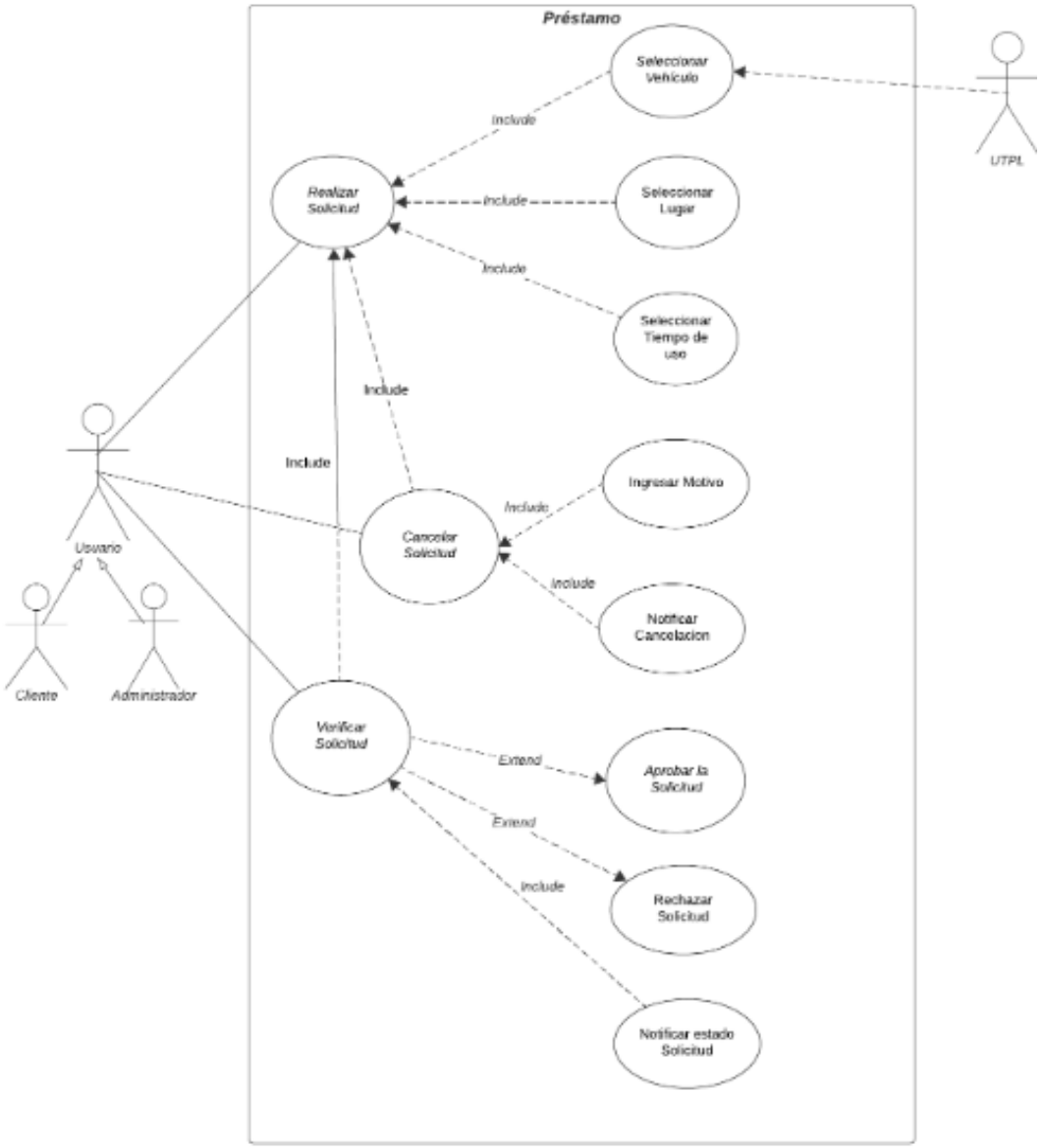
5.4. Casos de Uso

5.4.1. Registro



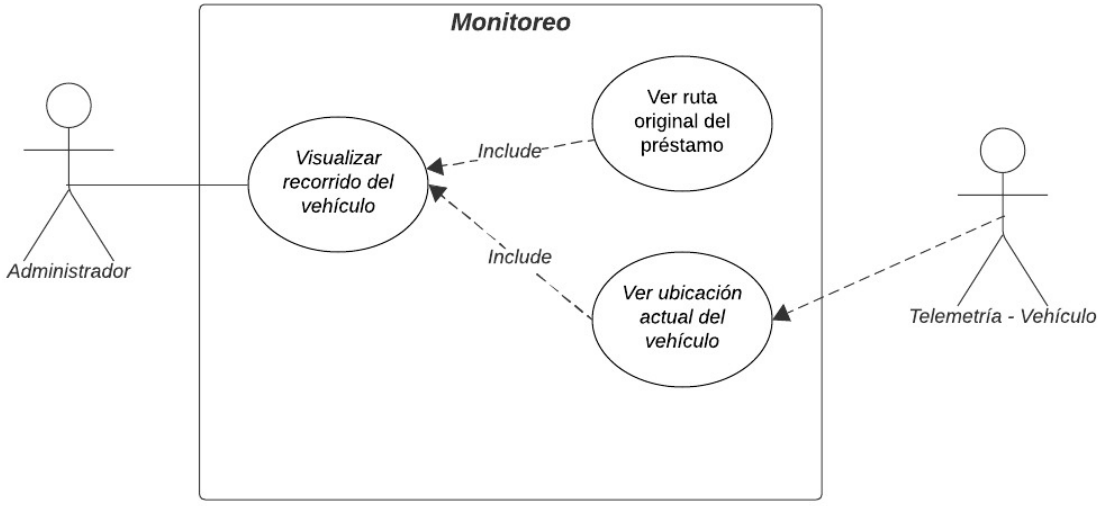
NOMBRE	REGISTRO
ACTORES	- Administrador - UTPL (fuente de datos)
FLUJO NORMAL	1. El administrador busca a los docentes o administrativos en UTPL. 2. UTPL devuelve la información de los usuarios buscados al administrador. 3. El administrador registra un nuevo vehículo en UTPL.
FLUJO ALTERNATIVO	1ª. Si el administrador no encuentra coincidencias en las búsquedas, UTPL muestra un mensaje de "no se encontraron resultados" al administrador. 2ª. Si el administrador ingresa información incorrecta o incompleta para el registro de vehículos, UTPL muestra un mensaje de error y solicita correcciones al administrador.

5.4.2. Prestamos



NOMBRE	PRÉSTAMO
ACTORES	- Cliente - Administrador - UTPL (fuente de datos)
FLUJO NORMAL	1. El cliente ingresa una solicitud de préstamo de vehículo al administrador. 2. El administrador aprueba la solicitud ingresada en UTPL. 3. UTPL notifica el estado de solicitud al cliente.
FLUJO ALTERNATIVO	1ª. Si el cliente cancela la solicitud ingresada, UTPL notifica el estado de la solicitud al administrador. 2ª. Si el administrador rechaza la solicitud enviada por el cliente, UTPL notifica el estado de la solicitud al cliente.

5.4.3. Monitoreo



NOMBRE	MONITOREO
ACTORES	- Administrador - Telemetría-Vehículo (fuente de datos)
FLUJO NORMAL	1. El administrador monitorea los vehículos de UTPL. 2. Telemetría-Vehículo muestra la ruta y ubicación del vehículo.
FLUJO ALTERNATIVO	1ª. Si la telemetría no está disponible, se muestra un mensaje de error al administrador.

5.4.4. Devolución

NOMBRE	DEVOLUCIÓN
ACTORES	- Guardia - Administrador
FLUJO NORMAL	1. El guardia ingresa imágenes del vehículo, hace el checklist del vehículo y notifica al administrador. 2. El administrador acepta la devolución de la solicitud y actualiza los datos del vehículo en UTPL. 3. UTPL muestra los datos actualizados al administrador. 4. El administrador genera un informe de solicitud y revisa el historial de solicitudes.
FLUJO ALTERNATIVO	1ª. Si se detecta algún daño o anomalía en el vehículo durante la validación, el guardia solicita una revisión adicional al administrador. 2ª. Si la información ingresada para actualizar los datos del vehículo es inconsistente, UTPL solicita una verificación al administrador.

6. Vista Lógica

6.1. Descripción general

Describe la funcionalidad del sistema desde el punto de vista de los usuarios y desarrolladores. Muestra cómo se organiza el software en módulos o componentes que colaboran para cumplir con los requisitos funcionales.

6.2. Diagrama de Clases

6.2.1. ¿Qué es un diagrama de clases?

Un *diagrama de clases* es un tipo de diagrama estático en la Programación Orientada a Objetos (POO) que modela la estructura de un sistema mostrando:

1. *Clases*: Representan las entidades o conceptos del sistema.
2. *Atributos*: Definen las características o propiedades de cada clase.
3. *Métodos*: Especifican los comportamientos o acciones que puede realizar la clase.
4. *Relaciones*: Muestran las conexiones entre las clases, como asociaciones, composiciones o herencias.

Es ampliamente utilizado en la fase de diseño de software para definir cómo interactúan las diferentes partes del sistema

6.2.2. Estructura general de un diagrama de clases

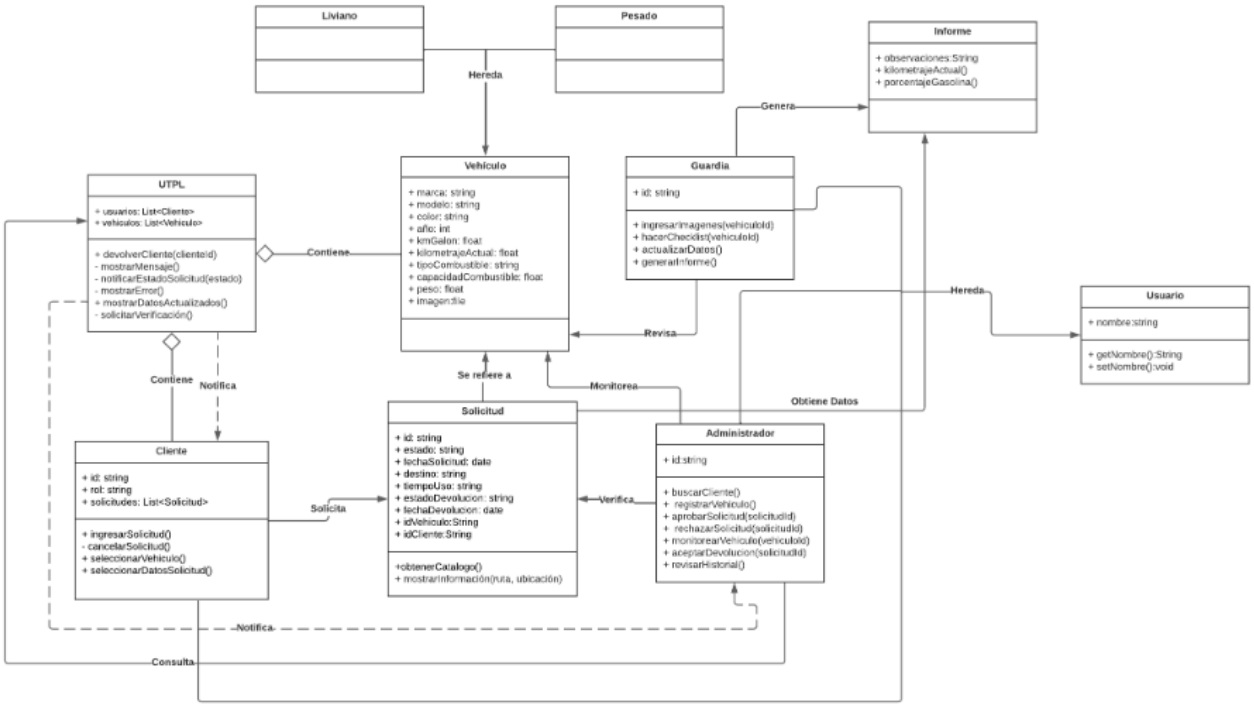
Un diagrama de clases sigue la siguiente estructura:

1. *Clases*:
    - Representadas como rectángulos divididos en tres secciones:
        - *Nombre* de la clase (parte superior).
        - *Atributos* (parte media).
        - *Métodos* (parte inferior).
2. *Relaciones entre clases*:
    - *Asociaciones*: Indican conexiones entre clases.
    - *Composición*: Una clase contiene a otra como parte esencial.
    - *Agregación*: Una clase contiene a otra, pero no de manera esencial.
    - *Herencia*: Una clase hereda propiedades y métodos de otra.
3. *Visibilidad de atributos y métodos*:
    - Público: Accesible desde cualquier lugar.
    - Privado: Accesible solo dentro de la clase.

- Protegido: Accesible dentro de la clase y sus subclases.

4. *Cardinalidad*: Define la cantidad de objetos relacionados, como 1:1, 1:N o N:M.

6.2.3. Diagrama de Clases



7. VISTA PROCESOS

7.1. Descripción general

Explica cómo interactúan los procesos del sistema para cumplir con los requisitos funcionales y no funcionales, como el rendimiento, la concurrencia y la comunicación entre procesos.

7.2. Diagramas de Secuencia

7.2.1. ¿Qué son los diagramas de secuencias?

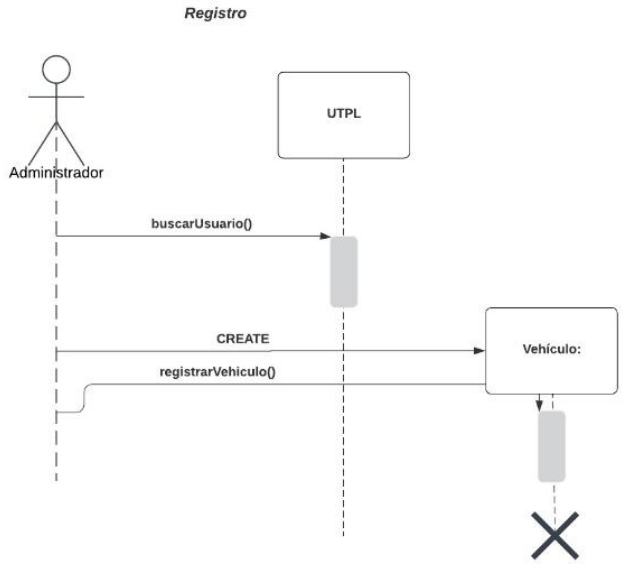
Los diagramas de secuencia son una herramienta de modelado en la Ingeniería de Software que forma parte del Lenguaje Unificado de Modelado (UML). Se utilizan para describir cómo los objetos de un sistema interactúan entre sí a través del tiempo, mostrando el flujo de mensajes o llamadas entre ellos. Estos diagramas son especialmente útiles para representar procesos y casos de uso en sistemas orientados a objetos.

7.2.2. Estructura que siguen los diagramas de secuencia

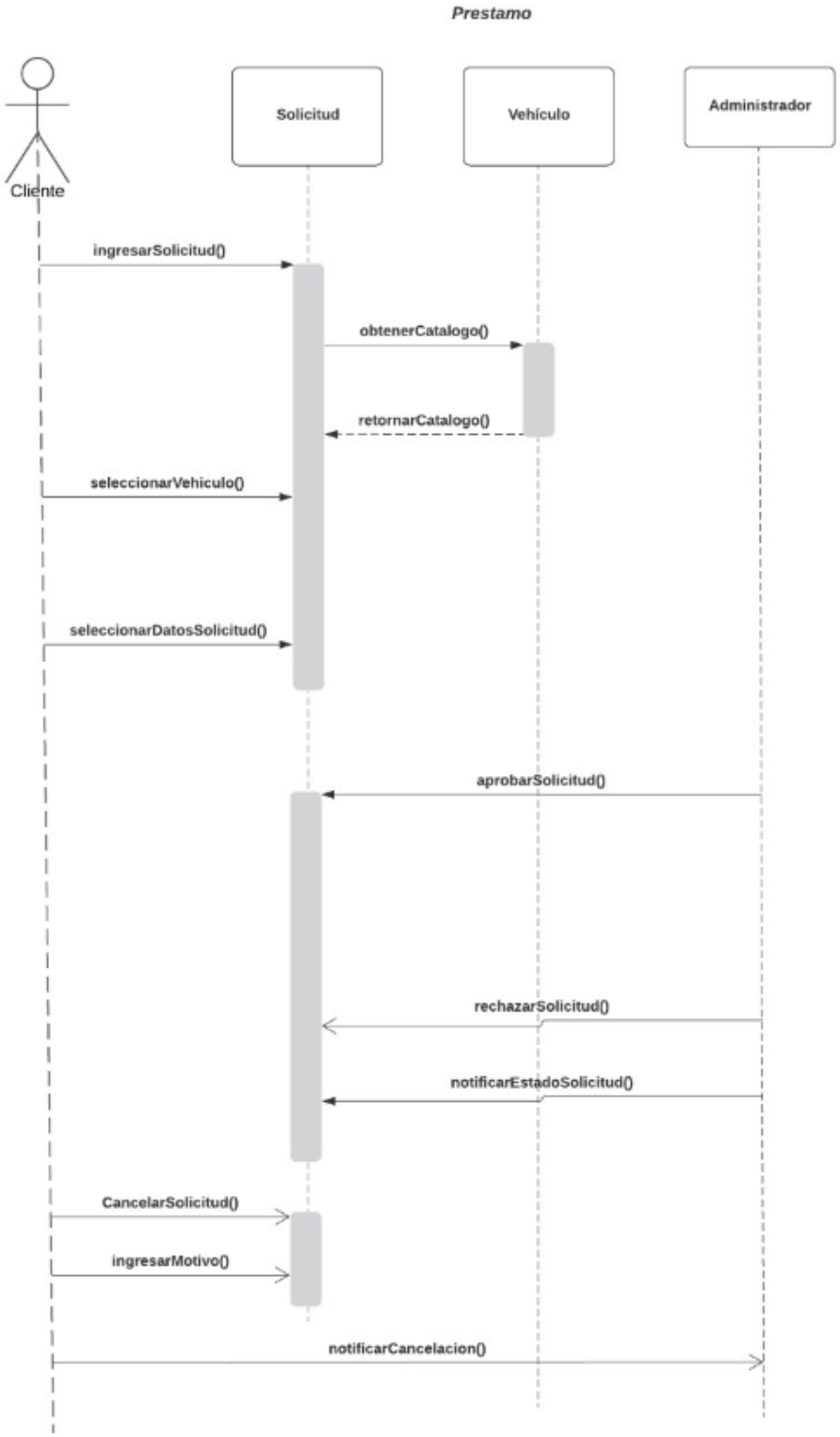
1. *Actores y Objetos*: Representan las entidades que participan en la interacción. Los actores se representan como figuras humanas, y los objetos como rectángulos.
2. *Líneas de vida*: Indican la existencia de un actor u objeto durante el proceso. Son líneas verticales que comienzan debajo de cada actor u objeto.
3. *Mensajes*: Representan la comunicación entre actores y objetos mediante flechas horizontales.
4. *Bloques de ejecución*: Rectángulos sobre las líneas de vida que representan la ejecución de un proceso o método.
5. *Tiempo*: Se representa de forma implícita en la dirección descendente del diagrama.

.

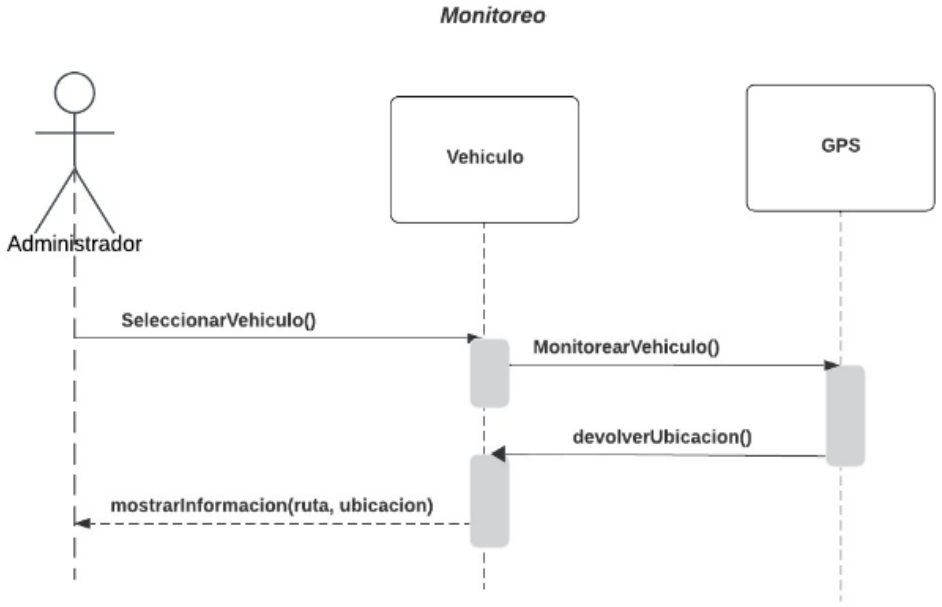
7.2.3. Registro



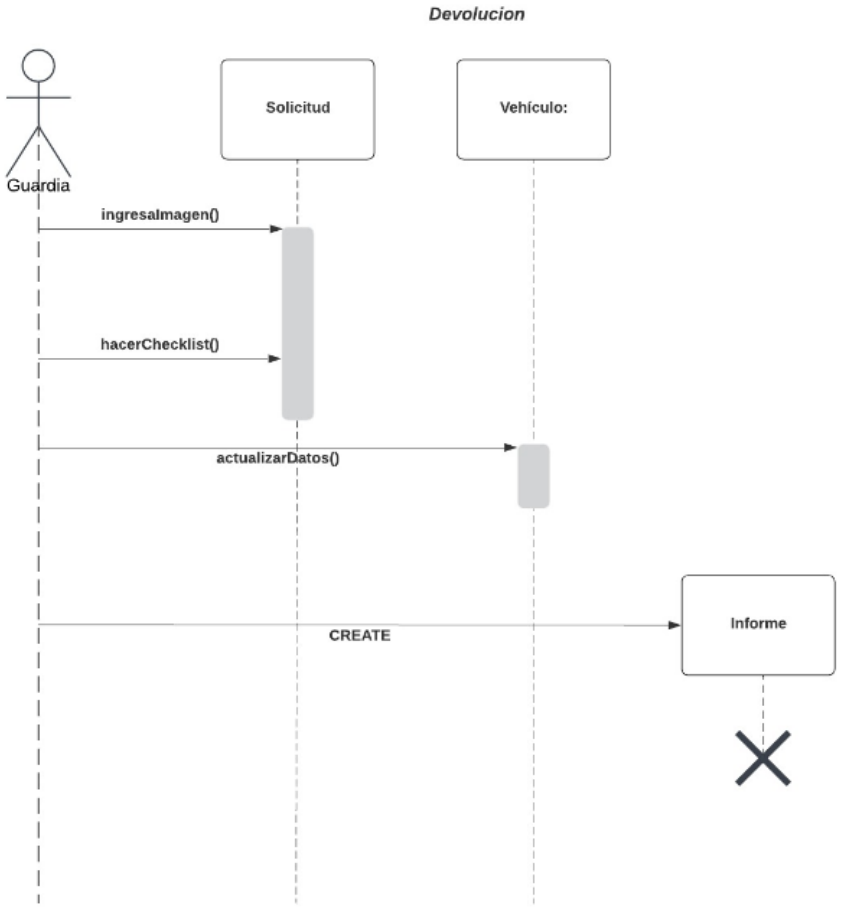
7.2.4. Préstamo



7.2.5. Monitoreo



7.2.6. Devolución



7.3. Diagramas de Robustez

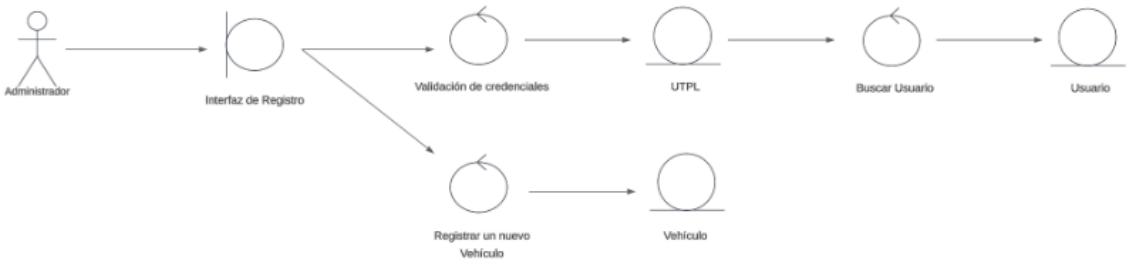
7.3.1. ¿Qué son los diagramas de robustez?

Los diagramas de robustez son una herramienta de modelado utilizada en el diseño de software para detallar cómo los actores, objetos, y procesos interactúan dentro de un sistema. Sirven como un puente entre los diagramas de casos de uso y los diagramas de diseño detallado, permitiendo identificar elementos clave del sistema como actores, controladores y entidades.

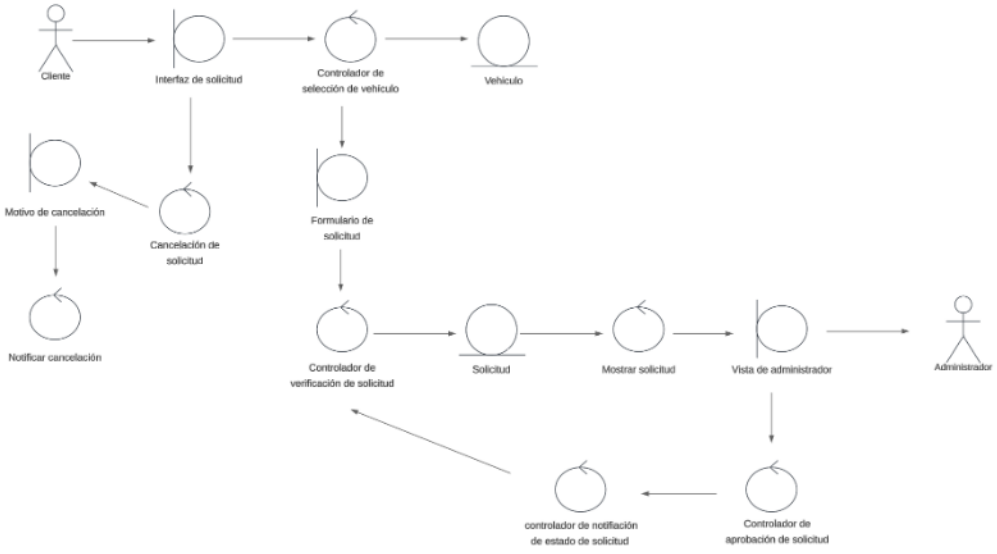
7.3.2. Estructura que siguen los diagramas de robustez

- Actores: Representan los usuarios u otros sistemas que interactúan con el sistema principal. Se dibujan como figuras humanas o íconos.
- Entidades: Son los elementos de datos o conceptos principales del dominio, representados como rectángulos.
- Controladores: Representan la lógica de la aplicación y la interacción entre actores y entidades. Se muestran como círculos u óvalos.
- Relaciones: Representan las conexiones entre actores, controladores y entidades mediante líneas con etiquetas que describen la interacción.

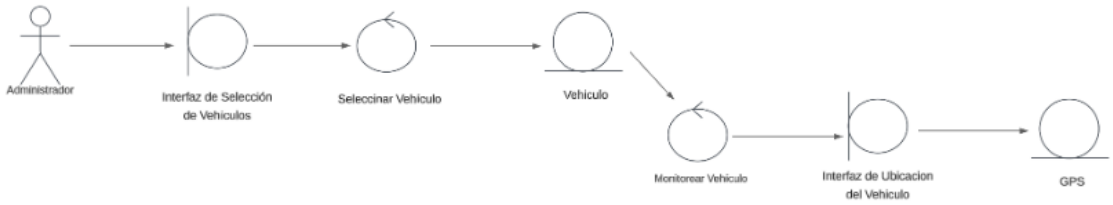
7.3.3. Registro



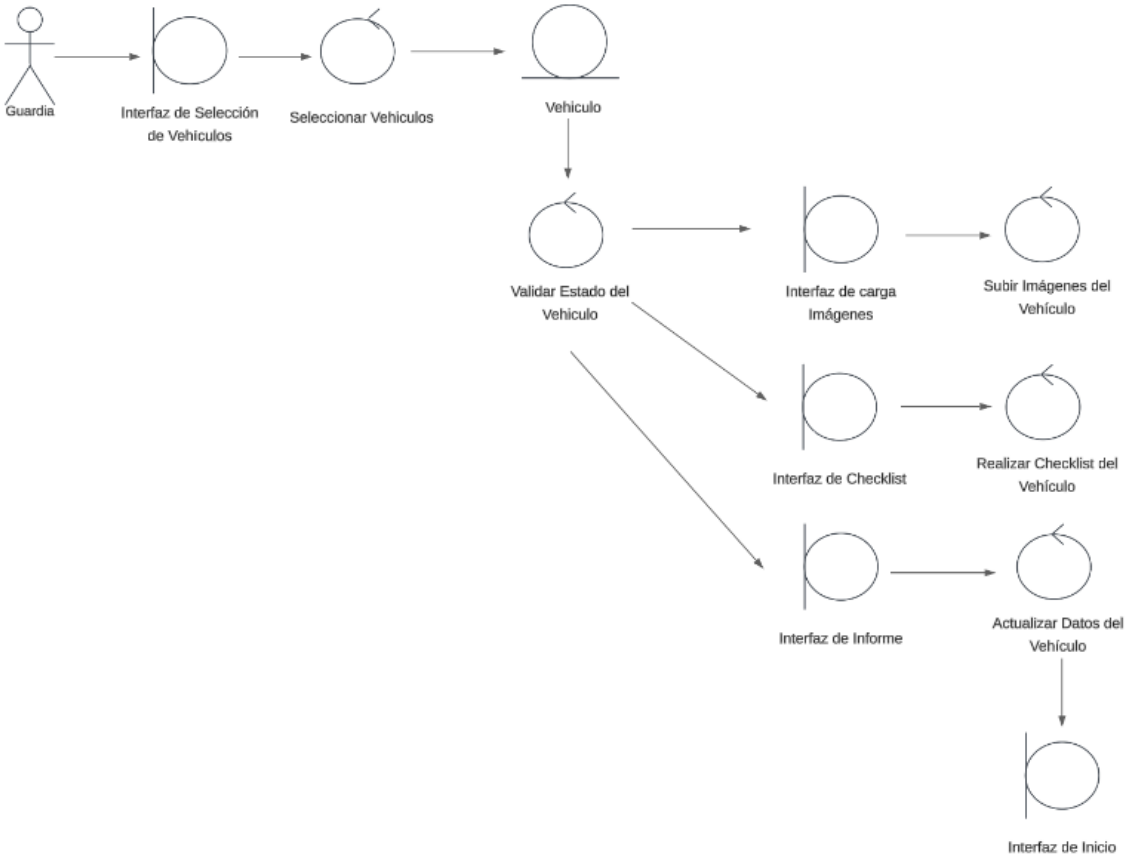
7.3.4. Prestamos



7.3.5. Monitoreo



7.3.6. Devolución



8. VISTA DE DESARROLLO

8.1. Descripción general

Representa la organización del software desde la perspectiva del equipo de desarrollo. Detalla la estructura del código, los módulos, bibliotecas y cómo se empaqueta el sistema.

8.1.1. ¿Qué es CI/CD?

En este proyecto se ha decidido usar CI/CD, que significa *Integración Continua y Entrega/Despliegue Continuo* y es un conjunto de prácticas que automatizan las etapas de desarrollo, prueba y despliegue del software. Su objetivo principal es mejorar la velocidad, calidad y confiabilidad de las entregas de software, por lo que será vital en el desarrollo del proyecto.

- *Integración Continua (CI)*:
  - Es el proceso de integrar regularmente el código de diferentes desarrolladores en un repositorio compartido.
  - Incluye la ejecución automática de pruebas para identificar errores rápidamente.
  - *Esto permitirá que el proyecto se desenvuelva con mayor agilidad.*
- *Entrega Continua (CD - Continuous Delivery)*:
  - Extiende la CI al automatizar la preparación de entregas del software en cualquier momento.
  - Garantiza que el código esté siempre en un estado listo para producción.
  - Requiere pruebas adicionales y validaciones antes del despliegue.
  - *Permite al proyecto estar siempre en condiciones para poder visualizarlo*
- *Despliegue Continuo (CD - Continuous Deployment)*:
  - Va un paso más allá y automatiza también el proceso de despliegue en producción.
  - Cada cambio que pasa las pruebas se despliega automáticamente.
  - *Automatización completa del proyecto.*

El enfoque CI/CD fomenta ciclos de desarrollo cortos, iterativos y más seguros.

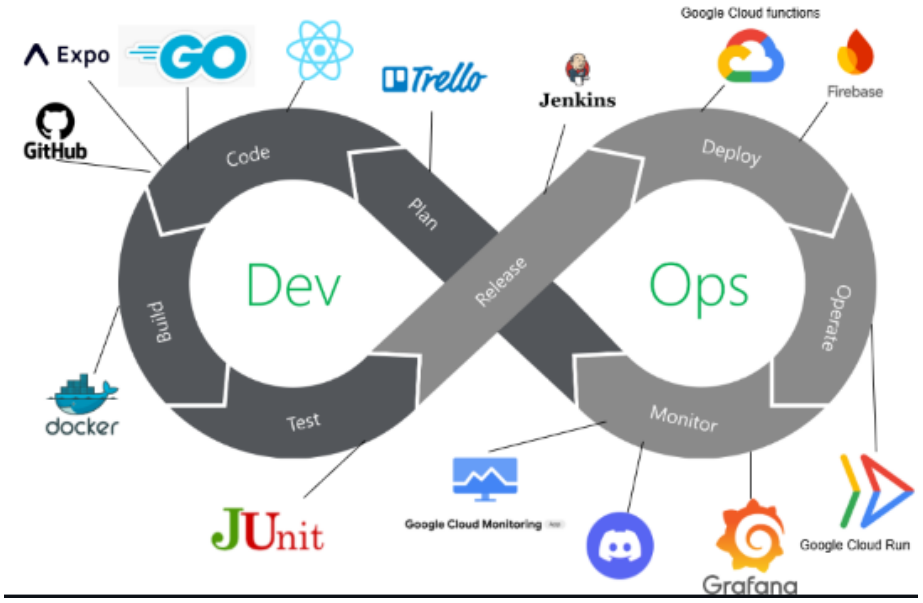
8.2. DevOps

8.2.1. ¿Qué es DevOps?

DevOps es una cultura, metodología y conjunto de prácticas que busca integrar los equipos de desarrollo (Development) y operaciones (Operations) para mejorar la colaboración, automatización y entrega continua de software. DevOps no es solo una herramienta, sino una filosofía que combina personas, procesos y tecnologías.
Se ha decidido usar DevOps para este proyecto porque presenta algunos beneficios importantes.

- *Beneficios de DevOps*:
  - Ciclos de entrega más rápidos.
  - Mayor estabilidad y calidad del software.
  - Mejor alineación entre objetivos técnicos y empresariales.

8.2.2. PipeLine



8.2.3. Tabla de Descripción

Fase	Herramienta	Descripción
Plan	Trello	Se ha usado Trello para planificar, desarrollando las historias de usuario con sus tareas, las cuales se asignan a los integrantes del equipo según su función, para esto se ha usado el apartado de *Power-ups* de Trello que nos otorga integración directa con ramas de un repositorio de github. *Turorial de integración:* https://platzi.com/tutoriales/1400-trello/3181-conectar-trello-con-github/
Código	Expo	Se ha decidido usar esta plataforma de código abierto React Native para el desarrollo móvil multiplataforma, ya que se ha observado sencillez para el desarrollo.
-	Go	Lenguaje de programación de Google para el desarrollo de una gran variedad de aplicaciones. Se escogió debido a que la creación de APIs es ágil.
-	React Native	Framework de código abierto para el desarrollo de aplicaciones para Android, iOS, macOS, Web, Windows, etc. Ha sido indicado como herramienta de uso obligatorio en cuanto al desarrollo frontend.
-	Github	GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se escogió por su sencilla integración con herramientas como Trello o Discord.
Construcción	Docker	Software de código abierto que permite crear, probar e implementar aplicaciones de forma rápida. Se escogió por su empaquetamiento de microservicios.
Pruebas	JUnit	Conjunto de bibliotecas utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. Su completitud ha sido la principal razón para ser escogido como herramienta de pruebas.
Lanzamiento	Jenkins	Servidor de automatización de código abierto escrito en Java. Compila y prueba proyectos de forma automática. Es una herramienta potente para el lanzamiento del proyecto.
Despliegue	Google Cloud	Servicio de computación serverless, ideal para funciones de uso único que se conectan hacia otros servicios. Será de gran utilidad en el proyecto para

	Functions	incorporar funciones al proyecto.
-	Firebase	Plataforma de Google para el desarrollo de aplicaciones web y aplicaciones móviles. Su integración con otros servicios de Google es vital para el desarrollo del proyecto.
Operación	Google Cloud Run	Plataforma gestionada que permite ejecutar código directamente en la infraestructura escalable de Google. Herramienta para manejar la operación del proyecto de forma efectiva.
Monitoreo	Grafana	Software libre que permite la visualización y el formato de datos métricos. Escogido por permitir un monitoreo a través de métricas fijas.
-	Discord	Servicio de mensajería instantánea y chat de voz. Se ha integrado con el repositorio github para las notificaciones y monitoreo del proyecto. *Tutorial de integración:* https://www.youtube.com/watch?v=RdifjyEuUbE&ab_channel=BGHDDevelopment
-	Google Cloud Monitoring	Servicio de Google que reduce la complejidad y ofrece soluciones para almacenamiento, estadísticas, macrodatos, aprendizaje automático y desarrollo de aplicaciones. Se escogió porque al ser de Google es de fácil integración con Firebase y Google Cloud Functions

8.2.4. Cronograma de actividades de desarrollo Backend y FrontEnd

Fase	Tarea	Backend	Frontend	Fecha Inicio	Fecha Finalización
Diseño	Diseñar modelos de datos y endpoints	✓		27/11/2024	04/12/2024
Desarrollo	Implementar endpoint de autenticación	✓		11/12/2024	18/12/2024
Desarrollo	Crear lógica de negocio para flujos principales	✓		20/12/2024	02/01/2025
Desarrollo	Integrar servicios externos	✓		02/01/2025	09/01/2025
Pruebas	Probar y validar la seguridad	✓		09/01/202	16/01/2025
Diseño	Crear componentes clave (formularios, dashboards)		✓	04/12/2024	10/12/2024
Desarrollo	Implementar autenticación y manejo de sesiones		✓	11/12/2024	18/12/2024
Desarrollo	Integrar APIs y validar datos del backend		✓	02/01/2025	09/01/2025
Desarrollo	Asegurar responsividad y accesibilidad		✓	16/01/2025	20/01/2025
Pruebas	Realizar pruebas funcionales y visuales		✓	20/01/2025	23/01/2025

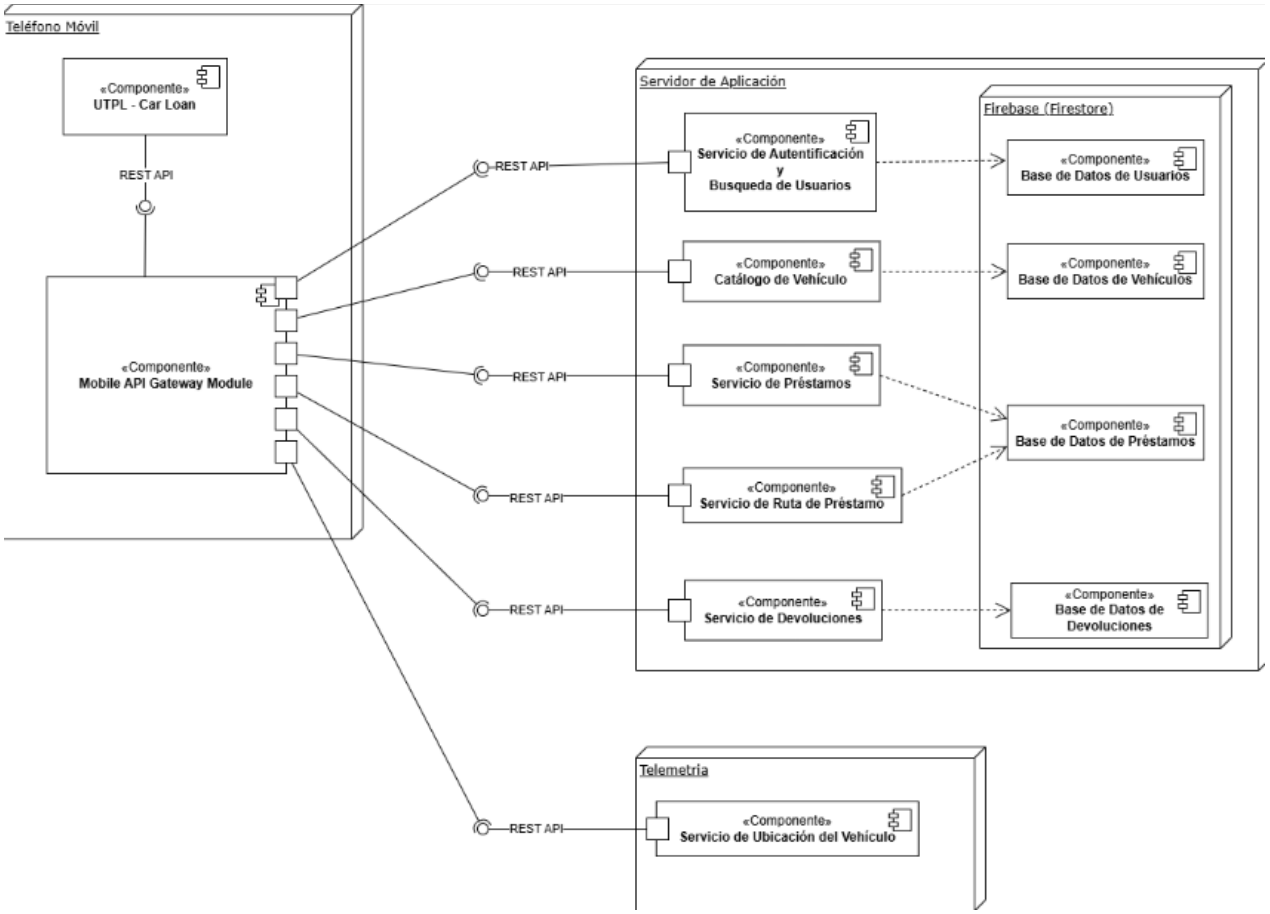
9. VISTA DE DESPLIEGUE

9.1. Descripción general

Describe cómo se implementa el sistema en la infraestructura física, mostrando servidores, redes y cómo se distribuyen los componentes en el hardware.

9.1.1. ¿Qué es un diagrama de despliegue?

Un *diagrama de despliegue* ayuda a modelar el aspecto físico de un sistema de software orientado a objetos. Modela la configuración en tiempo de ejecución en una vista estática y visualiza la distribución de los componentes en una aplicación. En la mayoría de los casos, implica modelar las configuraciones de hardware junto con los componentes de software existentes



9.1.2. Teléfono Móvil

Función: Representa el dispositivo móvil utilizado por los usuarios finales (administradores o clientes). Componentes: UTPL - Car Loan: Es la interfaz móvil (aplicación) que permite a los usuarios interactuar con el sistema. Mobile API Gateway Module: Módulo que gestiona las solicitudes REST API entre el cliente y el servidor de aplicación.

9.1.3. Servidor de Aplicación

- Función: Alberga la lógica de negocio y conecta con las bases de datos.
- Componentes: Servicio de Autenticación y Búsqueda de Usuarios: Gestiona la autenticación de usuarios y permite buscar información. Conecta con la Base de Datos de Usuarios en Firebase.
- Catálogo de Vehículo: Administra el inventario de vehículos disponibles. Conecta con la Base de Datos de Vehículos en Firebase.
- Servicio de Préstamos: Procesa las solicitudes y asignaciones de préstamos de vehículos. Conecta con la Base de Datos de Préstamos en Firebase.
- Servicio de Ruta de Préstamo: Administra la ubicación y las rutas relacionadas con los vehículos prestados.
- Servicio de Devoluciones: Gestiona la información y el estado de la devolución de los vehículos. Conecta con la Base de Datos de Devoluciones en Firebase.

9.1.4. Firebase (Firestore)

Función: Actúa como la infraestructura de base de datos en la nube, proporcionando almacenamiento para los datos críticos del sistema. Bases de Datos: Base de Datos de Usuarios: Información de autenticación y usuarios. Base de Datos de Vehículos: Información sobre el catálogo de vehículos. Base de Datos de Préstamos: Registros de préstamos realizados. Base de Datos de Devoluciones: Datos sobre las devoluciones realizadas.

9.1.5. Telemetría

U*Función: Nodo independiente para la gestión de datos de ubicación en tiempo real de los vehículos. Componente: Servicio de Ubicación del Vehículo. Ofrece información de ubicación y telemetría de los vehículos a través de REST API. Podría interactuar con otros servicios en el servidor de aplicación.*

10. Modelo de Datos

10.1. Descripción general

Un *modelo de datos* es una representación estructurada que define cómo se organizan, almacenan y manipulan los datos dentro de un sistema o base de datos

10.1.1. ¿Qué es un modelo de datos?

Un modelo de datos es una representación estructurada que define cómo se organizan, almacenan y manipulan los datos dentro de un sistema o base de datos. Es un diseño conceptual que describe:

- *Entidades: Objetos o conceptos del mundo real, como "Clientes", "Productos" o "Órdenes".*
- *Atributos: Propiedades o características de las entidades, como el "nombre" de un cliente o el "precio" de un producto.*
- *Relaciones: Conexiones o asociaciones entre entidades, como la relación entre "Clientes" y "Órdenes".*

El propósito principal de un modelo de datos es facilitar la comprensión y gestión de los datos en sistemas complejos, asegurando que estén organizados de manera lógica y eficiente.

10.1.2. Entidades

Representan los objetos principales que necesitan ser modelados dentro del sistema. Ejemplo: Un cliente puede ser una entidad en un sistema de ventas.

10.1.3. Atributos

Son las características o propiedades que describen una entidad. Ejemplo: Los atributos de un cliente pueden incluir:

- *Nombre*
- *Dirección*
- *Teléfono*

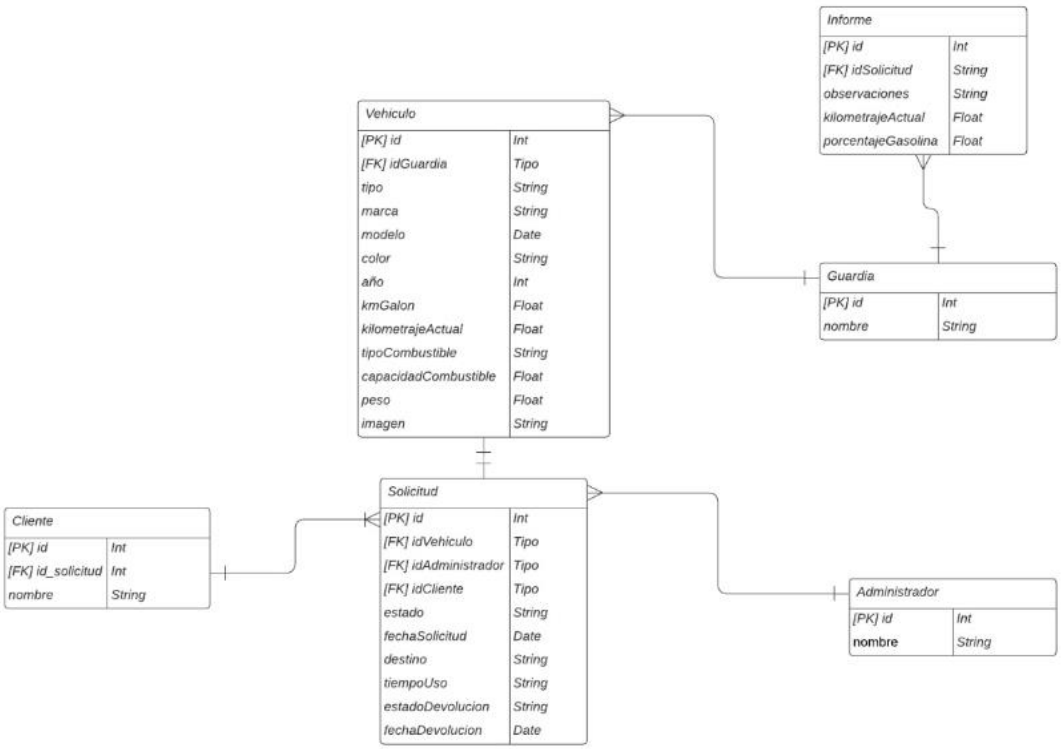
10.1.4. Relaciones

Describen cómo las entidades están conectadas entre sí y cómo interactúan. Ejemplo: Un cliente puede tener una relación de "realiza" con varias órdenes en un sistema de ventas.

10.1.5. Tipos comunes de modelo de datos

- *Modelo conceptual: Representación abstracta, como diagramas Entidad-Relación (ER).*
- *Modelo lógico: Especifica estructuras detalladas como tablas y columnas.*
- *Modelo físico: Implementación en un sistema de base de datos específico.*

10.1.6. Modelo de Datos



11. Tamaño y Rendimiento

11.1. Tamaño

Las restricciones de tamaño aplicadas a los datos intercambiados entre los sistemas de back office afectan directamente la arquitectura y configuración del sistema, aunque no a la aplicación o sus componentes en sí.

Para permitir el intercambio de archivos binarios grandes, la API del complemento admite el envío de carga útil por referencia. Esto significa que la arquitectura basada en Krutchen (vistas 4+1) permite la descarga de una carga útil desde un URI determinado. Además, las cargas útiles pueden almacenarse en el sistema de archivos en lugar de en la base de datos para evitar el procesamiento de grandes volúmenes de datos.

Dado que algunos perfiles de intercambio de mensajes no proporcionan mecanismos de división y unión de archivos grandes, la transferencia de datos está limitada por el ancho de banda y la memoria disponible.

Se pueden implementar restricciones adicionales mediante el proceso de negocio definido en las vistas de la arquitectura. Estas restricciones incluyen el tamaño máximo de una carga útil y el número máximo de cargas útiles permitidas en un mensaje.

11.2. Rendimiento

Una decisión arquitectónica clave que mejora el rendimiento del sistema basado en la arquitectura de Krutchen es el desacoplamiento de los sistemas administrativos (vista lógica y vista de desarrollo) de los puntos de acceso y transporte de mensajes (vista de procesos y vista física).

Los sistemas de back office (vista lógica) interactúan con los puntos de acceso mediante interfaces expuestas (servicios web, JMS, REST, etc.), garantizando una mejor organización modular y escalabilidad.

El uso de colas JMS dentro del sistema permite gestionar de manera eficiente el procesamiento de mensajes, asegurando un mejor rendimiento y distribución de carga dentro de la arquitectura.

12. Calidad

La arquitectura basada en las vistas 4+1 de Krutchen contribuye a mejorar la extensibilidad, confiabilidad y portabilidad del sistema.

12.1. Extensibilidad

El sistema está diseñado de manera modular y organizado en capas interconectadas. Este diseño

facilita la actualización y mantenimiento, permitiendo reemplazar módulos existentes o agregar nuevos sin afectar el funcionamiento general.

12.2. Confiabilidad

La confiabilidad del sistema se mejora mediante el desacoplamiento de cada capa arquitectónica a través de colas JMS. Un mecanismo de almacenamiento y reenvío, junto con una política de reintentos automáticos, garantiza que el sistema continúe funcionando sin pérdida de datos ante fallas en componentes específicos.

12.3. Portabilidad

Actualmente, la aplicación puede desplegarse en Google Cloud Run, utilizando Google Cloud Functions y Firebase para la gestión de la infraestructura en la nube. Además, se integra con herramientas de monitoreo como Google Cloud Monitoring y Grafana para garantizar su estabilidad y rendimiento en entornos de producción.

Con cambios menores, la aplicación podría ser desplegada en cualquier plataforma que soporte contenedores Docker y entornos compatibles con servidores de aplicaciones basados en Java Servlet 3.0. También puede conectarse a cualquier Sistema de Gestión de Bases de Datos Relacional (RDBMS) según las necesidades del proyecto.

Además de ser extensible, la arquitectura está diseñada para ser independiente de los sistemas externos con los que se comunica. El uso de una API de complementos genérica garantiza que las distintas capas de la aplicación permanezcan sin modificaciones cuando sea necesario integrar nuevos servicios externos.

El uso de JPA (Java Persistence API) para acceder a la base de datos facilita la migración entre diferentes gestores de bases de datos sin afectar la lógica central de la aplicación.

13. Información de Contacto

Nombres y Apellidos: José Miguel Regalado Valarezo

CI: 1104857071

Fecha de nacimiento: 10/03/2003

Dirección de domicilio: Loja, Loja, Av. Pío Jaramillo Alvarado y Argentina

Teléfono celular: 0960947889

Correo electrónico: mjregalado2@utpl.edu.ec

Nombres y Apellidos: Tais Belen Balcázar Albán CI: 1150083242
Fecha de nacimiento: 21/12/2003
Dirección de domicilio: Loja, Loja, Av. Turunuma y São Paulo
Teléfono celular: 0990823668
Correo electrónico: tbbalcazar@utpl.edu.ec

Nombres y Apellidos: Jeremy Fabricio Jaramillo Peña CI: 1105638025
Fecha de nacimiento: 07/03/2003
Dirección de domicilio: Loja, Loja, Eduardo Mora e Ibarra
Teléfono celular: 0980978748
Correo electrónico: jfjaramillo19@utpl.edu.ec

Nombres y Apellidos: Miguel Alejandro Álvarez Lima CI: 1105652604
Fecha de nacimiento: 28/04/2004
Dirección de domicilio: Loja, Loja,
Sevilla de Oro y Argentina
Teléfono celular: 0979114141
Correo electrónico: malvarez11@utpl.edu.ec

Nombres y Apellidos: Carlos Agustin Salas Churo

CI: 1104524507
Fecha de nacimiento: 29/07/2003
Dirección de domicilio: Loja, Loja, Ciudadela Bellavista entre Republica Dominicana y Chile
Teléfono celular: 0969606481
Correo electrónico: casalas8@utpl.edu.ec