Universidad Nacional de Cuyo
Septiembre 2025

Integrador Parte N°1
Software para Gimnasio

Grupo 2: ElvaStudio

- Trinidad Perea
- Perla Valerio
- Giovanny Azurduy
- Adriano Fabris

ELVA

← QR al repositorio GitHub: código + informe con links hacia la documentación

INDICE

Introducción

Desarrollo de un sistema de gestión para el gimnasio “Sport” (Mendoza). Administra socios, usuarios, cuotas, facturas, deudas y rutinas.

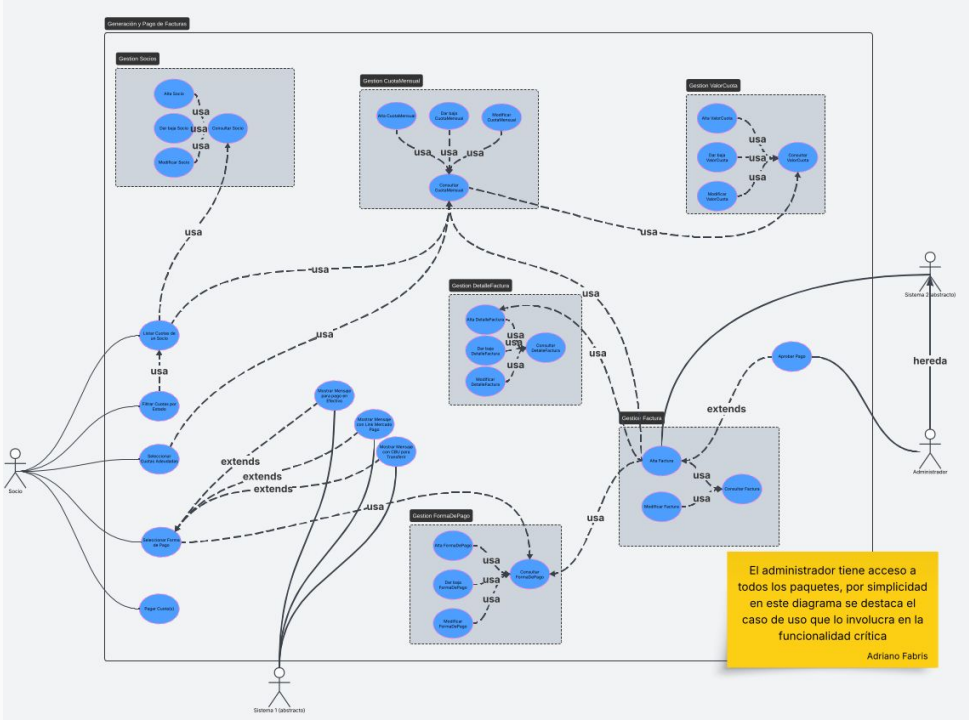
Se agregó un módulo de recompensas para motivar la fidelización.

Caso de Uso Crítico

“Pago de Cuotas con Generación de Factura y Método de Pago”

Diagrama de Caso de Uso

Veámoslo parte por parte, en vivo y en directo

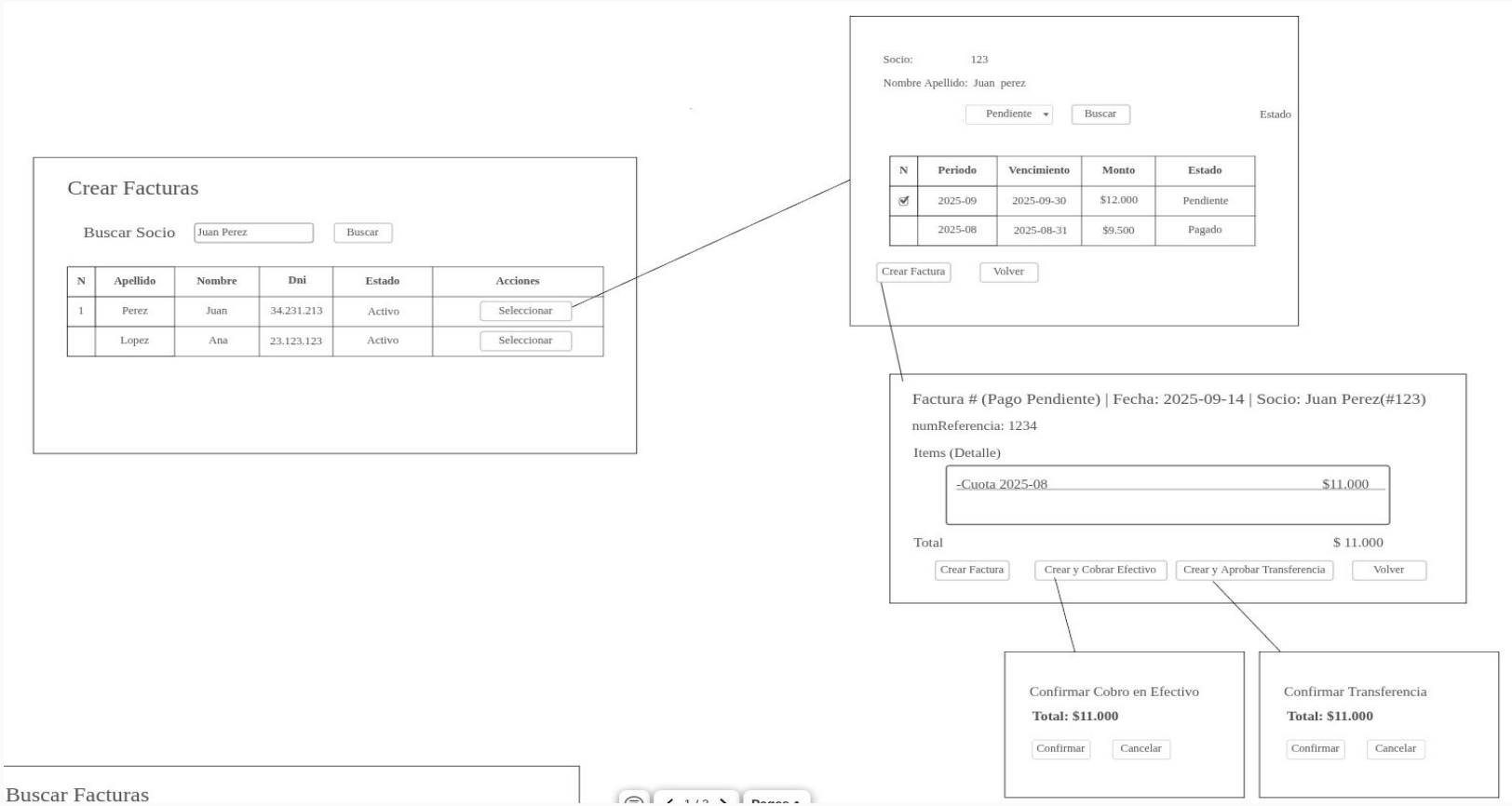


Escenario de Caso de Uso

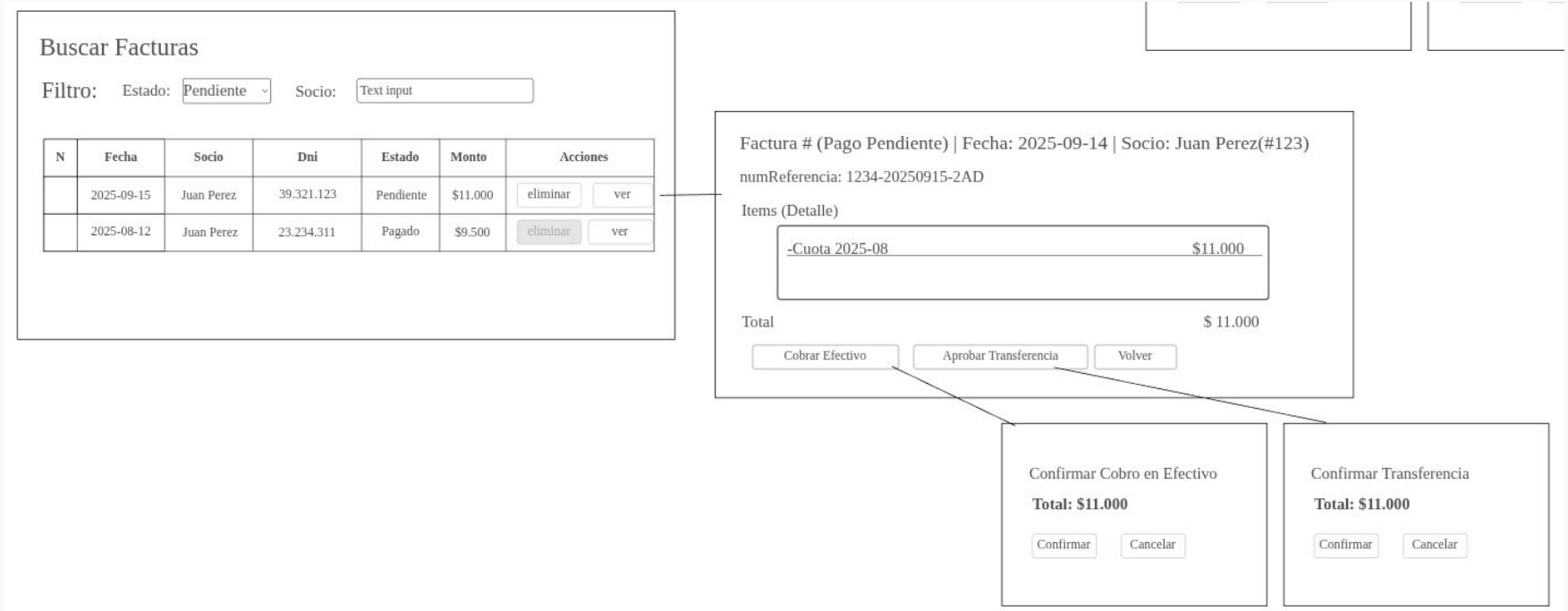
ESCENARIO DE CASO DE USO #1	
Prioridad	9/10 (muy alta)
Nombre	Pago de cuota con su generación de factura y forma de pago
Descripción	Un socio quiere pagar una o más cuotas
Actores principales	Socio Administrador
Actores secundarios (si aplica)	Sistema (abstracto)
Precondiciones	El socio debe existir, 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	-

Prototipado de la UI

Caso de uso crítico



Caso de uso crítico



Caso de uso crítico

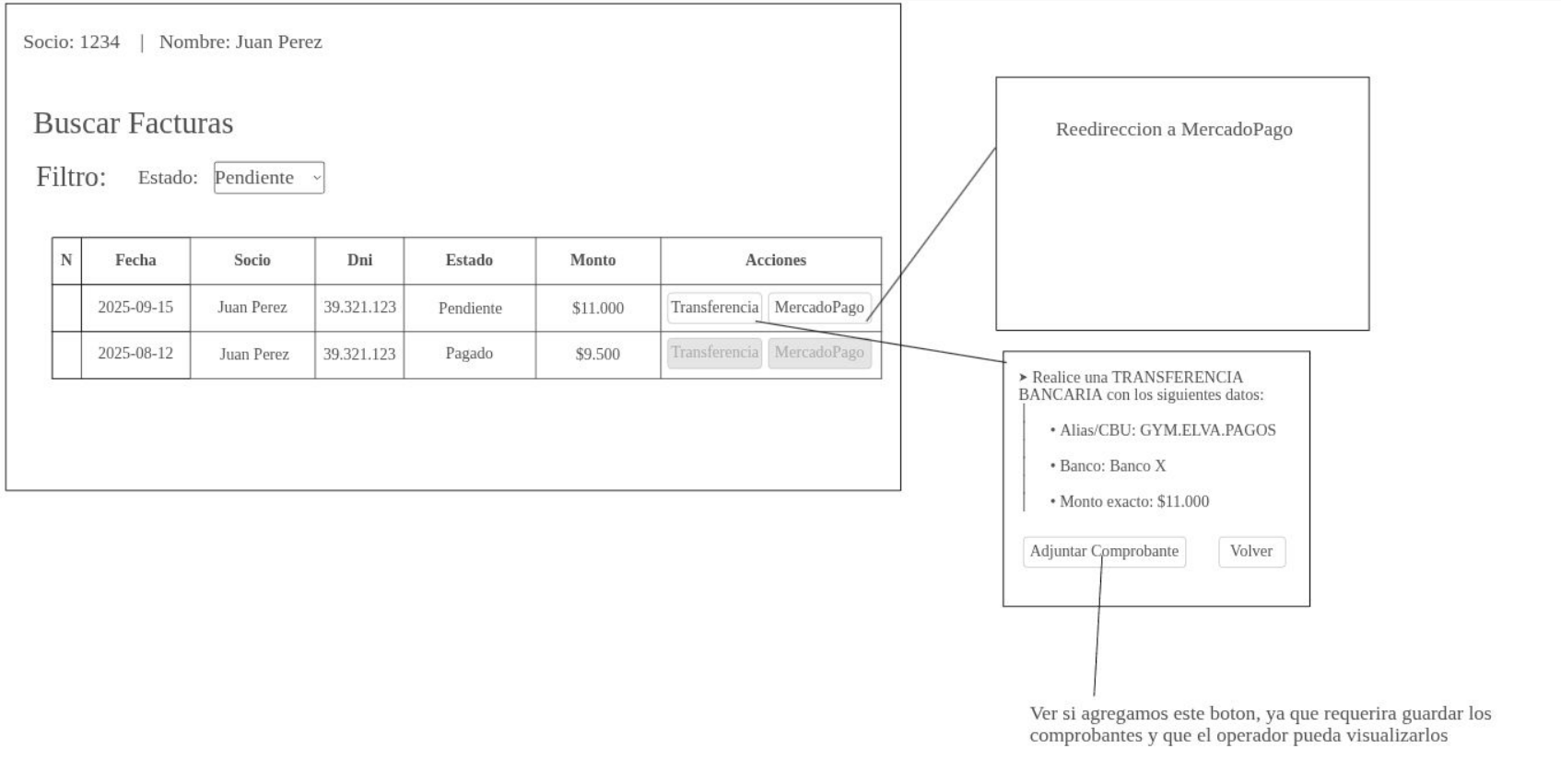
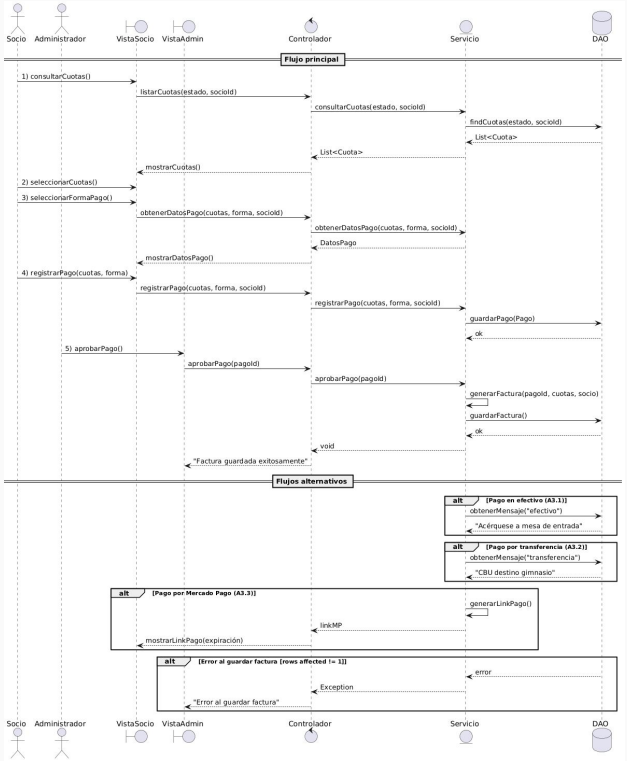
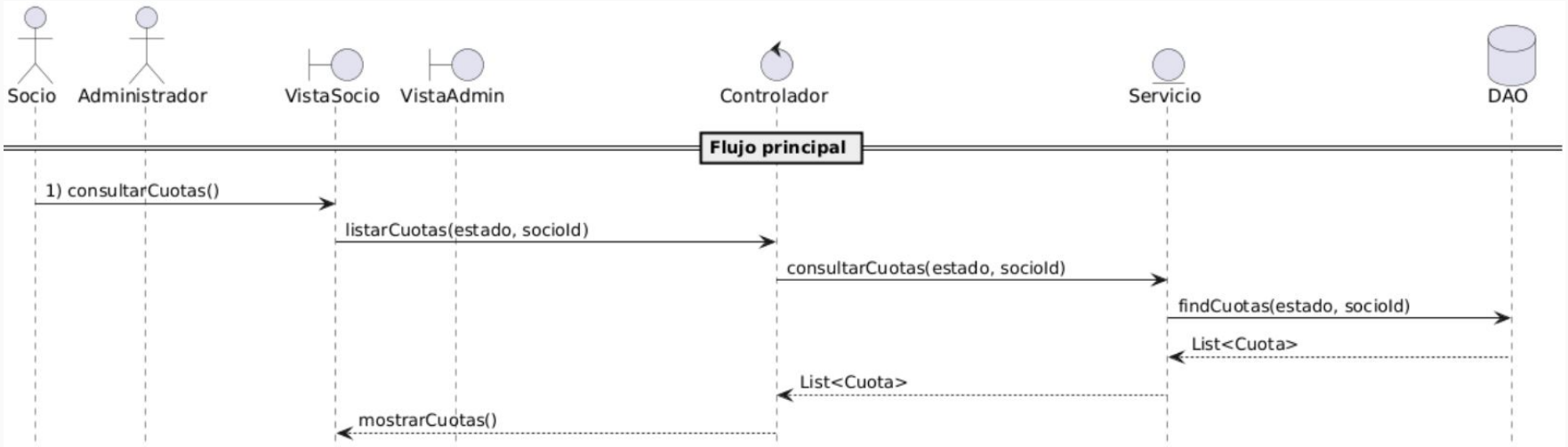


Diagrama de Secuencia

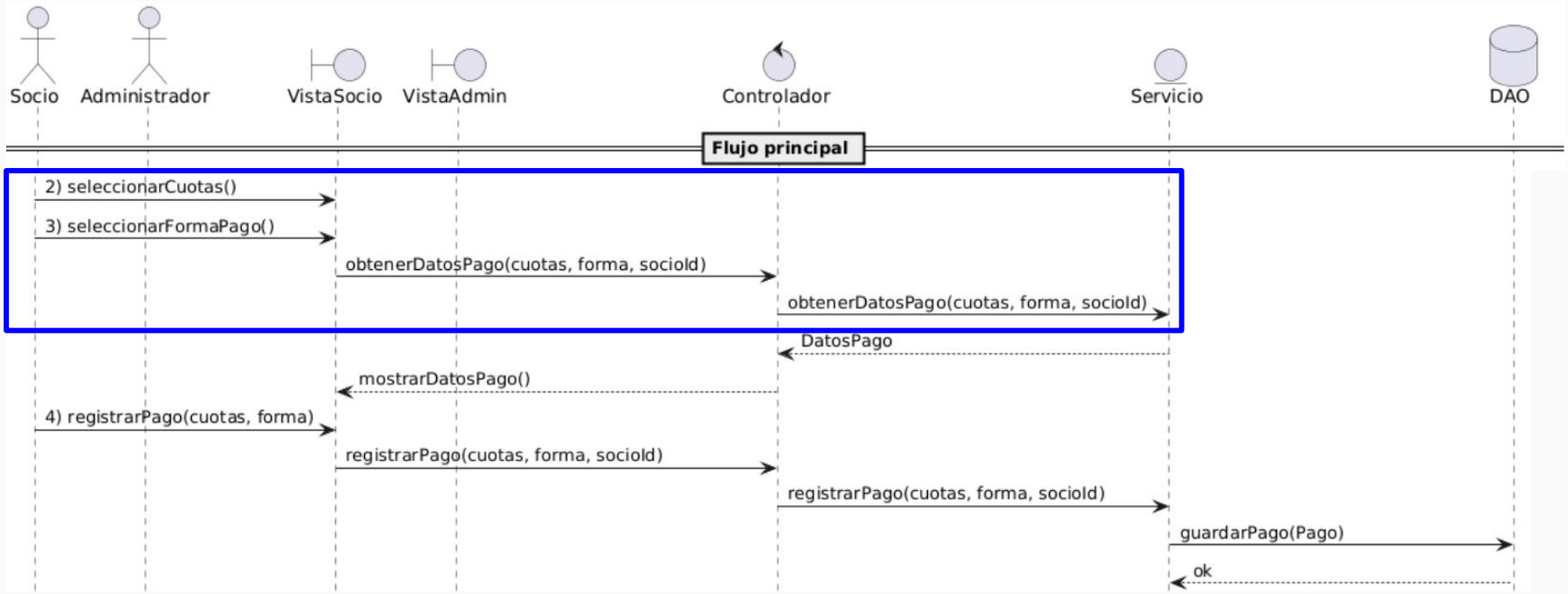
De nuevo, veámoslo de a partes...



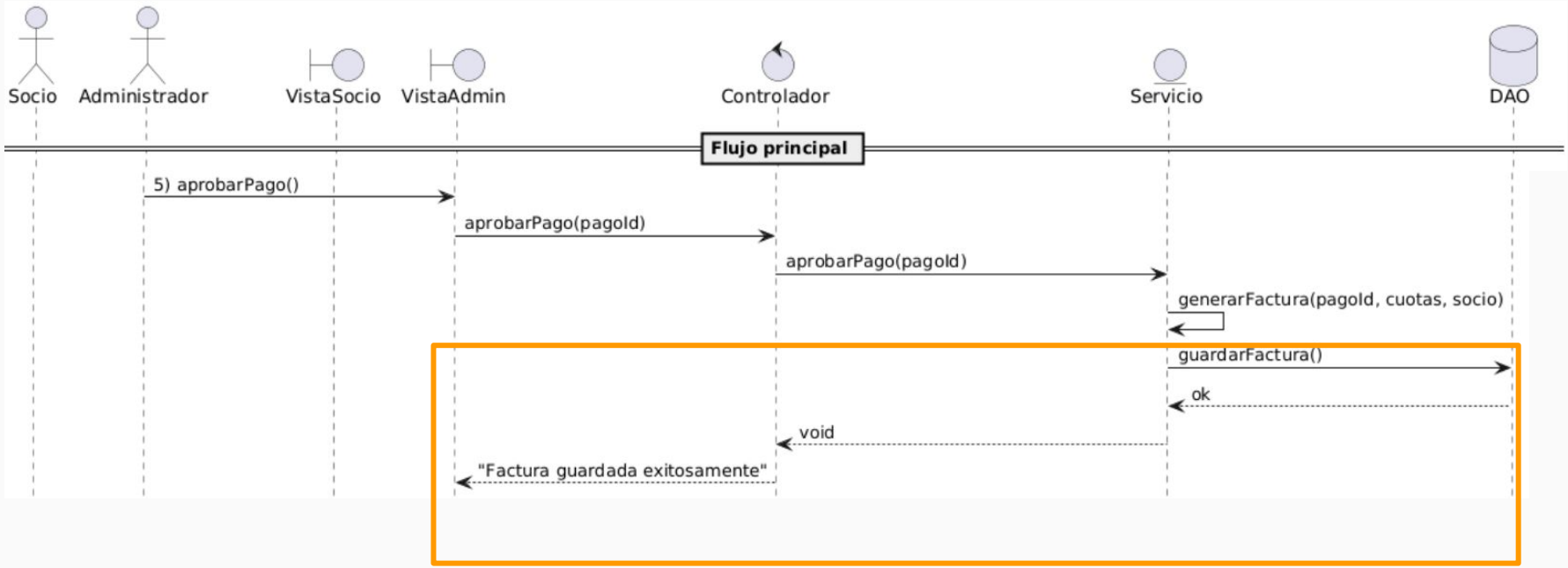
Caso de uso crítico



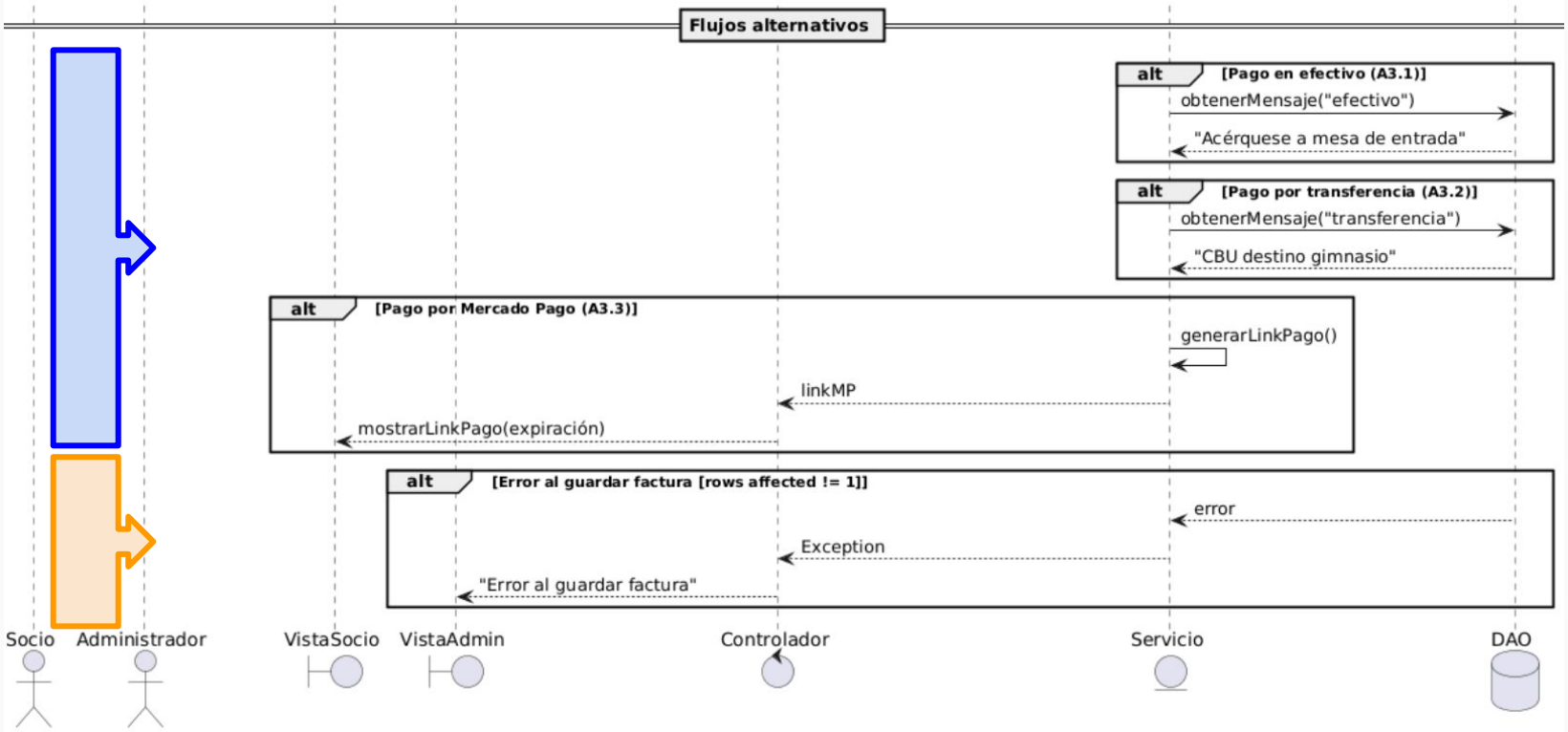
Caso de uso crítico



Caso de uso crítico



Caso de uso crítico

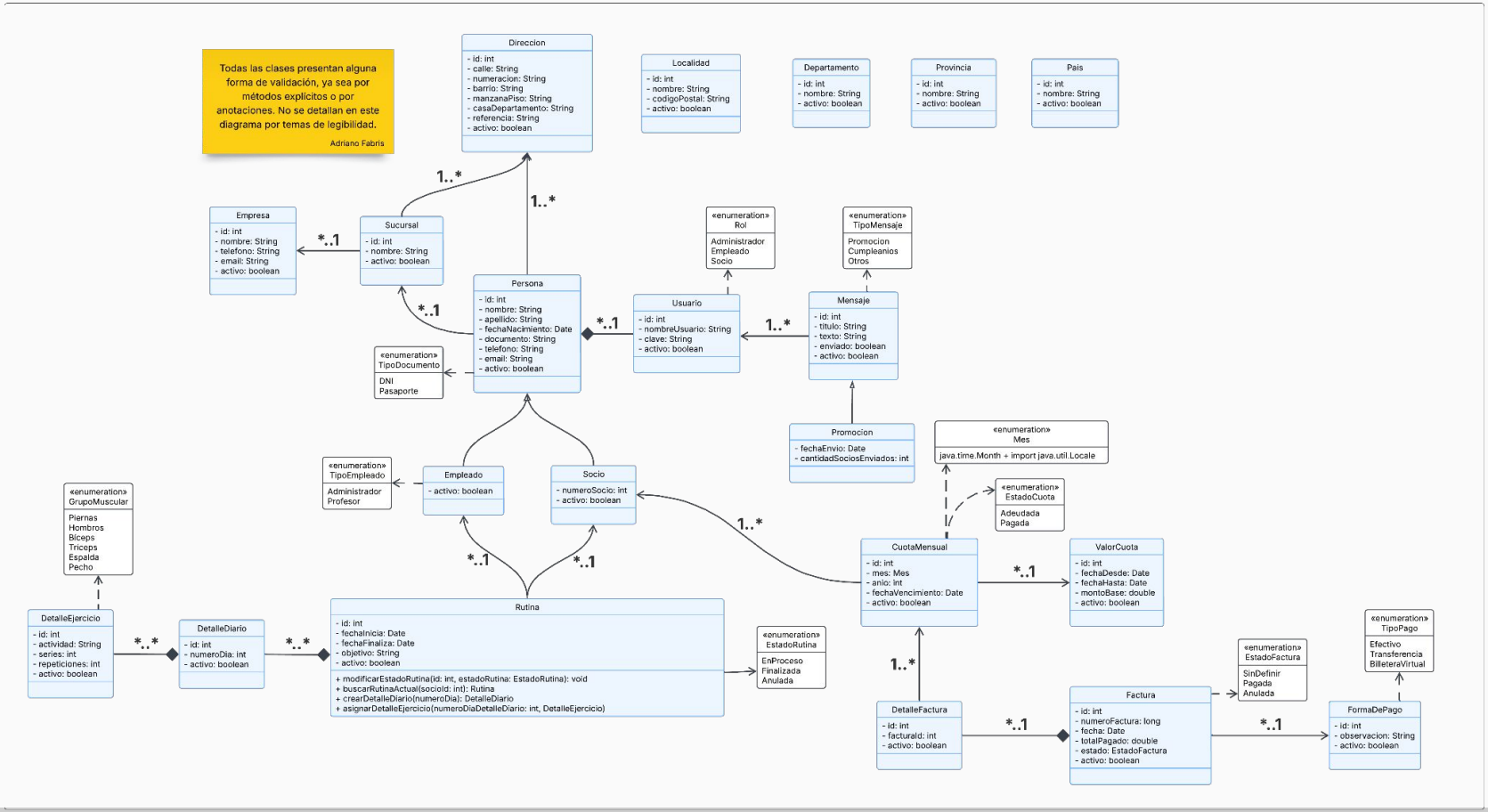


Diseño del Software

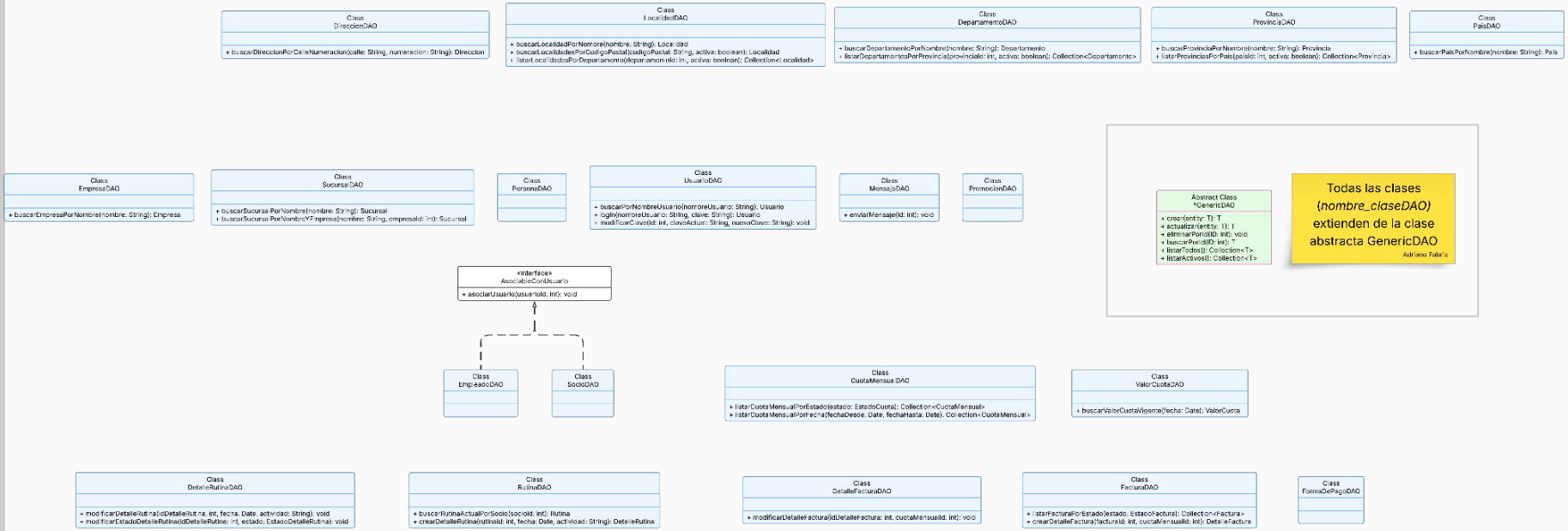
A modo de diferenciar las Entidades de sus Servicios y por cuestiones de legibilidad, separamos el diagrama en 2 capas:

1) Entidad
2) DAO

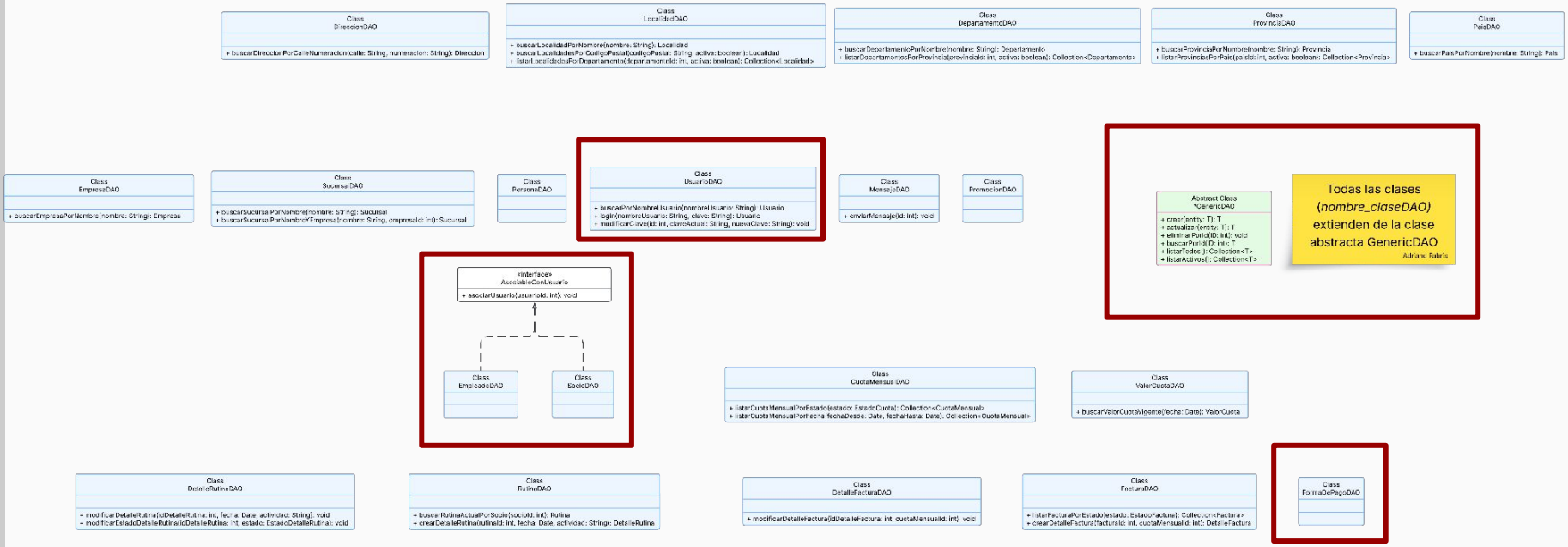
Entidad

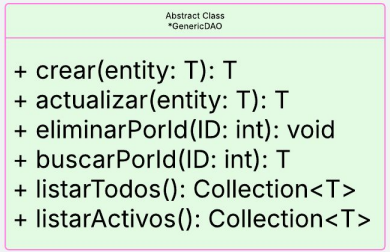

Todas las clases presentan alguna forma de validación, ya sea por métodos explícitos o por anotaciones. No se detallan en este diagrama por temas de legibilidad.
Adriano Fabris
Direccion
- id: int
- calle: String
- numeracion: String
- barrio: String
- manzanaPiso: String
- casaDepartamento: String
- referencia: String
- activo: boolean
Localidad
- id: int
- nombre: String
- codigoPostal: String
- activo: boolean
Departamento
- id: int
- nombre: String
- activo: boolean
Provincia
- id: int
- nombre: String
- activo: boolean
Pais
- id: int
- nombre: String
- activo: boolean
Empresa
- id: int
- nombre: String
- telefono: String
- email: String
- activo: boolean
Sucursal
- id: int
- nombre: String
- activo: boolean
Persona
- id: int
- nombre: String
- apellido: String
- fechaNacimiento: Date
- documento: String
- telefono: String
- email: String
- activo: boolean
«enumeration» Rol
Administrador
Empleado
Socio
«enumeration» TipoMensaje
Promocion
Cumpleanios
Otros
Usuario
- id: int
- nombreUsuario: String
- clave: String
- activo: boolean
Mensaje
- id: int
- titulo: String
- texto: String
- enviado: boolean
- activo: boolean
«enumeration» TipoDocumento
DNI
Pasaporte
Promocion
- fechaEnvio: Date
- cantidadSociosEnviados: int
«enumeration» Mes
java.time.Month + import java.util.Locale
«enumeration» TipoEmpleado
Administrador
Profesor
Empleado
- activo: boolean
Socio
- numeroSocio: int
- activo: boolean
«enumeration» EstadoCuota
Adeudada
Pagada
«enumeration» GrupoMuscular
Piernas
Hombros
Biceps
Triceps
Espalda
Pecho
CuotaMensual
- id: int
- mes: Mes
- anio: int
- fechaVencimiento: Date
- activo: boolean
ValorCuota
- id: int
- fechaDesde: Date
- fechaHasta: Date
- montoBase: double
- activo: boolean
DetalleEjercicio
- id: int
- actividad: String
- series: int
- repeticiones: int
- activo: boolean
DetalleDiario
- id: int
- numeroDia: int
- activo: boolean
Rutina
- id: int
- fechaInicia: Date
- fechaFinaliza: Date
- objetivo: String
- activo: boolean
+ modificarEstadoRutina(id: int, estadoRutina: EstadoRutina): void
+ buscarRutinaActual(socioId: int): Rutina
+ crearDetalleDiario(numeroDia): DetalleDiario
+ asignarDetalleEjercicio(numeroDiaDetalleDiario: int, DetalleEjercicio)
«enumeration» EstadoRutina
EnProceso
Finalizada
Anulada
«enumeration» EstadoFactura
SinDefinir
Pagada
Anulada
«enumeration» TipoPago
Efectivo
Transferencia
BilleteraVirtual
DetalleFactura
- id: int
- facturaId: int
- activo: boolean
Factura
- id: int
- numeroFactura: long
- fecha: Date
- totalPagado: double
- estado: EstadoFactura
- activo: boolean
FormaDePago
- id: int
- observacion: String
- activo: boolean
1..*
*..1
..

DAO


Class
DireccionDAO
+ buscarDireccionPorCalleNumeracion(calle: String, numeracion: String): Direccion
Class
LocalidadDAO
+ buscarLocalidadPorNombre(nombre: String): Localidad
+ buscarLocalidadesPorCodigoPostal(codigoPostal: String, activa: boolean): Localidad
+ listarLocalidadesPorDepartamento(departamentoId: int, activa: boolean): Collection<Localidad>
Class
DepartamentoDAO
+ buscarDepartamentoPorNombre(nombre: String): Departamento
+ listarDepartamentosPorProvincia(provinciaId: int, activa: boolean): Collection<Departamento>
Class
ProvinciaDAO
+ buscarProvinciaPorNombre(nombre: String): Provincia
+ listarProvinciasPorPais(paisId: int, activa: boolean): Collection<Provincia>
Class
PaisDAO
+ buscarPaisPorNombre(nombre: String): Pais
Class
EmpresaDAO
+ buscarEmpresaPorNombre(nombre: String): Empresa
Class
SucursalDAO
+ buscarSucursalPorNombre(nombre: String): Sucursal
+ buscarSucursalPorNombreYEmpresa(nombre: String, empresaId: int): Sucursal
Class
PersonaDAO
Class
UsuarioDAO
+ buscarPorNombreUsuario(nombreUsuario: String): Usuario
+ login(nombreUsuario: String, clave: String): Usuario
+ modificarClave(id: int, claveActual: String, nuevaClave: String): void
Class
MensajeDAO
+ enviarMensaje(id: int): void
Class
PromocionDAO
Abstract Class
*GenericDAO
+ crear(entity: T): T
+ actualizar(entity: T): T
+ eliminarPorId(ID: int): void
+ buscarPorId(ID: int): T
+ listarTodos(): Collection<T>
+ listarActivos(): Collection<T>
Todas las clases
(nombre_claseDAO)
extienden de la clase
abstracta GenericDAO
Adriano Fabris
«interface»
AsociableConUsuario
+ asociarUsuario(usuarioId: int): void
Class
EmpleadoDAO
Class
SocioDAO
Class
CuotaMensualDAO
+ listarCuotaMensualPorEstado(estado: EstadoCuota): Collection<CuotaMensual>
+ listarCuotaMensualPorFecha(fechaDesde: Date, fechaHasta: Date): Collection<CuotaMensual>
Class
ValorCuotaDAO
+ buscarValorCuotaVigente(fecha: Date): ValorCuota
Class
DetalleRutinaDAO
+ modificarDetalleRutina(idDetalleRutina: int, fecha: Date, actividad: String): void
+ modificarEstadoDetalleRutina(idDetalleRutina: int, estado: EstadoDetalleRutina): void
Class
RutinaDAO
+ buscarRutinaActualPorSocio(socioId: int): Rutina
+ crearDetalleRutina(rutinaId: int, fecha: Date, actividad: String): DetalleRutina
Class
DetalleFacturaDAO
+ modificarDetalleFactura(idDetalleFactura: int, cuotaMensualId: int): void
Class
FacturaDAO
+ listarFacturaPorEstado(estado: EstadoFactura): Collection<Factura>
+ crearDetalleFactura(facturaId: int, cuotaMensualId: int): DetalleFactura
Class
FormaDePagoDAO

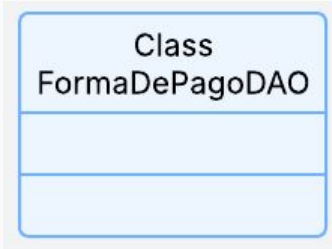
DAO

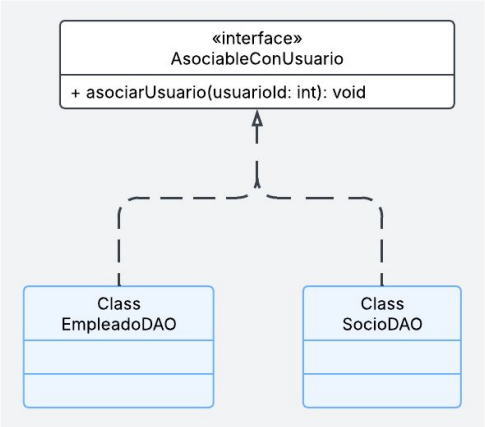

Class
DireccionDAO
+ buscarDireccionPorCalleNumeracion(calle: String, numeracion: String): Direccion
Class
LocalidadDAO
+ buscarLocalidadPorNombre(nombre: String): Localidad
+ buscarLocalidadesPorCodigoPostal(codigoPostal: String, activa: boolean): Localidad
+ listarLocalidadesPorDepartamento(departamentoId: in, activa: boolean): Collection<Localidad>
Class
DepartamentoDAO
+ buscarDepartamentoPorNombre(nombre: String): Departamento
+ listarDepartamentosPorProvincia(provinciaId: int, activa: boolean): Collection<Departamento>
Class
ProvinciaDAO
+ buscarProvinciaPorNombre(nombre: String): Provincia
+ listarProvinciasPorPais(paisId: int, activa: boolean): Collection<Provincia>
Class
PaisDAO
+ buscarPaisPorNombre(nombre: String): Pais
Class
EmpresaDAO
+ buscarEmpresaPorNombre(nombre: String): Empresa
Class
SucursalDAO
+ buscarSucursalPorNombre(nombre: String): Sucursal
+ buscarSucursalPorNombreYEmpresa(nombre: String, empresaId: Int): Sucursal
Class
PersonaDAO
Class
UsuarioDAO
+ buscarPorNombreUsuario(nombreUsuario: String): Usuario
+ login(nombreUsuario: String, clave: String): Usuario
+ modificarClave(id: Int, claveActual: String, nuevaClave: String): void
Class
MensajeDAO
+ enviarMensaje(id: Int): void
Class
PromocionDAO
Abstract Class
*GenericDAO
+ crear(entity: T): T
+ actualizar(entity: T): T
+ eliminarPorId(ID: Int): void
+ buscarPorId(ID: Int): T
+ listarTodos(): Collection<T>
+ listarActivos(): Collection<T>
Todas las clases
(nombre_claseDAO)
extienden de la clase
abstracta GenericDAO
Adriano Fabris
«interface»
AsociableConUsuario
+ asociarUsuario(usuarioId, Int): void
Class
EmpleadoDAO
Class
SocioDAO
Class
CuotaMensualDAO
+ listarCuotaMensualPorEstado(estado: EstadoCuota): Collection<CuotaMensual>
+ listarCuotaMensualPorFecha(fechaDesde: Date, fechaHasta: Date): Collection<CuotaMensual>
Class
ValorCuotaDAO
+ buscarValorCuotaVigente(fecha: Date): ValorCuota
Class
DetalleRutinaDAO
+ modificarDetalleRutina(idDetalleRutina: int, fecha: Date, actividad: String): void
+ modificarEstadoDetalleRutina(idDetalleRutina: int, estado: EstadoDetalleRutina): void
Class
RutinaDAO
+ buscarRutinaActualPorSocio(socioId, int): Rutina
+ crearDetalleRutina(rutinaId: Int, fecha: Date, actividad: String): DetalleRutina
Class
DetalleFacturaDAO
+ modificarDetalleFactura(idDetalleFactura: int, cuotaMensualId: Int): void
Class
FacturaDAO
+ listarFacturaPorEstado(estado: EstadoFactura): Collection<Factura>
+ crearDetalleFactura(facturaId: Int, cuotaMensualId: Int): DetalleFactura
Class
FormaDePagoDAO

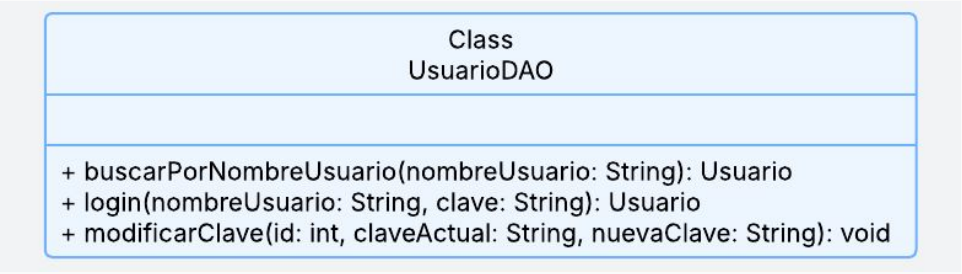


Todas las clases (*nombre_claseDAO)* extienden de la clase abstracta GenericDAO

Adriano Fabris



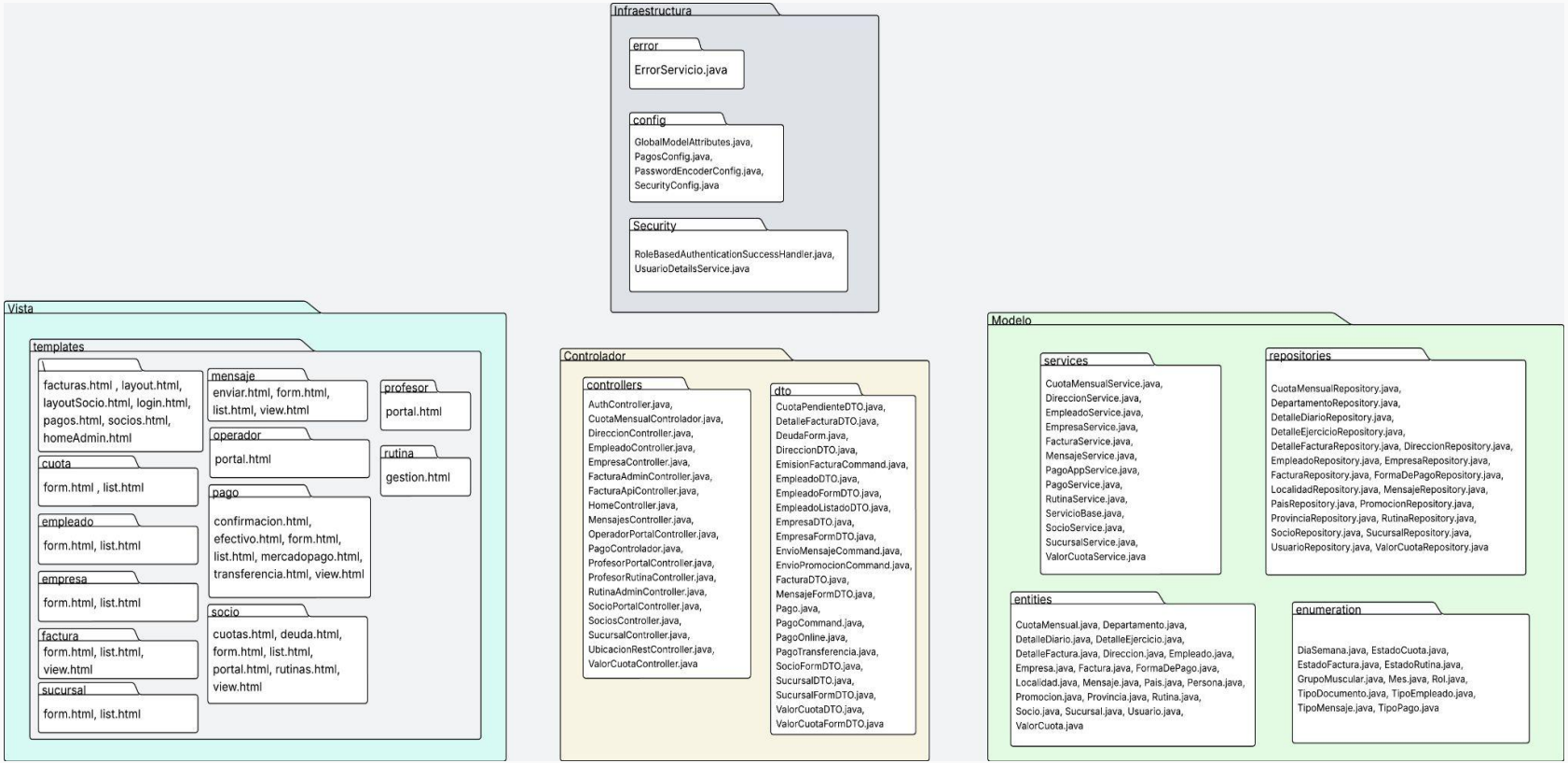




Modelo de Datos

Diagrama de Paquetes

Proyecto


Infraestructura
error
ErrorServicio.java
config
GlobalModelAttributes.java,
PagosConfig.java,
PasswordEncoderConfig.java,
SecurityConfig.java
Security
RoleBasedAuthenticationSuccessHandler.java,
UsuarioDetailsService.java
Vista
templates
facturas.html , layout.html,
layoutSocio.html, login.html,
pagos.html, socios.html,
homeAdmin.html
cuota
form.html , list.html
empleado
form.html, list.html
empresa
form.html, list.html
factura
form.html, list.html,
view.html
sucursal
form.html, list.html
mensaje
enviar.html, form.html,
list.html, view.html
operador
portal.html
pago
confirmacion.html,
efectivo.html, form.html,
list.html, mercadopago.html,
transferencia.html, view.html
socio
cuotas.html, deuda.html,
form.html, list.html,
portal.html, rutinas.html,
view.html
profesor
portal.html
rutina
gestion.html
Controlador
controllers
AuthController.java,
CuotaMensualControlador.java,
DireccionController.java,
EmpleadoController.java,
EmpresaController.java,
FacturaAdminController.java,
FacturaApiController.java,
HomeController.java,
MensajesController.java,
OperadorPortalController.java,
PagoControlador.java,
ProfesorPortalController.java,
ProfesorRutinaController.java,
RutinaAdminController.java,
SocioPortalController.java,
SociosController.java,
SucursalController.java,
UbicacionRestController.java,
ValorCuotaController.java
dto
CuotaPendienteDTO.java,
DetalleFacturaDTO.java,
DeudaForm.java,
DireccionDTO.java,
EmisionFacturaCommand.java,
EmpleadoDTO.java,
EmpleadoFormDTO.java,
EmpleadoListadoDTO.java,
EmpresaDTO.java,
EmpresaFormDTO.java,
EnvioMensajeCommand.java,
EnvioPromocionCommand.java,
FacturaDTO.java,
MensajeFormDTO.java,
Pago.java,
PagoCommand.java,
PagoOnline.java,
PagoTransferencia.java,
SocioFormDTO.java,
SucursalDTO.java,
SucursalFormDTO.java,
ValorCuotaDTO.java,
ValorCuotaFormDTO.java
Modelo
services
CuotaMensualService.java,
DireccionService.java,
EmpleadoService.java,
EmpresaService.java,
FacturaService.java,
MensajeService.java,
PagoAppService.java,
PagoService.java,
RutinaService.java,
ServicioBase.java,
SocioService.java,
SucursalService.java,
ValorCuotaService.java
repositories
CuotaMensualRepository.java,
DepartamentoRepository.java,
DetalleDiarioRepository.java,
DetalleEjercicioRepository.java,
DetalleFacturaRepository.java, DireccionRepository.java,
EmpleadoRepository.java, EmpresaRepository.java,
FacturaRepository.java, FormaDePagoRepository.java,
LocalidadRepository.java, MensajeRepository.java,
PaisRepository.java, PromocionRepository.java,
ProvinciaRepository.java, RutinaRepository.java,
SocioRepository.java, SucursalRepository.java,
UsuarioRepository.java, ValorCuotaRepository.java
entities
CuotaMensual.java, Departamento.java,
DetalleDiario.java, DetalleEjercicio.java,
DetalleFactura.java, Direccion.java, Empleado.java,
Empresa.java, Factura.java, FormaDePago.java,
Localidad.java, Mensaje.java, Pais.java, Persona.java,
Promocion.java, Provincia.java, Rutina.java,
Socio.java, Sucursal.java, Usuario.java,
ValorCuota.java
enumeration
DiaSemana.java, EstadoCuota.java,
EstadoFactura.java, EstadoRutina.java,
GrupoMuscular.java, Mes.java, Rol.java,
TipoDocumento.java, TipoEmpleado.java,
TipoMensaje.java, TipoPago.java

Requisitos Funcionales

- 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: Gestión 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.

Requisitos No Funcionales

- Usabilidad:

Interfaz intuitiva: Diseño claro y consistente para operadores y socios.
Compatibilidad multiplataforma: Accesible desde computadoras de escritorio y dispositivos móviles.

- Seguridad:

Autenticación y autorización: Acceso protegido con roles (administrador, operador, socio).
Protección de datos: Cifrado de contraseña.

- Rendimiento

Por el momento no se efectuaron pruebas de carga, sin embargo el sistema responde rápido en los casos que probamos (pago, consultas, alta de socio).

- Portabilidad:

Compatibilidad con distintas bases de datos: Uso de ORM para facilitar migraciones.

- Mantenibilidad y Escalabilidad:

Código modular: Uso de arquitectura en capas (por ejemplo, MVC con Spring Boot) para facilitar futuras ampliaciones.
La jerarquía de clases **Usuario, Socio y Empleado** permite incorporar fácilmente nuevos tipos de usuarios si el gimnasio amplía sus servicios o personal.

Patrones de Diseño

Patrones GRASP (1/2)

Information Expert → cada clase maneja sus propios datos (`Factura`, `CuotaMensual`).

Controller → `PagoControlador`, `FacturaControlador` median entre UI y lógica.

Creator → `PersonaService` crea `Socio` al conocer la info necesaria.

Low Coupling / High Cohesion → capas bien separadas

Patrones GRASP (2/2)

Pure Fabrication → servicios y DTOs (`PagoService`, `DeudaForm`).

Polymorphism → herencia (`Persona` abstracta → `Socio`, `Empleado`).

Indirection → servicios median entre controladores y repositorios.

Patrones GoF

Builder → creación de objetos con Lombok (`PagoOnline.builder()`).

Facade → `PagoService` simplifica coordinación interna.

Singleton → servicios/repos de Spring son instancias únicas.

Adapter → repositorios adaptan métodos a SQL.

Mejoras posibles:

- *Strategy*: distintos medios de pago.
- *State*: manejo avanzado de estados de cuotas/facturas.

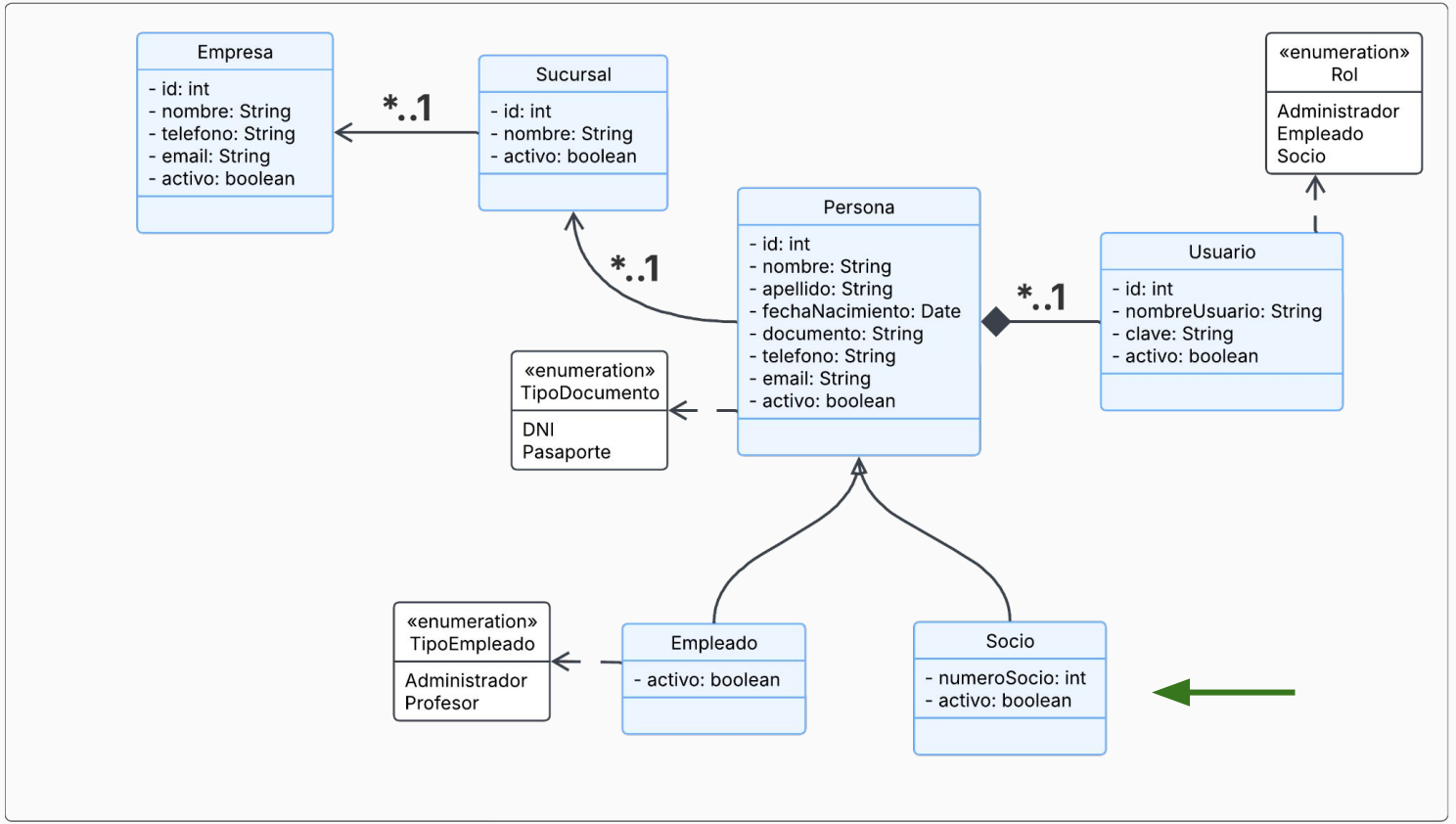
Software en Funcionamiento

Nueva Funcionalidad

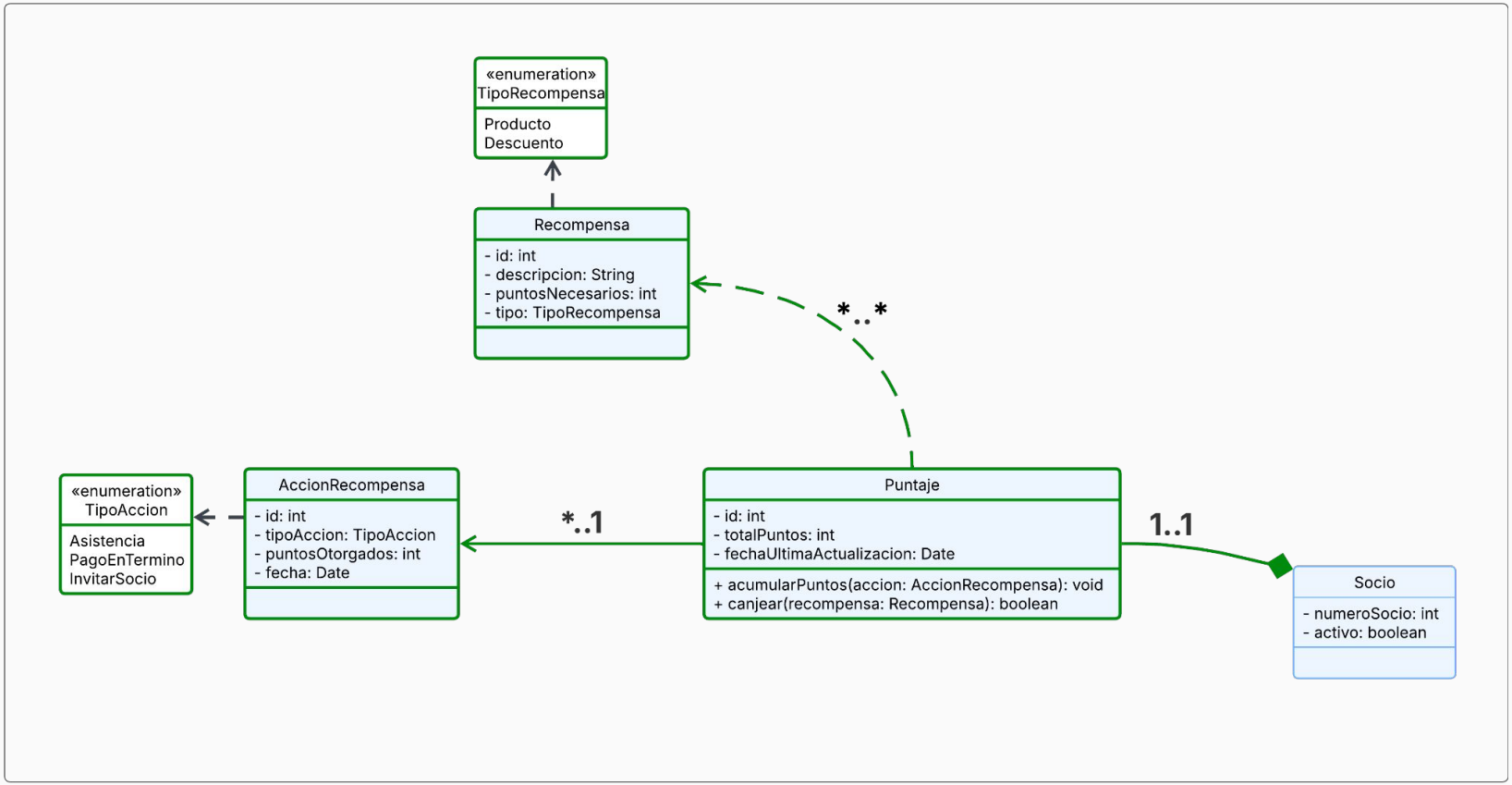
Socios acumulan puntos por acciones (asistencia, pago en término, referidos) y los canjean por beneficios.

Ejemplo: Mes del Amigo

Funcionalidad



Funcionalidad



Conclusión

Trabajo sinérgico entre documentación e implementación.

División de tareas +
Comunicación constante

Trabajo sinérgico entre documentación e implementación.

División de tareas +
Comunicación constante

Trabajo sinérgico entre documentación e implementación.

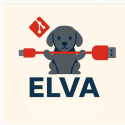
División de tareas +
Comunicación constante

¿Preguntas?

¡Muchas Gracias!

¡Muchas Gracias!

```
System.out.print
(“Muchas
Gracias”);
```



Fin