



# NEXORA

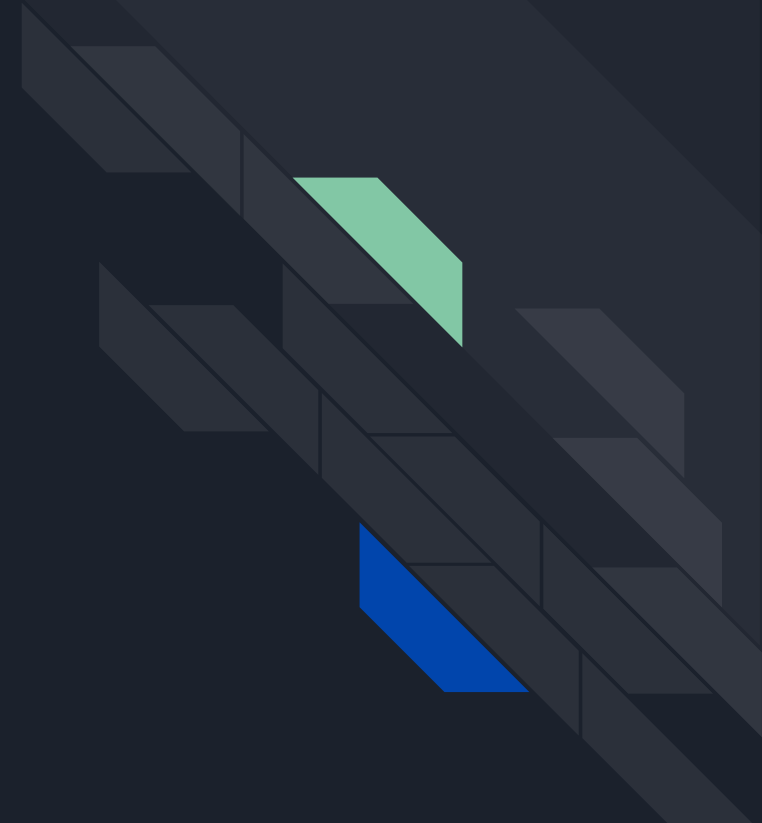
## Integrantes:

Longhino, Emmanuel  
Puentes, Luciana  
Cohen, Ari  
Massacesi, Juan Ignacio

REPOSITORIO

# Índice

1. Casos de uso
2. Diagramas de Casos de Uso
3. Escenarios de Casos de Uso
4. Diagramas de Secuencia
5. Prototipo Interfaz Gráfica
6. Diagrama de Clases de Diseño
7. Diagramas de Paquete de Diseño
8. Requisitos no Funcionales
9. Patrones de Diseño
10. Propuesta de Mejora
11. Demo
12. Fin





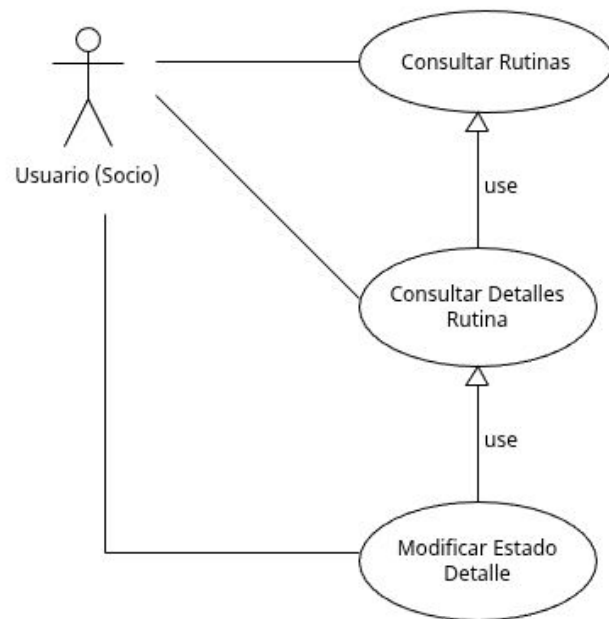
## CASOS DE USO

- Completar Detalle Rutinas.
- Mostrar Rutinas incumplidas y mostrar porcentaje de progreso de Rutina.

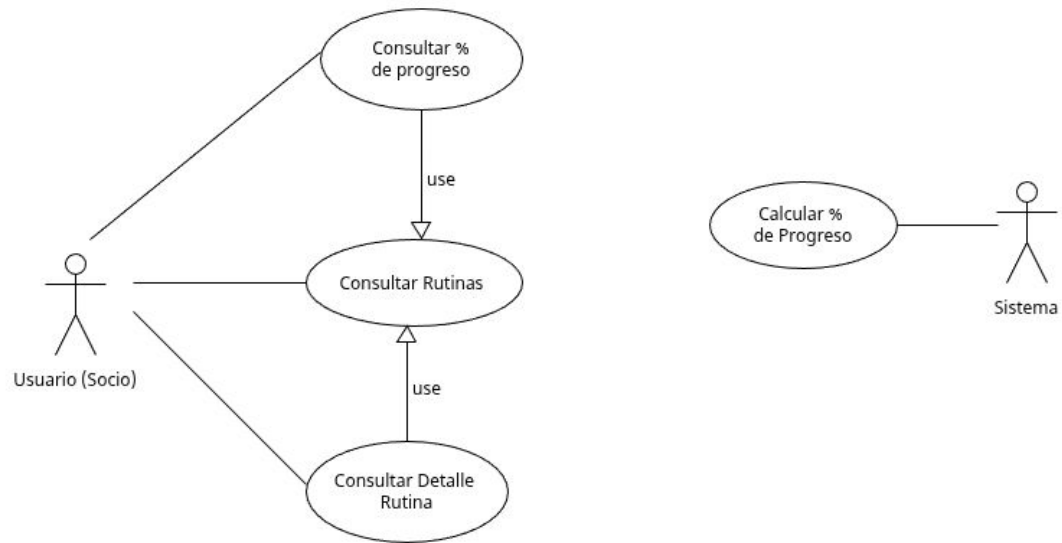
# Diagramas de Casos de Uso



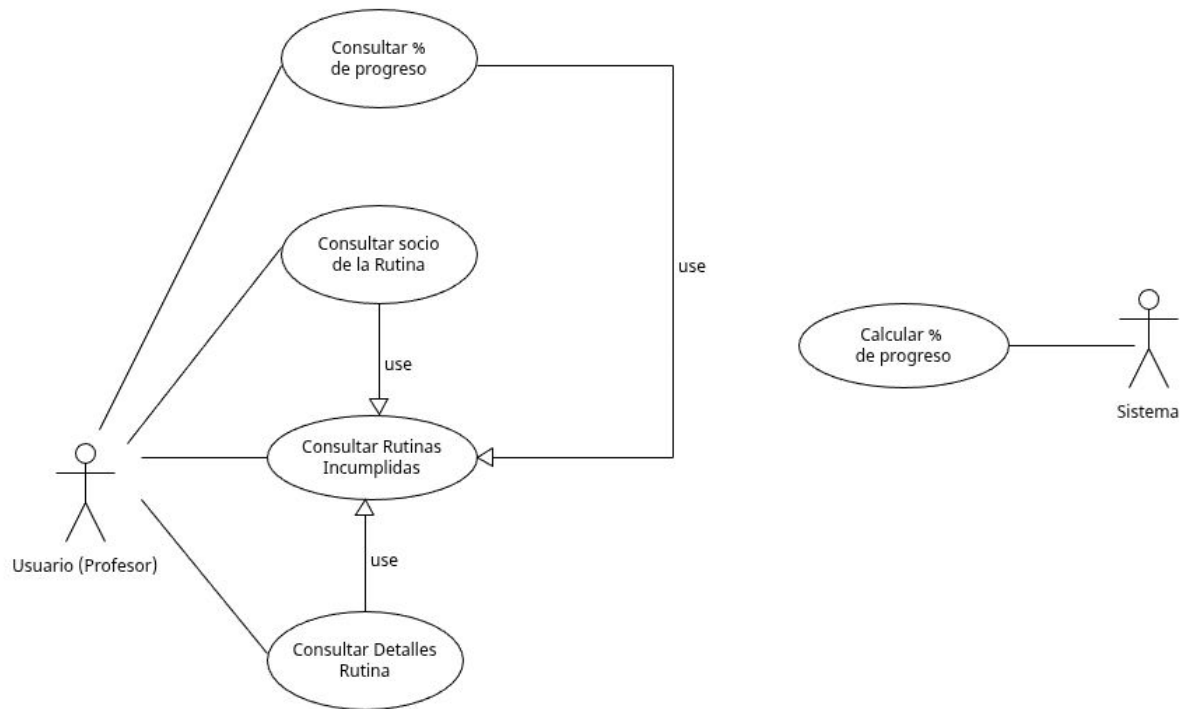
## CASO DE USO: Completar Detalle Rutinas Realizadas



CASO DE USO: Mostrar Rutinas Incumplidas y porcentaje cumplido de Rutina



CASO DE USO: Mostrar Rutinas Incumplidas y porcentaje cumplido de Rutina



# Escenarios de Casos de Uso





# Completar Detalle Rutinas

Caso de Uso	Completar Detalle Rutinas Realizadas		ID: 1	
Prioridad	Alta	Estimación	-	ID Pantalla Prototipo: -
Precondición	El usuario debe estar logueado en el sistema.			
Descripción	El usuario (asociado/a) podrá completar sus rutinas marcando como finalizada cada actividad que la rutina tiene.			
Escenario Principal	Paso	Acción		
	1	El sistema busca las rutinas asociadas al usuario		
	2	El sistema muestra al usuario sus rutinas con sus actividades		
	3	El usuario selecciona un botón para ver las actividades de su rutina		
	4	El sistema muestra las actividades de la rutina seleccionada por el usuario		
	5	El usuario modifica el estado de la actividad		
	6	El usuario vuelve a ver sus rutinas		
	7	El sistema repite 1		
Poscondición	El sistema muestra el listado de rutinas del usuario			
Excepciones	1.1	El sistema muestra el mensaje "No tiene rutinas registradas"		
	2.1	El sistema muestra el mensaje "La rutina no tiene actividades asociadas"		
Comentarios				Dependencias

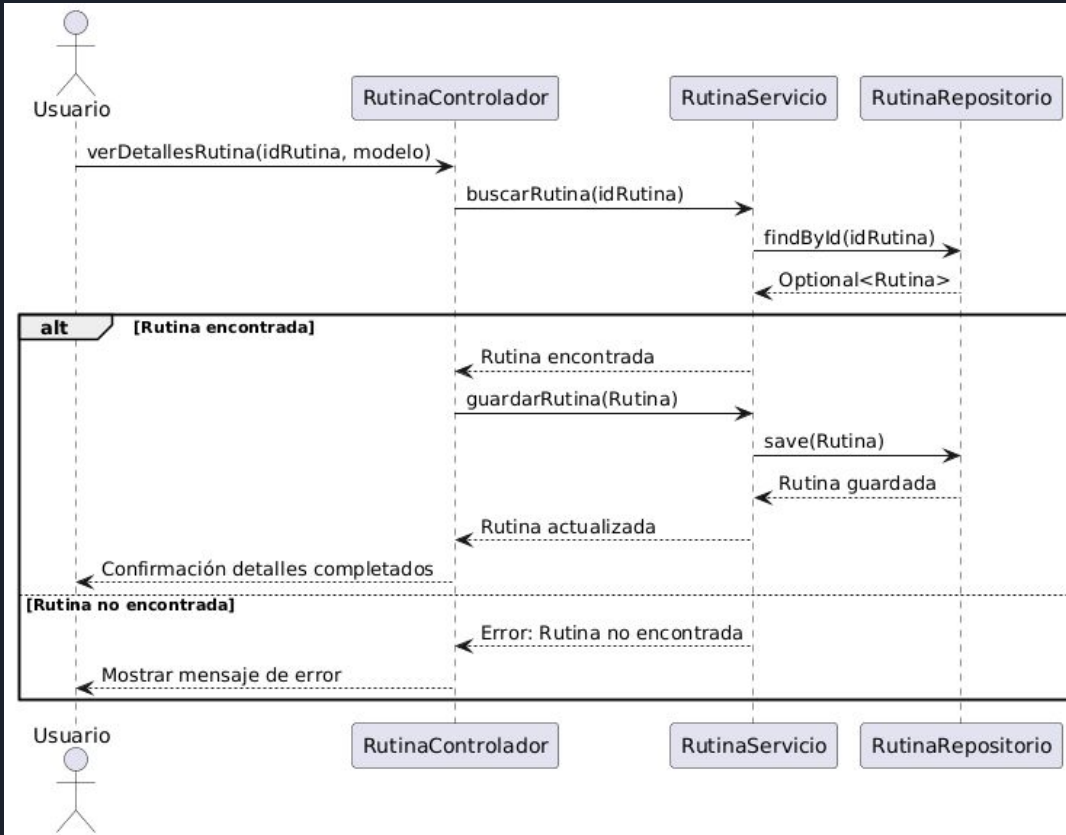
# Mostrar Rutinas incumplidas y mostrar porcentaje de progreso de Rutina

Caso de Uso	Mostrar Rutinas incumplidas con porcentaje de progreso		ID: 2	
Prioridad	Alta	Estimación	-	ID Pantalla Prototipo: -
Precondición	El usuario debe estar dado de alta y logueado en el sistema.			
Descripción	El sistema debe mostrar al usuario (profesor) las rutinas incumplidas de sus asesorados y el porcentaje de progreso.			
Escenario Principal	Paso	Acción		
	1	El usuario consulta sus rutinas		
	2	El sistema busca las rutinas hechas por el usuario		
	3	El sistema muestra al usuario sus rutinas con sus actividades, progreso actual y asociado/alumno		
	4	El usuario selecciona el filtro de En_Proceso para las rutinas		
	5	El sistema busca las rutinas con estado En_Proceso		
	6	Se repite el paso 2		
Poscondición	El sistema muestra las rutinas del usuario.			
Excepciones	1.1	El sistema muestra el mensaje "No tiene rutinas registradas"		
	2.1	El sistema muestra el mensaje "La rutina no tiene actividades asociadas"		
	3.1	El sistema muestra el mensaje "No tiene rutinas incompletas/En proceso"		
Comentarios				Dependencias

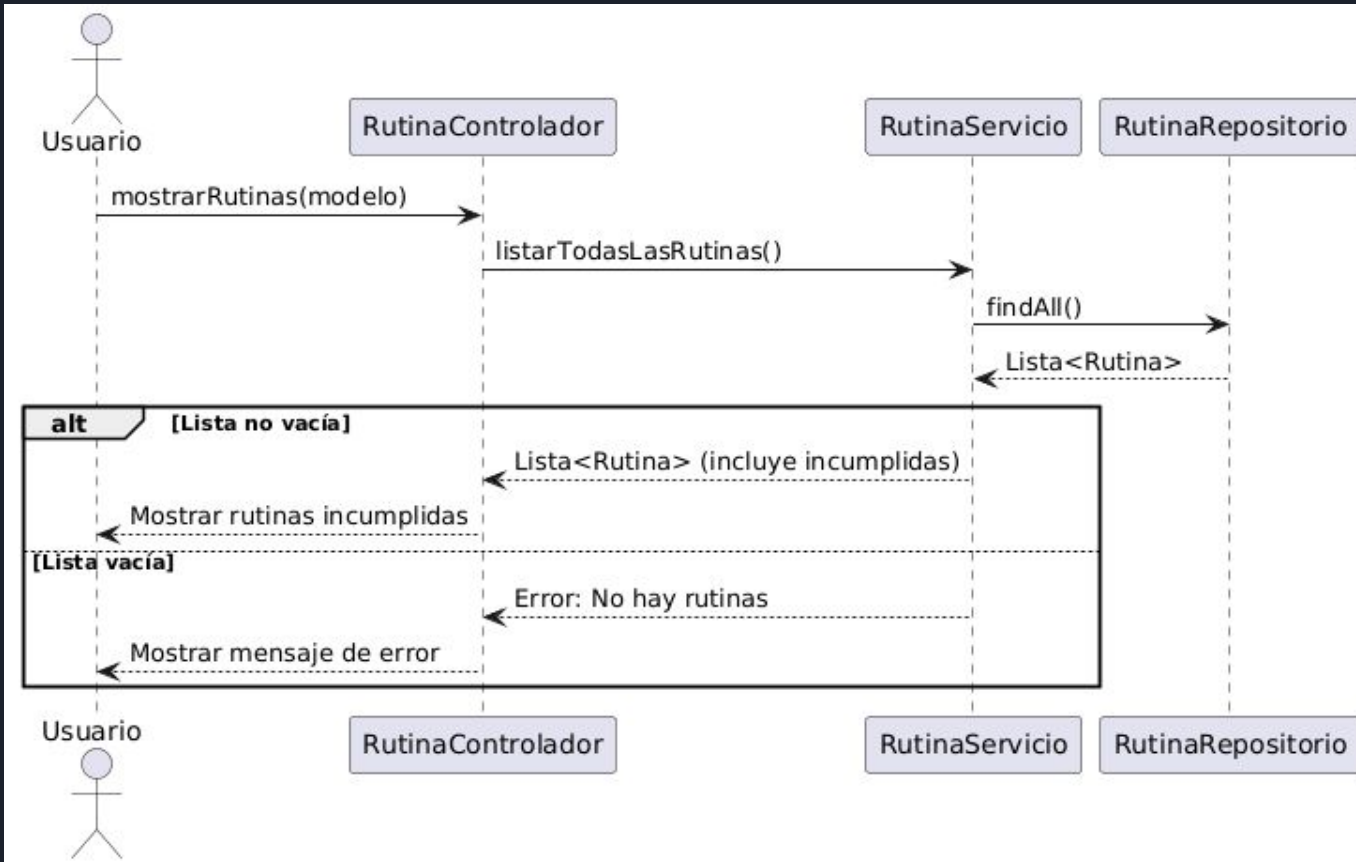
# Diagramas de Secuencia



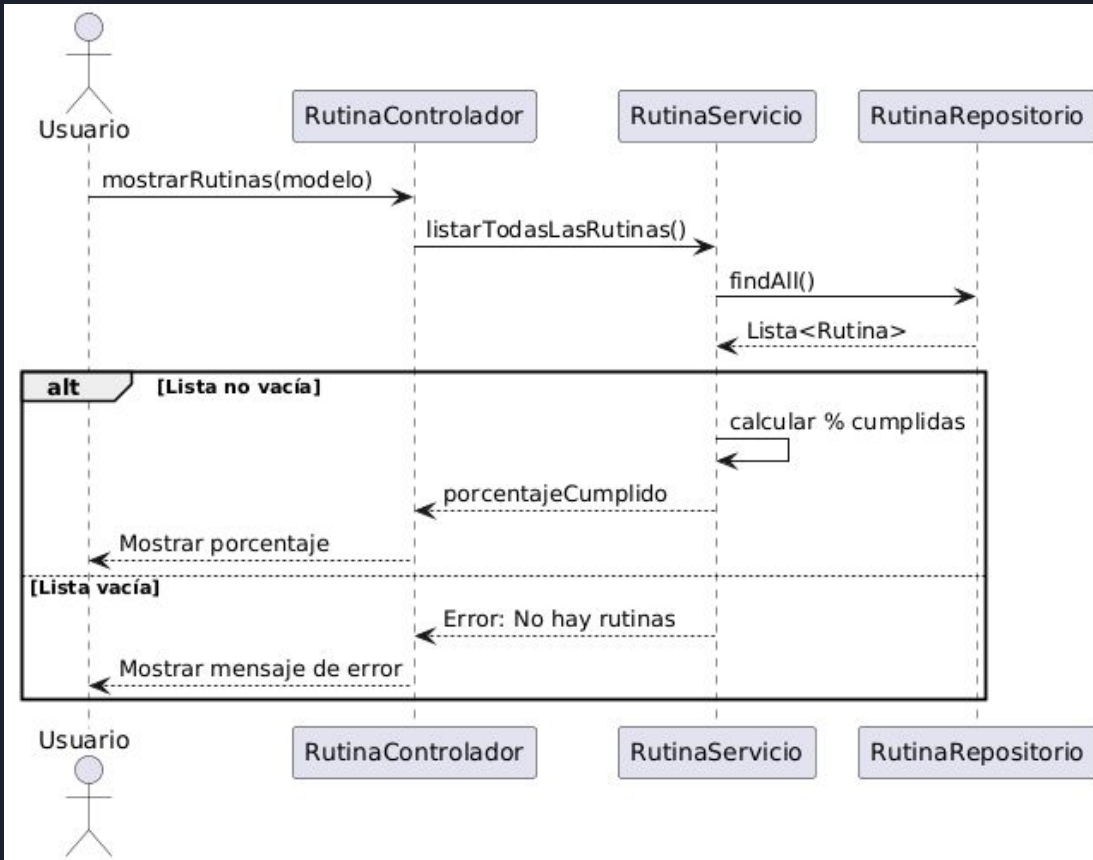
# Diagramas de secuencia



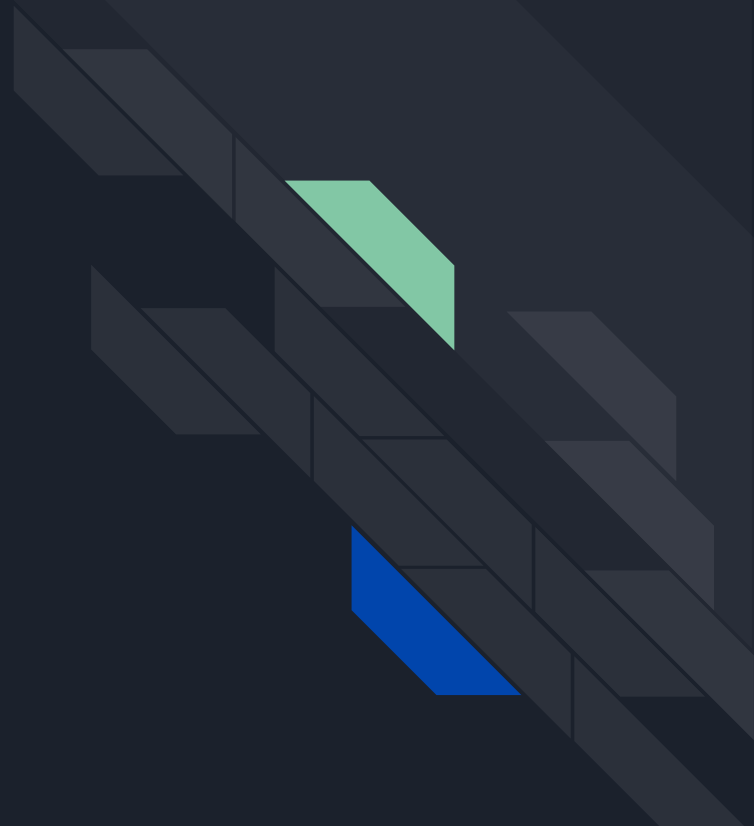
# Diagramas de secuencia



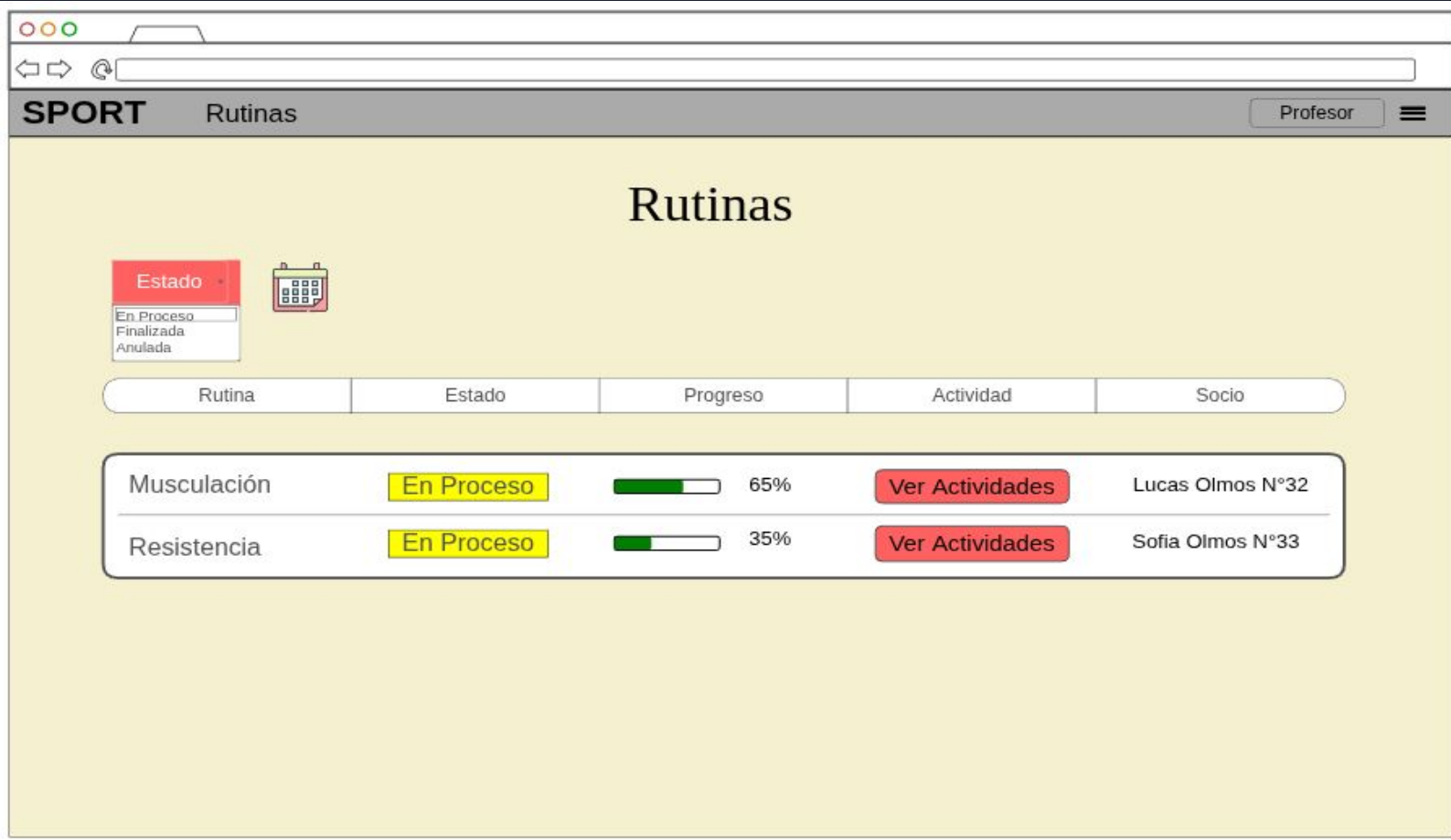
# Diagramas de secuencia



# Prototipo Interfaz Gráfica



# Mostrar Rutinas incumplidas y mostrar porcentaje de progreso de Rutina



The screenshot shows a web application interface for 'SPORT Rutinas'. The header includes the 'SPORT' logo, the title 'Rutinas', a user role 'Profesor', and a menu icon. The main content area is titled 'Rutinas'. On the left, there is a filter for 'Estado' (Status) with options: 'En Proceso' (selected), 'Finalizada', and 'Anulada'. Below the filter is a calendar icon. A table displays the routine data with columns: Rutina, Estado, Progreso, Actividad, and Socio. Two routines are listed: 'Musculación' and 'Resistencia', both with a status of 'En Proceso'. Each routine has a progress bar and a percentage value. A 'Ver Actividades' button is present for each routine.

Rutina	Estado	Progreso	Actividad	Socio
Musculación	En Proceso	<div><div></div></div> 65%	<a href="#">Ver Actividades</a>	Lucas Olmos N°32
Resistencia	En Proceso	<div><div></div></div> 35%	<a href="#">Ver Actividades</a>	Sofia Olmos N°33



## Completar Detalle Rutinas

SPORT Rutinas

Socio

### Actividades



←

15/09/2025

21/09/2025

→

LUN. - 15

MAR. - 16

MIÉ. - 17

JUE. - 18

VIE. - 19

SÁB. - 20

DOM. - 21

Brazo

Estado

Realizada

Sin Realizar

Pecho

Estado

Pierna

Estado

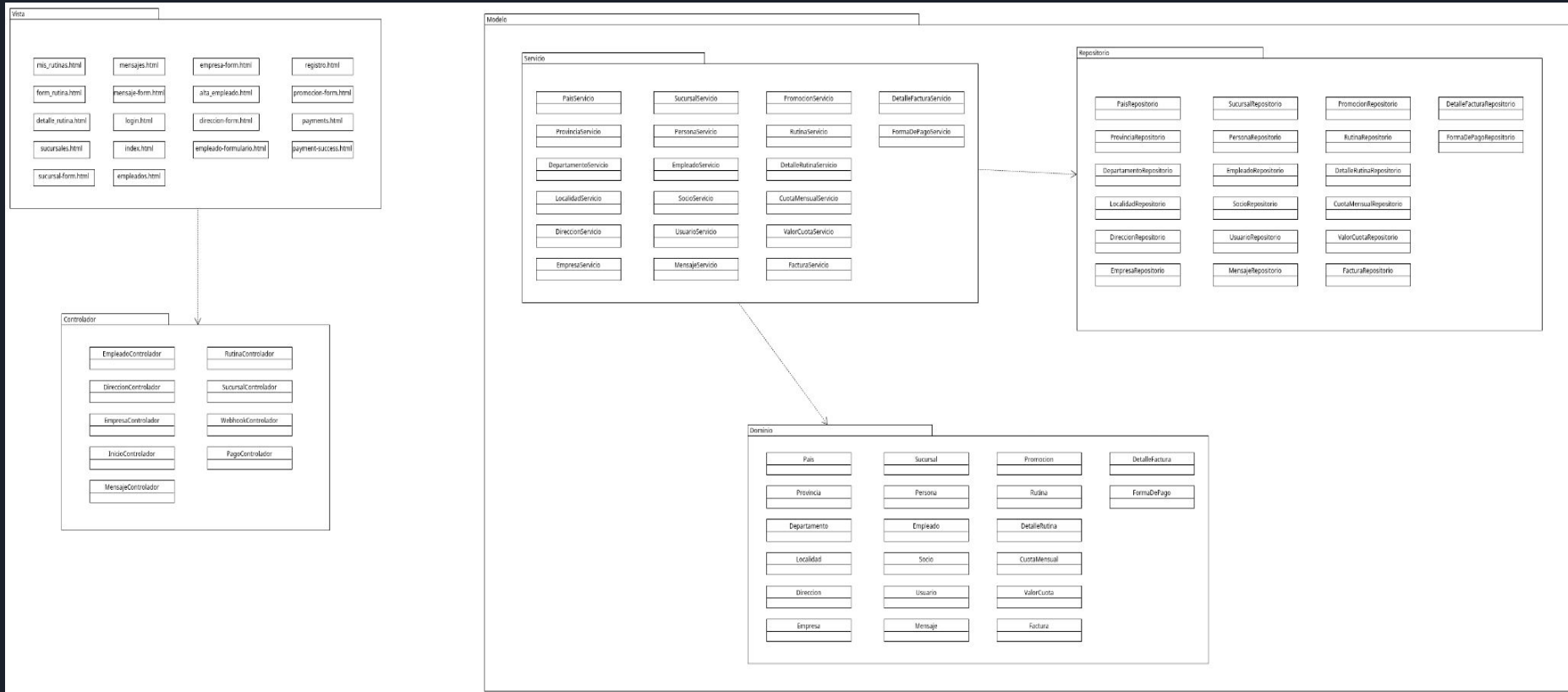
# Diagrama de Clases de Diseño





# Diagrama de Paquete de Diseño





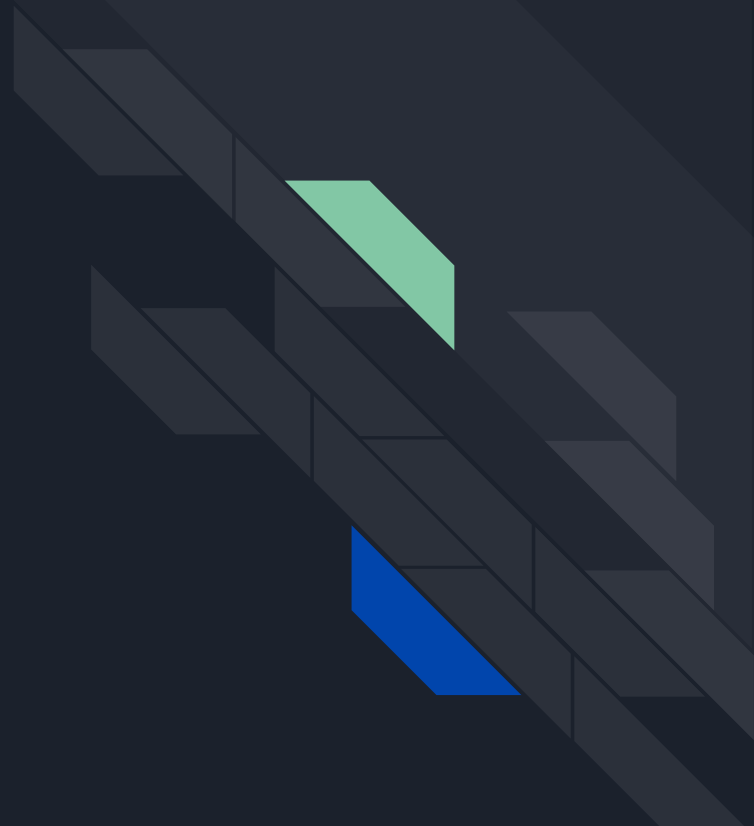
Link: [Diagrama de Paquete](#)

# Requisitos no Funcionales

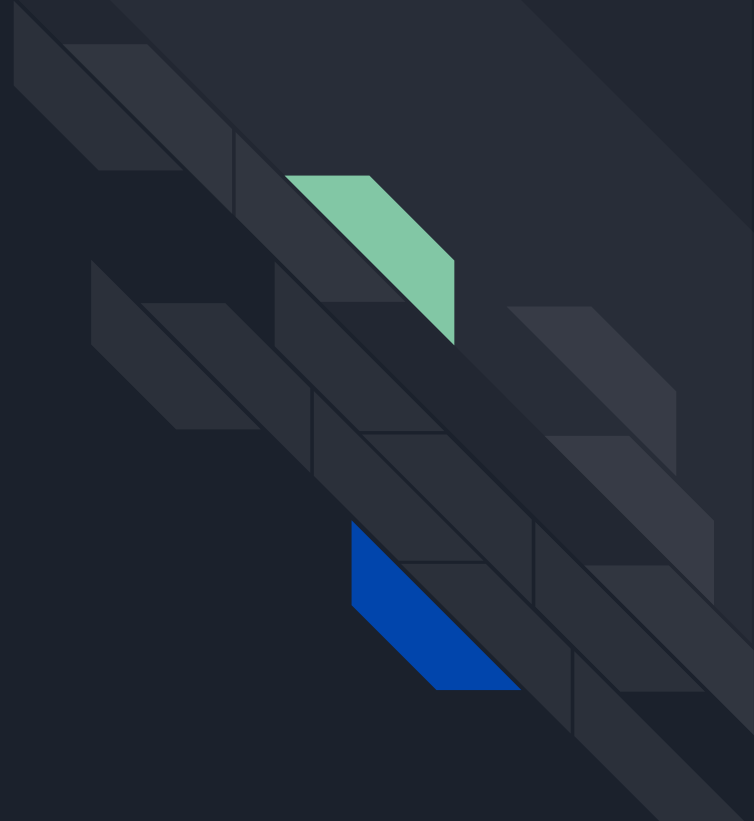


# Requisitos no Funcionales

- ❖ **Facilidad de Uso:** Interfaz intuitiva para los usuarios del sistema, navegación consistente, ayudas (tooltips), los usuarios contarán con tutoriales para facilitar el uso del sistema, mensajes de error amigables.
- ❖ **Fiabilidad:** El sistema debe funcionar correctamente bajo condiciones normales y recuperarse de forma controlada ante fallos.
- ❖ **Rendimiento:** Respuestas rápidas para vista fluida y capacidad para picos de carga previstos sin degradación crítica. El sistema debe hacer un buen uso de los recursos.
- ❖ **Soporte:** Debe ser escalable a futuro, con un mantenimiento sencillo y tener documentación completa para facilitar resolución rápida de incidentes.
- ❖ **Seguridad:** Se deben resguardar los datos personales y financieros del usuario. Solo deben ingresar a la aplicación usuarios registrados.
- ❖ **Disponibilidad:** El sistema debe estar funcionando todo el día para los usuarios del sistema. Puede que por mantenimiento el sistema no permita el ingreso a los socios y profesores en horario de madrugada.



# Patrones de Diseño





# Patrones Aplicados

## MVC (Modelo/Vista/Controlador)

MVC separa la presentación en tres piezas: el Modelo (datos y lógica), la Vista (representación visual) y el Controlador (gestión de eventos y coordinación). Esta separación permite que la UI cambie sin tocar la lógica de negocio y que la lógica se pruebe independientemente, además de facilitar la reutilización y organización en aplicaciones web.



# Patrones Aplicados

## Inyección de Dependencia

Es una técnica para proporcionar a las clases sus dependencias desde el exterior en lugar de que las creen internamente. Esto reduce el acoplamiento y mejora la mantenibilidad del código.

```
@Service 4 usages  👤 Juan Ignacio +2 *
public class EmpleadoServicio {
    @Autowired
    private EmpleadoRepositorio empleadoRepositorio;

    @Autowired
    private UsuarioServicio usuarioServicio;
```

## DAO (Data Access Object)

Es una técnica para proporcionar a las clases sus dependencias desde el exterior en lugar de que las creen internamente. Esto reduce el acoplamiento y mejora la mantenibilidad del código.

```
12  @Repository 6 usages  👤 Ari Rafael Cohen +1
13  public interface EmpleadoRepositorio extends JpaRepository<Empleado,String> {
14      @Query("SELECT e FROM Empleado e WHERE e.tipoEmpleado = 'PROFESOR' and e.eliminado = false and e.id=:id") 3
15      Empleado findProfesor(@Param("id") String id);
16      @Query("SELECT e FROM Empleado e WHERE e.usuario.id = :idUsuario and e.eliminado = false") 1 usage  👤 lucianapi
17      Empleado findEmpleadoByIdUsuario(String idUsuario);
```

# Patrones Aplicados

## Experto en Responsabilidad (GRASP)

Este principio indica que la responsabilidad debe asignarse a la clase que tiene la información necesaria para cumplirla.

```
@Transactional no usages lucianapuentes *
public Rutina crearRutina(String id_socio, String id_empleado, Collection<DetalleRutina> detalle,
                          LocalDate fechaInicio, LocalDate fechaFin) throws ErrorServicio {
    // Lógica para crear una nueva rutina
    validar(id_empleado, id_socio, fechaInicio, fechaFin, detalle);
    Rutina rutina = new Rutina();
    rutina.setFechaInicio(fechaInicio);
    rutina.setFechaFin(null);
    rutina.setDetalleRutina(detalle);
    rutina.setEstado(EstadoRutina.EN_PROCESO);
    if(socioRepositorio.findByNumeroSocio(Long.parseLong(id_socio)).isPresent()){
        rutina.setSocio(socioRepositorio.findByNumeroSocio(Long.parseLong(id_socio)).get());
    }
    else{
        throw new ErrorServicio("No existe un socio con ese ID.");
    }
    rutina.setProfesor(empleadoRepositorio.findProfesor(id_empleado));
    return rutinaRepositorio.save(rutina);
}
```

# Patrones Aplicados

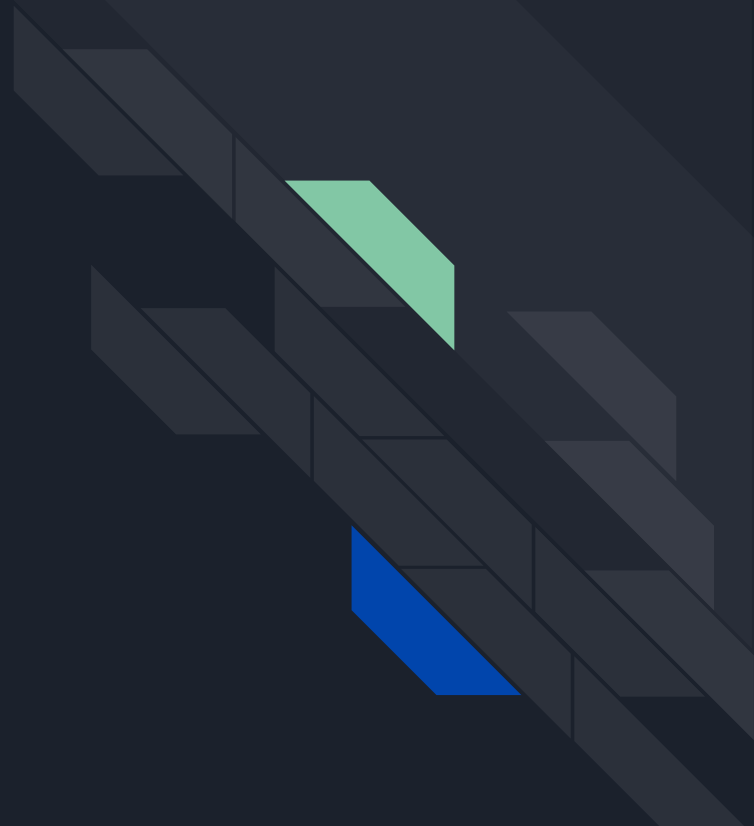
Alta cohesión / Bajo acoplamiento

Alta cohesión: Una clase o módulo realiza tareas relacionadas entre sí (está bien enfocada).

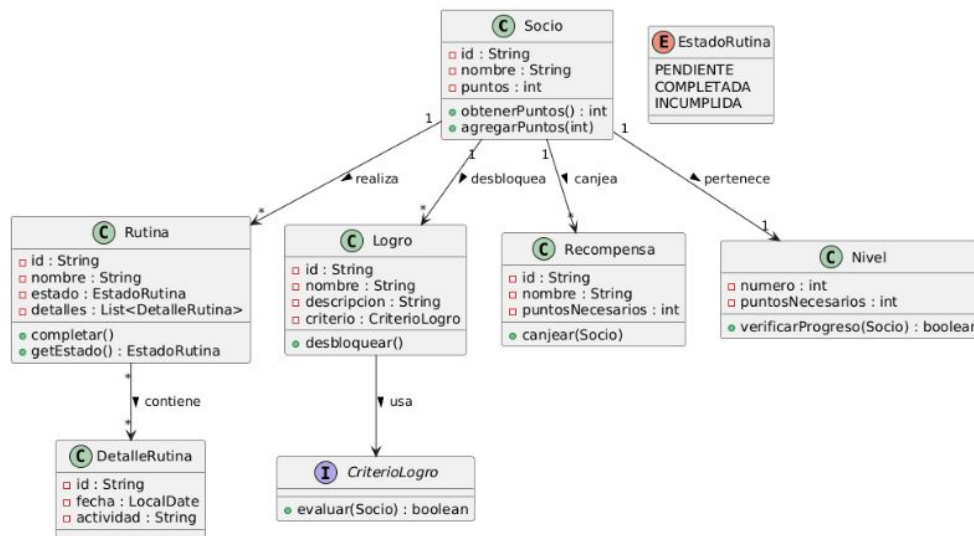
Bajo acoplamiento: Implica que las dependencias entre módulos son mínimas y relegadas a interfaces o inyección.

```
18  @Service 4 usages Juan Ignacio +2 *
19  public class EmpleadoServicio {
20      @Autowired
21      private EmpleadoRepositorio empleadoRepositorio;
22
23      @Autowired
24      private UsuarioServicio usuarioServicio;
25
26      @Transactional 3 usages Ari Rafael Cohen
27      public Empleado buscarEmpleadoPorIdUsuario(String idUsuario) {
28          return empleadoRepositorio.findEmpleadoByIdUsuario(idUsuario);
29      }
30      @Transactional no usages Ari Rafael Cohen
31      public List<Empleado> obtenerProfesores() { return empleadoRepositorio.findAllProfesores(); }
32
33      @Transactional 1 usage Juan Ignacio
34      public List<Empleado> listaEmpleadosActivos() throws ErrorServicio {
```

# Propuesta de Mejora



# Propuesta de Mejora



La propuesta de Mejora presentada es la de utilizar Gamificación en el seguimiento de rutinas. Esencialmente recompensando la realización de actividades con puntajes virtuales, logros y demás herramientas derivadas de videojuegos para aquellos que necesitan motivación extrínseca.

**DEMO**

¡Muchas Gracias por su Atención!