

a. Modelo de dominio

Clase	Atributos	Relaciones
Usuario (abstracta)	<ul style="list-style-type: none"> • login: String • password: String • nombre: String • saldo: double 	Superclase de Cliente, Organizador y Administrador
Cliente	<ul style="list-style-type: none"> • idCliente:String 	El Cliente se conecta con Tiquete (boletas que adquiere) y con Transacción (operaciones de compra o reembolso).
Organizador	<ul style="list-style-type: none"> • idOrganizador: int 	El Organizador está conectado con Evento , porque es quien los crea y gestiona, y con Oferta , porque define los descuentos aplicados a localidades de sus eventos.
Administrador	<ul style="list-style-type: none"> • idAdministrador:String 	El Administrador , como subclase de Usuario , se conecta con Venue (creación/aprobación) y con Evento (cancelación).
Evento	<ul style="list-style-type: none"> • idEvento: int • nombre: String • fecha: Date • hora: Time • tipoEvento: TipoEvento 	El Evento está ligado por composición con Venue (se realiza en un lugar específico), se conecta con Tiquete (define las entradas disponibles) y con TiqueteTemporada (puede formar parte de pases múltiples), además de relacionarse con el Organizador , que lo crea y gestiona.

Venue	<ul style="list-style-type: none"> • idVenue: int • nombre: String • ubicacion: String • capacidad: int • restriccionesUso: String 	El Venue se relaciona con el Administrador (puede crearlo o aprobarlo), tiene composición con Evento (los eventos ocurren en un lugar específico) y se conecta con Localidad , que define la distribución interna del espacio.
Localidad	<ul style="list-style-type: none"> • nombre: String • precioBase: double • numerada: boolean 	La Localidad forma parte por composición del Venue (secciones del lugar), se conecta con

	<ul style="list-style-type: none"> • numAsientos: int 	Tiquete (las boletas dependen de la localidad) y con Oferta , que aplica descuentos sobre ella.
Tiquete (abstracta)	<ul style="list-style-type: none"> • idTiquete: int • precio: double • cargoServicio: double • cargoEmision: double • estado: String <p>“Como el enunciado dice que la fecha y hora del tiquete dependen del evento, no es un atributo propio del Tiquete, sino que viene del Evento.”</p>	El Tiquete se relaciona con el Cliente (quien lo adquiere), con el Evento (al que da acceso), con la Localidad (que determina su precio y características) y con la Transacción (obligatoria para su compra). Además, es clase padre de TiqueteBásico , TiqueteMúltiple , TiqueteTemporada y PaqueteDeluxe .
TiqueteBasico	<ul style="list-style-type: none"> • numeroAsiento: int • localidadNumerada : bool 	Hereda de Tiquete
TiqueteNumerado	<ul style="list-style-type: none"> • numeroAsiento: int 	Hereda de Tiquete
TiqueteMultiple	<ul style="list-style-type: none"> • cantidadEntradas: int • precioTotal: double 	Compone Tiquetes Hereda de Tiquete
PaqueteDeluxe	<ul style="list-style-type: none"> • beneficios: String • tiquetesIncluidos: List<Tiquete> 	Hereda Tiquetes Agregacion Tiquetes
Transaccion	<ul style="list-style-type: none"> • idTransaccion: int • fecha: Date • monto: double 	La Transacción está vinculada al Cliente que la realiza y tiene una relación de obligatoriedad con el Tiquete , ya que toda compra de boletas debe quedar registrada en una transacción.
Oferta	<ul style="list-style-type: none"> • descuento: double • fechaInicio: Date • fechaFin: Date 	La Oferta es creada por un Organizador y aplica sobre una Localidad , estableciendo descuentos en sus tiquetes durante un tiempo definido.

b. Reglas del dominio

- Cada tiquete debe tener un identificador único.
- Cada tiquete tiene un precio base, un cargo porcentual por servicio y una cuota fija de emisión.
- Puede existir un número máximo de tiquetes por transacción.

- Un cliente puede transferir un ticket a otro, validando login y contraseña.
 - Los tickets Deluxe no se pueden transferir.
 - Los tickets múltiples pueden transferirse de forma individual o completa, pero no como paquete completo si ya hay vencidos o transferidos.
 - Los tickets múltiples no pueden venderse individualmente.
 - En los tickets múltiples y de temporada, el tope de compra aplica al paquete, no a los tickets internos.
 - El precio de los paquetes puede ser distinto a la suma de los tickets que incluye.
 - Todo evento debe tener un organizador (no puede existir evento sin promotor).
 - Todo evento debe tener un venue asociado.
 - Un venue no puede tener dos eventos el mismo día.
 - Las localidades no son inherentes al venue (se configuran según el evento).
 - Todas las boletas de una misma localidad tienen el mismo precio.
 - Si una localidad es numerada, cada ticket debe tener un asiento único en esa localidad.
-
- Un organizador puede generar ofertas sobre localidades de un evento.
-
- Una oferta aplica un descuento porcentual en una ventana de tiempo definida.
 - Los venues sugeridos por promotores deben ser aprobados por el administrador antes de usarse.
 - El administrador fija el cargo por servicio y la cuota fija de emisión.
 - El administrador puede establecer tarifas distintas según tipo de evento.
 - El administrador puede cancelar un evento y definir el tipo de reembolso.
 - Un cliente puede pedir reembolso por calamidad, a discreción del administrador.
 - Todos los usuarios tienen un saldo virtual inicial en 0, donde se cargan los reembolsos.
 - El administrador no puede comprar tickets.

c. Restricciones del dominio

ID	Área/Proceso	Descripción	Comportamiento esperado del sistema	Observaciones
RES-001	Persistencia	La información debe ser persistente y almacenada en archivos en una carpeta separada del código.	Los datos se almacenan correctamente en la carpeta definida.	El sistema no debe usar la misma carpeta para código fuente y datos.

RES-002	Estructura de Archivos	La persistencia no debe realizarse en un solo archivo. Se debe definir cuántos archivos y cómo estarán estructurados.	Los datos deben ser distribuidos en múltiples archivos organizados según el dominio.	Puede ser necesario organizar diferentes archivos para usuarios, eventos, transacciones, etc.
RES-003	Lenguaje de Programación	El sistema debe ser implementado en Java .	Todo el código debe ser desarrollado usando Java .	Este es un requisito técnico que define la plataforma de implementación.
RES-004	Interfaz de Usuario	No es necesario implementar una interfaz de usuario en este primer proyecto (solo consola para pruebas).	El sistema debe ser probado usando la consola para mostrar su funcionamiento.	En el proyecto #2 se implementará la interacción con el usuario.
RES-005	Funcionalidades no solicitadas	El sistema no debe implementar funcionalidades que no estén descritas en el enunciado, salvo que faciliten las pruebas.	El desarrollo debe enfocarse exclusivamente en lo solicitado.	Funcionalidades adicionales pueden ser incluidas si ayudan a mejorar las pruebas o la implementación.
RES-006	Estrategia de Pruebas	Se pueden implementar pruebas automatizadas (JUnit) en futuras etapas del proyecto, pero no es obligatorio ahora.	Las pruebas del sistema deben validarse con programas de consola.	En proyectos futuros, se podrá integrar JUnit o pruebas de integración.

d. Programas de prueba

- **Prueba de Cliente–Transacción–Tiquete**
Crear un cliente, simular que compra un tiquete y verificar que queda registrada una transacción.
- **Prueba de Organizador–Evento–Localidad**
Un organizador crea un evento, lo asocia a un venue y define localidades. Mostrar que cada localidad tiene un precio y puede generar tiquetes.
- **Prueba de Tiquete Básico Numerado**
Crear una localidad numerada y generar tiquetes con asiento único. Verificar que no se repiten los números de asiento.
- **Prueba de Tiquete Múltiple / Temporada**
Crear un pase múltiple (ej: palco o pase de temporada).
Confirmar que se asocian varios tiquetes y que el precio no es solo la suma.
- **Prueba de Paquete Deluxe**
Crear un paquete con tiquetes + beneficios adicionales. Mostrar que no puede transferirse.
- **Prueba de Administrador–Venue–Evento**
Administrador aprueba un venue.
Administrador cancela un evento y se genera reembolso al saldo del cliente.
- **Prueba de Oferta**
Organizador crea una oferta sobre una localidad.
Verificar que el precio del tiquete cambia si la fecha está dentro de la ventana de la oferta.

Historias de usuario

Cliente (comprador)

1. **Comprar tiquetes**
Como cliente, quiero comprar tiquetes de eventos, para asistir. Acepta pago por pasarela o con saldo en la plataforma; cada tiquete tiene precio, cargo % de servicio y cargo fijo de emisión; puede haber tope máximo por transacción.
2. **Elegir tipo de tiquete** (*básico, numerado con asiento, múltiple/palco, temporada, Deluxe*)
Como cliente, quiero escoger el tipo de tiquete disponible para el evento, para ajustar precio/beneficios. En localidades numeradas, cada tiquete lleva asiento único. Los múltiples y temporada se venden como agrupación (precio propio, tope aplicado al paquete). Deluxe puede incluir beneficios y cortesías.
3. **Transferir un tiquete**
Como cliente (dueño), quiero transferir un tiquete a otro usuario con su login y mi contraseña, para cederlo. Deluxe no se transfiere. Los múltiples se pueden transferir por entradas o completo, pero no

completo si alguna entrada está vencida o ya transferida.

4. Recibir reembolsos al saldo virtual

Como cliente, quiero recibir reembolsos en mi saldo virtual (inicia en 0) para reutilizarlo en compras cuando aplique una cancelación o calamidad aprobada

Organizador

1. Crear evento

Como organizador, quiero crear un evento con fecha/hora y venue, para publicarlo y vendertiquetes.

2. Configurar localidades del evento

Como organizador, quiero definir localidades con precio y si son numeradas, para habilitar la venta.

3. Generar ofertas por localidad

Como organizador quiero crear una oferta con descuento porcentual y ventana de tiempo, para incentivar la compra.

4. Proponer un venue para mi evento

Como organizador, quiero proponer un venue, para que pueda usarse en mi evento.

5. Solicitar cancelación de evento

Como organizador, quiero solicitar cancelar un evento por insolvencia u otra razón, para detener la venta.

6. Consultar ventas/avance (sin recargos)

Como organizador, quiero ver ingresos sin recargos y % de venta por evento y por localidad, para monitorear desempeño.

Administrador

1. Aprobar venues propuestos

Como administrador, quiero aprobar o rechazar venues sugeridos, para habilitar su uso en eventos.

2. Configurar cargos de la tiquetera

Como administrador, quiero fijar la cuota fija de emisión y el cargo porcentual por servicio (por tipo de evento si corresponde), para que se apliquen en la compra.

3. Cancelar evento y procesar reembolsos

Como administrador, quiero cancelar un evento y aplicar la política de reembolso, para proteger a los compradores según las reglas.

4. Decidir reembolso por calamidad (cliente)

Como administrador, quiero evaluar solicitudes por calamidad, para aprobar o negar el reembolso.

5. Consultar ganancias de la tiquetera

Como administrador, quiero ver ganancias por fecha, evento u organizador, para evaluar el desempeño.

Requerimientos funcionales:

Cliente 1.1

Nombre	RF1. Registrarse como cliente
Resumen	El cliente crea una nueva cuenta en la plataforma para poder usar el sistema.
Entradas	Nombre, login, password
Resultados	Nuevo usuario registrado y además el cliente es creado con saldo 0

Nombre	RF2. comprar tiquete
Resumen	El cliente reserva un tiquete para un evento específico.
Entradas	Evento, localidad, tipo de tiquete, cantidad, pago,
Resultados	Tiquete(s) comprados

Organizador 1.1

Nombre	RF3. Crear evento
Resumen	El organizador crea un nuevo evento para la venta de tiquetes.
Entradas	Fecha, Venue, número de tiquetes
Resultados	Nuevo evento creado

Nombre	RF4. Consultar ganancias
Resumen	El organizador consulta las ganancias de sus eventos, sin incluir cargos de la tiquetera.
Entradas	Ninguna

Resultados	Ganancia por evento
-------------------	---------------------

Nombre	RF5. Generar oferta
Resumen	El organizador crea una oferta con descuento sobre una localidad de un evento.
Entradas	Localidad, porcentaje de descuento, ventana de tiempo
Resultados	Oferta creada y aplicada a la localidad

Administrador 1.1

Nombre	RF6. Ver ganancias
Resumen	El administrador consulta las ganancias netas de la tiquetera.
Entradas	Ninguna
Resultados	Ganancias calculadas

Nombre	RF7. Fijar cargo fijo
Resumen	El administrador define un cargo adicional fijo para cada tiquete vendido.
Entradas	Cargo fijo
Resultados	Cargo aplicado a los tiquetes

Nombre	RF8. Fijar cargo porcentual
Resumen	El administrador define el porcentaje de cargo por servicio para los tiquetes según tipo de evento.
Entradas	Cargo porcentual
Resultados	Cargo aplicado a los tiquetes

Nombre	RF9. Reembolso de tiquete
Resumen	El administrador decide si reembolsar o no un tiquete bajo ciertas circunstancias.
Entradas	Tiquete para reembolsar
Resultados	Monto reembolsado o rechazo del reembolso

Nombre	RF10. Transferir tiquete
Resumen	“Es posible transferir un tiquete indicando el login del receptor y la contraseña del emisor. Los Deluxe NO se pueden transferir
Entradas	Tiquete para reembolsar
Resultados	Monto reembolsado o rechazo del reembolso

Nombre	RF11. Aprobar venues propuestos
Resumen	Los venues pueden ser creados por el administrador o sugeridos dichos venues deben ser aprobados por el administrador antes de usarse.”
Entradas	Tiquete para reembolsar

Resultados	Monto reembolsado o rechazo del reembolso
-------------------	---

Relacion de RF con historia de usuario

Cliente

- RF1 Registrarse
- RF2 Comprar tiquete
- RF3 Transferir tiquete

Organizador

- RF4 Crear evento
- RF5 Configurar localidades / Generar oferta
- RF6 Consultar ganancias

Administrador

- RF7 Aprobar venues
- RF8 Fijar cargo fijo
- RF9 Fijar cargo porcentual
- RF10 Cancelar evento y reembolsar tiquete
- RF11 Consultar ganancias

Descomposición de requisitos funcionales

RF 1 (Registrarse como cliente)

- **registrarCliente(nombre, login, password) -> Cliente.**
Entradas: nombre, login, password.
Salida: objeto Cliente creado (saldo inicial = 0) o error.
- **validarCamposBasicos(nombre, login, password)**
Entradas: nombre, login, password.
Salida: (continúa si es válido; detiene con error si no).
- **normalizarLogin(login) -> loginNormalizado**
Descripción:
Entradas: login escrito por la persona.
Salida: loginNormalizado (p. ej., trim y minúsculas).
Efecto/garantías: el sistema compara y almacena el login en forma consistente.
- **loginDisponible(loginNormalizado) -> boolean**
Entradas: loginNormalizado.
Salida: true si no existe; false si ya está registrado.
Efecto/garantías: solo se continúa con la creación si el login está libre
- **crearCliente(nombre, loginNormalizado, password) -> Cliente**

Descripción:

Construye el objeto de dominio con sus valores iniciales, aplicando las reglas del negocio.

Entradas: nombre, loginNormalizado, password.

Salida: instancia **Cliente** en memoria.

- **guardarCliente(cliente) -> Cliente**

Descripción:

Persiste la entidad creada en el mecanismo de almacenamiento definido para la entrega (lista/archivo, etc.).

Entradas: objeto Cliente recién construido.

Salida: **Cliente** persistido (opcionalmente con identificador asignado).

RF 2 (Comprar como cliente)

- **comprarTiquetes(cliente, evento, localidad, tipoTiquete, cantidad, medioPago)**

Entradas: cliente, evento, localidad, tipoTiquete, cantidad, medioPago

Salida: List<Tiquete>

- **validarDisponibilidad(localidad, tipoTiquete, cantidad)**

Entradas: localidad, tipoTiquete, cantidad

Salida: void

- **validarTopeTransaccion(tipoTiquete, cantidad)**

Entradas: tipoTiquete, cantidad

Salida: void

- **calcularPrecioBase(localidad)**

Entradas: localidad

Salida: Money

- **aplicarOferta(localidad, fechaHoraCompra, subtotal)**

Entradas: localidad, fechaHoraCompra, subtotal

Salida: Money

- **aplicarCargos(tipoEvento, subtotal)**

Entradas: tipoEvento, subtotal

Salida: Money

- **calcularTotal(subtotalConCargos, cantidad, tipoTiquete)**

Entradas: subtotalConCargos, cantidad, tipoTiquete

Salida: Money

- **asignarAsientosSiNumerada(localidad, n)**

Entradas: localidad, n

Salida: List<Integer>

- **emitirTiquetes(cliente, evento, localidad, tipoTiquete, cantidad, asientos?)**

Entradas: cliente, evento, localidad, tipoTiquete, cantidad, asientos?

Salida: List<Tiquete>

- **pagarConSaldo(cliente, total)**

Entradas: cliente, total

Salida: void

- **pagarPorPasarela(total)**

Entradas: total

Salida: boolean

- **registrarTransaccion(cliente, total, medioPago, tiquetes)**

Entradas: cliente, total, medioPago, tiquetes

Salida: Transaccion

RF 3 crear evento (por organizador)

- **crearEvento(fecha, venue, numeroTiquetes) -> Evento**

Entradas: fecha, venue, numeroTiquetes.

Salida: objeto Evento creado (estado ACTIVO) o error.

Descripción: crea un nuevo evento para venta de tiquetes con la fecha y el venue indicados y el stock inicial solicitado.

- **validarFecha(fecha)**

Entradas: fecha.

Salida: continúa si la fecha es válida (no pasada); error si no.

Efecto/garantías: el evento se programa en una fecha aceptable.

- **validarVenueAprobado(venue)**

Entradas: venue.

Salida: continúa si el venue está aprobado; error si no.

Efecto/garantías: solo se usan venues aprobados.

- **validarDisponibilidadFechaEnVenue(venue, fecha)**

Entradas: venue, fecha.

Salida: continúa si el venue no tiene otro evento ese mismo día; error si ya existe.

Efecto/garantías: un venue no aloja dos eventos en la misma fecha.

- **validarNumeroTiquetes(numeroTiquetes)**

Entradas: numeroTiquetes.

Salida: continúa si es > 0 ; error si no.

Efecto/garantías: el stock inicial es positivo.

- **generarIdEvento() -> idEvento**

Entradas: —

Salida: idEvento único.

Efecto/garantías: identificación unico del evento.

- **construirEvento(idEvento, fecha, venue, numeroTiquetes) -> Evento**

Entradas: idEvento, fecha, venue, numeroTiquetes.

Salida: instancia Evento en memoria con estado ACTIVO y stock inicial.

Efecto/garantías: el objeto queda listo para registrarse.

- **registrarEvento(evento) -> Evento**

Entradas: evento.

Salida: evento persistido.

Efecto/garantías: el evento queda almacenado para consultas y operaciones posteriores.

RF 4 consultar ganancias

- **consultarGanancias() -> (evento → ganancia)**
Entradas: —
Salida: colección con pares (Evento, Monto de ganancia del organizador).
Descripción: obtiene todos los eventos del organizador y calcula la ganancia de cada uno **sin incluir cargos de la tiquetera**.
Efecto/garantías: retorna solo valores propios del organizador.
- **obtenerEventosDelOrganizador() -> Lista<Evento>**
Entradas: —
Salida: lista de eventos creados por el organizador.
Efecto/garantías: limita el cálculo a eventos del organizador autenticado.
- **listarTiquetesVendidos(evento) -> Lista<Tiquete>**
Entradas: evento.
Salida: tiquetes emitidos asociados al evento (ventas vigentes).
Efecto/garantías: la base del cálculo incluye únicamente ventas efectivas.
- **calcularGananciaPorEvento(evento, tiquetesVendidos) -> double**
Entradas: evento, tiquetesVendidos.
Salida: monto de ganancia del organizador para ese evento.
Descripción: suma el **precio para el organizador** de cada tiquete vendido (precio del evento/localidad ofrecido al cliente **sin cargos de servicio ni de emisión**).
Efecto/garantías: excluye completamente cargos de la tiquetera.
- **precioOrganizador(tiquete) -> double**
Entradas: tiquete.
Salida: valor atribuible al organizador por ese tiquete (**sin** cargos de servicio/emisión).
Efecto/garantías: aísla el componente que corresponde al organizador para la suma.
- **construirResumenGanancias(evento, monto) -> (evento → ganancia)**
Entradas: evento, monto.
Salida: par (Evento, Ganancia) para la salida final.
Efecto/garantías: estructura uniforme para presentar “ganancia por evento”.
- **RF5 (Generar oferta)**
-
- **generarOferta(idOrganizador, idEvento, idLocalidad, porcentajeDescuento, fechaInicio, fechaFin) -> Oferta**
Entradas: idOrganizador, idEvento, idLocalidad, porcentajeDescuento, fechaInicio, fechaFin.
Salida: Oferta creada y persistida o Error (permisos inválidos / solapamiento / rango inválido).
Descripción: valida que el organizador sea propietario y que la ventana sea coherente; registra oferta para uso en cálculos de precio.
Efecto/garantías: oferta disponible durante su ventana; porcentaje en rango válido; evita solapamientos según política establecida.
- **validarEventoPerteneceAOrganizador(idEvento, idOrganizador)**
Entradas: idEvento, idOrganizador.

Salida: boolean o error.

Efecto/garantías: impide crear oferta por organizador que no es dueño.

- validarPorcentaje(porcentajeDescuento)

Entradas: porcentajeDescuento.

Salida: continúa si $0 < p \leq 100$; error si no.

Efecto/garantías: evita descuentos inválidos.

- validarVentana(fechaInicio, fechaFin, fechaEvento)

Entradas: fechaInicio, fechaFin, fechaEvento.

Salida: continúa si coherente; error si no.

Efecto/garantías: ventana lógica y, opcionalmente, $\text{fechaFin} \leq \text{fechaEvento}$.

- verificarSolapamientoOfertas(idEvento, idLocalidad, fechaInicio, fechaFin) -> boolean

Entradas: datos de ventana.

Salida: true si solapa (según política), false si no.

Efecto/garantías: evita políticas de solapamiento no permitidas.

- construirOferta(...) -> Oferta

Entradas: parámetros de oferta.

Salida: objeto Oferta en memoria listo para persistir.

Efecto/garantías: contiene la ventana y porcentaje.

- guardarOferta(oferta) -> Oferta

Entradas: oferta.

Salida: Oferta persistida.

Efecto/garantías: disponible para cálculo de precios en compras.

- **RF6 (Ver ganancias — Tiquetera / Administrador)**

- consultarGananciasPlataforma(filtro) -> GananciasReport

Entradas: filtro opcional (fechaDesde, fechaHasta, idEvento, idOrganizador).

Salida: GananciasReport (totalIngresos, ingresosPorServicio, cuotasEmision, netoTiquetera, detalle por evento/organizador).

Descripción: agrega transacciones dentro del filtro y calcula componentes; descuenta reembolsos.

Efecto/garantías: ganancias calculadas = $\text{suma}(\text{cargoServicio} + \text{cuotaEmision})$ menos reembolsos; datos consistentes con transacciones; operación de solo lectura (no modifica base).

- listarTransacciones(filtro) -> List<Transaccion>

Entradas: filtro.

Salida: lista de transacciones relevantes.

Efecto/garantías: fuente de datos para el cálculo.

- calcularComponentesTransaccion(transaccion) -> {precioBase, cargoServicio, cuotaEmision}

Entradas: transaccion.

Salida: desglose numérico.

Efecto/garantías: permite separar lo atribuible a la tiquetera.

- agregarPorEventoYOrganizador(transacciones) -> Map

Entradas: lista de transacciones.

Salida: agregados por evento/organizador.

Efecto/garantías: desagregación correcta para reportes.

- `calcularNetoTiquetera(agregados) -> double`
Entradas: agregados.
Salida: neto final de la tiquetera.
Efecto/garantías: excluye transacciones reembolsadas.
- **RF7 (Fijar cargo fijo — Cuota de emisión)**
 - `setCuotaEmision(adminId, cuotaFija) -> Configuración`
Entradas: adminId, cuotaFija (≥ 0).
Salida: configuración actualizada y persistida o Error.
Descripción: valida admin y monto; persiste configuración que afectará compras futuras.
Efecto/garantías: valor persistido; aplica de forma inmediata a compras nuevas (no retroactivo salvo especificación).
 - `validarSesionAdmin(adminId)`
Entradas: adminId.
Salida: boolean o error.
Efecto/garantías: solo admin puede cambiar configuración.
 - `actualizarConfiguracion("cuotaEmision", cuotaFija)`
Entradas: cuotaFija.
Salida: configuración actualizada en memoria.
Efecto/garantías: lectura futura recupera ese valor.
 - `persistirConfiguracion()`
Entradas: —
Salida: configuración guardada en archivo.
Efecto/garantías: durable tras reinicios.
- **RF8 (Fijar cargo porcentual — Cargo por servicio)**
 - `setCargoPorcentual(adminId, tipoEvento, porcentaje) -> Configuración`
Entradas: adminId, tipoEvento (o global), porcentaje ($0 \leq p \leq 100$).
Salida: configuración actualizada o Error.
Descripción: valida y persiste regla de cargo porcentual para tipo de evento.
Efecto/garantías: porcentaje aplicado en compras posteriores; soporta override por tipo.
 - `validarPorcentaje(porcentaje)`
Entradas: porcentaje.
Salida: continúa si válido; error si no.
Efecto/garantías: evita valores fuera de rango.
 - `actualizarConfiguracion("cargoPorcentual", tipoEvento, porcentaje)`
Entradas: tipoEvento, porcentaje.
Salida: configuración en memoria.
Efecto/garantías: persistida y utilizada en cálculos posteriores.

- **RF9 (Reembolso de ticket)**

- procesarReembolso(adminId, idTicket, motivo, tipoReembolso) -> ReembolsoResult
Entradas: adminId, idTicket, motivo, tipoReembolso (CANCELACION_ADMIN | CANCELACION_ORG_AUTORIZADA | CALAMIDAD).
Salida: ReembolsoResult {exito, montoReembolsado, destino} o Error.
Descripción: valida admin, obtiene transacción y calcula monto según regla; actualiza saldo/estado y registra reembolso.
Efecto/garantías: actualiza estado de ticket y transacción; actualiza saldo virtual o emite reembolso según política; aplica reglas:
 - CANCELACION_ADMIN → reembolsa precioPagado - cuotaEmision;
 - CANCELACION_ORG_AUTORIZADA → reembolsa precioBase;
 - CALAMIDAD → a discreción del admin;registro persistente del reembolso.
- obtenerTicket(idTicket) -> Ticket
Entradas: idTicket.
Salida: Ticket o NotFound.
Efecto/garantías: base para cálculo.
- obtenerTransaccionPorTicket(idTicket) -> Transaccion
Entradas: idTicket.
Salida: Transaccion.
Efecto/garantías: contiene monto pagado y desglose.
- calcularMontoReembolso(transaccion, tipoReembolso) -> double
Entradas: transaccion, tipoReembolso.
Salida: monto numérico.
Efecto/garantías: aplica regla documental y ajustes por reembolsos previos.
- actualizarSaldoCliente(idCliente, monto)
Entradas: idCliente, monto.
Salida: saldo actualizado persistente.
Efecto/garantías: consistente y trazable.
- registrarReembolso(transaccion, monto, adminId, motivo)
Entradas: transaccion, monto, adminId, motivo.
Salida: registro persistente.
Efecto/garantías: auditoría completa.

- **RF10 (Transferir ticket)**

- transferirTicket(idTicket, loginEmisor, passwordEmisor, loginReceptor, opcionTransferencia) -> TransferResult
Entradas: idTicket (o idPaquete), loginEmisor, passwordEmisor, loginReceptor, opcionTransferencia (INDIVIDUAL/COMPLETO).
Salida: TransferResult {exito, mensaje} o Error.
Descripción: autentica emisor, verifica reglas de transferibilidad (Deluxe no transferible; paquetes con

restricciones) y actualiza propietario en persistencia; registra historial.

Efecto/garantías: solo con autenticación del emisor; no permite transferir tiquetes Deluxe; para paquetes completos garantiza que ningún sub-tiquete esté vencido o transferido; propietario actualizado y registro en historial.

- autenticarUsuario(login, password) -> boolean
Entradas: login, password.
Salida: true si credenciales correctas.
Efecto/garantías: seguridad sobre la operación.
- buscarTiquete(idTiquete) -> Tiquete
Entradas: idTiquete.
Salida: Tiquete.
Efecto/garantías: obtener estado y tipo.
- verificarTransferibilidad(tiquete, opcion) -> boolean
Entradas: tiquete, opcionTransferencia.
Salida: true/false.
Efecto/garantías: aplica reglas (Deluxe, vencimiento, transferencias previas).
- actualizarPropietario(tiquete, loginReceptor)
Entradas: tiquete, loginReceptor.
Salida: persistencia actualizada.
Efecto/garantías: propietario actualizado y consistente.
- registrarHistorialTransferencia(...)
Entradas: datos de la operación.
Salida: registro persistido.
Efecto/garantías: trazabilidad.
- **RF11 (Aprobar venues propuestos)**
- aprobarVenue(adminId, idVenue, decision, comentarios) -> Venue
Entradas: adminId, idVenue, decision (APROBAR/RECHAZAR), comentarios (opcional).
Salida: Venue actualizado (estado APROBADO/RECHAZADO) o Error.
Descripción: valida admin, cambia estado del venue y persiste; notifica al promotor si aplica.
Efecto/garantías: solo venues APROBADOS pueden asociarse a eventos activos; decisión y comentarios quedan registrados; cambios aplican en validaciones futuras.
- buscarVenue(idVenue) -> Venue
Entradas: idVenue.
Salida: Venue o NotFound.
Efecto/garantías: base para decisión.
- cambiarEstadoVenue(venue, estado, comentarios)
Entradas: venue, estado, comentarios.
Salida: venue actualizado en memoria.
Efecto/garantías: persistencia y disponibilidad inmediata para validaciones.
- guardarVenue(venue) -> Venue

Entradas: venue.

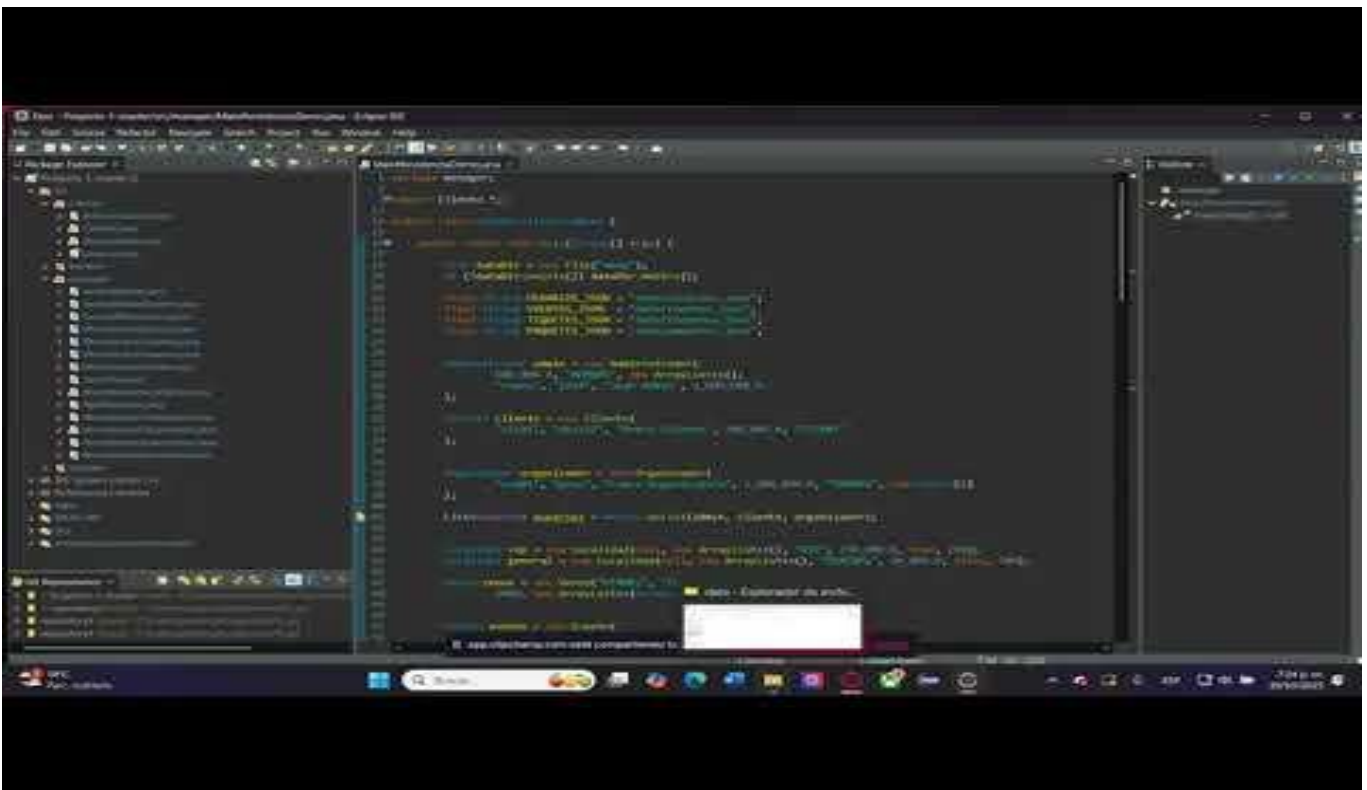
Salida: Venue persistido.

Efecto/garantías: durable y consultable.

- `notificarPromotor(promotorId, decision, comentarios)`
Entradas: `promotorId`, `decision`, `comentarios`.
Salida: notificación enviada (opcional).
Efecto/garantías: promotor informado del estado del venue.

Persistencia y funcionamiento en el proyecto

Realizamos un video para entender el funcionamiento de la persistencia y el correcto funcionamiento de este en java https://www.youtube.com/watch?v=ZCz179d_MS0



De la libreria que se habla en el video es <https://repo1.maven.org/maven2/org/json/json/20240303/>

Esto demuestra que el programa funciona correctamente es evidente que falta que persisten los metodos, esto se resolvera cuando tengamos un menu ya que es mas evidente y organizado de esta manera pero para esta primera version los datos persisten.

Proyecto 2: pruebas y despliegue

Historias de usuario para el nuevo requerimiento de Market Place

Cliente (CL)

CL-01. Registrarse como cliente

Como visitante

quiero crear una cuenta de cliente

para poder comprar/vender boletas en el Marketplace.

- **Entradas:** nombre, login, password.
- **Salidas:** cuenta creada, sesión iniciada (o mensaje de éxito).
- **Criterios de aceptación:**
 - Se valida unicidad de login.
 - El saldo inicial = 0.
 - Se puede iniciar sesión luego con login/password.

CL-02. Publicar una oferta de reventa

Como cliente vendedor

quiero publicar una boleta con precio de venta

para revenderla a otro usuario.

- **Entradas:** ID de boleta, precio de venta.
- **Salidas:** confirmación de oferta publicada; entrada en log.
- **Criterios de aceptación:**
 - Solo boletas **no** pertenecientes a paquete **Deluxe**.
 - Estado inicial de la oferta = **ACTIVA**.
 - Se registra en log: usuario, oferta, precio, fecha/hora.

CL-03. Consultar ofertas disponibles

Como cliente

quiero ver ofertas activas de otros

para decidir compra o contraoferta.

- **Entradas:** filtros opcionales (evento/ciudad/rango).

- **Salidas:** listado con precio, evento, fecha, vendedor.
- **Criterios de aceptación:**
 - No aparecen ofertas de boletas **Deluxe**.
 - No aparecen mis propias ofertas.
 - Solo estado **ACTIVA**.

CL-04. Hacer una contraoferta

Como cliente comprador
quiero proponer un nuevo precio
para negociar la compra.

- **Entradas:** ID oferta, valor contraoferta.
- **Salidas:** confirmación; entrada en log.
- **Criterios de aceptación:**
 - No se contraoferta en **Deluxe**.
 - Contraoferta queda **PENDIENTE**.
 - Se registra en log usuario, oferta, monto, fecha/hora.

CL-05. Aceptar una contraoferta recibida

Como cliente vendedor
quiero aceptar una contraoferta
para concretar la venta.

- **Entradas:** ID de contraoferta.
- **Salidas:** confirmación de venta; actualización de saldos; log.
- **Criterios de aceptación:**
 - Saldo: +monto al vendedor, -monto al comprador (si aplica).
 - Transferir propiedad de la boleta al comprador.
 - Log con compradores/vendedores, boleta y precio.

CL-06. Rechazar una contraoferta

Como cliente vendedor
quiero rechazar contraofertas no deseadas
para mantener el precio.

- **Entradas:** ID de contraoferta.
- **Salidas:** confirmación; log.
- **Criterios de aceptación:**
 - Estado contraoferta → **RECHAZADA**.
 - Registro en log con fecha/hora.

CL-07. Comprar a precio publicado (aceptar oferta)

Como cliente comprador

quiero aceptar directamente el precio publicado
para comprar sin negociar.

- **Entradas:** ID de la oferta.
- **Salidas:** confirmación; actualización de saldos; log.
- **Criterios de aceptación:**
 - Solo ofertas **ACTIVAS** y no-**Deluxe**.
 - Saldo actualizado; transferencia de propiedad.
 - Registro en log.

CL-08. Borrar/retirar mi oferta

Como cliente

quiero retirar una oferta mía
para dejar de vender esa boleta.

- **Entradas:** ID de la oferta.
- **Salidas:** confirmación; log.
- **Criterios de aceptación:**
 - Solo el **dueño** de la oferta puede borrarla.
 - La oferta pasa a estado **BORRADA_POR_CLIENTE** (o equivalente).
 - El log deja trazabilidad de que **existió** y fue borrada (fecha/hora).

CL-09. Ver mis ofertas activas

Como cliente

quiero ver mis ofertas activas
para seguir su estado.

- **Entradas:** ninguna.
- **Salidas:** lista de mis ofertas con estado actual.
- **Criterios de aceptación:**
 - Incluye **ACTIVAS** y con contraofertas en curso.
 - Excluye **borradas** y **vendidas**.

CL-10. Ver mis transacciones

Como cliente

quiero ver mis compras y ventas
para llevar control de movimientos.

- **Entradas:** ninguna.
- **Salidas:** lista con fecha, tipo (compra/venta), monto y estado.
- **Criterios de aceptación:**
 - Solo transacciones **completadas**.
 - Cada transacción debe existir en el log.

CL-11. Ver mi historial (trazabilidad)

Como cliente

quiero revisar el historial de mis acciones
para tener transparencia.

- **Entradas:** ninguna.
- **Salidas:** lista cronológica con fecha/hora, acción y descripción.
- **Criterios de aceptación:**
 - Deriva del **log** del sistema.
 - Muestra tipo de acción, usuarios involucrados y timestamp.

CL-12. Usar saldo para comprar

Como cliente

quiero pagar con mi saldo
para completar compras en Marketplace o plataforma original.

- **Entradas:** monto de compra, medio = **saldo**.
- **Salidas:** confirmación, nuevo saldo.
- **Criterios de aceptación:**
 - No se modela pasarela; solo saldo interno.
 - Compra falla si saldo insuficiente.

Administrador (AD)

AD-01. Aprobar nuevos promotores

Como administrador

quiero aprobar a los promotores registrados
para habilitar su uso del sistema como organizadores.

- **Entradas:** lista de solicitudes, selección de promotor.
- **Salidas:** promotor **APROBADO**; notificación breve.
- **Criterios de aceptación:**
 - Solo el **administrador** puede cambiar estado de promotor a **APROBADO**.
 - El promotor aprobado puede iniciar sesión como organizador.

AD-02. Consultar el log del Marketplace

Como administrador

quiero ver el log completo
para auditar ofertas, contraofertas, rechazos y transacciones.

- **Entradas:** filtros opcionales (rango de fechas, tipo de acción, usuario).
- **Salidas:** lista cronológica con fecha/hora, tipo, descripción.

- **Criterios de aceptación:**
 - Solo el administrador puede acceder al log.
 - Debe traer eventos: **publicación/contraoferta/aceptación/rechazo/venta/borrado por cliente/eliminación por admin.**

AD-03. Eliminar una oferta

Como administrador
quiero eliminar una oferta publicada
para moderar el Marketplace cuando sea necesario.

- **Entradas:** ID de oferta, motivo opcional.
- **Salidas:** confirmación; registro en log.
- **Criterios de aceptación:**
 - La oferta pasa a estado **ELIMINADA_POR_ADMIN**.
 - El log registra la eliminación con fecha/hora y responsable (admin).

Promotor / Organizador (PR)

PR-01. Registrarse como promotor

Como visitante
quiero registrarme como promotor
para solicitar acceso como organizador.

- **Entradas:** nombre, login, password, datos de promotor.
- **Salidas:** solicitud registrada con estado **PENDIENTE_APROBACIÓN**.
- **Criterios de aceptación:**
 - Unicidad de login.
 - No tiene permisos de organizador hasta ser **aprobado**.

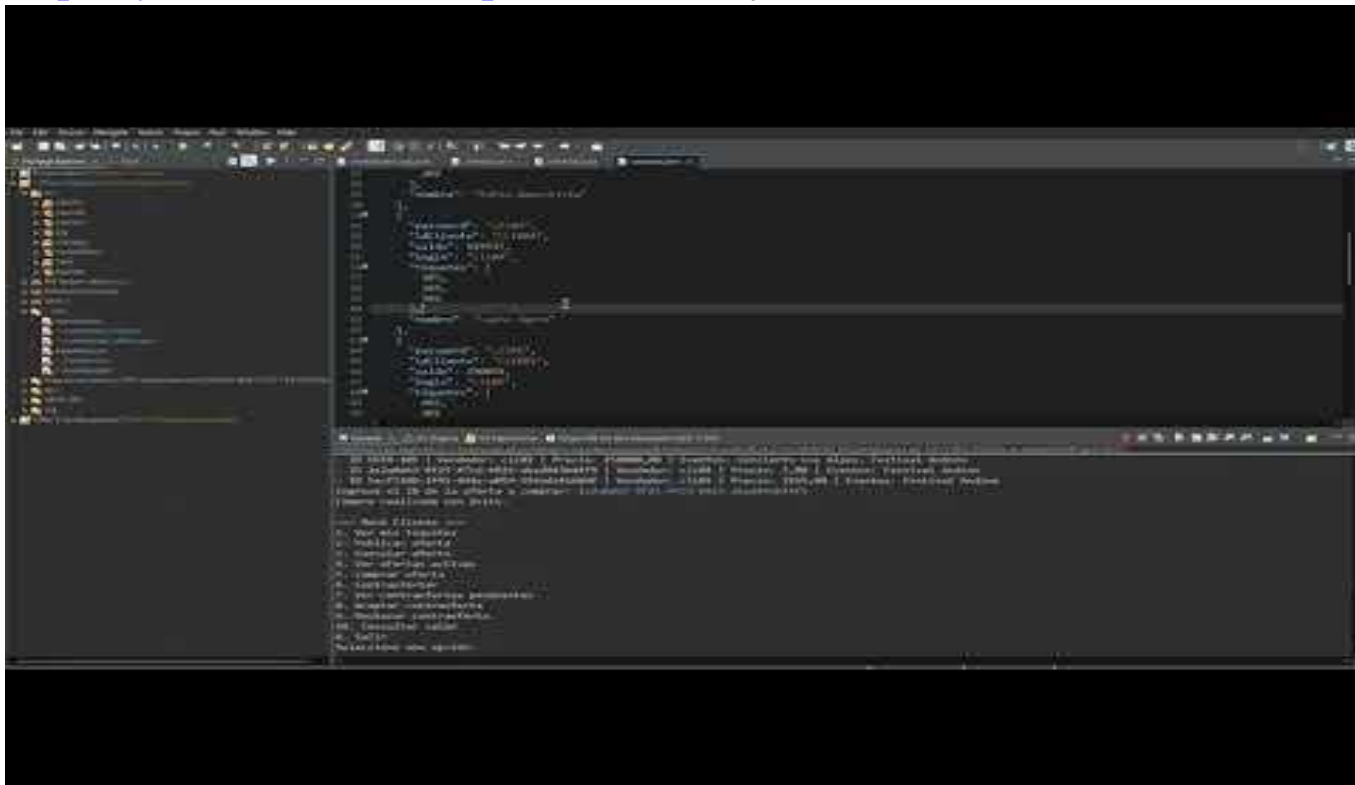
PR-02. Ser aprobado por el administrador

Como promotor
quiero que el admin apruebe mi solicitud
para poder operar como organizador.

- **Entradas:** ninguna (evento disparado por admin).
- **Salidas:** cambio de estado a **APROBADO**.
- **Criterios de aceptación:**
 - Solo admin aprueba.
 - Tras aprobación, puede autenticarse como organizador.

Tutorial para hacer pruebas del menú de market place:

<https://youtu.be/OZEkQeq1Fs4> video abajo



Para hacer las pruebas y cambiar de menus rapido es darle a la felchita y entras rapido a los menus

