**《软件工程》**

**实验设计报告**

**项目题目： 面向对象的商业综合体管理系统**

**Shopping Mall Information System**

**姓 名： 王子贤 （组长）学 号： 1191170420**

**同组姓名： 闫逸飞 张泽龙 冯香睿**

**班 级： 数字媒体技术1704**

**评 语：**

签名：

**成 绩：**

**江南大学数字媒体学院**

**2019年4月**

**目 录**

[第一章 概述 1](#_Toc6235842)

[1.1 项目背景 1](#_Toc6235843)

[1.2 系统简介 1](#_Toc6235844)

[1.3 开发环境 1](#_Toc6235845)

[1.4 主要技术 1](#_Toc6235846)

[第二章 系统设计 2](#_Toc6235847)

[2.1 UI界面设计 2](#_Toc6235848)

[2.2 软件设计 2](#_Toc6235849)

[第三章 软件实现 3](#_Toc6235850)

[3.1 功能描述 3](#_Toc6235851)

[3.2 系统运行界面 3](#_Toc6235852)

[3.3 主要设计及代码 3](#_Toc6235853)

[第四章 结束语 4](#_Toc6235854)

[4.1 总结 4](#_Toc6235855)

[4.2 不足之处 4](#_Toc6235856)

[参考文献 5](#_Toc6235857)

# 第一章 概述

## 1.1 项目背景

1.1.1 需求分析：

某城市商业综合体，现有数栋大楼，共有500个商业铺位，对外招租，进行商业经营。该单位有招商中心、市场管理部、办公室、财务部、物业部等部门构成。现需开发一套软件系统，用于管理综合体的招商信息、客户信息、合同信息、租金信息、物业费以及水电费等。

系统的用户分为场长、市场管理员、业务接待员、办公室、财务总监、物业业主、总经理，其操作软件的权限不同。

商铺业主可以向招商中心申请进场，并与市场管理部进行租赁洽谈；确认租赁的业主将与办公室签署合同，并向财务部缴纳租金或保证金；进场后，业主需向物业部定期缴纳物业费、水电费。业主还可以申请转租业务，确认转租的业主将与办公室签署铺位转租合同，签署完毕后更新铺位信息。

为方便管理商业综合体，系统应具有高效性，可方便地产生、修改、删除各种信息，包括客户、招商、合同、物业费、租金或保证金、水电费等。相关人员应能方便快速地查看所有商业铺位的关联信息，包括缴费状态、合同状态、对应客户等。系统同时应具有非常好的图形用户界面，并且在程序架构上应具有很好的可扩展性。

1.1.2 企业级信息化系统软件的发展现状：

信息系统作为企业的业务支撑, 随着企业信息化的深入, 其作用和重要性日益显现, 大规模信息系统软件开发越来越普遍[1]。

在信息化社会的发展过程中，越来越多的企业需要一个可靠性高的、分布式的、实时的、拥有嵌入式特性的软件系统，来管理越来越庞大的企业信息库，而软件的发展使海量的数据存储和复杂的管理操作得以实现[2]。

利用企业级信息化系统软件，国内诸多企业，例如服装企业逐步走向信息化，改善企业生产供应链，建立了信息化平台。信息化系统的建立使得企业管理更加高效快捷、提高了企业竞争力和生产力[3][4]。

在网络信息技术的不断发展中，软件工程作为一种基础型工具对于企业信息化平台建设不可或缺，软件工程技术应用的成功与否直接决定着企业信息化平台建设的成果[5]。遵循软件工程方法开发计算机软件，能够提升计算机软件开发效率、提升软件产品的体验效果、推动计算机软件的智能化发展[6]。因此，合理的规划软件系统、在企业及信息化系统软件的实现过程中**严谨遵循软件工程技术的开发模式**在保证企业利益和生命力方面显得尤为重要。

信息系统和软件工程技术能够对企业发展提供巨大的帮助，但在信息和软件工程产业发展的过程中，出现了一系列实际应用方面的问题：

在企业软件项目技术管理过程中，出现了软件需求定义模糊、项目参与者沟通强度不足、定制带来的软件自身内在复杂度高、文档资料丢失致使无法有效继承相关技术的问题[7]。

在系统设计和实际开发的企业管理方面，中小企业管理基础薄弱，缺乏系统的管理理念和行之有效的管理模式。不牢靠的管理基础阻碍了管理信息化的实施，影响着最终信息化系统的质量[8]。

除以上所述外，我国企业管理信息系统常存在组织结构混乱的问题：管理人员的职能和管理范围不明确；企业的信息化程度较低；相关管理人员能力不足等问题仍然存在[9]。

1.1.3 现代高度信息化的软件系统新需求：

基于上述考虑，新时代高度信息化软件系统应在原有软件系统和信息技术要求的基础上，满足更加符合现代需求的、保证企业生命力的新的系统需求：

1）程序设计方法：

顺应现代软件工程的发展趋势，新型高度信息化软件系统应以**面向对象**的编程方法为主流，实现多元化的软件产业管理模式[10]。

2）系统设计思路：

系统应在智能化和自动化的基础上应用**软件构件技术和软件重构技术**，并以**模块化**思维构建系统[11][12]。

3）团队合作与项目控制：

在系统设计和实际开发的过程中，对于开发者，应强化**项目需求控制**，降低由于低效处理需求而导致的开发和运维压力的提升。加强对需求的理解和与需求方的交流，以减少由于对需求不够了解而造成的企业开发流程混乱。在技术和管理层次**提升沟通效率**，确保软件和业务人员有效的建立彼此相互了解的关系，高效达成共识[7]。

4）用户界面设计：

在软件系统制作的基础上，**用户界面（UI）仍应受到重视**，研究表明用户界面对于用户的适应性极为重要地影响着企业级信息系统实现的成功与否[13]。

在实际的用户界面设计过程中，应考虑以用户为中心的产品界面设计流程和方法，即通过用户分析界定出产品的使用概念；通过模型设计汲取设计灵感；通过使用性测试对设计方案进行评估，最终达到降低制作成本、优化产品功能的效果[14]。同时，界面的设计也可从信息过载、控制熟悉度和用户界面匹配方面对交互界面方案进行评估[13]，来保证最终界面设计的用户适用性。

## 1.2 系统简介

本系统实现了基于数据的商场信息管理系统，它包括后台的一套商场信息管理程序和一个可视化的客户端操作界面，采用模块化设计。

1.2.1 系统概要设计

本系统的客户端分为四个主要部分，分别为基于PyQt的界面控制系统、控制类组成的逻辑控制系统、实体类组成的数据存取系统以及数据库系统。

**1）界面控制系统**

在系统工作时，与用户直接相关的是最外层的界面控制系统，用户通过界面操作实现一系列软件功能。

界面控制系统基于PyQt5设计，包含一系列界面模块，负责软件系统不同界面的显示、更新等功能，是应用程序中处理数据显示的部分。

**2）逻辑控制系统**

在软件后台，逻辑控制系统与界面直接关联，用户对界面的一系列输入，由逻辑控制系统进行处理。

逻辑控制系统负责从界面控制系统中读取数据进行处理，控制用户输入，并向数据存取系统发送数据。它由一系列控制类模块组成，是应用程序中处理用户交互并做出响应的部分，它链接系统界面和系统的数据存取系统，是软件系统中较为关键的一层。

**3）数据存取系统**

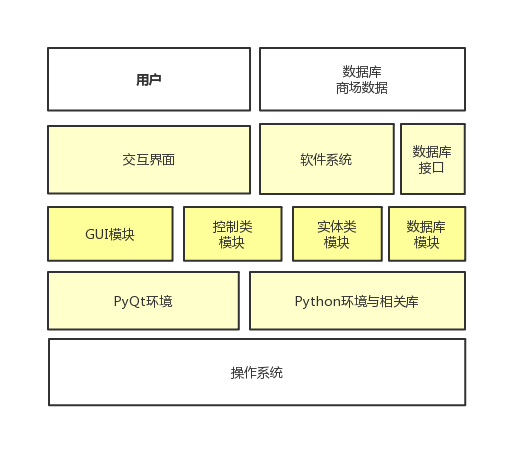
数据存取系统负责从数据库中存取数据并对应用程序数据逻辑进行处理，它直接与数据库相关联，是逻辑控制系统对数据库数据进行一系列操作的接口层。

数据存储系统由一系列实体类组成，根据软件系统所要实现的功能封装了软件内对数据的各种处理操作。

**4）数据库系统**

数据库系统负责商场信息的存储，同时封装了对数据的增删查改操作，方便修改和使用。

1.2.2 总体结构和模块接口设计



图

如图，在Python和PyQt5的基础上，我们采用了一系列模块，这些模块分别组成了逻辑控制系统和数据存取系统，实现了对数据的存取操作和根据数据进行内部逻辑处理并更新界面的操作。

## 1.3 开发环境

1.3.1 基于PyCharm的Python开发环境（Python 3.6版本）

本系统采用了PyCharm进行系统逻辑设计，PyCharm是一个Python集成开发环境，它带有一整套可以帮助用户在使用Python语言开发时提高效率的工具，为开发团队提供了项目管理、项目单元测试、项目版本测试等友好的功能。

在使用过程中，PyCharm方便了项目整体的管理，在界面中，代码结构一目了然，项目控制台的直观助力了代码调试。同时，PyCharm还集成了Git管理，为小组人员的团队协作提升效率。

利用Python这门“优雅、明确、简单”的高级语言，结合PyCharm这款专门为Python开发者设计的工具，本项目的开发效率大大提高，最终得到了满意的结果。

1.3.2 PyQt5界面开发环境

本系统的界面设计方面采用了PyQt5开发环境。

Qt作为一个跨平台C++图形用户界面应用程序开发框架，具有如下几个优点：

1. 优良的跨平台特性，支持多种操作系统。
2. 面向对象 ，Qt良好的封装机制使得Qt的模块化程度非常高，可重用性较好，对于用户开发十分方便，并且提供了signals/slots的安全类型来替代callback，使得各个元件之间的协同工作变得十分简单。
3. 丰富的API，Qt包括多达250个以上的C++类，还提供基于模板的collections、serialization、file、I/O device和date/time类，甚至还包括正则表达式的处理功能。
4. 支持2D/3D图形渲染，支持OpenGL，拥有大量开发文档等。

而PyQt是一个创建GUI应用程序的工具包。它是Python编程语言和Qt库的成功融合，为本项目开发提供了较高的开发效率。

## 1.4 主要技术

1.4.1 基于Python的软件核心实现

本项目组的项目核心语言为Python，利用面向对象的编程技术从用例及需求中抽象出相对应的类，并建立各种类之间的联系，最终通过各个类之间的交互完成整个系统的运作。

1.4.2 基于SQLite与Python的数据库系统实现

在数据库系统的构建方面，本系统运用了SQLite数据库作为底层数据库，负责数据存储和数据基本操作；同时利用Python中提供的Sqlite3库实现Python项目与本地数据库的链接与交互。

1.4.3 基于PtQt5的界面实现

在界面设计方面，系统利用PyQt5作为界面系统的技术实现，在PyQt5提供的一系列类的基础上实现界面类之间的交互以及界面类和逻辑类之间的交互。

1.4.4 MVC框架模式实现

本项目利用基于MVC框架模式构建起整个系统，在系统中分出模型（MODEL）、视图（VIEW）、控制（CONTROLLER）三层架构，分离了系统的业务逻辑、数据以及界面显示。

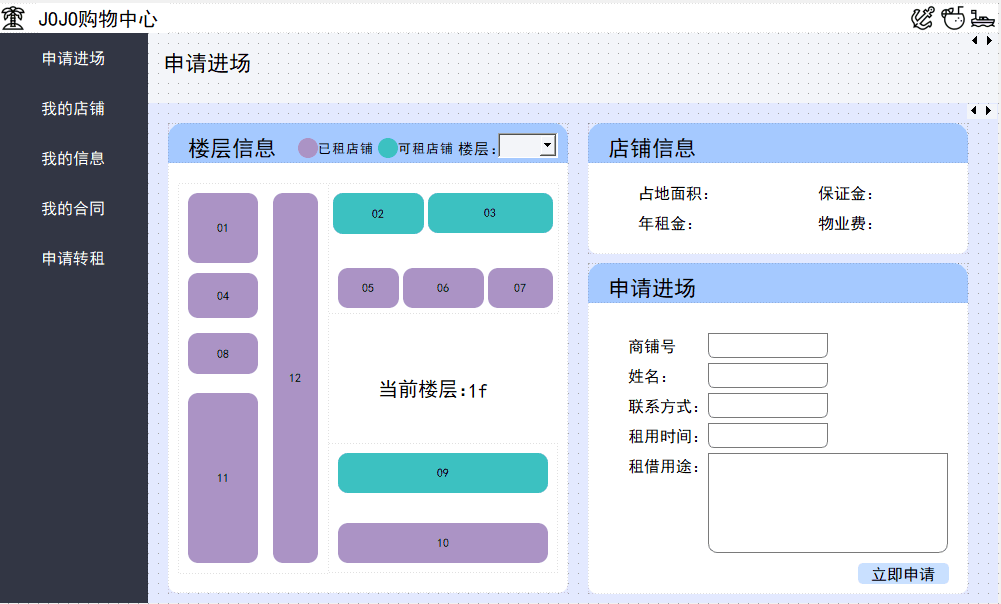
MVC分层有助于管理复杂的应用程序，在一个时间内可以只专注一个方面，同时也让应用程序的测试更加容易。MVC分层同时也简化了分组开发，不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

在实际的开发过程中，各分组人员在极大程度上关注于自己所负责的部分，而不用对整个系统有具体的了解，提高了开发效率、减小了开发阻碍。

# 第二章 系统设计

## 2.1 UI界面设计

2.1.1 整体呈现



本软件系统的整体界面设计如上。

本系统的界面设计有如下两个优点：

1）整体风格现代化，亲和用户设计

本软件采用扁平化圆角式设计，符合现代化设计的简约理念，同时左右侧的撞色引导用户视线对左右侧进行区分和分层，用户很容易理解当前页面的信息，并且通过观察就能知道如何对界面进行操作。

2）重要信息展示清晰

在具体信息栏（右侧），重要的信息分别展示出来，其中必要的信息被合理的展示，除必要信息外，没有过多冗余的文字和设计，整体风格清新自然。

2.1.2 界面风格

1）配色方案

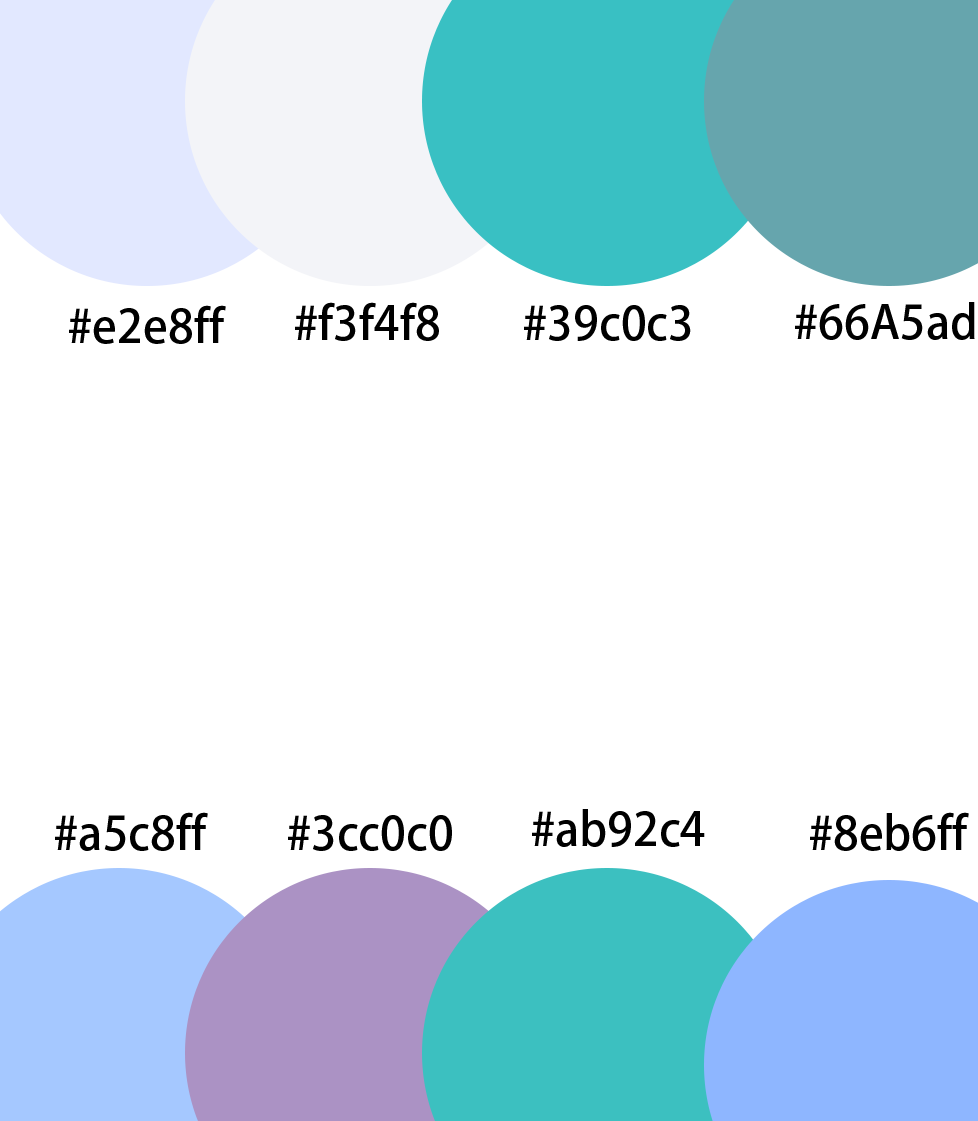
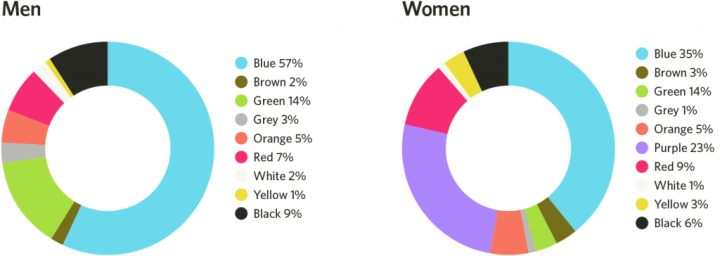


图2.2

本系统的配色方案如上图。主色调：淡蓝色。

**人们就是喜欢蓝色。**问卷调查显示大部分人把蓝色当做它们最喜欢的颜色。因此蓝色被认为是全球最安全的颜色。

色彩偏好是视觉体验中的重要一环：



主要考虑到我们的主要用户是商铺业主，我们在颜色方面选择蓝色使用户在使用中更加舒服。色彩在一定程度上作为一个软件的重要组成元素，主要目的就是使用户在感官体验上获得更好的感受，所以结合主要功能，方便用户进行操作，同时蓝色也代表着大自然的颜色，与清空大海等自然景观产生强烈的关联，提高了人们对其喜好程度，最后，蓝色可以体现科技感，对于一个实体公司来说，软件的颜色选择蓝色不仅彰显着其蓬勃的生命力还可以从中看出来它正积极顺应时代潮流，与时俱进，增加公司在用户眼中的可信度。

2）UI按钮风格

本系统的UI按钮风格设计如上图。整体风格趋于圆角、扁平化，具有现代设计感。

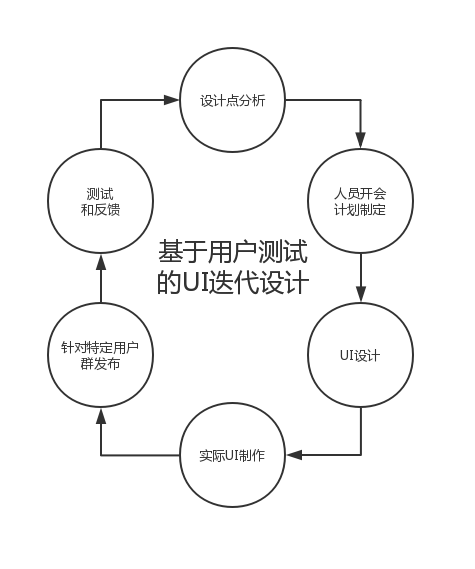
3）文字风格

本系统采用黑体，具有商务风格，作为软件字体较为稳健。

2.1.3 基于用户测试的UI迭代设计流程

本软件系统的UI设计在整个过程中遵循基于用户测试的UI迭代设计流程，采用快速确定需求、确认成员分工及目标、成果综合整理、收集用户反馈、整理并确定下一阶段界面设计需求的工作流，在保证了界面设计符合用户审美的基础上，对用户使用界面的整体体验和整体软件功能的展现程度进行迭代式提升。

下图对本小组进行界面设计的工作流进行了直观阐述。

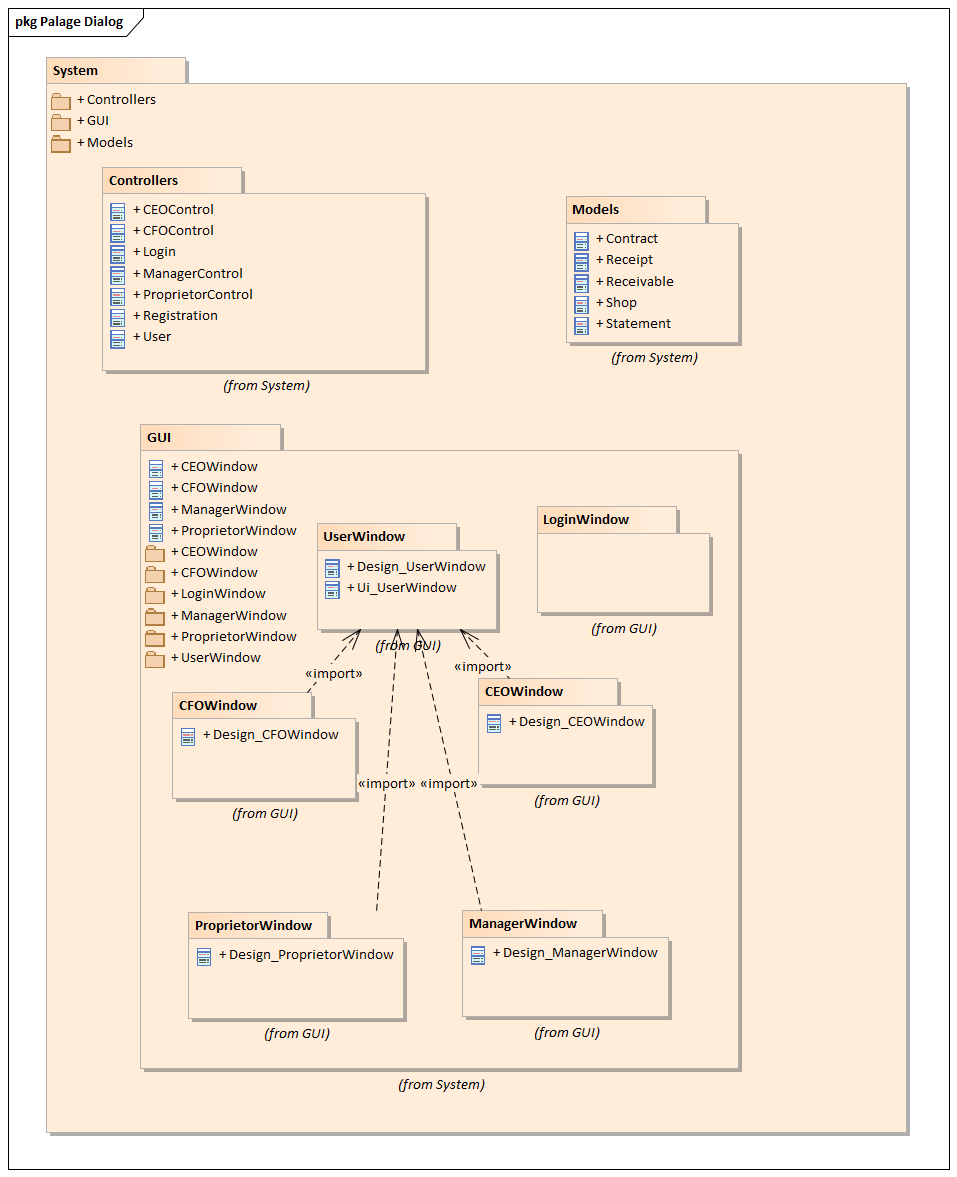


## 2.2 软件设计

2.2.1 结构设计

结构设计定义软件系统各主要部件之间的关系。

本系统的设计分为三大部分：GUI、Controllers、Models，分别对应边界类、控制类、实体类所对应的集合，便于管理与控制。如下图为系统的包图设计。



1. **GUI**

GUI包中包括四个类与五个子包，便于多人合作共同开发，在UserWindow包中的Ui\_UserWindow中位界面的布局实现，Design开头的类实现了界面逻辑，在GUI包中的类为系统逻辑实现边界类。

1. **Controllers**

Controllers包中包含一系列控制类，它们负责链接界面和数据层，并对界面的交互进行响应、进行后台数据处理。

1. **Models**

Models包中包含一系列实体类，它们直接连接数据库，负责从数据库中取回数据和向数据库中写入数据，同时也负责对数据库中数据进行定义和封装。

2.2.2 数据设计

数据设计是将模型转换成数据结构的定义。

在本系统中，通过实体类来把数据进行综合和定义。具体的实体类有合同信息（Contract）、收据（Receipt）、应收额（Receivable）、商铺信息（Shop）、报表（Statement）。

1. **合同信息**

合同信息描述与业主合同相关的一系列信息。包含合同的内容信息（information）、状态（status）、期限（year）。在数据库中，还包含合同签字情况、合同id以及签署人员id。

1. **收据**

收据描述业主当前的缴费情况。包含业主的电费缴纳额（electricCharge）、保证金缴纳额（guaranteeCharge）、物业费（propertyCharge）、税费缴纳额（waterCharge）。在数据库中，还包含缴纳人的用户id。

1. **应收额**

应收额描述业主应当的缴费情况。包含业主的电费缴纳额（electricCharge）、保证金缴纳额（guaranteeCharge）、物业费（propertyFee）、税费缴纳额（waterCharge）。在数据库中，还包含缴纳人的用户id。

1. **商铺信息**

商铺信息由以上三项（合同、收据、应收额）组合而成，是业主直接可以接触到的实体类，在数据库中，还包含商铺id、商铺拥有者id。

1. **报表**

报表是市场管理员、财务总监与总经理在查看商场状况时直接接触的实体类，包含所有业主的商铺信息。

2.2.3 接口设计

接口设计定义了软件内部，软件和操作系统间以及软件和人之间如何通信。

在本系统中，实体类所提供的外部接口以及控制类所定义的一系列函数构成了本系统的接口设计。具体如下：

1. **合同信息类的接口**

在系统中，对合同的操作有确认合同（affirm）、获取信息（get\_information）、签字（sign）。

1. **收据类的接口**

在系统中，对收据的操作有录入费用（entry）、设置水费（water）、设置电费（electric）、设置物业费（propertyfee）和保证金（guarantee）。

1. **应收额类的接口**

在系统中，对应收额的操作是对各项费用的设置，与收据类相同。

1. **商铺信息类的接口**

在系统中，对于商铺信息类的操作有获取合同信息、获取收据信息、获取应收额和方便业主使用的打印方法。

1. **报表类的接口**

在系统中，对报表类的操作有获取所有商铺合同、获取所有商铺收据、获得所有商铺应收额以及方便使用的打印选项。

1. **用户控制类的接口**  
   在系统中，用户类是最基本的一个控制类之一，CEO、CFO等控制类皆继承自User类。  
   User类提供了最基本的两个接口：登录（login）和注册（registration）。
2. **CEO控制类的接口**  
   CEO控制类提供了检查所有信息（check\_all\_information）、获得所有合同（get\_all\_contracts）、获得所有收据（get\_all\_receipts）、获得所有应收额（get\_all\_receivable）、确认合同（confirm\_contract）、提醒安排和到期（remind）以及签署合同（sign\_contract）的功能。
3. **CFO控制类的接口**  
   CFO控制类提供了检查所有信息（check\_all\_information）、提醒安排和到期（remind）的功能。
4. **业主控制类（ProprietorControl）接口**  
   业主控制类提供了申请进场（apply\_to\_enter）、申请转租（apply\_to\_sublet）、缴费（pay）、签订合同（sign\_contract）以及查看状态（view\_status）的接口。
5. **市场管理员控制类接口**  
   市场管理员控制类提供了查看合同（enter\_contract）、查看费用（enter\_fee）、以及提醒（remind）功能。
6. **GUI相关接口**  
   系统的GUI通过Qt构建，用户与界面的交互通过信号/槽函数（signal/slots）来实现，当用户与界面进行交互时，对应的控件发出特定的信号，而槽函数紧接着对特定的信号进行处理，对信号的处理过程可能包含切换界面、更新数据、调用控制类函数等操作。
7. **数据库相关接口**  
   数据库系统提供了对特定数据（如商铺信息、用户信息、合同信息等）进行增删查改的操作，同时还具有对不同数据进行综合搜索整理的特殊操作。

2.2.4 过程设计

过程设计是系统结构部件转换成软件的过程描述。

在本软件系统中，主要的两个过程为申请商铺和查询商铺信息。

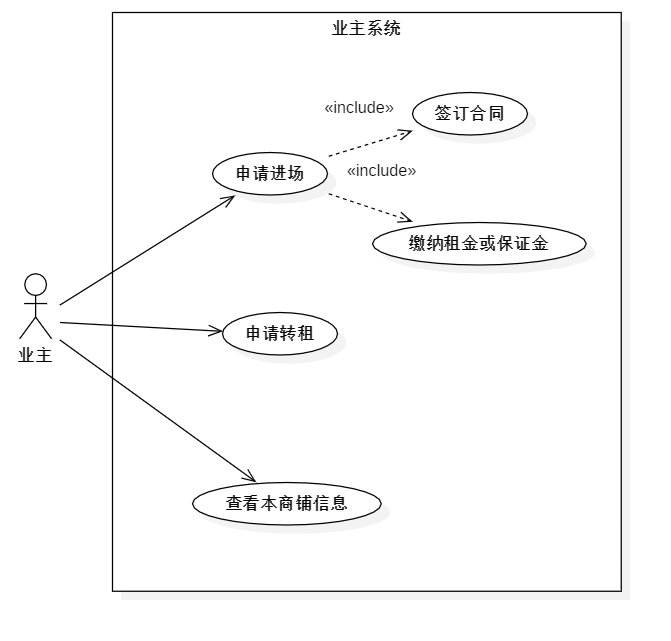
**一、申请商铺**  
 申请商铺的过程涉及界面层、控制层、实体层和数据库层。

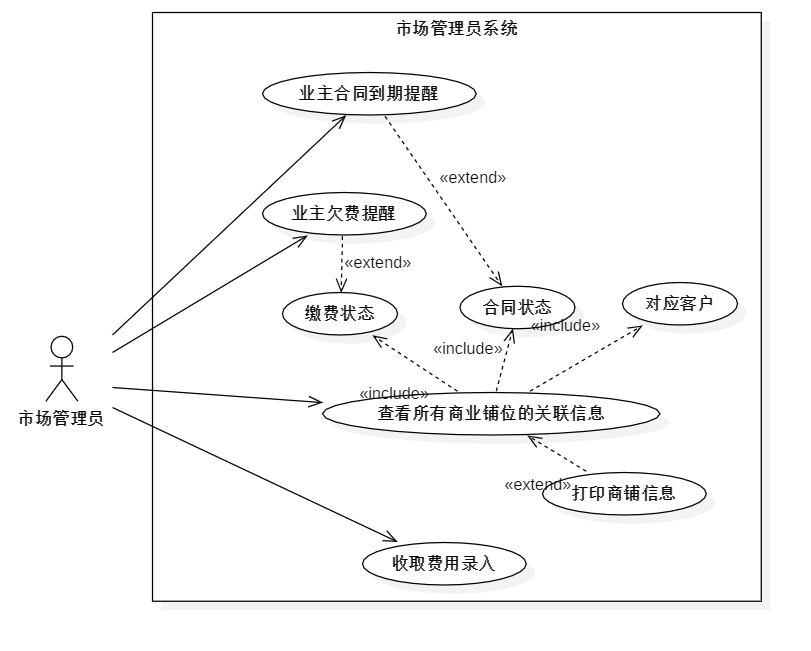
整个过程由用户在界面进行商铺信息的录入和确认提交商铺信息开始，紧接着界面层会与控制层进行交互，通知控制层进行相关信息的更新，在控制层的信息更新过程中，实体层负责数据的封装和处理，数据库层直接控制对数据的增删查改操作。

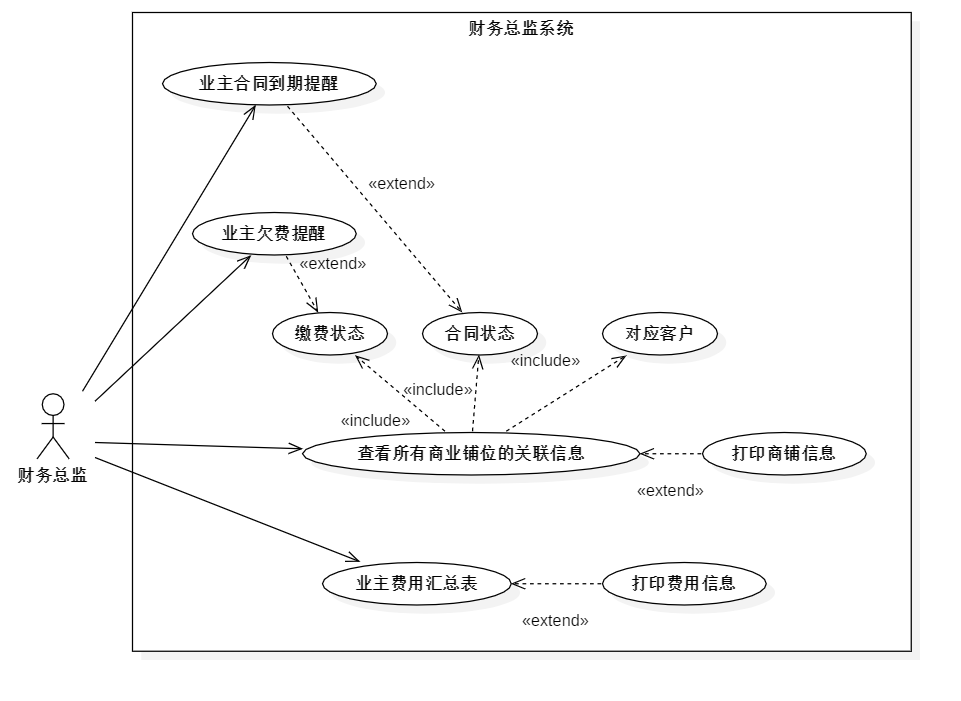
在整个过程中，不同的用户会参与整个过程，其中不同的用户（例如普通用户、业主、CEO、市场管理员）将操作不同的界面，同时调用不同控制类的函数，通过统一的实体类对数据库进行修改。

**二、查询商铺信息**  
 查询商铺信息的系统结构部件交互过程与申请商铺的过程类似，同样是由界面层出发，调用控制层的函数，并根据实体层从数据库层中取回的数据进行逻辑处理和界面更新操作。最终在外部界面更新出商铺信息。

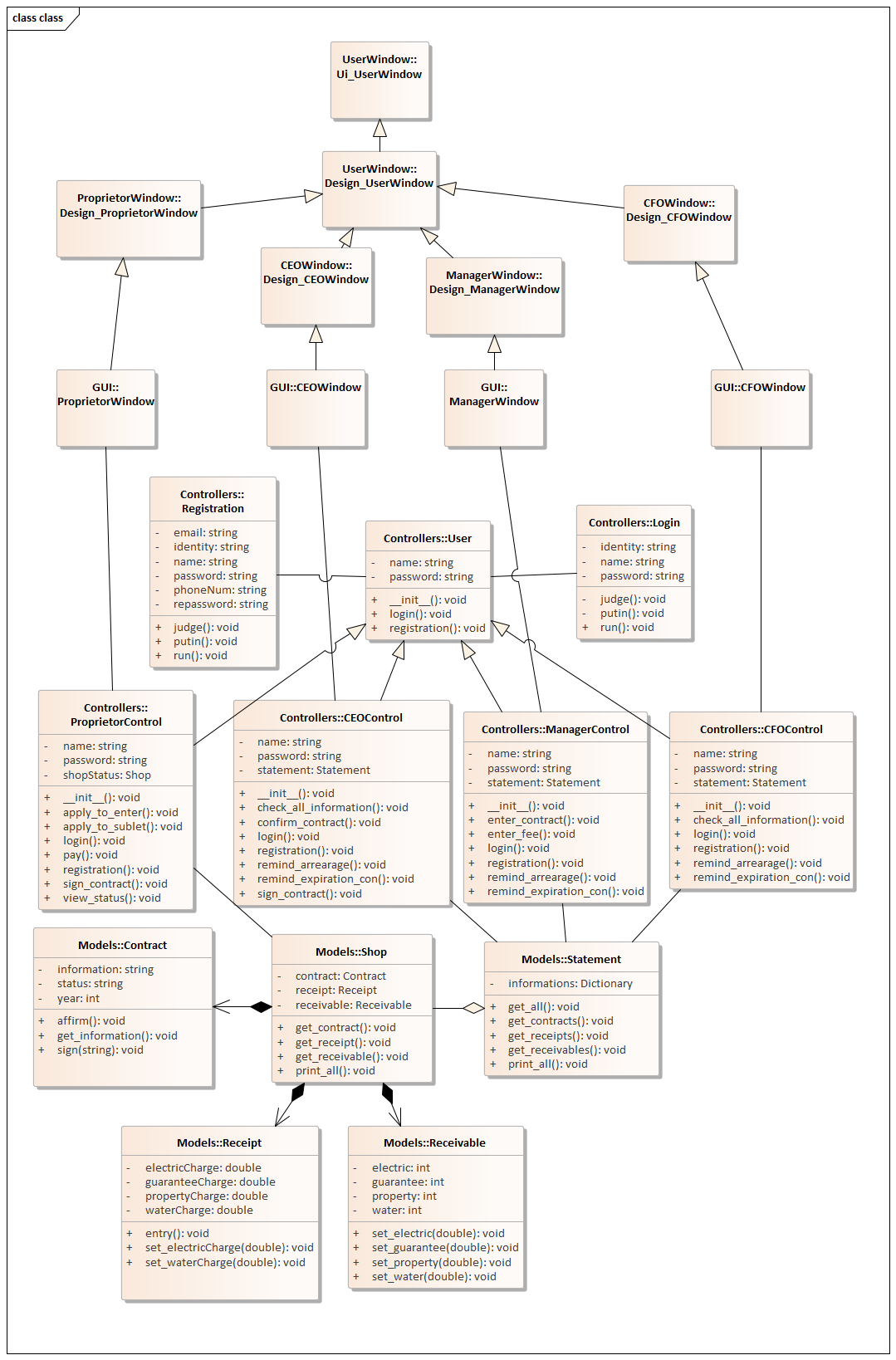
2.2.5 系统用例图







2.2.6 系统类图设计



# 第三章 软件实现

## 3.1 功能描述

应用程序的界面根据不同的用户会划分不同的板块：业主具有申请进场、我的店铺、我的信息、我的合同和申请转租五个板块；CEO具有全部店铺、审核合同和我的信息三个板块；Manager具有全部店铺、申请处理和拟定合同三个板块。

软件系统主要实现了**商铺申请、查看商铺信息和用户信用系统**。

3.1.1 商铺申请

**一、软件操作流程描述**

用户在“申请进场”板块，可以看到不同楼层的楼层信息。用户可以手动选择要查看的楼层，在点击某个店铺图标后，在商铺信息栏下会显示对应商铺的相关信息。

在用户点击某个可租店铺后，可以在申请进场栏进行商铺申请，填入必要信息后，即可点击“立即申请”按钮提交一个商铺申请。

市场管理员可以在“申请处理”板块查看待处理的申请，并通过点击不同的申请来查看这些申请的具体信息，显示在当前申请信息栏下。市场管理员可以选择点击“批准进场”或者“拒接进场”按钮，分别对应批准用户入场和拒绝用户入场。

接着，在市场管理员的“拟定合同”板块，可以进行合同内容的填入，同时，合同信息中还包括业主签字情况、总经理审核情况和总经理签字情况。在填写相关信息后，市场管理员即可点击“提交审核”按钮，将合同信息交给总经理以待审核。

然后，在总经理的“审核合同”板块，可以查看对应的合同信息，并通过“确认审核”按钮，对合同进行审核确认。

最后，在业主和总经理的合同查看界面，双方进行线上信息签订后，对应的商铺合同最终被确定，业主的店铺信息也被更新。

3.1.2 查看商铺信息

**一、软件操作流程描述**

用户可以在“申请进场”板块，看到不同楼层的楼层信息。用户可以在楼层信息栏进行楼层的选择，用户可以对楼层俯视图中的店铺方块进行点击，相应地，在店铺信息栏处会有店铺信息的更新，包括店铺的占地面积、保证金、年租金、物业费。

3.1.3 用户信用系统

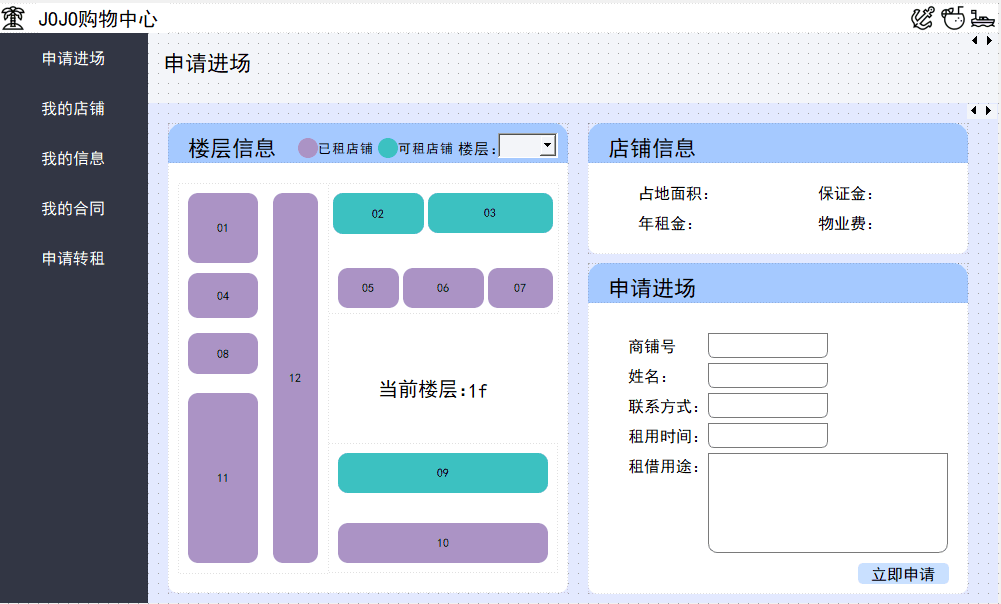
在系统中，用户具有信用积分，利用应用内信用积分可以参与平台活动等。

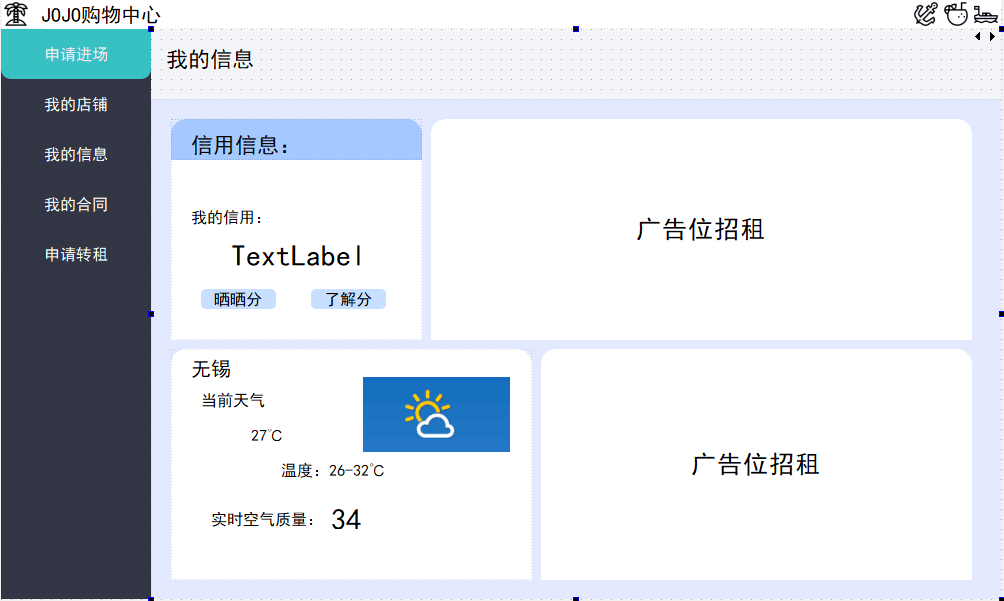
3.1.4 视觉障碍语音播报助手

考虑到使用本系统的用户可能为盲人，我们针对此类人群设计了视觉障碍语音播报助手，在用户点击按钮触发事件后，会播放对应按钮的语音，以便用户清楚当前处于的界面板块。

## 3.2 系统运行界面







如上图为本系统运行界面。其他界面在此不再过多展示。

## 3.3 主要设计及代码

3.3.1 软件核心系统设计及代码展示

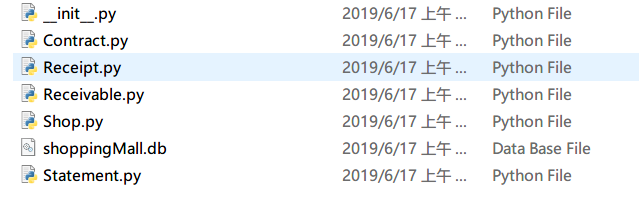
软件核心系统设计包含一系列控制类和实体类。

控制类所包含的所有类如下图所示：

图



实体类所包含的所有类如下所示：

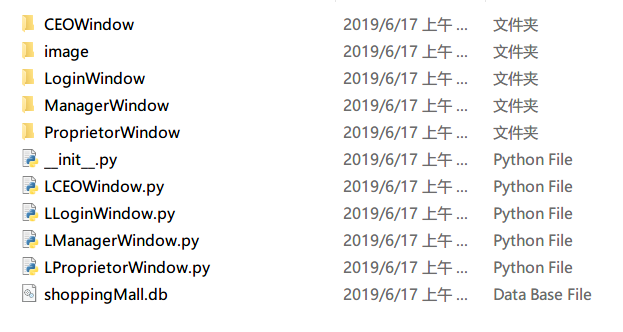


相关核心代码如下所示：

*#!/usr/bin/env python   
# -\*- coding:utf-8 -\*-  
from* Models.DataBase *import* DatabaseAccessor  
  
*class* Contract(*object*):  
 DB = DatabaseAccessor()  
 *def \_\_init\_\_*(*self*,ID=*None*):  
 *if* ID *is not None*:  
 *self*.\_\_ID = ID  
 *self*.\_\_contract = *self*.DB.get\_contract\_info\_by\_id(ID)  
  
 *self*.\_\_information = *self*.\_\_contract["contractInfo"]  
 *self*.\_\_status = *self*.\_\_contract["contractStatus"]  
 *self*.\_\_year = *self*.\_\_contract["contractYear"]  
  
 *self*.\_\_proprietorSign = *self*.\_\_contract["proprietorSign"] *# 业主是否签字  
 self*.\_\_CEOAffirm = *self*.\_\_contract["CEOAffirm"] *# 总经理是否确认  
 self*.\_\_CEOSign = *self*.\_\_contract["CEOSign"] *# 总经理是否签字  
  
 self*.\_\_userName = *self*.\_\_contract["userName"]  
 *self*.\_\_shopIndex = *self*.\_\_contract["shopIndex"]  
 *self*.\_\_userTelContract = *self*.\_\_contract["userTelContract"]  
  
 *else*:  
 *pass  
  
 # self.\_\_information = ''  
 # self.\_\_status = ''  
 # self.\_\_year = -1  
  
 # self.\_\_proprietorSign = False # 业主是否签字  
 # self.\_\_CEOAffirm = False # 总经理是否确认  
 # self.\_\_CEOSign = False # 总经理是否签字  
  
 def* add\_new\_contract\_info(*self*,ID,shopNum,userName,userTelContract,contractInformation,rentTimeManager):  
 *self*.DB.add\_new\_contract\_info(ID,shopNum,userName,userTelContract,contractInformation,rentTimeManager)  
  
 *def* set\_contract\_ceoaffirm\_by\_id(*self*, user\_id, ceoaffirm):  
  
 *self*.DB.set\_contract\_ceoaffirm\_by\_id(user\_id, ceoaffirm)  
  
 @property  
 *def* proprietorSign(*self*):  
 *return self*.\_\_proprietorSign  
  
 @property  
 *def* CEOAffirm(*self*):  
 *return self*.\_\_CEOAffirm  
  
 @property  
 *def* CEOSign(*self*):  
 *return self*.\_\_CEOSign  
 @property  
 *def* information(*self*):  
 *return self*.\_\_information  
  
 @information.setter  
 *def* information(*self*, \_information):  
 *self*.\_\_information = \_information  
  
 @property  
 *def* status(*self*):  
 *return self*.\_\_status  
  
 @status.setter  
 *def* status(*self*, \_status):  
 *self*.\_\_status = \_status  
  
 @property  
 *def* year(*self*):  
 *return self*.\_\_year  
  
 @year.setter  
 *def* year(*self*, \_year):  
 *self*.\_\_year = \_year  
  
 *def* affirm(*self*):  
 *"""  
 总经理确认* ***:return****: None  
 """  
 self*.\_\_CEOAffirm = 1  
 *self*.DB.set\_contract\_ceoaffirm\_by\_id(*self*.\_\_ID,1)  
  
 *return None  
  
 def* sign(*self*, status):  
 *"""  
 业主或总经理签字* ***:return****:  
 """  
 # 业主签字  
 if* status == 'p':  
 *self*.\_\_proprietorSign = *True  
 elif* status == 'c':  
 *self*.\_\_CEOSign = *True  
  
 return None* @property  
 *def* allInfo(*self*):  
 allInfo = {"information": *self*.\_\_information,  
 "status": *self*.\_\_status,  
 "year": *self*.\_\_year}  
 *return* allInfo  
  
 @allInfo.setter  
 *def* allInfo(*self*, \_allInfo):  
 *if* \_allInfo["information"] != '':  
 *self*.information = \_allInfo["information"]  
  
 *if* \_allInfo["status"] != '':  
 *self*.status = \_allInfo["status"]  
  
 *if* \_allInfo["year"] != '':  
 *self*.year = \_allInfo["year"]  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 c = Contract()  
  
 c.information = "asas"  
  
 info = {"information": '信息',  
 "status": "状态",  
 "year": 3}  
 c.allInfo = info  
  
 *print*(c.information)  
 *print*(c.year)  
 *print*(c.status)

3.3.2 UI设计及代码展示

UI部分的设计包含一系列与界面相关的.py文件和资源文件，如图所示：



关键界面在上文中已示。

界面部分代码实现如下所示（代码量过大，在此不宜展示）：

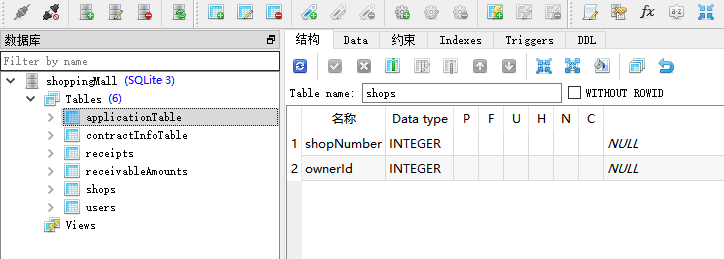
*from* PyQt5 *import* QtCore, QtGui, QtWidgets  
  
*class* Ui\_LoginInterface(*object*):  
 *def* setupUi(*self*, LoginInterface):  
 LoginInterface.setObjectName("LoginInterface")  
 LoginInterface.setEnabled(*True*)  
 LoginInterface.resize(457, 356)  
 LoginInterface.setStyleSheet("")  
 *self*.frame = QtWidgets.QFrame(LoginInterface)  
 *self*.frame.setGeometry(QtCore.QRect(10, 10, 430, 330))  
 *self*.frame.setStyleSheet("\*{\n"  
" font-family:century gothic;\n"  
" font-size:16px\n"  
"}\n"  
"\n"  
"#frame\n"  
"{\n"  
"background-image: url(:/icon/背景图片.png);\n"  
"}\n"  
"QFrame\n"  
"{\n"  
" border-top-left-radius:15px;\n"  
" border-bottom-right-radius:15px;\n"  
" \n"  
"}\n"  
"\n"  
"QLabel\n"  
"{\n"  
" color:white;\n"  
"}\n"  
"\n"  
"\n"  
"\n"  
"\n"  
"\n"  
"QLineEdit\n"  
"{\n"  
"background:transparent;\n"  
"border:none;\n"  
"color:#717072;\n"  
"border-bottom:1px solid#717072;\n"  
"}\n"  
"\n"  
"QCheckBox\n"  
"{\n"  
" color:white;\n"  
"}\n"  
"\n"  
"\n"  
"")

3.3.3 数据库设计及代码展示

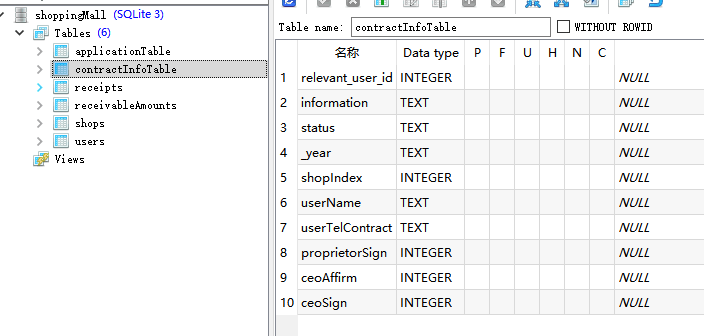
数据库系统包含位于本地的数据库文件和访问数据库的一系列接口。

本地数据库包含的一系列表格采用一系列结构化设计，具体的数据结构如图：

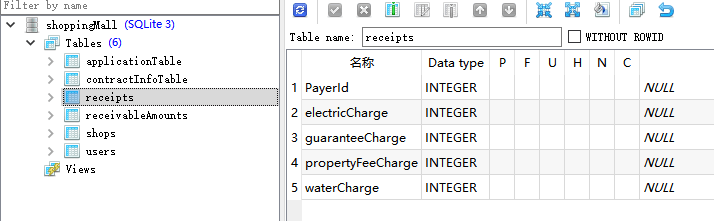
一、商铺表



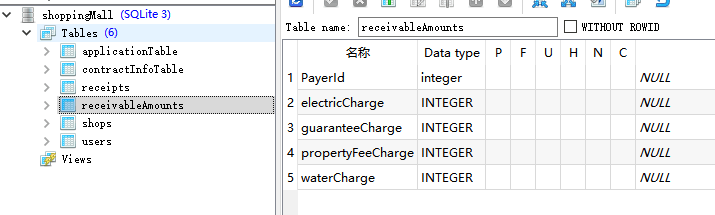
二、合同信息表



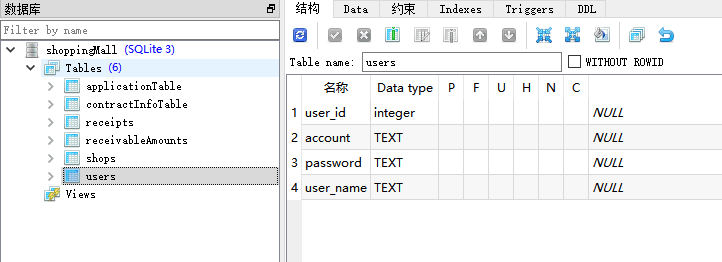
三、收据信息表



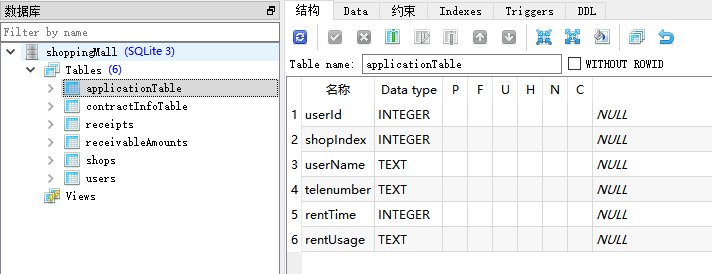
四、应收额信息表



五、用户表



1. 申请表



数据库系统的实现包含一系列对不同表进行操作的函数，如添加、删除、更改、查询操作，根据实际系统内部逻辑的需求，数据库系统提供的具体功能较为灵活，如根据用户id查询与用户相关的合同信息、商铺信息和资金信息或者获得所有现存商铺的相关信息等等。

具体的数据库系统部分实现代码如下：

*import* sqlite3  
*import* os  
  
*class* DatabaseAccessor(*object*):  
  
 electricCharge = "ElectricCharge"  
 guaranteeCharge = "GuaranteeCharge"  
 propertyFeeCharge = "PropertyFeeCharge"  
 waterCharge = "WaterCharge"  
  
 *def \_\_init\_\_*(*self*):  
 *self*.connect = sqlite3.connect(os.getcwd() + "//shoppingMall.db")  
 *self*.cursor = *self*.connect.cursor()  
 *self*.name = "DatabaseAccessor"  
 *self*.create\_user\_table()  
 *self*.create\_contract\_table()  
 *self*.create\_receipt\_table()  
 *self*.create\_receivable\_amount\_table()  
 *self*.create\_shop\_table()  
 *self*.create\_application\_table()  
 *self*.connect.commit()  
 *self*.clear\_empty\_form()  
  
 *def* clear\_empty\_form(*self*):  
 *self*.cursor.execute('DELETE FROM applicationTable WHERE userId = ""')  
 *self*.cursor.execute('DELETE FROM contractInfoTable WHERE relevant\_user\_id = ""')  
 *self*.cursor.execute('DELETE FROM receivableAmounts WHERE PayerId = ""')  
 *self*.cursor.execute('DELETE FROM receipts WHERE PayerId = ""')  
 *self*.cursor.execute('DELETE FROM shops WHERE shopNumber = ""')  
 *self*.cursor.execute('DELETE FROM users WHERE user\_id = ""')  
 *self*.connect.commit()  
  
  
 *# user table  
  
 def* create\_user\_table(*self*):  
 *"""  
 如果发现数据库中没有用户table，则创建* ***:return****:  
 """  
 print*("创建用户表")  
 *self*.cursor.execute('CREATE TABLE IF NOT EXISTS '  
 'users(user\_id integer, account TEXT,'  
 'password TEXT, user\_name TEXT)')  
  
 *def* add\_new\_user(*self*, user\_id, account, password, username):  
 *"""  
 添加用户，添加操作无法撤回，小心调用这个函数* ***:param*** *user\_id:* ***:param*** *account:* ***:param*** *password:* ***:param*** *username:* ***:return****:  
 """  
 print*("添加用户：")  
 *self*.cursor.execute('INSERT INTO users VALUES (?, ?, ?, ?)',  
 (user\_id, account, password, username))  
 *self*.connect.commit()  
  
 *def* get\_user\_info\_by\_id(*self*, user\_id):  
 *"""  
 根据id获取用户所有信息（合同、已收额、应收额）* ***:param*** *user\_id: id* ***:return****:返回与用户id相关的合同、已收额、应收额信息  
 """  
 print*("获取对应id的合同、已收额、应收额")  
 *self*.cursor.execute('SELECT \* FROM receivableAmounts WHERE PayerId= ?',  
 *str*(user\_id))  
 receivable\_data = *self*.cursor.fetchall()  
 *self*.cursor.execute('SELECT \* FROM receipts WHERE PayerId= ?',  
 *str*(user\_id))  
 receipt\_data = *self*.cursor.fetchall()  
 *self*.cursor.execute('SELECT \* FROM contractInfos WHERE relevant\_user\_id = ?',  
 *str*(user\_id))  
 contract\_data = *self*.cursor.fetchall()  
 user\_info = {}  
 *# contract  
 for* user, i *in zip*(contract\_data, *range*(*len*(contract\_data))):  
 user\_info["contractInfo{}".format(i)] = user[1]  
 user\_info["contractStatus{}".format(i)] = user[2]  
 user\_info["contractYear{}".format(i)] = user[3]  
  
 *# receivable  
 for* user, i *in zip*(receivable\_data, *range*(*len*(receivable\_data))):  
 user\_info["receivedEleCharge{}".format(i)] = user[1]  
 user\_info["receivedGuarCharge{}".format(i)] = user[2]  
 user\_info["receivedPropCharge{}".format(i)] = user[3]  
 user\_info["receivedWaterCharge{}".format(i)] = user[4]  
 *# receipt  
 for* user, i *in zip*(receipt\_data, *range*(*len*(receipt\_data))):  
 user\_info["receiptEleCharge{}".format(i)] = user[1]  
 user\_info["receiptGuarCharge{}".format(i)] = user[2]  
 user\_info["receiptPropCharge{}".format(i)] = user[3]  
 user\_info["receiptWaterCharge{}".format(i)] = user[4]  
 *return* user\_info  
  
 *# receivable  
 def* create\_receivable\_amount\_table(*self*):  
 *"""  
 创建receivableAmountTable* ***:return****:  
 """  
 print*("创建应收额信息表")  
 *self*.cursor.execute('CREATE TABLE IF NOT EXISTS '  
 'receivableAmounts(PayerId integer, '  
 'electricCharge INTEGER,'  
 ' guaranteeCharge INTEGER, '  
 'propertyFeeCharge INTEGER,'  
 ' waterCharge INTEGER)')  
  
 *def* add\_new\_receivable\_amount\_info(*self*, payer\_id, electric\_charge,  
 guarantee\_charge, property\_fee\_charge, water\_charge):  
 *"""  
 添加应收额信息* ***:param*** *payer\_id:* ***:param*** *electric\_charge:* ***:param*** *guarantee\_charge:* ***:param*** *property\_fee\_charge:* ***:param*** *water\_charge:* ***:return****:  
 """  
 print*("添加应收额信息")  
 *self*.cursor.execute('INSERT INTO receivableAmounts VALUES (?, ?, ?, ?, ?)',  
 (payer\_id, electric\_charge, guarantee\_charge, property\_fee\_charge, water\_charge))  
 *self*.connect.commit()  
 *# 字典的列表  
 # list\_.append({})  
  
 def* get\_receivable\_info\_by\_id(*self*, user\_id):  
 *"""  
 根据id获取应收额信息* ***:param*** *user\_id:* ***:return****:  
 """  
 self*.cursor.execute('SELECT \* FROM receivableAmounts WHERE PayerId= ?',  
 (*str*(user\_id),))  
 receivable\_data = *self*.cursor.fetchall()  
 user\_info = {}  
 user\_info["electricCharge"] = receivable\_data[0][1]  
 user\_info["guaranteeCharge"] = receivable\_data[0][2]  
 user\_info["propertyFeeCharge"] = receivable\_data[0][3]  
 user\_info["waterCharge"] = receivable\_data[0][4]  
 *return* user\_info

# 第四章 结束语

## 4.1 总结

4.1.1 总用时和代码量

**一、总用时**

本系统分为三个阶段：设计阶段、编写系统阶段、收尾阶段。

设计阶段开始于4月底左右，结束于五月中旬，期间进行了系统的用例分析、类图设计和通信图设计。

编写系统阶段开始于五月中旬，结束于六月中旬，期间组员利用Python和PyQt以及SQLite进行了软件系统的实现工作。

最后是项目收尾阶段，从六月中旬直到项目交题，进行项目的整体收尾、debug和有关文档的收尾工作。

项目总用时大约为两个月，人均投入时间约为1.5h/day。

**二、代码量**

代码分为三个部分：核心逻辑系统、GUI部分以及数据库系统部分。

核心逻辑系统的代码量为：1300行左右。

GUI部分的代码量为：3500行左右。

数据库系统的代码量为：500行左右。

4.1.2 有关分工及合作

在项目进行过程中，项目组长王子贤，项目组员闫逸飞、冯香睿和张泽龙共同完成了本系统的设计和构建过程。

1. **分析和设计过程**

在项目开发前期，小组人员进行了多次会议，进行本商场软件系统的需求分析和软件设计工作，几次会议中，讨论内容涵盖了许多内容，其中包括：项目使用的编程语言、界面的开发环境、重要用例的分析、系统中所使用的类的提取、程序框架的选择、数据库语言的选择以及人员分工问题。

分析和设计过程由全部小组成员共同完成。

1. **系统实现和项目结题过程**

在开始实际的系统实现阶段后，小组人员各司其职，尽量地减少了组员工作的交叉和不必要的学习成本、沟通用时。

软件系统的主要框架设计和后台系统编写由王子贤完成，界面系统的设计和实现由冯香睿和张泽龙共同完成，数据库系统的设计和实现由闫逸飞和张泽龙共同完成，最终软件系统的描述文档由闫逸飞进行内容设计、所有组员共同贡献文档内容完成。

## 4.2 不足之处

4.2.1 有关界面

**一、整体风格**

在整体风格设计方面，本小组缺少较为广泛的调研，且对于整体风格的设计方法没有一个系统的学习，在实际设计过程中，感到缺少一个贯穿始终的设计方法，最终对于整体风格的把握有部分欠缺。

**二、细节处理**

在细节处理方面，由于对于UI设计方面缺少理论支持，对于界面的审视角度有限，无法发现较为细节的设计点，最终导致整体感受同市面上成功的设计产品较为松散，这也是遗憾之一。

**三、图案设计**

在图案设计方面，由于小组人员对具体的设计工具掌握不足，同时缺少UI设计的理论支持，最终缺少一个足够具体和系统的、能够表达产品理念的图案设计，对于总体的外部表现有部分影响。

4.2.2 有关数据库

**一、SQLite的局限性**

不难否认，SQLite作用于小体量软件时，其功能实现和修改都较为简约。SQLite的简约不仅体现在功能实现方面上，而是存在于这个数据库语言的各处：较少的数据结构、微型的数据库文件甚至连安装都十分简单，安装完毕后，竟然还没有附带SQLite数据库查看器。

使用这种轻量级、简单化的数据库语言，有利于设计人员更快地进行功能的实现和测试，同时，简单化还带来功能的稳定性，因为越简单越不容易出错。

但简单化在一个数据库引擎中也可以是一个缺点，为了达到简单化，SQLite省略了一些比较有用的特性，比如高并发性、严格的存取控制、复杂的SQL语言特性、超大的万亿级别的数据测量等等。由于简单化，在应用SQLite数据库时，项目的数据安全性、存取高效性、大体量特质可能会得不到较高保证。

由于本项目对应的数据体量较小，也没有较严格的存取效率要求，所以以SQLite作为数据库较为合适：保证功能稳定达标的情况下，又有较高的开发效率。

而当软件系统绑定的数据量慢慢提升，其对数据处理的要求逐渐上升时，SQLite的局限性会慢慢显现。为了防止这种情况的发生，项目组应在设计阶段确认好可能达到的项目数据规模，并对长期效益和实际开发效率等因素综合考虑，做出合适的数据库类型选择。

**二、数据库规模和数据库系统设计**

较为遗憾的是，在本项目开发过程中，实际的数据规模并没有达到一个大型商场的数据量大小，为了达到数据的可控和为了方便程序Debug，本项目的数据规模限制在一个较小的数值。

在数据库系统的设计过程中，开发人员缺乏较为高级的数据库语言编写能力，所幸软件所要求的数据处理需求并不过于复杂，也不涉及大型数据、大范围数据的处理，最终的数据库系统实现了所有软件系统的需求。

但以长远的软件开发的角度考虑，数据库系统应对数据有较好的控制和管理，并且易于更改和理解，本系统的数据库系统在这些方面仍有提升空间。

4.2.3 系统扩展性

由于时间原因，本项目未能够进行充分的调研和系统设计，导致最终的软件系统的扩展性有限：数据库系统的扩展性和界面设计的扩展性在面临功能修改时较为有限，往往一个更改的需求需要进行一系列修改，比如界面的颜色风格需要进行更换时，就需要打开所有的界面依次对控件的设置进行修改。

在经过思考后，小组人员考虑能够通过抽象数据库系统的方法对数据库的可扩展性加以提升，同时也可以对界面类加以封装，方便后续的界面修改。

# 参考文献

1. 谢若承,陈顺德,完泾平.企业信息化软件应用生命周期管理及应用[J].电力信息与通信技术,2014,12(01):33-37.
2. –c.f.Tripakis(2016).
3. 甘霖,潘虹.关于软件工程对现代化生活造成影响的思考[J].计算机产品与流通,2019(05):31.
4. 徐斌.软件工程在企业信息化平台建设中的作用[J].电子技术与软件工程,2019(06):241-242.
5. 马颖.软件工程在企业信息化平台建设中的应用研究[J].计算机产品与流通,2019(02):21.
6. 伍琴兰.软件工程方法在计算机软件开发中的应用分析[J].信息系统工程,2018(10):52.
7. 贾磊,倪振兴,李玉萍.企业信息化软件项目技术初探[J].数字技术与应用,2012(05):250.
8. 匡南.浅谈中小企业管理信息系统现状及对策[J].商场现代化,2014(19):100.
9. 潘家骏.浅析我国中小企业管理信息系统的现状及对策[J].信息系统工程,2017(11):65.
10. 张宏佳.网络信息时代下软件工程技术的发展[J].化工管理, 2017, (08) :232.
11. 范萱萱,周天宏.软件工程产业现状与未来发展对策[J].电脑编程技巧与维护,2019(04):55-57.
12. Stavros Tripakis. The Science of Software and System Design ⁎ ⁎ Some of the ideas in this paper appear also in Tripakis (2016, 2018).[J]. IFAC PapersOnLine,2018,51(7).
13. Turetken,Ondracek,IJsselsteijn. Influential Characteristics of Enterprise Information System User Interfaces[J]. Journal of Computer Information Systems,2019,59(3).
14. 林影丽,李德杰,郭隆健.以用户为中心的产品界面设计流程和方法[J].机电产品开发与创新,2019,32(01):31-33+37.