Place To Live: Home Shopping Reimagined

Jun Chen
Georgia State University

Josh Ritz
Georgia State University

Saahil Karnik
Georgia State University

Kemar Douglas
Georgia State University

Abstract

The era we live in is one where human life and technology have become so intertwined that developing new and innovative software to keep up with changing demand can be challenging. Software can become outdated and online business models can become stagnant. Further, given the current climate with keen focus on individual health and well being, we are seeing more people than ever interacting in an online capacity. Many businesses that were formerly in-person businesses have moved online as best as possible.[1] One thing that will always remain a necessity for people is shopping for a place to live. Prior to the pandemic, many people shopped in-person, going to real estate agencies or coordinating in-person tours of homes, attending open houses, and generally handling much of this process in-person. The existing websites that people can peruse in an effort to locate local real estate have been around for a long time, but many of them are limited in the online support they offer. With this research project, our team aims to explore new and innovative ways that we can improve upon the online experience for individuals looking for a place to live. Following the idea above, we have named our project and website "Place To Live". We used various resources for development ranging from HTML and CSS for the front end, to Python and PHP for the back end. While the architectural model and design for Places To Live is likely similar to existing websites (a layered architecture design pattern), one of our key focuses was incorporating a chat module into the website in order to take a people based approach.

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130. Copyright 2021 by the author(s).

1 Introduction

From a high-level perspective, our website (Place To Live) is a searchable database of properties listed for sale, lease, or rent. The website aims to connect individuals searching for a new place to live with realtors and private sellers or landlords. We recognize the need for shifting as much of the process of finding a place to live to an online space. Place To Live is being designed to appeal to any person or group of people looking for a property to buy, lease, or rent.

Place To Live solves the problem of endless searching for a property that meets the user's requirements. Home buying, or even renting, can be a hectic nightmare. Many people are first-time buyers or renters. Place To Live seeks to make the process for people in such a situation as simple as possible, and as personal as possible.

While alternatives to our website exist, such as Zillow, Forrent, and others, we seek to take the existing processes identified in these websites and simplify them. We want as smooth and simple a property purchasing process (or renting process) as possible. If a user gets frustrated with a website, it's possible they will discontinue use. Some of the market competitors don't have the simplest of processes. We seek to take advantage of this. Additionally, we recognize that browsing for a property, even online, can be a daunting and stressful task. One of the focuses of Place To Live is ensuring that users have access to real, live human beings that can assist them in their online experience and buying process.

An eventual goal of the project would be to connect with existing Real Estate agencies in effort to partner with them and offer them an online space where they can connect with customers. While some Real Estate agencies have their own websites, the majority of traffic as it relates to people shopping for a place to live, happens online. According to the NAR (National Association of Realtors) "**95%** of home buyers will search for their next home onilne." Additionally, where Zillow.com averages 173 million monthly visitors, Real Estate Agency websites like Remax.com only average

5.3 million.[2] This makes it crystal clear that Real Estate Agencies needs a better way of connecting with prospective buyers. They need to be able to tap into the market of 173 million monthly views. Our website seeks to model itself after websites like Zillow.com that receive those massive view numbers, and while Zillow has finally added an "agent finder" to their website, it is essentially just a google search by zip code of Real Estate Agents in that area. It does not actually connect the Zillow website users with those agents. Many of the agents that you can call yourself either have their own platforms for virtual meetings, or meet in person. Our aim is to ensure the website user can get everything in one place. No extra phone calls, no additional websites or software for virtual visits. We aim to create an all-in-one online experience.

Aside from the above idea of providing real-time connection with an agent that can assist in the search, we seek to standardize the level of information required to display a listing. Quite often, listings will merely have one picture of a property, no virtual walk-through, no additional interior pictures, and scant information about the property. One of the aims of our website is to set a standard of what information is required about a property for that property to be listed on Place To Live. The idea would be to eventually have associates or contractors whose job it is to collect information, pictures, and videos of the property to be used for a listing. This would eventually be offered as a for purchase service offered by the website.

The technology needed to create this system is readily available and we will be using a combination of HTML, CSS, PHP, Javascript, annd Python to impelment our designs. We will also use MySQL and AmazonAWS in order to create our databases and store user and property information. At this time, we have even reserved our web domain from Domain.com, were this website to go live, as PlaceToLive.net was indeed available.

Ultimately, the market for home buying, apartment renting, condo leasing, townhouse purchasing; the market for looking for a place to live is a multi-Billion dollar industry. This is not a niche' market, and we certainly see room for innovation in the marketplace. This research project allows us to explore new ways of approaching old problems. From a technical point of view, this project is interesting because we have an opportunity to take things we dislike about existing websites and improve upon them. We can identify processes within the home buying or renting experience that can be improved upon and streamlined, providing the simplest user experience possible. We can solicit feedback from users of popular sites and find out what features they would like to see that currently aren't offered. This research project allows us to work with system designs and models we are unfamiliar with, and work with coding languages some of us have worked with before. Therefore this is a useful, worthwhile, and valuable exercise in preparing for a position as a software developer.

2 Dataset analysis

Our group chooses to use the Amazon RDS database management system to implement our database and tables. The database management system is MySQL. According to Tony[3], Using MySQL has a lot of advantages such as data security, on-demand scalability, high performance, round-the-clock uptime, comprehensive transactional support, complete workflow control, reduce total cost of ownership, and the flexibility of open source. John [4] mentioned that MySQL is easy to use because the commands are not too much, and it is easy to install. The difficulty is that we need to remember how to use and when to use.

Our group uses the MySQL command to create a lot of tables such as chatters, messages, property_info, property_type, user_info, user_login, user_save, user_type. The chatters table will stores which is online. The messages table stores the chatting content. The property_info table stores information about the property. The property_type table will store the property type. The user_info table will store the user's information. The user_login will store the user's login information. The user_save table will store the user and the property. The user_type table will store the user's type.

2.1 Data Exploration

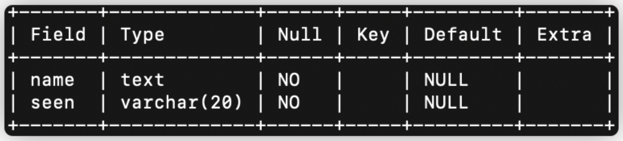
Different tables have different attributes and settings. The chatters table has two attributes, which are name and seen. The attribute name's type is text, and the attribute seen's type is varchar. The messages table contains three attributes, which are name, msg, and posted. The type of the attribute name and msg is text. The type of attribute posted is varchar.

The property_info table has 12 attributes. The attribute PropertyID's type is int. The attribute PtypeID's type is int. The attribute Address's type is varchar. The attribute City's type is varchar. The attribute State's type is varchar. The attribute Zip_Code's type is int. The attribute price's type is decimal. The attribute Features' type is varchar. The attribute Year's type is the year. The attribute Bedrooms' type is int. The attribute Bathrooms' type is decimal. The attribute Square_Feet's type is int. The property_type table has two attributes. The type of the attribute PtypeID is int. The type of attribute Type_Name is varchar.

The user_info has 11 attributes. The attribute UserID's type is int. The attribute TypeID's type is tinyint. The attribute First_Name's type is varchar. The attribute Last_Name's type is varchar. The attribute Address's type is varchar. The attribute City's type is varchar. The attribute State's type is varchar. The attribute Zip_Code's type is int. The attribute Phone_Number's type is int. The attribute Gender's type is ENUM. The attribute Marital_Status's type is ENUM. The user_type table has two attributes. One is TypeID, and the type is tinyint. The other is Type_Name, and the type is varchar. The user_login table has four attributes. The attribute UserID's type is int. The attribute User_Name's type is varchar. The attribute Hashed_Password's type is char. The attribute Email_Address's type is varchar. The user_save table has two attributes. One is UserID, which type is int. The other is PropertyID, and the type is int.

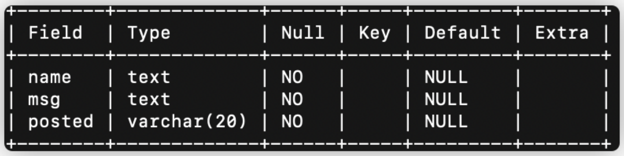
2.1.1 Visualization

By using the "CREAT" command in MySQL, we can get these tables. Then, we use "INSERT" to add some intial values.All the tables can be seen in Fig.1, Fig.2, Fig.3, Fig. 4, Fig.5, Fig.6, Fig.7, and Fig.8 The initial value tables can be found in these figures 9, 10, 11 and 12. We need to specific the property's type and set special index for the type. When we add the property's information, it will be very convenient for us to take actions. The type of the property is specified as House, Apartment, Townhouse and Other. Also, we specific the user type, so when the user update their user information, they can choose their type.



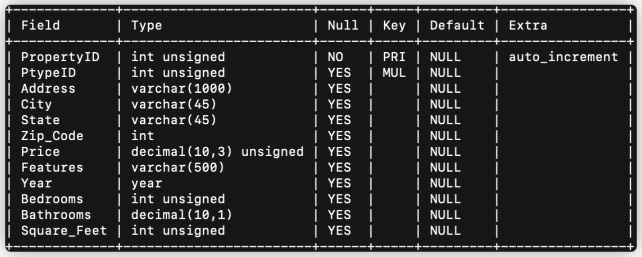
```
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| name  | text        | NO   |     | NULL    |       |
| seen  | varchar(20) | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Figure 1: chatters



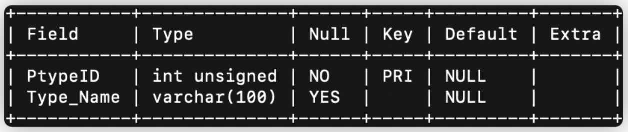
```
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| name   | text        | NO   |     | NULL    |       |
| msg    | text        | NO   |     | NULL    |       |
| posted | varchar(20) | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

Figure 2: messages



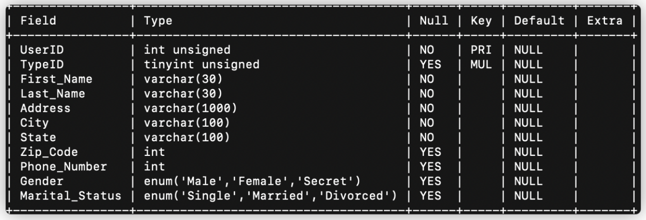
```
+-------------+-------------------------+------+-----+---------+----------------+
| Field       | Type                    | Null | Key | Default | Extra          |
+-------------+-------------------------+------+-----+---------+----------------+
| PropertyID  | int unsigned            | NO   | PRI | NULL    | auto_increment |
| PtypeID     | int unsigned            | YES  | MUL | NULL    |                |
| Address     | varchar(1000)           | YES  |     | NULL    |                |
| City        | varchar(45)             | YES  |     | NULL    |                |
| State       | varchar(45)             | YES  |     | NULL    |                |
| Zip_Code    | int                     | YES  |     | NULL    |                |
| Price       | decimal(10,3) unsigned  | YES  |     | NULL    |                |
| Features    | varchar(500)            | YES  |     | NULL    |                |
| Year        | year                    | YES  |     | NULL    |                |
| Bedrooms    | int unsigned            | YES  |     | NULL    |                |
| Bathrooms   | decimal(10,1)           | YES  |     | NULL    |                |
| Square_Feet | int unsigned            | YES  |     | NULL    |                |
+-------------+-------------------------+------+-----+---------+----------------+
```

Figure 3: property_info



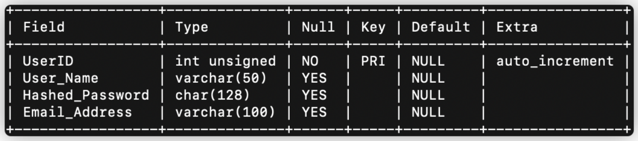
```
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| PtypeID   | int unsigned | NO   | PRI | NULL    |       |
| Type_Name | varchar(100) | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
```

Figure 4: property_type



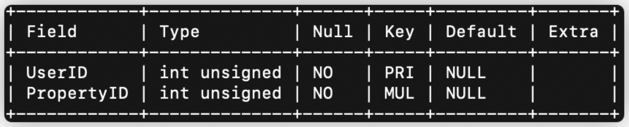
```
+----------------+------------------------------------+------+-----+---------+-------+
| Field          | Type                               | Null | Key | Default | Extra |
+----------------+------------------------------------+------+-----+---------+-------+
| UserID         | int unsigned                       | NO   | PRI | NULL    |       |
| TypeID         | tinyint unsigned                   | YES  | MUL | NULL    |       |
| First_Name     | varchar(30)                        | NO   |     | NULL    |       |
| Last_Name      | varchar(30)                        | NO   |     | NULL    |       |
| Address        | varchar(1000)                      | NO   |     | NULL    |       |
| City           | varchar(100)                       | NO   |     | NULL    |       |
| State          | varchar(100)                       | NO   |     | NULL    |       |
| Zip_Code       | int                                | YES  |     | NULL    |       |
| Phone_Number   | int                                | YES  |     | NULL    |       |
| Gender         | enum('Male','Female','Secret')     | YES  |     | NULL    |       |
| Marital_Status | enum('Single','Married','Divorced')| YES  |     | NULL    |       |
+----------------+------------------------------------+------+-----+---------+-------+
```

Figure 5: user_info



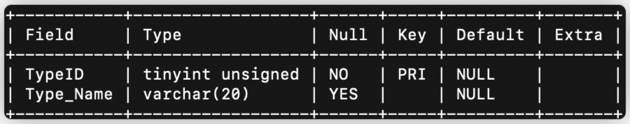
```
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| UserID          | int unsigned | NO   | PRI | NULL    | auto_increment |
| User_Name       | varchar(50)  | YES  |     | NULL    |                |
| Hashed_Password | char(128)    | YES  |     | NULL    |                |
| Email_Address   | varchar(100) | YES  |     | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
```

Figure 6: user_login



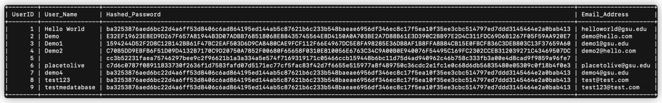
```
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| UserID     | int unsigned | NO   | PRI | NULL    |       |
| PropertyID | int unsigned | NO   | MUL | NULL    |       |
+------------+--------------+------+-----+---------+-------+
```

Figure 7: user_save



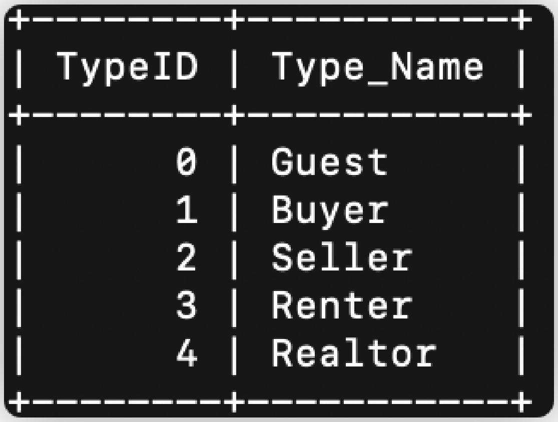
```
+-----------+------------------+------+-----+---------+-------+
| Field     | Type             | Null | Key | Default | Extra |
+-----------+------------------+------+-----+---------+-------+
| TypeID    | tinyint unsigned | NO   | PRI | NULL    |       |
| Type_Name | varchar(20)      | YES  |     | NULL    |       |
+-----------+------------------+------+-----+---------+-------+
```

Figure 8: user_type



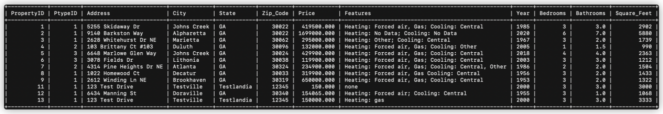
```
+--------+----------------+----------------------------------------------------------------------------------------------------------------------------------+----------------------+
| UserID | User_Name      | Hashed_Password                                                                                                                  | Email_Address        |
+--------+----------------+----------------------------------------------------------------------------------------------------------------------------------+----------------------+
|      1 | Hello World    | ba3253876aed6bc22d4a6ff53d8406c6ad864195ed144ab5c87621b6c233b548baeae6956df346ec8c17f5ea10f35ee3cbc514797ed7ddd3145464e2a0bab413 | helloworld@gsu.edu   |
|      2 | Demo           | E32EF19623E8ED9D267F657A81944B3D07ADBB768518068E88435745564E8D4150A0A703BE2A7D88B61E3D390C2BB97E2D4C311FDC69D6B1267F05F59AA920E7 | demo@hello.com       |
|      3 | Demo1          | 1594244D52F2D8C12B142BB61F47BC2EAF503D6D9CA8480CAE9FCF112F66E4967DC5E8FA98285E36DB8AF1B8FFA8B84CB15E0FBCF836C3DEB803C13F37659A60 | demo1@gsu.edu        |
|      4 | Demo2          | C70B5DD9EBFB6F51D09D4132B7170C9D20750A7852F00680F65658F0310E810056E6763C34C9A00B0E940076F54495C169FC2302CCEB312039271C43469507DC | demo2@hello.com      |
|      5 |                | cc3b52231faea75746297bee9c2f96621b1a3a334a5e574f7169319171c05466ccb159448b6bc11d75d4ad940962c46b758c333fb3a00e4d8cad9f9859a96fe7 |                      |
|      6 | placetolive    | c7d6c0787f08911833730f2636f1d7583fafd07d5171ec77cf5fac83f42d7f6655e515977a8f489750c36cdc2e1fc1e0c68d6db56835480e05309c0f18b4f0e3 | placetolive@gsu.edu  |
|      7 | demo4          | ba3253876aed6bc22d4a6ff53d8406c6ad864195ed144ab5c87621b6c233b548baeae6956df346ec8c17f5ea10f35ee3cbc514797ed7ddd3145464e2a0bab413 | demo4@gsu.edu        |
|      8 | test123        | ba3253876aed6bc22d4a6ff53d8406c6ad864195ed144ab5c87621b6c233b548baeae6956df346ec8c17f5ea10f35ee3cbc514797ed7ddd3145464e2a0bab413 | test@test.com        |
|      9 | testmedatabase | ba3253876aed6bc22d4a6ff53d8406c6ad864195ed144ab5c87621b6c233b548baeae6956df346ec8c17f5ea10f35ee3cbc514797ed7ddd3145464e2a0bab413 | test123@test.com     |
+--------+----------------+----------------------------------------------------------------------------------------------------------------------------------+----------------------+
```

Figure 9: user_login



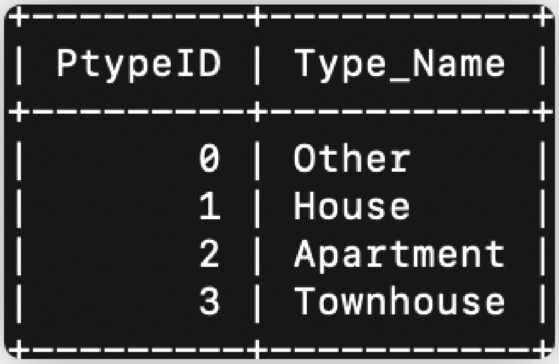
```
+--------+-----------+
| TypeID | Type_Name |
+--------+-----------+
|      0 | Guest     |
|      1 | Buyer     |
|      2 | Seller    |
|      3 | Renter    |
|      4 | Realtor   |
+--------+-----------+
```

Figure 10: user_type



```
+------------+---------+------------------------+--------------+------------+----------+-------------+------------------------------------------------------+------+----------+-----------+-------------+
| PropertyID | PtypeID | Address                | City         | State      | Zip_Code | Price       | Features                                             | Year | Bedrooms | Bathrooms | Square_Feet |
+------------+---------+------------------------+--------------+------------+----------+-------------+------------------------------------------------------+------+----------+-----------+-------------+
|          1 |       1 | 5255 Skidaway Dr       | Johns Creek  | GA         |    30022 |  419500.000 | Heating: Forced air, Gas; Cooling: Central           | 1985 |        3 |       3.0 |        2902 |
|          2 |       1 | 9140 Barkston Way      | Alpharetta   | GA         |    30022 | 1699000.000 | Heating: No Data; Cooling: No Data                   | 2020 |        6 |       7.0 |        5880 |
|          3 |       1 | 2628 Whitehurst Dr NE  | Marietta     | GA         |    30062 |  295000.000 | Heating: Other; Cooling: Central                     | 1967 |        3 |       2.0 |        1739 |
|          4 |       2 | 103 Brittany Ct #103   | Duluth       | GA         |    30096 |  132000.000 | Heating: Forced air, Gas; Cooling: Other             | 2005 |        1 |       1.5 |         990 |
|          5 |       3 | 6648 Marlowe Glen Way  | Johns Creek  | GA         |    30024 |  429900.000 | Heating: Forced air, Gas; Cooling: Central           | 2018 |        4 |       4.0 |        2363 |
|          6 |       3 | 3078 Fields Dr         | Lithonia     | GA         |    30038 |  119900.000 | Heating: Forced air, Gas; Cooling: Central           | 2003 |        3 |       3.0 |        1212 |
|          7 |       2 | 4314 Pine Heights Dr NE | Atlanta     | GA         |    30324 |  234900.000 | Heating: Forced air, Gas; Cooling: Central, Other    | 1986 |        2 |       2.0 |        1504 |
|          8 |       1 | 1022 Homewood Ct       | Decatur      | GA         |    30033 |  319900.000 | Heating: Forced air, Gas Cooling: Central            | 1956 |        3 |       2.0 |        1433 |
|          9 |       1 | 2612 Winding Ln NE     | Brookhaven   | GA         |    30319 |  650000.000 | Heating: Forced air, Gas; Cooling: Central           | 1953 |        3 |       2.0 |        1322 |
|         11 |       1 | 123 Test Drive         | Testville    | Testlandia |    12345 |     150.000 | none                                                 | 2000 |        3 |       3.0 |        3000 |
|         12 |       1 | 6434 Manning St        | Doraville    | GA         |    30340 |  154065.000 | Heating: Forced air; Cooling: Central                | 1955 |        3 |       1.0 |        1068 |
|         13 |       1 | 123 Test Drive         | Testville    | Testlandia |    12345 |  150000.000 | Heating: gas                                         | 2000 |        3 |       3.0 |        3333 |
+------------+---------+------------------------+--------------+------------+----------+-------------+------------------------------------------------------+------+----------+-----------+-------------+
```

Figure 11: property_info

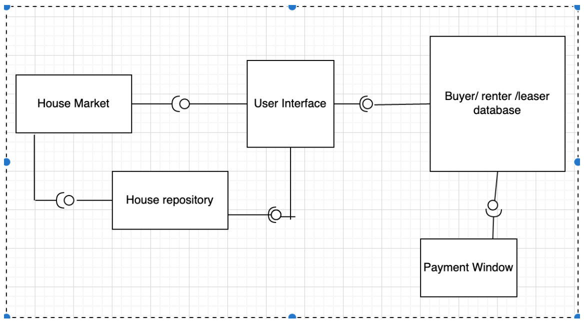


```
+---------+-----------+
| PtypeID | Type_Name |
+---------+-----------+
|       0 | Other     |
|       1 | House     |
|       2 | Apartment |
|       3 | Townhouse |
+---------+-----------+
```

Figure 12: property_type

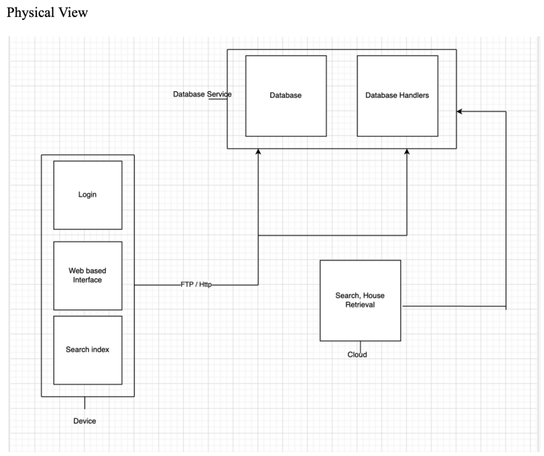
3 Architectural Model

3.1 Logical View



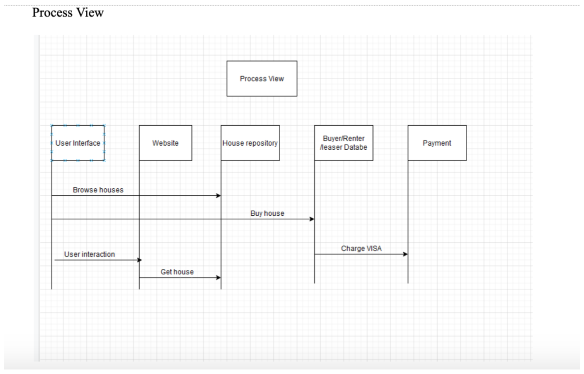
Logical View

3.1.1 Physical View



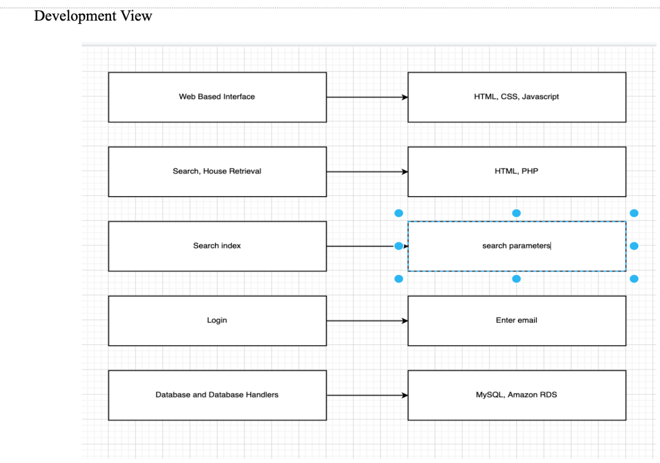
Physical View

3.1.2 Process View



Process View

3.1.3 Development View



Development View

3.1.4 Use case Scenarios

1) Interact with the website
2) Make payments
3) Access list of buyers/sellers/renters/lessees
4) Access list of properties

3.1.5 Architectural Design Pattern

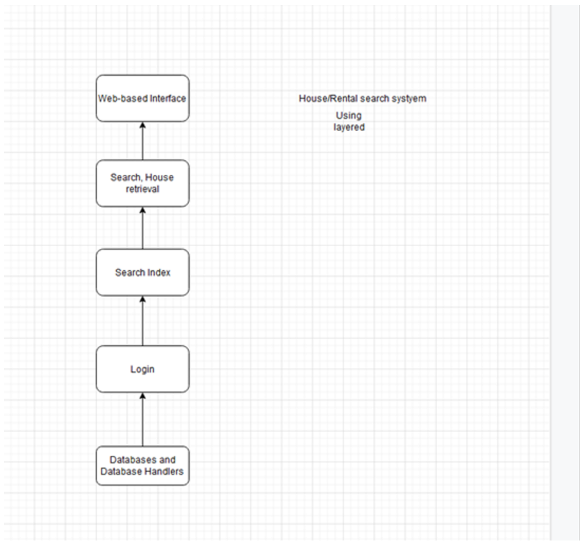


Figure 13: Architectural Design Pattern

Figure 13 displays our project's architectural design pattern. The architectural design pattern for our system is a layered architecture pattern. The first layer is the web-based interface. The second layer is search house retrieval, the third layer is search index, the fourth layer is login, and the lowest layer is databases and database handlers.

3.1.6 Class Diagram

Figure 14, 15, 16, and 17 are our project's class diagram. The class diagram can show the system's structure, class, and attributes. Let's look at Figure 14. This is communication system class diagram. The system admin can control one or more user and this action means that if the atmosphere of communication between the users is terrible such as conflict, the system admin has the right to stop the communication and bans several user account. The user has several types such as buyer, seller, renter and realtor and they can communicate with each other.

Figure 15 is search for the property class diagram. All users such as buyer, seller, renter or realtor can type the zip code or city name to search the property. One or more users can type the zip code or city name and use property database to get one or more results. Figure 16 is create a user account class diagram. Buyer, seller, renter and realtor can create their account by typing their user name, password, email address and they also can type their personal information. Figure 17 is advertising the property class diagram. The users can choose to advertise their property and the website will display the property's information.

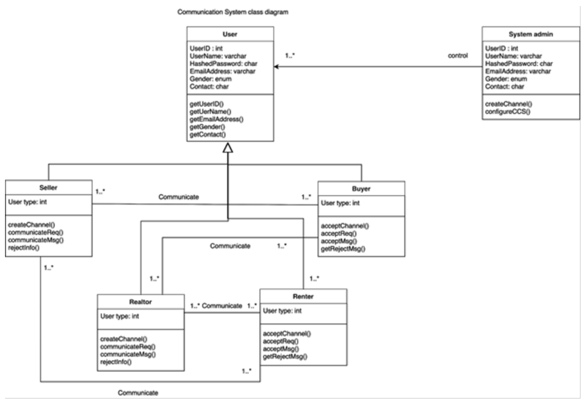


Figure 14: Communication system class diagram

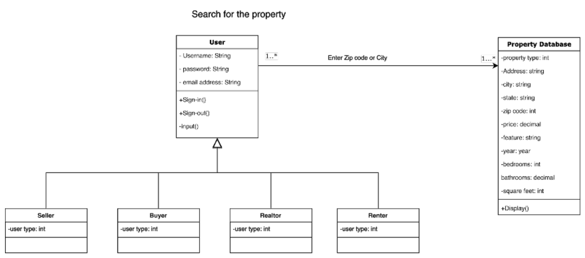


Figure 15: Search for the property class diagram

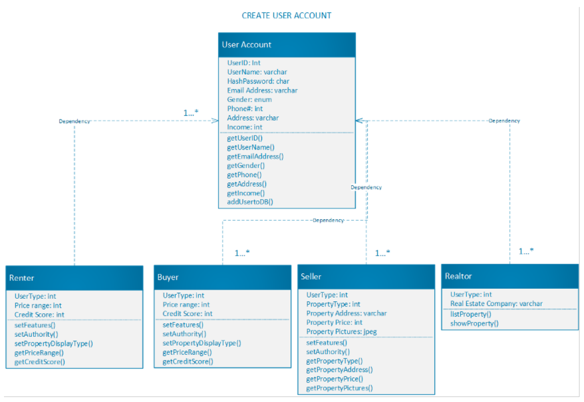


Figure 16: Create user account class diagram

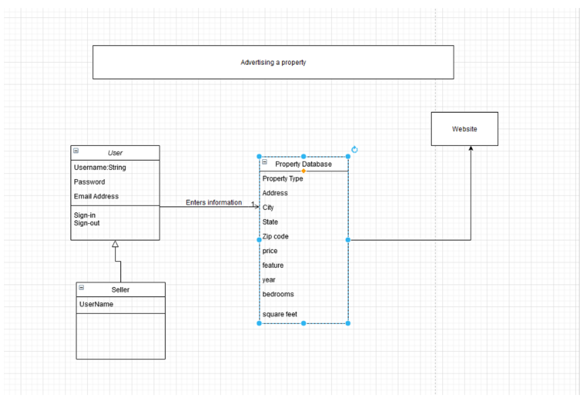


Figure 17: Advertising a property class diagram

3.1.7 Design Pattern

Figure 18, 19, 20, and 21 are our class diagram's design pattern. Figure 18 is Search for the property class diagram and it is easy to connect to other use cases. It is loosely coupled. This class diagram has one purpose which is searching. It means that the search for the property has high cohesion. The design pattern is chosen to use the Strategy pattern.

Figure 19 is Communication class diagram and it can connect to other use cases easily. It is loosely coupled. This class diagram has one purpose, which is chatting. It means that it is highly cohesive. The design pattern is chosen to use the Bridge Pattern.

Figure 20 is Advertising a property and it is not tough to connect Advertising a property class diagram to other use cases. This class diagram is loosely coupled. The purpose of this class diagram is to show the property. It means that the class diagram has high cohesion. The advertising of a property class diagram uses the singleton pattern as its design pattern.

Figure 21 is create a user account and create a new user design pattern is like a factory to churn out a new user each time someone creates a new account. There are three subclasses of the user; buyer/renter, seller, and realtor. The factory can quickly churn out one of any of the three user types quickly and effectively. The design is loosely coupled and has high cohesion.

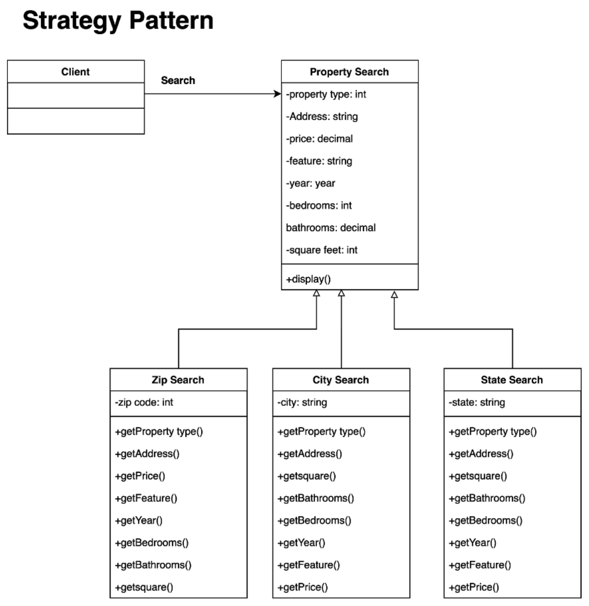


Figure 18: search for the property class diagram with Strategy Pattern (Behavioral)

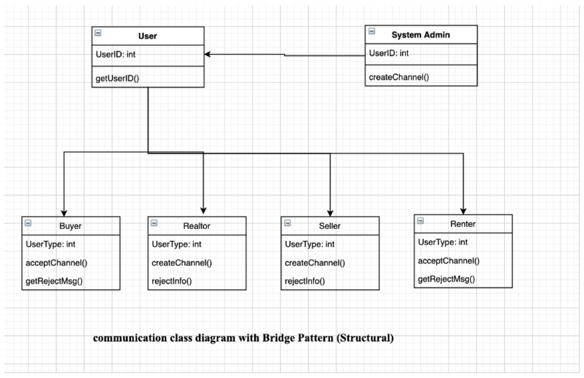


Figure 19: Communication class diagram with Bridge Pattern(Structural)

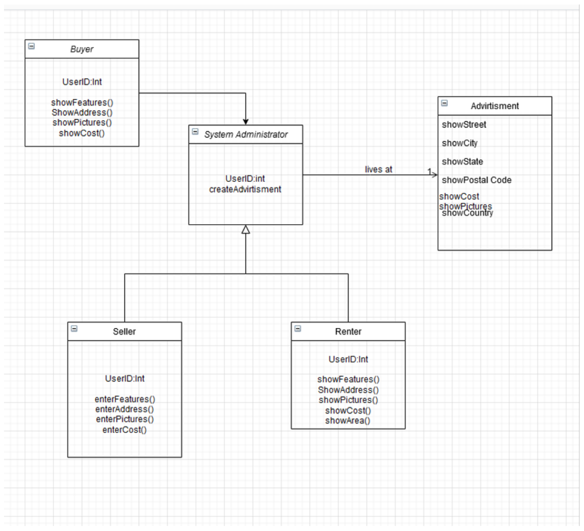


Figure 20: Advertising a property with the singleton pattern (Creational)

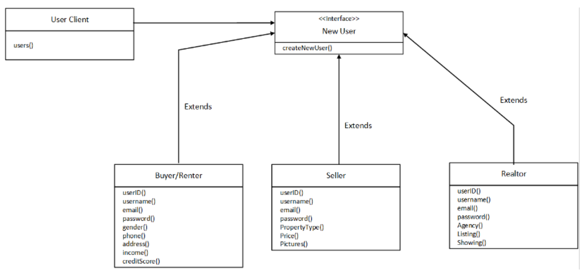


Figure 21: Create a new user (Creative design pattern – Abstract Factory Pattern)

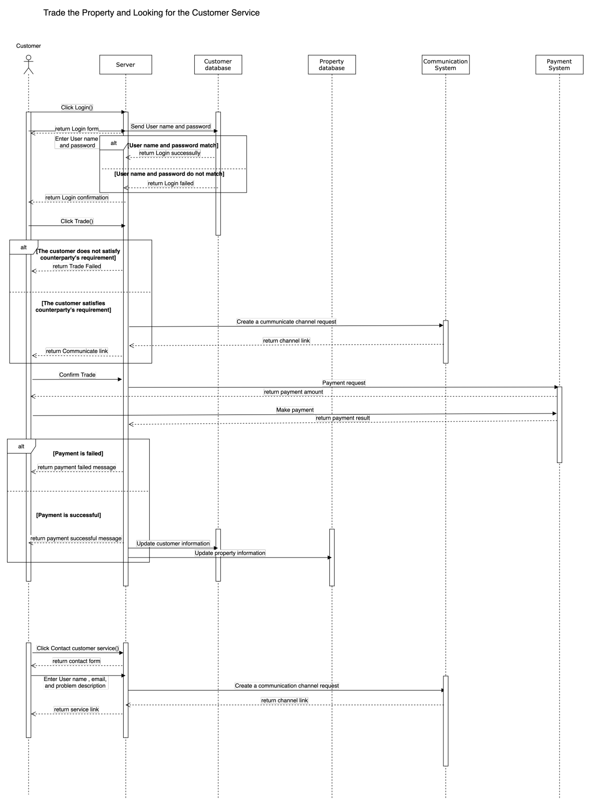

Trade the Property and Looking for the Customer Service
Customer
Server
Customer database
Property database
Communication System
Payment System
Click Login()
return Login form
Send User name and password
Enter User name and password
alt
[User name and password match]
return Login successully
[User name and password do not match]
return Login failed
return Login confirmation
Click Trade()
alt
[The customer does not satisfy counterparty's requirement]
return Trade Failed
[The customer satisfies counterparty's requirement]
Create a cummunicate channel request
return channel link
return Communicate link
Confirm Trade
Payment request
return payment amount
Make payment
return payment result
alt
[Payment is failed]
return payment failed message
[Payment is successful]
return payment successful message
Update customer information
Update property information
Click Contact customer service()
return contact form
Enter User name , email, and problem description
Create a communication channel request
return channel link
return service link

Figure 22: Trade the Property and Looking for the Customer Service

3.1.8 Behavioral Modeling

Figure 22 shows the sequence diagram of trading the property and looking for the customer service. Before trading the property, the customer needs to login into the system. Then, the customer can click the trade button. If the customer satisfies the counterparty's requirement, the system will create a communication channel for them to hold the final discussion. Otherwise, the trade will be over. After the discussion, if the customer confirms the trade, the server will send the request to the payment system. The payment system will send the payment amount to the customer. Once the customer makes the payment, the payment system will check the result. If the payment is failed, the customer will receive the failed message. Otherwise, the customer will receive a successful message. Then, the server will update the customer information and property information.

Also, when the customer clicks the Contact customer service, the system will send a form to the customer. The customer types the username, email address, and the problem description. Then, the server will send the create communication channel request to the communication system. The communication system will generate a link and send it to the server. The customer service will use this link to join. Then, the server will send this link to the customer.

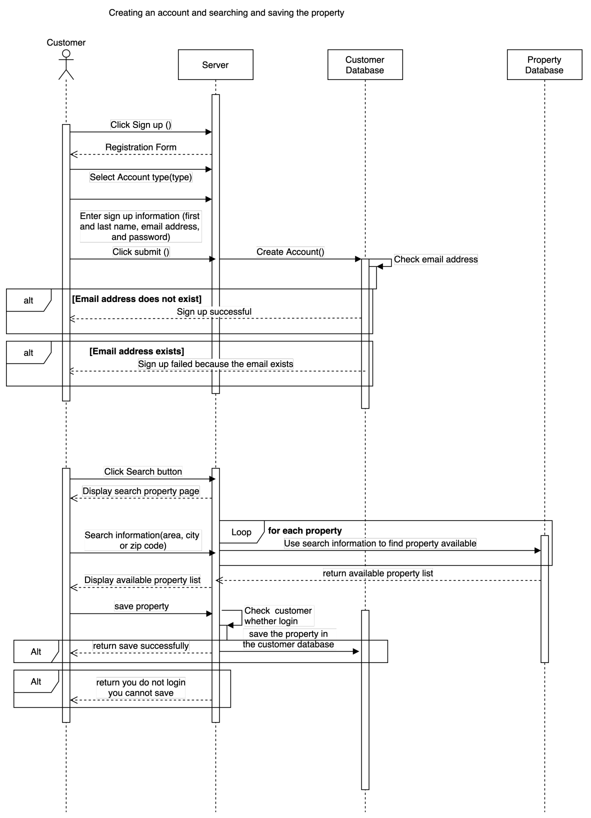

Creating an account and searching and saving the property
Customer
Server
Customer Database
Property Database
Click Sign up ()
Registration Form
Select Account type(type)
Enter sign up information (first and last name, email address, and password)
Click submit ()
Create Account()
Check email address
alt
[Email address does not exist]
Sign up successful
alt
[Email address exists]
Sign up failed because the email exists
Click Search button
Display search property page
Search information(area, city or zip code)
Loop
for each property
Use search information to find property available
return available property list
Display available property list
save property
Check customer whether login
save the property in the customer database
Alt
return save successfully
Alt
return you do not login you cannot save

Figure 23: Creating an account and Searching and Saving the property

Figure 23 shows 2 use cases which are creating an account and searching and saving the property. The first use case is creating an account. When the customer clicks the sign-up button, the website server will return a page, which is a registration form. Then, the customer should select the account type and enter the basic sign up information such as first name, last name, email address, and password. Then, the customer should click the submit button, and the customer database will check the email address whether it exists. If the email address does not exist, the customer will receive sign up successful message. If the email address exists, the customer will receive sign up failure.

The second use case is searching and saving the property. The customer clicks the search button, and the website server will display the search page. The customer types the searching parameters such as city or zip code. Then, the property database will search each property to find the one that fits the requirements and display the results. If the customer clicks the save button, the website server will check whether the customer login. If the customer logs in, the customer database will update and notify the customer that the save action is successful. If the customer does not log in, the website server will send the message to the customer that the customer has to log in to save the property.

Acknowledgments

First of all, we would like to extend our sincere gratitude to Dr. Sudha Tushara Sadasivuni, for her instructive advice and useful suggestions on our project. We are deeply grateful of her help in the completion of this project.

We are also deeply indebted to our group members. They work hard and cooperate to complete this project. They draw diagrams, write reports, revise the diagrams and reports. They made a great contribution.

We would like to thank and acknowledge the developers that make projects like this possible by providing free software and database hosting such as Amazon aws or software required to work with the databases like phpmyadmin, MySql Workbench, and XAMPP.

To conclude, we hope that we can inspire each other as well as other developers to take the initiative in working on projects that interest them. We hope we can inspire others and push ourselves in order to add something valuable to the digital marketplace that makes others' lives better.

References

[1] T. Walk-Morris, "How the covid-19 pandemic is pushing brands to connect digitally," *https://www.retaildive.com/news/how-the-covid-19-pandemic-is-pushing-brands-to-connect-digitally/576835/ April 28, 2020.*

[2] S. FitzMaurice, "Top 10 real estate websites by traffic – 2020 update," *https://realestateagentpdx.com/top-10-real-estate-websites-by-traffic-2020-update/16956 February 14, 2020.*

[3] T. Branson, "8 major advantages of using mysql," Nov 2016.

[4] J. Mack, "Five advantages & disadvantages of mysql," May 2014.