# SUMMARY

Topic title: Real estate business

Student name: Nguyen Duc Van

Student ID: 102210195. Class: 21TCLC\_DT2

|  |  |
| --- | --- |
| DA NANG UNIVERSITY  **UNIVERSITY OF SICIENCE AND TECHNOLOGY**  FALCUTY………………………………………… | **THE SOCIALIST REPUBLIC OF VIETNAM**  Independence - Freedom - Happiness |

# GRADUATION PROJECT REQUIREMENTS

Student Name: …..…………….………….…….. Student ID :………………...

Class:…………… Faculty:..................................... Major:…………...........................

1. *Topic title:*

………………………………………………..…………………………………………

…………………………………………………………………………………………..

1. *Project topic :* ☐*has signed intellectual property agreement for final result*
2. *Initial figure and data:*

……………………………………..……………………………………………..……......……………………………………………………………………………………………..………………………………….…..………………………..………………………*Content of the explanations and calculations:*

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

1. *Drawings, charts (specify the types and sizes of drawings):*

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

…...………………………………………………………………………………………

|  |  |
| --- | --- |
| 1. Name of instructor: | *Content parts:* |
|  |  |
|  |  |
|  |  |

1. *Date of assignment: ……../……./201…..*
2. *Date of completion: ……../……./201…..*

|  |  |
| --- | --- |
|  | *Đà Nẵng, date month year 201* |
| **Head of Division**…………………. | **Instructor** |

# PREFACE

# ASSURRANCE

I hereby declare that:

1. The graduation project report, Topic name: Real Estate Business System, is my own research work under the direct guidance of the lecturer MSc. Do Thi Tuyet Hoa.
2. I have read, researched, translated documents, and synthesized the knowledge that has made this report, and ensured that it has not been copied anywhere.
3. The theories in the thesis are all used from the documents as I have referred to in the reference section in the report.

If there is any violation, I will take full responsibility.

Student Performed

Nguyen Duc Van

# TABLE OF CONTENTS

[SUMMARY 3](#_Toc198219523)

[GRADUATION PROJECT REQUIREMENTS 4](#_Toc198219524)

[PREFACE i](#_Toc198219525)

[ASSURRANCE ii](#_Toc198219526)

[TABLE OF CONTENTS iii](#_Toc198219527)

[LIST OF PICTURES v](#_Toc198219528)

[LIST OF TABLES vii](#_Toc198219529)

[LIST OF SYMBOLS, ACRONYMS viii](#_Toc198219530)

[INTRODUCTION 1](#_Toc198219531)

[1. Purpose of implementation 1](#_Toc198219532)

[2. Objective of the topic 1](#_Toc198219533)

[2.1. System Target 1](#_Toc198219534)

[2.2. Features 1](#_Toc198219535)

[3. Scope 2](#_Toc198219536)

[4. Target customers 2](#_Toc198219537)

[5. Development technology 2](#_Toc198219538)

[CHAPTER 1: THEORETICAL FOUNDATION 3](#_Toc198219539)

[1.1. Overview of Python and Jupyter Notebook 3](#_Toc198219540)

[1.1.1. Python 3](#_Toc198219541)

[1.1.2. Jupyter Notebook 3](#_Toc198219542)

[1.2. Overview of Java and MySQL 4](#_Toc198219543)

[1.2.1. Java 4](#_Toc198219544)

[1.2.2. Spring Boot 5](#_Toc198219545)

[1.2.3. MVC model 5](#_Toc198219546)

[1.2.4. MySQL 8](#_Toc198219547)

[1.3. Overview of ReactJS and ChakraUI 9](#_Toc198219548)

[1.3.1. React 9](#_Toc198219549)

[1.3.2. ChakraUI 12](#_Toc198219550)

[1.4. Chapter conclusion 12](#_Toc198219551)

[CHAPTER 2: SYSTEM DESIGN AND ANALYSIS 13](#_Toc198219552)

[2.1. Business analysis 13](#_Toc198219556)

[2.1.1. Buyer's business 13](#_Toc198219557)

[2.1.2. Broker's business 13](#_Toc198219558)

[2.1.3. Admin's business 13](#_Toc198219559)

[2.2. System design 14](#_Toc198219560)

[2.2.1. Use-case diagram 14](#_Toc198219561)

[2.2.2. Activity diagram 18](#_Toc198219562)

[2.2.3. Sequence diagram 19](#_Toc198219563)

[2.3. Database design 23](#_Toc198219564)

[2.3.1. ER (entity-relationship) 23](#_Toc198219565)

[2.3.2. Converting an implementation model into a system model 23](#_Toc198219566)

[2.3.3. Database schema 29](#_Toc198219567)

[2.4. Real estate prediction system development 30](#_Toc198219568)

[2.4.1. Data crawling 30](#_Toc198219569)

[2.4.2. Data cleaning 30](#_Toc198219570)

[2.4.3. Feature Engineering 37](#_Toc198219571)

[2.4.4. Real estate valuation model 39](#_Toc198219572)

[2.4.5. Real estate price forecast model 40](#_Toc198219573)

[2.5. Apply NSFW image and toxic content classification models 40](#_Toc198219574)

[2.5.1. NSFW image classification models 40](#_Toc198219575)

[2.6. Chapter conclusion 41](#_Toc198219576)

[CHAPTER 3: IMPLEMENTATION AND RESULTS EVALUATION 41](#_Toc198219577)

[3.1. Description of achieved functionalities 41](#_Toc198219578)

[3.2. Chapter conclusion 41](#_Toc198219579)

[CHAPTER 4: CONCLUSION AND DEVELOPMENT DIRECTION 42](#_Toc198219580)

[REFERENCES 43](#_Toc198219581)

[APPENDIX 1 1](#_Toc198219582)

[APPENDIX 2 2](#_Toc198219583)

# LIST OF PICTURES

[Figure 1.1 Python 3](#_Toc198218931)

[Figure 1.2 Jupyter 4](#_Toc198218932)

[Figure 1.3 Java 4](#_Toc198218933)

[Figure 1.4 The compilation and interpretation process of Java 5](#_Toc198218934)

[Figure 1.5 Spring Boot 5](#_Toc198218935)

[Figure 1.6 Flow of operation in the MVC model 6](#_Toc198218936)

[Figure 1.7 Three-tier architecture 7](#_Toc198218937)

[Figure 1.8 Three-Tier architecture vs MVC pattern 8](#_Toc198218938)

[Figure 1.9 MySQL 8](#_Toc198218939)

[Figure 1.10 React 9](#_Toc198218940)

[Figure 1.11 How React uses Virtual DOM 10](#_Toc198218941)

[Figure 1.12 ReactJS lifecycle 11](#_Toc198218942)

[Figure 1.13 Chakra UI 12](#_Toc198218943)

[Figure 2.1 General use case diagram 14](#_Toc198218944)

[Figure 2.2 User management use-case diagram 14](#_Toc198218945)

[Figure 2.3 Post moderation use case diagram 15](#_Toc198218946)

[Figure 2.4 Profile management use case diagram 15](#_Toc198218947)

[Figure 2.5 Account upgrade management use case diagram 16](#_Toc198218948)

[Figure 2.6 Post management use case diagram 16](#_Toc198218949)

[Figure 2.7 Favourite post management use case diagram 17](#_Toc198218950)

[Figure 2.8 Client Operation Principle 18](#_Toc198218951)

[Figure 2.9 Admin Operation Principle 18](#_Toc198218952)

[Figure 2.10 Registration/Login Flow 19](#_Toc198218953)

[Figure 2.11 Real estate article management flow as a user 21](#_Toc198218954)

[Figure 2.12 Account upgrade flow 22](#_Toc198218955)

[Figure 2.13 The ER entity relationship model 23](#_Toc198218956)

[Figure 2.14 Relational Model 29](file:///D:\VisualStudioCode\Project\RealEstate\GraduationProjectReport\MyReport\PL04_B_RuotThuyetMinh.docx#_Toc198218957)

[Figure 2.15 Statistics to check features with values less than 0 30](#_Toc198218958)

[Figure 2.16 Statistics of missing values 30](#_Toc198218959)

[Figure 2.17 Distribution of rows based on the number of missing values 31](#_Toc198218960)

[Figure 2.18 Width distribution before and after preprocessing 31](#_Toc198218961)

[Figure 2.19 Length distribution before and after preprocessing 32](#_Toc198218962)

[Figure 2.20 Direction distribution before and after preprocessing 32](#_Toc198218963)

[Figure 2.21 Property features distribution before and after preprocessing 33](#_Toc198218964)

[Figure 2.22 Floor distribution before and after preprocessing 33](#_Toc198218965)

[Figure 2.23 Toilet distribution before and after preprocessing 34](#_Toc198218966)

[Figure 2.24 Furnishing sell distribution before and after preprocessing 34](#_Toc198218967)

[Figure 2.25 Real estate valuation model features 35](#_Toc198218968)

[Figure 2.26 Real estate price forecast features 35](#_Toc198218969)

[Figure 2.27 Histogram plots of feature distributions 36](#_Toc198218970)

[Figure 2.28 Visual Comparison of Outlier Impact on Feature Distributions 36](#_Toc198218971)

[Figure 2.29 Q-Q Plots after optimal normalization per feature 37](#_Toc198218972)

[Figure 2.30 Comparison of Q-Q Plots of Price per m² across different transformation methods 38](#_Toc198218973)

[Figure 2.31 Scatter plot of features vs. Price 38](#_Toc198218974)

[Figure 2.32 Selected features in order from best to worst. 39](#_Toc198218975)

[Figure 2.33 Optimal number of features 39](#_Toc198218976)

[Figure 2.34 Recursive Feature Elimination with correlated features 39](#_Toc198218977)

# LIST OF TABLES

[Table 2.1 Describe role table attributes 14](#_Toc197611443)

[Table 2.2 Describe user table attributes 15](#_Toc197611444)

[Table 2.3 Describe token table attributes 15](#_Toc197611445)

[Table 2.4 Describe transaction histories table attributes 16](#_Toc197611446)

[Table 2.5 Describe favourite properties table attributes 16](#_Toc197611447)

[Table 2.6 Describe category table attributes 16](#_Toc197611448)

[Table 2.7 Describe property table attributes 17](#_Toc197611449)

[Table 2.8 Describe house table attributes 17](#_Toc197611450)

[Table 2.9 Describe furnished status table attributes 17](#_Toc197611451)

[Table 2.10 Describe type of house table attributes 18](#_Toc197611452)

[Table 2.11 Describe house characteristics table attributes 18](#_Toc197611453)

[Table 2.12 Describe house characteristic mapping table attributes 18](#_Toc197611454)

[Table 2.13 Describe land table attributes 18](#_Toc197611455)

[Table 2.14 Describe type of land table attributes 18](#_Toc197611456)

[Table 2.15 Describe land characteristics table attributes 19](#_Toc197611457)

[Table 2.16 Describe land characteristic mapping table attributes 19](#_Toc197611458)

# LIST OF SYMBOLS, ACRONYMS

|  |  |
| --- | --- |
| Symbol, Acronym | Detailed explanation |
| PMML | Predictive Model Markup Language |
| HTTPS | HyperText Transfer Protocol Secure |
| PKL | Extension for files created with Pickle |
| API | Application Programming Interface |
| ONNX | Open format for storing machine learning models |
| NSFW | Not safe for work |
| XML | eXtensible Markup Language |
| CSS | Cascading Style Sheets |
| HTML | HyperText Markup Language |
| WAR | Web Application Archive |
| JPA | Java Persistence API |
| DOM | Document Object Model |
| **ACID** | Atomicity, Consistency, Isolation, Durability |
| **JSX** | JavaScript XML |

# INTRODUCTION

## Purpose of implementation

* Searching and buying real estate today is difficult, especially in determining the real value of the property, comparing suitable options and accessing transparent information. Buyers often spend a lot of time searching for properties that meet their needs in terms of location, area, amenities and finance, while sellers have difficulty reaching potential customers.
* Therefore, an online real estate business system can help display information visually, support filtering and comparing options, and integrate price prediction technology to help buyers and sellers make more accurate decisions

## Objective of the topic

### System Target

* Help users quickly post and search for suitable real estate based on filters with many characteristics.
* The real estate valuation feature provides a reference price close to the market, helping users minimize the risk of incorrect valuation, supporting the brokerage team or real estate sellers and buyers to save time in valuation and negotiation.
* The system will also support predicting real estate fluctuations to support customers' buying or selling decisions.
* Integrating VNPay, safe and convenient online payment when customers want to upgrade their account to use more features of the application.

### Features

* For guest
  + Search and filter by needs: price, area, location, property type, ...
  + View property details with images, prices and legal information, ...
  + Displays detailed information of the property along with the price determined by the property valuation system.
* For member
  + Post real estate with full information: images, descriptions, prices, legal status, ...however, the number of posts will be limited to 3.
  + You can save posts that interest you as well as upgrade your account to be able to use more features of the application.
* For brokers
  + Customers with upgraded accounts will be supported by the system to forcast real estate price in the future.
  + The number of posts posted by the brokerage account will also be up to 30.
* For admins:
  + Update account status.
  + View article details and moderate it.

## Scope

* Property type: house, land.
* Geographical scope: Da Nang
* Language: Vietnamese

## Target customers

* Real Estate Buyers/Sellers
* Real Estate Companies or Brokers

## Development technology

* Project management: Github
* Language: Java, Python, Reactjs
* Framework: Spring boot
* Database: Mysql
* Encryption & security: Security, HTTPS Token

# THEORETICAL FOUNDATION

## Overview of Python and Jupyter Notebook

### Python

Python is a widely used high-level programming language applied in web development, software engineering, data science, and machine learning (ML). Developers favor it due to its simplicity, efficiency, and cross-platform compatibility.

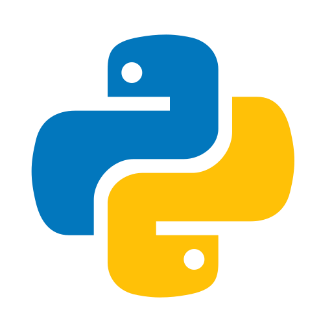


Figure 1.1 Python

**Benefits of Python include:**

* Developers can easily read and understand Python programs because the language has a simple syntax that resembles English.
* Python improves developer productivity as fewer lines of code are typically needed to write a program compared to other programming languages.
* Python has a large standard library that contains reusable code for almost any task, so developers don't need to write everything from scratch.
* Developers can easily integrate Python with other popular programming languages such as Java, C, and C++.
* Python is cross-platform and can be used on various operating systems, including Windows, macOS, Linux, and Unix.

### Jupyter Notebook

Jupyter Notebook is an open-source web application that allows users to create and share documents containing code, equations, visualizations, and narrative text. It is widely used in data science, machine learning, academic research, and educational settings.



Figure 1.2 Jupyter

Benefits of Jupyter Notebook:

* Jupyter allows users to run code in small chunks, making it ideal for testing and debugging.
* The integration with libraries like Matplotlib, Seaborn, and Plotly makes visualizing data easy within the notebook itself.
* Users can mix narrative text (using Markdown) with code and outputs, creating comprehensive reports or tutorials.
* Notebooks make it easy to share code, data, and results, ensuring that research is reproducible.

## Overview of Java and MySQL

### Java

Java is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is intended to let application developers Write Once and Run Anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java was developed by James Gosling at Sun Microsystems Inc. in May 1995 and later acquired by Oracle Corporation, and is widely used for developing applications for desktop, web, and mobile devices.



Figure 1.3 Java

Java is known for its simplicity, robustness, and security features, making it a popular choice for enterprise-level applications. Java applications are compiled to bytecode that can run on any Java Virtual Machine. The syntax of Java is similar to C/C++.

Since Java runs on the JVM, it may not be as fast as languages that are compiled directly, such as C/C++. This can be a concern for applications that require high performance.

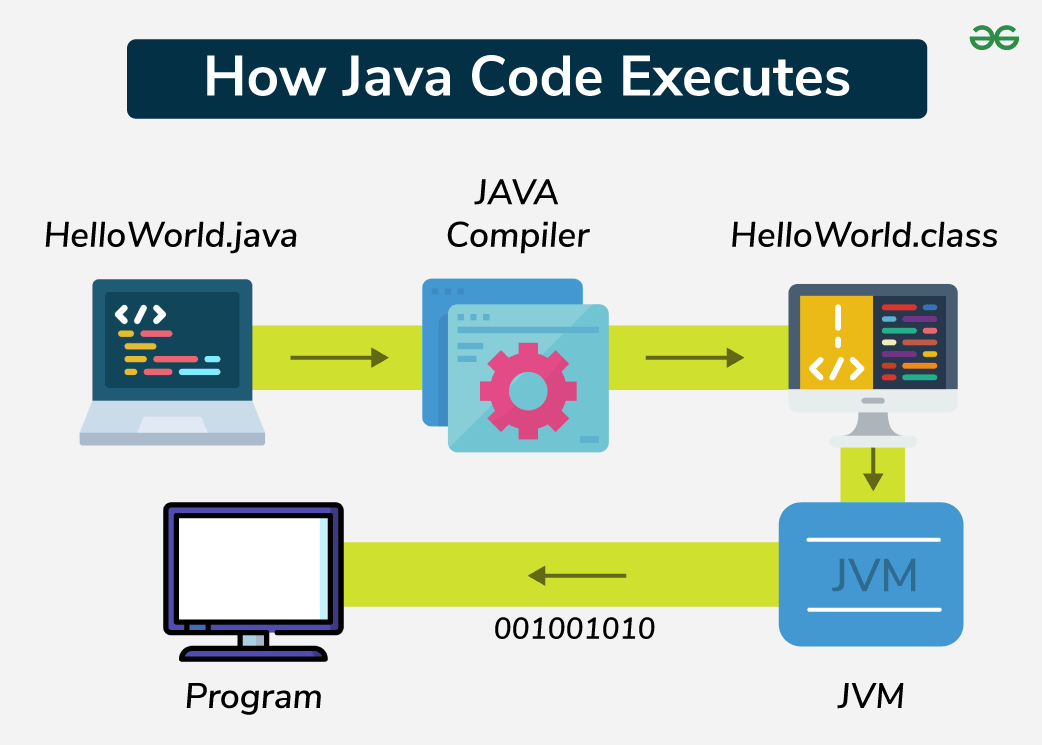


Figure 1.4 The compilation and interpretation process of Java

### Spring Boot

Spring Boot is a combination of the Spring Framework and an embedded server, allowing you to build web applications, microservices, and standalone JAR applications. It supports features like Microservices, REST API, and JPA databases, making app development simpler and more efficient.



Figure 1.5 Spring Boot

Advantages of Spring Boot:

* It creates standalone Spring applications.
* It makes testing web apps easier with embedded HTTP servers like Tomcat, Jetty, and others. No need to distribute WAR files.
* No XML configuration required.

### MVC model

* **MVC** stands for **Model-View-Controller**.
* It is a design pattern used in software engineering.
* MVC is a software architecture pattern used to create computer user interfaces.
* MVC is divided into three connected parts; each part has its responsibility and works independently from the others.
  + **Model**
    - Responsible for working with the **database**.
    - It contains all the functions and methods that directly query or update the data.
    - The **Controller** uses these functions to get data and sends it to the **View**.
  + **View**
    - The **user interface**.
    - Contains all the elements that the user interacts with, such as menus, buttons, images, text, etc.
    - Receives data from the **Controller** and displays it.
  + **Controller**
    - Acts as the **middle layer** between Model and View.
    - Handles user requests, uses the **Model** to get data, and then passes that data to the **View** to show to the user.

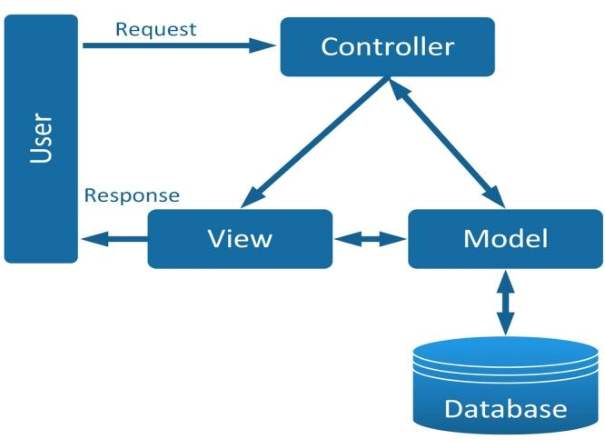


Figure 1.6 Flow of operation in the MVC model

* MVC workflow
  + First, a request from the user is sent from the client to the server.
  + Then, the Controller, based on the user’s request, communicates with the Model to get the required data from the database.
  + Finally, the Controller sends the retrieved data to the View, which displays it to the user in the browser.
* Three-tier model: This is a common source code structure used in Spring Boot applications. Specifically, the application is divided into three tiers (or layers) as follows:
  + Presentation layer: This layer interacts directly with the user. It can include Views and Controllers (in traditional MVC) or APIs (in RESTful applications).
  + Business logic layer: This layer contains the core logic of the application. Most of the application’s code resides here. It is responsible for processing data and enforcing business rules.
  + Data access layer: This layer interacts directly with the database, executing queries and returning results to the business logic layer.

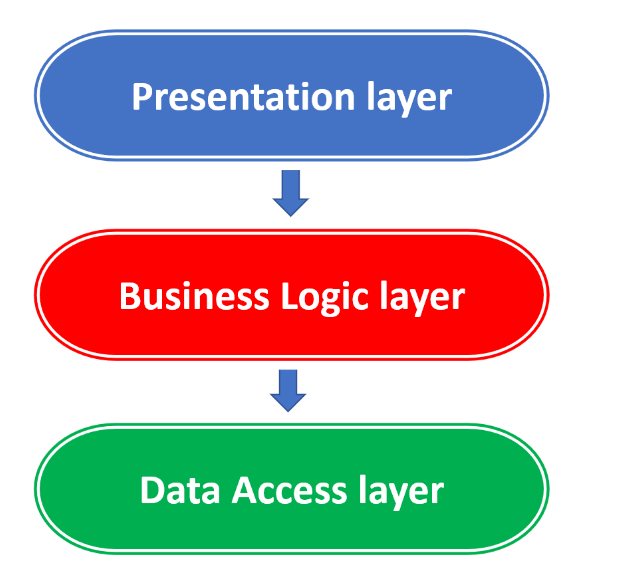


Figure 1.7 Three-tier architecture

* By combining the two models, we get a complete Spring Boot application consisting of the following components:
  + Controller: The controller returns a View (which contains the data ready for display in HTML format) or a Model represented through an API for the View (the View is separately written using React, Vue, or Angular).
  + Service: The Service contains the logic and processing code. When the Controller requests it, the Service will handle the request and return the data (in the form of a Model) back to the Controller. The Controller will then pass this data to the View as described earlier.
  + Repository: The Repository is used by the Service to interact with the database. The Repository directly interacts with the database, reading and writing data, and then returning the results to the Service.

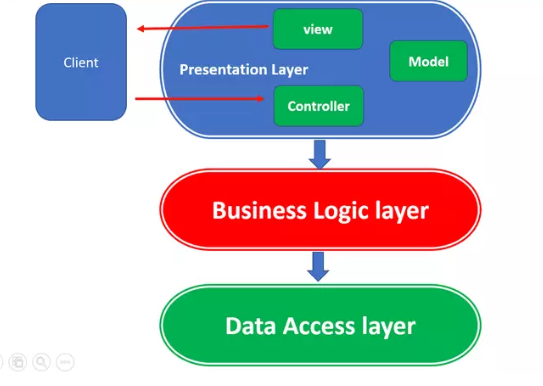


Figure 1.8 Three-Tier architecture vs MVC pattern

### MySQL

* MySQL is an open-source, relational database management system (RDBMS) that uses Structured Query Language (SQL) to manage and manipulate data. It is one of the most popular database systems used in web applications, known for its speed, reliability, and ease of use.



Figure 1.9 MySQL

* Key Features of MySQL:
  + **Developed by Oracle Corporation.**
  + **Supports multiple platforms**: MySQL runs on operating systems such as Windows, Linux, and macOS.
  + **Widely used by developers** due to its scalability, data security features, and large community support.
* Advantages of MySQL:
  + **Open-source**: MySQL is free and open-source, allowing modification and redistribution.
  + **High performance**: MySQL offers fast data retrieval and processing for large datasets.
  + **ACID compliance**: Ensures data integrity and reliability, especially with the InnoDB storage engine.
  + **Scalability**: It supports large databases and high traffic with features like partitioning and clustering.
  + **Variety of storage engines**: MySQL offers multiple storage engine options (e.g., InnoDB, MyISAM) for flexible usage.
  + **Replication**: It supports master-slave replication for data redundancy and high availability.

## Overview of ReactJS and ChakraUI

### React

* React (ReactJS) is a free and open-source JavaScript library for building web user interfaces. It handles only the visual part (the view) and does not control how business logic or app structure is organized.

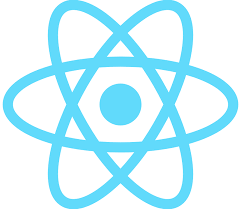


Figure 1.10 React

* Key feature of React
  + **Virtual DOM**:
    - Creates a lightweight copy of the DOM.
    - Compares it with the previous version to detect changes (diffing).
    - Updates only the changed parts in the actual DOM (reconciliation), improving performance.
  + **Components** are the basic building blocks of React that define the structure of the user interface. Each component is encouraged to be as small and independent as possible to allow for easy reuse across different parts of the application or even in other applications.
  + **JSX** is a syntax extension that allows you to write HTML-like code within JavaScript. In other languages, HTML and JavaScript are typically written separately. However, with JSX, React lets you combine both in the same source code file.
  + React uses **one-way data binding**, meaning data flows in a single direction from parent components to child components via props. This provides better control over data and helps maintain predictable behavior.
* React operates by creating an in-memory virtual DOM rather than directly manipulating the browser’s DOM. It performs necessary manipulations within this virtual representation before applying changes to the actual browser DOM.

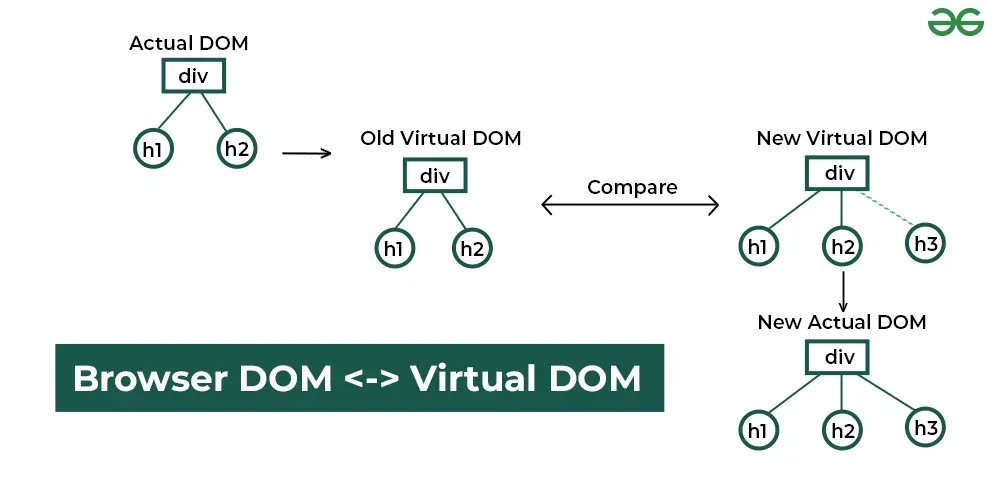


Figure 1.11 How React uses Virtual DOM

* Here’s how the process works:
  1. Actual DOM and Virtual DOM
     + Initially, an Actual DOM contains a div with two child elements: h1 and h2.
     + React maintains a previous Virtual DOM to track the UI state before any updates.
  2. Detecting Changes
     + When a change occurs (e.g., adding a new h3 element), React generates a New Virtual DOM.
     + React compares the previous Virtual DOM with the New Virtual DOM using a process called reconciliation.
  3. Efficient DOM Update
     + React identifies the differences (in this case, the new h3 element).
     + Instead of updating the entire DOM, React updates only the changed part in the New Actual DOM, making the update process more efficient.
* ReactJS lifecycle

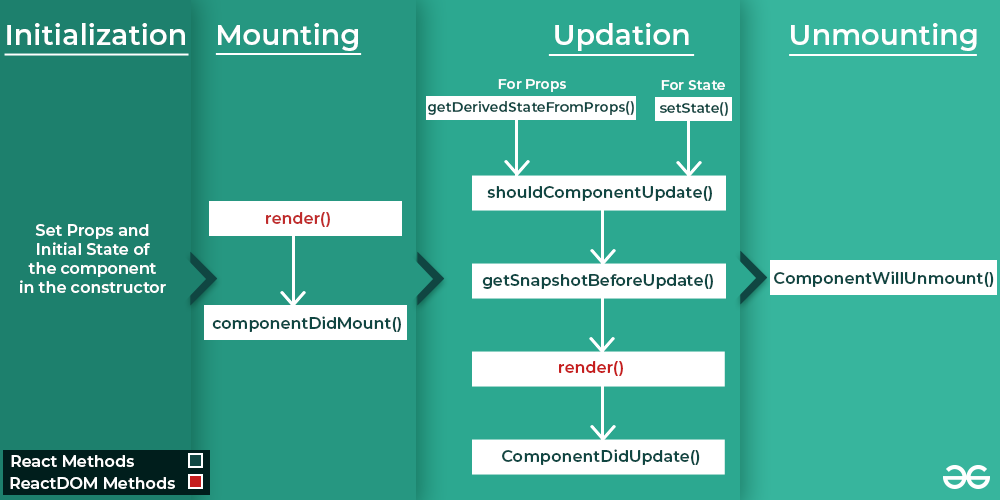


Figure 1.12 ReactJS lifecycle

* 1. Initialization: This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.
  2. Mounting Phase
     + Constructor: The constructor method initializes the component. It’s where you set up the initial state and bind event handlers.
     + **render():** This method returns the JSX representation of the component. It’s called during initial rendering and subsequent updates.
     + **componentDidMount()**: After the component is inserted into the DOM, this method is invoked. Use it for side effects like data fetching or setting timers.
  3. Updating Phase
     + **componentDidUpdate(prevProps, prevState)**: Called after the component updates due to new props or state changes. Handle side effects here.
     + **shouldComponentUpdate(nextProps, nextState)**: Determines if the component should re-render. Optimize performance by customizing this method.
     + **render():** Again, the render() method reflects changes in state or props during updates.
  4. Unmounting Phase
     + **componentWillUnmount()**: Invoked just before the component is removed from the DOM. Clean up resources (e.g., event listeners, timers).

### ChakraUI

* Chakra UI is a modern, modular, and accessible component library for React applications. Its main goal is to simplify the development of user interfaces by providing a set of pre-built, composable, and themeable components that follow best practices for accessibility and responsive design.
* Unlike traditional UI libraries that often require developers to write a lot of custom CSS or rely on additional styling tools, Chakra UI is built with a focus on developer experience. It uses a style props system that allows developers to apply CSS directly to components using simple and intuitive syntax, reducing or even eliminating the need for separate stylesheets.
* With Chakra UI, you can easily build sleek, modern interfaces that are highly customizable and compliant with UI/UX standards. The library offers a wide range of commonly used components such as buttons, forms, navigation bars, modals, and more—helping you get started quickly without having to build everything from scratch.



Figure 1.13 Chakra UI

## Chapter conclusion

In this chapter, I reviewed key technologies and frameworks essential for modern software development. I covered popular programming languages like Python and Java, tools such as Jupyter Notebook, and powerful frameworks like Spring Boot with its MVC and three-tier architecture.

I also discussed MySQL’s strengths as a reliable and scalable database system. On the frontend side, we introduced ReactJS with its component-based design and Virtual DOM, along with Chakra UI for faster and easier UI development.

Together, these technologies form a solid foundation for building effective and maintainable applications today.

# SYSTEM DESIGN AND ANALYSIS



## Business analysis

### Buyer's business

* **Visiting customers**
  + View property list: can view all properties posted on the system.
  + View property details: view description, images, price, area, number of rooms, etc. Atthe same time, the real estate price will be displayed as determined by the system.
  + Register account: if you want to save posts or make transactions, visitors must register an account.
  + Search and filter properties: search by keywords, filter by characteristics of property.
* **Members** are those who have registered an account and can use all the features:
  + All rights of visitors.
  + Update profile and account information: Change contact information and fullname.
  + Upgrade account: choose an account upgrade package to use more features and pay directly through the payment gateway.
  + View transaction history: track completed or pending transactions.
  + Buyer accounts can also post properties for sale, but are limited to 3
  + Manage personal post: edit, update, or delete posted real estate post.
  + Save to favorites: add properties to your favorites list for easy access later.

### Broker's business

* Brokerage account is a registered and upgraded account
* Having all rights of member
* This account will be used for additional features of the system: predict real estate trends.
* The article limits the posts that can be posted by this account to 30

### Admin's business

* Account status control: lock or unlock client accounts.
* Post moderation: review, approve or reject real estate posts.

## System design

### Use-case diagram

#### General use-case diagram

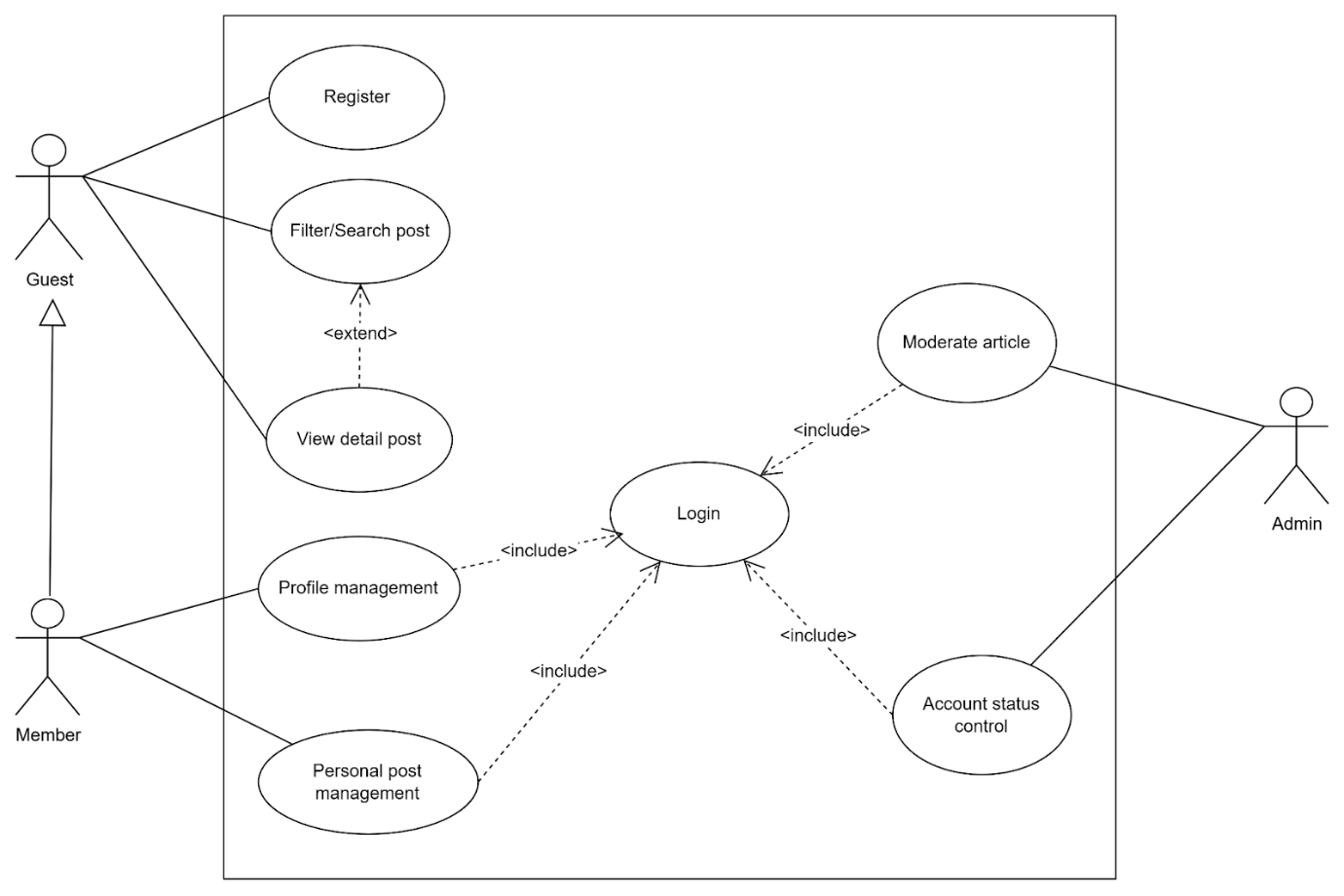


Figure 2.1 General use case diagram

#### Admin use-case diagram

* Account status control

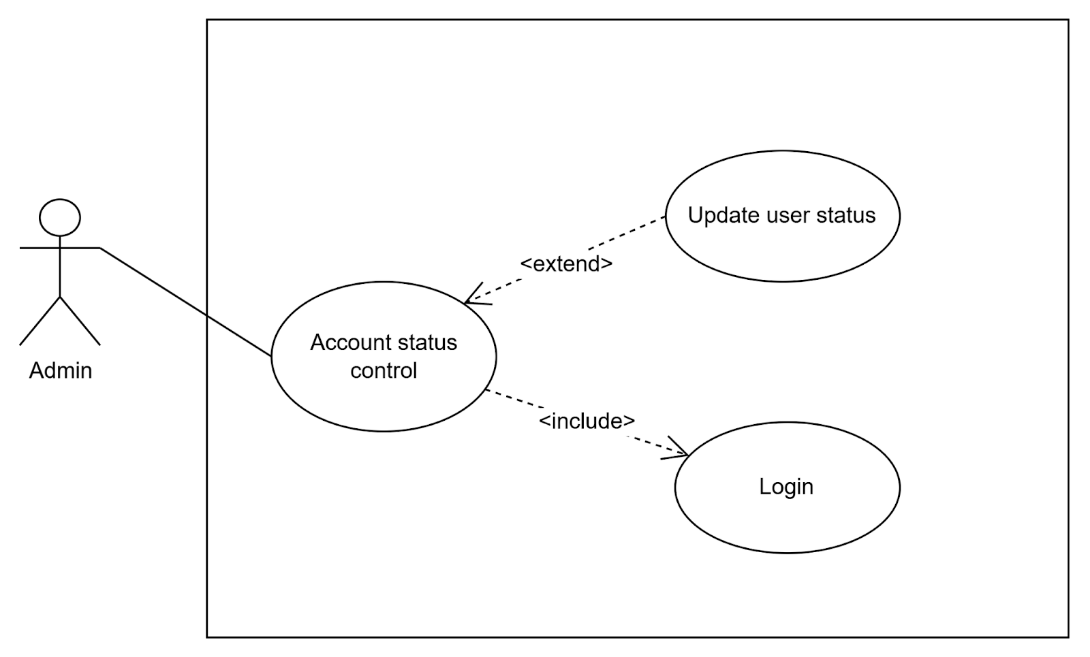


Figure 2.2 User management use-case diagram

* Post moderation

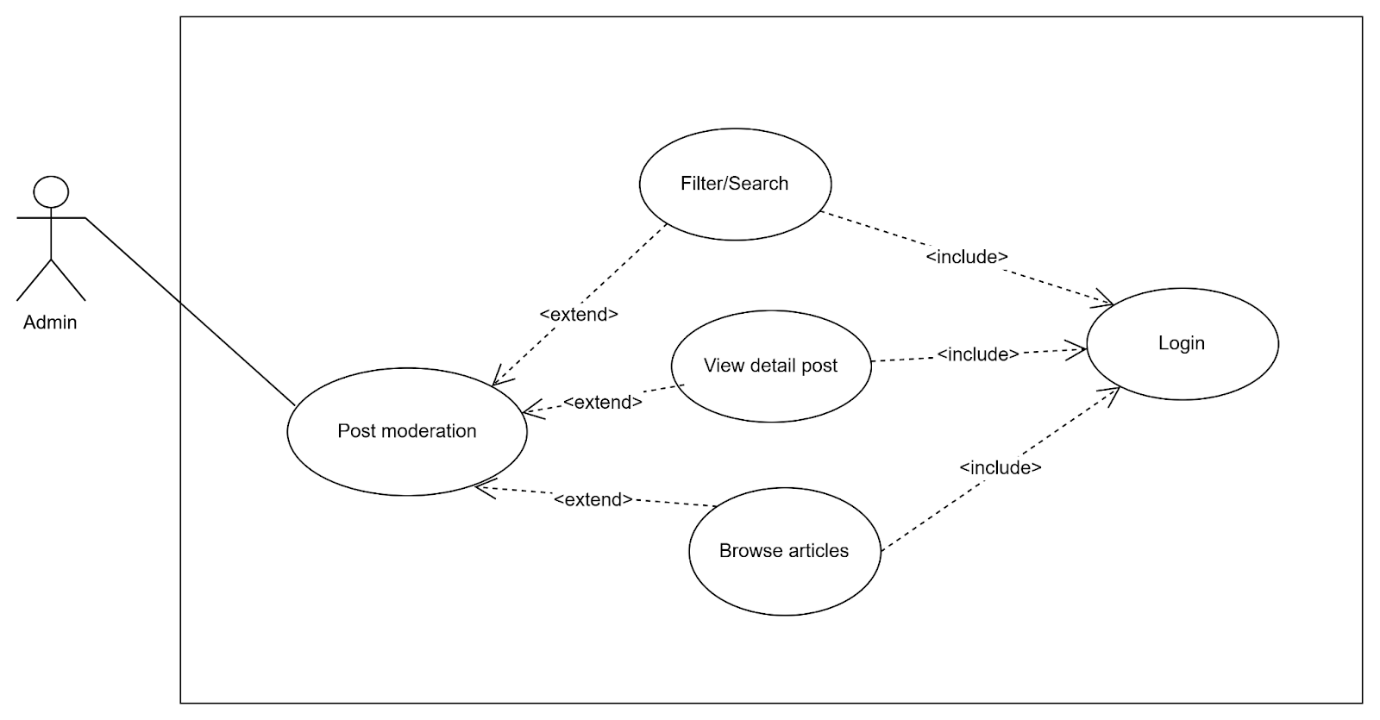


Figure 2.3 Post moderation use case diagram

#### User use-case diagram

* Profile management

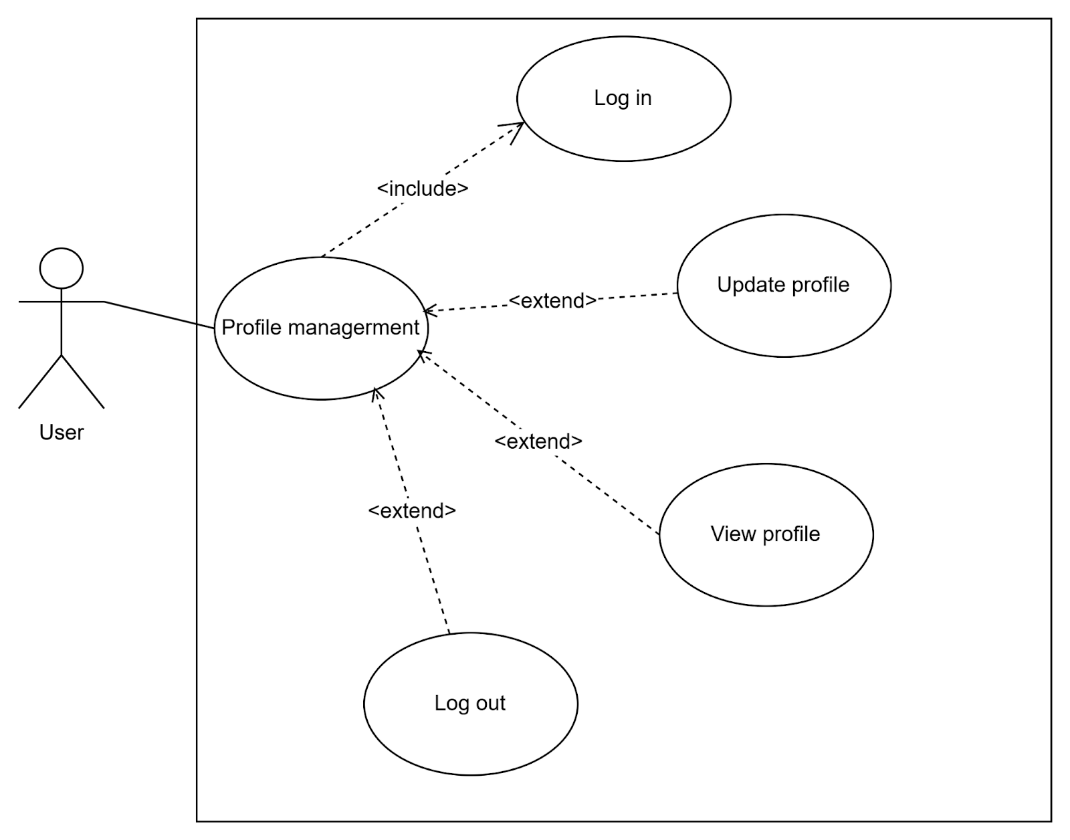


Figure 2.4 Profile management use case diagram

* Account upgrade management

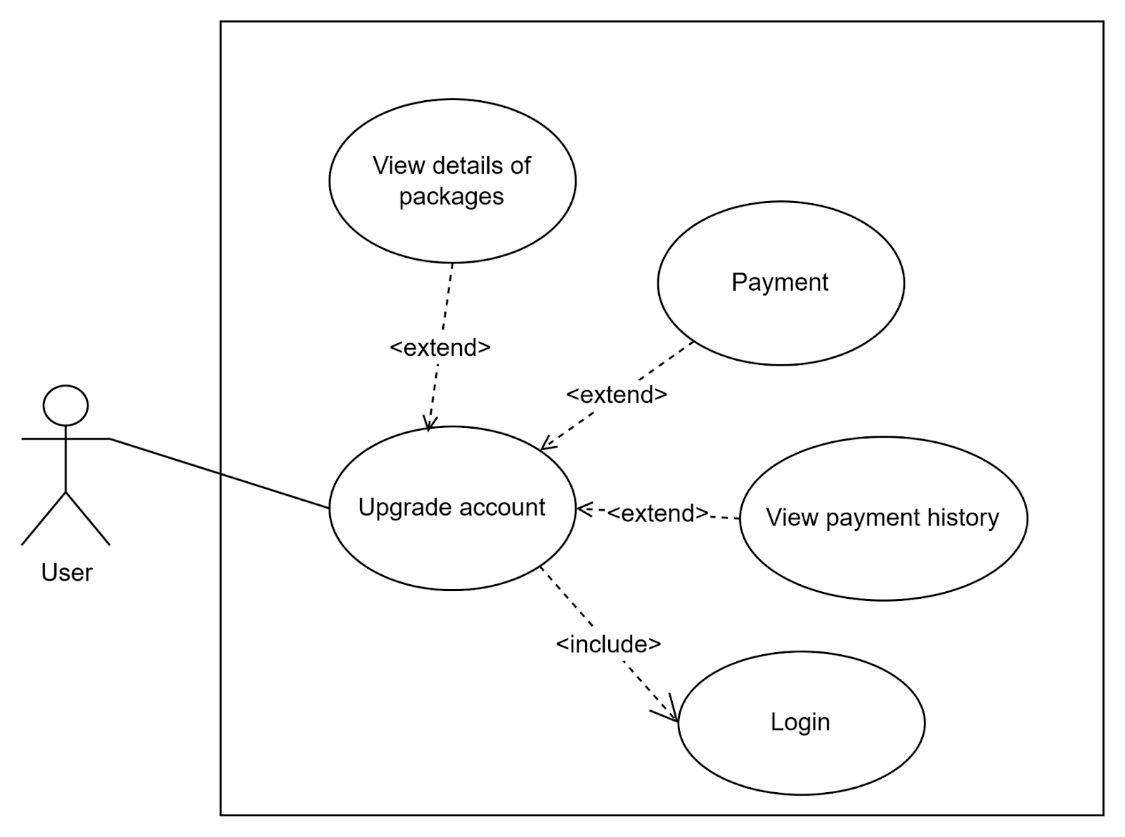


Figure 2.5 Account upgrade management use case diagram

* Post management

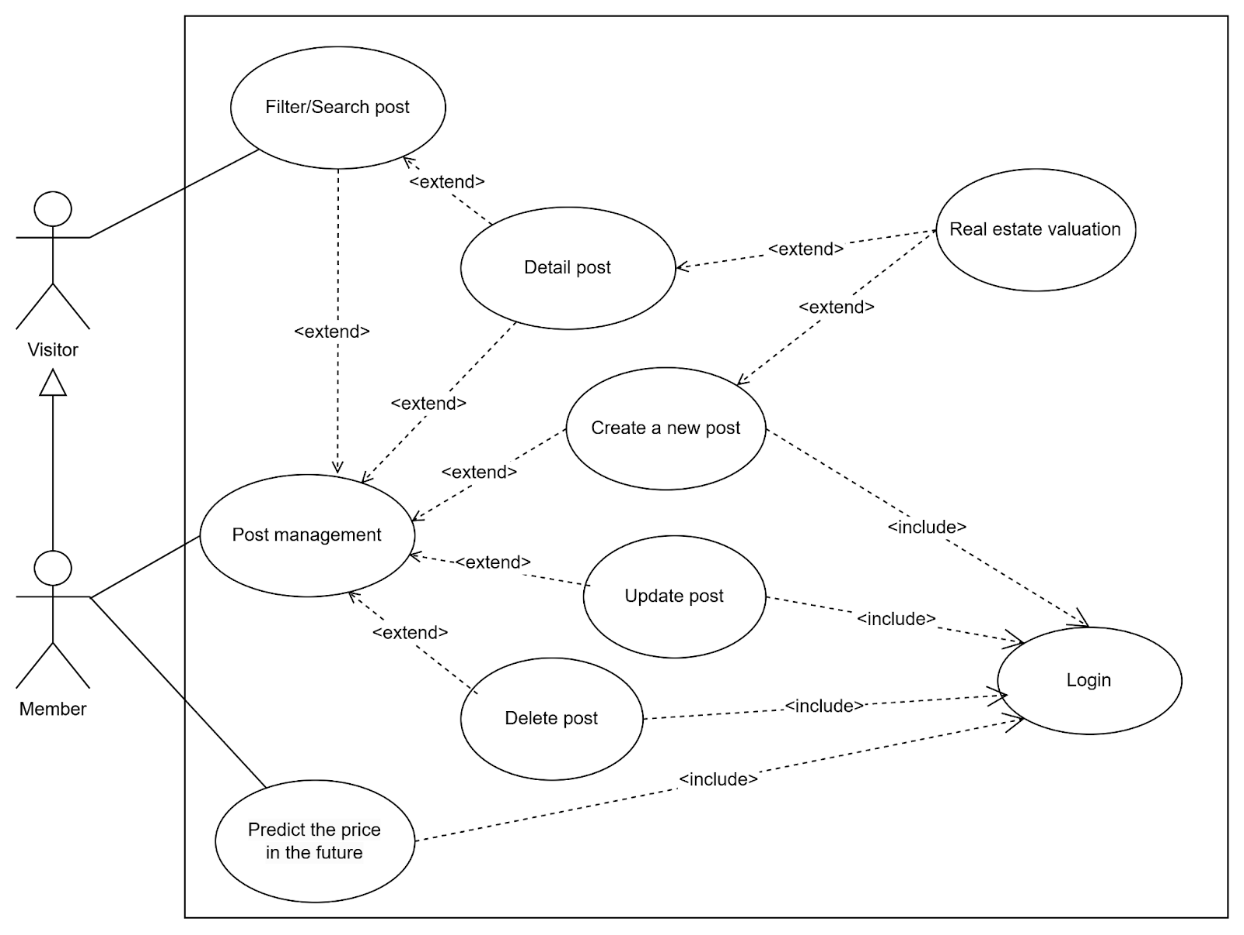


Figure 2.6 Post management use case diagram

* Favourite post management

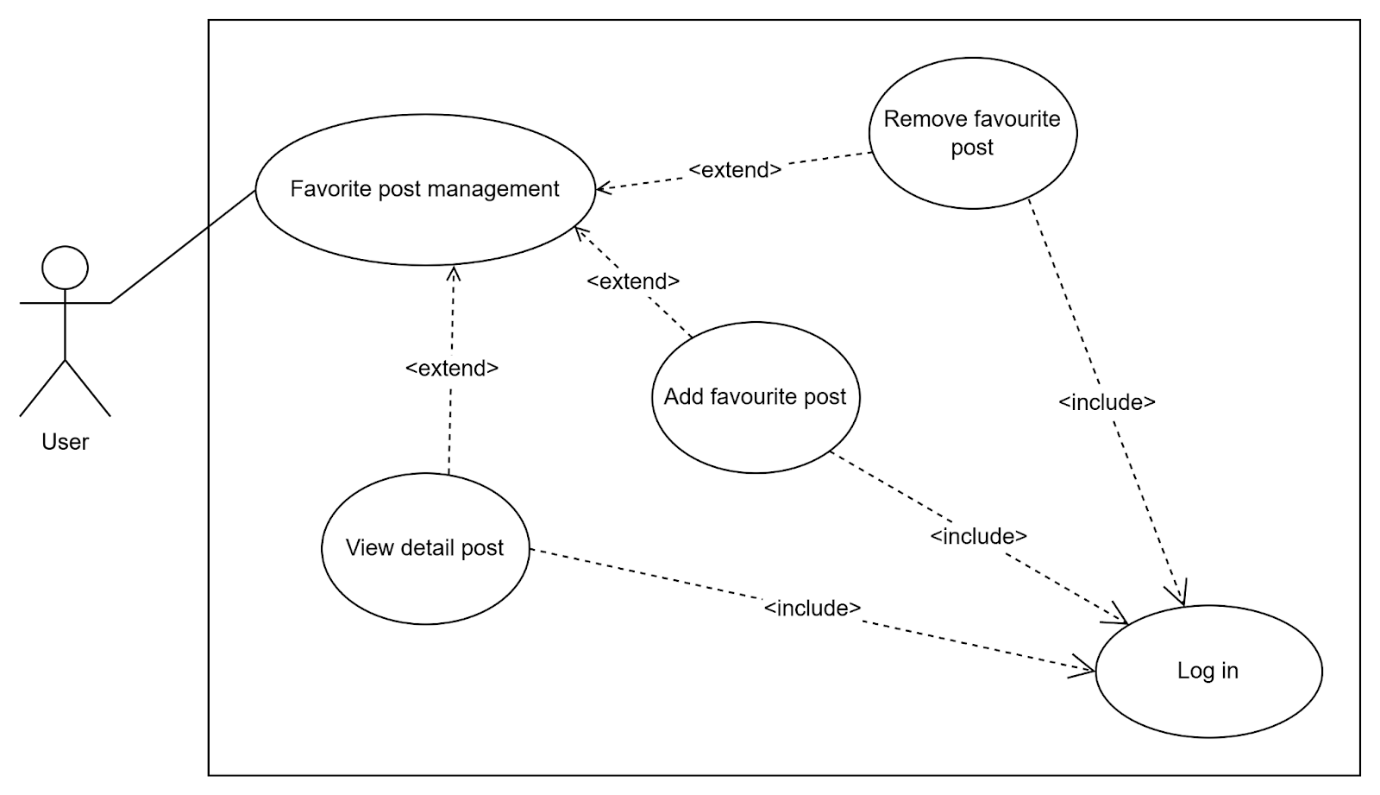


Figure 2.7 Favourite post management use case diagram

### Activity diagram

* The stream of activities provides access to the buyer/seller's website

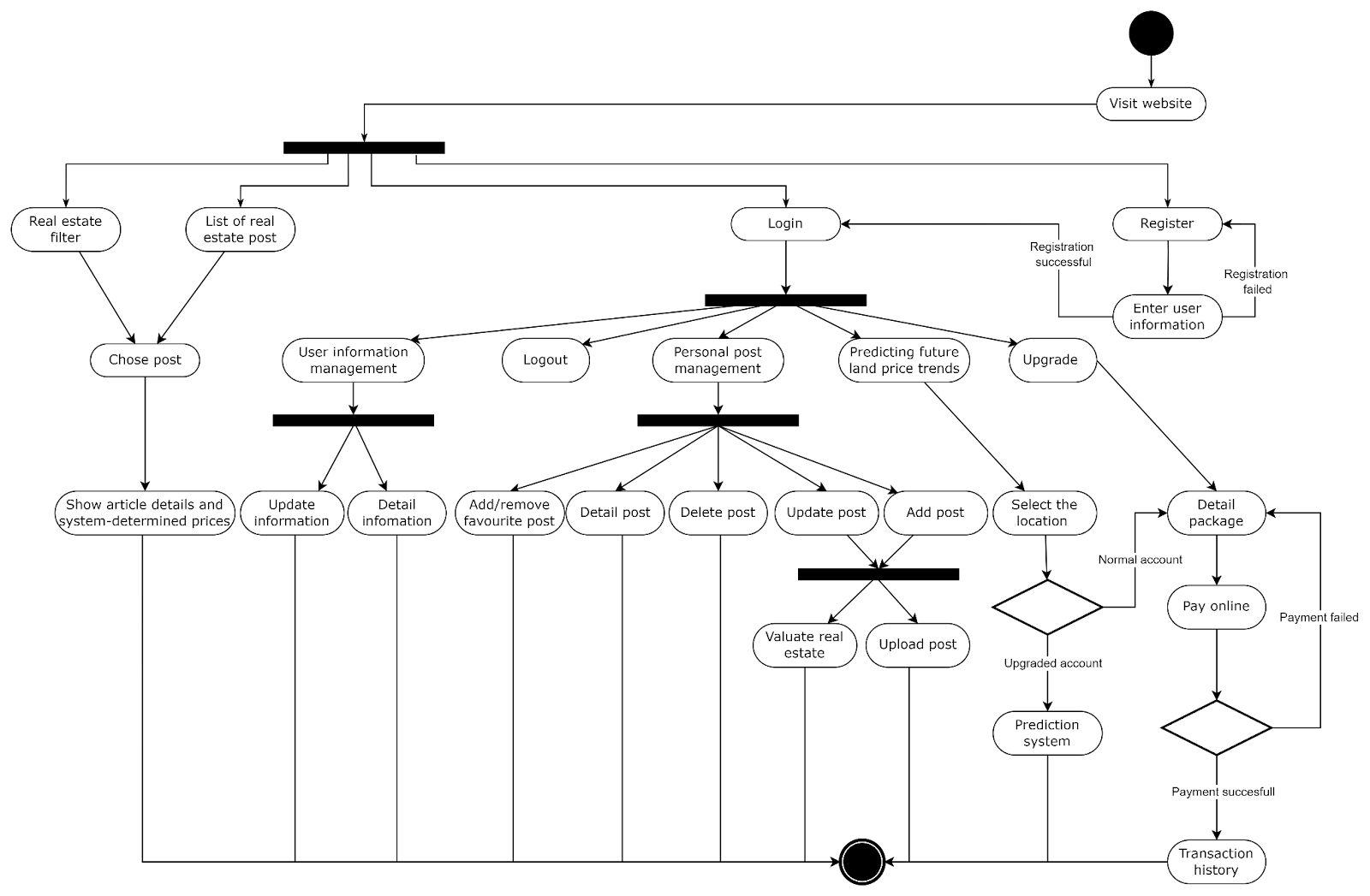


Figure 2.8 Client Operation Principle

* The stream of activities provides access to the admin website

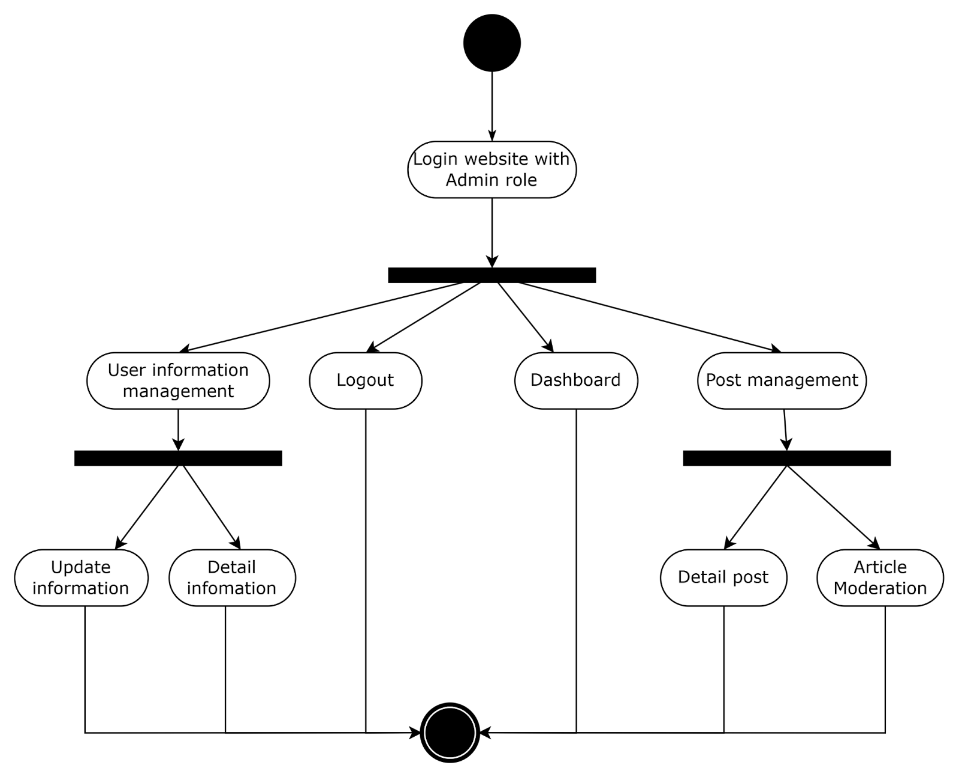


Figure 2.9 Admin Operation Principle

### Sequence diagram

* Login/Signup flow design

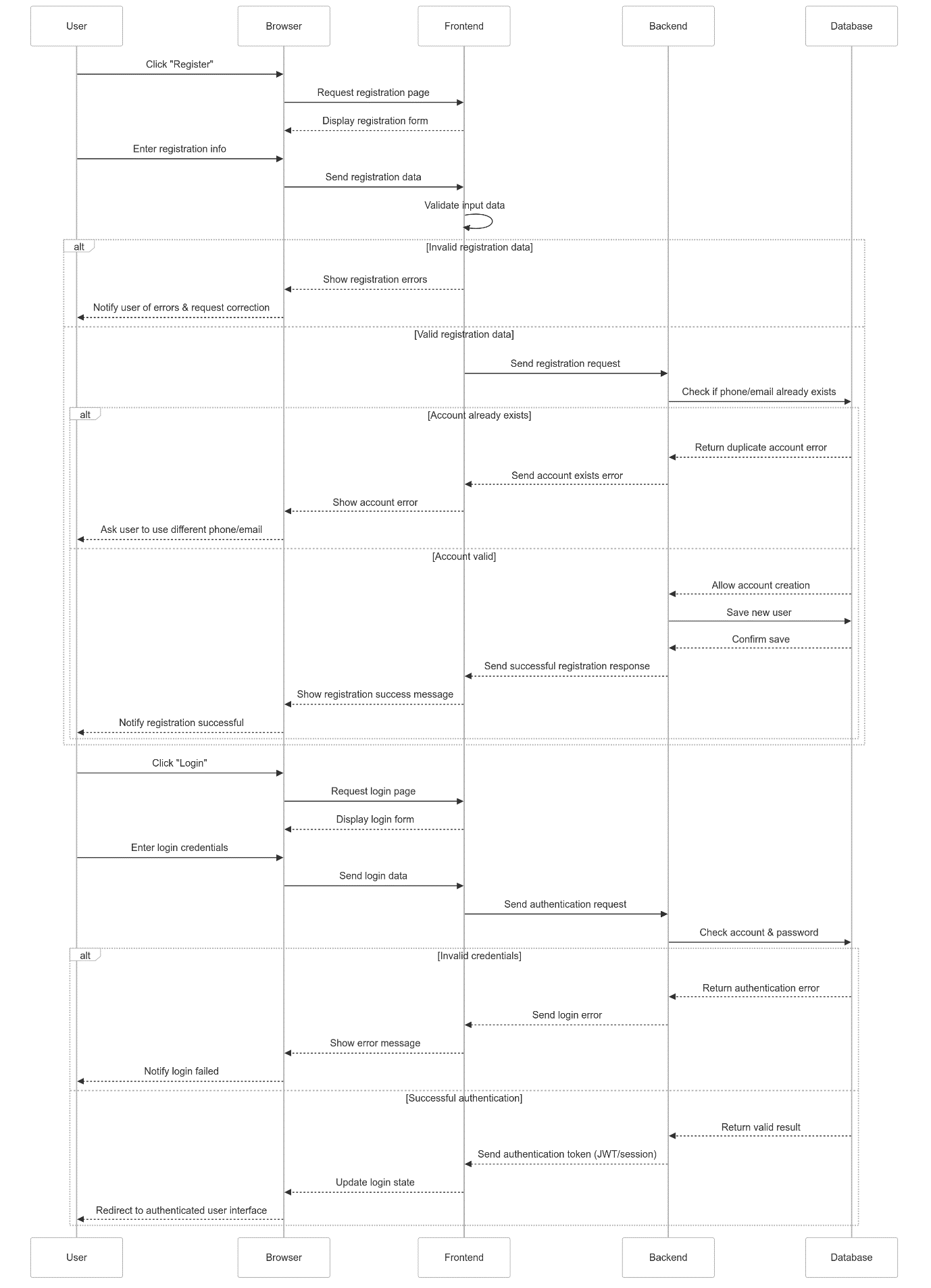
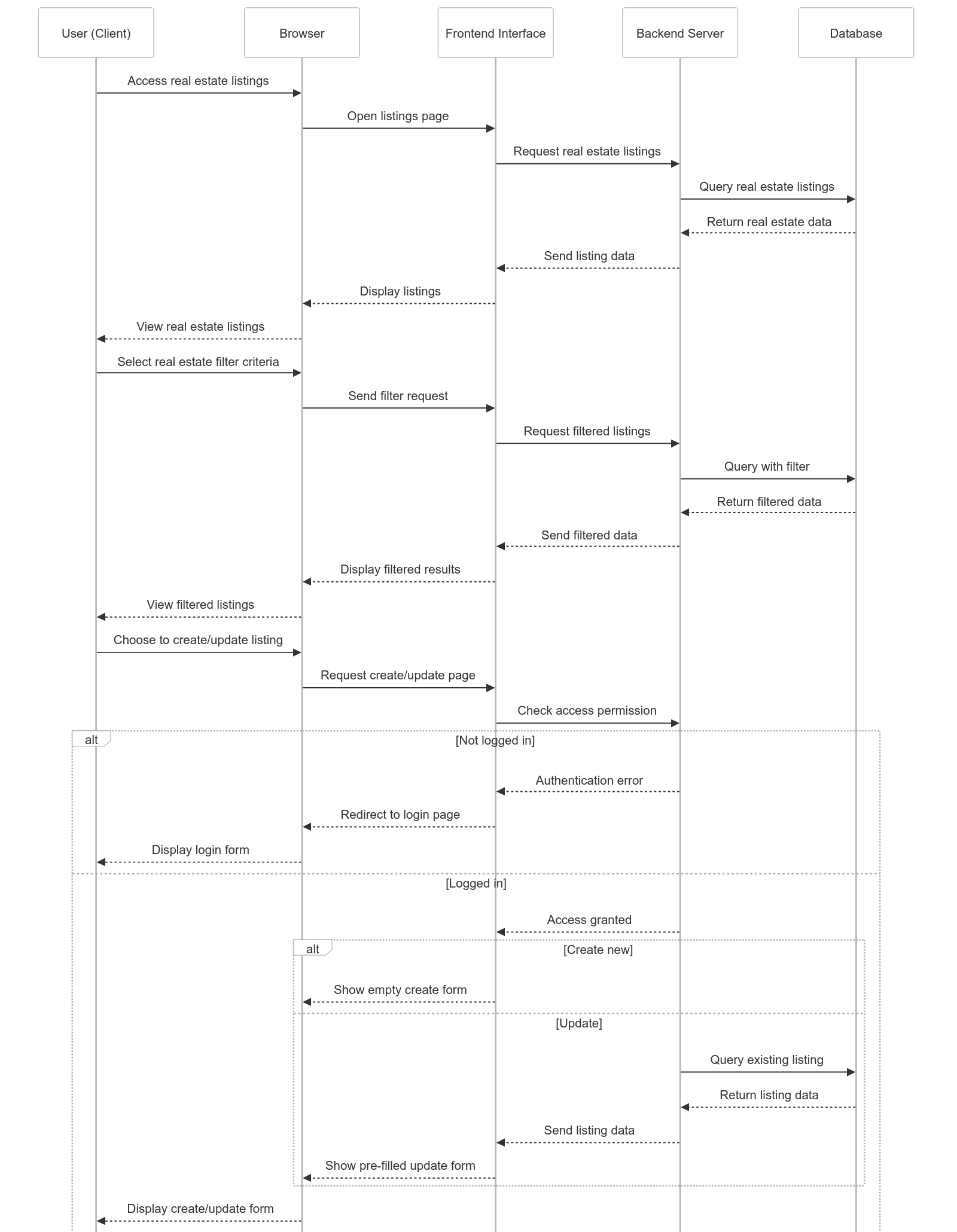


Figure 2.10 Registration/Login Flow

* Real estate article management flow design



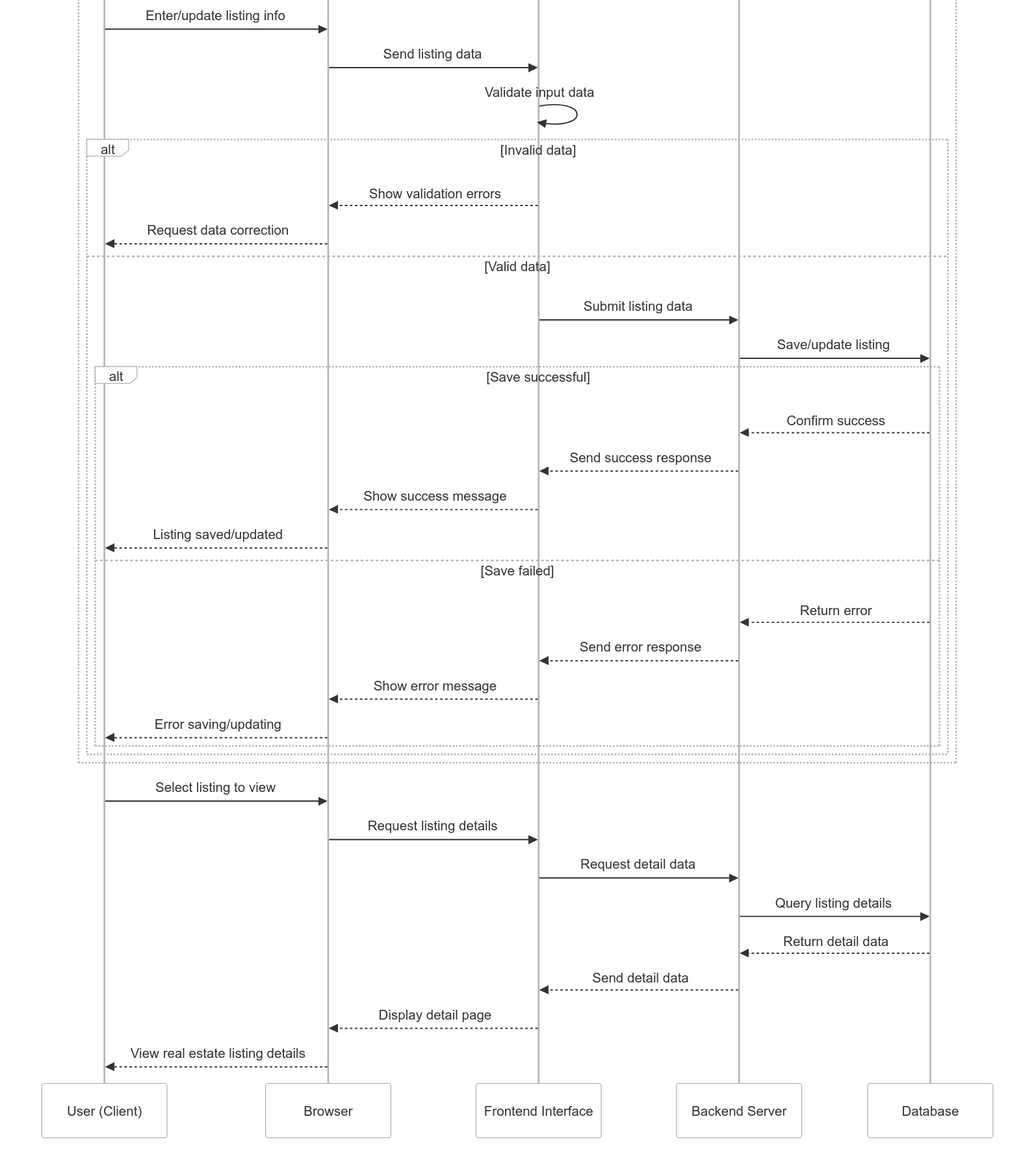


Figure 2.11 Real estate article management flow as a user

* Design of account upgrade flow, integrating online payment

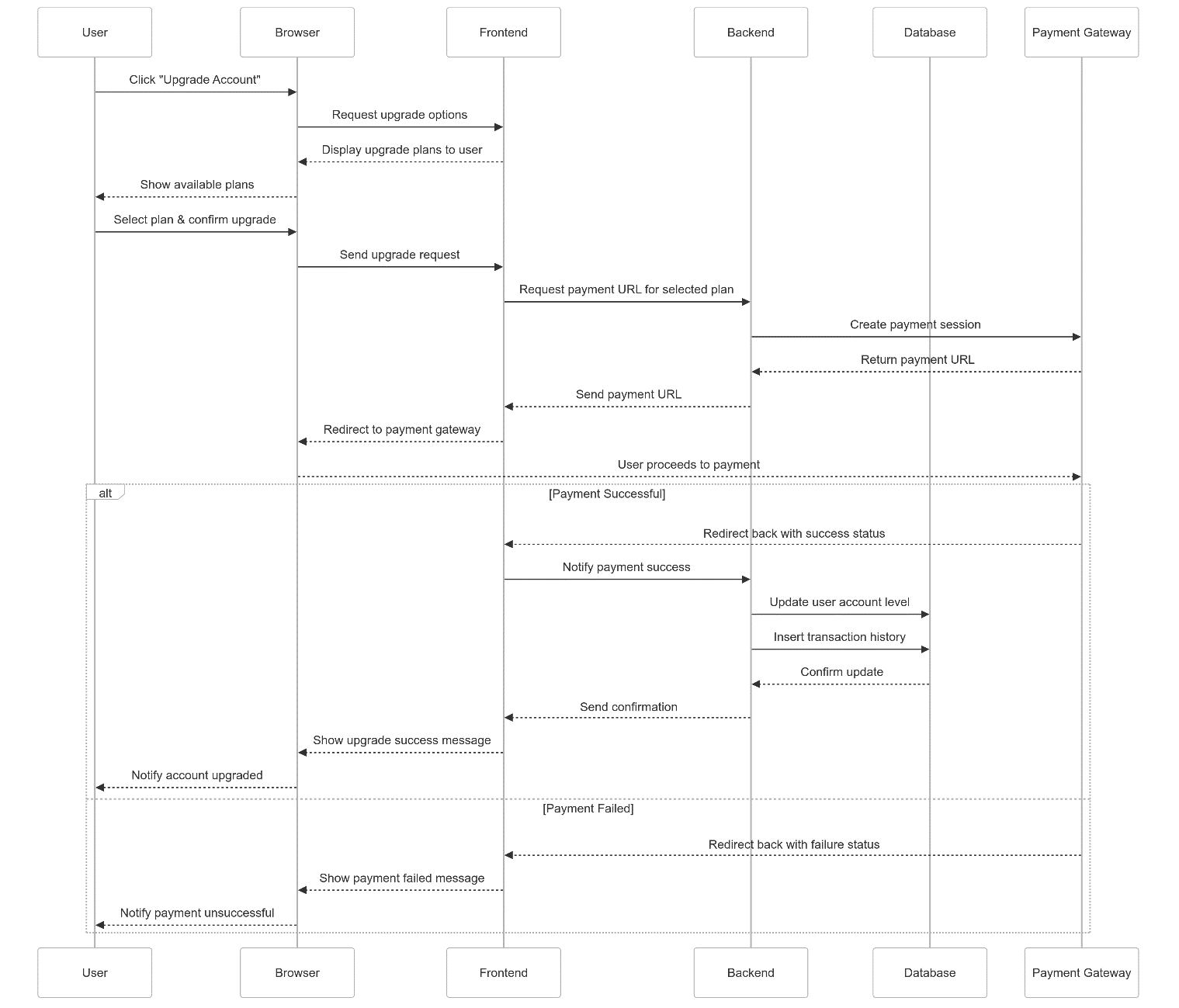


Figure 2.12 Account upgrade flow

## Database design

### ER (entity-relationship)

The ER entity relationship model is represented as

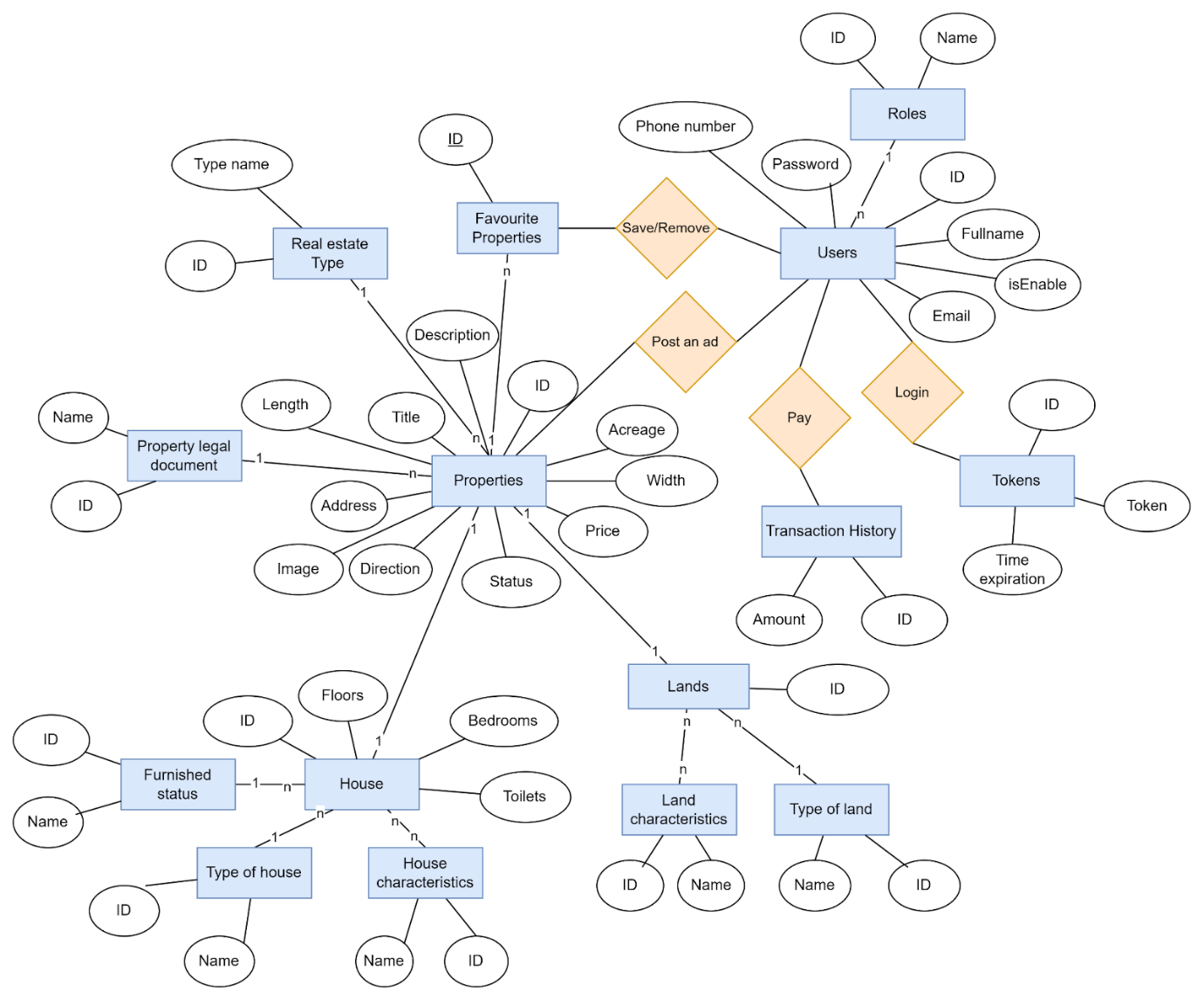


Figure 2.13 The ER entity relationship model

### Converting an implementation model into a system model

* Role table (roles)

Table . Describe role table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| role\_id | Integer | Not null | Primary Key | Code of permission |
| name | Enum | Not null |  | Name of permission |

* User table (users)

Table . Describe user table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| user\_id | Integer | Not null | Primary Key | Code of the user |
| role\_id | Integer | Not null | Foreign Key | Permission |
| isEnable | Boolean | Not null |  | The user account is active or inactive |
| fullName | String | Not null |  | Full name of user |
| email | String | Not null |  | Email of user |
| password | String | Not null |  | Password of user(will be encrypted before entering the database) |
| phone | String | Not null |  | Phone number of user |
| created\_at | LocalDataTime |  |  |  |
| updated\_ad | LocalDataTime |  |  |  |

* Token table (tokens)

Table . Describe token table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| token\_id | Integer | Not null | Primary Key | Code of token |
| user\_id | Integer | Not null | Foreign Key | User |
| token | String | Not null |  | A temporary or persistent string used to authenticate and authorize a user or system |
| created\_at | LocalDataTime |  |  |  |
| updated\_ad | LocalDataTime |  |  |  |

* Transaction history table

Table 2.4 Describe transaction histories table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| transaction\_id | Integer | Not null | Primary Key | Code of token |
| user\_id | Integer | Not null | Foreign Key | User |
| amount | Number | Not null |  | Amount paid |
| created\_at | LocalDataTime |  |  |  |
| updated\_ad | LocalDataTime |  |  |  |

* Favourite property table (favourite\_properties)

Table . Describe favourite properties of table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| favourite\_ property\_id | Integer | Not null | Primary Key | Code of favourite property |
| user\_id | Integer | Not null | Foreign Key | User |
| property\_id | Integer | Not null | Foreign Key | Property |
| created\_at | LocalDataTime |  |  |  |

* Category table (categories)

Table . Describe category table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| category\_id | Integer | Not null | Primary Key | Code of category |
| Name | String | Not null |  | Name of category |

* Property table (properties)

Table . Describe property table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| property\_id | Integer | Not null | Primary Key | Code of Property |
| user\_id | Integer | Not null | Foreign Key | User |
| category\_id | Integer | Not null | Foreign Key | Type of property |
| property\_legal \_document\_id | Integer | Not null | Foreign Key | Type of property legal document |
| status | Enum | Not null |  | Approved or canceled |
| title | String | Not null |  | Title appears at the top of the article |
| description | String | Not null |  | Detailed description of the article |
| region | String | Not null |  | Province name |
| district\_name | String | Not null |  | District name |
| ward\_name | String | Not null |  | Ward name |
| street\_name | String | Not null |  | Street name |
| longitude | Number | Not null |  | Coordinates |
| latitude | Number | Not null |  |
| direction | Number | Not null |  | Main direction of real estate |
| area | Number | Not null |  |  |
| length | Number | Not null |  |  |
| width | Number | Not null |  |  |
| images | String | Not null |  | Contains image links |
| price | Number | Not null |  |  |
| created\_at | LocalDataTime |  |  |  |
| updated\_ad | LocalDataTime |  |  |  |

* House table (houses)

Table 2.8 Describe house table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| house\_id | Integer | Not null | Primary Key | Code of house |
| property\_id | Integer | Not null | Foreign Key | Code of property |
| furnished\_ status\_id | Integer | Not null | Foreign Key | Interior condition |
| house\_ type\_id | Integer | Not null | Foreign Key | Type of house |
| floors | Integer | Not null |  |  |
| bedrooms | Integer | Not null |  |  |
| toilets | Integer | Not null |  |  |

* Furnished status table (furnished\_status)

Table . Describe furnished status table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| furnished\_ status\_id | Integer | Not null | Primary Key | Code of furnished status |
| name | String | Not null |  | Describe the condition of the interior |

* Type of house table (house\_type)

Table . Describe type of house table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| house\_ type\_id | Integer | Not null | Primary Key | Code of house type |
| name | String | Not null |  | House type description |

* House characteristics table (house\_characteristics)

Table . Describe house characteristics table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| house\_ characteristic\_id | Integer | Not null | Primary Key | Code of house  characteristics |
| name | String | Not null |  | House characteristics description |

* + Here create an additional table (house\_characteristic\_mappings) to hold the characteristic keys of the house object.

Table . Describe house characteristic mapping table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| house\_characteristic\_ mapping\_id | Integer | Not null | Primary Key | Code of house  characteristic mapping |
| house\_ characteristics\_id | Integer | Not null | Foreign Key | House  characteristics |
| house\_id | Integer | Not null | Foreign Key | House |

* Land table (lands)

Table . Describe land table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| land\_id | Integer | Not null | Primary Key | Code of house |
| property\_id | Integer | Not null | Foreign Key | Code of property |
| land\_type\_id | Integer | Not null | Foreign Key | Type of land |

* Type of land table (land\_types)

Table . Describe type of land table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| land\_type\_id | Integer | Not null | Primary Key | Code of land type |
| name | String | Not null |  | Land type description |

* Land characteristics table (land\_characteristics)

Table . Describe land characteristics table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| land \_ characteristic\_id | Integer | Not null | Primary Key | Code of land  characteristics |
| name | String | Not null |  | Land characteristics description |

* + Here, create an additional table (land\_characteristic\_mappings) to hold the characteristic keys of the land object.

Table . Describe land characteristic mapping table attributes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field Name | Data Type | Required | Constraint | Description |
| land \_characteristic\_ mapping\_id | Integer | Not null | Primary Key | Code of land  characteristic mapping |
| land \_ characteristics\_id | Integer | Not null | Foreign Key | Land  characteristics |
| house\_id | Integer | Not null | Foreign Key | Land |

### Database schema

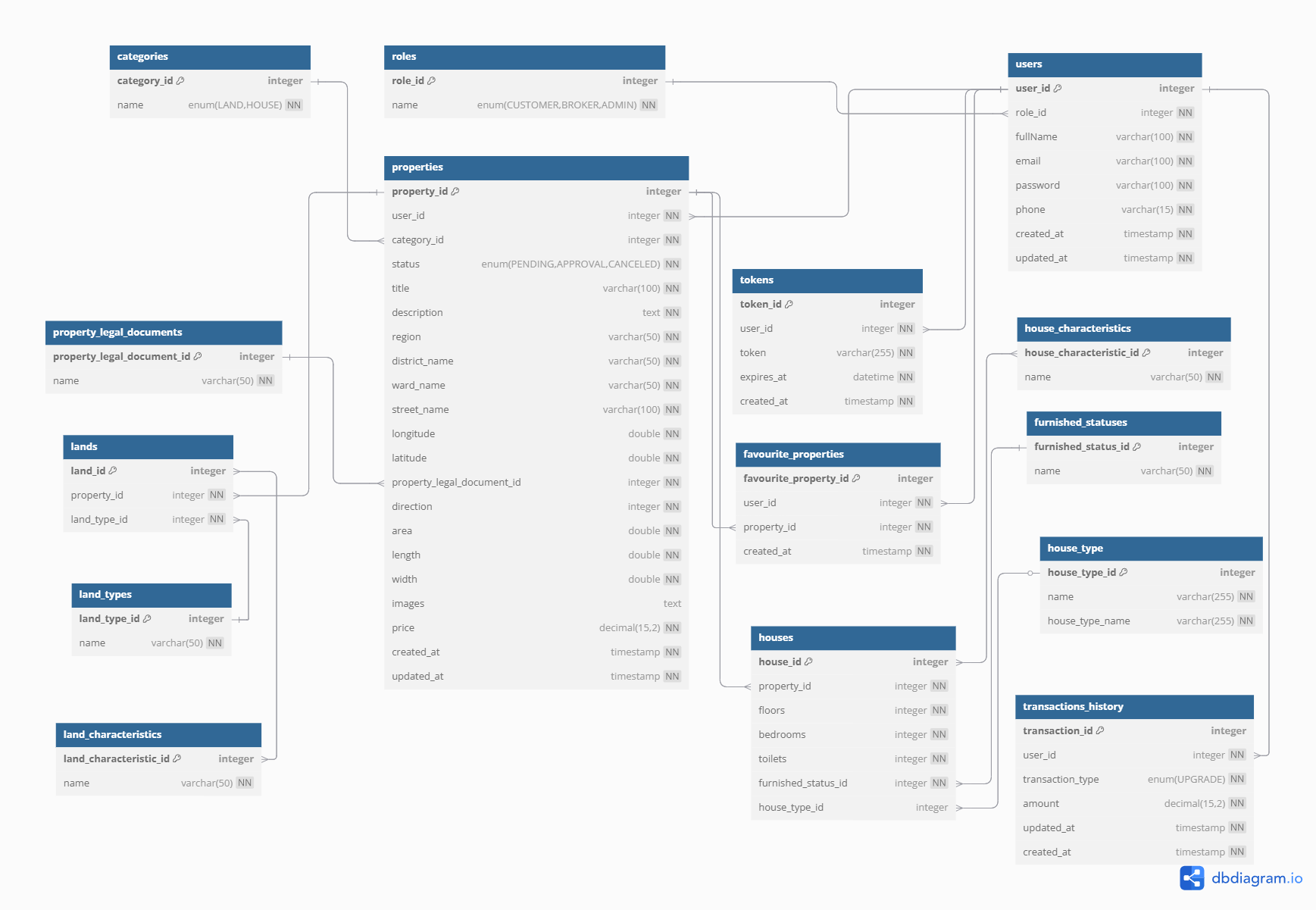


Figure 2.15 Relational Model

## Real estate prediction system development

* The problem of predicting real estate prices will be divided into two types:
  + Real estate valuation: Users will provide information related to the land or the house they want to sell or buy for the system to evaluate.
  + Real estate price forecast: The system will predict the price shortly after a certain area is given by users at the same level.
* Common processing steps of the two models

### Data crawling

* Collect data from the “Nhà tốt” website and have a “RawData.csv” file with … samples.
  + This dataset contains two property types: house and land. For entries where the property type is land, the numerical values for **Rooms, Toilets, Floors,** and **Furnishing Sell** are set to 0.

### Data cleaning

* Validate values: Check if there are any values less than 0

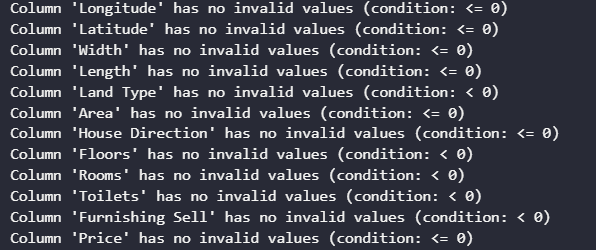


Figure 2.16 Statistics to check features with values less than 0

#### Handling missing

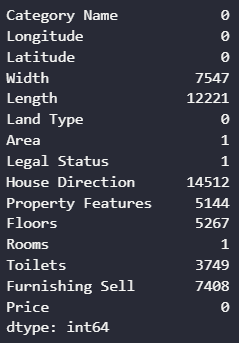


Figure 2.17 Statistics of missing values

* Since a large number of missing values can significantly alter the overall data distribution and characteristics when imputed, I decided to remove all samples with three or more missing features to preserve the integrity of the dataset.

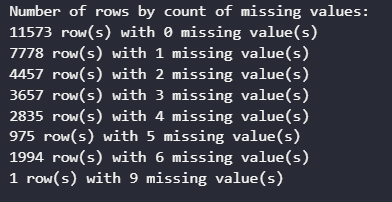


Figure 2.18 Distribution of rows based on the number of missing values

* Confirming that the area data is complete, we proceed to handle missing values for **width** and **length** as follows:
  + If either width or length is missing, we estimate the missing value using the area calculation formula (Area = Width × Length).
  + If both width and length are missing, we assign a default value of 5 meters for both width and length.

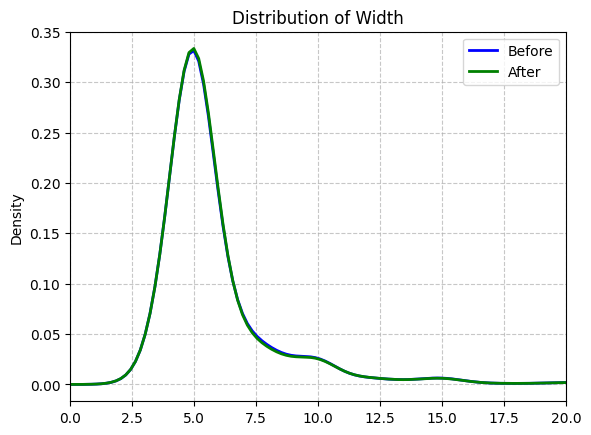


Figure 2.19 Width distribution before and after preprocessing

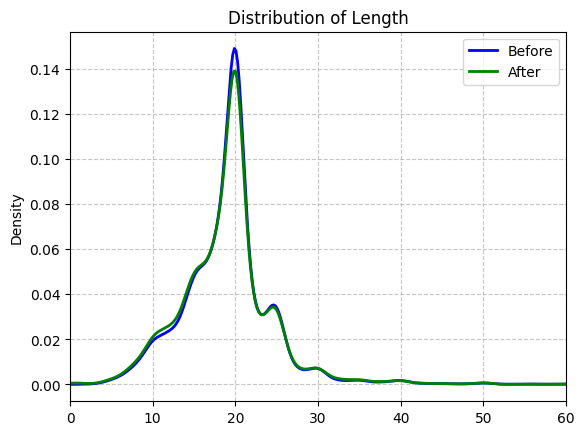


Figure 2.20 Length distribution before and after preprocessing

* For the **direction** feature, missing values are imputed using a random sampling method.

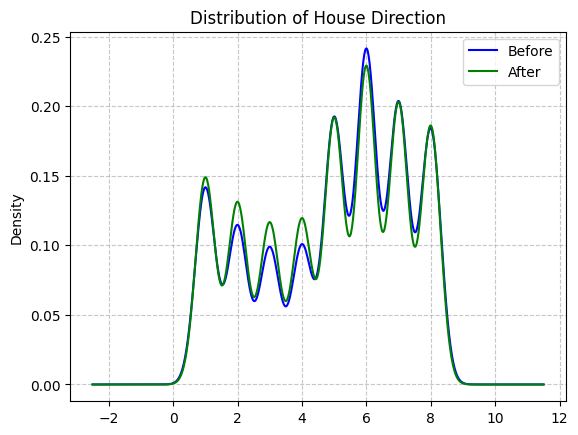


Figure 2.21 Direction distribution before and after preprocessing

* Regarding the **property features** variable:
  + If the land type is residential land, the default value for property features is set to 3 (partly residential land).
  + Otherwise, it is set to 2 (non-residential land).

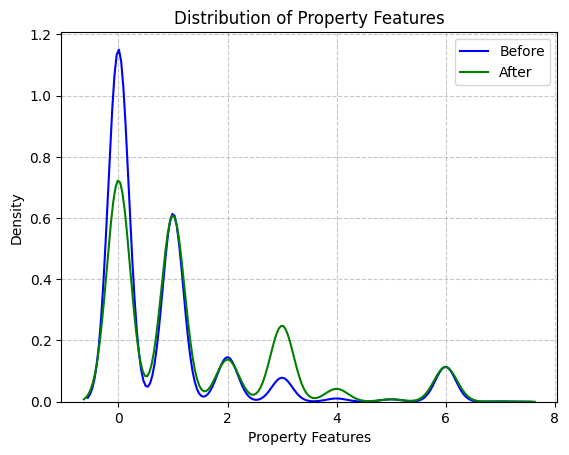


Figure 2.22 Property features distribution before and after preprocessing

* As for the **floors** feature, based on common housing layouts:
  + The ground floor typically contains 1 bedroom.
  + Each upper floor usually has 2 bedrooms.

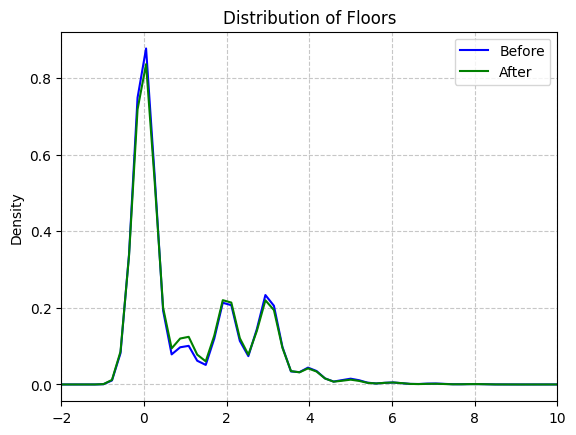


Figure 2.23 Floor distribution before and after preprocessing

* The number of **toilets** is assumed to be equal to the number of floors.

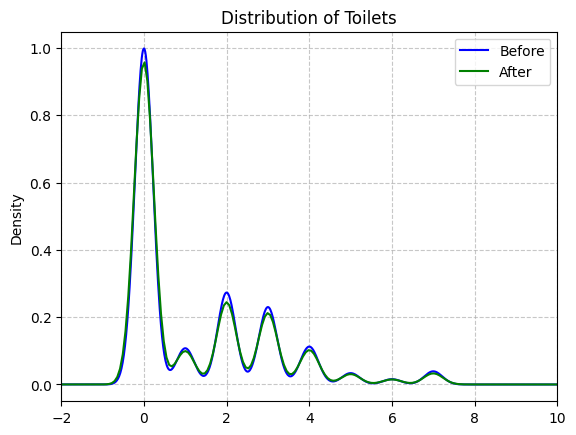


Figure 2.24 Toilet distribution before and after preprocessing

* Finally, for the **furnishing sell** feature, missing values are also imputed using a random sampling method.

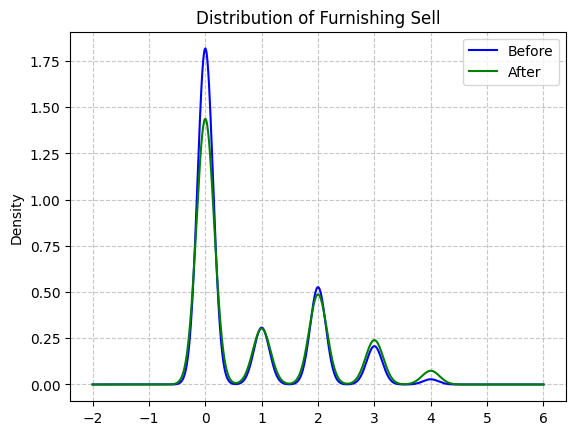


Figure 2.25 Furnishing sell distribution before and after preprocessing

***Notes****: Most plots show that data cleaning preserved the original distribution shape, indicating that the overall data structure remained largely intact. In the "Distribution of Length" and "Distribution of Width" plots, cleaning effectively removed some outliers in the right tail. Key distribution peaks were maintained across most features, suggesting minimal impact on the core data.*

*However, in the "Distribution of Property Features" plot, there is a noticeable difference between the before (blue) and after (green) curves, especially in the 0–1 range, which may indicate a loss of important information. Similarly, in the "Distribution of Furnishing Sell" plot, the main peak’s density decreased after cleaning, potentially affecting analyses involving this feature.*

* The process returns a cleaned dataset containing the following fields and is exported as “Cleandata.csv”:
  + Real estate valuation:

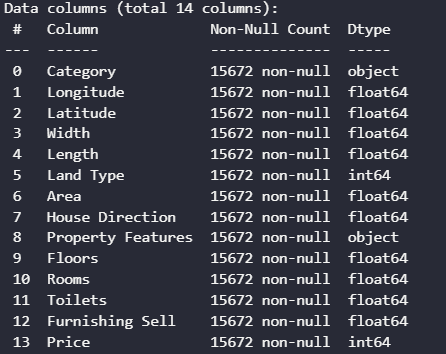


Figure 2.26 Real estate valuation model features

* + Real estate price forecast:

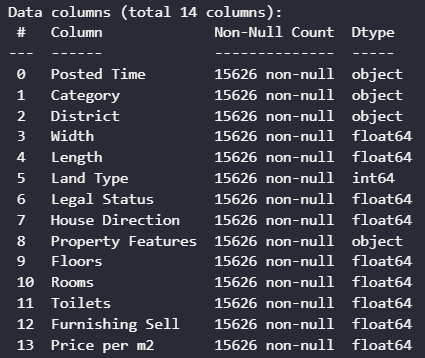


Figure 2.27 Real estate price forecast features

* The two problems use slightly different features. For the valuation model, exact coordinates are included since price can vary significantly even along the same street. For the forecasting model, time becomes a crucial factor, and detailed coordinate data is sparse. Therefore, we generalize location by district and use **price per m²** as the target variable.

#### Handling outliers

* These charts reveal several serious issues with the data distribution, as most variables are heavily skewed and concentrated within a very narrow range of values.

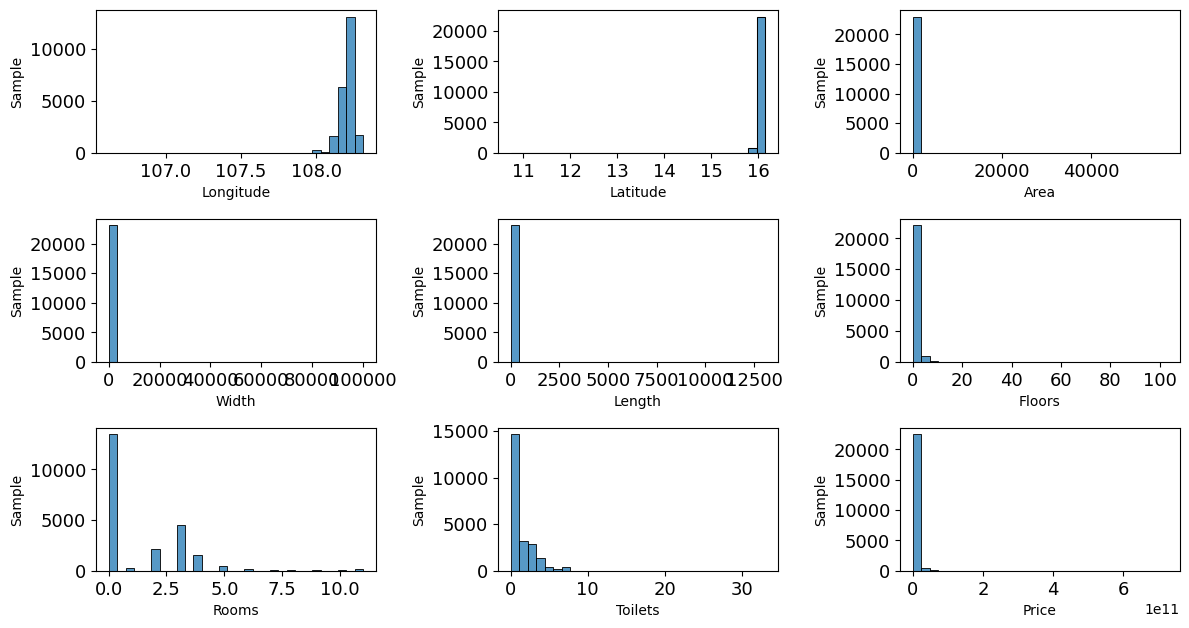


Figure 2.28 Histogram plots of feature distributions

* To improve data quality, we applied outlier removal techniques to variables with high skewness. This helped reduce the impact of extreme values and enhanced the reliability of subsequent analyses and modeling

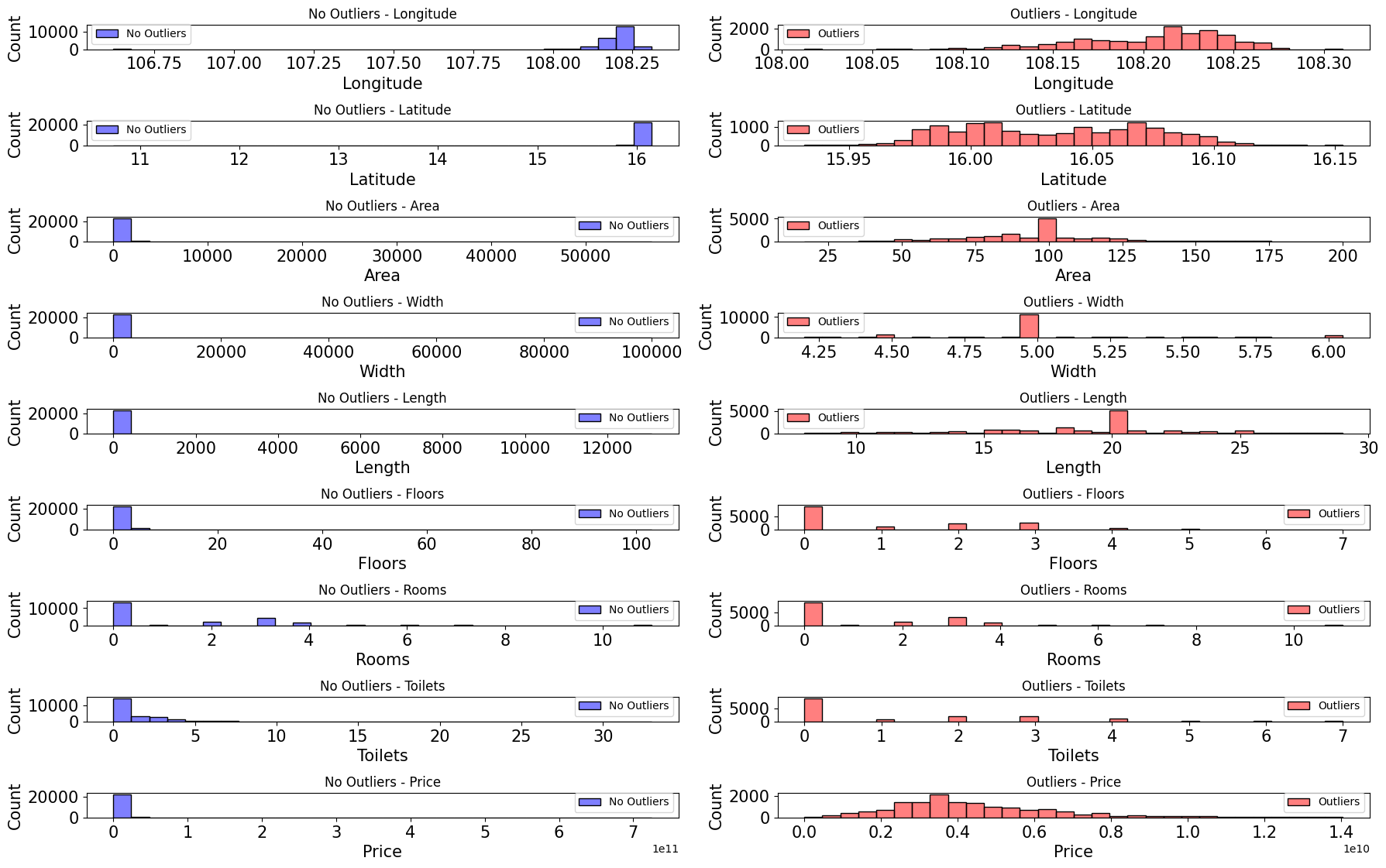


Figure 2.29 Visual Comparison of Outlier Impact on Feature Distributions

### **Feature Engineering**

* Feature Transformation:
  + First, one-hot encoding is applied to categorical variables such as **property category, land type, direction, furnishing status,** and **district.**
  + For numerical features, various normalization techniques (e.g., Z-score, log, sqrt, min-max, arsinh) are tested. The most effective method is selected based on distribution analysis.
  + The result is a transformed dataset exported as “TransformedData.csv”.

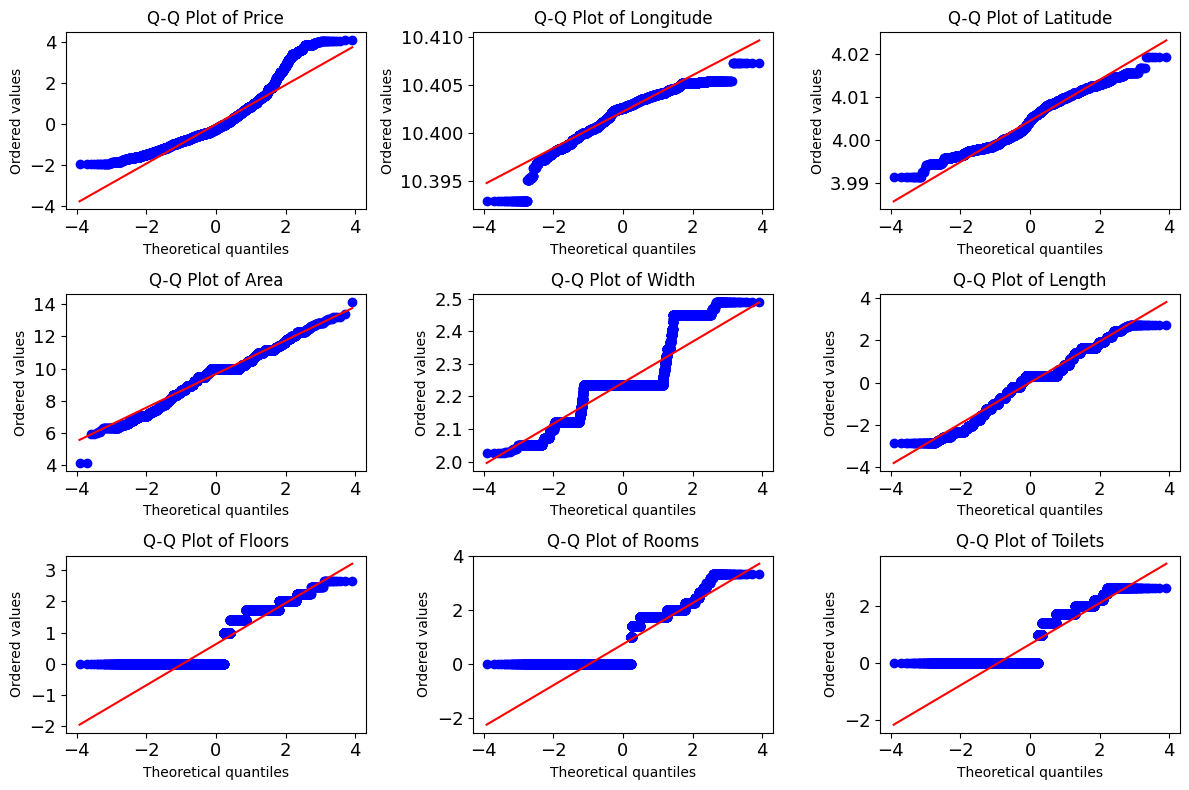


Figure 2.30 Q-Q Plots after optimal normalization per feature

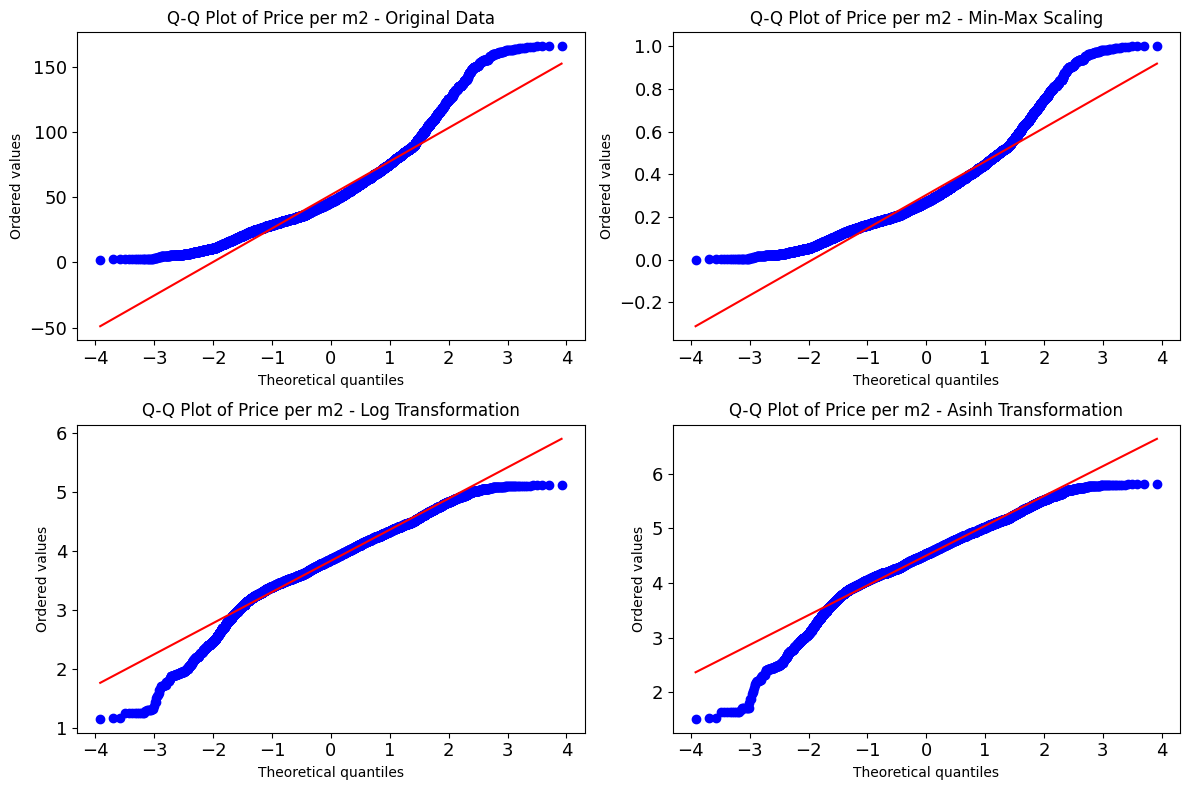


Figure 2.31 Comparison of Q-Q Plots of **Price per m²** across different transformation methods

* Feature Selection:
  + Observing the relationships between the coordinates (Longitude, Latitude), area, and width with the target variable (Price), it was noted that these features have a slight correlation with the target.

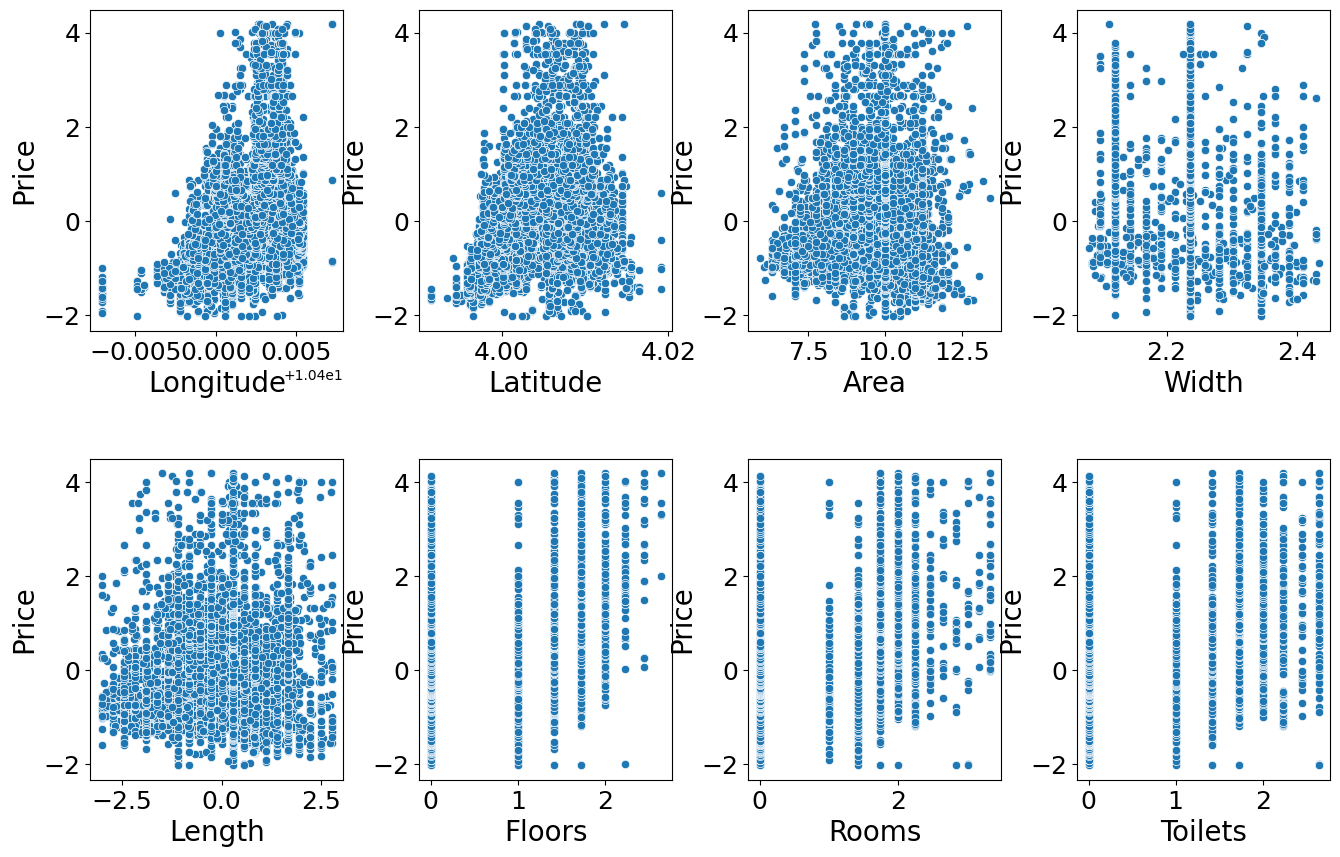


Figure 2.32 Scatter plot of features vs. **Price**

* + **SelectKBest** was used to rank the features based on their correlation with the target, with **Longitude** and **Latitude** being the most influential.

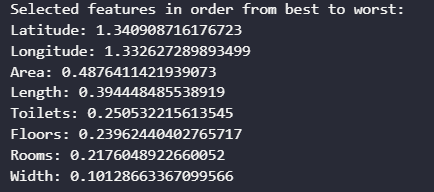


Figure 2.33 Selected features in order from best to worst.

* + The **RFECV** method was then applied, selecting the optimal number of features, which is 8 in this case. The best set of features for predicting the target variable includes **Longitude, Latitude, Area, Width, Length, Floors, Rooms,** and **Toilets**.

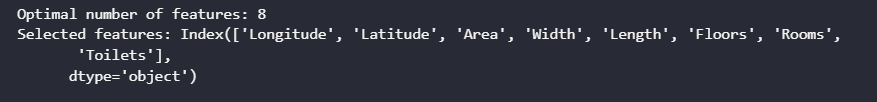


Figure 2.34 Optimal number of features

* + The plot of the **RFECV** shows the relationship between the number of selected features and the mean test accuracy.

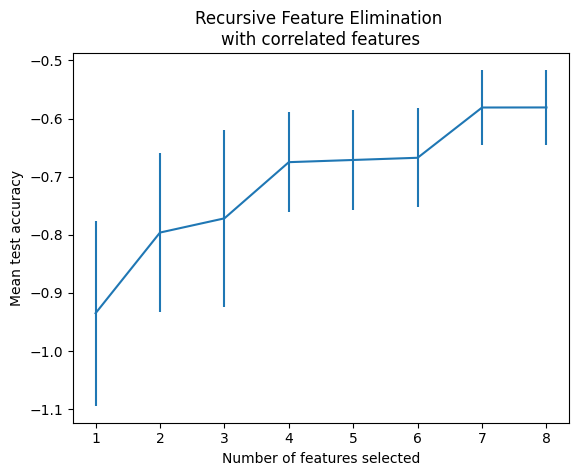


Figure 2.35 Recursive Feature Elimination with correlated features

### Real estate valuation model

* The dataset is divided into two parts:
  + 80% for the training set (…sample)
  + 20% for the test set (…sample)
* Three regression algorithms are applied to build the models: Linear Regression, Ridge Regression, and Random Forest Regression.
* The evaluation metrics for the Random Forest Regression model that achieved the best performance are presented below:

Table . Evaluation metrics table for the real estate valuation models

|  |  |  |
| --- | --- | --- |
|  | Train set | Test set |
| MSE |  |  |
| RMSE |  |  |
| MAPE |  |  |
| R-Squared | 82.82% | 77.75% |

### Real estate price forecast model

* To handle time-based patterns, we extracted temporal features from the **Posted Time** field:
  + **year, month, day, day of week,** and **quarter**: These help capture seasonality and time-related trends.
* The dataset was split into two parts:
  + 80% for training (… samples)
  + 20% for testing (…samples)
* Three forecasting models were applied for comparison: XGBoost Regressor, LightGBM Regressor, and Facebook Prophet.
* The evaluation metrics for the XGBoost Regressor model that achieved the best performance is presented below

Table . Evaluation metrics table for the real estate price forecast models

|  |  |  |
| --- | --- | --- |
|  | Train set | Test set |
| MSE |  |  |
| RMSE |  |  |
| MAPE |  |  |
| R-Squared | 80.42% | 70.77% |

## Apply NSFW image and toxic content classification models

### NSFW image classification models

In this project, I integrated the NudeNetv2 model to detect and classify sexually explicit content in images. I utilized the pre-trained “classifier\_model.onnx” for efficient inference.

The model categorizes content into two classes:

* safe – The image/video is not sexually explicit
* unsafe – The image/video contains nudity or explicit content

I cloned the repository and used the available ONNX version of the classifier to perform predictions on image data.

This classification component plays a crucial role in automatically filtering out inappropriate content, enhancing the safety and compliance of our platform.

## Chapter conclusion

This chapter provides the reader with an overview of the real estate business operations, offering insights into the key activities and processes involved in the industry. It also includes an analysis and design of the system architecture, along with a detailed discussion of the database model. Additionally, the chapter focuses on the development of a price support system, which aims to assist buyers by offering pricing assistance and making real estate transactions more accessible.

# IMPLEMENTATION AND RESULTS EVALUATION

## Description of achieved functionalities

## Chapter conclusion

# CONCLUSION AND DEVELOPMENT DIRECTION

# REFERENCES

<https://aws.amazon.com/vi/what-is/python/>

<https://200lab.io/blog/jupyter-notebook-la-gi>

<https://docs.jupyter.org/en/latest/what_is_jupyter.html>

<https://www.geeksforgeeks.org/introduction-to-java/>

<https://www.geeksforgeeks.org/spring-boot/>

https://viblo.asia/p/tat-tan-tat-ve-mo-hinh-mvc-Rk74avjAJeO

https://viblo.asia/p/luong-di-trong-spring-boot-ORNZqdELK0n

https://www.geeksforgeeks.org/what-is-mysql/

https://200lab.io/blog/reactjs-la-gi

https://www.geeksforgeeks.org/reactjs-introduction/

https://refine.dev/blog/chakra-ui/

# APPENDIX 1

# APPENDIX 2