

SEBASTIÁN ARIAS

PROYECTO final

INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

ING. KENNETH JIMÉNEZ PLÚAS, MGTR

DISEÑO DE SOFTWARE

15 de mayo 2024

# ENUNCIADO

**Para el sistema definido por grupo, elaborar lo siguiente:**

1. Diagrama de casos de uso (15 ptos)

2. Diagrama de clases:

a. Aplicar por lo menos 2 patrones de diseño, indicar cuales y justificar los seleccionados. (10 ptos)

b. Clases (20 pts)

c. Cardinalidad (5 ptos)

d. Relaciones (10 pts)

3. Diagramas de secuencia de los casos de uso (20 pts)

4. Diagrama de diseño modular con alta cohesión y bajo acoplamiento. (10 ptos)

5. Diagrama de la arquitectura. (10 ptos)

**Se solicita:**

1. Crear carpeta con nombre “doc/UML” dentro del proyecto java

2. Almacenar los diagramas (.xml) en la carpeta “doc/UML”

3. Crear una nueva versión con git.

4. Crear nuevo repositorio con nombre “Proyecto final” en Gitlab

5. Publicar la nueva versión en Gitlab sobre la rama main.

Entregable

URL de gitlab en Blackboard

# DESARROLLO

Los diagramas ayudaran a visualizar las funciones principales que cada tipo de usuario dentro del sistema.

Para el Administrador:

Autenticarse en el sistema.

Ingresar repuestos nuevos.

Registrar mecánicos.

Recibir solicitudes de repuestos.

Asignar repuestos a mecánicos.

Actualizar estado de los repuestos.

Para el Mecánico:

Autenticarse en el sistema.

Registrar el camión a cargo.

Consultar repuestos disponibles.

Enviar formulario de solicitud de repuesto.

Consultar historial y estado de repuestos.

Para el Usuario de Lectura:

Autenticarse en el sistema.

Ver lista de todos los pedidos y sus estados.

## Diagrama de casos de uso.

Para crear un diagrama de casos de uso en diagrams.net que represente el Sistema de Gestión de Repuestos para Camiones (SGRC), se utilizará dos tipos de formas principalmente: "Actors" (Actores) y "Use Cases" (Casos de Uso), representados por óvalos.



## Diagrama de Clase

**Aplicar 2 patrones de diseño:**

Para el Sistema de Gestión de Repuestos para Camiones (SGRC), se puede implementar dos patrones de diseño útiles y complementarios: Singleton y Observer. La elección de estos patrones está fundamentada en las necesidades específicas del sistema y en cómo cada patrón puede optimizar el diseño y funcionamiento del mismo.

**Singleton:** Este patrón es ideal para gestionar la conexión a la base de datos en el sistema SGRC. Dado que es común y eficiente mantener una única instancia de conexión a la base de datos a lo largo de toda la aplicación para evitar sobrecargas de conexiones y garantizar un manejo coherente de los datos, el patrón Singleton permite asegurar que sólo exista una instancia de la clase de conexión, evitando conflictos y problemas de rendimiento.

**Observer:** Este patrón se utilizará para gestionar las actualizaciones en el inventario de repuestos. El sistema necesita actualizar a varios componentes (como interfaces de usuario y reportes de estado) cada vez que hay un cambio en el inventario de repuestos (por ejemplo, cuando se añaden nuevos repuestos, o cuando cambian los estados de los mismos). Utilizando el patrón Observer, la clase Inventario puede notificar automáticamente a todos los observadores registrados sobre cualquier cambio, facilitando la sincronización y actualización en tiempo real sin intervención manual.

**Clases:**

* DatabaseConnection (Singleton)
* Usuario (Clase base abstracta)
* Administrador
* Mecanico
* Inventario (Observer Subject)
* InventoryObserver (Interface)
* Repuesto
* SolicitudRepuesto
* Camion

**Relaciones y cardinalidad:**

La Clase Administrador:

Mecanico: 1 a muchos (Un administrador puede registrar muchos mecánicos)

SolicitudRepuesto: 1 a muchos (Un administrador puede gestionar muchas solicitudes de repuestos)

La ClaseMecanico

Camion: 1 a muchos (Un mecánico puede registrar varios camiones)

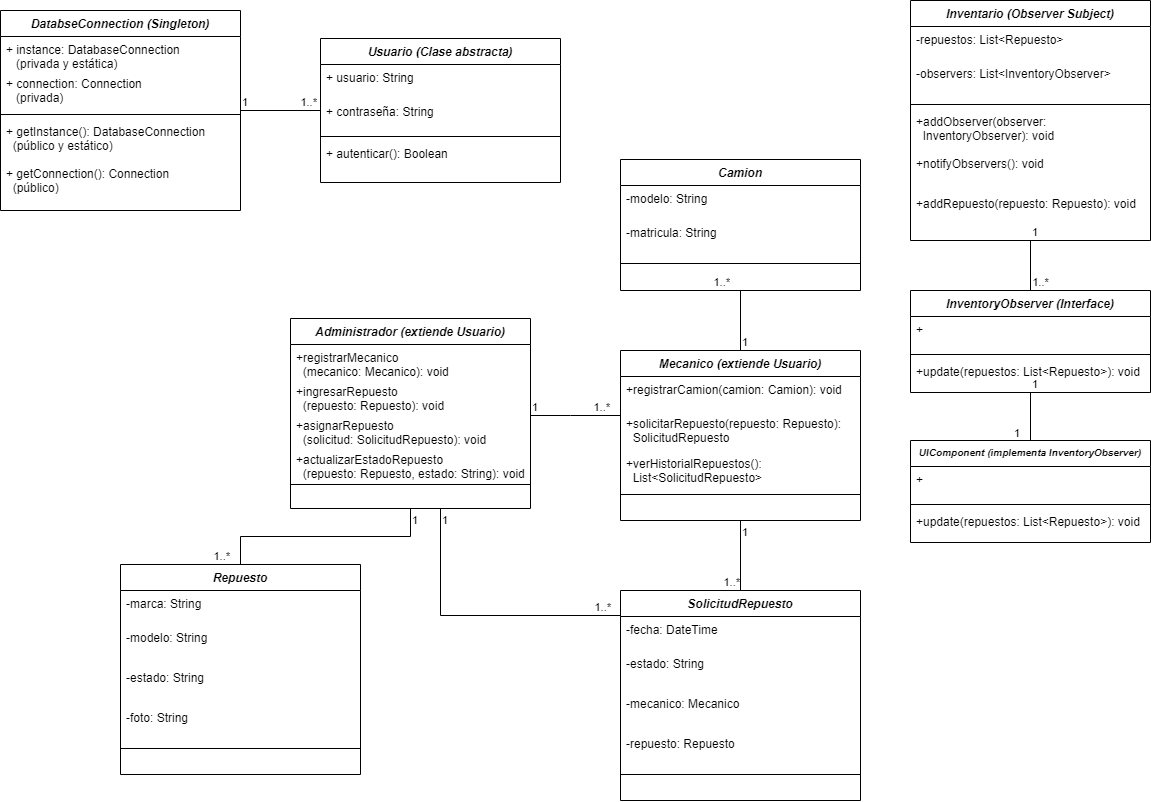
SolicitudRepuesto: 1 a muchos (Un mecánico puede hacer varias solicitudes de repuestos)

La Clase Inventario - InventoryObserver

Cardinalidad: 1 a muchos (El inventario puede notificar a muchos observadores)

La clase UIComponent implementa InventoryObserver

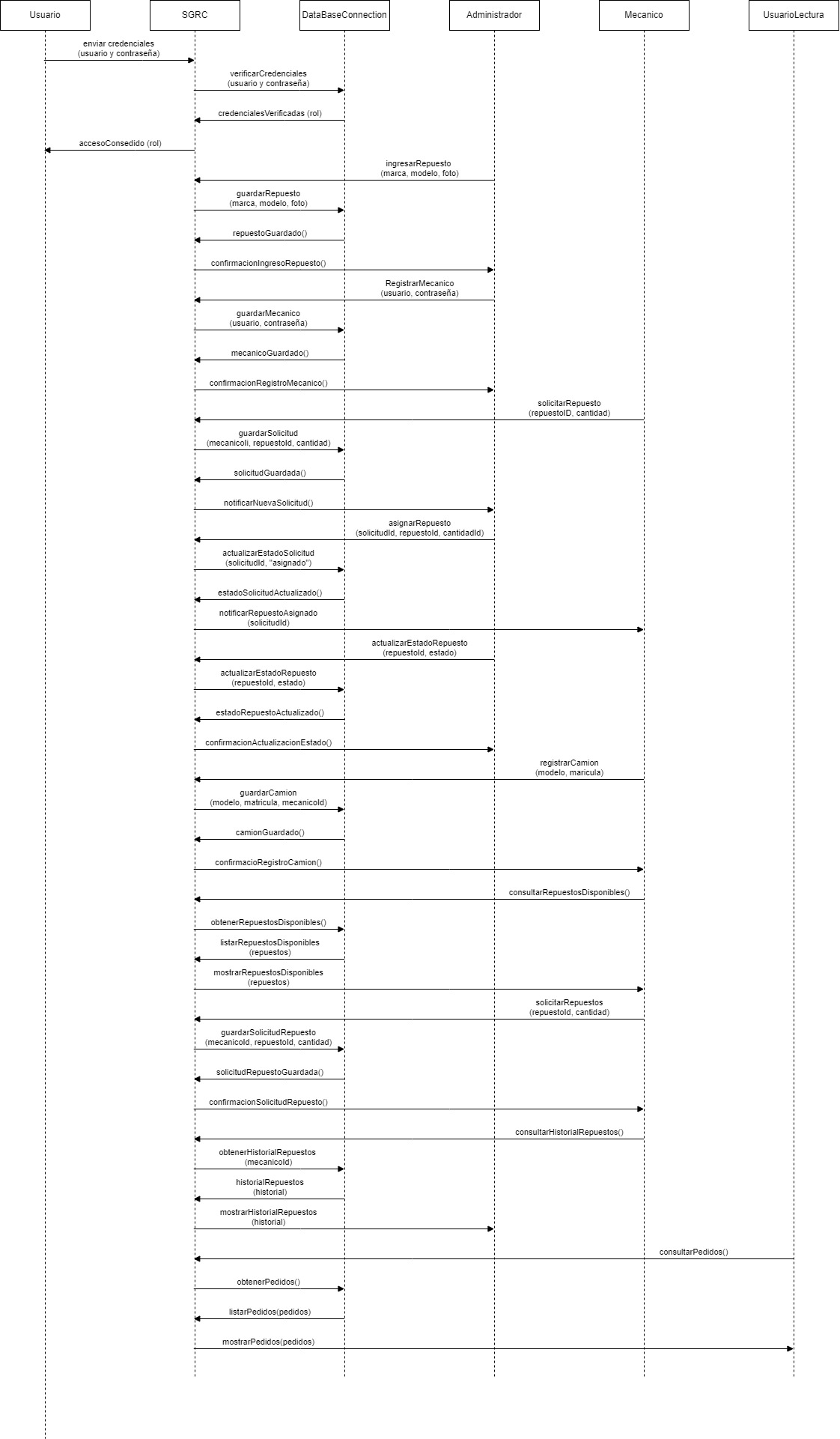
Cardinalidad: 1 a 1 (Cada componente de UI implementa un observador)

****

## Diagramas de secuencia de los casos de uso.

Este diagrama de secuencia para un sistema como el SGRC (Sistema de Gestión de Repuestos para Camiones), sirve para ver los elementos clave y las interacciones importantes.

Estos elementos y flujos muestran cómo los diferentes roles de usuario interactúan con el sistema SGRC y con la base de datos a través de varias operaciones clave. Cada interacción representa un caso de uso crítico para el sistema, destacando la funcionalidad principal y cómo se gestiona la información en cada paso del proceso.



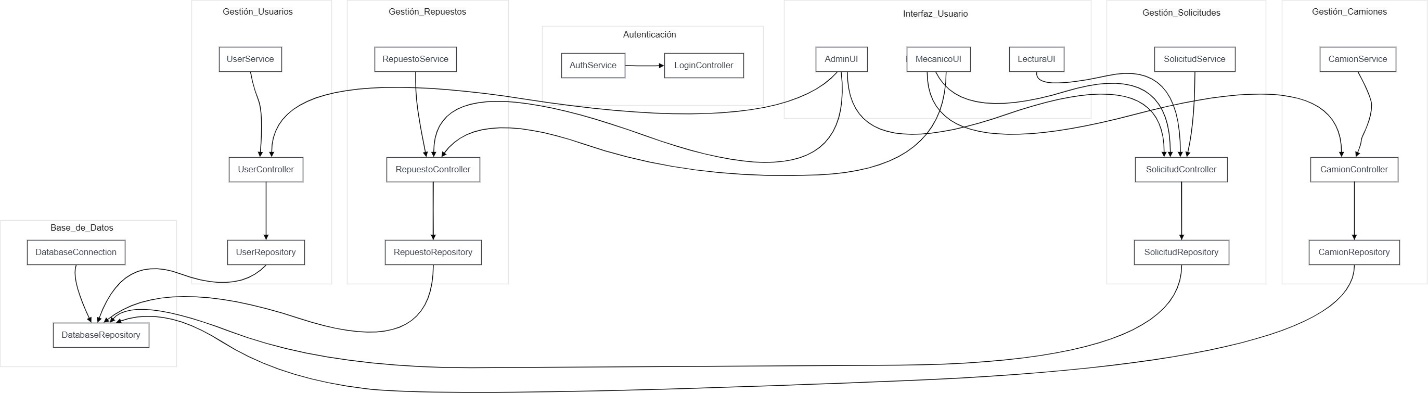
## Diagrama de diseño modular con alta cohesión y bajo acoplamiento.

Para crear un diagrama de diseño modular con alta cohesión y bajo acoplamiento para el Sistema de Gestión de Repuestos para Camiones (SGRC), es esencial dividir el sistema en módulos que agrupan funcionalidad relacionada y minimizar las dependencias entre ellos.

Relaciones Clave:

* Los componentes dentro de cada módulo se comunican entre sí para mantener la alta cohesión.
* Los controladores de cada módulo se comunican con sus respectivos servicios.
* Los servicios se comunican con los repositorios para las operaciones de datos.
* La interfaz de usuario (UI) se comunica con los controladores.
* Los repositorios interactúan con DatabaseRepository para realizar operaciones de base de datos.
* DatabaseConnection gestiona la conexión a la base de datos y es utilizado por DatabaseRepository.

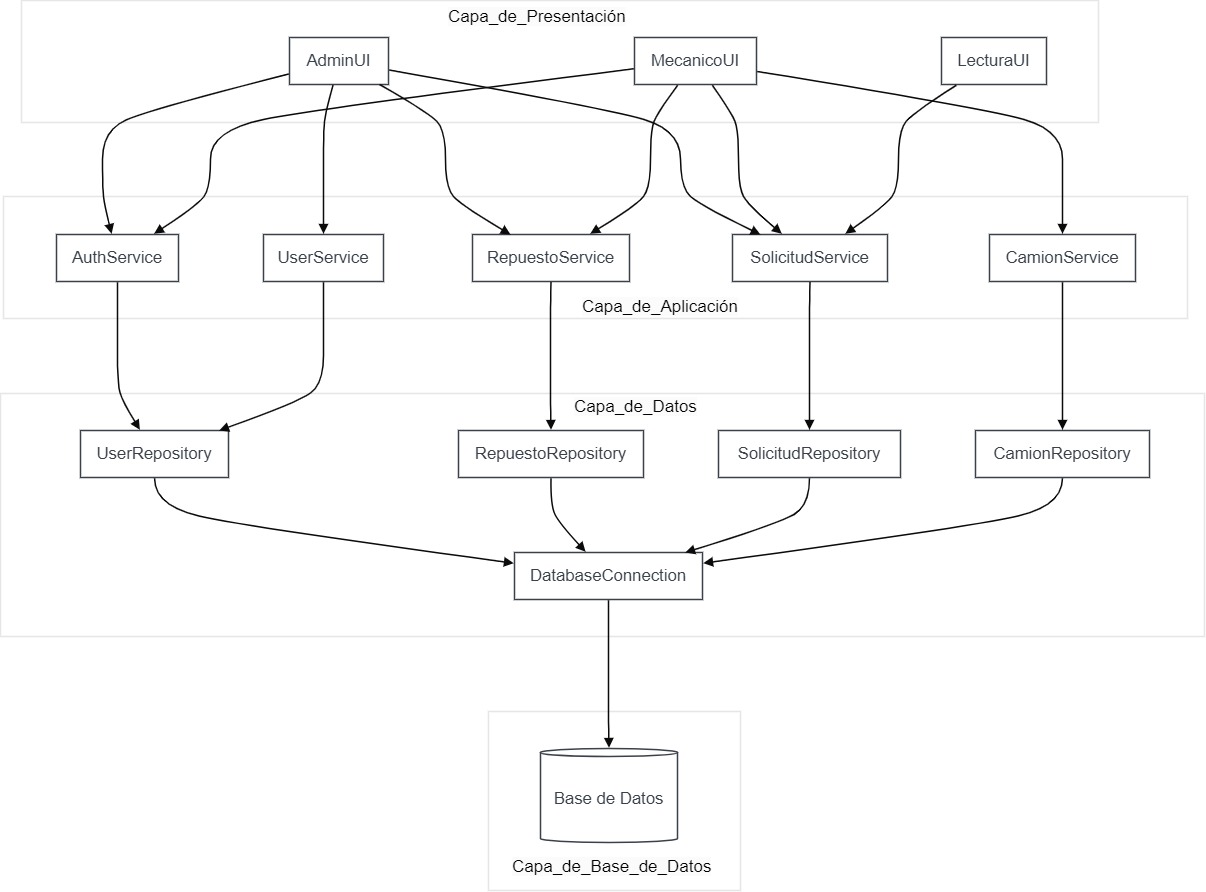
Este diseño asegura que cada módulo tenga una responsabilidad clara y que las interacciones entre módulos sean mínimas, manteniendo el sistema con alta cohesión y bajo acoplamiento.



## Diagrama de la arquitectura.

Para crear un diagrama de arquitectura para el Sistema de Gestión de Repuestos para Camiones (SGRC), se puede usar un enfoque basado en capas, que ayuda a organizar y gestionar las responsabilidades y dependencias del sistema de manera clara.

Este diseño asegura que cada capa y componente tiene una responsabilidad clara y que las interacciones entre las capas se mantienen bien definidas, promoviendo así una arquitectura con alta cohesión y bajo acoplamiento.



# CONCLUSIÓN

A lo largo del desarrollo del Sistema de Gestión de Repuestos para Camiones (SGRC), se han realizado diversos diagramas y se han aplicado patrones de diseño para garantizar una arquitectura sólida y bien estructurada.

**Diagrama de Casos de Uso:**

Se realizó el diagrama de casos de uso para identificar y documentar todas las interacciones posibles entre los usuarios del sistema (Administrador, Mecánicos y Usuarios de Lectura) y el sistema SGRC. Este diagrama ayuda a entender los requisitos funcionales y proporciona una base clara para el diseño del sistema.

**Diagrama de Clases:**

Se diseñó el diagrama de clases, aplicando dos patrones de diseño: Singleton y Observer. Se establecieron las relaciones y la cardinalidad entre las clases para reflejar adecuadamente la estructura y las interacciones del sistema, asegurando un diseño cohesivo y correctamente interrelacionado.

**Diagrama de Secuencia:**

Se realizó el diagrama de secuencia para los principales casos de uso del sistema, incluyendo la autenticación, la gestión de repuestos, la gestión de usuarios, y las solicitudes de repuestos. Estos diagramas ilustran el flujo de mensajes entre los diferentes componentes y usuarios del sistema, proporcionando una visión clara de la dinámica y las interacciones en tiempo de ejecución.

**Diagrama de Diseño Modular:**

Se diseñó un diagrama de diseño modular con alta cohesión y bajo acoplamiento. Este diagrama organiza el sistema en módulos funcionales (Autenticación, Gestión de Usuarios, Gestión de Repuestos, Gestión de Solicitudes, Gestión de Camiones, Interfaz de Usuario y Base de Datos), cada uno con responsabilidades claras y bien definidas. Este enfoque modular facilita el mantenimiento, la escalabilidad y la reutilización de componentes.

**Diagrama de Arquitectura:**

Se realizó el diagrama de arquitectura del sistema, distribuyendo los componentes en capas (Presentación, Aplicación, Datos y Base de Datos). Este diagrama proporciona una visión global de la estructura del sistema, destacando las relaciones y dependencias entre las capas y los componentes. La separación en capas mejora la organización y facilita la gestión de la complejidad del sistema.

Implementación y Publicación

**Creación de la Carpeta de Documentación:**

Se crear una carpeta con nombre “doc/UML” dentro del proyecto java.

Se almacena los diagramas (.xml) en la carpeta “doc/UML”

Se crear una nueva versión con git.

Se crear un nuevo repositorio con nombre “Proyecto final” en Gitlab.

Se publica la nueva versión en Gitlab sobre la rama main.

**Entregable URL de gitlab:**

<https://github.com/LinkinCypher/java>