实验三：软件实现与构建实验报告

- 学号：231220051
- 姓名：丁锐
- 日期：2025年11月11日

1. 实验目的

- 熟练使用构建工具
- 学习使用大模型辅助软件开发
- 熟练使用Git管理本地及远程仓库

2. 实验内容

在本次实验中，完成了以下三个任务：

- 根据实验二中设计的UML图，将设计的软件实现为C++代码，代码量大于500行。
- 使用g++编译器编译代码并运行。
- 使用Git建立本地仓库并托管至远程仓库。

3. 软件实现

3.1 UML图到代码的实现

本次实验基于实验二中设计的UML图，将网络商场系统的核心功能模块实现为C++代码。项目采用模块化的设计思想，每个核心实体都对应独立的头文件（.h）和源文件（.cpp），确保了代码的清晰性、可维护性和可扩展性。

- **核心实体与模块**：
  根据实验二的类图设计，系统主要包含以下核心实体，并在 `code/` 目录下进行了相应的C++实现：
  - **User (用户)**: `User.h`, `User.cpp`。负责用户账户管理、身份认证、个人资料等。
  - **Category (商品类别)**: `Category.h`, `Category.cpp`。管理商品的分类信息。
  - **Listing (商品)**: `Listing.h`, `Listing.cpp`。处理商品的发布、管理、状态变更等。
  - **Conversation (会话)**: `Conversation.h`, `Conversation.cpp`。管理用户间的交流会话。
  - **Message (消息)**: `Message.h`, `Message.cpp`。处理会话中的具体消息内容。
  - **ContactExchange (联系方式交换)**: `ContactExchange.h`, `ContactExchange.cpp`。实现买卖双方联系方式的安全交换机制。
  - **Favorite (收藏)**: `Favorite.h`, `Favorite.cpp`。允许用户收藏感兴趣的商品。
  - **Report (举报)**: `Report.h`, `Report.cpp`。处理用户对商品或用户的举报。
  - **AuditLog (审计日志)**: `AuditLog.h`, `AuditLog.cpp`。记录系统中的关键操作和事件，用于追溯和审计。
  - **Utils (工具类)**: `Utils.h`, `Utils.cpp`。提供通用的辅助函数，如时间戳生成等。
  - **Types (类型定义)**: `Types.h`。定义了系统中使用的枚举类型和结构体，如用户状态、商品状态、举报状态等。

  这些模块的实现严格遵循了实验二中类图所定义的属性和方法，以及各实体之间的关系。例如，`Listing` 类包含了商品的基本信息、状态管理和图片添加功能；`ContactExchange` 类则实现了双向确认机制，确保联系方式的隐私和安全。

- **源代码规模**:
  本次实验实现的C++代码（不含空行、注释及预处理指令）总计 **690** 行，满足实验要求。
- **实现细节**:
  - **联系方式交换 (ContactExchange)**:
    实验二中的"联系方式交换"活动图和时序图详细描绘了买卖双方在会话中确认交换联系方式的流程。在代码实现中，`ContactExchange` 类封装了这一业务逻辑。
    - `ContactExchange` 类的构造函数接收 `conversation_id`，表明其与特定会话关联。
    - `ProvideBuyerContact()` 和 `ProvideSellerContact()` 方法用于记录买卖双方提供的联系信息，但在交换完成前，这些信息通过 `buyer_contact()` 和 `seller_contact()` 访问时会返回"未公开"，体现了隐私保护。
    - `ConfirmBuyer()` 和 `ConfirmSeller()` 方法分别用于记录买卖双方的确认状态。只有当 `buyer_confirmed_` 和 `seller_confirmed_` 都为 `true` 时，`IsExchangeComplete()` 方法才返回 `true`，此时联系方式才真正对外可见，并且 `exchanged_at_` 记录交换完成时间。这与活动图中双向确认的流程完全一致。
  - **商品 (Listing)**:
    `Listing` 类实现了商品的核心属性和行为。
    - 属性如 `id_`, `seller_id_`, `title_`, `description_`, `price_`, `condition_`, `category_id_`, `campus_`, `delivery_type_`, `status_`, `created_at_` 等直接对应类图中的字段。
    - `Publish()`, `OffShelf()`, `Delete()` 方法实现了商品状态的流转，例如从 `DRAFT` 或 `OFFSHELF` 到 `PUBLISHED`，或从 `PUBLISHED` 到 `OFFSHELF`。
    - `AddImage()` 方法限制了图片数量不能超过9张，与需求中的约束相符。
  - **用户 (User)**:
    `User` 类管理用户基本信息和认证逻辑。
    - 属性如 `id_`, `phone_`, `nickname_`, `campus_`, `department_`, `role_`, `status_`, `created_at_` 等与类图一致。
    - `VerifySmsCode()` 方法模拟了短信验证码的验证过程，成功后将用户状态从 `PENDING_VERIFICATION` 设置为 `ACTIVE`。
    - `SetPassword()` 方法模拟了密码设置，但在实际应用中会进行哈希处理以增强安全性。

3.2 大模型辅助软件开发

在本次软件实现过程中，大模型（如 Gemini）作为辅助工具，在提高开发效率和代码质量方面发挥了重要作用。

- **辅助工作**:
  - **代码结构与骨架生成**：针对实验二中设计的各个领域实体（如 `User`, `Listing`, `Conversation` 等），利用大模型快速生成了类的基本结构（包括成员变量、构造函数、析构函数、Getter/Setter方法等），极大地减少了手动编写样板代码的工作量。
  - **枚举类型与常量定义**：在 `Types.h` 中定义 `RoleType`, `UserStatus`, `Condition` 等枚举类型时，大模型协助梳理了所有可能的枚举值，并提供了清晰的命名建议。
  - **辅助函数实现**：对于 `Utils.cpp` 中的 `GetCurrentTimestamp()` 等辅助函数，大模型提供了标准库的实现示例，确保了代码的正确性和效率。
  - **代码风格检查与优化建议**：在编写C++代码时，大模型能够根据Google C++ Style Guide等规范，对代码进行初步的风格检查，并提出格式化、命名约定、常量使用等方面的优化建议，帮助代码保持一致性和可读性。
  - **消除代码异味与重构建议**：对于一些可能存在的重复代码或复杂逻辑，大模型能够识别出"代码异味"，并给出潜在的重构方案，例如将重复逻辑提取为独立函数，或优化条件判断结构。

    - **注释与文档生成**：大模型辅助生成了类、方法和复杂逻辑的注释，提高了代码的可理解性。
- **不符合预期情况的处理**：
  在使用大模型辅助开发时，也遇到了一些生成内容不完全符合预期的情况。
    - **领域特定逻辑的细化**：大模型生成的代码骨架通常是通用的，对于 `ContactExchange` 中双向确认的复杂逻辑，以及 `Listing` 中图片数量限制等业务规则，需要人工进行详细的补充和调整，确保其完全符合实验二的需求设计。
    - **C++语言特性与最佳实践**：虽然大模型能生成C++代码，但在某些高级特性（如智能指针的恰当使用、RAII原则的贯彻）或性能优化方面，仍需开发者结合C++最佳实践进行审阅和修改。例如，在 `main.cpp` 中，虽然使用了 `std::unique_ptr` 的头文件，但实际示例中并未广泛使用，这需要人工判断并决定是否引入更复杂的内存管理。
    - **调试与错误排查**：大模型在生成代码时可能会引入细微的逻辑错误或编译警告。此时，开发者需要手动调试，并根据编译器错误信息或运行时行为，对大模型生成的代码进行修正。

  总体而言，大模型是强大的辅助工具，但其产出仍需开发者进行批判性思考、审阅和验证，尤其是在涉及具体业务逻辑和语言高级特性时。通过迭代式的交流和明确的指令，可以更有效地利用大模型提升开发效率。

4. 代码编译与运行

- **编译环境**：
  本次实验的C++代码使用 `g++` 编译器进行编译。在Windows环境下，可以通过MinGW或WSL等方式获取 `g++`。
  编译命令如下：

```
g++ -o D:\大三\软件工程\实验三\code\main.exe \
    D:\大三\软件工程\实验三\code\main.cpp \
    D:\大三\软件工程\实验三\code\Report.cpp \
    D:\大三\软件工程\实验三\code\Message.cpp \
    D:\大三\软件工程\实验三\code\Listing.cpp \
    D:\大三\软件工程\实验三\code\Favorite.cpp \
    D:\大三\软件工程\实验三\code\Conversation.cpp \
    D:\大三\软件工程\实验三\code\ContactExchange.cpp \
    D:\大三\软件工程\实验三\code\Category.cpp \
    D:\大三\软件工程\实验三\code\AuditLog.cpp \
    D:\大三\软件工程\实验三\code\Utils.cpp
```

  该命令将所有 `.cpp` 源文件编译并链接为一个名为 `main.exe` 的可执行文件。

- **运行结果**：
  编译成功后，运行 `main.exe` 可执行文件。由于控制台编码问题，中文字符可能显示为乱码，但这不影响程序逻辑的正确性。程序将按照 `main.cpp` 中定义的顺序，演示网络商城系统中各个核心类的创建、属性设置、方法调用以及对象间的交互。

  以下是运行输出的示例（已对乱码部分进行人工解读和修正）：

```
--- 演示网络商城系统类 ---

正在创建用户...
用户 Alice 已创建。
用户 Bob 已创建。
用户 Admin 已创建。

演示用户方法
正在为用户 Alice 设置密码。
密码已设置。
```

```
正在验证用户 Alice 的短信验证码: 123456
短信验证码已验证。用户状态设置为 ACTIVE。
用户 1 状态: 活跃

正在创建类别...
类别 '电子产品' 已创建。
类别 '手机' 已创建。
类别 '书籍' 已创建。

正在创建商品...
商品 'iPhone 13 Pro' 已创建为草稿。
添加图片: http://example.com/iphone1.jpg
图片已添加。总图片数: 1
添加图片: http://example.com/iphone2.jpg
图片已添加。总图片数: 2
发布商品 'iPhone 13 Pro'。
商品现在是 PUBLISHED 状态。
商品 1 状态: 已发布

商品 'C++ 编程书籍' 已创建为草稿。
发布商品 'C++ 编程书籍'。
商品现在是 PUBLISHED 状态。

用户 2 正在收藏商品 1...
用户 uuid-2 对商品 uuid-7 的收藏已创建。

正在为商品 1 创建用户 2 (买家) 和用户 1 (卖家) 之间的会话...
会话 uuid-9 已为买家 uuid-2 和卖家 uuid-1 (商品 uuid-7) 创建。

正在交换消息...
消息 uuid-10 已在会话 uuid-9 中由发送者 uuid-2 创建。
消息 uuid-11 已在会话 uuid-9 中由发送者 uuid-1 创建。

正在启动联系方式交换...
联系方式交换已为会话 uuid-9 启动。
买家联系方式已提供。
卖家联系方式已提供。
买家联系方式 (确认前): 未公开
卖家联系方式 (确认前): 未公开
买家已确认联系方式交换。
卖家已确认联系方式交换。
联系方式交换已于 2025-11-11 21:34:59 完成。
联系方式交换完成: 是
买家联系方式 (确认后): user2_wechat_id
卖家联系方式 (确认后): user1_phone_number

正在举报商品 2...
举报 uuid-12 已由 uuid-1 为 商品 uuid-8 创建。
举报 1 状态: 审核中

正在记录审计事件...
审计日志 uuid-13: 操作者 uuid-3 对 商品 uuid-8 执行了 更新商品状态 操作。

演示商品下架
将商品 'C++ 编程书籍' 下架。
商品现在是 OFFSHELF 状态。
商品 2 状态: 已下架
```

```
--- 演示完成 ---
```

5. Git远程代码管理

本次实验使用Git进行版本控制，并将代码托管至远程仓库。

- **初始化本地仓库**：
  在项目根目录 `D:\大三\软件工程\实验三\` 下执行以下命令初始化Git仓库：

  ```
  git init
  ```

- **添加文件到暂存区**：
  将 `code/` 和 `UML/` 目录下的所有相关文件添加到Git的暂存区：

  ```
  git add code/
  git add UML/
  git add 软工2025实验三.md
  # ... 其他需要添加的文件，例如实验报告本身
  ```

- **提交到本地仓库**：
  提交暂存区的文件到本地仓库，并附上提交信息：

  ```
  git commit -m "feat: 完成实验三代码实现与UML图"
  ```

- **关联远程仓库**：
  假设远程仓库地址为 `[YOUR_REMOTE_REPOSITORY_URL]`（例如 GitHub 或 Gitee），执行以下命令关联远程仓库：

  ```
  git remote add origin [YOUR_REMOTE_REPOSITORY_URL]
  ```

  如果已经存在 `origin`，可以使用 `git remote set-url origin [YOUR_REMOTE_REPOSITORY_URL]` 更新。

- **推送到远程仓库**：
  将本地代码推送到远程仓库的 `main` 分支（或 `master` 分支）：

  ```
  git push -u origin main
  ```

  通过以上步骤，实验三的代码和相关文档被有效地管理在Git版本控制之下，并同步到远程仓库，便于协作和备份。

6. 总结与展望

- **总结**：
  本次实验成功地将实验二中设计的网络商场系统UML图转化为C++代码实现，并达到了500行代码量的要求。通过模块化的设计和实现，验证了面向对象设计原则在实际项目中的应用。大模型在代码骨架生成、枚举定义、辅助函数实现以及代码风格检查等方面提供了有效的辅助，显著提升了开发效率。同时，熟练掌握了Git进行版本控制和远程仓库管理。

- **展望**：
  当前实现是一个控制台演示程序，未来可以考虑：

  1. **集成数据库**：将当前内存中的对象操作替换为与数据库（如MySQL、SQLite）的交互，实现数据的持久化存储。

2. **构建图形用户界面 (GUI)**：使用Qt、MFC或Web框架（如Wt）为系统添加用户友好的图形界面，提升用户体验。
3. **完善业务逻辑**：增加更多的业务功能，如搜索、筛选、用户认证（短信验证码、邮箱验证）、后台管理功能等。
4. **引入测试框架**：编写单元测试和集成测试，确保代码质量和功能稳定性。
5. **优化大模型使用**：进一步探索更高效的Prompt Engineering技巧，让大模型在复杂逻辑实现、错误排查和性能优化方面提供更深层次的辅助。

7. 参考文献

- Google C++ Style Guide. (n.d.). Retrieved from https://google.github.io/styleguide/cppguide.html
- [在此处列出其他参考资料，例如关于Prompt Engineering的论文或文章]

8. AIGC检测报告

请在此处附上AIGC检测报告的PDF文件。