Departamento de Ingeniería de Sistemas y Computación
Diseño y Programación Orientada a Objetos
ISIS-1226

DOC PROYECTO 2

Daniel Manzanera, 202411443, d.manzanerat@uniandes.edu.co

Santiago Gomez, 202221885, s.gomezp2@uniandes.edu.co

Santiago Pinilla, 202315246, s.pinillap2@uniandes.edu.co

Índice

Diseño general del sistema

(Los diagramas en alta resolución se encuentran en el repositorio para su mejor visualización).

Visión general

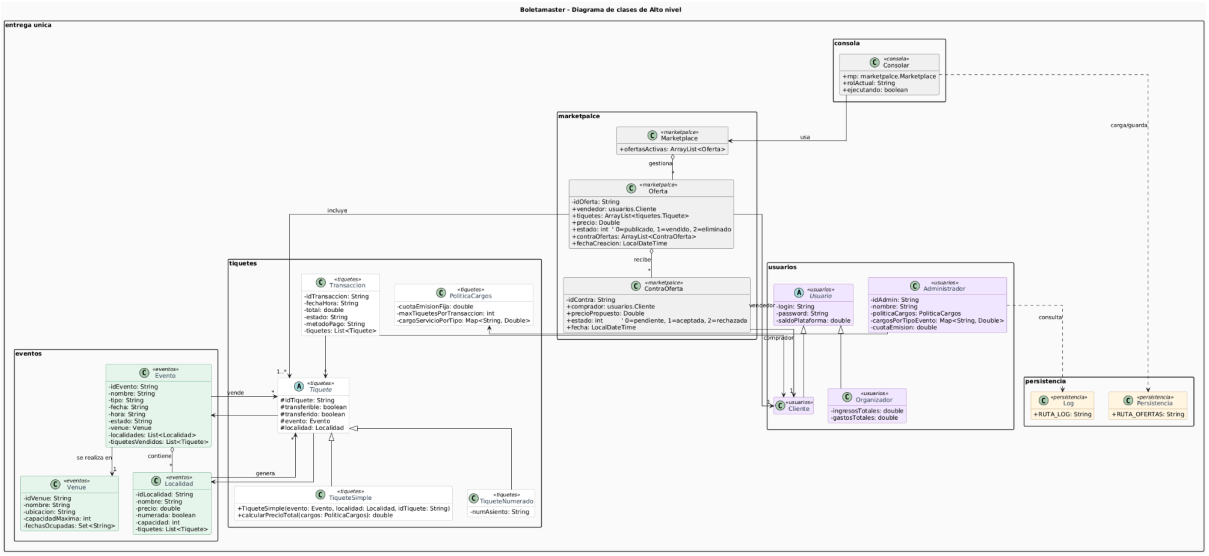
El sistema ofrece una interfaz de consola desde la cual distintos roles interactúan con el marketplace para publicar, negociar, comprar y administrar tiquetes de diversos eventos. Los roles principales son: Cliente, Organizador y Administrador.

Cambios realizados

Los UML (alto y bajo nivel) fueron actualizados según las correcciones del proyecto anterior. Se realizaron ajustes de estilo y color para mejorar la legibilidad y se agregaron tres paquetes:

marketplace, persistencia y consola. Con estos cambios se logra una interpretación más clara del sistema y se da cumplimiento a lo solicitado en la entrega.

Diagrama de de alto nivel



El sistema está organizado por paquetes que describen las distintas áreas que maneja el sistema: consola, marketplace, persistencia, usuarios, eventos y tiquetes. En el diagrama de alto nivel estas secciones agrupan las clases principales y sus relaciones sin detallar métodos:

- **consola**: interfaz de consola por rol usuario y maneja las entradas/salidas del usuario para que este pueda acceder, consultar y actualizar los datos de marketplace y persistencia a través de la lógica del sistema.

- **marketplace**: contiene la lógica de negocio relacionada apara publicar, listar, negociar y concretar operaciones sobre tiquetes. Incluye clases como Marketplace (fachada), Oferta y ContraOferta, y controla estados y validaciones del flujo (por ejemplo, que un tiquete sea transferible y no esté ya transferido).

- **persistencia**: provee la carga y guardado del estado del sistema.

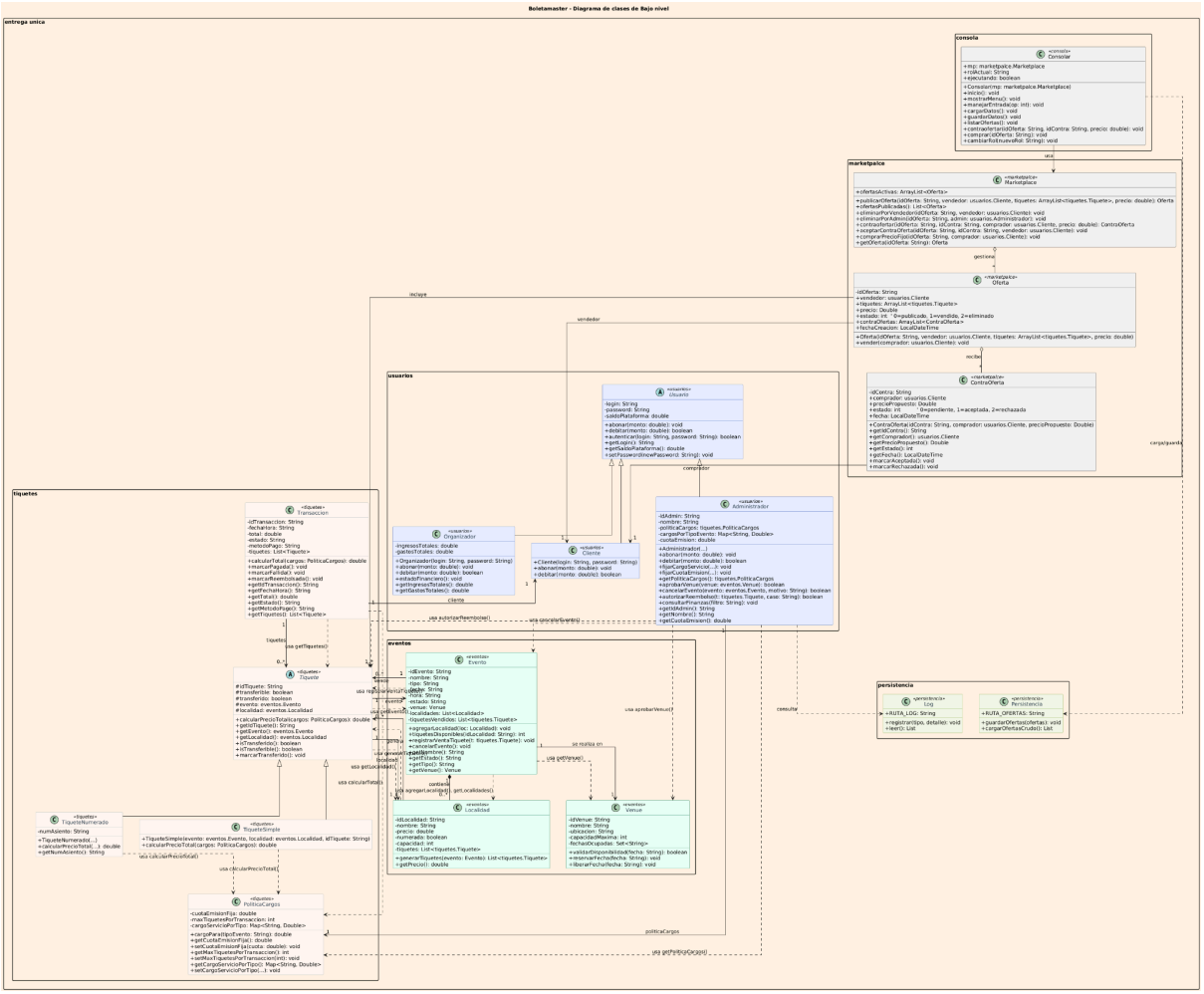
- **usuarios**: define los roles y sus permisos: Cliente (compra y puede publicar ofertas), Organizador (crea/administra eventos) y Administrador (supervisa y aplica acciones de control).

- **eventos**: contiene el evento, el venue, la localidad y su relación con la oferta de tiquetes.

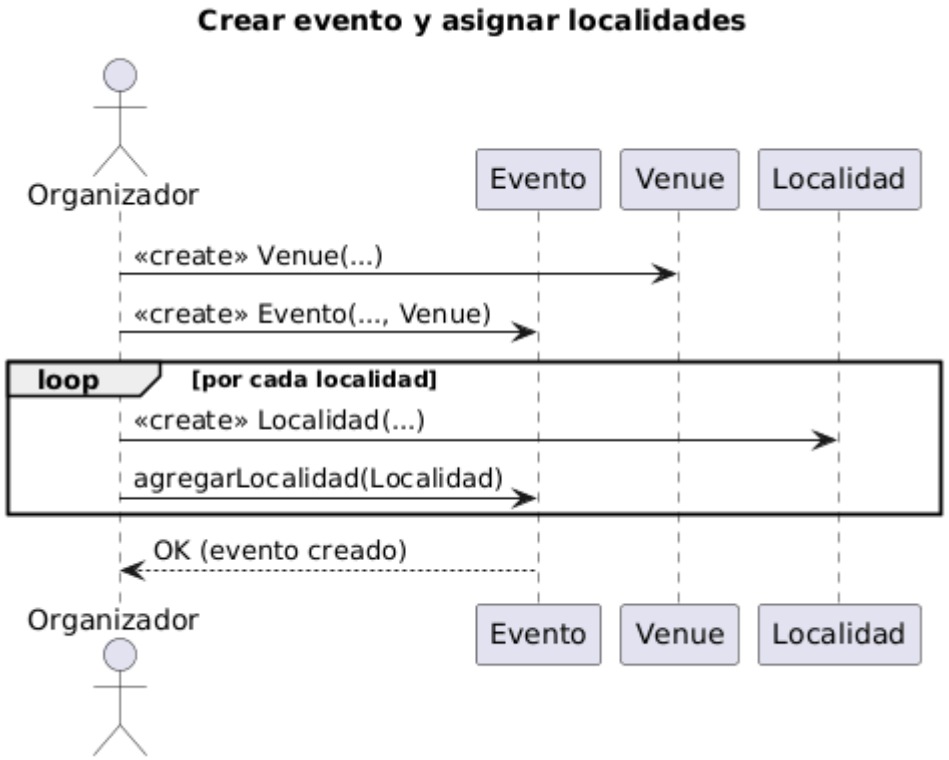
- **tiquetes**: contiene Tiquete (abstracta) , sus subtipos y transacción y política de cargos.

La consola se conecta con marketplace para ejecutar los casos de uso y con persistencia para reflejar los cambios; usuarios interactúan con marketplace según su rol; eventos se asocian a tiquetes, y estos pueden formar parte de ofertas en el marketplace.

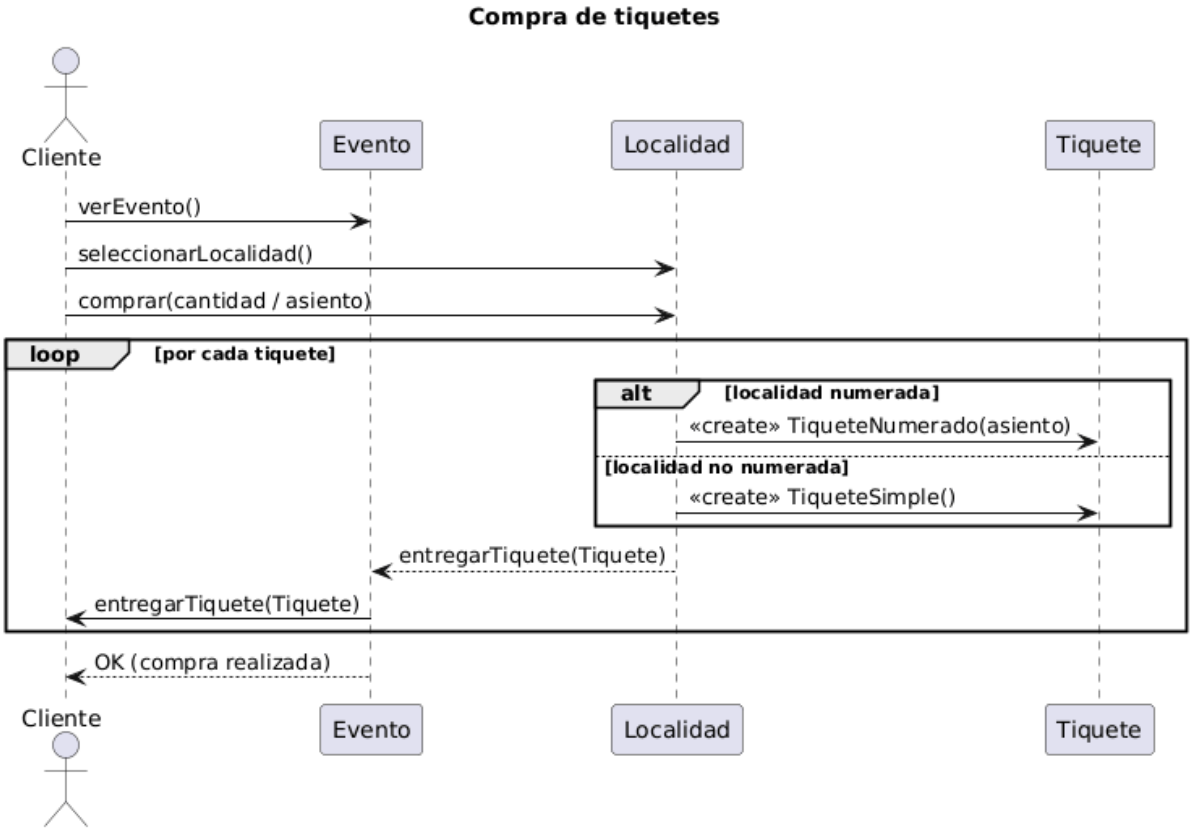
Diagrama de bajo nivel



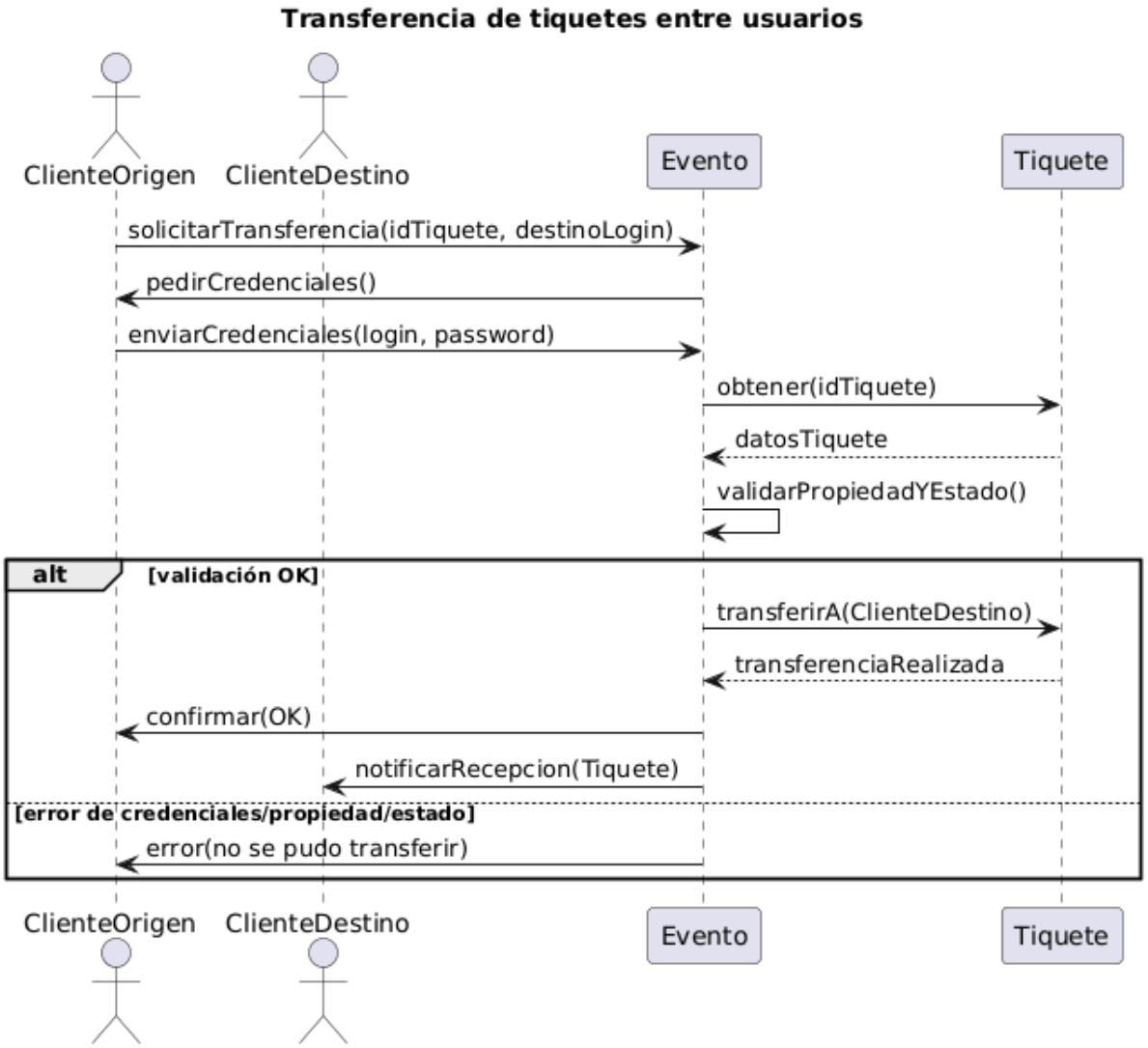
Diagramas de secuencia



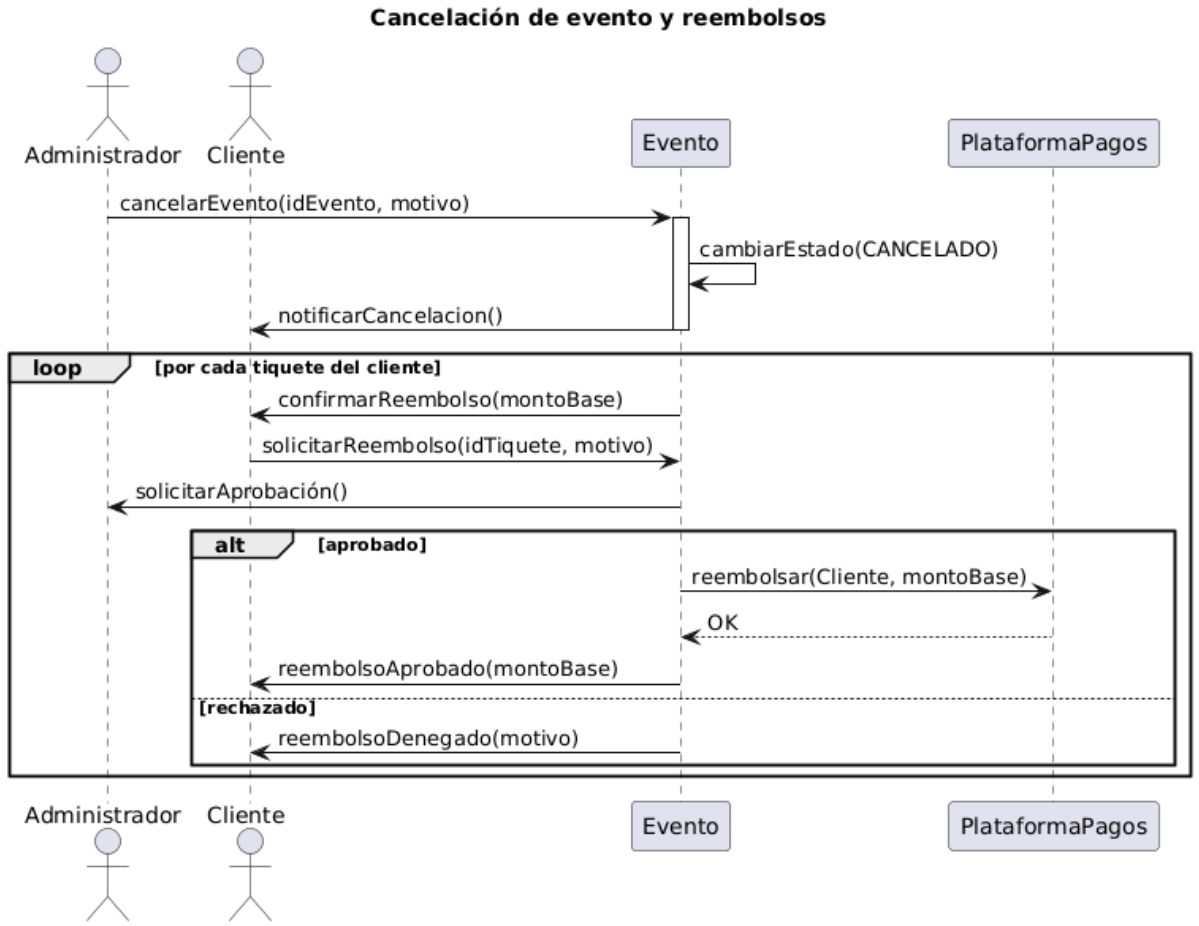
Iniciamos con el diagrama de secuencia para la creación de un evento, porque el evento es el objeto que dispara la interacción con el organizador y luego sirve de base para las demás operaciones. El flujo es: se crea el Venue, con ese venue se crea el Evento, y en un bucle por cada localidad se crea la Localidad y se llama a agregarLocalidad(Localidad) sobre el evento. Al final el sistema devuelve OK (evento creado).



A partir de un evento ya creado (como en la secuencia anterior), la interacción es del Cliente: primero verEvento(), luego seleccionarLocalidad() y, con base en lo que necesita, comprar(cantidad/asiento); dentro de un bucle [por cada tiquete] se evalúa alt según el tipo de localidad: [localidad numerada] → «create» TiqueteNumerado(asiento), [localidad no numerada] → «create» TiqueteSimple(); finalmente el sistema entregarTiquete(Tiquete) al cliente y cierra con OK (compra realizada).



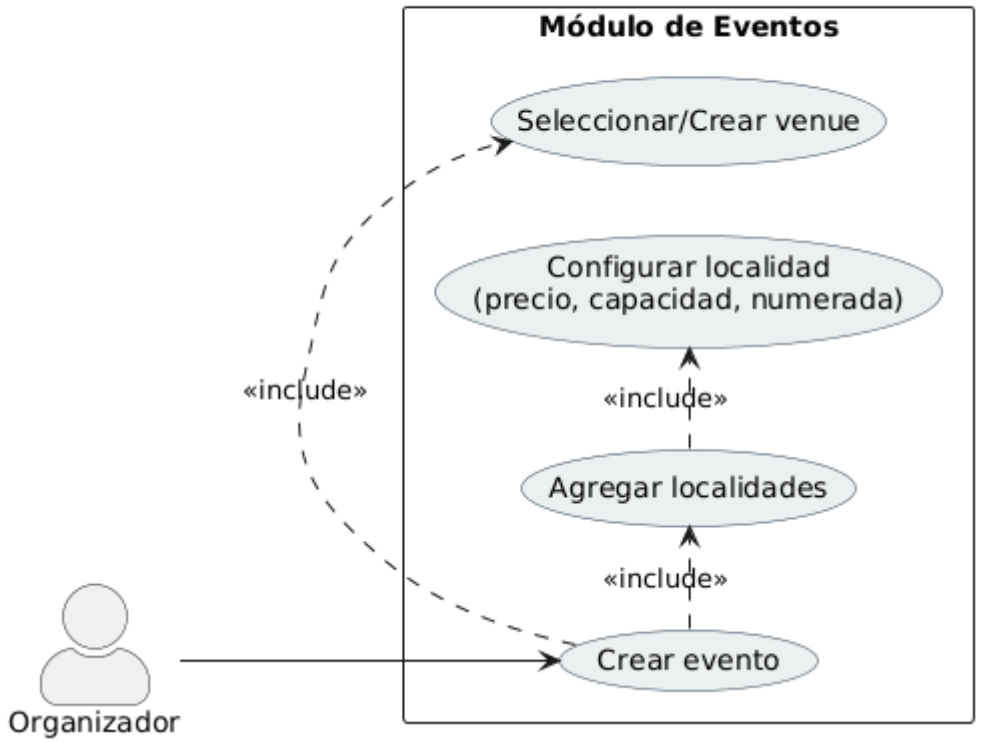
La secuencia muestra cómo usuarios tipo Cliente se transfieren tiquetes: el ClienteOrigen inicia con solicitarTransferencia(idTiquete, destinoLogin), el sistema pedirCredenciales() al ClienteDestino, este enviarCredenciales(login, password) y luego se obtener(idTiquete), cargar datosTiquete y validarPropiedadYEstado(); si la validación es correcta (alt [validación OK]), el sistema transferirA(ClienteDestino), marca transferenciaRealizada y notificarRecepcion(Tiquete) al destino, retornando confirmar(OK) al origen; en caso de error de credenciales/propiedad/estado, se responde error(no se pudo transferir).

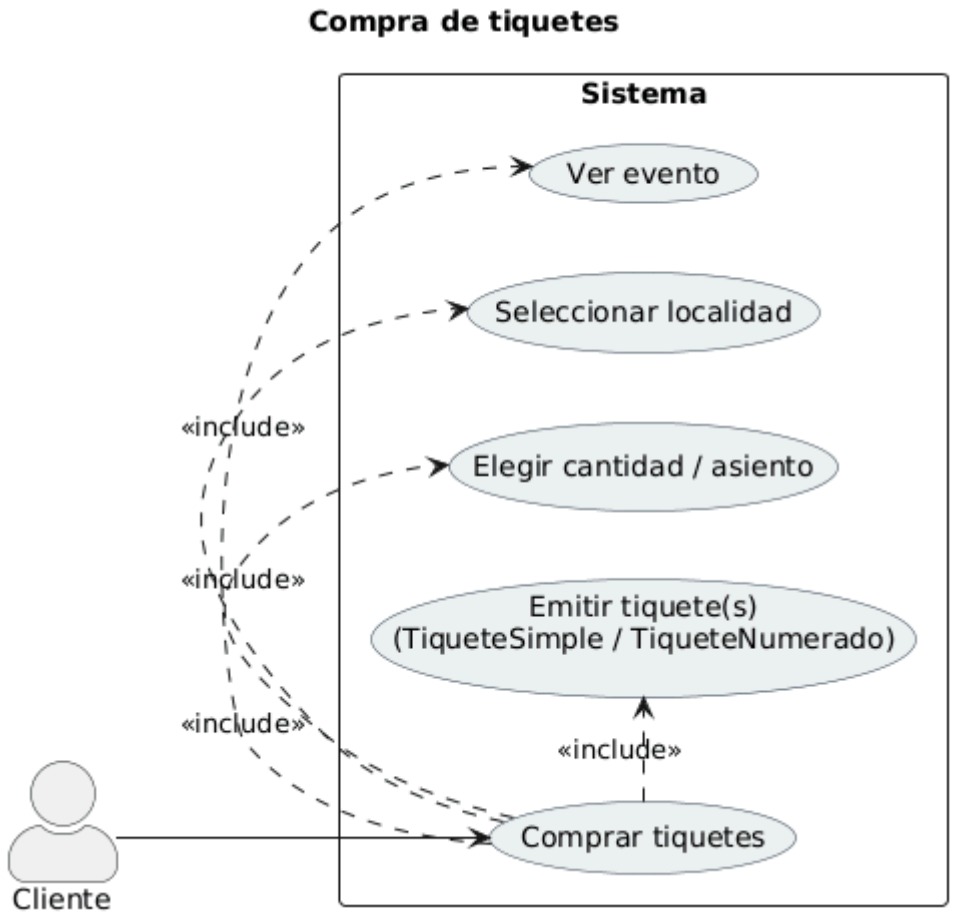


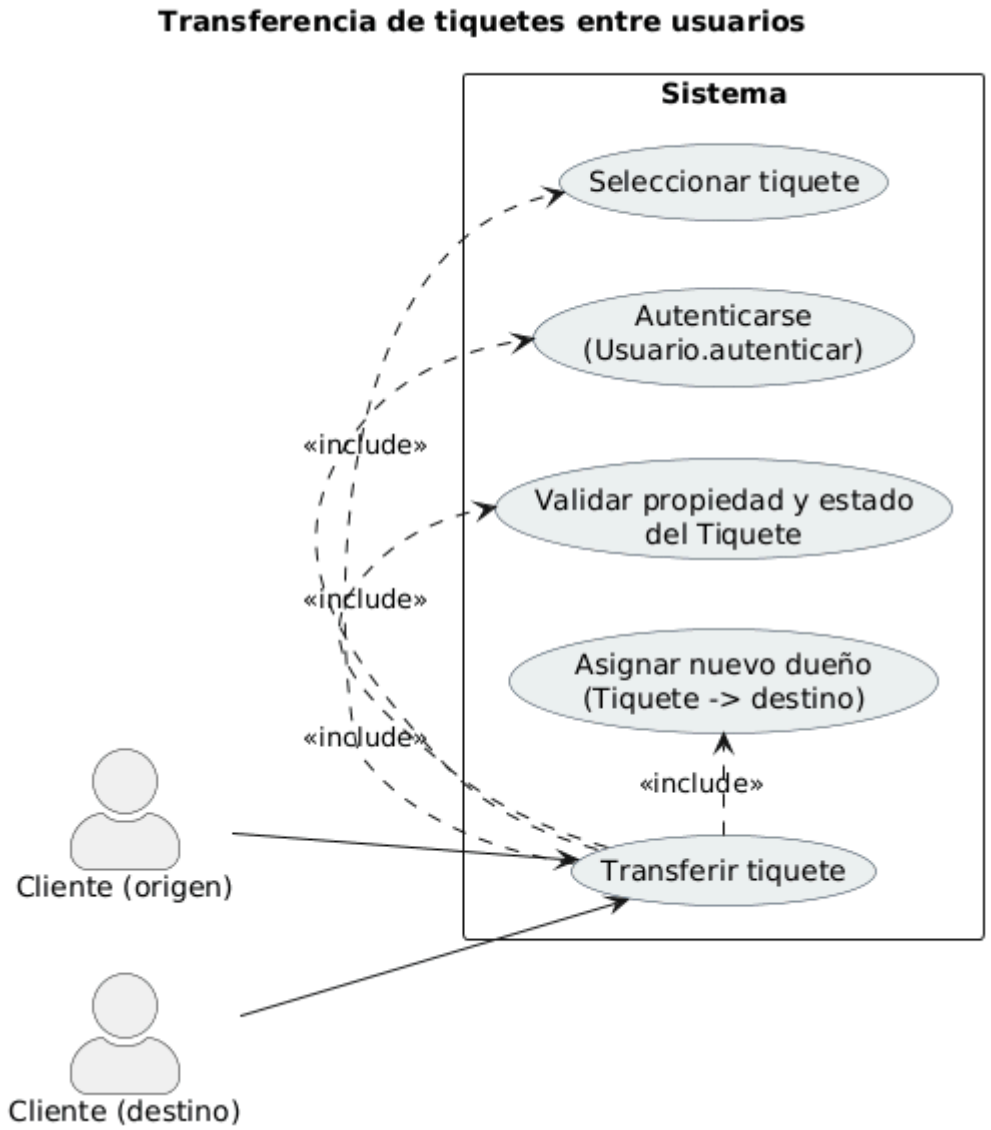
Esta secuencia corresponde a funciones del Administrador: el Administrador invoca cancelarEvento(idEvento, motivo), el Evento cambiarEstado(CANCELADO) y notificarCancelación(); luego, en loop [por cada tiquete del cliente], se gestiona el reembolso del montoBase vía PlataformaPagos → reembolsar(Cliente, montoBase) (OK) y el cliente confirmarReembolso(montoBase); si el cliente solicitarReembolso(idTiquete, motivo), el Administrador solicitarAprobación() y, según alt: [aprobado] la PlataformaPagos reembolsar(Cliente, montoBase) (OK) y se notifica reembolsoAprobado(montoBase); [rechazado] se informa reembolsoDenegado(motivo).

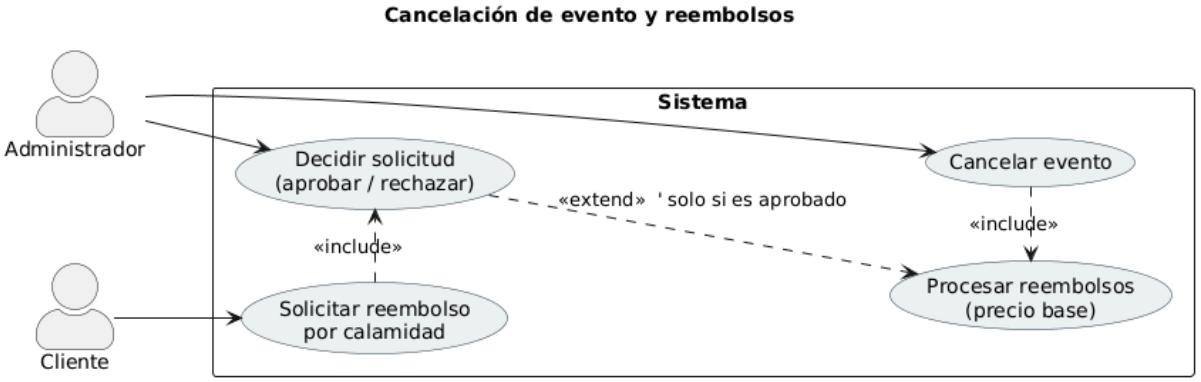
Diagramas de casos de Uso

Crear evento y asignar localidades




Compra de tiquetes
Sistema
Ver evento
Seleccionar localidad
Elegir cantidad / asiento
Emitir tiquete(s)
(TiqueteSimple / TiqueteNumerado)
Comprar tiquetes
«include»
«include»
«include»
«include»
Cliente





Justificación

Organización Lógica y Claridad

El sistema fue diseñado con una arquitectura modular basada en paquetes (eventos, tiquetes, usuarios), lo que promueve una separación clara de responsabilidades y facilita la mantenibilidad del código.

Cada paquete agrupa clases relacionadas con un mismo dominio funcional:

- usuarios: gestiona los distintos tipos de usuarios y sus operaciones.
- eventos: modela la estructura de los eventos, localidades y venues.
- tiquetes: maneja las reglas de negocio asociadas a las políticas de cobro y transacciones.

Esta organización lógica favorece la escalabilidad del sistema y permite integrar nuevas funcionalidades sin afectar las existentes.

Uso de Herencia y Polimorfismo

La clase abstracta Usuario agrupa atributos y comportamientos comunes.
A partir de ella se extienden tres subclases:

- Cliente: usuario que compra y transfiere tiquetes.
- Organizador: usuario que gestiona ingresos y gastos asociados a eventos.
- Administrador: usuario con permisos de gestión, configuración de políticas y control financiero.

El uso de herencia y polimorfismo facilita la extensión del sistema y evita duplicación de código, permitiendo que cada rol redefina o especialice los métodos según sus necesidades.

Encapsulamiento y Control de Errores

Se aplicó un estricto control de acceso a los atributos mediante encapsulamiento, garantizando la integridad de los datos.
Además, se incorporaron validaciones en los constructores y métodos para evitar inconsistencias, como descuentos fuera del rango permitido o montos negativos en transacciones.

Gestión de Reglas de Negocio

La clase PoliticaCargos centraliza las reglas de cobro por tipo de evento, permitiendo que el administrador defina cargos variables según la categoría del evento y una cuota de emisión fija.
Esto facilita la actualización de políticas sin alterar otras partes del sistema.

La clase Transaccion consolida las compras y reembolsos de tiquetes, manteniendo el estado y los totales calculados según las políticas vigentes.

Diseño Orientado a Casos de Uso

El diseño responde directamente a los principales casos de uso del sistema:

- Creación y cancelación de eventos.
- Generación de localidades y tiquetes.
- Compra, transferencia y reembolso de tiquetes.
- Aplicación de descuentos y ofertas.

Cada caso de uso se encuentra representado por una clase o método que refleja su comportamiento dentro del dominio del sistema.

Extensibilidad y Mantenibilidad

El diseño modular y el uso de interfaces hacen que el sistema sea fácilmente ampliable.
Por ejemplo:

- Pueden añadirse nuevos tipos de Usuario o Evento sin modificar la estructura existente.
- Nuevos servicios o políticas de precios pueden integrarse mediante la implementación de las interfaces del paquete servicios.

Esto asegura que el sistema pueda evolucionar con nuevas funcionalidades sin comprometer su estabilidad.