



Proyecto: Parking Pellegrini Suites

Seminario de Práctica de Informática

Módulo 4: Patrones de diseño y JDBC

Profesora: Ana Carolina Ferreyra

Alumno: Gonzalo Sian

Leg.: VINF013776

18 de noviembre de 2024

Índice

1. Proyecto: Parking Pellegrini Suites	4
2. Introducción	4
2.1. D-1: Antecedentes	4
2.2. D-2: Áreas a mejorar	4
2.3. D-3: Desafíos específicos del sistema	5
3. Justificación	5
3.1. D-1: Razones para llevar a cabo el proyecto	6
3.2. D-2: Beneficios esperados	6
4. Definiciones del proyecto	7
4.1. Objetivo general del proyecto	7
4.2. D-1: Objetivos específicos	7
4.3. D-2: Diagrama de Gantt de las actividades a desarrollar	8
5. Definiciones del sistema	8
5.1. Objetivo general del sistema	8
5.2. D-1: Límites, alcances y restricciones del sistema	8
6. Elicitación	9
6.1. D-1: Profundización del proceso	11
6.2. D-2: Identificación de la actividad del cliente y tecnologías a utilizar (TIC)	11
7. Conocimiento del negocio	13
7.1. Diagrama de dominio	14
7.2. D-1: Relevamiento de procesos de negocio	14
7.3. D-2: Diagnóstico de los procesos relevados	16
8. Propuesta de solución	17
8.1. Propuesta funcional	17
8.2. Propuesta técnica	18
8.3. D-1: Diagrama de arquitectura de la propuesta	18
9. Requerimientos	19
9.1. Requerimientos funcionales	19
9.2. Requerimientos no funcionales	21
9.3. D-1: Requerimientos candidatos	22

10. Inicio del análisis: casos de uso	23
10.1. Diagrama de caso de uso	23
10.2. Identificación de actores	23
10.3. Trazabilidad	24
10.4. Descripción de casos de uso	25
10.5. D-1: Descripción de otros casos de uso	26
11. Etapa de análisis	27
12. Etapa de diseño	28
13. Etapa de implementación	30
14. Etapa de pruebas	31
15. Interfaz gráfica	32
16. Definición de base de datos para el sistema	33
17. Diagrama entidad-relación de la base de datos	34
18. Creación de las tablas MySQL	35
19. Inserción, consulta y borrado de registros	37
19.1. Población de tablas	38
19.2. Insertar, modificar, eliminar y consultar registros	38
20. Presentación de las consultas SQL	40
21. Desarrollo del sistema utilizando Java	40
22. Presentación del desarrollo en Java	53
23. Fuentes	55

1. Proyecto: Parking Pellegrini Suites

Desarrollo de un sistema integral de gestión para la playa de estacionamiento y el centro de lavado de Parking Pellegrini Suites, con énfasis en optimizar la organización, mejorar la seguridad y garantizar el cuidado de los vehículos.

2. Introducción

Parking Pellegrini Suites, con una sólida trayectoria en la gestión de playas de estacionamiento, se compromete a ofrecer una experiencia de excelencia a sus clientes. Su estrategia se basa en tres pilares fundamentales: organización eficiente, seguridad mejorada y un cuidado minucioso de los vehículos.

El proyecto tiene como objetivo diseñar y desarrollar un sistema de gestión que registre de manera precisa los ingresos y egresos de los vehículos, optimizando los tiempos en los procesos de check-in y check-out, y generando los comprobantes correspondientes. Además, el sistema gestionará todas las tareas relacionadas con el área de lavado, y controlará los cambios de matafuegos, asegurando el cumplimiento de las normativas de seguridad.

A continuación, se detallan los antecedentes, la descripción de las áreas a mejorar y la formulación específica de los desafíos que enfrentará el sistema.

2.1. D-1: Antecedentes

Parking Pellegrini Suites ha operado durante varios años utilizando sistemas manuales y poco integrados para la gestión de sus servicios de estacionamiento y lavado de vehículos. Los registros de ingreso y egreso de vehículos, así como los comprobantes, se llevaban en hojas de cálculo, lo que generaba ineficiencias y errores. La creciente demanda y el aumento en la cantidad de vehículos gestionados han expuesto la necesidad de un sistema automatizado que optimice estos procesos.

2.2. D-2: Áreas a mejorar

- 1. Registro de ingresos y egresos:** Actualmente, el proceso de check-in y check-out de vehículos es lento, generando filas en horas pico. Se requiere una solución que agilice estos procedimientos.
- 2. Emisión de comprobantes:** La generación manual de comprobantes es propensa a errores, dificultando la trazabilidad de los servicios prestados.

-
3. **Gestión del lavado de vehículos:** No existe un sistema eficiente que permita la programación, registro y control de los servicios de lavado.
 4. **Control de matafuegos:** La gestión manual de cambios y revisiones de matafuegos es inconsistente y no garantiza el cumplimiento de las normas de seguridad.

2.3. D-3: Desafíos específicos del sistema

1. **Integración de funciones:** El sistema debe unificar las áreas de estacionamiento, lavado y control de matafuegos en una única plataforma intuitiva, accesible para el personal y gerentes.
2. **Automatización y precisión:** El nuevo sistema deberá automatizar procesos clave como el registro de vehículos, la emisión de comprobantes y la gestión del lavado, minimizando el margen de error y mejorando la eficiencia.
3. **Seguridad y trazabilidad:** Deberá garantizar que cada vehículo y servicio esté correctamente registrado, permitiendo un control completo sobre el ingreso, egreso y tareas adicionales, como el mantenimiento de matafuegos y el servicio de lavado.
4. **Escalabilidad y adaptabilidad:** El sistema debe ser capaz de adaptarse al crecimiento futuro de la empresa y a la incorporación de nuevas tecnologías o servicios.

3. Justificación

Parking Pellegrini Suites, ubicado en la ciudad de Rosario, Santa Fe, Argentina, entiende que la eficiencia en la gestión de estacionamientos y centros de lavado es clave para ofrecer un servicio de calidad y garantizar la satisfacción del cliente. Actualmente, la falta de un sistema integral que automatice y centralice los procesos ha limitado el potencial de crecimiento y ha afectado la experiencia del cliente, especialmente en momentos de alta demanda.

El desarrollo de un sistema integral de gestión no solo mejorará la organización interna y la seguridad, sino que también permitirá optimizar los tiempos de registro de ingresos y egresos de vehículos, la emisión de comprobantes, y la gestión de los servicios de lavado. Además, el sistema incluirá la administración de tareas esenciales como el control y cambio de matafuegos, lo que incrementará los niveles de seguridad dentro del establecimiento.

Este proyecto no solo beneficiará a la empresa al mejorar la eficiencia operativa, sino que también contribuirá a la seguridad de los vehículos y a una mayor transparencia en los servicios ofrecidos, brindando confianza tanto a clientes como a empleados. La implementación de este sistema también permitirá la escalabilidad y adaptación a futuras necesidades tecnológicas, posicionando a Parking Pellegrini Suites como un líder en su sector dentro de la ciudad de Rosario.

El proyecto permitirá optimizar los procesos de gestión, mejorar la calidad del servicio, y garantizar la seguridad, aspectos clave para satisfacer las expectativas de los clientes y fomentar el crecimiento sostenible de la empresa.

3.1. D-1: Razones para llevar a cabo el proyecto

1. **Demanda creciente:** La ciudad de Rosario, siendo una urbe en expansión, está experimentando un aumento en el número de vehículos. Un sistema integral de gestión permitirá a Parking Pellegrini Suites adaptarse a esta demanda, evitando congestiones y tiempos de espera excesivos.
2. **Competitividad:** La implementación de un sistema moderno y eficiente posicionaría a Parking Pellegrini Suites como una opción líder frente a la competencia. Esto aumentaría la confianza de los clientes y atraería a nuevos usuarios en busca de un servicio rápido y confiable.
3. **Optimización del personal:** La automatización de procesos como la emisión de comprobantes y la gestión de ingresos y egresos reducirá la carga de trabajo manual, permitiendo al personal concentrarse en tareas más estratégicas y de valor añadido.
4. **Sustentabilidad operativa:** Un sistema eficiente reduce el uso innecesario de papel y minimiza el consumo energético, contribuyendo a la adopción de prácticas más sostenibles dentro de la empresa.
5. **Mejor control de seguridad:** La gestión centralizada de las actividades en el estacionamiento, junto con el seguimiento de los cambios de matafuegos, permitirá cumplir con regulaciones de seguridad y aumentar la protección de los bienes.

3.2. D-2: Beneficios esperados

1. **Mejora en la experiencia del cliente:** Con un sistema más ágil y automatizado, los tiempos de espera disminuirán, lo que proporcionará una experiencia

más satisfactoria para los usuarios, mejorando la percepción del servicio.

2. **Eficiencia en la gestión de recursos:** La automatización de procesos permitirá una mejor utilización de los recursos disponibles, tanto humanos como materiales, reduciendo errores operativos y mejorando la calidad del servicio ofrecido.
3. **Mayor rentabilidad:** Al optimizar los procesos y atraer a más clientes gracias a un servicio mejorado, la empresa verá un incremento en su rentabilidad. Además, el control eficiente de las operaciones permitirá reducir costos operativos.
4. **Escalabilidad:** La implementación del sistema facilitará la adaptación a futuras necesidades, como la expansión de servicios adicionales, la integración con nuevas tecnologías, o incluso la apertura de nuevas sucursales.
5. **Mejor toma de decisiones:** Un sistema que centraliza la información y genera reportes automáticos permitirá a los gerentes y propietarios tomar decisiones basadas en datos en tiempo real, lo que contribuirá a una gestión más estratégica y efectiva del negocio.

4. Definiciones del proyecto

4.1. Objetivo general del proyecto

Desarrollar un sistema integral de gestión que permita optimizar la administración de las operaciones de estacionamiento y lavado de vehículos en Parking Pellegrini Suites, mejorando la eficiencia en la organización, seguridad y atención al cliente durante el próximo año.

4.2. D-1: Objetivos específicos

1. **Automatizar los procesos de ingreso y egreso de vehículos:** Implementar un sistema que registre automáticamente los movimientos de entrada y salida, optimizando los tiempos de check-in y check-out y reduciendo errores manuales en la generación de comprobantes.
2. **Optimizar la gestión del servicio de lavado:** Desarrollar una herramienta que permita organizar y monitorear las tareas del centro de lavado, asegurando un uso eficiente de los recursos y tiempos, y garantizando la satisfacción del cliente.

- 3. Gestionar de forma centralizada la seguridad del estacionamiento:** Incorporar un módulo que controle la inspección y cambio de matafuegos, así como la supervisión de cámaras de seguridad y el mantenimiento preventivo de las instalaciones, asegurando el cumplimiento de las normativas.

4.3. D-2: Diagrama de Gantt de las actividades a desarrollar

Fase	Duración	Actividad	2024				2025		
			septiembre	octubre	noviembre	diciembre	enero	febrero	marzo
Planificación	1 mes	Relevamiento de necesidades y especificaciones							
Diseño del sistema	1 mes	Relevamiento de necesidades y especificaciones							
Desarrollo del sistema	2 meses	Programación de módulos (ingreso, lavado, seguridad)							
Pruebas y ajustes	1 mes	Pruebas de integración y corrección de errores							
Implementación	1 mes	Implementación final del sistema							
Capacitación del personal	1 mes	Formación del personal para el uso del sistema							

5. Definiciones del sistema

5.1. Objetivo general del sistema

Centralizar y automatizar la gestión del estacionamiento y centro de lavado en Parking Pellegrini Suites, con el objetivo de mejorar la eficiencia operativa, garantizar la seguridad de los vehículos y facilitar la administración del personal y recursos.

5.2. D-1: Límites, alcances y restricciones del sistema

■ Límites:

- El sistema gestionará únicamente las operaciones internas de Parking Pellegrini Suites, excluyendo operaciones externas o asociadas a sucursales.
- Solo manejará los procesos relacionados con la administración de vehículos, servicios de lavado y control de seguridad (como el cambio de matafuegos), sin incluir otros servicios externos no directamente relacionados.

■ Alcances:

- Permitirá la automatización completa del ingreso y egreso de vehículos, con registro digital y emisión automática de comprobantes.

-
- Integrará el control de las tareas de lavado, programando los turnos y generando alertas para la gestión de recursos.
 - Centralizará la supervisión de la seguridad interna, gestionando los controles de mantenimiento de matafuegos.

■ **Restricciones:**

- El sistema deberá ajustarse al presupuesto y cronograma definidos en la planificación del proyecto.
- El sistema no podrá operar sin conexión a la red local o Internet, por lo que se requerirá una infraestructura tecnológica adecuada.
- Habrá limitaciones en el acceso a datos sensibles, por lo que se deberá implementar un sistema robusto de permisos y seguridad para proteger la información.

6. Elicitación

La elicitation es el proceso mediante el cual se recolecta y documenta toda la información necesaria para modelar los requerimientos de un dominio específico (Loucopoulos, 1995). Para el caso de Parking Pellegrini Suites, es crucial entender los desafíos y necesidades de un estacionamiento y centro de lavado en una ciudad como Rosario, donde la eficiencia operativa y la satisfacción del cliente son primordiales.

Proceso de elicitation del proyecto Parking Pellegrini Suites

Para obtener los requerimientos del sistema, se llevaron a cabo las siguientes actividades de elicitation:

1. **Encuestas:**

Se distribuyeron encuestas a los clientes actuales y potenciales de Parking Pellegrini Suites con el fin de obtener datos cuantitativos sobre sus experiencias, necesidades y expectativas. Estas encuestas proporcionaron una visión general sobre:

- Frecuencia de uso del estacionamiento
- Tiempo promedio de espera para ingreso y egreso.
- Opinión sobre los servicios de lavado.
- Expectativas sobre la seguridad de sus vehículos.

Las encuestas fueron diseñadas utilizando herramientas de recolección de datos como Google Forms y enviadas por correo electrónico y redes sociales, lo que permitió obtener una muestra más amplia de usuarios.

2. Entrevistas:

Se realizaron entrevistas semi-estructuradas con los siguientes actores clave:

- **Personal del estacionamiento y centro de lavado:** Para entender los desafíos operativos, se entrevistó al personal que trabaja en la playa de estacionamiento y realiza las tareas de lavado. Esto permitió identificar cuellos de botella en los procesos y necesidades de automatización.
- **Propietarios y gerentes:** Se entrevistaron a los gerentes y dueños de Parking Pellegrini Suites para entender sus objetivos comerciales, expectativas del sistema y preocupaciones en torno a la seguridad y eficiencia.

Las entrevistas se realizaron en persona y a través de videollamadas, utilizando preguntas abiertas que permitieron explorar en detalle temas como la gestión del tiempo de ingreso y salida, la calidad del servicio de lavado y el cumplimiento de normativas de seguridad.

3. Observación directa:

Se observó el flujo diario de vehículos en el estacionamiento y las operaciones del centro de lavado durante una semana. Esto ayudó a mapear los puntos críticos en los procesos de check-in y check-out, y a identificar problemas recurrentes en la gestión del área de lavado y el control de seguridad (cámaras, matafuegos, etc.).

Conclusiones del proceso de elicitation:

1. **Necesidad de automatización:** Tanto clientes como empleados coincidieron en que los tiempos de espera y la gestión manual de las operaciones generan ineficiencias. Se requiere un sistema que agilice el proceso de check-in/check-out.
2. **Optimización del lavado de vehículos:** La falta de un sistema que registre y programe los turnos de lavado ha sido una queja recurrente, tanto de los clientes como del personal. Se necesita un sistema que permita la gestión digital de estos servicios.
3. **Seguridad y control:** El personal de seguridad y mantenimiento destacó la importancia de automatizar el control de seguridad, como la gestión de matafuegos y la vigilancia por cámaras.

6.1. D-1: Profundización del proceso

El proceso de elicitación se llevó a cabo en varias etapas, utilizando técnicas mixtas para garantizar una comprensión completa de las necesidades del proyecto. El siguiente diagrama de flujo muestra cómo se estructuró la recolección de información:

Diagrama de Competencia:



El gráfico representa un proceso de investigación que combina tres métodos de recolección de datos: encuestas (cuantitativas), entrevistas (cualitativas) y observación. Cada uno de estos métodos aporta información distinta, que luego es analizada y validada para identificar las necesidades claves. Las encuestas ofrecen datos numéricos para un análisis inicial, las entrevistas permiten una mayor profundización, y la observación valida los datos para asegurar su precisión y relevancia. Todo esto converge en la identificación de aspectos esenciales para la toma de decisiones.

6.2. D-2: Identificación de la actividad del cliente y tecnologías a utilizar (TIC)

Actividades del cliente: El cliente principal de Parking Pellegrini Suites busca un servicio eficiente de estacionamiento y lavado de vehículos, donde se valoran los tiempos de espera mínimos, la seguridad de los automóviles y la posibilidad de obtener servicios adicionales de manera rápida.

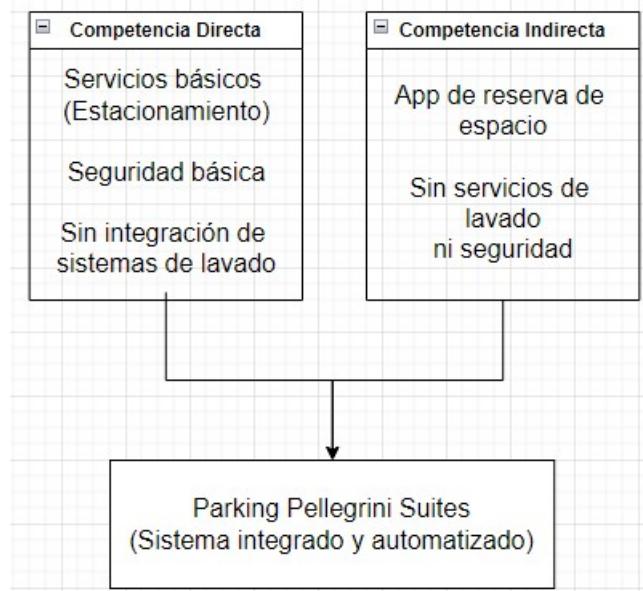
Tecnologías a utilizar (TIC):

- **Software de gestión de estacionamientos:** Un sistema desktop, cuyo servidor es On Premise, permitirá la gestión del ingreso/egreso, la emisión de comprobantes y la programación de servicios de lavado.
- **Tótem para autoservicio de ticket:** Para permitir a los clientes hacer check-in/check-out de manera cómoda, sin bajar del auto, reduciendo el tiempo de espera y la interacción con el personal.

Análisis de la competencia:

1. **Competencia directa:** Los estacionamientos en Rosario que ya ofrecen servicios básicos de gestión y seguridad digitalizados, aunque con menos enfoque en la optimización del lavado de autos.
2. **Competencia indirecta:** Empresas que ofrecen aplicaciones de gestión de estacionamientos y permiten a los usuarios realizar reservaciones de espacios, aunque no cubren servicios adicionales como el lavado o la seguridad con cámaras integradas.

Diagrama de Competencia:



Lo expuesto cubre los pasos y herramientas necesarias para la correcta elicitation de los requerimientos del sistema, junto con un análisis de la actividad del cliente y las tecnologías a implementar.

7. Conocimiento del negocio

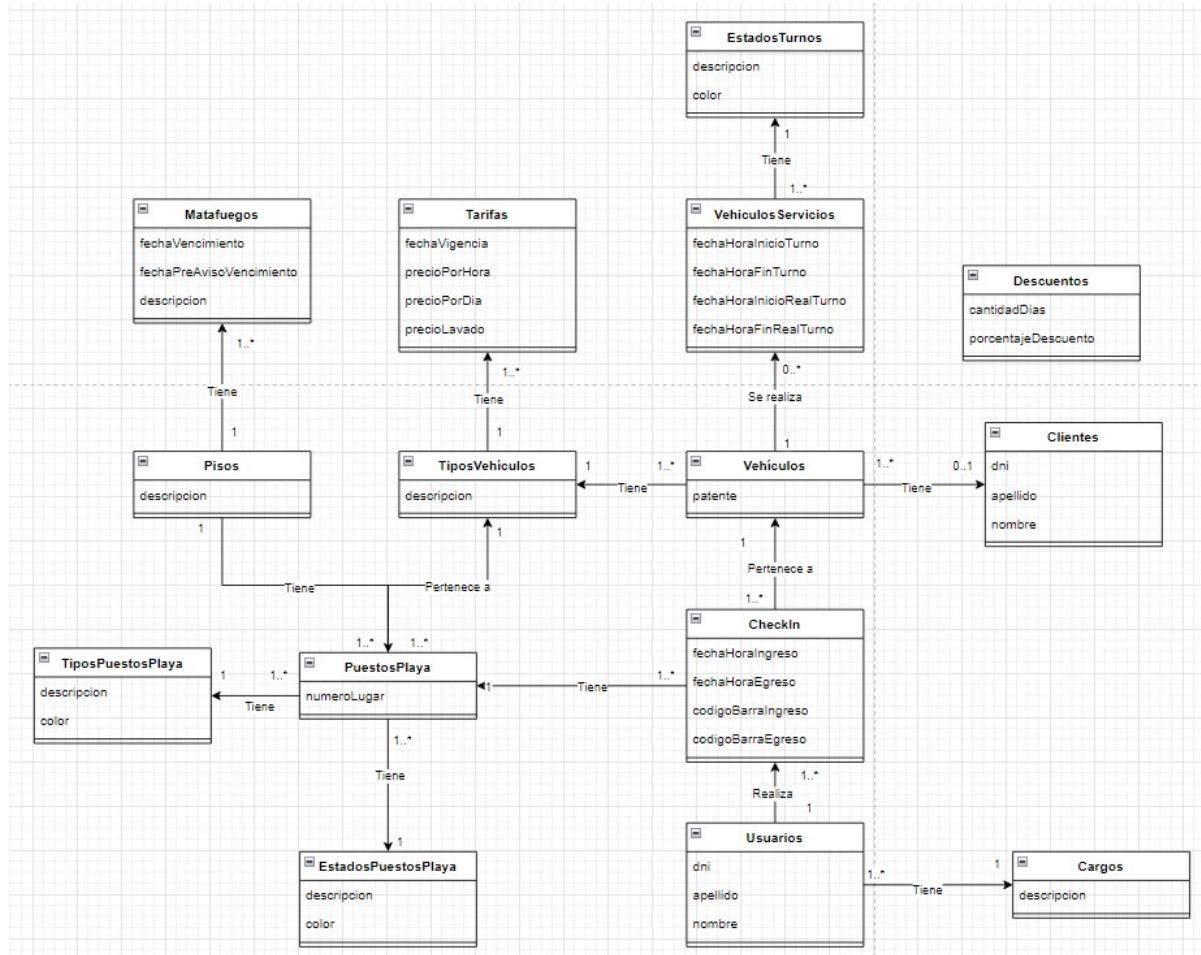
El sistema que se va a desarrollar para Parking Pellegrini Suites debe permitir la gestión integral del estacionamiento y el centro de lavado, optimizando el control de ingreso y egreso de vehículos, el manejo de servicios adicionales (como el lavado y cambios de matafuegos), y garantizando la seguridad de los automóviles. El sistema debe automatizar estas operaciones para mejorar la eficiencia y la satisfacción del cliente.

En un estacionamiento como el de Parking Pellegrini Suites, el flujo constante de vehículos requiere un sistema que gestione de manera efectiva la disponibilidad de espacios, el registro de entradas y salidas, y la emisión de comprobantes. Adicionalmente, el centro de lavado opera como un servicio complementario, lo cual implica que el sistema debe programar y optimizar los turnos para que los clientes puedan solicitar el servicio mientras estacionan sus vehículos.

Para mejorar la experiencia del cliente y la eficiencia operativa, es fundamental automatizar la gestión de los ingresos y egresos, el control de seguridad mediante cámaras y sistemas de vigilancia, y la gestión de servicios de lavado. Además, la administración de los cambios de matafuegos debe ser programada para garantizar el cumplimiento de las normas de seguridad.

7.1. Diagrama de dominio

El siguiente modelo de dominio ilustra los conceptos clave del sistema de gestión de Parking Pellegrini Suites:



7.2. D-1: Relevamiento de procesos de negocio

A continuación, se detallan los principales procesos involucrados en el sistema de Parking Pellegrini Suites: **Proceso: Ingreso de vehículos**

- **Roles:** Cliente, Operador de ingreso, Sistema de gestión.
- **Pasos:**

-
1. El cliente llega al estacionamiento.
 2. El operador verifica la disponibilidad y registra el ingreso del vehículo mediante un sistema automatizado.
 3. El sistema genera un ticket con la información del vehículo (patente, hora de ingreso).
 4. El cliente estaciona el vehículo en el espacio asignado.

Proceso: Egreso de vehículos

- **Roles:** Cliente, Operador de salida, Sistema de gestión.

- **Pasos:**

1. El cliente solicita el retiro de su vehículo.
2. El operador verifica el ticket del cliente.
3. El sistema calcula el costo del servicio según el tiempo transcurrido.
4. El cliente paga, y se emite un comprobante.
5. Se habilita la salida del vehículo.

Proceso: Lavado de vehículos

- **Roles:** Cliente, Operador de lavado, Sistema de gestión.

- **Pasos:**

1. El cliente solicita el servicio de lavado al ingresar.
2. El sistema agenda el turno de lavado, dependiendo de la disponibilidad.
3. El operador del centro de lavado recibe la notificación del turno por sistema.
4. El vehículo es lavado según el servicio solicitado.
5. Se actualiza el estado en el sistema, y el cliente recibe una notificación de WhatsApp.

Proceso: Cambio de matafuegos

- **Roles:** Operador de mantenimiento, Sistema de gestión.

- **Pasos:**

1. El sistema programa el mantenimiento periódico de los matafuegos.
2. El operador de mantenimiento recibe la notificación por sistema.
3. Se realiza el cambio o recarga del matafuego.
4. El sistema actualiza el estado y registra el próximo mantenimiento.

7.3. D-2: Diagnóstico de los procesos relevados

A continuación, se detallan los problemas y causas identificados en cada proceso:

Proceso: Ingreso de vehículos

■ **Problemas:**

- Largas tiempos de espera en horas pico.
- Errores humanos al registrar la patente del vehículo.

■ **Causas:**

- Falta de automatización completa en la verificación y registro de vehículos.
- Escasez de personal para gestionar el alto volumen de autos.

Proceso: Egreso de vehículos

■ **Problemas:**

- Tiempos de espera prolongados para el cálculo de tarifas y pagos.
- Congestión en las salidas durante las horas pico.

■ **Causas:**

- El sistema actual no permite pagos automáticos ni por anticipado.
- Falta de integración con métodos de pago digitales.

Proceso: Lavado de vehículos

■ **Problemas:**

- Dificultad para gestionar la programación de los turnos de lavado.
- Retrasos en la prestación del servicio por falta de organización.

■ **Causas:**

- El sistema manual de asignación de turnos no optimiza la capacidad del centro de lavado.
- No hay una previsión en tiempo real de la demanda de los servicios.

Proceso: Cambio de matafuegos

■ **Problemas:**

-
- El mantenimiento de los matafuegos no siempre se realiza a tiempo.
 - Incumplimiento de normativas de seguridad.

▪ **Causas:**

- El proceso de mantenimiento depende de recordatorios manuales.
- Falta de un sistema de seguimiento de seguridad integrado.

8. Propuesta de solución

8.1. Propuesta funcional

Para mejorar la eficiencia operativa y garantizar una excelente experiencia para los usuarios en Parking Pellegrini Suites, se propone desarrollar un sistema integral de gestión de estacionamiento y centro de lavado. Este sistema automatizará los siguientes procesos clave:

1. Ingreso y egreso de vehículos:

- El sistema registrará automáticamente los ingresos y salidas de vehículos, generando comprobantes digitales y eliminando el proceso manual de registro.
- Permitirá el monitoreo en tiempo real de la ocupación de espacios.

2. Gestión de servicios de lavado:

- Los clientes podrán solicitar el servicio de lavado al realizar el check-in. El sistema programará los turnos en función de la disponibilidad y permitirá la gestión eficiente del personal en el área de lavado.

3. Mantenimiento de matafuegos:

- El sistema incluirá una funcionalidad de seguimiento y programación de cambios y recargas de matafuegos, alertando al personal cuando se acerque la fecha límite.

El sistema estará orientado a mejorar la experiencia del usuario, optimizar los recursos internos y garantizar el cumplimiento de normas de seguridad.

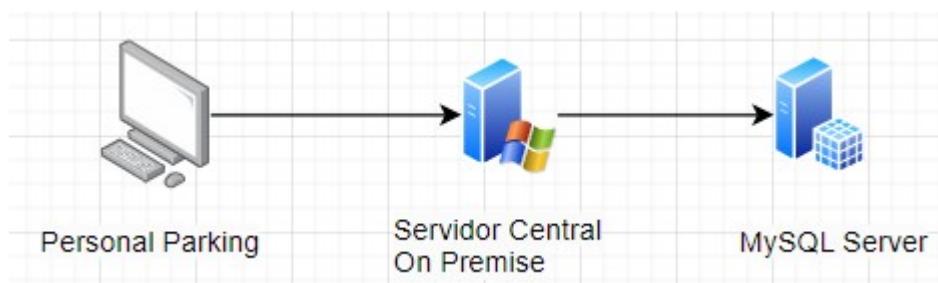
8.2. Propuesta técnica

El desarrollo del sistema de Parking Pellegrini Suites se llevará a cabo con las siguientes tecnologías y herramientas:

- **Lenguaje de programación:** El sistema será desarrollado en Java, lo que permitirá aprovechar el entorno de desarrollo Eclipse y garantizar un rendimiento óptimo y fácil mantenimiento. Esto también permitirá la integración con futuros módulos o funcionalidades.
- **Base de datos:** Se utilizará MySQL Server para la persistencia de los datos, dada su robustez y su capacidad para gestionar grandes volúmenes de información en tiempo real.
- **Interfaz de usuario:** El sistema será una aplicación desktop desarrollada en Java para la administración del estacionamiento y el lavado. Los usuarios podrán interactuar con el sistema a través de una interfaz sencilla.
- **Comunicación y conectividad:**
 - Se empleará una red de conexión Ethernet dentro de las instalaciones para garantizar una transmisión de datos rápida y confiable.
 - Para gestionar el acceso remoto y la administración del sistema desde ubicaciones externas, se implementará un RDP al Servidor central. Esto permitirá el monitoreo y la gestión en tiempo real desde cualquier ubicación.

8.3. D-1: Diagrama de arquitectura de la propuesta

A continuación se describe la arquitectura planteada:



- **Servidor Central On Premise:** Responsable de gestionar la comunicación entre los módulos del sistema y coordinar los flujos de datos.

-
- **SQL Server:** Base de datos centralizada que almacena la información de los vehículos, los servicios y la facturación.
 - **Personal Parking:** Interfaz de usuario para el personal encargado del control de ingreso, egreso y la administración del lavado de vehículos.

9. Requerimientos

9.1. Requerimientos funcionales

RFS01 El sistema debe permitir registrar un Cargo
RFS02 El sistema debe permitir dar de baja un Cargo
RFS03 El sistema debe permitir modificar un Cargo
RFS04 El sistema debe permitir consultar un Cargo
RFS05 El sistema debe permitir listar Cargas
RFS06 El sistema debe permitir registrar un Usuario
RFS07 El sistema debe permitir dar de baja un Usuario
RFS08 El sistema debe permitir modificar un Usuario
RFS09 El sistema debe permitir consultar un Usuario
RFS10 El sistema debe permitir listar Usuarios
RFS11 El sistema debe permitir registrar un Piso
RFS12 El sistema debe permitir dar de baja un Piso
RFS13 El sistema debe permitir modificar un Piso
RFS14 El sistema debe permitir consultar un Piso
RFS15 El sistema debe permitir listar Pisos
RFS16 El sistema debe permitir registrar un Tipo de Vehículo
RFS17 El sistema debe permitir dar de baja un Tipo de Vehículo
RFS18 El sistema debe permitir modificar un Tipo de Vehículo
RFS19 El sistema debe permitir consultar un Tipo de Vehículo
RFS20 El sistema debe permitir listar Tipos de Vehículos
RFS21 El sistema debe permitir registrar un Estado de Puesto Playa
RFS22 El sistema debe permitir dar de baja un Estado de Puesto Playa
RFS23 El sistema debe permitir modificar un Estado de Puesto Playa
RFS24 El sistema debe permitir consultar un Estado de Puesto Playa
RFS25 El sistema debe permitir listar Estados de Puestos Playa
RFS26 El sistema debe permitir registrar un Estado de Turno
RFS27 El sistema debe permitir dar de baja un Estado de Turno
RFS28 El sistema debe permitir modificar un Estado de Turno
RFS29 El sistema debe permitir consultar un Estado de Turno
RFS30 El sistema debe permitir listar Estados de Turnos

-
- RFS31 El sistema debe permitir registrar un Tipo de Puesto Playa
 - RFS32 El sistema debe permitir dar de baja un Tipo de Puesto Playa
 - RFS33 El sistema debe permitir modificar un Tipo de Puesto Playa
 - RFS34 El sistema debe permitir consultar un Tipo de Puesto Playa
 - RFS35 El sistema debe permitir listar Tipos de Puesto Playa
 - RFS36 El sistema debe permitir registrar Tarifa
 - RFS37 El sistema debe permitir dar de baja Tarifa
 - RFS38 El sistema debe permitir modificar Tarifa
 - RFS39 El sistema debe permitir consultar Tarifa
 - RFS40 El sistema debe permitir listar Tarifas
 - RFS41 El sistema debe permitir registrar un Matafuego
 - RFS42 El sistema debe permitir dar de baja un Matafuego
 - RFS43 El sistema debe permitir modificar un Matafuego
 - RFS44 El sistema debe permitir consultar un Matafuego
 - RFS45 El sistema debe permitir listar Matafuegos
 - RFS46 El sistema debe permitir registrar un Descuento
 - RFS47 El sistema debe permitir dar de baja un Descuento
 - RFS48 El sistema debe permitir modificar un Descuento
 - RFS49 El sistema debe permitir consultar un Descuento
 - RFS50 El sistema debe permitir listar Descuentos
 - RFS51 El sistema debe permitir registrar un Cliente
 - RFS52 El sistema debe permitir dar de baja un Cliente
 - RFS53 El sistema debe permitir modificar un Cliente
 - RFS54 El sistema debe permitir consultar un Cliente
 - RFS55 El sistema debe permitir listar Clientes
 - RFS56 El sistema debe permitir registrar un Vehículo
 - RFS57 El sistema debe permitir dar de baja un Vehículo
 - RFS58 El sistema debe permitir modificar un Vehículo
 - RFS59 El sistema debe permitir consultar un Vehículo
 - RFS60 El sistema debe permitir listar un Vehículo
 - RFS61 El sistema debe permitir registrar un Servicio a Vehículo
 - RFS62 El sistema debe permitir dar de baja un Servicio a Vehículo
 - RFS63 El sistema debe permitir modificar un Servicio a Vehículo
 - RFS64 El sistema debe permitir consultar un Servicio a Vehículo
 - RFS65 El sistema debe permitir listar un Servicio a Vehículo
 - RFS66 El sistema debe permitir registrar un Puesto de Playa
 - RFS67 El sistema debe permitir dar de baja un Puesto de Playa
 - RFS68 El sistema debe permitir modificar un Puesto de Playa
 - RFS69 El sistema debe permitir consultar un Puesto de Playa
 - RFS70 El sistema debe permitir listar Puestos de Playa

-
- RFS71 El sistema debe permitir registrar un Check-In
 - RFS72 El sistema debe permitir registrar un Check-Out
 - RFS73 El sistema debe permitir dar de baja un Check-In y/o Check-Out
 - RFS74 El sistema debe permitir dar de baja un Check-In y/o Check-Out
 - RFS75 El sistema debe permitir consultar un Check-In y/o Check-Out
 - RFS76 El sistema debe permitir listar Check-In y/o Check-Out
 - RFS77 El sistema debe permitir re-imprimir un Check-In
 - RFS78 El sistema debe permitir re-imprimir un Check-Out
 - RFS79 El sistema debe permitir el ingreso mediante un usuario y contraseña
 - RFS80 El sistema debe calcular el importe total al egresar un vehículo
 - RFS81 El sistema debe mostrar un mensaje de alerta si un matafuego llegó a la fecha de pre-aviso.
 - RFS82 El sistema debe mostrar un mensaje de alerta si un matafuego llegó a la fecha de vencimiento.
 - RFS83 El sistema debe verificar horarios disponibles para un Servicio a Vehículo.
 - RFS84 El sistema debe tomar como fecha hora de check-in y check-out la hora del sistema.

9.2. Requerimientos no funcionales

- RNFS01 El sistema debe estar desarrollado en Java.
- RNFS02 El sistema debe contar con una base de datos MySQL.
- RNFS03 Se requiere la disponibilidad de impresora láser para garantizar la velocidad de las transacciones.
- RNFS04 Debe garantizarse la seguridad de los datos almacenados.
- RNFS05 La interfaz debe ser intuitiva y fácil de usar.
- RNFS06 El sistema debe tener una disponibilidad de 7 x 24, con un tiempo de inactividad de un 1 % en el mes para mantenimiento.
- RNFS07 El sistema debe ser escalable, es decir, debe permitir la incorporación de nuevas centrales en caso de requerirse.
- RNFS08 Para la persistencia y consulta de datos en la BD se debe utilizar un patrón MVC
- RNFS09 El sistema debe estar instalado en una infraestructura propia en la oficina de la playa de estacionamiento
- RNFS10 Se requiere la disponibilidad de un lector de barras para garantizar la velocidad de las transacciones.

9.3. D-1: Requerimientos candidatos

Los siguientes son aquellos requerimientos funcionales y no funcionales factibles de ser aplicados a futuro:

Requerimientos funcionales candidatos:

- El sistema debe permitir registrar un Perfil de Empresa
- El sistema debe permitir modificar un Perfil de Empresa
- El sistema debe permitir consultar un Perfil de Empresa
- El sistema debe permitir registrar una Cámara Web
- El sistema debe permitir dar de baja una Cámara Web
- El sistema debe permitir modificar una Cámara Web
- El sistema debe permitir consultar una Cámara Web
- El sistema debe permitir listar Cámaras Web

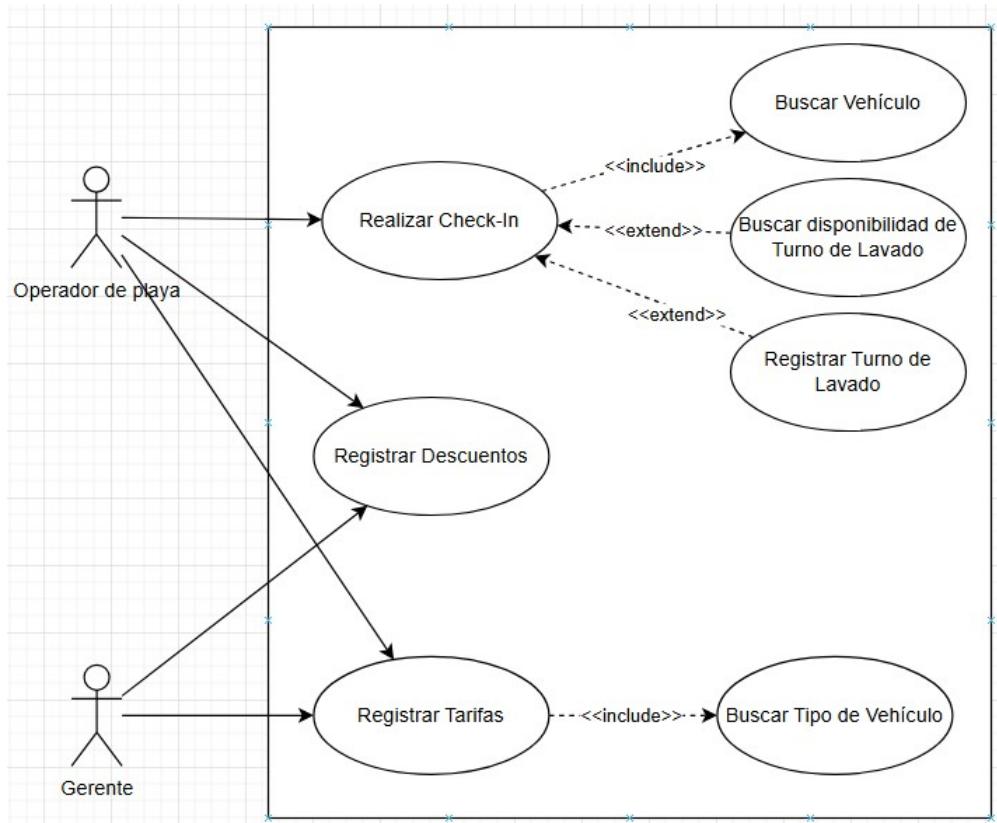
Requerimientos no funcionales candidatos

- El sistema debe dejar de ser OnPremise y migrar a AWS o GCP.
- El sistema debe ser una WebApp.

10. Inicio del análisis: casos de uso

10.1. Diagrama de caso de uso

A continuación se presentan algunos casos de uso.



10.2. Identificación de actores

- **Cliente:** Persona que utiliza los servicios de la empresa. Actualmente, no interactúa directamente con el sistema.
- **Operador de Playa:** Empleado de la empresa autorizado para utilizar el sistema y brindar servicios a los clientes. Posee permisos operativos, pero no de administrador.
- **Gerente:** Empleado con privilegios administrativos, responsable de la gestión y administración del sistema.

10.3. Trazabilidad

Matriz de Trazabilidad			
Requerimiento	Descripción	Caso de Uso	Nombre
RFS71	El sistema debe permitir registrar un Check-In	CU001	Registrar Check-In de Vehículo
RFS84	El sistema debe tomar como fecha hora de check-in y check-out la hora del sistema.		
RFS72	El sistema debe permitir registrar un Check-Out	CU002	Registrar Check-Out de Vehículo
RFS84	El sistema debe tomar como fecha hora de check-in y check-out la hora del sistema.		
RFS80	El sistema debe calcular el importe total al egresar un vehículo	CU003	Mostrar Alerta de Matafuegos por Vencimiento
RFS81	El sistema debe mostrar un mensaje de alerta si un matafuego llegó a la fecha de pre-aviso.		
RFS82	El sistema debe mostrar un mensaje de alerta si un matafuego llegó a la fecha de vencimiento.		
RFS46	El sistema debe permitir registrar un Descuento	CU004	Registrar Descuentos

10.4. Descripción de casos de uso

Caso de Uso	CU001 Registrar Check-In de Vehículo
Actores	Operador de Playa. Cliente
Referencias	RFS56, RFS71, RFS79, RFS83, RFS84
Descripción	Permite al operador de la playa registrar el Check-In de un vehículo, y de corresponder, se lo asocia a un Servicio de Lavado.
Precondición	El operador debe estar autenticado en el sistema.
	El vehículo debe estar registrado en el sistema.
	Debe haber un servicio de lavado disponible en la agenda si el cliente opta por esta opción
Flujo Principal	1 El operador selecciona la opción "Registrar Check-in" en el sistema. 2 El sistema solicita los datos del vehículo (patente, tipo de vehículo) 3 El operador ingresa los datos del vehículo o lo selecciona de la base de datos. 4 Si el cliente requiere un Servicio de Lavado, el sistema verifica la disponibilidad en la agenda. 5 Si hay disponibilidad, el operador selecciona el horario, completa los datos del cliente y confirma el servicio 6 El sistema registra el check-in del vehículo, y si corresponde, el servicio de lavado 7 El sistema genera el ticket con la información del vehículo, el puesto asignado y la hora de ingreso.
	El check-in del vehículo queda registrado correctamente en el sistema.
	Si el cliente ha solicitado el servicio de lavado y hay disponibilidad, el servicio queda programado en la agenda y registrado en el sistema.
Flujo Alternativo	S1 Si no hay disponibilidad en la agenda para el servicio de lavado, el sistema notifica al operador y no permite seleccionar el servicio. S2 Si el cliente no desea el servicio de lavado, el operador continúa con el check-in normal del vehículo.
	E1 Si los datos del cliente no son válidos o están incompletos, el sistema solicita la corrección de los campos. E2 Si el sistema no puede acceder a la base de datos para verificar la disponibilidad de la agenda de lavado, muestra un mensaje de error y solicita intentar más tarde. E3 Si la conexión a la base de datos falla, el sistema notifica al operador y no realiza el registro.

10.5. D-1: Descripción de otros casos de uso

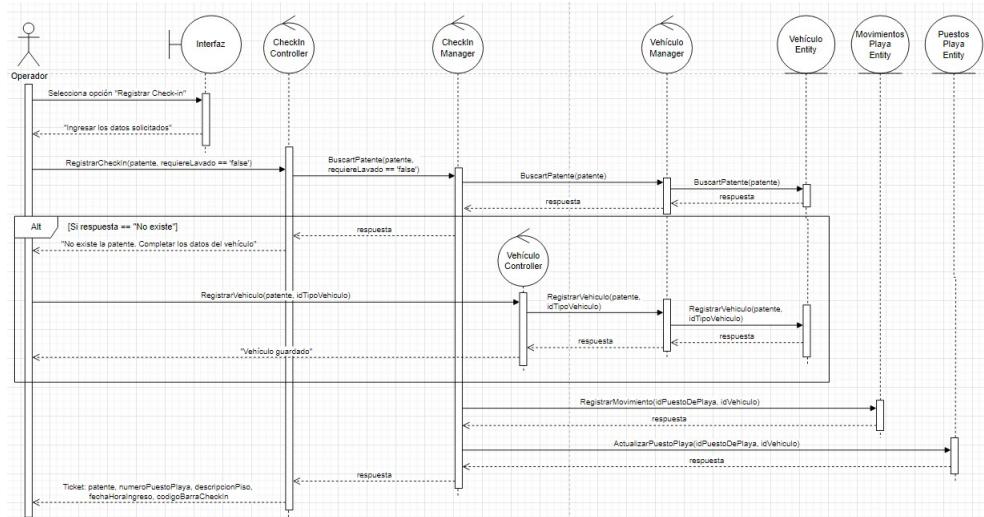
Caso de Uso	CU004 Registrar Descuentos	
Actores	Gerente. Operador de Playa	
Referencias	RFS46, RFS79	
Descripción	Permite al Gerente u Operador de Playa registrar Descuentos	
Precondición	El operador debe estar autenticado en el sistema.	
Flujo Principal	1	El operador selecciona la opción "Registrar Descuentos" en el sistema.
	2	El sistema solicita una cantidad de hora desde, una cantidad de horas hasta y un porcentaje
	3	El sistema verifica que no exista un descuento el rango de horas ingresado
	4	El sistema registra el Descuento.
Postcondición	El descuento quedó registrado correctamente en el sistema.	
Flujo Alternativo	S1	Si existe un descuento en el rango de horas ingreso, el sistema notifica al operador y no permite guardar el descuento..
Excepciones	E1	Si el rango es incorrecto, el sistema muestra un error y solicita el ingreso correcto.
	E2	Si el porcentaje de descuento no se ingresó, el sistema muestra un mensaje de error.
	E3	Si el sistema no puede registrar el descuento debido a un error en la base de datos, muestra un mensaje de error.

Caso de Uso	CU005 Registrar Tarifas	
Actores	Gerente. Operador de Playa	
Referencias	RFS16, RFS36, RFS79	
Descripción	Permite al Gerente u Operador de Playa registrar Tarifas	
Precondición	El operador debe estar autenticado en el sistema. El Tipo de Vehículo debe estar registrado	
Flujo Principal	1	El operador selecciona la opción "Registrar Tarifas" en el sistema.
	2	El sistema solicita el tipo de vehículo, fecha de vigencia, precio por hora, precio por día y precio del servicio de lavado.
	3	El sistema verifica que no existan Tarifas cargadas para el tipo de vehículo y la fecha de vigencia ingresada.
	4	El sistema registra las Tarifas
Postcondición	La Tarifas quedó registrada correctamente en el sistema.	
Flujo Alternativo	S1	Si existe Tarifas cargadas para el tipo de vehículo y la fecha de vigencia ingresada, el sistema notifica al operador y no permite guardar las mismas.
Excepciones	E1	Si la fecha de vigencia es incorrecta, el sistema muestra un error y solicita el ingreso correcto.
	E2	Si el Tipo de Vehículo no se ingresó, el sistema muestra un mensaje de error.
	E3	Si el sistema no puede registrar las Tarifas debido a un error en la base de datos, muestra un mensaje de error.

11. Etapa de análisis

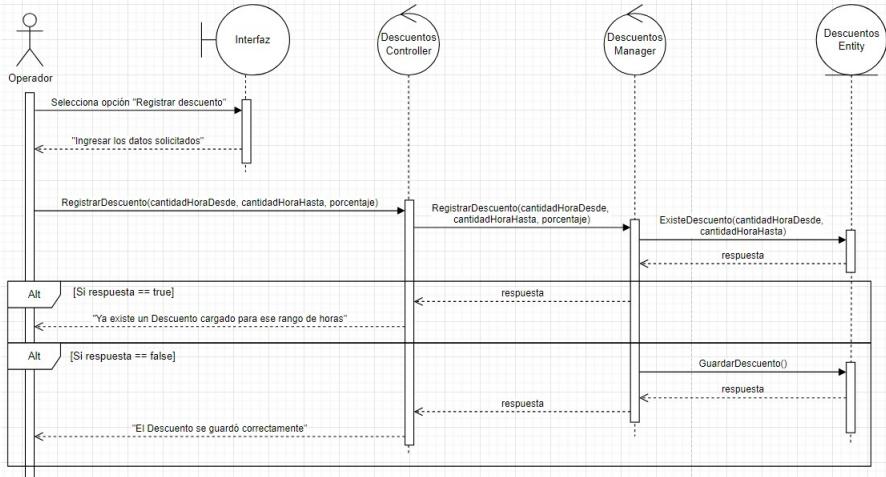
Los diagramas de secuencia en UML (Unified Modeling Language) son una herramienta visual que sirve para modelar la interacción entre los objetos o componentes de un sistema a lo largo del tiempo. Su propósito principal es describir cómo los objetos se comunican entre sí a través de mensajes o eventos en un flujo cronológico.

A continuación, se presenta el diagrama de secuencia para el CU001, "Registrar Check-In de Vehículo".

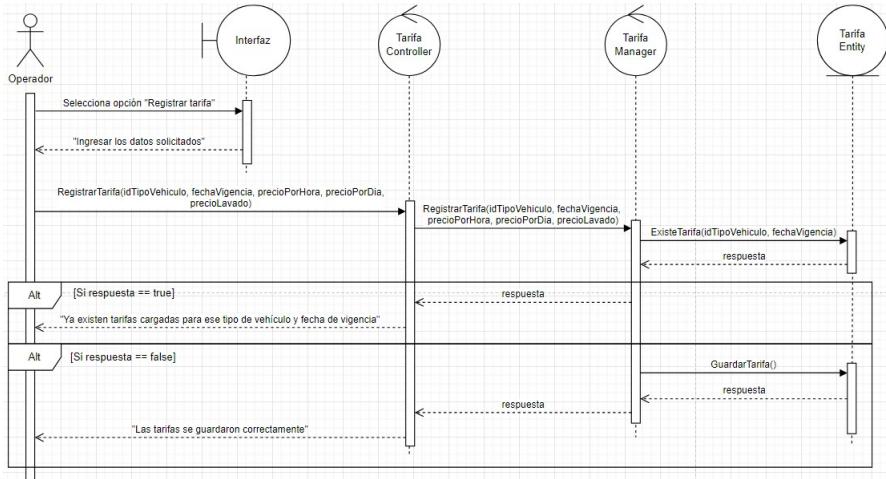


A continuación, se presenta el diagrama de secuencia para el CU004, "Registrar

Descuentos".



Así como también el diagrama de secuencia para el CU005, "Registrar Tarifas".

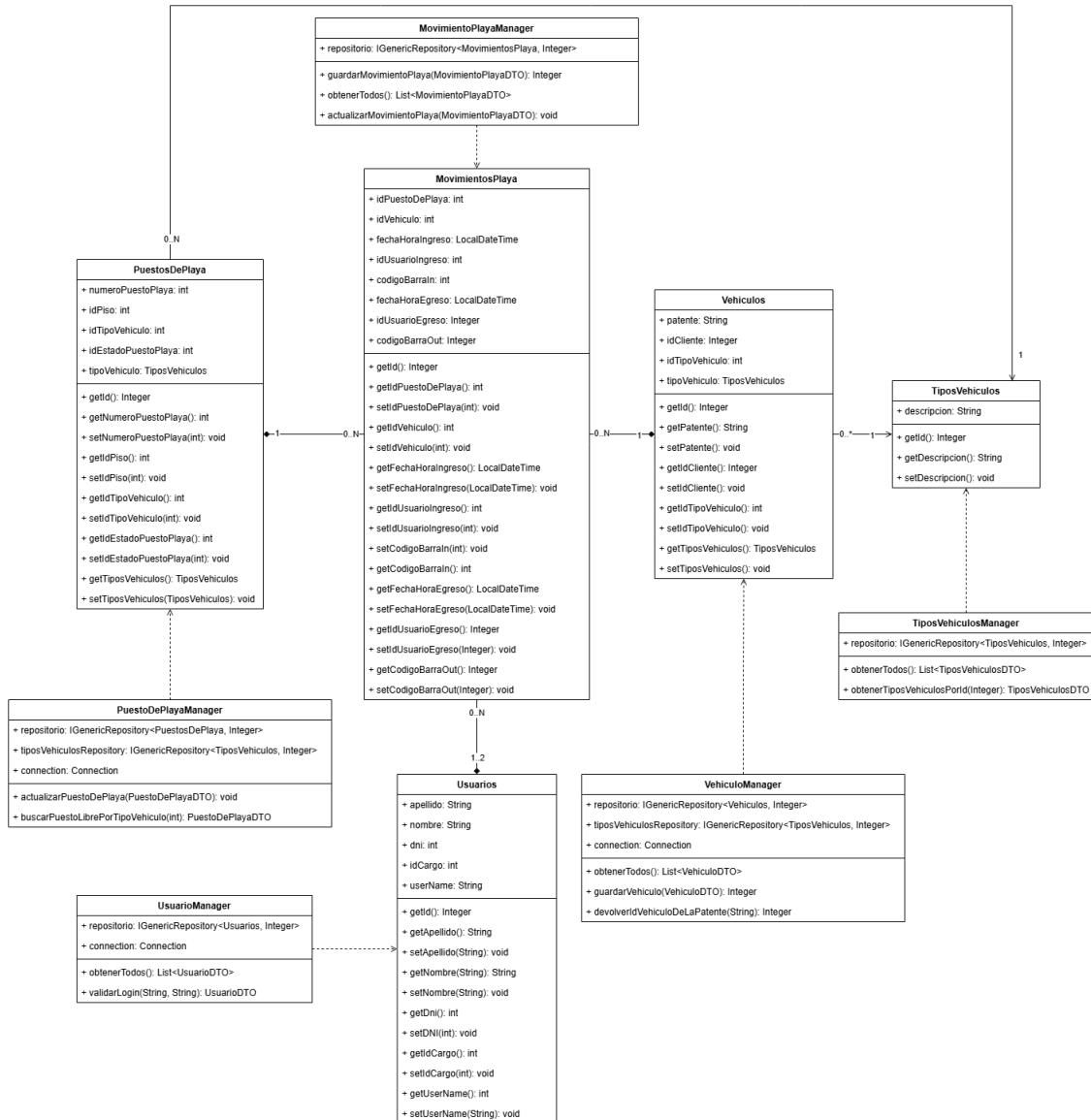


12. Etapa de diseño

Un diagrama de clases del diseño es una representación visual de las clases, sus atributos y relaciones en un sistema de software. Se utiliza para describir la estructura estática y las interacciones entre las clases. Es una herramienta fundamental en la etapa de diseño de software, ya que permite comprender la organización y la jerarquía de las clases, así como los atributos y métodos que contienen. Proporciona una

visión general de cómo se componen las diferentes partes del sistema y promueve la reutilización del código al mostrar las relaciones de herencia y composición entre las clases.

A continuación, se representa el diagrama de clases del subsistema correspondiente al CU001, "Registrar Check-In de Vehículo"



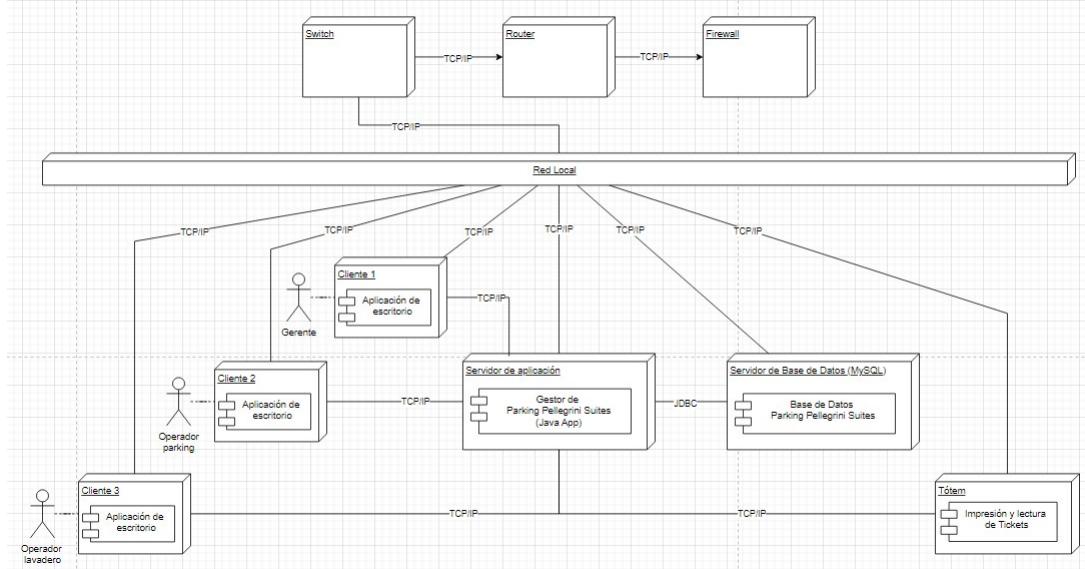
13. Etapa de implementación

Un diagrama de despliegue representa la distribución de los componentes de un sistema en los nodos físicos (servidores, dispositivos, computadoras) y cómo interactúan entre sí en una implementación real. Es de utilidad para entender cómo se va a desplegar el sistema en un hardware físico y cómo se comunican los componentes.

Para esta etapa se van a incorporar a los requisitos funcionales (RF) que se están trabajando, además del siguiente listado de requisitos no funcionales (RNF).

Requerimientos no funcionales	
Requerimiento	Descripción
RNF01	El sistema debe estar desarrollado en Java.
RNF02	El sistema debe contar con una base de datos MySQL.
RNF03	Se requiere la disponibilidad de impresora láser para garantizar la velocidad de las transacciones.
RNF04	Debe garantizarse la seguridad de los datos almacenados.
RNF05	La interfaz debe ser intuitiva y fácil de usar.
RNF06	El sistema debe tener una disponibilidad de 7x24, con un tiempo de inactividad de un 1% en el mes para mantenimiento.
RNF07	El sistema debe ser escalable, es decir, debe permitir la incorporación de nuevas centrales en caso de requerirse.
RNF08	Para la persistencia y consulta de datos en la BD se debe utilizar un patrón MVC.
RNF09	El sistema debe estar instalado en una infraestructura propia en la oficina de la playa de estacionamiento
RNF10	Se requiere la disponibilidad de un lector de barras para garantizar la velocidad de las transacciones.

A partir de esta consideración, a continuación se presenta el diagrama de despliegue correspondiente.



14. Etapa de pruebas

Durante este flujo de trabajo, se planifican las pruebas y se verifica el resultado de la implementación, tanto en las versiones intermedias como la versión final del sistema que se está desarrollando. Los artefactos a considerar en esta etapa son los siguientes.

- Plan de prueba.
- Modelo de pruebas: casos de prueba, procedimiento de prueba y componente de prueba.
- Tratamiento de defectos.
- Evaluación de prueba.

15. Interfaz gráfica

Interfaz de usuario para el caso de uso CU001:

Registrar check-in

Ingrese patente del vehículo:

Ingrese tipo de vehículo:

Cancelar Check-in

Interfaz de usuario para el caso de uso CU004:

Registrar descuentos

Cantidad de horas:

Porcentaje de descuento:

Cancelar Guardar

Interfaz de usuario para el caso de uso CU005:

Registrar tarifas

Tipo de Vehículo:

Precio por hora:

Precio por día:

Precio por lavado:

Cancelar Guardar

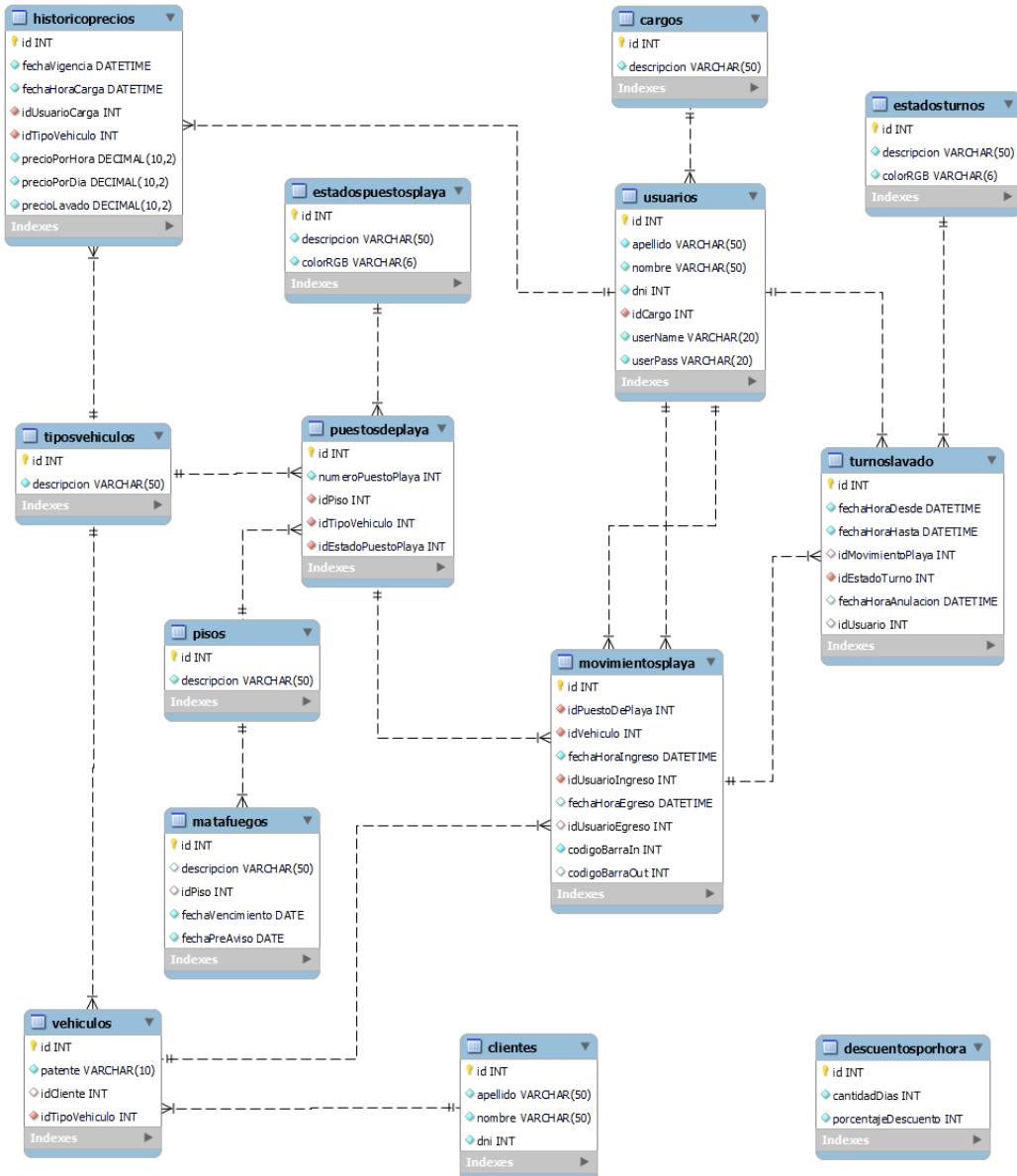
16. Definición de base de datos para el sistema

Para el desarrollo del sistema de gestión y monitoreo, se requiere contar con una base de datos relacional. Tal cual como se especifica en los requerimientos no funcionales, en el prototipo se utiliza una base MySQL, que ofrece un buen rendimiento, flexibilidad y escalabilidad. Al considerar que se está trabajando con un prototipo, estas características son muy necesarias para avanzar de forma ordenada con la implementación del proyecto completo.

El uso de una base de datos es fundamental para lograr la persistencia de datos clave, tales como los movimientos en la playa de estacionamiento, los turnos asignados para el servicio de lavado de auto y la gestión de los matafuegos.

Adicionalmente, el almacenamiento de estos datos permite realizar informes para analizar el mercado, identificar tendencias, detectar problemas y tomar decisiones para mejorar la eficiencia.

17. Diagrama entidad-relación de la base de datos



18. Creación de las tablas MySQL

Se avanzó con el armado de la estructura de datos en la base, realizando la creación de las tablas. Las tablas permiten organizar y almacenar los datos de una manera coherente, ya que cada una representa una entidad dentro del dominio del sistema Parking Pellegrini Suites.

```
-  
10 •    CREATE DATABASE `PARKING`  
11      CHARACTER SET 'utf8mb4'  
12      COLLATE 'utf8mb4_general_ci';  
13  
14 •    USE `PARKING`;  
15  
16 •    CREATE TABLE `Cargos` (  
17      `id` integer AUTO_INCREMENT NOT NULL,  
18      `descripcion` varchar(50) NOT NULL,  
19      PRIMARY KEY (`id`)  
20  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;  
21  
22 •    CREATE TABLE `TiposVehiculos` (  
23      `id` integer AUTO_INCREMENT NOT NULL,  
24      `descripcion` varchar(50) NOT NULL,  
25      PRIMARY KEY (`id`)  
26  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;  
27  
28 •    CREATE TABLE `EstadosTurnos` (  
29      `id` integer AUTO_INCREMENT NOT NULL,  
30      `descripcion` varchar(50) NOT NULL,  
31      `colorRGB` varchar(6) NOT NULL,  
32      PRIMARY KEY (`id`)  
33  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;  
34  
35 •    CREATE TABLE `Pisos` (  
36      `id` integer AUTO_INCREMENT NOT NULL,  
37      `descripcion` varchar(50) NOT NULL,  
38      PRIMARY KEY (`id`)  
39  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;  
40  
41 •    CREATE TABLE `EstadosPuestosPlaya` (  
42      `id` integer AUTO_INCREMENT NOT NULL,  
43      `descripcion` varchar(50) NOT NULL,  
44      `colorRGB` varchar(6) NOT NULL,  
45      PRIMARY KEY (`id`)  
46  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;  
47  
48 •    CREATE TABLE `DescuentosPorHora` (  
49      `id` integer AUTO_INCREMENT NOT NULL,  
50      `cantidadDias` integer NOT NULL,  
51      `porcentajeDescuento` integer NOT NULL,  
52      PRIMARY KEY (`id`)  
53  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```

55 • CREATE TABLE `Matafuegos` (
56     `id` integer AUTO_INCREMENT NOT NULL,
57     `descripcion` varchar(50) NULL,
58     `idPiso` integer NULL,
59     `fechaVencimiento` date NOT NULL,
60     `fechaPreAviso` date NOT NULL,
61     PRIMARY KEY (`id`),
62     KEY `fk_Pisos_id` (`idPiso`),
63     CONSTRAINT `fk_Pisos_id` FOREIGN KEY (`idPiso`) REFERENCES `Pisos` (`id`) ON UPDATE CASCADE
64 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
65
66 • CREATE TABLE `Usuarios` (
67     `id` integer AUTO_INCREMENT NOT NULL,
68     `apellido` varchar(50) NOT NULL,
69     `nombre` varchar(50) NOT NULL,
70     `dni` integer NOT NULL,
71     `idCargo` integer NOT NULL,
72     PRIMARY KEY (`id`),
73     KEY `fk_Cargos_id` (`idCargo`),
74     CONSTRAINT `fk_Cargos_id` FOREIGN KEY (`idCargo`) REFERENCES `Cargos` (`id`) ON UPDATE CASCADE
75 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
76
77 • CREATE TABLE `Clientes` (
78     `id` integer AUTO_INCREMENT NOT NULL,
79     `apellido` varchar(50) NOT NULL,
80     `nombre` varchar(50) NOT NULL,
81     `dni` integer NOT NULL,
82     PRIMARY KEY (`id`)
83 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
84
85 • CREATE TABLE `Vehiculos` (
86     `id` integer AUTO_INCREMENT NOT NULL,
87     `patente` varchar(10) NOT NULL,
88     `idCliente` integer NULL,
89     `idTipoVehiculo` integer NOT NULL,
90     PRIMARY KEY (`id`),
91     KEY `fk_Clientes_id` (`idCliente`),
92     CONSTRAINT `fk_Clientes_id` FOREIGN KEY (`idCliente`) REFERENCES `Clientes` (`id`) ON UPDATE CASCADE,
93     KEY `fk_TiposVehiculos_id` (`idTipoVehiculo`),
94     CONSTRAINT `fk_TiposVehiculos_id` FOREIGN KEY (`idTipoVehiculo`) REFERENCES `TiposVehiculos` (`id`) ON UPDATE CASCADE
95 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
96
97 • CREATE TABLE `HistoricoPrecios` (
98     `id` integer AUTO_INCREMENT NOT NULL,
99     `fechaVigencia` datetime NOT NULL,
100    `fechahoraCarga` datetime NOT NULL,
101    `idUsuarioCarga` integer NOT NULL,
102    `idTipoVehiculo` integer NOT NULL,
103    `precioPorHora` DECIMAL(10, 2) NOT NULL,
104    `precioPorDia` DECIMAL(10, 2) NOT NULL,
105    `precioLavado` DECIMAL(10, 2) NOT NULL,
106    PRIMARY KEY (`id`),
107    KEY `fk_HistoricoPrecios_Usuarios_id` (`idUsuarioCarga`),
108    CONSTRAINT `fk_HistoricoPrecios_Usuarios_id` FOREIGN KEY (`idUsuarioCarga`) REFERENCES `Usuarios` (`id`) ON UPDATE CASCADE,
109    KEY `fk_HistoricoPrecios_TiposVehiculos_id` (`idTipoVehiculo`),
110    CONSTRAINT `fk_HistoricoPrecios_TiposVehiculos_id` FOREIGN KEY (`idTipoVehiculo`) REFERENCES `TiposVehiculos` (`id`) ON UPDATE CASCADE
111 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
112

```

```

113 • CREATE TABLE `PuestosDePlaya` (
114     `id` integer AUTO_INCREMENT NOT NULL,
115     `numeroPuestoPlaya` integer NOT NULL,
116     `idPiso` integer NOT NULL,
117     `idTipoVehiculo` integer NOT NULL,
118     `idEstadoPuestoPlaya` integer NOT NULL,
119     PRIMARY KEY (`id`),
120     KEY `fk_PuestosDePlaya_Pisos_id` (`idPiso`),
121     CONSTRAINT `fk_PuestosDePlaya_Pisos_id` FOREIGN KEY (`idPiso`) REFERENCES `Pisos` (`id`) ON UPDATE CASCADE,
122     KEY `fk_PuestosDePlaya_TiposVehiculos_id` (`idTipoVehiculo`),
123     CONSTRAINT `fk_PuestosDePlaya_TiposVehiculos_id` FOREIGN KEY (`idTipoVehiculo`) REFERENCES `TiposVehiculos` (`id`) ON UPDATE CASCADE,
124     KEY `fk_EstadosPuestosPlaya_id` (`idEstadoPuestoPlaya`),
125     CONSTRAINT `fk_EstadosPuestosPlaya_id` FOREIGN KEY (`idEstadoPuestoPlaya`) REFERENCES `EstadosPuestosPlaya` (`id`) ON UPDATE CASCADE
126 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
127

128 • CREATE TABLE `MovimientosPlaya` (
129     `id` integer AUTO_INCREMENT NOT NULL,
130     `idPuestoDePlaya` integer NOT NULL,
131     `idVehiculo` integer NOT NULL,
132     `fechaHoraIngreso` datetime NOT NULL,
133     `idUsuarioIngreso` integer NOT NULL,
134     `fechaHoraEgreso` datetime NULL,
135     `idUsuarioEgreso` integer NULL,
136     `codigoBarraIn` integer NOT NULL,
137     `codigoBarraOut` integer NULL,
138     PRIMARY KEY (`id`),
139     KEY `fk_PuestosDePlaya_id` (`idPuestoDePlaya`),
140     CONSTRAINT `fk_PuestosDePlaya_id` FOREIGN KEY (`idPuestoDePlaya`) REFERENCES `PuestosDePlaya` (`id`) ON UPDATE CASCADE,
141     KEY `fk_MovimientosPlaya_Vehiculos_id` (`idVehiculo`),
142     CONSTRAINT `fk_MovimientosPlaya_Vehiculos_id` FOREIGN KEY (`idVehiculo`) REFERENCES `Vehiculos` (`id`) ON UPDATE CASCADE,
143     KEY `fk_MovimientosPlaya_Usuarios_Ingreso_id` (`idUsuarioIngreso`),
144     CONSTRAINT `fk_MovimientosPlaya_Usuarios_Ingreso_id` FOREIGN KEY (`idUsuarioIngreso`) REFERENCES `Usuarios` (`id`) ON UPDATE CASCADE,
145     KEY `fk_MovimientosPlaya_Usuarios_Egreso_id` (`idUsuarioEgreso`),
146     CONSTRAINT `fk_MovimientosPlaya_Usuarios_Egreso_id` FOREIGN KEY (`idUsuarioEgreso`) REFERENCES `Usuarios` (`id`) ON UPDATE CASCADE
147 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
148

149 • CREATE TABLE `TurnosLavado` (
150     `id` integer AUTO_INCREMENT NOT NULL,
151     `fechaHoraDesde` datetime NOT NULL,
152     `fechaHoraHasta` datetime NOT NULL,
153     `idMovimientoPlaya` integer NULL,
154     `idEstadoTurno` integer NOT NULL,
155     `fechaHoraAnulacion` datetime NULL,
156     `idUsuario` integer NULL,
157     PRIMARY KEY (`id`),
158     KEY `fk_MovimientosPlaya_id` (`idMovimientoPlaya`),
159     CONSTRAINT `fk_MovimientosPlaya_id` FOREIGN KEY (`idMovimientoPlaya`) REFERENCES `MovimientosPlaya` (`id`) ON UPDATE CASCADE,
160     KEY `fk_EstadosTurnos_id` (`idEstadoTurno`),
161     CONSTRAINT `fk_EstadosTurnos_id` FOREIGN KEY (`idEstadoTurno`) REFERENCES `EstadosTurnos` (`id`) ON UPDATE CASCADE,
162     KEY `fk_Usuarios_id` (`idUsuario`),
163     CONSTRAINT `fk_Usuarios_id` FOREIGN KEY (`idUsuario`) REFERENCES `Usuarios` (`id`) ON UPDATE CASCADE
164 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
165

```

19. Inserción, consulta y borrado de registros

Se van a insertar datos de prueba en algunas tablas para verificar que el modelo de datos y las funcionalidades de la base de datos funcionen correctamente antes de implementar la aplicación en un entorno de producción. Luego, se mostrará con una consulta cómo obtener los resultados y finalmente, se procederá a limpiar las tablas.

19.1. Población de tablas

```
166 •     insert into Cargos (descripcion) values ('Empleado'),('Gerente');
167 •     insert into TiposVehiculos (descripcion) values ('Auto'),('Camiñeta'),('Moto'),('Camión ligero');
168 •     insert into EstadosTurnos (descripcion, colorRGB) values ('Libre','c2f898'),('Ocupado','f8b198'),('Reservado','a49894');
169 •     insert into Pisos (descripcion) values ('Planta baja'),('Piso uno'),('Piso dos'),('Piso tres'),('Piso cuatro');
170
171 •     insert into EstadosPuestosPlaya (descripcion, colorRGB) values ('Libre','84ff92'),('Ocupado','92d2ff'),
172 ('En curso','d692ff'),('Finalizado','ff929c'),('Anulado','89484e');
173
174 •     insert into DescuentosPorHora (cantidadDias, porcentajeDescuento) values (30, 10),(40, 15),(50,20);
175
176 •     insert into Matafuegos (idPiso, fechaVencimiento, fechaPreAviso) values
177 (1, '2024-10-31', '2024-10-21'),(1, '2024-11-24', '2024-11-14'),(2, '2025-10-31', '2025-10-21'),
178 (2, '2025-5-20', '2025-5-10'),(3, '2024-12-31', '2024-12-21'),(3, '2024-12-31', '2024-12-21'),
179 (4, '2025-8-20', '2025-8-10'),(4, '2025-2-20', '2025-2-10');
180
181 •     insert into Usuarios (apellido, nombre, dni, idCargo) values ('Perez', 'Juan', 30222888, 1),
182 ('Gomez', 'Carlos', 30111333, 1),('Sian', 'Gonzalo', 31444999, 2);
183
184 •     insert into Clientes (apellido, nombre, dni) values ('Cabral', 'Daniel', 10222333),
185 ('Diaz', 'Leon', 15666777), ('Frete', 'Gregorio', 20333444);
186
187 •     insert into Vehiculos (patente, idCliente, idTipoVehiculo) values ('AAAA1234', 1, 1),
188 ('BBBB1234', null, 1), ('CCCC1234', null, 2), ('DDDD1234', null, 3);

189
190 •     insert into HistoricoPrecios (fechaVigencia,fechaHoraCarga,idUsuarioCarga,idTipoVehiculo,precioPorHora,
191     precioPorDia,precioLavado) values
192 ('2024-08-05 12:00:00', '2024-07-01 12:00:00', 2, 1, 1000, 12000, 5000),
193 ('2024-09-05 12:00:00', '2024-08-01 12:00:00', 2, 2, 1500, 15000, 7000),
194 ('2024-10-05 12:00:00', '2024-09-01 12:00:00', 2, 3, 500, 6000, 1000);
195
196 •     insert into PuestosDePlaya (numeroPuestoPlaya, idPiso, idTipoVehiculo, idEstadoPuestoPlaya) values
197 (1,1,1,2),(2,1,1,2),(3,1,1,1),(4,1,1,1),(5,1,1,1),(6,1,1,1),(7,1,2,1),(8,1,3,1),(9,1,3,1),(10,1,3,1);
198
199 •     insert into MovimientosPlaya (idPuestoDePlaya,idVehiculo,fechaHoraIngreso,idUsuarioIngreso,
200     fechaHoraAgreso,idUsuarioAgreso,codigoBarraIn,codigoBarraOut) values
201 (1,1,'2024-10-05 11:00:00',2,'2024-10-05 14:00:00',2,1000000,1000001),
202 (2,2,'2024-10-05 13:20:00',null,null,1000002,null),
203 (1,1,current_timestamp(),2,null,null,1000003,null);
204
205 •     insert into TurnosLavado (fechaHoraDesde, fechaHoraHasta, idMovimientoPlaya, idEstadoTurno,
206     fechaHoraAnulacion, idUsuario) values
207 ('2024-10-05 12:00:00', '2024-10-05 13:00:00', 1, 2, null, 2),
208 ('2024-10-05 13:00:00', '2024-10-05 14:00:00', null, 3, null, 2),
209 ('2024-10-05 14:00:00', '2024-10-05 15:00:00', 2, 2, null, 2),
210 ('2024-10-05 15:00:00', '2024-10-05 16:00:00', null, 1, '2024-10-05 12:00:00', 2);
```

19.2. Insertar, modificar, eliminar y consultar registros

Con esta consulta inicial se puede observar como se poblaron algunas de las tablas de la base de datos. Desde este punto iniciaremos algunas pruebas de CRUD.

```
select m.id as idMovimiento, pp.numeroPuestoPlaya, v.patente, m.fechaHoraingreso, m.fechaHoraegreso, t.fechaHoraDesde as fechaHoraDesdeLavado,
      t.fechaHoraHasta as fechaHoraHastaLavado, et.descripcion as estadoTurnoLavado, epp.descripcion as estadoPuesto
from movimientosplaya m
left join vehiculos v on v.id=m.idvehiculo
left join TurnosLavado t on t.idMovimientoPlaya=m.id
left join estadosturnos et on et.id=t.idEstadoTurno
left join PuestosDePlaya pp on pp.id=m.idPuestoDePlaya
left join EstadosPuestosPlaya epp on epp.id=pp.idEstadoPuestoPlaya;
```

idMovimiento	numeroPuestoPlaya	patente	fechahoraingreso	fechahoraegreso	fechaHoraDesdeLavado	fechaHoraHastaLavado	estadoTurnoLavado	estadoPuesto
1	1	AAAA1234	2024-10-05 11:00:00	2024-10-05 14:00:00	2024-10-05 12:00:00	2024-10-05 13:00:00	Finalizado	Ocupado
2	2	BBBB1234	2024-10-05 13:20:00	NULL	2024-10-05 14:00:00	2024-10-05 15:00:00	Finalizado	Ocupado
3	1	AAAA1234	2024-10-06 00:46:40	NULL	NULL	NULL	NULL	Ocupado

Se realiza el Check-In de un vehículo, asignándole un puesto y actualizando el estado del mismo.

```
-- INGRESA UN VEHÍCULO ASIGNÁNDOLE UN PUESTO EN LA PLAYA. NO DESEA TURNO DE LAVADO.
insert into MovimientosPlaya (idPuestoDePlaya,idVehiculo,fechaHoraIngreso,idUsuarioIngresos,
fechaHoraEgreso,idUsuarioEgreso,codigoBarraIn,codigoBarraOut) values
(3,3,current_timestamp(),2,null,null,1000004,null);

-- SE ACTUALIZA EL ESTADO DEL PUESTO DE PLAYA
update PuestosDePlaya set idEstadoPuestoPlaya=2 where id=3;
```

idMovimiento	numeroPuestoPlaya	patente	fechahoraingreso	fechahoraegreso	fechaHoraDesdeLavado	fechaHoraHastaLavado	estadoTurnoLavado	estadoPuesto
1	1	AAAA1234	2024-10-05 11:00:00	2024-10-05 14:00:00	2024-10-05 12:00:00	2024-10-05 13:00:00	Finalizado	Ocupado
2	2	BBBB1234	2024-10-05 13:20:00	NULL	2024-10-05 14:00:00	2024-10-05 15:00:00	Finalizado	Ocupado
3	1	AAAA1234	2024-10-06 00:46:40	NULL	NULL	NULL	NULL	Ocupado
4	3	CCCC1234	2024-10-06 12:20:15	NULL	NULL	NULL	NULL	Ocupado

Se realiza el Check-Out, actualizando el movimiento y liberando el puesto que se había asignado.

```
-- SE HACE EL CHECK-OUT
update MovimientosPlaya set
    fechaHoraEgreso=current_timestamp(),
    idUsuarioEgreso=2,
    codigoBarraOut=1000005
where codigoBarraIn=1000004;

-- SE ACTUALIZA EL PUESTO, LIBERANDO EL MISMO.
update PuestosDePlaya set idEstadoPuestoPlaya=1 where id=3;
```

idMovimiento	numeroPuestoPlaya	patente	fechahoraingreso	fechahoraegreso	fechaHoraDesdeLavado	fechaHoraHastaLavado	estadoTurnoLavado	estadoPuesto	codigoBarraIn
1	1	AAAA1234	2024-10-05 11:00:00	2024-10-05 14:00:00	2024-10-05 12:00:00	2024-10-05 13:00:00	Finalizado	Ocupado	1000000
2	2	BBBB1234	2024-10-05 13:20:00	NULL	2024-10-05 14:00:00	2024-10-05 15:00:00	Finalizado	Ocupado	1000002
3	1	AAAA1234	2024-10-06 00:46:40	NULL	NULL	NULL	NULL	Ocupado	1000003
4	3	CCCC1234	2024-10-06 12:20:15	2024-10-06 12:31:44	NULL	NULL	NULL	Libre	1000004

Por último, se procede a eliminar estos registros de prueba.

```
-- SE ELIMINAN LOS REGISTROS DE PRUEBA
delete from MovimientosPlaya where codigoBarraIn=1000004;
```

20. Presentación de las consultas SQL

El script utilizado se encuentra a disposición mediando el siguiente repositorio de GitHub:

https://github.com/gonzalosian/Parking_Pellegrini_Suites.git

21. Desarrollo del sistema utilizando Java

En esta sección se presentará parte del código utilizado para desarrollar la aplicación de consola. Así mismo, se hará una breve explicación de cada imagen.

Esta clase centraliza los recursos de la aplicación. Usa un singleton para evitar múltiples instancias, lo que asegura que los recursos sean únicos y controlados. También se encarga de manejar y cerrar el Scanner correctamente, manteniendo

todo encapsulado y bien organizado.

```
ApplicationContext.java X
32
33 public class ApplicationContext {
34     private static ApplicationContext applicationContext;
35     private final Scanner scanner;
36     private final DatabaseConnection dbConnection;
37     private final Connection connection;
38     private final Map<Class<?>, Object> beans = new HashMap<>();
39
40     private ApplicationContext() {
41         this.dbConnection = new DatabaseConnection();
42         this.scanner = new Scanner(System.in);
43         this.connection = dbConnection.getConnection();
44         initializeBeans();
45     }
46
47     public static ApplicationContext getInstance() {
48         if (applicationContext == null) {
49             applicationContext = new ApplicationContext();
50         }
51         return applicationContext;
52     }
53
54     /** Obtiene la instancia del Scanner */
55     public Scanner getScanner() {
56         return scanner;
57     }
58
59     /** Cierra el Scanner */
60     public void closeScanner() {
61         scanner.close();
62     }
63 }
```

Este método se encarga de armar todo el "esqueleto" para que las distintas partes de la app trabajen juntas. Primero crea las piezas (repositories, managers, controladores) y las conecta en un mapa para que después puedan ser reutilizadas sin tener que instanciarlas otra vez. Es como un ensamblador detrás de escena que deja todo

listo para cuando se necesite.

```
ApplicationContext.java X
54  private void initializeBeans() {
55      // Instancia de repositorios y managers
56      GenericRepository<DescuentosPorHora, Integer> descuentosRepo = new GenericRepository<>(connection,
57          DescuentosPorHora.class);
58      DescuentosPorHoraManager descuentosManager = new DescuentosPorHoraManager(descuentosRepo, connection);
59      DescuentosPorHoraController descuentosController = new DescuentosPorHoraController(descuentosManager);
60      // Guardar en el map
61      beans.put(DescuentosPorHoraController.class, descuentosController);
62
63      GenericRepository<Usuarios, Integer> usuarioRepo = new GenericRepository<>(connection, Usuarios.class);
64      UsuarioManager usuarioManager = new UsuarioManager(usuarioRepo, connection);
65      UsuarioController usuarioController = new UsuarioController(usuarioManager);
66      beans.put(UsuarioController.class, usuarioController);
67
```

Método getBean: Es el "despachador de instancias". Cuando se necesita un objeto (como un controlador), se pide por medio de su clase y este método lo devuelve ya listo para usar.

Método closeDataBase: Se encarga de apagar la conexión a la base de datos cuando ya no se usa, cuidando que todo quede limpio y sin errores.

```
public <T> T getBean(Class<T> beanClass) {
    return beanClass.cast(beans.get(beanClass));
}

/** Cierra la conexión a la DB al final del ciclo de vida de la aplicación */
public void closeDataBase() {
    try {
        if (connection != null && !connection.isClosed()) {
            connection.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Cuando se crea un objeto DatabaseConnection, se busca el archivo db.properties para cargar las configuraciones de la base de datos. El método getConnection usa esos datos para establecer la conexión. Si algo falla (como no encontrar el archivo o tener credenciales incorrectas), la clase maneja los errores sin que la aplicación crashee.

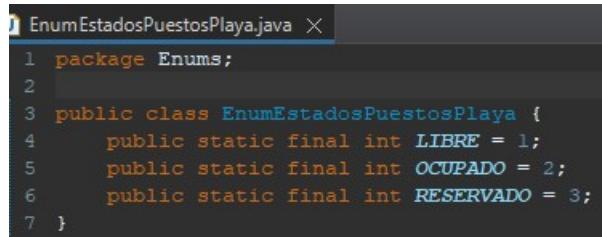
Este diseño centraliza y simplifica el manejo de conexiones.

```
DatabaseConnection.java ×
1 package database;
2
3 import java.sql.Connection;
4
5 public class DatabaseConnection {
6     private static Connection connection = null;
7     private Properties properties = new Properties();
8
9     public DatabaseConnection() {
10         try (InputStream input = getClass().getClassLoader().getResourceAsStream("db.properties")) {
11             if (input == null) {
12                 System.out.println("No se encontró db.properties");
13                 return;
14             }
15             // Cargar propiedades desde el archivo
16             properties.load(input);
17         } catch (IOException ex) {
18             ex.printStackTrace();
19         }
20     }
21
22     public Connection getConnection() {
23         String url = properties.getProperty("db.url");
24         String username = properties.getProperty("db.username");
25         String password = properties.getProperty("db.password");
26
27         try {
28             connection = DriverManager.getConnection(url, username, password);
29             System.out.println("Conectado a la base de datos.");
30         } catch (SQLException e) {
31             e.printStackTrace();
32         }
33
34         return connection;
35     }
36 }
```

El contenido del archivo db.properties contiene las configuraciones necesarias para conectarse a una base de datos MySQL.

```
db.properties ×
1 db.url=jdbc:mysql://localhost:3306/parking
2 db.username=root
3 db.password=123456789
4 db.driver=com.mysql.cj.jdbc.Driver
```

Este código define una clase EnumEstadosPuestosPlaya que simula un enumerado para representar los estados posibles de los puestos en la playa de estacionamiento.

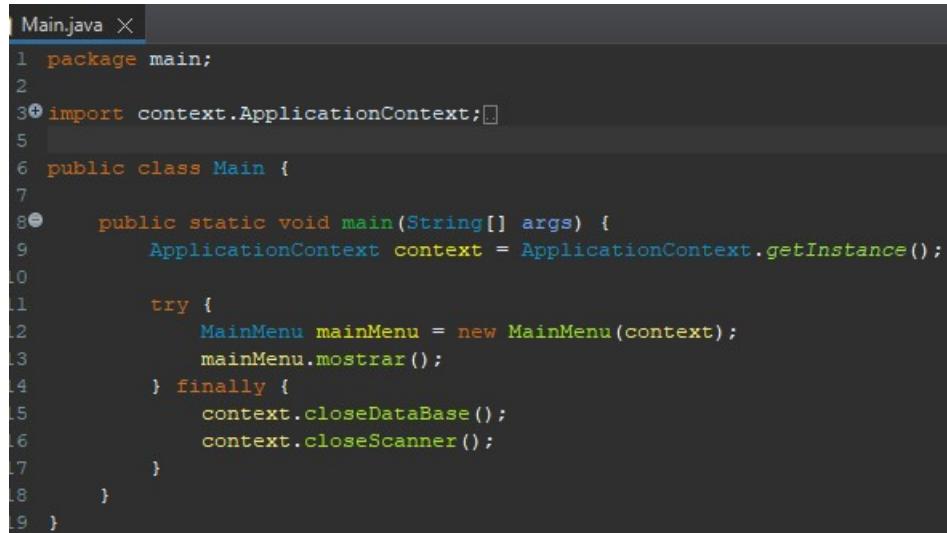


```
1 package Enums;
2
3 public class EnumEstadosPuestosPlaya {
4     public static final int LIBRE = 1;
5     public static final int OCUPADO = 2;
6     public static final int RESERVADO = 3;
7 }
```

Este main contiene el punto de entrada de la aplicación. Acá se puede ver como centralizo la gestión de recursos en ApplicationContext.

Adicionalmente, se trató de garantizar un cierre adecuado de recursos, previniendo errores.

Considero que este desarrollo me permite expandir el menú principal o agregar nuevos menús sin cambiar la estructura básica.



```
1 package main;
2
3 import context.ApplicationContext;
4
5 public class Main {
6
7     public static void main(String[] args) {
8         ApplicationContext context = ApplicationContext.getInstance();
9
10        try {
11            MainMenu mainMenu = new MainMenu(context);
12            mainMenu.mostrar();
13        } finally {
14            context.closeDataBase();
15            context.closeScanner();
16        }
17    }
18 }
19 }
```

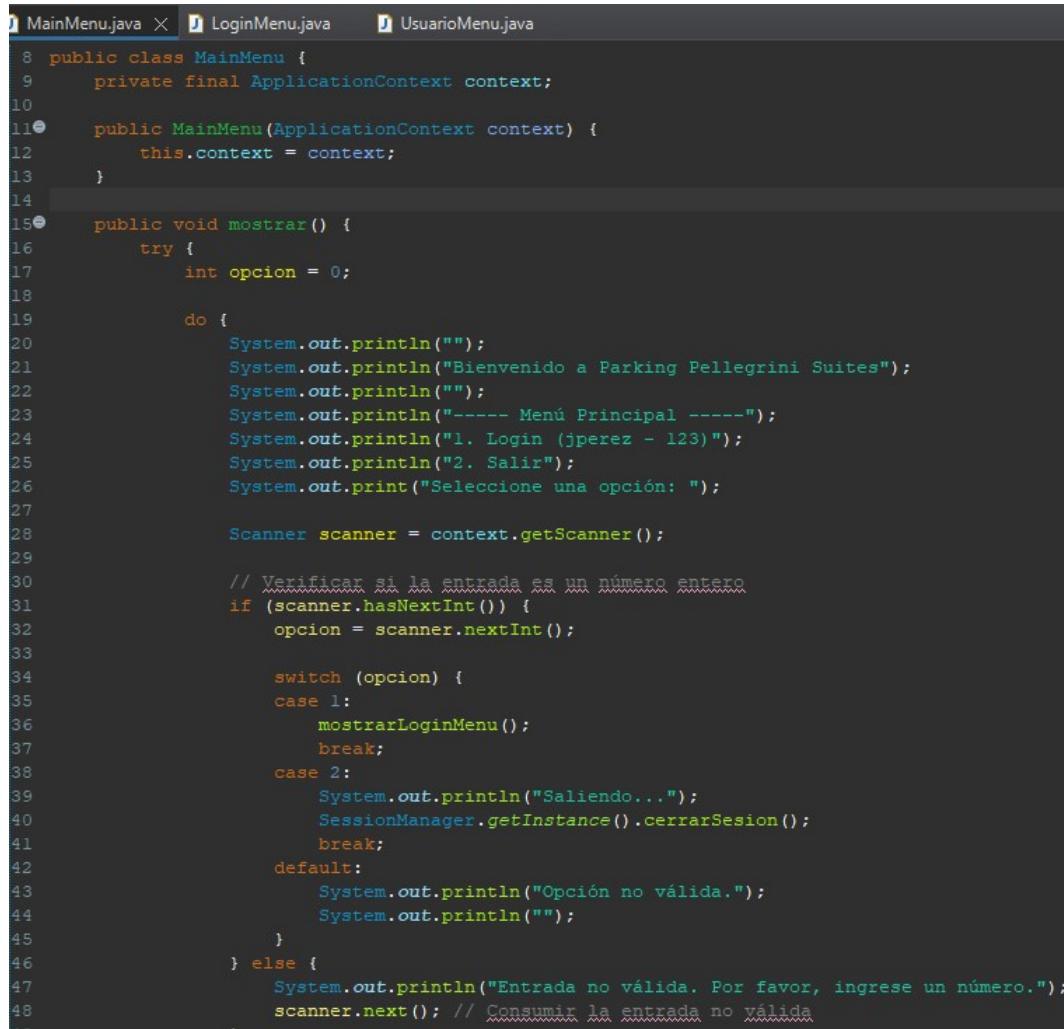
El diseño del SessionManager está orientado al patrón Singleton, lo cual asegura una gestión centralizada y eficiente del estado del usuario logueado. Esto permite al sistema ser flexible y asegurar la consistencia y seguridad de los datos a lo largo del ciclo de vida de la aplicación. Esta implementación es adecuada para el contexto de una aplicación en la que el estado del usuario y la autenticación son factores

críticos.

```
SessionManager.java X
1 package manager;
2
3 import dto.UsuarioDTO;
4
5 public class SessionManager {
6     private static SessionManager instance;
7     private UsuarioDTO usuarioLogueado;
8
9     private SessionManager() {
10 }
11
12     // Para obtener la única instancia de la clase (Singleton)
13     public static SessionManager getInstance() {
14         if (instance == null) {
15             instance = new SessionManager();
16         }
17         return instance;
18     }
19
20     // Para establecer el usuario logueado
21     public void setUsuarioLogueado(UsuarioDTO usuario) {
22         this.usuarioLogueado = usuario;
23     }
24
25     // Para obtener el usuario logueado
26     public UsuarioDTO getUsuarioLogueado() {
27         return usuarioLogueado;
28     }
29
30     // Para cerrar sesión
31     public void cerrarSesion() {
32         this.usuarioLogueado = null;
33     }
34 }
```

En la clase MainMenu se trató de implementar buenas prácticas, como el manejo de excepciones, validación de entradas, y delegación de responsabilidades, lo que hace que el flujo sea claro y manejable. Además, se cumple con principios como encapsulamiento y una correcta separación de responsabilidades entre clases. Esta

estructura mejora la modularidad y la mantenibilidad del sistema.



```
1 MainMenu.java X 2 LoginMenu.java 3 UsuarioMenu.java
8 public class MainMenu {
9     private final ApplicationContext context;
10
11●    public MainMenu(ApplicationContext context) {
12         this.context = context;
13     }
14
15●    public void mostrar() {
16         try {
17             int opcion = 0;
18
19             do {
20                 System.out.println("");
21                 System.out.println("Bienvenido a Parking Pellegrini Suites");
22                 System.out.println("");
23                 System.out.println("----- Menú Principal -----");
24                 System.out.println("1. Login (jperez - 123)");
25                 System.out.println("2. Salir");
26                 System.out.print("Seleccione una opción: ");
27
28                 Scanner scanner = context.getScanner();
29
30                 // Verificar si la entrada es un número entero
31                 if (scanner.hasNextInt()) {
32                     opcion = scanner.nextInt();
33
34                     switch (opcion) {
35                         case 1:
36                             mostrarLoginMenu();
37                             break;
38                         case 2:
39                             System.out.println("Saliendo...");
40                             SessionManager.getInstance().cerrarSesion();
41                             break;
42                         default:
43                             System.out.println("Opción no válida.");
44                             System.out.println("");
45                     }
46                 } else {
47                     System.out.println("Entrada no válida. Por favor, ingrese un número.");
48                     scanner.next(); // Consumir la entrada no válida
49             }
50         } catch (Exception e) {
51             e.printStackTrace();
52         }
53     }
54 }
```

En las clases LoginMenu y UsuarioMenu hago uso de la inyección de dependencias, validación de entrada y control de flujo. Si bien es funcional y modular, considero que la podría mejorar añadiendo el manejo de excepciones para anticipar errores inesperados y agregar algunas optimizaciones. El código permite una fácil expan-

sión y mantenimiento del sistema.



```
mainMenu.java  LoginMenu.java  UsuarioMenu.java

public class LoginMenu {
    private final ApplicationContext context;
    private final UsuarioController usuarioController;

    public LoginMenu(ApplicationContext context) {
        this.context = context;
        this.usuarioController = context.getBean(UsuarioController.class);
    }

    public void mostrar() {
        Scanner scanner = context.getScanner();
        System.out.println("-----");
        System.out.print("Ingrese su usuario: ");
        String user = scanner.next();
        System.out.print("Ingrese su contraseña: ");
        String pass = scanner.next();

        UsuarioDTO usuarioLogueado = authUsuario(user, pass);

        if (usuarioLogueado != null) {
            mostrarUsuarioMenu();
        } else {
            this.mostrarMainMenu();
        }
    }

    public UsuarioDTO authUsuario(String user, String pass) {
        UsuarioDTO usuario = usuarioController.validarLogin(user, pass);

        if (usuario != null) {
            System.out.println("-----");
            System.out.println("Se logueo correctamente");
            System.out.println("-----");
            SessionManager.getInstance().setUsuarioLogueado(usuario);
        } else {
            System.out.println("-----");
            System.out.println("Usuario y/o contraseña incorrecto");
            System.out.println("-----");
        }

        return usuario;
    }
}
```

```
binMenu.java  LoginMenu.java  UsuarioMenu.java X
public class UsuarioMenu {
    private final ApplicationContext context;

    public UsuarioMenu(ApplicationContext context) {
        this.context = context;
    }

    public void mostrar() {
        try {
            int opcion = 0;

            do {
                System.out.println("");
                System.out.printf("Bienvenido/a %s %s \n",
                    SessionManager.getInstance().getUsuarioLogueado().getNombre(),
                    SessionManager.getInstance().getUsuarioLogueado().getApellido());
                System.out.println("");
                System.out.println("----- Menú de Usuario -----");
                System.out.println("1. Check-In");
                System.out.println("2. Descuentos");
                System.out.println("3. Tarifas");
                System.out.println("4. Salir");
                System.out.print(">> Seleccione una opción: ");

                Scanner scanner = context.getScanner();

                if (scanner.hasNextInt()) {
                    opcion = scanner.nextInt();

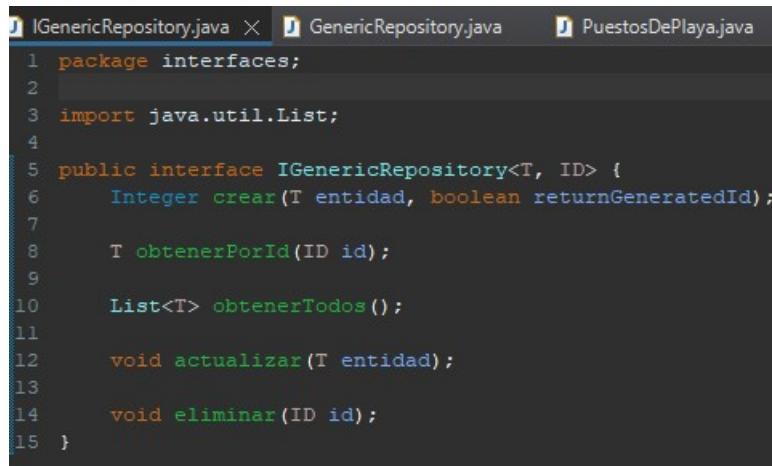
                    switch (opcion) {
                        case 1:
                            mostrarCheckInMenu();
                            break;
                        case 2:
                            mostrarDescuentosMenu();
                            break;
                        case 3:
                            mostrarTarifasMenu();
                            break;
                        case 4:
                            System.out.println("Saliendo...");
                            SessionManager.getInstance().cerrarSesion();
                            break;
                    }
                }
            } while (opcion != 4);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
private void mostrarCheckInMenu() {
    ChekInMenu chekInMenu = new ChekInMenu(context);
    chekInMenu.mostrar();
}

private void mostrarDescuentosMenu() {
    DescuentosMenu descuentosMenu = new DescuentosMenu(context);
    descuentosMenu.mostrar();
}

private void mostrarTarifasMenu() {
    HistoricoPreciosMenu tarifasMenu = new HistoricoPreciosMenu(context);
    tarifasMenu.mostrar();
}
```

La interfaz `IGenericRepository< T, ID >` es una base para un patrón Repository genérico en Java.

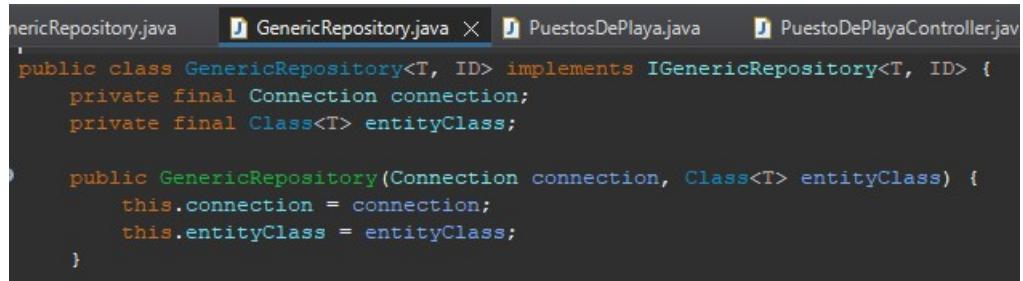


```
IGenericRepository.java × | GenericRepository.java | PuestosDePlaya.java
1 package interfaces;
2
3 import java.util.List;
4
5 public interface IGenericRepository<T, ID> {
6     Integer crear(T entidad, boolean returnGeneratedId);
7
8     T obtenerPorId(ID id);
9
10    List<T> obtenerTodos();
11
12    void actualizar(T entidad);
13
14    void eliminar(ID id);
15 }
```

Este código define la clase `GenericRepository` que implementa la interfaz `IGenericRepository`.

La clase tiene dos cosas importantes: una conexión a la base de datos y una referencia a la clase de la entidad (`entityClass`), que es el tipo de objeto con el que se

estará trabajando.

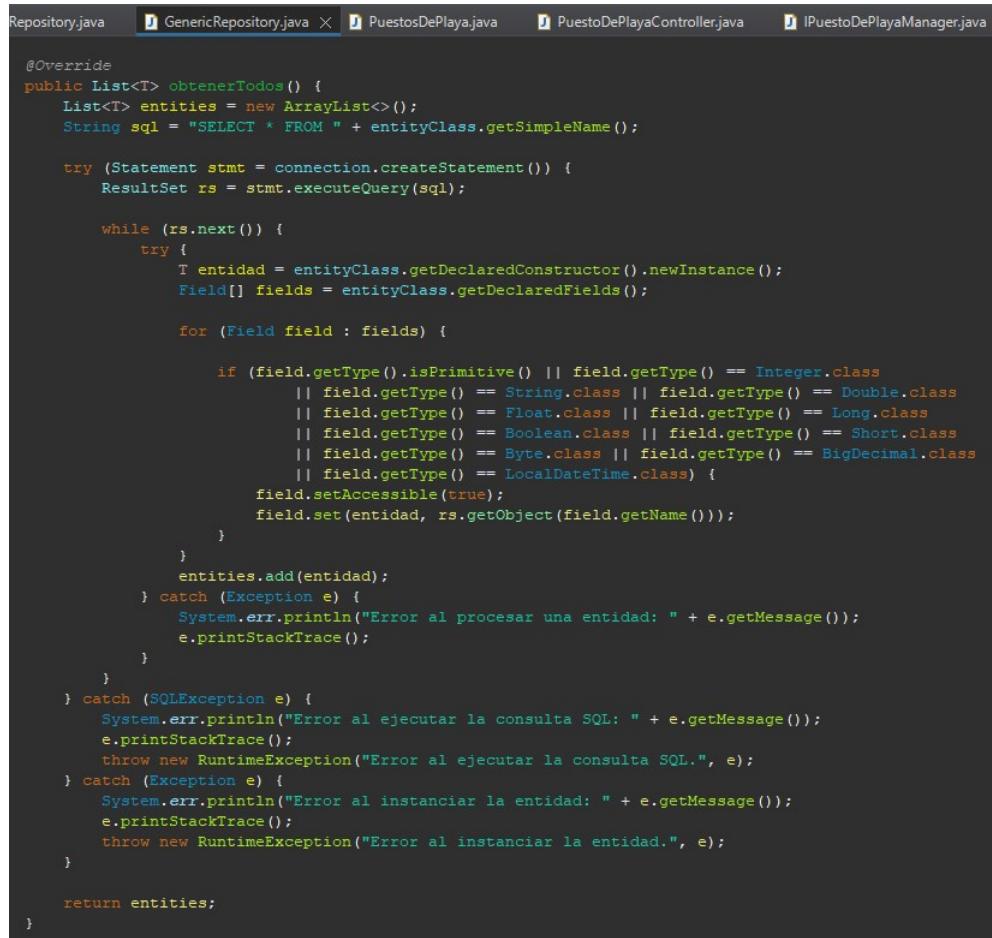


```
GenericRepository.java  GenericRepository.java X  PuestosDePlaya.java  PuestoDePlayaController.java
public class GenericRepository<T, ID> implements IGenericRepository<T, ID> {
    private final Connection connection;
    private final Class<T> entityClass;

    public GenericRepository(Connection connection, Class<T> entityClass) {
        this.connection = connection;
        this.entityClass = entityClass;
    }
}
```

Este método se encarga de ejecutar una consulta SQL para obtener todos los registros de una tabla de la base de datos y convertir esos registros en objetos de la clase correspondiente. Luego, esos objetos se almacenan en una lista y se devuelven. Es una forma genérica de obtener todos los registros de cualquier entidad que

esté configurada en el repositorio.



```
Repository.java  GenericRepository.java  PuestosDePlaya.java  PuestoDePlayaController.java  IPuestoDePlayaManager.java

@Override
public List<T> obtenerTodos() {
    List<T> entities = new ArrayList<>();
    String sql = "SELECT * FROM " + entityClass.getSimpleName();

    try (Statement stmt = connection.createStatement()) {
        ResultSet rs = stmt.executeQuery(sql);

        while (rs.next()) {
            try {
                T entidad = entityClass.getDeclaredConstructor().newInstance();
                Field[] fields = entityClass.getDeclaredFields();

                for (Field field : fields) {

                    if (field.getType().isPrimitive() || field.getType() == Integer.class
                        || field.getType() == String.class || field.getType() == Double.class
                        || field.getType() == Float.class || field.getType() == Long.class
                        || field.getType() == Boolean.class || field.getType() == Short.class
                        || field.getType() == Byte.class || field.getType() == BigDecimal.class
                        || field.getType() == LocalDateTime.class) {
                        field.setAccessible(true);
                        field.set(entidad, rs.getObject(field.getName()));
                    }
                }
                entities.add(entidad);
            } catch (Exception e) {
                System.err.println("Error al procesar una entidad: " + e.getMessage());
                e.printStackTrace();
            }
        }
    } catch (SQLException e) {
        System.err.println("Error al ejecutar la consulta SQL: " + e.getMessage());
        e.printStackTrace();
        throw new RuntimeException("Error al ejecutar la consulta SQL.", e);
    } catch (Exception e) {
        System.err.println("Error al instanciar la entidad: " + e.getMessage());
        e.printStackTrace();
        throw new RuntimeException("Error al instanciar la entidad.", e);
    }

    return entities;
}
```

La clase PuestosDePlaya representa un puesto dentro de un estacionamiento, con información sobre su ubicación (número, piso), el tipo de vehículo que puede estacionarse en él, su estado (ocupado o libre), y un identificador único. Tiene constructores que permiten crear un puesto con los detalles mínimos o completos, y los

métodos getters y setters necesarios.

```
PuestosDePlaya.java × PuestoDePlayaController.java IPuestoDePlayaManager.java PuestoDePlayaManager.java
1 public class PuestosDePlaya {
2     private Integer id;
3     private int numeroPuestoPlaya;
4     private int idPiso;
5     private int idTipoVehiculo;
6     private int idEstadoPuestoPlaya;
7     private TiposVehiculos tiposVehiculos;
8
9     public PuestosDePlaya(Integer id, int numeroPuestoPlaya, int idPiso, int idTipoVehiculo,
10                           int idEstadoPuestoPlaya) {
11         this(id, numeroPuestoPlaya, idPiso, idTipoVehiculo, idEstadoPuestoPlaya, null);
12     }
13
14     public PuestosDePlaya(Integer id, int numeroPuestoPlaya, int idPiso, int idTipoVehiculo, int idEstadoPuestoPlaya, TiposVehiculos tiposVehiculos) {
15         this.id = id;
16         this.numeroPuestoPlaya = numeroPuestoPlaya;
17         this.idPiso = idPiso;
18         this.idTipoVehiculo = idTipoVehiculo;
19         this.idEstadoPuestoPlaya = idEstadoPuestoPlaya;
20         this.tiposVehiculos = tiposVehiculos;
21     }
22
23     public Integer getId() {
24         return id;
25     }
26 }
```

```
PuestoDePlayaController.java × IPuestoDePlayaManager.java PuestoDePlayaManager.java
1 package controller;
2
3 import dto.PuestoDePlayaDTO;
4
5 public class PuestoDePlayaController {
6     private IPuestoDePlayaManager objPuestoDePlayaManager;
7
8     public PuestoDePlayaController(IPuestoDePlayaManager oPuestoDePlayaManager) {
9         this.objPuestoDePlayaManager = oPuestoDePlayaManager;
10    }
11
12    public void actualizarPuestoDePlaya(PuestoDePlayaDTO puestoDePlayaDTO)
13    {
14        objPuestoDePlayaManager.actualizarPuestoDePlaya(puestoDePlayaDTO);
15    }
16
17    public PuestoDePlayaDTO buscarPuestoLibrePorTipoVehiculo(int idTipoVehiculo) {
18        PuestoDePlayaDTO puestoLibre = objPuestoDePlayaManager.buscarPuestoLibrePorTipoVehiculo(idTipoVehiculo);
19        return puestoLibre;
20    }
21 }
```

```
IPuestoDePlayaManager.java × PuestoDePlayaManager.java
1 package interfaces;
2
3 import dto.PuestoDePlayaDTO;
4
5 public interface IPuestoDePlayaManager {
6     void actualizarPuestoDePlaya(PuestoDePlayaDTO puestoDePlayaDTO);
7
8     PuestoDePlayaDTO buscarPuestoLibrePorTipoVehiculo(int idTipoVehiculo);
9 }
```

```

PuestoDePlayaManager.java ×
16 public class PuestoDePlayaManager implements IPuestoDePlayaManager {
17     private IGenericRepository<PuestosDePlaya, Integer> repositorio;
18     private final IGenericRepository<TiposVehiculos, Integer> tiposVehiculosRepository;
19     private final Connection connection;
20
21     public PuestoDePlayaManager(IGenericRepository<PuestosDePlaya, Integer> oRepositorio, Connection connection) {
22         this.repositorio = oRepositorio;
23         this.connection = connection;
24         this.tiposVehiculosRepository = new GenericRepository<>(connection, TiposVehiculos.class);
25     }
26
27     public void actualizarPuestoDePlaya(PuestoDePlayaDTO puestoDePlayaDTO) {
28         PuestosDePlaya puestoDePlaya = new PuestosDePlaya(puestoDePlayaDTO.getId(),
29             puestoDePlayaDTO.getNumeroPuestoPlaya(), puestoDePlayaDTO.getIdPiso(),
30             /* puestoDePlayaDTO.getIdTipoVehiculo(), */ puestoDePlayaDTO.getIdEstadoPuestoPlaya(), null);
31         repositorio.actualizar(puestoDePlaya);
32     }
33 }

```

```

/**
 * Busca el primer puesto libre para un Tipo de Vehículo específico, ordenado
 * por piso y número de puesto.
 *
 * @param idTipoVehiculo
 * @return PuestoDePlayaDTO
 */
public PuestoDePlayaDTO buscarPuestoLibrePorTipoVehiculo(int idTipoVehiculo) {
    String sql = "SELECT pp.id, pp.numeroPuestoPlaya, pp.idPiso, pp.idTipoVehiculo, pp.idEstadoPuestoPlaya "
        + "FROM PuestosDePlaya pp " + "LEFT JOIN EstadosPuestosPlaya ep ON pp.idEstadoPuestoPlaya = ep.id "
        + "WHERE ep.descripcion = ? AND pp.idTipoVehiculo = ? " + "ORDER BY pp.idPiso, pp.numeroPuestoPlaya "
        + "LIMIT 1 ";

    PuestoDePlayaDTO puestoLibre = null;

    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setString(1, "libre"); // Asigna "libre" a la consulta
        stmt.setInt(2, idTipoVehiculo); // Asigna el idTipoVehiculo a la consulta

        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            // Obtiene TiposVehiculos usando el repositorio genérico y el idTipoVehiculo
            TiposVehiculos tiposVehiculos = tiposVehiculosRepository.obtenerPorId(rs.getInt("idTipoVehiculo"));

            puestoLibre = new PuestoDePlayaDTO(rs.getInt("id"), rs.getInt("numeroPuestoPlaya"), rs.getInt("idPiso"),
                rs.getInt("idTipoVehiculo"), rs.getInt("idEstadoPuestoPlaya"),
                new TiposVehiculosDTO(tiposVehiculos.getId(), tiposVehiculos.getDescripcion()));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

return puestoLibre;
}

```

22. Presentación del desarrollo en Java

A continuación se brinda la url con la explicación del proyecto:
<https://www.youtube.com/@gonzalosian>

Disculpe Profesora, estoy teniendo un inconveniente y no puedo subir el video.
Dejo enlace a mi canal y veré si puedo dejarlo allí. Gracias.

23. Fuentes

- Cátedra de SEMINARIO DE PRÁCTICA DE INFORMÁTICA (2024). UES21.
- Cédulas de identificación digital de tu vehículo.
<https://www.argentina.gob.ar/miargentina/servicios/cedulas-de-tu-vehiculo-digital>
- Oliveros, A. y Antonelli, L. (2015). Técnicas de elicitación de requerimientos. XXI Congreso Argentino de Ciencias de la Computación, Junín, Argentina.
- Kendall, K. y Kendall, J. (2011). Análisis y Diseño de Sistemas. Pearson Education.
- Software Overleaf. Editor colaborativo de LaTeX basado en la nube que se utiliza para escribir, editar y publicar documentos científicos.