SHOWPASS

Memoria trabajo fin de grado

Fecha: 1/12/2025

Autores: Marcos Beas Sánchez y Dylan Molina Amariles

2º Desarrollo Aplicaciones Multiplataforma 2024-2025 I.E.S Villablanca

1. Introducción

1.1. Descripción de la aplicación a desarrollar

El proyecto **Showpass** consiste en el desarrollo de una plataforma digital integral para la gestión, compra y venta de entradas para eventos. Se estructura como una solución **multiplataforma**, ofreciendo una **aplicación web** y una **aplicación móvil** (App) que consumen los servicios de un **Backend** propio desarrollado por el equipo (Dylan y Marcos).
El objetivo principal es eliminar las fricciones en el proceso tradicional de venta de entradas, proporcionando una herramienta eficiente, segura e intuitiva tanto para los usuarios finales (Clientes) como para los organizadores de eventos (Vendedores) y los gestores internos (Administradores).

1.2. Finalidad y objetivos

- **Finalidad**: Proveer un mercado digital directo y transparente que conecte organizadores de eventos y consumidores. Esto se logra ofreciendo una plataforma tecnológica robusta que simplifica la publicación, venta, compra y gestión post-venta de entradas.
- **Objetivos de los desarrolladores**:

  1. Arquitectura en capas: Implementar una arquitectura limpia y modular de tres capas (Presentación, Lógica de Negocio/API, Acceso a Datos).

  2. API de servicios: Desarrollar una API REST para gestionar la lógica de negocio y las operaciones CRUD (Crear, Leer, Actualizar, Borrar) de Usuarios, Eventos y Tickets.

  3. Multiplataforma: Asegurar la funcionalidad completa y la coherencia visual/operacional entre la aplicación web y la aplicación móvil.

  4. Control de acceso: Implementar un sistema de autenticación y autorización basado en roles (Cliente, Vendedor, Administrador).

- **Objetivos de Usuario**

  1. Experiencia del Cliente: Ofrecer una navegación fluida, un proceso de compra rápido (carrito y simulación de pago) y múltiples opciones de gestión de tickets ("Mis Tickets": descarga local, envío por correo, eliminación).

  2. Experiencia del Vendedor: Proveer un panel de control intuitivo que permita crear, editar y visualizar sus propios eventos de forma autónoma.

  3. Experiencia del Administrador: Suministrar herramientas para el control total del sistema (moderación de eventos, gestión de usuarios).

1.3. Requisitos y restricciones

El cumplimiento de estos requisitos será la base para las pruebas de aceptación del proyecto

Requisitos Funcionales (RF)

Rol	Descripción del Requisito
Cliente	Añadir eventos al carrito de compra.
Cliente	Descargar los tickets localmente (Web/Móvil).
Cliente	Enviar los tickets al correo electrónico de registro.
Cliente	Modificar datos del perfil (nombre, foto, simulación de tarjeta bancaria).
Cliente	Simular la compra de tickets.
Cliente	Poder registrarse y crear un perfil de usuario.
Cliente	Navegar, buscar y visualizar eventos.
Vendedor	Registrarse y seleccionar el rol de Vendedor.
Vendedor	Crear nuevos eventos.

Vendedor	Editar los eventos creados por sí mismo.
Admin	Borrar cualquier evento del sistema.
Admin	Bloquear, desbloquear o eliminar cuentas de usuario mediante correo electrónico.

Requisitos No Funcionales (RNF)

Descripción del Requisito
Seguridad: Los datos sensibles (contraseñas, simulación de datos bancarios) deben ser almacenados de forma segura (hashing) en el Backend
Rendimiento: El tiempo de carga de las listas de eventos y el proceso de compra no debe exceder los 3 segundos.
Usabilidad: Las interfaces de usuario (Web y Móvil) deben ser intuitivas, accesibles y consistentes entre plataformas.
Escalabilidad: La arquitectura del Backend debe permitir un crecimiento futuro en el número de usuarios y eventos sin degradación severa del servicio.
Multiplataforma: El sistema debe funcionar de manera óptima en las principales plataformas de escritorio (Web) y dispositivos móviles (iOS/Android).

1.1.Quick Introduction in English

1.1. Application Description

The Showpass project involves developing a comprehensive digital platform for managing, buying, and selling event tickets. It is designed as a cross-platform solution, offering both a web application and a mobile application that consume services from a dedicated backend developed by the team (Dylan and Marcos).

The primary goal is to remove the frictions present in traditional ticketing processes, providing an efficient, secure, and intuitive tool for end users (Customers), event organizers (Sellers), and internal managers (Administrators).

1.2. Purpose and Objectives

Purpose: To provide a direct and transparent digital marketplace connecting event organizers with consumers. This is achieved through a robust technological platform that simplifies ticket publishing, sales, purchasing, and post-sale management.

Developers Objectives:

- **Layered Architecture:** Implement a clean, modular three-layer architecture (Presentation, Business Logic/API, Data Access).
- **Service API:** Develop a REST API to manage business logic and CRUD operations (Create, Read, Update, Delete) for Users, Events, and Tickets.
- **Cross-Platform:** Ensure full functionality and visual/operational consistency between the web and mobile applications.
- **Access Control:** Implement a role-based authentication and authorization system (Customer, Seller, Administrator).

User Objectives:

- **Customer Experience:** Provide smooth navigation, a fast purchase process (cart and payment simulation), and multiple ticket management options ("My Tickets": local download, email delivery, deletion).
- **Seller Experience:** Offer an intuitive control panel allowing sellers to create, edit, and view their own events independently.
- **Administrator Experience:** Provide tools for complete system control (event moderation, user management).

1.3. Functional Requirements and Restrictions

Customers can add events to the cart, register, simulate ticket purchases, manage profiles, and download or email tickets. Sellers can register, create, and edit their own events. Administrators can block, unblock, or delete user accounts and remove any event from the system.

1.4. Non-Functional Requirements (NFRs)

- **Security:** Sensitive data (passwords, simulated payment information) must be securely stored (hashed), and all communication with the backend must be encrypted.
- **Performance:** Event listings and purchase processes should load within 3 seconds.
- **Usability:** User interfaces (Web and Mobile) must be intuitive, accessible, and consistent across platforms.
- **Scalability:** Backend architecture should support future growth in users and events without significant service degradation.
- **Cross-Platform Compatibility:** The system must function optimally on major desktop platforms (Web) and mobile devices (iOS/Android).

2. Análisis del entorno

2.1. Descripción de las oportunidades de negocio

- **Modelo de negocio enfocado al Vendedor:** Existe una oportunidad para atraer a organizadores de eventos que buscan plataformas con comisiones más bajas o con herramientas de gestión más potentes y directas que las ofrecidas por los grandes monopolios del sector.
- **Experiencia de usuario (UX) unificada:** La mayoría de las soluciones existentes ofrecen experiencias fragmentadas. **Showpass**, al ser desarrollado bajo un mismo Backend y un diseño coherente en Web y Móvil, puede diferenciarse ofreciendo una UX superior.
- **Nicho de eventos específicos:** Posibilidad de captar eventos locales o de menor escala que son a menudo ignorados por las grandes plataformas.

2.2. Obligaciones fiscales, laborales y de prevención de riesgos

- **Protección de Datos :** Es fundamental el cumplimiento de la normativa de protección de datos personales al manejar información de perfiles, correos electrónicos y, especialmente, la simulación de datos bancarios. La política de privacidad debe ser visible y clara.
- **Comercio Electrónico:** La aplicación debe cumplir con la Ley de Servicios de la Sociedad de la Información y de Comercio Electrónico (LSSI), incluyendo términos y condiciones de venta claros y política de devoluciones/cancelaciones en caso de desarrollo profesional.

2.3. Posibles ayudas o subvenciones

El proyecto podría calificar para:

- **Ayudas a la Digitalización y Emprendimiento:** Subvenciones de organismos autonómicos o nacionales (como el SEPE, cámaras de comercio, o programas como el Kit Digital en España) destinados a la creación de nuevas empresas de base tecnológica.
- **Financiación Inicial (Startups):** Posibilidad de buscar rondas de financiación inicial (*seed funding*) o préstamos a emprendedores una vez superada la fase de prototipo (TFG).

3. Análisis del sistema actual

3.1. Funcionamiento y de los problemas que acarrea a la empresa o cliente la aplicación actual

Dado que **Showpass** es un proyecto nuevo, el "sistema actual" se refiere a las soluciones que existen en el mercado y los problemas que resuelven o crean.

- **Sistemas Intermediarios:** Las plataformas tradicionales (ej. Ticketmaster, Eventbrite) actúan como intermediarios con altas comisiones, lo que reduce el margen de beneficio de los Vendedores y encarece el precio final al Cliente.
- **Problemas de los Vendedores:** Falta de control directo sobre sus eventos una vez publicados. Dificultad para editar detalles de última hora y acceso limitado a los datos de sus propios clientes.
- **Problemas de los Clientes:** Necesidad de usar diferentes aplicaciones para diferentes tipos de eventos. Proceso de post-compra poco flexible (la gestión de tickets, como descargar, borrar o enviar por correo, es a menudo limitada).

3.2. Necesidades que presenta

El proyecto Showpass surge de las siguientes necesidades no cubiertas o mejorables en el mercado:

1. **Centralización de la Gestión de Tickets:** El Cliente necesita un único lugar (*Mis Tickets*) desde donde pueda gestionar sus entradas de forma digital completa (descarga, reenvío, eliminación), independientemente de si usa la web o la app móvil.
2. **Control Total del Vendedor:** El Vendedor requiere autonomía para **crear y editar sus propios eventos** sin depender de aprobaciones largas, lo que agiliza la publicación.
3. **Sistema de Roles con Privilegios:** Se necesita un sistema robusto que diferencie claramente las funcionalidades por rol, permitiendo al Administrador las tareas de moderación sensibles (borrado de eventos y gestión de cuentas) para mantener la calidad y seguridad de la plataforma.
4. **Experiencia de Usuario (UX) Multiplataforma Optimizada:** La necesidad de una app móvil que garantice un rendimiento excelente para la navegación y la descarga de tickets, manteniendo las acciones que se realizan en esta app también aplicadas en la web.

4. Solución propuesta

4.1. Elección y justificación de las tecnologías seleccionadas

Capa	Tecnología Propuesta	Justificación de la Elección
Backend (API y Lógica de Negocio)	Java con Framework Spring Boot	Uso de java con el framework de Spring Boot. Spring Boot permite crear una API REST robusta, ideal para una aplicación con gestión de roles y alto tráfico potencial.
Base de Datos	H2	Se elige una base de datos relacional escrita en Java, muy ligera y veloz. Guarda datos en archivos o en memoria, arranca rápido y necesita muy poca configuración.
Frontend Web	React y Vite	React facilita que tu interfaz crezca sin enredarse: organiza tu app en componentes reutilizables, actualiza la pantalla con eficiencia casi coreográfica y cuenta con un ecosistema enorme que acelera el desarrollo.
Frontend Móvil	Kotlin con Jetpack compose	Forma moderna y declarativa de crear interfaces en Android, reduce la cantidad de código al eliminar el uso de XML, gestiona el estado de manera clara y eficiente, mejora el rendimiento al actualizar solo lo necesario, se integra de forma natural con Kotlin todo respaldado por el soporte oficial de Google

Además la app web cuenta con una orquestación con Docker compose que es perfecto, porque permite definir y manejar múltiples contenedores de forma sencilla usando un solo archivo de configuración (Docker-compose.yml), facilita

la replicación de entornos completos (desarrollo, pruebas, producción), asegura consistencia entre distintos sistemas, automatiza la orquestación de dependencias entre servicios y hace más fácil escalar, actualizar o reconstruir servicios sin afectar el resto del entorno.

4.2. Recursos humanos y materiales necesarios

- **Recursos Humanos:** El proyecto está desarrollado por un equipo de dos personas: **Dylan y Marcos** ambos desarrollamos el Frontend y Backend por igual.
- **Recursos Materiales (Hardware):** Ordenadores personales con capacidad para ejecutar Docker, y realizar un comando para levantar todos los servicios en un solo comando, y Android Studio para cambiar la dirección de IP dentro de las constantes de la app para que el Frontend de la app móvil pueda conectarse a los servicios del backend desplegados por docker.
- **Recursos Materiales (Software/Infraestructura):**
    - **IDEs:** Visual Studio Code (Frontend web y Machine-learning), Eclipse (Backend), Android Studio (Frontend Web),
    - **Control de versiones:** Git y plataforma (GitHub/GitLab) para trabajo colaborativo.

4.3. Estudio económico. Valoración de presupuestos

Concepto	Coste Estimado (Simulación)	Justificación
Recursos Humanos (Personal)	**3 meses** (trabajo TFG)	Coste valorado en el tiempo dedicado por Dylan y Marcos. (En un escenario real, sería el mayor tiempo dinero).
Software y Licencias	**0 €**	Uso de tecnologías open source , que minimizan los costes de licencia.
Infraestructura (Cloud)	**0 €** (trabajo TFG) **50 – 100 €/año** (Escenario real)	Coste mínimo asociado a un servicio de hosting básico o un VPS para desplegar el Backend de forma accesible para la demostración.
Dominio y Certificado SSL	**0 €** (trabajo TFG) **15 – 30 €/año** (Escenario real)	Coste asociado a un dominio “.com” o similar y un certificado de seguridad inicial (muchos hostings ya lo incluyen).
Total Estimado	**0 €** (TFG) **2.000 €** (en un escenario hipotético de producción)	Cifra estimada para una puesta en marcha con costes reales mínimos, excluyendo el coste salarial del equipo de desarrollo

5. Planificación temporal del desarrollo de proyecto

Se propone una planificación dividida en 5 fases, estructurada en un horizonte de tiempo de **17 semanas** (aproximadamente 3-4 meses) de dedicación, adecuada para la duración del TFG.

Fase	Duración Estimada	Responsables	Tareas Clave
Fase 1: Análisis y Diseño	2 Semanas	Dylan & Marcos	1. Diseño de la Base de Datos (Diagrama ER). 2. Definición completa de la API. 3. Diseño UX/UI 4. Definición de la seguridad por roles.
Fase 2: Desarrollo del Backend (API)	4 Semanas y media	Dylan & Marcos	1. Configuración del entorno (BD,Spring Boot). 2. Desarrollo de la Autenticación y Autorización por Roles. 3. Implementación de la lógica de Eventos (CRUD). 4. Implementación de la lógica de Cliente, Vendedor y Admin.
Fase 3: Desarrollo Frontend Web	4 Semanas y media	Dylan & Marcos	1. Desarrollo de las vistas principales del Cliente (Navegación, Perfil). 2. Implementación del Carrito y el proceso de Compra. 3. Desarrollo del Panel del Vendedor (Creación/Edición de Eventos). 4.Desarrollo panel Admin
Fase 4: Desarrollo Frontend Móvil	4 Semanas	Dylan & Marcos	1. Configuración del proyecto. 2. Adaptación de las vistas del Cliente a la App Movil. 3. Desarrollo de la funcionalidad de Mis Tickets (Descarga/Envío).
Fase 5: Integración, Pruebas y Documentación	3 Semanas	Dylan & Marcos	1. Pruebas de integración Backend-Frontend. (Docker) 2. Pruebas unitarias y funcionales (QA de los 3 roles). 3. Corrección de errores. 4. Redacción final de la Memoria y Manuales

6. Estudio de la viabilidad del proyecto

Este proyecto no ha sido financiado por ningúna fuente externa y nos limitamos a dar dos puntos de vista para cada caso.

6.1. Viabilidad Técnica

Alta Viabilidad. Las tecnologías seleccionadas son maduras, cuentan con abundante documentación y una gran comunidad de apoyo. El equipo (Dylan y Marcos) tiene las habilidades técnicas necesarias adquiridas durante el Ciclo Formativo para abordar la arquitectura propuesta.

6.2. Viabilidad Herramientas y aprendizaje

Alta Viabilidad. El sistema satisface todas las necesidades identificadas: proporciona una solución a los Clientes para la gestión de sus tickets y ofrece las herramientas de control necesarias a los Vendedores y Administradores. La simplicidad de uso de las interfaces garantiza una curva de aprendizaje mínima para todos los tipos de usuario.

6.3. Viabilidad Económica

TFG Viabilidad Alta. Los costes de desarrollo se limitan principalmente al esfuerzo del equipo (TFG). Al apoyarse en *software* de código abierto, los costes de licencias son nulos.

En caso Real Viabilidad Media. La plataforma tiene un claro potencial de generación de ingresos a futuro (a través de comisiones de venta).
La infraestructura necesaria es escalable y se puede empezar con soluciones de bajo coste.

7. Documentación del diseño e implementación de la solución adoptada

7.1.Prototipo de la aplicación y diseño de interfaces

La solución propuesta se compone de dos entornos principales:

Aplicación Web (Versión Escritorio y Web Responsive)

Diseñada para que los usuarios puedan descubrir eventos, gestionarlos, comprar entradas y visualizar recomendaciones personalizadas.

La interfaz mantiene una estructura limpia y moderna:

- **Página principal**: listado dinámico de eventos con filtros por categoría, precio, ubicación y fecha. Incluye carrusel destacado.
- **Vista de evento**: ficha completa con imágenes, descripción ampliada, ubicación, invitados, aforo disponible y botón de compra.
- **Carrito de compra**: permite modificar cantidades, eliminar eventos y procesar pagos.
- **Perfil de usuario**: gestión de datos personales, tickets, tarjeta bancaria y preferencias.
- **Panel del vendedor**: CRUD de eventos, visualización de estadísticas y ventas.
- **Administrador**: gestión global de usuarios, reportes e incidencias.

Aplicación Móvil (Android)

Desarrollada siguiendo guías de diseño Material Design:

- Navegación inferior con accesos a inicio, búsqueda, recomendaciones y perfil.
- Visualización optimizada de eventos para pantallas reducidas.
- Sección dedicada a códigos QR para uso en accesos físicos.

Principios de diseño empleados

- **Coherencia visual** entre web y móvil.
- **Accesibilidad** (contrastes adecuados, tamaño mínimo de fuentes, etiquetas).
- **Responsive** en toda la interfaz web.
- **Interactividad** basada en animaciones suaves y retroalimentación visual.

7.2.Diseño lógico de la aplicación (UML y flujo de datos)

Arquitectura general

La aplicación está dividida en tres capas principales:

1. **Frontend web y móvil**

   React Web / Aplicación Android

   Consumen API REST del backend

2. **Backend (Spring Boot – Java)**
   - Controladores REST
   - Servicios de negocio
   - Capa de persistencia con JPA/Hibernate
   - Seguridad con JWT

   Gestión integral de usuarios, carritos, tickets y eventos.
3. **Microservicio de IA (Python + FastAPI)**
   - Modelo híbrido de recomendación (SVD + TF-IDF)
   - Búsquedas inteligentes
   - Reentrenamiento automático

   Comunicación por HTTP REST con el backend

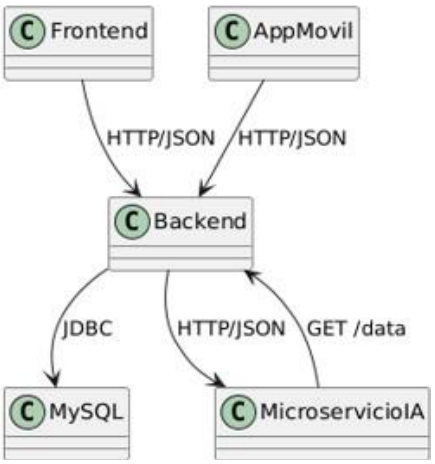
Diagramas UML incluidos

- **Diagrama de clases principal**

  Usuarios, Eventos, Tickets, Invitados, Carrito, Items, Tarjeta Bancaria.
- **Diagrama de casos de uso**
  - Cliente
  - Vendedor
  - Administrador
- **Diagrama de secuencia**
  - Proceso de compra de un ticket
  - Solicitud de recomendaciones al microservicio Python
  - Creación de un evento por un vendedor

Flujo de Datos (DFD – Nivel 1)

1. El usuario **interactúa** con la app/web
2. Envía **datos al Backend**
3. Backend **accede a la Base de Datos**
4. Backend **consulta el microservicio de IA** cuando es necesario o a los otros servicios
5. Microservicio procesa y devuelve recomendaciones
6. Backend **responde** a la app/web
7. Usuario **recibe** resultados



7.3.Descripción modular del software

Backend (Spring Boot)

- **Controladores:** Exponen endpoints REST para usuarios, eventos, tickets, compras, etc.
- **Servicios:** Contienen la lógica empresarial (validaciones, cálculos, asignaciones).
- **Repositorios:** Acceso a datos mediante JPA.
- **DTOs:** Transporte de información ligera entre capas y hacia el frontend.
- **Configuración:** Seguridad JWT, CORS, carga de recursos.
- **Modelo:** Entidades JPA que representan las tablas del sistema.

Microservicio Python

- **Módulo de carga de datos:** obtiene usuarios, eventos y tickets del backend.
- **Entrenamiento SVD:** filtrado colaborativo basado en compras.
- **Entrenamiento TF-IDF:** modelo de similitud textual por contenido.
- **Motor híbrido:** combinación de ambos modelos.
- **Sistema de auto-reentrenamiento:** detecta cambios en datos y ajusta los modelos.
- **Endpoints:** búsqueda inteligente, recomendaciones por usuario y por evento.

Frontend (Web/Móvil)

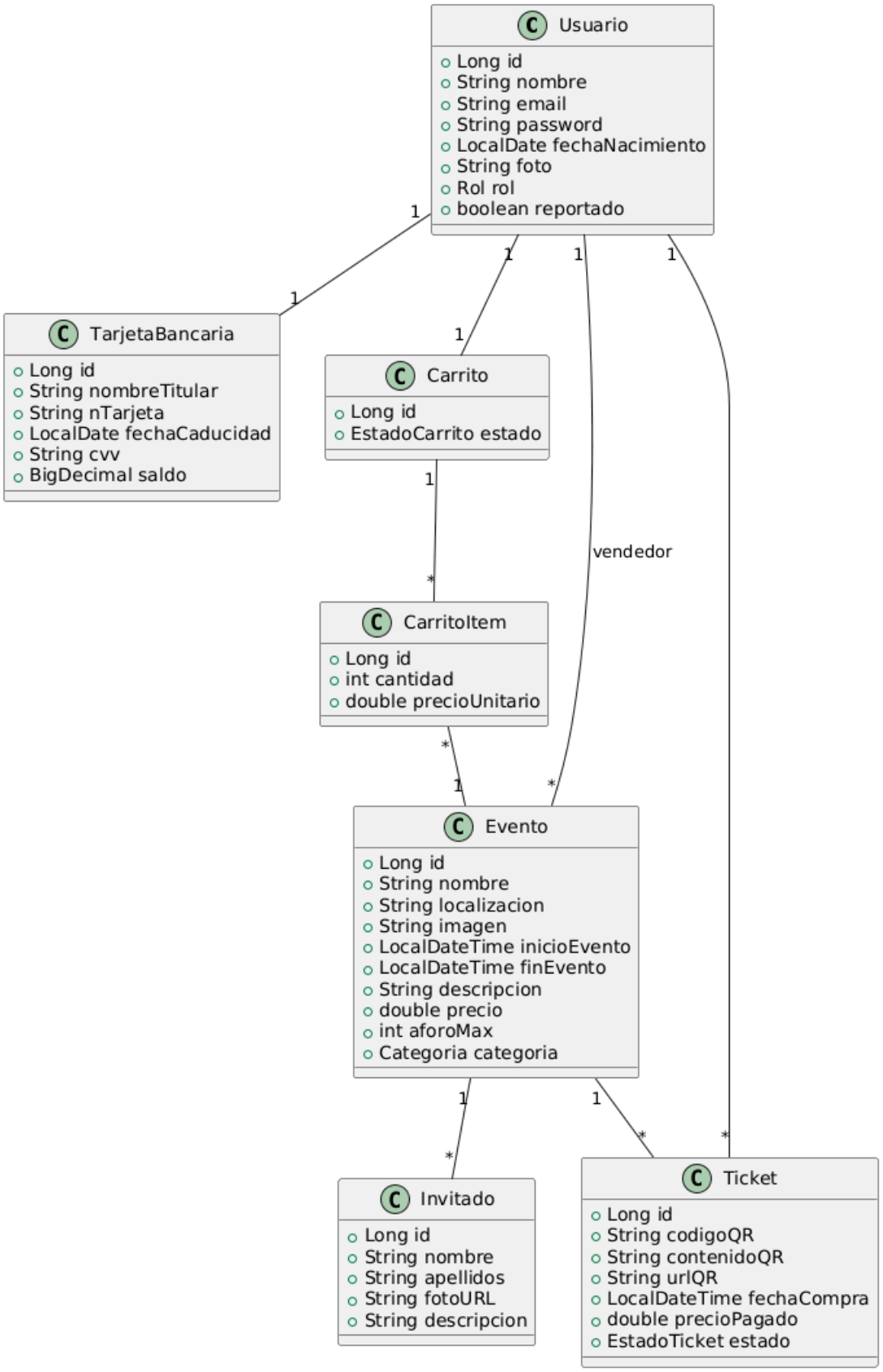
- **Servicios API:** comunicación unificada con el backend.
- **Componentes UI:** tarjetas de eventos, carruseles, formularios, modal de compra.
- **Gestión de estado:** autenticación, carrito, perfil del usuario.
- **Pantallas de navegación** Mobile/Web.

7.4.Diseño de bases de datos y diagramas

El diagrama ER incluye:

- **Usuario** (1:1 Tarjeta, 1:1 Carrito, 1:N Tickets y 1:N Eventos creados)
- **Evento** (1:N Invitados, 1:N Tickets, 1:N CarritoItem)
- **Carrito** (1:N CarritoItem)
- **CarritoItem** (N:1 Usuario, N:1 Evento)
- **Ticket** (N:1 Usuario, N:1 Evento)
- **TarjetaBancaria** (N:1 Usuario)

Estructura optimizada con claves primarias auto-generadas y relaciones bien definidas con JPA/Hibernate.


Usuario
Long id
String nombre
String email
String password
LocalDate fechaNacimiento
String foto
Rol rol
boolean reportado
TarjetaBancaria
Long id
String nombreTitular
String nTarjeta
LocalDate fechaCaducidad
String cvv
BigDecimal saldo
Carrito
Long id
EstadoCarrito estado
CarritoItem
Long id
int cantidad
double precioUnitario
vendedor
Evento
Long id
String nombre
String localizacion
String imagen
LocalDateTime inicioEvento
LocalDateTime finEvento
String descripcion
double precio
int aforoMax
Categoria categoria
Invitado
Long id
String nombre
String apellidos
String fotoURL
String descripcion
Ticket
Long id
String codigoQR
String contenidoQR
String urlQR
LocalDateTime fechaCompra
double precioPagado
EstadoTicket estado

7.5.Otras estructuras de datos utilizadas

- **Listas y colecciones dinámicas** para representar tickets, invitados, ítems del carrito.
- **DTOs** para transportar datos entre servicios sin exponer entidades internas.
- **Matrices dispersas** utilizadas internamente por el modelo SVD en Python.
- **Vectores TF-IDF** generados por scikit-learn para búsqueda y similitud.

7.6.Estudio de seguridad de la aplicación

Principales medidas aplicadas:

- **JWT para autenticación** con expiración renovable.
- **Roles y permisos:** Cliente, Vendedor y Administrador.
- **Cifrado de contraseñas** mediante BCrypt.
- **Protección de datos bancarios** en TarjetaBancaria (no se almacenan CVV reales).
- **Cabeceras de seguridad:** CORS, CSRF deshabilitado solo en API REST, XSS y sanitización.
- **Aislamiento del microservicio IA** en red interna Docker.

8.Código fuente documentado.

El código fuente está documentado en en el USB adjunto, junto con esta memoria.

9.Manual de Configuración y Funcionamiento de la Aplicación

Este apartado describe los pasos necesarios para la puesta en marcha del sistema completo **SHOWPASS**, incluyendo el backend en Spring Boot, el frontend web en React, y el microservicio de recomendaciones desarrollado en **FastAPI**.
Toda la infraestructura está contenida y orquestada mediante **Docker Compose**, facilitando su despliegue en cualquier entorno.

9.1.Requisitos previos

Para ejecutar la aplicación es necesario disponer de:

- **Docker Engine o Docker Desktop** instalado y configurado.
- **Docker Compose** integrado en el sistema (incluido en versiones recientes de Docker).
- Acceso al directorio raíz del proyecto, donde se encuentran:
    - `docker-compose.yml`
    - Dockerfiles de cada servicio
    - Archivo `.env` con las variables de entorno necesarias.

9.2.Estructura de servicios incluidos en Docker Compose

El proyecto levanta automáticamente los siguientes módulos:Todos los servicios son desplegados y conectados entre sí mediante la red interna definida en Docker.

Servicio	Tecnología	Función
showpass_backend	Spring Boot (Java 17)	API REST, lógica de negocio, seguridad y base de datos
showpass_ recomendador	FastAPI (Python)	Microservicio de IA: recomendaciones y búsqueda inteligente
showpass_ frontend_web	React + Vite	Interfaz web pública

9.3.Puesta en marcha del sistema

Para construir e iniciar todos los servicios, se utiliza el siguiente comando:

docker compose up --build -d

Significado de los parámetros:

- **up**: Crea e inicia todos los contenedores.
- **--build**: Fuerza la reconstrucción de las imágenes (útil tras cambios en el código).
- **-d**: Ejecuta los servicios en segundo plano (modo detached).

Si se desea ver logs en tiempo real durante la primera ejecución:

```
docker compose up --build
```

Verificación del estado de los servicios

Una vez ejecutado el comando, se puede comprobar el estado de los contenedores mediante:

```
docker compose ps
```

Todos los contenedores deben aparecer con estado:

`running`

Servicios visibles:

- `showpass_backend`
- `showpass_recomendador`
- `showpass_frontend_web`

9.4.Pruebas de acceso y funcionamiento

Backend – Spring Boot (Puerto 8080)

Endpoint de prueba:

`http://localhost:8080/tfg/utilidades/data`

Si responde un JSON con eventos y usuarios, el backend funciona correctamente.

Microservicio de Recomendación – FastAPI (Puerto 8000)

Forzar reentrenamiento de los modelos:

`http://localhost:8000/reload`

Endpoint para comprobar que funciona:

`http://localhost:8000/prueba`

Frontend Web – React / Nginx (Puerto 80)

Interfaz accesible en:

`http://localhost/5`

9.5.Supervisión y depuración de logs

Cada microservicio puede ser monitorizado mediante:

Backend (Spring Boot)

```
docker logs showpass_backend
```

Recomendador (FastAPI)

```
docker logs showpass_recomendador
```

Frontend Web

```
docker logs showpass_frontend_web
```

Los logs permiten detectar errores de compilación, conexiones de red y reentrenamientos automáticos del motor IA.

9.6.Detención y limpieza del entorno

A. Detener todos los contenedores

```
docker compose down
```

B. Eliminación completa (incluye volúmenes y datos)

Advertencia: esto borra datos persistentes, incluyendo bases de datos.

```
docker compose down --volumes
```

9.7.Contenido fuera de docker

La aplicación móvil está fuera de docker ya que no hay método de crear un contenedor para una aplicación móvil, aparte de que la IP de la app era una variable que cambiaba debido a que se trabajaba en la aplicación desde distintos lugares y la wifi cambiaba en uso en local.

10. Manual de usuario.

ShowPass es una aplicación **multiplataforma** (web y móvil) diseñada para la compra y venta de entradas a eventos. Permite a los clientes explorar un catálogo de eventos, comprar tickets y recibirlos en formato digital, a los vendedores gestionar sus propios eventos y a los administradores supervisar el correcto funcionamiento de la plataforma. Este manual tiene como objetivo **guiar al usuario en el uso básico de la aplicación**, desde el registro hasta la gestión de entradas.

10.1.Requisitos del sistema

Versión web

- Navegador actualizado (Google Chrome, Mozilla Firefox, Microsoft Edge).
- Conexión a internet estable.

Versión móvil (Android)

- Sistema operativo Android 9.0 o superior.
- Conexión a internet estable.
- Almacenamiento disponible para guardar tickets en formato PDF.

10.2.Acceso a la aplicación

Registro de usuario

1. Ingresar en la aplicación web o móvil.
2. Seleccionar la opción “Registrarse”, en caso de móvil en el icono de usuario arriba a la derecha te saldrán las opciones de Login o Registrarse.

3. Completar el formulario con nombre, correo electrónico, contraseña y rol (cliente o vendedor).

Inicio de sesión

1. Seleccionar la opción “Iniciar sesión”.
2. Introducir correo electrónico y contraseña.
3. Acceder al panel de usuario (cada rol tiene unas funcionalidades).

10.3. Roles principales

Cliente

- Buscar y filtrar eventos por nombre,categoría etc .
- Visualizar detalles de un evento (descripción, imágenes, horarios, recomendaciones de otros eventos).
- Añadir entradas al carrito de compra.
- Realizar pago simulado. Descargar la entrada en formato PDF con código QR único o recibirla por correo electrónico.

Vendedor

- Acceder al panel de Vendedor(gestión).
- Crear y publicar nuevos eventos (título, descripción, aforo, precio, imágenes).
- Editar eventos existentes.

Administrador

- Acceder al panel de Administrador(Administración).
- Supervisar el correcto funcionamiento de la plataforma.
- Eliminar eventos caducados o inadecuados.
- Bloquear o suspender usuarios (clientes o vendedores) que incumplan las normas.

10.4. Navegación en la aplicación

- **Menú principal**: acceso rápido a secciones: (Inicio(Eventos), Mi Perfil, Carrito, Tickets y Panel de Admin o vendedor(en caso que exista).
- **Eventos(inicio)**:listado entradas disponibles para ver más información o comprar, etc, estas pueden filtrarse por categoría o búsqueda
- **Perfil de usuario**: gestión de datos personales y tarjeta bancaria.
- **Carrito de compra**: listado de entradas seleccionadas antes del pago.
- **Tickets**: Listado de tickets comprados por el usuario, podrá descargarse en PDF o enviarse al correo del logueado.
- **Panel Admin**: En caso de ser un administrador, automáticamente al hacer login también aparecerá este apartado para desempeñar las funciones de un administrador
- **Panel Vendedor**: En caso de ser un vendedor, automáticamente al hacer login aparecerá este apartado para realizar las funciones de un vendedor

11. Plan de Formación a Usuarios de la aplicación

Este plan está diseñado para enseñar a futuros usuarios o personal de empresa a utilizar la aplicación.

Tema	Duración	Perfil
Introducción a la aplicación	10 min	Todos
Uso del catálogo de eventos	3 min	Cliente
Gestión de carrito y pagos	4 min	Cliente
Uso de la app móvil	15 min	Cliente
Gestión de eventos	7 min	Vendedor
Uso del panel administrador	5 min	Admin
Instalación y mantenimiento	15 min	Técnicos
Uso del microservicio IA	6 min	Técnicos

12. Bibliografía y fuentes de información

En el desarrollo del presente proyecto se ha requerido consultar múltiples fuentes de información con el fin de **garantizar una implementación correcta**, actualizada y fundamentada tanto a nivel técnico como metodológico. Las fuentes abarcan documentación oficial de tecnologías empleadas, artículos sobre aprendizaje automático aplicado a sistemas de recomendación y recursos digitales utilizados para la resolución de problemas concretos.

A continuación, se detallan las **fuentes empleadas**, indicando por qué se han consultado, qué aportaron al proyecto y cómo se incluyen en el documento siguiendo las normativas de citación académica.

a) Frameworks de backend (Spring Boot)

Se utilizó la documentación oficial de Spring Boot para comprender en profundidad:

- Gestión de entidades JPA
- Relaciones OneToMany, ManyToOne y OneToOne
- CascadeType y FetchType
- Uso de controladores, servicios, repositorios y DTOs

Estas consultas fueron fundamentales para garantizar un diseño robusto de la base de datos y de la API REST.

b) Machine Learning y Recomendadores (Python, SVD, TF-IDF)

Se consultó documentación técnica sobre:

- Librería Surprise (SVD)
- Cálculo de TF-IDF y similitud coseno
- Modelos híbridos de recomendación

Esto permitió diseñar un microservicio de IA optimizado, capaz de entrenar modelos automáticamente cuando cambian los datos.

c) Desarrollo móvil (Android/Kotlin o Flutter, si aplica)

Documentación consultada para:

- Arquitectura MVVM
- Consumo de API REST
- Adaptabilidad de UI (Material Design)

d) Seguridad informática

Se estudió documentación acerca de:

- Cifrado de contraseñas (BCrypt)
- Validación en formularios
- Autenticación y autorización

e) Guardado de imágenes en el backend

video informativo para:

- Guardar imágenes en base64 o url a una ruta, para poder almacenar en el backend

f) React + vite

Documentación consultada para:

- Diseño de pagina web Tailwind
- Api REST

A continuación se detallan las fuentes de información utilizadas, siguiendo el estilo bibliográfico:

Artículos y documentación técnica

Spring Framework Documentation.

https://spring.io/projects/spring-boot

React

https://react.dev/learn

Machine Learning y Recomendadores (Python, SVD, TF-IDF)

construcción del modelo híbrido de recomendación:

https://medium.com/@sales_33021/ml-building-recommendation-system-using-surprise-tf-idf-and-the-hybrid-model-fec7885f4676

FastAPI:

https://fastapi.tiangolo.com/#deploy-your-app-optional

TF-IDF:

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Cosine similarity:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html

Texto y pipelines de NLP en Python :

https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

NLP: (Natural Language Processing) es el área de la inteligencia artificial que se encarga de que las computadoras entiendan y generen lenguaje humano.

Desarrollo móvil (Android/Kotlin)

MVVM con ViewModel + LiveData / StateFlow

https://developer.android.com/topic/libraries/architecture/viewmodel?hl=es-419

Retrofit – documentación oficial

Llamadas al backend con retrofit

https://square.github.io/retrofit/

Guía de clientes HTTP en Android

https://developer.android.com/training/volley?hl=es-419

Docker Documentation.

Orquestación y docker-compose

https://docs.docker.com

Páginas web consultadas

StackOverflow. Soluciones a errores de JPA y JSON.

https://stackoverflow.com/

Guardado de imágenes en el backend

Consultado en video de youTube por la creadora *LaMalditaProgramadora*, con el título del vídeo “Cómo guardar imágenes en Base de Datos - Una "alternativa" con Spring Boot y Thymeleaf ”

https://www.youtube.com/watch?v=tgP-3nNyDD8&t=54s

React + vite

Diseño e implementación de estilos.

https://tailwindcss.com/docs/installation/using-vite

Consumición de una API :

Documento creado por : Joel Olawanle es ingeniero de software y escritor técnico

https://www.freecodecamp.org/news/how-to-consume-rest-apis-in-react/

Inteligencia artificial IA

OpenAI (ChatGPT), Gemini y Github Copliot.