Universidad Nacional de Cuyo

Facultad de Ingeniería

Licenciatura en Ciencias de la Computación

Ingeniería del Software II

Trabajo Integrador N°1: *Sistema de Software para Gimnasio*

Grupo 2 - ElvaStudio

Trinidad Perea

Perla Valerio

Giovanni Azurduy

Adriano Fabris

Septiembre 2025

Índice

1. Introducción

El presente trabajo práctico integrador tiene como objetivo aplicar los conceptos de Ingeniería de Software que vimos durante la materia en el desarrollo de un sistema realista. El proyecto consiste en el diseño e implementación de un software para la gestión de un gimnasio, llamado «Sport», que cuenta con varias sucursales en la provincia de Mendoza.

A lo largo del trabajo fuimos pasando por distintas etapas: primero el análisis de requisitos, después la elaboración de modelos UML (casos de uso, diagramas de clases, diagramas de secuencia), y finalmente la implementación en código. En la práctica notamos que los modelos iniciales sirvieron de guía, pero varias cosas cambiaron en la implementación final, lo cual nos parece normal ya que fue nuestro primer proyecto completo y porque el software siempre evoluciona durante el desarrollo.

El sistema que construimos permite gestionar socios, usuarios y sus roles, la cobranza de cuotas mensuales, la generación de facturas, la gestión de deudas y también rutinas de entrenamiento que los profesores asignan a los socios. Además, pensamos y agregamos una funcionalidad adicional: un módulo de recompensas por puntos, que busca motivar a los socios a asistir, pagar en término e invitar a nuevos miembros.

El enfoque de este informe es mostrar tanto los modelos previos como la implementación resultante, explicando las decisiones de diseño, los patrones aplicados y la experiencia de trabajo en equipo.

2. Caso de Uso Crítico: Pago de Cuotas

2.1. Diagrama de Caso de Uso

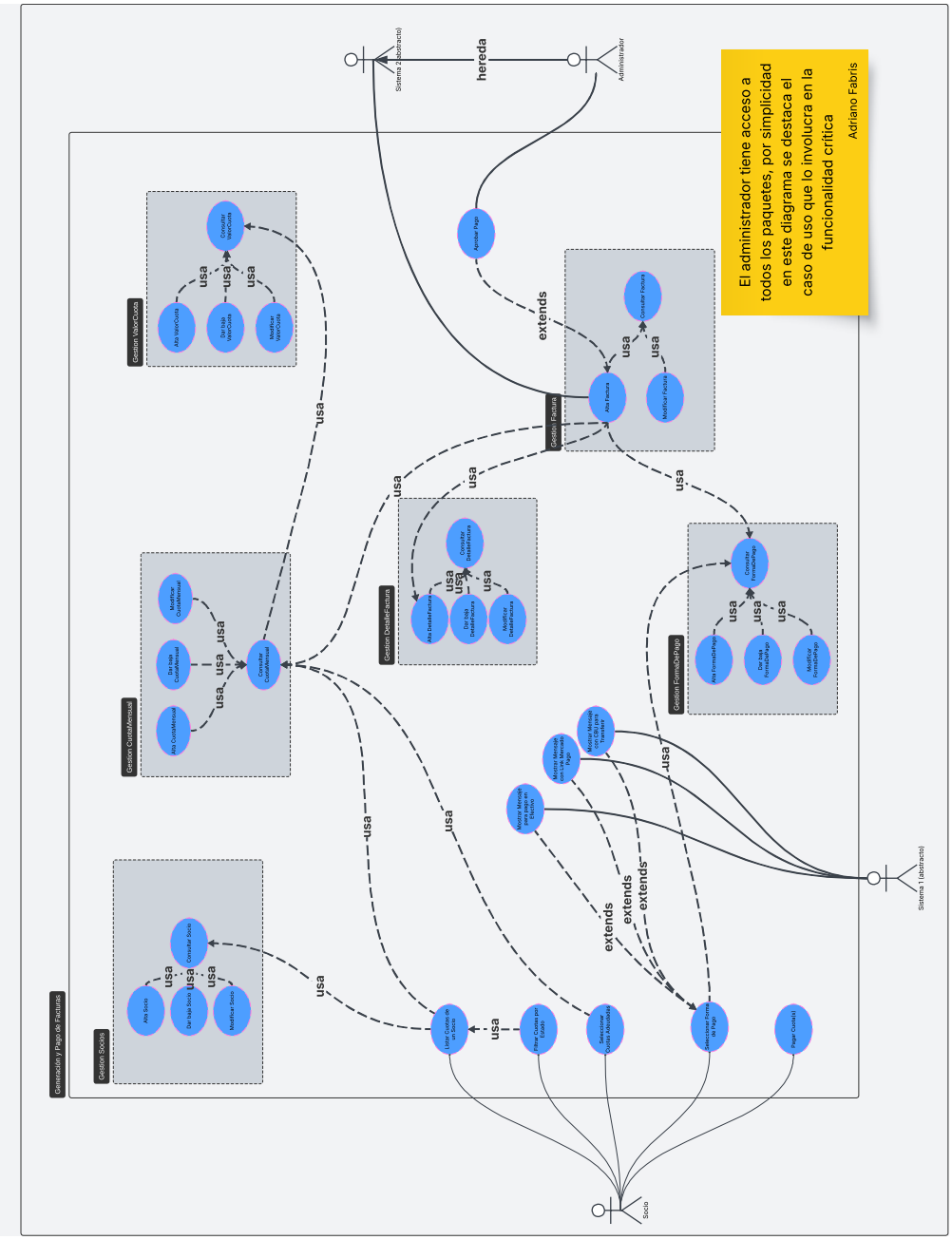

Figura 1: Diagrama de Caso de Uso

Enlace: `https://lucid.app/lucidchart/75a89fc9-ae9e-469b-a08a-86d133243093/edit?view_items=jFBTfsF-ByK_&invitationId=inv_ef5d30df-b4d8-4ee1-bddc-cb6337fbbc2b`

2.2. Escenario de Caso de Uso

ESCENARIO DE CASO DE USO #1	
Prioridad	9/10 (muy alta)
Nombre	Pago de cuota con su generación de factura y forma de pago
Descripción	Un socio quiere pagar una o más cuotas
Actores principales	Socio Administrador
Actores secundarios (si aplica)	Sistema (abstracto)
Precondiciones	El socio debe existir, tener un estado activo, y tener una o más cuotas adeudadas El sistema debe permitir pagos mediante efectivo, transferencia o billetera virtual (Mercado Pago)
Postcondiciones	El socio habrá pagado una o más cuotas adeudadas El sistema habrá generado una factura para las cuotas seleccionadas y pagadas por el socio
Flujo principal de eventos	1. El socio accede al listado de cuotas correspondientes a su perfil 2. El socio selecciona las cuotas adeudadas que pagará 3. El socio selecciona un método de pago 4. El socio efectúa el pago 5. El administrador aprueba el pago 6. El sistema genera la factura correspondiente a las cuotas pagadas
Flujos alternativos / excepciones	(A1) El socio puede filtrarlas según su estado: adeudada/pagada (A3.1) Si el método de pago es en efectivo el sistema muestra: "Acérquese a mesa de entrada" (A3.2) Si el método de pago es por transferencia el sistema muestra el CBU destino del gimnasio (A3.3) Si el método de pago es por Mercado Pago el sistema muestra un link de pago
Requisitos especiales	El link de pago para una cuota cuyo pago se efectuará por Mercado Pago tiene un tiempo de expiración predeterminado
Frecuencia de uso	Se estima que se ejecutará n/2 veces entre el 1er y 5to día del correspondiente mes, y el resto de veces en el período restante, n es el número de socios activos en el mes previo.
Notas adicionales	-

Figura 2: Escenario de Caso de Uso

Enlace: `https://lucid.app/lucidchart/75a89fc9-ae9e-469b-a08a-86d133243093/edit?view_items=5aMRDlw8F6nU&invitationId=inv_ef5d30df-b4d8-4ee1-bddc-cb6337fbbc2b`

2.3. Diagrama de Secuencia de Diseño

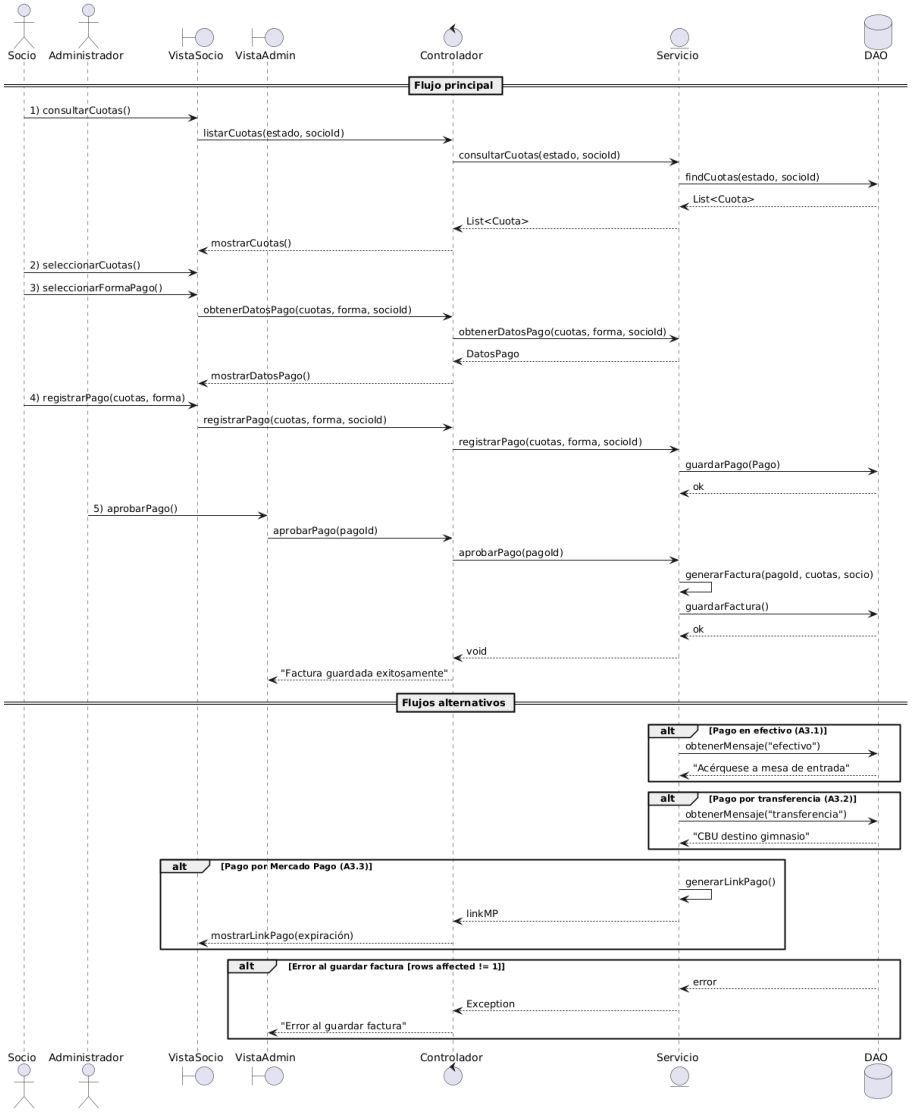

Figura 3: Diagrama de Secuencia de Diseño

Enlace: `https://lucid.app/lucidchart/75a89fc9-ae9e-469b-a08a-86d133243093/edit?view_items=fCKTkNhf-5qu&invitationId=inv_ef5d30df-b4d8-4ee1-bddc-cb6337fbbc2b`

2.4. Prototipado de Interfaz de Usuario

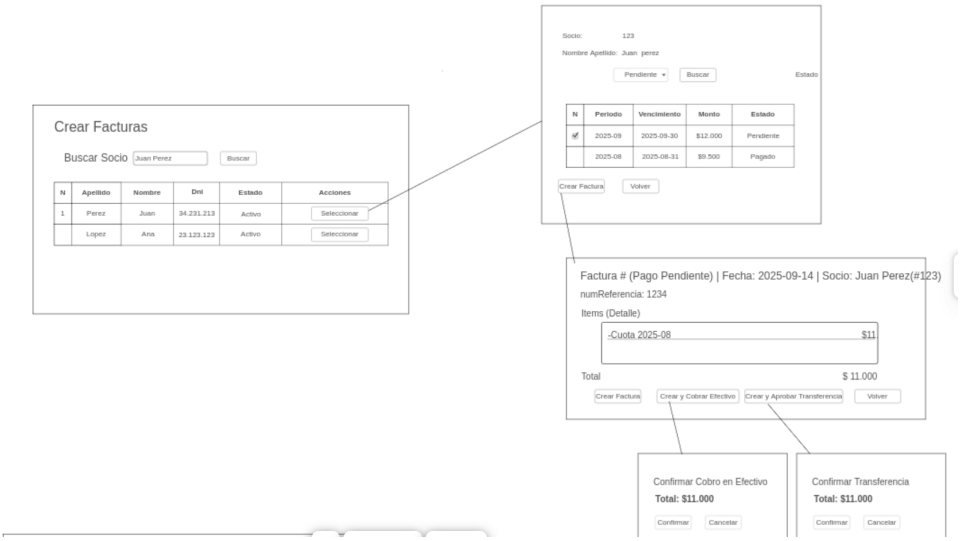

Figura 4: Prototipado 1/3

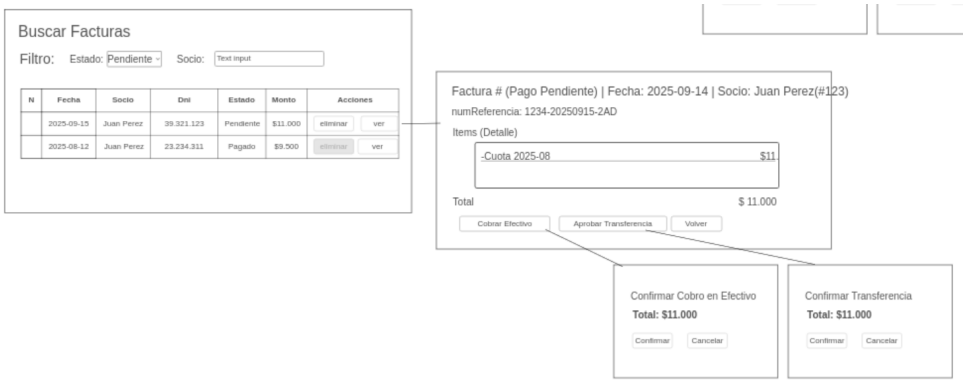

Figura 5: Prototipado 2/3

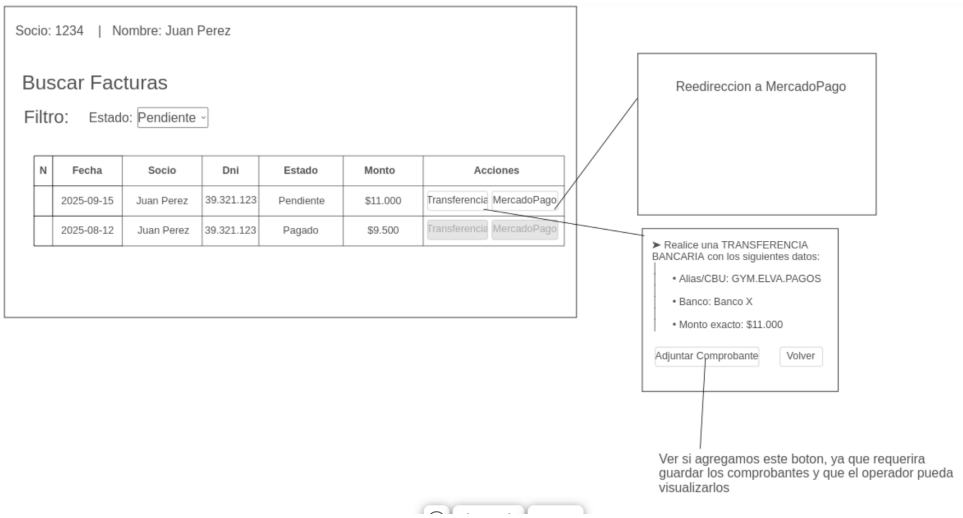

Figura 6: Prototipado 3/3

Enlace: `https://ninjamock.com/s/TVS89Lx`

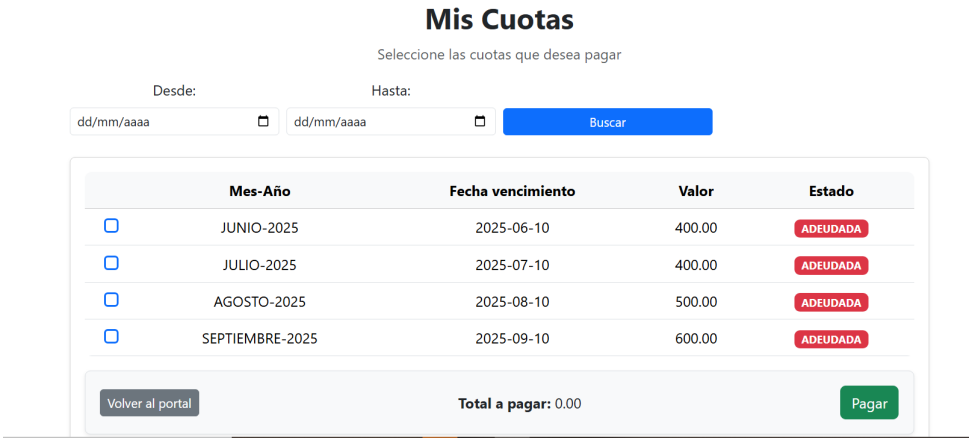

Figura 7: Interfaz Resultante 1/7

Figura 8: Interfaz Resultante 2/7

Figura 9: Interfaz Resultante 3/7

Figura 10: Interfaz Resultante 4/7

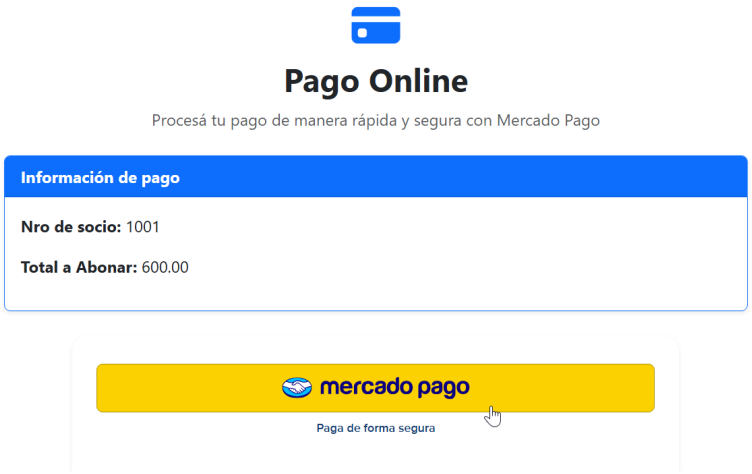

Figura 11: Interfaz Resultante 5/7

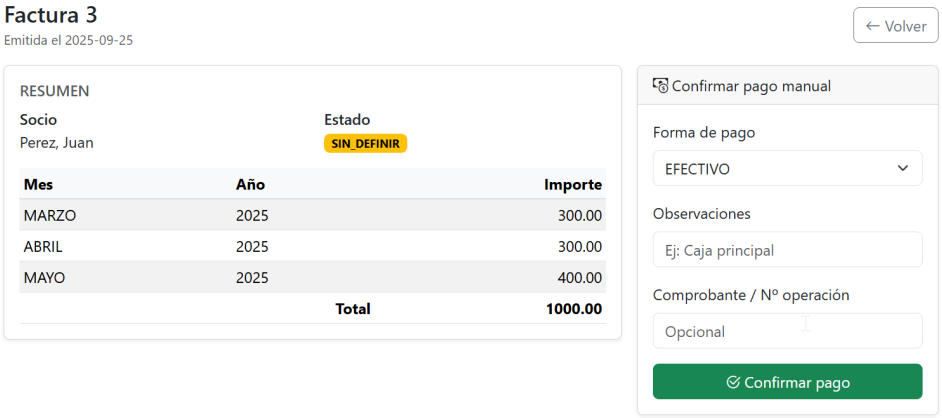

Figura 12: Interfaz Resultante 6/7

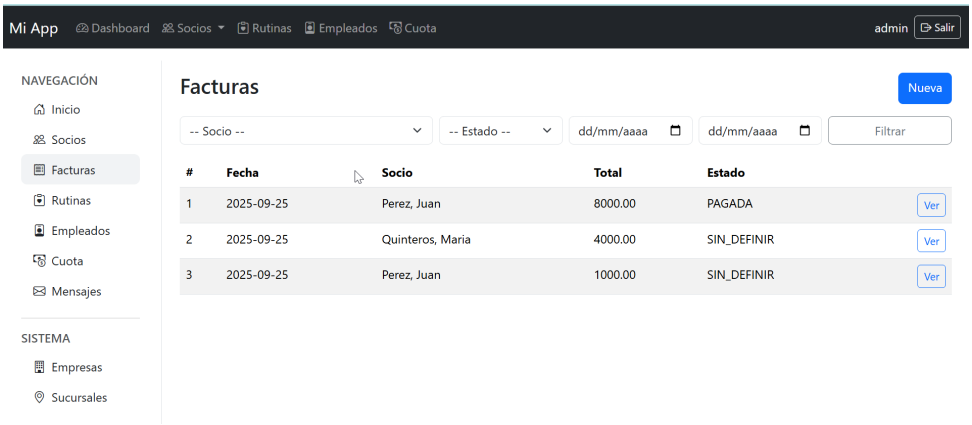

Figura 13: Interfaz Resultante 7/7

3. Diseño del Software

3.1. Diagrama de Clases

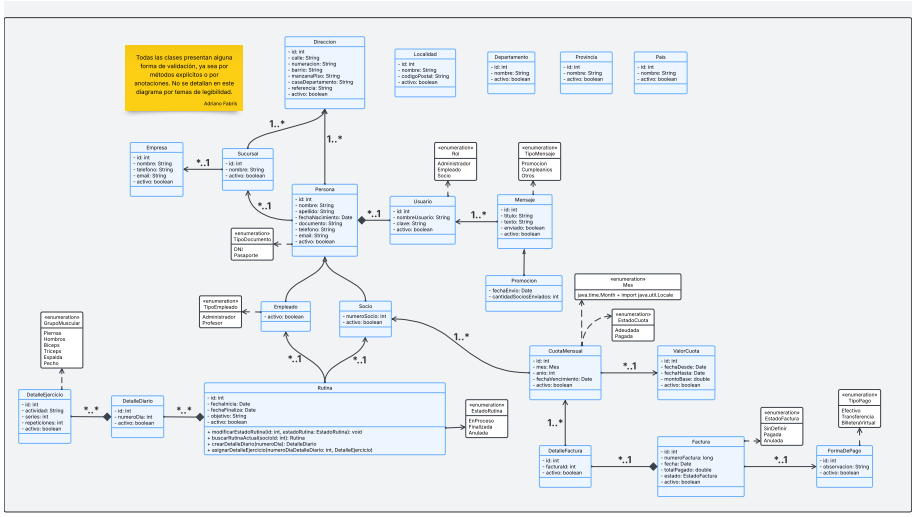

Figura 14: Diagrama de Clases del Dominio - Modelo

Enlace: `https://lucid.app/lucidchart/75a89fc9-ae9e-469b-a08a-86d133243093/edit?view_items=C7BT8wLoO5My&invitationId=inv_ef5d30df-b4d8-4ee1-bddc-cb6337fbbc2b`

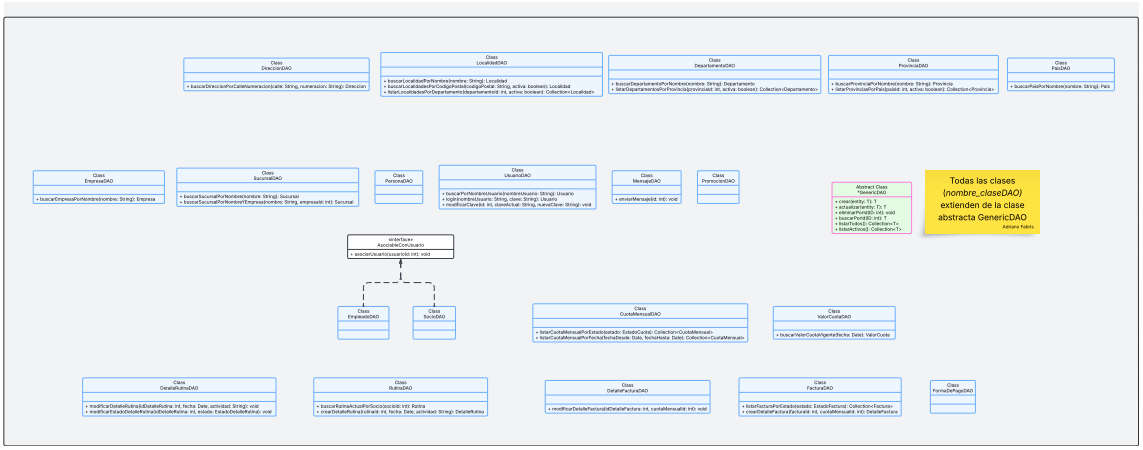

Figura 15: Diagrama de Clases del Dominio - DAO

Enlace: `https://lucid.app/lucidchart/75a89fc9-ae9e-469b-a08a-86d133243093/edit?view_items=p-BTdY7y-Yqd&invitationId=inv_ef5d30df-b4d8-4ee1-bddc-cb6337fbbc2b`

3.2. Diagrama de Paquetes

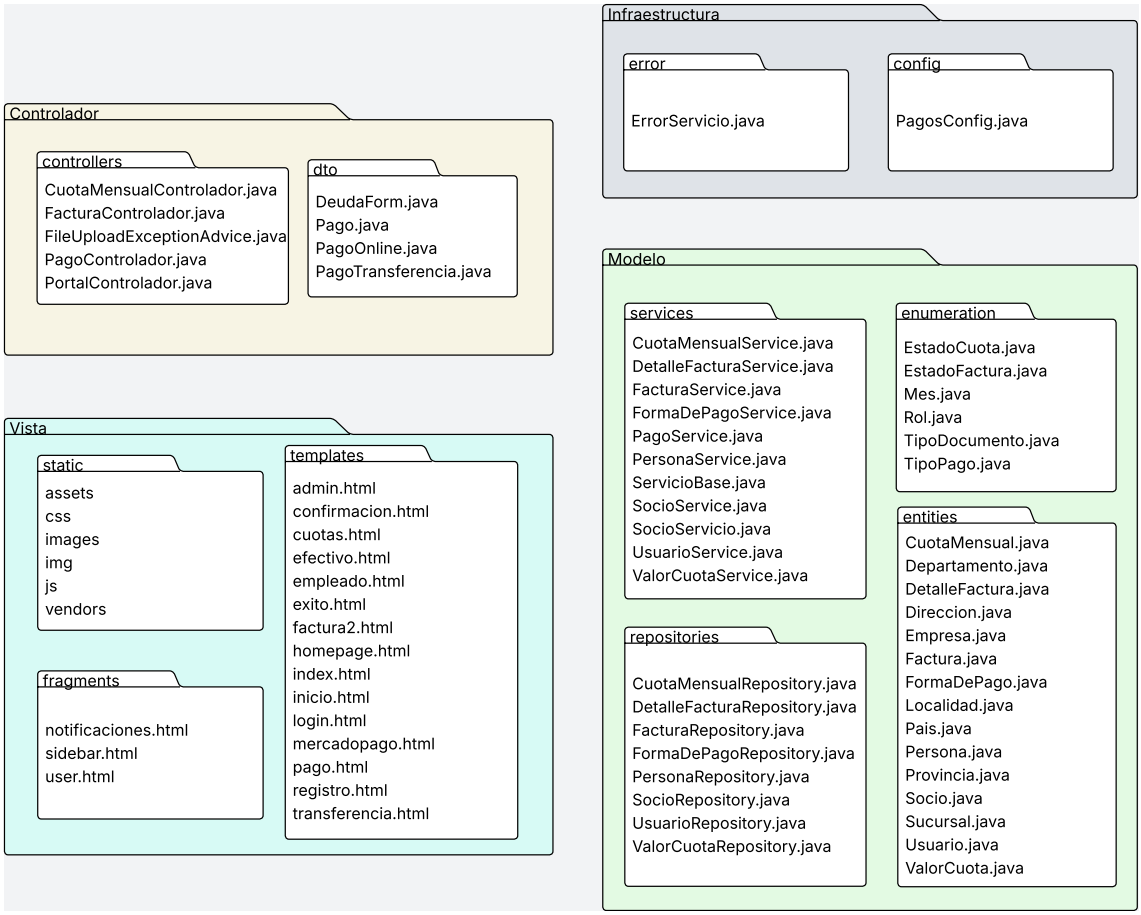

Figura 16: Diagrama de Paquetes

Enlace: `https://lucid.app/lucidchart/75a89fc9-ae9e-469b-a08a-86d133243093/edit?viewport_loc=-2087%2C14800%2C2962%2C2204%2C0_0&invitationId=inv_ef5d30df-b4d8-4ee1-bddc-cb6337fbbc2b`

4. Modelo de Datos

4.1. Diagrama Entidad-Relación

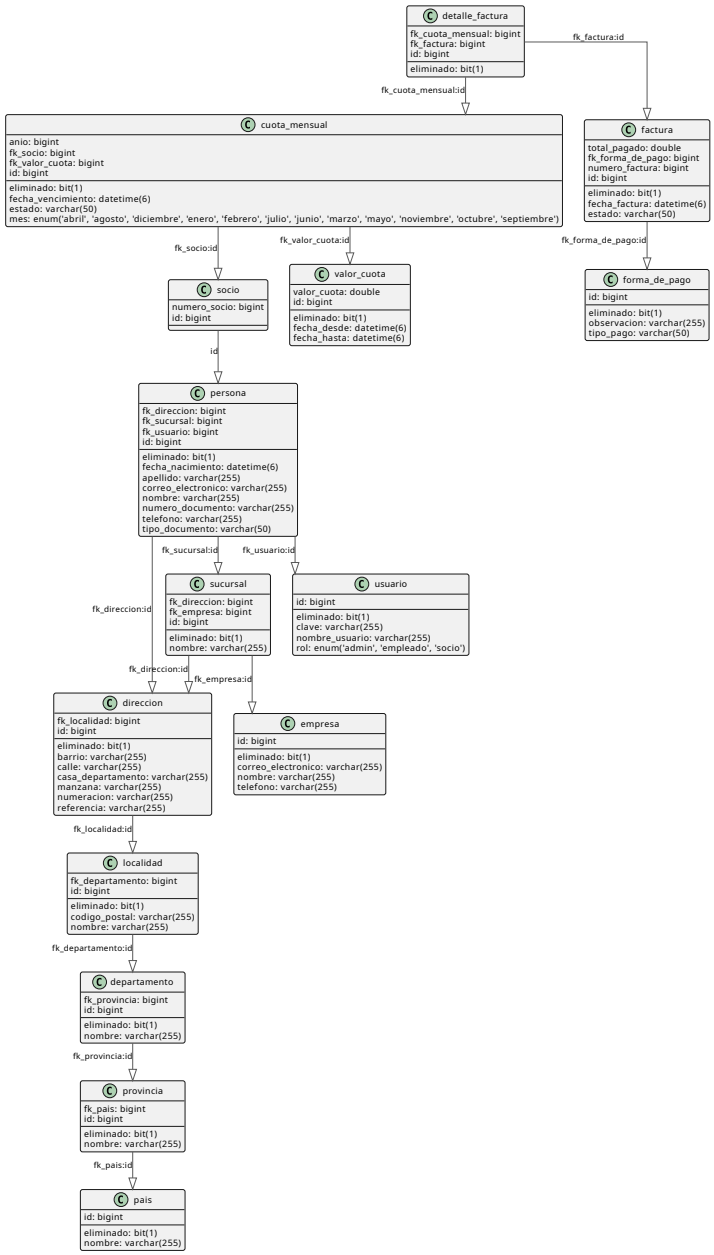

Figura 17: Diagrama Entidad-Relación

Enlace: `https://lucid.app/lucidchart/75a89fc9-ae9e-469b-a08a-86d133243093/edit?view_items=4yKT0Mf.aIp0&invitationId=inv_ef5d30df-b4d8-4ee1-bddc-cb6337fbbc2b`

5. Requisitos

5.1. Requisitos Funcionales

Durante el análisis inicial, a partir del enunciado que nos dieron, detectamos las siguientes funcionalidades que el sistema debía cubrir. Las enumeramos como requisitos funcionales (RF):

RF1: Alta, consulta, modificación y baja de socios/as.

RF2: Alta, consulta, modificación y baja de usuarios del sistema con sus respectivos roles.

RF3: Gestión del cobro de la cuota mensual.

RF4: Gestión de la deuda de los socios/as.

RF5: Generación y envío de campañas promocionales por correo electrónico.

RF6: Envío automático de saludos por cumpleaños a los socios/as.

RF7: Gestión y seguimiento de rutinas de entrenamiento que los profesores entregan a los socios/as.

RF8: Control de perfiles de usuario con distintos niveles de acceso:

- Administrador: acceso total.
- Empleado: acceso casi total, salvo gestión de usuarios y alta de valor de cuota.
- Socio/a: acceso a su rutina, pago de cuota mensual (Mercado Pago) e informe de deuda.

RF9: Visualización de la rutina diaria y posibilidad de marcar si fue completada.

5.2. Requisitos No Funcionales

Por otro lado, los requisitos no funcionales los fuimos definiendo nosotros en base a lo que veíamos necesario:

- **Usabilidad**: incorporamos una interfaz responsive por medio de una plantilla de Bootstrap, esto nos simplificó el desarrollo y permite que el sistema se use tanto en PC como en celular sin problemas.
- **Seguridad**: se incorporó autenticación y autorización mediante acceso protegido por roles (administrador, operador, socio); y el cifrado bcrypt de contraseñas para la protección de datos.
- **Portabilidad**: se hizo uso de ORM para facilitar migraciones hacia distintas bases de datos.
- **Rendimiento**: por el momento no se efectaron pruebas de carga, sin embargo el sistema responde rápido en los casos que probamos (pago, consultas, alta de socio). Lo mínimo que pedimos es que las operaciones comunes respondan "en un tiempo razonable".
- **Mantenibilidad y Escalabilidad**: intentamos que el diseño aguante crecimiento. Un ejemplo es la herencia entre Usuario, Socio y Empleado, que permitiría agregar más tipos de usuarios si el gimnasio crece. Un ejemplo de mantenibilidad se presenta en el módulo de domicilio (país, provincia, localidad, dirección), que hace más fácil escalar a distintos lugares.

6. Patrones de Diseño

En esta sección describimos los patrones que aplicamos en nuestro proyecto. Primero damos una breve definición del patrón a modo de repaso teórico.

6.1. Patrones GRASP: General Responsibility Assignment *Software Principles*

- **Information Expert**: Este patrón dice que la responsabilidad de manejar cierta información debe estar en la clase que realmente conoce esos datos. En nuestro caso, la clase `Factura` maneja su número, fecha y monto, y la clase `CuotaMensual` maneja mes, año y estado. Así evitamos tener datos desperdigados.
- **Controller**: Según GRASP, el Controller coordina la interacción entre la interfaz de usuario y la lógica del sistema. Nosotros lo aplicamos con controladores web como `PagoControlador` y `FacturaControlador`, que reciben las solicitudes de la vista y llaman a los servicios necesarios.
- **Creator**: Este patrón dice que una clase debería ser la responsable de crear objetos de otra clase si tiene la información necesaria. Un ejemplo en nuestro sistema es `PersonaService`, que crea instancias de Persona o Socio porque ya conoce los datos requeridos y sabe cómo persistirlos.
- **Low Coupling y High Cohesion**: Estos dos patrones van de la mano. Bajo acoplamiento significa que las clases dependan lo menos posible de otras, y alta cohesión significa que cada clase tenga un foco claro en su tarea. En nuestro sistema lo logramos separando las responsabilidades en servicios, repositorios y controladores, y usando inyección de dependencias con Spring.
- **Pure Fabrication**: Es un patrón que sugiere inventar clases que no existen en el dominio real, pero que ayudan a organizar el sistema. Nosotros lo aplicamos al crear clases como `PagoService`, `PersonaService` (servicios en general), y las clases incluídas en el paquete de DTO, como el formulario de deuda `DeudaForm`, que no son conceptos del gimnasio, pero sí sirven para mantener el código ordenado.

6.2. Patrones GoF: Gang of Four - *Software Desing Patterns*

- **Builder**: Es un patrón de creación que permite armar objetos complejos paso a paso, sin necesidad de un constructor gigante. En nuestro caso lo usamos gracias a Lombok, que genera el builder automáticamente. Por ejemplo, para crear un `PagoOnline` usamos la sintaxis `PagoOnline.builder()...build()`.
- **Facade**: Este patrón propone tener una clase que funcione como "fachada", ofreciendo una interfaz simple para operaciones que por dentro son complejas. En nuestro sistema, `PagoService` es un buen ejemplo, porque desde afuera parece un único servicio, pero internamente coordina cuotas, facturas y repositorios.
- **Singleton**: Garantiza que una clase tenga una única instancia en todo el sistema. Nosotros no lo programamos manualmente, pero en Spring los servicios y repositorios anotados con `@Service` y `@Repository` ya funcionan como singletons automáticamente.
- **Adapter**: Permite que dos interfaces incompatibles trabajen juntas. En nuestro caso se da con Spring Data JPA: los repositorios adaptan operaciones de alto nivel (findAll, save, etc.) a consultas SQL, sin que nosotros veamos el SQL.
- **Strategy (posible mejora)**: Strategy define una familia de algoritmos y permite intercambiarlos fácilmente. En nuestro proyecto lo pensamos como una mejora para los distintos tipos de pago: podríamos tener estrategias separadas para efectivo, transferencia o Mercado Pago, en lugar de un enum.
- **State (posible mejora)**: Este patrón encapsula los distintos estados de un objeto y sus transiciones. En el sistema tenemos enums para `EstadoCuota` y `EstadoFactura`. Por ahora solo guardan el estado, pero si en el futuro hay reglas de transición más complejas, podrían evolucionar a un State.

7. Nueva Funcionalidad: Sistema de Recompensas

Como parte del trabajo, incorporamos un módulo adicional al sistema: un **sistema de recompensas por puntos**. La idea surge de la necesidad de motivar a los socios y promover la fidelización dentro del gimnasio. El funcionamiento es sencillo: los socios acumulan puntos al realizar distintas acciones, como asistir regularmente, pagar sus cuotas en término o invitar a nuevos socios. Luego, estos puntos pueden canjearse por descuentos en la cuota, productos promocionales o beneficios especiales.

Con esta funcionalidad se busca no solo mejorar la experiencia del usuario, sino también generar un incentivo para que los socios permanezcan activos y colaboren con el crecimiento del gimnasio. Además, el sistema es flexible y podría ampliarse en el futuro con nuevas reglas o tipos de recompensas.

7.1. Diagrama de Clases del Módulo

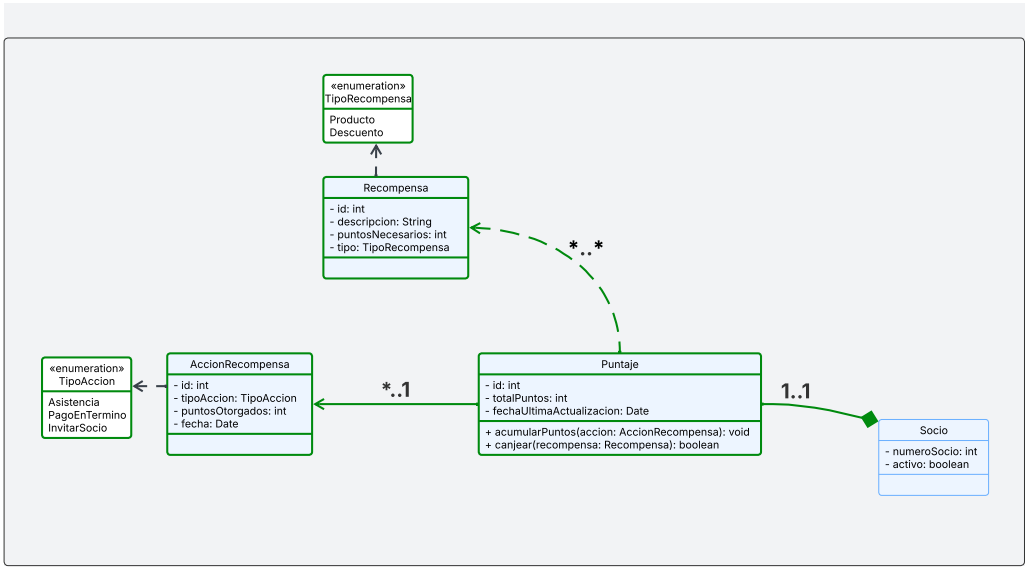

Figura 18: Diagrama de Clases de la Nueva Funcionalidad

Enlace: `https://lucid.app/lucidchart/75a89fc9-ae9e-469b-a08a-86d133243093/edit?view_items=5YJTan_uGaTs&invitationId=inv_ef5d30df-b4d8-4ee1-bddc-cb6337fbbc2b`

8. Relaciones y Justificación

En el módulo de recompensas se definieron las siguientes relaciones principales:

- **Socio – Puntaje:** asociación 1 a 1, cada socio posee un único puntaje acumulado.
- **Puntaje – AccionRecompensa:** asociación 1 a N, un puntaje se forma a partir de múltiples acciones realizadas por el socio.
- **Puntaje – Recompensa:** relación de dependencia, ya que sólo interviene al momento del canje de una recompensa.
- **Enumeraciones:** *TipoRecompensa* y *TipoAccion* funcionan como dependencias de *Recompensa* y *AccionRecompensa*, tipificando su comportamiento.

9. Conclusión

Este trabajo nos permitió experimentar con un ciclo completo de desarrollo mediante trabajo en equipo. Una cosa que aprendimos es que los diagramas iniciales sirven de guía, pero después en la implementación cambian varias cosas, y está bien que pase así porque el software evoluciona.

En cuanto a la forma de trabajo, nos dividimos en dos grupos: unos se enfocaron en la implementación y otros en la documentación. De todas formas la división nunca fue total: los que programaban consultaban la documentación y los que documentaban también ayudaban a probar o entender la implementación para basar en ellas la documentación o los futuros diagramas. Eso hizo que el trabajo se mantuviera coordinado y que todos tuviéramos una idea general del sistema.

En resumen, fue un proyecto que nos ayudó a poner en práctica conceptos de la materia (como diagramas UML, requisitos, patrones de diseño, código) y también a organizarnos como equipo de desarrollo por primera vez.