**摘 要**

本课程设计旨在设计并实现一个基于 Web 的在线航空订票系统。系统采用 B/S 架构，利用 Python Flask 框架进行后端开发，结合 SQLAlchemy 进行数据库操作，以及 HTML、CSS 和 Jinja2 模板引擎构建用户界面。系统实现了用户注册、登录与身份认证功能，区分普通用户和管理员两类角色，并对不同角色赋予相应的操作权限。核心功能包括：航班信息的在线查询与展示；普通用户进行机票预订、查看个人订单和自主退票；管理员对航班信息进行增、删、改、查的管理，以及查看和取消所有用户订单。系统设计注重业务流程的清晰性和数据管理的准确性，通过数据库事务确保预订和退票操作的数据一致性。项目的实现验证了基于 Flask 构建小型 Web 应用的可行性，并为理解 Web 开发、数据库交互和权限管理提供了实践经验。

**关键词：** 航空订票系统；Web 开发；Flask；SQLAlchemy；数据库；用户认证；管理员管理；在线预订；退票

第一章 绪论

1.1 项目背景与意义

1.2 项目背景与功能概述

1.3 Flask 开发框架介绍

第二章 系统分析

2.1 功能分析

2.1.1 用户功能模块

2.1.2 管理员功能模块

2.2 非功能需求分析

2.3 系统总体结构

第三章 数据库设计

3.1 数据库需求分析

3.2 数据库概念模型设计

3.3 数据库逻辑模型设计

第四章 详细设计与实现

4.1 技术选型与架构设计

4.2 数据库模型设计与实现

4.3 各功能模块的详细设计与实现

4.3.1 用户认证与权限管理模块实现

4.3.2 航班信息管理模块实现

4.3.3 机票预订模块实现

4.3.4 订单管理模块实现

4.3.5 退票处理模块实现

4.4 关键技术应用

第五章 系统测试

5.1 测试方案

5.2 测试用例设计

第六章 总结与展望

6.1 项目总结

6.2 遇到的问题及解决方法

6.3 展望与未来改进方向

**第一章 绪论**

**1.1 项目背景与意义**

在现代信息技术飞速发展的背景下，Web 应用开发已成为计算机科学与技术专业学生必须掌握的核心技能。为了提供一个综合性的实践平台，本项目选择开发一个在线航空订票系统。航空订票系统作为一个复杂的业务应用，其开发过程涵盖了需求分析、系统设计、前后端编程、数据库交互、用户界面实现以及安全性考虑等多个关键环节。通过参与本项目的全过程，我们能够将课堂上学习的理论知识应用于实际，重点提升编程开发能力、系统设计能力和解决实际问题的能力。这不仅是对 Web 技术栈的全面实践，也是对软件工程思想的初步体验，为我们未来从事软件开发工作奠定坚实基础。

**1.2 项目背景与功能概述**

本项目作为课程设计实践，构建具备核心在线订票功能的 Web 系统原型。模拟实际业务流程，涵盖用户和管理员功能。

主要功能包括：

* 用户认证与权限管理: 用户注册、登录、退出，按角色控制权限。
* 航班信息管理: 管理员增删改查航班；所有用户查询航班。
* 机票预订: 用户预订机票、填写信息、生成订单、更新座位。
* 订单管理: 用户查看个人订单；管理员查看、搜索、取消所有订单。
* 退票处理: 用户或管理员退票，恢复座位。

通过实现这些功能，旨在深入理解 Web 开发流程、前后端交互、数据持久化及权限控制。

**1.3 Flask 开发框架介绍**

本项目后端采用 Python Flask 框架。Flask 是轻量级微框架，提供灵活性。选用 Flask 因其简洁易上手，适合课程设计项目。核心精简，通过扩展可轻松添加功能（如 Flask-SQLAlchemy）。结合 Flask 核心、SQLAlchemy 和 Jinja2 构建 Web 应用。

**第二章 系统分析**

**2.1 功能分析**

本航空订票系统根据不同用户角色的需求，主要包含以下功能模块：

**2.1.1 用户功能模块**

* **用户注册:** 系统允许用户以唯一的用户名和密码(哈希加密口令存到数据库)注册，并在成功后提示其登录。
* **用户登录:** 已注册用户可通过验证用户名和密码登录，成功则系统创建用户会话以记录其登录状态与身份信息，失败则提供错误提示。
* **用户退出登录:** 已登录用户可安全退出系统，其会话信息将被清除，恢复至未登录状态。
* **航班查询:** 所有用户可根据出发地、目的地、日期或航班号等条件查询航班信息，系统将展示匹配航班的详细列表，含航班号、航程、时刻、座位及票价。
* **机票预订 :** 仅限已登录用户选择航班并填写乘机人信息（姓名、身份证号）进行预订，系统在校验座位及事务成功后创建订单、更新座位并提示用户。
* **查看我的订单:** 已登录用户可查看其所有订单列表，包含订单号、订票时间、航班详情、乘机人信息及座位号。
* **自主退票:** 已登录用户可申请退订本人名下订单，系统在验证权限并通过事务成功取消订单、恢复座位后，会提供相应提示**。**

**2.1.2 管理员功能模块**

* **管理员登录:** 管理员使用特定账户密码登录，系统验证身份成功后建立管理员会话并授予相应权限。
* **添加航班信息:** 管理员可通过表单添加新航班，系统在校验数据有效性及航班号唯一性后，将航班信息存入数据库并初始化可用座位。
* **修改航班信息:** 管理员可修改现有航班信息，系统将校验数据有效性、处理座位调整及航班号唯一性检查后更新数据库。
* **删除航班信息:** 管理员可删除现有航班记录，但若航班存在关联订单，系统将基于数据完整性限制操作并提示。
* **管理员查看所有订单:** 管理员可查看系统中所有用户的订单列表，并能按航班号、客户姓名或身份证号等条件进行搜索和查看详情。
* **管理员取消订单 :** 管理员可取消系统中任何用户的订单，系统将执行删除订单及恢复航班座位的操作。

**2.2 非功能需求分析**

* + 安全性 (Security): 用户密码哈希存储；敏感操作严格权限控制；防范SQL注入等Web漏洞；安全管理用户Session。
  + 性能 (Performance): 系统快速响应用户请求，尤其高峰期；优化数据库查询（如索引、JOIN）提升效率。
  + 可用性 (Usability): 提供简洁直观的用户界面，易于操作；提供清晰提示和反馈；确保注册、登录、预订等流程顺畅。
  + 可靠性 (Reliability): 系统稳定运行，不易崩溃；关键业务（预订、退票）使用数据库事务，保障数据一致性。
  + 可扩展性 (Scalability): 系统设计考虑未来功能扩展；采用模块化设计降低耦合。
  + 数据完整性 (Data Integrity): 数据库设置主键、外键、约束，保障数据准确关联；预订退票操作确保相关数据同步更新。

**2.3 系统总体结构**

本系统采用经典的浏览器/服务器 (B/S) 三层架构。

* 浏览器: 用户界面展示与输入，发送HTTP请求。
* 应用服务器 (Flask): 接收请求，处理业务逻辑，通过SQLAlchemy与数据库交互。
* 数据库服务器 (MySQL): 存储系统所有数据。

这些模块通过Flask视图函数、SQLAlchemy ORM和模板渲染协同工作，共同完成系统功能。

**第三章 数据库设计**

数据库设计是系统设计的基石，它决定了系统如何存储和管理数据。这部分将阐述系统所需的数据，并设计相应的数据库结构。

**3.1 数据库需求分析**

根据系统分析阶段确定的功能需求，航空订票系统需要持久化存储以下关键信息：

* **用户信息：** 用于用户认证和授权，包括用户名、密码和用户角色。
* **航班信息：** 包括航班的基本属性，如航班号、飞机号、起降地、时间、座位数和票价。
* **客户信息：** 预订机票的乘客信息，包括姓名和身份证号。由于同一个用户可能为不同乘客订票，或不同用户为同一个乘客订票，客户信息需要独立存储并与用户关联。
* **订单信息：** 记录每次成功的订票操作，包括关联的航班、客户、订票时间、座位号等。订单是连接用户行为与航班、客户信息的桥梁。

**3.2 数据库概念模型设计**

根据需求分析，主要实体包括：

* 用户 (User): 系统注册用户 (用户名, 密码, 角色)。
* 航班 (Flight): 具体航班信息 (航班号, 起降地, 时间, 座位数, 票价等)。
* 客户 (Customer): 乘客信息 (姓名, 身份证号)。
* 订单 (Order): 预订记录 (关联航班, 客户, 座位等)。

实体间关系如下（均为一对多）：

* 用户与客户: 一个用户对应多个客户。
* 航班与订单: 一个航班对应多个订单。
* 客户与订单: 一个客户对应多个订单。

**3.3 数据库逻辑模型设计**

根据概念模型，我们将实体和关系转换为关系模式，设计出数据库中的表结构。本系统将创建以下四个主要的表：

* users 表：存储用户信息。
* flights 表：存储航班信息。
* customers 表：存储客户（乘客）信息。
* orders 表：存储订单信息。

下面详细说明各个表的字段定义：

**表名：users (用户表)**

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **数据类型** | **约束** | **说明** |
| id | INTEGER | 主键，自增 | 用户唯一标识 |
| username | VARCHAR(80) | 唯一，非空 | 用户名 |
| password | VARCHAR(512) | 非空 | 密码哈希值 |
| role | VARCHAR(20) | 非空，默认'user' | 用户角色 (user/admin) |

**表名：flights (航班表)**

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **数据类型** | **约束** | **说明** |
| id | INTEGER | 主键，自增 | 航班唯一标识 |
| flight\_number | VARCHAR(50) | 唯一，非空 | 航班号 |
| aircraft\_number | VARCHAR(50) | 非空 | 飞机号 |
| origin | VARCHAR(100) | 非空 | 出发地 |
| destination | VARCHAR(100) | 非空 | 目的地 |
| departure\_time | DATETIME | 非空 | 起飞时间 |
| arrival\_time | DATETIME | 非空 | 到达时间 |
| total\_seats | INTEGER | 非空 | 总座位数 |
| available\_seats | INTEGER | 非空 | 可用座位数 |
| price | FLOAT 或 DECIMAL | 非空 | 票价 |

**表名：customers (客户表)**

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **数据类型** | **约束** | **说明** |
| id | INTEGER | 主键，自增 | 客户唯一标识 |
| user\_id | INTEGER | 外键(users.id), 可空 | 关联的系统用户ID |
| name | VARCHAR(100) | 非空 | 客户姓名 |
| id\_card\_number | VARCHAR(50) | 唯一，非空 | 客户身份证号 |

**表名：orders (订单表)**

|  |  |  |  |
| --- | --- | --- | --- |
| **字段名** | **数据类型** | **约束** | **说明** |
| id | INTEGER | 主键，自增 | 订单唯一标识 |
| flight\_id | INTEGER | 外键(flights.id), 非空 | 关联的航班ID |
| customer\_id | INTEGER | 外键(customers.id), 非空 | 关联的客户ID |
| booking\_time | DATETIME | 非空 | 订票时间 |
| seat\_number | VARCHAR(20) | 非空 | 座位号 |

**外键关系说明：**

* customers.user\_id 引用 users.id
* orders.flight\_id 引用 flights.id
* orders.customer\_id 引用 customers.id

通过上述数据库设计，系统可以有效地存储、关联和管理用户、航班、客户和订单信息，为上层应用逻辑提供数据支持。我们使用了 SQLAlchemy 作为 ORM 工具，它能够将 Python 对象（模型）与数据库表进行映射，简化了数据库操作。

**第四章 详细设计与实现**

**4.1 技术选型与架构设计**

本项目采用 Python Flask 作为后端 Web 框架，结合 SQLAlchemy ORM 进行数据库交互，使用 MySQL 数据库存储数据，并通过 Jinja2 模板引擎生成前端 HTML 页面。系统遵循经典的浏览器/服务器 (B/S) 三层架构模式。浏览器作为表现层，负责用户界面的展示和用户交互。Flask 应用作为业务逻辑层，处理用户请求和核心业务逻辑。MySQL 数据库作为数据访问层，负责数据的持久化存储。Flask-Migrate 用于数据库 schema 的版本管理。

（在这里可以插入之前生成的简单系统结构图，并在图下方进行简要说明）。

**4.2 数据库模型设计与实现**

数据库模型在 Flight/models.py 文件中定义，使用了 SQLAlchemy 的 ORM 将 Python 类映射到数据库表。主要模型包括 User（用户）、Flight（航班）、Customer（客户/乘客）和 Order（订单）。User 包含用户名、密码哈希和角色 (role)。Flight 包含航班详情和座位信息 (total\_seats, available\_seats)。Customer 存储乘客姓名和身份证号，并通过 user\_id 外键关联到 User 模型。Order 记录预订详情，通过 flight\_id 和 customer\_id 外键分别关联到 Flight 和 Customer 模型。db.relationship 定义了模型间的关联关系，方便进行关联查询和操作。数据库 schema 的创建和更新通过 Flask-Migrate 的 flask db migrate 和 flask db upgrade 命令进行管理。

**4.3 各功能模块的详细设计与实现**

系统的各项功能通过 Flight/routes.py 文件中的视图函数实现，每个函数对应一个或一组 URL 路径，并处理特定的 HTTP 请求方法（GET/POST）。视图函数通过 SQLAlchemy 与数据库交互，并使用 render\_template 渲染页面或使用 redirect 进行跳转。

**4.3.1 用户认证与权限管理**

用户认证与权限管理保障系统只有合法用户才能访问相应功能。

用户注册功能由 register 视图函数处理。该函数接收用户提交的用户名和密码，进行非空和唯一性检查。通过 werkzeug.security.generate\_password\_hash 对密码进行安全哈希处理后，创建新的 User 对象并使用 db.session.add 和 db.session.commit 保存到数据库。注册成功后，使用 url\_for('main.login') 重定向至登录页。

用户登录功能由 login 视图函数实现。它接收用户名和密码，使用 db.session.query(User).filter\_by(username=username).first() 查询用户，并通过 werkzeug.security.check\_password\_hash 验证密码。登录成功时，将用户 ID (user.id) 存储在 Flask 的 session['user\_id'] 中，并通过 url\_for('main.index') 重定向到主页。

用户退出登录功能由 logout 视图函数处理，通过 session.pop('user\_id', None) 清除用户登录状态，并重定向。

系统权限控制通过 @login\_required 和 @admin\_required 装饰器实现。@login\_required 检查 session 中是否存在 user\_id，确保用户已登录。@admin\_required 在已登录基础上，通过查询数据库中用户的 role 字段是否为 'admin'，限制只有管理员才能访问被装饰的路由。

**4.3.2 航班信息管理**

航班管理模块实现航班数据的维护和查询功能。

航班查询和显示由 list\_flights 视图函数处理 GET 请求。它获取 URL 参数中的搜索条件（flight\_number, origin, destination, date），使用 db.session.query(Flight) 构建查询，并通过 filter 方法结合 ilike (模糊匹配) 和 db.func.date (日期匹配) 添加过滤条件。执行查询获取航班列表，并按 departure\_time 排序。函数还判断当前用户是否为管理员 (is\_admin)，将航班列表、搜索参数和 is\_admin 传递给 flights.html 模板进行渲染。

管理员添加航班由 @admin\_required 保护的 add\_flight 视图函数处理。POST 方法获取表单数据，进行非空、格式（如 datetime.fromisoformat 转换日期时间）、唯一性检查。创建新的 Flight 对象（available\_seats 初始化为 total\_seats），使用 db.session.add 和 db.session.commit 保存。成功后重定向。

管理员修改航班由 @admin\_required 保护的 edit\_flight 视图函数实现，通过 URL 参数获取 flight\_id。GET 方法获取航班数据并渲染 edit\_flight.html 表单。POST 方法处理更新数据，进行验证（包括修改总座位数时，根据已预订数量调整 available\_seats 的逻辑，以及修改航班号的唯一性检查）。更新 Flight 对象并提交。

管理员删除航班由 @admin\_required 保护的 delete\_flight 视图函数实现，只接受 POST 请求。根据 flight\_id 获取航班，尝试通过 db.session.delete 删除。为处理有关联订单的外键约束，使用 try...except 捕获 IntegrityError 并提示错误。

**4.3.3 机票预订**

机票预订是用户核心业务流程。

预订航班由 @login\_required 保护的 book\_flight 视图函数实现，通过 URL 参数获取 flight\_id。GET 方法显示预订表单 (book\_flight.html)。POST 方法获取客户姓名和身份证号，进行非空验证。检查航班 available\_seats 是否大于 0。如果可用，进入数据库事务处理：通过 db.session.query(Customer).filter\_by(id\_card\_number=...).first() 查找客户，不存在则创建新客户；将当前登录用户 ID (user\_id) 与客户关联（如果未关联）；创建 Order 对象，关联航班和客户；将航班的 available\_seats 减一。使用 db.session.commit 提交事务，确保操作原子性。预订失败时 db.session.rollback 回滚。

**4.3.4 订单管理**

订单管理包括用户查看自己的订单和管理员查看所有订单。

用户查看个人订单由 @login\_required 保护的 my\_orders 视图函数实现。它通过 Session 中的用户 ID 查找关联到该用户的 Customer 记录 (db.session.query(Customer).filter\_by(user\_id=user\_id).all())，然后获取这些客户的所有 Order 列表（通过 customer.orders）。将订单列表传递给 my\_orders.html 模板展示。

管理员查看所有订单由 @admin\_required 保护的 admin\_list\_orders 视图函数实现。它获取搜索参数，通过 db.session.query(Order) 并使用 join 关联 Flight 和 Customer 表，根据搜索条件进行过滤。获取所有订单列表并传递给 admin\_orders.html 模板展示，模板中包含搜索表单和订单列表。

**4.3.5 退票处理**

退票操作允许用户或管理员取消订单并恢复座位。

退票由 @login\_required 保护的 refund\_ticket 视图函数实现，通过 URL 参数获取 order\_id，只接受 POST 请求。它获取当前登录用户，并验证权限：是管理员或订单的合法所有者（通过检查 order.customer 是否存在以及 order.customer.user\_id 是否等于登录用户 ID）。权限验证通过后，在 try...except 块中执行事务：获取订单和关联航班对象，使用 db.session.delete 删除订单，将航班 available\_seats 加一。使用 db.session.commit 提交。退票成功或失败时 flash 提示并重定向到“我的订单”页面（或管理员订单列表页）。

**4.4 关键技术应用**

* **Flask 路由与请求处理：** 使用 @main\_bp.route 装饰器将 URL 映射到视图函数，通过 request 对象获取请求数据。
* **SQLAlchemy ORM：** 将数据库操作对象化，简化数据模型的定义和数据库交互。
* **数据库事务：** 在预订和退票等关键业务中使用事务 (db.session.commit, db.session.rollback) 确保数据一致性。
* **Jinja2 模板引擎：** 将后端数据动态渲染到前端页面，实现前后端分离。
* **Session 管理：** 使用 Flask Session 维护用户登录状态。
* **权限控制装饰器：** 将权限验证逻辑与业务逻辑分离，提高代码可读性和可维护性。

**第五章 系统测试**

**5.1 测试方案**

本项目采用手动黑盒测试，从用户角度验证系统功能是否符合预期。

* **测试环境：**
  + **硬件：** 运行开发环境的个人计算机。
  + **软件：** 操作系统 ([windows]), Python ([3.13]), Flask([3.1.0]), MySQL数据库 ([8.0.41]), 浏览器 ([edge])。
* **测试范围：**
  + 覆盖用户认证、航班管理、机票预订、订单管理、退票等主要功能。
  + 测试不同角色（未登录、普通用户、管理员）权限。
  + 测试正常及异常场景（无效输入、余票不足等）。

**5.2 测试用例设计**

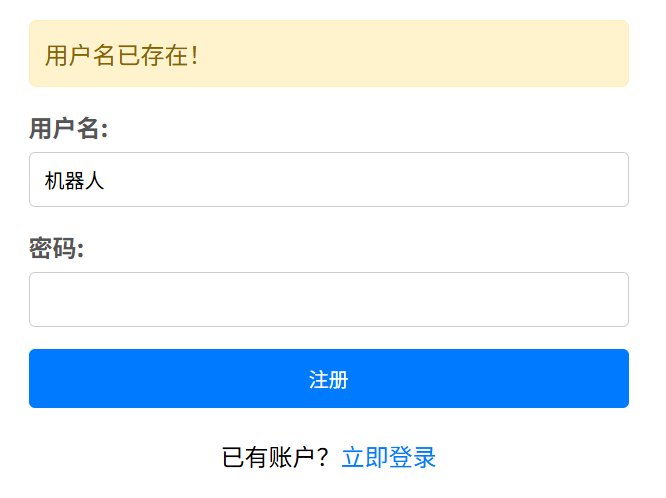
根据系统功能需求，设计以下典型测试用例。对每个测试用例执行后，记录实际结果并判断测试通过性。

**用户功能测试用例**

* (用户注册成功): 访问注册页 -> 输入有效用户名/密码 -> 点击注册。
  + 预期结果：注册成功，提示显示，重定向到登录页。



* (用户注册-用户名重复): 访问注册页 -> 输入已存在的用户名/密码 -> 点击注册。
  + 预期结果：注册失败，提示用户名已存在。



* (用户登录成功): 访问登录页 -> 输入有效用户名/密码 -> 点击登录。
  + 预期结果：登录成功，提示显示，重定向到主页。



* (用户登录-用户名/密码错误): 访问登录页 -> 输入错误用户名/密码 -> 点击登录。
  + 预期结果：登录失败，提示用户名或密码错误。

图片包含 表格

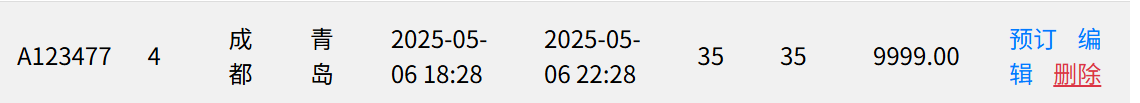
AI 生成的内容可能不正确。

* (用户退出登录成功): 用户已登录 -> 点击退出登录链接。
  + 预期结果：退出成功，提示显示，Session清除，重定向到主页。

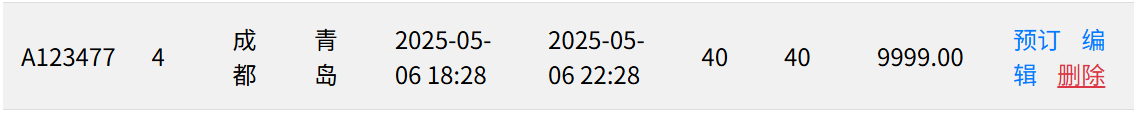


**航班管理功能测试用例**

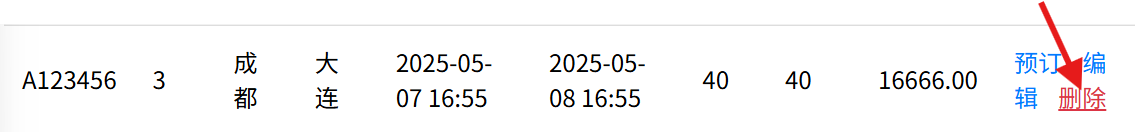
* (管理员添加航班成功): 管理员登录 -> 访问添加航班页 -> 输入有效航班信息 -> 点击添加。
  + 预期结果：添加成功，提示显示，重定向，列表中出现新航班。



* (管理员修改航班成功): 管理员登录 -> 访问航班列表 -> 点击编辑链接 -> 修改信息 -> 点击更新。
  + 预期结果：修改成功，提示显示，重定向，列表中航班信息更新，可用座位数调整。

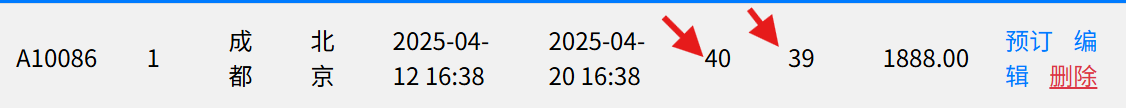


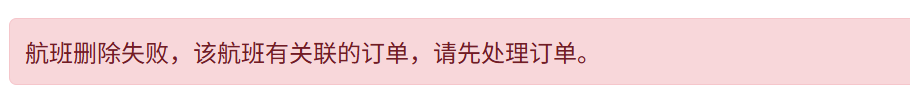
* (管理员删除无订单航班成功): 管理员登录 -> 访问航班列表 -> 点击无订单航班删除按钮 -> 确认。
  + 预期结果：删除成功，提示显示，重定向，列表中航班消失。



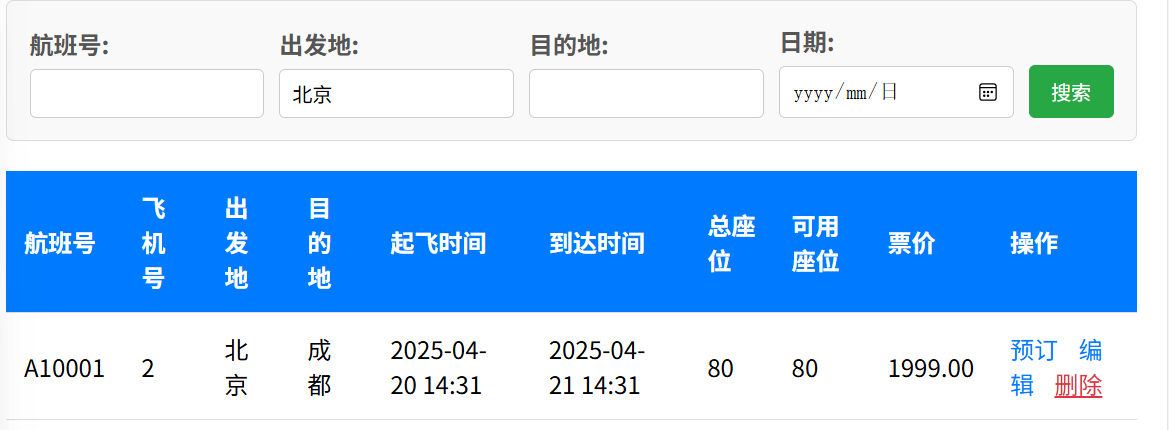


* (管理员删除有关联订单航班): 管理员登录 -> 确保航班有订单 -> 点击删除按钮 -> 确认。
  + 预期结果：删除失败，提示航班有关联订单。





* (航班查询-按出发地): 访问航班查询页 -> 输入出发地 -> 点击搜索。
  + 预期结果：显示出发地匹配航班。



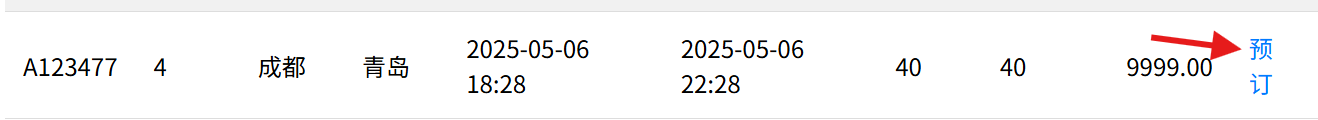
* (航班查询-按航班号): 访问航班查询页 -> 输入航班号 -> 点击搜索。
  + 预期结果：显示航班号匹配航班。

图形用户界面, 应用程序

AI 生成的内容可能不正确。

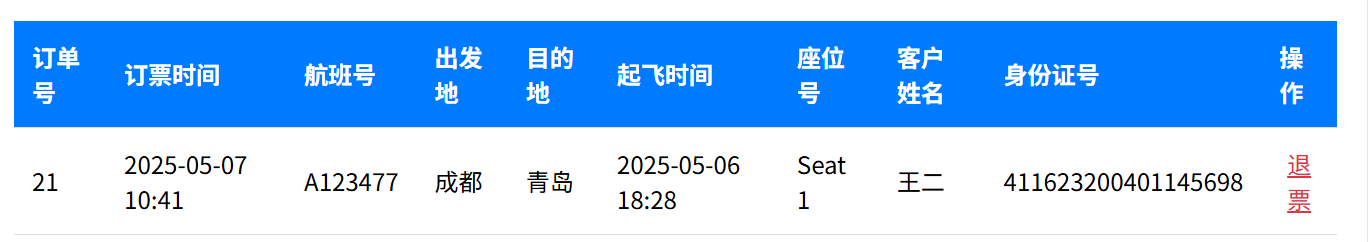
**订票与订单管理功能测试用例**

* (用户预订成功): 用户登录 -> 航班查询 -> 选择有余票航班预订 -> 输入客户信息 -> 确认预订。
  + 预期结果：预订成功，提示显示，重定向至我的订单页，订单出现，可用座位减。

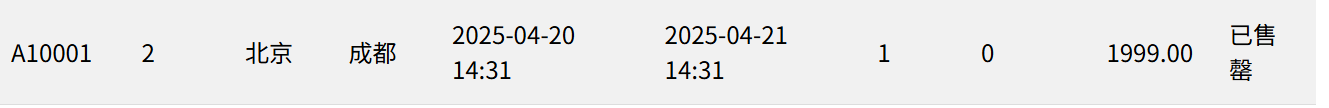


图形用户界面, 应用程序

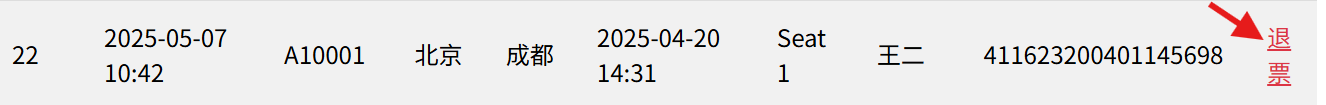
AI 生成的内容可能不正确。



* (用户预订已售罄航班): 确保航班售罄 -> 访问预订页 -> 输入客户信息 -> 确认预订。
  + 预期结果：提示航班已售罄。



* (用户自主退票成功): 用户登录 -> 确保有自己订单 -> 访问我的订单页 -> 点击自己订单退票按钮 -> 确认。
  + 预期结果：退票成功，提示显示，订单消失，航班可用座位恢复。

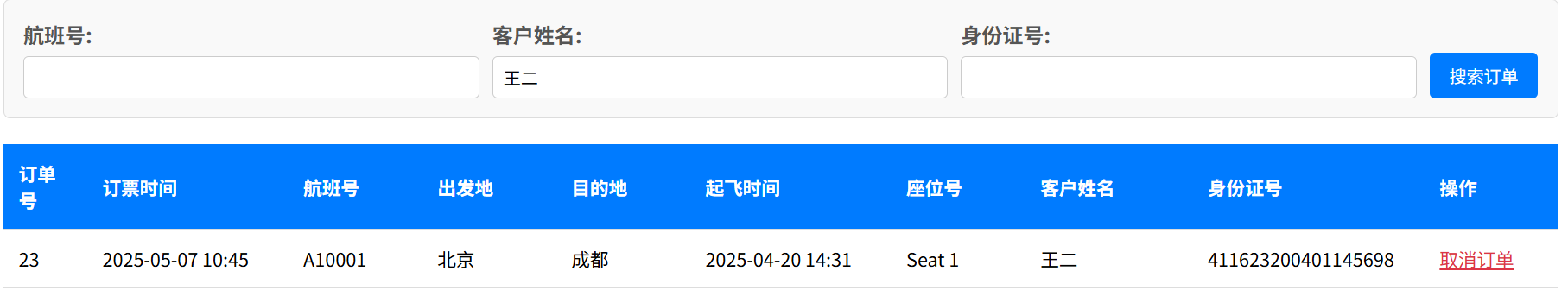
图片包含 文本

AI 生成的内容可能不正确。

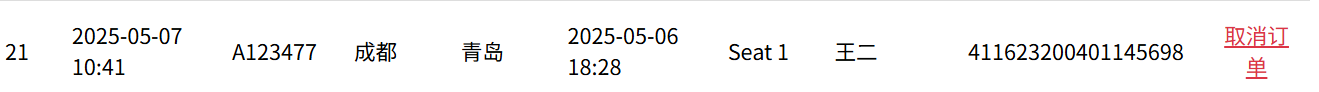
* (管理员查看所有订单): 管理员登录 -> 访问管理员订单管理页。
  + 预期结果：显示系统中所有订单列表。



* (管理员搜索订单): 管理员登录 -> 访问管理员订单管理页 -> 输入搜索条件 -> 点击搜索。
  + 预期结果：显示符合搜索条件的订单。



* (管理员取消订单): 管理员登录 -> 访问管理员订单管理页 -> 点击某个订单取消订单按钮 -> 确认。
  + 预期结果：取消订单成功，提示显示，订单消失，航班可用座位恢复。



图片包含 文本

AI 生成的内容可能不正确。

**6.1 项目总结**

本项目基于Flask框架，成功开发Web版航空订票系统。实现了用户注册/登录/退出，航班查询，用户预订/查看订单/退票，及管理员航班/订单管理功能。系统采用B/S架构，后端Flask/SQLAlchemy，前端HTML/CSS/Jinja2。利用ORM简化数据库操作，Flask-Migrate管理数据库的结构的更新。实现了Session用户管理和角色权限控制，保障基本安全。预订/退票关键业务采用事务，确保数据一致性。

**6.2 遇到的问题及解决方法**

开发中遇到环境配置、依赖、循环导入（通过调整导入解决）、路由与URL生成（BuildError，通过检查路由定义、模板url\_for、确保代码加载解决）、数据类型/验证（通过模型、迁移、后端验证解决）、事务一致性等问题。通过调试、查阅文档、实践，积累了问题解决经验。