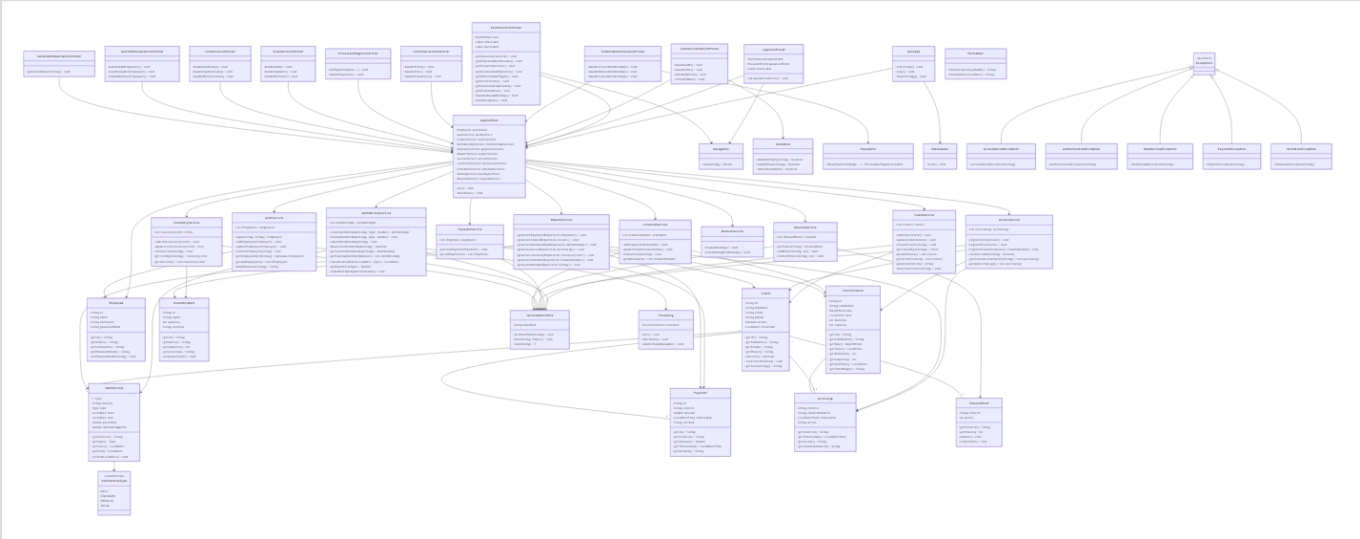
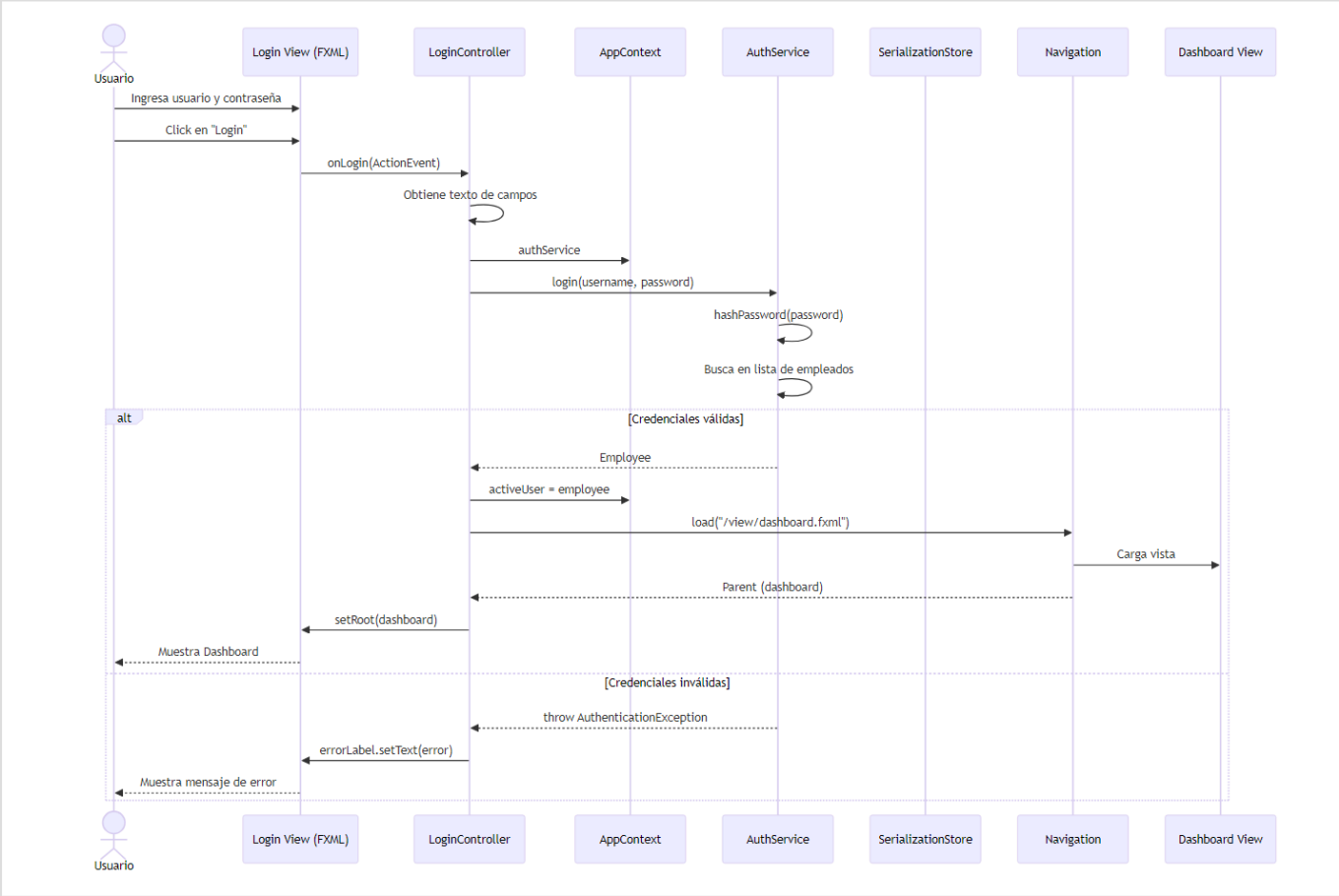
Diagramas UML del Sistema GymPOS

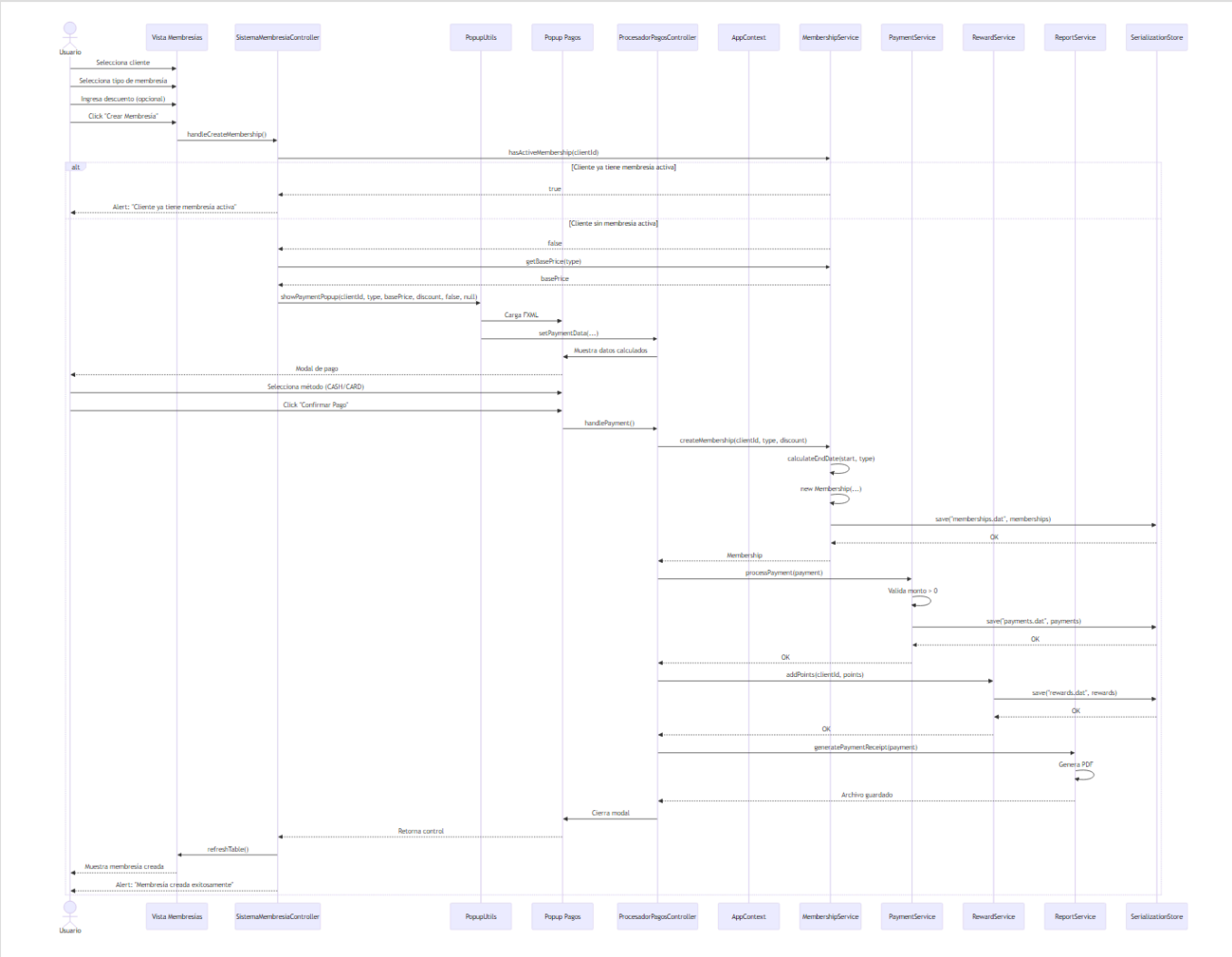
1. Diagrama de Clases Completo del Sistema



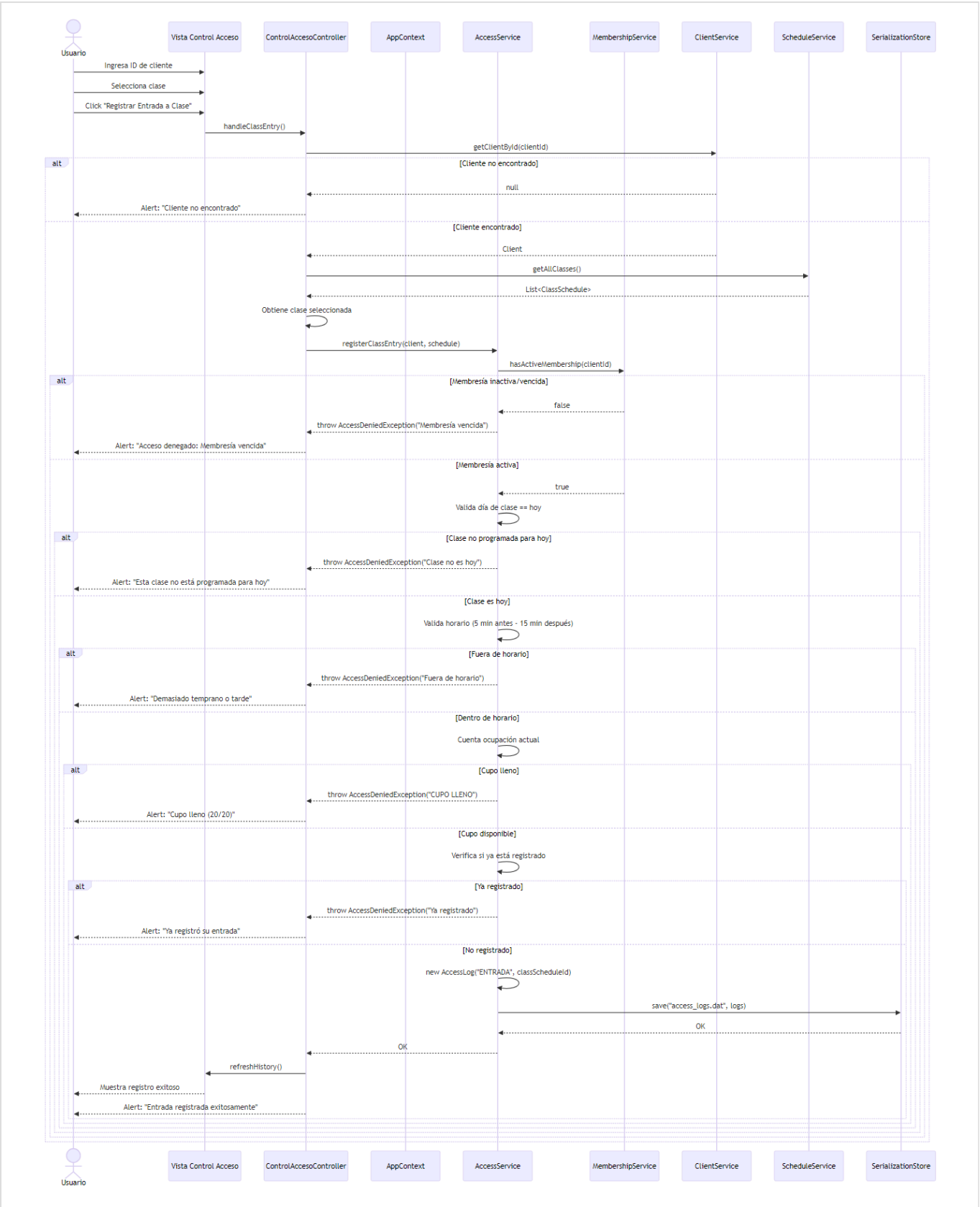
2. Diagrama de Secuencia: Proceso de Login



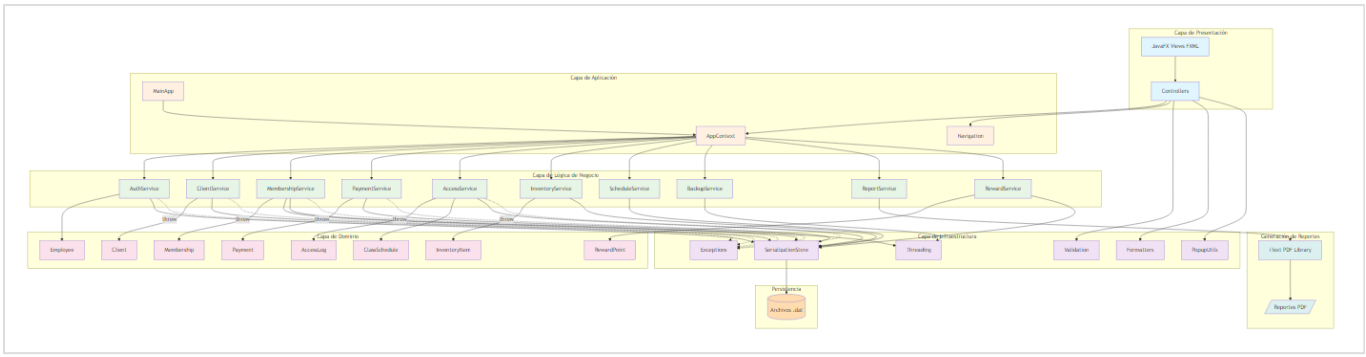
3. Diagrama de Secuencia: Creación de Membresía con Pago



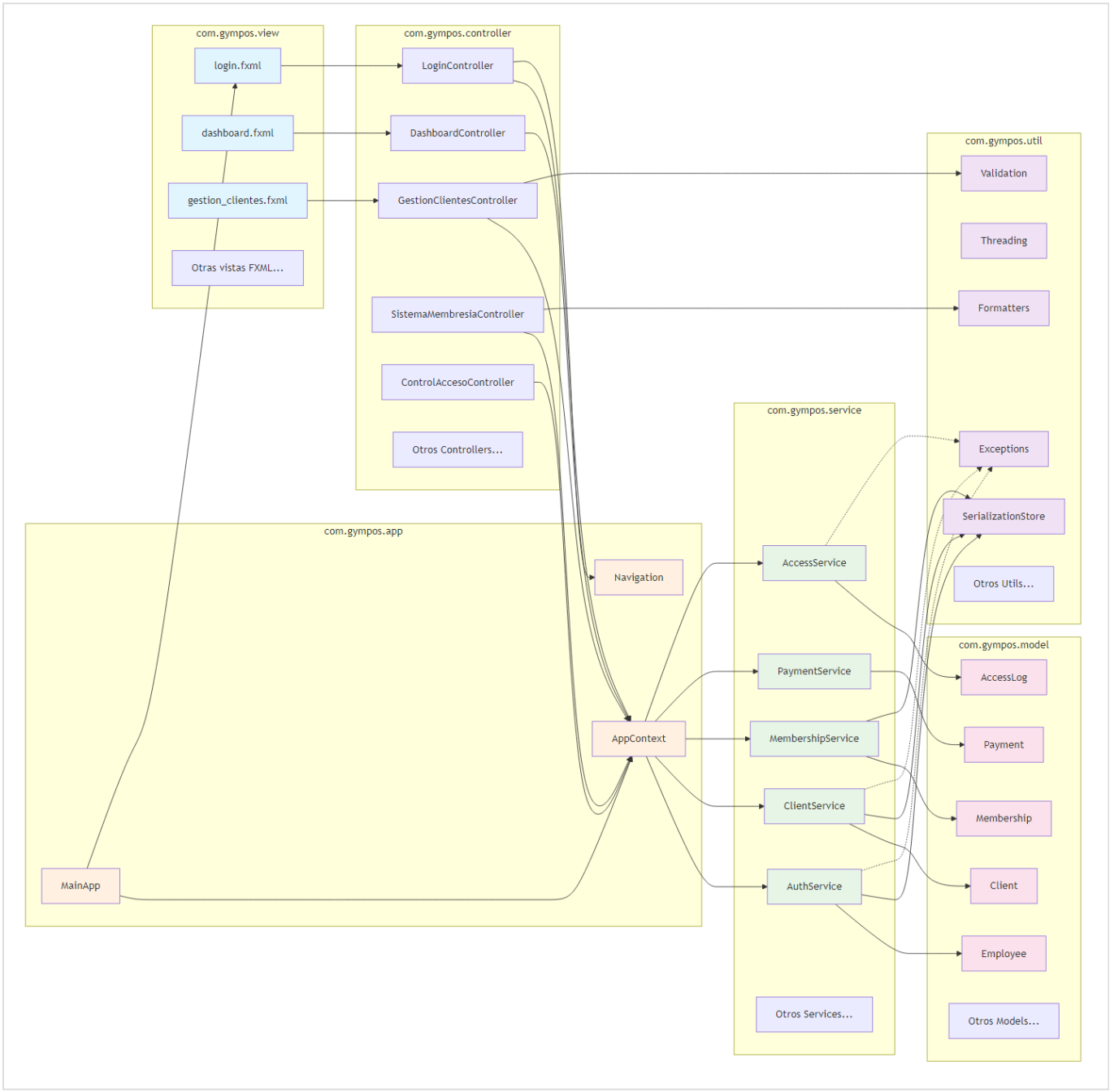
4. Diagrama de Secuencia: Control de Acceso a Clase



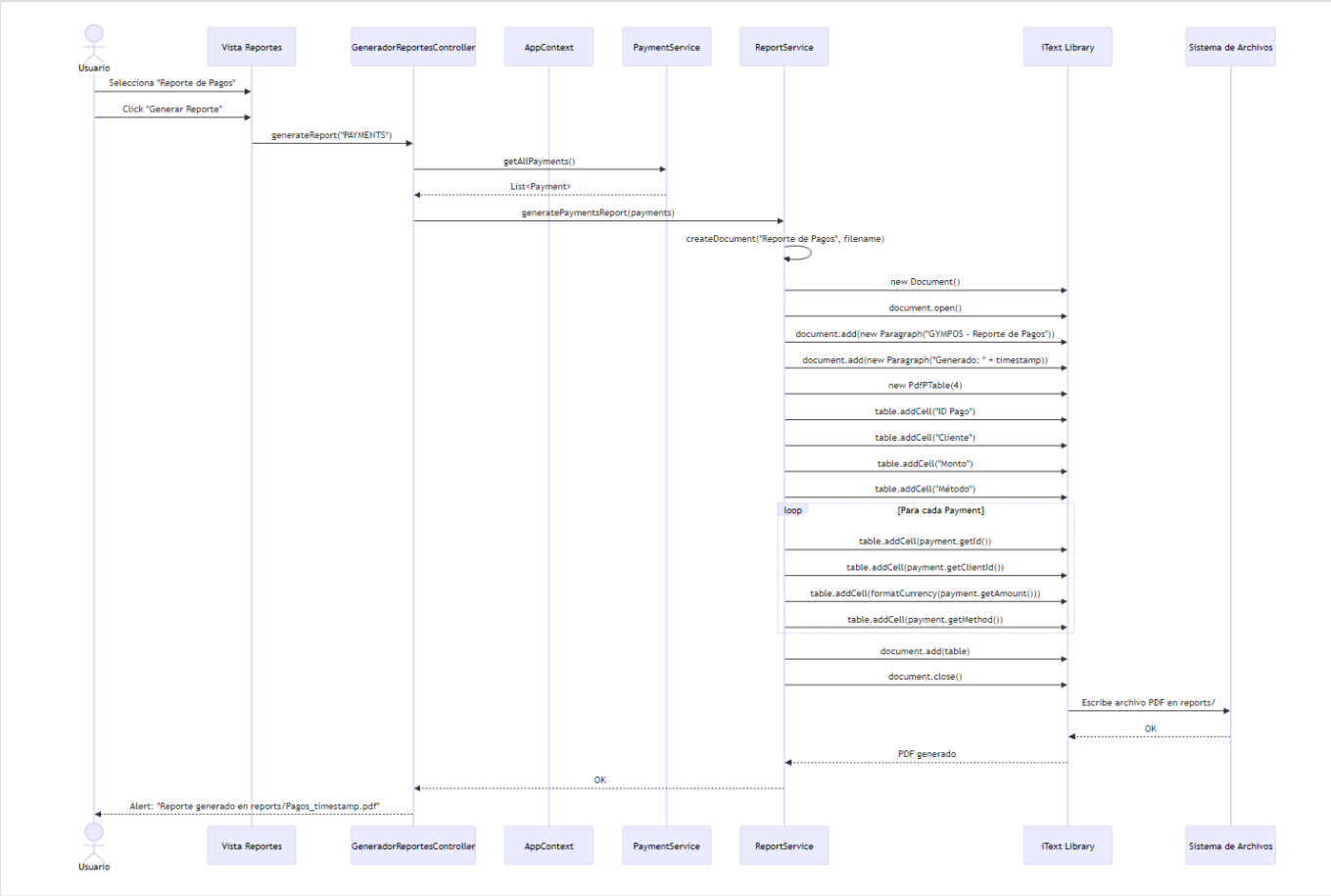
5. Diagrama de Arquitectura General del Sistema



6. Diagrama de Componentes: Estructura de Módulos



7. Diagrama de Secuencia: Generación de Reporte PDF



Descripción de la Arquitectura

Patrón Arquitectónico

El sistema GymPOS implementa una **arquitectura en capas** con separación clara de responsabilidades:

1. **Capa de Presentación**: Vistas FXML y Controllers JavaFX 2. **Capa de Aplicación**: Punto de entrada y coordinación global (MainApp, AppContext) 3. **Capa de Lógica de Negocio**: Services que encapsulan reglas de negocio 4. **Capa de Dominio**: Entidades del modelo de datos 5. **Capa de Infraestructura**: Utilidades transversales y persistencia

Principios Aplicados

- **Separation of Concerns**: Cada capa tiene responsabilidades específicas - **Dependency Injection**: AppContext actúa como contenedor de servicios - **Single Responsibility**: Cada clase tiene un propósito único - **Data Transfer**: Modelos como DTOs entre capas - **Facade Pattern**:

AppContext simplifica acceso a servicios - **Observer Pattern**: JavaFX properties para binding de UI - **Template Method**: Controllers siguen flujo estándar de validación-servicio-actualización

Flujo de Datos

`` Usuario → Vista FXML → Controller → Service → Model → SerializationStore → Archivo .dat ``

Características Técnicas

- **Persistencia**: Serialización Java (archivos .dat) - **Concurrencia**: ExecutorService para tareas background - **Reportes**: iText para generación de PDFs - **UI**: JavaFX con FXML + CSS - **Validación**: Centralizada en capa de servicios - **Excepciones**: Jerarquía personalizada para errores de negocio

Módulo App

Propósito y Responsabilidad

El módulo **app** es el punto de entrada de la aplicación GymPOS y se encarga de la inicialización del sistema, la gestión del ciclo de vida de la aplicación JavaFX y la coordinación de los servicios principales. Este módulo actúa como el orquestador central que conecta todos los componentes del sistema y proporciona utilidades para la navegación entre vistas.

Componentes Clave

`MainApp`

Clase principal de JavaFX que inicia la aplicación. Se encarga de:

- Cargar la ventana de login inicial
- Inicializar el contexto de la aplicación (`AppContext`)
- Poblar datos de prueba mediante `DataSeeder`
- Gestionar el cierre ordenado de recursos al finalizar

`` java public class MainApp extends Application { @Override public void start(Stage stage) throws Exception { AppContext.init(); DataSeeder.seed(); // Carga la vista de login } } `

AppContext

`Contenedor estático singleton que centraliza todos los servicios del sistema y mantiene el estado de sesión. Proporciona acceso global a:`

- `Usuario actualmente autenticado (` activeUser `)`
- `Instancias de todos los servicios (auth, clientes, membresías, pagos, etc.)`
- `Inicialización de hilos de ejecución y respaldos programados`

Navigation

Utilidad para cargar vistas FXML de manera segura y centralizada. Simplifica la navegación entre diferentes pantallas de la aplicación manejando excepciones de carga.

```java Parent view = Navigation.load("/com/gympos/view/dashboard.fxml"); ```
```

Módulo Model

Propósito y Responsabilidad

El módulo **model** contiene las entidades de dominio que representan los conceptos fundamentales del negocio del gimnasio. Estas clases implementan `Serializable` para permitir la persistencia mediante serialización Java y encapsulan los datos y reglas básicas de validación de cada entidad.

Componentes Clave

`Client`

Representa a un cliente del gimnasio con sus datos personales y estado de actividad.

Atributos:

- `id` : Identificador único (formato: C-XXX)
- `fullName` : Nombre completo del cliente
- `email` : Correo electrónico
- `phone` : Número telefónico
- `active` : Estado de la cuenta (activo/inactivo)
- `createdAt` : Fecha de registro

Métodos destacados:

- `getStatusString()` : Retorna "Activo" o "Inactivo" según el estado

``` java Client client = new Client("C-001", "Juan Pérez", "juan@gym.com", "555-0100"); client.setActive(false); // Desactivar cuenta ```

Employee

`Representa un empleado del sistema con credenciales de acceso.`

Atributos:
```

- id : Identificador único
- name : Nombre del empleado
- username : Usuario para login
- passwordHash : Contraseña hasheada con SHA-256

Membership

Representa una membresía de cliente con vigencia y tipo de servicio.

Tipos de membresía (Enum Type):

- BASIC : Membresía básica mensual
- STANDARD : Membresía estándar mensual
- PREMIUM : Membresía premium mensual
- ANUAL : Membresía anual

Atributos:

- clientId : ID del cliente propietario
- type : Tipo de membresía
- start : Fecha de inicio de vigencia
- end : Fecha de vencimiento
- pricePaid : Monto pagado por la membresía
- discountApplied : Descuento aplicado (decimal, ej: 0.15 = 15%)

Payment

Registra transacciones de pago realizadas por clientes.

Atributos:

- id : Identificador único del pago
- clientId : Cliente que realiza el pago
- amount : Monto pagado

- timestamp : Fecha y hora del pago (automática)
- method : Método de pago ("CASH" o "CARD")

AccessLog

Registra entradas y salidas de clientes al gimnasio o clases.

Atributos:

- clientId : Cliente que accede
- classScheduleId : ID de clase (null para acceso general)
- timestamp : Fecha y hora del acceso
- action : Tipo de acción ("ENTRADA" o "SALIDA")

Constructores: `java // Acceso general al gimnasio AccessLog log1 = new AccessLog("C-001", "ENTRADA");

// Acceso a clase específica AccessLog log2 = new AccessLog("C-001", "ENTRADA", "CLS-001"); `

ClassSchedule

Define horarios de clases grupales programadas.

Atributos:

- id : Identificador único de la clase
- className : Nombre de la clase (ej: "Yoga", "Spinning")
- day : Día de la semana (DayOfWeek)
- time : Hora de inicio (LocalTime)
- duration : Duración en minutos
- capacity : Cupo máximo de participantes

Métodos de utilidad:

- getEndTime() : Calcula hora de finalización

- getTimeRange() : Retorna rango "HH:MM - HH:MM"

`java ClassSchedule yoga = new ClassSchedule( "CLS-001", "Yoga Matutino", DayOfWeek.MONDAY, LocalTime.of(8, 0), 60, 20 ); `

InventoryItem

Representa un artículo o producto en el inventario del gimnasio.

Atributos:

- id : Identificador único
- name : Nombre del producto
- quantity : Cantidad en stock
- location : Ubicación física en el gimnasio

RewardPoint

Gestiona puntos de recompensa acumulados por clientes.

Atributos:

- clientId : Cliente propietario de los puntos
- points : Cantidad de puntos acumulados

Métodos:

- add(int p) : Suma puntos
- redeem(int p) : Canjea puntos (no puede ser negativo)

Relaciones Entre Entidades

` Client (1) ----< (0..n) Membership Client (1) ----< (0..n) Payment Client (1) ----< (0..n) AccessLog Client (1) ----< (0..1) RewardPoint ClassSchedule (1) ----< (0..n) AccessLog `

Características Comunes

Todas las entidades:

- Implementan Serializable para persistencia
- Usan tipos inmutables de Java 8+ (LocalDate, LocalDateTime, DayOfWeek)
- Sobrescriben equals() y hashCode()` basándose en el ID
- Siguen convenciones JavaBean con getters/setters
- Incluyen constructores sin argumentos para compatibilidad con frameworks

Módulo Service

Propósito y Responsabilidad

El módulo **service** implementa la capa de lógica de negocio del sistema, encapsulando operaciones complejas, reglas de negocio, validaciones y coordinación entre entidades. Cada servicio gestiona un aspecto específico del dominio y proporciona una API limpia para los controladores. Todos los servicios utilizan `SerializationStore` para persistencia de datos.

Componentes Clave

`AuthService`

Gestiona autenticación, autorización y administración de empleados.

Responsabilidades:

- CRUD completo de empleados
- Autenticación mediante usuario/contraseña
- Hashing de contraseñas con SHA-256
- Validación de credenciales únicas

Métodos principales: ``java Employee login(String username, String password) // Autenticar usuario void addEmployee(Employee employee) // Registrar empleado void updateEmployee(Employee updated) // Actualizar datos void removeEmployee(String employeeId) // Eliminar empleado List getAllEmployees() // Listar todos ``

Lógica de negocio:

- Las contraseñas se hashean automáticamente al agregar empleados
- No se permiten nombres de usuario duplicados
- El hash SHA-256 es irreversible (seguridad)

Ejemplo de uso: `java Employee user = authService.login("admin", "123"); if (user != null) { // Login exitoso } `

ClientService

Administra el ciclo de vida de clientes del gimnasio.

Responsabilidades:

- CRUD de clientes
- Generación automática de IDs secuenciales
- Filtrado de clientes activos
- Activación/desactivación de cuentas

Métodos principales: `java void addClient(Client client) void updateClient(Client updated) void removeClient(String clientId) Client getClientById(String clientId) List getAllClients() List getActiveClients() String generateNextId() // Genera C-001, C-002, etc. void deactivateClient(String clientId) `

MembershipService

Gestiona membresías con lógica compleja de precios, renovaciones y vencimientos.

Responsabilidades:

- Creación de membresías con cálculo de fechas de vigencia
- Renovación de membresías (extensión de fecha)
- Cancelación de membresías
- Detección de membresías próximas a vencer
- Notificaciones programadas de vencimiento
- Cálculo de precios base por tipo

Precios configurados (personalizables por matrícula): `java BASIC (DAVILA): $4,122 MXN - 1 mes STANDARD (FLORES): $6,033 MXN - 1 mes PREMIUM (SEGOBIA): $7,528 MXN - 1 mes ANUAL (ZACATENCO): $7,791 MXN - 12 meses `

Métodos principales: `java Membership createMembership(String clientId, Type type, double discount) void renewMembership(String clientId, Type type, double pricePaid) void cancelMembership(String clientId) boolean hasActiveMembership(String clientId) Membership getCurrentMembership(String clientId) List getExpiringMemberships(int daysThreshold) double getBasePrice(Type type) LocalDate calculateEndDate(LocalDate start, Type type) `

Lógica de renovación:

- Si existe membresía activa: extiende la fecha de vencimiento
- Si no existe o ya venció: crea nueva membresía desde hoy
- Previene duplicación de membresías activas

Notificaciones automáticas: `java scheduleExpiringNotifications() // Verifica cada hora si hay membresías próximas a vencer (7 días) `

PaymentService

Registra y valida transacciones de pago.

Responsabilidades:

- Procesamiento de pagos
- Validación de montos positivos
- Registro histórico de transacciones

Métodos principales: `java void processPayment(Payment payment) List getAllPayments() `

Validaciones:

- El monto debe ser mayor a cero
- Se registra timestamp automáticamente al crear Payment

AccessService

Controla el acceso físico al gimnasio y a clases grupales con reglas de negocio estrictas.

Responsabilidades:

- Registro de entradas/salidas generales
- Control de acceso a clases con validación de horario y cupo
- Verificación de membresía activa
- Prevención de accesos duplicados
- Historial de accesos por cliente

Métodos principales: `java void registerEntry(Client client) // Entrada general void registerExit(Client client) // Salida general void registerClassEntry(Client client,

ClassSchedule schedule) // Entrada a clase boolean isClientInside(String clientId) List getClientAccessHistory(String clientId) `

Reglas de negocio para clases: 1. El cliente debe tener membresía activa 2. La clase debe estar programada para el día actual 3. Acceso permitido: 5 minutos antes hasta 15 minutos después del inicio 4. Verificación de cupo disponible (capacidad máxima) 5. No se permiten entradas duplicadas a la misma clase en el mismo día

Ejemplo de validación: `java // Clase a las 8:00 AM con capacidad de 20 personas // Acceso permitido: 7:55 AM - 8:15 AM // Si ya hay 20 registros de entrada: CUPO LLENO `

InventoryService

Gestiona inventario de productos e insumos del gimnasio.

Responsabilidades:

- CRUD de artículos de inventario
- Control de stock y ubicaciones
- Búsqueda de productos por ID

Métodos principales: `java void addItem(InventoryItem item) void updateItem(InventoryItem updated) void removeItem(String id) InventoryItem getItemById(String id) List getAllItems() `

ScheduleService

Administra calendario de clases con validación de traslapes de horarios.

Responsabilidades:

- CRUD de horarios de clases
- Detección de conflictos de horario
- Validación de disponibilidad al crear/editar clases

Métodos principales: `java void addClass(ClassSchedule classSchedule) void updateClass(ClassSchedule updated) void removeClass(String id) List getAllClasses() `

Lógica de traslape: `java // Detecta si dos clases del mismo día se traslapan // Clase A: 8:00 - 9:00 // Clase B: 8:30 - 9:30 → CONFLICTO // Clase C: 9:00 - 10:00 → OK (no se traslapa) `

BackupService

Gestiona respaldos de datos del sistema.

Responsabilidades:

- Respaldos manuales bajo demanda
- Programación de respaldos nocturnos automáticos
- Copia de archivos .dat a carpeta timestamped

Métodos principales: `java void manualBackup() // Crea backup inmediato void scheduleNightlyBackup() // Programa backup automático `

Estructura de respaldos: ` backups/ backup_20251114_193045/ clients.dat memberships.dat payments.dat ... `

ReportService

Genera reportes PDF profesionales con iText.

Responsabilidades:

- Generación de múltiples tipos de reportes
- Formato profesional con tablas y metadatos

- Almacenamiento en carpeta reports/

Reportes disponibles: `java void generatePaymentsReport(List payments) void generateClientsReport(List clients) void generateMembershipsReport(List memberships) void generateAccessReport(List logs) void generateInventoryReport(List items) void generateScheduleReport(List classes) void generateBackupsReport(List backupNames) `

Formato de archivos:

- Nombre: TipoReporte_timestamp.pdf
- Incluye fecha de generación
- Tablas con datos relevantes según tipo de reporte

RewardService

Sistema de puntos de recompensa para clientes.

Responsabilidades:

- Acumulación de puntos por pagos
- Canje de puntos por beneficios
- Consulta de balance de puntos

Métodos principales: `java RewardPoint getPoints(String clientId) void addPoints(String clientId, int points) void redeemPoints(String clientId, int pointsToRedeem) `

Regla de negocio:

- Los puntos canjeados no pueden resultar en balance negativo
- Típicamente: 10% del monto pagado = puntos otorgados

Flujo de Datos Típico

` Controller ↓ Service (validación + lógica de negocio) ↓ Model (entidad) ↓ SerializationStore (persistencia) `

Principios de Diseño

1. **Single Responsibility**: Cada servicio gestiona un aspecto del dominio 2. **Encapsulación**: La lógica de negocio está oculta de los controladores 3. **Validación centralizada**: Los servicios validan antes de persistir 4. **Manejo de excepciones**: Lanzan excepciones personalizadas para errores de negocio 5. **Persistencia automática**: Todos los cambios se guardan inmediatamente

Excepciones de Negocio

Los servicios lanzan excepciones específicas definidas en Exceptions :

- AccessDeniedException : Acceso denegado por reglas de negocio
- AuthenticationException : Credenciales inválidas

- MembershipException `: Error en operación de membresía`
- PaymentException `: Error en procesamiento de pago`
- ValidationException`: Datos inválidos

Módulo Util

Propósito y Responsabilidad

El módulo **util** proporciona clases utilitarias y componentes de infraestructura que brindan funcionalidad transversal al sistema. Incluye mecanismos de persistencia, validación, formateo, manejo de excepciones, concurrencia, diálogos emergentes y población de datos de prueba.

Componentes Clave

`SerializationStore`

Sistema de persistencia mediante serialización Java que guarda objetos en archivos binarios.

Responsabilidades:

- Configuración de ruta base de almacenamiento
- Serialización de objetos a archivos .dat
- Deserialización de objetos desde archivos
- Manejo transparente de errores de I/O

Métodos principales: `` `java void initBasePaths(String path) // Configura directorio base (./data) void save(String filename, Object) // Guarda objeto en archivo T load(String filename) // Carga objeto desde archivo ` ``

Uso típico: `` `java SerializationStore.initBasePaths("./data"); List clients = new ArrayList<>(); SerializationStore.save("clients.dat", clients);

// Más tarde... List loaded = SerializationStore.load("clients.dat"); ` ``

Ventajas:

- `Persistencia simple sin base de datos`
- `Serialización automática de grafos de objetos`
- `Retorna` null `si el archivo no existe (primera ejecución)`

Threading

Administrador centralizado de hilos para tareas asíncronas y de fondo.

Responsabilidades:

- Inicialización de pool de hilos (10 threads)
- Ejecución de tareas en segundo plano
- Cierre ordenado de recursos al finalizar aplicación

Métodos principales: `java void init() // Crea ExecutorService con 10 threads void shutdown() // Cierra pool de hilos void submitTask(Runnable task) // Ejecuta tarea en background `

Uso típico: `java Threading.init();

// Tarea de larga duración sin bloquear UI Threading.submitTask(() -> { while(true) { // Verificar vencimientos cada hora Thread.sleep(3600000); checkExpiredMemberships(); } }); `

Validation

Utilidades de validación de datos de entrada.

Responsabilidades:

- Validación de campos nulos o vacíos
- Validación de formato de email con regex
- Validación de números positivos

Métodos principales: `java boolean isNullOrEmpty(String text) boolean isValidEmail(String email) boolean isPositive(double number) `

Patrones de validación:

- Email: ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,6}$

Ejemplo: `java if (Validation.isNullOrEmpty(nameField.getText())) { showError("El nombre es obligatorio"); return; }

if (!Validation.isValidEmail(emailField.getText())) { showError("Email inválido"); return; } `

Formatters

Formateadores de datos para presentación consistente en la UI.

Responsabilidades:

- Formateo de moneda en pesos mexicanos (MXN)
- Formateo de fechas en formato local (dd/MM/yyyy)

Métodos principales: `java String formatCurrency(double amount) // $1,234.56 MXN String formatDate(LocalDate date) // 14/11/2025 `

Configuración:

- Locale: español de México (es_MX)
- Formato de fecha: día/mes/año

Ejemplo: `java String precio = Formatters.formatCurrency(6033.00); // "$6,033.00" String fecha = Formatters.formatDate(LocalDate.now()); // "14/11/2025" `

Exceptions

Jerarquía de excepciones personalizadas para reglas de negocio.

Excepciones definidas:

AccessDeniedException

Se lanza cuando se deniega acceso al gimnasio o clase por:

- Membresía vencida
- Fuera de horario permitido
- Cupo lleno
- Cliente ya registrado

AuthenticationException

Se lanza cuando fallan las credenciales de login:

- Usuario inexistente

- Contraseña incorrecta

MembershipException

Se lanza en operaciones inválidas de membresía:

- Intento de crear membresía duplicada
- Cliente sin membresía al intentar renovar

PaymentException

Se lanza en errores de procesamiento de pago:

- Monto inválido (cero o negativo)

ValidationException

Se lanza en validaciones generales:

- Datos obligatorios faltantes
- Formato de datos incorrecto
- Entidad no encontrada

Todas heredan de RuntimeException (no checked exceptions)

Ejemplo de uso: ` java if (amount <= 0) { throw new PaymentException("El monto debe ser mayor a cero"); } try { service.doSomething(); } catch (ValidationException e) { showAlert(e.getMessage()); } `

PopupUtils

Utilidad para mostrar ventanas modales reutilizables.

Responsabilidades:

- Creación de ventanas emergentes modales
- Carga de vistas FXML para popups

- Aplicación de estilos CSS
- Bloqueo de ventana principal hasta cerrar popup

Método principal: `java ProcesadorPagosController showPaymentPopup( String clientId, Type type, double basePrice, double discount, boolean isRenewal, String currentEndDate ) `

Características:

- Modal: bloquea interacción con ventana padre
- Sin decoración (sin barra de título estándar)
- Centrado en pantalla
- Estilos personalizados aplicados

Ejemplo: `java ProcesadorPagosController ctrl = PopupUtils.showPaymentPopup("C-001", Membership.Type.PREMIUM, 7528.00, 0.10, false, null);

if (ctrl.isPaymentSuccessful()) { // Procesar resultado } `

DataSeeder

Generador de datos de prueba para desarrollo y demostración.

Responsabilidades:

- Población inicial de 20 clientes de ejemplo
- Creación automática de membresías
- Generación de pagos de prueba
- Solo ejecuta si no hay datos existentes

Lógica de generación:

- 20 clientes con nombres predefinidos
- Emails generados: nombre.apellido@gym.com
- Teléfonos: 555-01XX
- 1 de cada 5 clientes inactivo (20%)
- Tipos de membresía aleatorios
- Método de pago alternado (CASH/CARD)

Ejemplo de datos generados: ` C-001 | Juan Pérez | juan.pérez@gym.com | PREMIUM | ACTIVO C-002 | María López | maría.lópez@gym.com | BASIC | ACTIVO C-003 | Carlos Ruiz

| carlos.ruiz@gym.com | ANUAL | ACTIVO C-004 | Ana Torres | ana.torres@gym.com | STANDARD| ACTIVO C-005 | Pedro Sánchez | pedro.sánchez@gym.com | BASIC | INACTIVO `

Invocación: `java public static void seed() { if (!AppContext.clientService.getAllClients().isEmpty()) { return; // Ya hay datos, no poblar } // Generar 20 clientes con membresías y pagos } `

Principios de Diseño

1. **Reutilización**: Código común extraído a utilidades 2. **Separación de responsabilidades**: Cada clase tiene un propósito específico 3. **Facilidad de mantenimiento**: Cambios centralizados (ej: formato de fecha) 4. **Testabilidad**: Métodos estáticos facilitan pruebas unitarias 5. **Configurabilidad**: Paths y formatos configurables

Flujo de Uso Típico

` Aplicación inicia ↓ SerializationStore.initBasePaths("./data") ↓ Threading.init() ↓ DataSeeder.seed() // Si es primera vez ↓ Servicios usan SerializationStore para persistencia ↓ Controladores usan Validation/Formatters para UI ↓ Threading.submitTask() para tareas background ↓ Threading.shutdown() al cerrar aplicación ``

Módulo Controller

Propósito y Responsabilidad

El módulo **controller** implementa el patrón MVC (Model-View-Controller) actuando como intermediario entre las vistas JavaFX (archivos FXML) y la lógica de negocio encapsulada en los servicios. Cada controlador maneja las interacciones del usuario, valida entradas, invoca servicios correspondientes y actualiza la interfaz gráfica según los resultados.

Componentes Clave

`LoginController`

Gestiona la autenticación de usuarios:

- Captura credenciales (usuario y contraseña)
- Valida mediante `AuthService`
- Redirige al Dashboard tras login exitoso
- Muestra mensajes de error en caso de credenciales inválidas

``` java @FXML public void onLogin(ActionEvent e) { AppContext.activeUser = AppContext.authService.login(username, password); // Navegar al Dashboard } ```

DashboardController

Pantalla principal del sistema que proporciona:

- Navegación a todos los módulos funcionales
- Reloj en tiempo real actualizado cada segundo
- Gestión de respaldos manuales
- Control de acceso a funciones administrativas (gestión de empleados)
- Cierre de sesión con confirmación

Flujo de seguridad para funciones administrativas: 1. Verifica que el usuario sea "admin" 2. Solicita reconfirmación de contraseña mediante diálogo modal 3. Permite acceso solo si la validación es exitosa
```

GestionClientesController

CRUD completo de clientes:

- Registro de nuevos clientes con validación de campos
- Edición de información de clientes existentes
- Activación/desactivación de cuentas
- Búsqueda y filtrado de clientes
- Visualización en tabla con datos: ID, nombre, email, teléfono, estado

SistemaMembresiaController

Administración del ciclo de vida de membresías:

- Creación de nuevas membresías con cálculo automático de fechas
- Renovación de membresías existentes (extiende fecha de vencimiento)
- Cancelación de membresías
- Aplicación de descuentos (porcentaje personalizable)
- Tipos de membresía: BASIC, STANDARD, PREMIUM, ANUAL

- Integración con PaymentService para registro de pagos

Lógica de precios:

- BASIC: $4,122 (1 mes)
- STANDARD: $6,033 (1 mes)
- PREMIUM: $7,528 (1 mes)
- ANUAL: $7,791 (12 meses)

ControlAccesoController

Control de entrada y salida del gimnasio:

- Registro de entradas generales (gimnasio)
- Registro de entradas a clases específicas con validación de horario
- Validación de membresía activa antes de permitir acceso

- Verificación de cupo disponible en clases
- Prevención de registros duplicados
- Visualización de historial de accesos por cliente

Reglas de negocio:

- Solo clientes con membresía activa pueden ingresar
- Entrada a clases: 5 minutos antes hasta 15 minutos después del inicio
- Control de capacidad máxima por clase
- No se permiten entradas duplicadas a la misma clase

ProcesadorPagosController

Ventana modal para procesar pagos:

- Muestra información de la membresía a pagar
- Calcula monto final con descuentos aplicados
- Registra método de pago (CASH/CARD)

- Integra con MembershipService y PaymentService
- Otorga puntos de recompensa (10% del monto pagado)
- Genera recibo en PDF mediante ReportService

InventarioController

Gestión de artículos e insumos:

- Alta, baja y modificación de productos
- Control de cantidades y ubicaciones
- Búsqueda y filtrado de inventario
- Actualización en tiempo real de stock

SchedulesController

Calendario de clases grupales:

- Creación de horarios de clases

- Edición de clases existentes
- Eliminación de clases
- Validación de traslapes de horarios
- Configuración de capacidad, duración y día de la semana

GestionEmpleadosController

Administración de personal (solo para admin):

- Registro de nuevos empleados
- Asignación de credenciales de acceso
- Actualización de contraseña con doble confirmación
- Hash SHA-256 de contraseñas para seguridad
- Eliminación de empleados

GeneradorReportesController

Generación de reportes PDF:

- Reportes de pagos
- Reportes de clientes
- Reportes de membresías
- Reportes de accesos
- Reportes de inventario
- Reportes de calendario de clases
- Reportes de respaldos

Flujo de Datos Típico

1. **Usuario interactúa con la vista** (botones, campos de texto) 2. **Controlador captura el evento** mediante anotaciones @FXML 3. **Validación de entrada** usando Validation utilities 4. **Invocación de servicio** correspondiente a través de AppContext 5. **Actualización de la interfaz** con los resultados (tablas, alertas, navegación) 6. **Manejo de excepciones** con diálogos informativos al usuario

```java @FXML private void onSave() { try { // Validar datos if (Validation.isNullOrEmpty(nameField.getText())) { showError("El nombre es obligatorio"); return; } // Invocar servicio Client client = new Client(...); AppContext.clientService.addClient(client); // Actualizar vista refreshTable(); showSuccess("Cliente guardado exitosamente"); } catch (Exception e) { showError(e.getMessage()); } } ```
```

Módulo View

Propósito y Responsabilidad

El módulo **view** contiene las definiciones de interfaz de usuario mediante archivos FXML que describen la estructura visual y layout de cada pantalla del sistema. Estos archivos son interpretados por JavaFX para renderizar componentes gráficos y vincularlos con sus respectivos controladores.

Archivos FXML

El módulo incluye las siguientes vistas principales:

- **login.fxml**: Pantalla de autenticación con campos de usuario, contraseña y botón de acceso.
- **dashboard.fxml**: Menú principal con botones de navegación a todos los módulos, reloj en tiempo real y opciones de respaldo/logout.
- **gestion_clientes.fxml**: Interfaz CRUD para administración de clientes con tabla, formularios y botones de acción.
- **sistema_membresias.fxml**: Gestión de membresías con selección de tipo, cálculo de precios y fechas de vigencia.
- **control_acceso.fxml**: Control de entradas/salidas al gimnasio y clases con validación de membresías.
- **procesador_pagos.fxml**: Ventana modal para procesamiento de pagos con método de pago y confirmación.
- **inventario.fxml**: Administración de productos con control de stock y ubicaciones.
- **schedules.fxml**: Calendario de clases grupales con validación de horarios y capacidad.
- **gestion_empleados.fxml**: CRUD de empleados con gestión de credenciales (solo admin).
- **generador_reportes.fxml**: Generación de reportes PDF con opciones de filtrado y exportación.

Cada vista está vinculada a su controlador correspondiente mediante el atributo `fx:controller` y utiliza `fx:id` para referenciar componentes desde código Java. Los estilos se

aplican mediante el archivo CSS compartido en `resources/css/app.css` .