**UNIVERSIDAD NACIONAL DE INGENIERÍA**

**FACULTAD DE INGENIERÍA INDUSTRIAL Y DE SISTEMAS**



**ÁREA DE SISTEMAS Y TELEMÁTICA**

**Curso:**

**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Proyecto:**

**TRANSPORTE**

**Autores:**

Águila Salvador, Elisson Gabriel (0009-0002-7792-4845/20230046G)

Huanca Méndez, Luís Abel (0009-0000-2318-1340/20231059E)

Pinedo (orcid.org/código)

Samaniego Azañero, David Alexander(0009-0000-6663-7387/20202100K)

Yaures Casanca, Joel Leonardo (0009-0005-7746-4844/20232581G)

**Docente:**

Mag. Coronel Castillo, Eric Gustavo (orcid.org/0000-0003-0494-5629)

**LIMA – PERÚ**

**2024**

# DEDICATORIA

A nuestros queridos padres, con amor y gratitud dedicamos este trabajo. Ustedes son nuestra fuente de inspiración, fortaleza y apoyo incondicional. Gracias por creer en nosotros, por sus sacrificios y por su aliento constante en nuestro camino académico. Su sabiduría y dedicación nos han guiado hacia nuestros sueños. Este logro es un reflejo de su amor y guía, sin los cuales no hubiéramos llegado hasta aquí.

# ÍNDICE GENERAL

Pág.

DEDICATORIA 3

ÍNDICE GENERAL 4

RESUMEN 7

ABSTRACT 8

INTRODUCCIÓN 9

OBJETIVOS 11

Objetivo General 11

Objetivos Específicos 11

REQUERIMIENTOS 12

Requerimientos Funcionales 12

Requerimientos NO Funcionales 12

Alcances 13

Limites 13

ANALISIS DE REQUERIMIENTOS 14

Identificación de Actores 14

Casos de Uso 14

Diagrama de Casos de Uso 14

CU01 – PROGRAMACIÓN DE VIAJES 15

Especificación del caso de uso 15

Prototipo 16

Diagrama de secuencia 16

Modelo de base de dato 17

Criterios de aceptación 17

CU02 – REGISTRO DE MANTENIMIENTO 19

Especificación del caso de uso 19

Prototipo 20

Diagrama de secuencia 20

Modelo de base de dato 21

Criterios de aceptación 21

CU03 – CANTIDAD DE INCIDENTES DE UN CONDUCTOR 24

Especificación del caso de uso 24

Prototipo 25

Diagrama de secuencia 25

Modelo de base de dato 26

Criterios de aceptación 26

CU04 – REGISTRO INCIDENTE 27

Especificación del caso de uso 27

Prototipo 28

Diagrama de secuencia 28

Modelo de base de dato 28

Criterios de aceptación 28

CU05 – REGISTRO DE UNA REPARACIÓN 32

Especificación del caso de uso 32

Prototipo 33

Diagrama de secuencia 33

Modelo de base de dato 34

Criterios de aceptación 34

CU06 – FRECUENCIA DE INCIDENTES 35

Especificación del caso de uso 35

Prototipo 35

Diagrama de secuencia 36

Modelo de base de dato 37

Criterios de aceptación 37

MODELO DE BASE DE DATOS 38

Modelo Conceptual 38

Modelo Relacional 39

Diccionario de Datos 39

IMPLEMENTACIÓN DE LA BASE DE DATOS 47

Creación de la Base de Datos 47

Creación de los objetos de la base de datos 47

Cargar Datos de Prueba 50

CODIGO FUENTE 53

CONCLUSIONES 69

LECCIONES APRENDIDAS 70

BIBLIOGRAFIA 71

ANEXOS 72

Anexo 1: Diagramas del Sistema 73

Anexo 2: Código Fuente del Servicio 74

INDICE DE TABLAS

[Tabla 1. Empleado 39](#_Toc182854984)

[Tabla 2. Incidente 40](#_Toc182854985)

[Tabla 3. Reparacion 40](#_Toc182854986)

[Tabla 4. Taller 41](#_Toc182854987)

[Tabla 5. Mantenimiento 42](#_Toc182854988)

[Tabla 6. Estado del Mantenimiento 43](#_Toc182854989)

[Tabla 7. Programacion 43](#_Toc182854990)

[Tabla 8. Conductor 44](#_Toc182854991)

[Tabla 9. Ruta 45](#_Toc182854992)

[Tabla 10. Carro 45](#_Toc182854993)

[Tabla 11. Estado 46](#_Toc182854994)

[Tabla 12. Tipo de incidente 46](#_Toc182854995)

# RESUMEN

El proyecto se centra en la creación de un sistema integral de gestión que será usado para el desarrollo de una aplicación web, sobre una empresa de transporte, usando herramientas de trabajo como Spring para hacer la lógica del negocio, SQL Server para la base de datos y Postman para hacer las pruebas. El modelo de base de datos fue diseñado con un enfoque relacional, asegurando integridad entre los datos y soporte para futuras ampliaciones del proyecto, esto podrá ser observado con mayor detalle en el diccionario de datos del proyecto. Por otro lado, a lo largo del desarrollo, se hizo hincapié en las buenas prácticas de programación, como la implementación de transacciones seguras en la base de datos y la validación rigurosa de datos para asegurar la integridad de la información.

Asimismo, el sistema realizado para este proyecto permite gestionar de manera correcta la programación de viajes asignando un conductor a un vehículo y a una ruta , también permite registrar mantenimientos tomando en cuenta factores como el taller y el carro, otros registros que estarán presentes en el proyecto serán el registro de incidentes y reparaciones, además también es posible saber el tipo de incidente más frecuente gracias a la generación de reportes sobre los incidentes más frecuentes en un determinado rango de fecha. Todo esto podrá ser observado con más detalle en los casos de uso. Por último, se dará algunas conclusiones y lecciones aprendidas que se adquirió a lo largo del desarrollo del proyecto.

**Palabras clave:** Servicios, aplicación, negocio, información y proyecto.

# ABSTRACT

The project focuses on the creation of a comprehensive management system that will be used for the development of a web application for a transportation company, using work tools such as Spring to make the business logic, SQL Server for the database. and Postman to do the testing. The database model was designed with a relational approach, ensuring integrity between the data and support for future expansions of the project, this can be observed in greater detail in the project's data dictionary. On the other hand, throughout development, emphasis was placed on good programming practices, such as the implementation of secure transactions in the database and rigorous data validation to ensure the integrity of the information.

Likewise, the system created for this project allows you to correctly manage the travel schedule by assigning a driver to a vehicle and a route. It also allows you to record maintenance taking into account factors such as the workshop and the car, other records that will be present in the project. They will be the record of incidents and repairs, in addition it is also possible to know the most frequent type of incident thanks to the generation of reports on the most frequent incidents in a certain date range. All this can be observed in more detail in the use of cases. Finally, some conclusions and lessons learned that were acquired throughout the development of the project will be given.

**key words:** Service, application, business, information and project.

# INTRODUCCIÓN

El sector del transporte es un pilar fundamental para la economía global, ya que conecta puntos de origen y destino, permitiendo la movilización de bienes, servicios y personas. En un mundo cada vez más digitalizado, gestionar eficazmente una empresa de transporte implica enfrentar retos complejos relacionados con la organización de rutas, la programación de conductores, el mantenimiento de vehículos, y el control de recursos humanos. Ante estos desafíos, surge la necesidad de una solución tecnológica integral que permita optimizar las operaciones, minimizar los tiempos improductivos y garantizar la seguridad y calidad en el servicio.

Este proyecto se centra en el desarrollo de una aplicación web utilizando el framework Spring, que servirá como una herramienta de gestión integral para una empresa de transporte. La aplicación está diseñada para interactuar con una base de datos robusta, estructurada de manera eficiente, con el fin de administrar múltiples aspectos del negocio, como la asignación de conductores, el estado y mantenimiento de vehículos, la programación de rutas y la gestión de incidentes. De esta forma, se busca ofrecer un sistema intuitivo y funcional que responda a las necesidades del sector y permita una toma de decisiones basada en datos en tiempo real.

En este trabajo, se aplica una estructura relacional en la base de datos para garantizar la integridad de los datos y facilitar futuras expansiones. Por ejemplo, la base de datos incluye tablas clave para gestionar a los conductores y su disponibilidad, los estados de los vehículos, los periodos de mantenimiento, así asegurando que las asignaciones de recursos se realicen de forma óptima y conforme a las normativas legales aplicables.

El proyecto también aborda las necesidades específicas de planificación y monitoreo. Por ejemplo, la aplicación permitirá verificar la disponibilidad de conductores en rangos de fechas específicos, identificar períodos en los que los vehículos necesitan mantenimiento y registrar incidentes o reparaciones realizadas. Estas funciones se integran en una interfaz web diseñada para facilitar su uso tanto por administradores como por otros usuarios clave dentro de la empresa.

El objetivo general del proyecto es desarrollar una aplicación web para gestionar y optimizar la administración de una flota de vehículos de una empresa de transporte, a través del análisis de los requerimientos solicitados y una consiguiente aplicación de servicios para satisfacer los mismos.

En resumen, este proyecto busca demostrar cómo una combinación de tecnología web moderna, buenas prácticas de diseño de bases de datos y un enfoque centrado en la solución de problemas empresariales puede transformar la forma en que se gestionan las operaciones de una empresa de transporte, ofreciendo una herramienta que no solo mejora la eficiencia, sino que también fomenta la transparencia y la toma de decisiones estratégicas basadas en datos.

# OBJETIVOS

## Objetivo General

Desarrollar una aplicación web para gestionar y optimizar la administración de una flota de vehículos de una empresa de transporte.

## Objetivos Específicos

 Analizar los requisitos funcionales y no funcionales necesarios para el sistema de gestión de una empresa de transporte.

 Crear una base de datos apropiada con la que se pueda manejar la información de los vehículos, conductores, programaciones, empleados, y demás factores intervinientes, a través del sistema

 Implementar clases de servicios que permitan registrar, actualizar y gestionar información detallada sobre cada vehículo, incidentes, reparaciones, mantenimientos, talleres de mantenimiento o reparaciones, incidentes, fechas y rutas programadas, conductores y empleados.

 Realizar pruebas unitarias y de integración de cada servicio para garantizar el correcto funcionamiento del sistema.

# REQUERIMIENTOS

## Requerimientos Funcionales

 **Gestión de vehículos:**

- El sistema debe permitir registrar un nuevo vehículo con su número de placa, su estado actual (disponible, en mantenimiento, etc.) y la fecha de su próximo mantenimiento.

- El sistema debe permitir al administrador editar la información de un vehículo registrado.

- El sistema debe permitir eliminar un vehículo de la flota.

 **Programación de conductores, vehículos y rutas:**

- El sistema debe permitir asignar un conductor a un vehículo y a una ruta específica.

 **Registro de mantenimientos, incidentes y reparaciones:**

- El sistema debe permitir registrar los mantenimientos realizados en los vehículos, indicando tipo de servicio, fecha, taller y costo del servicio.

- El sistema debe permitir consultar el historial de mantenimientos de un vehículo.

- El sistema debe permitir registrar incidentes de tránsito en los que se haya visto involucrado un vehículo de la flota.

- El sistema debe permitir asociar un incidente a un vehículo y un conductor (programación).

- El sistema debe permitir consultar el tipo de incidente más frecuente y el conductor con más incidentes registrados.

 **Gestión de talleres:**

- El sistema debe permitir registrar y gestionar información de talleres de reparación, como nombre, dirección, tipo de servicio y contacto.

- El sistema debe permitir asociar reparaciones y mantenimientos realizadas a un taller específico.

## Requerimientos NO Funcionales

- La interfaz debe ser intuitiva y fácil de usar para los administradores y empleados, sin necesidad de formación técnica avanzada.

- El empleado debe poder completar el proceso de registro en pocos minutos.

- El sistema debe ser accesible desde navegadores modernos.

- El sistema debe ser fácil de mantener y actualizar.

- El sistema debe estar disponible en todo momento de la jornada laboral.

## Alcances

**Incluido en el Alcance:**

- Desarrollo de una aplicación web para gestionar flota de vehículos, asignación de rutas, conductores y vehículos.

- Funcionalidades de mantenimiento, reparaciones, incidentes, y talleres.

- Reportes acerca de la cantidad y tipos de incidentes más frecuentes para la toma de decisiones.

- Interfaz de usuario responsiva y accesible desde navegadores modernos.

**Excluido del Alcance:**

- Desarrollo de aplicaciones móviles nativas.

- Integración con redes sociales o plataformas externas.

- Funcionalidades de colaboración en equipo.

- Integraciones externas con GPS o APIs de tráfico.

- Soporte multilingüe en la fase inicial.

- Optimización para grandes volúmenes de usuarios o datos (en fases posteriores).

- Inteligencia artificial o análisis predictivo (en fases posteriores).

## Limites

- El proyecto se debe completar hasta la semana 15 del presente ciclo académico.

- El proyecto es de financiamiento propio y a pequeña escala.

- El proyecto es solo aplicable a empresas pequeñas.

- No es integrable a otro tipo de plataformas y funciona de manera local.

# ANALISIS DE REQUERIMIENTOS

## Identificación de Actores

|  |  |  |
| --- | --- | --- |
| **CÓDIGO** | **NOMBRE** | **DESCRIPCIÓN** |
| **ACT01** | Empleados | Personas que registran las actividades y registro del sistema. |
| **ACT02** | Administradores | Personas que buscan información para una mejor toma de decisiones. |
| **ACT03** | Conductores | Personas que se encargan de transportar la mercadería con un vehículo. |

## Casos de Uso

|  |  |  |  |
| --- | --- | --- | --- |
| **CÓDIGO** | **ACTOR** | **NOMBRE** | **DESCRIPCIÓN** |
| **CU01** | ACT01 | Programación de rutas, vehículos y conductores. | Se asignarán conductores, vehículos y rutas para un determinado viaje, con una fecha de inicio y fin esperadas, así como la fecha de fin real, y el empleado que registró dicha programación. |
| **CU02** |  |  |  |
| **CU03** | ACT03 | Cantidad de incidentes de un conductor en un rango de fecha | Se describirá la cantidad de incidentes que a tenido el conductor consultado durante un rango específico de fecha. |
| **CU04** |  |  |  |
| **CU05** |  |  |  |

## Diagrama de Casos de Uso

# CU01 – PROGRAMACIÓN DE VIAJES

## Especificación del caso de uso

|  |  |  |
| --- | --- | --- |
| **Código** | **CU01** | |
| **Nombre** | **Programación de rutas, vehículos y conductores** | |
| **Descripción** | Este caso de uso permite a los empleados asignar rutas a vehículos y conductores de manera eficiente, indicando las fechas en las que se inició, se planificó que finalizara y cuando realmente terminó. | |
| **Actores** | Empleado | |
| **Requerimiento asociado** | El sistema debe permitir asignar un conductor a un vehículo y a una ruta específica. | |
| **Casos de uso asociados** | CU03 – Registro de incidentes  CU04 – Cantidad de incidentes por conductor | |
| **Precondición** | El empleado debe estar autenticado en el sistema.  El carro, la ruta, el conductor y el empleado deben existir en la base de datos. | |
| **Secuencia normal** | **Paso** | **Acción** |
| **1** | El empleado inicia el registro de una programación |
| **2** | El sistema valida la existencia del empleado, así como la ruta, carro y conductor a programar |
| **3** | El sistema valida que el carro y el conductor estén disponibles en las fechas a programar |
| **4** | El sistema valida que la fecha de inicio sea posterior a la fecha final real del último viaje realizado con dicho carro |
| **5** | El sistema valida que la fecha final programada sea después de la fecha de inicio |
| **Postcondición** | La programación queda registrada | |
| **Excepciones** | **Paso** | **Acción** |
| **1** | Si el empleado, carro, ruta o conductor no existen, se lanza una excepción con un mensaje de error. |
| **2** | Si las fechas son inválidas (fecha de inicio dentro de una programación existente o fecha de fin anterior a la fecha de inicio), se lanza una excepción. |
| **Comentarios** | Se debe asegurar que el empleado exista antes de iniciar con el proceso de programación | |

## Prototipo

**Figura 1:**

*Diagrama de secuencia del servicio de programación*



*Nota:* Elaboración propia, hecha en PowerPoint.

## Diagrama de secuencia

**Figura 2:**

*Diagrama de secuencia del servicio de programación*



*Nota:* Elaboración propia, hecha en PowerPoint.

## Modelo de base de dato

**Figura 3:**

*Modelo de base de datos del servicio programación*



*Nota*: Elaboración propia, hecha en Erwin Model Data

## Criterios de aceptación

- El sistema debe permitir asignar una ruta a un vehículo y un conductor específicos.

- La asignación debe registrar los detalles completos de la ruta, el vehículo, el conductor, la fecha de inicio, la fecha de fin programada y la fecha de fin real en la base de datos.

- El sistema debe validar que no haya rutas duplicadas para un mismo vehículo y conductor en el mismo periodo.

- El sistema debe mostrar un historial de asignaciones previas, incluyendo la fecha de asignación, ruta, vehículo, conductor y cualquier cambio realizado.

# CU02 – REGISTRO DE MANTENIMIENTO

## Especificación del caso de uso

|  |  |  |
| --- | --- | --- |
| **Código** | **CU02** | |
| **Nombre** | **Registro del Mantenimiento** | |
| **Descripción** | Permite registrar un mantenimiento para un carro en un taller específico, validando la existencia del empleado, carro y taller, así como las fechas de inicio y fin del mantenimiento. | |
| **Actores** | Empleado, Taller, Carro | |
| **Requerimiento asociado** | Validación de datos de mantenimiento  Actualización de estado del carro | |
| **Casos de uso asociados** | CU01 (Registro de Empleado)  CU02 (Registro de Taller) | |
| **Precondición** | El empleado debe estar autenticado en el sistema.  El carro, taller y empleado deben existir en la base de datos. | |
| **Secuencia normal** | **Paso** | **Acción** |
| **1** | El empleado inicia el registro de mantenimiento. |
| **2** | El sistema valida la existencia del empleado, del carro y del taller. |
| **3** | El sistema valida que la fecha de inicio sea anterior a la fecha de fin. |
| **4** | El sistema actualiza el estado del carro a "en mantenimiento". |
| **5** | El sistema registra el mantenimiento en la base de datos. |
| **6** | El sistema calcula y actualiza la calificación del taller. |
| **Postcondición** | El mantenimiento queda registrado y el carro cambia su estado. | |
| **Excepciones** | **Paso** | **Acción** |
| **1** | Si el empleado, carro o taller no existen, se lanza una excepción con un mensaje de error. |
| **2** | Si las fechas son inválidas (fecha de inicio posterior a la fecha de fin), se lanza una excepción. |
| **3** | Si ocurre un error al registrar el mantenimiento, se lanza una excepción. |
| **Comentarios** | Se debe asegurar que el taller y el empleado existan antes de registrar el mantenimiento. | |

## Prototipo

Figura 7:

Prototipo del servicio:



Nota: Elaboración propia, hecho en PowerPoint.

## Diagrama de secuencia

Figura 8:

*Diagrama de secuencia*



*Nota:* Elaboración propia, hecha en PowerPoint.

## Modelo de base de dato

**Figura 9:**

*Modelo de Base de Datos*

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

*Nota:* Elaboración propia, hecha en Erwin Model Data.

## Criterios de aceptación

1. **Validación de Identidad del Empleado:**
   * **Criterio:** El sistema debe verificar que el empleado que intenta registrar el mantenimiento esté autenticado y tenga los permisos necesarios.
   * **Aceptación:** Si el empleado no está autenticado, el sistema debe mostrar un mensaje claro que indique que se requiere iniciar sesión. Si el empleado no tiene permisos, debe recibir un mensaje que explique que no tiene acceso a la función de registro de mantenimiento.
2. **Existencia del Carro:**
   * **Criterio:** El sistema debe comprobar que el carro que se está registrando para mantenimiento existe en la base de datos.
   * **Aceptación:** Si el carro no existe, el sistema debe mostrar un mensaje de error que indique que el carro no se encuentra registrado. Además, debe ofrecer la opción de registrar un nuevo carro.
3. **Existencia del Taller:**
   * **Criterio:** El sistema debe validar que el taller donde se realizará el mantenimiento esté registrado y activo.
   * **Aceptación:** Si el taller no está en la base de datos, el sistema debe notificar al usuario con un mensaje que indique que el taller no es válido y sugerir la posibilidad de registrar un nuevo taller.
4. **Validación de Fechas:**
   * **Criterio:** El sistema debe asegurarse de que la fecha de inicio del mantenimiento sea anterior a la fecha de finalización.
   * **Aceptación:** Si la fecha de inicio es posterior a la fecha de fin, el sistema debe mostrar un mensaje de error claro que explique que las fechas son incorrectas y debe permitir al usuario corregirlas.
5. **Actualización del Estado del Carro:**
   * **Criterio:** Al registrar el mantenimiento, el estado del carro debe actualizarse automáticamente a "en mantenimiento".
   * **Aceptación:** Después de un registro exitoso, el sistema debe reflejar el nuevo estado del carro en la base de datos y en la interfaz de usuario, asegurando que cualquier consulta posterior muestre el estado actualizado.
6. **Registro en la Base de Datos:**
   * **Criterio:** El mantenimiento debe ser registrado correctamente en la base de datos con todos los detalles relevantes (empleado, carro, taller, fechas, etc.).
   * **Aceptación:** Tras completar el registro, se debe poder consultar el mantenimiento en la base de datos y verificar que todos los datos ingresados son correctos y están completos.
7. **Cálculo de Calificación del Taller:**
   * **Criterio:** El sistema debe calcular y actualizar la calificación del taller basado en el mantenimiento registrado.
   * **Aceptación:** Después de registrar el mantenimiento, el sistema debe reflejar el cambio en la calificación del taller, y esta debe ser visible en la interfaz de usuario, mostrando cómo el nuevo mantenimiento afecta la calificación.
8. **Manejo de Errores:**
   * **Criterio:** El sistema debe manejar adecuadamente cualquier error que ocurra durante el proceso de registro.
   * **Aceptación:** Si ocurre un error (por ejemplo, problemas de conexión a la base de datos), el sistema debe mostrar un mensaje de error amigable que informe al usuario del problema y sugiera intentar nuevamente más tarde.
9. **Interfaz de Usuario:**
   * **Criterio:** La interfaz de usuario debe ser intuitiva y fácil de usar, permitiendo al empleado completar el registro sin confusiones.
   * **Aceptación:** Los elementos de la interfaz deben estar claramente etiquetados, y el flujo de registro debe ser lógico. Se debe realizar una prueba de usabilidad para asegurar que los empleados puedan completar el registro sin necesidad de asistencia.
10. **Documentación y Ayuda:**
    * **Criterio:** Debe existir documentación accesible que explique cómo registrar un mantenimiento y qué hacer en caso de errores.
    * **Aceptación:** La documentación debe estar disponible en línea y ser fácil de encontrar. Además, debe incluir ejemplos y respuestas a preguntas frecuentes.

# CU03 – CANTIDAD DE INCIDENTES DE UN CONDUCTOR

## Especificación del caso de uso

|  |  |  |
| --- | --- | --- |
| **Código** | **CU03** | |
| **Nombre** | **Cantidad de incidentes de un conductor en un rango de fecha** | |
| **Descripción** | Con este código puedo evaluar la cantidad de incidentes de un conductor en un rango de fecha. | |
| **Actores** | Conductor, Incidente, Programación | |
| **Requerimiento asociado** | Consultar a un conductor válido en el sistema. | |
| **Casos de uso asociados** | CU04 (Registro de incidente) | |
| **Precondición** | El conductor debe existir en la base de datos, además de que las fechas estén en el formato correcto y que la fecha inicial sea menor a la fecha final. | |
| **Secuencia normal** | **Paso** | **Acción** |
| **1** | Valida que el conductor exista. |
| **2** | Valida que las fechas no sean nulas. |
| **3** | Valida que las fechas estén en el formato correcto. |
| **4** | Valida que la fecha inicial sea menor a la fecha final. |
| **5** | Busca el id\_conductor dentro de la tabla incidente considerando que la fecha que pone la tabla incidente sea mayor a la fecha inicial y menor a la fecha final. |
| **6** | Entrega un Dto en formato JSON que entrega el nombre, apellido del conductor y la cantidad de incidentes. |
| **Postcondición** | Se entrega el nombre, apellido y cantidad de incidentes. | |
| **Excepciones** | **Paso** | **Acción** |
| **1** | Si el conductor no existe, se lanza la excepción indicando que no existe el id\_conductor entregado. |
| **2** | Si la fecha inicio es menor a la fecha final, se indicará que el que elabora la consulta debe ordenar las fechas. |
| **Comentarios** | Es importante que los usuarios usen formatos válidos de fecha (DD/MM/AAAA) para evitar excepciones durante la consulta.  Este caso de uso es fundamental para mantener el historial de incidentes y el estado actualizado de los vehículos. | |

## Prototipo

**Figura 7:**

*Prototipo de la consulta*



Nota: Elaboración propia, hecha en PowerPoint.

## Diagrama de secuencia

**Figura 8:**

*Diagrama de secuencia del servicio de programación*



*Nota*: Elaboración propia, hecha un platuml.

## Modelo de base de dato

**Figura 9:**

*Modelo de la base de datos*



*Nota:* Elaboración propia, hecha en Erwin Model Data.

## Criterios de aceptación

1. **Validación de Conductor**

* Identificara si es que el conductor existe dentro de la base de datos, en caso contrario, te indicará que el conductor al que intentas consultar no existe.

1. **Validación de Fechas**

* Después de validar el id\_conductor, identificara si es que no escribiste nada en donde debe ir las fechas.
* Después de validar que no es nulo, validará si están en el formato correcto.
* Finalmente, el programa verificará que la fecha colocada en fecha inicio sea menor a la fecha final.

1. **Devolución del JSON**

* Se consultará a la tabla incidente sobre los incidentes generados por un conductor dentro del rango de fecha y se entregara su nombre, apellido y la cantidad de incidentes cometidos en el rango de fecha.

# CU04 – REGISTRO INCIDENTE

## Especificación del caso de uso

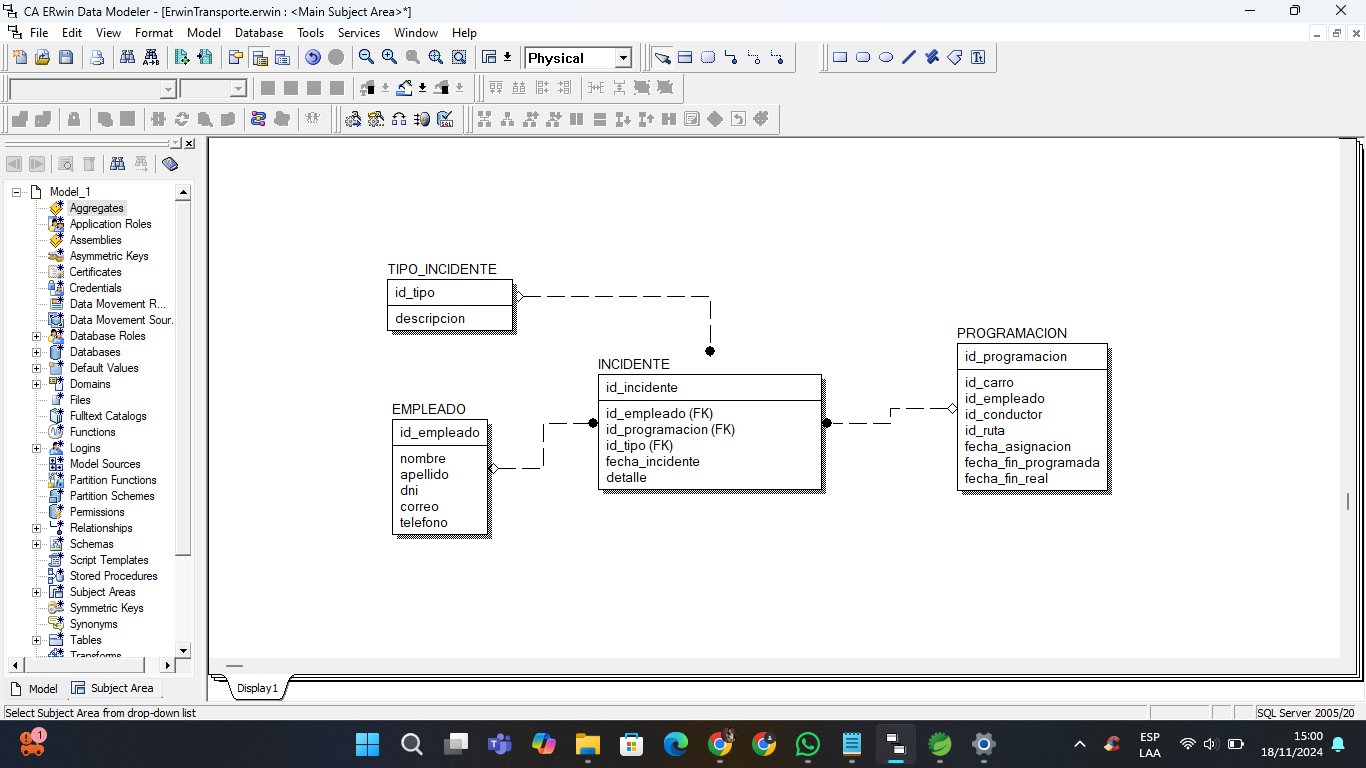
|  |  |  |
| --- | --- | --- |
| **Código** | **CU04** | |
| **Nombre** | **Registro del incidente** | |
| **Descripción** | Permite al sistema registrar un incidente asociado a una programación, validando que el empleado, la programación y el tipo de incidente existan en la base de datos. Incluye la fecha y detalles del incidente, y actualiza el estado del vehículo asociado a la programación en caso de éxito. | |
| **Actores** | Usuario del sistema (chofer), Empleado | |
| **Requerimiento asociado** | Registrar un incidente válido en el sistema, verificando las entidades relacionadas y actualizando el estado del vehículo en caso de ser necesario. | |
| **Casos de uso asociados** | CU01 (Registro de Programación) | |
| **Precondición** | El empleado debe existir en la base de datos, así como la Programación del vehículo del incidente. | |
| **Secuencia normal** | **Paso** | **Acción** |
| **1** | Valida que el empleado existe en la base de datos |
| **2** | Valida que la programación existe en la base de datos |
| **3** | Verifica que el tipo de incidente es válido |
| **4** | Inserta el incidente en la base de datos (INCIDENTE) con la información proporcionada. |
| **5** | Actualiza el estado del vehículo asociado a la programación a "En Reparación" (CARRO). |
| **6** | Devuelve la información registrada del incidente. |
| **Postcondición** | El incidente queda registrado y el vehículo es mandado en reparación. | |
| **Excepciones** | **Paso** | **Acción** |
| **1** | Si el empleado o programación no existen, se lanza una excepción con un mensaje de error. |
| **2** | Si la fecha del incidente no coindice con la ruta programada, se enviará un mensaje de excepción |
| **Comentarios** | Es importante que los usuarios usen formatos válidos de fecha (DD/MM/YYYY) para evitar excepciones durante el registro.  Este caso de uso es fundamental para mantener el historial de incidentes y el estado actualizado de los vehículos. | |

## Prototipo



## Diagrama de secuencia

## Modelo de base de dato



## Criterios de aceptación

1. **Validación de Empleado**

**Criterio de Aceptación**:

* Dado que el sistema recibe una solicitud de registro de incidente,
* Cuando se consulta la base de datos para verificar la existencia del empleado especificado,
* Entonces si el empleado existe, el sistema debe continuar con el proceso.
* De lo contrario, si el empleado no existe, el sistema debe arrojar un error con el mensaje "El empleado no existe".

**2. Validación de Programación**

**Criterio de Aceptación**:

* Dado que el sistema recibe una solicitud para registrar un incidente,
* Cuando se consulta la base de datos para verificar la existencia de la programación especificada,
* Entonces si la programación existe, el sistema debe continuar con el proceso.
* De lo contrario, si la programación no existe, el sistema debe arrojar un error con el mensaje "La programación no existe".

**3. Validación de Tipo de Incidente**

**Criterio de Aceptación**:

* Dado que el sistema recibe una solicitud de registro de incidente,
* Cuando se consulta la base de datos para verificar la existencia del tipo\_incidente especificado,
* Entonces si el tipo de incidente es válido, el sistema debe continuar con el proceso.
* De lo contrario, si el tipo de incidente no existe, el sistema debe arrojar un error con el mensaje "El tipo de incidente no es válido".

**4. Registro del Incidente en la Base de Datos**

**Criterio de Aceptación**:

* Dado que el empleado, la programación y el tipo de incidente han sido validados con éxito,
* Cuando el incidente es registrado en la base de datos,
* Entonces el sistema debe insertar un nuevo registro en la tabla INCIDENTE con los siguientes campos:
  + id\_empleado,
  + id\_programacion,
  + id\_tipo,
  + fecha\_incidente,
  + detalle.
* De lo contrario, si ocurre un error durante la inserción, el sistema debe revertir la transacción y no guardar el incidente.

**5. Actualización del Estado del Vehículo**

**Criterio de Aceptación**:

* Dado que el incidente se ha registrado correctamente,
* Cuando el sistema actualiza el estado del vehículo asociado a la programación,
* Entonces el estado de ese vehículo debe actualizarse en la base de datos a "En Reparación".
* De lo contrario, si ocurre un error durante la actualización del estado, el sistema debe revertir la transacción y no realizar el cambio en el estado del vehículo.

**6. Manejo de Transacciones**

**Criterio de Aceptación**:

* Dado que el sistema está registrando un incidente,
* Cuando ocurre un error en cualquiera de los pasos de la validación o actualización (empleado, programación, tipo de incidente, registro o actualización del estado),
* Entonces la transacción completa debe revertirse, y no debe persistir ningún cambio en la base de datos (incidente no registrado, estado del vehículo no actualizado).
* De lo contrario, si todo el proceso es exitoso, los cambios deben persistir en la base de datos.

**7. Manejo de Excepciones**

**Criterio de Aceptación**:

* Dado que el sistema recibe una solicitud de registro de incidente,
* Cuando cualquier validación o paso en el proceso falla,
* Entonces el sistema debe lanzar la excepción correspondiente con un mensaje claro que explique el error (por ejemplo, "El empleado no existe", "La programación no existe", "El tipo de incidente no es válido", etc.).
* De lo contrario, si no se produce ningún error, el proceso debe continuar y completarse exitosamente.

# CU05 – REGISTRO DE UNA REPARACIÓN

## Especificación del caso de uso

|  |  |  |
| --- | --- | --- |
| **Código** | **CU05** | |
| **Nombre** | **Registro de una reparación** | |
| **Descripción** | Permite al empleado poder registrar en el sistema una reparación completada de un auto que tuvo un incidente, validando varios aspectos como el incidente o la programación del transporte | |
| **Actores** | Empleado, Sistema | |
| **Requerimiento asociado** | Registrar los datos necesarios relacionados a una reparación en el sistema | |
| **Casos de uso asociados** | Registro de Programación  Registro de Incidente | |
| **Precondición** | El empleado debe estar registrado en el sistema y tener acceso a este para poder registrar una reparación | |
| **Secuencia normal** | **Paso** | **Acción** |
| **1** | Validar que exista un incidente del cuál se registrará la reparación del auto |
| **2** | Validar que el auto se encuentre en estado de reparación. |
| **3** | Validar el taller en el cual se realizó la reparación. |
| **4** | Validar que la fecha registrada de la reparación sea la correcta. |
| **5** | El sistema valida la información ingresada. |
| **6** | El sistema registra la reparación en la base de datos y actualiza el estado del auto. |
| **Postcondición** | La reparación quedará registrada y el vehículo volverá a estar en estado disponible. | |
| **Excepciones** | **Paso** | **Acción** |
| **1** | Si los detalles de la reparación están incorrectos el sistema muestra un error. |
| **2** | Cualquier fallo en el proceso del registro de la reparación no será registrado en la base de datos. |
| **Comentarios** | Tener en cuenta que para que haya un incidente del cuál se registrará la reparación de un auto también debe existir una programación de un auto del cuál ocurre el incidente. | |

## Prototipo

**Figura 13:**

*Prototipo del servicio de registrar una reparación*

Interfaz de usuario gráfica

Descripción generada automáticamente

*Nota:* Elaboración propia, hecha en PowerPoint.

## Diagrama de secuencia

**Figura 14:**

*Diagrama de secuencia del servicio de registrar una reparación*



*Nota:* Elaboración propia, hecha en PowerPoint.

## Modelo de base de dato

**Figura 14:**

*Modelo de base de datos del servicio de registrar una reparación*



*Nota*: Elaboración propia en Erwin Model Data

## Criterios de aceptación

- La reparación se registra correctamente en el sistema.

- El sistema valida los datos ingresados y asegura su consistencia.

- La información queda disponible para consulta inmediata.

# CU06 – FRECUENCIA DE INCIDENTES

## Especificación del caso de uso

|  |  |  |
| --- | --- | --- |
| **Código** | **CU06** | |
| **Nombre** | **Frecuencia de incidentes** | |
| **Descripción** | Se dará a conocer la cantidad de veces que a ocurrido un tipo de incidente. | |
| **Actores** | ACT01 | |
| **Requerimiento asociado** | Registrar un id tipo incidente válido en el sistema | |
| **Casos de uso asociados** | CU04 (Registro de Incidente) | |
| **Precondición** | El id tipo debe existir en la base de datos. | |
| **Secuencia normal** | **Paso** | **Acción** |
| **1** | Validará la existencia del id tipo incidente. |
| **2** | Buscará dentro de la tabla incidente la cantidad de incidentes ocurridos. |
| **3** | Devolverá un JSON que indicara el id tipo, descripción del incidente y la cantidad de veces que hubo un incidente |
| **Postcondición** | Se entrega un JSON que involucra la descripción del incidente y la cantidad de veces que ocurrieron. | |
| **Excepciones** | **Paso** | **Acción** |
| **1** | Si el id tipo no existe, te indica error y q ese tipo no existe |
| **Comentarios** | Este servicio es importante para considerar que tipo de incidente es el más común, gracias a los conocimientos de estadistíca se puede llegar a evaluar que incidentes serán los más problemáticos y solucionar los problemas que pueden existir. | |

## Prototipo

**Figura 13:**

*Prototipo del servicio*

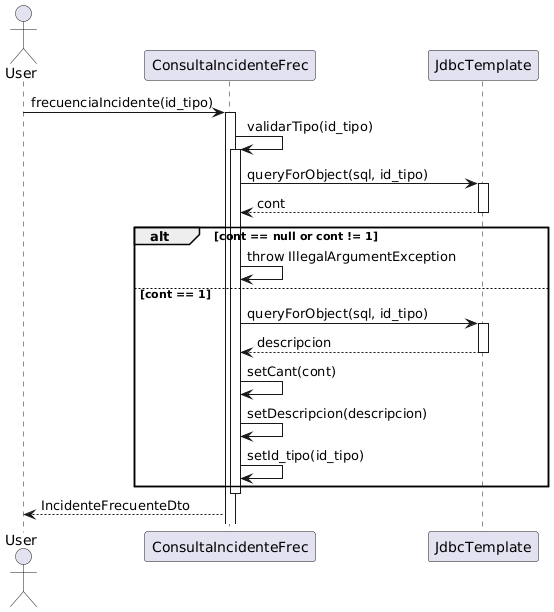


*Nota:* Elaboración propia, hecho en PowerPoint.

## Diagrama de secuencia

**Figura 14:**

Diagrama de secuencia del servicio



*Nota:* Elaboración propia, hecho en plantUml.

## Modelo de base de dato

**Figura 14:**

*Modelo de la base de datos del servicio*



*Nota:* Elaboración propia, hecho en Erwin Model Data.

## Criterios de aceptación

* Valida la existencia del id\_tipo en la tabla tipo\_incidente.

# MODELO DE BASE DE DATOS

## Modelo Conceptual

El modelo conceptual del sistema de transporte fue diseñado utilizando un diagrama Entidad-Relación (E-R), el cual representa las principales entidades, atributos y relaciones del sistema. A continuación, se describe su estructura:

- Entidades principales: Reparación, Estado, Conductor, Ruta, Taller, Mantenimiento, Programación, Empleado, Carro, Incidente, Tipo de Incidente, Estado de Mantenimiento.

- Relaciones clave: Cada carro puede tener asignado un estado y estar relacionado con programaciones, mantenimientos o reparaciones.

- Los incidentes pueden estar vinculados a un tipo específico y a programaciones.

- Los mantenimientos son realizados en talleres y tienen un estado asociado.

- Cardinalidades: Estas se detallan gráficamente en el diagrama incluido más adelante.

Diagrama conceptual del diseño realizado en Erwin

Diagrama, Esquemático

Descripción generada automáticamente

## Modelo Relacional

- Tablas principales: Reparación, Estado, Conductor, Ruta, Taller, Mantenimiento, Programación, Empleado, Carro, Incidente, Tipo de Incidente, Estado de Mantenimiento.

- Claves primarias y foráneas: Cada tabla tiene su identificador único como clave primaria, y las relaciones se implementan mediante claves foráneas. Por ejemplo, 'id\_empleado' en la tabla Reparación es una clave foránea referenciando a la tabla Empleado.

## Diccionario de Datos

**Tabla 1**:

*Empleado*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_empleado | INT |  | IDENTITY |  | Identificador único del empleado. |
| nombre | VARCHAR | 50 | NOT NULL |  | Nombre del empleado. |
| apellido | VARCHAR | 50 | NOT NULL |  | Apellido del empleado |
| dni | VARCHAR | 8 | NOT NULL |  | Documento de identificación |
| correo | VARCHAR | 150 | NOT NULL |  | Correo del empleado |
| telefono | VARCHAR | 20 | NOT NULL |  | Numero de contacto |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Incidente

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_incidente | INT |  | IDENTITY |  | Identificador |
| id\_empleado | INT |  | NOT NULL | KF -> EMPLEADO (id\_empleado) | Nombre del empleado. |
| id\_programacion | INT |  | NOT NULL | KF -> PROGRAMACION (id\_programacion) | Apellido del empleado |
| Id\_tipo | INT |  | NOT NULL | KF -> TIPO\_INCIDENTE (id\_tipo) | Documento de identificación |
| Fecha\_incidente | DATE |  | NOT NULL |  | Correo del empleado |
| detalle | VARCHAR | 200 | NOT NULL |  | Numero de contacto |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Reparacion

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_reparacion | INT |  | IDENTITY |  | identificador único de la reparación |
| id\_empleado | INT |  | NULL | KF -> EMPLEADO (id\_empleado) | identificador único del empleado |
| id\_incidente | INT |  | NULL | KF -> TIPO\_INCIDENTE (id\_tipo) | identificador único del incidente |
| id\_taller | INT |  | NULL | KF -> TALLER (id\_taller) | identificador único en el taller |
| fecha\_reparacion | DATE |  | NOT NULL |  | fecha de la reparación completada |
| calificacion | DECIMAL (5,1) | 200 | NOT NULL |  | calificación del servicio |
| costo | DECIMAL (10,2) |  | NOT NULL |  | costo del servicio |
| detalle | VARCHAR | 2000 | NOT NULL |  | detalle de la reparación |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Taller

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| ID\_taller | INT |  | IDENTITY |  | identificador único De El taller |
| nombre\_ taller | VARCHAR | 50 | NOT NULL |  | nombre de El taller |
| dirección | VARCHAR | 50 | NOT NULL |  | ubicación del taller |
| teléfono | VARCHAR | 20 | NOT NULL |  | número de contacto del taller |
| calificación | DECIMAL(5,1) |  | NOT NULL |  | calificación del servicio |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Mantenimiento

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_mantenimiento | INT |  | IDENTITY |  | identificador único del mantenimiento |
| id\_empleado | INT |  | NULL | KF -> EMPLEADO (id\_empleado) | identificador único del empleado |
| id\_taller | INT |  | NULL | KF -> TALLER (id\_taller) | identificador único del taller |
| Id\_est\_mant | INT |  | NULL | KF -> EST\_MANTENIMIENTO (id\_est\_mant) | identificador único del estado del mantenimiento |
| id\_ carro | INT |  |  | KF -> CONDUCTOR (id\_conductor) | identificador único del carro |
| calificación | DECIMAL(5,1) |  | NOT NULL |  | calificación del servicio |
| fecha\_ inicio | DATE |  | NOT NULL |  | fecha de inicio del mantenimiento |
| fecha\_ salida\_ programada | DATE |  | NOT NULL |  | fecha de la salida programada del mantenimiento |
| fecha\_ salida\_ real | DATE |  | NULL |  | fecha de la salida si ocurre un percance |
| costo | DECIMAL(10,1) |  | NOT NULL |  | costo del servicio |
| detalle | VARCHAR | 2000 | NOT NULL |  | detalle del servicio |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

**Tabla 6.**

*Estado del Mantenimiento*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_est\_mant | INT |  | IDENTITY |  | identificador único del estado del mantenimiento |
| descripcion | VARCHAR | 50 | NOT NULL |  | Descripción del estado del mantenimiento |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Programacion

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_programación | INT |  | IDENTITY |  | identificador único una programación |
| id\_carro | INT |  | NOT NULL | KF -> CARRO (id\_carro) | identificador único del carro |
| id\_empleado | INT |  | NOT NULL | KF -> EMPLEADO (id\_empleado) | identificador único del empleado |
| Id\_conductor | INT |  | NOT NULL | KF -> CONDUCTOR (id\_conductor) | identificador único del conductor |
| id\_ ruta | INT |  | NOT NULL | KF -> RUTA (id\_ruta) | identificador único de una ruta |
| fecha\_asignación | DATE | 200 | NOT NULL |  | fecha del inicio de la programación |
| fecha \_ fin\_ programada | DATE |  | NOT NULL |  | fecha del fin de la programación |
| fecha\_ fin\_ real | DATE | 2000 | NOT NULL |  | fecha real del fin de la programación |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Conductor

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_conductor | INT |  | IDENTITY |  | Identificador único del conductor |
| nombre | VARCHAR | 100 | NOT NULL |  | Nombre del conductor |
| apellido | VARCHAR | 100 | NOT NULL |  | Apellido del conductor |
| dni | VARCHAR | 8 | NOT NULL |  | Documento de identificación |
| correo | VARCHAR | 150 | NOT NULL |  | Correo del conductor |
| telefono | VARCHAR | 20 | NOT NULL |  | Numero de contacto |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Ruta

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_ruta | INT |  | IDENTITY |  | Identificador único de una ruta |
| nombre\_ ruta | VARCHAR | 100 | NOT NULL |  | Nombre del conductor |
| origen | VARCHAR | 100 | NOT NULL |  | Lugar de salida de la ruta |
| destino | VARCHAR | 100 | NOT NULL |  | Lugar de destino de la ruta |
| Distancia\_Km | VARCHAR | DECIMAL(5,2) | NOT NULL |  | distancia el de ruta |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Carro

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_carro | INT |  | IDENTITY |  | identificador único del carro |
| id\_ estado | INT |  | NOT NULL | KF -> ESTADO(id\_estado) | estado del carro |
| placa | VARCHAR | 6 | NOT NULL |  | placa del carro |
| prox\_mant | DATE |  | NOT NULL |  | fecha próxima del mantenimiento del carro |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Estado

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_estado | INT | - | IDENTITY |  | estado o situación del carro |
| descripcion | VARCHAR | 50 | NOT NULL |  | Describe el estado en el cual se encuentra del carro |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

Tabla . Tipo de incidente

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Campo | Tipo | Longitud | Restricciones | Relación | Descripción |
| id\_tipo | INT |  | IDENTITY |  | identificador único del incidente |
| descripcion | VARCHAR | 50 | NOT NULL |  | Describe el tipo de incidente |

*Nota:* La columna 1 contiene el nombre del atributo en la base de datos; la columna 2 especifica el tipo de dato; la columna 3, la longitud máxima permitida; la columna 4, las restricciones aplicables; la columna 5, las relaciones con otras tablas, y la columna 6, una descripción breve del atributo.

# IMPLEMENTACIÓN DE LA BASE DE DATOS

## Creación de la Base de Datos

Para crear la base de datos se debe utilizar la instrucción **CREATE DATABASE**, como se ilustra a continuación:

USE master;

GO

-- Si la base de datos ya existe, se elimina

IF EXISTS (SELECT name FROM sys.databases WHERE name = 'TRANSPORTE')

BEGIN

DROP DATABASE TRANSPORTE;

END;

GO

-- Crear la base de datos TRANSPORTE

CREATE DATABASE TRANSPORTE;

GO

-- Usar la base de datos recién creada

USE TRANSPORTE;

GO

En este caso se está creando la base de datos **TRANSPORTE**;

## Creación de los objetos de la base de datos

-- Tabla de Estado

CREATE TABLE ESTADO (

id\_estado INT IDENTITY PRIMARY KEY,

descripcion VARCHAR(50) NOT NULL

);

-- Tabla de Carro

CREATE TABLE CARRO (

id\_carro INT IDENTITY PRIMARY KEY,

id\_estado INT NOT NULL FOREIGN KEY REFERENCES ESTADO(id\_estado),

placa VARCHAR(6) NOT NULL UNIQUE,

prox\_mant DATE NOT NULL

);

-- Tabla de Estado de Mantenimiento

CREATE TABLE EST\_MANTENIMIENTO (

id\_est\_mant INT IDENTITY PRIMARY KEY,

descripcion VARCHAR(50) NOT NULL

);

-- Tabla de Conductor

CREATE TABLE CONDUCTOR (

id\_conductor INT IDENTITY PRIMARY KEY,

nombre VARCHAR(100) NOT NULL,

apellido VARCHAR(100) NOT NULL,

dni VARCHAR(8) NOT NULL,

correo VARCHAR(150) NOT NULL,

telefono VARCHAR(20) NOT NULL

);

-- Tabla de Empleado

CREATE TABLE EMPLEADO (

id\_empleado INT IDENTITY PRIMARY KEY,

nombre VARCHAR(50) NOT NULL,

apellido VARCHAR(50) NOT NULL,

dni VARCHAR(8) NOT NULL,

correo VARCHAR(150) NOT NULL,

telefono VARCHAR(20) NOT NULL

);

-- Tabla de Rutas

CREATE TABLE RUTA (

id\_ruta INT IDENTITY PRIMARY KEY,

nombre\_ruta VARCHAR(100) NOT NULL,

origen VARCHAR(100) NOT NULL,

destino VARCHAR(100) NOT NULL,

distancia\_km DECIMAL(5, 2) NOT NULL

);

-- Tabla de Taller

CREATE TABLE TALLER (

id\_taller INT IDENTITY PRIMARY KEY,

nombre\_taller VARCHAR(50) NOT NULL,

direccion VARCHAR(50) NOT NULL,

telefono VARCHAR(20) NOT NULL,

calificacion DECIMAL(5, 1) NOT NULL

);

-- Tabla de Mantenimientos

CREATE TABLE MANTENIMIENTO (

id\_mantenimiento INT IDENTITY PRIMARY KEY,

id\_empleado INT NOT NULL FOREIGN KEY REFERENCES EMPLEADO(id\_empleado),

id\_taller INT NOT NULL FOREIGN KEY REFERENCES TALLER(id\_taller),

id\_est\_mant INT NOT NULL FOREIGN KEY REFERENCES EST\_MANTENIMIENTO(id\_est\_mant),

id\_carro INT NOT NULL FOREIGN KEY REFERENCES CARRO(id\_carro),

calificacion DECIMAL(5, 1) NOT NULL,

fecha\_inicio DATE NOT NULL,

fecha\_salida\_programada DATE NOT NULL,

fecha\_salida\_real DATE,

costo DECIMAL(10, 2) NOT NULL,

detalle VARCHAR(2000) NOT NULL

);

-- Tabla de Tipo de Incidente

CREATE TABLE TIPO\_INCIDENTE (

id\_tipo INT IDENTITY PRIMARY KEY,

descripcion VARCHAR(50) NOT NULL

);

-- Tabla de Programación

CREATE TABLE PROGRAMACION (

id\_programacion INT IDENTITY PRIMARY KEY,

id\_carro INT NOT NULL FOREIGN KEY REFERENCES CARRO(id\_carro),

id\_empleado INT NOT NULL FOREIGN KEY REFERENCES EMPLEADO(id\_empleado),

id\_conductor INT NOT NULL FOREIGN KEY REFERENCES CONDUCTOR(id\_conductor),

id\_ruta INT NOT NULL FOREIGN KEY REFERENCES RUTA(id\_ruta),

fecha\_asignacion DATE NOT NULL,

fecha\_fin\_programada DATE NOT NULL,

fecha\_fin\_real DATE

);

-- Tabla de Incidentes

CREATE TABLE INCIDENTE (

id\_incidente INT IDENTITY PRIMARY KEY,

id\_empleado INT NOT NULL FOREIGN KEY REFERENCES EMPLEADO(id\_empleado),

id\_programacion INT NOT NULL FOREIGN KEY REFERENCES PROGRAMACION(id\_programacion),

id\_tipo INT NOT NULL FOREIGN KEY REFERENCES TIPO\_INCIDENTE(id\_tipo),

fecha\_incidente DATE NOT NULL,

detalle VARCHAR(2000) NOT NULL

);

-- Tabla de Reparaciones

CREATE TABLE REPARACION (

id\_reparacion INT IDENTITY PRIMARY KEY,

id\_empleado INT FOREIGN KEY REFERENCES EMPLEADO(id\_empleado),

id\_incidente INT FOREIGN KEY REFERENCES INCIDENTE(id\_incidente),

id\_taller INT FOREIGN KEY REFERENCES TALLER(id\_taller),

fecha\_reparacion DATE NOT NULL,

calificacion DECIMAL(5, 1) NOT NULL,

costo DECIMAL(10, 2) NOT NULL,

detalle VARCHAR(2000) NOT NULL

);

GO

## Cargar Datos de Prueba

INSERT INTO ESTADO (descripcion) VALUES

('Disponible'),

('En Mantenimiento'),

('En Reparación'),

('Fuera de Servicio'),

('Reservado');

Go

INSERT INTO CARRO (id\_estado, placa, prox\_mant) VALUES

(1, 'ABC123', '2024-11-15'),

(2, 'DEF456', '2024-12-20'),

(1, 'GHI789', '2025-01-10'),

(3, 'JKL012', '2025-02-05'),

(4, 'MNO345', '2024-12-30');

Go

INSERT INTO EST\_MANTENIMIENTO (descripcion) VALUES

('Programado'),

('En Proceso'),

('Completado'),

('Cancelado'),

('Pendiente');

Go

INSERT INTO CONDUCTOR (nombre, apellido, dni, correo, telefono) VALUES

('Juan', 'Pérez', '12345678', 'juan.perez@example.com', '987654321'),

('María', 'Gómez', '23456789', 'maria.gomez@example.com', '987654322'),

('Carlos', 'Sánchez', '34567890', 'carlos.sanchez@example.com', '987654323'),

('Ana', 'Martínez', '45678901', 'ana.martinez@example.com', '987654324'),

('Luis', 'Torres', '56789012', 'luis.torres@example.com', '987654325');

Go

INSERT INTO EMPLEADO (nombre, apellido, dni, correo, telefono) VALUES

('Pedro', 'Ramírez', '67890123', 'pedro.ramirez@example.com', '987654326'),

('Laura', 'Hernández', '78901234', 'laura.hernandez@example.com', '987654327'),

('Jorge', 'Díaz', '89012345', 'jorge.diaz@example.com', '987654328'),

('Clara', 'Vásquez', '90123456', 'clara.vasquez@example.com', '987654329'),

('Fabian', 'Cruz', '01234567', 'diego.cruz@example.com', '987654330');

Go

INSERT INTO RUTA (nombre\_ruta, origen, destino, distancia\_km) VALUES

('Ruta 1', 'Ciudad A', 'Ciudad B', 120.50),

('Ruta 2', 'Ciudad B', 'Ciudad C', 150.75),

('Ruta 3', 'Ciudad C', 'Ciudad D', 200.00),

('Ruta 4', 'Ciudad D', 'Ciudad E', 175.25),

('Ruta 5', 'Ciudad E', 'Ciudad A', 300.00);

Go

INSERT INTO TALLER (nombre\_taller, direccion, telefono, calificacion) VALUES

('Taller Mecánico A', 'Av. Principal 123', '987654331',3),

('Taller Mecánico B', 'Calle Secundaria 456', '987654332',4),

('Taller Mecánico C', 'Calle Tercera 789', '987654333',3.5),

('Taller Mecánico D', 'Av. Cuarta 321', '987654334',4.5),

('Taller Mecánico E', 'Calle Quinta 654', '987654335',1.5);

Go

INSERT INTO MANTENIMIENTO (id\_taller, id\_empleado , id\_est\_mant, id\_carro, calificacion, fecha\_inicio, fecha\_salida\_programada, fecha\_salida\_real, costo, detalle) VALUES

(1, 1, 1, 1, 2,'2024-01-10', '2024-01-15', '2024-01-14', 500.00, 'Carro en buen estado, no se encontraron fallas'),

(2, 2, 2, 2, 3,'2024-02-01', '2024-02-20', '2024-02-19', 600.00, 'Se inflaron las llantas'),

(3, 3, 1, 3, 4,'2024-03-01', '2024-03-10', '2024-03-09', 450.00, 'Se realizó cambio de aceite'),

(4, 4, 3, 4, 5,'2024-04-01', '2024-04-05', '2024-04-04', 700.00, 'Carro en buen estado'),

(5, 5, 2, 5, 1,'2024-05-01', '2024-05-30', '2024-05-29', 550.00, 'Carro en buen estado, no se encontraron fallas');

Go

INSERT INTO TIPO\_INCIDENTE (descripcion) VALUES

('Accidente en carretera'),

('Falla Mecánica'),

('Problema Eléctrico'),

('Incidente de Seguridad'),

('Daños por Vandalismo');

Go

INSERT INTO PROGRAMACION (id\_carro, id\_empleado, id\_conductor, id\_ruta, fecha\_asignacion, fecha\_fin\_programada, fecha\_fin\_real) VALUES

(1, 1, 1, 1, '2024-01-01', '2024-01-15', '2024-01-14'),

(2, 2, 2, 2, '2024-02-01', '2024-02-20', '2024-02-20'),

(3, 3, 3, 3, '2024-03-01', '2024-03-10', '2024-03-09'),

(4, 4, 4, 4, '2024-04-01', '2024-04-05', '2024-04-04'),

(5, 5, 5, 5, '2024-05-01', '2024-05-30', '2024-05-30');

Go

INSERT INTO INCIDENTE (id\_empleado, id\_programacion, id\_tipo, fecha\_incidente, detalle) VALUES

(1, 1, 1, '2024-01-12', 'Ocurrió 30 minutos de partir. No hubo heridos'),

(2, 2, 2, '2024-02-15', 'Fallo en el motor. No hubo heridos'),

(3, 3, 3, '2024-03-05', 'Problema con la batería. No hubo heridos'),

(4, 4, 4, '2024-04-03', 'La puerta no cerraba correctamente'),

(5, 5, 5, '2024-05-25', 'Cinco adolescentes intervenidos. No hubo heridos');

Go

INSERT INTO REPARACION (id\_incidente, id\_empleado, id\_taller, fecha\_reparacion, calificacion, costo, detalle) VALUES

(1, 1, 1, '2024-01-13', 4, 200.00, 'Capot reparado'),

(2, 2, 2, '2024-02-16', 5, 300.00, 'Motor reparado'),

(3, 3, 3, '2024-03-06', 3, 250.00, 'Cambio de batería'),

(4, 4, 4, '2024-04-04', 4, 400.00, 'Puerta reparada'),

(5, 5, 5, '2024-05-26', 2, 350.00, 'Ventanas cambiadas. Se aplicó una capa de pintura al carro');

Go

# CODIGO FUENTE

Se presentan los códigos fuente de alguno de los servicios más importantes:

1. **Registrar Programación:**
   1. Código fuente:

package pe.edu.uni.proyecto.service;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.time.format.DateTimeParseException;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import pe.edu.uni.proyecto.dto.ProgramacionDto;

@Service

public class ProgramacionService {

@Autowired

private JdbcTemplate jdbcTemplate;

@Transactional(propagation = Propagation.REQUIRES\_NEW, rollbackFor = Exception.class)

public ProgramacionDto registrarProg(ProgramacionDto bean) {

validarEmpleado(bean.getIdEmpleado());

validarCarro(bean.getIdCarro());

validarEstadoCarro(bean.getIdCarro());

validarConductor(bean.getIdConductor());

validarEstadoConductor(bean.getIdConductor());

validarRuta(bean.getIdRuta());

bean.setFechaAsignacion(convertirFecha(bean.getFechaAsignacion()));

validarFechaPartida(bean.getIdConductor(), bean.getFechaAsignacion());

bean.setFechaFinProgramada(convertirFecha(bean.getFechaFinProgramada()));

validarFechaFin(bean.getFechaFinProgramada(), bean.getFechaAsignacion());

//Registrar Programacion

registrarProgamacion(bean);

return bean;

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarEmpleado(int idEmpleado) {

String sql = "select count(1) cont from EMPLEADO where id\_empleado = ?";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, idEmpleado);

if(cont != 1) {

throw new RuntimeException("Empleado " + idEmpleado + " no existe");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarCarro(int idCarro) {

String sql = "select count(1) cont from CARRO where id\_carro = ?";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, idCarro);

if(cont != 1) {

throw new RuntimeException("Carro " + idCarro + " no existe");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarEstadoCarro(int idCarro) {

String sql = "select id\_estado estado from CARRO where id\_carro = ?";

int estado = jdbcTemplate.queryForObject(sql, Integer.class, idCarro);

if(estado != 1) {

throw new RuntimeException("Carro " + idCarro + " no disponible");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarConductor(int idConductor) {

String sql = "select count(1) cont from CONDUCTOR where id\_conductor = ?";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, idConductor);

if(cont != 1) {

throw new RuntimeException("Conductor " + idConductor + " no existe");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarEstadoConductor(int idConductor) {

String sql = "select id\_estado estado from CONDUCTOR where id\_conductor = ?";

int estado = jdbcTemplate.queryForObject(sql, Integer.class, idConductor);

if(estado != 1) {

throw new RuntimeException("Conductor " + idConductor + " no disponible");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarRuta(int idRuta) {

String sql = "select count(1) cont from RUTA where id\_ruta = ?";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, idRuta);

if(cont != 1) {

throw new RuntimeException("Ruta " + idRuta + " no existe");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

public String convertirFecha(String fecha) {

try {

// Definir los formatos: de entrada (dd/MM/yyyy) y de salida (yyyy-MM-dd)

DateTimeFormatter inputFormatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

DateTimeFormatter outputFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

// Parsear la fecha de entrada y formatearla al nuevo formato

LocalDate date = LocalDate.parse(fecha, inputFormatter);

return date.format(outputFormatter);

} catch (DateTimeParseException e) {

throw new RuntimeException("Formato de fecha inválido. Asegúrese de usar el formato dd/MM/yyyy");

} catch (NullPointerException e) {

throw new RuntimeException("Las fechas no pueden ser nulas.");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarFechaPartida(int idConductor, String fechaPartida) {

String sql = "select top 1 cast(fecha\_fin\_programada as date) ult\_fecha from PROGRAMACION ";

sql += "where id\_conductor = ? order by fecha\_fin\_programada desc";

String fecha = jdbcTemplate.queryForObject(sql, String.class, idConductor);

sql = "select DATEDIFF(DAY,'" + fecha + "','" + fechaPartida + "')";

int dif = jdbcTemplate.queryForObject(sql, Integer.class);

if(dif<0) {

throw new RuntimeException("Fecha de partida " + fechaPartida + " no valida");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarFechaFin(String fechaFin, String fechaPartida) {

fechaPartida = "'" + fechaPartida + "'";

fechaFin = "'" + fechaFin + "'";

String sql = "select DATEDIFF(DAY," + fechaPartida + "," + fechaFin + ")";

int dif = jdbcTemplate.queryForObject(sql, Integer.class);

if(dif<=0) {

throw new RuntimeException("Fecha final " + fechaFin + " no valida");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void registrarProgamacion(ProgramacionDto bean) {

String sql = "insert into PROGRAMACION values(?,?,?,?,?,?,NULL)";

jdbcTemplate.update(sql,bean.getIdCarro(),bean.getIdEmpleado(),bean.getIdConductor(),

bean.getIdRuta(),bean.getFechaAsignacion(),bean.getFechaFinProgramada());

}

}

* 1. Prueba en postman:

1. {
2. "idEmpleado": 2,
3. "idCarro": 3,
4. "idConductor": 5,
5. "idRuta": 2,
6. "fechaAsignacion": "20/01/2025",
7. "fechaFinProgramada": "23/01/2025"
8. }

Resultado:

{

    "idProgramacion": 0,

    "idCarro": 3,

    "idEmpleado": 2,

    "idConductor": 5,

    "idRuta": 2,

    "fechaAsignacion": "2025-01-20",

    "fechaFinProgramada": "2025-01-23",

    "fechaFinReal": null

}

1. **Registrar Incidente:**

2.1) Código fuente:

package pe.edu.uni.proyecto.service;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import pe.edu.uni.proyecto.dto.IncidenteDto;

@Service

public class IncidenteService {

@Autowired

private JdbcTemplate jdbcTemplate;

@Transactional(propagation = Propagation.REQUIRES\_NEW, rollbackFor = Exception.class)

public IncidenteDto reportarincidente(IncidenteDto bean) {

// Validaciones

validarEmpleado(bean.getEmpleado());

validarProgramacion(bean.getProgramacion());

validarFecha(bean.getFecha(), bean.getProgramacion());

// Proceso

registrarIncidente(bean.getEmpleado(), bean.getProgramacion(), bean.getTipo\_incidente(), bean.getFecha(), bean.getDetalle());

actualizarEstadoCarro(bean.getProgramacion());

// Reporte final

System.out.println("Proceso ok.");

return bean;

}

// Validaciones para registrar el incidente

// validar el empleado

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarEmpleado(int empleado) {

String sql = "SELECT COUNT(1) cont FROM EMPLEADO where id\_empleado = ?";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, empleado);

if (cont != 1) {

throw new RuntimeException("El empleado no existe.");

}

}

// validar que la programación exista

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarProgramacion(int programacion) {

String sql = "SELECT COUNT(1) cont FROM PROGRAMACION where id\_programacion = ?";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, programacion);

if (cont != 1) {

throw new RuntimeException("La programacion no existe.");

}

}

// validar que la fecha del incidente esté en el rango de de las fechas programadas

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarFecha(String fecha, int programacion)

{

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

LocalDate Fecha = LocalDate.parse(fecha, formatter);

String sql1 = "SELECT fecha\_asignacion FROM PROGRAMACION where id\_programacion = ?";

LocalDate fecha\_inicio = jdbcTemplate.queryForObject(sql1, LocalDate.class, programacion);

String sql2 = "SELECT fecha\_fin\_programada FROM PROGRAMACION where id\_programacion = ?";

LocalDate fecha\_fin = jdbcTemplate.queryForObject(sql2, LocalDate.class, programacion);

if ((Fecha.isEqual(fecha\_inicio) || Fecha.isAfter(fecha\_inicio)) &&

(Fecha.isEqual(fecha\_fin) || Fecha.isBefore(fecha\_fin))) {

} else {

throw new RuntimeException("La fecha no está en el rango de fechas programado.");

}

}

// registrar el incidente

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void registrarIncidente(int empleado, int programacion, int tipo\_incidente, String fecha, String detalle) {

String sql = "INSERT INTO INCIDENTE(id\_empleado, "

+ "id\_programacion,id\_tipo,fecha\_incidente, "

+ "detalle) VALUES(?,?,?,CONVERT(DATETIME,?,105),?)";

jdbcTemplate.update(sql, empleado, programacion, tipo\_incidente, fecha, detalle);

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void actualizarEstadoCarro(int programacion) {

String sql = "update carro set id\_estado = 3 where id\_carro "

+ "= (select id\_carro from programacion where id\_programacion = ?)";

jdbcTemplate.update(sql, programacion);

}

}

2.2) Prueba en Postman:

{

    "empleado":3,

    "programacion":1,

    "tipo\_incidente":3,

    "fecha":"2024-01-10",

    "detalle":"Fallos en el motor de arranque"

}

Resultado:



1. **Registrar Reparación:**

3.1) Código Fuente:

package pe.edu.uni.proyecto.service;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

import pe.edu.uni.proyecto.dto.ReparacionDto;

@Service

public class ReparacionService {

@Autowired

private JdbcTemplate jdbcTemplate;

@Transactional(propagation = Propagation.REQUIRES\_NEW, rollbackFor = Exception.class)

public ReparacionDto reparacion(ReparacionDto bean) {

// validaciones

validarIncidente(bean.getIdincidente());

validarEstadoAuto(bean.getIdincidente());

validarEmpleado(bean.getIdempleado());

validarTaller(bean.getIdtaller());

bean.setFechareparacion(convertirFecha(bean.getFechareparacion()));

validarIngresoFecha(bean.getIdincidente(), bean.getFechareparacion());

// registro

registrarReparacion(bean.getIdempleado(), bean.getIdincidente(), bean.getIdtaller(), bean.getFechareparacion(),

bean.getCalificacion(), bean.getCosto(), bean.getDetalle());

actualizarEstadoCarro(bean.getIdincidente());

double calificacionfinal = obtenerCalificacionTaller(bean.getIdtaller());

actualizarpromediotaller(bean.getIdtaller(),calificacionfinal);

System.out.println("Proceso ok.");

return bean;

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarTaller(int idtaller) {

String sql = """

select count(1) cont from TALLER where id\_taller = ?

""";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, idtaller);

if (cont == 0) {

throw new RuntimeException("El taller no se encuentra registrado");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarIncidente(int idincidente) {

String sql = """

select count(1) cont from INCIDENTE where id\_incidente = ?

""";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, idincidente);

if (cont == 0) {

throw new RuntimeException("El incidente no se encuentra registrado");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarEmpleado(int idempleado) {

String sql = """

select count(1) cont from EMPLEADO where id\_empleado = ?

""";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, idempleado);

if (cont == 0) {

throw new RuntimeException("El empleado no existe");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void actualizarpromediotaller(int idtaller, double calificacionfinal) {

String sql = """

UPDATE TALLER

SET calificacion = ?

WHERE id\_taller = ?

""";

jdbcTemplate.update(sql, calificacionfinal, idtaller);

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private double obtenerCalificacionTaller(int idtaller ) {

String sql = """

SELECT CAST(AVG(totales.calificacion) AS DECIMAL(10, 1)) AS promedio

FROM (

SELECT t1.calificacion FROM REPARACION t1 WHERE t1.id\_taller = ? UNION ALL

SELECT t2.calificacion FROM MANTENIMIENTO t2 WHERE t2.id\_taller = ?

) AS totales;

""";

double calificacionfinal= jdbcTemplate.queryForObject(sql, double.class, idtaller , idtaller );

return calificacionfinal;

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void actualizarEstadoCarro(int idincidente) {

String sql = """

UPDATE CARRO

SET id\_estado = 1

WHERE id\_carro = (SELECT t2.id\_carro FROM INCIDENTE t1

INNER JOIN PROGRAMACION t2 ON t1.id\_programacion = t2.id\_programacion

WHERE t1.id\_incidente = ? )

""";

jdbcTemplate.update(sql, idincidente);

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void registrarReparacion(int idempleado, int idincidente, int idtaller, String fechareparacion,

double calificacion, double costo, String detalle) {

String sql = """

INSERT INTO REPARACION(id\_empleado, id\_incidente, id\_taller, fecha\_reparacion,calificacion,costo,detalle)

VALUES (?, ?, ?, ?, ?, ?, ?)

""";

jdbcTemplate.update(sql, idempleado, idincidente, idtaller, fechareparacion, calificacion, costo, detalle);

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarIngresoFecha(int idincidente, String fechareparacion) {

String sql = """

SELECT COUNT(1) FROM INCIDENTE t1

INNER JOIN PROGRAMACION t2 ON t1.id\_programacion = t2.id\_programacion

WHERE t1.id\_incidente = ? and ? BETWEEN CONVERT(VARCHAR(10), t1.fecha\_incidente, 120)

AND CONVERT(VARCHAR(10), t2.fecha\_fin\_programada, 120)

""";

int cont = jdbcTemplate.queryForObject(sql, Integer.class, idincidente, fechareparacion);

if (cont == 0) {

throw new RuntimeException(

"La fecha es incorrecta , no se encuentra en el intervalo de fecha correspondiente");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

private void validarEstadoAuto(int idincidente) {

String sql = """

SELECT t3.id\_estado FROM INCIDENTE t1

INNER JOIN PROGRAMACION t2 ON t1.id\_programacion = t2.id\_programacion

INNER JOIN CARRO t3 ON t2.id\_carro = t3.id\_carro

where t1.id\_incidente = ?

""";

int estado = jdbcTemplate.queryForObject(sql, Integer.class, idincidente);

if (estado != 3) {

throw new RuntimeException("El carro no se encuentra en reparacion.");

}

}

@Transactional(propagation = Propagation.MANDATORY, rollbackFor = Exception.class)

public String convertirFecha(String fecha) {

// Definir los formatos: de entrada (dd/MM/yyyy) y de salida (yyyy-MM-dd)

DateTimeFormatter inputFormatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

DateTimeFormatter outputFormatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

// Parsear la fecha de entrada y formatearla al nuevo formato

LocalDate date = LocalDate.parse(fecha, inputFormatter);

return date.format(outputFormatter);

}

}

# CONCLUSIONES

* El diseño de una buena base de datos es fundamental, ya que su estructura permitirá o no realizar correctamente las transacciones y consultas que ayudan a cumplir su propósito de ser una herramienta para la gestión y la buena toma de decisiones.
* Las pruebas son una parte fundamental en la programación, no solo para corroborar que funcione el código, si no también, para encontrar errores en el código, excepciones que no se validaron y datos que no deberían registrarse.
* Gracias a las validaciones incluidas, el sistema garantiza la integridad de los datos al verificar la existencia de las entidades relacionadas, como empleados, talleres y vehículos, antes de registrar un mantenimiento. Además, la actualización automática del estado de los vehículos y el cálculo dinámico de las calificaciones de los talleres refuerzan la fiabilidad del sistema, asegurando que la información refleje las operaciones en tiempo real.
* El sistema del proyecto desarrollado gracias a un buen trabajo en equipo tiene una base sólida para que en un futuro pueda incorporarse nuevas funcionalidades, como la optimización de rutas o generación de reportes avanzados.

# LECCIONES APRENDIDAS

* A lo largo de este proyecto aprendí a usar mejor los repositorios git, el cual resultaba en un completo misterio a comienzos del ciclo, resulto en un muy buen apoyo para mejorar la comunicación con mis compañeros, además de ayudar en mantener organizado los diferentes entregables de este proyecto.
* Debe tomarse el tiempo necesario para definir claramente los requisitos al principio del proyecto. Los detalles, como las funcionalidades exactas que se esperan de la aplicación o los casos de uso, deben estar bien establecidos para un correcto desarrollo y ejecución del sistema.
* Cumplir con los estándares REST, como usar códigos HTTP adecuados (201 Created para éxito y 400 Bad Request para errores), hace que la API sea más predecible y fácil de integrar con otros sistemas.
* Aprendimos a interactuar y entender la utilidad tanto de SQL Server para poder crear nuestra base de datos como de Erwin Data Modeler para poder tener una mejor visualización de las interacciones entre las tablas creadas de nuestra base de datos
* Aprendimos la importancia de actualizar dinámicamente los estados del sistema, como el cambio del estado del carro a "en mantenimiento" y la recalificación del taller después de cada registro. Este ejercicio nos enseñó a integrar lógica empresarial en nuestras bases de datos de manera eficiente, utilizando triggers y procedimientos almacenados.

# BIBLIOGRAFIA

La bibliografía que has consultado es muy importante, debes utilizar el formato APA.

Mínimo debes tener 8 referencias bibliográficas, deben ser artículos científicos y/o libros.

**A continuación, tienes un ejemplo:**

Ohmae, K. (2004). *La mente del estratega.* McGraw-Hill Interamericana.

Ruiz, L., & Ruiz, Y. (2019). *Enseñar hoy una lengua extranjera.* Octaedro. Retrieved from https://octaedro.com/wp-content/uploads/2020/12/30809-Ensenar-hoy-una-lengua-extranjera.pdf

Sosa Mayanga, M. L. (2022). *Entornos virtuales y aprendizaje del idioma inglés de estudiantes de cuarto grado de Secundaria, de una institución educativa de Chiclayo.* UNIVERSIDAD CESAR VALLEJO, CHICLAYO-PERU. Retrieved from https://repositorio.ucv.edu.pe/handle/20.500.12692/96668

Yankovskaya, V. V., Mustafin, T. A., Endovitsky, D. A., & Krivosheev, A. V. (2022). Corporate Social Responsibility as an Alternative Approach to Financial Risk Management: Advantages for Sustainable Development. *Risks, 10*(5). doi:https://doi.org/10.3390/risks10050106

# ANEXOS

## Anexo 1: Diagramas del Sistema

Este anexo debe incluir los diagramas principales del sistema, como el modelo de base de datos y los diagramas de flujo de los servicios implementados.

Diagrama Entidad-Relación (ER): Representa las entidades clave del sistema (Empleado, Carro, Taller, Programación, Mantenimiento, entre otros) y sus relaciones. Es fundamental para entender la estructura de datos y cómo se relacionan en el contexto de transporte.

(Incluir el diagrama ER elaborado en Erwin o cualquier herramienta utilizada, exportado como imagen.)

Diagrama de Flujo de Operaciones: Muestra el flujo de procesos, como la programación de viajes, el registro de mantenimientos e incidentes, y cómo las acciones afectan a las bases de datos o los estados de los vehículos.

(Adjuntar un diagrama de flujo claro y ordenado que explique el proceso paso a paso.)

Diagramas UML: Si fueron desarrollados, incluir diagramas de caso de uso, de secuencia y de clases para ilustrar cómo interactúan las entidades y los actores del sistema.

## Anexo 2: Código Fuente del Servicio

Incluye los fragmentos de código más importantes que muestran la implementación de los casos de uso principales del proyecto. Este anexo permite validar las funcionalidades desarrolladas y sirve como referencia técnica.

Servicios REST Implementados:

Código del Servicio de Registro de Programación.

Código del Servicio de Registro de Mantenimiento.

Código del Servicio de Registro de Incidente.

Código del Servicio de Registro de Reparación.

(Agregar fragmentos de código relevantes que muestren las validaciones, lógica de negocio y manejo de excepciones, como los incluidos en tu documento.)

Scripts SQL:

Scripts para la creación de tablas en la base de datos, asegurando relaciones y restricciones de integridad.

Scripts para la carga de datos de prueba.

**Pruebas Unitarias e Integración:** Incluir aquí los Postman