84-作业报告

严天乐 2310307218

程序功能介绍：

本应用程序基于 QT 工具开发，使用 C++语言，实现了一个简单的铁路订单管理系统，可以实现管理员身份下对车票、订单、用户等的管理以及用户身份下对车票的选中、购买、收藏等功能，另外集成了行程收藏、车票生成等功能。

在登录界面可以选择管理员或用户身份进行登录，管理员账号以硬编码的形式写入后端数据库，整个程序仅有一个管理员账号，登录后主界面被分为 5 个模式，分别是“用户信息”、“订单记录”、“所有车次”、“售票统计”、“反馈信息”，在用户信息界面可以看到当前注册过这个程序的所有用户的信息，并且可以进行增删查改。订单记录部分记录了本程序所有用户进行的所有的订单信息。所有车次是在用户界面可以被查询到的所有的车次的信息，由管理员进行手动录入，支持增删查改，售票统计记录了每个在“所有车次”中存在的车次的车票售出数量与销售额。反馈信息是当以用户模式登录时，用户模式可以对车次进行留言，留言会展示在这个模块。

如果以用户身份进行登录，同样也有 5 个功能模块，主界面是“所有车次”，管理员界面录入的所有车次会显示在这个界面，可以使用“查询”功能对需要的车次进行查询，查询使用的关键字可为车次名称、起点城市、终点城市等，对需要的车次选中后将其加入待支付订单，后续该车次会被加入第二个模块待支付订单。

在待支付订单界面，可以通过选中来进行付款，只要账户内余额充足就会支付并买下车票，删除待支付订单中的订单，如果余额不足则会弹出提示，此时可以在设置界面通过充值来提高账户余额，进行付款。

在行程记录模块，记录了之前所有的已经支付过的订单，双击行程即可进入行程界面，会记录这次购票对应的车票信息，在行程内部界面可以通过点击收藏来对指定的行程进行收藏，对应的车次将会被转到收藏与记录模块。

收藏与记录这个模块记录了之前在行程记录收藏的所有车票，另外在这个模块当中集成了车票生成器，点击“生成车票”的按钮之后会进入车票编辑界

面，一张待编辑的车票会展示出来，可以对上面的既有控件进行修改以满足自己的需求，点击界面上的保存按钮可以选择文件路径，就可以将生成的车票以.jpg 文件的形式保存下来。

最后在我的车次板块，可以支持自定义添加自己的车次，可以是自己未来的出行计划，也可以是一些理想中的车次，完成添加之后管理员界面可以看到对应的车次。

总的来说本程序参考主流的车票预定软件，设计了管理员和用户两个不同的界面，分别实现了多个模块，核心功能就是对车次进行增删查改，并对车次进行一些诸如购买、收藏、车票生成等功能。

项目各模块与类设计细节

系统总体架构：本项目是基于 QT 框架下的一个粗糙度火车票购票管理系统，采用 MVC 设计模式，实现了用户管理、车次管理、购票管理、订单管理、收藏管理等功能模块。系统分为前台用户界面和后台管理界面，支持不同角色的用户登录和操作。前端使用 QT 自带的 QT Widgets 框架，后端使用 C++语言，数据存储使用 SQLite。将整个程序的功能划分为多个模块：用户管理、车次管理、票务管理、订单管理等功能模块。系统的设计架构为：表示层、业务逻辑层、数据访问层，分别对应创建的用户界面、处理核心业务流程的部分、数据库交互与数据持久化。

系统核心类结构：本项目主要实现用户管理、车次管理、票务管理、订单管理、收藏管理等功能，分别将这些功能分发给对应的类来控制对应的窗口进行处理。

首先是程序的核心管理类 Runtime 类，Runtime 类在程序运行时托管整个程序的运行逻辑，负责全局状态管理、数据库协调、处理业务逻辑与管理资源等，负责维护应用程序运行时的全局状态、控制设置参数、封装所有的数据库访问操作、实现核心的业务逻辑、负责各模块间的数据交流、处理信息的加载与保存等，Runtime 类与 Config 类、DBManager 类进行协作管理全局程序。在 Runtime 类中，定义关键的成员变量 m_context、m_config、m_dbmanager 三个对象，用来管理角色、基础设置、数据操作。自身构造全局唯一实例，在程序

启动时自动初始化，程序退出时自动清理。

在 Runtime 类中实现了本程序中需要用到的绝大部分数据库操作方法，包括查询与购票等业务逻辑，在每一个逻辑中都添加了充足的条件检查与错误处理，以提升程序整体的稳定性。在设计时选用了单例模式，以避免重复的资源分配，也可以简化各个模块的访问方式。整体上，界面层与业务层都在 Runtime 类中实现，而数据由 DBManager 类进行封装。实现了业务逻辑的统一性，减少了各个类直接相互访问的不便与可读性降低，简化了整体复杂度。

数据管理相关的类有 DBManager 类与 Config 类，DBManager 类作为数据库访问的核心封装，其实现采用了分层设计理念，首先通过 check()方法验证数据库驱动可用性，在 start()方法中根据配置选择启动 SQLite 连接，直接连接本地数据库文件，通过 QSqlDatabase 实现底层操作。该类提供了 query()方法获取查询对象，将原始数据库操作封装在内部，对外暴露统一的接口，其 getDB()方法允许高级用户直接访问数据库引用，resetDatabase()方法则实现了数据库重置功能，通过读取资源文件中的 SQL 脚本重新初始化数据库结构。DBManager 类在设计时进行了充分的异常情况考虑，对每一个操作都包含了异常处理与报错，确保程序的稳定性。

Config 类作为配置管理系统，采用 INI 文件格式存储应用程序设置，其 load()方法从 config.ini 文件中读取包括应用名称、数据库连接参数、UI 样式等配置信息，如果配置文件不存在则调用 init()方法从资源文件初始化默认配置。该类使用 QSettings 进行配置读写，将配置项按功能分组管理，SQLConfig 组包含所有数据库相关参数，UIConfig 组管理界面样式设置 Config 类在 Runtime 初始化时被加载，为 DBManager 提供必要的连接参数，同时服务于整个应用程序的配置需求，通过值语义来传递配置数据，确保了和数据一致性。Config 在找不到配置文件时会自动创建默认配置，这种自恢复机制增强了系统的健壮性，同时也可以在设置被不小心修改后进行简单的恢复（虽然这也只能用于开发时的调试情况）。

以上是与数据管理、程序设置、运行核心逻辑相关的类，后续是具体负责各项业务的类，分别有窗口类 UserWindow 和 AdminWindow 类，对话框类 LoginDialog、UserDialog、AdminDialog、GoodsDialog、PurchaseDialog、

RecordationDialog、RechargeDialog、SearchDialog。

UserWindow 类作为用户主界面，通过 tabWidget 控件实现了多视图管理，包含车次查询、购物车、订单记录、我的车次、收藏管理五个子模块。在初始化阶段，该类首先调用 initTableView 方法对各子模块的 QTableView 进行统一配置，设置选择行为、编辑策略和交替行颜色等视觉属性，并分别为其绑定对应的数据模型，包括 QSqlQueryModel 和 QStandardItemModel 两种类型。refreshTableView 系列方法通过 Runtime 类的对应接口获取最新数据，其中车次查询模块还实现了按车次名称、起止城市等条件的动态筛选功能。事件处理方面，on_tableViewGoods_doubleClicked 等槽函数实现了双击查看详情的交互，而 on_pushButtonGoodsCart_clicked 等按钮响应函数则封装了购票、退票等核心业务流程，通过创建 PurchaseDialog 等子对话框完成具体操作。对于“我的车次”功能，额外提供的车次管理功能通过 on_pushButtonMyGoodsAdd_clicked 等方法实现车次的增删改查。

AdminWindow 类作为管理员主界面，采用类似的框架设计但功能侧重不同，主要包含用户管理、车次管理、订单管理、数据统计、反馈管理五大功能模块。该类同样使用 initTableView 方法对各模块的表格视图进行标准化配置，并建立与对应数据模型的关联。refreshTableViewUser 等方法通过 Runtime 提供的管理接口（如 getUserList）加载全量数据，search 系列方法（如 on_pushButtonUserSearch_clicked）则实现多条件组合查询功能。在用户管理模块，通过 on_pushButtonUserAdd_clicked 等方法提供用户账号的创建、修改和删除功能；车次管理模块管理车次信息；数据统计模块利用 getStatisticsList 方法生成销售数据分析报表。所有数据操作都通过 Runtime 统一进行权限校验和数据持久化。两个窗口类都实现了状态栏动态显示当前用户信息，并使用一致的视觉处理。

对话框类由主窗口上的业务逻辑唤起，用于处理具体的业务，针对上面的主窗口上不同的模块，显然需要根据需要建立一些具体的对话框类来处理不同情况下的需求，下面是对每个类实现功能与细节的简介：

LoginDialog 作为系统入口，主要处理用户认证流程。验证普通用户身份的 loginUser()和管理员验证的 loginAdmin()通过数据库查询核对凭据；而

on_pushButtonRegister_clicked()负责启动注册流程，创建用户信息对话框。登录成功后，该对话框会触发主窗口的初始化并传递用户上下文。

UserDialog 负责管理用户信息，提供三种操作模式。download()方法从数据库加载用户数据填充表单，upload()处理数据提交，根据当前模式调用不同的 SQL 语句；on_pushButtonRecharge_clicked()会调出充值对话框，而界面元素会根据当前用户角色动态调整可编辑字段。

AdminDialog 类负责管理管理员账号信息，将唯一的管理员账号以硬编码的形式填充在程序当中。

GoodsDialog 是车次信息管理界面，其核心方法包括处理图片上传的 on_pushButtonImage_clicked()，管理讨论区消息的 updateMessageList()，以及根据当前模式（创建/修改/查看）动态调整控件状态的初始化逻辑。表单提交时会验证时间、票价等关键业务数据的有效性。

PurchaseDialog 处理购票数量选择和订单生成的行为。通过 download()实时获取车次余票信息，通过 on_pushButton_clicked()方法会验证库存并调用 Runtime 的接口，界面中的 spinBox 控件实现了购票数量与总价的联动计算。

RecordationDialog 负责订单详情展示和后续操作。实现基本的订单数据加载功能，on_recshoucang_clicked()方法实现了车次的收藏，将车次信息写入本地文件，文件会存放在可执行文件的同级目录，以.txt 形式存储。

RechargeDialog 实现充值功能。其 on_pushButton_clicked()方法通过 Runtime 调用余额更新接口，内置的数值范围验证确保充值金额合理性，操作结果会即时反馈给用户。

SearchDialog 负责所有的查询工作。通过 addOption()动态添加查询字段，formatLike()和 formatTime()等方法将界面输入转换为 SQL 条件，最终生成的过滤表达式可供各个模块复用，实现了查询逻辑的统一管理。

另外还有一个较为独立的类：TicketGenerator 类，用于处理车票生成这个小业务，在这个类中用槽函数将所有的文本框与修改文字的行为联系起来，并且实现了车票的选择路径保存功能。

最后，在完成这些类的内部功能之后，通过 main.cpp 主程序将类之间彼此串联起来，作为程序入口文件，负责初始化应用程序框架，协调核心组件的生

命周期管理，在运行时首先初始化 Runtime 全局单例并加载系统配置，随后创建并显示 LoginDialog 作为初始界面；根据登录结果动态决定实例化 UserWindow 或 AdminWindow 主界面，并在主窗口关闭后执行资源清理。通过简洁的 while 循环结构支持多次登录会话而不重启进程，避免额外的开销。

通过以上运行时类、数据类、窗口类、对话框类彼此配合，并集成在主函数中，构成了整个程序的运行逻辑。

组内分工：

功能性文档——严天乐

源代码编写——严天乐

程序界面美化——严天乐（5 月 31 号前佩依力做了一部分，但终版是我做的）

程序运行录屏——严天乐

GitHub 仓库的建立与源码上传——严天乐

作业报告撰写——严天乐

（总之几乎所有事情都是一个人做的，两个队友在期中后就先后决定缓考了，整个队伍只剩我一个人）

项目总结与反思：

在这次大作业的全过程中，最重要的感受就是——难!! 感觉真正上手去写一个项目真的好难，与平时的作业有很大的不同在于，真正需要去管理一个有特别多文件的项目。头文件、C++源代码文件、ui 文件每个都需要去手动管理，要求对整个程序架构都需要做到相当熟悉，否则写到一半都不知道自己在干什么，以及如何将这些类之间合理地串联起来。

第一个难点就是架构的搭建。在本次项目的书写过程中，我认识到面对具体的项目，一定不能闭门造车，可能自己一个人辛辛苦苦钻研也比不上别人的架构模板，在本次编写的前期，我都是自己设计所有的类的关系，试图通过一个主窗口类串联所有其他类，最后虽然可以运行但效果很差，后面我对整个项目进行了全面的重写，学习了在项目设计中的标准流程，提前设计了 Runtime

类用于实现全局的管理，并且将类分为窗口类、对话框类、数据类，仅每个类专注于实现自身的功能，而不涉及彼此复杂的嵌套关系，减少互相的耦合以降低编写难度。最后由 Runtime 类集成所有的运行逻辑，使用单例模式，便于整个程序运行时的管理。

第二个难点就是 debug 之难。只要某一步有一点错误，QT 就无法运行项目文件，因此每次修改都得小心谨慎，修改一步尝试运行一步，否则如果一次修改太多，其中某一步出现错误，甚至可能被迫需要放弃全部的修改。而且由于项目里的文件越来越多，需要时刻清楚认识每个类之间的关系，否则实在是难以找到合适的地方下手修改。因此给每个类、每个方法命名时需要不厌其烦地取有具体意义的名字（哪怕很长）。

第三个难点就是构建界面并将界面与源代码中的内容联系起来。在进行界面构建时，还需要提前对窗口里的内容进行充足的思考，将控件进行合理的摆放，然后将信号与合适的槽函数相连。在一开始写的时候我偷懒先粗略设计界面，再随着源码的书写逐步补全控件，但这样反而事倍功半，经常会造成难以察觉的 bug，大大降低效率。后面我只能先花很大的功夫设计界面，再思考进一步的实现。而且整体写好后我感觉很难进行修改，无论是修改源码还是控件，可能都会造成牵一发而动全身的问题，使得修改举步维艰。

在写本次大作业之前本来是有很丰满的理想的，但无奈现实很骨感。现在回看当时自己所写下的功能性文档，唯有羞愧能表达我的心情。可以说是该实现的功能一个也没实现，虽然当时在做可行性评估时就已经有感觉这些功能还是很难实现的，但是也没有料想到结果如此之差。通过本次程设大作业练习，我深刻做以下反思：

1. 在设计项目的功能时，一定不能好高骛远，从最基础的功能做起，不要过分高估自己的实力，还是要脚踏实地去写一些东西才能体会到整个过程的不容易，才能发现自己的空想是多么幼稚。同时设计功能时一定要心中有大致的想法，而不是仅仅一个空壳，导致最后实现的时候才发现做不出来或者没有好的想法，白白浪费时间。
2. 在项目的全过程尤其是初期一定不能闭门造车，作为这个领域的新手小白，还是应该跳出舒适圈去多看看别人的项目，多学学别人的方法、范

式等，可以避免自己少走好多弯路，甚至提前解决一些问题，同时也可以寻找一些灵感来辅助创作。

3. 产品经理是重要的。只有先明确想要做什么才能做得出来，在前期设计的时候就应该更详细地将想法写下来，而不是写到哪一行算哪一行，能更好地提升效率。
4. 项目开发完全是心急吃不了热豆腐，一定不能把工作压缩到几天内完成，因为那样会导致时间紧张忙中出错，但是也不能三天打鱼两天晒网，必须每天都写，保持写项目的状态才是最佳的。

虽然有那么多可以反思的地方，但是本次大作业我同时也踩了这些坑，犯了这些错误，这也就是我这次作业质量如此之低的原因，如果再有下次大作业，我会吸收这次的教训，希望能借这些经验达到更好的效果。

最后，小组分工需要进行更合理的分配，把所有任务全部堆给一个人是不合理的。下次再有组队大作业时，会更加合理地考虑工作量分配问题，保证每个组员的实际工作量相近以确保公平性以及更好地组织工作。