Proyecto 1 DPOO: Boletmaster

- Contexto del problema:
    - Se nos ha pedido como desarrolladores, la construcción de una aplicación que emule el funcionamiento de una empresa de venta de tiquetes para eventos varios (Musicales, culturales, religiosos, deportivos, etc.). La plataforma debe servir como la base de operaciones de la empresa, encargada de todas las transacciones y actividades administrativas que sean necesarias para el correcto funcionamiento de la empresa.

        - Usuarios:
            - Clientes:
                - De todas las edades y categorías. Al tener una variedad tan grande en cuanto a los eventos proporcionados y administrados por la aplicación, no existe distinción entre posibles clientes. Es decir: no hay una demografía específica. Son los principales compradores, y quienes acceden a los servicios de venta de tiquetes.
            - Promotores de eventos:
                - Los creadores y organizadores de los eventos dentro de la aplicación. Pueden ser organizadores de uno o más eventos, que a su vez pueden ser simultáneos. Determinan fechas, horas y venues para cada evento, además de ser los que dictan el número de tiquetes disponibles al público, ofertas posibles, y variaciones de precio por localidad del venue. Debe poder revisar sus ganancias por evento y localidad de este.
                - El organizador también puede comportarse como un cliente, comprar los tiquetes normalmente o si lo hace para su propio evento, no se le genera cobro.
            - Administradores:
                - El administrador pertenece a la empresa tiquetera misma. Debe poder fijar los porcentajes de ganancia de la tiquetera, el cobro fijo de emisión y aprobar los venues del promotor.
                - Puede cancelar eventos, y como el promotor revisar el estado fianciero de las ventas, basado en los recargos diseñados por él, mostrados por fecha, organizador o evento.
                - No puede comprar tiquetes.
        - Dispositivos:

        - Al ser una herramienta web, la plataforma existe en celulares (como aplicacón móvil o como página web). Todo basado en internet.
    - Comunicaciones:
        - Todas las comunicaciones deben realizarse por internet, por medio de la misma aplicación.
    - Otras personas involucradas:
        - Toda acción de venta y lucro dee estar vigilada por el ente gubernamental competente: Superintendencia de Industria y comercio.
        - Puede haber un interés particular de patrocinadores por evento, o permanentes de la misma aplicación.
    - Intereses involucrados:
        - Los clientes desean poder comprar tiquetes al evento de su preferencia, en todas sus posibles presentaciones, iteraciones o modificaciones.
        - Los organizadores desean promover y facilitar el acceso a sus eventos. Así como acceder al desempeño financiero de estos eventos, una vez hayan sido propiamente planeados y diseñados en la misma aplicación. Los organizadores también desean poder utilizar la aplicación como clientes, pero tendiendo trato preferencial para sus propios eventos.
        - Los administradores desean poder determinar la aprobación de los eventos presentados dentro de la aplicación, llevar un control de las ganancias generadas por comisión en cada evento, así como poder cancelar los eventos a discreción.
- Problema:
    - Se nos pide desarrollar una herramienta web que permita la planeación, administración, venta de tiquetes y posteriormente el reportaje de ganancias de varios eventos culturales.
    - Para esto debemos crear tres tipos de usuarios, con diferentes capacidades y potestades. Así como tres interfaces distintas, que se utilizarán para surtir las necesidades de cada uno.
- Requerimientos funcionales:
    - Vamos a dividir los requerimientos funcionales, basados en las necesidades de cada tipo de usuario:
        - Cliente:
            - La aplicación debe poder encargarse de transacciones monetarias (tercerizado).
            - La aplicación debe poder llevar un saldo interno para realizar transacciones virtuales sin necesidad de terceros.

- La aplicación debe poder emitir tiquetes de diversos tipos, en las cantidades que el cliente desee, mientras sean acordes a los máximos delimitados por evento.

- Organizador:
  - La aplicación debe poder facilitar la planeación del evento, permitiéndole al administrador definir horarios, fechas, venues, capacidad máxima y localidades disponibles.
  - La aplicación debe permitirle al organizador ver la información financiera pertinente en todos sus eventos, por evento puntual o por localidad de este.
  - La aplicación debe también permitir a los organizadores convertirse en clientes, darles la capacidad de comprar boletos, y ofrecerles un descuento total en el caso de que quieran asistir a uno de sus propios eventos.

- Administrador:
  - La aplicación debe poder darle al administrador la capacidad de definir las tarifas y recargos pertinentes que se quedará la tiquetera.
  - La aplicación debe permitirle revisar las ganancias de la tiquetera, por fecha, evento u organizador puntual.
  - La aplicación debe poder permitirle a un administrador la cancelación de un evento. Encargarse de realizar los reembolsos o acciones correctivas pertinentes, y el cese de la actividad monetaria y la emisión de tiquetes para ese evento en particular.
  - El administrador NO PUEDE convertirse en cliente.

- Requerimientos no funcionales:
  - La aplicación debe tener una base de datos que incluya:
    - Todos los eventos próximos (por fecha y organizador)
    - Todos los eventos pasados (por fecha, organizador y ganancia)
      - Por evento (se le añade a la información base del evento):
        - Capacidad Máxima (en tiquetes)
        - Tiquetes vendidos (En tiquetes y ganancias totales)
          - Ganancias para el organizador
          - Ganancias para la tiquetera
    - Todos los organizadores asociados a la tiquetera
      - Por organizador:
        - Eventos próximos
        - Eventos Pasados
        - Tiquetes vigentes

            - Tiquetes no vigentes (usados o vencidos)
        - Todos los clientes (incluye organizadores)
            - Tiquetes vigentes
            - Tiquetes no vigentes
        - Todos los venues posibles
    - La aplicación debe implementar una herramienta de construcción de evento, que, al utilizarse, debe guardar el evento en la lista de eventos próximos con la siguiente información básica:
        - Por evento:
            - Nombre
            - Fecha
            - Hora de inicio
            - Hora de cierre
            - Venue
            - Localidades
                - Aumento de precio por localidad
                    - Vamos a basar los cambios de costo de los tiquetes basado en la localidad por porcentajes. Cada localidad puede subir (o no) el precio estándar del tiquete basado en un porcentaje del pecio base determinado por el organizador.
            - Precio de tiquete base
            - Tipo (Religioso, deportivo, musical, cultural, etc.)
            - Organizador
            - Capacidad máxima
    - La aplicación debe tener una tienda online que permita la venta de tiquetes, provea las ofertas que sean pertinentes y emita los tiquetes que el cliente le pida, basado en la información del evento, los recargos del administrador, la capacidad máxima y los cambios de costo por localidad.
    - La aplicación debe tener un portal de revisión financiera por evento para los organizadores. Dónde por evento y localidad de mismo puedan revisar el estado final de las ventas tras haber sido tramitadas por la tienda, sin incluir los recargos de la tiquetera (solo se ven los ingresos del organizador).
    - La aplicación debe tener un portal administrativo que permita al administrador cancelar eventos (con las consecuencias pertinentes), revisar las ganancias generadas a la tiquetera una vez cerrada la venta de tiquetes por evento. Revisar las ganancias generadas en una fecha, o las ganancias asociadas a un organizador.

- Diagrama de clases (UML):

Versión HD: https://lucid.app/lucidchart/89f574a4-c43a-4db9-85df-19f7dfee55fd/edit?viewport_loc=-397%2C161%2C4020%2C1592%2CHWEp-vi-RSFO&invitationId=inv_e14bfeb8-fd5f-4b86-85fe-ada9d6c3d463

- Restricciones:

1. Restricciones Funcionales
    a. Cualquier tipo de evento presentado, debe estar asociado a un organizador y a un venue valido
    b. El venue no puede presentar más de un evento en la misma fecha y hora
    c. El número de tiquetes disponibles para la venta no puede superar la capacidad máxima del venue
    d. En las localidades numeradas del venue, se debe tener un numero identificador único
    e. Se debe respetar el numero máximo de ventas de tiquetes para cada usuario por evento
    f. Los tiquetes de entrada múltiple deben de venderse en conjunto
    g. No se pueden transferir los tiquetes Deluxe
    h. Solo se pueden transferir tiquetes por medio de usuarios con login y password
    i. Las tarifas de tiquetes y servicios pueden variar según el tipo de evento
    j. Los reembolsos de una boleta se acreditarán a un saldo virtual dentro de la plataforma del usuario

- Programas de Prueba:

Programa 1: Carga inicial de datos

- **Objetivo:** Validar que el sistema lea correctamente archivos con información de usuarios, eventos y venues.
- **Entradas:** Nombre del archivo (ejemplo: eventos.txt, usuarios.txt).
- **Salidas esperadas:** Confirmación de carga exitosa y listado de entidades cargadas (usuarios, eventos, localidades).
- **Requerimiento validado:** Persistencia y base de datos de eventos, clientes, organizadores y venues.

Programa 2: Creación de evento (organizador)

- **Objetivo:** Verificar que un organizador pueda registrar un evento con su respectivo venue y tiquetes.
- **Entradas:** Nombre del evento, fecha, hora, venue, capacidad, localidades y precios.
- **Salidas esperadas:** Confirmación de creación del evento y listado de eventos activos con sus atributos.
- **Requerimiento validado:** Planeación de eventos (horarios, venues, capacidad máxima, localidades).

Programa 3: Compra de tiquetes (cliente)

- **Objetivo:** Validar que un cliente pueda comprar tiquetes respetando los límites y precios.

- **Entradas:** ID del evento y cantidad de tiquetes.
- **Salidas esperadas:** Confirmación de compra, generación de IDs únicos de tiquetes, cálculo del costo total (precio base + recargos). Mensaje de error si se exceden los límites.
- **Requerimiento validado:** Compra de tiquetes, aplicación de recargos del administrador, límites máximos por evento.

Programa 4: Transferencia de tiquete

- **Objetivo:** Probar que un cliente pueda transferir un tiquete a otro usuario bajo las reglas establecidas.
- **Entradas:** ID del tiquete, login del receptor y contraseña del dueño actual.
- **Salidas esperadas:** Confirmación de transferencia o mensaje de restricción (si el tiquete es Deluxe, está vencido o ya fue transferido).
- **Requerimiento validado:** Transferencia de tiquetes con seguridad y restricciones.

Programa 5: Cancelación de evento (administrador)

- **Objetivo:** Validar que el administrador pueda cancelar un evento y realizar los reembolsos correspondientes.
- **Entradas:** ID del evento.
- **Salidas esperadas:** Confirmación de cancelación, listado de clientes afectados y reembolsos depositados en el saldo virtual. El cálculo debe respetar las reglas de devolución.
- **Requerimiento validado:** Cancelación de eventos y gestión de reembolsos en el saldo interno de los usuarios.

Programa 6: Reportes financieros

- **Objetivo:** Verificar que los organizadores y el administrador puedan consultar sus ganancias.
- **Entradas:** ID del evento o rango de fechas.
- **Salidas esperadas:** Reporte con tiquetes vendidos, ingresos generados, ganancias del organizador y recargos de la tiquetera.
- **Requerimiento validado:** Revisión financiera por evento, localidad, organizador y tiquetera.

Programa 7: Funcionamiento del marketplace

- **Objetivo:** Verificar el correcto funcionamiento de la función general de marketplace.
- **Entradas:** Cliente con sesión iniciada.
- **Salidas esperadas:** Capacidad para generar ofertas de venta de tiquetes, explorar las ofertas disponibles y hacer contraofertas, revisar, aceptar o ignorar ofertas hechas para tiquetes propios.
- **Requerimiento validado:** Función de marketplace generalizada y local de cada cliente.

Programa 8: Revisión final de aplicación

- **Objetivo:** Verificar que las funciones establecidas en consola funcionen correctamente.
- **Entradas:** N.A

- **Salidas esperadas:** Todos los requerimientos funcionales en cada tipo de usuario dado.
- **Requerimiento validado:** Funcionamiento correcto general de la aplicación.

- Historias de Usuario:

  Nuestra aplicación maneja tres tipos diferentes de usuarios con intereses diferentes para con la aplicación de Boletmaster. Para enunciar nuestro ejercicio, presentamos las siguientes historias de usuario, teniendo en cuenta las funciones y necesidades de todos:

  - Administrador:
    - **Como un** administrador.
    - **Quiero** poder moderar, manipular y controlar el flujo de la aplicación, acceder al dinero generado por cada evento, organizador y fecha, además de aceptar y cancelar los eventos que se van a presentar en la plataforma de Boletmaster.
    - **Para así** poder encargarme del correcto funcionamiento de la aplicación y llevar la contabilidad de nuestra tiquetera, así como llevar un histórico de compras y transacciones que se hayan llevado a cabo en la plataforma.

    - **Detalles adicionales:**
      - El administrador puede generarse como usuario en la aplicación, pues es un empleado real y puede ser reemplazado, así como puede haber varios administradores trabajando para Boletmaster. Sin embargo, el administrador necesita una clave maestra específica para generar un usuario, confidencial para los empleados de Boletmaster.
      - El administrador no puede ser cliente, ni puede generar eventos. Es decir: el administrador no posee ni emite tiquetes de ninguna forma.
      - El administrador define las tasas de aumento que se le añaden a los tiquetes, basados en porcentajes.
    - **Criterios de aceptación**
      - Poder revisar las solicitudes de creación de eventos y venues de los organizadores.
      - Poder aceptar o rechazar las solicitudes de los organizadores.
      - Poder cancelar eventos a discreción, con todas las consecuencias y garantías de reembolso para los clientes.
      - Poder revisar las ganancias de la tiquetera por organizador, evento y fecha.

- Cliente:
    - **Como un** cliente.
    - **Quiero** poder utilizar la aplicación como fuente de compra, organización y reventa de mis tiquetes para varios eventos.
    - **Para así** obtener acceso a los eventos del tipo que sea, fácilmente desde mi aplicación, o en su defecto transferir o revenderlos a otros usuarios.

    - **Detalles adicionales:**
        - El cliente no puede crear ni solicitar nada a los organizadores o los administradores.
        - El cliente tiene la potestad de revender y transferir sus tiquetes por medio del sistema de marketplace de la aplicación. Pero los tiquetes deluxe no pueden ser transferidos o vendidos.
        - El cliente tiene acceso a la función markeplace de la aplicación, que funciona mediante a un sistema de subastas dónde es su discreción vender o no sus tiquetes.
        - El cliente debe recibir una compensación en el caso de que un evento se cancele.
    - **Criterios de aceptación**
        - Poder revisar los tiquetes activos y usados que haya comprado.
        - Acceder al marketplace.
        - Transferir sus tiquetes a discreción.
        - Utilizar sus tiquetes para entrar a una función.
        - Revisar el catálogo de eventos disponible, y comprar tiquetes, sea con saldo o con pago tercerizado.
        - Poder consultar su saldo en la aplicación.

TESTS

UsuarioTest:

- void constructor_inicializaCamposCorrectamente(): Verifica que el constructor inicialice correctamente
- void toString_camposVaciosoNulos_seIncluyen(): Verifica el comportamiento del toString con valores vacíos o nulos. Si se incluyen literalmente

LocalidadTest:

- void formatearStringIncluyendoVenue(): Verifica que el toString se realice correctamente, incluyendo venue
- void imprimirVenue(): Verifica que se imprime N/A cuando no hay venue

VenueTest:

- void eventoPorFechaYRechazarDuplicados(): Revisa que se permita un evento por fecha y rechace duplicados
- void LocalidadConCapacidadNoPositiva(): Revisa que no se permita crear localidades con capacidad menor o igual a cero
- void noPermitirLocalidadesSuperenCapacidadDelVenue(): Revisa que no se permita que la suma de localidades supere la capacidad maxima del venue
- void permitirSumaExactaAlaCapacidadMaxima(): Verifica que se permita la suma exacta a la capacidad maxima del venue

VenueNumeradoTest:

- void disponiblesConCapacidadMaxima(): Verifica que se cree un arreglo y su longitud coincida con la capacidad maxima
- void toStringMuestraDatosBasicos(): Verifica que se muestren los datos basicos en el toString

TiquetesTest:

- void calculoCostoConPorcentajeDeLocalidad(): Verifica que se calcule el costo con el porcentaje de la localidad
- void deberiaCopiarFechaYHoraDelEvento(): Verifica que se copie la fecha y hora desde el evento pasado
- void identificadorIniciaYFinalizaCorrectamente(): Verifica que el identificador inicie con login y termine en un numero
- void toStringDelegadoAImprimir(): Verifica la delegacion correcta de toString en imprimir

ClienteTest:

- void constructorInicializaCorrectamente(): Verifica que el constructor inicializa tipo de listas y saldo por defecto
- void imprimirFormateaCredencialesCorrectamente(): Verifica que imprimir formatea correctamente saldo y lista de ids

TiqueteBasicoTest:

- void costoEsBaseMasPorcentajeDeLocalidad(): Verifica que el calculo del costo sea el costo base mas el porcentaje de la localidad
- void imprimirCorrectamenteCuandoFaltenReferencias(): Comprueba que cuando hay campos nulos se imprime N/A

TiqueteDeluxeTest:

- void aplicaTasaDelEventoAlCostoInicial(): Verifca que el costo del tiquete deluxe sume el recargo de la tasa del evento

- void inclcusionDeBeneficiosEnImpresion(): Verifica que en imprimir agrega los beneficios y mantiene deluxe como tipo
- void imprimeNAEnBeneficiosNulos(): Verifica que se imprime N/A cuando no hay beneficios definidos

Revisiones:

- Corregir costo en constructor, condicional si el ev no es nulo el costo es igual al costo mas la tasa del evento entre 100
- Verificar por que no esta imprimiendo N/A al tener beneficios nulos

TiqueteGrupalTest:

- void calculaCorrectamenteCostoGrupal(): verifica que el costo sea (base + porcentaje * base) * individuos
- void impresionIncluyeIndividuosYCostoTotal(): verifica que imprimir refleje la cantidad de personas y el costo total

Revisiones:

- Para la segunda revisar:

TiqueteNumeradoTest:

- void asignacionPrimerPuestoLibreYMarcacionEnVenue(): Verifica que se toma el primer indice libre en disponibles y se guarda el tiquete ahi
- void noAsignacionSiNoHayCupos(): Verifica que si no hay espacios, no asigna y deja el numero por defecto en –1
- void inclusionDeNumeroDeAsientoEnImpresion(): Verifica que imprimir inluye el numero de asiento como ultimo campo

AdministradorTest:

- void constructorVacioDefineTipoAdministrador(): Verifica que el constructor vacio defina como tipo administrador
- void constructorConArgsFijaLoginPassYTipo(): verifica que cuando el constructor este con argumentos se fije el login, la contrasena y el tipo
- void toStringSigueElFormato(): Verifica que el toString siga el formato Login,Pass,Tipo

OrganizadorTest:

- void constructorVacioInicializaCorrectamente(): Verifica que el constructor vacio inicialice el tipo, las listas y el saldo
- void constructorConArgsInicializaCorrectamente(): Verifica que el contructor con argumentos inicializa el tipo y las listas
- void imprimirIncluyeTiquetesVigentesNoVigentesYEventos(): Verifica que imprimir incluye correctamente el formato log,pass,tipo,saldo,IDsVigentes|IDsNoVigentes|eventosProx|eventosPas

EventoTest:

- void getCapacidadUsadaEnLocalidadesSumaCapacidades(): Verifica que la distribucion interna se contabilice correctamente, mediante el recorrido de localidades y la suma de sus capacidades
- void totalEventoNoSuperaCapacidadVenue(): Verifica que se cumpla la restriccion de que la cantidad del evento no supere la capacidad del venue
- void tiquetesProcesadosYRestantesSeCalculanCorrectamente(): Verifica que los tiquetes procesados y restantes se calculan con base en la capacidad total
- void toStringIncluyeDatosClave(): Verifica que el toString incluya nombre, tipo, fechas, precio, venue y organizador

AplicacionIntegrationTest:

- void testCreacionArchivosJSON(): Verifica que se creen los archivos de apoyo de la página web correctamente y demuestra su existencia.
- void testRegistroYGuardadoCliente(): Verifica que el cliente se puede registrar y generar dentro de la aplicación, y que correctamente se añade a las listas pertinentes y sobrescribe los archivos.
- void testRegistroYGuardadoAdministrador(): Verifica que el administradorse puede registrar y generar dentro de la aplicación, y que correctamente se añade a las listas pertinentes y sobrescribe los archivos.
- void testRegistroYGuardadoOrganizador(): Verifica que el organizador se puede registrar y generar dentro de la aplicación, y que correctamente se añade a las listas pertinentes y sobrescribe los archivos.
- void testCargaPersistencia(): Verifica que, al guardar datos, la carga automática que se realiza lee correctamente y carga la información pertinente.
- void testCrearEvento(): Verifica que la función dle administrador para crear un evento funciona correctamente, y queda registrada como es debido tanto en la lista actual como en el archivo que corresponde.