**迭代评估报告**

　　　　　　　　　　　　　　　　　　评估日期：2025.7.10.

|  |  |  |  |
| --- | --- | --- | --- |
| 组号 | 10 | 项目名称 | 在线车票服务系统 |
| 迭代名称 | 技术原型迭代 | 实际起止日期 | 7.1-7.10 |
| 任务达成情况：  本次迭代的核心目标是将Mini-12306在线车票服务系统从纯前端原型（第一次迭代）提升至具备完整后端逻辑和真实数据交互的技术原型。我们成功完成了用户注册/登录、车次查询、购票/退票（改签未完善）、订单和车票管理与详情，以及个人中心等关键模块的后端实现。   1. 数据库设计与搭建：   完成情况：成功设计了用户、车次、订单、车票、支付等核心实体关系图（ER图），并细致定义了所有关键字段，特别是针对passengers表的student\_type\_left、users表的related\_passenger和passenger\_id，以及seats表的is\_available等。  数据量：已成功搭建MySQL表结构，并初始化导入了超过30,000条真实车次数据及相关基础数据，满足了非功能性需求中关于数据量的要求。  架构基础：配置了Redis连接池，为高并发场景下的库存预减和分布式锁提供了支撑。  亮点：在处理座位区间占用问题时，我们没有简单粗暴地通过添加大量冗余数据或列来表示，这会导致数百万条数据量，极大影响性能。相反，我们巧妙地采用了二进制与位操作的创新思路：用两位二进制表示一个站点（始发站用10，终到站用01，中间站用11），将整个行程区间表示为一个二进制掩码。通过位操作（如AND, OR）高效判断和操作区间占用情况，极大地优化了数据存储空间和查询效率，显著提升了系统的可伸缩性与性能，是本次迭代在技术方案上的重大突破。   1. 高并发架构和方案设计：完成情况：已成功设计并初步实现了基于“Redis预减库存 + RabbitMQ异步下单 + Redisson分布式锁”的抢票架构。这套机制能够有效应对瞬间大量购票请求，保证数据一致性，并提升系统响应速度。 2. 接口规范制定与编程规范确认：   完成情况：初步明确了后端RESTful API的URL结构、请求体格式和状态码约定。虽然尚未生成Swagger文档，但通过Postman Collection文件已完成了接口的初步定义和测试用例。  规范落地：团队确认并遵循Google Java编程规范进行开发，并计划在后续迭代中集成Checkstyle插件，以进一步保证代码质量和一致性。   1. 模块实现（后端核心逻辑）：   用户模块：完成了用户注册、登录（包括JWT签发）和实名认证（身份证校验、密码加密）的后端逻辑。个人中心模块（获取/更新个人信息、添加乘车人等）已完全实现，并通过了详尽的功能测试。  基础架构：成功搭建了基于Spring Boot和Spring Data JPA的后端框架，并无缝集成了Redis和RabbitMQ等核心中间件。  车次查询：实现了根据时间、出发地和目的地查询车次的接口，并为后续中转查票等进阶需求预留了扩展接口。  订单管理：完成了订单的生成、详情查询、状态流转（待支付、已支付、已完成、已取消）以及购票事务管理和退票库存回滚的核心逻辑。  票务核心：实现了购票和退票的API接口，包含了Redis预减库存、座位分配、库存回滚、旧票作废和新票生成等复杂业务逻辑，改签计划于下次迭代中完善。  抢票模块：高并发设计体现在RedisService中的原子性库存操作（decrStock()）和QueueService的RabbitMQ消息队列异步下单机制，并预留了超时处理逻辑。  支付模块：实现了模拟银行支付回调功能，驱动订单状态更新，并为电子票证的本地签名验证预留了接口。   1. 进度与质量：   本次迭代基本按计划完成了所有后端核心功能的实现，成功从纯前端模拟迈向了具备真实数据交互的技术原型；代码结构清晰，遵循分层架构设计，各模块职责明确；通过持续的代码评审和自测，保证了代码的初步质量。 | | | |
| 评审/测试的结果：  本次迭代主要通过以下方式进行评审和测试：   1. 单元测试：针对Service层中的核心业务逻辑（如购票事务、退票回滚、密码加密验证等）编写并执行了单元测试，确保了各独立功能的正确性。 2. API接口测试（Postman）：   个人中心模块：利用Postman对个人信息获取（GET /api/profile/{userId}）、更新（PUT /api/profile/update）、密码更改（POST /api/profile/change-password）、最后登录时间更新（POST /api/profile/update-last-login）和账户状态更新（POST /api/profile/update-account-status）等接口进行了全面测试。测试涵盖了成功场景、无效用户ID、验证失败（如密码不符合规范、邮箱/手机号重复）等多种情况，均按预期工作。  票务核心功能：使用预设的postman\_collection.json、postman\_prepare\_order\_test.json等集合，对购票（单人、多人、不同区间、不同席别）、退票、取消订单、支付、本人车票查询（按状态、按日期范围）以及车票详情查询（含权限验证）等接口进行了详尽的功能测试，改签计划放在下次迭代中完善。结果显示各功能均符合预期，订单状态流转和库存管理逻辑正确。  高并发测试：通过对/api/test/concurrent-test端点进行初步的并发压力测试（例如模拟1000并发请求），验证了Redis预减库存和Redisson分布式锁在保证库存原子性扣减方面的有效性，核心高并发机制已验证可行。   1. 代码评审：团队成员之间进行了关键代码模块的交叉评审，对代码风格、逻辑清晰度、潜在bug和性能优化点进行了讨论和改进。 2. 结果总结：绝大部分后端核心业务逻辑已按设计实现并验证通过；个人中心模块接口功能稳定，能够支撑前端的正常交互；高并发的底层机制已成功集成并初步验证有效，为后续的性能优化奠定了基础；数据库交互和数据导入工作正常；特别地，在座位区间占用问题上，我们采用的二进制位操作方案在测试中展现了其高效性，证明了该创新方案的有效性和先进性。 | | | |
| 问题、变更和返工：   1. ProfileController 接口路径参数问题：   问题：在ProfileController的getUserProfile方法中，最初将URL路径中的userId错误地注解为@RequestParam，导致Spring Boot无法正确映射该接口，从而引发404和400错误。  变更/返工：已及时发现并修正为正确的@PathVariable注解。通过这次返工，我们进一步加深了对Spring MVC路径变量和请求参数注解的理解，并对其他接口的注解使用进行了全面检查，确保了一致性和正确性。   1. 数据量与并发处理挑战：   挑战：在项目初期阶段，如何设计一套能够高效处理大规模数据（10k+）和高并发（1000并发）的库存和座位管理方案是核心难题。  方案变更：我们放弃了传统的简单数据添加和复杂查询方案，转而采用了更优化的技术选型和设计：结合Redis进行库存预减和分布式锁保证一致性，并通过RabbitMQ进行异步下单，最大化系统吞吐量。最关键的变更在于对“座位区间占用”问题的创新性解决，采用二进制位操作，避免了传统方案带来的巨量数据和性能损耗，这是在迭代过程中经过深入讨论和技术验证后采纳的重要优化方案。   1. 数据库字段的迭代调整：   根据对用户、乘客和车票业务理解的深入，我们对users、passengers和seats等数据库表结构进行了微调，例如为users表添加related\_passenger和passenger\_id，为passengers表添加student\_type\_left等。这些调整确保了后端数据模型能够更准确地支持业务需求。 | | | |
| 经验和教训：   1. 架构先行，细节验证：在复杂系统开发中，早期进行高并发架构设计和关键算法（如座位区间占用二进制位操作）的方案论证至关重要。这避免了在后期实现阶段陷入泥潭，并为后续开发奠定了坚实的基础。实践证明，我们对座位区间占用问题的创新性解决方案，在满足功能需求的同时，实现了卓越的性能表现，避免了潜在的数据爆炸问题。 2. API规范与早期测试的重要性：严格遵循RESTful API规范，并在开发初期配合Postman进行详细的API测试，能够显著减少联调阶段的问题。即使是细小的注解错误，也可能导致整个接口不可用，因此细致的接口测试和及时的问题发现机制至关重要。 3. 善用中间件，解决高并发：本次迭代通过实际集成和应用Redis（缓存、原子操作）、RabbitMQ（异步消息）和Redisson（分布式锁）等中间件，有效解决了抢票场景下的高并发和数据一致性挑战。这加深了我们对分布式系统原理和实践的理解。 4. 持续集成与交付的意识：虽然本次迭代主要侧重后端实现，但明确的前后端职责分离和接口定义，为后半段的前后端联调做好了充分准备。每次提交代码后进行自测和团队内部评审，确保了代码质量的逐步提升。 5. 文档的重要性：详尽的迭代计划、功能说明文档和Postman测试集是团队协作的基石。它们不仅清晰记录了设计思路和实现细节，也极大地便利了团队成员之间的沟通和测试验证工作。 6. 迭代式开发的灵活性：通过将项目分解为多次迭代，我们能够更灵活地应对需求变化和技术挑战。第一次迭代的前端原型为后端开发提供了明确的目标，而本次后端迭代的完成，也为后续的性能优化和创意功能扩展提供了坚实的基础。 | | | |