



**UNCUYO**  
UNIVERSIDAD  
NACIONAL DE CUYO



**FACULTAD  
DE INGENIERÍA**

## Ingeniería del Software II

### GUÍA INFORME

#### **PROYECTO “Software para Gimnasio”**



#### **GRUPO Nexora:**

- Luciana Puentes– Legajo: 14339
- Cohen Ari – Legajo: 13672
- Longhino Emmanuel – Legajo: 13233
- Massacesi Juan Ignacio – Legajo: 14169

#### **PROFESORES:**

- Lucia Cortez
- Leandro Spadaro

#### **AÑO:**

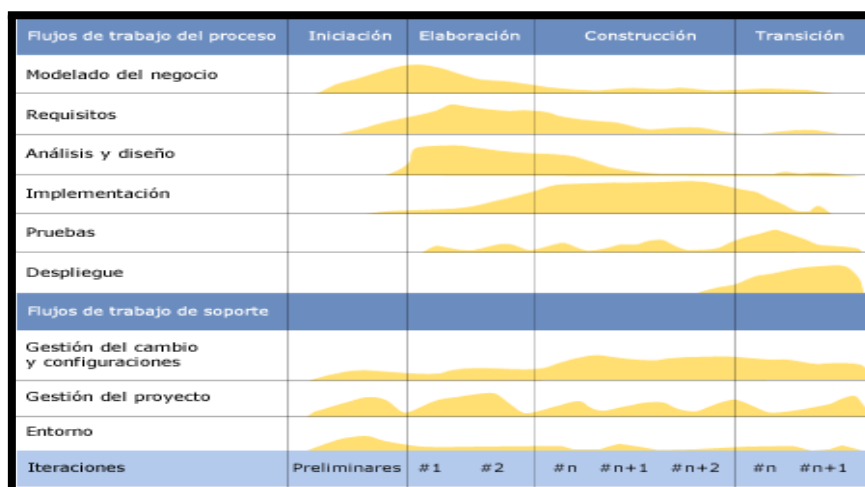
- 2025

# Índice

<b>1. Metodología de desarrollo.....</b>	<b>4</b>
<b>2. Desarrollo de Software.....</b>	<b>6</b>
Iniciación.....	6
2.1 Modelado de Negocio.....	6
2.1.1 Especificación del Proyecto.....	6
2.1.2 Entrevista con el cliente (Preguntas y Respuestas).....	6
2.1.3 Diagrama de clases de Negocio.....	6
2.1.4 Diagrama de paquetes de dominio.....	7
2.2 Requisitos.....	7
2.2.1 Requisitos Funcionales.....	7
2.2.2 Diagrama de caso de uso.....	9
2.2.3 Escenarios Caso de Uso.....	9
2.2.4 Diagrama de secuencia del sistema.....	9
2.2.5 Prototipado de Pantallas.....	9
2.2.6 Requisitos No funcionales.....	10
Elaboración.....	11
2.3 Análisis y Diseño.....	11
2.3.1 Diagrama de clases de diseño.....	11
2.3.2 Diagrama de secuencia de diseño.....	11
2.3.3 Diagrama de paquetes de diseño.....	11
2.3.4 Diagrama Entidad Relación Base de Datos.....	11
Construcción.....	11
2.4 Implementación.....	11
2.4.1 FrameWork de trabajo.....	11
2.4.2 Base de Datos.....	12
2.4.3 Codificación.....	12
Transición.....	12
2.5 Pruebas.....	12
2.5.1 Pruebas Unitarias.....	12
2.5.2 Pruebas de Interfaz.....	12
2.5.3 Pruebas de Carga.....	13
2.5.4 Pruebas de Stress.....	13
Actividades sombrilla.....	13
2.6 Gestión de Proyecto.....	13
2.6.1 Viabilidad.....	13
2.6.2 Estimación de tiempos de desarrollo e implementación.....	13
2.6.3 Estimación de riesgos.....	13
2.6.4 Estimación de costos.....	13
2.6.5 Forma de realización del proyecto. Comprar/Construir.....	13
2.6.6 Presupuesto.....	13

2.6.7 Seguimiento Avance Proyecto.....	13
2.7 Gestión del Cambio y Configuración.....	13
2.7.1 Modificación de Artefactos.....	13
2.8 Entorno.....	13
2.8.1 Arquitectura.....	13
2.8.2 Tecnología.....	13
2.8.3 Base de Datos.....	13
2.8.4 Tipo de Software.....	13
2.8.5 Herramienta de Comunicación del Equipo.....	13
2.8.6 Herramienta para la captura de Requerimiento.....	14
2.8.7 Herramienta de prototipado Interfaces Gráficas UI.....	14
2.8.8 Herramienta de Modelado.....	14
2.8.9 Herramienta de Codificación.....	14
2.8.10 Herramienta de Pruebas de Software.....	14
2.8.11 Herramienta de Gestión de Proyecto.....	14
<b>3. Propuesta de Mejora.....</b>	<b>14</b>
A) REFERENCIA DE MERCADO.....	14
B) PROPUESTA DE MEJORA.....	14
<b>4. Propuesta de Refactorización.....</b>	<b>14</b>
A) VISTA.....	14
B) CONTROLADOR.....	15
C) MODELO.....	15
D) ACCESO A DATOS.....	15
<b>5. Material de Exposición en Conferencia.....</b>	<b>15e</b>
A) POWERPOINT PRESENTACIÓN.....	15
<b>6. Referencias.....</b>	<b>15</b>

# 1. Metodología de desarrollo



El Proceso Racional Unificado (RUP) es una metodología de desarrollo de software que se caracteriza por su flexibilidad y adaptabilidad a las necesidades específicas de cada proyecto.

Una de las principales fortalezas de RUP es su enfoque iterativo e incremental. En lugar de abordar el desarrollo de manera lineal, RUP divide el proyecto en ciclos o iteraciones, cada uno de los cuales representa una fase de desarrollo completa. Este enfoque permite a los equipos refinar y mejorar el software de manera gradual, asegurando que el producto final se ajuste de manera más precisa a las necesidades y requisitos del usuario.

Además, RUP pone un fuerte énfasis en la gestión de requisitos, la arquitectura del sistema y la documentación del proceso. Estas características ayudan a garantizar que el software desarrollado sea de alta calidad, mantenible y fácil de evolucionar a lo largo del tiempo. En resumen, RUP es una metodología flexible y estructurada que facilita la entrega de productos de software de manera más eficiente y predecible.

La metodología se divide en cuatro etapas principales:

1. **Iniciación:** Esta etapa se centra en definir el alcance del proyecto, identificar a los principales interesados y sus necesidades, y establecer la visión general del sistema a desarrollar. Se realiza un análisis de viabilidad y se toman las decisiones iniciales sobre el enfoque y la planificación del proyecto.
2. **Elaboración:** En esta etapa, el equipo se enfoca en definir y estabilizar la arquitectura del sistema. Se realiza un análisis más detallado de los requisitos, se diseña la solución técnica y se planifica el desarrollo del proyecto.
3. **Construcción:** Esta es la etapa de desarrollo propiamente dicha. El equipo se centra en la implementación y prueba del software, siguiendo un enfoque iterativo e incremental. Se van construyendo y entregando versiones funcionales del sistema.

4. Transición: En la última etapa, el software desarrollado se entrega a los usuarios finales. Se realizan las pruebas de aceptación, se capacita a los usuarios y se resuelven los problemas finales antes de la puesta en producción. El objetivo es asegurar una transición sin problemas del sistema al entorno operativo.

Estas cuatro etapas, junto con las iteraciones y actividades específicas dentro de cada una, permiten a los equipos de desarrollo seguir un proceso organizado y estructurado para la entrega de software de calidad.

## **2. Desarrollo de Software**

### ***Iniciación***

#### ***2.1 Modelado de Negocio***

##### **2.1.1 Especificación del Proyecto**

El equipo de desarrollo fue contratado por el Gimnasio “Sport” para realizar un Software que permita administrar la gestión de asociados/as que el mismo posee.

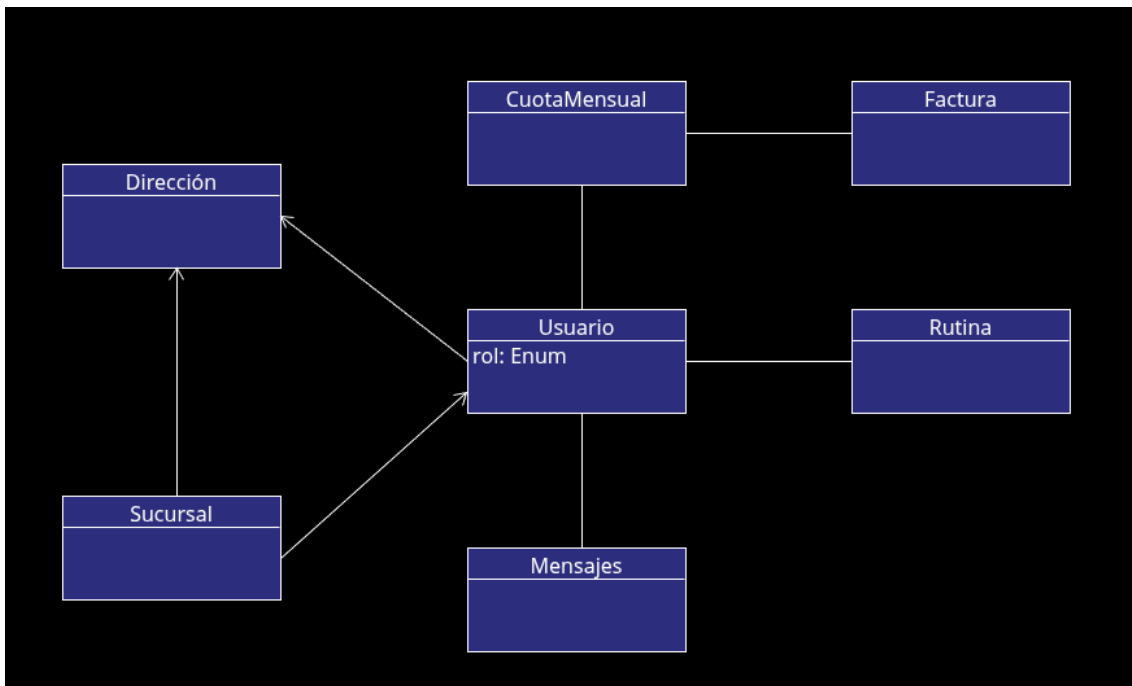
El sistema cuenta con tres tipos de usuarios (Administrador, Profesor, Socio) donde cada uno contará con funcionalidades acordes a su rol.

El Administrador debe tener acceso a la gestión de cuotas y deudas de los socios, además de notificar al socio sobre promociones vía mail. También debe crear los usuarios para los Profesores en el sistema.

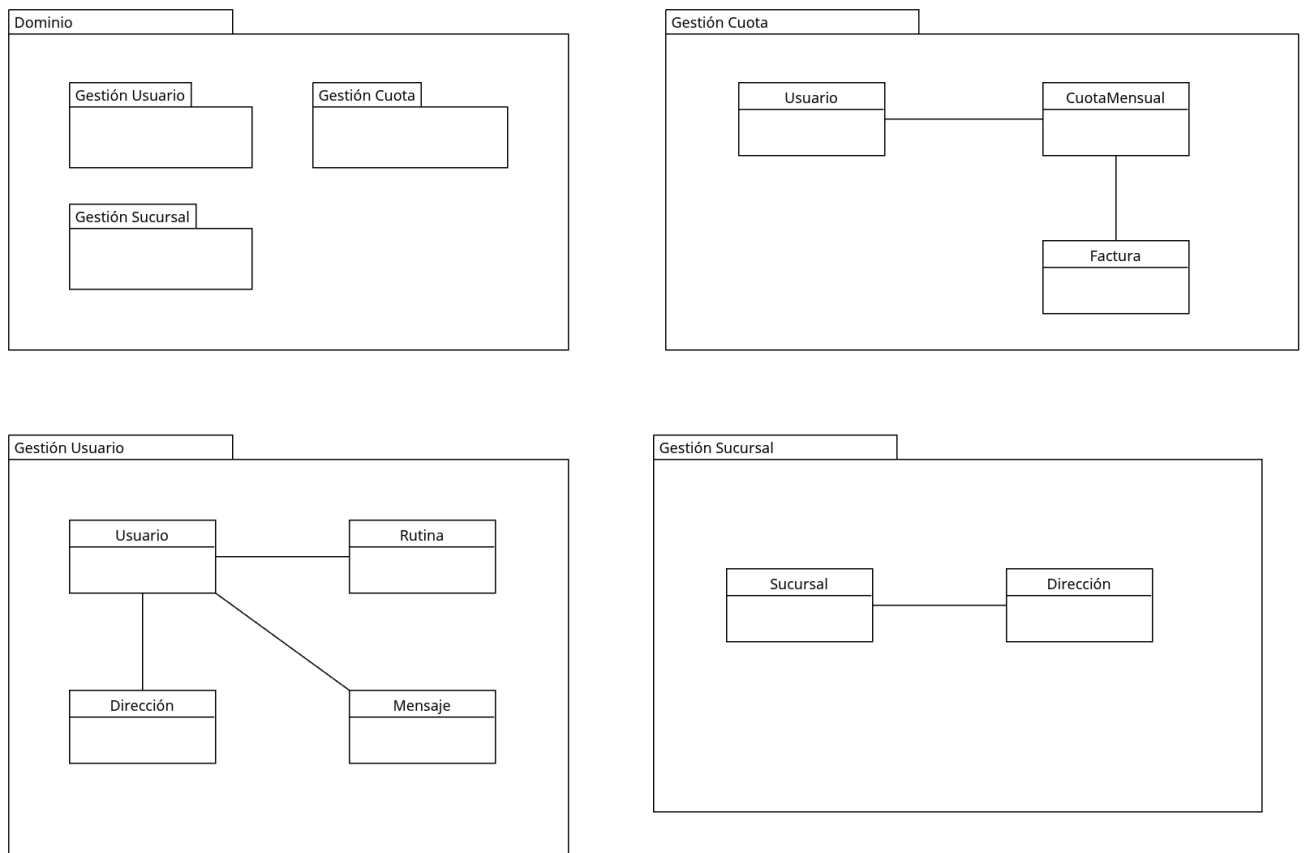
El Profesor debe dar de alta rutinas con sus respectivos detalles para los socios, así puede tener un seguimiento de sus rutinas hechas y a quienes están asignadas. También puede modificarlas y eliminarlas.

El Socio/Asociado puede ver sus rutinas y modificar el detalle de cada una para ir avanzando con el progreso de la misma. Puede pagar la cuota mensual para el caso que se realice con Mercado Pago. Además, podrá acceder al informe de deuda donde se detalla el total de la deuda y los meses adeudados. Los asociados podrán visualizar la rutina en los días que asisten al gimnasio e indicar si la misma fue completada de forma exitosa.

### 2.1.3 Diagrama de clases de Negocio



### 2.1.4 Diagrama de paquetes de dominio



## **2.2 Requisitos**

### **2.2.1 Requisitos Funcionales**

#### Gestión de usuarios

- ABM Usuario

#### Gestión de socios

- ABM Socios
- Consultar Socios

#### Gestión de empleados

- ABM empleados
- Consultar empleados

#### Gestión de cuota mensual y pagos

- Registro de valor de cuota mensual
- Generar cuota mensual para socio
- Pago de cuota (integración Mercado Pago)
- Consulta de estado de cuotas

#### Gestión de rutinas y seguimiento

- Creación y asignación de rutina
- Visualización de rutina por socio
- Marcar actividad/ejercicio como completado

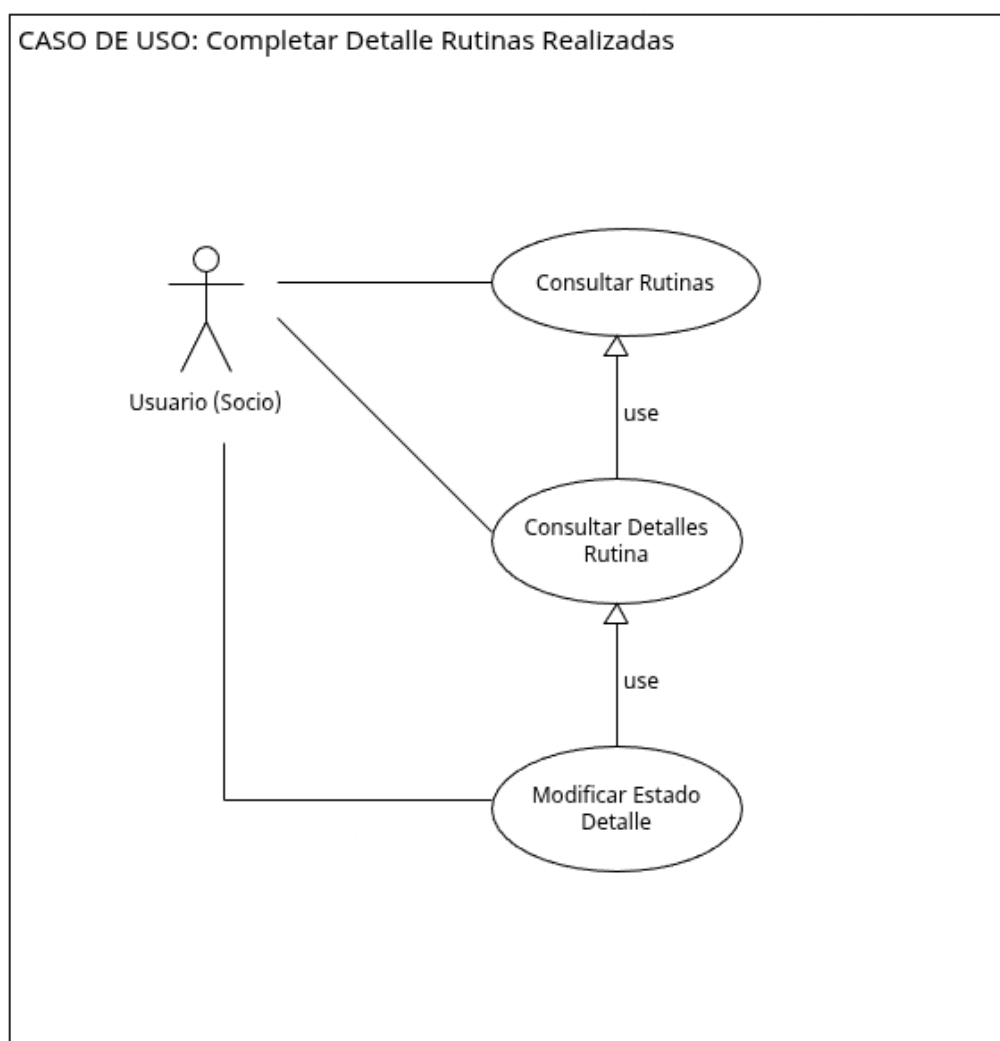
#### Gestión de deuda

- Cálculo de deuda por socio
- Informe de deuda para socio

#### Comunicaciones y notificaciones

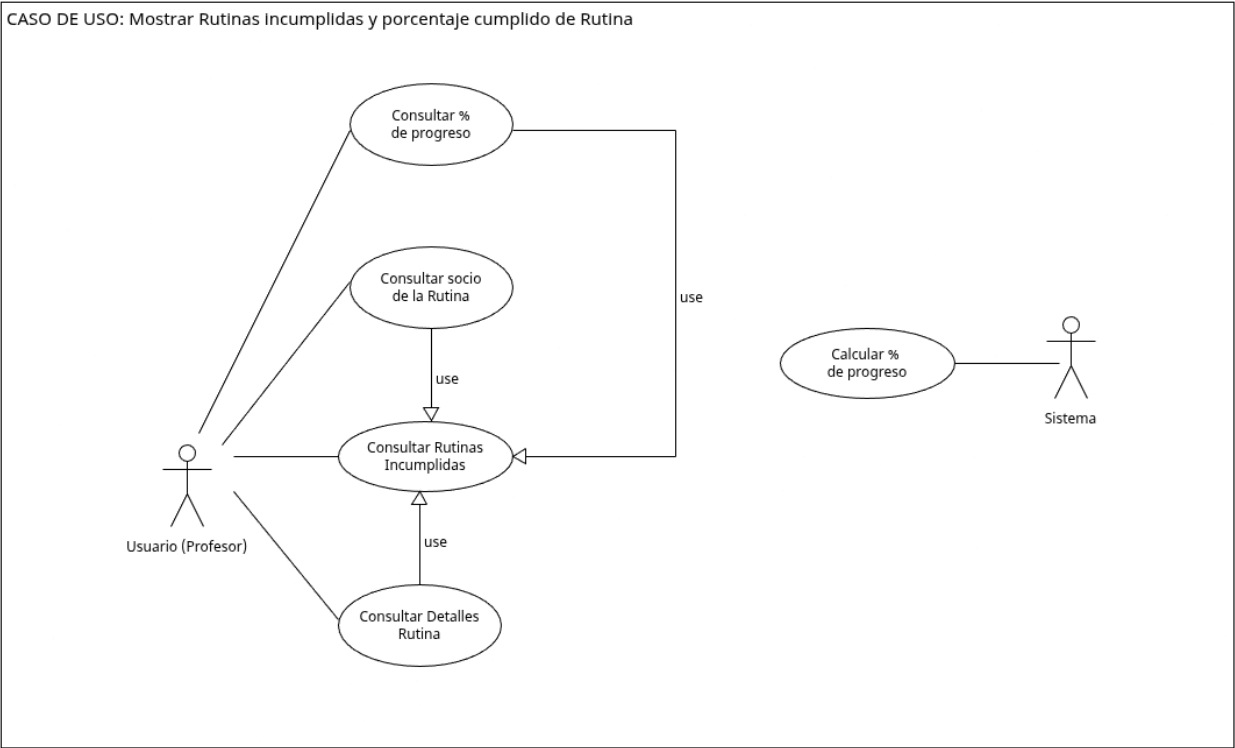
- Envío de campañas promocionales por correo
- Mensaje por cumpleaños a socios

### 2.2.2 Diagrama de caso de uso

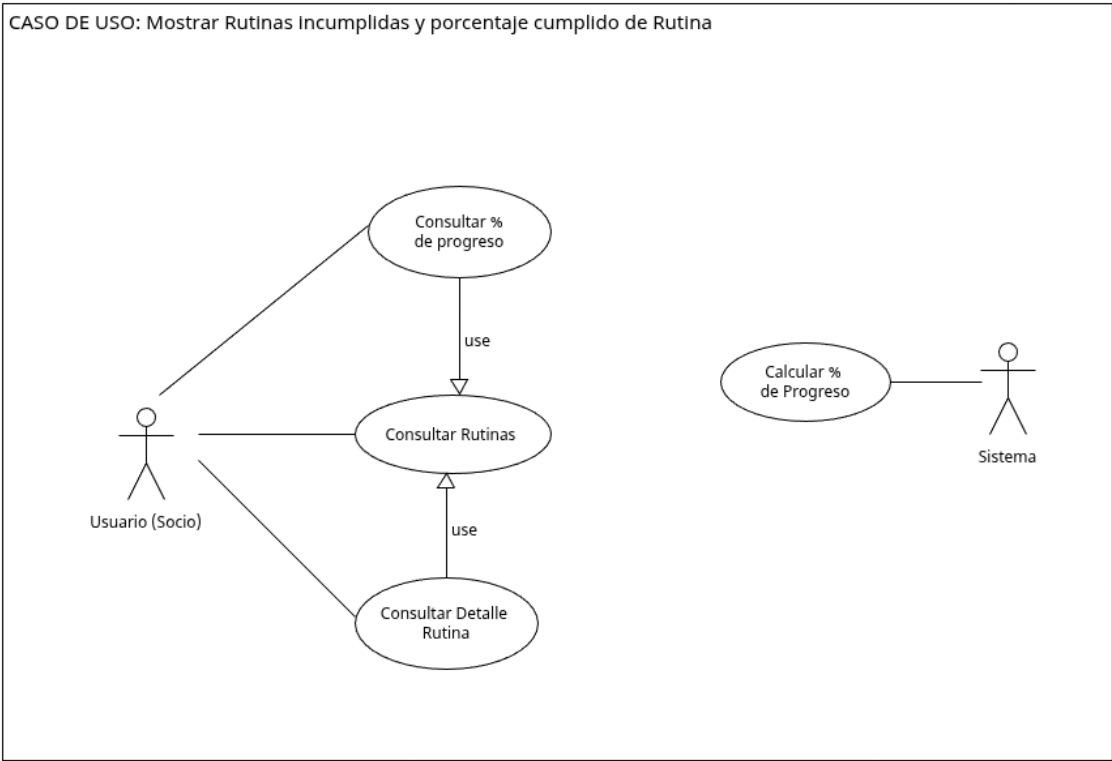




**Perspectiva del Profesor:**



**Perspectiva del Socio:**



### 2.2.3 Escenarios Caso de Uso

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 (asocioado/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	

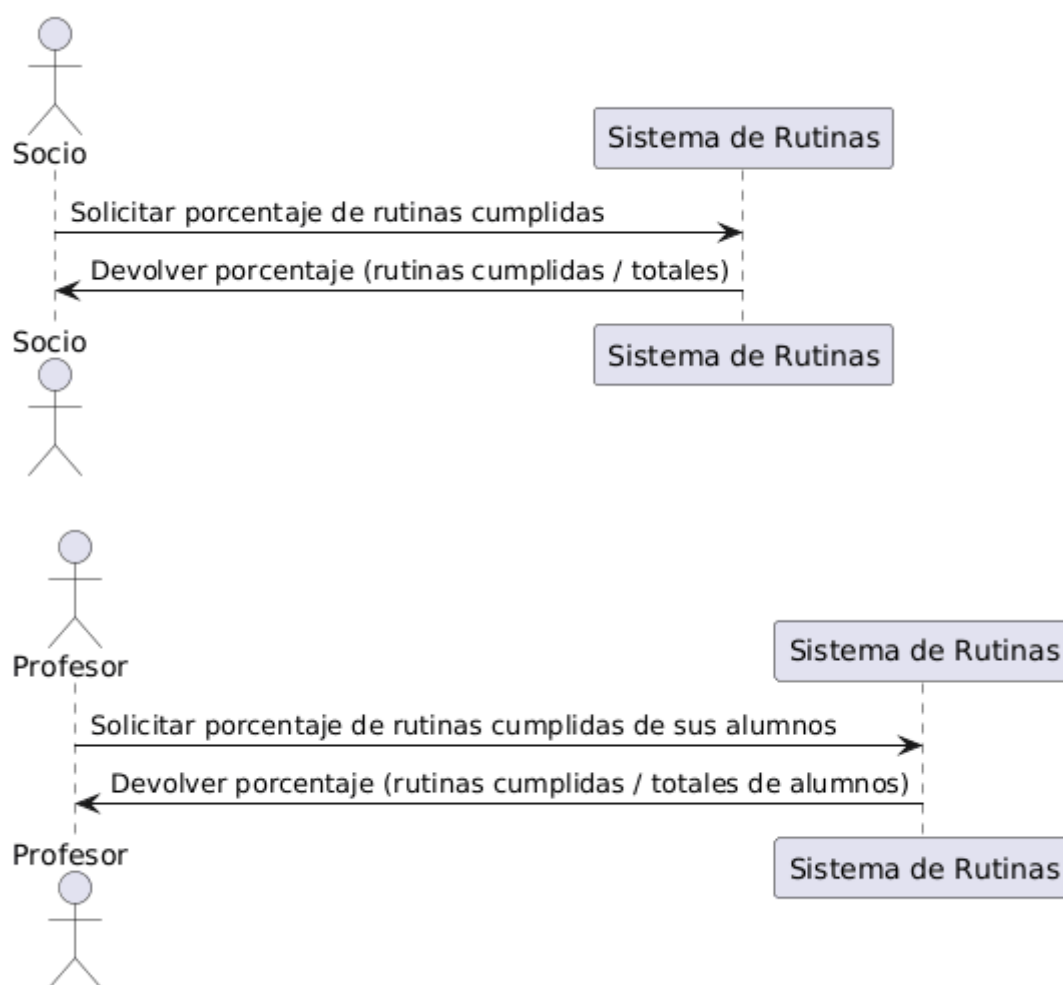
#### Perspectiva del Socio

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 sus rutinas incumplidas y el porcentaje de progreso de las mismas.				
Escenario Principal	Paso	Acción			
	1	El sistema busca las rutinas y su progreso actual asociadas al usuario			
	2	El sistema muestra al usuario sus rutinas con sus actividades y el progreso actual de cada una			
	3	El usuario selecciona el filtro de En_Proceso			
	4	El sistema busca las rutinas del paso 3			
	5	Se repite el paso 1			
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"			
Comentarios				Dependencias	

#### Perspectiva del Profesor

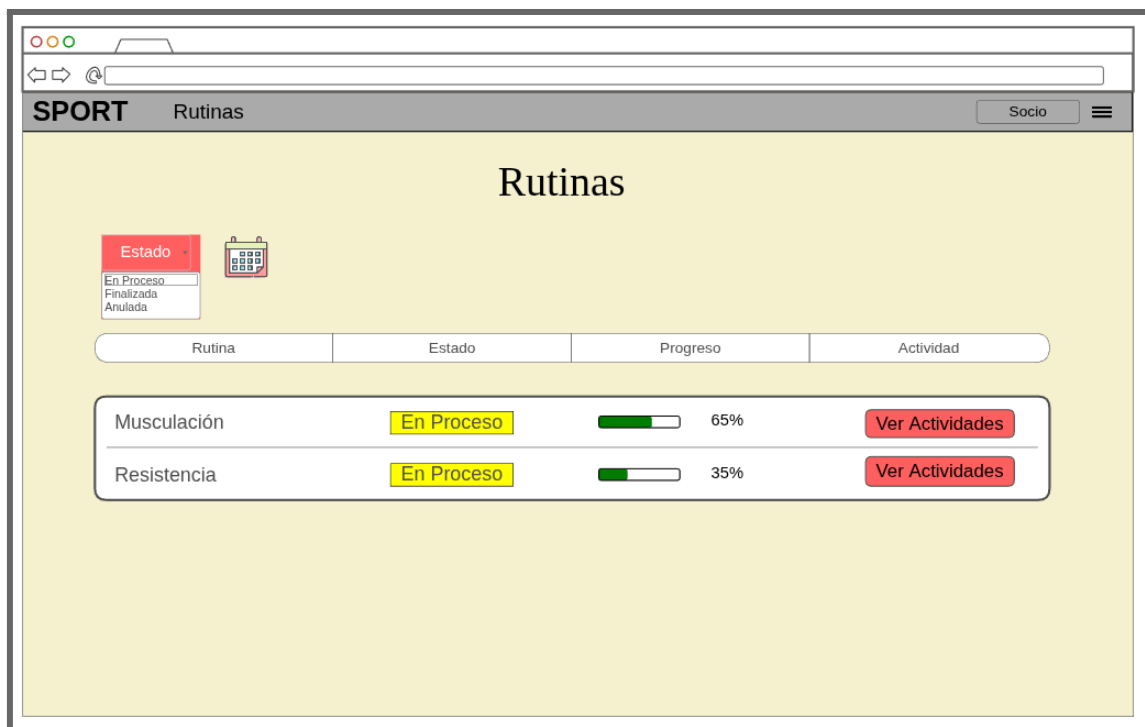
Caso de Uso	Mostrar Rutinas incumplidas con porcentaje de progreso			ID: 2
Prioridad	Alta	Estimación	-	ID Pantalla Prototipo: -
Precondición	El usuario debe estar dado de alta y logueado en el sistema.			
Descripción	El sistema debe mostrar al usuario (profesor) las rutinas incumplidas de sus asesorados 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

## 2.2.4 Diagrama de secuencia del sistema

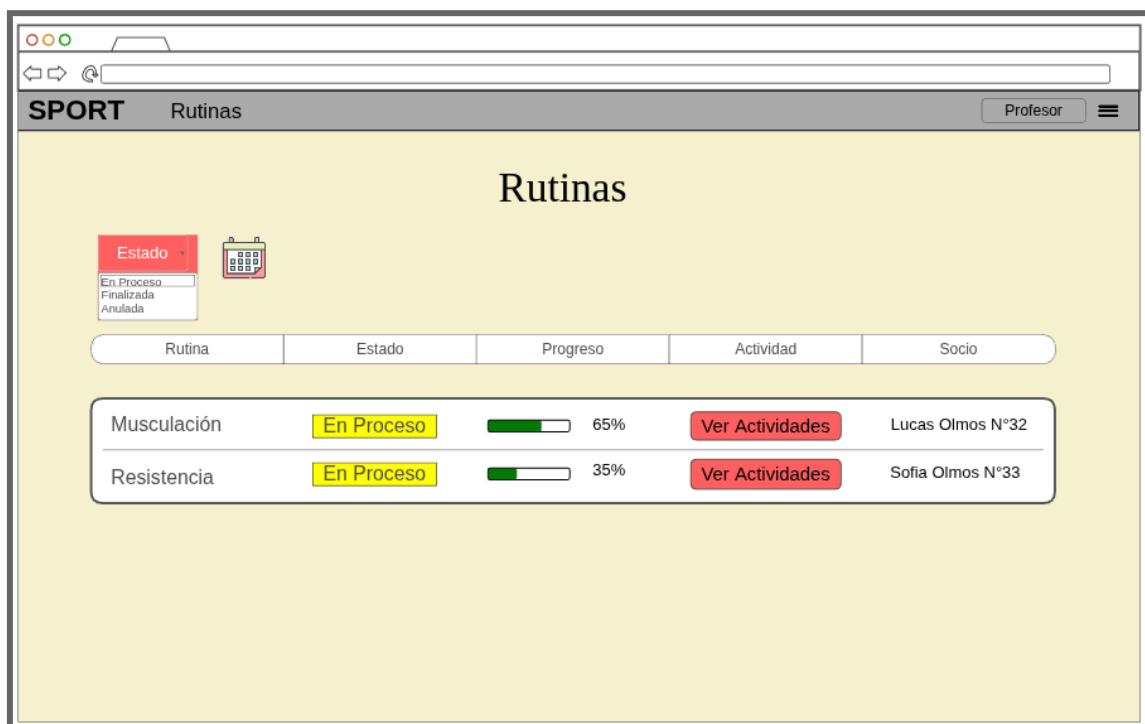


## 2.2.5 Prototipado de Pantallas

**Perspectiva del Socio:**



**Perspectiva del profesor:**





### 2.2.6 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.
- ❖ **Implementación:** Se utilizará Java como lenguaje de servidor de la aplicación, MySQL para la base de datos y patrón de diseño MVC para lograr una mejor mantenibilidad y organización del sistema.
- ❖ **Interfaz:** La interfaz de usuario está diseñada con HTML5 + CSS + Bootstrap + Thymeleaf.
- ❖ **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.
- ❖ Legales: Protección y seguridad de los datos de los usuarios. La aplicación en su totalidad será entregada al cliente.

## ***Elaboración***

### **2.3 *Análisis y Diseño***

#### 2.3.1 Diagrama de clases de diseño

[https://drive.google.com/file/d/1Qbbu-0-ZjkxcmQKGxV2ea\\_ukrjMnLT83/view?usp=drive\\_link](https://drive.google.com/file/d/1Qbbu-0-ZjkxcmQKGxV2ea_ukrjMnLT83/view?usp=drive_link)

#### 2.3.2 Diagrama de secuencia de diseño

[https://drive.google.com/file/d/1LToPTdDYFc9srZS-dvdtjNI\\_ZBUUY7In/view?usp=drive\\_link](https://drive.google.com/file/d/1LToPTdDYFc9srZS-dvdtjNI_ZBUUY7In/view?usp=drive_link)

[https://drive.google.com/file/d/1Ew8TuA47\\_5BilQ5cQGeOzt5sRumJsJZU/view?usp=drive\\_link](https://drive.google.com/file/d/1Ew8TuA47_5BilQ5cQGeOzt5sRumJsJZU/view?usp=drive_link)

[https://drive.google.com/file/d/1Kutz3lhrfbpR3gsKEYjBdaA-Jx8F9Zsw/view?usp=drive\\_link](https://drive.google.com/file/d/1Kutz3lhrfbpR3gsKEYjBdaA-Jx8F9Zsw/view?usp=drive_link)

#### 2.3.3 Diagrama de paquetes de diseño

[https://drive.google.com/file/d/1XRmxqkPnj22iGEfkytwuRzOwdZx4MRQG/view?usp=drive\\_link](https://drive.google.com/file/d/1XRmxqkPnj22iGEfkytwuRzOwdZx4MRQG/view?usp=drive_link)

#### 2.3.4 Diagrama Entidad Relación Base de Datos

<https://drive.google.com/file/d/1GGTiHoHp5rTkd-ISANL5X4HAVVBPH11z/view?usp=sharing>

## ***Construcción***

### **2.4 *Implementación***

#### 2.4.1 Framework de trabajo

Para el desarrollo del sistema se utilizó Java 17 como lenguaje principal, aprovechando las ventajas de una versión moderna, estable y con soporte a largo plazo (LTS).

El proyecto se implementó utilizando el framework Spring, el cual proporciona un ecosistema completo para el desarrollo de aplicaciones. En particular, se hizo uso de Spring Boot para simplificar la configuración inicial y la gestión de dependencias, favoreciendo una arquitectura modular, escalable y fácilmente integrable con otros componentes.

Para la capa de persistencia se empleó JPA (Java Persistence API) junto con Hibernate como proveedor de implementación. Esta combinación permitió mapear las entidades del dominio a la base de datos relacional de manera transparente, automatizando gran parte de las operaciones CRUD y asegurando la consistencia en el acceso a los datos.

#### 2.4.2 Base de Datos

Se utilizó MySQL 8.0.43 con interfaz MySQL Workbench 8.0.36

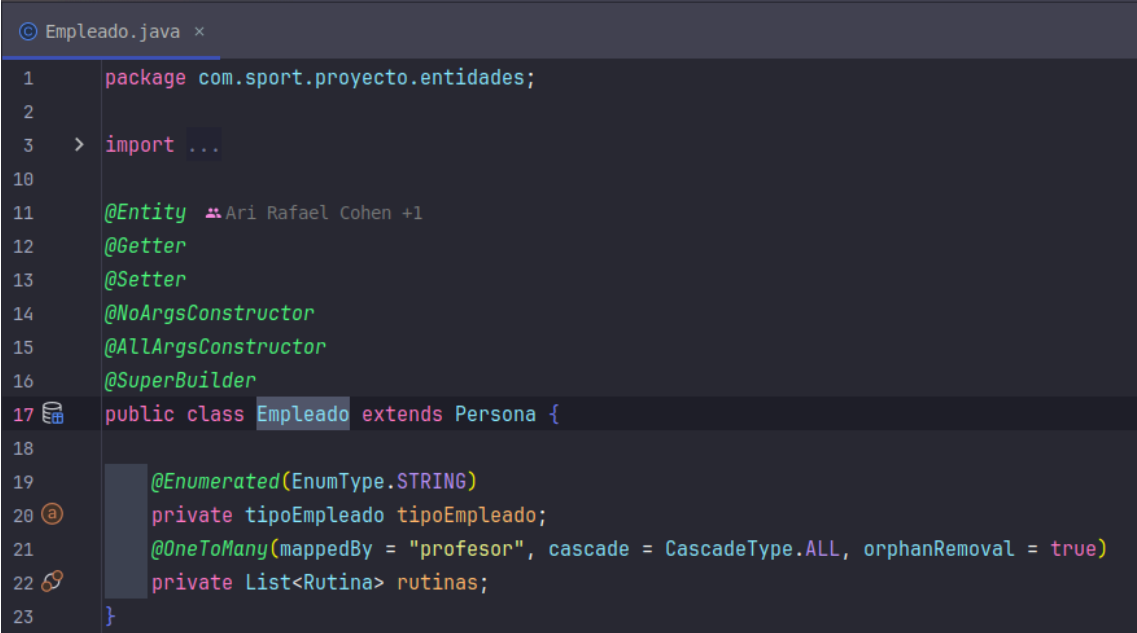
#### 2.4.3 Codificación

Mediante el siguiente link se puede acceder a la codificación del proyecto:

<https://github.com/AriC001/IS2-2025>

#### 2.4.4 Patrones Utilizados

MVC (Modelo/Vista/Controlador) y Capas



```
Empleado.java x
1 package com.sport.proyecto.entidades;
2
3 > import ...
10
11 @Entity
12 @Getter
13 @Setter
14 @NoArgsConstructor
15 @AllArgsConstructor
16 @SuperBuilder
17 public class Empleado extends Persona {
18
19     @Enumerated(EnumType.STRING)
20     private tipoEmpleado tipoEmpleado;
21     @OneToMany(mappedBy = "profesor", cascade = CascadeType.ALL, orphanRemoval = true)
22     private List<Rutina> rutinas;
23 }
```

```
EmpleadoRepositorio.java x
1 package com.sport.proyecto.repositorios;
2
3 > import ...
11
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 usages lucianapuentes
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 lucianapuentes
17     Empleado findEmpleadoByIdUsuario(String idUsuario);
18     @Query("SELECT e FROM Empleado e WHERE e.tipoEmpleado = 'PROFESOR' and e.eliminado = false") 1 usage Ari Rafael Cohen
19     public List<Empleado> buscarEmpleadosActivos();
20     @Query("SELECT e FROM Empleado e WHERE e.tipoEmpleado = 'PROFESOR' and e.eliminado = false") 1 usage Ari Rafael Cohen
21     public List<Empleado> findAllProfesores();
22     @Query("SELECT e FROM Empleado e WHERE e.numeroDocumento = :numeroDocumento and e.eliminado = false") no usages Ari Rafael Cohen
23     public Optional<Empleado> findByNumeroDocumento(@Param("numeroDocumento") String numeroDocumento);
```



```

EmpleadoServicio.java x
1  package com.sport.proyecto.servicios;
2
3  > import ...
17
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(); }
34
35      @Transactional 1 usage Juan Ignacio
36      public List<Empleado> listaEmpleadosActivos() throws ErrorServicio {

```

```

EmpleadoControlador.java x
1  package com.sport.proyecto.controladores;
2
3  > import ...
21
22  @Controller Ari Rafael Cohen +2
23  @RequestMapping("/empleados")
24  public class EmpleadoControlador {
25      @Autowired
26      private EmpleadoServicio empleadoServicio;
27
28      @GetMapping("") Juan Ignacio +1
29      public String empleados(Model model) {
30          try{
31              List<Empleado> empleados = empleadoServicio.listaEmpleadosActivos();
32              model.addAttribute(attributeName: "empleados", empleados);
33              return "views/empleados";
34          } catch (Exception e){
35              model.addAttribute(attributeName: "error", e.getMessage());
36              return "views/empleados";
37          }
38      }
39  }

```

```

empleados.html x
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4   <meta charset="utf-8"/>
5   <meta name="viewport" content="width=device-width, initial-scale=1"/>
6   <title>Empleados - Gimnasio</title>
7   <!-- Bootstrap CSS -->
8   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">
9 </head>
10 <body>
11
12 <!-- Navbar -->
13 <div th:replace="~{components/navbar :: nav_bar}"></div>
14
15 <div class="container mt-4">
16   <div class="d-flex justify-content-between align-items-center mb-3">
17     <h2>Empleados</h2>
18     <a th:href="@{/empleados/nuevo}" class="btn btn-primary">+ Nuevo Empleado</a>
19   </div>

```

Tomando como ejemplo Empleado, el patrón MVC se utilizó para separar UI (vista), lógica de presentación (controlador) y datos/dominio (modelo).

### Inyección de Dependencia

```

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

    @Autowired
    private UsuarioServicio usuarioServicio;
}

```

Tomando como ejemplo EmpleadoServicio se le proporcionan sus dependencias (empleadoRepositorio y usuarioServicio) desde fuera (Spring las inyecta), evitando que la clase las cree directamente.

### Experto en Responsabilidad (GRASP)

```

@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);
}

```

La clase RutinaServicio tiene la responsabilidad de crear la Rutina porque tiene la información necesaria para cumplirla (tiene la información y dependencias).

## DAO (Data Access Object)

```

EmpleadoRepositorio.java x
1 package com.sport.proyecto.repositorios;
2
3 > import ...
11
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 usages lucianapuentes
15     Empleado findProfesor(@Param("id") String id);
16     @Query("SELECT e FROM Empleado e WHERE e.usuario.id = :idUser and e.eliminado = false") 1 usage lucianapuentes
17     Empleado findEmpleadoByIdUsuario(String idUsuario);
18     @Query("SELECT e FROM Empleado e WHERE e.tipoEmpleado = 'PROFESOR' and e.eliminado = false") 1 usage Ari Rafael Cohen
19     public List<Empleado> buscarEmpleadosActivos();
20     @Query("SELECT e FROM Empleado e WHERE e.tipoEmpleado = 'PROFESOR' and e.eliminado = false") 1 usage Ari Rafael Cohen
21     public List<Empleado> findAllProfesores();
22     @Query("SELECT e FROM Empleado e WHERE e.numeroDocumento = :numeroDocumento and e.eliminado = false") no usages Ari Rafael Cohen
23     public Optional<Empleado> findByNumeroDocumento(@Param("numeroDocumento") String numeroDocumento);

```

EmpleadoRepositorio representa el DAO en el proyecto. Implementado con Spring Data JPA, centraliza las consultas JPQL y los accesos CRUD, manteniendo separada la persistencia de la lógica de negocio.

## Alta Cohesión / Bajo Acoplamiento

```
EmpleadoServicio.java x
1 package com.sport.proyecto.servicios;
2
3 > import ...
17
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
31     @Transactional no usages Ari Rafael Cohen
32     public List<Empleado> obtenerProfesores() { return empleadoRepositorio.findAllProfesores(); }
33
34
35     @Transactional 1 usage Juan Ignacio
36     public List<Empleado> listaEmpleadosActivos() throws ErrorServicio {
```

**Alta cohesión:** EmpleadoServicio sólo hace cosas relacionadas con la gestión de empleados (validar, creación/edición/eliminación, reglas de negocio).

**Bajo acoplamiento:** depende de interfaces/servicios externos inyectados (repositorios u otros servicios), no los crea ni conoce su implementación concreta, así es fácil sustituirlos.

## Transición

### 2.5 Pruebas

#### 2.5.1 Pruebas Unitarias

Las pruebas de los distintos métodos se realizaron de forma manual, verificando en cada caso el correcto funcionamiento individual. Se incluyeron en las pruebas tanto los métodos de los controladores como de los servicios, asegurando que los resultados obtenidos fueran consistentes con los esperados. Asimismo, se evaluaron las consultas a la base de datos con el fin de garantizar la adecuada persistencia de la información y la correcta recuperación de los datos almacenados.

#### 2.5.2 Pruebas de Interfaz

Las pruebas de interfaz se llevaron a cabo con el objetivo de verificar la correcta interacción entre el usuario y el sistema. Se evaluó la disposición de los componentes gráficos, la coherencia en la navegación entre pantallas y la respuesta de los distintos elementos de la interfaz frente a las acciones del

usuario. Asimismo, se comprobó que los formularios permitieran la carga, modificación y visualización de datos de manera intuitiva y sin errores. Estas pruebas garantizaron que la experiencia de uso fuera clara, accesible y consistente con los requisitos planteados.

## ***Actividades sombrilla***

### **2.6 Gestión de Proyecto.**

#### **2.6.1 Viabilidad**

La viabilidad del software se analiza considerando los distintos factores que influyen en su desarrollo, implementación y posterior mantenimiento.

- **Viabilidad técnica:**

El sistema puede desarrollarse utilizando tecnologías ampliamente probadas y disponibles, tales como Java, Spring Boot y MySQL, lo que garantiza estabilidad, escalabilidad y soporte a largo plazo. Además, el equipo cuenta con el conocimiento necesario para la implementación, y los requerimientos de hardware/infraestructura son compatibles con los recursos disponibles.

- **Viabilidad económica:**

El costo de desarrollo se encuentra dentro de los márgenes aceptables para la organización, dado que se aprovechan herramientas open source. Los costos de mantenimiento y capacitación son reducidos en comparación con los beneficios obtenidos como mejoras en la eficiencia y reducción de riesgos.

- **Viabilidad operativa:**

El software responde directamente a las necesidades de los usuarios finales, aportando mejoras en la gestión de la información, la comunicación interna y la toma de decisiones. La interfaz es sencilla, lo que facilita su uso.

#### **2.6.2 Estimación de tiempos de desarrollo e implementación**

Fases de desarrollo:

1. Análisis de requerimientos: 4 días
2. Diseño: 4 días
3. Desarrollo: 15 días
4. Pruebas: 3 días

#### **2.6.3 Estimación de riesgos**

- **Técnicos:** Problemas con la integración de sistemas externos y sistema de notificaciones.

- Recursos Humanos: Posibles ausencias o demoras por curva de aprendizaje en las tecnologías usadas.
- Presupuesto: Gastos no previstos en infraestructura o servicios externos.
- Cronograma: Retrasos en entregas por dependencias de sistemas externos y pruebas adicionales.

#### 2.6.4 Estimación de costos

El mayor costo del proyecto se concentra en el recurso humano, principalmente en el equipo de desarrollo. Los programadores son responsables del diseño, implementación y pruebas del sistema. Los costos asociados incluyen horas de trabajo, capacitaciones necesarias y posibles horas extras en etapas críticas del desarrollo.

#### 2.6.5 Forma de realización del proyecto. Comprar/Construir

Se optó por construir el sistema. La decisión se basa en la necesidad de contar con un software a medida, que se adapte a los requerimientos específicos del proyecto y permita mayor flexibilidad para futuras modificaciones e integraciones.

#### 2.6.7 Seguimiento Avance Proyecto

El seguimiento del proyecto se realizó mediante reuniones virtuales en Google Meet y comunicación constante a través de WhatsApp para la coordinación de tareas y resolución de dudas.

## 2.7 **Gestión del Cambio y Configuración**

### 2.7.1 Modificación de Artefactos

Las modificaciones de artefactos (código fuente, documentación y base de datos) se gestionaron mediante Git como sistema de control de versiones.

Cada cambio realizado por los desarrolladores fue registrado en ramas específicas y posteriormente integrado en la rama principal tras su validación.

De esta manera, se mantuvo un historial completo de los cambios, lo que facilitó el seguimiento, la reversión en caso de errores y la correcta configuración del sistema en cada etapa del desarrollo.

## 2.8 **Entorno**

### 2.8.1 Arquitectura

- ❖ Arquitectura: Cliente Servidor con patrón de arquitectura MVC.
- ❖ Capas: Vista, Controlador y Modelo que maneja la lógica de negocio y la persistencia de datos.

### 2.8.2 Tecnología

- ❖ Lenguaje de Programación: Java
- ❖ FrameWork: Spring

- ❖ Servidor de aplicaciones: Tomcat

### 2.8.3 Base de Datos

- ❖ Base de datos: MySQL
- ❖ Persistencia: ORM con JPA/Hibernate

### 2.8.4 Tipo de Software

- ❖ El sistema desarrollado constituye un software a medida, diseñado en base a los requisitos específicos del cliente. Se ha concebido con un enfoque que facilite su mantenimiento y garantice la posibilidad de escalarlo, permitiendo la incorporación de futuras mejoras y nuevas funcionalidades.
- ❖ Software libre.

### 2.8.5 Herramienta de Comunicación del Equipo

- ❖ Herramientas de comunicación: WhatsApp y Meet
- ❖ Funciones: Mensajes (Chat) y Videollamadas.

### 2.8.6 Herramienta para la captura de Requerimiento

- ❖ Herramienta: WhatsApp
- ❖ Funciones: Manejo organizado de la información, acceso rápido a los datos, control de las actividades realizadas.

### 2.8.7 Herramienta de prototipado Interfaces Gráficas UI

- ❖ Herramienta: NinjaMock

### 2.8.8 Herramienta de Modelado

- ❖ Herramienta: UMLet/UMLetino
- ❖ Funciones: Realizar diagramas de casos de uso, clases, paquete y secuencia.

### 2.8.9 Herramienta de Codificación

- ❖ Herramienta: IntelliJ (IDE)
- ❖ Versionador: Git y Github para seguimiento remoto.

### 2.8.10 Herramienta de Pruebas de Software

- ❖ Herramienta: La prueba de Software se hace de forma manual.

### 2.8.11 Herramienta de Gestión de Proyecto

- ❖ Herramientas: WhatsApp y Github
- ❖ Funciones: Asignación de tareas, seguimiento de código y comunicación rápida.

### 3. Propuesta de Mejora

#### **A) REFERENCIA DE MERCADO**

JEFIT es una aplicación móvil utilizada para el registro y seguimiento de rutinas, además de el progreso del usuario a lo largo del tiempo mediante estadísticas y comparaciones. Posee templates de rutinas modificables y categorización detallada de las distintas tareas para poder planear y seguir la rutina preferida.

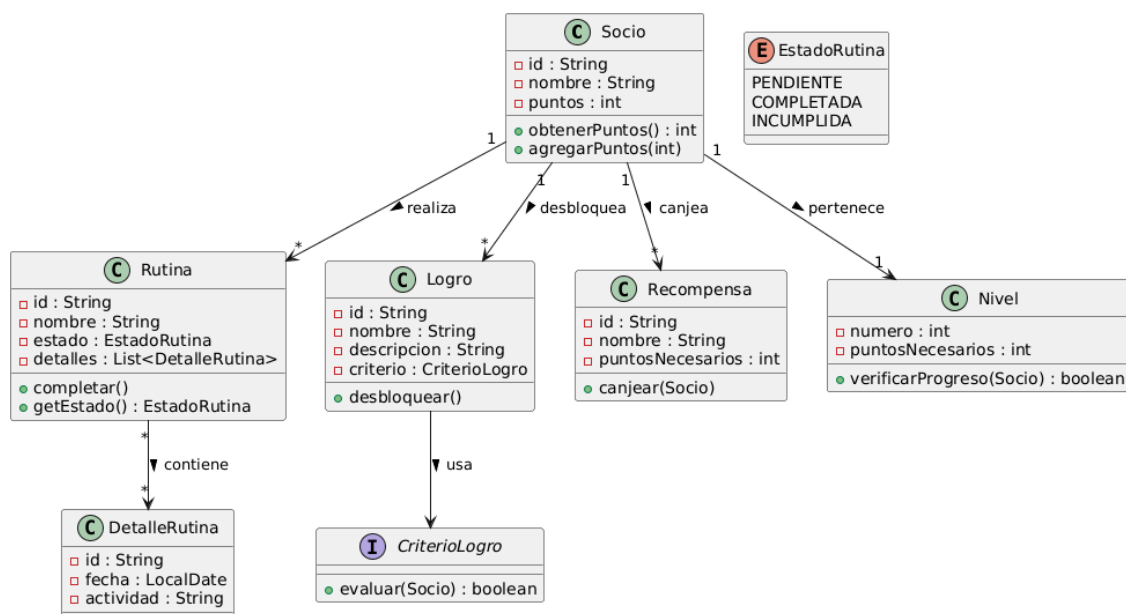
Existen múltiples aplicaciones comparables como Hevy y SmartGym, donde se utilizan para mantener un logbook del progreso del entrenamiento. Sin embargo, estas aplicaciones se enfocan principalmente en dar la información de forma clara y esteril. Esto es útil en general, pues permite conocer cómo está progresando el usuario, pero para personas la información presentada así no les incita a progresar, puede que sientan que es muy lento el progreso, o no es visible.

Para esta gente que necesita motivación extra, se puede utilizar una técnica que se ha estado volviendo popular recientemente, la Gamificación.

#### **B) 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.

El siguiente diagrama mostraria una versión hipotética del concepto:





Otra propuesta interesante podría ser que el gimnasio si tuviese diversas actividades, las tenga listadas con cupo, y cada socio puede reservar su actividad en determinado horario con determinado profesor. Esto implicaría manejar un sistema de reservas con horarios, manejo de actividades, actividades que se dan de baja, que se modifican, usuarios que se puedan dar de baja a una actividad y el cupo disponible debería cambiar en tiempo real.

## **4. Propuesta de Refactorización**

### ***A) VISTA***

Extraer campos comunes a fragmentos de Thymeleaf para reutilizar formularios.

Resultado: menos duplicación y UX consistente.

### ***B) CONTROLADOR***

No pasar entidades a la vista, usar DTOs para transferir datos entre capas.

### ***C) MODELO***

Optimizar las consultas SQL añadiendo índices en columnas de búsqueda/ordenación.

### ***D) ACCESO A DATOS***

Implementar una caché en el sistema para no realizar consultas repetitivas en la base de datos.

## **5. Material de Exposición en Conferencia**

### ***A) POWERPOINT PRESENTACIÓN***

Acceda a la presentación mediante el siguiente link: [Presentación](#)

### ***B) Porcentaje de Código***

El estado actual del sistema/software es de un 70%.

## **6. Referencias**

1. Oracle. Técnica de Sun Java Enterprise System  
<https://docs.oracle.com/cd/E19528-01/820-0888/6ncjkpnh6/index.html>