|  |  |
| --- | --- |
|  |  |

**本科生毕业论文（设计）**

基于Python的租赁房屋管理

系统与实现

专 业 电气工程及其自动化

姓 名 王举

学 号 16031107

指 导 教 师 陈垚

完 成 时 间

**教务处 印**

**摘要**

随着我国经济的快速发展的日益加速，城市流动人口逐年增多，社会对房屋租赁的需求也因此快速增加，房屋的出租和租赁已经成为生活的重要组成元素。房屋租赁的管理本身就是一项琐碎、复杂而且又十分细致的工作，传统的管理方式通常会耗费工作人员的大量时间和精力，而且相关信息也容易出错。伴随着这项业务的开展，高效、合理的管理房屋租赁的重要性也日益凸现。另一方面，由于互联网技术在近几十年的飞快发展，传统的商业活动逐渐引入了互联网技术，企业利用互联网技术极大提高了商务运作效率，房屋租赁业务也从传统的手工登记模式走入了网络空间。“基于Python的租赁房屋管理系统”正是为了实现高效、准确进行房屋管理而形成的一种新型信息化房屋管理系统。

本文首先对该系统的开发环境（PyCharm、HBuilder X和微信小程序开发者工具）进行了概述，在仔细考量了各种技术的优点和缺点之后，结合本系统的需求和特点提出本系统采用的前台和后台实现方式:该系统分前台和后台两大模块，考虑到Python的Django框架强大的后台管理能力，因此后台管理功能的实现主要采用Django框架；此外， Python的Flask框架又特别的小巧、灵活，因此实现前台API接口的实现采用的是Flask框架，值得一提的是由于DCloud公司推出的uni-app前端框架具有跨平台、可转换、小巧等优点，故前台的前端代码完全采用uni-app实现。需要指出，在该系统中数据的缓存采用的是Redis缓存技术，数据的持久性存储采用的是MySQL技术。同时也引入阿里云的短信验证、身份认证以及OSS存储等第三方技术。

该系统的后台模块针对的用户群体主要有超级管理员、赞助管理员和审核管理员。其中超级管理员的权限有角色管理、用户管理、公告发布和删除、评论管理、投诉管理；审核管理员则可以进行房屋信息的审核和公告审核；赞助管理员就只能进行广告的管理和代金券的管理。

该系统的前台针对的群体则主要是普通租户或房东。其中普通租户便只能进行房屋信息查看以及评论、投诉等基本操作，如果希望发布房源，便可以在升级为房东之后发生该操作。

**关键字:**房屋租赁，Python，Django，Flask，uni-app，MySQL，后台，前台

**Abstract**

With the rapid acceleration of China's rapid economic elopment， the city's floating population is increasing year by year， and the society's demand for housing leasing has therefore rapidly increased. Housing leasing and leasing have become an important element of life. The management of house leasing itself is a trivial， complicated and very meticulous job. The traditional management method usually consumes a lot of time and energy of the staff， and the related information is also prone to errors. With the development of this business， the importance of efficient and reasonable management of house leasing has become increasingly prominent. On the other hand， due to the rapid development of Internet technology in recent decades， traditional business activities have gradually introduced Internet technology. Enterprises have used Internet technology to greatly improve the efficiency of business operations. Housing leasing business has also entered from the traditional manual registration model. Cyberspace. "Python-based rental housing management system" is a new type of information-based housing management system formed for efficient and accurate housing management.

This article first outlines the development environment of the system (PyCharm， HBuilder X， and WeChat applet developer tools). After carefully considering the advantages and disadvantages of various technologies， this paper combines the needs and characteristics of the system and proposes the use of the system. Front-end and back-end implementation methods: The system is divided into front-end and back-end modules. Considering the powerful background management capabilities of Python's Django framework， the background management function is mainly implemented by Django framework; in addition， Python's Flask framework is particularly small， Flexible， so the front-end API interface is implemented using the Flask framework. It is worth mentioning that the uni-app front-end framework introduced by DCloud has the advantages of cross-platform， convertible， and compact， so the front-end code of the front desk is completely uni- app implementation. It should be pointed out that the data cache in this system uses Redis cache technology， and the persistent storage of data uses MySQL technology. At the same time， third-party technologies such as Alibaba Cloud SMS verification， identity authentication and OSS storage are also introduced.

The user groups targeted by the background module of the system mainly include super administrators， sponsor administrators and audit administrators. Among them， the super administrator's permissions include role management， user management， announcement release and deletion， comment management， and complaint management; the audit administrator can conduct house information audit and announcement audit; Management of coupons.

The group targeted by the front desk of the system is mainly ordinary tenants or landlords. Among them， ordinary tenants can only perform basic operations such as viewing house information， comments， and complaints. If you want to release a house， you can do this after upgrading to a landlord.

**Keywords:** housing lease， Python， Django， Flask， uni-app， MySQL， background， front desk

**目录**

**第一章 绪论1**

1.1研究的目的和意义1

1.1.1 研究目的1

1.1.2 研究意义1

1.2课题的国内外研究现状1

1.2.1 国内现状1

1.2.2 国外现状3

1.3论文的主要功能与结构3

1.3.1 论文的主要作用3

1.3.2 论文的主要结构4

**第二章 开发环境及相关技术介绍**4

2.1开发环境介绍4

2.1.1 PyCharm4

2.1.2 HBuilder X5

2.2相关技术介绍5

2.2.1 Python5

2.2.2 Web框架5

2.2.3数据库技术6

2.2.4 uni-app6

**第三章 系统分析**7

3.1可行性分析7

3.1.1 技术可行性7

3.1.2 经济可行性7

3.1.3 操作可行性7

3.2 业务分析7

3.2.1 用户群体7

3.2.2 业务模型图7

3.2.3 模型表建立8

**第四章 系统的设计与实现**8

4.1系统后台实现8

4.1.1 登录界面9

4.1.2 超级管理员界面9

4.1.3 审核管理员界面10

4.1.4 赞助管理员界面10

4.2系统前台设计10

4.2.1 首页模块10

4.2.2 收藏模块11

4.2.3 聊天模块11

4.2.4 我的模块11

**第五章 总结**12

**参考文献**13

**附录**14

**致谢**15

**第一章 绪论**

**1.1 课题研究的目的和意义**

**1.1.1 研究目的**

近年来，我国经济发展迅速，城市化建设步伐日益加速。作为世界第一人口大国，第二大经济体，我国各大城市流动人口逐步增加，导致对房屋租赁的需求也日益剧增。房屋中介也大量地涌现了出来，出租、租赁的大量业务量产生了对高效合理的租赁业务信息管理系统的迫切需求。另一方面，因为计算机的崛起，设计一款信息管理系统，可以极大的提高人们工作效率。

**1.1.2 研究意义**

由于国外信息技术起源比较早，在各个行业的应用也较为普及，国外信息化在房屋租赁方面的管理应用也比较成熟。在欧美等发达国家约有一半的中低收入者都是依靠房屋租赁来免租住宿需求，这一数字远远多于发展中国家的房屋租赁人群。在国内，经过多年的经济建设和城镇化建设，国内农民所占全国总人口的比例逐渐减少，越来越多的农民主键有乡村转移到城市，还包括大批从乡村流动到城市的大学生，这些大量流动的人口组成了租赁市场的主力军。庞大的流动人口数目产生了蓬勃的房屋租赁市场，然而，国内的房屋租赁市场还不发达。进入信息化时代，房产行业的发展和居民的租房需求的多元化带动了房产中介的发展，虽然存在这比较多的中介网站，单中介的信息化管理还不够完善，不能够对提高晓丽和降低成本起到至关重­要的作用。在我国房产中介仍然限于传统的方式，信息都是孤立的。未来房产租赁信息市场面临着网络化发展的趋势，从而实现房产交易中共享信息的功能，增加交易的机会，缩短交易的时间。

**1.2 课题研究的国内外现状**

**1.2.1 国内现状**

随着我国城镇化的快速发展，越来越多的人民进入城市工作和学习。同时，我国高校的大面积扩展，将更多的毕业高校学生留在了城市工作。这为房屋租赁带来了大量的客源。这也促进了我国房屋租赁市场的开发以及各种系统网站的发布。而我国的房屋租赁市场并不是一撮而就地发展，而是经过了几个过程。

中国加入 WTO 后，房屋租赁行业受到国外竞争者的挑战。国外企业凭借先进的信息分享平台和互联网技术，对我国房屋租赁市场造成很多冲击。面对国外的竞争，原先以手工方式进行管理房屋信息的我国企业开始自己摸索和探究房屋租赁系统的开发。

进入新世纪，我国的互联网企业得到快速发展，一大部分互联网企业掌握了核心技术并结合我国市场的特色都已发展长较大的企业。例如赶集网、58 同城等。房屋租赁市场也都称为各大第三方平台业务的一部分，得到了快速发展。而我国的中小企业的房屋租赁系统发展较缓慢，各自之间没有数据共享，各自为阵。系统设计开发也不规范，没有留下相关的数据接口供第三方使用，以利用“互联网+”的思维进行数据挖掘。现如今我国房屋租赁系统还存在如下一些问题：

（1）房屋信息不正确。很多企业发布的信息与实际房屋信息不符合，并且太多的房屋信息给用户甄别带来了很大的难度。

（2）信息化程度低、专业化程度不高。租赁管理系统还无法适应现代信息管理系统的要求。

（3）信息管理系统专业人才短缺。

基于上述问题，我国很多学者对我国的房屋租赁管理系统做了一些研究。还有部分学者根据存在问题设计了一些简单的系统供借鉴。主要的研究学者有，张哲使使用 C#，ASP.NET 开发环境，采用 MVC 框架设计实现了房屋销售与租赁信息网站。唐娜从社区暂住人口为切入点使用.net 实现了房屋出租管理系统。施磊设计了基于 B/S 结构，使用.net 技术设计了房屋租赁管理系统；方义从房屋租赁管理系统的设计与开发中的问题出发，进行了相关研究，供其参考。任小艳使用 UML 建模语言分析房屋租赁管理系统。李宁使用 PHP 开发设计了一个房屋租赁管理系统。吴琼和张旭超使用 B/S 模式对房屋租赁管理系统进行了设计。高海静.采用 MVC 架构设计了一个新型房屋租售管理系统。另一些学者从别的学科借鉴技术设计了房屋租赁管理系统。例如李国华使用 GIS 技术进行了设计。

**1.2.2 国外现状**

由于信息技术产生于欧美，房屋租赁系统的开发也较早在欧美国家进行，房屋租赁市场的信息化早已完成。但国外技术房屋租赁系统的完善，使得很多学者关心使用管理思想解决在房屋租赁过程中产生的问题以及结合功能模块自己开发系统的框架。主要的研究学者包括：Forgionne G A 使用一种决策系统方法分配军队租房距离的算法，尽可能减少距离。Park M L 等提出了一种新的技术管理住房问题。Darinka Czischke 认为租房部门应保持高度警惕，应采用新的战略和技术提高房屋租赁管理的效率，这样有利于管理租赁房屋信息，并提出设计了一个可以提高经理管理效率的出租屋管理系统。对技术和方法改进的还有 Wei P X和 Roberts A等学者的研究。

国外成熟的房屋租赁公司利用Internet 实现对房源和客源双重信息的管理，通过 MLS(多重上市服务)对房源和客源信息对接，实现了两个数据库的联合管理。此外还有一些学者强调了房屋租赁管理系统的重要性。

**1.3 论文的主要功能与结构**

**1.3.1 论文的主要作用**

本文首先认真研究了当下各优秀的Web框架及数据库相关技术的理论知识以及各自的优缺点，然后从该项目的业务流程、功能性需求以及使用场景等多个方面从何考虑，基于Python开发出了一套适用于国内现状的房屋租赁系统。本论文所做的主要研究工作如下：

(1)对国内房屋租赁市场进行了一定的需求分析和研究，同时结合市面上已经成熟的租赁系统，搭建出了一套适用于国内的反诬租赁系统的业务流程图。

(2)分析管理系统的结构的特点以及各Web框架的优缺点，选择了合适的Web进行来针对性实现功能。

(3)针对数据的存储的目的以及使用场景进行分析，采用MySQL数据库进行数据的持久性存储，同时选择Redis进行了数据的缓存。

(4)采用相关技术，完成基于Python的房屋租赁系统的整体结构设计和功能模块实现。

**1.3.2 论文的主要结构**

本文共分五章。

第一章，介绍该系统的研究意义和背景、国内国外的研究现状以及本论文的主要功能与结构；

第二章，介绍本系统采用的开发环境以及主要技术；

第三章，对本系统进行可行性分析和业务分析；

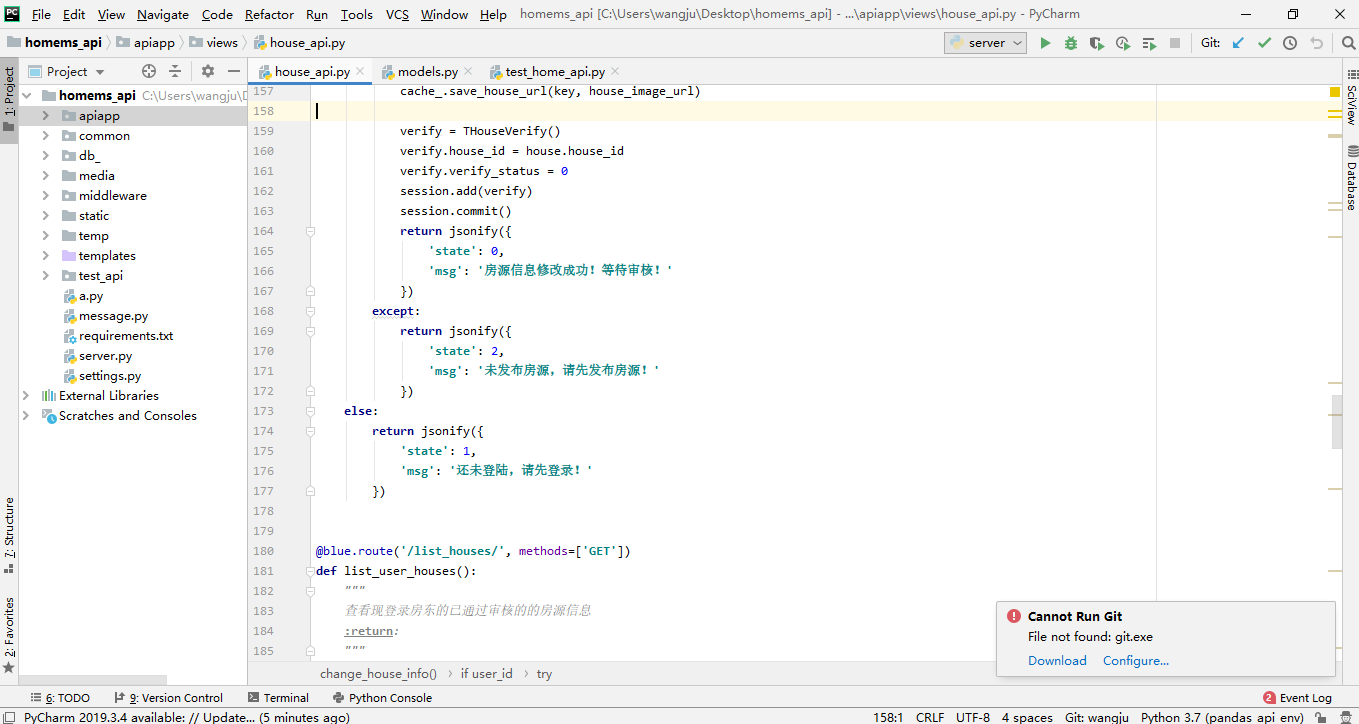
第四章，完成本系统的及数据库设计、前后台的功能实现以及测试；

第五章，总结。

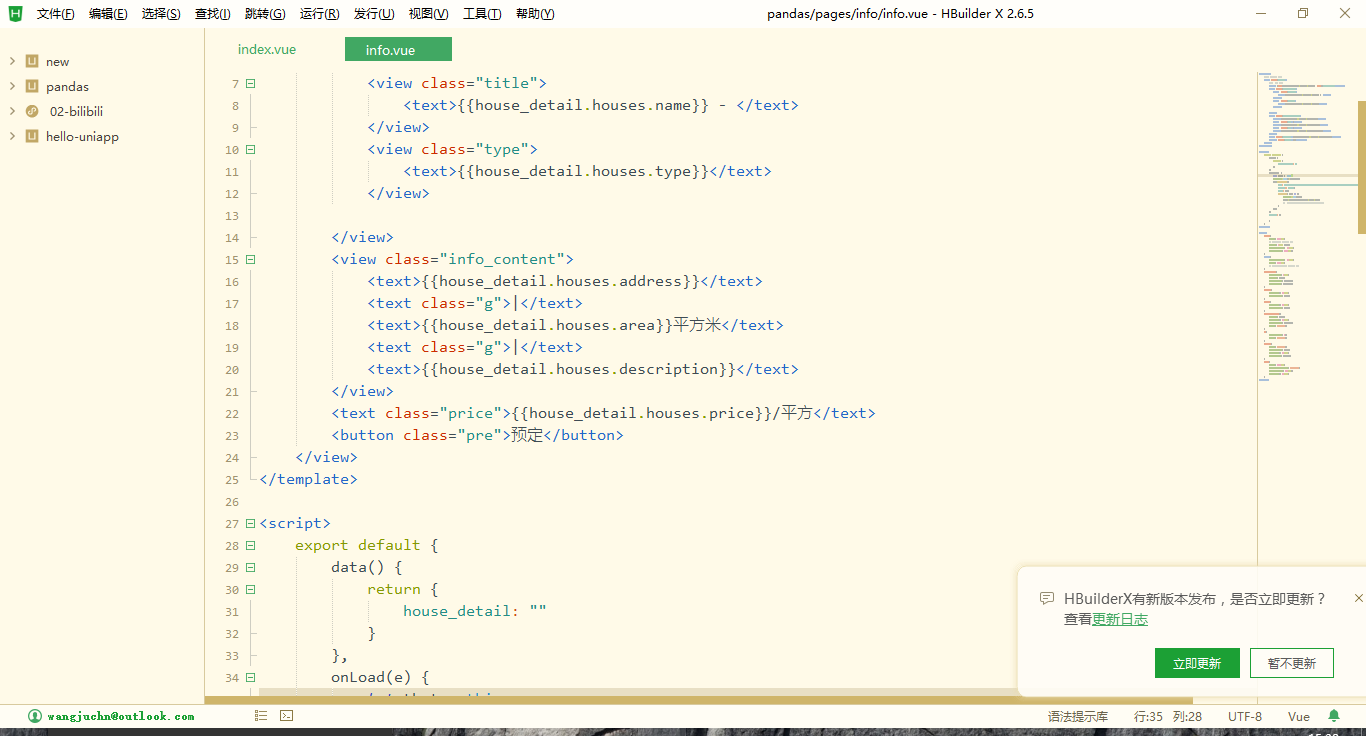
**第二章 开发环境及相关技术介绍**

**2.1 开发环境介绍**

**2.1.1 PyCharm**

PyCharm是由JetBrains打造的一款Python IDE，带有一整套可以帮助用户在使用Python语言开发时提高其效率的工具，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该IDE提供了一些高级功能，以用于支持Django框架下的专业Web开发。截图如下：

**2.1.2 HBuilder X**

HBuilder X是DCloud（数字天堂）专为vue推出的一款Web开发IDE。HBuilder X同时具有着不但轻巧、急速，而且有着相当强大的语法提示功能。HBuilder X可视化的方式比较简单，HBuilder X内置相关环境，开箱即用，无需配置nodejs。HBuilder X是通用的前端开发工具，但为uni-app做了特别强化。截图如下：

**2.2 相关技术介绍**

**2.2.1 Python**

Python是一种跨平台的[计算机程序设计语言](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E8%AF%AD%E8%A8%80/7073760" \t "_blank)，是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。最初被设计用于编写自动化脚本(shell)，随着版本的不断更新和语言新功能的添加，越多被用于独立的、大型项目的开发，同时随着人工智能时代的到来，Python语言的使用范畴也正在迅速扩大，并且，Python也拥有者相当多的第三方库，也可以直接和各种数据库进行交互，极大地简化了程序员的开发成本。

**2.2.2 Web框架**

Python下有许多款不同的 Web 框架，在该系统中主要用到了Django和Flask框架。可以这样理解，如果把Flask比喻成一个毛坯房，内部家具基本没有，然而Django就相当于是一栋别墅，内部家具齐全。因此Django是重量级选手中最有代表性的一位，许多成功的网站和APP都基于Django，Django是一个开放源代码的Web应用框架，是基于Flask用Python写成的，特点则是大而全；相比于Django，Flask框架就显得比较轻量级。

**2.2.3 数据库技术**

数据库简单来说，其实就是数据的仓库，被称为DataBase，简称为DB，指的是长期保存在计算机的存储设备上，按照一定的规则将指定的数据组织起来，可以被各种应用使用的数据集合，其本质就是用于存储数据的容器，与普通的数据容器不同的地方在于：数据库依据数据结构来存储数据，正因为有数据结构的存在，导致数据库中的数据的存储比较条理化或者格式化。市面上的常见的数据库有：Oracle、SQL Server、MySQL、Redis、MongoDB等，

在本系统中主要用到了MySQL和Redis数据库。值得一提的是，这两种数据的差别决定了其不同的用途，首先在类型上MySQL数据库属于关系型数据库，然而Redis属于NoSQL，其次在数据存储位置上有非常大的差别，MySQL数据库将数据存储在硬盘上，然而Redis一般将数据存储于内存中，因此，MySQL一般用于持久化数据，偏向于存数据，然而Redis一般用于对数据进行缓存，偏向于取数据。

**2.2.4 uni-app**

uni-app是由DCloud（数字天堂）开发的可以使用[Vue.js](https://vuejs.org/)开发所有前端应用的框架，开发者编写一套代码，可运行和发布到iOS、Android、H5、以及各种小程序（微信/支付宝/百度/头条/QQ/钉钉）等多个平台，因此，uni-app的产生，极大地降低了开发者的学习成本，一套代码可以多平台转换运行。同时uni-app也引入了众多的第三方接口，如：地图、支付、导航等，极大地增强了开发效率。

**第三章 系统分析**

**3.1 可行性分析**

该系统的可行性分析(Feasibility Analysis)是在对该业务进行了初步分析之后，同时主要结合采用技术的优缺点、经济成本等因素，最终确定该系统的开发方案的方法。它是更深层次的思考该系统采用技术的适配性和经济成本等，从而确定其技术、经济最优的方案。可行性分析是项目开发过程中必须进行的步骤，决定了整个系统的大体方向和产品质量。本节从技术、经济、操作是哪个方面对房屋租赁系统进行了全面分析。

**3.1.1 技术可行性**

在对该系统进行了全面分析之后，用Django做后台管理，Flask前台的想法是完全合适和可行的。Django强大后台管理模块可以非常完美的解决后台管理设计繁琐的问题，且其自带的ORM完美的实现了关系型数据库和Django服务器之间的交互；其次，Flask的灵活多变性也相当方便接口的实现，而且，Flask中的sqlAlchemy插件也完美的解决了Flask服务器和数据库之间的交互问题。

**3.1.2 经济可行性**

经济上考虑的是系统相应的开发成本、维护成本与系统使用中租户和房东之间的关系。该系统开发成本较低，只需要一台云服务器以及短信服务便可以完成项目的开发和部署。另外，该系统的前台前端采用uni-app开发，可以适应更多的平台，极大地降低了开发成本。此外，系统的使用能大大增加房屋成交的机会，极大地提高房东和租客的劳动成本。

**3.1.3 操作可行性**

由于该系统无论前台还是后台服务，都是具有前端界面和后端服务器的，因此，本系统只需要将该后端Django和Flask服务器成功部署到云服务器中，系统管理员便可以直接通过浏览器访问指定URL就可以访问该后台管理服务，而用户便可以便可以直接安装有uni-app转换来的Apk文件便可以，从而非常方便进行各种前台操作。

**3.2 系统需求分析**

系统需求分析，是基于可行性分析基础上的一个重要环节，关系到一个系统设计和实现的成败和软件的运行质量。需求分析的主要目的是界定系统需要满足的功能模块，真实全面的体现用户的真实需求，并将此结果作为系统开发与实现的前提条件，是后续系统设计与开发的重要依据。充分的系统需求分析在较大程度上能够避免或剔除前期错误，增强软件开发效率，降低软件开发成本，提升软件运行质量。结合对城市房屋租赁管理信息系统现状的基本了解和实际调查，存在以下不同的需求。

**3.2.1 系统目标**

通过引入Python和数据库等技术，以国内房屋租赁市场需求研究为基础，将房屋的租赁方、出租方以及各管理部门之间的关系进行逻辑化、信息化，从而提高房屋租赁管理部门的管理效率、业务流程以及决策水平，实现房屋租赁业务流程中的信息共享、信息集成与流程整合。基于Python开发的房屋租赁管理信息平台，能够很方便地提供房源的详细的租赁信息，极大地减少了房屋租赁的交易成本，扩大了房屋租赁的交易量，实现了对房屋租赁关键指标的有效监控，引导着房屋租赁市场朝着健康、繁荣、有序的方向发展。此外，本课题设计与开发的基于Python的房屋租赁管理系统，首先参考了已有房屋租赁信息管理的优点，其次符合国内的房屋租赁管理制度，并满足了操作直观、方便、实用、安全等要求。具体而言，该系统的功能目标包括：具有一个舒适的交互界面；具有较好的权限管理功能；具有迅速、快捷的数据查询功能；具有安全、方便、快捷的交易功能。

**3.2.2 系统功能需求分析**

本课题基于Python设计与开发的房屋租赁系统在充分结合房屋租赁的业务流程、租赁管理部门的实际需求以及租赁市场的信息需求的基础上，将该房屋租赁系统划分为后台管理系统和前台展示系统，其中后台管理中包含管理员登录模块、角色管理模块、用户管理模块、公告管理模块、评论管理模块、投诉管理模块、轮播图管理模块、代金券管理模块等几个主要模块；前台展示主要包含房源展示模块、快速检索模块、定位模块、收藏模块、聊天模块、用户注册模块、登录模块、订单查询模块、升级房东模块、实名认证模块等。下面简要说明各个功能模块的具体需求。

（1）后台管理系统

管理员登录模块的功能是检验登录该后台管理系统的管理员是否合法。每个管理员都有自己的角色，角色主要有：超级管理员、审核管理员和赞助管理员。当用户输入用户名和密码后，如果该用户的身份合法，则可以经过确认后直接进入房屋租赁系统的后台管理系统，并且不同角色的呈现的界面是不一样的。如果该登录的信息是非法的，则系统拒绝该用户的进入。对于超级管理员，可以进行角色的创建、编辑和删除，可以创建不同角色的管理员，同时，超级管理员也是可以进行公告的发布、查看和删除；审核管理员可以对房屋的信息进行审核，其中包括是否实名认证，如果审核通过，则可以在前台进行显示，也可以对公告信息进行审核，通过则任何管理员都可以直接看到该公告信息；此外，赞助管理员则可以直接对代金券和轮播图进行管理，添加的代金券和轮播图可以直接在前台展示。

（2）前台展示系统

在前台展示系统中，首页可以进行轮播图和房源信息的展示，同时在首页中可以通过关键字进行房源搜索，同时，首页中也可以进行定位操作，在首页中也可以进行房源信息的收藏，同时点击对应的房源信息，会跳转到详情页面，可以对该房源进行收藏和预定，其中预定可以采用支付宝和微信支付；在收藏模块中，可以查看当前账户收藏的房源信息，如果还未登录会跳转到登录页面，在登录页面中，输入用户名和密码，如果验证通过，则可以登录成功，返回到首页页面，同时在收藏页面中，也是可以进行预定的；在聊天页面中，可以列出最近30天的聊天记录，点击任意一个聊天记录，便可以打开一个临时会话页面；在我的页面中，除了可以进行基本地用户注册、登录，还可以进行当前用户的实名认证，之后便可以通过后台进行审核，当然，我的页面中，也可以查询订单，点击查询订单便可以将当前用户所有的订单信息列出来，同时，对用普通租客，在我的页面中也可以升级为房东。当然是建立在已经进行实名认证的前提下。

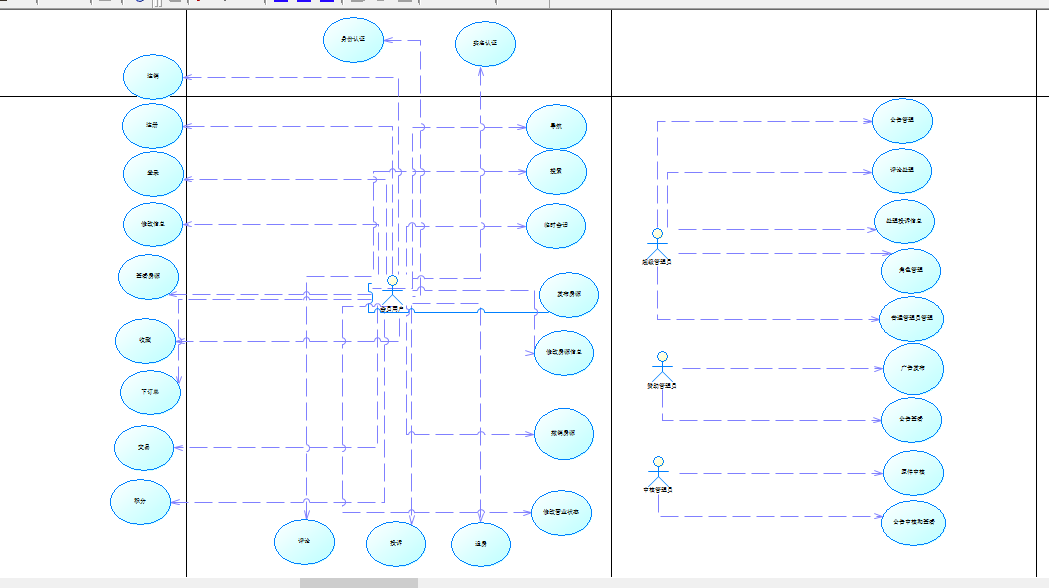
**3.2.3 系统性能需求分析**

**3.3 业务分析**

**3.3.1 用户群体**

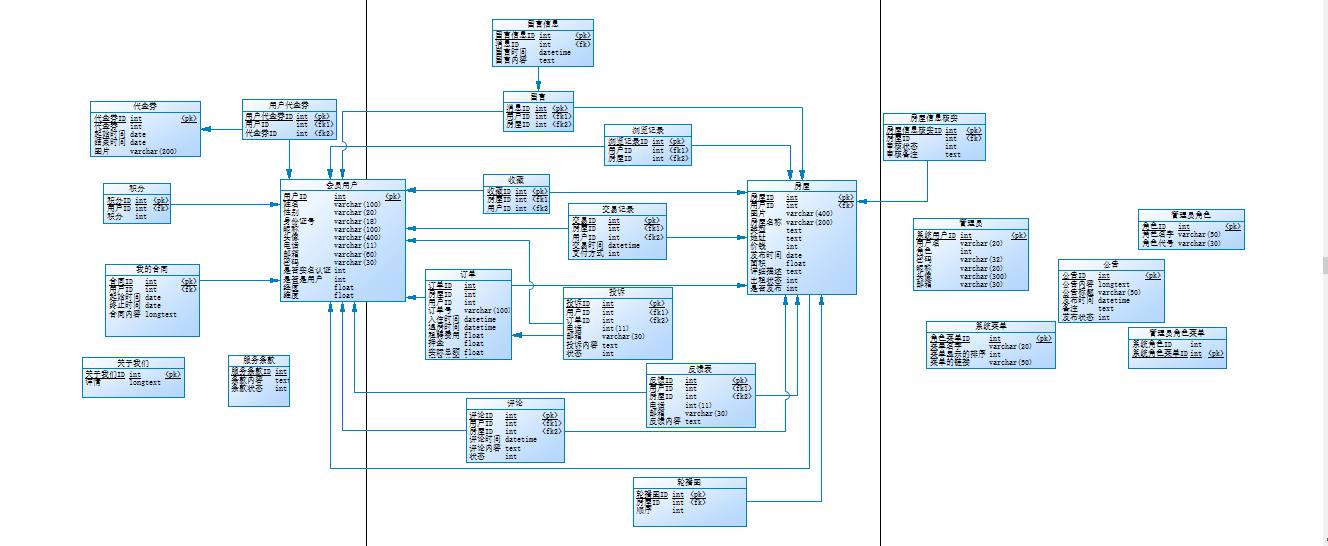
该系统的用户群体主要分前台和后台，前台针对的主要是普通租客和房东，而后台则主要针对的是超级管理员、审核管理员和赞助管理员。

**3.3.2 业务模型图**

对该系统进行了全面的业务分析，并对该系统真的每个群体都进行了分析，并通过PowerDesigner集合分析的结果，画出了该系统的业务用例图，截图如下：

**3.3.3 模型表建立**

根据该系统的业务用例图进行分析，同样通过PowerDesier建出了物理模型图，截图如下：

**第四章 系统的设计与实现**

在系统设计中，究其本质，任何的操作其实都是建立在对数据库的增、删、改、查的基础上的，本文分后台管理系统和前台操作系统来进行详细阐述。

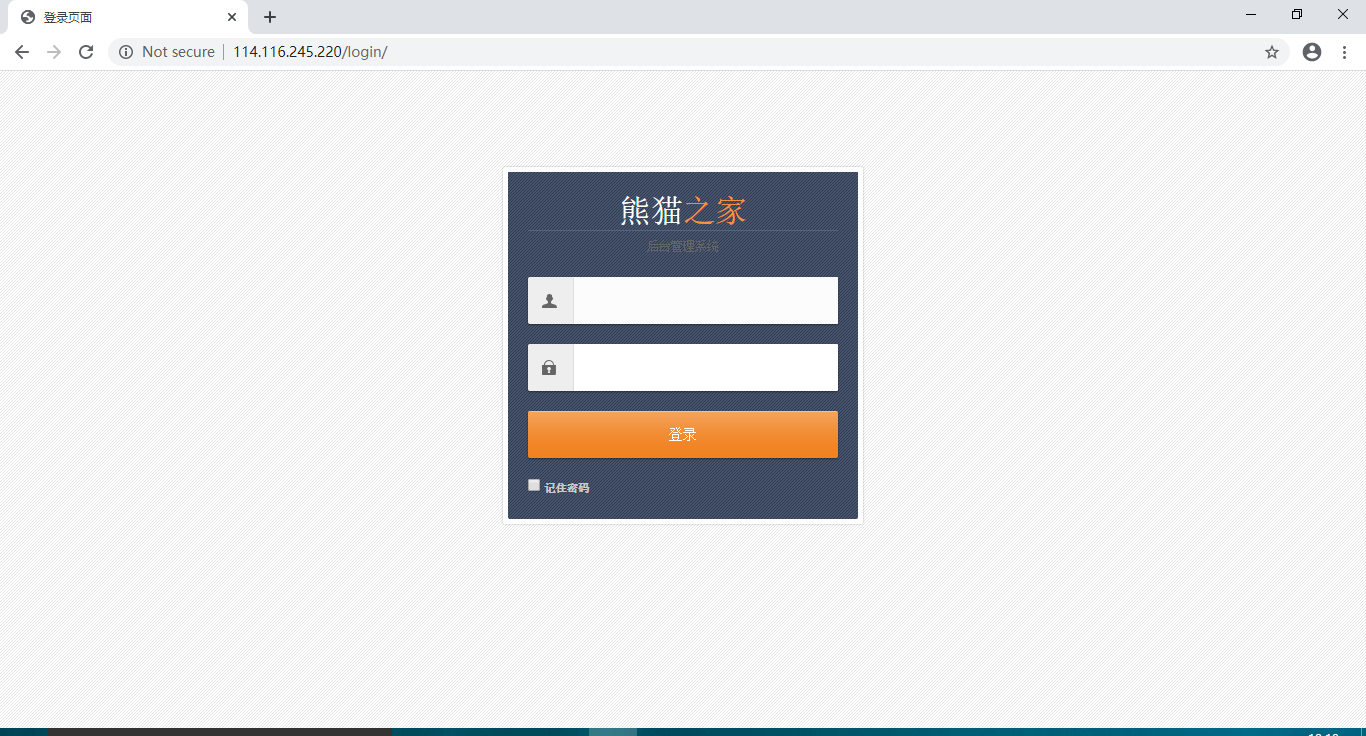
**4.1 后台管理系统**

在后台管理系统中采用了Django中的表单验证功能，从前端向后端传入响应的数据Django表单进行了相应的验证，最终进行相应的操作或跳转。

**4.1.1 登录界面**

在前段通过表单向后端传入用户名和密码，通过后端进行数据的获取和验证，如果传入数据和数据库匹配的话，则登录成功，否则，登陆失败，通过Ajax弹出提醒信息。

同时，如果后端验证登录成功，那么后端会通过Session将登录用户的信息进行保存，方便后续前后端全局进行调用，值得一提的是，在后续的超级管理员、审核管理员、赞助管理员的头像、用户名等信息都是可以通过Session来获取，极大地增强了系统的开发的灵活性。

打开浏览器，输入该系统后台管理登录地址，即出现如下图所示的登录界面：

如果己经在本系统的后台系统中已经注册为用户，只需输入用户名与密码点击“登录”即可进入后台管理系统。

如果不是本系统的用户，请联系系统超级管理员进行系统用户注册，然后才能登录本后台系统。详情请见4.1.2超级管理员详细介绍。

**4.1.2 超级管理员界面**

超级管理员的权限有角色管理、系统用户管理、评论管理、投诉管理以及公告的发布和删除，但却并不能进行公告的审核。其中的角色管理、系统用户管理、评论管理、投诉管理主要就是对数据库中的响应表进行增删改查，然而，对于超级管理员，只能对公告表进行增加和删除的操作。

在浏览器中通过登录超级管理员之后，可以看到如下图所示界面：



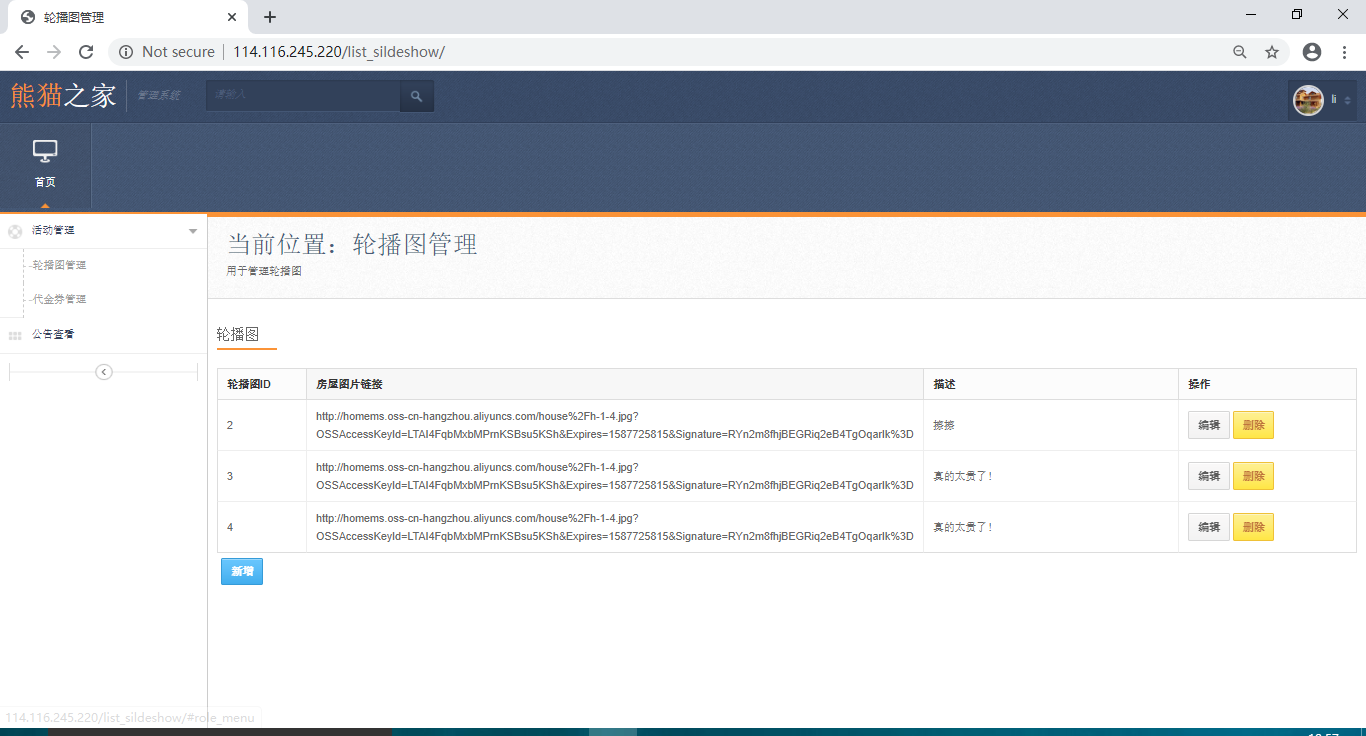
**4.1.3 审核管理员界面**

对于审核管理员，其权限就比较少了，完全是和各种信息审核相关了，有房屋审核、公告审核，同时也可以查看审核已经通过的公告信息。房屋审核和公告审核其实就是更改数据库中房屋表和公告表的某一字段值。

如果通过登录界面登录的是审核管理员，成功之后，便会到达如下界面：

在房屋审核的过程中，如果点击审核不通过，那么审核界面会弹出一个对话窗，可以填写未通过的理由。

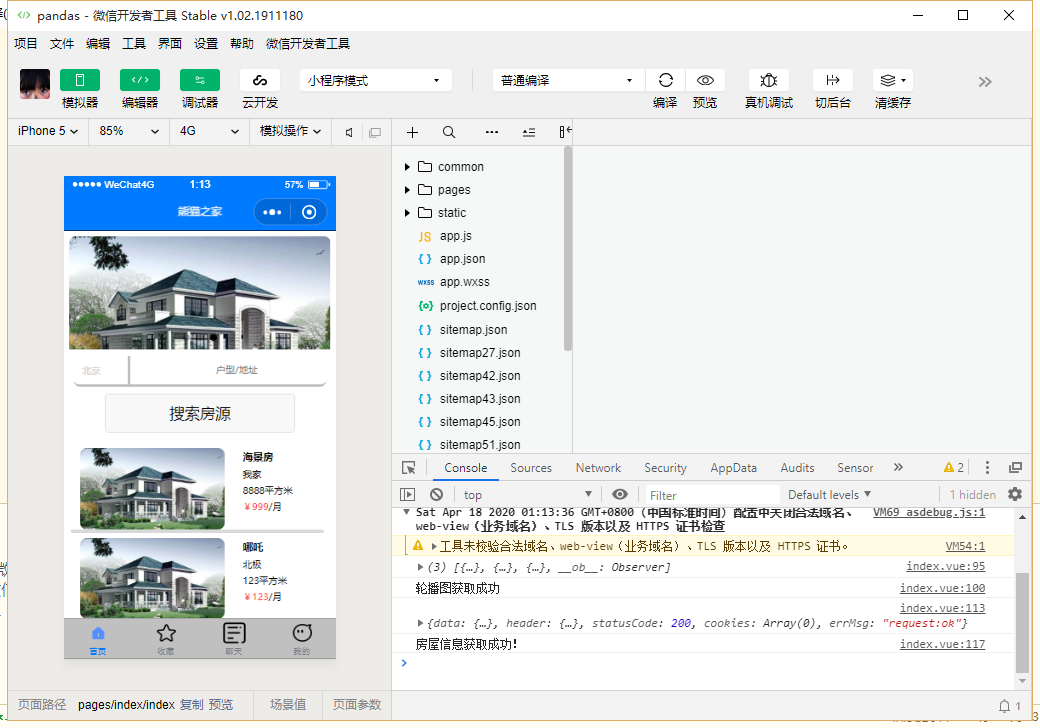
**4.1.4 赞助管理员界面**

最后，对于赞助管理员，其界面就比较简单了，如下所示：

由上图可以非常清楚的看到，赞助管理员的权限主要就是广告的管理（轮播图管理）和代金券的管理，即可以对轮播图和代金券表中的信息进行发布、编辑以及删除操作，最终呈现给用户。

**4.2前台操作系统**

**4.2.1 首页模块**

首页主要是广告、房源信息的展示，本质其实就是将广告、房源信息从数据库查询出来，然后通过uni-app进行展示，首页如下图所示：

**4.2.2 收藏模块**

收藏页面做的其实就是收藏的房屋信息进行展示出来，就是对收藏表的查询操作，然后通过瀑布流的方式进行展示，同时，在该页面中，用户是可以对所收藏的房源信息进行预订和删除的。

**4.2.3 消息模块**

通过消息模块，用户可与房东进行线上交流，主要采用的是UDP协议，建立在非连接的基础上完成，无论房东或者租客是否在线，都可以向对方发送消息，首先通过Socket编程完成服务端、客户端的创建，便可完成在非连接的基础上进行消息通信的目标。

**4.2.4 我的模块**

在我的模块里面，用户可以进行登录/注册、实名认证、订单查询、以及修改房源信息等。这些操作本质上都是对数据表的增、删、改、查，然而值得一提的是，实名认证由于引入了第三方技术，因此其实是通过一定的规则，修改用户表中的某些字段。

**第五章 测试**

**总结**

本文在认真的市场需求调研和选用最合适的开发技术的基础上，分别从系统整体设计目标、模块、功能、业务流程等几个方面考虑，设计出了一套开发成本最低的设计框架。设计并开发了一套基于Python的房屋租赁管理系统。本文所做的主要工作如下：

分析和研究了我国房屋租赁市场的需求和研究现状，结合目前市面上已经较为成熟的房屋租赁平台和失败的案例，制定出了最为合适的开发思路。

更加深入研究和分析了Python各个Web框架的优缺点以及适用场景，选择了最合适的技术处理相应的问题。

本系统在大的方向上，主要分为2个设计方向：前台系统和后台系统，同时，无论是前台和后台系统都包括有客户端和服务端。在此基础上再结合数据库技术开发出来的房屋租赁系统在功能上将更加的完善，并且，在用户使用上，使用感受也将更加的轻松，最重要的是，在房屋信息的管理上，也更加的安全和方便。

该系统一大特色就是，前台完全采用uni-app小程序开发，可以适配各种移动端和PC端，实用性得到了极大的提高，而且也达到了“秒开”的效果。并且极大地节省了开发的成本。

**参考文献**

[1]张旭超. 基于B/S架构的房屋租赁管理系统的设计与实现[D].北京工业大学,2015.

[2]陈萍. 基于B/S的房屋中介管理信息系统的分析与设计[D].云南大学,2012.

[3]胡世民. 基于互联网+的房屋租赁管理系统的设计与实现[D].江西财经大学,2018.

[4]李谊. 城市房屋租赁管理信息系统设计与实现[D].电子科技大学,2012.

[5]赵迎芳. 基于J2EE的房屋租赁管理信息系统设计与实现[D].河北科技大学,2014.

[6]朱园园,邵雯娟,黄志远.基于Scrapy的房屋租赁信息搜索系统设计与实现[J].电脑与电信,2019(06):14-19.

[7]武海龙,李国平.基于SpringBoot的房屋租赁系统设计[J].电脑与信息技术,2019,27(03):76-78.

[8]董莹. 基于用户意向分析的房屋租赁系统的设计与实现[D].大连海事大8,2017.

[9]张基.基于Android平台的房屋租赁管理系统的设计[J].老字号品牌营销,2019(07):70-71.

[10]李佳雯. 基于区块链技术的房屋租赁平台设计与实现[D].济南大学,2019.

[11]王飞.论房屋租赁信息系统的设计与实现[J].电脑迷,2017(07):194-195.

[12]Ran Liu,Tingzhu Li,Richard Greene. Migration and inequality in rental housing: Affordability stress in the Chinese cities[J]. Applied Geography,2020,115.

[13]Wei LI. Design and Implementation of the Management System of House Lease[C]. Science And Engineering Research Center.Proceedings of 2016 International Conference on Wireless Communication and Network Engineering (WCNE2016).Science And Engineering Research Center:Science and Engineering Research Center,2016:137-141.

**附录**

**后台系统代码**

**models.py**

*# This is an auto-generated Django model module.  
# You'll have to do the following manually to clean this up:  
# \* Rearrange models' order  
# \* Make sure each model has one field with primary\_key=True  
# \* Make sure each ForeignKey has `on\_delete` set to the desired behavior.  
# \* Remove `managed = False` lines if you wish to allow Django to create, modify, and delete the table  
# Feel free to rename the models, but don't rename db\_table values or field names.***from** datetime **import** datetime  
  
**from** django.db **import** models  
**from** tinymce.models **import** HTMLField  
  
**from** common **import** md5\_  
  
**class** TUser(models.Model):  
 user\_id = models.AutoField(primary\_key=**True**)  
 name = models.CharField(max\_length=100, blank=**True**, null=**True**)  
 sex = models.CharField(max\_length=20, blank=**True**, null=**True**)  
 identity\_number = models.CharField(max\_length=18, blank=**True**, null=**True**)  
 nickname = models.CharField(max\_length=100, blank=**True**, null=**True**)  
 img = models.CharField(max\_length=200, blank=**True**, null=**True**)  
 phone = models.CharField(max\_length=11, blank=**True**, null=**True**)  
 email = models.CharField(max\_length=60, blank=**True**, null=**True**)  
 password = models.CharField(max\_length=30, blank=**True**, null=**True**)  
 has\_real\_name = models.IntegerField(blank=**True**, null=**True**)  
 is\_member = models.IntegerField(blank=**True**, null=**True**)  
 longitude = models.FloatField(blank=**True**, null=**True**)  
 dimension = models.FloatField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_user'  
  
class** Contract(models.Model): *# 合同模型* t\_contract\_id = models.AutoField(primary\_key=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 start\_time = models.DateField(blank=**True**, null=**True**)  
 stop\_time = models.DateField(blank=**True**, null=**True**)  
 content = models.TextField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'contract'  
  
  
class** TBrowsingHistory(models.Model):  
 browsing\_history\_id = models.AutoField(primary\_key=**True**)  
 usert = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 house = models.ForeignKey(**'THouse'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_browsing\_history'  
  
  
class** TChatmsg(models.Model): *# 留言* chatmsg\_id = models.AutoField(primary\_key=**True**)  
 msg = models.ForeignKey(**'TMessage'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 chatmsg\_time = models.DateTimeField(blank=**True**, null=**True**)  
 content = models.TextField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_chatmsg'  
  
  
class** TComment(models.Model): *#* comment\_id = models.AutoField(primary\_key=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 house = models.ForeignKey(**'THouse'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 create\_time = models.DateTimeField(blank=**True**, null=**True**)  
 content = models.TextField(blank=**True**, null=**True**)  
  
 states = (  
 (0, **'审核中'**),  
 (1, **'已通过'**),  
 (2, **'未通过'**),  
 )  
 state = models.IntegerField(blank=**True**, default=0)  
  
 @property  
 **def** state\_label(self):  
 **return** self.states[self.state][-1]  
  
 **def** save(self, force\_insert=**False**, force\_update=**False**, using=**None**,  
 update\_fields=**None**):  
 **if** self.create\_time **is None**:  
 self.create\_time = datetime.now()  
  
 super(TComment, self).save()  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_comment'  
  
class** TComplaint(models.Model):  
 fadeback\_id = models.AutoField(primary\_key=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 order = models.ForeignKey(**'TOrder'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 phone = models.IntegerField(blank=**True**, null=**True**)  
 email = models.CharField(max\_length=30, blank=**True**, null=**True**)  
 content = models.TextField(blank=**True**, null=**True**)  
  
 states = (  
 (0, **'审核中'**),  
 (1, **'已通过'**),  
 (2, **'未通过'**),  
 )  
 state = models.IntegerField(blank=**True**, default=0)  
  
 @property  
 **def** state\_label(self):  
 **return** self.states[self.state][-1]  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_complaint'  
  
  
class** TFavorite(models.Model):  
 favorite\_id = models.AutoField(primary\_key=**True**)  
 house = models.ForeignKey(**'THouse'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_favorite'  
  
  
class** TFeedback(models.Model):  
 fadeback\_id = models.AutoField(primary\_key=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 house = models.ForeignKey(**'THouse'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 phone = models.IntegerField(blank=**True**, null=**True**)  
 email = models.CharField(max\_length=30, blank=**True**, null=**True**)  
 content = models.TextField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_feedback'  
  
  
  
class** THouse(models.Model):  
 house\_id = models.AutoField(primary\_key=**True**)  
 user = models.ForeignKey(TUser, models.DO\_NOTHING, blank=**True**, null=**True**)  
 image = models.CharField(max\_length=200, blank=**True**, null=**True**)  
 *# image = models.ImageField()* name = models.CharField(max\_length=200, blank=**True**, null=**True**)  
 type = models.TextField(blank=**True**, null=**True**)  
 address = models.TextField(blank=**True**, null=**True**)  
 price = models.IntegerField(blank=**True**, null=**True**)  
 publish\_time = models.DateField(blank=**True**, null=**True**)  
 area = models.FloatField(blank=**True**, null=**True**)  
 description = models.TextField(blank=**True**, null=**True**)  
 sale\_status = models.IntegerField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_house'  
  
  
  
class** THouseVerify(models.Model): *# 房屋审核模型* verify\_id = models.AutoField(primary\_key=**True**)  
 house = models.ForeignKey(THouse, models.DO\_NOTHING, blank=**True**, null=**True**)  
 *# verify\_status = models.IntegerField(blank=True, null=True)* remarks = models.TextField(blank=**True**, null=**True**)  
 states = (  
 (0, **'审核中'**),  
 (1, **'已通过'**),  
 (2, **'未通过'**)  
 )  
 verify\_status = models.IntegerField(choices=states, default=0)  
  
 @property  
 **def** state\_label(self):  
 **return** self.states[self.verify\_status][-1]  
  
 **def** save(self, force\_insert=**False**, force\_update=**False**, using=**None**,  
 update\_fields=**None**):  
 *# if self.create\_time is None:  
 # self.create\_time = datetime.now()* super(THouseVerify, self).save()  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_house\_verify'  
  
  
class** TLuckyTicket(models.Model):  
 lucky\_ticket\_id = models.AutoField(primary\_key=**True**)  
 money = models.IntegerField(blank=**True**, null=**True**)  
 begin\_time = models.DateField(blank=**True**, null=**True**)  
 end\_time = models.DateField(blank=**True**, null=**True**)  
 image = models.CharField(max\_length=20, blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_lucky\_ticket'  
  
class** TMessage(models.Model):  
 msg\_id = models.AutoField(primary\_key=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 house = models.ForeignKey(THouse, models.DO\_NOTHING, blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_message'  
  
  
class** TOrder(models.Model):  
 order\_id = models.AutoField(primary\_key=**True**)  
 house = models.ForeignKey(THouse, models.DO\_NOTHING, blank=**True**, null=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 order\_number = models.CharField(max\_length=16, blank=**True**, null=**True**)  
 enter\_time = models.TimeField(blank=**True**, null=**True**)  
 exit\_time = models.TimeField(blank=**True**, null=**True**)  
 hire\_price = models.FloatField(blank=**True**, null=**True**)  
 cash\_price = models.FloatField(blank=**True**, null=**True**)  
 total = models.FloatField(blank=**True**, null=**True**)  
 order\_status = models.IntegerField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_order'  
  
  
class** TPanda(models.Model):  
 panda\_id = models.AutoField(primary\_key=**True**)  
 detail\_content = models.TextField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_panda'  
  
  
class** TPublicNotice(models.Model):  
 public\_notice\_id = models.AutoField(primary\_key=**True**)  
 content = HTMLField(null=**True**)  
 public\_title = models.CharField(max\_length=50, null=**True**)  
 public\_time = models.DateTimeField(auto\_created=**True**, blank=**True**)  
 public\_remarks = models.TextField(null=**True**, blank=**True**)  
  
 states = (  
 (0, **'待审核'**),  
 (1, **'已通过'**),  
 (2, **'未通过'**)  
 )  
 public\_status = models.IntegerField(choices=states, default=0)  
  
 @property  
 **def** state\_label(self):  
 **return** self.states[self.public\_status][-1]  
  
 **def** save(self, force\_insert=**False**, force\_update=**False**, using=**None**,  
 update\_fields=**None**):  
  
 **if** self.public\_time **is None**:  
 self.public\_time = datetime.now()  
  
 super(TPublicNotice, self).save()  
  
 **class** Meta:  
 *# managed = False* db\_table = **'t\_public\_notice'** ordering = [**'-public\_time'**]  
  
  
**class** TScore(models.Model):  
 score\_id = models.AutoField(primary\_key=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 score = models.IntegerField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_score'  
  
  
class** TService(models.Model):  
 service\_id = models.AutoField(primary\_key=**True**)  
 content = models.TextField(blank=**True**, null=**True**)  
 service\_status = models.IntegerField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_service'  
  
  
class** TSlidesshow(models.Model):  
 slidesshow\_id = models.AutoField(primary\_key=**True**)  
 house = models.ForeignKey(THouse, models.DO\_NOTHING, blank=**True**, null=**True**)  
 ord = models.IntegerField()  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_slidesshow'  
  
  
class** TSysMenu(models.Model):  
 menu\_id = models.AutoField(primary\_key=**True**)  
 menu\_name = models.CharField(max\_length=20, blank=**True**, null=**True**)  
 ord = models.IntegerField(blank=**True**, null=**True**)  
 url = models.CharField(max\_length=50, blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_sys\_menu'  
  
  
class** TSysRole(models.Model):  
 role\_id = models.AutoField(primary\_key=**True**)  
 role\_name = models.CharField(max\_length=20)  
 role\_code = models.CharField(max\_length=10)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_sys\_role'  
  
  
class** TSysRoleMenu(models.Model):  
 role\_id = models.IntegerField(blank=**True**, null=**True**)  
 sys\_menu\_id = models.AutoField(primary\_key=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_sys\_role\_menu'  
  
  
class** TSysUser(models.Model):  
 user\_id = models.AutoField(primary\_key=**True**)  
 username = models.CharField(max\_length=20, blank=**True**, null=**True**)  
 role\_id = models.IntegerField(blank=**True**, null=**True**)  
 password = models.CharField(max\_length=32, blank=**True**, null=**True**)  
 nick\_name = models.CharField(max\_length=20, blank=**True**, null=**True**)  
 head = models.ImageField(upload\_to=**'upload\_head/%Y/%m'**, null=**True**, blank=**True**)  
 *# head = models.CharField(max\_length=300, null=True, blank=True)* email = models.CharField(max\_length=30, blank=**True**, null=**True**)  
  
 @property  
 **def** role(self):  
 **return** TSysRole.objects.get(role\_id=self.role\_id)  
  
 **def** save(self, force\_insert=**False**, force\_update=**False**, using=**None**,  
 update\_fields=**None**):  
  
 **if** len(self.password) != 32:  
 self.password = md5\_.hash\_encode(self.password)  
  
 super(TSysUser, self).save()  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_sys\_user'  
  
  
class** TTradingrecord(models.Model):  
 tradingrecord\_id = models.AutoField(primary\_key=**True**)  
 house = models.ForeignKey(THouse, models.DO\_NOTHING, blank=**True**, null=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 payment\_date = models.DateTimeField(blank=**True**, null=**True**)  
 payment\_type = models.IntegerField(blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_tradingrecord'  
  
  
class** TULuckyTicket(models.Model):  
 u\_lucky\_ticketid = models.AutoField(primary\_key=**True**)  
 user = models.ForeignKey(**'TUser'**, models.DO\_NOTHING, blank=**True**, null=**True**)  
 lucky\_ticket = models.ForeignKey(TLuckyTicket, models.DO\_NOTHING, blank=**True**, null=**True**)  
  
 **class** Meta:  
 managed = **False** db\_table = **'t\_u\_lucky\_ticket'**

**views.py**

**import** json  
  
**from** django.core.serializers **import** serialize  
**from** django.db.models **import** Q  
**from** django.http **import** HttpResponse  
**from** django.shortcuts **import** render, redirect  
**from** django.urls **import** reverse  
**from** django.views **import** View  
  
**from** common **import** md5\_  
**from** . **import** get\_house\_image, get\_user\_image  
**from** .models **import** TSysRole, TSysUser, TPublicNotice, TSlidesshow, TLuckyTicket, THouseVerify  
  
  
*# Create your views here.***def** login(request):  
 *# 分两种用户，一个是会员，一个管理员（系统）  
 # print('--->', request.method)* **if** request.method == **'POST'**:  
 *# print(request.POST)* error = **None** username = request.POST[**'username'**].strip()  
 password = request.POST[**'password'**].strip()  
 remeber = request.POST.get(**'remeber'**, **''**) *# checkbox* password\_ = md5\_.hash\_encode(password) *# 转成md5后的密文* print(password\_)  
  
 *# 验证用户名和口令是否为空* **if not** all((username, password)):  
 error = **f'用户名或口令不能为空！'  
  
 else**:  
 login\_user = TSysUser.objects.filter(username=username, password=password\_).first()  
 **if** login\_user:  
 *# 超级管理员* role\_ = login\_user.role  
  
 login\_info = {  
 **'user\_id'**: login\_user.user\_id,  
 **'head'**: str(login\_user.head),  
 **'email'**: login\_user.email,  
 **'nick\_name'**: login\_user.nick\_name,  
 **'role\_name'**: role\_.role\_name,  
 **'role\_code'**: role\_.role\_code,  
 }  
  
 **else**:  
 error = **f'{**username**} 用户名或口令错误！'  
  
 if not** error:  
 request.session[**'login\_user'**] = login\_info  
 **return** redirect(reverse(**'main:dash'**))  
 **return** render(request, **'login.html'**, locals())  
  
  
**def** dashboard(request):  
 *"""  
 登陆之后主页界面* **:param** *request:* **:return***:  
 """* notices = TPublicNotice.objects.filter(public\_status=1)  
 **return** render(request, **'dashboard.html'**, locals())  
  
  
**def** role(request):  
 *"""  
 查看角色页面和删除角色* **:param** *request:* **:return***:  
 """* action = request.GET.get(**'action'**, **''**)  
 **if** action == **'del'**:  
 role\_id = request.GET.get(**'role\_id'**)  
 TSysRole.objects.get(role\_id=role\_id).delete()  
 user\_list = TSysUser.objects.filter(role\_id=role\_id)  
 **for** obj **in** user\_list:  
 obj.delete()  
  
 roles = TSysRole.objects.all()  
 **return** render(request, **'role/list.html'**, locals()) *#***def** list\_sys\_user(request):  
 *"""  
 查看所有管理员和删除管理员* **:param** *request:* **:return***:  
 """* action = request.GET.get(**'action'**, **''**)  
 **if** action == **'del'**:  
 TSysUser.objects.get(user\_id=request.GET.get(**'user\_id'**)).delete()  
  
 *# 查询系统时，除去超级管理员的用户* users = TSysUser.objects.filter(~Q(user\_id=request.session[**'login\_user'**][**'user\_id'**])).all()  
 **return** render(request, **'sys\_user/list.html'**, locals())  
  
  
**class** EditRoleView(View): *# 编辑角色信息  
 """  
 编辑角色信息和增加角色信息  
 """* **def** get(self, request):  
 role\_id = request.GET.get(**'role\_id'**, **''**)  
 **if** role\_id:  
 role = TSysRole.objects.get(role\_id=role\_id)  
 **return** render(request, **'role/edit.html'**, locals())  
  
 **def** post(self, request):  
 **from** .forms **import** RoleForm  
 role\_id = request.POST.get(**'role\_id'**, **''**)  
 **if** role\_id:  
 form = RoleForm(request.POST, instance=TSysRole.objects.get(role\_id=role\_id))  
 **else**:  
 form = RoleForm(request.POST)  
 **if** form.is\_valid():  
 form.save()  
 **return** redirect(reverse(**'main:role'**))  
  
 errors = json.loads(form.errors.as\_json())  
 **return** render(request, **'role/edit.html'**, locals())  
  
  
**class** EditSysUserView(View):  
 *"""  
 编辑管理员信息和增加管理员  
 """* **def** get(self, request):  
 user\_id = request.GET.get(**'user\_id'**, **''**)  
 **if** user\_id:  
 obj = TSysUser.objects.get(user\_id=user\_id)  
  
 roles = TSysRole.objects.filter(~Q(role\_code=**'admin'**))  
 **return** render(request, **'sys\_user/edit.html'**, locals())  
  
 **def** post(self, request):  
 **from** .forms **import** SysUserForm  
 *# print(request.POST,"+++++++++++++++++++++")* user\_id = request.POST.get(**'user\_id'**, **''**)  
 **if** user\_id:  
 form = SysUserForm(request.POST, request.FILES, instance=TSysUser.objects.get(user\_id=user\_id))  
 **else**:  
 form = SysUserForm(request.POST, request.FILES)  
 **if** form.is\_valid():  
 form.save()  
 **return** redirect(reverse(**'main:list\_sysuser'**))  
  
 errors = json.loads(form.errors.as\_json())  
  
 roles = TSysRole.objects.filter(~Q(code=**'admin'**))  
  
 **return** render(request, **'sys\_user/edit.html'**, locals())  
  
  
**def** logout(request):  
 *"""  
 注销* **:param** *request:* **:return***:  
 """* **del** request.session[**'login\_user'**]  
 **return** redirect(reverse(**"main:login"**))  
  
  
**def** notice(request):  
 *"""  
 公告展示* **:param** *request:* **:return***:  
 """* notices = TPublicNotice.objects.all()  
 action = request.GET.get(**'action'**, **''**)  
 **if** action:  
 TPublicNotice.objects.get(public\_notice\_id=request.GET.get(**'public\_notice\_id'**)).delete()  
 **return** render(request, **'notice/list.html'**, locals())  
  
  
**class** EditPublicNotice(View):  
 *# 编辑公告* **def** get(self, request):  
 public\_notice\_id = request.GET.get(**'public\_notice\_id'**, **''**)  
 **if** public\_notice\_id:  
 notice = TPublicNotice.objects.get(public\_notice\_id=public\_notice\_id)  
 **return** render(request, **'notice/edit.html'**, locals())  
  
 **def** post(self, request):  
 **from** .forms **import** NoticeForm  
  
 public\_notice\_id = request.POST.get(**'public\_notice\_id'**, **''**)  
 **if** public\_notice\_id:  
 notice = TPublicNotice.objects.get(public\_notice\_id=public\_notice\_id)  
 form = NoticeForm(request.POST, instance=notice)  
  
 **else**:  
 form = NoticeForm(request.POST)  
  
 **if** form.is\_valid():  
 form.save()  
 **return** redirect(reverse(**'main:notice'**))  
  
 errors = json.loads(form.errors.as\_json())  
 **return** render(request, **'notice/edit.html'**, locals())  
  
  
**def** list\_slide\_show(request):  
 *"""  
 轮播图展示* **:param** *request:* **:return***:  
 """* action = request.GET.get(**'action'**, **''**)  
 **if** action == **'del'**:  
 TSlidesshow.objects.get(pk=request.GET.get(**'id\_'**)).delete()  
 slides = TSlidesshow.objects.all()  
 **for** obj **in** slides:  
 h\_key = obj.house.image  
 h\_image\_url = get\_house\_image(h\_key)  
 obj.house.image = h\_image\_url  
  
 **return** render(request, **'sys\_user/list\_sildeshow.html'**, locals())  
  
  
**class** EditSlideWhowView(View):  
 *# 编辑轮播图* **def** get(self, request):  
 id\_ = request.GET.get(**'id\_'**, **''**)  
 **if** id\_:  
 obj = TSlidesshow.objects.get(pk=id\_)  
 h\_key = obj.house.image  
 h\_image\_url = get\_house\_image(h\_key)  
 **return** render(request, **'sys\_user/edit\_slide.html'**, locals())  
  
 **def** post(self, request):  
 **from** .forms **import** SlideForm  
 id\_ = request.POST.get(**'id'**, **''**)  
 **if** id\_:  
 form = SlideForm(request.POST, instance=TSlidesshow.objects.get(pk=id\_))  
 **else**:  
 form = SlideForm(request.POST)  
  
 **if** form.is\_valid():  
 form.save()  
 **return** redirect(**'/list\_sildeshow/'**)  
  
 errors = json.loads(form.errors.as\_json())  
  
 **return** render(request, **'sys\_user/edit\_slide.html'**, locals())  
  
  
**def** list\_lucky(request):  
 *"""  
 代金券展示* **:param** *request:* **:return***:  
 """* action = request.GET.get(**'action'**, **''**)  
 **if** action == **'del'**:  
 TLuckyTicket.objects.get(pk=request.GET.get(**'id\_'**)).delete()  
 tickets = TLuckyTicket.objects.all()  
 **return** render(request, **'sys\_user/list\_lucky.html'**, locals())  
  
  
**class** EditLuckyView(View):  
 *"""代金券管理"""* **def** get(self, request):  
 id\_ = request.GET.get(**'id\_'**, **''**)  
 **if** id\_:  
 obj = TLuckyTicket.objects.get(pk=id\_)  
  
 **return** render(request, **'sys\_user/edit\_lucky.html'**, locals())  
  
 **def** post(self, request):  
 **from** .forms **import** LuckyForm  
 id\_ = request.POST.get(**'id'**, **''**)  
 begin = request.POST.get(**'begin\_time'**)  
 print(**f'------->>>{**begin**}---{**type(begin)**}'**)  
 **if** id\_:  
 form = LuckyForm(request.POST, instance=TLuckyTicket.objects.get(pk=id\_))  
 **else**:  
 form = LuckyForm(request.POST)  
  
 **if** form.is\_valid():  
 form.save()  
 **return** redirect(**'/lucky\_ticket/'**)  
  
 errors = json.loads(form.errors.as\_json())  
  
 **return** render(request, **'sys\_user/edit\_lucky.html'**, locals())  
  
  
**class** AuditMessage(View):  
 *# 房屋审核* **def** get(self, request):  
 action = request.GET.get(**'action'**, **''**)  
 **if** action:  
 obj = THouseVerify.objects.get(pk=request.GET.get(**'id\_'**))  
  
 **if** action == **'yes'**:  
 obj.verify\_status = 1  
 **elif** action == **'no'**:  
 obj.verify\_status = 2  
 obj.remarks = request.GET.get(**'remarks'**, **''**)  
 obj.save()  
 obj.full\_clean()  
  
  
 objs = THouseVerify.objects.filter(verify\_status=0).all()  
 **for** obj **in** objs:  
 h\_key = obj.house.image  
 h\_image\_url = get\_house\_image(h\_key)  
 obj.house.image = h\_image\_url  
 u\_key = obj.house.user.img  
 u\_image\_url = get\_user\_image(u\_key)  
 obj.house.user.img = u\_image\_url  
 **return** render(request, **'message/list\_audit.html'**, locals())  
  
  
**class** TPublic(View):  
 *"""公告管理"""* **def** get(self, request):  
 action = request.GET.get(**'action'**, **''**)  
 public\_status = request.GET.get(**'public\_status'**,**''**)  
 **if** action:  
 obj = TPublicNotice.objects.get(pk=request.GET.get(**'id\_'**))  
 **if** action == **'yes'**:  
 obj.public\_status = 1  
 **elif** action == **'no'**:  
 obj.public\_status = 2  
 obj.public\_remarks = request.GET.get(**'public\_remarks'**, **''**)  
 obj.save()  
 obj.full\_clean()  
  
 **if** public\_status:  
 objs = TPublicNotice.objects.filter(public\_status=1).all()  
 **else**:  
 objs = TPublicNotice.objects.filter(public\_status=0).all()  
 **return** render(request, **'message/list\_public.html'**, locals())  
  
  
**from** mainapp.models **import** TComplaint  
**def** get\_complain(request):  
 *# get请求参数：action 有三个可选值（yes, del, query)  
 # yes和del 都有一个额外参数：id\_(投诉的ID）  
 # query 有一个参数wd:可以按照订单号或者用户账号、手机号搜索都行* action = request.GET.get(**'action'**, **''**)  
 **if** action==**'yes'**:  
 obj = TComplaint.objects.get(pk=request.GET.get(**'id\_'**))  
 obj.state = 1  
 obj.save()  
 obj.full\_clean()  
 **if** action==**'del'**:  
 TComplaint.objects.get(pk=request.GET.get(**'id\_'**)).delete()  
 *# 查看评论主页* complains = TComplaint.objects.all()  
 **return** render(request, **'complaint/list.html'**, locals())  
  
  
**from** mainapp.models **import** TComment  
**def** get\_comment(request):  
 *# get请求参数：action 有三个可选值（yes, del, query)  
 # yes和del 都有一个额外参数：id\_(投诉的ID）  
 # query 有一个参数wd:可以按照订单号或者用户账号、手机号搜索都行* action = request.GET.get(**'action'**, **''**)  
 **if** action == **'yes'**:  
 obj = TComment.objects.get(pk=request.GET.get(**'id\_'**))  
 obj.state = 1  
 obj.save()  
 obj.full\_clean()  
 **if** action == **'del'**:  
 TComment.objects.get(pk=request.GET.get(**'id\_'**)).delete()  
 comments = TComment.objects.all()  
 **return** render(request, **'comment/list.html'**, locals())

**forms.py**

*#!/usr/bin/python3  
# coding: utf-8  
  
"""  
用来对数据进行检验  
  
"""***from** django **import** forms  
**from** .models **import** TSysRole, TSysUser, TPublicNotice, TLuckyTicket, TSlidesshow  
  
  
**class** RoleForm(forms.ModelForm):  
  
 **class** Meta:  
 model = TSysRole  
 fields = [**'role\_name'**, **'role\_code'**] *# '\_\_all\_\_'* error\_messages = {  
 **'role\_name'**: {  
 **'required'**: **'角色名不能为空'** },  
 **'role\_code'**: {  
 **'required'**: **'角色代码不能为空'** }  
 }  
  
**class** SysUserForm(forms.ModelForm):  
  
 **class** Meta:  
 model = TSysUser  
 fields = [**'username'**, **'password'**, **'role\_id'**, **'nick\_name'**, **'head'**]  
 error\_messages = {  
 **'username'**: {  
 **'required'**: **'账号不能为空'** },  
 **'nick\_name'**: {  
 **'required'**: **'昵称不能为空'** },  
 **'password'**: {  
 **'required'**: **'口令不能为空'** },  
 **'role\_id'**: {  
 **'required'**: **'系统用户角色不能为空'** }  
 }  
  
  
**class** NoticeForm(forms.ModelForm):  
  
 **class** Meta:  
 model = TPublicNotice  
 fields = [**'content'**, **'public\_title'**, **'public\_time'**, **'public\_remarks'**, **'public\_notice\_id'**]  
 error\_messages = {  
 **'public\_title'**: {  
 **'required'**: **'标题不能为空'** },  
  
 **'content'**:{  
 **'required'**: **'内容不能为空'** }  
 }  
  
**class** SlideForm(forms.ModelForm): *# 轮播图* **class** Meta:  
 model = TSlidesshow  
 fields = [**'house'**, **'ord'**] *# '\_\_all\_\_'* error\_messages = {  
 **'house'**: {  
 **'required'**: **'房屋ID不能为空'** },  
 **'ord'**: {  
 **'required'**: **'顺序不能为空'** }  
 }  
  
  
**class** LuckyForm(forms.ModelForm): *# 代金券* **class** Meta:  
 model = TLuckyTicket  
 fields = [**'money'**, **'begin\_time'**, **'end\_time'**, **'image'**] *# '\_\_all\_\_'* error\_messages = {  
 **'money'**: {  
 **'required'**: **'面值不能为空'** },  
 **'begin\_time'**: {  
 **'required'**: **'起始时间不能为空'** },  
 **'end\_time'**: {  
 **'required'**: **'结束时间不能为空'** },  
 **'image'**: {  
 **'required'**: **'图片路径不能为空'** }  
 }

**oss\_.py**

*#!/usr/bin/python3  
# coding: utf-8***import** oss2  
  
small\_style = **'image/interlace,1/resize,m\_lfit,w\_100/quality,q\_90/bright,3/contrast,-21'  
  
  
def** get\_bucket():  
 *# 阿里云主账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您创建并使用RAM账号进行API访问或日常运维，请登录 https://ram.console.aliyun.com 创建RAM账号。* auth = oss2.Auth(**'LTAI4FqbMxbMPrnKSBsu5KSh'**, **' copeGy6ElAJ0fdleAkYxcOME5TR4CJ'**)  
 *# Endpoint以杭州为例，其它Region请按实际情况填写。* bucket = oss2.Bucket(auth, **'http://oss-cn-hangzhou.aliyuncs.com'**, **'homems'**)  
  
 **return** bucket  
  
  
**def** upload\_head(user\_id, file\_name, file\_path):  
 bucket = get\_bucket()  
  
 *# <yourObjectName>上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。  
 # <yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt。* key = **f'head/{**user\_id**}-{**file\_name**}'** ret = bucket.put\_object\_from\_file(key, file\_path)  
  
 **if** ret.status == 200:  
 **return** bucket.sign\_url(**'GET'**, key, 3600 \* 24 \* 7, params={**'x-oss-process'**: small\_style})  
  
  
**def** get\_oss\_img\_url(key):  
 bucket = get\_bucket()  
  
 **return** bucket.sign\_url(**'GET'**, **f'head/{**key**}'**, 3600 \* 24 \* 7, params={**'x-oss-process'**: small\_style})  
  
  
**def** upload\_house\_image(user\_id, file\_name, file\_path):  
 bucket = get\_bucket()  
  
 *# <yourObjectName>上传文件到OSS时需要指定包含文件后缀在内的完整路径，例如abc/efg/123.jpg。  
 # <yourLocalFile>由本地文件路径加文件名包括后缀组成，例如/users/local/myfile.txt。* key = **f'house/h-{**user\_id**}-{**file\_name**}'** ret = bucket.put\_object\_from\_file(key, file\_path)  
  
 **if** ret.status == 200:  
 **return** bucket.sign\_url(**'GET'**, key, 3600 \* 24 \* 7)  
  
  
**def** get\_oss\_house\_image(key):  
 bucket = get\_bucket()  
  
 **return** bucket.sign\_url(**'GET'**, **f'house/{**key**}'**, 3600 \* 24 \* 7)

前台系统代码

**\_\_init\_\_.py**

**from** flask **import** Flask  
**from** flask\_cors **import** CORS  
  
**import** settings  
**from** apiapp.views **import** user\_api, house\_api, slide\_api  
  
  
app = Flask(\_\_name\_\_,  
 static\_folder=settings.STATIC\_DIR,  
 static\_url\_path=**'/s/'**)  
  
app.config[**'ENV'**] = **'development'** *# production*app.config[**'DEBUG'**] = **True**app.register\_blueprint(user\_api.blue, url\_prefix=**'/api/'**)  
app.register\_blueprint(house\_api.blue, url\_prefix=**'/api/'**)  
app.register\_blueprint(slide\_api.blue, url\_prefix=**'/api/'**)  
CORS(app) *# 全局方式支持跨域请求（前端服务器和后端API服务器分开部署）*

**house\_api.py**

**import** datetime  
**import** os  
  
**from** flask **import** Blueprint, request  
**from** flask.json **import** jsonify  
**from** sqlalchemy **import** and\_  
  
**import** settings  
**from** apiapp.views **import** validate\_json, validate\_params, get\_house\_image  
**from** common **import** cache\_, oss\_  
**from** common.serializer **import** to\_json  
**from** db\_ **import** session  
**from** apiapp.models **import** THouse, THouseVerify  
  
blue = Blueprint(**'house\_api'**, \_\_name\_\_)  
  
  
  
*# 房源发布*@blue.route(**'/add/'**, methods=[**'POST'**])  
**def** add\_house():  
 *"""  
 房源发布* **:return***:  
 """  
 # resp = validate\_json()  
 # if resp: return resp  
 # resp = validate\_params(['token', 'image', 'name', 'type', 'address', 'price', 'area', 'discription'])  
 # if resp: return resp* upload\_file = request.files.get(**'image'**)  
 file\_name = upload\_file.filename  
 *# print(file\_name)* save\_file\_path = os.path.join(settings.TEMP\_HOUSE\_DIR, file\_name)  
 *# 保存上传的文件到临时的目录中* upload\_file.save(save\_file\_path)  
  
 user\_id = cache\_.get\_user\_id(request.form.get(**'token'**))  
 **if** user\_id:  
 **try**:  
 house = THouse()  
 house.user\_id = user\_id  
 *# house.image = 'http://localhost:5000/s/' + file\_name* key = **f'h-{**user\_id**}-{**file\_name**}'** *# 存储oss上的key对象* house.image = key  
 house.name = request.form.get(**'name'**)  
 house.type = request.form.get(**'type'**)  
 house.address = request.form.get(**'address'**)  
 house.price = request.form.get(**'price'**)  
 house.area = request.form.get(**'area'**)  
 house.description = request.form.get(**'discription'**)  
 house.publish\_time = datetime.datetime.now()  
 house.sale\_status = 0  
 house.is\_public = 1  
 session.add(house)  
 session.commit()  
  
  
 house\_image\_url = oss\_.upload\_house\_image(user\_id, file\_name, save\_file\_path)  
  
 *# 将房屋图片的url存到redis中* cache\_.save\_house\_url(key, house\_image\_url)  
  
 verify = THouseVerify()  
 verify.house\_id = house.house\_id  
 verify.verify\_status = 0  
 session.add(verify)  
 session.commit()  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'发布房源成功'** })  
 **except**:  
 **return** jsonify({  
 **'state'**: 2,  
 **'msg'**: **'网络延迟，请等会再尝试！'** })  
 **return** jsonify({  
 **"state"**: 1,  
 **'msg'**: **'未登录，请先登录！'** })  
  
  
@blue.route(**'/del\_house/'**, methods=[**'POST'**])  
**def** del\_house():  
 *"""  
 删除房源信息的房源信息* **:return***:  
 """  
 # 验证参数完整性  
 # resp = validate\_json()  
 # if resp: return resp  
 # resp = validate\_params('token', 'house\_id')  
 # if resp: return resp* data = request.get\_json()  
 house\_id = data[**'house\_id'**]  
 token = data[**'token'**]  
 user\_id = cache\_.get\_user\_id(token)  
 house = session.query(THouse).filter(house\_id == house\_id, user\_id == user\_id)  
 print(house)  
 **if** house:  
 obj = house.first()  
 obj.is\_public = 0  
 session.add(obj)  
 session.commit()  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'撤销房源成功'** })  
 **else**:  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'登录已过期或房屋信息已删除'** })  
  
  
@blue.route(**'/change\_house\_info/'**, methods=[**'POST'**])  
**def** change\_house\_info():  
 *"""  
 改变房屋信息  
 :return:  
 """  
 # resp = validate\_json()  
 # if resp: return resp  
 # resp = validate\_params('image', 'name', 'type', 'address', 'price', 'area', 'description', 'token', 'house\_id')  
 # if resp: return resp* upload\_file = request.files.get(**'image'**)  
 file\_name = upload\_file.filename  
 *# print(file\_name)* save\_file\_path = os.path.join(settings.TEMP\_HOUSE\_DIR, file\_name)  
 *# 保存上传的文件到临时的目录中* upload\_file.save(save\_file\_path)  
  
 data = request.form  
 house\_id = data[**'house\_id'**]  
 token = data[**'token'**]  
 user\_id = cache\_.get\_user\_id(token)  
 **if** user\_id:  
 **try**:  
 house = session.query(THouse).filter(THouse.house\_id == house\_id, THouse.user\_id == user\_id).first()  
 key = **f'h-{**user\_id**}-{**file\_name**}'** *# 存储oss上的key对象* house.image = key  
 house.name = data[**'name'**]  
 house.type = data[**'type'**]  
 house.address = data[**'address'**]  
 house.price = data[**'price'**]  
 house.area = data[**'area'**]  
 house.description = data[**'description'**]  
 house.publish\_time = datetime.datetime.now()  
 session.add(house)  
 session.commit()  
  
 house\_image\_url = oss\_.upload\_house\_image(user\_id, file\_name, save\_file\_path)  
  
 *# 将房屋图片的url存到redis中* cache\_.save\_house\_url(key, house\_image\_url)  
  
 verify = THouseVerify()  
 verify.house\_id = house.house\_id  
 verify.verify\_status = 0  
 session.add(verify)  
 session.commit()  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'房源信息修改成功！等待审核！'** })  
 **except**:  
 **return** jsonify({  
 **'state'**: 2,  
 **'msg'**: **'未发布房源，请先发布房源！'** })  
 **else**:  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'还未登陆，请先登录！'** })  
  
  
@blue.route(**'/list\_houses/'**, methods=[**'GET'**])  
**def** list\_user\_houses():  
 *"""  
 查看现登录房东的已通过审核的的房源信息  
 :return:  
 """* data = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(data)  
 print(user\_id)  
 **if** user\_id:  
 ret = session.query(THouse).filter(THouse.user\_id == user\_id).all()  
 **try**:  
 data = to\_json(ret)  
  
 **for** house\_info **in** data:  
 h\_key = house\_info.get(**'image'**)  
 h\_image\_url = get\_house\_image(h\_key)  
 *# print("+++++++++++++++", image\_url)* house\_info[**'image'**] = h\_image\_url  
 u\_key = house\_info[**'user'**][**'img'**]  
 u\_image\_url = get\_house\_image(u\_key)  
 house\_info[**'user'**][**'img'**] = u\_image\_url  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'查询成功'**,  
 **'data'**: data  
 })  
 **except**:  
 **return** jsonify({  
 **'state'**: 2,  
 **'msg'**: **'暂未发布房源'**,  
 })  
 **else**:  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'未登录'** })  
  
  
@blue.route(**'/list\_all\_houses/'**, methods=[**'GET'**])  
**def** list\_all\_houses():  
 *"""  
 查看所有房源详细信息  
 :return:  
 """* **try**:  
 ret = session.query(THouse).all()  
 data = to\_json(ret)  
 **for** (house\_info, verify\_obj) **in** zip(data, ret):  
 h\_key = house\_info.get(**'image'**)  
 h\_image\_url = get\_house\_image(h\_key)  
 *# print("+++++++++++++++", image\_url)* house\_info[**'image'**] = h\_image\_url  
 u\_key = house\_info[**'user'**][**'img'**]  
 u\_image\_url = get\_house\_image(u\_key)  
 house\_info[**'user'**][**'img'**] = u\_image\_url  
 house\_info[**"verify\_status"**] = verify\_obj.t\_house\_verifies[0].verify\_status  
 print(**"verify\_status"**, house\_info[**"verify\_status"**])  
  
 **return** jsonify({  
 **'state'**: 0,  
 **'data'**: data  
 })  
 **except** Exception **as** e:  
 session.rollback()  
 print(e)  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'获取房屋信息失败！'** })  
  
@blue.route(**'/detail/'**, methods=[**'GET'**])  
**def** detail\_info():  
 *"""  
 查看房屋的详情  
 :return:  
 """* token = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(token)  
 **if** user\_id:  
 house\_id = request.args.get(**'house\_id'**)  
  
 verify\_info = session.query(THouseVerify).filter(THouseVerify.house\_id == house\_id)  
 **if** verify\_info:  
 obj = verify\_info.first()  
 house\_info = to\_json(obj.house)  
 *# print("+++++++++++++++", type(house\_info), house\_info)* h\_key = house\_info.get(**'image'**)  
 h\_image\_url = get\_house\_image(h\_key)  
 *# print("+++++++++++++++", image\_url)* house\_info[**'image'**] = h\_image\_url  
 u\_key = house\_info[**'user'**][**'img'**]  
 u\_image\_url = get\_house\_image(u\_key)  
 house\_info[**'user'**][**'img'**] = u\_image\_url  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'查询成功'**,  
 **'data'**: {  
 **'verify\_state'**: obj.verify\_status,  
 **'houses'**: house\_info  
 }  
 })  
 **else**:  
 **return** jsonify({  
 **'state'**: 2,  
 **'msg'**: **'查询失败，发生了未知错误，请重新尝试！'** })  
  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'未登录，请先登录！'** })

**user\_api.py**

**import** os  
  
**from** flask **import** Blueprint, Response  
**from** flask **import** request, jsonify  
**from** sqlalchemy **import** or\_  
  
**import** settings  
**from** apiapp.models **import** TUser, TScore, TPanda, Contract  
**from** apiapp.views **import** validate\_json, validate\_params  
**from** common **import** code\_, token\_, cache\_, oss\_  
**from** common.serializer **import** to\_json  
**from** common.simple\_check **import** simple\_check  
**from** db\_ **import** session  
*# from db.models import TUser***from** db\_.raw **import** query  
  
blue = Blueprint(**'user\_api'**, \_\_name\_\_)  
  
  
*# 发送短信验证码*@blue.route(**'/code/'**, methods=[**'get'**])  
**def** get\_code():  
 *"""  
 发送短信验证码* **:return***:  
 """* phone = request.args.get(**'phone'**)  
 **if** phone:  
 **try**:  
 code\_.send\_code(phone)  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'验证码已发送'** })  
 **except**:  
 **return** jsonify({  
 **'state'**: 2,  
 **'msg'**: **'获取验证码失败'** })  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'手机号不能为空'** })  
  
  
*# 用户注册*@blue.route(**'/regist/'**, methods=[**'GET'**, **'POST'**])  
**def** regist():  
 *# 获取前端传来的JSON格式数据* valid\_fields = {**"phone"**, **"code"**, **"password"**}  
 data = request.get\_json()  
 **if** data **is None**:  
 **return** jsonify({  
 **'state'**: 4,  
 **'msg'**: **'必须提供json格式的参数'** })  
  
 *# 验证参数的完整性* **if** set(data.keys()) == valid\_fields:  
 *# 验证输入的验证码是否正确* **if not** code\_.valid\_code(data[**'phone'**], data[**'code'**]):  
 **return** jsonify({  
 **'state'**: 2,  
 **'msg'**: **'验证码输入错误，请确认输入的验证码'** })  
 user = TUser()  
 *# user.name = data.get('name')* user.phone = data.get(**'phone'**)  
 user.password = data.get(**'password'**)  
  
 session.add(user)  
 session.commit()  
 *# 向前端返回信息中，包含一个与用户匹配的token(有效时间为一周)  
 # 1. 基于uuid+user\_id生成token  
 # 2. 将token和user\_id保存到缓存（cache\_.save\_token(token, user\_id)）  
 # JWT 单点授权登录* token = token\_.gen\_token(user.user\_id)  
 cache\_.add\_token(token, user.user\_id)  
 **else**:  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'参数不完整，详情查看接口文档'** })  
  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'注册成功'** })  
  
  
*# 用户登陆*@blue.route(**'/login/'**, methods=[**'POST'**])  
**def** login():  
 data = request.get\_json()  
 print(**"data="**, data)  
 **try**:  
 user = session.query(TUser).filter(or\_(TUser.phone == data[**'phone'**],  
 TUser.nickname == data[**'nickname'**]),  
 TUser.password == data[**'password'**]).one()  
  
  
 **if** user:  
 token = token\_.gen\_token(user.user\_id)  
 cache\_.add\_token(token, user.user\_id)  
 head\_url = **''  
 if** user.img:  
 head\_url = cache\_.get\_head\_url(user.img)  
 *# print(head\_url)* **if not** head\_url:  
 head\_url = oss\_.get\_oss\_img\_url(user.img)  
 cache\_.save\_head\_url(user.img, head\_url)  
 resp: Response = jsonify({  
 **'state'**: 0,  
 **'msg'**: **'登录成功'**,  
 **'token'**: token,  
 **'head'**: head\_url  
 })  
 *# 设置响应对象的cookie，向客户端响应cookie* resp.set\_cookie(**'token'**, token)  
 resp.set\_cookie(**'head'**, head\_url)  
 **return** resp  
 **except**:  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'用户名或口令输入错误'**,  
 })  
  
  
*# 普通用户升级为会员用户*@blue.route(**'/become\_superuser/'**, methods=[**'POST'**])  
**def** become\_superuser():  
 *"""  
 普通用户升级为会员用户,并进行实名认证  
 :return:  
 """  
 # 验证用户是否输入json格式数据* resp = validate\_json()  
 **if** resp: **return** resp  
 *# 验证用户输入参数是否齐全* resp = validate\_params(**'token'**, **'nickname'**, **'sex'**, **'phone'**, **'email'**, **'name'**, **'identity\_number'**)  
 **if** resp: **return** resp  
 data = request.get\_json()  
 print(**"data="**, data)  
 token = data.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(token)  
 print(user\_id)  
 **if** user\_id:  
 user = session.query(TUser).get(user\_id)  
 **if** simple\_check(data[**'identity\_number'**], data[**'name'**]):  
 user.name = data[**'name'**]  
 user.nickname = data[**'nickname'**]  
 user.sex = data[**'sex'**]  
 user.phone = data[**'phone'**]  
 user.identity\_number = data[**'identity\_number'**]  
 user.is\_member = 1  
 user.has\_real\_name = 1  
 session.add(user)  
 session.commit()  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'已成功成为会员用户，您可以开始发布自己的房源了！'**,  
 })  
 **return** jsonify({  
 **'state'**: 2,  
 **'msg'**: **'请填写正确的身份信息！'** })  
 **else**:  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'未登录， 请先登录！'** })  
  
  
*# 用户修改密码*@blue.route(**'/modify\_password/'**, methods=[**'POST'**])  
**def** modify\_password():  
 *# 验证用户是否输入json格式数据* resp = validate\_json()  
 **if** resp: **return** resp  
  
 *# 验证用户输入参数是否齐全* resp = validate\_params(**'new\_password\_str'**, **'password\_str'**, **'token'**)  
 **if** resp: **return** resp  
  
 data = request.get\_json()  
 print(data[**'token'**])  
 **try**:  
 user\_id = cache\_.get\_user\_id(data[**"token"**])  
 print(user\_id)  
 **if not** user\_id:  
 **return** jsonify({  
 **'state'**: 3,  
 **'msg'**: **'登录已期，需要重新登录并获取新的token'**,  
 })  
 user = session.query(TUser).get(int(user\_id))  
 **if** user.password == data[**'password\_str'**]:  
 user.password = data[**'new\_password\_str'**]  
 session.add(user)  
 session.commit()  
  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'修改成功'** })  
 **return** jsonify({  
 **'state'**: 4,  
 **'msg'**: **'原口令不正确'** })  
 **except**:  
 **pass  
  
 return** jsonify({  
 **'state'**: 3,  
 **'msg'**: **'token已无效，尝试重新登录'**,  
 })  
  
  
*# 用户注销*@blue.route(**'/login\_out/'**)  
**def** login\_out():  
 token = request.args.get(**'token'**)  
 cache\_.del\_token(token)  
  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'退出登陆成功'** })  
  
  
*# 用户上传头像*@blue.route(**'/upload\_head/'**, methods=[**"POST"**,**"GET"**])  
**def** upload\_head():  
 *# 前端上传图片的两种方式(文件上传，base64字符串上传)* upload\_file = request.files.get(**'head'**)  
 token = request.cookies.get(**'token'**)  
 *# print(upload\_file.filename)  
 # print(token)* user\_id = cache\_.get\_user\_id(token)  
 filename = upload\_file.filename  
 save\_file\_path = os.path.join(settings.TEMP\_USER\_DIR, filename)  
 *# 保存文件到临时的目录中* upload\_file.save(save\_file\_path)  
  
 *# 将头像保存到用户表中* user = session.query(TUser).get(user\_id)  
 **if** user:  
 **try**:  
 user.img = **f"{**user\_id**}-{**filename**}"** *# 存储oss上的key对象* session.add(user)  
 session.commit()  
  
 *# 将保存的文件上传到oss服务中，并获取到缩小后的图片的url* head\_url = oss\_.upload\_head(user\_id, filename, save\_file\_path)  
  
 *# 将头像的url存到redis中* cache\_.save\_head\_url(user.img, head\_url)  
  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'上传成功'** })  
 **except**:  
 **return** jsonify({  
 **'state'**: 2,  
 **'msg'**: **'上传失败'** })  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'登录已超时，请先登录！'** })  
  
  
*# 获取用户头像*@blue.route(**'/head/'**, methods=[**"GET"**])  
**def** get\_head():  
 token = request.args.get(**"token"**)  
 user\_id = cache\_.get\_user\_id(token)  
 **if not** user\_id:  
 **return** jsonify({  
 **'state'**: 3,  
 **'msg'**: **'登录已期，需要重新登录并获取新的token'**,  
 })  
  
 *# sql = 'select nickname,img,is\_member from t\_user where user\_id=%s'  
 # data = query(sql, user\_id)* user = session.query(TUser).get(user\_id)  
 nickname = user.nickname  
 img = user.img  
 is\_member = user.is\_member  
 data = {  
 **'nickname'**: nickname,  
 **'img'**: img,  
 **'is\_member'**: is\_member  
 }  
 **return** jsonify({  
 **'state'**: 0,  
 **'data'**: data  
 })  
  
  
*# 用户更新信息*@blue.route(**'/detail\_resource/'**, methods=[**"POST"**])  
**def** change\_user\_info():  
 *# 验证用户是否输入json格式数据* resp = validate\_json()  
 **if** resp: **return** resp  
  
 data = request.get\_json()  
  
 token = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(data[**"token"**])  
 **if not** user\_id:  
 **return** jsonify({  
 **'state'**: 3,  
 **'msg'**: **'登录已期，需要重新登录并获取新的token'**,  
 })  
  
 *# 根据id查询用户* user = session.query(TUser).get(user\_id)  
 user.nickname = data[**"nickname"**]  
 user.email = data[**"email"**]  
 session.add(user)  
 session.commit()  
  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'更新信息成功'**,  
 **'data'**: data  
 })  
  
  
*# 用户积分查询*@blue.route(**'/integral/'**, methods=[**"GET"**])  
**def** interal():  
 token = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(token)  
 **if not** user\_id:  
 **return** jsonify({  
 **'state'**: 3,  
 **'msg'**: **'登录已期，需要重新登录并获取新的token'**,  
 })  
 sql = **'select score from t\_score join t\_user on t\_score.user\_id=t\_user.user\_id where t\_score.user\_id=%s'** score = query(sql, user\_id)  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'ok'**,  
 **'data'**: score  
 })  
  
  
*# 关于我们*@blue.route(**'/about\_pandas/'**, methods=[**"GET"**])  
**def** about\_pandas():  
 sql = **'select \* from t\_panda where t\_panda.panda\_id=%s'** detail = query(sql, 1)  
  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'ok'**,  
 **'interal'**: detail  
 })  
  
  
*# 查看合同*@blue.route(**'/get\_contract/'**, methods=[**"GET"**])  
**def** get\_contract():  
 token = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(token)  
 **if not** user\_id:  
 **return** jsonify({  
 **'state'**: 3,  
 **'msg'**: **'登录已期，需要重新登录并获取新的token'**,  
 })  
 sql = **'select start\_time,stop\_time,content from t\_user join contract c2 on t\_user.user\_id = c2.user\_id where c2.user\_id=%s'** cont\_details = query(sql, user\_id)  
  
 **if** cont\_details:  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'已查询到合同信息'**,  
 **'data'**: cont\_details  
 })  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'您的合同信息不存在'** })  
  
  
*# 查看房屋交易记录*@blue.route(**'/traderecord/'**, methods=[**"GET"**])  
**def** traderecord():  
 token = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(token)  
 **if not** user\_id:  
 **return** jsonify({  
 **'state'**: 3,  
 **'msg'**: **'登录已期，需要重新登录并获取新的token'**,  
 })  
 sql = **'select th.\*,td.\* from t\_house th,t\_tradingrecord td,t\_user tu where th.user\_id=td.user\_id and td.user\_id=tu.user\_id and tu.user\_id=%s'** house\_details = query(sql, user\_id)  
 **if** house\_details:  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'已查询到房屋交记录'**,  
 **"data"**: house\_details  
 })  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'您暂时没有交易记录'** })  
  
  
*# 查看订单*@blue.route(**'/get\_order/'**, methods=[**"GET"**])  
**def** get\_order():  
 token = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(token)  
 **if not** user\_id:  
 **return** jsonify({  
 **'state'**: 3,  
 **'msg'**: **'登录已期，需要重新登录并获取新的token'**,  
 })  
 status = int(request.args.get(**"status"**))  
 print(status, type(status))  
 **if** status == 2:  
 *# 1查看全部订单* sql = **'select th.\*,td.\* from t\_user tu,t\_order td,t\_house th where tu.user\_id=td.user\_id and tu.user\_id=th.user\_id and tu.user\_id=%s'** all\_orders = query(sql, user\_id)  
 print(all\_orders)  
 **if** all\_orders:  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'全部订单信息'**,  
 **'order'**: all\_orders  
 })  
 **elif** status == 0:  
 *# 2.查看已支付订单* sql = **'select th.\*,td.\* from t\_user tu,t\_order td,t\_house th where tu.user\_id=td.user\_id and tu.user\_id=th.user\_id and td.order\_status=%s'** pay\_order = query(sql, 0)  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'有效订单信息'**,  
 **'order'**: pay\_order  
 })  
  
 **elif** status == 1:  
 *# 3.查看待支付订单* sql = **'select th.\*,td.\* from t\_user tu,t\_order td,t\_house th where tu.user\_id=td.user\_id and tu.user\_id=th.user\_id and td.order\_status=%s'** nopay\_order = query(sql, 1)  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'待支付订单信息'**,  
 **'order'**: nopay\_order  
 })  
  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'暂无订单'** })  
  
  
*# 查看代金券*@blue.route(**'/cash/'**, methods=[**"GET"**])  
**def** cash():  
 token = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(token)  
 **if not** user\_id:  
 **return** jsonify({  
 **'state'**: 3,  
 **'msg'**: **'登录已期，需要重新登录并获取新的token'**,  
 })  
 sql = **'select tl.\* from t\_u\_lucky\_ticket tul,t\_lucky\_ticket tl,t\_user tu where tu.user\_id=tul.user\_id and tul.lucky\_ticket\_id=tl.lucky\_ticket\_id and tu.user\_id=%s'** cash\_ = query(sql, user\_id)  
 print(cash\_)  
  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'代金券查询成功'**,  
 **'data'**: cash\_  
 })

**slide.py**

**import** uuid  
  
**from** flask **import** Blueprint, request, g, redirect, url\_for  
**from** flask.json **import** jsonify  
  
**from** apiapp.views **import** validate\_json, validate\_params, get\_house\_image  
**from** common **import** cache\_  
**from** common.serializer **import** to\_json  
**from** db\_ **import** session  
**from** apiapp.models **import** TUser, THouse, THouseVerify, TSlidesshow, TFavorite, TOrder  
  
blue = Blueprint(**'slide\_api'**, \_\_name\_\_)  
  
  
@blue.route(**'/slide/'**, methods=[**'GET'**])  
**def** slide\_view(): *# 获取首页轮播图* **try**:  
 slides\_list = session.query(TSlidesshow).all()  
 *# house\_info = session.query(THouse).filter(THouse.house\_id == s.house\_id)* res = to\_json(slides\_list)  
 **for** ad **in** res:  
 key = ad[**'house'**][**"image"**]  
 ad[**'house'**][**"image"**] = get\_house\_image(key)  
  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'请求成功'**,  
 **'data'**: res  
 })  
  
 **except** Exception **as** e:  
 session.rollback()  
 print(**"获取轮播图数据错误"**, e)  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'服务器错误'** })  
  
  
*# 首页房屋展示*@blue.route(**'/house\_display/'**, methods=[**"GET"**])  
**def** house\_display():  
 houses = session.query(THouse).all()  
 **if** houses:  
 **return** jsonify({  
 **"state"**: 0,  
 **'data'**: to\_json(houses)  
 })  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'房屋信息不存在'** })  
  
  
*# 房屋详细信息页*@blue.route(**'/house\_detail/'**, methods=[**'GET'**])  
**def** detail\_info():  
 *"""  
 查看房屋的详情* **:return***:  
 """* token = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(token)  
 **if** user\_id:  
 house\_id = request.args.get(**'house\_id'**)  
  
 verify\_info = session.query(THouseVerify).filter(THouseVerify.house\_id == house\_id)  
 **if** verify\_info:  
 obj = verify\_info.first()  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'查询成功'**,  
 **'data'**: {  
 **'verify\_state'**: obj.verify\_status,  
 **'houses'**: to\_json(obj.house)  
 }  
 })  
 **else**:  
 **return** jsonify({  
 **'state'**: 2,  
 **'msg'**: **'查询失败，发生了未知错误，请重新尝试！'** })  
  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'未登录，请先登录！'** })  
  
  
*# 收藏房屋*@blue.route(**'/insert\_collection/'**, methods=[**"POST"**])  
**def** collection():  
 data = request.get\_json()  
 user\_id = cache\_.get\_user\_id(data[**"token"**])  
 **if** user\_id:  
 house\_id = data[**"house\_id"**]  
 fav = TFavorite()  
 fav.house\_id = house\_id  
 fav.user\_id = user\_id  
 session.add(fav)  
 session.commit()  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'房屋收藏成功'**,  
 })  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'房屋添加不成功'** })  
  
  
*# 查询收藏房屋的信息*@blue.route(**'/get\_collection/'**, methods=[**"GET"**])  
**def** get\_collection():  
 data = request.args.get(**'token'**)  
 user\_id = cache\_.get\_user\_id(data)  
 **if** user\_id:  
 ret = session.query(TFavorite).filter(TFavorite.user\_id == user\_id).all()  
 *# if house\_detail:* **return** jsonify({  
 **'state'**: 0,  
 **'data'**: to\_json(ret)  
 })  
 **return** ({  
 **'state'**: 1,  
 **'msg'**: **'暂无收藏的房屋'** })  
  
  
*# 下订单*ORDER\_PAY = 0 *# 已支付订单*ORDER\_NOT\_PAY = 1 *# 待支付订单  
  
# 添加订单*@blue.route(**'/order/'**, methods=[**"POST"**])  
**def** order():  
 *"""  
 添加订单* **:return***:  
 """* resp = validate\_json()  
 **if** resp: **return** resp  
 resp = validate\_params(**'token'**, **'house\_id'**, **'hire\_price'**, **'cash\_price'**, **'enter\_time'**, **'exit\_time'**)  
 **if** resp: **return** resp  
  
 order\_num = uuid.uuid4().hex  
 data = request.get\_json()  
 user\_id = cache\_.get\_user\_id(data[**"token"**])  
 *# total\_price = (data["exit\_time"] - data["enter\_time"]) \* float(data["hire\_price"]) + float(data["cash\_price"])* order = TOrder()  
 order.order\_number = order\_num  
 order.user\_id = user\_id  
 order.house\_id = data[**"house\_id"**]  
 order.hire\_price = data[**"hire\_price"**]  
 order.cash\_price = data[**"cash\_price"**]  
 order.enter\_time = data[**"enter\_time"**]  
 order.exit\_time = data[**"exit\_time"**]  
 *# order.total = total\_price* order.order\_status = ORDER\_NOT\_PAY  
 session.add(order)  
 session.commit()  
 order\_info = session.query(TOrder).filter(TOrder.user\_id==user\_id, TOrder.order\_number==order\_num)  
 *# print(order\_info)* **if** order\_info:  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'下单成功'**,  
 **'data'**: to\_json(order\_info.all())  
 })  
  
  
@blue.route(**'/del\_order/'**, methods=[**'POST'**])  
**def** del\_order():  
 resp = validate\_json()  
 **if** resp: **return** resp  
 resp = validate\_params(**'token'**, **'order\_number'**)  
 **if** resp: **return** resp  
  
 data = request.get\_json()  
 user\_id = cache\_.get\_user\_id(data[**'token'**])  
 **if** user\_id:  
 order\_number = data[**'order\_number'**]  
 order = session.query(TOrder).filter(TOrder.order\_number==order\_number, TOrder.user\_id==user\_id).first()  
 **if** order:  
 session.delete(order)  
 session.commit()  
 **return** jsonify({  
 **'state'**: 0,  
 **'msg'**: **'该订单已删除！'** })  
 **else**:  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'订单不存在'** })  
 **else**:  
 **return** jsonify({  
 **'state'**: 1,  
 **'msg'**: **'登录已超时，请重新登录！'** })

**致谢**

从毕业设计的开题到论文和程序的完成，将近有差不多一年的时间，在这个漫长过程中，首先特别感谢我的指导老师陈垚老师。因为陈垚老师总是特别认真和耐心的回答我的各种问题，在陈垚老师的细心指导之下，我学到了特别多的新东西。陈垚老师也经常会询问我们的进度情况，而且当我把已经完成的论文发给老师后，老师也会及时的进行检查和批注。

同时也特别的感谢我的同学和身边的朋友，在毕业设计期间，无论是我的学习、生活上遇到任何问题向他们请教时，他们也都总会力所能及的对我进行帮助;当我的毕设遇到难以解决的问题时，他们也一直也都一直在尽力的帮助我。我真的特别感谢他们，正是在他们的无私帮助之下，我的毕设才能圆满完成。

当然我也非常的感谢我的学校一一商洛学院，它为我们提供了一个特别好的学习平台，也给我们配备了最优秀的专业导师，让我们接触并且学习到了非常多的专业知识知识。“不忘初心，继续前进”的校训我将铭记在心!