票务管理系统课程设计

一. 项目背景

1.1 项目概述

本系统是一个票务管理系统，旨在为用户提供活动票务的发布、购买、管理和分析功能。主要针对事件（如演唱会、会议、展览等）的票务流程，支持用户注册、活动发布与审核、购票、订单管理以及统计分析与推荐。系统采用前后端分离架构，确保高效、安全和可扩展性。

1.2 需求范围

- **主要业务流程：**
  - 用户注册与登录。
  - 活动发布与审核。
  - 购票流程（包括座位选择、支付）。
  - 统计分析与推荐模块（基于用户行为）。
- **数据库设计：**MySQL 处理核心 CRUD 操作；NoSQL（如 MongoDB）结合向量数据库（如 Pinecone 或 Milvus）用于用户行为分析和推荐系统。
- **模块划分：**用户管理、活动模块、票务模块、订单模块、统计模块（含集成测试）。

1.3 目标用户

- 普通用户：注册、浏览活动、购票。
- 活动发布者：发布活动、查看订单。
- 管理员：审核活动、管理用户、查看统计。

二. 系统架构

- **前端：**使用 Vue3 构建交互界面，支持响应式设计。考虑到服务器端渲染（SSR）。
- **后端**：fastapi 作为 API 服务层，处理业务逻辑、数据库交互和权限控制。
- **数据库：**

- MySQL：存储结构化数据，如用户、活动、订单、票务信息。
- NoSQL (MongoDB)：存储用户行为日志（如浏览、点击记录）。
- 向量数据库 (e.g., Pinecone)：存储用户 embedding 用于推荐（基于行为分析生成向量）。

- **其他组件：**
  - 认证：JWT for token 管理。
  - 支付集成：使用内部积分方式
  - 推荐系统：使用 ML 模型（e.g., via TensorFlow.js 或后端集成 scikit-learn）分析行为数据。（可选）

- **部署：**使用 `vercel` 进行部署

三. 模块设计

3.1 用户管理模块

- **负责功能：**
  - 用户注册/登录/认证：支持邮箱/手机号注册，密码哈希（bcrypt），OAuth（可选 Google/WeChat）。
  - 登录状态监测和自动登出：前端使用 Vuex/Pinia 存储 token，后端中间件验证过期（e.g., 30 分钟无操作登出）。
  - JWT token 管理：登录后生成 token，包含角色（user/admin/publisher），刷新 token 机制。
  - 权限控制中间件：基于角色（RBAC），Express middleware 检查路由权限（e.g., admin-only for 审核）。
- **API 示例：**
  - POST /api/auth/register：注册用户。
  - POST /api/auth/login：登录，返回 JWT。
  - GET /api/user/profile：获取个人信息（需 token）。
- **前端实现：**Vue 组件（Login.vue, Register.vue），路由守卫（beforeEach）检查 token。

3.2 活动模块

- **负责功能：**
  - 活动 CRUD 操作：创建/更新/删除/查询活动（标题、描述、时间、地点）。
  - 活动类别管理：分类（如演唱会、会议），支持树形结构。

    - 活动审核流程：发布者提交 -> 管理员审核（状态：pending/approved/rejected），通知机制（email/push）。
    - 活动搜索和筛选：全文搜索（MySQL FULLTEXT 或 Elasticsearch 集成），筛选（类别、日期、地点）。
- **API 示例：**
    - POST /api/events/create：创建活动（publisher only）。
    - GET /api/events/search?q=keyword&category=concert：搜索活动。
    - PUT /api/events/:id/approve：审核活动（admin only）。
- **前端实现：**EventList.vue（列表视图），EventForm.vue（表单），使用 Vue Router 导航。

3.3 票务模块

- **负责功能：**
    - 场次管理：为活动添加场次（时间、场地）。
    - 票价设置：多票种（VIP/普通），动态定价。
    - 座位库存管理：座位图生成，库存计数。
    - 实时库存控制：使用 Redis 锁机制防止超卖，WebSocket 实时更新库存。
- **API 示例：**
    - POST /api/tickets/sessions/create：添加场次。
    - GET /api/tickets/:eventId/stock：查询库存。
    - PUT /api/tickets/lock/:seatId：锁定座位（事务处理）。
- **前端实现：**SeatMap.vue（互动座位图，使用 SVG 或 Canvas）。

3.4 订单模块

- **负责功能：**
    - 订单创建和管理：选座 -> 生成订单 -> 支付。
    - 支付接口集成：集成 Stripe/Alipay，回调处理。
    - 电子票生成：QR 码生成（qrcode library），订单状态（pending/paid/refunded）。
    - 状态管理：退票流程（库存回滚），锁定位置（定时释放未支付订单）。
- **API 示例：**
    - POST /api/orders/create：创建订单。
    - POST /api/orders/pay/:orderId：支付回调。
    - PUT /api/orders/refund/:orderId：退票。
- **前端实现：**OrderCheckout.vue（支付页面），OrderHistory.vue（订单列表）。

3.5 统计模块 + 集成测试

- **负责功能：**

    - 数据统计 API + 报表生成：销售统计、用户活跃度，使用 Chart.js 生成图表（e.g., 销售额趋势）。
    - 推荐模块：基于 NoSQL 日志分析用户行为，生成向量 embedding，查询相似活动推荐。
    - 系统集成测试：覆盖 API 端到端测试，使用 Supertest + Jest。
- **API 示例：**
    - GET /api/stats/sales?date=2023-01-01：销售统计。
    - GET /api/recommend/:userId：推荐活动。
- **前端实现：**Dashboard.vue（管理员面板，集成 ECharts）。

四、系统实现

4. 1 数据库设计

- **Mysql**:

USERS

列名	类型	约束/说明
id	int	PK
username	varchar(150)	NOT NULL
email	varchar(254)	NOT NULL, UNIQUE
password	varchar(255)	NOT NULL
role	enum(user_role)	默认 customer
avatar	varchar(512)	
created_at	timestamp	
updated_at	timestamp	

EVENTS

列名	类型	约束/说明
id	int	PK
name	varchar(200)	NOT NULL
description	text	
start_time	timestamp	NOT NULL
end_time	timestamp	
location	varchar(255)	
cover_image	varchar(512)	
status	enum(event_status)	默认 draft
created_by	int	FK → users.id（管理员）
created_at	timestamp	

updated_at	timestamp	

EVENT_SESSIONS

列名	类型	约束/说明
id	int	PK
event_id	int	FK → events.id
session_time	timestamp	NOT NULL
capacity	int	≥ 0
created_at	timestamp	

TICKET_TYPES

列名	类型	约束/说明
id	int	PK
event_id	int	FK → events.id
name	varchar(120)	NOT NULL
price	decimal(10,2)	≥ 0
total_stock	int	≥ 0
available_stock	int	≥ 0
description	text	
created_at	timestamp	
updated_at	timestamp	

SEATS

列名	类型	约束/说明
id	int	PK
event_id	int	FK → events.id
section	varchar(50)	
row	varchar(20)	
number	varchar(20)	
status	enum(seat_status)	默认 available
locked_until	timestamp	
created_at	timestamp	

TICKETS

列名	类型	约束/说明
id	int	PK
ticket_type_id	int	FK → ticket_types.id
session_id	int	FK → event_sessions.id
user_id	int	FK → users.id
seat_id	int	FK → seats.id 可空

status	enum(ticket_status)	默认 pending
qr_code	varchar(256)	UNIQUE
purchase_time	timestamp	
created_at	timestamp	
updated_at	timestamp	

PAYMENTS

列名	类型	约束/说明
id	int	PK
ticket_id	int	FK → tickets.id，
user_id	int	FK → users.id
amount	decimal(10,2)	≥ 0
payment_method	enum(payment_method)	
status	enum(payment_status)	默认 pending
transaction_id	varchar(128)	UNIQUE
payment_time	timestamp	
created_at	timestamp	
updated_at	timestamp	

REFUNDS

列名	类型	约束/说明
id	int	PK
ticket_id	int	FK → tickets.id
user_id	int	FK → users.id（申请人）
reason	text	
amount	decimal(10,2)	≥ 0
status	enum(refund_status)	默认 requested
refund_time	timestamp	
reviewed_by	int	FK → users.id（管理员）
created_at	timestamp	

- **MongoDB**:

```
{
        "_id": ObjectId,
        "user_id": int,
        "action": string "view, click, purchase",
        "event_id": int,
        "timestamp": datetime,
        "details": object "e.g., {duration: 120,
        page: 'event_detail'}",
        "session_id": string
}
```

- Neo4j：
  - Namespace: user_embeddings
  - Vector 示例：{id: user_id, vector: [float array] (e.g., 128-dim from behavior embedding), metadata: {last_updated: timestamp, preference_tags: ["concert", "conference"]}}
  - 查询：相似度搜索 for 推荐 (e.g., find similar users/events)。
  - 扩展：集成 ML pipeline (e.g., using Sentence Transformers to generate embeddings from behavior logs)。

4. 2 数据库选择

- MySQL：用户、事件、订单表（关系型，确保 ACID）。
- MongoDB：用户行为集合（{userId, action, timestamp, eventId}）。
- 向量 DB：用户向量索引，用于相似度搜索（e.g., cosine similarity）。

五. 业务流程

5.1 用户注册与登录

1. 用户提交注册信息 -> 后端验证 -> 存 MySQL -> 返回 token。
2. 登录：验证凭证 -> 生成 JWT -> 前端存储。

5.2 活动发布与审核

1. 发布者创建活动 -> 存 MySQL (pending) -> 通知 admin。
2. Admin 审核 -> 更新状态 -> 通知发布者。

5.3 购票流程

1. 用户浏览活动 -> 选场次/座位 -> 创建订单 -> 支付 -> 生成电子票。

5.4 统计分析 + 推荐

1. 收集行为 (MongoDB) -> 分析生成 embedding (向量 DB) -> 查询推荐。
2. 统计：聚合查询 MySQL 数据 -> 生成报表。

六、测试与部署

- **部署流程：**`vercel CI/CD with GitHub Actions`，Docker 镜像推送。