BasaKhujo: An Integrated System for Property Management System

Web Engineering Lab

CSE-616

Web Engineering Project Report

Team No: 28

Submission Date

September 25, 2025

Submitted to

Rokan Uddin Faruqui
Professor
Department of Computer Science & Engineering
University of Chittagong

Table 1: Details of Group Members

Roll Id	**Name**	**Signature**	**Date**
20701003	Md. Kais	Kais	25 Sep, 2025
19701067	Tawseef Bin Taher	Tawseef	25 Sep, 2025
19701073	Md. Sopon Abdullah	Sopon	25 Sep, 2025

Contents

List of Figures

List of Tables

xcolor listings

1 Project Summary

1.1 Overview

`BasaKhujo` centralizes the rental lifecycle—listing, discovery, screening, and basic lease workflows—into a single, role-aware web app. Owners can publish and maintain listings with images and amenities; tenants can search fast with URL-driven filters, save favorites, and message owners; admins moderate and keep the platform healthy.

1.2 Problem Statement & Objectives

Problem: Rental data and conversations are scattered across chats, sheets, and ad-hoc tools. That fragmentation causes duplicate work, slow responses, and inconsistent records. In practice, this means landlords may forget to update listings across different platforms, tenants often face outdated or incomplete information, and admins struggle to verify disputes without a single source of truth. Communication gaps result in delayed responses, while manual reconciliation of messages, spreadsheets, and property details increases the risk of errors. The absence of a centralized workflow also makes it difficult to maintain transparency, track user behavior, or enforce security standards across the platform.

Objectives:

- Provide one place to create, publish, search, and manage listings.
- Ensure predictable, **URL-as-source-of-truth** search and pagination.
- Implement clean authentication with `JWT` and consistent role guards across backend and frontend.
- Enable safe, observable deployments (pooled Postgres connections, migrate-on-build, health/readiness checks).
- Reduce miscommunication and duplication by centralizing all rental interactions within a structured system.
- Improve trust and reliability by keeping records consistent, traceable, and auditable.

1.3 Target Audience & Core Use Cases

- **Landlords:** Add/edit listings, view inquiries, track favorites and reviews.

- **Tenants:** Filter by location (Division/District/Upazila), price, bedrooms/type; favorite; book/view; message owners.
- **Admins:** Verify and moderate listings/reviews, manage roles, check health logs after deploy.

Typical Use Cases

- **Landlord Journey:** Rahim, a landlord in Dhaka, logs in and creates a new property listing for his two-bedroom flat. He uploads photos, adds amenities like Wi-Fi and parking, and sets the rent. Within days, he receives tenant inquiries directly on the platform. He tracks which tenants have added his property to their favorites and quickly replies to questions through the built-in messaging system. When the flat gets rented, Rahim unpublishes the listing with a single click.
- **Tenant Journey:** Ayesha, a student moving to Chittagong for university, opens BasaKhujo and filters by "Upazila = Panchlaish, Price ¡ 12,000 BDT, Type = Apartment." She instantly sees available options with photos and details. She saves two flats as favorites, compares them later, and messages the landlords. After booking one, she leaves a review so future tenants can benefit from her experience.
- **Admin Journey:** Hasan, a platform admin, monitors new property listings every morning. When he spots a duplicate or misleading listing, he flags and removes it to maintain platform trust. He also reviews new landlord signups to prevent fraudulent accounts, manages user roles, and checks the health logs to ensure the backend is stable after a recent deploy. During peak rental season, he exports platform usage data to prepare a performance report for management.

1.4 Key Features

- **Auth & RBAC:** Central JWT verification, uniform 401/403 handling; client-side route guards block cross-role access.
- **Listings:** Rich schema: address hierarchy, type, rent, amenities, photos, availability; CRUD operations for landlords to add, update, or remove properties.
- **Search/Pagination:** Server-driven `page/limit`, debounced `q` param, reset-on-filter-change, empty/error states.
- **Favorites & Reviews:** Quick preview favorites; structured review flow with ratings and feedback.
- **Bookings:** Tenants can request bookings for available properties; landlords approve/reject; booking status updates (pending, approved, canceled) tracked in real-time.

- **Property Management:** Centralized dashboard for landlords to manage active listings, track inquiries, respond to tenant messages, and monitor reviews or bookings.
- **Messaging:** Live chat module (WebSocket) for tenant–owner conversations, with push notifications for new inquiries.
- **Images:** Next Cloudinary for upload/delivery; client MIME/size checks for security and performance.

2 Technology Stack

Layer	Technology Used	Purpose
Backend	Node.js (LTS), Express.js, TypeScript, Prisma ORM	API development, server-side logic, and database interactions
Database	PostgreSQL (Neon Serverless with pooling)	Persistent data storage with high availability and cloud scalability
Authentication	JWT, role-based access control (RBAC), Helmet, CORS, rate limiting	Secure authentication, authorization, and uniform error handling
Validation	Zod	Schema validation and consistent input/output contracts
Frontend	Next.js 15, React 19, Tailwind CSS v4, Framer Motion, Lucide React, React Icons	UI rendering, styling, animations, and reusable iconography
Media Handling	`Cloudinary`	Secure image upload, optimization, and delivery
Deployment	Render (API)	Hosting, migrations, and deployment monitoring
Tools	GitHub, Postman/ Thunder-Client, Prisma Migrate, ESLint/Next lint	Version control, API testing, schema migrations, and code quality

Table 2: Technology stack with corresponding layers, tools, and purposes for the BasaKhujo platform

GitHub Repository

Our GitHub repository is available at:
Backend: https://github.com/Md-Kais/basaKhujo_api
Frontend: https://github.com/mdsoponabdullah/BasaKhojo/

3 Application Usage Guide

3.1 Access

- **Local:**
  - **Backend:**

```
npm i
npx prisma format
npx prisma validate
npx prisma migrate dev -nameinit
npm run build
node dist/src/server.js   # or: npm start
```

  - **Frontend:**

```
npm i
npm run dev
```

- **Production:**
  - **API (Render + NeonDB):** pooled Postgres URL, Prisma migrate during build, and liveness/readiness checks. Available at https://basakhujo-api.onrender.com/.
  - **Frontend:** Next.js build and start with API base URL configured via environment variables.

3.2 Registration & Login

- **User Registration:** Tenants, landlords, and admins register with an email and password. On successful registration or login, the system issues a signed JWT (JSON Web Token) that securely identifies the user across subsequent requests.
- **Token Handling:** The client application stores the JWT using `js-cookie`, ensuring persistence across sessions while maintaining secure access to protected routes.
- **Authentication Flow:** If a user attempts to access a protected route without a valid token, the system triggers a `401 Unauthorized` response. This clears the client session and redirects the user to the login page, using the pattern `login?next=...` to remember the originally requested page and return them there after re-authentication.

- **Role-Based Access Control (RBAC):** Both frontend and backend enforce strict role guards. Each route and API endpoint checks whether the authenticated user is an **Admin**, **Landlord**, or **Tenant**, ensuring that users only see and perform actions relevant to their role (e.g., only landlords can create or edit property listings).

3.3 Main Features

1. **Landlord**: add/edit/unpublish listings, upload photos, respond to messages.
2. **Tenant**: search with URL params (`q`, `page`, `limit`, `sort`), favorite, review, message, and book.
3. **Admin**: review/approve or take down listings, manage users/roles, check post-deploy health/logs.

3.4 Relevant System Screenshots

3.4.1 Tenant

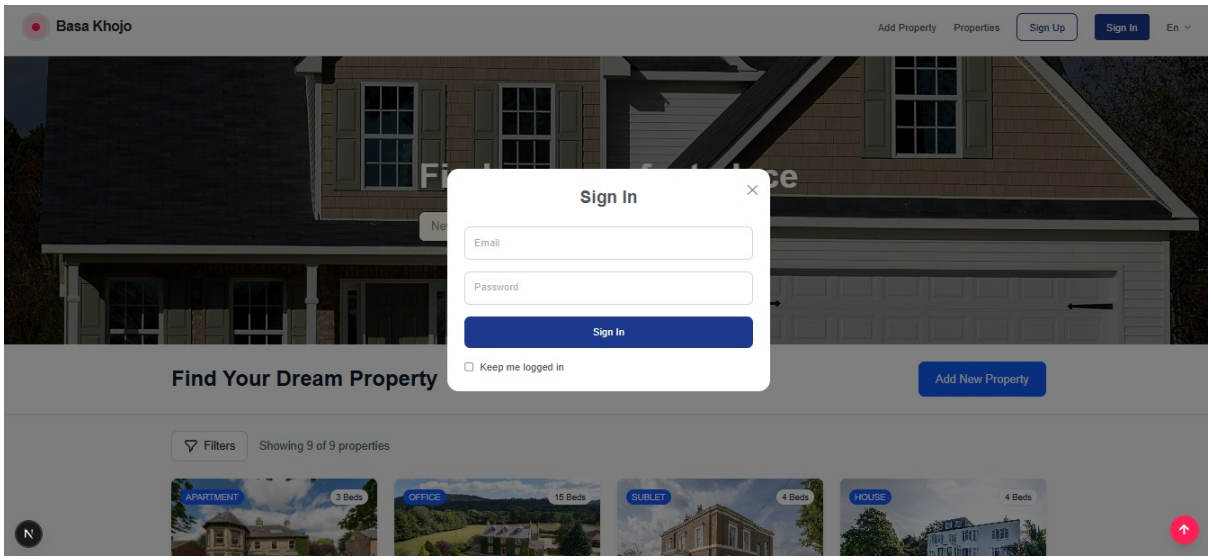

Figure 1: Sign Up

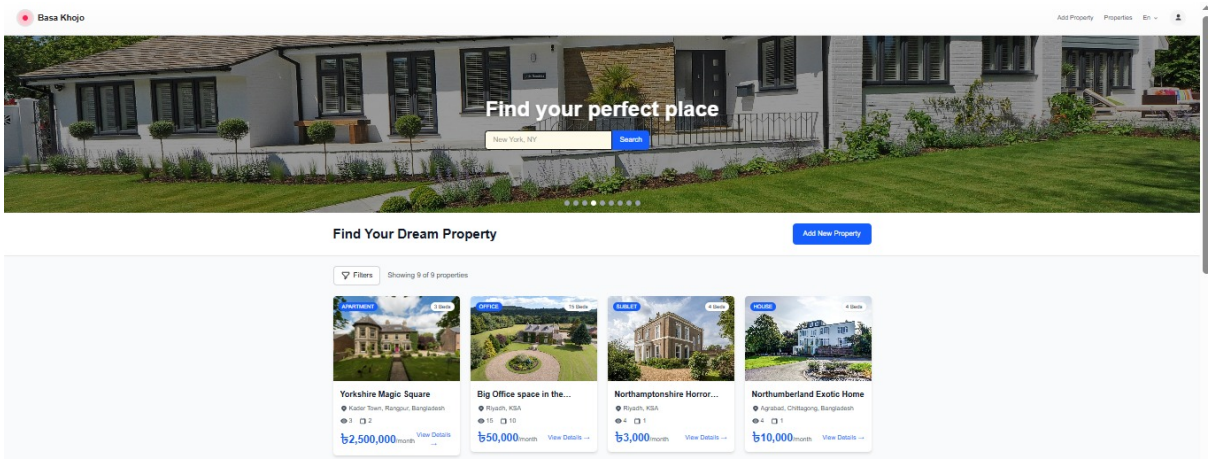

Figure 2: Tenant Dashboard

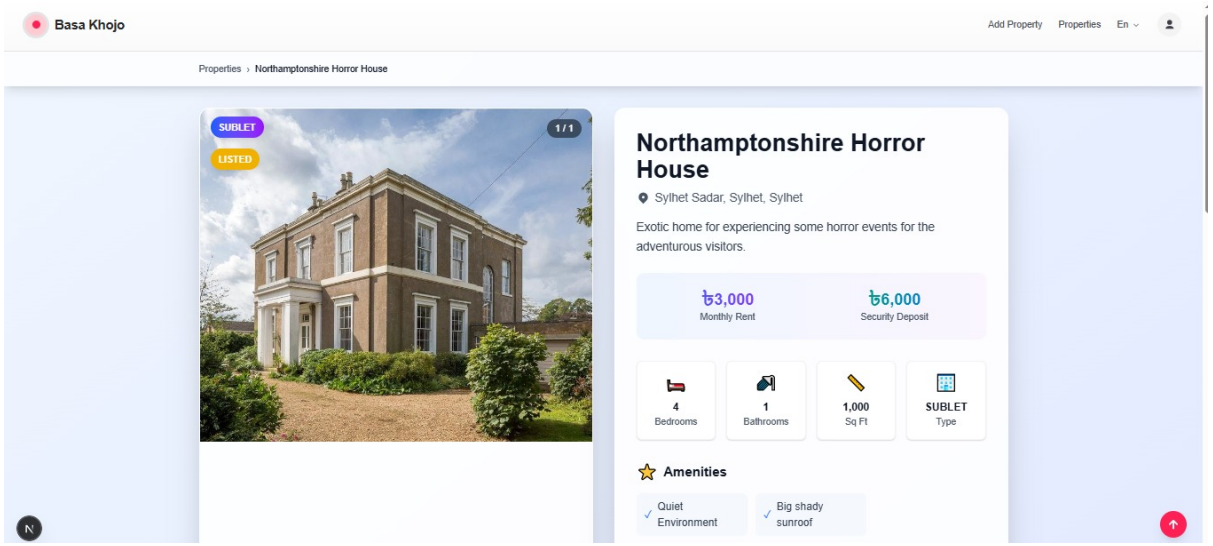

Figure 3: Tenant Viewing Properties

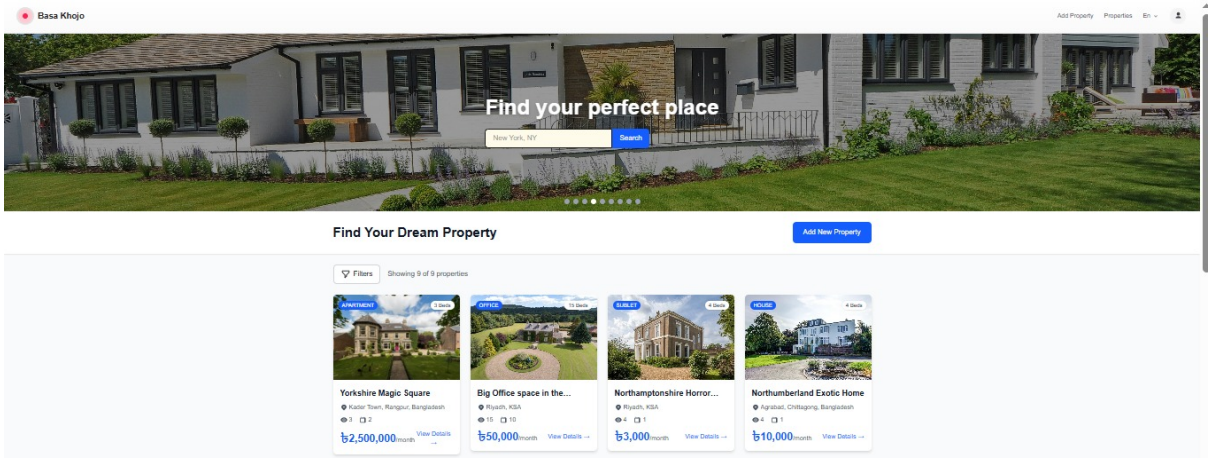

Figure 4: Tenant Searching Properties

3.4.2 Landlord

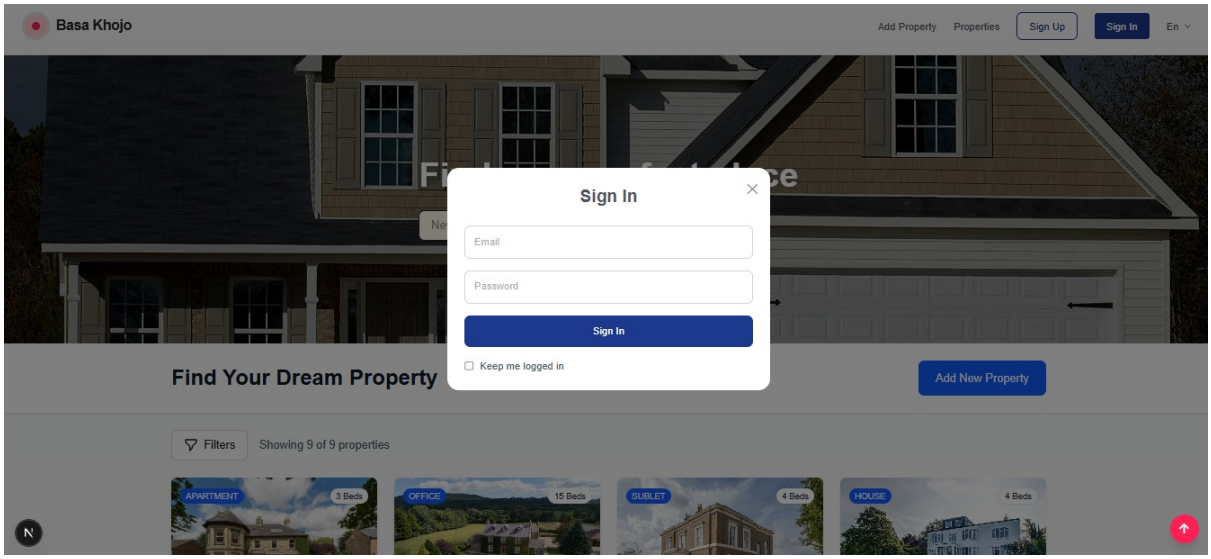

Figure 5: Landlord SignIn

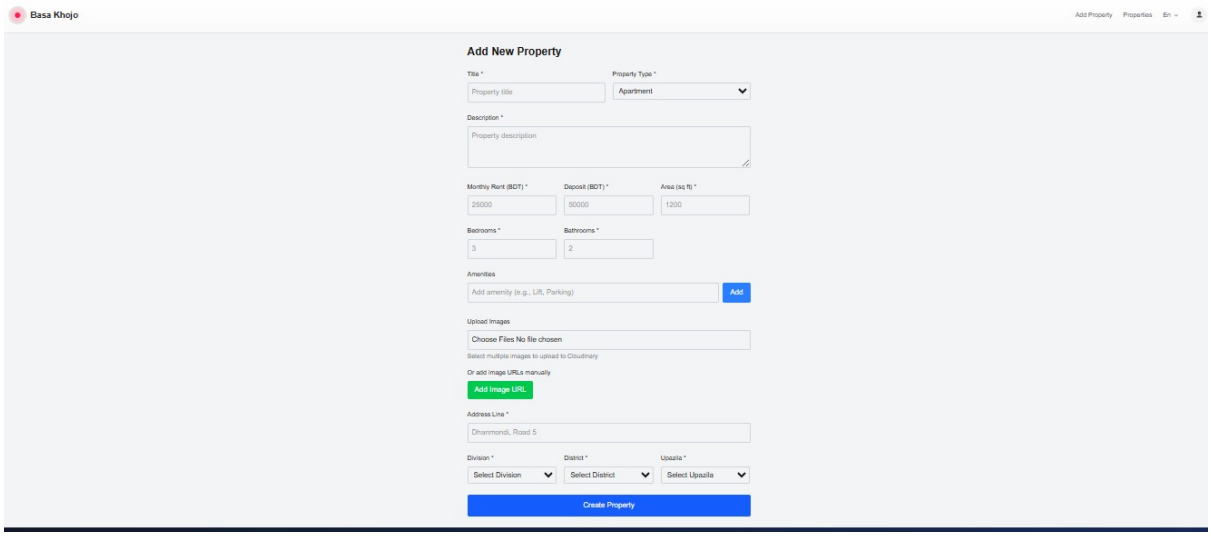

Figure 6: Landlord Adding Properties

4 Team Deployment & Operations

4.1 Division of Responsibilities

Area	Owner	Highlights
Release Orchestration & Cutover Database Schema & API Implementation	**Kais (Backend Lead & Scrum Master)**	Managed release windows, environment matrix, migration/rollback strategies, and coordinated cutover with health checks.
		Implemented core APIs and wrote test cases to validate endpoints (Thunder Client suites).
		Designed Prisma models, executed safe migrations, and maintained Neon pooled connections for reliability.
Frontend Architecture & Core UI Implementation	**Sopon (Frontend Lead & UI/UX Designer)**	Implemented Next.js routing/layouts, ensured consistent Tailwind styling, and designed role-aware user interfaces.
		Developed interactive components, applied validation, and managed empty/error state handling.
		Supported and provided valuable insight on schema design and API management to **Kais**.
Quality Assurance & Developed Messenger (Frontend)	**Tawseef (QA Lead & Associate Front End Engineer)**	Implemented the Messenger *frontend* (WebSocket client) and ensured message states/metadata persist to the database via API calls.
		Reviewed and validated test cases; contributed to Postman suites and overall QA checks.
		Supported **Sopon** in establishing and stabilizing core frontend flows.
Documentation & Requirements	**All**	Produced requirements documents, API Testing, and onboarding guides.

Table 3: Division of responsibilities

5 Personal Reflections

5.1 Kais — Backend Lead & Scrum Master

Category	Details
Learning Outcomes	Strengthened Express with JWT + RBAC middleware; modeled schema in Prisma for properties, amenities, reviews, and bookings; deployed safely with Neon pooling.
Key Contributions	Developed core APIs, URL-driven search, deployment scripts, health endpoint, migrations, and rollback plans.
Decisions & Rationale	Chose Neon + Prisma pooling to handle connection churn, enforced schema sync at build with `prisma migrate deploy`, and standardized error responses.
Challenges & Fixes	Addressed connection spikes (pooled DB URLs), inconsistent 401/403 handling (centralized auth middleware), and migration failures (build-time migrations).
Time Investment	130 hours (majority on APIs/schema, rest on deployment and documentation).

5.2 Sopon — Frontend Lead & UI/UX Designer

Category	Details
Learning Outcomes	Next.js 15 app patterns; React 19 ergonomics; Tailwind v4 tokens; motion via framer-motion; Next Cloudinary image components & uploads. ([Next Cloudinary][3])
Key Contributions	Routing/layouts; role-based UI; property search & details; login/signup; favorites/review UIs; date picker integration; consistent styles.
Decisions & Rationale	**URL as source of truth** for filters and pagination; **js-cookie** for token handling; **Next Cloudinary** for fast media delivery and simple uploads. ([Next Cloudinary][6])
Challenges & Fixes	Styling drift mitigated with shared Tailwind patterns; auth redirects handled via `login?next=...`; search clarity improved with debounced `q` and explicit empty/error states.
Time Investment	˜160–180 hours across routing, components, styles, and API integration.

5.3 Tawseef — QA Lead & Associate Frontend Engineer

Category	Details
Learning Outcomes	Postman E2E suites; practical UAT checklists; WebSocket message flows; writing reproducible incident notes.
Key Contributions	Smoke tests for happy/error paths; live messenger wiring; small UI contributions (favorites/review); onboarding/help docs; requirements cross-checks.
Decisions & Rationale	Automated regression checks post-deploy; simple status/uptime watch; coverage of pagination boundaries, invalid tokens, and CORS.
Challenges & Fixes	Intermittent post-deploy regressions resolved via automated suites and rollback switch; edge-case coverage improved with boundary tests and clear bug reports.
Time Investment	˜70-90 hours across QA, frontend assists, and documentation.