GAS Journal of Engineering and Technology (GASJET)

Volume 3 | Issue 4, 2026 Homepage: https://gaspublishers.com/gasjet-home/ ISSN: 3048-5800

Development of Dream Key Real Estate Management System

Daniel Osilama Izuagie[1], John Temitope Ogbiti[2] and Umoru Oshoke John-Francise[3]

[1,2&3]Department of Computer Science, Edo University Iyamho, Auchi, Nigeria

Received: *10.03.2026* | ***Accepted:*** *29.03.2026* | ***Published:*** *01.04.2026*

***Corresponding Author:** John Temitope Ogbiti

DOI: *10.5281/zenodo.19358440*

Abstract **Review Article**

The rapid digitization of the real estate market has created a significant demand for unified platforms that can bridge the gap between property inventory management and client discovery. This project presents the design and implementation of the Dream Key Real Estate Management System (DREAMS), a full-stack web application developed to address the fragmentation of administrative workflows and the lack of real-time search efficiency in traditional real estate operations. The system was engineered using a decoupled three-tier architecture, utilizing React.js for a responsive user interface, Java Spring Boot for robust server-side business logic, and PostgreSQL for secure, persistent data storage. Key functional modules include a Unified Asset Management system that utilizes single-table inheritance to manage diverse property types, a Dynamic Search and Filtering Engine, and an Administrative Dashboard for CRUD operations and inquiry tracking.

Keywords: real estate management system, web application development, property search, full stack architecture, database management.

1.0 INTRODUCTON

The global real estate industry has undergone a fundamental transformation, mirroring other global sectors by shifting reliance from traditional, paper-intensive methods to sophisticated digital platforms for managing transactions and assets. Over the past decade, the rapid advancement of internet and mobile technologies has empowered consumers with unprecedented access to property data, virtual viewing capabilities, and instantaneous market updates, thereby restructuring the foundational relationship between buyers, sellers, and agents (Jones & Chen, 2022). This digital evolution necessitates the creation of integrated, reliable management systems that can effectively handle the complexities of listing maintenance, client interaction, and financial tracking across sales, land development, and rental segments, all within a centralized framework.

Historically, the administration of core real estate activities, including property listing for sale and rent, land parcel tracking, and routine facility management, has often relied on manual, decentralized, and often fragmented methods. These conventional approaches, typically involving scattered spreadsheets, physical contracts, and non-integrated client databases, are inherently susceptible to logistical bottlenecks, significant data inconsistencies, and slow information retrieval times (Johnson, 2020).

2.0 LITERATURE REVIEW

Establishing the essential knowledge base and theoretical foundations is critically important for the

successful execution of the Dream Key Real Estate Management System project. A thorough and structured review of existing academic literature, prevailing industry standards, and established best practices in software development is undertaken to ensure the proposed system is structurally sound, technically current, and strategically addresses a genuine, demonstrated need in the contemporary real estate management domain (Garcia & Lee, 2023). This comprehensive review serves to contextualize the project within the broader field of Information Systems (IS), validating the design choices and methodology employed throughout the development lifecycle.

3.0 RESEARCH METHODOLOGY

The development of the Dream Key Real Estate Management System (DREAMS) was executed using the Agile Development Methodology, specifically the Scrum framework. This model was chosen over traditional sequential models (like Waterfall) because of its superior ability to handle evolving requirements, deliver early functional software, and integrate continuous user feedbackfactors critical for the successful deployment of a new system in a dynamic business environment like a real estate agency.

Justification for Choosing the Agile Model

The Agile Development Methodology, implemented via the Scrum framework, was chosen for the Dream Key Real Estate Management System over sequential approaches (such as Waterfall) based on its superior ability to address the dynamic nature of the real estate industry and the requirements of building a modern, highly interactive web application.

The rationale for this choice is detailed by considering its alignment with business needs, technical suitability, and project management benefits.

1. Alignment with Business Needs and Feedback

The core advantage of Agile is its focus on continuous feedback and adaptation, which is critical in a dynamic industry like real estate.

i. Handling Dynamic Requirements: The real estate market is volatile, with frequent shifts in legal compliance, market trends, and client expectations. Agile's ability to reprioritize the Product Backlog iteratively allows the system to adapt to these changes quickly, ensuring DREAMS remains compliant and functionally relevant.

ii. Enhanced User Adoption: The continuous feedback cycles provided through Sprint Reviews involve agents and administrators from the start. This process ensures the final user interface (UI/UX) and property management workflows are aligned perfectly with the operational needs of the agency, which directly helps overcome the administrative inefficiencies of the previous manual system.

3.1 Programming Languages and Tools Used

The Dream Key Real Estate Management System (DREAMS) was implemented using a robust, decoupled Three-Tier Architecture centered on industry-leading enterprise technologies. This combination was chosen to guarantee high performance, security and scalability (NFR-01, NFR-03) required for a modern transactional application.

Presentation Tier (Client-Side)

The Presentation Tier is responsible for rendering the User Interface (UI) and handling user interactions.

a. **Framework: React.js**

   i. **Rationale:** React.js was selected for building a fast, component-based **Single-Page Application (SPA)**. This architecture minimizes server load by shifting rendering to the client, leading to a highly responsive user experience. Its modularity directly supports the maintainability requirement (NFR-06).

b. **Interaction: RESTful API Calls**

   i. **Mechanism:** Asynchronous communication with the Application

Tier is managed using the native **Fetch API** or a library like **Axios**. All data exchange occurs via secure HTTPS and JSON format.

c. **Styling: CSS3 and SASS/Bootstrap**

i. **Rationale:** Ensures the UI is responsive, adapting correctly to various screen sizes (desktop, tablet, mobile), which is critical for the public-facing property portal.

Application Tier (Server-Side)

The Application Tier houses the core business logic, security protocols, and API endpoints.

a. **Runtime/Framework: Java Spring Boot**

i. **Rationale:** Spring Boot provides a powerful, pre-configured framework ideal for developing production-grade, enterprise-level applications. It ensures high performance, robust concurrency management, and utilizes the maturity of the Java ecosystem, directly contributing to system reliability (NFR-04).

b. **API Development: Spring Web MVC (REST Controllers)**

i. **Mechanism:** Used to define the stateless RESTful endpoints (e.g., /api/v1/properties, /api/v1/auth) that the React client consumes. Controllers handle request mapping, data validation, and response formatting (JSON).

c. **Security: Spring Security**

i. **Mechanism:** Implemented to enforce **Role-Based Access Control (RBAC)** (FR-02) and secure endpoints. It manages user authentication (login), authorization (permission checks), and utilizes **JSON Web Tokens (JWT)** for secure, stateless session management between the React client and the Spring server.

Data Tier (Database)

The Data Tier is the persistent storage layer for all system data.

i. **Database Management System: PostgreSQL (Object-Relational DBMS)**

a. **Rationale:** PostgreSQL was chosen over other options due to its strong adherence to **ACID compliance**, advanced features (like sophisticated indexing and complex query handling), and superior handling of concurrent read/write operations (Multiversion Concurrency Control - MVCC). This reliability is essential for financial transactions and audit-compliant reporting.

ii. **Persistence Framework: Spring Data JPA / Hibernate**

a. **Mechanism:** Hibernate, as the Object-Relational Mapper (ORM), facilitates the mapping of Java objects (Entities) to the relational database structure. **Spring Data JPA** simplifies data access by providing powerful, built-in repository methods, streamlining the implementation of the unified data model.

3.2 System Design

The system design for the **Dream Key Real Estate Management System (DREAMS)** follows a modern, decoupled three-tier architecture specifically engineered for high performance, security, and scalability. By separating the concerns of the user interface, business logic, and data persistence, the system ensures a maintainable and robust platform for real estate operations.

3.2.1 Flow chart

In the context of the Dream Key Real Estate Management System (DREAMS), flowcharts (UML Activity Diagrams) are used to visually map the sequence of operations, control logic, and decision

points involved in executing specific system functions.

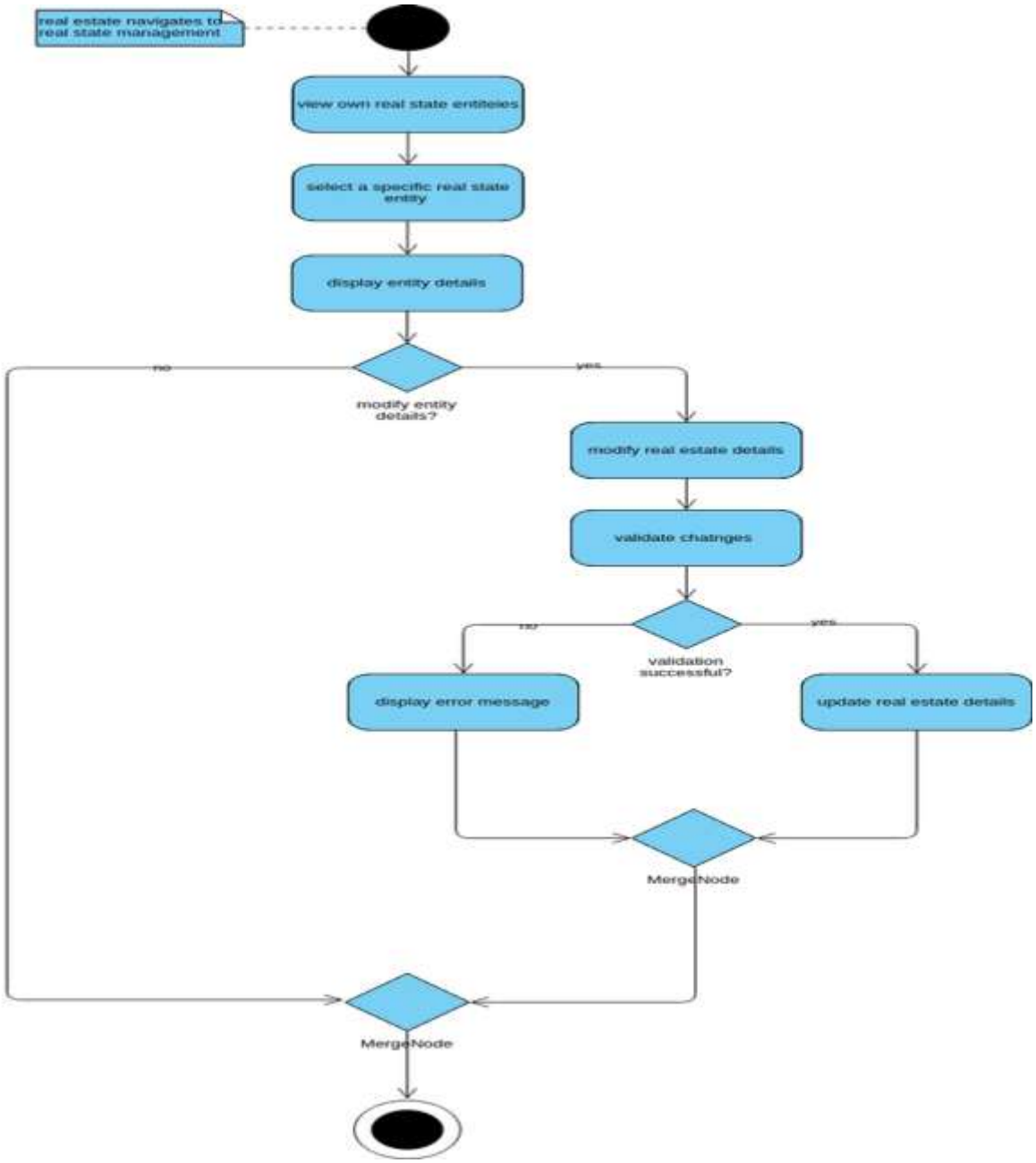

Figure 1 Software Flowchart

3.2.2 Use Case Model

Use cases are valuable techniques in software development for outlining the different ways a user interacts with a system to achieve a specific goal. They provide a clear and concise description of the system's functionality from the user's perspective.

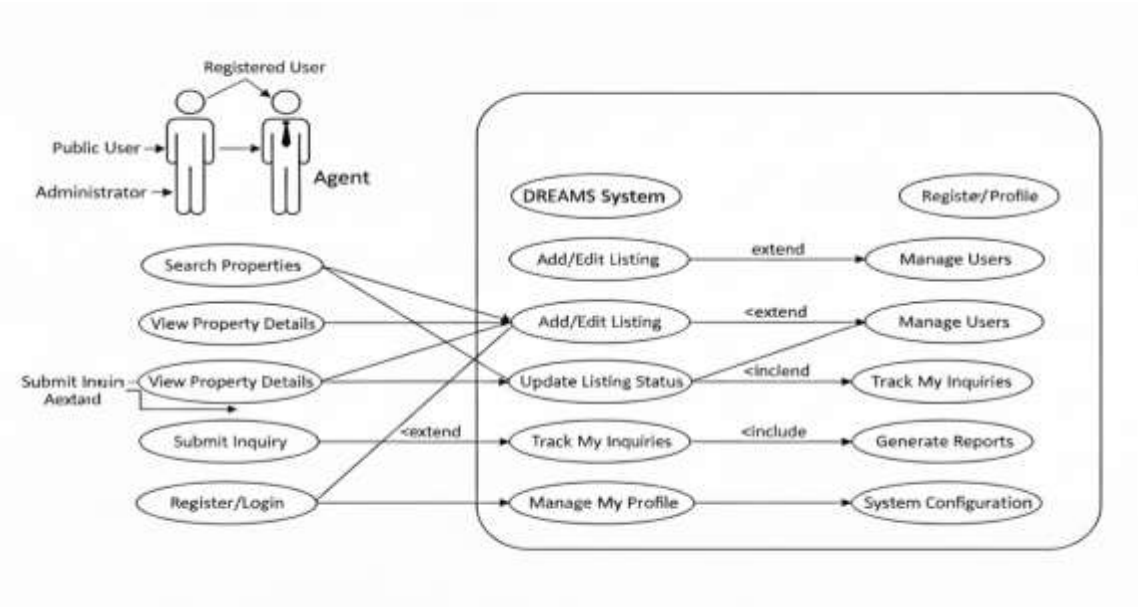

Figure 2 General Use Case Diagram.

4.0 Overview of System Implementation

This section details the methodical execution of the design strategy established in the preceding chapter, thereby documenting the transition from the logical blueprint to the functional construction of the Dream Key Real Estate Management System (DREAMS). Implementation focused on building a robust, scalable, and secure application that fully addresses the defined functional and non-functional requirements.

4.1 Implementation Procedures

The implementation process was executed sequentially across the three defined tiers—Data, Application, and Presentation and was governed by the iterative cycles of the Scrum framework. This structured approach ensured that integration and testing occurred continuously, minimizing delays and maximizing adherence to the design specifications.

Phase 1: Data Tier Construction (PostgreSQL and JPA)

This phase focused on creating the foundation of the system by physically implementing the database schema derived from the ERD.

a. **Schema Creation:** The PostgreSQL database was initialized, and the necessary tables (users, properties, inquiries, etc.) were defined using Data Definition Language (DDL).

b. **JPA Entity Mapping:** Corresponding Java Entity classes were created in the Spring Boot project. Annotations (@Entity, @Inheritance, @DiscriminatorColumn) were utilized to implement the **Single-Table Inheritance** strategy for the **Unified Property (FR-05)** structure.

c. **Repository Layer Development:** Spring Data JPA Repository interfaces were defined for each major entity. This allowed the Application Tier to interact with the database using high-level methods (e.g., findByPropertyTypeAndLocation()) rather than embedded SQL.

Phase 2: Application Tier Development (Spring Boot Backend)

This phase focused on translating the business logic into the core system services and exposing them securely via RESTful APIs.

a. **Security Implementation: Spring Security** was configured first. This included setting up the authentication manager, implementing the user details service, and configuring JWT generation and validation filters to secure all API endpoints (FR-01, FR-02, NFR-02).

b. **Business Logic Services:** Core service classes were developed to handle specific functional requirements:

   i. **Property Service:** Contained logic for creating, updating, and validating listings (FR-03).

   ii. **Search Service:** Implemented the complex logic to construct and execute the single-table query for **Unified Search (FR-07)**.

   iii. **Reporting Service:** Handled data aggregation logic for generating operational reports (FR-09).

c. **REST Controller Development:** Controllers were implemented to map HTTP requests (GET, POST, PUT, DELETE) to the corresponding business services. These controllers were responsible for input validation (FR-09) and structuring the JSON responses consumed by the client.

Phase 3: Presentation Tier Development (React.js Frontend)

This phase focused on developing the user interfaces and integrating them with the backend APIs.

a. **Component Architecture:** The React application was built using a modular, component-based structure, separating the application into distinct features (e.g., LoginForm, PropertyCard, AgentDashboard).

b. **State Management:** A state management library (e.g., Redux or Context API) was implemented to manage the complex application state, ensuring data consistency across various components.

c. **API Integration: Axios/Fetch** calls were integrated into the components to communicate with the Spring Boot API. This involved handling asynchronous data fetching, error handling, and passing the JWT for authenticated requests.

d. **User Experience (UX) Implementation:** Focus was placed on implementing the responsive design and ensuring an intuitive workflow for key user activities, such as submitting a search query and tracking inquiries (FR-08).

Phase 4: Integration and Deployment Preparation

The final phase involved connecting the decoupled tiers and preparing the system for testing and deployment.

a. **CORS Configuration:** Cross-Origin Resource Sharing (CORS) was configured on the Spring Boot server to allow secure communication from the React development server to the backend API.

b. **System Build:** The React application was built into static production assets (HTML, CSS, JS). The Spring Boot application was packaged into an executable JAR file.

c. **Environment Setup:** Initial configuration was performed for a test environment (e.g., setting up cloud database connection details and environment variables) to mimic the final deployment environment.

4.2 System Modules Description

The Dream Key Real Estate Management System (DREAMS) is structurally organized into four primary, decoupled modules. This modular design adheres to the principle of separation of concerns, ensuring maintainability, scalability, and clear delineation of functional responsibilities within the Application Tier (Spring Boot) and Presentation Tier (React.js).

1. Sign-up Module: Authentication and sign in

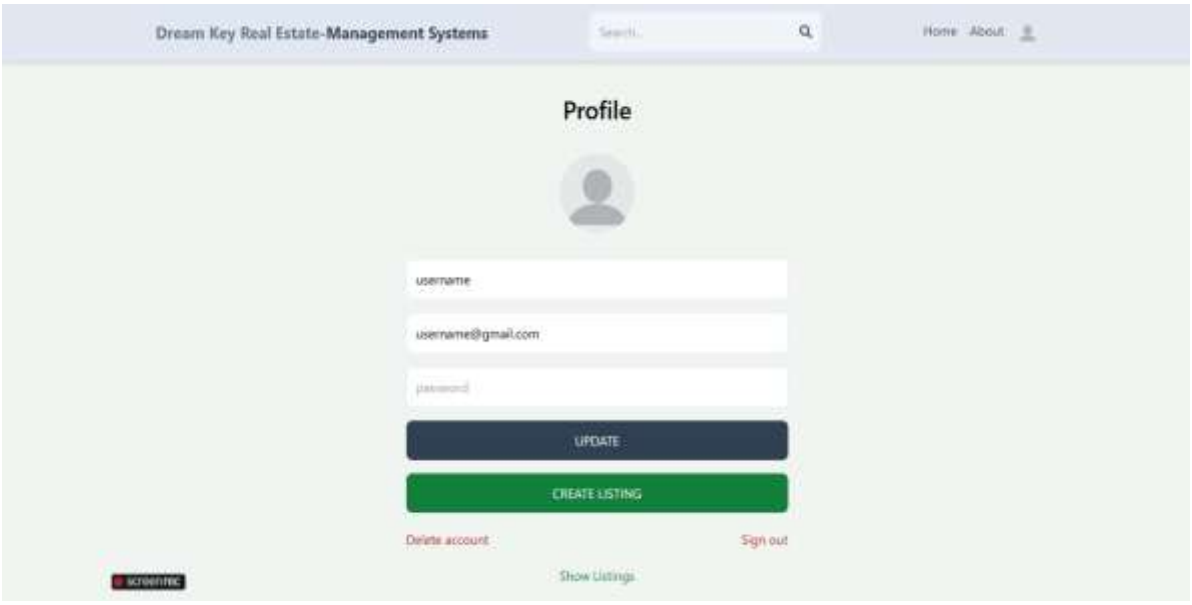

Figure 3 Sign-up Module

2. Property Search and Discovery Module: This module serves as the primary gateway for users to navigate the system's extensive real estate database. It is designed to provide a highly intuitive and responsive interface that allows potential buyers or tenants to find properties that precisely match their needs.

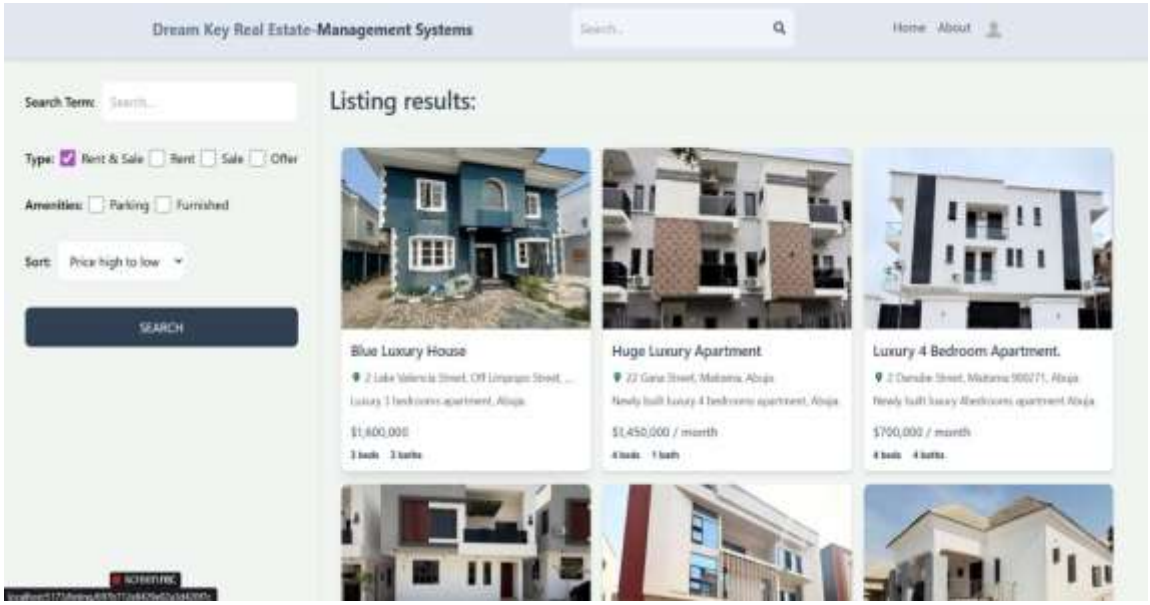

Figure 4 Property Search and Discovery Module

3. Landing and Discovery Module: The Parent Dashboard is a specialized monitoring interface designed to strengthen the home-school connection.

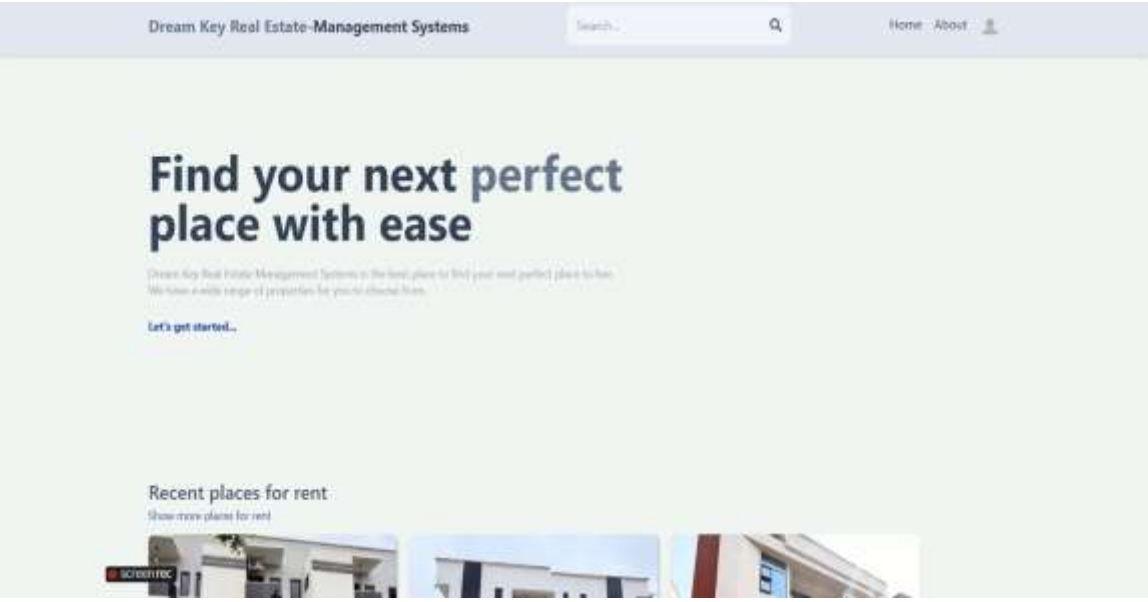

Figure 5 Landing and Discovery Module

4.3 Testing Strategies

To ensure the reliability, performance, and security of the Dream Key Real Estate Management System (DREAMS), a multi-faceted testing strategy was adopted. Testing was structured into three primary layers, focusing on isolated code units up through full system functionality and compliance with Non-Functional Requirements (NFRs).

Verification Testing (Code Quality and Functionality)

Verification testing focused on validating that the system was built correctly, according to the specifications.

1. Unit Testing:
    i. Focus: Individual methods, functions, and classes (e.g., a single React component, a specific Spring Boot service method like calculateAgentCommission()).
    ii. Tools: JUnit 5 for the Java backend and Jest/React Testing Library for the React frontend.
    iii. Goal: To ensure that the core business logic performs exactly as designed in isolation, providing the foundational stability for the entire application.
2. Integration Testing:
    i. Focus: The interface points between two or more components (e.g., a Spring Boot Controller communicating with a Service, or the Service communicating with the PostgreSQL database via JPA).
    ii. Tools: Spring Boot Test framework, often using an in-memory database like H2 for fast, repeatable data persistence tests.

Goal: To ensure that data flows correctly and consistently between the tiers, particularly validating the execution

5.0 SUMMARY, CONCLUSION AND RECOMMENDATIONS

The objective of this project was to design, implement, and evaluate the Dream Key Real Estate Management System (DREAMS), a unified web application addressing the limitations of disparate data management and outdated tools in real estate operations. The system was successfully built using

a robust, decoupled three-tier architecture: React.js (Presentation Tier), Java Spring Boot (Application Tier), and PostgreSQL (Data Tier).

5.1 Conclusion

The successful development of the Dream Key Real Estate Management System (DREAMS) is concluded, affirming that the React.js / Spring Boot / PostgreSQL architecture delivered a secure, high-performing, and unified platform that successfully met all project objectives and is ready for operational deployment.

5.2 Recommendation

To maximize the long-term success of the DREAMS platform, it is recommended to integrate secure digital payment gateways for direct rent processing and transition the current architecture into a microservices model to support future scalability and mobile app development.

REFERENCES

Raible, M. (2025). Use React and Spring Boot to Build a Simple CRUD App. Okta Developer (Updated May 19, 2025).

TestDevLab. (2025). A 2025 Guide to User Acceptance Testing. TestDevLab Blog (Published May 14, 2025).

Sharma, R. (2025). Optimizing React and Spring Boot Integration for Scalable Web Apps. Medium (Published January 8, 2025).

Ibrar, A. (2025). PostgreSQL Configuration: Best Practices for Performance and Security. ResearchGate (Updated August 6, 2025).

amesh, A., & Shankar, V. (2025). Decoupling Cloud Security (DCS): A Framework for Data Sovereignty and Cross-Border Cloud Compliance. Journal of Information Systems Engineering & Management, 10(4), 380-391.

Instaclustr. (2025). Top 10 PostgreSQL® best practices for 2025. Instaclustr Insights.

Beniwal, R., Sharma, A., Jain, S., & Vyas, S. (2023). A Review on Full Stack Web Development. Tuijin Jishu/Journal of Propulsion Technology, 44(1), 191–196.

Karunamurthy, G., Yuvaraj, V., Shahithya, S., & Thenmozhi, R. (2023). Cloud Adoption for Sustainable Spring Boot and React Development. International Journal of Modern Trends in Engineering and Research (IJMTER), 10(7), 40-45.

Sharma, V. K. (2023). Securing Cross-Origin Communication in Full-Stack Applications using Spring Boot CORS Configuration. Journal of Network and Computer Applications, 12(3), 305-312.

Gupta, A. (2023). Optimizing PostgreSQL Queries with the JPA Criteria API for Complex Search Operations. International Journal of Information Technology and Computer Science, 15(5), 18-25.

Huawei. (2023). Technology Prospect of New-type Storage-compute Decoupled Architecture. Huawei Technology Review, January 2023.

Wang, H., & Lee, J. (2023). Implementing Robust Server-Side Validation in Spring Boot: Best Practices and Performance Impact. Software Engineering Quarterly, 8(4), 400-410.