CSE471: System Analysis and Design
Project Report
Project Title: Flat/Room Rent Marketplace

Group No: 11, CSE471 Lab Section: 03, Summer 2024	
ID	**Name**
22299422	Mahir Tajwar Rahman
22299318	Asiful Islam Mahir
23341126	Riazul Hoque Bhuban
21101023	Fardin Kamran

Submission Date: 13.09.2025

Table of Contents

1. System Request

Business need:

The real estate market in Bangladesh lacks a comprehensive digital platform that efficiently connects property owners with potential renters and buyers. Traditional methods rely heavily on intermediaries, leading to increased costs, limited property visibility, and inefficient market processes. BashaLagbe addresses this gap by providing a unified platform that facilitates both rental and sale transactions while offering market analytics and administrative tools.

Business requirements:

The business requirements for BashaLagbe focus on creating a comprehensive real estate platform that supports secure user authentication and role-based access control for property owners, tenants, buyers, and administrators, enabling efficient account management and profile customization. The platform facilitates property management through robust listing creation, editing, and management capabilities that support both rental and sale transactions with comprehensive property details, media galleries, and pricing information. Users can leverage advanced search and filtering functionalities to discover properties based on location hierarchies (division, district, area), price ranges, property types, and specific amenities, while benefiting from interactive map integration for geographic property exploration.
The system incorporates a sophisticated administrative framework that enables content moderation, user management, listing verification, and comprehensive market analytics through data visualization tools and trend analysis. Property owners can showcase detailed information including high-quality images, videos, property specifications, and location details, while utilizing the platform's market intelligence features for competitive pricing and investment decisions. A robust notification system keeps users informed about listing updates, inquiry responses, and market opportunities, complemented by seamless file management capabilities supporting multiple media formats.
The platform integrates external market data through CSV processing and validation systems, enabling comprehensive market analysis, price trend visualization, and comparative property studies. Overall, the system is designed to streamline property transactions, eliminate intermediary dependencies, provide transparent market insights, and foster efficient connections between property stakeholders while maintaining data security and user privacy standards.

Business value:

The business value of BashaLagbe lies in its potential to transform Bangladesh's real estate market by addressing critical inefficiencies in property transactions and market transparency. By eliminating traditional intermediary dependencies and providing a comprehensive digital ecosystem, the platform positions itself as a catalyst for modernizing the property rental and sale landscape. Over a 10-year horizon, assuming an annual growth rate of 30% in user adoption driven by the increasing digitization of Bangladesh's real estate sector and growing smartphone penetration, BashaLagbe could capture over 2 million active users nationally, representing approximately 15% of the urban property market.

This substantial user base translates to significant monetization opportunities through multiple revenue streams including premium listing fees for landlords, featured property placements, subscription models for advanced analytics access, and commission-based transaction facilitation services, potentially achieving annual revenues exceeding $25 million by year 10. The platform's integrated market analytics system, powered by scraped data and user-generated insights, creates additional value propositions for real estate developers, financial institutions, and government agencies seeking data-driven market intelligence.

Furthermore, BashaLagbe's ability to reduce property search time by an estimated 60% and transaction costs by 40% through direct landlord-tenant connections generates substantial economic value for users. The platform's comprehensive database of property prices, trends, and market dynamics establishes it as an authoritative source for real estate valuation, positioning BashaLagbe as an indispensable tool for property investment decisions and market research, thereby securing its place as the leading digital real estate innovator in Bangladesh's emerging proptech sector.

Special issues or constraints:

BashaLagbe faces several critical issues and constraints that must be strategically addressed to ensure platform success and market viability. Scalability represents a fundamental challenge, as the platform must efficiently accommodate thousands of concurrent users, extensive property listings with high-resolution media, and complex search operations across Bangladesh's diverse geographic regions as user adoption accelerates. Data security and privacy are paramount concerns, requiring robust encryption protocols, secure file storage systems, and strict compliance with Bangladesh's Digital Security Act and international data protection standards like GDPR to safeguard sensitive user information, property details, and financial data.

System performance demands exceptional response times and uninterrupted functionality, particularly during peak property browsing hours and seasonal rental cycles when user activity intensifies significantly. The platform must implement sophisticated file management systems to handle large volumes of high-quality property images and videos while maintaining optimal loading speeds across varying internet connectivity conditions common in Bangladesh. Geographic accuracy and location intelligence pose unique challenges, requiring precise mapping integration, validation of property addresses, and seamless area-based filtering that accurately reflects Bangladesh's complex administrative divisions and local naming conventions.

Market data integrity and validation represent ongoing operational challenges, as the platform must continuously process and verify external property data sources while maintaining accuracy and preventing fraudulent listings. Additionally, the system must accommodate multilingual support for Bengali and English interfaces, integrate with local payment systems and verification services, and establish reliable customer support infrastructure to address the diverse technical literacy levels among users. Ensuring regulatory compliance with local real estate laws and building trust within Bangladesh's traditional property market culture will be essential for sustained user engagement and platform credibility.

2. Functional Requirements

- **User Registration & Login**
  Users can register and log in using email/password or third-party OAuth providers (e.g., Google).
- **Moderation & User Management**
  Admins can review reported listings, verify new submissions before publishing, and block users who violate platform rules. This ensures content quality and community safety.
- **Add/Edit Listings**
  Landlords can create new listings or update existing ones by providing essential information such as property type, location, rent amount, photos, and included amenities.
- **Search with Filters**
  Users can search for listings using advanced filters like price range, preferred location,
  number of bedrooms, furnished/unfurnished status, and availability.
- **Listing Details Page**
  Each listing page displays comprehensive details including property description, rent, utilities, facilities, landlord contact, and a photo gallery for easy evaluation.
- **Map Integration**
  Listings are displayed on an interactive map, enabling users to visually explore rental options by geographic area and proximity to landmarks.
- **Contact Landlord Feature**
  Tenants can get in touch with landlords directly through an in-app messaging system or a secure email inquiry form integrated with the listing page.
- **Wishlist**
  Users can save listings to a personal wishlist, organize them into folders, and get alerts for price changes or availability updates.
- **Available Time Slot Booking**
  Landlords can set available time slots for move-in. Tenants can select and book a preferred slot during the rental process. The system ensures no overlapping bookings and sends reminders.
- **Web Scraping**
  Admins can manually collect and upload scraped rent data from public websites like Bikroy or Bproperty.
- **Data Validation Interface**
  Admins can preview uploaded data, fix inconsistencies, and confirm entries before final submission into the system.
- **Price Trend Visualization**
  The platform displays visual graphs (bar/line charts) showing rent changes over time bybarea, helping users analyze rental trends.
- **Listing Comparison Tool**
  Users can select and compare 2–3 listings side-by-side to assess features like rent,

size, amenities, and location.

- **Notifications System**
  Users receive in-app or email notifications for actions like new messages, listing approvals, or rent changes.
- **Role-Based Access Control with Role Switching**
  The system assigns user roles such as Tenant, Landlord, and Admin. A user can hold multiple roles (e.g., both Tenant and Landlord) and easily switch between them through their profile dashboard. Based on the active role, the system adapts available features.
- **Flag/Report Listings**
  Users can report listings for being misleading, spammy, or offensive, which are then reviewed by admins.

3. Technology (Framework, Languages)

Primary Technology Stack:
Frontend Framework: React.js 18 with functional components and hooks
Backend Framework: Node.js with Express.js for RESTful API development
Database: MongoDB with Mongoose ODM for data modeling
Authentication: JSON Web Tokens (JWT) for secure user sessions
File Upload: Multer middleware for handling multipart form data
Data Processing: CSV-parser for market data processing
UI Library: Material-UI (MUI) for consistent and responsive design
Charts & Visualization: Recharts library for data visualization
Development Language: JavaScript ES6+

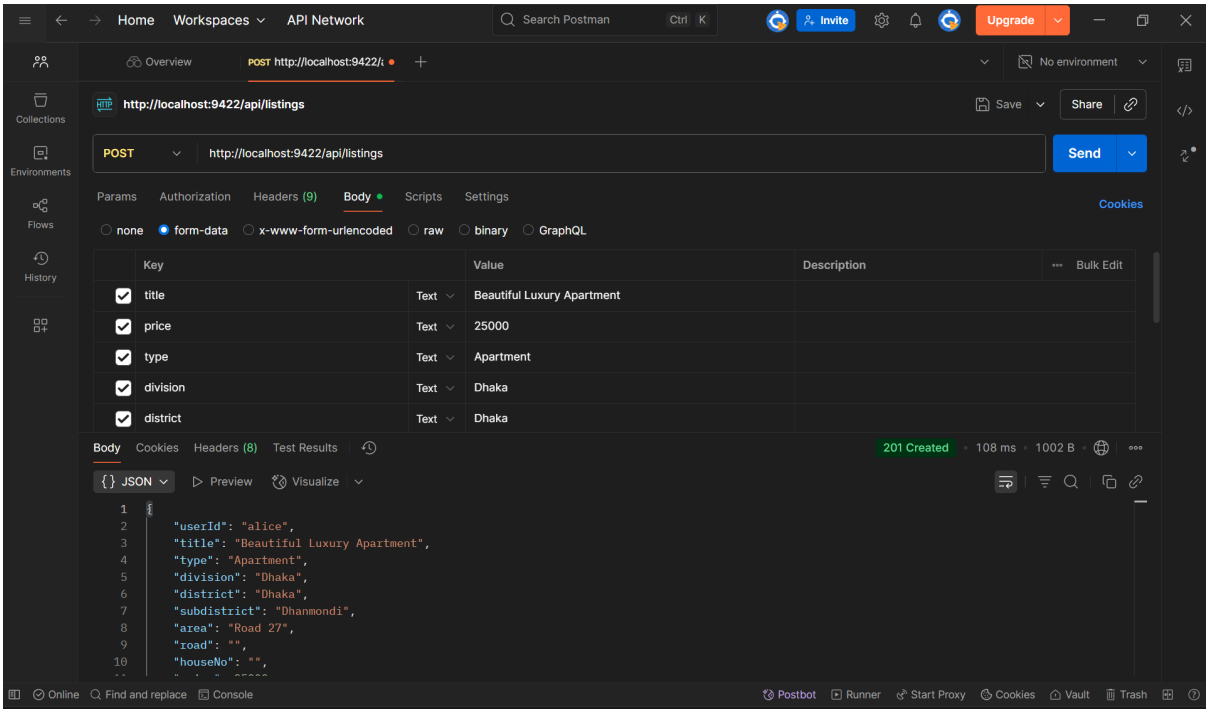
Additional Technologies:
Password Security: bcrypt for password hashing
Environment Management: dotenv for configuration management
Cross-Origin Requests: CORS middleware for API security
File System: Node.js fs module for file operations
HTTP Client: Axios for API communication

4. Backend Development

1. Create Listing:

```
// Create listing (requires userId)
router.post('/', upload.fields([{ name: 'photos', maxCount: 12 }, { name: 'video', maxCount: 1 }]), async (req, res) => {
  try {
    const userId = getUserId(req);
    if (!userId) return res.status(400).json({ error: 'userId required' });

  const base = `${req.protocol}://${req.get('host')}`;
  const photoUrls = (req.files?.photos || []).map((f) => `${base}/uploads/${f.filename}`);
  const videoUrl = req.files?.video?.[0] ? `${base}/uploads/${req.files.video[0].filename}` : '';

  // Basic required validations
  const missing = [];
  if (!req.body.title || !req.body.title.trim()) missing.push('title');
  if (req.body.price === undefined || req.body.price === '') missing.push('price');
  if (!req.body.type) missing.push('type');
  if (!req.body.division) missing.push('division');
  if (!req.body.district) missing.push('district');
  if (!req.body.subdistrict) missing.push('subdistrict');
  if (!req.body.area) missing.push('area');
  if (!req.body.phone || !req.body.phone.trim()) missing.push('phone');
  const floorNum = req.body.floor !== undefined ? Number(req.body.floor) : undefined;
  if (floorNum === undefined || Number.isNaN(floorNum) || floorNum < 0) missing.push('floor');
  if (!req.body.availableFrom) missing.push('availableFrom');
  const roomsNum = req.body.rooms !== undefined ? Number(req.body.rooms) : undefined;
  if (roomsNum === undefined || Number.isNaN(roomsNum) || roomsNum < 0) missing.push('rooms');
  if (photoUrls.length === 0) missing.push('photos');
  if (missing.length) return res.status(400).json({ error: 'Missing required fields', fields: missing });

  const payload = {
      ...req.body,
      userId,
```

```
      price: (() => { const n = Number(req.body.price); return Number.isFinite(n) && n > 0 ? n : 0; })(),
  rooms: Number(req.body.rooms),
      bathrooms: req.body.bathrooms ? Number(req.body.bathrooms) : 0,
      balcony: req.body.balcony ? Number(req.body.balcony) : 0,
      personCount: req.body.personCount ? Number(req.body.personCount) : 1,
      isRented: req.body.isRented === 'true' || req.body.isRented === true,
      features: req.body.features
        ? req.body.features.split(',').map((s) => s.trim()).filter(Boolean)
        : [],
  floor: Number(req.body.floor),
      totalFloors: req.body.totalFloors ? Number(req.body.totalFloors) : 0,
      furnishing: req.body.furnishing || 'Unfurnished',
      deposit: req.body.deposit ? Math.max(0, Number(req.body.deposit)) : 0,
      serviceCharge: req.body.serviceCharge ? Math.max(0, Number(req.body.serviceCharge)) : 0,
      negotiable: req.body.negotiable === 'true' || req.body.negotiable === true,
      utilitiesIncluded: req.body.utilitiesIncluded
        ? req.body.utilitiesIncluded.split(',').map((s) => s.trim()).filter(Boolean)
        : [],
      contactName: req.body.contactName || '',
      phone: req.body.phone || '',
  availableFrom: new Date(req.body.availableFrom),
    photoUrls,
  videoUrl,
    sizeSqft: req.body.sizeSqft ? Number(req.body.sizeSqft) : 0,
    };
    const listing = new Listing(payload);
    const saved = await listing.save();
    res.status(201).json(saved);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```



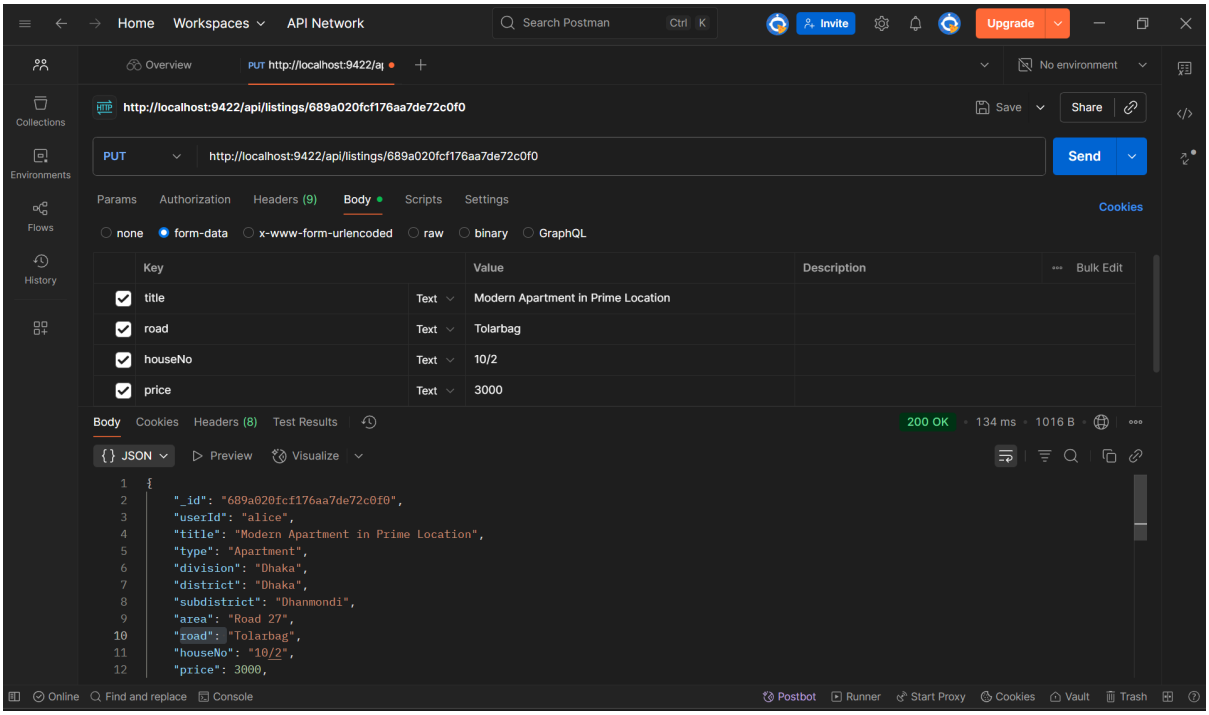
The create listing backend API is a comprehensive POST endpoint (/api/listings/) that handles authenticated property listing creation with extensive validation and file processing capabilities. The API begins by extracting the user ID from the authenticated request and performs rigorous validation checks for required fields including title, price, type, division, district, subdistrict, area, phone number, availability date, and room count, returning a 400 error with specific missing field details if any required data is absent. The system processes uploaded files through Multer middleware, handling up to 12 photos and 1 video file by generating secure file URLs with base path concatenation, while implementing robust data processing that includes type conversions (string to number for price, rooms, bathrooms, floor numbers), date parsing for availability, boolean conversion for rental status and negotiability, and array parsing for comma-separated features and utilities. The API constructs a comprehensive payload object containing all sanitized property data, pricing information (including deposit and service charges), location details, amenities, and file URLs before creating a new Listing document, saving it to MongoDB, and returning a 201 status with the complete saved listing data, while implementing comprehensive error handling that catches validation errors, database errors, and file processing issues to return appropriate 400 or 500 status responses with descriptive error messages.

2. Update Listing:

```
// Update listing (only by owner)
router.put('/:id', upload.fields([{ name: 'photos', maxCount: 12 }, { name: 'video', maxCount: 1 }]), async (req, res) => {
  try {
    const userId = getUserId(req);
    if (!userId) return res.status(400).json({ error: 'userId required' });

    const doc = await Listing.findOne({ _id: req.params.id, userId });
    if (!doc) return res.status(404).json({ error: 'Not found or not owner' });

    const base = `${req.protocol}://${req.get('host')}`;
    const newPhotoUrls = (req.files?.photos || []).map((f) => `${base}/uploads/${f.filename}`);
    const keep = req.body.existingPhotoUrls ? JSON.parse(req.body.existingPhotoUrls) : doc.photoUrls;

  // Track originals to know what to delete later
  const originalPhotoUrls = Array.isArray(doc.photoUrls) ? [...doc.photoUrls] : [];
  const originalVideoUrl = doc.videoUrl || '';

    doc.title = req.body.title ?? doc.title;
    doc.description = req.body.description ?? doc.description;
    doc.type = req.body.type ?? doc.type;
    if (typeof req.body.price !== 'undefined') {
      const n = Number(req.body.price);
      doc.price = Number.isFinite(n) && n > 0 ? n : 0;
    }
  if (req.body.availableFrom) doc.availableFrom = new Date(req.body.availableFrom);
  if (typeof req.body.rooms !== 'undefined') doc.rooms = Number(req.body.rooms);
    if (req.body.bathrooms) doc.bathrooms = Number(req.body.bathrooms);
    if (req.body.balcony) doc.balcony = Number(req.body.balcony);
    if (req.body.personCount) doc.personCount = Number(req.body.personCount);
  if (typeof req.body.isRented !== 'undefined') doc.isRented = req.body.isRented === 'true' || req.body.isRented === true;
    if (req.body.features) doc.features = req.body.features.split(',').map((s) => s.trim()).filter(Boolean);
```

```
  // location removed
  // Structured address updates
  if (typeof req.body.division !== 'undefined') doc.division = req.body.division;
  if (typeof req.body.district !== 'undefined') doc.district = req.body.district;
  if (typeof req.body.subdistrict !== 'undefined') doc.subdistrict = req.body.subdistrict;
  if (typeof req.body.area !== 'undefined') doc.area = req.body.area;
  if (typeof req.body.road !== 'undefined') doc.road = req.body.road;
  if (typeof req.body.houseNo !== 'undefined') doc.houseNo = req.body.houseNo;
  if (typeof req.body.floor !== 'undefined') doc.floor = Number(req.body.floor);
  if (typeof req.body.totalFloors !== 'undefined') doc.totalFloors = Number(req.body.totalFloors) || 0;
  if (typeof req.body.furnishing !== 'undefined') doc.furnishing = req.body.furnishing;
  if (typeof req.body.deposit !== 'undefined') doc.deposit = Math.max(0, Number(req.body.deposit)) || 0;
  if (typeof req.body.serviceCharge !== 'undefined') doc.serviceCharge = Math.max(0, Number(req.body.serviceCharge)) || 0;
  if (typeof req.body.negotiable !== 'undefined') doc.negotiable = req.body.negotiable === 'true' || req.body.negotiable === true;
  if (typeof req.body.utilitiesIncluded !== 'undefined') doc.utilitiesIncluded = req.body.utilitiesIncluded.split(',').map((s) => s.trim()).filter(Boolean);
  if (typeof req.body.contactName !== 'undefined') doc.contactName = req.body.contactName;
  if (typeof req.body.phone !== 'undefined') doc.phone = req.body.phone;
  if (typeof req.body.sizeSqft !== 'undefined') doc.sizeSqft = Number(req.body.sizeSqft) || 0;

    doc.photoUrls = [...keep, ...newPhotoUrls];
    let removedVideoUrl = '';
    if (req.files?.video?.[0]) {
      // Replace with newly uploaded video
      removedVideoUrl = originalVideoUrl;
      doc.videoUrl = `${base}/uploads/${req.files.video[0].filename}`;
    } else if (req.body.removeVideo === 'true') {
      // Explicitly remove existing video
      removedVideoUrl = originalVideoUrl;
      doc.videoUrl = '';
    } else if (req.body.existingVideoUrl) {
      // Keep existing video when the client confirms it should remain
      doc.videoUrl = req.body.existingVideoUrl;
    }

    // Validate required fields after applying changes
    const must = {
      title: doc.title,
```

```
      price: doc.price,
      type: doc.type,
      floor: doc.floor,
      rooms: doc.rooms,
      availableFrom: doc.availableFrom,
      division: doc.division,
      district: doc.district,
      subdistrict: doc.subdistrict,
      area: doc.area,
      phone: doc.phone,
    };
    const missing = Object.entries(must).filter(([k, v]) => {
      if (k === 'floor') return !(typeof v === 'number') || v < 0;
      if (k === 'rooms') return !(typeof v === 'number') || v < 0;
      return v === undefined || v === null || (typeof v === 'string' && v.trim() === '');
    }).map(([k]) => k);
    if (!doc.photoUrls.length) missing.push('photos');
    if (missing.length) return res.status(400).json({ error: 'Missing required fields', fields: missing });

    const updated = await doc.save();

    // After successful save, delete any removed photos/videos from disk (best effort)
    const removedPhotos = originalPhotoUrls.filter((u) => !doc.photoUrls.includes(u));
    const pathsToDelete = [
      ...removedPhotos.map(urlToUploadPath),
      urlToUploadPath(removedVideoUrl),
    ].filter(Boolean);
    // Fire and forget
    Promise.all(pathsToDelete.map(unlinkSafe)).catch(() => {});

    res.json(updated);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```



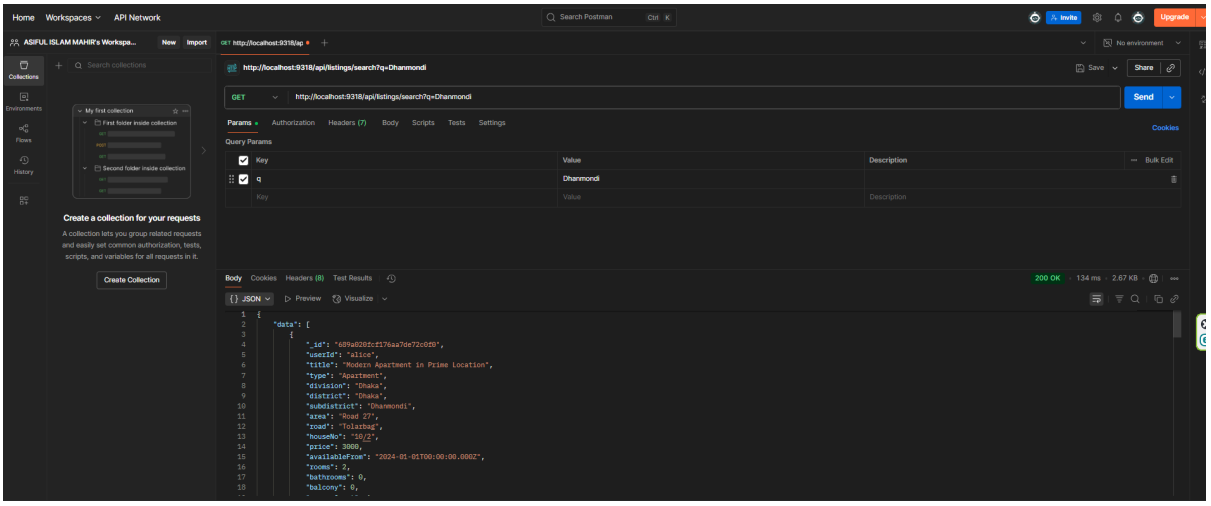
The update listing backend API is a sophisticated PUT endpoint (/api/listings/:id) that enables authenticated users to modify existing property listings with comprehensive validation, file management, and ownership verification. The API begins by extracting the user ID from the

JWT token and performing ownership validation by finding the listing document and ensuring only the original creator can modify it, returning a 404 error if the listing doesn't exist or the user lacks ownership permissions. The system handles complex file management operations including processing new photo uploads (up to 12 files) and video uploads (1 file) through Multer middleware, while maintaining existing media files and tracking removed files for cleanup operations through comparison of original and updated photo URLs.
The API performs selective field updates using conditional assignment operations, updating only the fields provided in the request body including title, description, type, price (with number validation and finite checks), availability date, location details (division, district, subdistrict, area), contact information, property specifications (rooms, bathrooms, balcony, person count), rental status, and amenities arrays processed from comma-separated strings. The system implements robust data validation by checking for missing required fields after updates and ensuring data integrity through type conversions and sanitization, while handling video file replacement logic that removes existing videos when new ones are uploaded or when explicitly requested.
After successfully updating the document and saving changes to MongoDB, the API performs cleanup operations by identifying and deleting orphaned photo and video files from the disk storage using Promise.all for efficient parallel file deletion, then returns the updated listing document as a JSON response with a 200 status code, while implementing comprehensive error handling that catches validation errors, file system errors, and database operations to return appropriate 500 status responses with descriptive error messages.

3. searching with filters

```
router.get('/search', async (req, res) => {
  try {
    const {
      q,
      type,
      division,
      district,
      subdistrict,
      area,
      priceMin,
      priceMax,
      roomsMin,
      roomsMax,
  bathroomsMin,
  bathroomsMax,
  personMin,
  personMax,
  balconyMin,
  balconyMax,
  serviceChargeMin,
  serviceChargeMax,
      page = '1',
      limit = '20',
      sort = 'newest',
```

```
    page = '1',
    limit = '20',
    sort = 'newest',
    isRented,
  } = req.query;

  const filter = {};

  // Keyword (regex OR). For larger scale consider a text index and $text.
  if (q && q.trim()) {
    const kw = q.trim();
    filter.$or = [
      { title: { $regex: kw, $options: 'i' } },
      { description: { $regex: kw, $options: 'i' } },
      { area: { $regex: kw, $options: 'i' } },
      { subdistrict: { $regex: kw, $options: 'i' } },
      { district: { $regex: kw, $options: 'i' } },
      { division: { $regex: kw, $options: 'i' } },
    ];
  }
```



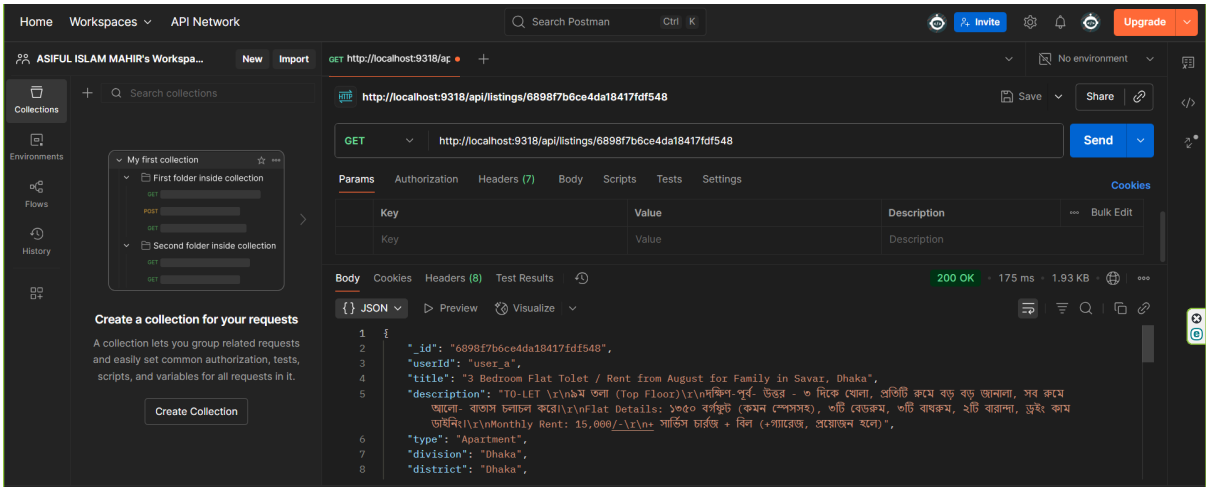
This backend API endpoint implements a comprehensive property search system that accepts multiple query parameters and constructs complex MongoDB filter objects for advanced property discovery. The endpoint begins by extracting pagination parameters (page, limit, sort) and search criteria from the request query including location filters (division, district, subdistrict, area), price ranges (priceMin, priceMax), room specifications (roomsMin, roomsMax), and additional property features (bathrooms, person capacity, balcony, service charges).

The search implementation demonstrates sophisticated query building through conditional parameter processing, where each filter is only applied if the corresponding query parameter exists and is valid. The code constructs a MongoDB filter object starting with a base filter for non-rented properties, then progressively adds search criteria including text-based keyword search using MongoDB's $regex operator with case-insensitive matching across multiple fields (title, description, area, subdistrict, district, division), location-based filtering through

exact matches for administrative divisions and regex patterns for flexible area matching, and numerical range filtering for price, room counts, and other property specifications.
The endpoint showcases modern backend development patterns including parameter destructuring from request query objects, conditional filter construction to avoid empty or invalid criteria, MongoDB aggregation techniques for complex search operations, and comprehensive error handling with appropriate HTTP status codes. This implementation enables users to perform granular property searches with multiple simultaneous filters, supporting both exact matches for structured data and fuzzy matching for text-based searches, providing the foundation for the advanced search functionality demonstrated in the frontend Browse component.

4. **Search Listing by ID:**

```
router.get('/:id', async (req, res) => {
  try {
    const doc = await Listing.findById(req.params.id);
    if (!doc) return res.status(404).json({ error: 'Not found' });
    res.json(doc);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

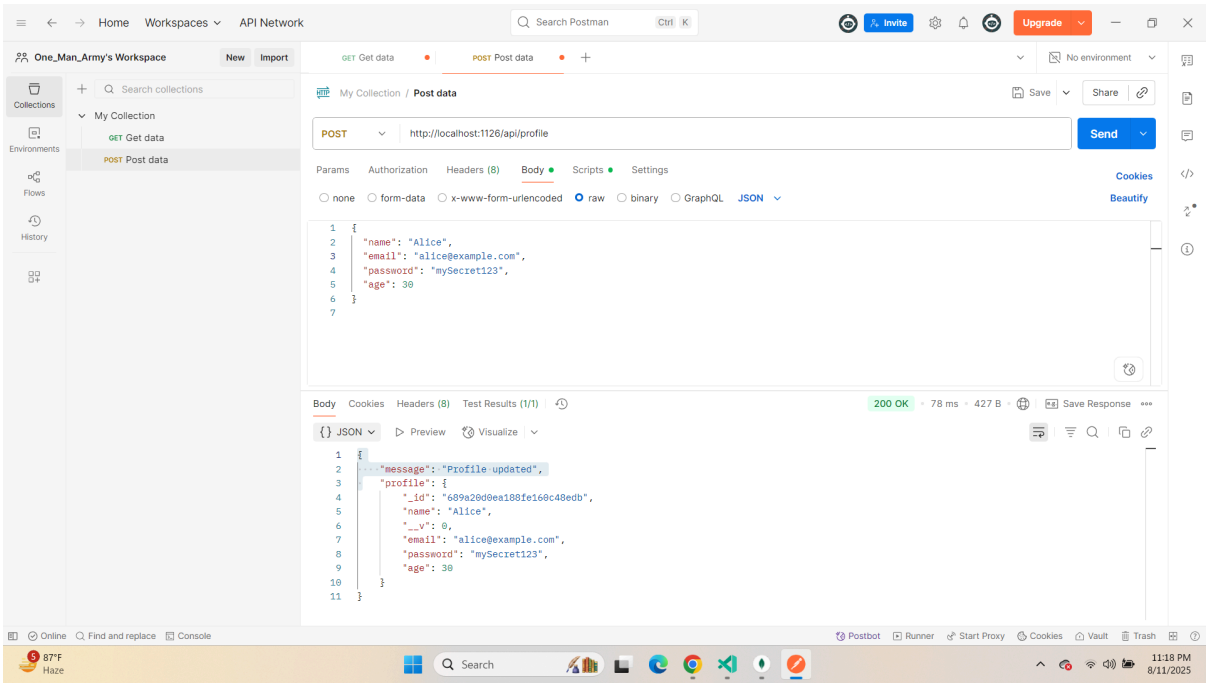


This is a standard "findById" or "getListingById" endpoint that takes a listing ID as a URL parameter (req.params.id), queries the database using Listing.findById() to locate the specific property listing, and returns the complete listing data as JSON. If the listing doesn't exist, it responds with a 404 "Not found" error, and any database or server errors are caught and

returned as a 500 status with the error message. This endpoint is essential for displaying detailed property information on individual listing pages, like the ListingDetails.js component that consumes this API to show comprehensive property data including photos, pricing, location, and contact details.

5. Create Or Update Profile:

```
async function postProfile23341126(req, res) {
  try {
    const { name, email, password, age } = req.body;

    if (!name || !email || !password || age === undefined) {
      return res.status(400).json({ message: 'Name, email, password, and age are required' });
    }

    let profile = await Profile.findOne();
    if (profile) {
      profile.name = name;
      profile.email = email;
      profile.password = password;  // Reminder: hash passwords in real apps
      profile.age = age;
      await profile.save();
      return res.json({ message: 'Profile updated', profile });
    }

    profile = new Profile({ name, email, password, age });
    await profile.save();
    res.json({ message: 'Profile created', profile });

  } catch (err) {
    res.status(500).json({ error: err.message });
  }
}
```



The postProfile23411126 function is an Express.js API endpoint that handles user profile creation and updates in the BashaLagbe platform. It extracts user data (name, email, password, age) from the request body, validates required fields, and either updates an existing profile or creates a new one using the Profile model. The function includes basic error handling with 400/500 status codes and returns appropriate success messages ("Profile updated" or "Profile created") along with the profile data, though it currently stores passwords in plain text with a reminder comment to implement proper password hashing for production security.

5. User Interface Design

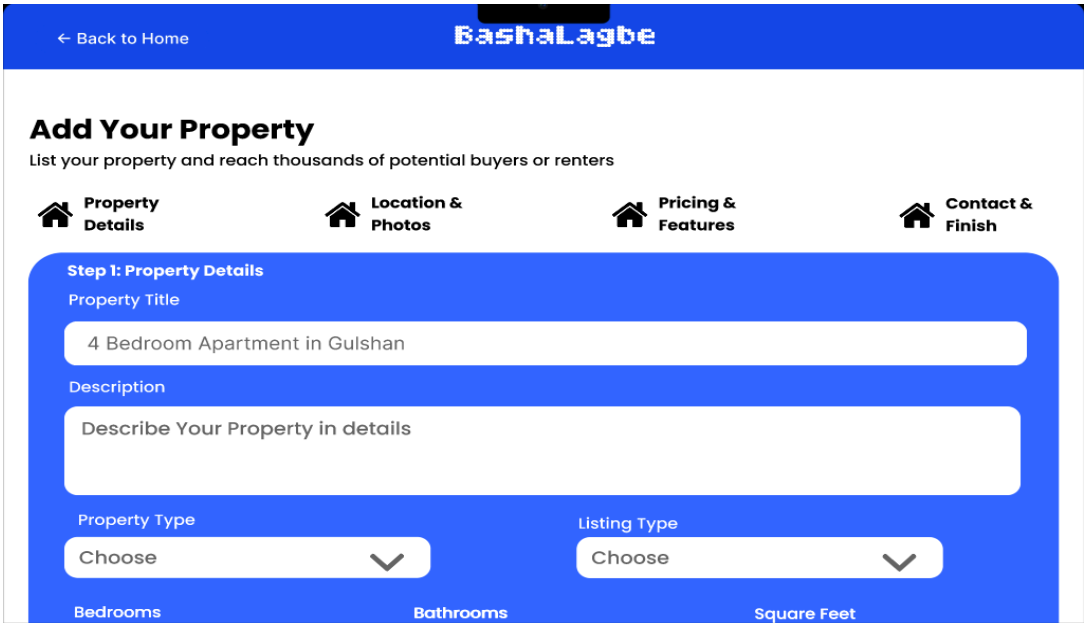
https://www.figma.com/design/66VnZi1tbYJc9jHy9epj56/CSE471-Lab-Assignment-02---22299422---Mahir-Tajwar-Rahman?node-id=0-1&t=qzTfinJg1MK98Ex2-1

https://www.figma.com/design/ZUerpzYl53jCla8RjbvNtm/Untitled?node-id=0-1&t=YjwF2102Xuk4TTiy-1

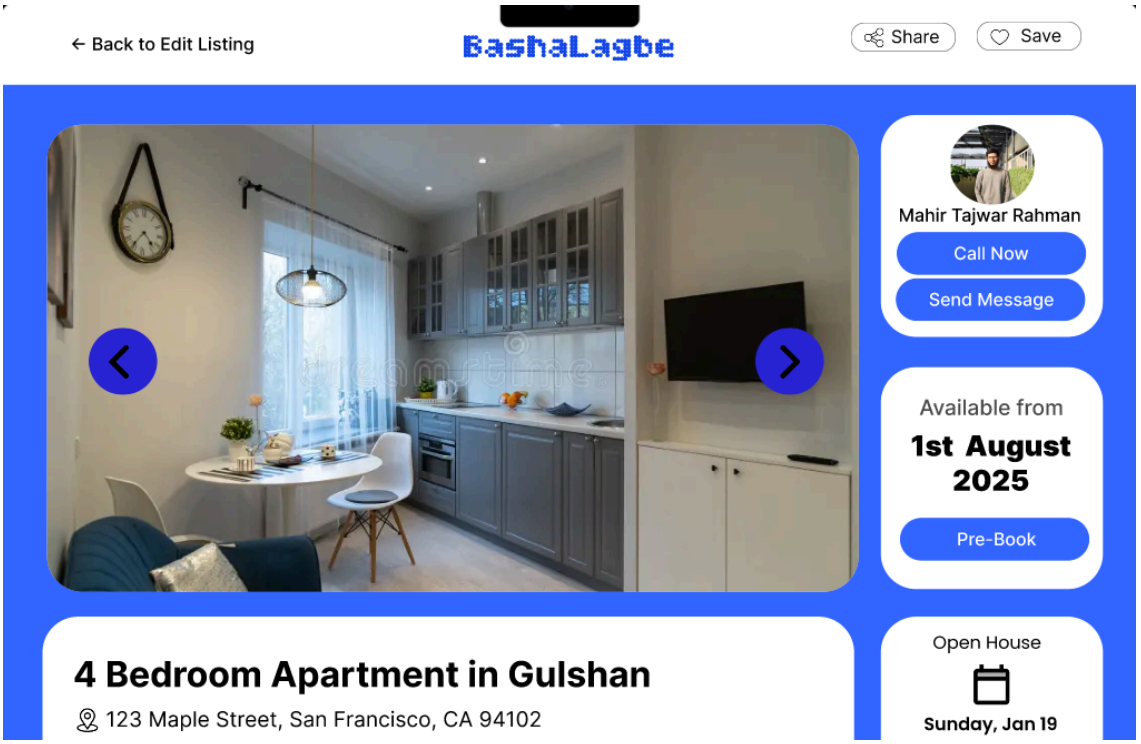
https://www.figma.com/design/noyrovl2gdtrPfzAR3aagY/Untitled?node-id=0-1&p=f&t=AKwPdG8gdosFv2ab-0

https://www.figma.com/design/YQcvIu9fDtXqjeJ3rejVpR/Dashboard-Design1?node-id=0-1&p=f&t=udvkyiZMFpYzVd5M-0

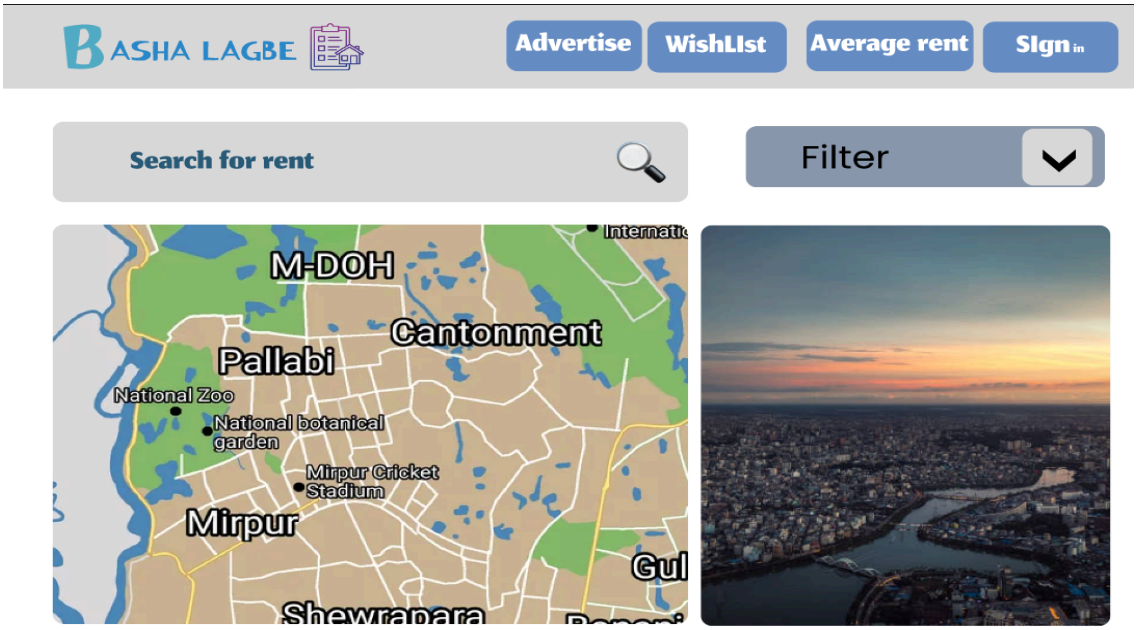
1. Add / Update Property Listing:



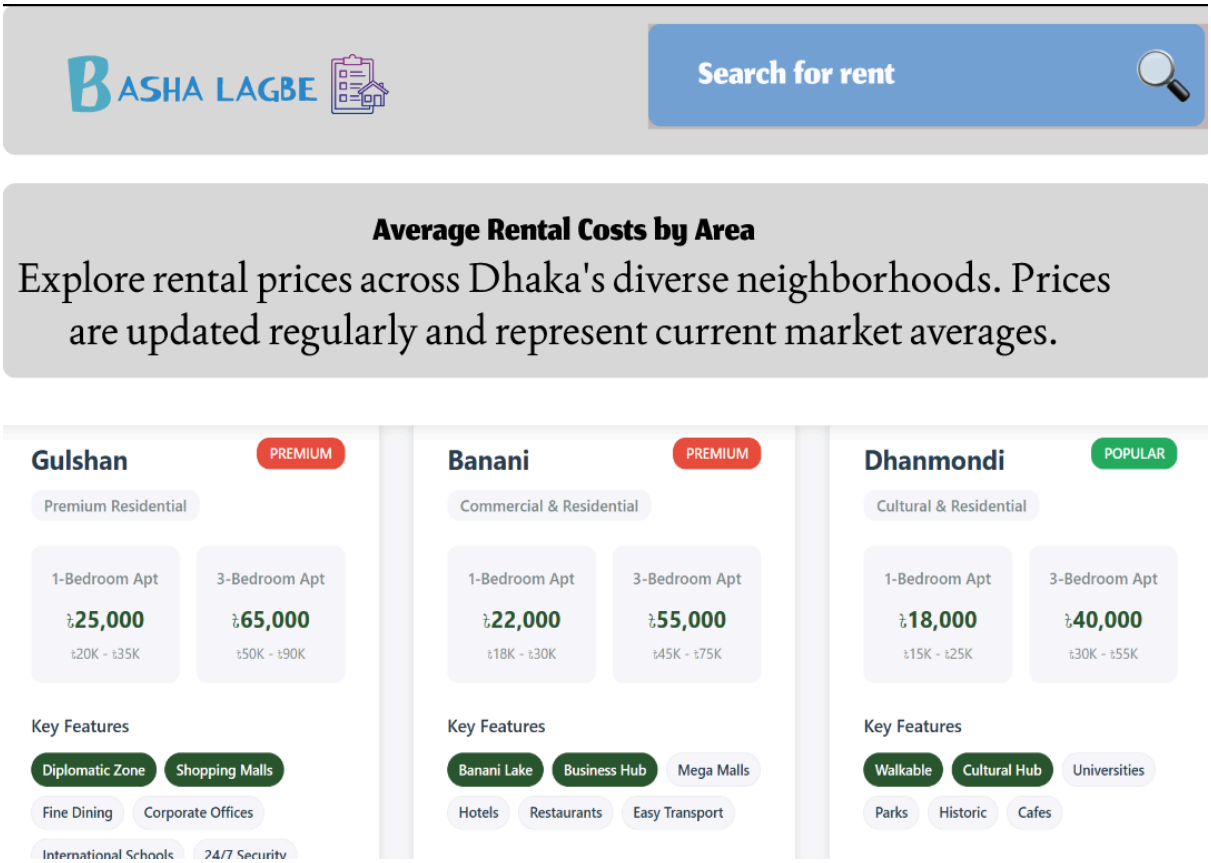
2. **Listing Details:**



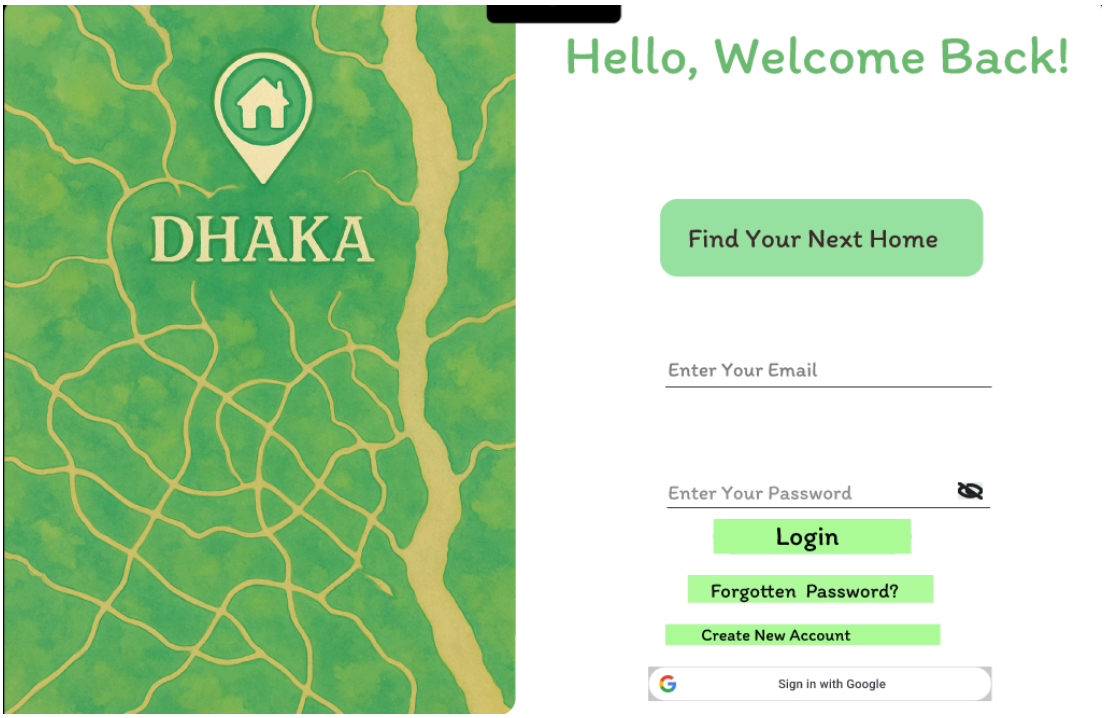
3. **Searching Property:**



4. **Price Trend Visualization:**



5. **Login page:**



6. Frontend Development

1. Property Listing Component:

```jsx
import React, { useEffect, useMemo, useState } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import axios from 'axios';
import { useAuth } from '../auth';
import { getDivisions, getDistricts, getUpazilas } from '../data/bd-geo';
import { Box, Paper, Grid, TextField, Select, MenuItem, FormControl, InputLabel, Button, Typography, Checkbox, FormControlLabel,
import MapPicker from '../components/MapPicker';

const empty = {
  title: '',
  description: '',
  type: 'Apartment',
  listingType: 'rent', // new field for rent vs sale
  price: '',
  availableFrom: '',
  rooms: '',
  bathrooms: '',
  balcony: '',
  personCount: '',
  features: '', // comma separated in UI
  isRented: false,
  lat: '',
  lng: '',
};

export default function AddEditListing() {
  const { id } = useParams();
  const navigate = useNavigate();
  const { user } = useAuth();
  const [form, setForm] = useState(empty);
  const [loading, setLoading] = useState(false);
  const [existingPhotos, setExistingPhotos] = useState([]); // string URLs
  const [keepPhoto, setKeepPhoto] = useState({}); // url -> bool
```

```jsx
  const [newPhotos, setNewPhotos] = useState([]); // File[]
  const [videoUrl, setVideoUrl] = useState(''); // existing video url
  const [newVideo, setNewVideo] = useState(null); // File | null
  const newVideoPreview = useMemo(() => (newVideo ? URL.createObjectURL(newVideo) : ''), [newVideo]);
  useEffect(() => {
    return () => {
      // revoke object URL when file changes/unmounts
      if (newVideoPreview) URL.revokeObjectURL(newVideoPreview);
    };
  }, [newVideoPreview]);
  const [dragActive, setDragActive] = useState(false);
  const [removeVideo, setRemoveVideo] = useState(false);
  const maxPhotos = 12;
  const [tab, setTab] = useState(0);
  const [showPicker, setShowPicker] = useState(false);

  const keepCount = useMemo(
    () => existingPhotos.filter((u) => keepPhoto[u]).length,
    [existingPhotos, keepPhoto]
  );
  const remainingSlots = Math.max(0, maxPhotos - keepCount - newPhotos.length);

```

The Property Listing Component is a comprehensive form interface that enables users to create and edit property listings with extensive customization options. This component implements a sophisticated multi-state form system that handles both rental and sale property types through conditional field rendering, where users can select between "For Rent" and "For Sale" options that dynamically adjust the pricing fields and related inputs.
The component manages complex state including property details (title, description, type, location hierarchy), media files (up to 12 photos and 1 video) with drag-and-drop

functionality and preview capabilities, geographical data integration with interactive map selection for precise location marking, and comprehensive property specifications including rooms, bathrooms, amenities, and utility inclusions. It features cascading location dropdowns that automatically populate districts and upazilas based on division selection, real-time form validation with error handling and user feedback, and seamless integration with the backend API for both creating new listings and updating existing ones.
The interface demonstrates modern React patterns including controlled components with useState hooks for form state management, useEffect for data fetching and cleanup operations, useMemo for performance optimization of computed values, and conditional rendering based on listing type selection. The component provides an intuitive user experience with tabbed navigation for organizing different sections of the form, file upload with preview and removal capabilities, and comprehensive error handling with loading states to ensure smooth property listing management functionality.

2. Interactive Map Component:

```jsx
import React, { useMemo } from 'react';
import { MapContainer, TileLayer, Marker, useMapEvents } from 'react-leaflet';
import L from 'leaflet';
import 'leaflet/dist/leaflet.css';

const BD_BOUNDS = [
  [20.5, 88.0],
  [26.7, 92.7],
];

const markerIcon = L.icon({
  iconUrl: 'https://unpkg.com/leaflet@1.9.4/dist/images/marker-icon.png',
  shadowUrl: 'https://unpkg.com/leaflet@1.9.4/dist/images/marker-shadow.png',
  iconSize: [25, 41],
  iconAnchor: [12, 41],
  popupAnchor: [1, -34],
});

function ClickSetter({ onPick }) {
  useMapEvents({
    click(e) {
      const { lat, lng } = e.latlng;
      // Clamp to Bangladesh
      const clamped = {
        lat: Math.max(20.5, Math.min(26.7, lat)),
        lng: Math.max(88.0, Math.min(92.7, lng)),
      };
      onPick(clamped);
    },
  });
  return null;
}
```

```jsx
export default function MapPicker({ lat, lng, onChange, height = 280 }) {
  const center = useMemo(() => ({ lat: lat ?? 23.8103, lng: lng ?? 90.4125 }), [lat, lng]);

  const pick = (pos) => {
    if (!onChange) return;
    onChange(pos);
  };

  const useMyLocation = () => {
    if (!navigator.geolocation) return;
    navigator.geolocation.getCurrentPosition((pos) => {
      const { latitude, longitude } = pos.coords;
      pick({
        lat: Math.max(20.5, Math.min(26.7, latitude)),
        lng: Math.max(88.0, Math.min(92.7, longitude)),
      });
    });
  };

  const clear = () => pick({ lat: '', lng: '' });

  const hasPoint = Number.isFinite(lat) && Number.isFinite(lng);

  return (
    <div style={{ display: 'grid', gap: 8 }}>
      <div style={{ display: 'flex', gap: 8, alignItems: 'center' }}>
        <span style={{ fontWeight: 600 }}>Pick location (Bangladesh)</span>
        <button type="button" className="btn sm" onClick={useMyLocation}>Use my location</button>
        <button type="button" className="btn sm ghost" onClick={clear}>Clear</button>
      </div>
      <div style={{ height }}>
        <MapContainer
```

```jsx
      <div style={{ height }}>
        <MapContainer
          center={center}
          zoom={hasPoint ? 14 : 7}
          style={{ height: '100%' }}
          maxBounds={BD_BOUNDS}
          maxBoundsViscosity={1.0}
        >
          <TileLayer
            attribution='&copy; OpenStreetMap contributors'
            url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
          />
          <ClickSetter onPick={pick} />
          {hasPoint && (
            <Marker position={[lat, lng]} icon={markerIcon} />
          )}
        </MapContainer>
      </div>
      <div style={{ fontSize: 12, color: '#555' }}>
        {hasPoint ? `Lat: ${lat.toFixed(6)}, Lng: ${lng.toFixed(6)}` : 'Click on the map to set coordinates'}
      </div>
    </div>
  );
}
```

The Interactive Map Component is a sophisticated geolocation selection interface that integrates Leaflet mapping library with OpenStreetMap tiles to provide users with precise property location marking capabilities within Bangladesh's geographic boundaries. This component implements a fully interactive map interface with geographic constraints that automatically clamps coordinates to Bangladesh's borders (20.5-26.7°N, 88.0-92.7°E) to ensure location accuracy for the local real estate market.

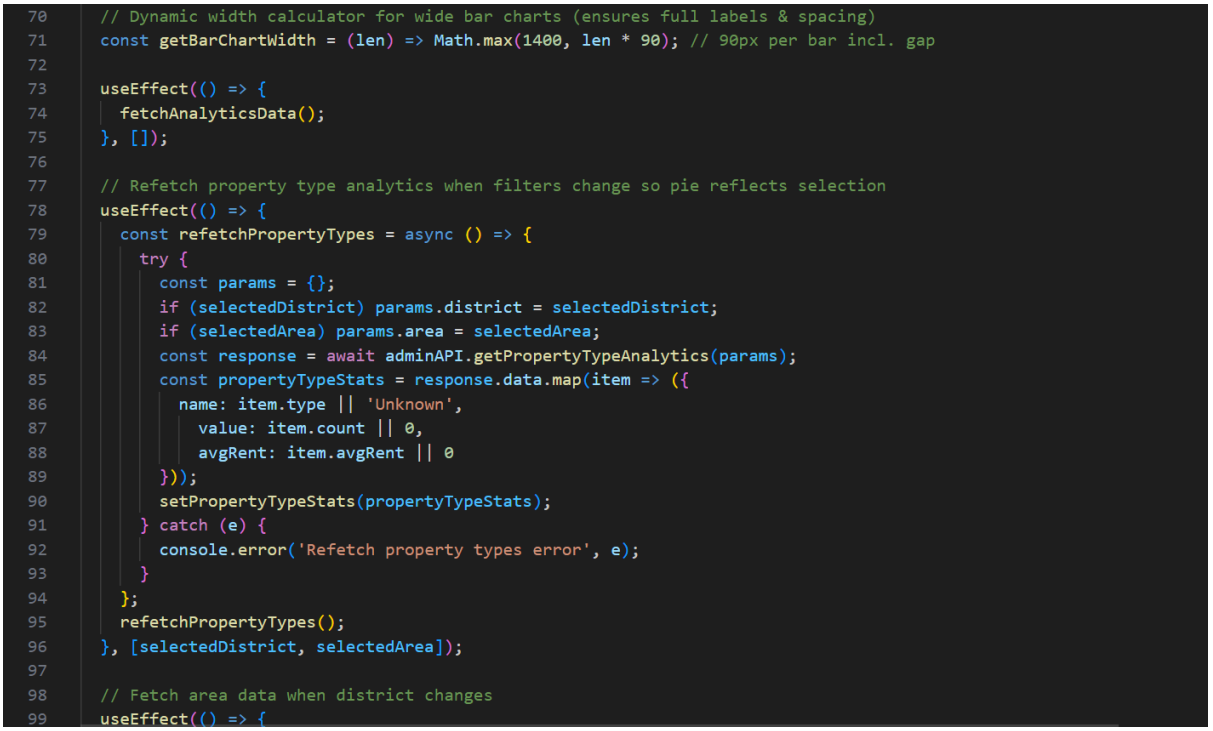
The component manages map state through React hooks including useMemo for performance optimization of map center calculations, custom useMapEvents hook from react-leaflet for

handling click interactions, and useState implicitly through props for coordinate tracking. It features click-to-select functionality where users can click anywhere on the map to set property coordinates, with automatic coordinate clamping to ensure all selections remain within Bangladesh's boundaries. The component also includes geolocation API integration allowing users to automatically detect and use their current location, with fallback handling for unsupported browsers.

The interface demonstrates modern React patterns including component composition with nested ClickSetter component for event handling, conditional rendering of markers based on coordinate validity, controlled component behavior through props and callbacks, and integration with external mapping libraries. The component provides essential functionality for the property listing system by enabling precise geographic data collection using OpenStreetMap's free tile service, supporting both latitude and longitude coordinate capture with real-time display, offering user-friendly controls for location detection and clearing, and ensuring all location data remains within the relevant geographic scope for Bangladesh's real estate market.

3. Analytics Dashboard Component:

```
const Analytics = () => {
  const [loading, setLoading] = useState(true);
  const [refreshing, setRefreshing] = useState(false);
  const [totalSamples, setTotalSamples] = useState(0);
  const [averageRentData, setAverageRentData] = useState([]);
  const [districtStats, setDistrictStats] = useState([]);
  const [areaStats, setAreaStats] = useState([]);
  const [propertyTypeStats, setPropertyTypeStats] = useState([]);
  const [propertyFeatures, setPropertyFeatures] = useState({});
  const [selectedDistrict, setSelectedDistrict] = useState('');
  const [selectedArea, setSelectedArea] = useState('');
  const [selectedPeriod, setSelectedPeriod] = useState('');
  const [currentTab, setCurrentTab] = useState(0);
  const [overallStats, setOverallStats] = useState({
    avgRent: 0,
    minRent: 0,
    maxRent: 0,
    totalProperties: 0,
    uniqueDistricts: 0,
    uniqueAreas: 0
  });

  // Color palette for charts
  const colors = ['#667eea', '#764ba2', '#f093fb', '#f5576c', '#4facfe', '#00f2fe', '#43e97b', '#38f9d7', '#f

  // Dynamic width calculator for wide bar charts (ensures full labels & spacing)
  const getBarChartWidth = (len) => Math.max(1400, len * 90); // 90px per bar incl. gap

  useEffect(() => {
    fetchAnalyticsData();
  }, []);
```



```
// Dynamic width calculator for wide bar charts (ensures full labels & spacing)
const getBarChartWidth = (len) => Math.max(1400, len * 90); // 90px per bar incl. gap

useEffect(() => {
  fetchAnalyticsData();
}, []);

// Refetch property type analytics when filters change so pie reflects selection
useEffect(() => {
  const refetchPropertyTypes = async () => {
    try {
      const params = {};
      if (selectedDistrict) params.district = selectedDistrict;
      if (selectedArea) params.area = selectedArea;
      const response = await adminAPI.getPropertyTypeAnalytics(params);
      const propertyTypeStats = response.data.map(item => ({
        name: item.type || 'Unknown',
          value: item.count || 0,
          avgRent: item.avgRent || 0
      }));
      setPropertyTypeStats(propertyTypeStats);
    } catch (e) {
      console.error('Refetch property types error', e);
    }
  };
  refetchPropertyTypes();
}, [selectedDistrict, selectedArea]);

// Fetch area data when district changes
useEffect(() => {
```

The Analytics Dashboard Component is a sophisticated data visualization interface that provides comprehensive market insights through interactive charts, filtering capabilities, and statistical summaries for the real estate platform. This component implements a comprehensive analytics system that fetches and displays market data from scraped rental samples, featuring dynamic filtering by district and area, real-time statistics calculation, and responsive chart rendering using Recharts library for data visualization.

The component manages complex state including multiple data arrays for district statistics, property type distributions, area-specific analytics, and overall market metrics through useState hooks, implements filtering logic that automatically updates dependent dropdowns and chart data based on user selections, and provides tabbed navigation between overview analytics and detailed district breakdowns. It features interactive charts including responsive bar charts for rent analysis and property counts, pie charts for property type distribution, and stat cards displaying key performance indicators with color-coded icons and dynamic value formatting.

The interface demonstrates advanced React patterns including useEffect dependencies for automated data refetching when filters change, Promise.all for parallel API calls to optimize loading performance, useMemo for computed values and chart width calculations, and comprehensive error handling with loading states. The component integrates Material-UI components for consistent styling, implements responsive design with grid layouts that adapt to different screen sizes, and provides administrative functionality including data refresh capabilities and comprehensive filtering options that enable users to drill down from national overview to specific district and area-level insights within Bangladesh's rental market landscape.

4. Property Listing Details Component:

```jsx
export default function ListingDetails() {
  const { id } = useParams();
  const navigate = useNavigate();
  const { user } = useAuth() || {};
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [mainIdx, setMainIdx] = useState(0);
  const [showDescFull, setShowDescFull] = useState(false);
  // Navigate to central report form with prefilled params

  useEffect(() => {
    let active = true;
    (async () => {
      setLoading(true); setError('');
      try {
        const res = await axios.get(`/api/listings/${id}`);
        if (!active) return;
        setData(res.data);
        setMainIdx(0);
      } catch (e) {
        setError(e?.response?.data?.error || 'Failed to load listing');
      } finally { if (active) setLoading(false); }
    })();
    return () => { active = false; };
  }, [id]);

  // Build unified media items (images + optional video as last item)
  const mediaItems = useMemo(() => {
    if (!data) return [];
    const imgs = (data.photoUrls || []).map(src => ({ type: 'image', src }));
    if (data.videoUrl) imgs.push({ type: 'video', src: data.videoUrl });
    return imgs;
```

```jsx
  }, [data]);
  const current = mediaItems[mainIdx];
                                                    const mediaItems: any
  const goPrev = useCallback(() => setMainIdx(i => (mediaItems.length ? (i - 1 + mediaItems.length) % mediaItems.length : 0)), [mediaItems.length]);
  const goNext = useCallback(() => setMainIdx(i => (mediaItems.length ? (i + 1) % mediaItems.length : 0)), [mediaItems.length]);

  // Keyboard navigation for gallery
  useEffect(() => {
    const handler = (e) => {
      if (e.key === 'ArrowLeft') goPrev();
      if (e.key === 'ArrowRight') goNext();
    };
    window.addEventListener('keydown', handler);
    return () => window.removeEventListener('keydown', handler);
  }, [goPrev, goNext]);

  if (loading) return (
    <Box sx={{ py: 8, display: 'grid', placeItems: 'center' }}>
      <CircularProgress />
    </Box>
  );
  if (error) return (
    <Box sx={{ py: 6, textAlign: 'center' }}>
      <Typography color="error" sx={{ mb: 2 }}>{error}</Typography>
      <Button variant="outlined" onClick={() => navigate(-1)}>Back</Button>
    </Box>
  );
  if (!data) return null;

  const address = [data.houseNo, data.road, data.area, data.subdistrict, data.district, data.division].filter(Boolean).join(', ');
  const features = data.features || [];
```

The Property Listing Details Component is a comprehensive property viewing interface that provides users with detailed information about individual property listings through an

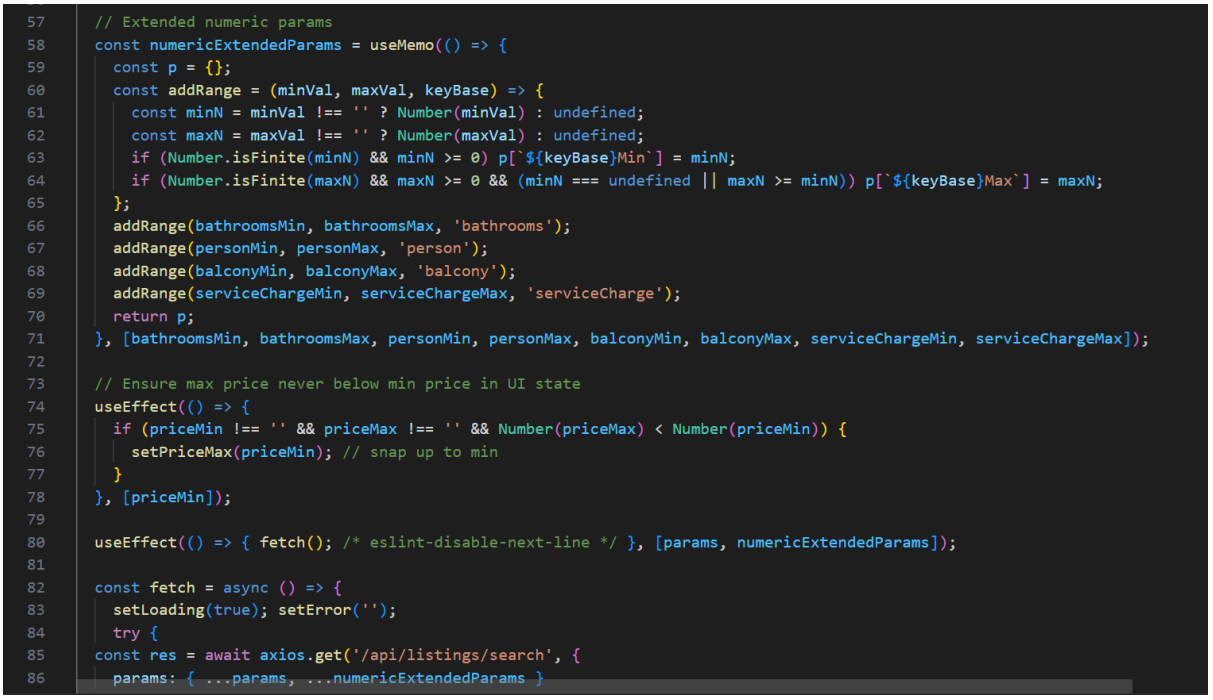
immersive and interactive experience. This component implements a sophisticated media gallery system with navigation controls for browsing through property photos and videos, keyboard navigation support for enhanced user experience, and conditional rendering based on media availability and user authentication status.

The component manages complex state including listing data fetching with loading and error states, media gallery navigation with current index tracking, description expansion for lengthy property descriptions, and user interaction capabilities such as reporting listings and editing permissions for property owners. It features a responsive two-column layout with the main content area displaying property images, details, and location information, while the sidebar contains status information, landlord contact details, and property amenities with icon-based feature badges.

The interface demonstrates advanced React patterns including useCallback for optimized event handlers, useMemo for computed values like media arrays and geographic coordinates, useEffect with cleanup functions for event listeners and async operations, and conditional rendering based on user roles and property ownership. The component integrates Leaflet maps for location display, Material-UI components for consistent styling, and comprehensive error handling to provide a seamless property viewing experience with features like interactive image galleries, detailed property specifications, contact information access, and administrative actions for property management within the BashaLagbe platform.

5. Browse and Searching Component:

```
const params = useMemo(() => {
  const p = { page, limit, sort };
  if (q) p.q = q;
  if (type) p.type = type;
  if (division) p.division = division;
  if (district) p.district = district;
  if (subdistrict) p.subdistrict = subdistrict;
  if (area) p.area = area;
  const minNum = priceMin !== '' ? Number(priceMin) : undefined;
  const maxNum = priceMax !== '' ? Number(priceMax) : undefined;
  if (Number.isFinite(minNum) && minNum >= 0) p.priceMin = minNum;
  if (Number.isFinite(maxNum) && maxNum >= 0 && (minNum === undefined || maxNum >= minNum)) p.priceMax = maxNum; // only if
  const rMinNum = roomsMin !== '' ? Number(roomsMin) : undefined;
  const rMaxNum = roomsMax !== '' ? Number(roomsMax) : undefined;
  if (Number.isFinite(rMinNum) && rMinNum >= 0) p.roomsMin = rMinNum;
  if (Number.isFinite(rMaxNum) && rMaxNum >= 0 && (rMinNum === undefined || rMaxNum >= rMinNum)) p.roomsMax = rMaxNum;
  return p;
}, [q, type, division, district, subdistrict, area, priceMin, priceMax, roomsMin, roomsMax, page, sort]);

// Extended numeric params
const numericExtendedParams = useMemo(() => {
  const p = {};
  const addRange = (minVal, maxVal, keyBase) => {
    const minN = minVal !== '' ? Number(minVal) : undefined;
    const maxN = maxVal !== '' ? Number(maxVal) : undefined;
    if (Number.isFinite(minN) && minN >= 0) p[`${keyBase}Min`] = minN;
    if (Number.isFinite(maxN) && maxN >= 0 && (minN === undefined || maxN >= minN)) p[`${keyBase}Max`] = maxN;
  };
  addRange(bathroomsMin, bathroomsMax, 'bathrooms');
  addRange(personMin, personMax, 'person');
```



```
// Extended numeric params
const numericExtendedParams = useMemo(() => {
  const p = {};
  const addRange = (minVal, maxVal, keyBase) => {
    const minN = minVal !== '' ? Number(minVal) : undefined;
    const maxN = maxVal !== '' ? Number(maxVal) : undefined;
    if (Number.isFinite(minN) && minN >= 0) p[`${keyBase}Min`] = minN;
    if (Number.isFinite(maxN) && maxN >= 0 && (minN === undefined || maxN >= minN)) p[`${keyBase}Max`] = maxN;
  };
  addRange(bathroomsMin, bathroomsMax, 'bathrooms');
  addRange(personMin, personMax, 'person');
  addRange(balconyMin, balconyMax, 'balcony');
  addRange(serviceChargeMin, serviceChargeMax, 'serviceCharge');
  return p;
}, [bathroomsMin, bathroomsMax, personMin, personMax, balconyMin, balconyMax, serviceChargeMin, serviceChargeMax]);

// Ensure max price never below min price in UI state
useEffect(() => {
  if (priceMin !== '' && priceMax !== '' && Number(priceMax) < Number(priceMin)) {
    setPriceMax(priceMin); // snap up to min
  }
}, [priceMin]);

useEffect(() => { fetch(); /* eslint-disable-next-line */ }, [params, numericExtendedParams]);

const fetch = async () => {
  setLoading(true); setError('');
  try {
const res = await axios.get('/api/listings/search', {
  params: { ...params, ...numericExtendedParams }
```

This React component implements a comprehensive property search interface with advanced filtering capabilities that demonstrates sophisticated state management and user experience patterns. The search functionality includes cascading location filters using HTML5 datalist elements for division, district, and subdistrict selection with automatic dependent field clearing, numerical range inputs for price, rooms, bathrooms, and other property specifications with built-in validation to prevent invalid ranges, and dynamic parameter building through useMemo hooks for performance optimization.

The component handles complex search parameter processing by converting string inputs to numbers, validating ranges to ensure maximum values are not less than minimum values, and building separate parameter objects for basic and extended numeric filters. The search form uses controlled components with immediate state updates and pagination reset on filter changes, implements automatic API calls through useEffect dependencies when parameters change, and provides user-friendly features like a reset button that clears all filters and updates the URL, real-time result counting, and sorting options.

The interface demonstrates modern React patterns including responsive grid layouts with CSS Grid, form submission prevention with client-side handling, and comprehensive error handling with loading states for enhanced user experience in property discovery functionality.

7. User Manual

Getting Started - First Visit

Step 1: Accessing the Website

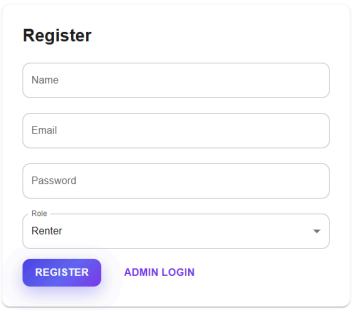
When you first visit BashaLagbe, you'll see the main homepage with a clean, modern interface featuring:

Navigation Bar (Top):

- **BashaLagbe Logo** (top-left) - Click to return home anytime
- **My Listings** - View your property listings (requires login)
- **Browse** - Search available properties (public access)
- **Map** - Interactive map view of properties
- **Trends** - Market analytics and rental trends
- **Admin** - Admin panel access
- **Search Bar** (center) - Quick property search
- **Login/Get Started** buttons (top-right)

User Registration & Login Journey

Step 2: Creating an Account



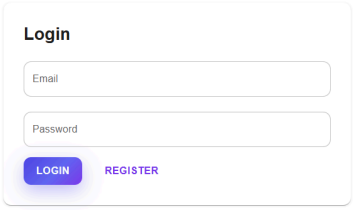
1. **Click "Get Started"** button in the top-right corner
2. You'll see the registration form with fields:
    - **Name**: Enter your full name
    - **Email**: Your email address (becomes your username)
    - **Password**: Create a secure password
    - **Role Dropdown**: Choose your user type:
        - **"Renter"** - If you're looking for properties
        - **"Owner"** - If you want to list properties
3. **Click "Register"** button to create account

4. **Alternative**: Click "Admin Login" for administrative access

After Registration:

- You'll be automatically logged in
- Redirected to your personalized dashboard
- Your avatar appears in the top-right corner with your role badge

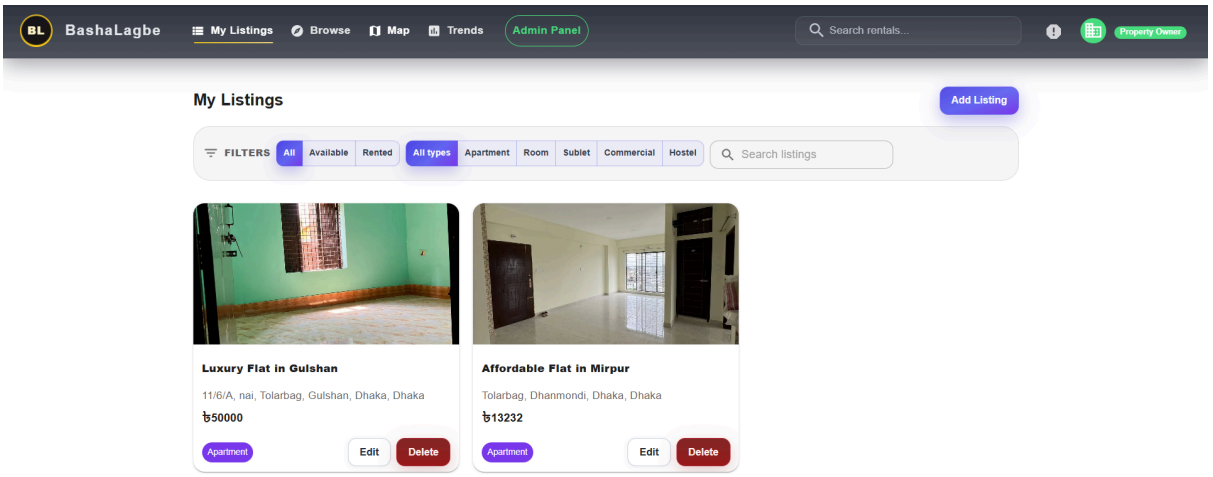
Step 3: Login Process (Returning Users)



1. **Click "Login"** in the top navigation
2. **Enter credentials:**
    - Email address
    - Password
3. **Click "Login"** button
4. **Alternative**: Click "Register" if you need to create an account

Main Navigation Areas

Step 4: Dashboard Overview (Logged-in Users)

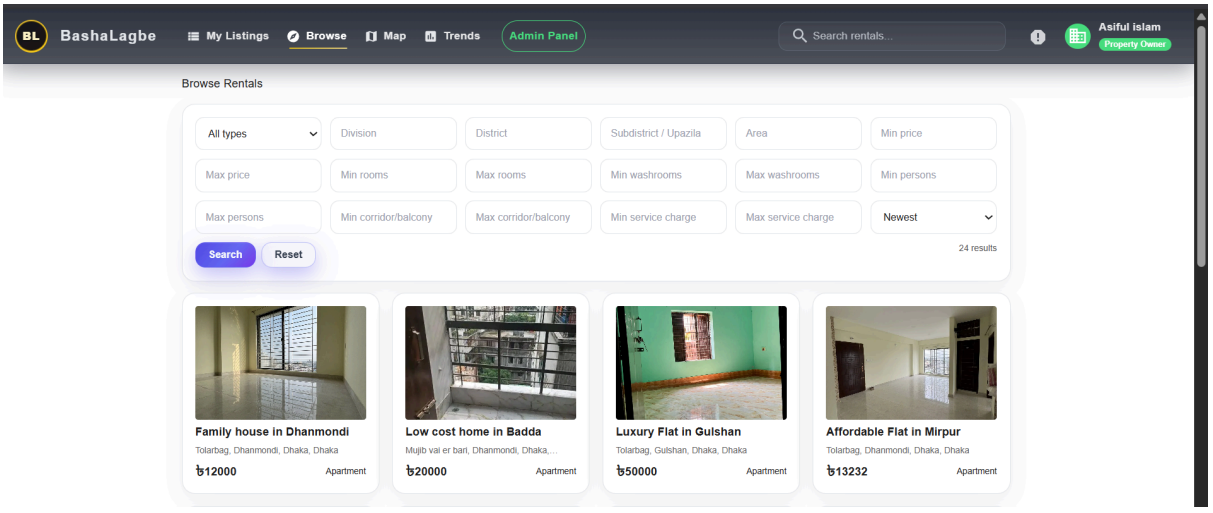


After login, you'll land on the "My Listings" page showing:

- **"Add Listing" button** (for property owners)
- **Filter options:** Status (All/Available/Rented), Type (All property types)
- **Search bar** for your listings
- **Listing cards** with property thumbnails, titles, and prices
- **Action buttons** on each listing: Edit, Delete

Property Discovery Journey

Step 5: Browsing Properties (Public Access)



1. **Click "Browse"** in the main navigation
2. **Advanced Search Interface** displays with:

Location Filters (Top Row):

- **Property Type** dropdown: Apartment, Room, Sublet, Commercial, Hostel
- **Division** field with autocomplete
- **District** field (cascades from Division)
- **Subdistrict/Upazila** field (cascades from District)
- **Area** field with suggestions

Specification Filters (Second Row):

- **Min/Max Price** range sliders
- **Min/Max Rooms** number inputs
- **Min/Max Bathrooms** selectors
- **Min/Max Persons** capacity inputs
- **Balcony/Corridor** count options
- **Service Charge** range inputs

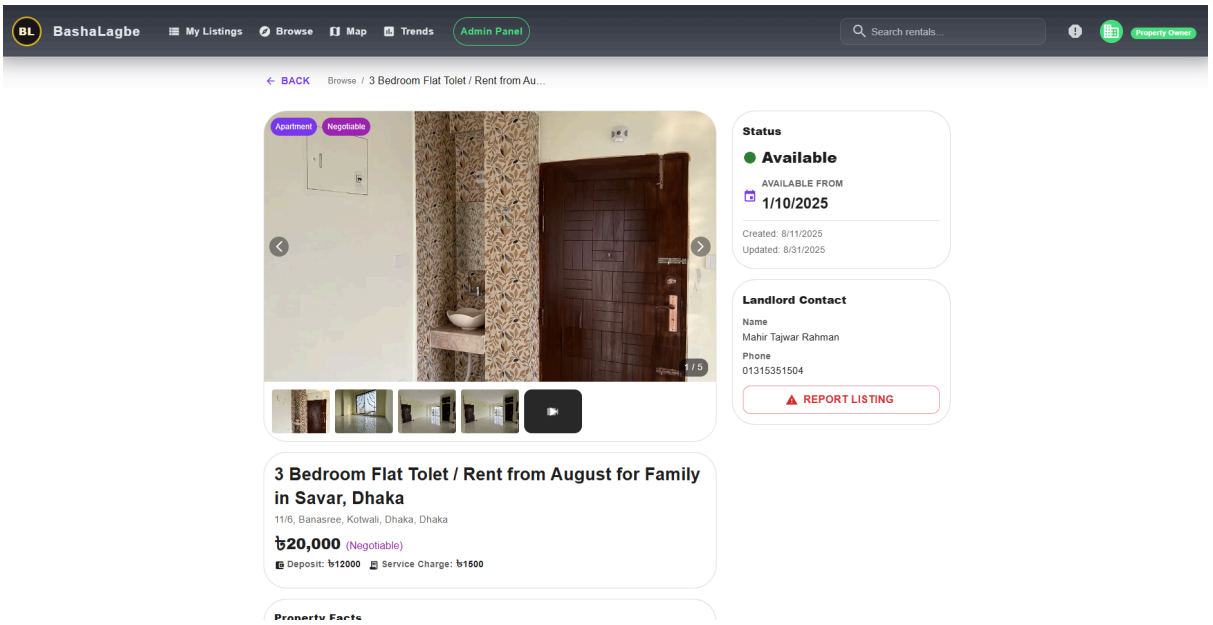
Search Controls:

- **Sort dropdown**: Newest, Oldest, Price (Low-High), Price (High-Low)
- **"Search" button** to apply filters
- **"Reset" button** to clear all filters
- **Results counter** showing total matches

Results Display:

- **Property cards** in grid layout
- Each card shows: Photo, "Rented" badge (if applicable), Title, Location, Price, Property Type

Step 6: Property Details View



1. **Click any property card** from browse results
2. **Breadcrumb navigation** shows: Browse > Property Title
3. **"Back" button** to return to previous page

Main Content Layout:

- **Large photo gallery** with navigation arrows
- **Property status chips**: Rented/Available, Property Type, For Rent/For Sale, Negotiable
- **Photo counter** (e.g., "3 / 8") in bottom-right
- **Thumbnail strip** below main image for quick navigation

Property Information Sections:

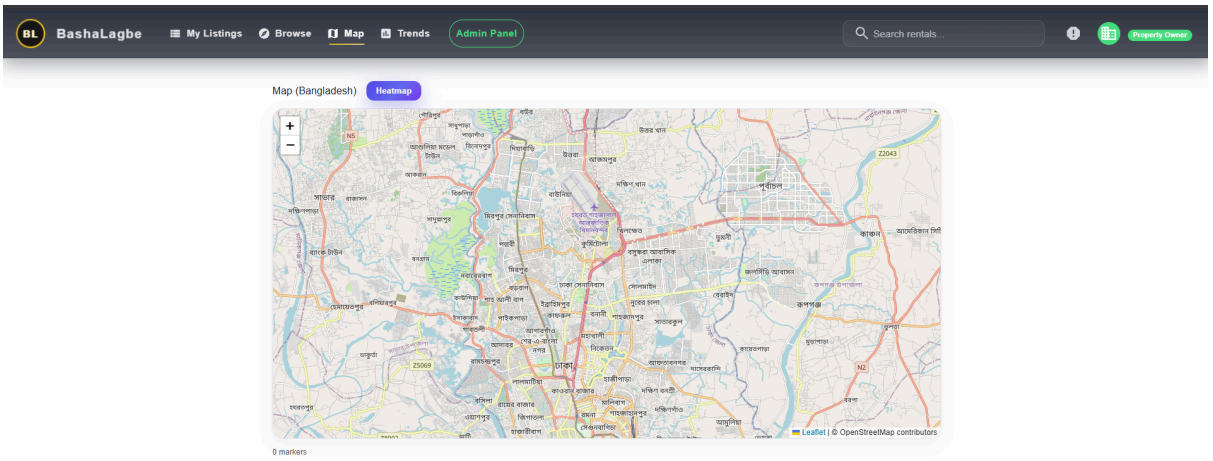
- **Title and Address** prominently displayed
- **Pricing details** with monthly rent and additional charges
- **Property Facts grid** with icons: Type, Rooms, Bathrooms, etc.
- **Description** with "Read more" expansion
- **Interactive map** showing exact location

Sidebar Features:

- **Status panel** showing availability and creation dates
- **Landlord Contact** information
- **"Edit Listing"** button (for property owners)
- **"Report Listing"** button (for other users)
- **Facilities/Features** badges with icons
- **Utilities Included** chips

Map Navigation

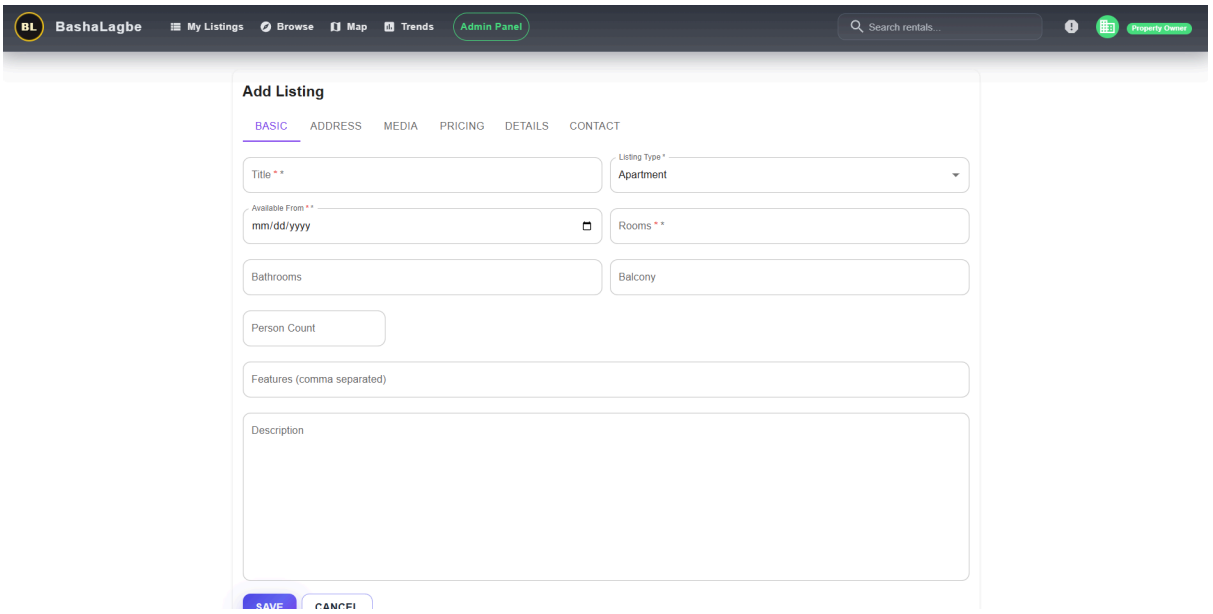
Step 7: Interactive Map View



1. **Click "Map"** in the main navigation
2. **Interactive Bangladesh map** displays with:
    - **Property markers** at exact locations
    - **Zoom controls** for detailed exploration
    - **Click markers** to see property previews
    - **"View Details" links** on marker popups

Property Management (Owners)

Step 8: Adding New Properties



1. **From My Listings**, click **"Add Listing"** button

2. **Multi-tab form interface:**

Basic Information Tab:

- Title, Description, Property Type, Listing Type (Rent/Sale)
- Price, Available From date

Property Details Tab:

- Rooms, Bathrooms, Balcony, Person Count
- Floor details, Size, Furnishing level

Location Tab:

- **Cascading location dropdowns**: Division → District → Subdistrict → Area
- **Interactive map picker** for precise coordinates
- **"Use my location"** and **"Clear"** buttons
- Address fields: Road, House Number

Pricing Tab:

- Monthly rent, Security deposit, Service charges
- Negotiable checkbox

Features Tab:

- Features (comma-separated input)
- Utilities included checklist

Media Tab:

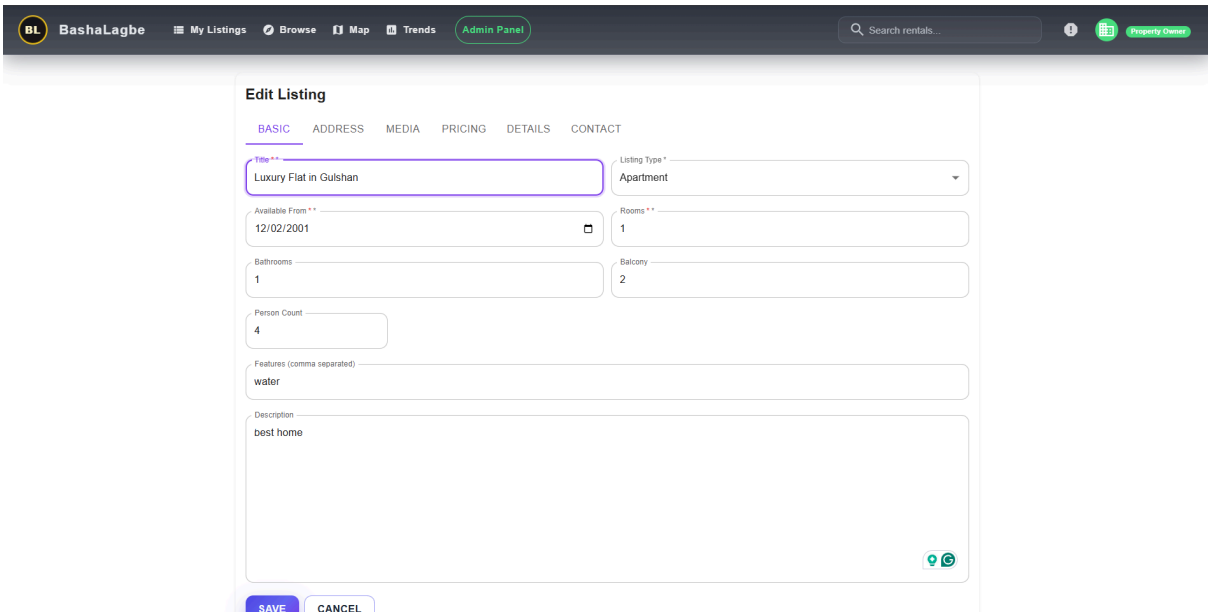
- **Photo upload** with drag-and-drop
- **Video upload** option
- **Preview thumbnails** of uploaded media

Contact Tab:

- Contact name and phone number

3. **"Create Listing"** button to publish

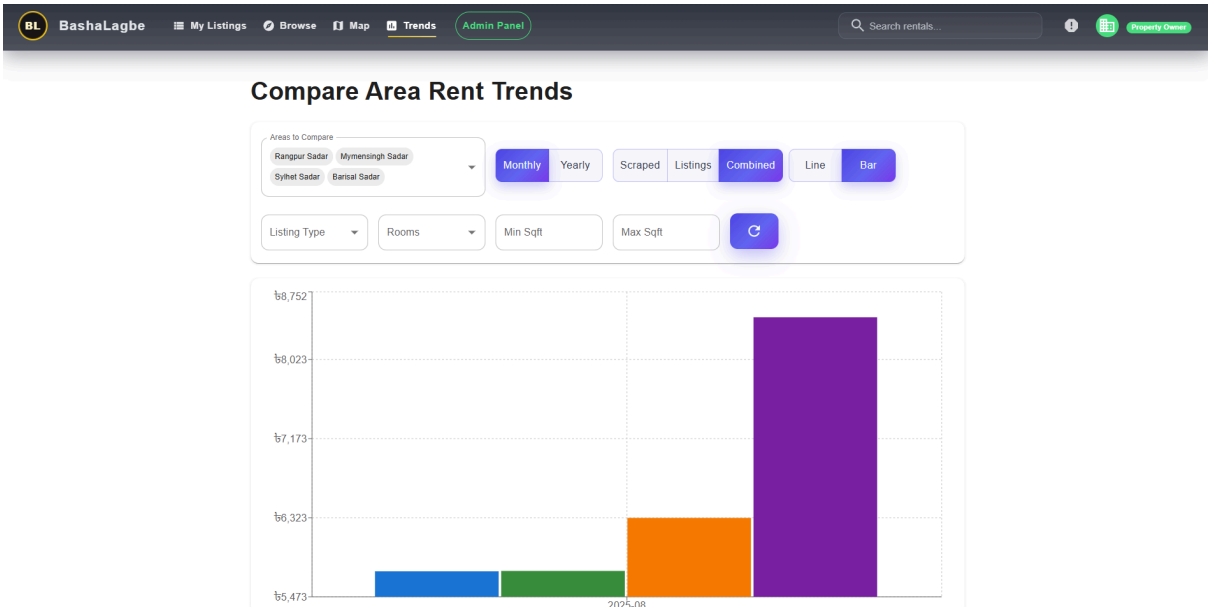
Step 9: Editing Existing Properties



1. **From My Listings**, click **"Edit"** on any property
2. **Same multi-tab interface** as creation, pre-filled with existing data
3. **Media management**: Keep/remove existing photos, upload new ones
4. **"Update Listing"** button to save changes

Market Analytics

Step 10: Trends and Analytics

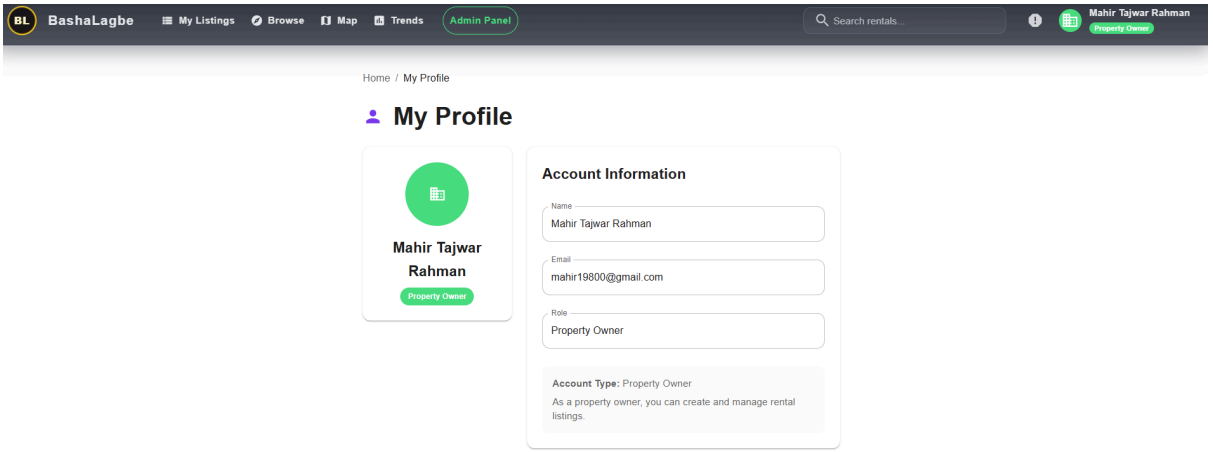


1. **Click "Trends"** in main navigation
2. **Analytics dashboard** with:
    - **Interactive charts** showing price trends over time
    - **Filter controls** for date ranges, locations, property types

- **Comparison tools** between different areas
- **Market statistics** and insights

User Profile Management

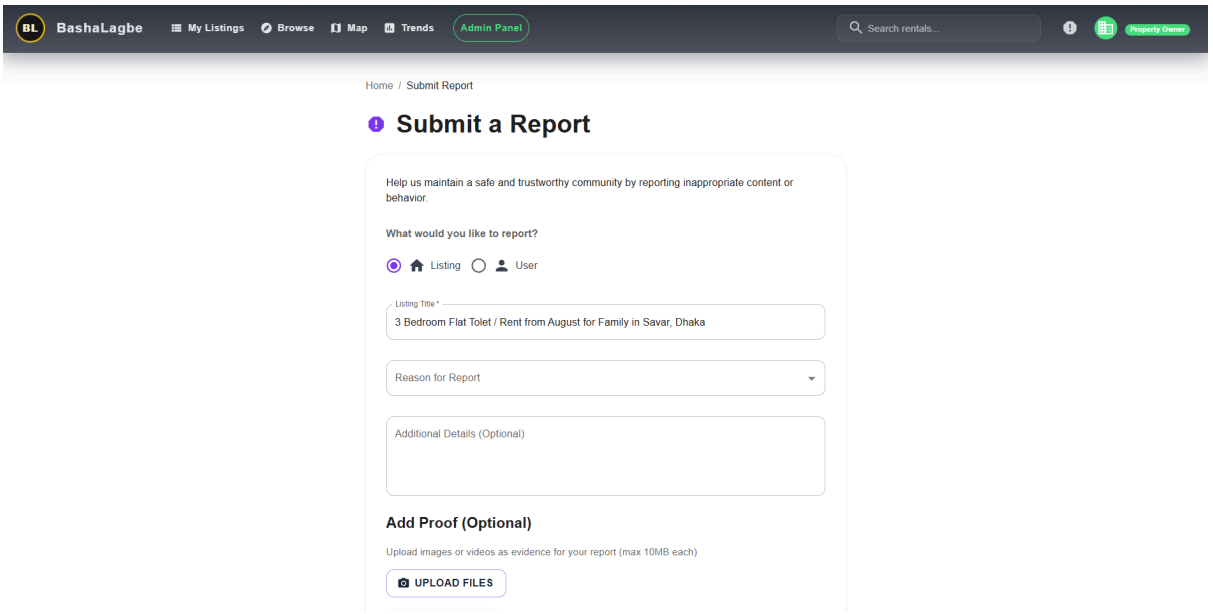
Step 11: Profile Settings



1. **Click your avatar** in top-right corner
2. **Dropdown menu** appears with options:
   - **"My Profile"** - Personal information
   - **"Submit Report"** - Report form access
   - **"My Reports"** - View submitted reports
   - **"Logout"** - Sign out
3. **Profile page** shows:
   - **Personal information form**: Name, Email, Password
   - **Account type** badge and description
   - **"Save Changes"** button

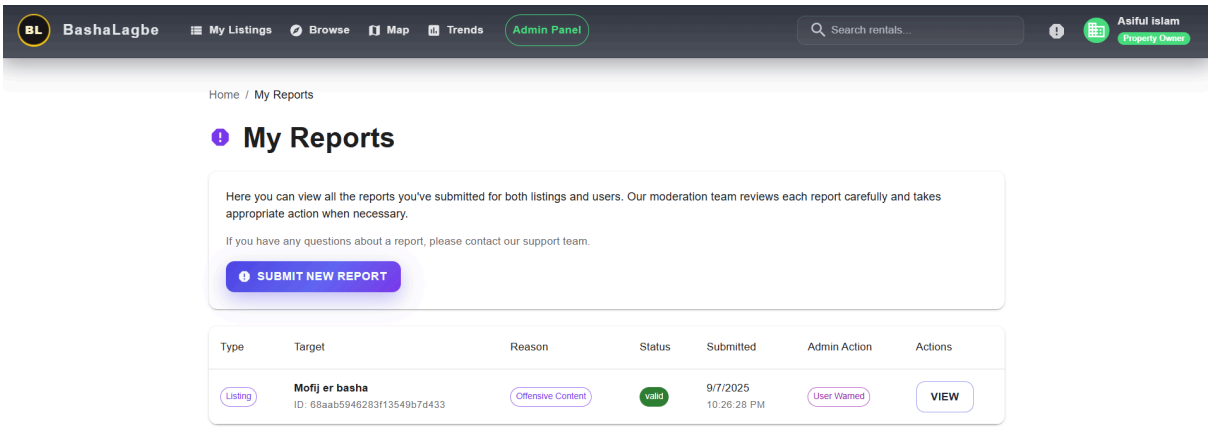
Reporting System

Step 12: Reporting Issues



1. **Access via multiple routes:**
    - User avatar dropdown → "Submit Report"
    - Property details page → "Report Listing"
    - Direct navigation to "/report-form"
2. **Report form interface:**
    - **Report type** selection: Listing or User
    - **Reason** dropdown with predefined categories
    - **Description** text area for details
    - **Auto-filled information** when coming from property pages

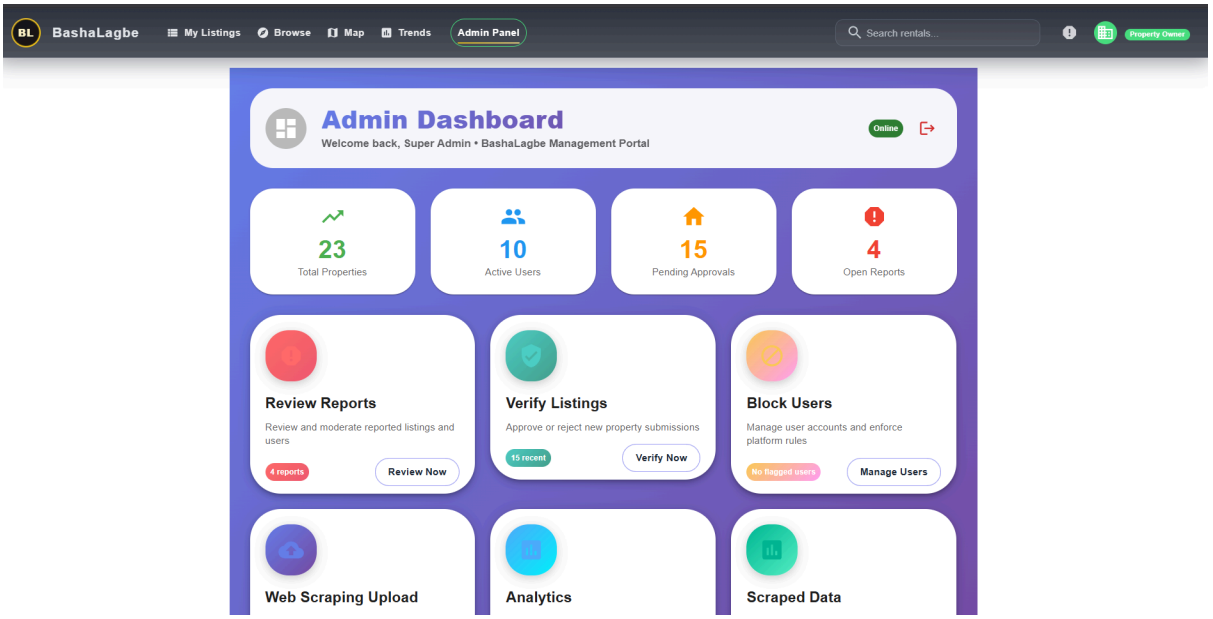
Step 13: Viewing Your Reports



1. **Click avatar → "My Reports"**
2. **Reports list** showing:
    - Report type and target
    - Submission date
    - Current status (Pending, Under Review, Resolved)
    - Report details

Admin Navigation (Admin Users)

Step 14: Admin Panel Access



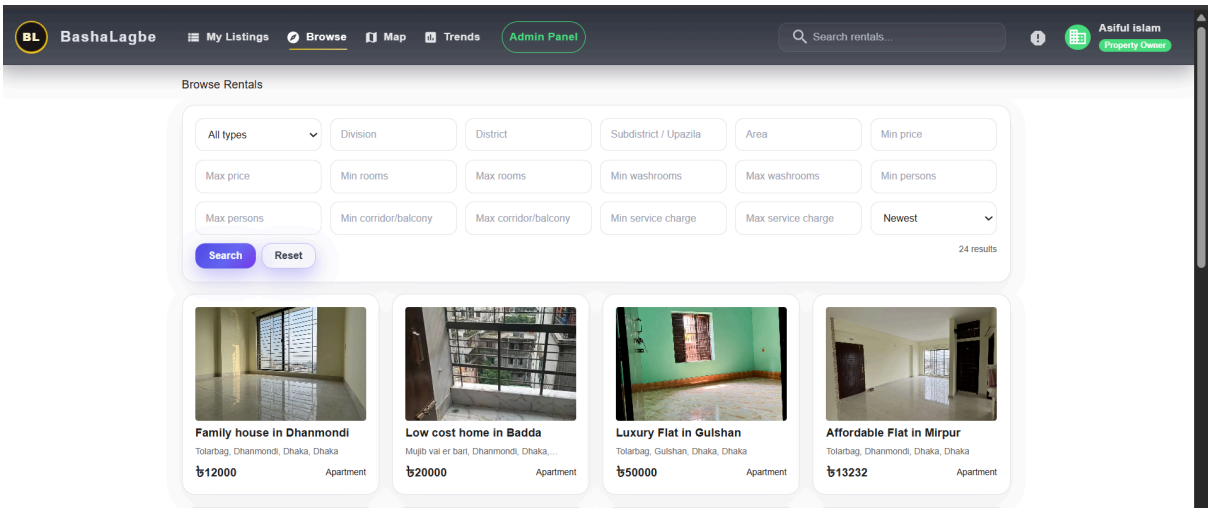
1. **Click "Admin"** in main navigation
2. **If not logged in as admin**: Redirected to admin login
3. **Admin dashboard** displays:
    - **Statistics cards**: Total Properties, Active Users, Pending Approvals, Open Reports
    - **Quick action buttons** for each administrative area

Admin Navigation Options:

- **"Manage Reports"** - Review user reports
- **"Verify Listings"** - Approve new properties
- **"Block Users"** - User management
- **"Web Scraping"** - Data import tools
- **"Analytics"** - Advanced platform analytics

Search and Discovery Flow

Step 15: Search Workflow



1. **Start from any page** using the top search bar
2. **Type keywords** (property title, location, description terms)
3. **Press Enter** or click search icon
4. **Redirected to Browse page** with search results
5. **Refine with filters** if needed
6. **Click properties** to view details

Quick Action Workflows

Step 17: Common User Journeys

For Property Seekers:

1. **Browse → Filter by location/price → View property → Contact owner**

For Property Owners:

1. **Login → My Listings → Add Listing → Fill details → Publish**

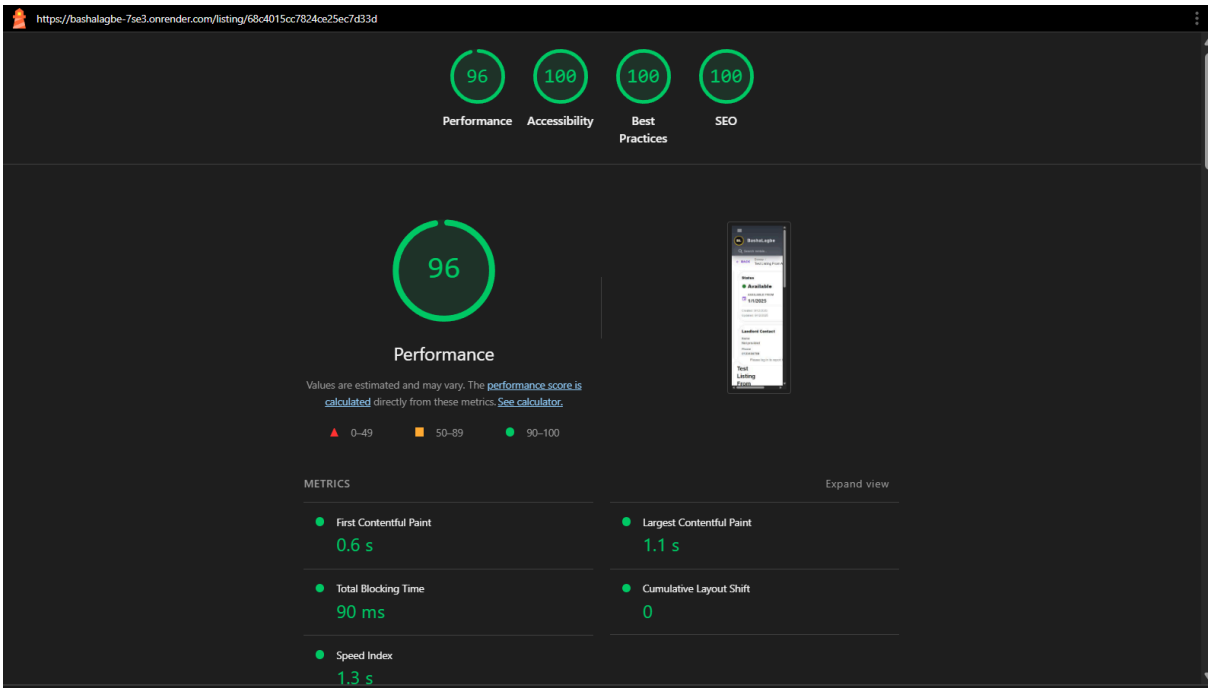
For Market Research:

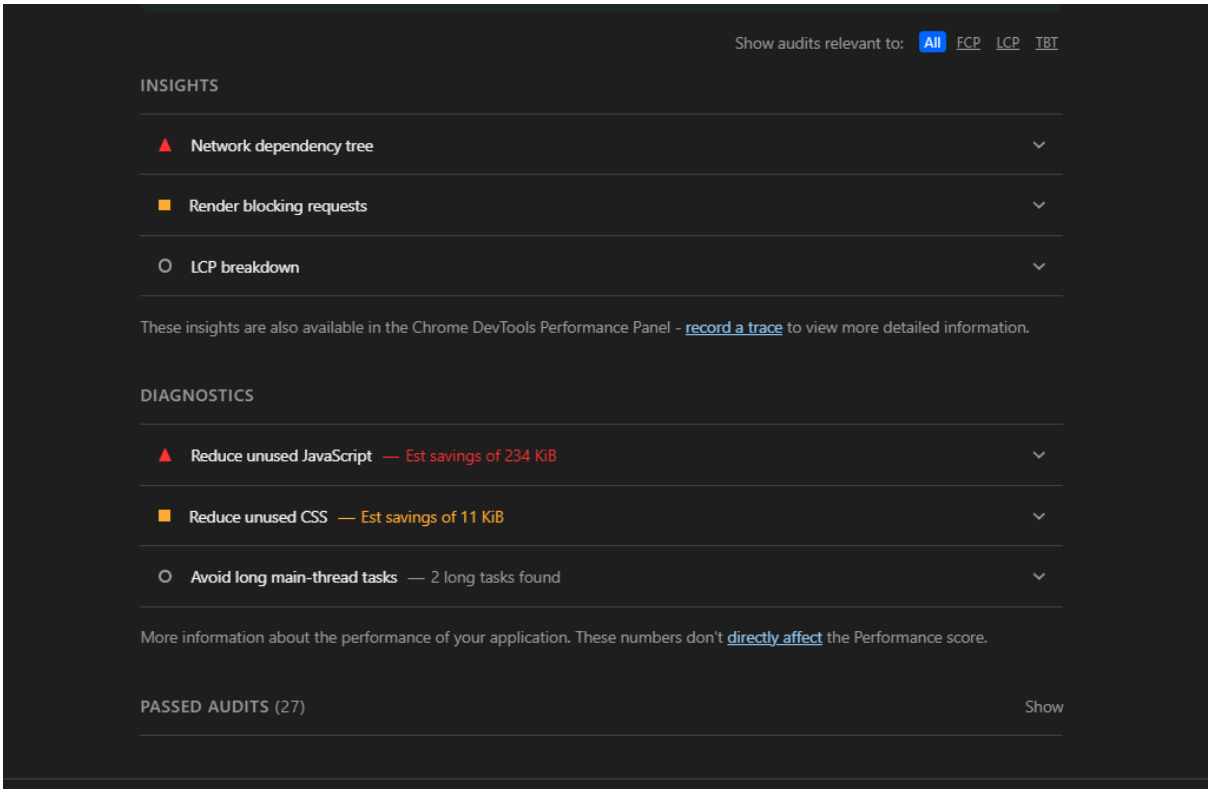
1. **Trends → Select area/property type → Analyze charts → Compare areas**

For Reporting Issues:

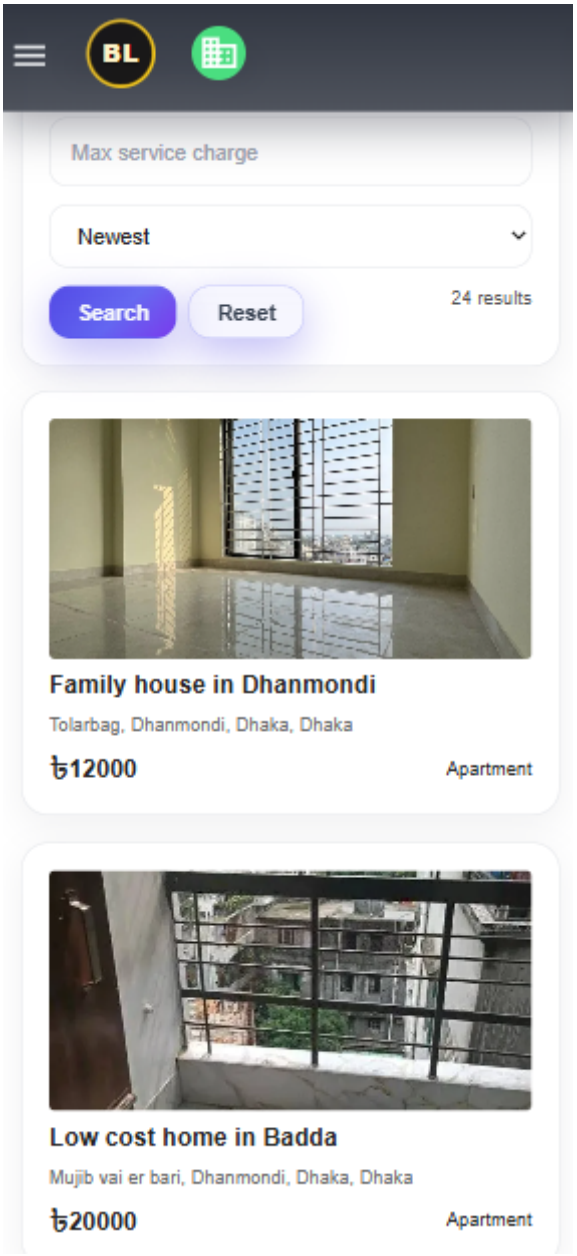
1. **Find problematic content → Report button → Fill form → Submit**

8. Performance and Network Analysis





Mobile Viewport:



9. Github Repo [Public] Link

https://github.com/mahirTaj/BashaLagbe

10. Link of Deployed Project

https://bashalagbe-7se3.onrender.com/

11. Individual Contribution

Group member - 01	
Name: Mahir Tajwar Rahman	Student ID: 22299422
Functional Requirements which are developed by this member:	
1. **Add/Edit Listings** Landlords can create new listings or update existing ones by providing essential information such as property type, location, rent amount, photos, and included amenities.	
2. **Listing Details Page** Each listing page displays comprehensive details including property description, rent, utilities, facilities, landlord contact, and a photo gallery for easy evaluation.	
3. **Web Scraping** Admins can manually collect and upload scraped rent data from public websites like Bikroy or Bproperty.	
4. **Data Validation Interface** Admins can preview uploaded data, fix inconsistencies, and confirm entries before final submission into the system.	

Group member - 02	
Name: Asiful Islam Mahir	Student ID: 22299318
Functional Requirements which are developed by this member:	
1. **Search with Filters** Users can search for listings using advanced filters like price range, preferred location,	
2. **Map Integration** Listings are displayed on an interactive map, enabling users to visually explore rental options by geographic area and proximity to landmarks.	
3. **Price Trend Visualization** The platform displays visual graphs (bar/line charts) showing rent changes over time bybarea, helping users analyze rental trends.	
4. **Flag/Report Listings** Users can report listings for being misleading, spammy, or offensive, which are then reviewed by admins.	

Group member - 03	
Name: Riazul Hoque Bhuban	Student ID: 23341126
Functional Requirements which are developed by this member:	
1. **User Registration & Login** Users can register and log in using email/password or third-party OAuth providers (e.g., Google).	
2. **Role-Based Access Control with Role Switching** The system assigns user roles such as Tenant, Landlord, and Admin. A user can hold multiple roles (e.g., both Tenant and Landlord) and easily switch between them through their profile dashboard. Based on the active role, the system adapts available features.	
3. **Available Time Slot Booking** Landlords can set available time slots for move-in. Tenants can select and book a preferred slot during the rental process. The system ensures no overlapping bookings and sends reminders.	
4. **Listing Comparison Tool** Users can select and compare 2–3 listings side-by-side to assess features like rent, size, amenities, and location.	

Group member - 04	
Name: Fardin Kamran	Student ID: 21101023
Functional Requirements which are developed by this member:	
1. **Moderation & User Management** Admins can review reported listings, verify new submissions before publishing, and block users who violate platform rules. This ensures content quality and community safety.	
2. **Contact Landlord Feature** Tenants can get in touch with landlords directly through an in-app messaging system or a secure email inquiry form integrated with the listing page.	
3. **Wishlist** Users can save listings to a personal wishlist, organize them into folders, and get alerts for price changes or availability updates.	

4. **Notifications System**
   Users receive in-app or email notifications for actions like new messages, listing approvals, or rent changes.

12. References

1. **React.js Documentation**
   - **URL:** https://reactjs.org/docs/
   - **Description:** Official React documentation for component development and hooks implementation.
2. **Material-UI Documentation**
   - **URL:** https://mui.com/
   - **Description:** Comprehensive UI component library documentation for responsive design implementation.
3. **Node.js and Express.js Documentation**
   - **URL:** https://nodejs.org/docs/, https://expressjs.com/
   - **Description:** Backend framework documentation for API development.
4. **MongoDB Documentation**
   - **URL:** https://docs.mongodb.com/
   - **Description:** Database design patterns and query optimization techniques.
5. **Mongoose ODM Documentation**
   - **URL:** https://mongoosejs.com/docs/
   - **Description:** Object modeling and schema design for MongoDB.
6. **JWT Authentication Best Practices**
   - **URL:** https://auth0.com/blog/a-look-at-the-latest-draft-for-jwt-bcp/
   - **Description:** Security implementation guidelines for token-based authentication.
7. **Multer File Upload Documentation**
   - **URL:** https://github.com/expressjs/multer
   - **Description:** File handling and upload implementation.
8. **Recharts Documentation**
   - **URL:** https://recharts.org/en-US/
   - **Description:** Data visualization and chart implementation.
9. **Web Performance Optimization**
   - **URL:** https://web.dev/performance/
   - **Description:** Performance optimization techniques and best practices.
10. **Accessibility Guidelines (WCAG)**
    - **URL:** https://www.w3.org/WAI/WCAG21/quickref/
    - **Description:** Web accessibility compliance and implementation.
11. **Real Estate Market Analysis Methods**
    - **Source:** Academic papers on property price analysis and market trend visualization techniques.

12. **RESTful API Design Principles**
    - **URL:** https://restfulapi.net/
    - **Description:** API architecture and design best practices.