

课 程 设 计

房屋销售系统

设计题目

葛青云

学生姓名

2020210506

学 号

信息与计算科学20-01

专业班级

何蕾

指导教师

**2023** 年 3 月 31 日

## 房屋销售系统

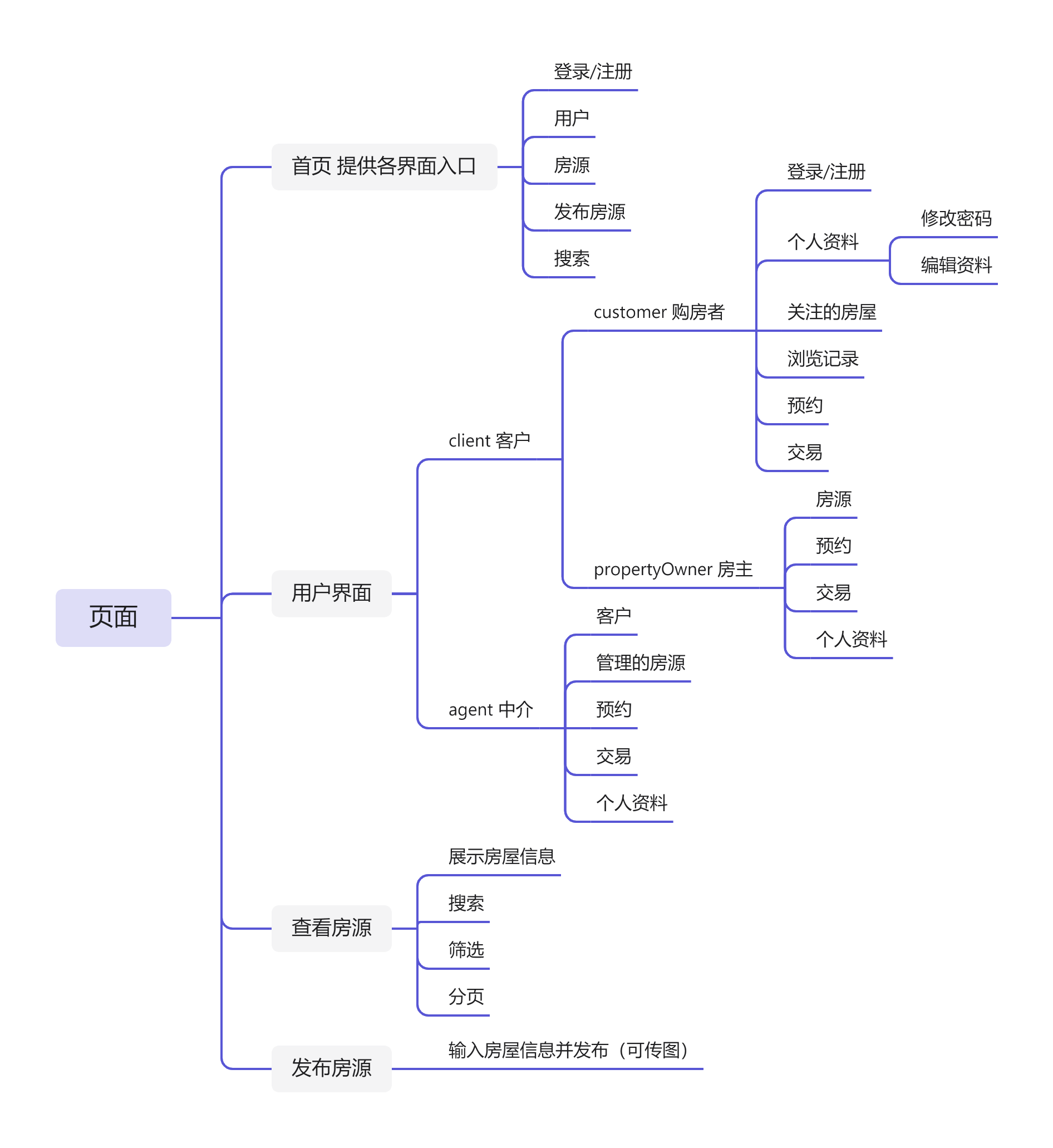
本项目建立了一个房屋销售网站

**一、问题描述及需求**

简要需求：需完成如下用户系统并实现其功能，不同用户的需求不同，要实现的功能也不同

1. 用户系统
   1. client 客户
      1. customer 购房者
      2. propertyOwner 房主
   2. agent 中介
2. 功能
   1. 登录/注册
   2. 查看房屋信息
   3. 搜索/筛选房屋
   4. 预约看房
   5. 交易

以下是以网站形式进一步明确的需求

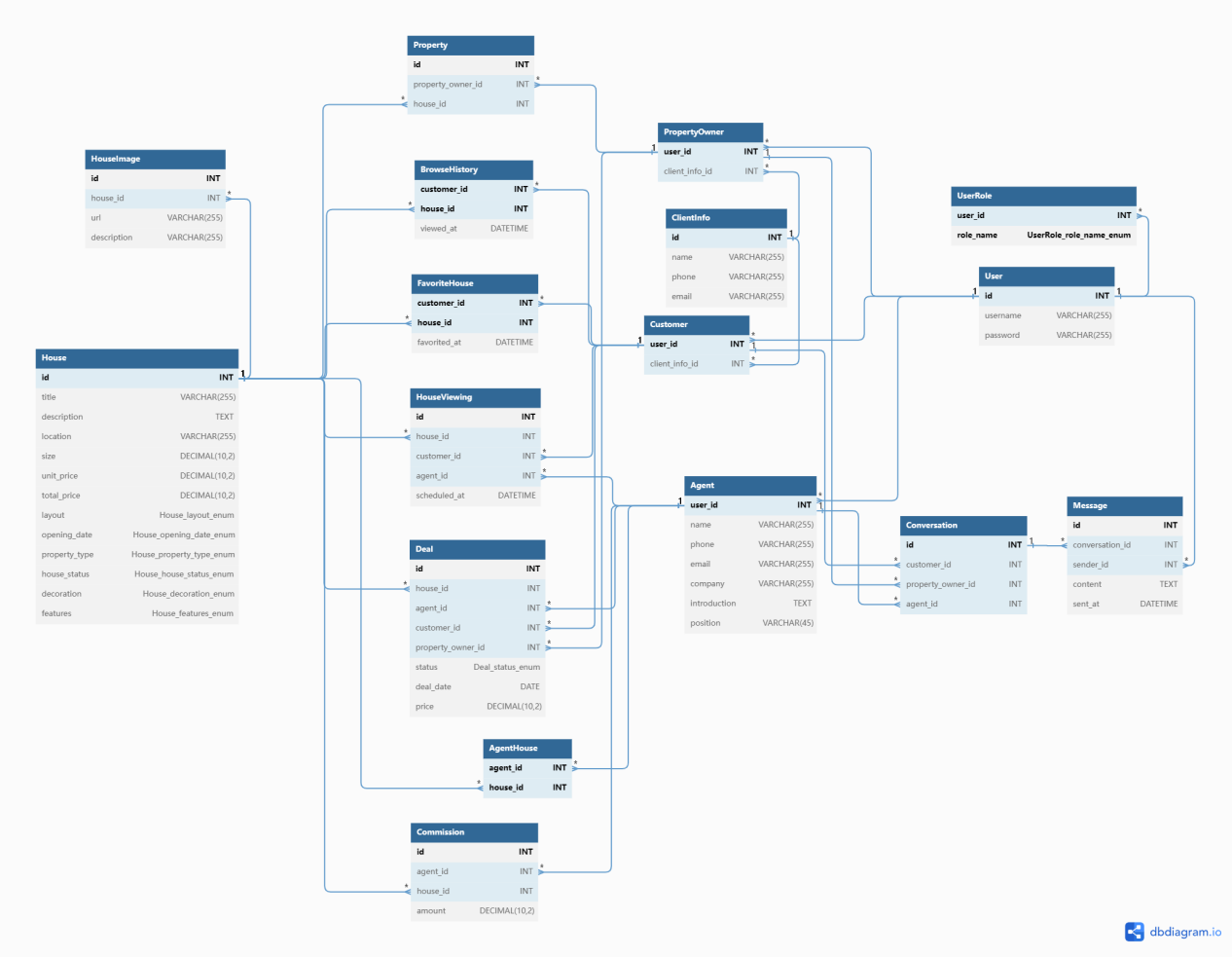


**二、解决方案**

1.设计数据库

以下是我的数据库图示，也可看作是关系模式的图例，图中的线表示参照完整性

[若想查看高清图或进入设计页面点击这里](https://dbdiagram.io/d/6425bc485758ac5f172587bd)



**具体介绍：  
第一列是房屋信息：**

**House涵盖了绝大多数房屋属性，其中除了描述（description）为text外，其他所有string类型的属性都像如下layout属性利用ENUM限定了其取值范围以简化数据库及非法输入。**

**layout ENUM('1居', '2居', '3居', '4居', '5居', '5居+') NOT NULL,**

**HouseImage用于储存房屋图片url及图片描述**

**第二列代表房屋和用户之间的联系，用于进行相关操作**

**其中agentHouse，Property，BrouseHistory，FavouriteHouse 用于操作用户管理、拥有、浏览过和喜欢的房子。**

**其中HouseViewing，Deal用于操作预约看房和交易的相关任务**

**第三列表示三种用户**

**第四列表示总的用户**

**这里我们详细介绍一下用户系统**

在本用户系统中，有两大类用户

* 1. client 客户
     1. customer 购房者
     2. propertyOwner 房主
  2. agent 中介

他们都会记录在User表中，其中client并没有对应table。

Use表中的password存放的是密码的密文，密码明文只有用户知道，除非能同时拥有数据库和加密方法及密码表

由于在本系统中customer和propertyOwner，可以是同一个人，因此它们的信息会存放在clientInfo中，以避免重复存储，而agent的信息则直接存在agent表中

又由于一个人可以同时为customer，propertyOwner，所以为User表配了一个userrole表来标明身份

最后右下角的Conversation和Message是用于用户间交流信息的，但是本项目并没有实现此功能。

**对应的关系模式：**

House (id PK, title, description, location, size, unit\_price, total\_price, layout, opening\_date, property\_type, house\_status, decoration, features)

HouseImage (id PK, house\_id FK, url, description)

User (id PK, username, password)

UserRole (user\_id FK, role\_name, PK(user\_id, role\_name))

ClientInfo (id PK, name, phone, email)

Customer (user\_id PK FK, client\_info\_id FK)

PropertyOwner (user\_id PK FK, client\_info\_id FK)

Agent (user\_id PK FK, name, phone, email, company, introduction, position)

BrowseHistory (customer\_id FK, house\_id FK, viewed\_at, PK(customer\_id, house\_id))

FavoriteHouse (customer\_id FK, house\_id FK, favorited\_at, PK(customer\_id, house\_id))

Property (id PK, property\_owner\_id FK, house\_id FK)

HouseViewing (id PK, house\_id FK, customer\_id FK, agent\_id FK, scheduled\_at)

AgentHouse (agent\_id FK, house\_id FK, PK(agent\_id, house\_id))

Commission (id PK, agent\_id FK, house\_id FK, amount)

Deal (id PK, house\_id FK, agent\_id FK, customer\_id FK, property\_owner\_id FK, status, deal\_date, price)

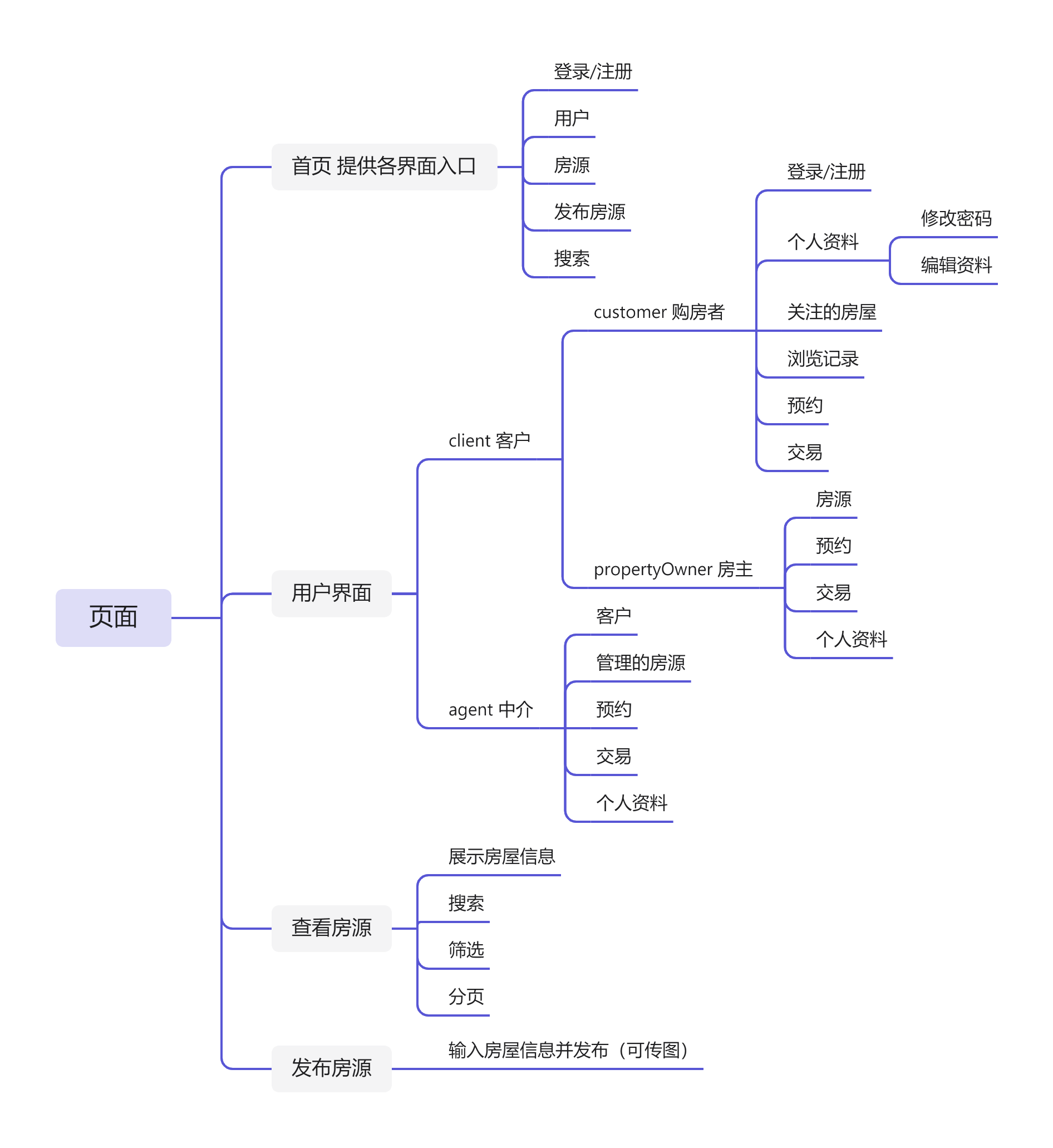
Conversation (id PK, customer\_id FK, property\_owner\_id FK, agent\_id FK)

Message (id PK, conversation\_id FK, sender\_id FK, content, sent\_at)

1. 前端

利用html, css, js来实现网站前端

所需实现的各个页面和功能已经在下图中展示了



1. 后端

使用nodejs实现后端

**后端实现的api/函数：**

**用户登录/注册：**

login = (req, res) => {

这个函数实现了用户登录的功能，接收了前端发送的请求体 req.body，其中包含用户输入的用户名和密码信息。

首先，函数会将用户名作为查询条件，从数据库中查找用户信息。

如果查询失败，则返回错误信息；如果查询结果不为1，则说明用户名不存在，也返回错误信息。

如果查询结果存在，则将结果和用户输入的密码使用bcrypt.compareSync()方法进行比较，如果比较结果不一致，则说明密码错误，也返回错误信息。如果比较结果一致，则表示用户登录成功。

接下来，函数会查询用户的角色信息，根据用户类型判断用户是**否有权限**登录。如果没有权限，则返回错误信息。

如果有权限，则在用户信息中添加角色信息，**并使用jwt库生成token字符串**，作为登录成功后的返回信息，同时将用户的角色信息也一起返回。

regUser = (req, res) => {

注册用户的处理函数，包含以下功能：

1. 校验用户提交的信息是否完整。

b. 查询用户名是否已被占用。

c. 将用户提交的密码加密后存入数据库。

d. 插入客户信息到数据库的 ClientInfo 表中。

e. 插入用户信息到数据库的 User 表中。

f. 在数据库的 UserRole 表中插入用户角色信息。

g. 根据用户类型，在对应的表中插入用户的详细信息。

h. 返回注册结果的响应信息。

**编辑用户信息：**

getUserType：获取当前用户的用户类型。

getUserInfo：获取当前用户的基本信息。

getAgentInfoById：根据经纪人 ID 获取经纪人的信息。

updateUserInfo：更新当前用户的信息，如姓名、电话和邮箱等。

updatePassword：更新当前用户的密码。

updateAvatar：更新当前用户的头像。

此代码中使用了 bcrypt 库来进行密码的哈希处理。另外，还使用了 db 模块来操作数据库，其中 db.query 方法用于执行 SQL 查询语句。每个函数都会通过 HTTP 响应的方式将结果返回给客户端。

**有关房屋的操作**

· createImages(house\_id, images): 在数据库中为指定房产添加图片。接受两个参数：房产 ID 和包含图片信息的数组（每个元素包含图片 URL 和描述信息）。在执行过程中，函数通过查询数据库并添加信息实现任务。如果有错误，函数将抛出一个错误。

· createListing(req, res): 为新的房产信息添加到数据库中。这个函数首先从请求体中获取房产信息，然后将它们添加到数据库中。然后，它会生成对应的图片 URL，并将图片添加到数据库中。最后，如果用户没有“propertyOwner”角色，函数将添加一个该角色。如果发生错误，函数将抛出错误。

· getHouseById(req, res): 根据 ID 查询房产信息和图片。该函数从请求参数中获取 ID，然后通过查询数据库来检索有关房产的信息。如果找到，则查询数据库以获取有关房产的所有图片。如果找不到房产，则函数将返回一个 404 状态码。

· getAgentByHouseId(req, res): 根据房产 ID 查询管理该房产的代理人。该函数从请求参数中获取房产 ID，然后查询数据库以查找相应的代理人。如果找到代理人，则返回代理人信息。否则，函数将返回一个 404 状态码。

· getHousesByPropertyOwner(req, res): 根据 propertyOwner 的 ID 查询该 propertyOwner 拥有的所有房产。函数从请求体中获取 propertyOwner 的 ID，然后在数据库中查找与该 ID 相关的所有房产信息。如果没有找到房产，则函数将返回一个空数组。

· getHousesByAgentId：用于获取某个经纪人管理的房屋信息。该函数会从数据库中查询符合条件的房屋数据，然后返回查询结果。

· viewHouse：用于创建一条新的房屋查看记录。当客户查看某个房屋时，该函数将根据请求中的信息创建一条新的记录并保存到数据库中。

· getHouseViewingsByAgentId：用于获取某个经纪人管理的房屋查看记录信息。该函数会从数据库中查询符合条件的房屋查看记录数据，然后返回查询结果。

· updateHouseViewing：用于更新一条房屋查看记录。当客户需要修改已有的查看记录时，该函数将根据请求中的信息更新数据库中对应的记录。

· createDeal：用于创建一条新的交易记录。当经纪人成功帮助客户完成房屋交易后，该函数将根据请求中的信息创建一条新的记录并保存到数据库中。

· getDealsByAgentId：用于获取某个经纪人管理的交易记录信息。该函数会从数据库中查询符合条件的交易记录数据，然后返回查询结果。

· getHousesByAgentId：用于获取某个经纪人管理的房屋信息。该函数会从数据库中查询符合条件的房屋数据，然后返回查询结果。

· viewHouse：用于创建一条新的房屋查看记录。当客户查看某个房屋时，该函数将根据请求中的信息创建一条新的记录并保存到数据库中。

· getHouseViewingsByAgentId：用于获取某个经纪人管理的房屋查看记录信息。该函数会从数据库中查询符合条件的房屋查看记录数据，然后返回查询结果。

· updateHouseViewing：用于更新一条房屋查看记录。当客户需要修改已有的查看记录时，该函数将根据请求中的信息更新数据库中对应的记录。

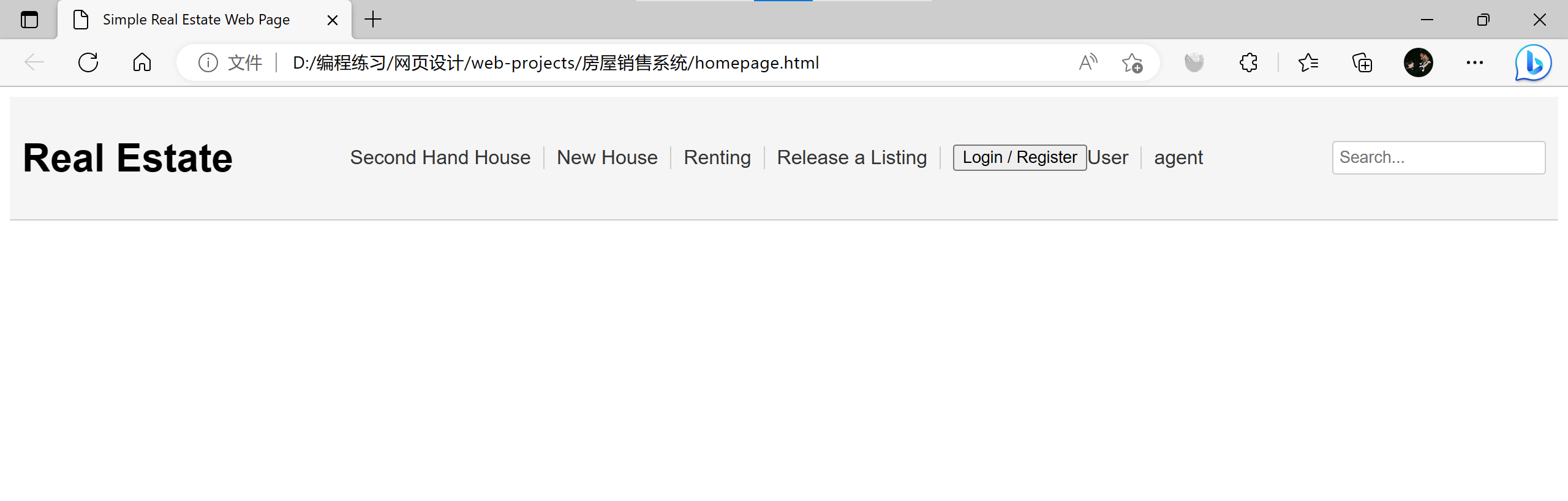
· createDeal：用于创建一条新的交易记录。当经纪人成功帮助客户完成房屋交易后，该函数将根据请求中的信息创建一条新的记录并保存到数据库中。

· getDealsByAgentId：用于获取某个经纪人管理的交易记录信息。该函数会从数据库中查询符合条件的交易记录数据，然后返回查询结果。

1. 具体实现及演示

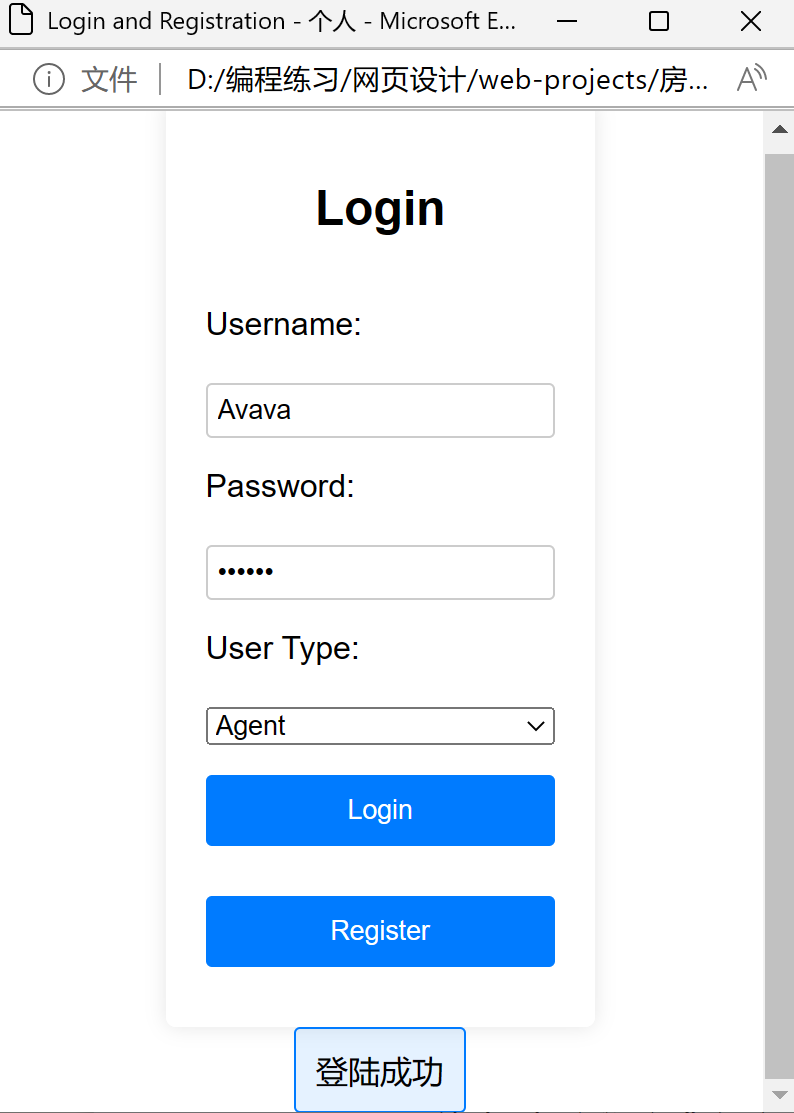
源码可在https://github.com/GQY-479/House-Sales-System 查看

网站首页



首先需要登录或者注册，不登录没有token无法像服务器发送请求

点击Login/Register之后进入登录页面

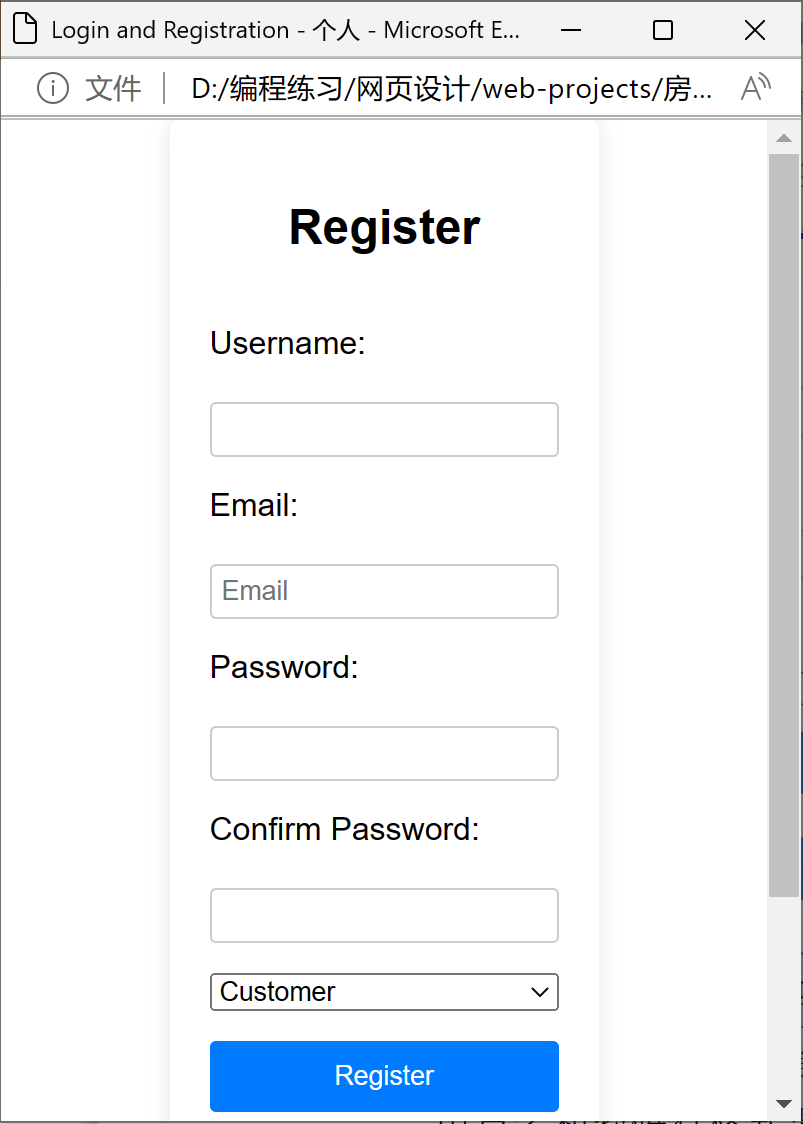


此处需要选择身份，身份不对无法登录

用户名和密码有格式要求，错误会提示

点击register进入注册页面

填写信息注册

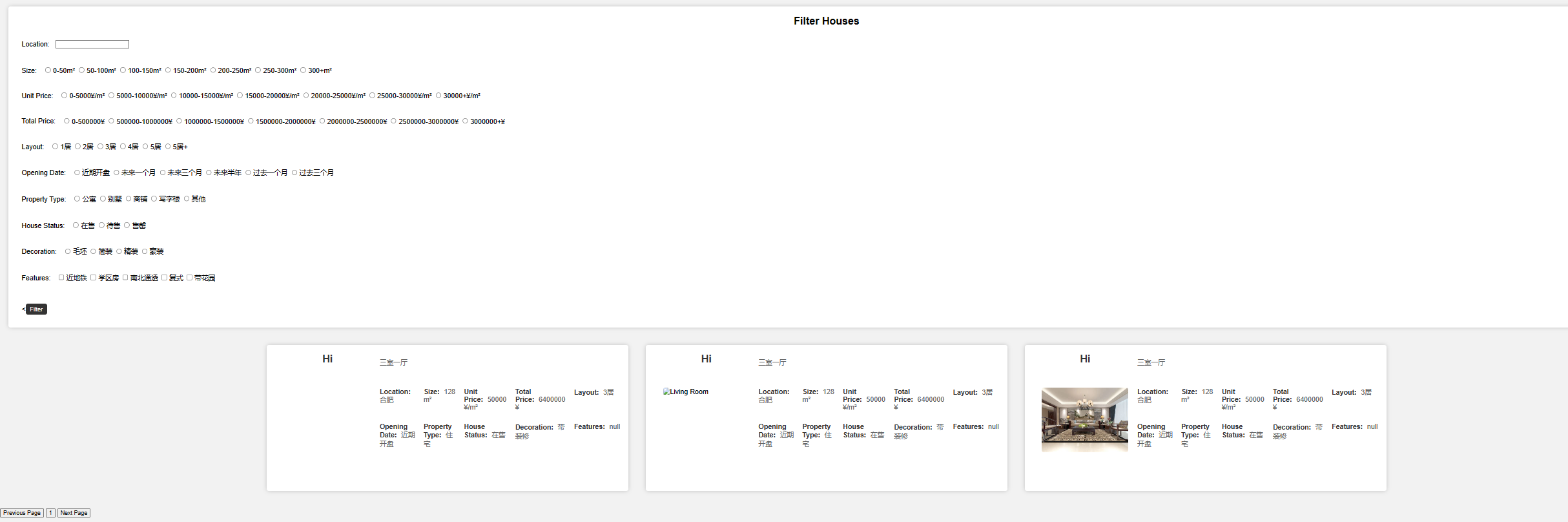


之后进入房屋浏览界面

此处展示房屋信息，如果有图，会展示图片

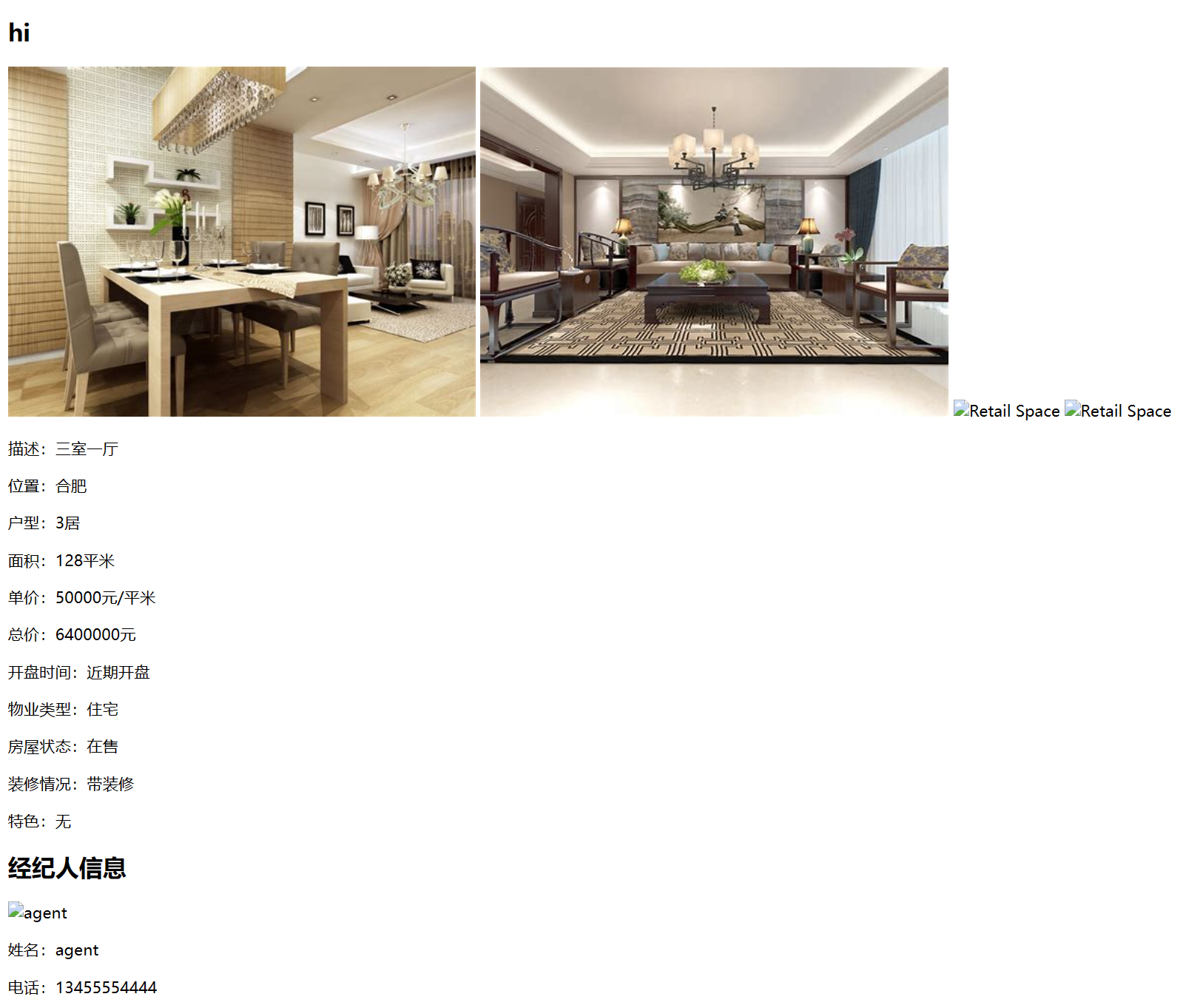
可以筛选房屋，

点击下一页后会显示下一页内容，会多出下一页的页码按钮



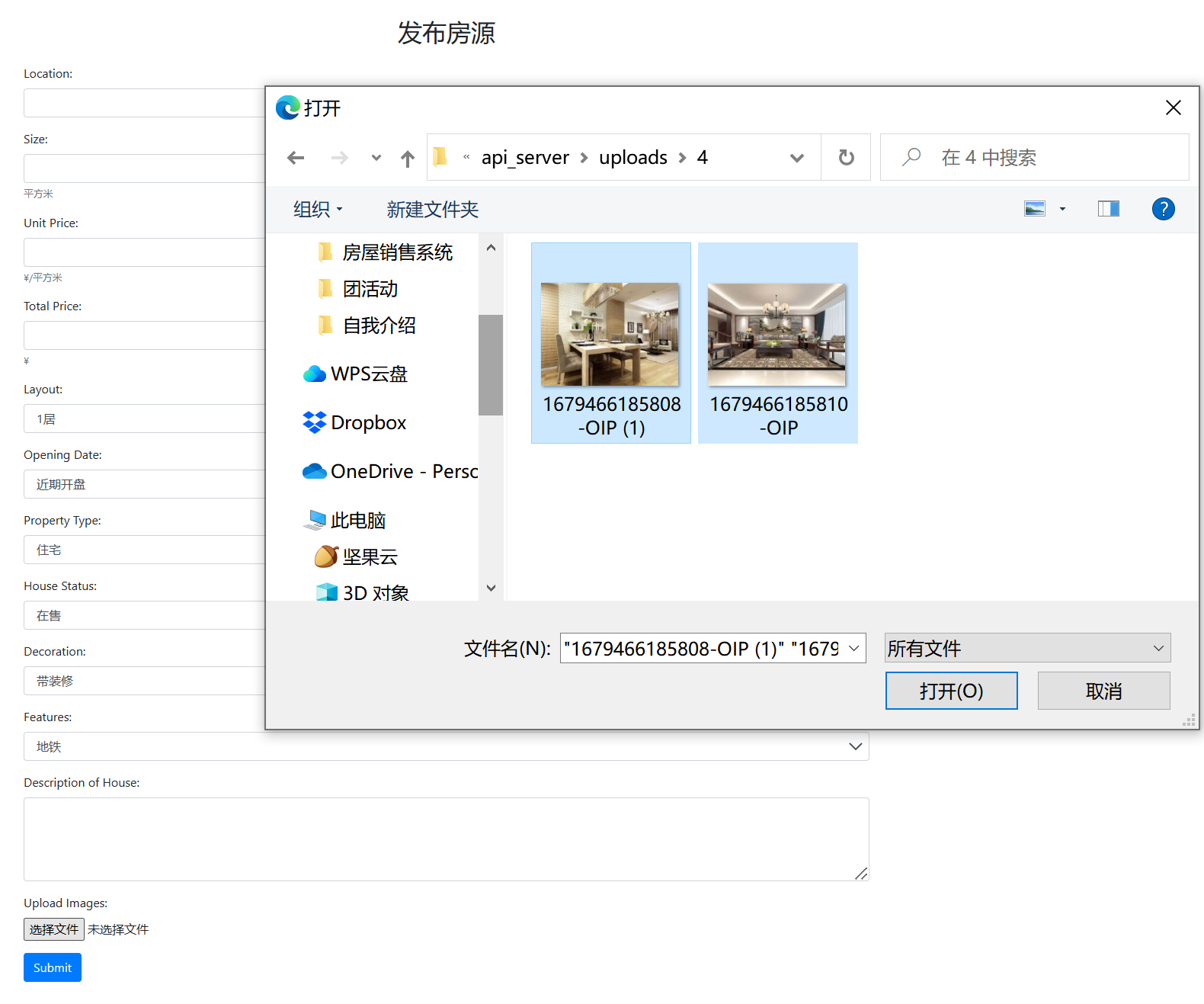
点击其中一个房子进入详情页

展示所有房屋图片和代理人信息，方便顾客联络



发布房源

可以上传图片

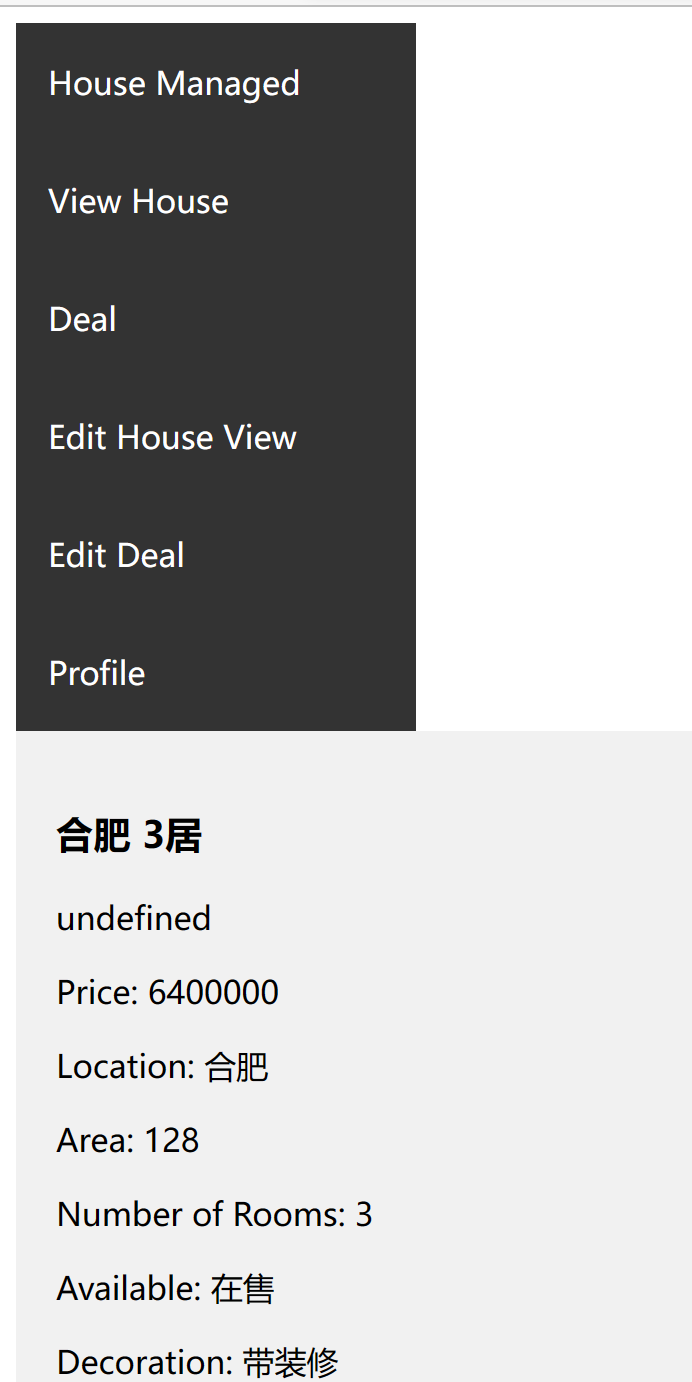


之后点击User进入用户界面

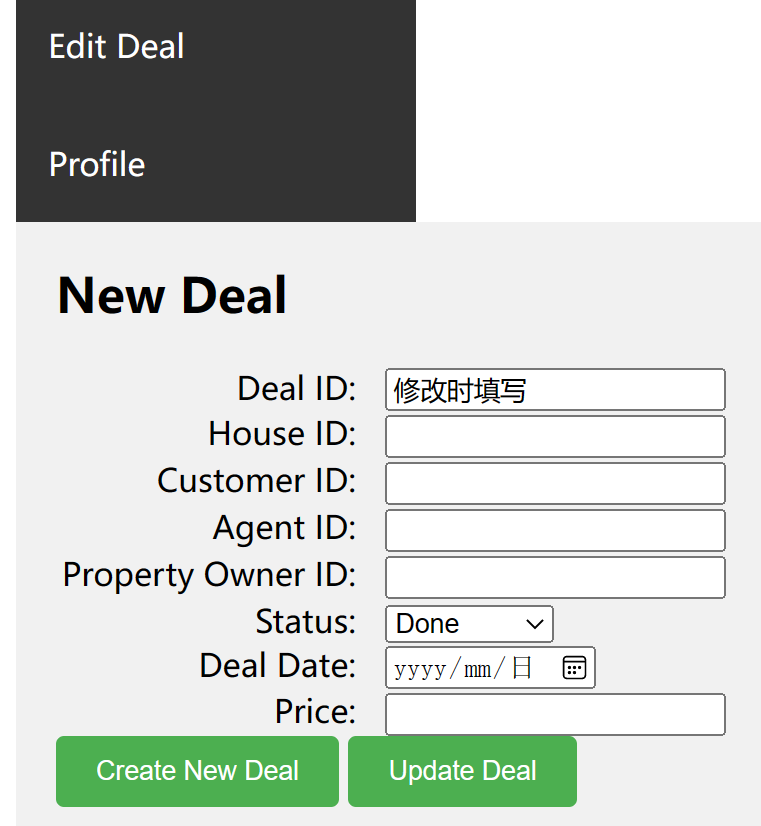
用户身份不同，进入的界面也不同

这是一个agent用户

点击导航栏进入不同板块，此处是查看管理的房屋



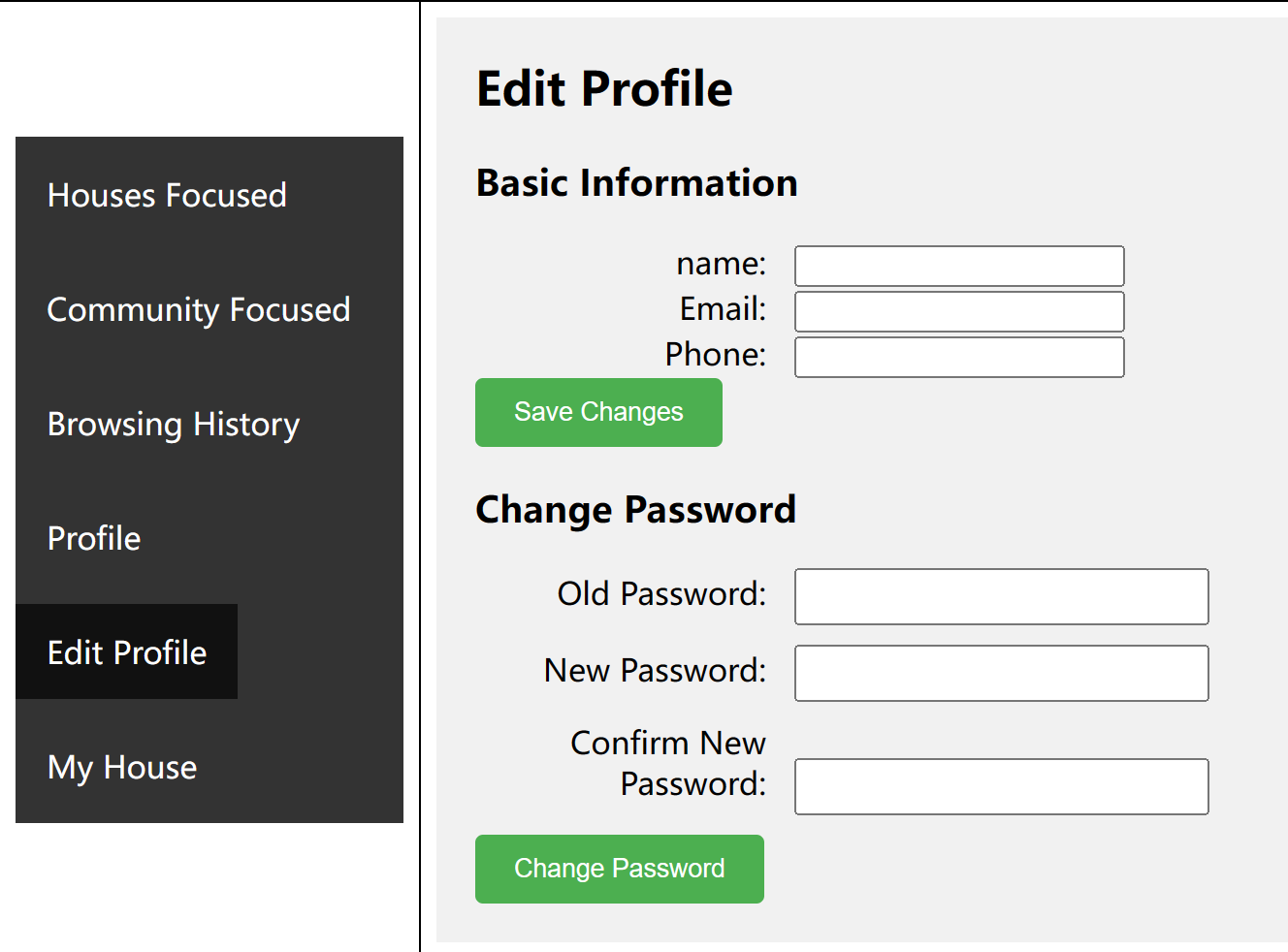
这里是添加或修改交易



还有房屋预约，个人信息等

这是一个客户界面

此处是编辑个人信息



如果你还是房主

可点击My House查看你的房子

