Proyecto 1

Documento de Diseño

Santiago Hernández Vélez – 202320909

David Varela Trujillo – 202321469

Bastien Quentin Clement Thirion - 202525085

Historias de usuario

Cliente:

Como cliente, quiero poder registrarme con un login y una contraseña para tener mi cuenta personal en la plataforma.

Criterio de aceptación: El login es único y la contraseña se almacena de forma segura.

Como cliente, quiero consultar los eventos disponibles para elegir a cuál asistir.

Criterio de aceptación: El sistema muestra la lista de eventos con nombre, fecha, lugar y tipo de boleta.

Como cliente, quiero comprar una o varias boletas para asistir a un evento.

Criterio de aceptación: Se descuenta el valor total de mi saldo virtual si es suficiente, y se me asignan los tiquetes.

Como cliente, quiero ver mis tiquetes comprados y sus detalles (tipo, precio, fecha, evento) para mantener un control de mis compras.

Como cliente, quiero transferir un tiquete a otro usuario, para que esa persona pueda asistir al evento en mi lugar.

Criterio de aceptación: No se permite la transferencia si el tiquete pertenece a un Pack Deluxe o si ya está vencido.

Como cliente, quiero solicitar un reembolso si el evento fue cancelado o si ya no puedo asistir. Criterio de aceptación: El administrador puede aceptar o rechazar la solicitud; el dinero se devuelve a mi saldo virtual.

Organizador:

Como organizador, quiero crear un nuevo evento asignándole un nombre, una fecha y un lugar (venue) para ofrecerlo al público.

Criterio de aceptación: Cada evento tiene un organizador y un lugar aprobados.

Como organizador, quiero definir los tipos y cantidades de tiquetes (simple, numerado, múltiple, pack deluxe) que estarán disponibles para mi evento.

Como organizador, quiero consultar mis ganancias y estadísticas de venta para evaluar el rendimiento de mis eventos.

Criterio de aceptación: El sistema muestra las ganancias sin incluir la tarifa de servicio.

Como organizador, quiero proponer nuevos lugares (venues) para realizar eventos futuros, que luego deberán ser aprobados por el administrador.

Criterio de aceptación: El administrador recibe la solicitud y puede aprobar o rechazar el lugar.

Como organizador, quiero cancelar un evento, con autorización del administrador, en caso de imprevistos o baja demanda.

Criterio de aceptación: El sistema actualiza el estado del evento y permite los reembolsos correspondientes.

Administrador:

Como administrador, quiero aprobar o rechazar los nuevos lugares propuestos por los organizadores para asegurar que cumplan con las condiciones necesarias.

Criterio de aceptación: El sistema muestra la lista de solicitudes pendientes y permite marcar cada lugar como “aprobado” o “rechazado”.

Como administrador, quiero fijar el cargo por servicio para cada tipo de evento, de manera que se pueda calcular la ganancia de la plataforma.

Criterio de aceptación: El cargo se guarda correctamente y se aplica en todas las compras nuevas de ese tipo de evento.

Como administrador, quiero definir el cargo fijo de emisión de tiquetes, el cual debe aplicarse a todos los tipos de eventos por igual.

El cargo se actualiza para todos los eventos y se refleja en el precio final mostrado al cliente.

Criterio de aceptación: El sistema actualiza el estado del evento a “cancelado” y los clientes reciben un reembolso completo.

Como administrador, quiero cancelar eventos cuando sea necesario para mantener el correcto funcionamiento del sistema.

Como administrador, quiero consultar las ganancias generales de la plataforma por fecha, evento u organizador, para llevar control financiero.

Criterio de aceptación: Los valores incluyen las tarifas de servicio.

Como administrador, quiero gestionar los usuarios (clientes y organizadores) para mantener actualizada la base de datos del sistema.

Criterio de aceptación: El sistema permite crear, modificar, suspender o eliminar usuarios y mantiene la unicidad del login.

Definición del dominio.

Una historia de usuario se considera terminada cuando todas las condiciones funcionales son respetadas, las reglas del dominio se aplican correctamente, las excepciones son manejadas y los objetos persistentes (clientes, organizadores, eventos, tiquetes) son guardados y leídos correctamente desde los archivos serializados.

Clases Tiquetes

Tiquete

Clase abstracta que genera el objeto Tiquete. Esta clase guarda la localidad a la que pertenece, el id del usuario al que pertenece, un atributo estático del precio de la impresión, de tal forma que emitir cualquier tiquete tiene el mismo costo, una fecha LocalDate, una hora LocalTime, un id final que depende de la localidad y el evento, el usuario al que pertenece la boleta, su precio base, un boolean que determina si el tiquete ya fue usado, el evento al que pertenece, un String que determina el tipo de tiquete, un HashMap de tipos de evento, que tiene asociado para cada tipo de evento un porcentaje asociado, el precio real y un atributo estático final tiquetesMax que es el límite de tiquetes individuales que se pueden comprar al tiempo.

Decisiones relevantes.

Se decide dejar fijo un valor máximo de tiquetes que se pueden comprar de forma simultánea, pues consideramos que es un atributo ligado a la forma de operar de la tiquetera y no de un evento o localidad específica.

Se menciona que existen cuotas de servicio específicas dependiendo del tipo de evento, por lo cual se agrega el HashMap estático tiposeventos sobre el cual se puede consultar este porcentaje siempre al momento de crear un tiquete.

TiqueteBasico

Clase que hereda de tiquete, cuenta con un atributo final estático “BASICO” que se asigna a su tipo cuando es creado.

TiqueteEnumerado

Clase que hereda de tiquete, cuenta con un atributo estático “ENUMERADO” que se asigna a su tipo cuando es creado. Cuenta con un atributo entero idSilla, que es el número de la silla que se compra.

TiqueteDeluxe

Clase que hereda de tiquete, cuenta con un ArrayList de Strings, donde almacena los beneficios asociados al tiquete, además, cuenta con un ArrayList de Tiquetes, donde se almacenan las cortesías a otros eventos asociadas al tiquete. Cuenta con dos constructores, uno donde recibe el idSilla en caso de que el tiquete Deluxe sea numerado, y un constructor sin él. Cuenta con un atributo estático precio.

Decisiones relevantes.

El tiquete Deluxe es una unidad, que a partir de su compra puede generar los beneficios y cortesías mencionadas. Se asume que, independientemente del evento, todos los tiquetes Deluxe van a costar lo mismo.

TiqueteMultiple

Clase abstracta que genera un objeto TiqueteMultiple, como atributos tiene el ID del tiquete múltiple, un bool transferible que indica si se puede transferir el tiquete múltiple, un string con el tipo de tiquete múltiple, un entero estático currentID que guarda cuál es el id del último tiquete múltiple creado y ayuda a asignar un id único al siguiente tiquete múltiple que se cree.

Decisiones relevantes.

Un tiquete multiple representa la compra de un paquete que incluye diferentes tiquetes. La gracia detrás de comprar tiquetes múltiples consiste en los precios asociados a su compra, los tiquetes que contiene dentro son entidades Tiquete descritas anteriormente. Todos los tiquetes que se generan a partir de un tiquete multiple son TiquetesBasicos.

TiqueteMultipleUnicoEvento

Esta clase hereda de TiqueteMmultiple, cuenta con un ArrayList con todos lo tiquetes generados por el paquete, un HashMap estático de precios, el cuál a partir de un id de localidad y la cantidad de tiquetes del paquete, tiene un precio asociado a este, y un string final estático “MULTIPLEUNICOEVENTO” que se asigna como el tipo de tiquete multiple.

Decisiones relevantes.

Esta clase es para comprar una cierta cantidad de tiquetes para un evento específico, todos los tiquetes comprados han de pertenecer a la misma localidad. Cuenta con un mapa estático de precios, funciona de la siguiente manera: Para un id de localidad, tiene un HashMap dentro que, para un numero de tiquetes tiene un precio asociado. Estas relaciones las asigna el administrador. Al ser estático, significa que, para todos los eventos, las localidades con el mismo identificador van a tener precios asociados al tamaño del tiquete múltiple iguales.

TiqueteMultipleVariosEventos

Clase que hereda de TiqueteMultiple, cuenta con un HashMap donde por cada evento tiene asociado un tiquete. Además tiene un string final estático "MULTIPLEVARIOSEVENTOS" que se asigna como el tipo de tiquete múltiple. Finalmente tiene un HashMap de precios, donde por cada cierta cantidad de tiquetes tiene asociado un precio.

Decisiones relevantes.

Se asume que, cuando se quiere comprar un tiquete con acceso a varios eventos, todos los tiquetes asociados son básicos, y todos los eventos a los que se quiere ir tienen un único Tiquete, como comprar un abono de un equipo de fútbol, que da acceso de 1 tiquete a diferentes partidos. Además, independientemente de los eventos, el valor de el tiquete múltiple depende exclusivamente de la cantidad de tiquetes que se compren, por eso se encuentra como estático en la clase.

Clases Usuario

Usuario

Un usuario cuenta con un login y contraseña únicos, un HashMap donde por cada evento hay una lista con los tiquetes que pertenecen a dicho evento. Un HashMap donde por cada identificador hay un tiquete múltiple asociado y finalmente, un HashMap estático donde por cada login hay un usuario asociado.

Decisiones relevantes.

Es necesario almacenar todos los usuarios que estén involucrados, pues cuando se realizan transferencias resulta muy importante poder encontrar el usuario al que se quiere transferir y actualizar su información con el nuevo tiquete. Además es importante trabajar por separado con los tiquetes múltiples e individuales, pues debemos estar en la capacidad de transferirlos todos si así se desea.

Cliente

Clase que hereda de Usuario, no tiene atributos o métodos diferentes a lo que hace un usuario común, instancia la clase que lo diferencia de organizador.

Organizador

Clase que hereda de usuario. Además de los atributos y métodos de un usuario común, tiene un HashMap con los eventos creados, dado el nombre del evento, almacena dicho evento.

Decisiones relevantes.

El organizador debe ser capaz de revisar sus estados financieros, para eso almacena los eventos que el ha organizado. Los eventos cuentan con sus localidades, y cada localidad con sus tiquetes, por lo cual puede consultar sus estados financieros globales, por evento y por localidad específica de un evento.

Otras Clases

Administrador

El administrador cuenta con un login y una contraseña, pero recordemos que no es un usuario del sistema. Además cuenta con una lista de solicitudes que debe atender.

Evento

Un evento cuenta con los siguientes atributos, un nombre, una fecha y hora, un Venue, un organizador, un HashMap donde, por cada identificador existe una localidad, y atributos finales estáticos que permiten ver los diferentes tipos de evento que puede tener: musical deportivo, religioso y cultural. Además cuenta con un String que es el estado del evento, un string que es el tipo del evento, y un HashMap estático de comisión por eventos, donde por cada tipo de evento, tiene asociado un porcentaje de emisión de tiquete específico.

Decisiones relevantes.

Se menciona que los sobrecargos por servicios son porcentajes que dependen del tipo de evento, por eso se guarda como estático para poder acceder siempre al sobrecargo dependiendo de los posibles tipos de evento: musical, cultural, deportivo, religioso. Además, al momento de crear una localidad, que puede ser o no numerada, cuando se añade al evento se le asigna un identificador para ese evento concreto, y se guarda en el HashMap mencionado.

Localidad

Cuenta con un nombre propio, el precio de los tiquetes que tienen acceso a la localidad, la capacidad, un boolean que determina si contiene tiquetes numerados, el venue del que hace parte, y el evento al evento. Tiene dos HashMaps para monitorear los tiquetes que se compran para dicho evento en dicha localidad, uno para tiquetes normales, y otro para tiquetes múltiples, ambos entran en el mapa por su localidad.

Decisiones relevantes.

Se decide separar los tiquetes múltiples de los normales, pues se menciona que existen casos donde los tiquetes múltiples representan entradas a espacios de las localidades destinados para grupos, por lo que es conveniente separarlos. Se decide almacenar todos los tiquetes en las localidades pues, dado que todas las localidades están ligadas a su evento, a través del evento se pueden consultar toda la información de los tiquetes por cada localidad, que para los estados financieros será muy útil.

Venue

Contiene la información de un venue, un HashMap con eventos que tiene reservados por fecha, una capacidad, un nombre y una ubicación.

Decisiones relevantes.

Se decide crear el HashMap de cada Venue de evento por fecha, para así al momento de crear un evento, no se vaya a crear en una misma fecha que otro, cosa que no se puede hacer.

Clase abstracta Solicitud

Sirve como entidad de organización para las solicitudes que se presentan entre entidades.

Decisiones relevantes.

Funciona como transporte de referencias a las entidades relacionadas en la solicitud, cuando la solicitud llegue al administrador, este ya tiene la referencia al objeto de la persona que hizo la solicitud (o sobre el evento que se hizo), no hay necesidad de busquedas adicionales.

Volver esta clase abstracta nos permite tener un comportamiento aislado para el caso de aceptación o rechazado de una solicitud, por ejemplo, para cancelar un evento el organizador envía la solicitud con el evento, y el administrador al recibirla puede invocar el método de aceptar la solicitud que automáticamente cancelara el evento y se encargara de todo lo que esto conlleva. Note que este proceso es mucho menos dispendioso debido a que la clase contiene dentro de ella referencias a todas las clases implicadas.

Funciona como un object value interno que se transfiere entre entidades.

Funciona como unidad de almacenamiento, el administrador puede consultar sus solicitudes que se guardan en un arraylist en su clase (se guardan polimorficamente)

Clase BoletasMaster

Esta es la clase centralizada del negocio, se encarga de controlar las interacciones entre entidades y más importante es la unica clase que se conecta con la persistencia y la consola (object value).

Decisiones relevantes.

Con la ayuda de los atributos esAdministrador, esOrganizador, esCliente es posible saber que usuario se ha logeado en cualquier momento, por ende, sabemos que métodos desplegar para que el usuario interactúe con ellos.

La clase almacena los objetos para ser procesados en persistencia garantizando la unicidad de la informacion. Esto es importante porque se puede almacenar la información de varias sesiones haciendo que, por ejemplo, las peticiones del organizador las pueda responder el administrador independientemente de que se hayan realizado en ejecuciones diferentes del programa

La unicidad de la informacion no solo se garantiza para el almacenamiento, sino para la escritura. En todo momento la clase logra que las interacciones tengan efecto sobre todas las clases pertinentes, por ejemplo, hacer una compra supone generar el nuevo Tiquete y añadirlo al Usuario y al Evento.

Se han almacenado los eventos en un arreglo secundario que los sorte por fecha, almacenar la información así mejora exponencialmente la complejidad temporal para acceder a eventos de una fecha.

Finalmente, se indican métodos de visualización para los casos donde es primordial para la entidad consultar información estructurada especifica como los estados financieros.

Clase ArchivoSerializable

La clase ArchivoSerializable pertenece al paquete Persistencia, se encarga de gestionar la persistencia de datos en el sistema mediante la serialización de objectos Java.

Es el componente que permite guardar y recuperar la información de los diferentes del dominio (usuarios, administrador, venues, eventos y tiquetes) gracias a archivos .ser de la carpeta Datos.

Decisiones relevantes.

Si el archivo de datos no existe, se crea automáticamente

Usamos el formato serializable que es de Java porqué es más fácil guardar y leer objetos complejos mientras que necesita más código con archivos planos

Las funciones de ArchivoSerialible se usan en la clase BoletasMaster para leer y escribir cada tipo de datos directamente desde el sistema

Responsabilidades

Usuario

Un usuario tiene la capacidad de comprar tiquetes, ya sean básicos, enumerados, deluxe múltiples de un solo evento y múltiples de varios eventos. Además, tiene la capacidad de transferir dichos tiquetes y tiquetes singulares de los tiquetes múltiples. Puede usar los tiquetes, que al hacerlo quedan marcados como usados y puede solicitar un reembolso por calamidad, cuando es aceptado, se ejecuta el procedimiento para realizar el reembolso.

Cliente

No cuenta con responsabilidades o capacidades diferentes al usuario.

Organizador

Además de las capacidades que hereda de usuario, tiene la responsabilidad de crear los eventos, solicitar la creación de un venue, asignar la cantidad y tipo de tiquetes disponibles. Además tiene la capacidad de revisar sus estados financieros, globales, es decir de todos los eventos, por evento, y por localidad de un evento.

Administrador

Tiene la responsabilidad de fijar el sobrecargo por servicio a cada tipo de evento que se ofrecen, también fija el cobro por emisión de los tiquetes, el cual es fijo e igual para todos los tiquetes. La creación y aprobación de nuevos venues también depende del administrador. Además, puede cancelar un evento, en dicho caso se hace un reembolso por el precio base de los tiquetes. Finalmente, el administrador tiene la capacidad de ver los estados financieros de la tiquetera, evaluando ganancias totales, por fecha, evento u organizador.

Reglas y restricciones

Estados financieros

Hay múltiples restricciones ligadas a los estados financieros. En primer lugar. Todos los tiquetes que se generan reciben su precio de la siguiente forma. A partir de un precio base, que depende de la localidad a la cual pertenezca el tiquete, se calcula el precio real sobre este precio base, el costo de impresión del tiquete y un porcentaje de cobro por servicio que depende del tipo de evento. Cuando se crea el tiquete, se verifica que el usuario que lo este comprando no sea el organizador del evento, en dicho caso, se asigna 0 a los precios base y real del tiquete.

Por otro lado, cuando se trata de un tiquete múltiple entendemos que la gracia de estos tiquetes es que resulten más económicos para el cliente, por lo cuál estos no dependen de las localidades si no de la asignación previa de un organizador o administrador. Para los tiquetes múltiples de único evento, existe un precio fijo para la totalidad de comprar cierta cantidad de tiquetes en cierta localidad. En estos casos, este precio se divide en el número de tiquetes que posee el tiquete múltiple y calcula su precio real sumando su precio de emisión y su cobro por servicio, se espera que esto genere normalmente un precio base más económico a los clientes. Por otro lado, para los tiquetes múltiples con varios eventos sucede algo similar, y es que para comprar cierta cantidad de tiquetes un precio asociado a esta totalidad. Sobre esta totalidad se calcula un nuevo precio base y sobre este se calcula el nuevo precio real.

Ahora, para obtener tanto los estados financieros del organizador como los del administrador, solo es necesario recorrer eventos, las localidades de sus eventos, y el valor de estos tiquetes, ya sea el base, el real o la diferencia entre los dos (corresponde a la ganancia por servicio de la tiquetera). Pues, cuando se creen tiquetes para el organizador del evento, estos quedan registrados como 0 y no se tienen en cuenta, cuando sean tiquetes múltiples, su valor ya no depende de la localidad si no del precio generado por comprar en un paquete, entonces toma un precio real de ganancia, y por último, los tiquetes normales generan su valor de forma convencional, por lo cual al momento de calcular estados financieros, la generación misma de los tiquetes se asegura de que no existan problemas.

Autenticación y operaciones

Todo usuario que interactúe con la plataforma debe autenticarse mediante un login y contraseña válidos. Si las credenciales son incorrectas, el sistema debe impedir el acceso y notificar el error. De igual forma, cualquier operación que no esté permitida para el rol del usuario será rechazada, garantizando que cada actor del sistema solo pueda ejecutar las acciones que le corresponden.

Compra de tiquetes

La compra de tiquetes está sujeta a un límite máximo por transacción. Si un cliente intenta adquirir más tiquetes de los permitidos, la operación se cancela. Además, el sistema valida que el usuario cuente con saldo suficiente para completar la compra; en caso contrario, la transacción no se ejecuta. Estas restricciones aseguran un control sobre la disponibilidad y la capacidad de pago.

Transferencia y uso de tiquetes

Los tiquetes pueden transferirse entre usuarios, pero existen condiciones que lo impiden. No se permite transferir tiquetes que pertenezcan a paquetes Deluxe, ni aquellos que estén vencidos o ya utilizados. Si el tiquete solicitado no existe o no pertenece al usuario que intenta transferirlo, la operación se rechaza. Estas reglas garantizan la integridad del sistema y evitan fraudes.

Eventos y venues

Cada evento debe estar asociado a un único venue, y un venue no puede albergar más de un evento en la misma fecha. Si el venue solicitado no está disponible, el sistema impide la creación del evento. Además, las localidades tienen una capacidad máxima; si se intenta asignar más tiquetes de los permitidos, la operación se cancela. Estas reglas garantizan la correcta organización de los eventos y la seguridad de los asistentes.

Reembolsos

Los reembolsos solo se permiten en casos específicos: cancelación del evento por parte del administrador o solicitud por calamidad, sujeta a aprobación. Si el reembolso no está autorizado, el sistema lo rechaza. Esta restricción asegura que las devoluciones se realicen bajo condiciones controladas.