House Rental Management System
Software Architecture Document
<Version 1.0>

Revision History

Date	Version	Description	Author
<13/02/2024>	<1.0>	<details>	• Mir Mohammod Tahasin MUH2025007M • Suraiya Akter BKH2025013F • Sumaiyea Akter Prema BKH2025023F • Wakil Ahammad ASH2025026M • Mithun Chandra Sarkar MUH2025029M

Table of Contents

1. Introduction

A House Rental System is a software application designed to assist with the management and organisation of properties that are available for rent. This system may be used by property owners, property management companies, and tenants to streamline the process of renting a house or apartment. The system may include features such as: A database of available properties, including information about location, size, price, and amenities.

1.1 Purpose

A house rental system is a software application that allows individuals or companies to manage the rental of houses or other properties. The system typically includes features for managing rental listings, handling rental applications and contracts, accepting payments, and managing maintenance requests and other tasks related to property management. Some house rental systems may also include additional features such as background check capabilities, communication tools for landlords and tenants, and integrations with other property management software. The specific requirements and functionality of a house rental system will depend on the needs and goals of the user.

1.2 Scope

The scope of a house rental system will depend on the specific requirements and goals of the project. However, some common elements that might be included in the scope of a house rental system are:

Rental listings: The system should allow landlords or property managers to create and manage listings for rental properties, including details such as the location, size, features, and rental rate of the property.

Rental applications and contracts: The system should provide a way for landlords to review and accept rental applications, and for tenants to sign rental agreements electronically.

Payment processing: The system should allow landlords to accept rent payments from tenants electronically, and track the payment history of each tenant.

Maintenance requests: The system should provide a way for tenants to submit maintenance requests, and for landlords or property managers to track and manage these requests.

Communication tools: The system should provide a platform for landlords and tenants to communicate with each other, such as through messaging or

email. Background checks: The system may include functionality for performing background checks on rental applicants.

Reporting and analytics: The system should provide landlords or property managers with access to reports and analytics on their rental business, such as information on rental income, occupancy rates, and maintenance expenses.

1.3 Glossary

This section provides definitions for all document names, acronyms, and abbreviations.The application domain's terms and concepts are defined.

I. HRMS-House Rental Management System
II. MB – Megabytes
III. UI – User Interface
IV. SRS – Software Requirement Specifications
V. API – Application Program Interface
VI. XML – Extensible Markup Language
VII. RESTful – Representational State Transfer
VIII. HTML – HyperText Markup Language

1.4 References

Software Architecture in Practice (SEI Series in Software Engineering) 3rd Edition by Len Bass (Author), Paul Clements (Author), Rick Kazman (Author)

1.5 Overview

A house rental system is a system that allows individuals or companies to rent out houses or apartments to tenants. The system typically involves a landlord or property owner who owns the rental unit and a tenant who pays rent to live in the unit. There are several different types of house rental systems, including: Traditional leasing: In this system, a tenant signs a lease agreement with a landlord, agreeing to rent the property. There are several ways that a house rental system can be set up. In some cases, the landlord or property owner may handle the rental process directly, advertising the property for rent and managing all aspects of the rental agreement themselves. In other cases, a third party such as a property management company may be responsible for managing the rental of the property on behalf of the landlord or owner. A house rental system may also involve the use of software or other technology to help manage the rental process. For example, a landlord may use a property management software program to track rent payments, schedule maintenance and repairs, and handle other aspects of the rental process. Overall, the goal of a house rental system is to provide a clear and organised way for landlords and tenants to manage the rental of a house or other residential property. This feature enables landlords to list their properties, including details such as property type, location,

price, and amenities. This feature enables landlords to collect rent payments online and track payment history. This feature allows landlords to manage maintenance requests from tenants, schedule repairs, and track the status of maintenance tasks.

2. Architecturally Significant Requirements

Number	ASR	Rationale
Scalability	The system must be able to handle a growing number of users, houses, and concurrent transactions as the user base expands.	As the platform gains popularity, it should be able to scale horizontally to accommodate increased demand without compromising performance.
Performance	The system should provide low-latency responses for actions such as searching, viewing houses, and real-time chat to ensure a responsive user experience.	Users expect quick and efficient interactions; hence, performance is crucial for user satisfaction.
Security	User data, including personal information, chat messages, and financial transactions, must be securely stored and transmitted.	Security is paramount to protect user privacy, prevent unauthorised access, and ensure the integrity of sensitive information.
Reliability and Availability	The system should have high availability, minimizing downtime and ensuring users can access the platform reliably.	The system should have high availability, minimizing downtime and ensuring users can access the platform reliably.
Integration with External Services	The system must seamlessly integrate with external services, such as mapping services (e.g., Google Maps) for accurate location information.	Integration enhances the user experience by providing additional functionalities and accurate data.
Usability and User Experience	The user interface should be intuitive, user-friendly, and	A well-designed and easy-to-use interface contributes to user satisfaction,

	accessible to ensure a positive user experience.	adoption, and overall success of the platform.
Maintainability	The system should be designed and implemented in a modular and maintainable manner to facilitate future updates, enhancements, and bug fixes.	Ease of maintenance is critical for ensuring the long-term viability and adaptability of the system.
Data Integrity and Consistency	The system should maintain data integrity and consistency, especially in scenarios where multiple users may be concurrently updating or accessing information.	Ensuring the accuracy and reliability of data is essential to avoid discrepancies and errors in the application.
Regulatory Compliance	The system must adhere to relevant legal and regulatory requirements governing real estate transactions, user data protection, and financial transactions.	Compliance with legal and regulatory standards is crucial to avoid legal issues, protect user rights, and ensure the system operates within the bounds of the law.
Auditability and Logging	The system should maintain comprehensive audit logs, tracking significant events, user interactions, and system activities for security, troubleshooting, and compliance purposes.	Detailed logs are essential for monitoring system behavior, identifying security incidents, and providing an audit trail to support investigations and compliance requirements.

3.Quality Attributes

Number	Quality Attributes
01	Scalability
02	Performance

03	Security
04	Reliability and Availability
05	Usability and User Experience
06	Maintainability

3.1 Quality Attributes Scenarios

Scalability

Scenario 1: Scalability under Increased User Load

Stimulus	A sudden influx of new users during a promotional campaign.
Source	Increased user registration and house listing activities.
Artifact	User registration and house listing modules.
Environment	Normal system operation with a sudden spike in user activity.
Response	The system should gracefully handle the increased load by dynamically allocating resources to ensure low-latency responses.
Response Measure	Maintain response times below a specified threshold (e.g., 95th percentile latency) even under increased load.

Scenario 2: Horizontal Scalability

Stimulus	Continuous growth in the user base.
Source	Incremental increase in the number of registered users and posted houses.

Artifact	Database and server infrastructure.
Environment	Ongoing operation with gradual user base expansion.
Response	The system should scale horizontally by adding more servers to distribute the load evenly.
Response Measure	Monitor and adjust server instances based on resource utilization to maintain system performance

Performance:

Scenario 1: Quick Search Response

Stimulus	Tenants initiate a search for available houses.
Source	Tenant interaction with the search feature
Artifact	Search functionality and database queries.
Environment	Normal system operation.
Response	The system should respond to search queries within a predefined time limit.
Response Measure	95% of search queries should return results within 2 seconds.

Scenario 2: Real-time Chat Responsiveness

Stimulus	Landlord and tenant engage in real-time chat.
Source	User interaction with the chat module.
Artifact	Chat messaging system.
Environment	Continuous chat activity during peak hours.
Response	The system should ensure minimal chat message delivery latency.

Response Measure	99% of chat messages should be delivered within 1 second.

Security:

Scenario 1: Secure Data Transmission

Stimulus	Tenant submits personal information during the house request process.
Source	Tenant interaction with the system.
Artifact	Data transmission protocols.
Environment	User interaction in a potentially insecure network.
Response	The system should encrypt sensitive data during transmission.
Response Measure	Use industry-standard encryption algorithms to ensure secure data transmission.

Scenario 2: Access Control

Stimulus	Unauthorized attempt to access landlord account settings
Source	Malicious user attempting unauthorized access.
Artifact	Malicious user attempting unauthorized access.

Environment	Normal system operation with potential security threats.
Response	The system should deny access and log the unauthorized attempt.
Response Measure	Implement multi-factor authentication and log all unauthorized access attempts.

Reliability and Availability:

Scenario 1: System Recovery after Failure

Stimulus	Unexpected server failure during peak hours.
Source	Hardware or software failure.
Artifact	Server infrastructure.
Environment	Normal system operation with potential failure events.
Response	The system should recover quickly without data loss.
Response Measure	Achieve a system recovery time of less than 5 minutes with minimal data inconsistency.

Scenario 2: High Availability during Maintenance

Stimulus	Planned system maintenance activities.
Source	Scheduled maintenance events.
Artifact	Load balancer and redundant server instances.
Environment	Scheduled maintenance period.

Response	The system should remain available with minimal disruption.
Response Measure	Ensure at least 99.9% availability during scheduled maintenance.

Usability and User Experience:

Scenario 1: Intuitive House Posting

Stimulus	Landlord initiates the process of posting a new house.
Source	Landlord interaction with the house posting feature.
Artifact	House posting user interface.
Environment	Landlord using the system.
Response	The system should provide clear instructions and an intuitive interface for easy house posting.
Response Measure	90% of landlords can successfully post a house on their first attempt.

.

Scenario 2: Responsive User Interface

Stimulus	Tenant interacts with the search and house viewing features.
Source	Tenant using the system on different devices.

Artifact	User interface design and responsiveness.
Environment	Tenant accessing the system from various devices.
Response	The system should adapt to different screen sizes and provide a seamless user experience.
Response Measure	Achieve a mobile responsiveness score of 90 or above on industry-standard tests.

Maintainability:

Scenario 1: Modular Feature Updates

Stimulus	Introduction of a new feature (e.g., enhanced search filters).
Source	Development team releasing a feature update.
Artifact	Feature modules and codebase.
Environment	Ongoing system maintenance and updates.
Response	The system should allow for the addition of new features with minimal impact on existing functionality.
Response Measure	Introduce a new feature with less than 5% codebase modification in existing modules.

Scenario 2: Codebase Refactoring

Stimulus	Identification of code smells and technical debt.
Source	Development team performing code reviews.

Artifact	Codebase and software architecture.
Environment	Continuous development and code maintenance.
Response	The system should support ongoing code refactoring to improve maintainability.
Response Measure	Reduce code smells by at least 20% in each major release cycle.

3.2 Utility Tree

A utility tree begins with the word "utility" as the root node. Utility is an expression of the overall "goodness" of the system. We then elaborate this root node by listing the major quality attributes that the system is required to exhibit.

Once the ASRs are recorded and placed in the tree, you can now evaluate them against the two criteria we listed above: the business value of the candidate ASR and the architectural impact of including it. You can use any scale you like, but we find that a simple "H" (high), "M" (medium), and "L" (low) suffice for each criterion.

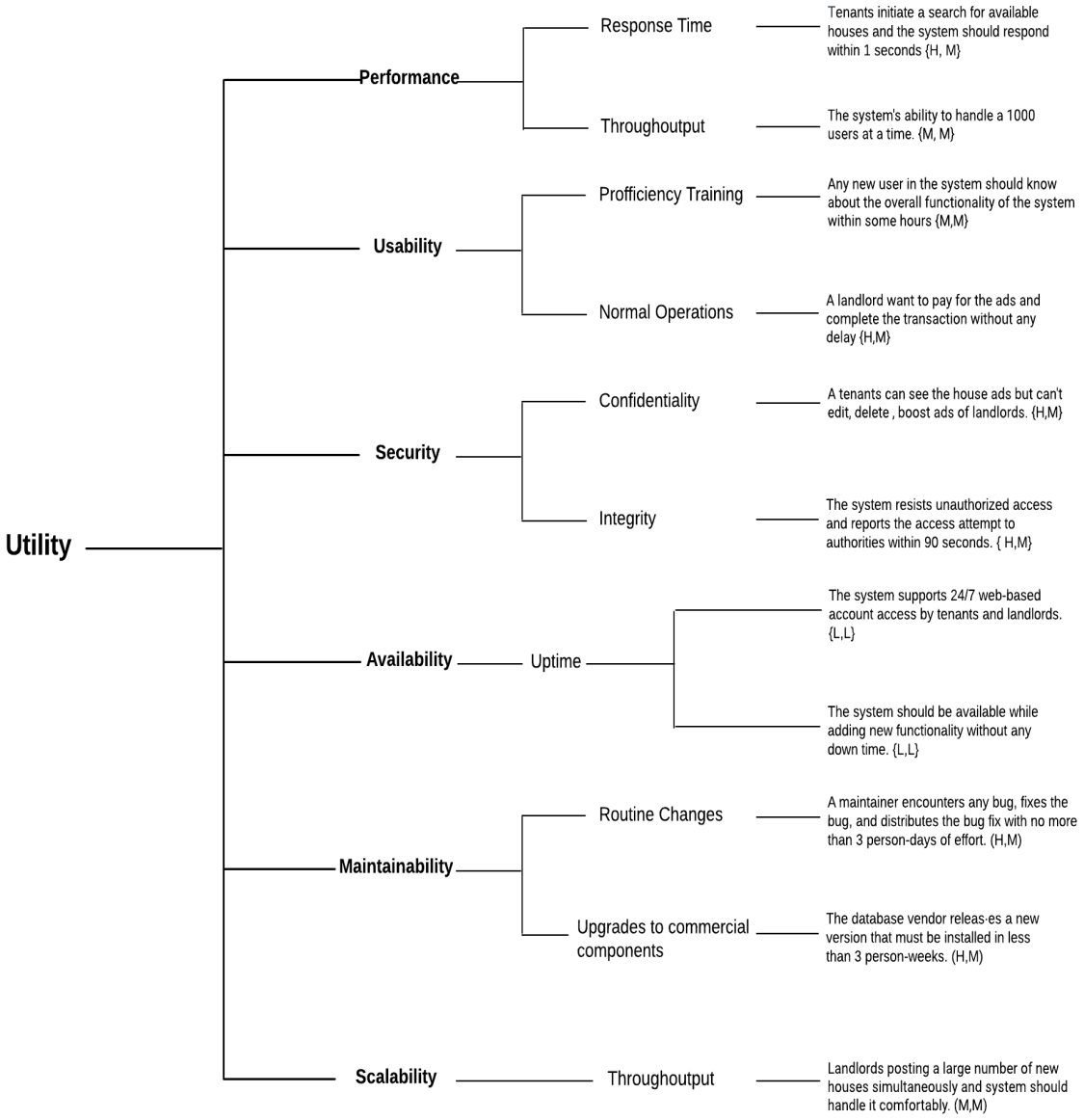

Figure 1: Utility Tree

3.3 Design Tactics

Design tactics are specific strategies or techniques employed to address and achieve the desired quality attributes identified in the utility tree. Here are design tactics for the key quality attributes in your system:

Performance:

Caching:

- Tactic: Implement caching mechanisms for frequently queried data.

- Rationale: Reduces response time by serving cached data, especially for search queries and house details.

Load Balancing:

- Tactic: Distribute incoming user requests evenly across multiple servers.
- Rationale: Supports horizontal scalability, ensuring even load distribution during increased user load.

Scalability:

Elastic Scaling:

- Tactic: Implement auto-scaling capabilities to dynamically adjust resources based on demand.
- Rationale: Ensures the system can handle varying loads, scaling up or down as needed.

Sharding:

- Tactic: Partition the database to distribute data across multiple servers.
- Rationale: Enhances horizontal scalability by spreading the data load across multiple storage nodes.

Security:

Encryption:

- Tactic: Use strong encryption algorithms for data transmission and storage.
- Rationale: Protects sensitive user data, addressing the secure data transmission scenario.

Multi-Factor Authentication (MFA):

- Tactic: Implement MFA for user authentication.
- Rationale: Enhances access control and mitigates the risk of unauthorized access.

Reliability and Availability:

Redundancy:

- Tactic: Deploy redundant components and servers.
- Rationale: Improves system reliability by ensuring backup resources are available in case of failures.

Fault Tolerance:

- Tactic: Design the system to gracefully handle and recover from failures.
- Rationale: Reduces downtime and ensures quick system recovery, aligning with reliability goals.

Usability and User Experience:

Responsive Design:

- Tactic: Adopt responsive design principles for the user interface.
- Rationale: Ensures a seamless user experience across different devices, addressing the responsive user interface scenario.

User Guidance:

- Tactic: Provide clear instructions and tooltips for users during critical tasks.
- Rationale: Enhances the intuitive house posting scenario, making the posting process user-friendly.

Maintainability:

Microservices Architecture:

- Tactic: Adopt a microservices architecture for modular feature updates.
- Rationale: Facilitates independent development and deployment of features with minimal impact on existing functionality.

Continuous Refactoring:

- Tactic: Enforce continuous code refactoring practices.
- Rationale: Supports ongoing codebase improvements, reducing technical debt, and enhancing maintainability.

4. Architectural Representations

4.1 Context Diagram

A context diagram provides an overview of a system and its interactions with external entities. In this case, the house rental management system involves several external entities.

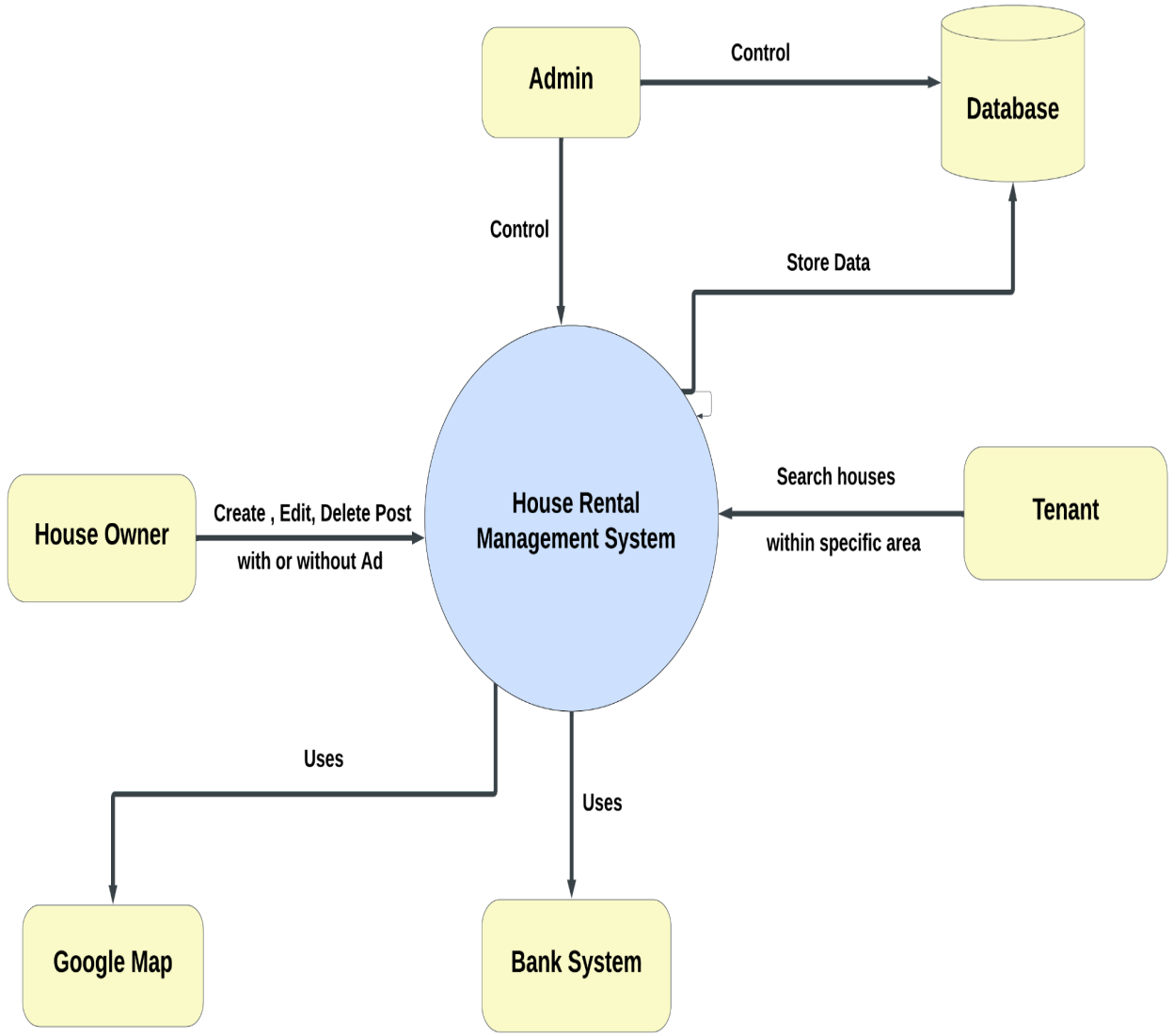

Figure 2: Context Diagram

The key external entities and their relationships with the House Rental Management System:

Entity	Relationship
Tenant	Rents a house from a house owner.Searches for Houses, uses House Rental Management System
Admin	Manages the house rental management system. Controls House Rental Management System.
GoogleMaps	A mapping service that is used to display the location of houses to tenants.Used by House Rental Management System
BankSystem	A financial institution that is used to process payments between tenants and house owners.Used by HouseRentalManagementSystem
HouseOwner	Owns a house that they want to rent out.Creates House, posts House, uses House Rental Management System
Database	Stores information about house owners, tenants, houses, and rental agreements.

4.2 Use Case Diagram

A Use Case Diagram is a vital tool in system design, it provides a visual representation of how users interact with a system. In the above use case diagram for House Rental Management System consist 5 actors and 11 use cases

Actors : Owner, Tenants, Admin, System, Bank System

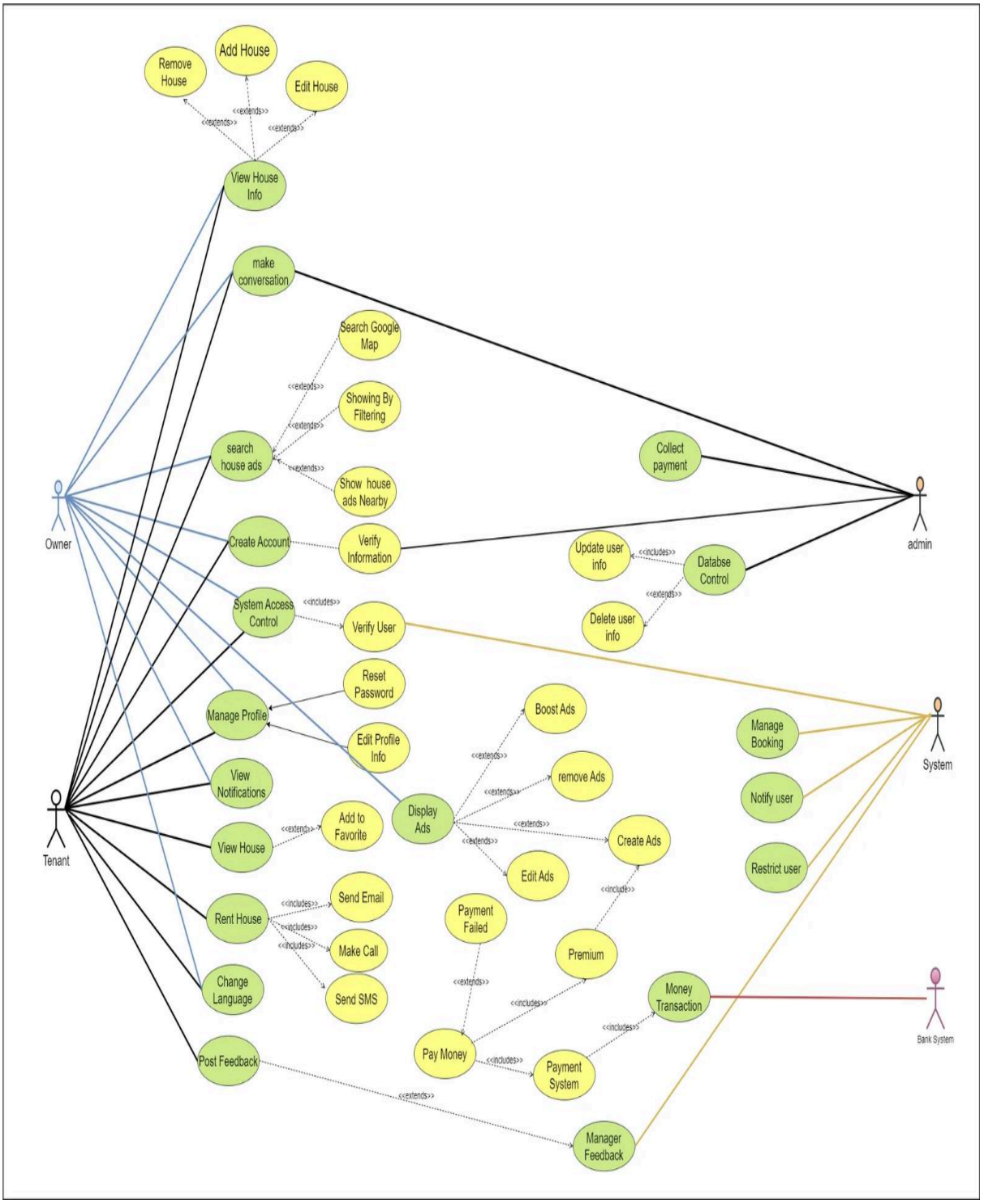

Figure 3: Use Case Diagram

4.3 Process View

The process view of a system provides a dynamic perspective, focusing on the interactions and communication among different components or processes within the system. It illustrates how various elements collaborate and communicate to achieve the overall functionality of the

system. In software engineering and systems architecture, the process view helps in understanding the runtime behavior of a system.

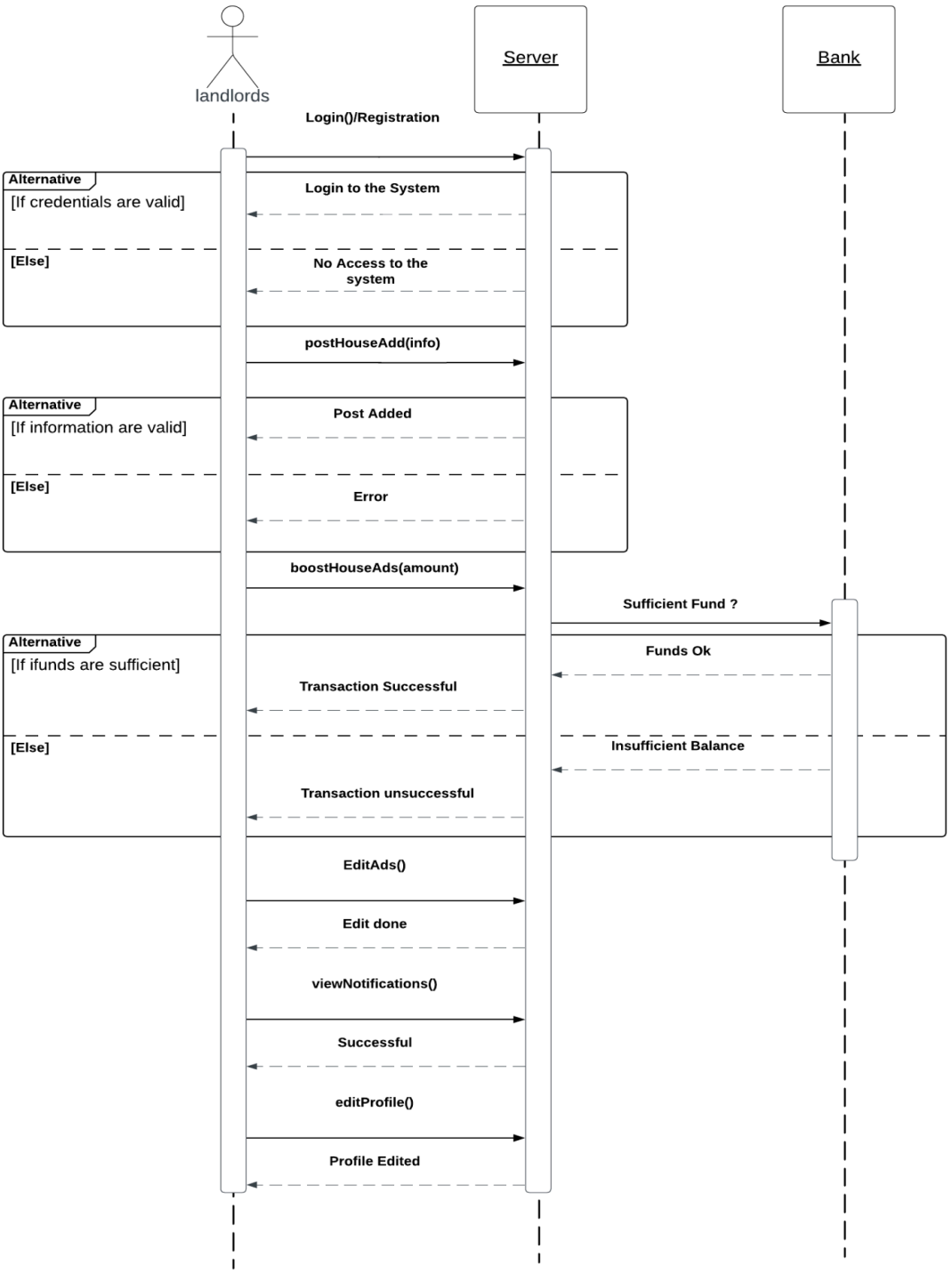

Figure 4: Sequence Diagram for Landlords

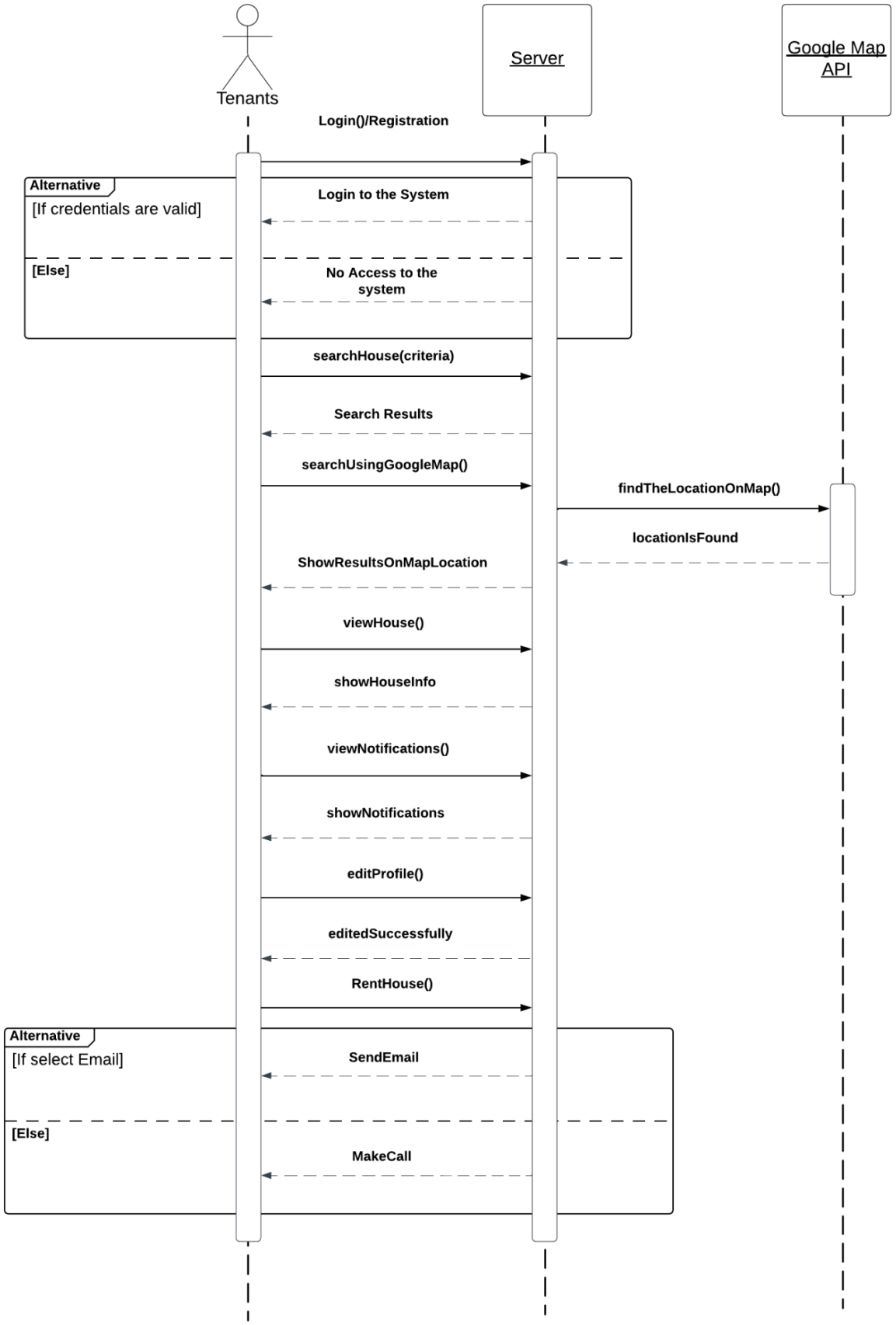

Figure 5 : Sequence Diagram For Tenants

4.4 Logical view

The logical view of a House Rental Management System provides a high-level representation of the system's functionality, emphasizing the organization of key components and their interactions.

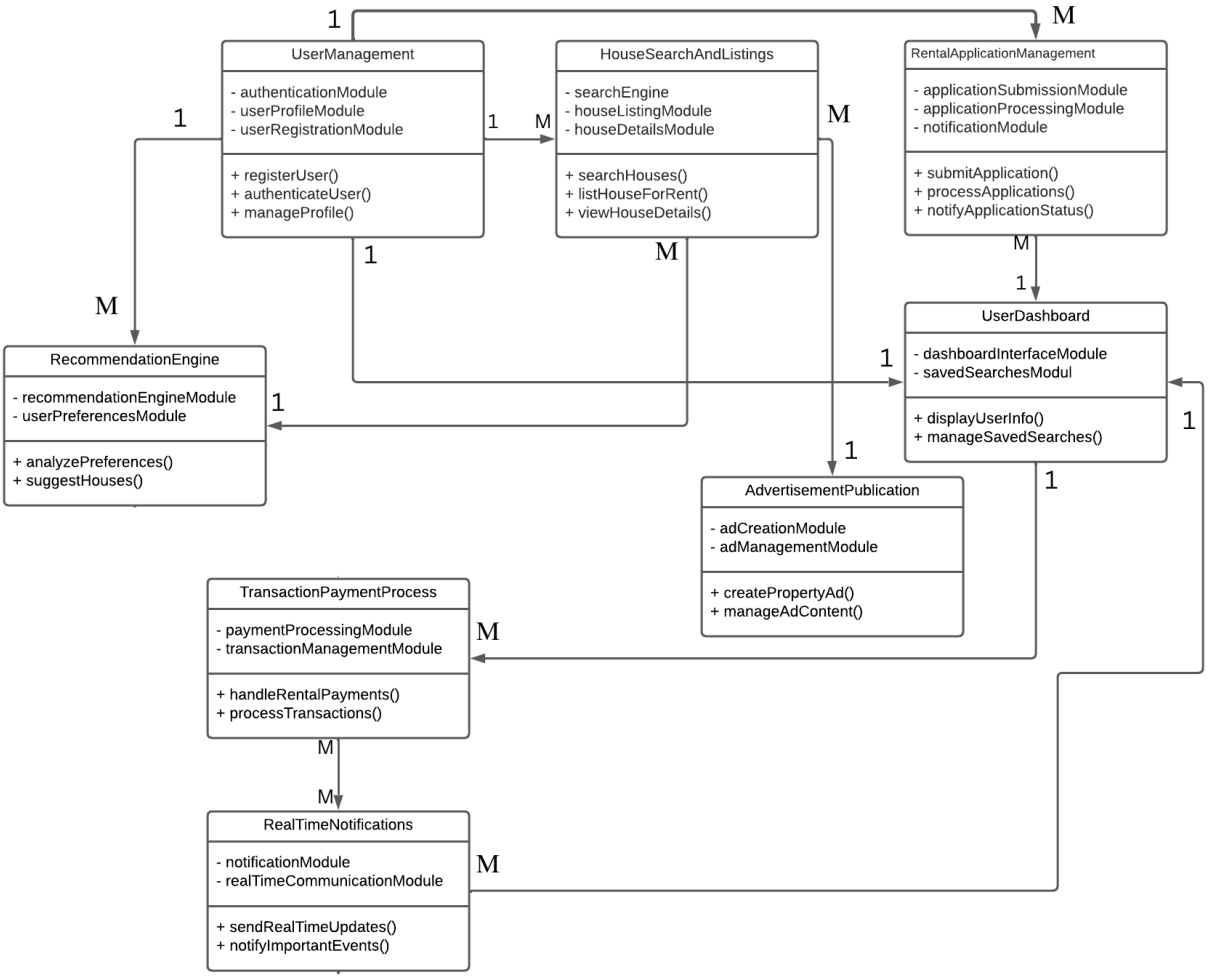

Figure 6 : Logical View of House Rental Management System

This UML class diagram represents the logical view of the House Rental System, including the main functions and components described. Each major function has its corresponding module, and components are encapsulated within the respective modules. The diagram illustrates the relationships and dependencies among different modules and functions in the system.

User Management:

- Functions: Manages user registration, authentication, and profiles.
- Components: User registration module, authentication module, user profile module.

House Search and Listing:

- Functions: Handles searching for houses, listing houses for rent, and viewing detailed house information.
- Components: Search engine, house listing module, house details module.

Rental Application Management:

- Functions: Manages the submission and processing of rental applications, notifies users of application status.
- Components: Application submission module, application processing module, notification module.

Recommendation Engine:

- Functions: Analyzes user preferences and suggests recommended houses.
- Components: Recommendation engine module, user preferences module.

Advertisement Publication:

- Functions: Creates and publishes property ads, manages ad content and images.
- Components: Ad creation module, ad management module.

User Dashboard:

- Functions: Displays user-specific information, manages saved searches or favorite properties.
- Components: Dashboard interface module, saved searches module.

Transaction and Payment Processing:

- Functions: Handles rental payments, processes transactions securely.
- Components: Payment processing module, transaction management module.

Real-Time Notifications:

- Functions: Sends and receives real-time updates, notifies users of important events.
- Components: Notification module, real-time communication module.

Property Availability Tracking:

- Functions: Monitors property availability, updates availability status in real-time.
- Components: Availability tracking module.

4.5 Deployment View

The deployment view provides an architectural perspective on how the components of a system are physically deployed across various hardware and software resources. In the context of a house rental management system, the deployment view illustrates

how different components of the system are distributed across servers, networks, and other infrastructure elements.

This section is organized by physical network configuration; each such configuration is illustrated by a deployment diagram,followed by a mapping of processes to each processor.

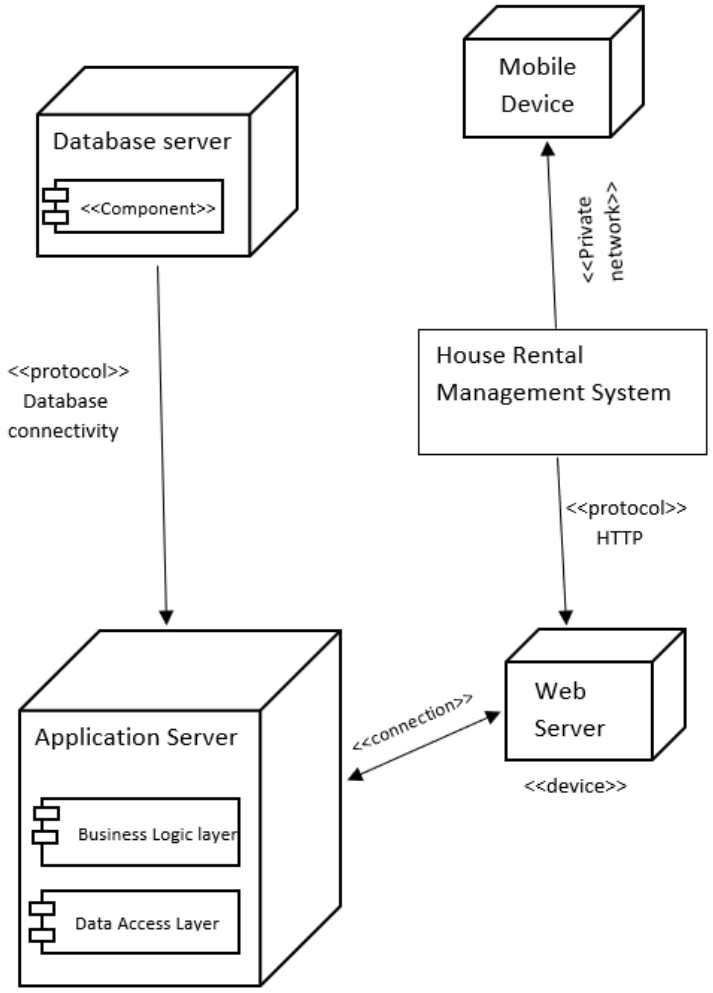

Figure 7 : Deployment view of House Rental Management System

1. Server Infrastructure:

Components:

- Web Server: Hosts the frontend components and handles user interactions.
- Application Server: Executes the backend logic and business processes.
- Database Server: Manages the storage and retrieval of data.

2. Client Devices:

Components:

- Web Browser: Renders the frontend interfaces for users.

● Mobile Devices: Devices used by tenants and house owners to interact with the system.

3. Communication Protocols:

Components:

● HTTP/HTTPS: Used for communication between the web server and clients. ● Database Connectivity Protocols: Facilitate communication between the application server and the database server.

4. Database System:

Components:

● Relational Database Management System (RDBMS): Manages and stores data in a structured manner.

4.6 Implementation View

The implementation view of a software system is concerned with the organisation of the software during implementation, including components, modules, layers, and the development workflow. Here are the key functions and components that fall under the implementation view in our "to-let" system:

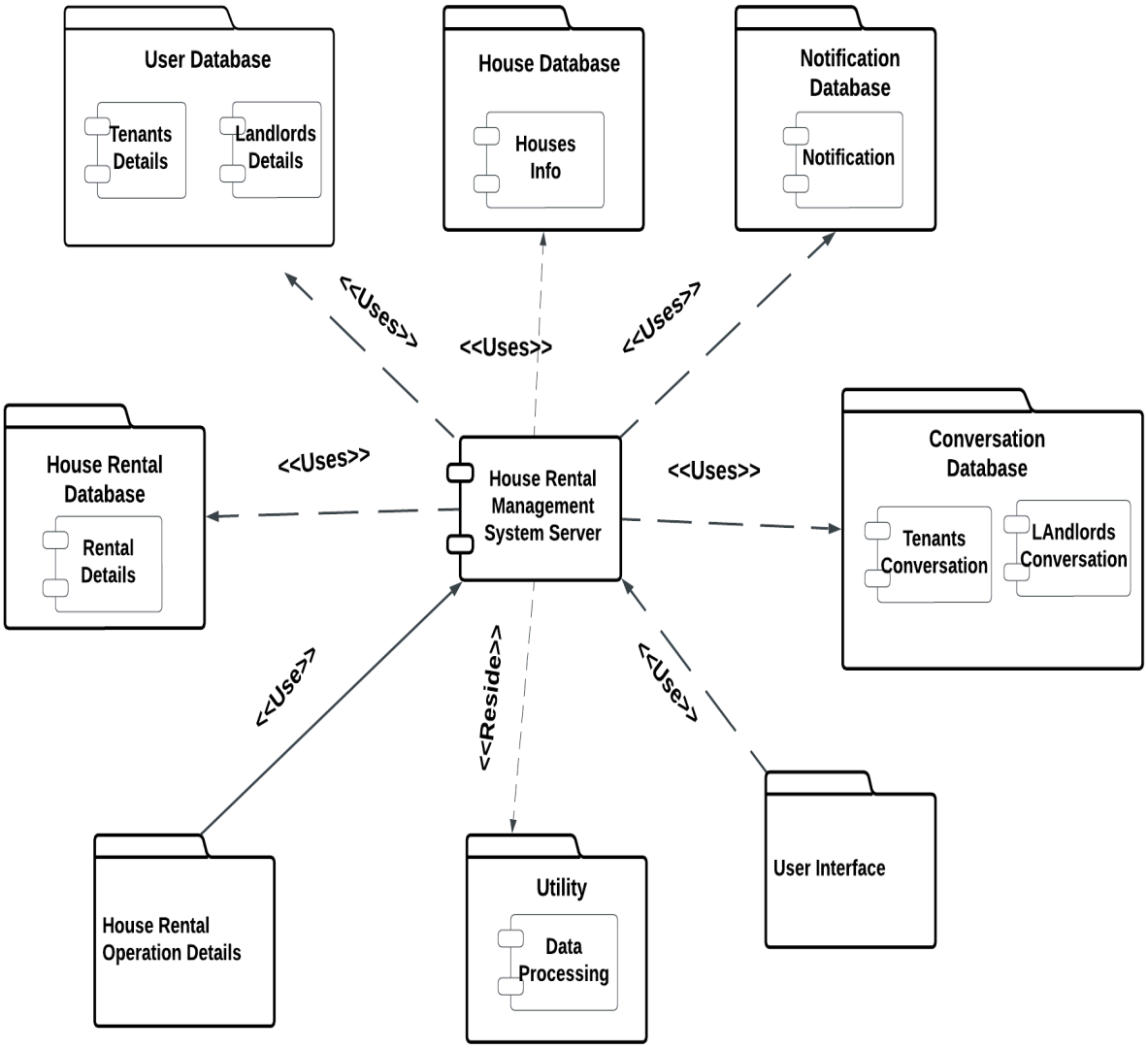

Figure 8 : Implementation View Of House Rental Management System

1. Module Organization

Components:

UserManagementModule: Organizes user-related functionalities.
HouseModule: Organizes house-related functionalities.
RentalApplicationModule: Organizes rental application-related functionalities.
AdvertisementModule: Organizes advertisement-related functionalities.
PaymentModule: Organizes payment-related functionalities.
NotificationModule: Organizes notification-related functionalities.
DashboardModule: Organizes user dashboard-related functionalities.

2. Layered Architecture

Layers:

- ➢ Presentation Layer: Handles user interactions, interfaces, and presentation logic.
- ➢ Business Logic Layer: Contains the core business logic for house management, application processing, payment handling, etc.
- ➢ Data Access Layer: Manages interactions with databases and data storage.

3. Component-Based Implement

Components:

- ➢ SearchEngineComponent: Manages house searches and queries the house database.
- ➢ AdManagementComponent: Handles the creation, updating, and publication of property advertisements.
- ➢ PaymentComponent: Processes payments and manages transaction details.
- ➢ NotificationComponent: Manages the generation and delivery of real-time notifications.
- ➢ MapsIntegrationComponent: Integrates with external mapping services for property locations and directions.

4. Database Schema

Tables:

- ➢ UserTable: Stores user information.
- ➢ HouseTable: Contains details about houses, their owners, and locations.
- ➢ ApplicationTable: Stores rental applications and their statuses.
- ➢ AdTable: Manages property advertisements.
- ➢ PaymentTable: Stores payment details.
- ➢ NotificationTable: Records real-time notifications.
- ➢ DashboardTable: Holds user-specific dashboard information.

5. Implement Tools and Technologies

Key Technologies:

- ➢ Frontend Technologies: HTML, CSS, JavaScript, React (or other frontend frameworks).
- ➢ Backend Technologies: Node.js, Express.js (or other server-side frameworks).
- ➢ Database Management System: PostgreSQL (or other relational database systems).
- ➢ External Service Integrations: Google Maps API, Recommendation Engine API.

6. Implement Workflow

Practices:

- ➢ Version Control: Git (or other version control systems).
- ➢ Collaboration Tools: Collaboration platforms like GitHub or GitLab.
- ➢ Continuous Integration/Continuous Deployment (CI/CD): Automated testing and deployment practices.

7. Interfaces and APIs

Key Interfaces:

- ➢ RESTful APIs: Used for communication between frontend and backend components.
- ➢ External Service APIs: Used for integrating with external services such as mapping and recommendation engines.

5. Architectural Goals and Constraints

5.1 Architectural Goals

Scalability: Design the system to handle a growing number of users, properties, and transactions without sacrificing performance or reliability. This includes the ability to scale both vertically (upgrading hardware resources) and horizontally (adding more servers).

Reliability: Ensure that the system is highly available and resilient to failures. Minimize downtime and data loss through redundancy, fault tolerance mechanisms, and automated failover procedures.

Security: Implement robust security measures to protect sensitive data such as tenant information, financial transactions, and property details. This includes user authentication, access control, data encryption, and protection against common security threats like SQL injection and cross-site scripting (XSS).

Performance: Optimize system performance to deliver fast response times and minimize latency, especially for interactive features like property search and booking. This involves efficient data retrieval, caching, and minimizing network overhead.

Usability: Design a user-friendly interface that makes it easy for landlords, tenants, and administrators to navigate the system, perform tasks, and access relevant information. Provide clear instructions, intuitive workflows, and responsive design across different devices.

Flexibility: Build a flexible and extensible architecture that can accommodate future changes, enhancements, and integrations with third-party services. Use modular components, well-defined APIs, and industry-standard protocols to facilitate interoperability and adaptability.
Maintainability: Develop clean, modular, and well-documented code that is easy to understand, debug, and maintain. Follow best practices for software engineering, version control, and automated testing to facilitate ongoing development and support.

5.2 Architectural Constraints

Budget: Adhere to budgetary constraints by selecting cost-effective technologies, optimizing resource utilization, and prioritizing features based on their value to users and stakeholders.
Time: Time constraints refer to the limitations on the available time to complete the project within a specified deadline or timeframe. These constraints can arise from various factors and can significantly impact the planning, execution, and delivery of the system.
Technology Stack: Work within the constraints of the chosen technology stack, including programming languages, frameworks, libraries, and database systems. Consider factors such as developer expertise, ecosystem support, **Resource Allocation:** Time constraints affect resource allocation, including the availability of developers, designers, testers, and other team members. Limited time may require prioritizing certain tasks or features, reallocating resources, or outsourcing certain components to meet deadlines.and licensing considerations.
Testing and Quality Assurance: Limited time may constrain the thoroughness of testing and quality assurance activities. Testing efforts may need to be streamlined, automated, or focused on high-priority areas to ensure that the system meets quality standards while adhering to deadlines.
Risk Management: Time constraints can increase the risk of project delays, scope creep, and technical debt. Project managers must identify and mitigate risks early in the development process to minimize the impact of unforeseen challenges on project timelines.
User Feedback and Iterative Improvement: Time constraints may limit the opportunity for extensive user testing and feedback collection before the system's launch. However, it's essential to plan for post-launch iterations and updates based on user feedback to continuously improve the system over time.

6. Appendix

A. Design Processes

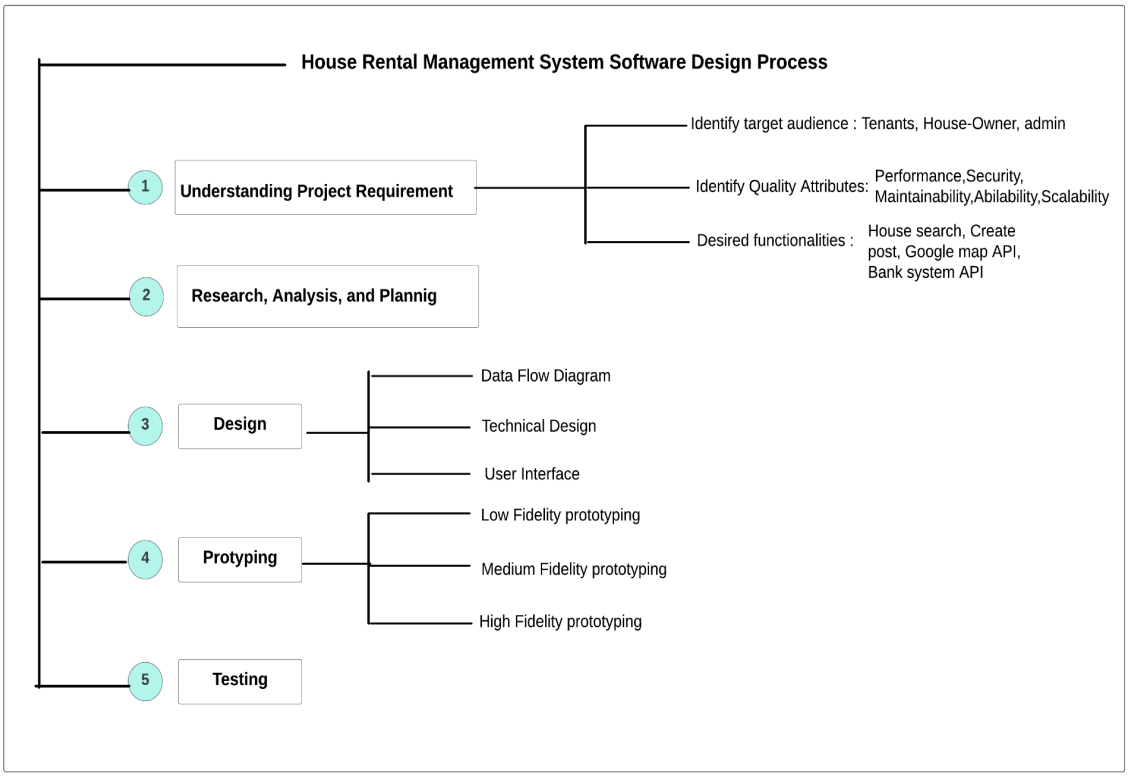

Figure 9 : Design Processes

1. Requirements Gathering and Analysis

- **Identify target audience:** This involves understanding who will be using the system, such as tenants, house owners, and administrators.
- **Identify Quality Attributes:** This involves defining the desired qualities of the system, such as performance, security, maintainability, availability, and scalability.
- **Desired functionalities:** This involves specifying the features and functions that the system should have, such as house search, creating posts, integrating with Google Maps API, and integrating with a bank system API.

2. Research, Analysis, and Planning

The research, analysis, and planning phase aims to dissect gathered requirements and subsequently formulate a comprehensive design plan.

3. **Design**

- **Data Flow Diagram**: This involves creating a visual representation of how data will flow through the system.
- **Technical Design:** Choose programming languages, frameworks, and databases based on requirements and expertise.
- **User Interface:** This involves designing the user interface of the system, which should be easy to use and navigate for all users.

4. **Prototyping**

- **Low Fidelity prototyping:** This involves creating a basic prototype of the system to get feedback from users.
- **Medium Fidelity prototyping:** This involves creating a more detailed prototype of the system to get further feedback from users.
- **High Fidelity prototyping:** This involves creating a close-to-final prototype of the system to get final feedback from users.

5. **Testing**

   This involves testing the system to ensure that it meets all of the requirements and that it is free of bugs.

B. Architecture Patterns

1. Client-Server pattern

In our system that is house rental management systems, we use the Client-Server pattern. It's preferable depends on various factors such as the requirements of the system, the expected user base, security considerations, scalability needs, and development constraints.

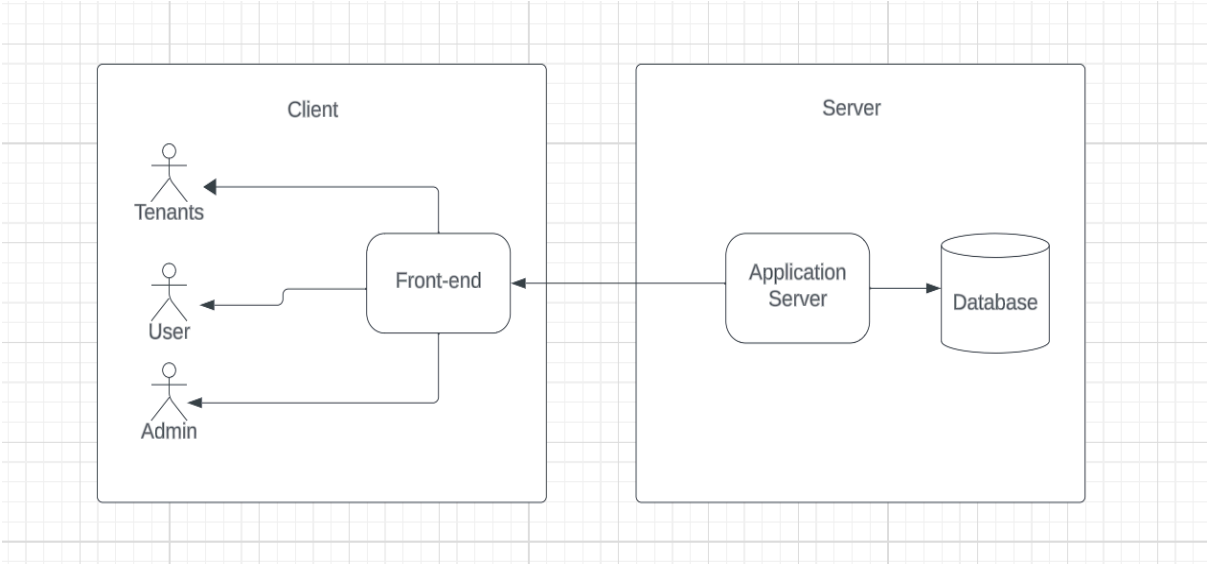

Figure10 : Client-server architecture

In the context of our house rental management system, here are some features that leverage the client-server pattern:

Feature	Description	
	Client side	Server side
User Authentication and Authorization	Users (house owners, renters) interact with the client-side interface to log in and authenticate.	The server manages user authentication, authorizes access, and ensures secure communication.
User Account Management	Users update their account information and preferences through the client-side interface.	The server handles requests to update user profiles, manages account details, and ensures data consistency.
Advertisement Posting and Management	House owners use the client-side interface to create and manage rental	The server handles incoming requests to create/update

	advertisements.	advertisements, validating and storing data in the database.
Google Maps Integration	Renters interact with the client-side map interface to visually explore available houses in specific areas.	The server coordinates with the Geospatial Module, integrates with Google Maps API, and manages location-based data.

2. MVC Design Pattern

Some functionality of our system also follows the **Model View Controller(MVC)** pattern.The Model-View-Controller (MVC) pattern is a design pattern that separates an application into three interconnected components: Model, View, and Controller. Each component has its specific role in managing and organizing the application's logic and user interface.

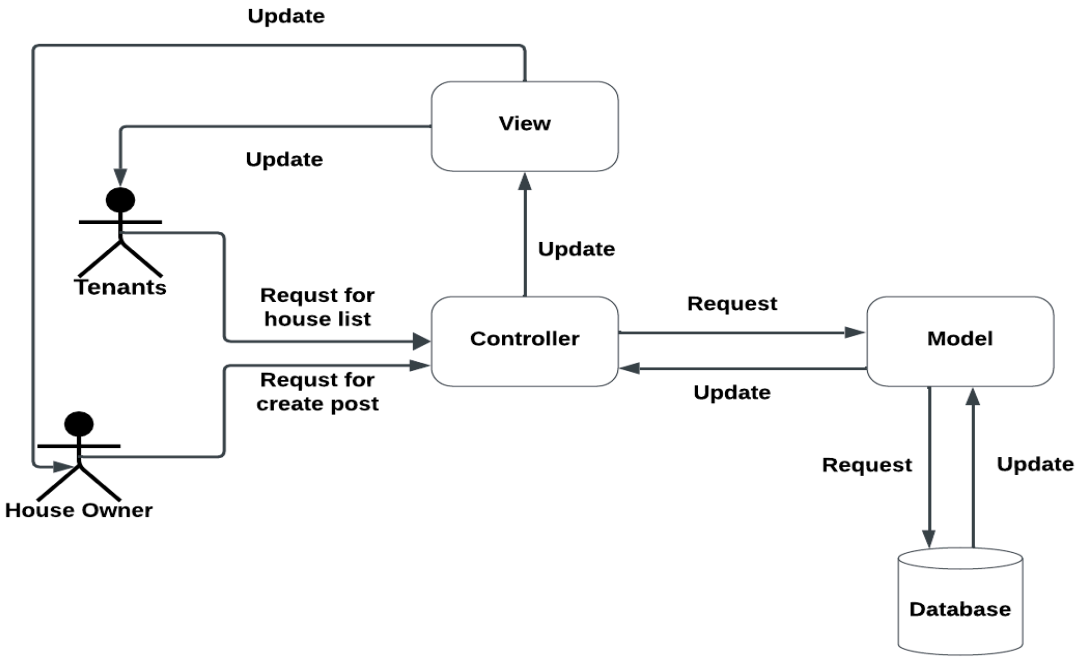

Figure11: MVC Pattern

In the context of our house rental management system, here are some features that utilize the MVC pattern:

Feature	**Description**		
	Model	View	Controller
House Search and Filtering	Stores and retrieves data related to available houses, including location, price, and amenities.	Displays search results and provides filters for users to refine their searches.	Processes user search queries, interacts with the Model to fetch relevant data, and updates the View.
Messaging System	Manages data related to messages, conversations, and user interactions.	Displays the messaging interface and conversation history to users.	Handles user input, manages message sending and receiving, and updates the Model.
Advertisement Posting and Management	Manages the data related to house advertisements, including details, pricing, and advertisement status.	Displays the advertisement creation form and information to the house owner.	Handles user input, validates data, and updates the Model with new advertisement details.

3. Blackboard Pattern

In the context of our house rental management system, the "blackboard" can be metaphorically represented by the central repository that holds and manages shared knowledge. In practical terms, this could be a database or another form of data storage that serves as a centralized hub for information related to house rental activities.

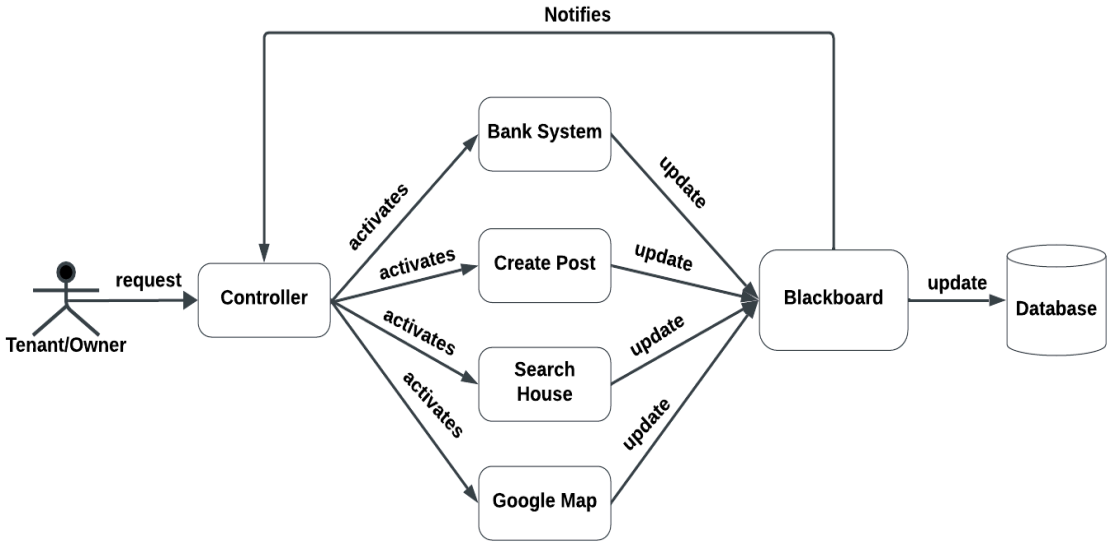

Figure12 : Blackboard Pattern

Here are some features that could leverage the Blackboard pattern:

Feature	**Description**		
	Blackboard	Knowledge Sources	Controller
Real-time Notifications and Alerts	Manages notification-related data, such as new messages, updated advertisements, or house availability.	Components responsible for event detection, notification generation, and delivery contribute their knowledge to the Blackboard.	The Blackboard Coordinator monitors system events, triggers notifications based on user preferences, and updates notification status.
	Stores information about house	Components responsible for	The Blackboard Coordinator

Advertisement Management	advertisements, including owner details, ad status, and payment information.	advertisement creation, payment processing, and status tracking contribute their knowledge to the Blackboard.	ensures that advertisement data is accurately updated, payments are processed, and notifications are sent to relevant parties.
House Search and Matching	Acts as a central repository for housing data, including details such as location, size, amenities, and rental prices.	Specialized components contribute information about available houses, their features, and rental status.	The Blackboard Coordinator orchestrates the integration of data from different sources and facilitates house search and matching algorithms.
Payment Processing	Stores payment-related data, tracks transaction status, and manages billing information for advertisement services.	Components responsible for payment processing, transaction tracking, and billing contribute their knowledge to the Blackboard.	The Blackboard Coordinator ensures that payment transactions are processed securely, updates payment status, and generates billing reports.