SEUAirline 航班预订系统 - 模块设计文档

文档创建时间: 2025-10-23
文档版本: v1.0
基于: 前后端开发总计划 (full-dev-plan.md)

---

1. 引言

1.1 文档目的

本文档旨在详细描述 SEUAirline 航班预订系统的核心模块设计，包括系统整体架构、前后端模块划分、数据库结构以及核心接口设计。本文档是详细设计阶段的产物，将作为后续开发和测试工作的重要依据。

1.2 项目概述

SEUAirline 航班预订系统是一个为用户和管理员设计的在线机票预订平台。它旨在提供一个便捷、安全、高效的航班查询、预订及管理解决方案。

---

2. 系统架构设计

2.1 整体架构

系统采用前后端分离的设计模式，通过 REST API 进行通信。

```
用户层 (PC Web / Mobile Web)
        ↓ HTTPS
前端层 (Vue3 + TypeScript)
        ↓ REST API
网关层 (Nginx / Spring Cloud Gateway) [可选]
        ↓
后端层 (Spring Boot)
        ↓
数据层 (MySQL + Redis + OSS)
```

2.2 技术栈

- **前端:** Vue3, TypeScript, Vite, Pinia, Tailwind CSS
- **后端:** Spring Boot, Maven, MySQL, Redis
- **开发工具:** VS Code, IntelliJ IDEA, Git

---

3. 前端模块设计

3.1 前端架构

前端项目遵循模块化的目录结构，以功能进行划分。

```
seu-airline-vue/
├── src/
|   ├── api/            # API 接口层 (按模块划分)
|   ├── components/     # 公共组件 (通用组件、布局组件)
|   ├── router/         # 路由配置 (index.ts)
|   ├── stores/         # 状态管理 (按模块划分)
|   ├── types/          # TypeScript 类型定义
|   ├── utils/          # 工具函数
|   ├── views/          # 页面级组件 (按用户端/管理端划分)
|   ├── App.vue         # 根组件
|   └── main.ts         # 入口文件
```

3.2 核心模块详述

3.2.1 用户认证模块

- **目的:** 管理用户注册、登录、会话和权限。
- **主要页面:**
  - `views/user/LoginPage.vue`: 用户登录页。
  - `views/user/RegisterPage.vue`: 用户注册页。
- **状态管理:** `stores/user.ts` (存储用户 Token、用户信息)。
- **核心逻辑:**
  - 调用 `api/user.ts` 中的登录/注册接口。
  - 使用 `localStorage` 或 `sessionStorage` 持久化 Token。
  - 通过路由守卫 (`router/index.ts`) 实现页面访问控制。

3.2.2 航班查询与预订模块

- **目的:** 提供航班搜索、筛选、列表展示和预订流程。
- **主要页面:**
  - `views/user/FlightSearchPage.vue`: 航班搜索及列表页。
  - `views/user/FlightBookPage.vue`: 航班预订信息填写页。
- **核心组件:**
  - `components/FlightCard.vue`: 单个航班信息卡片。
  - `components/FlightFilter.vue`: 航班筛选器（时间、价格、航司）。
- **状态管理:** `stores/flight.ts` (存储航班列表、搜索条件)。

3.2.3 订单与支付模块

- **目的:** 处理订单创建、支付、状态跟踪和展示。
- **主要页面:**
  - `views/user/PaymentPage.vue`: 订单支付页。
  - `views/user/OrderListPage.vue`: 用户订单列表。
  - `views/user/OrderSuccessPage.vue`: 支付成功提示页。
- **状态管理:** `stores/order.ts` (管理当前订单、订单列表)。
- **核心逻辑:**
  - 创建订单后跳转支付页，并启动支付倒计时。
  - 通过轮询或 WebSocket 更新支付状态。

3.2.4 管理后台模块

- **目的:** 为管理员提供数据管理和系统监控功能。
- **主要页面:**
  - `views/admin/DashboardPage.vue`: 系统概览，数据可视化。
  - `views/admin/FlightManagePage.vue`: 航班信息增删改查。
  - `views/admin/OrderManagePage.vue`: 订单查询与管理。
  - `views/admin/UserManagePage.vue`: 用户信息管理。
- **状态管理:** `stores/admin.ts` (管理后台相关状态)。
- **核心组件:**
  - `DataTable.vue` (规划中): 用于展示和操作表格数据的通用组件。
  - `ChartCard.vue` (规划中): 用于数据可视化的图表组件。

---

4. 后端模块设计

4.1 后端架构

后端采用经典的分层架构，实现高内聚、低耦合。

```
seu-airline-backend/
├── src/main/java/com/seu/airline/
|   ├── controller/     # 控制器层: 接收请求，调用服务
|   ├── service/        # 服务层: 实现核心业务逻辑
|   ├── repository/     # 数据访问层: 与数据库交互
|   ├── entity/         # 数据库实体类
|   ├── dto/            # 数据传输对象
|   ├── security/       # 安全认证 (Spring Security + JWT)
|   ├── exception/      # 全局异常处理
|   └── util/           # 工具类
```

4.2 核心模块详述

4.2.1 安全与认证模块 (Security)

- **目的:** 保护 API 接口，实现用户认证和授权。
- **核心技术:** Spring Security, JWT。
- **主要组件:**
  - `JwtTokenProvider.java`: 生成、解析和验证 JWT。
  - `JwtAuthFilter.java`: 拦截请求，从 Header 中提取 JWT 并进行验证。
  - `SecurityConfig.java`: 配置 HTTP 安全策略、密码编码器和认证规则。
  - `UserDetailsService` 实现: 从数据库加载用户信息用于认证。

4.2.2 用户模块 (User)

- **目的:** 管理用户信息、常用乘客等。
- **核心实体:** `User`。
- **主要组件:**
  - `UserController.java`: 提供用户注册、登录、信息查询/更新等接口。
  - `UserService.java`: 处理用户相关的业务逻辑。
  - `UserRepository.java`: 访问 `user` 表。

4.2.3 航班模块 (Flight)

- **目的:** 管理航班、机场信息。
- **核心实体:** `Flight`, `Airport`。
- **主要组件:**
  - `FlightController.java`: 提供航班搜索、详情查询等接口。
  - `FlightService.java`: 实现航班查询、筛选逻辑。
  - `FlightRepository.java`: 访问 `flight` 表。

4.2.4 订单模块 (Order)

- **目的:** 处理订单创建、查询、状态变更等核心交易流程。
- **核心实体:** `Order`, `OrderPassenger`。
- **主要组件:**
  - `OrderController.java`: 提供创建订单、查询订单、取消订单等接口。
  - `OrderService.java`: 实现订单创建、价格计算、库存扣减等复杂业务。
  - `OrderRepository.java`: 访问 `order` 表。
- **关键技术:**
  - 使用分布式锁 (Redis) 或数据库乐观锁处理并发下的库存超卖问题。
  - 订单状态机管理订单生命周期。

---

5. 数据库设计

数据库设计是系统的基石，以下是核心表的结构定义。

用户表 (user)

```sql
CREATE TABLE `user` (
  `id` BIGINT PRIMARY KEY AUTO_INCREMENT,
  `username` VARCHAR(50) UNIQUE NOT NULL,
  `password` VARCHAR(255) NOT NULL,
  `email` VARCHAR(100),
  `phone` VARCHAR(20),
  `real_name` VARCHAR(50),
  `id_card` VARCHAR(18),
  `avatar` VARCHAR(255),
  `member_level` TINYINT DEFAULT 1,
  `points` INT DEFAULT 0,
  `status` TINYINT DEFAULT 1 COMMENT '1:正常 0:冻结',
  `create_time` DATETIME DEFAULT CURRENT_TIMESTAMP,
  `update_time` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

航班表 (flight)

```sql
CREATE TABLE `flight` (
  `id` BIGINT PRIMARY KEY AUTO_INCREMENT,
  `flight_no` VARCHAR(20) NOT NULL,
  `airline` VARCHAR(50) NOT NULL,
  `departure_airport` VARCHAR(10) NOT NULL,
  `arrival_airport` VARCHAR(10) NOT NULL,
  `departure_time` DATETIME NOT NULL,
  `arrival_time` DATETIME NOT NULL,
  `economy_price` DECIMAL(10,2),
  `business_price` DECIMAL(10,2),
  `first_price` DECIMAL(10,2),
  `economy_seats` INT DEFAULT 0,
  `business_seats` INT DEFAULT 0,
  `first_seats` INT DEFAULT 0,
  `status` TINYINT DEFAULT 1 COMMENT '1:正常 2:延误 3:取消',
  `create_time` DATETIME DEFAULT CURRENT_TIMESTAMP,
  `update_time` DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

订单表 (order)

```sql
CREATE TABLE `order` (
  `id` BIGINT PRIMARY KEY AUTO_INCREMENT,
  `order_no` VARCHAR(32) UNIQUE NOT NULL,
  `user_id` BIGINT NOT NULL,
  `flight_id` BIGINT NOT NULL,
  `total_price` DECIMAL(10,2) NOT NULL,
```

```
  `status` TINYINT DEFAULT 1 COMMENT '1:待支付 2:已支付 3:已取消 4:已退款',
  `contact_name` VARCHAR(50),
  `contact_phone` VARCHAR(20),
  `create_time` DATETIME DEFAULT CURRENT_TIMESTAMP,
  `pay_time` DATETIME,
  FOREIGN KEY (user_id) REFERENCES user(id),
  FOREIGN KEY (flight_id) REFERENCES flight(id)
);
```

订单乘客表 (order_passenger)

```
CREATE TABLE `order_passenger` (
  `id` BIGINT PRIMARY KEY AUTO_INCREMENT,
  `order_id` BIGINT NOT NULL,
  `name` VARCHAR(50) NOT NULL,
  `id_card` VARCHAR(18) NOT NULL,
  `passenger_type` TINYINT DEFAULT 1 COMMENT '1:成人 2:儿童 3:婴儿',
  `seat_no` VARCHAR(10),
  FOREIGN KEY (order_id) REFERENCES `order`(id)
);
```

---

6. 接口设计规范

6.1 RESTful API 规范

- **请求方法:** 使用 `GET`, `POST`, `PUT`, `DELETE` 等标准 HTTP 方法表达操作。
- **URL 命名:** 资源使用名词复数，如 `/api/users`, `/api/orders`。

6.2 统一响应格式

所有接口返回统一的 JSON 结构，便于前端统一处理。

成功响应:

```
{
  "code": 200,
  "message": "success",
  "data": {
    "list": [],
    "total": 100
  }
}
```

失败响应:

```
{
  "code": 400,
  "message": "无效的请求参数",
  "data": null
}
```

6.3 核心接口列表

模块	接口路径	方法	说明
认证	`/api/auth/register`	`POST`	用户注册
	`/api/auth/login`	`POST`	用户登录
航班	`/api/flight/search`	`GET`	搜索航班
	`/api/flight/{id}`	`GET`	获取航班详情
订单	`/api/order`	`POST`	创建订单
	`/api/order/list`	`GET`	获取用户订单列表
	`/api/order/{id}`	`GET`	获取订单详情
	`/api/order/{id}/cancel`	`PUT`	取消订单
用户	`/api/user/profile`	`GET`	获取当前用户信息
	`/api/user/profile`	`PUT`	更新用户信息