# 目录


一 引言 ..... 4
1.1 编写目的 ..... 4
1.2 背景 ..... 4
二 任务概述 ..... 4
2.1 项目概述 ..... 4
2.1.1 项目来源及背景 ..... 4
2.1.2 项目目标 ..... 4
2.1.3 项目功能概述 ..... 5
2.2 用户特点 ..... 5
2.3 假定和约束 ..... 5
三 需求设计 ..... 6
3.1 需求概述 ..... 6
3.1.1 功能需求 ..... 6
3.1.2 非功能需求 ..... 9
四 总体设计 ..... 15
4.1 模块基本信息 ..... 15
4.2 功能概述 ..... 16
4.3 算法 ..... 19
4.4 模块处理逻辑 ..... 20
4.5 接口 ..... 23
4.6 性能 ..... 24
4.6.1 时间特性 ..... 24
4.6.2 灵活性 ..... 24
五 技术计划 ..... 24
5.1.时间安排 ..... 25
5.2.人员分工 ..... 25
六详细设计 ..... 25
6.1 项目的思路逻辑 ..... 25
6.2 功能说明 ..... 26
6.3 技术点说明 ..... 26
6.4 数据结构说明 ..... 27
6.5 重点代码说明 ..... 27
七 源代码 ..... 28
7.1 Main 模块 ..... 28
7.2 order 模块 ..... 32
7.3 PriorityQueue 模块 ..... 41
7.4 Trade 模块 ..... 49
7.5 user 模块 ..... 51
7.6 登录界面模块 ..... 52
八 测试 ..... 62
8.1 单元测试 ..... 62
8.1.1 MaxPriorityQueue 类测试 ..... 62

8.1.2 MinPriorityQueue 类测试 ..... 64
8.1.3 后端测试代码 ..... 66
8.2 系统测试 ..... 68
8.3 测试总结 ..... 72
九 用户手册 ..... 72
9.1 系统概述 ..... 72
9.2 使用说明 ..... 73

# 表目录

表 3.1 功能编号和优先级 ..... 8
表 3.2 用户名 ..... 9
表 3.3 密码 ..... 10
表 3.4 股票信息 ..... 10
表 3.5 订单号 ..... 10
表 3.6 订单金额 ..... 11
表 3.7 订单数量 ..... 11
表 3. 8 订单状态 ..... 12
表 3. 9 用户信息 ..... 12
表 3. 10 股票信息 ..... 12
表 3. 11 股票信息 ..... 13

表 4. 1 模块基本信息表 ..... 16
表 4. 2 用户信息 ..... 23
表 4. 3 股票信息 ..... 24
表 4. 4 订单信息 ..... 24

图 3. 1 顶层数据流图 ..... 6
图 3. 2 第 0 层数据流图 ..... 7
图 3. 3 第 1 层数据流图 ..... 7

# 图目录

图 4. 1 总系统程序流程图 ..... 16
图 4. 2 查询订单程序流程图 ..... 17
图 4. 3 添加订单程序流程图 ..... 18
图 4. 4 删除订单程序流程图 ..... 19
图 4. 5 一般用户系统流程图 ..... 20
图 4. 6 整体系统流程图 ..... 21

图 4. 7　撮合模块系统流程图 . . . . . 22
图 4. 8　匹配模块系统流程图 . . . . . 23

图 8. 1　 MaxPriorityQueue 测试结果.....64
图 8. 2　MiniPriorityQueue 测试结果.....66
图 8. 3　后端测试结果.....68
图 8. 4　登录界面.....69
图 8. 5　系统主界面.....69
图 8. 6　添加订单页面.....70
图 8. 7　删除订单界面.....70
图 8. 8　显示所有订单界面.....71
图 8. 9　自动撮合订单.....71
图 8. 10　显示交易历史.....72

# 一 引言

## 1.1 编写目的

这篇文章的编写目的主要是为了开发撮合交易系统为系统做一个总体的结构设计，经评审后进一步细化，分别对每一模块进行详细细化的解决方案、接口和数据库等方面的设计，明确描述所有输入输出参数、类型逻辑算法以及调用关系。作为开发人员和测试人员进一步变成和编写测试用例依据。

## 1.2 背景

a.待开发系统的名称：撮合交易系统

b.开发者：王子俊 李嘉康 徐凡鑫 郭思源

c.用户：在各种金融市场，如股票、期货、外汇等的投资者

# 二 任务概述

## 2.1 项目概述

### 2.1.1 项目来源及背景

随着互联网技术的不断发展，交易方式的创新与变革日新月异。撮合交易平台作为其中的一种重要形式，越来越受到市场的关注。撮合交易平台撮合交易系统是一种电子化交易系统，主要用于在金融市场中撮合买卖双方的交易订单。它的基本原理是将买卖双方的订单进行匹配，并在满足特定条件的情况下执行交易。撮合交易系统广泛应用于各种金融市场，如股票、期货、外汇等，为投资者提供了方便、快捷且安全的交易环境。

### 2.1.2 项目目标

实现撮合交易系统，系统通过一系列算法和规则，根据价格、数量和时间等因素，自动匹配合适的交易订单，促成交易的执行，通过信息发布、自动匹配、

交易协商等功能，降低交易成本、提高效率，增加透明度，方便快捷，信用评价机制保障安全。

2.1.3 项目功能概述

a.订单匹配与撮合：撮合交易系统的核心功能是对买卖双方的订单进行匹配和撮合。系统需要根据预设的规则和算法，如价格优先、时间优先等，对订单进行自动撮合。

b.交易执行与确认：一旦订单撮合成功，系统需要能够自动执行交易，并生成交易确认信息，通知交易双方。

c.订单状态查询与管理：用户应能够随时查询订单的状态，包括待撮合、已撮合、已成交、已取消等。系统还需要提供订单管理功能，允许用户修改或取消订单。

## 2.2 用户特点

a.追求效率与便利性：撮合交易平台通过自动化匹配算法，大幅度提高了交易的效率。用户倾向于利用这些平台来节省时间，快速找到合适的交易伙伴。这种对效率的追求也体现在用户偏好使用简单易操作的交易系统，以减少交易过程中的复杂性和人为错误。

b.注重成本效益：用户选择撮合交易平台的一个重要原因是能够显著降低交易成本。这包括减少了传统市场中需要耗费大量时间和精力寻找交易对象的环节。因此，这类用户通常对价格敏感，寻求最优的交易条件。

c.透明度和信用评价的重视：撮合交易平台上的所有信息公开透明，买卖双方可以更加了解市场情况，减少信息不对称带来的风险。用户高度重视平台的信用评价机制，这帮助他们降低了交易风险，确保了交易的安全性。

d.社区化和个性化服务的期望：随着撮合交易平台的发展，用户越来越期望平台能提供社区化和个性化的服务。他们希望通过加入特定的社区来增强与其他用户的互动，并通过定制化服务满足自己独特的需求和偏好。

e.适应新型交易模式的能力：撮合交易平台的用户需要适应数字化、网络化的新型交易模式，这对他们的学习和适应能力提出了要求。用户必须愿意接受新技术和新流程，才能充分利用平台的优势。

## 2.3 假定和约束

假定股市中存在恒大集团、山西焦煤、万达电影三种股票类型。

# 三 需求设计

## 3.1 需求概述

通过调查，撮合系统++主要面临的问题主要有用户查看股票信息、用户买入卖出订单选择、系统选择最优订单撮合、订单修改状态、用户查看撮合情况等，我们总结得出撮合系统需要解决的问题，让用户可以直观清晰地看到股票的公开信息，实现公开的便捷性，安全性。

3.1.1 功能需求

a. 功能划分

系统功能组成

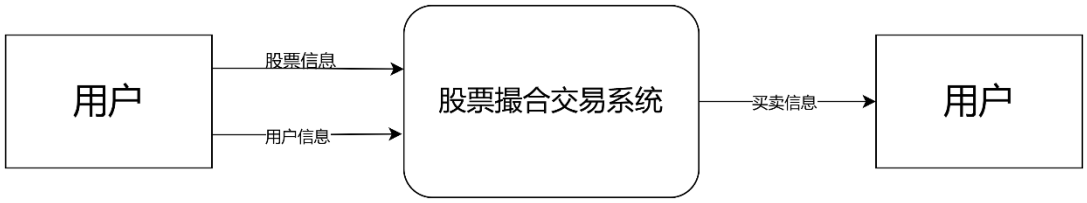


图 3. 1 顶层数据流图

当用户登录\注册撮合系统之后，通过股票信息买入或卖出股票订单，系统通过撮合操作，选取合适订单成功配对，反馈给用户。

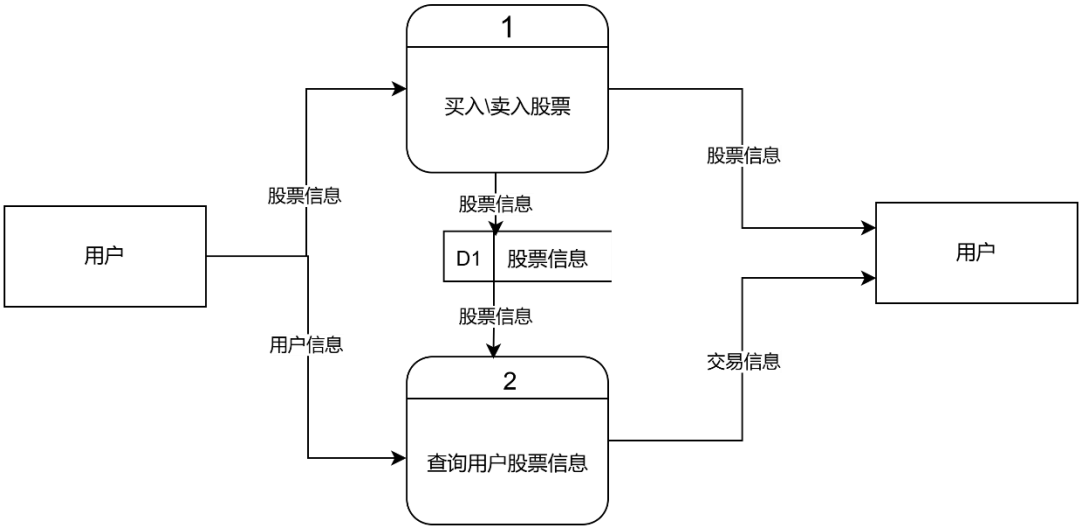


图 3. 2 第 0 层数据流图

用户通过提供用户信息与股票信息，通过系统提供的股票信息，完成买入或卖出股票，形成订单操作；查询用户股票信息操作，系统通过撮合与配对系统，向用户提供所需要的信息。

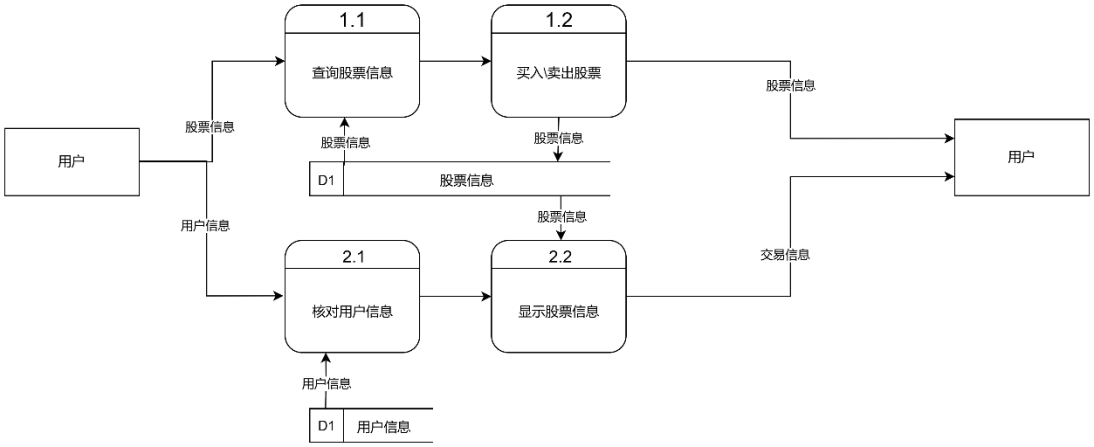


图 3. 3 第 1 层数据流图

用户通过用户信息登录用户系统，通过股票信息查询订单了解股票信息，选择买入或卖出操作，将形成的订单信息重新传回系统，系统将多个订单信息汇总撮合配对，选取合适的订单信息，改写订单信息，重新传给用户。

功能编号和优先级

| 功能编号 | 功能 | 优先级 |
|---|---|---|
| 1 | 查询订单 | 中 |
| 2 | 查询股票信息 | 中 |
| 3 | 买入股票 | 中 |
| 4 | 卖出股票 | 中 |
| 5 | 撮合订单 | 高 |
| 6 | 用户登录 | 高 |
| 7 | 用户注册 | 中 |

**表 3.1 功能编号和优先级**

b.功能描述

查询订单：选择查询订单后，选择所查询的股票类型后，会显示订单状态以及订单购买金额与数量

查询股票信息：点击添加订单时，可以查看股票类型

买入股票：选择查询订单后，选择所查询的股票类型后，选择买入操作，填写意愿购买金额与数量

卖出股票：选择查询订单后，选择所查询的股票类型后，选择卖出操作，填写意愿购买金额与数量

撮合订单：根据订单信息，将订单信息进行比对搭配，选择合适订单，修改此订单状态，完成一次撮合

用户登录：点击网址后，选择用户登录，填写正确后成功进入界面

用户注册：点击网址后，选择用户注册，注册成功后返回登陆界面

3.1.2 非功能需求

a.数据需求

静态数据：股票类型、用户订单号

动态数据：用户账号密码 用户订单数量选择 用户订单金额选择

数据字典

（1）数据流条目

用户名

| 名称 | 用户名 |
| --- | --- |
| 简述 | 用户登录\注册的账号 |
| 类型 | 字符串 |
| 长度 | 1024 |
| 来源 | 用户登录 |
| 去处 | 用户信息库 |

**表 3.2 用户名**

密码

| 名称 | 用户登录的账号对应的密码 |
| --- | --- |
| 简述 | 用户注册的密码 |

| | |
|---|---|
| 类型 | 字符串 |
| 长度 | 1024 |
| 来源 | 用户登录 |
| 去处 | 用户信息库 |

**表 3.3 密码**

股票信息

| | |
|---|---|
| 名称 | 股票信息 |
| 简述 | 系统中规定的股票信息 |
| 类型 | 字符串 |
| 长度 | 1024 |
| 来源 | 股票信息 |
| 去处 | 订单信息 |

**表 3.4 股票信息**

订单号

| | |
|---|---|
| 名称 | 订单号 |
| 简述 | 用户买卖股票时产生的订单号 |
| 类型 | 字符串 |
| 长度 | 1024 |
| 来源 | 买入卖出操作 |
| 去处 | 订单信息 |

**表 3.5 订单号**

订单金额

| 名称 | 订单金额 |
| --- | --- |
| 简述 | 用户买卖股票时填写的金额 |
| 类型 | Int |
| 长度 | 10000 |
| 来源 | 买入卖出操作 |
| 去处 | 订单信息 |

**表 3.6 订单金额**

订单数量

| 名称 | 订单数量 |
| --- | --- |
| 简述 | 用户买卖股票时填写的数量 |
| 类型 | Int |
| 长度 | 10000 |
| 来源 | 买入卖出操作 |
| 去处 | 订单信息 |

**表 3.7 订单数量**

订单状态

| 名称 | 订单状态 |
| --- | --- |
| 简述 | 用户查看订单买卖情况的状态 |
| 类型 | 布尔类型 |
| 长度 | NONE |

| | |
|---|---|
| 来源 | 撮合操作 |
| 去处 | 订单信息 |

表 3. 8 订单状态

（2）数据存储条目

用户信息

| | |
|---|---|
| 名称 | 用户信息 |
| 简述 | 描述用户信息 |
| 组成 | 用户账号、密码 |
| 组成方式 | 以用户名为关键字 |

表 3. 9 用户信息

股票信息

| | |
|---|---|
| 名称 | 股票信息 |
| 简述 | 描述股票 |
| 组成 | 股票类型 |
| 组成方式 | 以股票类型为关键字 |

表 3. 10 股票信息

订单信息

| | |
|---|---|
| 名称 | 订单信息 |
| 简述 | 描述订单信息 |
| 组成 | 订单号、股票类型、买卖属性、买卖金额、订单状态、买卖数量 |

| | |
|---|---|
| 组成方式 | 以订单号为关键字 |

表 3. 11 股票信息

b.性能需求

高效性：撮合交易系统需要能够快速处理大量的交易订单，保证交易的及时性和准确性。

稳定性：系统需要具有高可用性，能够持续稳定运行，避免因系统故障导致交易中断或数据丢失。

C.时间特性

(1) 响应时间：用户任意操作后 1 秒内系统给予反馈信息。

(2) 更新处理时间：由系统运行状态来决定。

(3) 数据的转换和传送时间：能够迅速完成。

d.运行需求

用户界面

登陆界面：显示登录账号密码填写，显示注册按钮，显示登录按钮

注册界面：显示注册账号密码填写，显示确定注册按钮

使用界面：显示添加订单，删除订单，查询订单，显示当前交易信息

添加订单：点击后选择股票类型，选择卖卖操作，填写订单数量金额

删除订单：点击后选择订单号点击删除

查询订单：点击后选择股票类型，显示订单数与相关信息

显示当前交易信息:点击后查看当前全部订单信息

e.软件接口

1.操作系统：Microsoft Windows 11、Microsoft Windows 10

2.软件设备：pycharm、vscode

f.安全性需求

数据安全：系统需要保证交易数据的安全性和完整性，防止数据被篡改或泄露。

交易安全：系统应能够防止欺诈交易和恶意操作，保护用户的合法权益。

g.扩展性要求

业务扩展：随着市场的发展和用户需求的变化，系统应能够支持新业务的快速接入和扩展。

技术升级：系统需要能够支持技术的不断升级和优化，以适应市场的快速发展。

h.易用性要求

界面友好：系统应提供简洁明了的用户界面，方便用户进行操作。

操作便捷：系统应提供便捷的操作流程，降低用户的学习成本和使用难度。

i.其他需求

验收标准:

一级撮合引擎将涵盖基本需求，侧重于模拟单用户与单品种的简单交易场景。

二级撮合引擎的探索，这一阶段将引入多用户与多品种的交易机制，使学生能够在更复杂的场景中应用所学知识。

三级撮合引擎将挑战学生的自学研究的能力，将引导学生考虑如何利用分布式存储等先进技术来优化撮合引擎的性能和效率，从而使学生探寻解决更高层次的现实需求。

j.质量属性:

可用性：用户可以使用

可靠性：在给定时间内可以大致上满足无错运行的要求

安全性：将用户订单信息保存，不可修改

# 四 总体设计

## 4.1 模块基本信息

模块基本信息表

| 名称 | 编号 | 设计者 | 所在文件 | 所在包 |
|---|---|---|---|---|
| 订单信息模块 | 1.1 | 王子俊 | .\PycharmProjects\dazuoye\new_total\order | Order |
| 订单交易模块 | 1.2 | 王子俊、李嘉康 | .\PycharmProjects\dazuoye\new_total\order | Order |
| 卖出订单模块 | 2.1 | 王子俊 | .\PycharmProjects\dazuoye\new_total\PriorityQueue | PriorityQueue |
| 买入订单模块 | 2.2 | 王子俊 | .\PycharmProjects\dazuoye\new_total\PriorityQueue | PriorityQueue |
| 交易信息模块 | 3.1 | 李嘉康 | .\PycharmProjects\dazuoye\new_total\Trade | Trade |
| 用户信息模块 | 4.1 | 李嘉康 | .\PycharmProjects\dazuo | User |

| | | | | |
|---|---|---|---|---|
| | | | ye\new_total\User | |

表 4. 1 模块基本信息表

如表所示，说明了各个模块的基本信息，包括模块名称、编号、设计者、所在文件和所在包。

## 4.2 功能概述

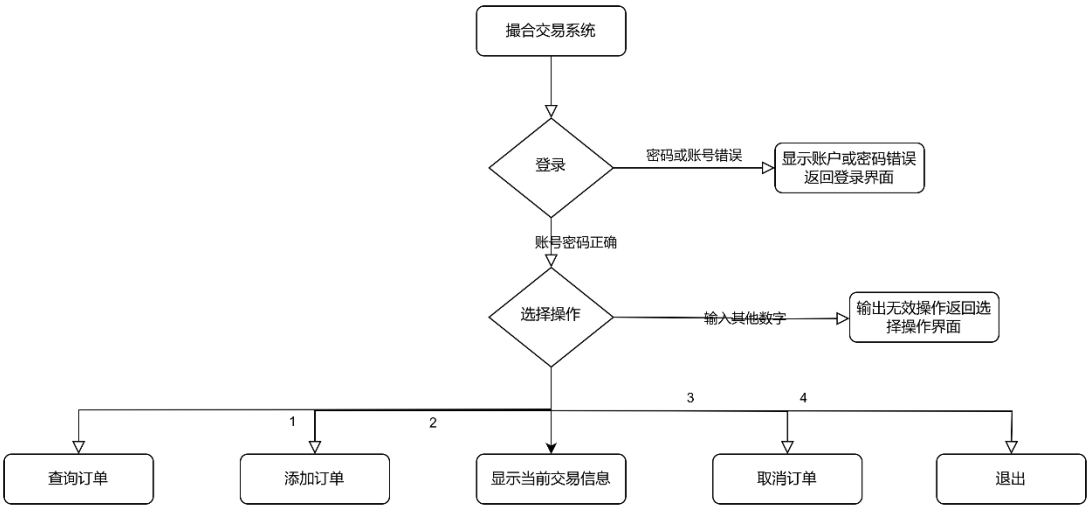


图 4. 1 总系统程序流程图

如图为总系统程序流程图，用户登录或注册系统后根据所需选择相关功能模块，完成添加订单，取消订单，查询订单，查看订单状态情况等

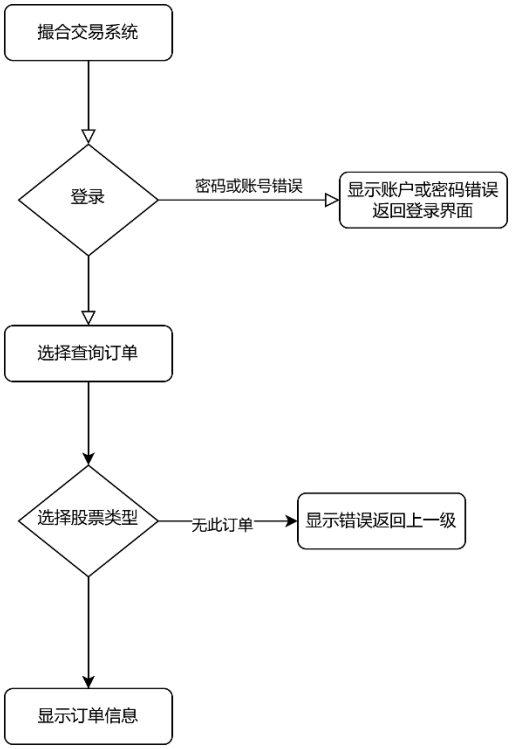


图 4. 2 查询订单程序流程图

如图为查询订单程序流程图，用户登录系统后选择查询订单模块，选择股票类型显示用户订单信息。

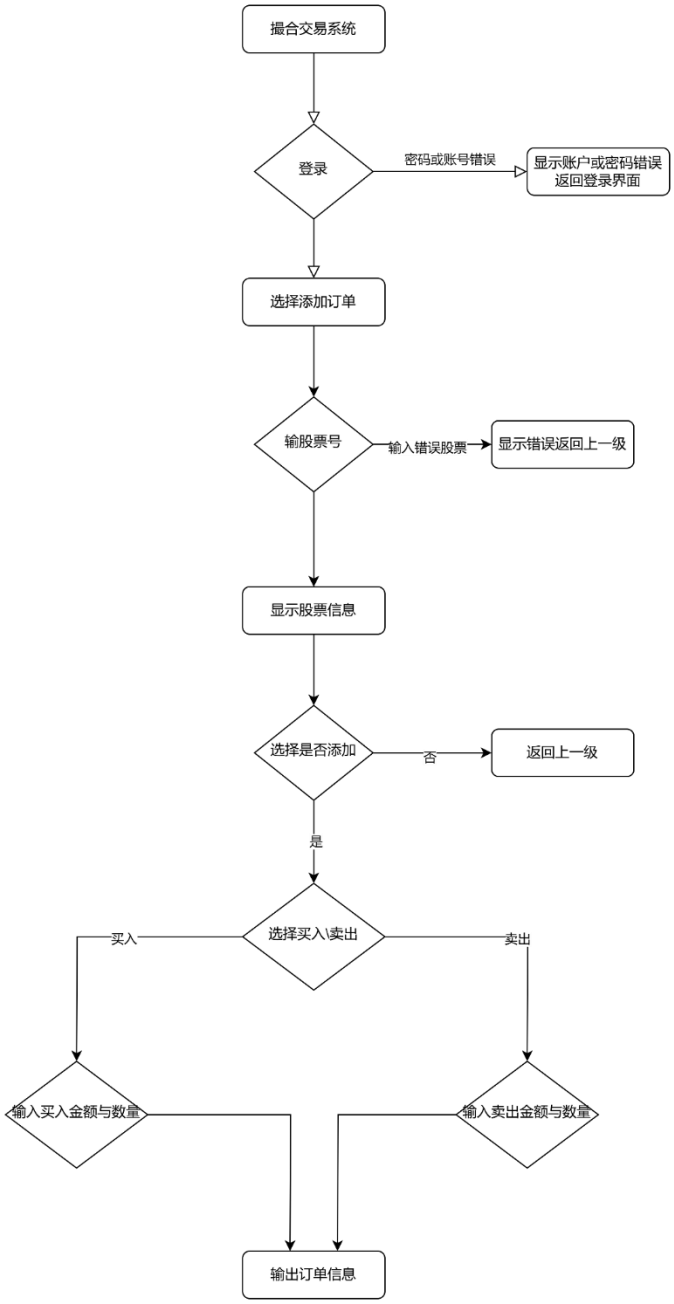


图 4. 3 添加订单程序流程图

如图为添加订单程序流程图，用户登录系统后选择添加订单模块，选择股票类型，买卖类型，输入目标金额，与目标数量，完成订单添加。

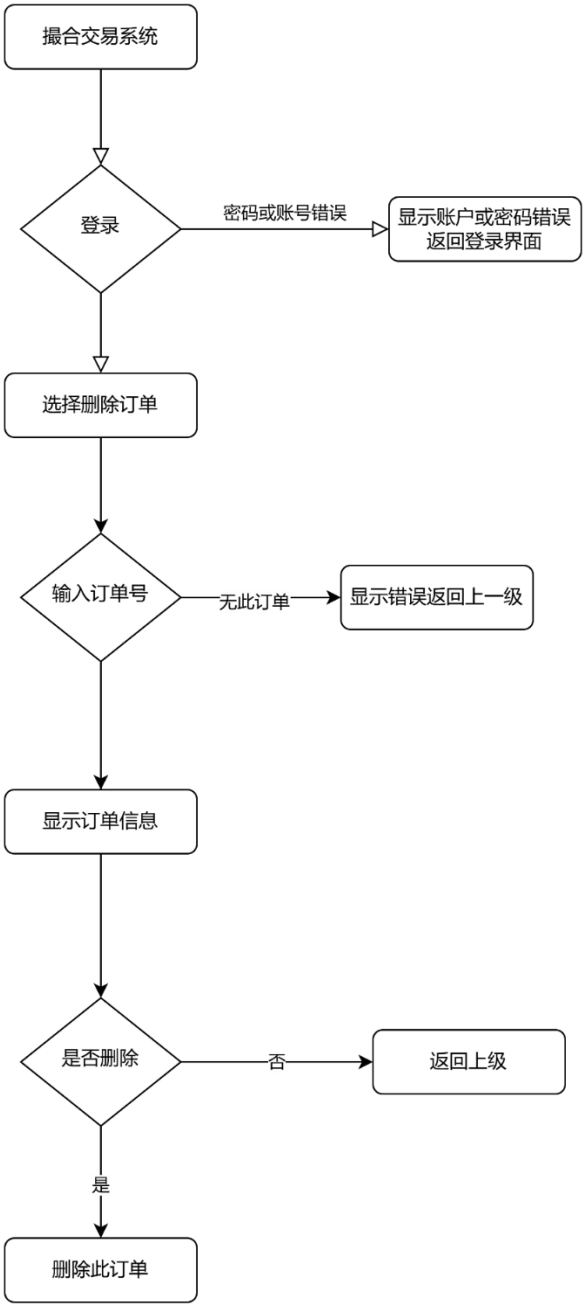


图 4. 4 删除订单程序流程图

如图为删除订单程序流程图，用户登录系统后选择删除订单模块，选择对应订单号，买卖类型，输入目标金额，与目标数量，完成订单添加。

## 4.3 算法

1.在 order 模块中，使用了最大优先队列（MaxPriorityQueue），用于存储买单，按照价格从高到低排序；最小优先队列（MiniPriorityQueue），用于存

储卖单，按照价格从低到高排序；堆排序算法：在最大优先队列和最小优先队列中实现，用于维护订单队列的顺序；匹配算法，用于在买单和卖单之间进行匹配，生成交易记录。

2.在 PriorityQueue 模块中，使用了堆排序算法（Heap Sort），在最大优先队列和最小优先队列中实现，用于维护订单队列的顺序；匹配算法，用于在买单和卖单之间进行匹配，生成交易记录。

## 4.4 模块处理逻辑

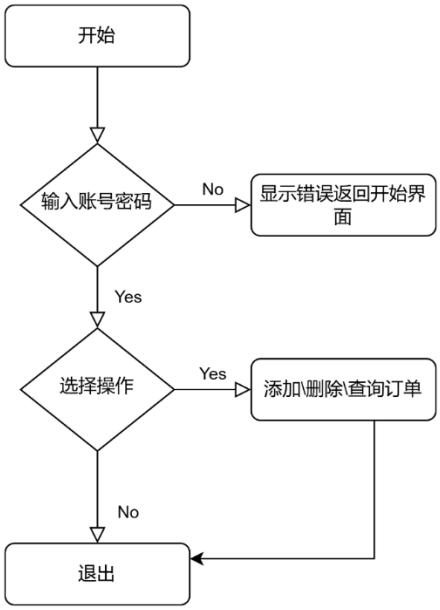


图 4. 5 一般用户系统流程图

如图为一般用户系统流程图，用户登录系统后进行选择，完成相关操作，结束后退出系统

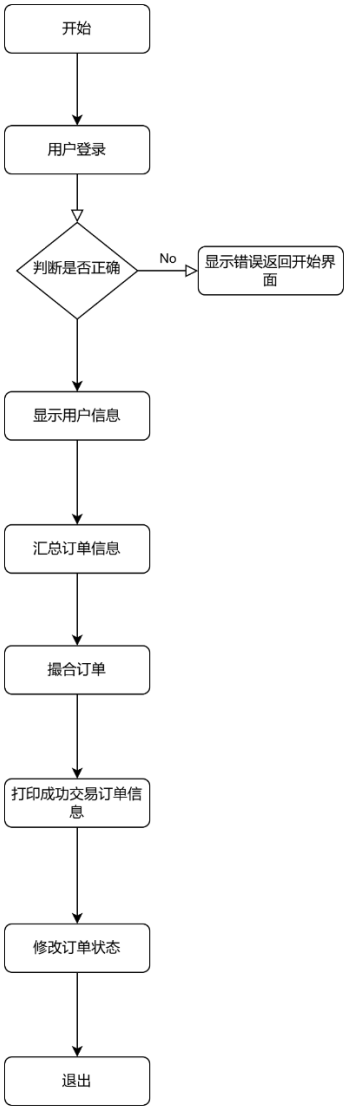


图 4. 6 整体系统流程图

如图为整体系统流程图，在用户进行注册登录后，完成添加订单操作，系统汇总所有买入卖出订单，进行撮合匹配，选择相符合的订单，完成交易，修改订单信息，完成一次撮合。

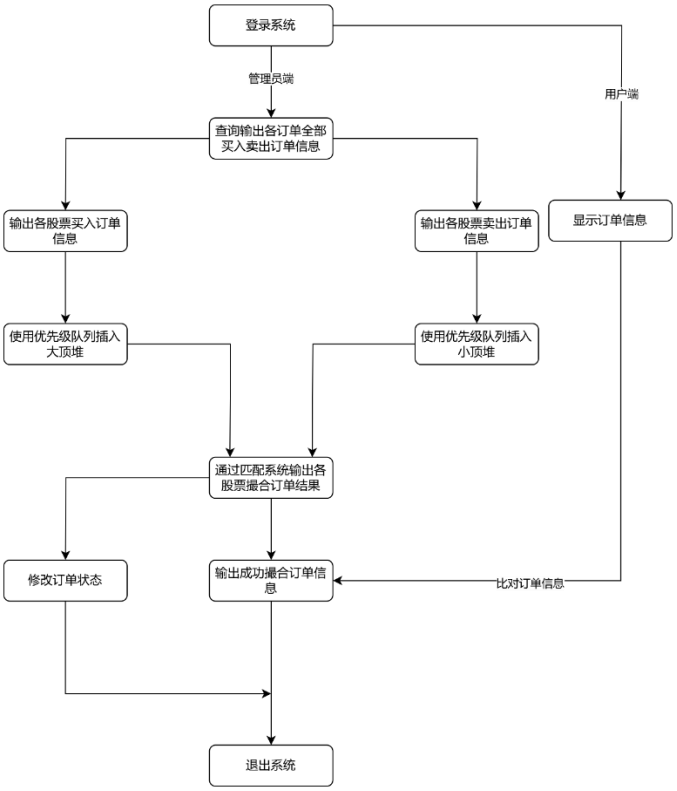


图 4. 7 撮合模块系统流程图

如图为撮合模块系统流程图，系统将买入股票订单插入大顶堆，将卖出股票订单插入小顶堆，通过匹配系统输出个股票撮合订单结果，修改成功订单信息状态，完成撮合。

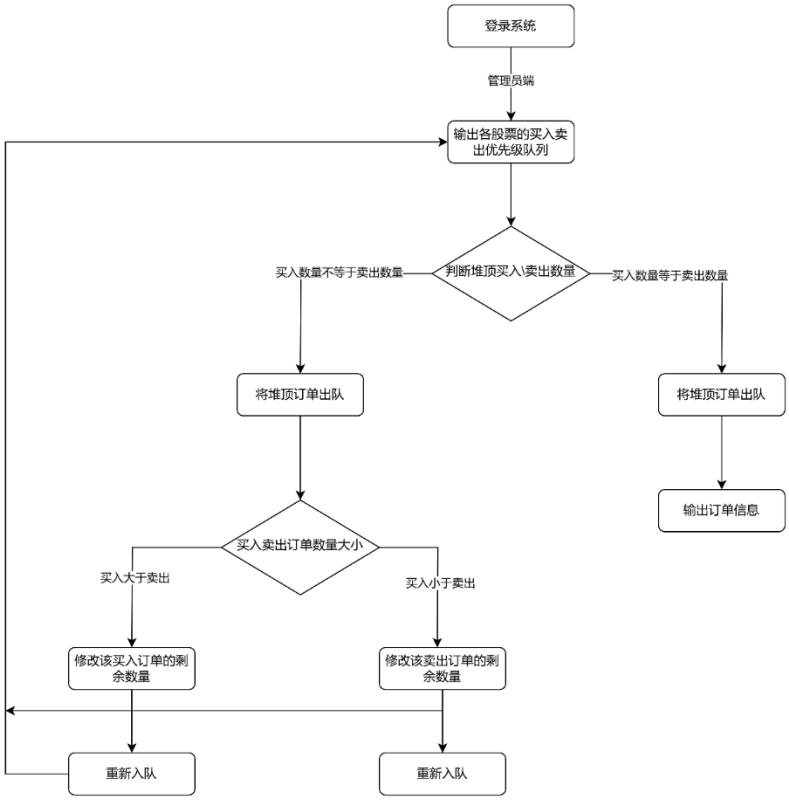


图 4. 8 匹配模块系统流程图

如图为匹配模块系统流程图，系统将买入股票订单插入大顶堆，将卖出股票订单插入小顶堆后，判断两堆顶订单数量，不相同则将订单剩余部分重新入队，重复进行匹配，知道数量相同时完成一次匹配，输出订单状态。

## 4.5 接口

用户信息

| 名称 | 数据类型 |
| --- | --- |
| 用户账号 | Char |
| 用户密码 | Char |

表 4. 2 用户信息

股票信息

| 名称 | 数据类型 |
| --- | --- |
| 股票类型 | Char |

**表 4. 3 股票信息**

订单信息

| 名称 | 数据类型 |
| --- | --- |
| 订单号 | Char |
| 订单状态 | boolean |
| 订单股票类型 | Char |
| 订单数量 | Int |
| 订单金额 | Int |

**表 4. 4 订单信息**

## 4.6 性能

4.6.1 时间特性

（1）响应时间：用户任意操作后 1 秒内系统给予反馈信息。

（2）更新处理时间：由系统运行状态来决定。

（3）数据的转换和传送时间：能迅速完成。

4.6.2 灵活性

当需求发生某些变化时，该软件的基本操作、数据结构、运行环境等等基本不会发生变化，只是对系统的文件和记录进行处理，就可以满足需求。

# 五 技术计划

## 5.1.时间安排

5.24 需求分析、分工

5.24-5.26 数据结构接口定义

5.26-5.31 基础代码设计

5.31-6.2 前端设计

6.2-6.5 代码整合、测试、文档设计

## 5.2.人员分工

王子俊:程序主体架构设计,关键方法 match 及其配套流程实现,虚拟信息生成、路演

李嘉康：优先级队列实现，Order 类，Trade 类代码实现，前端 GUI 设计，前端后端匹配

徐凡鑫：测试模块，命令行界面测试，GUI 美化

郭思源：程序文档设计，程序主体设计，相关流程图、数据流图设计

# 六详细设计

## 6.1 项目的思路逻辑

本项目以“完成大于完美”的基本思路进行设计。首先实现撮合系统的基本功能。我们定义了数据类型 `Order`，作为该程序的基本数据结构。所有的操作都是围绕 `Order` 的查看、修改、匹配和撮合进行设计的。接着我们设计了 `OrderQueue` 作为 `Order` 的存放和撮合。经过撮合后，产生 `Trade` 数据

类型以显示交易成功的信息。基于此前开发的简易版撮合系统，项目进一步从三个方向进行了升级：

1. 高效性：如何优化数据结构，以在实现相同功能的情况下更快速地完成任务。为此，我们做了大量的改进和迭代。例如，用堆结构取代了传统的优先级队列，大大降低了代码开销。

2.前后端访问便捷：为了使前端调用方便快捷，我们将所有类的调用最终封装到一个 `Progress` 类中。

3.前端和用户管理系统：设计合理的前端和用户管理系统，实现多只股票、多用户、多对多交易。

## 6.2 功能说明

1. 用户登录：提供用户登录和身份验证功能，确保交易安全和用户数据的私密性。

2.订单提交：用户可以提交买入或卖出的订单，并设置相关参数如数量和价格。

3.未完成订单撤回：允许用户撤回未被撮合的订单，以灵活应对市场变化。

4.订单自动撮合：系统根据买卖双方的订单进行自动撮合，生成交易记录并更新订单状态。

5.查看全部订单：用户可以查看所有订单的详细信息，包括已完成和未完成的订单。

## 6.3 技术点说明

1.Priority_Queue：使用堆（Heap）数据结构来实现优先级队列。对于买入订单使用最大堆（MaxHeap），对于卖出订单使用最小堆（MinHeap），以便高效地获取最优买卖订单。

2. match() 实现原理： match()方法实现了订单的自动撮合逻辑。具体步骤包括从买入和卖出队列中选取最优的订单进行匹配，生成交易记录并更新订单状态。详细实现如下：检查买卖订单队列是否为空；获取买入订单中的最高价和卖出订单中的最低价；如果买入价大于或等于卖出价，则进行撮合，生成 `Trade` 实例，更新买卖订单状态，并调整剩余数量；继续匹配直至不满足撮合条件。

## 6.4 数据结构说明

在本项目中，主要使用以下数据结构：

Order 类：表示一个交易订单，包含订单号、买卖类型、数量、价格等信息。示例代码如下：

OrderQueue 类：基于堆实现的优先级队列，用于存储和管理买入和卖出订单。示例代码如下：

Trade 类：表示一笔交易的数据结构，记录交易的买家、卖家、成交价格等信息。

Progress 类：整合了所有功能的类，提供用户接口和操作方法，包括订单提交、撤回、自动撮合和结果展示。

## 6.5 重点代码说明

Match 方法说明：

用于处理买卖订单的匹配。方法的主要逻辑如下：

1.初始化三个列表：trades（交易记录）、settle_buy_order（结算买入订单）和 settle_sell_order（结算卖出订单）。

2.当买单队列和卖单队列都不为空时，执行以下操作：

获取买单队列中价格最高的买单（buy_order）和卖单队列中价格最低的卖单（sell_order）。

如果买单的价格大于等于卖单的价格，计算实际成交价格(actual_price)，并根据买单和卖单的数量关系进行以下处理：

如果买单数量大于等于卖单数量，将买单状态设置为"部分成交"，卖单状态设置为"已成交"，并计算剩余数量（remaining_quantity）。如果剩余数量大于0，创建一个新的买单（new_buy_order），并将其状态设置为"部分成交"，然后将其添加到买单队列中。将买单和卖单分别从各自的队列中删除，并设置它们的状态、实际成交价格和剩余数量，然后将它们添加到 settle_buy_order 和 settle_sell_order 列表中。最后，将这笔交易添加到 trades 列表中。

如果买单数量小于卖单数量，将买单状态设置为"已成交"，卖单状态设置为"部分成交"，并计算剩余数量（remaining_quantity）。如果剩余数量大于 0，创建一个新的卖单（new_sell_order），并将其状态设置为"部分成交"，然后将其添加到卖单队列中。将买单和卖单分别从各自的队列中删除，并设置它们的状态、实际成交价格和剩余数量，然后将它们添加到 settle_buy_order 和 settle_sell_order 列表中。最后，将这笔交易添加到 trades 列表中。

如果买单的价格小于卖单的价格，跳出循环。

# 七 源代码

## 7.1 Main 模块

```
from order import Order, Order_queue

from random import randint, random


class Progress:

    def __init__(self):
```

```
        self.stock_0 = Order_queue()
        self.stock_1 = Order_queue()
        self.stock_2 = Order_queue()
        self.trades = []
        self.settled_buy_order = []
        self.settled_sell_order = []
        self.history = []
        self.all_order=[]
        self.all_sell_order = {}
        self.all_buy_order = {}


    def to_priority_queue(self, order):
        result = None
        if int(order.get_type()) == 0:
            self.stock_0.add_order(order)
            result = self.stock_0.match()
        elif int(order.get_type()) == 1:
            self.stock_1.add_order(order)
            result = self.stock_1.match()
        elif int(order.get_type()) == 2:
            self.stock_2.add_order(order)
            result = self.stock_2.match()
        self.history.append(order)
```

```
    return result


def to_trades(self, result):
    if 'trades' in result and result['trades']:
        self.trades.extend(result['trades'])


def to_buy_order(self, result):
    if 'settle_buy_order' in result and result['settle_buy_order']:
        self.settled_buy_order.extend(result['settle_buy_order'])


def to_sell_order(self, result):
    if 'settle_sell_order' in result and result['settle_sell_order']:
        self.settled_sell_order.extend(result['settle_sell_order'])


def show_all_order(self):


    # 检查 stock_0 的订单队列
    self.check_order_queue(self.stock_0)
    # 检查 stock_1 的订单队列
    self.check_order_queue(self.stock_1)
    # 检查 stock_2 的订单队列
    self.check_order_queue(self.stock_2)
```

```
    def check_order_queue(self, order_queue):
        for i in order_queue.buy_queue.heapArray[1:]:
            self.all_order.append(i[1])
        for i in order_queue.sell_queue.heapArray[1:]:
            self.all_order.append(i[1])




# 测试数据
def random_order(num):
    result = []
    for i in range(num):
        order = Order(
            str(i),
            randint(0, 1),
            randint(1, 99),
            round(4 * random(), 2),
            randint(0, 2),
            randint(0, 100)
        )
        result.append(order)
    return result
```

```
if __name__ == '__main__':
    pc = Progress()
    num = 10
    orders = random_order(num)  # 测试数量
    for order in orders:
        result = pc.to_priority_queue(order)
        pc.to_trades(result)
        pc.to_buy_order(result)
        pc.to_sell_order(result)

    # For better readability, printing the number of trades
    print(f"Number of trades: {len(pc.trades)}")
    for trade in pc.trades:
        print(trade)
    for i in range(len(pc.history)):
        print(pc.history[i].status_new)
```

## 7.2 order 模块

```
from PriorityQueue import MiniPriorityQueue, MaxPriorityQueue
from Trade import Trade
import time
```

```
import csv



# order_type=0/1   买/卖

# status= 0/1   未交易/已交易

# status_new='未交易','部分交易','全部交易'

# _type= 0,1,2  股票 0，1，2

class Order:

    def __init__(self, order_id, order_type, quantity, price, _type, owner_id):

        self.order_id = order_id

        self.order_type = order_type

        self.quantity = quantity

        self.status_new = '未交易'

        self.status = 0

        self.price = price

        self.type = _type

        self.time_stamp     =     time.strftime('%Y-%m-%d     %H:%M:%S', time.localtime())

        self.owner = owner_id

        self.actual_price = None



    def get_order_id(self):

        return self.order_id
```

```
def set_order_id(self, order_id):
    self.order_id = order_id
    return self

# Getter and Setter for order_type
def get_order_type(self):
    return self.order_type

def set_order_type(self, order_type):
    self.order_type = order_type
    return self

# Getter and Setter for quantity
def get_quantity(self):
    return self.quantity

def set_quantity(self, quantity):
    self.quantity = quantity
    return self

# Getter and Setter for status
def get_status(self):
```

```
    return self.status


def set_status(self, status):
    self.status = status
    return self


# Getter and Setter for price
def get_price(self):
    return self.price


def set_price(self, price):
    self.price = price
    return self


# Getter and Setter for type
def get_type(self):
    return self.type


def set_type(self, _type):
    self.type = _type
    return self


# Getter and Setter for owner
```

```
def get_owner(self):
    return self.owner


def set_actual_price(self, actual_price):
    self.actual_price = actual_price
    return self


def set_owner(self, owner):
    self.owner = owner
    return self


def __str__(self):
    return f"""
            order_id:{self.order_id},
            order_type:{self.order_type},
            quantity:{self.quantity},
            status:{self.status},
            price:{round(self.price, 2)},
            type:{self.type},
            time:{self.time_stamp},
            owner:{self.owner}
            actual:{self.actual_price}
```

"""

```
class Order_queue:
    def __init__(self):
        self.buy_queue = MaxPriorityQueue()
        self.sell_queue = MiniPriorityQueue()

    def add_order(self, order):
        if order.order_type == 0:
            self.buy_queue.add(order.price, order)
        elif order.order_type == 1:
            self.sell_queue.add(order.price, order)

    def match(self):
        trades = []
        settle_buy_order = []
        settle_sell_order = []
        while not self.buy_queue.isEmpty() and not self.sell_queue.isEmpty():
            buy_order = self.buy_queue.heapArray[1][1]
            sell_order = self.sell_queue.heapArray[1][1]
```

```
            if buy_order.get_price() >= sell_order.get_price():
                actual_price = (buy_order.get_price() +
sell_order.get_price()) / 2
                if buy_order.get_quantity() >= sell_order.get_quantity():
                    buy_order.status_new = "部分成交"
                    sell_order.status_new = "已成交"
                    remaining_quantity = buy_order.get_quantity() -
sell_order.get_quantity()
                    if remaining_quantity > 0:
                        new_buy_order = Order(buy_order.get_order_id() +
"*",

buy_order.get_order_type(),
                                              remaining_quantity,
                                              buy_order.get_price(),
                                              buy_order.get_type(),
                                              buy_order.get_owner())
                        new_buy_order.status_new = "部分成交"
                        self.add_order(new_buy_order)
                    settle_buy_order.append(

self.buy_queue.delMax().set_status(1).set_actual_price(actual_price).set
_quantity(
                                remaining_quantity))
                    settle_sell_order.append(
```

```
self.sell_queue.delMin().set_status(1).set_actual_price(actual_price).set_quantity(
                            remaining_quantity))
                    trades.append(Trade(buy_order, sell_order, actual_price, buy_order.type, remaining_quantity))
                else:
                    remaining_quantity = sell_order.get_quantity() - buy_order.get_quantity()
                    buy_order.status_new = "已成交"
                    sell_order.status_new = "部分成交"
                    if remaining_quantity > 0:
                        new_sell_order = Order(sell_order.get_order_id() + "*",
sell_order.get_order_type(),
                                               remaining_quantity,
                                               sell_order.get_price(),
                                               sell_order.get_type(),
                                               sell_order.get_owner())
                        new_sell_order.status_new = "部分成交"
                        self.add_order(new_sell_order)
                    settle_buy_order.append(
self.buy_queue.delMax().set_status(1).set_actual_price(actual_price).set_quantity(
                            remaining_quantity))
```

```
                    settle_sell_order.append(

self.sell_queue.delMin().set_status(1).set_actual_price(actual_price).set_quantity(

                            remaining_quantity))

                    trades.append(Trade(buy_order, sell_order, actual_price, buy_order.type, remaining_quantity))

            else:

                break



        return {'trades': trades, 'settle_sell_order': settle_sell_order, 'settle_buy_order': settle_buy_order}



    def cancel_sell_order(self, id):

        for item in self.sell_queue.heapArray:

            if id == item[1].order_id:

                self.sell_queue.heapArray.remove(item)

                self.sell_queue.buildHeap(self.sell_queue.heapArray[1:])

                break



    def cancel_buy_order(self, id):

        for item in self.buy_queue.heapArray[1:]:

            if id == item[1].order_id:

                self.buy_queue.heapArray.remove(item)

                self.buy_queue.buildHeap(self.buy_queue.heapArray[1:])
```

```
                break


    def check_first_order(self, order):
        if order == 0:
            return self.buy_queue.heapArray[1]
        elif order == 1:
            return self.sell_queue.heapArray[1]
```

## 7.3 PriorityQueue 模块

```
import unittest


class MiniPriorityQueue:
    def __init__(self):
        self.heapArray = [(0, 0)]
        self.currentSize = 0


    def buildHeap(self, alist):
        self.currentSize = len(alist)
        self.heapArray = [(0, 0)] + alist[:]
        i = len(alist) // 2
        while i > 0:
            self.percDown(i)
```

```
            i -= 1


    def percDown(self, i):
        while (i * 2) <= self.currentSize:
            mc = self.minChild(i)
            if mc == -1:  # 如果没有孩子，直接返回
                break
            if self.heapArray[i][0] > self.heapArray[mc][0]:
                self.heapArray[i], self.heapArray[mc] = self.heapArray[mc], self.heapArray[i]
            i = mc


    def minChild(self, i):
        if i * 2 > self.currentSize:
            return -1
        elif i * 2 + 1 > self.currentSize:
            return i * 2
        else:
            if self.heapArray[i * 2][0] < self.heapArray[i * 2 + 1][0]:
                return i * 2
            else:
                return i * 2 + 1
```

```
    def percUp(self, i):
        while i // 2 > 0:
            if self.heapArray[i][0] < self.heapArray[i // 2][0]:
                self.heapArray[i], self.heapArray[i // 2] = self.heapArray[i // 2], self.heapArray[i]
            i //= 2

    def add(self, k, order):
        self.heapArray.append((k, order))
        self.currentSize += 1
        self.percUp(self.currentSize)

    def delMin(self):
        retval = self.heapArray[1][1]
        self.heapArray[1] = self.heapArray[self.currentSize]
        self.currentSize -= 1
        self.heapArray.pop()
        self.percDown(1)
        return retval

    def isEmpty(self):
        return self.currentSize == 0
```

```
def decreaseKey(self, val, amt):
    done = False
    i = 1
    myKey = 0
    while not done and i <= self.currentSize:
        if self.heapArray[i][1] == val:
            done = True
            myKey = i
        else:
            i += 1
    if myKey > 0:
        self.heapArray[myKey] = (amt, self.heapArray[myKey][1])
        self.percUp(myKey)

def __contains__(self, vtx):
    return any(pair[1] == vtx for pair in self.heapArray)

def check_first(self):
    return self.heapArray[1][1]

def traverse(self):
    return [pair[1] for pair in self.heapArray[1:]]
```

```
class MaxPriorityQueue:
    def __init__(self):
        self.heapArray = [(0, 0)]
        self.currentSize = 0

    def buildHeap(self, alist):
        self.currentSize = len(alist)
        self.heapArray = [(0, 0)] + alist[:]
        i = len(alist) // 2
        while i > 0:
            self.percDown(i)
            i -= 1

    def percDown(self, i):
        while (i * 2) <= self.currentSize:
            mc = self.maxChild(i)
            if mc == -1:  # 如果没有孩子，直接返回
                break
            if self.heapArray[i][0] < self.heapArray[mc][0]:
                self.heapArray[i], self.heapArray[mc] = self.heapArray[mc], self.heapArray[i]
            i = mc
```

```
    def maxChild(self, i):
        if i * 2 > self.currentSize:
            return -1
        elif i * 2 + 1 > self.currentSize:
            return i * 2
        else:
            if self.heapArray[i * 2][0] > self.heapArray[i * 2 + 1][0]:
                return i * 2
            else:
                return i * 2 + 1


    def percUp(self, i):
        while i // 2 > 0:
            if self.heapArray[i][0] > self.heapArray[i // 2][0]:
                self.heapArray[i], self.heapArray[i // 2] = self.heapArray[i // 2], self.heapArray[i]
            i //= 2


    def add(self, k, order):
        self.heapArray.append((k, order))
        self.currentSize += 1
        self.percUp(self.currentSize)
```

```
def delMax(self):
    retval = self.heapArray[1][1]
    self.heapArray[1] = self.heapArray[self.currentSize]
    self.currentSize -= 1
    self.heapArray.pop()
    self.percDown(1)
    return retval


def isEmpty(self):
    return self.currentSize == 0


def decreaseKey(self, val, amt):
    done = False
    i = 1
    myKey = 0
    while not done and i <= self.currentSize:
        if self.heapArray[i][1] == val:
            done = True
            myKey = i
        else:
            i += 1
    if myKey > 0:
```

```
            self.heapArray[myKey] = (amt, self.heapArray[myKey][1])
            self.percUp(myKey)

    def __contains__(self, vtx):
        return any(pair[1] == vtx for pair in self.heapArray)

    def check_first(self):
        return self.heapArray[1][1]

    def traverse(self):
        return [pair[1] for pair in self.heapArray[1:]]

# 单元测试
class TestPriorityQueue(unittest.TestCase):
    def setUp(self):
        self.minPQ = MiniPriorityQueue()
        self.maxPQ = MaxPriorityQueue()

    def test_min_pq(self):
        self.minPQ.add(3, "task1")
        self.minPQ.add(1, "task2")
        self.minPQ.add(2, "task3")
        self.assertEqual(self.minPQ.delMin(), "task2")
```

```
        self.assertEqual(self.minPQ.delMin(), "task3")
        self.assertEqual(self.minPQ.delMin(), "task1")
        self.assertTrue(self.minPQ.isEmpty())

    def test_max_pq(self):
        self.maxPQ.add(1, "task1")
        self.maxPQ.add(3, "task2")
        self.maxPQ.add(2, "task3")
        self.assertEqual(self.maxPQ.delMax(), "task2")
        self.assertEqual(self.maxPQ.delMax(), "task3")
        self.assertEqual(self.maxPQ.delMax(), "task1")
        self.assertTrue(self.maxPQ.isEmpty())

if __name__ == "__main__":
    unittest.main()
```

## 7.4 Trade 模块

```
import time
import csv
class Trade:
    trade_counter = 0
```

```
    def __init__(self, order_buy, order_sell, price, _type, quantity):
        self.trade_id = self.generate_trade_id()
        self.trade_time = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())
        self.buyer_id = order_buy.get_order_id()
        self.seller_id = order_sell.get_order_id()
        self.price = price
        self.type = _type
        self.quantity = quantity

    def generate_trade_id(self):
        Trade.trade_counter += 1
        return f"TRADE_{Trade.trade_counter}"

    def get_trade_id(self):
        return self.trade_id

    def get_trade_time(self):
        return self.trade_time

    def get_buyer_id(self):
        return self.buyer_id
```

```
    def get_seller_id(self):
        return self.seller_id


    def __str__(self):
        return (f"Trade ID: {self.trade_id}\n"
                f"Trade Time: {self.trade_time}\n"
                f"Buyer ID: {self.buyer_id}\n"
                f"Seller ID: {self.seller_id}\n"
                f"Price: {self.price}\n"
                f"Type: {self.type}\n"
                f"Quantity: {self.quantity}")
```

## 7.5 user 模块

```
from order_queue import order_queue
class User:
    def __init__(self, user_id, name, password):
        self.user_id = user_id
        self.name = name
        self.password = password


    def order_buy_queue_show(self):
        buy_orders = order_queue.display_buy_order(self.user_id)
```

```
        print(f"用户{self.name}的买单队列：")

        for order in buy_orders:

            print(f"id:{order['order_id']}, 类 型 :{order['type']}, 价格:{order['price']},"

                  f"  数  量  :{order['quantity']},  时间:{order['time_stamp']}")


    def order_sell_queue_show(self):

        sell_orders = order_queue.display_sell_order(self.user_id)

        print(f"用户{self.name}的卖单队列：")

        for order in sell_orders:

            print(f"id:{order['order_id']}, 类 型 :{order['type']}, 价格:{order['price']},"

                  f"  数  量  :{order['quantity']},  时间:{order['time_stamp']}")
```

## 7.6 登录界面模块

```
import time

from main import Progress

from order import Order

from flask import Flask, render_template, request, redirect, url_for, flash

import pandas as pd

import csv
```

```
owner = ""

order_id = 1

app = Flask(__name__)

app.secret_key = 'your_secret_key'

progress = Progress()



@app.route('/')

def home():

    return redirect(url_for('login'))



@app.route('/login', methods=['GET', 'POST'])

def login():

    if request.method == 'POST':

        username = request.form['username']

        password = request.form['password']

        df = pd.read_csv("users.csv", encoding="gbk")

        users = []

        passwords = []

        for row in df.iterrows():

            usernameNew = row[1]['用户名']

            passwordNew = row[1]['用户密码']
```

```
            users.append(usernameNew)
            passwords.append(passwordNew)
        if username in users:
            if passwords[users.index(username)] == password:
                global owner
                owner = username
                return redirect(url_for('mainPage'))
            else:
                flash('密码错误！', 'danger')
        else:
            flash('无效用户名或密码！', 'danger')

    return render_template('login.html')


@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        df = pd.read_csv("users.csv", encoding="gbk")
        users = []
        for row in df.iterrows():
```

```
            usernameOld = row[1]['用户名']
            users.append(usernameOld)
        if username in users:
            flash('用户已经存在了！', 'danger')
        else:
            with open("users.csv", 'a+', encoding='gbk', newline="") as f:
                csv_write = csv.writer(f)
                data_row = [f"{username}", f"{password}"]
                csv_write.writerow(data_row)
            flash('注册成功，您现在可以登录了！', 'success')
            return redirect(url_for('login'))

    return render_template('register.html')


@app.route('/mainPage')
def mainPage():
    return render_template('mainPage.html')


@app.route('/page1', methods=['GET', 'POST'])
def page1():
    global owner
```

```
    global order_id
    if request.method == 'POST':
        stock_price = request.form['stock_price']
        stock_price = float(stock_price)
        print(stock_price)
        buy_sell = request.form['buy_sell']
        if buy_sell == '买':
            buy_sell = 0
        else:
            buy_sell = 1
        stock_quantity = request.form['stock_quantity']
        stock_quantity = int(stock_quantity)
        stock_type = request.form['stock_type']
        stock_type = int(stock_type)
        order = Order(str(order_id), buy_sell, stock_quantity, stock_price,
stock_type, owner)
        progress.to_priority_queue(order)
        order_id += 1
        flash('成功添加订单', 'success')

    return render_template('page1.html')
```

```
@app.route('/page2', methods=['GET', 'POST'])
def page2():
    if request.method == 'POST':
        stock_id = request.form['stock_id']
        stock_id = str(stock_id)
        found = False
        if progress.history == []:
            flash('当前无历史订单', 'danger')
            return render_template('page2.html')
        for stock in progress.history:
            if stock.order_id == stock_id:
                found = True
                if stock.type == 0:
                    if stock.order_type == 0:
                        progress.stock_0.cancel_buy_order(stock_id)
                    else:
                        progress.stock_0.cancel_sell_order(stock_id)
                    flash('成功删除订单', 'success')
                    return render_template('page2.html')
                elif stock.type == 1:
                    if stock.order_type == 0:
                        progress.stock_1.cancel_buy_order(stock_id)
                    else:
```

```
                        progress.stock_1.cancel_sell_order(stock_id)
                    flash('成功删除订单', 'success')
                    return render_template('page2.html')
                elif stock.type == 2:
                    if stock.order_type == 0:
                        progress.stock_2.cancel_buy_order(stock_id)
                    else:
                        progress.stock_2.cancel_sell_order(stock_id)
                    flash('成功删除订单', 'success')
                    return render_template('page2.html')
        if not found:
            flash('未找到订单', 'danger')

    return render_template('page2.html')



@app.route('/page3')
def page3():
    return render_template('page3.html')



@app.route('/page4')
def page4():
```

```
    return render_template('page4.html')



@app.route('/target_page1')
def target_page1():
    orders = []
    for order in progress.history:
        order_id = int(order.order_id)
        if order.order_type == 0:
            order_type = '买'
        else:
            order_type = '卖'
        dictNew = {'order_id': order_id, 'order_type': order_type,
'order_status': order.status_new,
                   'transaction_quantity': order.quantity,
                   'transaction_price': order.price}
        orders.append(dictNew)
    return render_template('target_page1.html', orders=orders)



@app.route('/target_page2')
def target_page2():
    return render_template('target_page2.html')
```

```
@app.route('/view_stock/<int:stock_id>')
def view_stock(stock_id):
    stock_orders = {0: [], 1: [], 2: []}
    progress.show_all_order()
    for stock in progress.all_order:
        if stock.type == 0:
            if stock.order_type == 0:
                order_type = '买'
            else:
                order_type = '卖'
            stock_orders[0].append(
                {'order_id': stock.order_id, 'order_type': order_type,
'order_status': stock.status_new,
                 'transaction_quantity': stock.quantity,
'transaction_price': stock.price})
        elif stock.type == 1:
            if stock.order_type == 0:
                order_type = '买'
            else:
                order_type = '卖'
            stock_orders[1].append(
```

```
                {'order_id': stock.order_id, 'order_type': order_type,
'order_status': stock.status_new,

                 'transaction_quantity':                  stock.quantity,
'transaction_price': stock.price})

        elif stock.type == 2:

            if stock.order_type == 0:

                order_type = '买'

            else:

                order_type = '卖'

            stock_orders[2].append(

                {'order_id': stock.order_id, 'order_type': order_type,
'order_status': stock.status_new,

                 'transaction_quantity':                  stock.quantity,
'transaction_price': stock.price})

    stock_info = stock_orders.get(stock_id, None)

    progress.all_order = []

    if stock_info is None:

        return "Stock not found", 404



    return     render_template('view_stock.html',     stock_id=stock_id,
orders=stock_info)




if __name__ == '__main__':

    app.run()
```

# 八 测试

## 8.1 单元测试

### 8.1.1 MaxPriorityQueue 类测试

```
from PriorityQueue import MaxPriorityQueue
import random


maxq = MaxPriorityQueue()


def create_maxP(i):
    alist = []
    for i in range(i):
        alist.append((random.randint(1, 100), f'{str(i)}'))
    maxq.buildHeap(alist)
    return maxq


if __name__ == '__main__':
    # buildHeap
    maxq = create_maxP(10)
    print(maxq.heapArray)
    # delmin
    for i in range(9):
        print(maxq.delMax())
```

```
print(maxq.check_first())

# isempty

print(maxq.isEmpty())

# add

maxq.add(5, '')

maxq.add(1, 'a')

# decreasekey

print("Before: ", maxq.heapArray)

maxq.decreaseKey('a', -1)

print("After: ", maxq.heapArray)

# contains

print('a' in maxq)

print('d' in maxq)

# check_first

print("First element:", maxq.check_first())
```

测试结果：

```
[(0, 0), (99, '4'), (86, '0'), (69, '5'), (52, '3'), (46, '9'), (57, '2'), (16, '6'), (35, '7'), (6, '8'), (20, '1')]
4
0
5
2
3
9
7
1
6
8
False
Before:  [(0, 0), (6, '8'), (5, ''), (1, 'a')]
After:  [(0, 0), (6, '8'), (5, ''), (-1, 'a')]
True
False
First element: 8

Process finished with exit code 0
```

图 8. 1　MaxPriorityQueue 测试结果

### 8.1.2 MinPriorityQueue 类测试

```
from PriorityQueue import MiniPriorityQueue
import random

minq = MiniPriorityQueue()

# buildheap

def creat_miniP(i):
    alist = []
    for i in range(i):
        alist.append((random.randint(1, 100), f'{str(i)}'))
    minq.buildHeap(alist)
    return minq

if __name__ == '__main__':
    # buildHeap
    minq = creat_miniP(10)
    print(minq.heapArray)
    # delmin
```

```
for i in range(9):
        print(minq.delMin())
print(minq.check_first())
# isempty
print(minq.isEmpty())
# add
minq.add(5, '')
minq.add(1, 'a')
# decreasekey
print("Before: ", minq.heapArray)
minq.decreaseKey('a', 0)
print("After: ", minq.heapArray)
# contains
print('a' in minq)
print('d' in minq)
# check_first
print("First element:", minq.check_first())
```

测试结果：

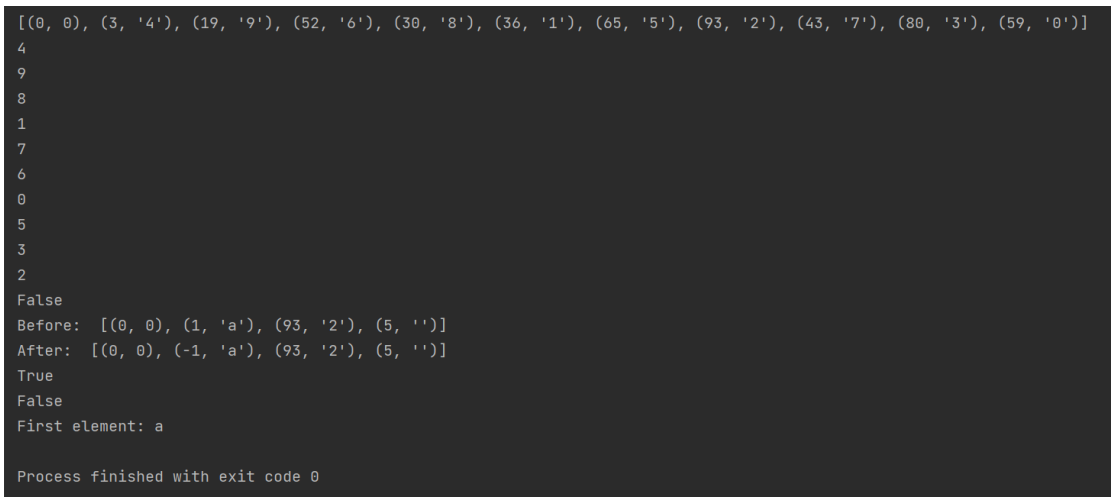


图 8. 2 MiniPriorityQueue 测试结果

### 8.1.3 后端测试代码

```
def random_order(num):
    result = []
    for i in range(num):
        order = Order(
            str(i),
            randint(0, 1),
            randint(1, 99),
            round(4 * random(), 2),
            randint(0, 2),
            randint(0, 100)
        )
        result.append(order)
    return result
```

```
if __name__ == '__main__':
    pc = Progress()
    num = 10
    orders = random_order(num)    # 测试数量
    for order in orders:
        result = pc.to_priority_queue(order)
        pc.to_trades(result)
        pc.to_buy_order(result)
        pc.to_sell_order(result)


    # For better readability, printing the number of trades
    print(f"Number of trades: {len(pc.trades)}")
    for trade in pc.trades:
        print(trade)
    for i in range(len(pc.history)):
        print(pc.history[i].status_new)
```

测试结果：

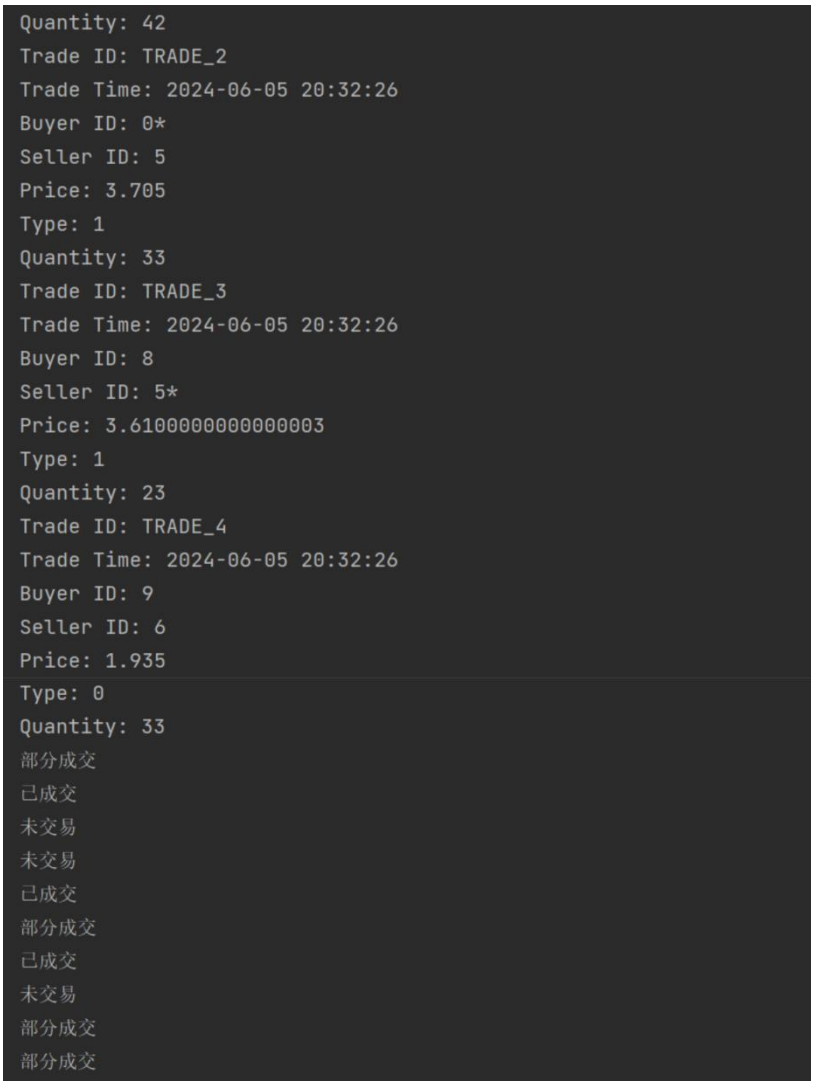


图 8. 3 后端测试结果

## 8.2 系统测试


图 8. 4 登录界面


图 8. 5 系统主界面


图 8. 6 添加订单页面


图 8. 7 删除订单界面


图 8. 8 显示所有订单界面


图 8. 9 自动撮合订单


图 8. 10 显示交易历史

## 8.3 测试总结

通过单元测试与系统测试，验证股票交易撮合系统中的单个组件（例如订单匹配引擎、订单管理模块等）是否按照预期工作，验证完整的股票交易撮合系统满足业务需求和性能要求，并确保不同组件之间的交互按预期工作，通过测试基本完成了交易撮合系统的基本功能，完成了系统的可视化，用户的注册登录，不同用户之间的交易等功能，实现了二级撮合引擎的探索，引入多用户与多品种的交易机制。

# 九 用户手册

## 9.1 系统概述

撮合交易系统，系统通过一系列算法和规则，根据价格、数量和时间等因素，自动匹配合适的交易订单，促成交易的执行，通过信息发布、自动匹配、交易协商等功能，降低交易成本、提高效率，增加透明度，方便快捷，信用评价机制保障安全。

## 9.2 使用说明

1.用户进入系统页面后点击注册按钮填写用户名与密码，后自动返回登录界面，填写对应注册好的用户名与密码进入系统

2.添加订单：可交易所需股票，进入系统后，选择添加订单按钮进入界面，选择要交易的股票类型，交易操作（买\卖），填写交易金额与数量，点击确定，完成一次订单添加。

3.删除（取消）订单：可撤回未交易订单，进入系统后，通过显示所有订单查看需要删除的订单号，返回后选择删除订单选项，输入订单号，即可删除未交易订单。

4.显示所有订单：可查看未交易订单，进入系统后，选择显示所有订单选项，选择要查看的股票类型，界面显示订单信息。

5.显示当前交易信息：可查看交易市场的订单详情，进入系统后，选择显示当前交易信息选项，可查看所有订单的交易情况