Documento de Diseño del Sistema de Venta de Tiquetes – BoletaMaster

Índice

1. Introducción

1.1 Propósito del documento

El presente documento tiene como finalidad exponer el diseño detallado del sistema BoletaMaster, una plataforma destinada a simular el funcionamiento de un sistema de venta de tiquetes para conciertos y distintos tipos de eventos.

Este informe corresponde a la segunda entrega del proyecto y su objetivo principal es evidenciar las decisiones de diseño, los modelos conceptuales y los diagramas UML que orientarán la implementación del sistema.

Asimismo, busca mantener la coherencia con el análisis realizado en la Entrega 1 (modelo de dominio), asegurando la trazabilidad entre los requerimientos funcionales, las clases, los métodos y las interacciones más relevantes dentro del sistema.

1.2 Alcance del sistema

El sistema BoletaMaster está diseñado para gestionar eventos y la venta de tiquetes, involucrando tres tipos de usuarios principales:

- Clientes: adquieren, consultan y transfieren tiquetes.
- Organizadores: crean eventos, configuran localidades, generan ofertas y consultan reportes financieros.
- Administradores: aprueban venues, definen parámetros de cobro y supervisan los aspectos financieros de la plataforma.

El alcance del diseño incluye:

- Gestión de usuarios con autenticación segura.
- Administración de eventos, localidades y venues.
- Control de transacciones y transferencias de tiquetes.
- Configuración de parámetros financieros y generación de reportes.

No se contempla la integración con pasarelas de pago reales ni el desarrollo de una interfaz gráfica, ya que esta entrega se centra exclusivamente en definir la estructura lógica y funcional del sistema.

1.3 Contexto general del sistema

BoletaMaster se enmarca dentro del dominio de los sistemas de venta de tiquetes digitales, encargándose de los procesos de compra, validación, transferencia y gestión de eventos.
La plataforma actúa como intermediaria entre los diferentes actores del sistema:

- Los clientes, que compran o transfieren entradas.
- Los organizadores, que gestionan y ofrecen los eventos.

- El administrador, quien supervisa el funcionamiento general del sistema y controla los cobros asociados.

Restricciones técnicas principales:

- Toda la información debe mantenerse de forma persistente en archivos.
- Los tiquetes no transferibles y los paquetes *Deluxe* no pueden cambiar de propietario.
- Los eventos deben ser aprobados antes de habilitarse para la venta.
- La aplicación debe desarrollarse en Java, garantizando portabilidad, claridad y robustez en la implementación.

2. Objetivos del diseño

2.1 Objetivos principales

1. Diseñar una arquitectura orientada a objetos que soporte todas las funcionalidades descritas en el enunciado.
2. Elaborar diagramas de clases detallados con sus atributos, métodos y relaciones.
3. Representar la estructura general del sistema mediante un diagrama de clases de alto nivel.
4. Modelar las interacciones críticas a través de diagramas de secuencia.
5. Asegurar que el diseño propuesto sea modular, extensible y coherente con su futura implementación en Java.

2.2 No-objetivos

- No se desarrollará una interfaz gráfica ni integración con pasarelas de pago reales.
- No se abordarán aspectos de infraestructura ni despliegue.
- No se realizarán pruebas de rendimiento ni optimizaciones avanzadas.

3. Contexto y alcance del sistema

3.1 Descripción general

El sistema BoletaMaster tiene como propósito administrar de manera integral todas las operaciones relacionadas con la compra, gestión y control financiero de eventos.

Los organizadores son responsables de crear y configurar los eventos, definiendo distintos tipos de tiquetes: individuales, múltiples, de temporada o paquetes Deluxe.

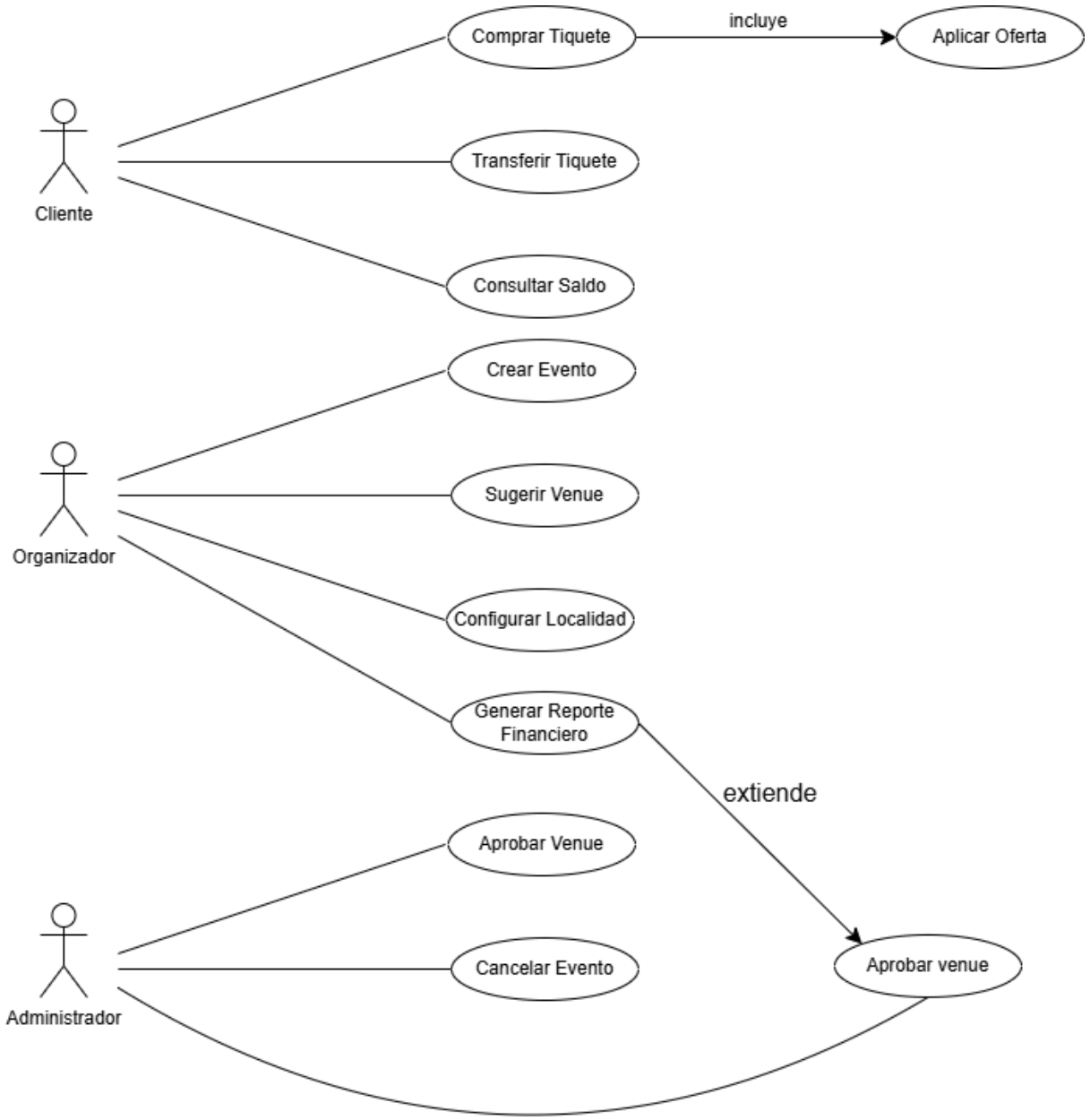
Por su parte, los clientes pueden adquirir y transferir tiquetes, mientras que los administradores supervisan el funcionamiento general del sistema y establecen las tarifas y políticas de cobro.

El diseño garantiza la trazabilidad de cada operación, asegurando la integridad, autenticación y seguridad de la información a lo largo de todo el proceso.

3.2 Actores y sus responsabilidades

Actor	Descripción	Responsabilidades principales
Cliente	Usuario comprador que interactúa con la plataforma para adquirir o transferir tiquetes.	Comprar tiquetes, transferir tiquetes, consultar saldo.
Organizador	Promotor de eventos que gestiona los conciertos y crea ofertas.	Crear eventos, configurar localidades, generar reportes financieros.
Administrador	Supervisor general de la plataforma.	Aprobar venues, configurar cobros, cancelar eventos.

3.3 Diagrama de contexto del sistema



El diagrama de requerimientos funcionales del sistema BoletaMaster representa las principales acciones que el sistema debe realizar según el rol de cada usuario.

El Cliente puede comprar, transferir y consultar tiquetes, además de aplicar ofertas durante la compra.

El Organizador tiene la capacidad de crear eventos, configurar localidades, sugerir venues y generar reportes financieros.

Por su parte, el Administrador es responsable de aprobar venues y cancelar eventos, además de definir parámetros financieros del sistema.

Las relaciones *incluye* y *extiende* ilustran dependencias entre funciones, reflejando el comportamiento esperado del sistema sin entrar en detalles técnicos de implementación.

4. Elementos del Análisis

4.1. Reglas del dominio y supuestos del negocio

Identificación única de tiquetes: cada tiquete debe tener un identificador irrepetible que evite falsificaciones.

Roles de usuario:

- **Cliente:** compra, transfiere y consulta tiquetes.
- **Organizador:** crea eventos, configura localidades, genera reportes y propone venues.
- **Administrador:** aprueba venues, define cobros y puede cancelar eventos.

Eventos y venues:

- Todo evento debe estar vinculado a un venue.
- Un venue solo puede tener un evento por fecha.
- Los venues sugeridos por organizadores deben ser aprobados por el administrador.

Localidades y precios:

- Las localidades determinan el precio del tiquete.
- Los tiquetes de una misma localidad comparten precio.
- Una localidad puede ser numerada o general.

Tipos de tiquetes y paquetes:

- Los tiquetes pueden ser **individuales, múltiples o de temporada**.
- Los **paquetes Deluxe** incluyen tiquetes, mercancía o beneficios adicionales.
- Los tiquetes dentro de un paquete Deluxe **no pueden transferirse**.

Transferencia de tiquetes:

- Solo los tiquetes **individuales o múltiples** son transferibles.
- No pueden transferirse tiquetes vencidos o previamente transferidos.
- Se debe validar la identidad mediante login y contraseña.

Restricciones de compra:

- Existe un límite de tiquetes por transacción.
- En tiquetes múltiples, el límite aplica al conjunto, no a los individuales.

Manejo de ofertas:

- El organizador puede aplicar descuentos por localidad y por tiempo limitado.
- Las ofertas solo afectan compras dentro del rango establecido.

Gestión financiera:

- El administrador define los porcentajes de cobro y la cuota fija de emisión.
- Los organizadores pueden consultar ventas y ganancias por evento o localidad.
- Los reembolsos se acreditan al saldo virtual del cliente.

Seguridad y autenticación:

- Todo usuario debe autenticarse con login y contraseña.
- Las operaciones sensibles deben registrarse para garantizar trazabilidad.

4.2. Restricciones relevantes del proyecto

- La información debe almacenarse en archivos fuera del código fuente.
- El desarrollo se implementará en Java.
- No se incluye interfaz gráfica en esta entrega.
- Solo puede existir un evento por venue y fecha.
- El administrador no puede comprar tiquetes.
- Los paquetes Deluxe son intransferibles.
- Los tiquetes vencidos o ya transferidos no pueden volver a transferirse.

4.3. Requerimientos funcionales (qué debe hacer el sistema)

- Permitir a los clientes comprar, transferir y consultar tiquetes.
- Permitir a los organizadores crear eventos, sugerir venues, configurar localidades y generar reportes.
- Permitir al administrador aprobar venues, configurar cobros y cancelar eventos.
- Registrar todas las transacciones con precios, recargos y comisiones.
- Aplicar ofertas temporales sobre localidades.
- Controlar los límites de compra y gestionar reembolsos cuando sea necesario.

4.4. Requerimientos no funcionales (calidad, seguridad, rendimiento, etc.)

- Persistencia: los datos deben almacenarse y recuperarse sin pérdida.
- Seguridad: autenticación por login y contraseña.
- Integridad: cada transacción debe mantener consistencia en los datos.
- Escalabilidad: soporte para múltiples usuarios y eventos simultáneos.
- Disponibilidad: acceso continuo para las operaciones.
- Mantenibilidad: código modular siguiendo principios SOLID.

5. Diseño del Sistema

5.1. Justificación de las decisiones clave de diseño

El diseño se estructuró en paquetes modulares para lograr alta cohesión y bajo acoplamiento. Las clases se agrupan en cuatro paquetes principales: Usuarios, Eventos, Transacciones y Financiero.

Esta organización facilita la comprensión, el mantenimiento y la futura implementación de una interfaz.

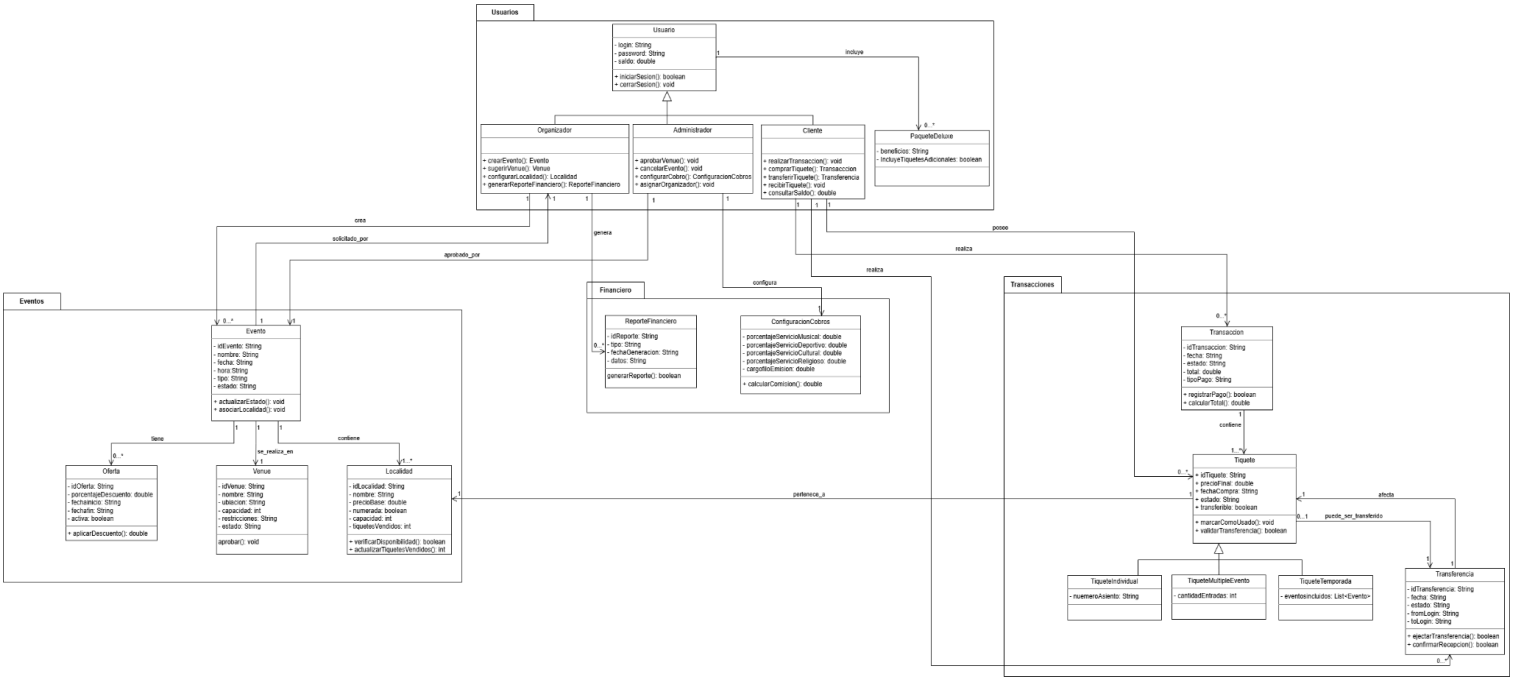
El modelo sigue principios SOLID, priorizando la responsabilidad única y la inversión de dependencias, lo que promueve flexibilidad y escalabilidad.

5.2. Estructura general del sistema (paquetes y módulos)

- **Usuarios:** Cliente, Organizador, Administrador, PaqueteDeluxe.
- **Eventos:** Evento, Localidad, Venue, Oferta.
- **Transacciones:** Tiquete, TiqueteIndividual, TiqueteMultipleEvento, TiqueteTemporada, Transaccion, Transferencia.
- **Financiero:** ConfiguracionCobros, ReporteFinanciero.

5.3. Diagrama de clases de diseño (1.a)

El diagrama de clases de diseño presenta todas las clases con sus atributos, métodos y relaciones.Permite visualizar las interacciones entre los principales componentes del sistema, las dependencias entre paquetes y la estructura base que soportará la implementación en Java.

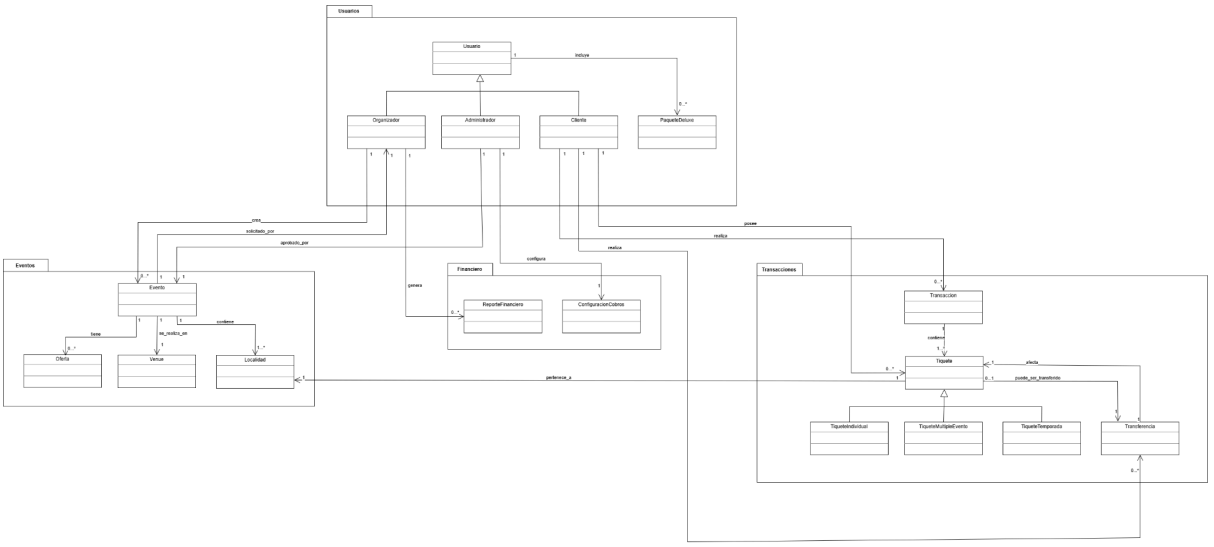


5.4. Diagrama de clases de alto nivel (1.b)

Este diagrama ofrece una visión general simplificada del sistema BoletaMaster. En él se muestran únicamente las clases y sus relaciones más relevantes, sin detallar atributos ni métodos.

Su objetivo es facilitar la comprensión de la arquitectura global, evidenciando cómo los paquetes (Usuarios, Eventos, Transacciones y Financiero) interactúan entre sí dentro de la

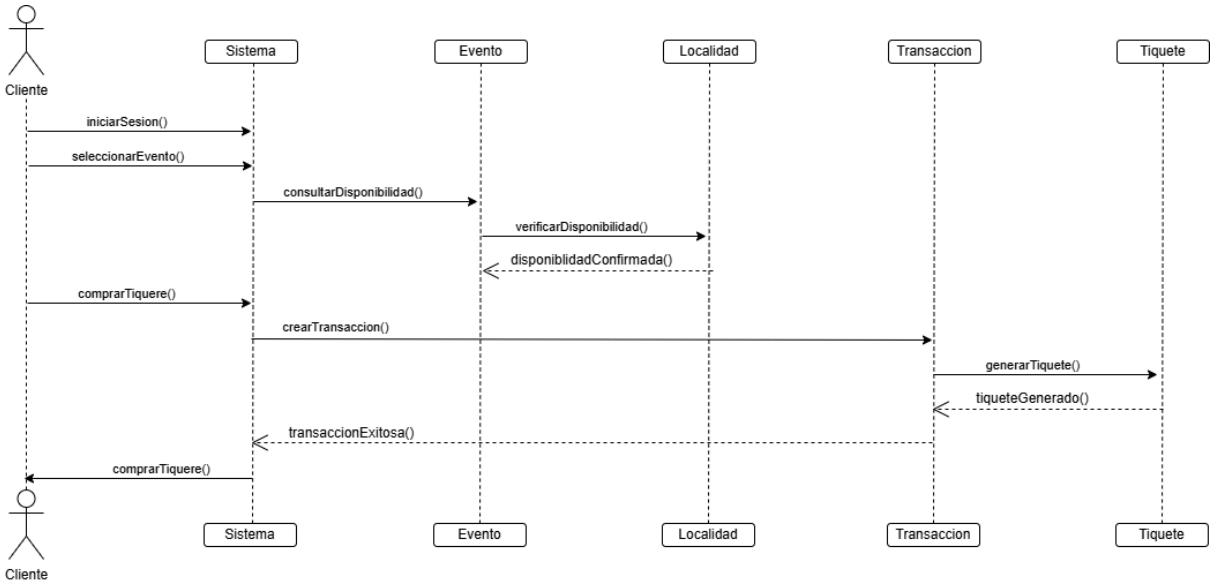
estructura del sistema.



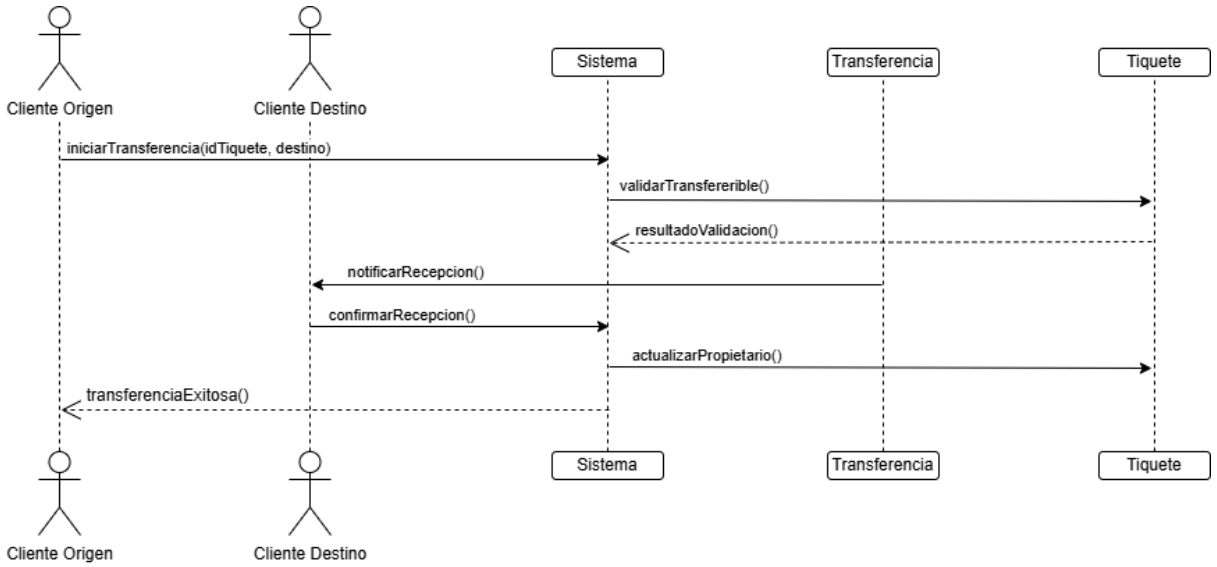
5.5. Diagramas de secuencia (1.c)

Se elaboraron diagramas de secuencia para representar las funcionalidades críticas del sistema, destacando la interacción dinámica entre los objetos involucrados:

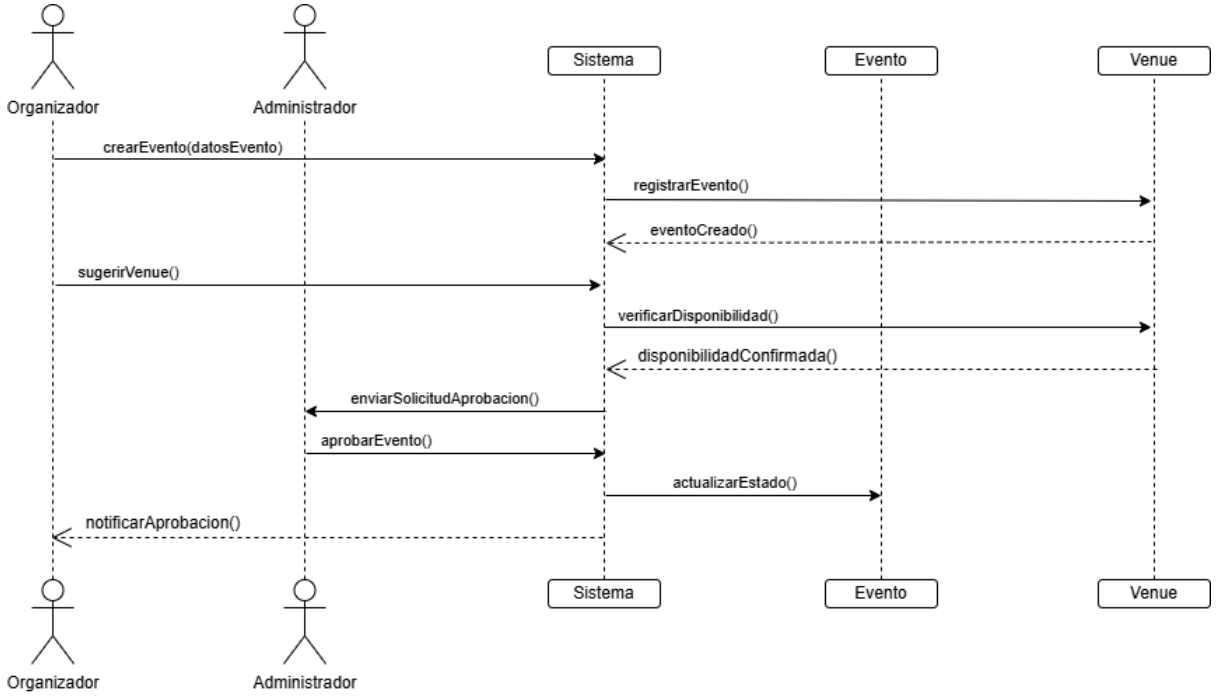
- Compra de tiquete: ilustra la comunicación entre el Cliente, la clase Transacción y la emisión del Tiquete, mostrando cómo se valida el saldo y se genera la compra.



- Transferencia de tiquete: describe el proceso de verificación de identidad, confirmación de la transferencia y cambio de titularidad entre usuarios.



- Creación de evento: muestra la colaboración entre el Organizador, el Venue y el Administrador durante el registro y aprobación de un nuevo evento.



5.6. Descripción de relaciones entre componentes y dependencias

Las clases de los distintos paquetes se comunican mediante asociaciones bien definidas.
Por ejemplo:

- Evento depende de Venue y Localidad para su configuración.
- Transacción interactúa con Cliente y Tiquete para gestionar las compras.
- Administrador colabora con ConfiguraciónCobros y ReporteFinanciero para el control financiero.

Esta distribución modular permite mantener la independencia de cada componente, garantizando que los cambios en un módulo no afecten el funcionamiento de los demás.

6. Persistencia de Datos

6.1. Entidades que requieren almacenamiento persistente

Las siguientes entidades se guardan en archivos para conservar la información entre sesiones del sistema:

- Usuario (subtipos: Cliente, Organizador, Administrador).
- Evento (datos básicos: fecha, hora, estado).
- Venue (identificador, ubicación, capacidad y estado).
- Localidad (id, nombre, numeración, precio y capacidad).
- Tiquete (id único, tipo, precio, fecha de compra, estado, transferible).
- Transacción (id, monto total, tipo de pago, estado).
- Transferencia (id, logins del emisor y receptor, fecha y estado).
- Oferta (id, descuento, fechas y estado).
- Configuración de cobros (porcentajes y tarifas fijas).
- Reporte financiero (id, tipo y datos consolidados).

6.2. Estrategia de manejo de transacciones

Todas las operaciones críticas como compras, transferencias o cancelaciones se manejan como transacciones atómicas.
Para ello:

- Se usa un registro temporal (log de respaldo) para revertir operaciones fallidas.
- Cada operación se valida antes de modificar los archivos persistentes.
- Se asegura la consistencia entre entidades relacionadas, por ejemplo, actualizando los tiquetes vendidos en una localidad tras cada compra.

7. Preocupaciones Transversales

7.1. Seguridad (autenticación, control de acceso)

- Todo usuario se autentica mediante login y contraseña.

- Se aplica control por roles:

  - Solo el Administrador puede aprobar venues o cancelar eventos.
  - Solo el Organizador puede crear y configurar eventos.

- Las contraseñas se almacenan con hash encriptado.

- Las operaciones sensibles se registran en un log de auditoría.

7.2. Transaccionalidad (integridad en operaciones financieras)

- Cada compra o reembolso se procesa como una transacción indivisible y segura.
- Se valida el saldo disponible antes de confirmar pagos o transferencias.
- Se evita la duplicación de registros y se garantiza la integridad de la información financiera ante fallos.

7.3. Escalabilidad y concurrencia

- La arquitectura modular permite añadir nuevos tipos de eventos o tiquetes sin afectar otros componentes.
- El acceso a archivos persistentes se sincroniza para prevenir conflictos de concurrencia.
- En futuras versiones, podrá migrar a una base de datos relacional sin alterar la lógica de negocio.

7.4. Mantenibilidad y extensibilidad del diseño

- Las clases aplican los principios SOLID, garantizando modularidad y bajo acoplamiento.
- La división por paquetes (Usuarios, Eventos, Transacciones, Financiero) facilita la comprensión y el mantenimiento.
- Nuevas funcionalidades (por ejemplo, nuevos tipos de ofertas o reportes) pueden incorporarse mediante herencia o composición.

8. Principios y Estilos de Diseño Adoptados

8.1. Estilo de control

Se optó por un modelo de control mixto:

- Centralizado, cuando el Administrador gestiona operaciones críticas como aprobaciones o cancelaciones.
- Delegado, cuando el Cliente u Organizador ejecutan funciones operativas, como compras o creación de eventos.

8.2. Aplicación de principios SOLID

- **Responsabilidad Única:** cada clase cumple una función concreta (por ejemplo, *Evento* gestiona únicamente información de eventos).
- **Abierto/Cerrado:** el sistema admite extensiones sin modificar su estructura base.
- **Sustitución de Liskov:** las subclases (*TiqueteIndividual*, *TiqueteMultipleEvento*) pueden reemplazar a *Tiquete* sin alterar el comportamiento esperado.
- **Segregación de Interfaces:** los roles (*Administrador*, *Organizador*, *Cliente*) cuentan con métodos específicos según sus funciones.
- **Inversión de Dependencias:** las clases dependen de **abstracciones**, no de implementaciones concretas, lo que facilita las pruebas y el mantenimiento.

9. Alternativas Consideradas y Decisiones Tomadas

9.1. Alternativas de modelado descartadas

Durante el proceso de diseño se evaluaron distintas alternativas que finalmente fueron descartadas:

- Se optó por no utilizar una única clase “Usuario” con múltiples responsabilidades, ya que violaba el *Principio de Responsabilidad Única* y dificulta la escalabilidad.
- Se evitó centralizar la lógica en una sola clase controladora para reducir el acoplamiento y favorecer una estructura modular
- No se implementó una base de datos relacional en esta fase, priorizando la persistencia mediante archivos de texto por simplicidad y para cumplir las restricciones establecidas en el enunciado del proyecto.

9.2. Justificación de la solución final seleccionada

- La arquitectura elegida busca un equilibrio entre claridad, modularidad y facilidad de mantenimiento.
- La división del sistema en paquetes (Usuarios, Eventos, Transacciones y Financiero) permite organizar las responsabilidades, reducir la complejidad y facilitar la incorporación de nuevas funcionalidades.
- La aplicación de los principios SOLID asegura un diseño limpio, coherente y extensible.
- Además, la elección de la persistencia basada en archivos cumple con los requerimientos del proyecto, garantizando que la información se mantenga segura, consistente y recuperable sin depender de infraestructura externa.

10. Historias de Usuario

Las siguientes historias de usuario representan las **funcionalidades críticas** del sistema *BoletaMaster*, redactadas con el formato estándar de historias ágiles y acompañadas de criterios de aceptación verificables.

Historia de Usuario 1 – Comprar Tiquete

Como cliente registrado,
quiero adquirir tiquetes para los eventos disponibles,
para poder asistir a los conciertos o espectáculos de mi interés.

Detalles adicionales:
El cliente selecciona el evento, la localidad y la cantidad de tiquetes, pudiendo aplicar ofertas vigentes y confirmar la compra mediante saldo disponible o un medio de pago externo.

Criterios de aceptación:

- Se muestran correctamente los eventos y localidades disponibles.
- Se permite elegir la cantidad de tiquetes sin exceder el límite máximo.
- El sistema calcula el precio total con cargos e impuestos.
- Las ofertas activas se aplican automáticamente.

- La compra se realiza solo si el pago o saldo es suficiente.
- Se genera un ID de transacción único y se actualiza el conteo de tiquetes vendidos.

Historia de Usuario 2 – Transferir Tiquete

Como cliente propietario de un tiquete, quiero transferir mi tiquete a otro usuario,
para que otra persona pueda asistir en mi lugar.

Detalles adicionales:
El usuario ingresa el login del destinatario y confirma la operación con su contraseña. Los tiquetes Deluxe no son transferibles.

Criterios de aceptación:

- El sistema valida que el tiquete sea transferible.
- Se verifica la existencia del usuario receptor.
- Se solicita confirmación final al emisor.
- El tiquete cambia de propietario y su estado se actualiza a “transferido”.
- No se permite transferir tiquetes vencidos o previamente transferidos.

Historia de Usuario 3 – Crear Evento

Como organizador registrado,
quiero crear y configurar un nuevo evento,
para que los clientes puedan adquirir tiquetes.

Detalles adicionales:
El organizador define la fecha, hora, tipo, localidades, precios y asocia un venue (nuevo o existente).

Criterios de aceptación:

- Solo los organizadores autenticados pueden crear eventos.
- Cada evento se asocia a un venue válido y disponible.
- No puede existir otro evento en la misma fecha y lugar.
- Las localidades deben tener capacidad y precios definidos.
- El evento se guarda con estado “pendiente” hasta aprobación del administrador.

Historia de Usuario 4 – Generar Reporte Financiero

Como organizador,
Quiero generar un reporte de ventas y ganancias,
Para conocer el rendimiento financiero de mis eventos.

Detalles adicionales:
El reporte incluye información sobre ingresos brutos y netos, descuentos aplicados y porcentaje de ventas por localidad.

Criterios de aceptación:

- El sistema lista los eventos asociados al organizador.
- El cálculo de ingresos es correcto y coherente.
- Se reflejan las tarifas de servicio descontadas.
- El reporte puede exportarse y almacenarse de forma persistente.

11. Aclaraciones Finales

Los diagramas UML del proyecto *BoletaMaster* (clases, secuencia, casos de uso y requerimientos funcionales) se encuentran en la carpeta **“Diagramas E2”.**

12. Conclusiones

12.1. Evaluación del diseño propuesto

- El diseño refleja con precisión la estructura funcional y lógica del dominio del problema.
- Las relaciones entre cliente, organizador y administrador están claramente definidas.
- El modelo de clases es modular, escalable y comprensible, facilitando su mantenimiento y crecimiento.
- Los diagramas de secuencia y alto nivel complementan la comprensión estructural y dinámica del sistema.

12.2. Beneficios y limitaciones del enfoque adoptado

Beneficios:

- Promueve una arquitectura limpia y organizada basada en orientación a objetos.
- Mantiene coherencia entre requerimientos funcionales y modelo técnico.
- Facilita la implementación en Java gracias a su estructura modular.
- Su diseño favorece la extensibilidad y la incorporación de nuevas funcionalidades.

Limitaciones:

- No incluye la capa de interfaz gráfica ni experiencia de usuario, ya que esta entrega se centra en la lógica y estructura del sistema.