



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	3
Business need:	3
Business requirements:	3
Business value:	3
Special issues or constraints:	4
2. Functional Requirements	5
3. Technology (Framework, Languages)	6
4. Backend Development	7
5. User Interface Design	15
6. Frontend Development	18
7. User Manual	26
8. Performance and Network Analysis	38
9. Github Repo [Public] Link	39
10. Link of Deployed Project	39
11. Individual Contribution	40
12. References	42

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:

```
47 // Create listing (requires userId)
48 router.post('/', upload.fields([{ name: 'photos', maxCount: 12 }, { name: 'video', maxCount: 1 }]), async (req, res) => {
49   try {
50     const userId = getUserId(req);
51     if (!userId) return res.status(400).json({ error: 'userId required' });
52
53     const base = `${req.protocol}://${req.get('host')}`;
54     const photoUrls = (req.files?.photos || []).map((f) => `${base}/uploads/${f.filename}`);
55     const videoUrl = req.files?.video?.[0] ? `${base}/uploads/${req.files.video[0].filename}` : '';
56
57     // Basic required validations
58     const missing = [];
59     if (!req.body.title || !req.body.title.trim()) missing.push('title');
60     if (req.body.price === undefined || req.body.price === '') missing.push('price');
61     if (!req.body.type) missing.push('type');
62     if (!req.body.division) missing.push('division');
63     if (!req.body.district) missing.push('district');
64     if (!req.body.subdistrict) missing.push('subdistrict');
65     if (!req.body.area) missing.push('area');
66     if (!req.body.phone || !req.body.phone.trim()) missing.push('phone');
67     const floorNum = req.body.floor !== undefined ? Number(req.body.floor) : undefined;
68     if (floorNum === undefined || Number.isNaN(floorNum) || floorNum < 0) missing.push('floor');
69     if (!req.body.availableFrom) missing.push('availableFrom');
70     const roomsNum = req.body.rooms !== undefined ? Number(req.body.rooms) : undefined;
71     if (roomsNum === undefined || Number.isNaN(roomsNum) || roomsNum < 0) missing.push('rooms');
72     if (photoUrls.length === 0) missing.push('photos');
73     if (missing.length) return res.status(400).json({ error: 'Missing required fields', fields: missing });
74
75     const payload = {
76       ...req.body,
77       userId,
78     };
79
80     price: ((() => { const n = Number(req.body.price); return Number.isFinite(n) && n > 0 ? n : 0; })(),
81     rooms: Number(req.body.rooms),
82     bathrooms: req.body.bathrooms ? Number(req.body.bathrooms) : 0,
83     balcony: req.body.balcony ? Number(req.body.balcony) : 0,
84     personCount: req.body.personCount ? Number(req.body.personCount) : 1,
85     isRented: req.body.isRented === 'true' || req.body.isRented === true,
86     features: req.body.features
87       ? req.body.features.split(',').map((s) => s.trim()).filter(Boolean)
88       : [],
89     floor: Number(req.body.floor),
90     totalFloors: req.body.totalFloors ? Number(req.body.totalFloors) : 0,
91     furnishing: req.body.furnishing || 'Unfurnished',
92     deposit: req.body.deposit ? Math.max(0, Number(req.body.deposit)) : 0,
93     serviceCharge: req.body.serviceCharge ? Math.max(0, Number(req.body.serviceCharge)) : 0,
94     negotiable: req.body.negotiable === 'true' || req.body.negotiable === true,
95     utilitiesIncluded: req.body.utilitiesIncluded
96       ? req.body.utilitiesIncluded.split(',').map((s) => s.trim()).filter(Boolean)
97       : [],
98     contactName: req.body.contactName || '',
99     phone: req.body.phone || '',
100    availableFrom: new Date(req.body.availableFrom),
101    photoUrls,
102    videoUrl,
103    sizeSqft: req.body.sizeSqft ? Number(req.body.sizeSqft) : 0,
104  };
105  const listing = new Listing(payload);
106  const saved = await listing.save();
107  res.status(201).json(saved);
108 } catch (err) {
109   res.status(500).json({ error: err.message });
110 }
```

```
79     price: ((() => { const n = Number(req.body.price); return Number.isFinite(n) && n > 0 ? n : 0; })(),
80     rooms: Number(req.body.rooms),
81     bathrooms: req.body.bathrooms ? Number(req.body.bathrooms) : 0,
82     balcony: req.body.balcony ? Number(req.body.balcony) : 0,
83     personCount: req.body.personCount ? Number(req.body.personCount) : 1,
84     isRented: req.body.isRented === 'true' || req.body.isRented === true,
85     features: req.body.features
86       ? req.body.features.split(',').map((s) => s.trim()).filter(Boolean)
87       : [],
88     floor: Number(req.body.floor),
89     totalFloors: req.body.totalFloors ? Number(req.body.totalFloors) : 0,
90     furnishing: req.body.furnishing || 'Unfurnished',
91     deposit: req.body.deposit ? Math.max(0, Number(req.body.deposit)) : 0,
92     serviceCharge: req.body.serviceCharge ? Math.max(0, Number(req.body.serviceCharge)) : 0,
93     negotiable: req.body.negotiable === 'true' || req.body.negotiable === true,
94     utilitiesIncluded: req.body.utilitiesIncluded
95       ? req.body.utilitiesIncluded.split(',').map((s) => s.trim()).filter(Boolean)
96       : [],
97     contactName: req.body.contactName || '',
98     phone: req.body.phone || '',
99     availableFrom: new Date(req.body.availableFrom),
100    photoUrls,
101    videoUrl,
102    sizeSqft: req.body.sizeSqft ? Number(req.body.sizeSqft) : 0,
103  };
104  const listing = new Listing(payload);
105  const saved = await listing.save();
106  res.status(201).json(saved);
107 } catch (err) {
108   res.status(500).json({ error: err.message });
109 }
```

The screenshot shows a POST request to `http://localhost:9422/api/listings`. The request body is defined as `form-data` with the following fields:

Key	Value	Description
<code>title</code>	<code>Beautiful Luxury Apartment</code>	
<code>price</code>	<code>25000</code>	
<code>type</code>	<code>Apartment</code>	
<code>division</code>	<code>Dhaka</code>	
<code>district</code>	<code>Dhaka</code>	

The response details are: `201 Created`, `108 ms`, `1002 B`.

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:

```

135 // Update listing (only by owner)
136 router.put('/:id', upload.fields([{ name: 'photos', maxCount: 12 }, { name: 'video', maxCount: 1 }]), async (req, res) => {
137   try {
138     const userId = getUserId(req);
139     if (!userId) return res.status(400).json({ error: 'userId required' });
140
141     const doc = await Listing.findOne({ _id: req.params.id, userId });
142     if (!doc) return res.status(404).json({ error: 'Not found or not owner' });
143
144     const base = `${req.protocol}://${req.get('host')}`;
145     const newPhotoUrls = (req.files?.photos || []).map((f) => `${base}/uploads/${f.filename}`);
146     const keep = req.body.existingPhotoUrls ? JSON.parse(req.body.existingPhotoUrls) : doc.photoUrls;
147
148     // Track originals to know what to delete later
149     const originalPhotoUrls = Array.isArray(doc.photoUrls) ? [...doc.photoUrls] : [];
150     const originalVideoUrl = doc.videoUrl || '';
151
152     doc.title = req.body.title ?? doc.title;
153     doc.description = req.body.description ?? doc.description;
154     doc.type = req.body.type ?? doc.type;
155     if (typeof req.body.price !== 'undefined') {
156       const n = Number(req.body.price);
157       doc.price = Number.isFinite(n) && n > 0 ? n : 0;
158     }
159     if (req.body.availableFrom) doc.availableFrom = new Date(req.body.availableFrom);
160     if (typeof req.body.rooms !== 'undefined') doc.rooms = Number(req.body.rooms);
161     if (req.body.bathrooms) doc.bathrooms = Number(req.body.bathrooms);
162     if (req.body.balcony) doc.balcony = Number(req.body.balcony);
163     if (req.body.personCount) doc.personCount = Number(req.body.personCount);
164     if (typeof req.body.isRented !== 'undefined') doc.isRented = req.body.isRented === 'true' || req.body.isRented === true;
165     if (req.body.features) doc.features = req.body.features.split(',').map((s) => s.trim()).filter(Boolean);

```

```

166     // location removed
167     // Structured address updates
168     if (typeof req.body.division !== 'undefined') doc.division = req.body.division;
169     if (typeof req.body.district !== 'undefined') doc.district = req.body.district;
170     if (typeof req.body.subdistrict !== 'undefined') doc.subdistrict = req.body.subdistrict;
171     if (typeof req.body.area !== 'undefined') doc.area = req.body.area;
172     if (typeof req.body.road !== 'undefined') doc.road = req.body.road;
173     if (typeof req.body.houseNo !== 'undefined') doc.houseNo = req.body.houseNo;
174     if (typeof req.body.floor !== 'undefined') doc.floor = Number(req.body.floor);
175     if (typeof req.body.totalFloors !== 'undefined') doc.totalFloors = Number(req.body.totalFloors) || 0;
176     if (typeof req.body.furnishing !== 'undefined') doc.furnishing = req.body.furnishing;
177     if (typeof req.body.deposit !== 'undefined') doc.deposit = Math.max(0, Number(req.body.deposit)) || 0;
178     if (typeof req.body.serviceCharge !== 'undefined') doc.serviceCharge = Math.max(0, Number(req.body.serviceCharge)) || 0;
179     if (typeof req.body.negotiable !== 'undefined') doc.negotiable = req.body.negotiable === 'true' || req.body.negotiable === true;
180     if (typeof req.body.utilitiesIncluded !== 'undefined') doc.utilitiesIncluded = req.body.utilitiesIncluded.split('.').map((s) => s.trim()).filter(Boolean);
181     if (typeof req.body.contactName !== 'undefined') doc.contactName = req.body.contactName;
182     if (typeof req.body.phone !== 'undefined') doc.phone = req.body.phone;
183     if (typeof req.body.sizeSqft !== 'undefined') doc.sizeSqft = Number(req.body.sizeSqft) || 0;
184
185     doc.photoUrls = [...keep, ...newPhotoUrls];
186     let removedVideoUrl = '';
187     if (req.files?.video?.[0]) {
188       // Replace with newly uploaded video
189       removedVideoUrl = originalVideoUrl;
190       doc.videoUrl = `${base}/uploads/${req.files.video[0].filename}`;
191     } else if (req.body.removeVideo === 'true') {
192       // Explicitly remove existing video
193       removedVideoUrl = originalVideoUrl;
194       doc.videoUrl = '';
195     } else if (req.body.existingVideoUrl) {
196       // Keep existing video when the client confirms it should remain
197       doc.videoUrl = req.body.existingVideoUrl;
198     }
199
200     // Validate required fields after applying changes
201     const must = {
202       title: doc.title,

```

```

203     price: doc.price,
204     type: doc.type,
205     floor: doc.floor,
206     rooms: doc.rooms,
207     availableFrom: doc.availableFrom,
208     division: doc.division,
209     district: doc.district,
210     subdistrict: doc.subdistrict,
211     area: doc.area,
212     phone: doc.phone,
213   };
214   const missing = Object.entries(must).filter(([k, v]) => {
215     if (k === 'floor') return !(typeof v === 'number') || v < 0;
216     if (k === 'rooms') return !(typeof v === 'number') || v < 0;
217     return v === undefined || v === null || (typeof v === 'string' && v.trim() === '');
218   }).map(([k]) => k);
219   if (!doc.photoUrls.length) missing.push('photos');
220   if (missing.length) return res.status(400).json({ error: 'Missing required fields', fields: missing });
221
222   const updated = await doc.save();
223
224   // After successful save, delete any removed photos/videos from disk (best effort)
225   const removedPhotos = originalPhotoUrls.filter(u => !doc.photoUrls.includes(u));
226   const pathsToDelete = [
227     ...removedPhotos.map(urlToUploadPath),
228     urlToUploadPath(removedVideoUrl),
229   ].filter(Boolean);
230   // Fire and forget
231   Promise.all(pathsToDelete.map(unlinkSafe)).catch(() => {});
232
233   res.json(updated);
234 } catch (err) {
235   res.status(500).json({ error: err.message });
236 }
237 );

```

The screenshot shows the Postman interface with a PUT request to `http://localhost:9422/api/listings/689a020fcf176aa7de72c0f0`. The request body is set to `form-data` and contains the following fields:

Key	Value	Description
<input checked="" type="checkbox"/> title	Text	Modern Apartment in Prime Location
<input checked="" type="checkbox"/> road	Text	Tolarbag
<input checked="" type="checkbox"/> houseNo	Text	10/2
<input checked="" type="checkbox"/> price	Text	3000

The response status is `200 OK` with a response time of `134 ms` and a size of `1016 B`.

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

```
114 router.get('/search', async (req, res) => {
115   try {
116     const {
117       q,
118       type,
119       division,
120       district,
121       subdistrict,
122       area,
123       priceMin,
124       priceMax,
125       roomsMin,
126       roomsMax,
127       bathroomsMin,
128       bathroomsMax,
129       personMin,
130       personMax,
131       balconyMin,
132       balconyMax,
133       serviceChargeMin,
134       serviceChargeMax,
135       page = '1',
136       limit = '20',
137       sort = 'newest'.
```

```

135     page = '1',
136     limit = '20',
137     sort = 'newest',
138     isRented,
139   } = req.query;
140
141   const filter = {};
142
143   // Keyword (regex OR). For larger scale consider a text index and $text.
144   if (q && q.trim()) {
145     const kw = q.trim();
146     filter.$or = [
147       { title: { $regex: kw, $options: 'i' } },
148       { description: { $regex: kw, $options: 'i' } },
149       { area: { $regex: kw, $options: 'i' } },
150       { subdistrict: { $regex: kw, $options: 'i' } },
151       { district: { $regex: kw, $options: 'i' } },
152       { division: { $regex: kw, $options: 'i' } },
153     ];
154   }

```

The screenshot shows a Postman collection named "My first collection" with two folders: "First folder inside collection" and "Second folder inside collection". A new request is being created for the URL `http://localhost:3118/api/listings/search?q=Dhamondi`. The method is set to GET. In the "Params" tab, there is a single parameter `q` with the value `Dhamondi`. The response is a 200 OK status with a content type of application/json. The JSON response is as follows:

```

1  {
2    "data": [
3      {
4        "_id": "609a020fcf176aa70e72c020",
5        "isRented": false,
6        "title": "Modern Apartment in Prime Location",
7        "type": "Apartment",
8        "division": "Dhaka",
9        "subdistrict": "Dhakadka",
10       "district": "Dhakadka",
11       "area": "Road 27",
12       "road": "Road 27",
13       "house": "House 22",
14       "price": 3000,
15       "availableFrom": "2024-01-01T00:00:00.000Z",
16       "rooms": 2,
17       "bathrooms": 1,
18       "personCapacity": 4,
19       "balcony": 0,
20       ...
21     }
22   ]
23 }

```

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 });
  }
});
```

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:9318/api/listings/6898f7b6ce4da18417fdf548`. The response body is a JSON object representing a listing:

```
{
  "_id": "6898f7b6ce4da18417fdf548",
  "userId": "user_a",
  "title": "3 Bedroom Flat Tolet / Rent from August for Family in Savar, Dhaka",
  "description": "TO-LET \n\n আপো- বাটস নামের করো।x\n\n Flat Details: ১০৫০ বর্গফুট (কমন প্রেসিডেন্সি), ৩ মিলে ঘোলা, প্রতিটি ঘরে বড় বড় জানালা, সব জায়ে\nআলো- বাতস নামের করো।x\n\n Monthly Rent: 15,000/-\n\n গার্ডেন চার্জিং + বিল (+গ্যারেজ, প্রয়োজন হলে)" ,
  "type": "Apartment",
  "division": "Dhaka",
  "district": "Dhaka"
}
```

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:

```

40  async function postProfile(req, res) {
41    try {
42      const { name, email, password, age } = req.body;
43
44      if (!name || !email || !password || age === undefined) {
45        return res.status(400).json({ message: 'Name, email, password, and age are required' });
46      }
47
48      let profile = await Profile.findOne();
49      if (profile) {
50        profile.name = name;
51        profile.email = email;
52        profile.password = password; // Reminder: hash passwords in real apps
53        profile.age = age;
54        await profile.save();
55        return res.json({ message: 'Profile updated', profile });
56      }
57
58      profile = new Profile({ name, email, password, age });
59      await profile.save();
60      res.json({ message: 'Profile created', profile });
61
62    } catch (err) {
63      res.status(500).json({ error: err.message });
64    }
65  }
66

```

The screenshot shows the Postman interface with a successful POST request to `http://localhost:1126/api/profile`. The request body is a JSON object:

```

1  {
2   "name": "Alice",
3   "email": "alice@example.com",
4   "password": "mySecret123",
5   "age": 30
6 }
7

```

The response is a 200 OK status with a JSON object:

```

1  {
2   "message": "Profile updated",
3   "profile": {
4     "_id": "699a20d0ea180fe160c48edb",
5     "name": "Alice",
6     "age": 30,
7     "email": "alice@example.com",
8     "password": "mySecret123",
9     "age": 30
10}
11

```

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/noyrovI2gdtrPfzAR3aagY/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:

The screenshot shows the 'Add Your Property' form on the BashaLagbe website. At the top, there's a blue header bar with a 'Back to Home' button and the BashaLagbe logo. Below the header, the main title 'Add Your Property' is displayed in bold, followed by a subtitle 'List your property and reach thousands of potential buyers or renters'. There are four tabs at the top: 'Property Details', 'Location & Photos', 'Pricing & Features', and 'Contact & Finish'. The 'Property Details' tab is active, indicated by a blue background. The form is divided into sections: 'Step 1: Property Details', 'Property Title' (containing '4 Bedroom Apartment in Gulshan'), 'Description' (containing 'Describe Your Property in details'), 'Property Type' (with a dropdown menu showing 'Choose'), 'Listing Type' (with a dropdown menu showing 'Choose'), 'Bedrooms' (with a dropdown menu showing 'Choose'), 'Bathrooms' (with a dropdown menu showing 'Choose'), and 'Square Feet' (with a dropdown menu showing 'Choose').

2. Listing Details:

The screenshot shows a property listing for a 4-bedroom apartment. At the top, there's a navigation link '← Back to Edit Listing' and the platform name 'BashaLagbe'. Below the image, the listing details are as follows:

- 4 Bedroom Apartment in Gulshan**
- 123 Maple Street, San Francisco, CA 94102**
- Available from 1st August 2025**
- Pre-Book**
- Open House** (Sunday, Jan 19)

On the right side, there's a contact section featuring a profile picture of 'Mahir Tajwar Rahman' with 'Call Now' and 'Send Message' buttons. There are also left and right arrows indicating more images are available.

3. Searching Property:

The screenshot shows the search interface of the BashaLagbe platform. At the top, there are several buttons: 'BASHA LAGBE' with a clipboard icon, 'Advertise', 'WishList', 'Average rent', and 'Sign in'. Below these are two main search components: a search bar labeled 'Search for rent' with a magnifying glass icon, and a 'Filter' button with a dropdown arrow. To the right of the search bar is a map of a city area with various neighborhoods labeled: M-DOH, Cantonment, Pallabi, Mirpur, Gulshan, and Shewranpara. Key landmarks like the National Zoo, National botanical garden, and Mirpur Cricket Stadium are marked. To the right of the map is a scenic aerial view of a city at sunset.

4. Price Trend Visualization:

The screenshot shows the BASHA LAGBE website's price trend visualization. At the top left is the logo 'BASHA LAGBE' with a house icon. To the right is a blue search bar with the text 'Search for rent' and a magnifying glass icon. Below the search bar is a section titled 'Average Rental Costs by Area'. It features three cards: 'Gulshan' (Premium Residential), 'Banani' (Commercial & Residential), and 'Dhanmondi' (Popular Cultural & Residential). Each card displays rental prices for 1-Bedroom and 3-Bedroom apartments, along with their respective price ranges and key features.

Area	Type	Price Range	Key Features
Gulshan	1-Bedroom Apt	₹25,000 ₹20K - ₹35K	Diplomatic Zone, Shopping Malls, Fine Dining, Corporate Offices, International Schools, 24/7 Security
Gulshan	3-Bedroom Apt	₹65,000 ₹50K - ₹90K	
Banani	1-Bedroom Apt	₹22,000 ₹18K - ₹30K	Banani Lake, Business Hub, Mega Malls, Hotels, Restaurants, Easy Transport
Banani	3-Bedroom Apt	₹55,000 ₹45K - ₹75K	
Dhanmondi	1-Bedroom Apt	₹18,000 ₹15K - ₹25K	Walkable, Cultural Hub, Universities, Parks, Historic, Cafes
Dhanmondi	3-Bedroom Apt	₹40,000 ₹30K - ₹55K	

5. Login page:

The screenshot shows the BASHA LAGBE login page. On the left is a green map of Dhaka city with a yellow house icon and the word 'DHAKA' in the center. To the right is a welcome message 'Hello, Welcome Back!' and a green button 'Find Your Next Home'. Below the button is a text input field 'Enter Your Email'. Further down is another text input field 'Enter Your Password' with a lock icon. Underneath the password field are three buttons: 'Login' (green), 'Forgotten Password?' (light green), and 'Create New Account' (light green). At the bottom is a 'Sign in with Google' button featuring the Google logo.

6. Frontend Development

1. Property Listing Component:

```
1 import React, { useEffect, useMemo, useState } from 'react';
2 import { useNavigate, useParams } from 'react-router-dom';
3 import axios from 'axios';
4 import { useAuth } from '../auth';
5 import { getDivisions, getDistricts, getUpazilas } from '../data/bd-geo';
6 import { Box, Paper, Grid, TextField, Select, MenuItem, FormControl, InputLabel, Button, Typography, Checkbox, FormControlLabel, MapPicker } from '../components/MapPicker';
7
8 const empty = [
9   title: '',
10  description: '',
11  type: 'Apartment',
12  listingType: 'rent', // new field for rent vs sale
13  price: '',
14  availableFrom: '',
15  rooms: '',
16  bathrooms: '',
17  balcony: '',
18  personCount: '',
19  features: '', // comma separated in UI
20  isRented: false,
21  lat: '',
22  lng: ''
];
23 ];
24
25
26 export default function AddEditListing() {
27   const { id } = useParams();
28   const navigate = useNavigate();
29   const { user } = useAuth();
30   const [form, setForm] = useState(empty);
31   const [loading, setLoading] = useState(false);
32   const [existingPhotos, setExistingPhotos] = useState([]); // string URLs
33   const [keepPhoto, setKeepPhoto] = useState({}); // url -> bool
34
35   const [newPhotos, setNewPhotos] = useState([]); // File[]
36   const [videoUrl, setVideoUrl] = useState(''); // existing video url
37   const [newVideo, setNewVideo] = useState(null); // File | null
38   const newVideoPreview = useMemo(() => (newVideo ? URL.createObjectURL(newVideo) : ''), [newVideo]);
39   useEffect(() => {
40     return () => {
41       // revoke object URL when file changes/unmounts
42       if (newVideoPreview) URL.revokeObjectURL(newVideoPreview);
43     };
44   }, [newVideoPreview]);
45   const [dragActive, setDragActive] = useState(false);
46   const [removeVideo, setRemoveVideo] = useState(false);
47   const maxPhotos = 12;
48   const [tab, setTab] = useState(0);
49   const [showPicker, setShowPicker] = useState(false);
50
51   const keepCount = useMemo(
52     () => existingPhotos.filter((u) => keepPhoto[u]).length,
53     [existingPhotos, keepPhoto]
54   );
55   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:

```
1  import React, { useMemo } from 'react';
2  import { MapContainer, TileLayer, Marker, useMapEvents } from 'react-leaflet';
3  import L from 'leaflet';
4  import 'leaflet/dist/leaflet.css';
5
6  const BD_BOUNDS = [
7    [20.5, 88.0],
8    [26.7, 92.7],
9  ];
10
11 const markerIcon = L.icon({
12   iconUrl: 'https://unpkg.com/leaflet@1.9.4/dist/images/icon.png',
13   shadowUrl: 'https://unpkg.com/leaflet@1.9.4/dist/images/icon-shadow.png',
14   iconSize: [25, 41],
15   iconAnchor: [12, 41],
16   popupAnchor: [1, -34],
17 });
18
19 function ClickSetter({ onPick }) {
20   useMapEvents({
21     click(e) {
22       const { lat, lng } = e.latlng;
23       // Clamp to Bangladesh
24       const clamped = {
25         lat: Math.max(20.5, Math.min(26.7, lat)),
26         lng: Math.max(88.0, Math.min(92.7, lng)),
27       };
28       onPick(clamped);
29     },
30   });
31   return null;
32 }
```

```

34  export default function MapPicker({ lat, lng, onChange, height = 280 }) {
35    const center = useMemo(() => ({ lat: lat ?? 23.8103, lng: lng ?? 90.4125 }), [lat, lng]);
36
37    const pick = (pos) => {
38      if (!onChange) return;
39      onChange(pos);
40    };
41
42    const useMyLocation = () => {
43      if (!navigator.geolocation) return;
44      navigator.geolocation.getCurrentPosition((pos) => {
45        const { latitude, longitude } = pos.coords;
46        pick({
47          lat: Math.max(20.5, Math.min(26.7, latitude)),
48          lng: Math.max(88.0, Math.min(92.7, longitude)),
49        });
50      });
51    };
52
53    const clear = () => pick({ lat: '', lng: '' });
54
55    const hasPoint = Number.isFinite(lat) && Number.isFinite(lng);
56
57    return (
58      <div style={{ display: 'grid', gap: 8 }}>
59        <div style={{ display: 'flex', gap: 8, alignItems: 'center' }}>
60          <span style={{ fontWeight: 600 }>Pick location (Bangladesh)</span>
61          <button type="button" className="btn sm" onClick={useMyLocation}>Use my location</button>
62          <button type="button" className="btn sm ghost" onClick={clear}>Clear</button>
63        </div>
64        <div style={{ height }}>
65          <MapContainer
66            center={center}
67            zoom={hasPoint ? 14 : 7}
68            style={{ height: '100%' }}
69            maxBounds={BD_BOUNDS}
70            maxBoundsViscosity={1.0}
71          >
72            <TileLayer
73              attribution='&copy; OpenStreetMap contributors'
74              url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
75            />
76            <ClickSetter onClick={pick} />
77            {hasPoint && (
78              <Marker position={[lat, lng]} icon={markerIcon} />
79            )}
80          </MapContainer>
81        </div>
82        <div style={{ fontSize: 12, color: '#555' }}>
83          {hasPoint ? `Lat: ${lat.toFixed(6)}, Lng: ${lng.toFixed(6)}` : 'Click on the map to set coordinates'}
84        </div>
85      </div>
86    );
87  }
88

```

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:

```
45 const Analytics = () => {
46   const [loading, setLoading] = useState(true);
47   const [refreshing, setRefreshing] = useState(false);
48   const [totalsamples, setTotalSamples] = useState(0);
49   const [averageRentData, setAverageRentData] = useState([]);
50   const [districtStats, setDistrictStats] = useState([]);
51   const [areaStats, setAreaStats] = useState([]);
52   const [propertyTypeStats, setPropertyTypeStats] = useState([]);
53   const [propertyFeatures, setPropertyFeatures] = useState({});
54   const [selectedDistrict, setSelectedDistrict] = useState('');
55   const [selectedArea, setSelectedArea] = useState('');
56   const [selectedPeriod, setSelectedPeriod] = useState('');
57   const [currentTab, setCurrentTab] = useState(0);
58   const [overallStats, setOverallStats] = useState({
59     avgRent: 0,
60     minRent: 0,
61     maxRent: 0,
62     totalProperties: 0,
63     uniqueDistricts: 0,
64     uniqueAreas: 0
65   });
66
67   // Color palette for charts
68   const colors = ['#667eea', '#764ba2', '#f093fb', '#f5576c', '#4facfe', '#00f2fe', '#43e97b', '#38f9d7', '#f1
69
70   // Dynamic width calculator for wide bar charts (ensures full labels & spacing)
71   const getBarChartWidth = (len) => Math.max(1400, len * 90); // 90px per bar incl. gap
72
73   useEffect(() => {
74     fetchAnalyticsData();
75   }, []);
}
```

```

70 // Dynamic width calculator for wide bar charts (ensures full labels & spacing)
71 const getBarChartWidth = (len) => Math.max(1400, len * 90); // 90px per bar incl. gap
72
73 useEffect(() => {
74   fetchAnalyticsData();
75 }, []);
76
77 // Refetch property type analytics when filters change so pie reflects selection
78 useEffect(() => {
79   const refetchPropertyTypes = async () => {
80     try {
81       const params = {};
82       if (selectedDistrict) params.district = selectedDistrict;
83       if (selectedArea) params.area = selectedArea;
84       const response = await adminAPI.getPropertyTypeAnalytics(params);
85       const propertyTypeStats = response.data.map(item => ({
86         name: item.type || 'Unknown',
87         value: item.count || 0,
88         avgRent: item.avgRent || 0
89       }));
90       setPropertyTypeStats(propertyTypeStats);
91     } catch (e) {
92       console.error('Refetch property types error', e);
93     }
94   };
95   refetchPropertyTypes();
96 }, [selectedDistrict, selectedArea]);
97
98 // Fetch area data when district changes
99 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:

```

57  export default function ListingDetails() {
58    const { id } = useParams();
59    const navigate = useNavigate();
60    const { user } = useAuth() || {};
61    const [data, setData] = useState(null);
62    const [loading, setLoading] = useState(true);
63    const [error, setError] = useState('');
64    const [mainIdx, setMainIdx] = useState(0);
65    const [showDescFull, setShowDescFull] = useState(false);
66    // Navigate to central report form with prefilled params
67
68    useEffect(() => {
69      let active = true;
70      (async () => {
71        setLoading(true); setError('');
72        try {
73          const res = await axios.get(`api/listings/${id}`);
74          if (!active) return;
75          setData(res.data);
76          setMainIdx(0);
77        } catch (e) {
78          setError(e?.response?.data?.error || 'Failed to load listing');
79        } finally { if (active) setLoading(false); }
80      })();
81      return () => { active = false; };
82    }, [id]);
83
84    // Build unified media items (images + optional video as last item)
85    const mediaItems = useMemo(() => {
86      if (!data) return [];
87      const imgs = (data.photoUrls || []).map(src => { type: 'image', src });
88      if (data.videoUrl) imgs.push({ type: 'video', src: data.videoUrl });
89      return imgs;
90    }, [data]);
91    const current = mediaItems[mainIdx];
92    const goPrev = useCallback(() => setMainIdx(i => (mediaItems.length ? (i - 1 + mediaItems.length) % mediaItems.length : 0)), [mediaItems.length]);
93    const goNext = useCallback(() => setMainIdx(i => (mediaItems.length ? (i + 1) % mediaItems.length : 0)), [mediaItems.length]);
94
95    // Keyboard navigation for gallery
96    useEffect(() => {
97      const handler = (e) => {
98        if (e.key === 'ArrowLeft') goPrev();
99        if (e.key === 'ArrowRight') goNext();
100      };
101      window.addEventListener('keydown', handler);
102      return () => window.removeEventListener('keydown', handler);
103    }, [goPrev, goNext]);
104
105    if (loading) return (
106      <Box sx={{ py: 8, display: 'grid', placeItems: 'center' }}>
107        <CircularProgress />
108      </Box>
109    );
110    if (error) return (
111      <Box sx={{ py: 6, textAlign: 'center' }}>
112        <Typography color="error" sx={{ mb: 2 }}>{error}</Typography>
113        <Button variant="outlined" onClick={() => navigate(-1)}>Back</Button>
114      </Box>
115    );
116    if (!data) return null;
117
118    const address = [data.houseNo, data.road, data.area, data.subdistrict, data.district, data.division].filter(Boolean).join(' ');
119    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:

```

38  const params = useMemo(() => {
39    const p = { page, limit, sort };
40    if (q) p.q = q;
41    if (type) p.type = type;
42    if (division) p.division = division;
43    if (district) p.district = district;
44    if (subdistrict) p.subdistrict = subdistrict;
45    if (area) p.area = area;
46    const minNum = priceMin !== '' ? Number(priceMin) : undefined;
47    const maxNum = priceMax !== '' ? Number(priceMax) : undefined;
48    if (Number.isFinite(minNum) && minNum >= 0) p.priceMin = minNum;
49    if (Number.isFinite(maxNum) && maxNum >= 0 && (minNum === undefined || maxNum >= minNum)) p.priceMax = maxNum; // only if
50    const rMinNum = roomsMin !== '' ? Number(roomsMin) : undefined;
51    const rMaxNum = roomsMax !== '' ? Number(roomsMax) : undefined;
52    if (Number.isFinite(rMinNum) && rMinNum >= 0) p.roomsMin = rMinNum;
53    if (Number.isFinite(rMaxNum) && rMaxNum >= 0 && (rMinNum === undefined || rMaxNum >= rMinNum)) p.roomsMax = rMaxNum;
54    return p;
55  }, [q, type, division, district, subdistrict, area, priceMin, priceMax, roomsMin, roomsMax, page, sort]);
56
57  // Extended numeric params
58  const numericExtendedParams = useMemo(() => {
59    const p = {};
60    const addRange = (minVal, maxVal, keyBase) => {
61      const minN = minVal !== '' ? Number(minVal) : undefined;
62      const maxN = maxVal !== '' ? Number(maxVal) : undefined;
63      if (Number.isFinite(minN) && minN >= 0) p[`_${keyBase}Min`] = minN;
64      if (Number.isFinite(maxN) && maxN >= 0 && (minN === undefined || maxN >= minN)) p[`_${keyBase}Max`] = maxN;
65    };
66    addRange(bathroomsMin, bathroomsMax, 'bathrooms');
67    addRange(personMin, personMax, 'person');

```

```

57 // Extended numeric params
58 const numericExtendedParams = useMemo(() => {
59   const p = {};
60   const addRange = (minVal, maxVal, keyBase) => {
61     const minN = minVal !== '' ? Number(minVal) : undefined;
62     const maxN = maxVal !== '' ? Number(maxVal) : undefined;
63     if (Number.isFinite(minN) && minN >= 0) p[`_${keyBase}Min`] = minN;
64     if (Number.isFinite(maxN) && maxN >= 0 && (minN === undefined || maxN >= minN)) p[`_${keyBase}Max`] = maxN;
65   };
66   addRange(bathroomsMin, bathroomsMax, 'bathrooms');
67   addRange(personMin, personMax, 'person');
68   addRange(balconyMin, balconyMax, 'balcony');
69   addRange(serviceChargeMin, serviceChargeMax, 'serviceCharge');
70   return p;
71 }, [bathroomsMin, bathroomsMax, personMin, personMax, balconyMin, balconyMax, serviceChargeMin, serviceChargeMax]);
72
73 // Ensure max price never below min price in UI state
74 useEffect(() => {
75   if (priceMin !== '' && priceMax !== '' && Number(priceMax) < Number(priceMin)) {
76     setPriceMax(priceMin); // snap up to min
77   }
78 }, [priceMin]);
79
80 useEffect(() => { fetch(); /* eslint-disable-next-line */ }, [params, numericExtendedParams]);
81
82 const fetch = async () => {
83   setLoading(true); setError('');
84   try {
85     const res = await axios.get('/api/listings/search', {
86       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

The screenshot shows the BashaLagbe homepage with a dark header bar. The header includes the BashaLagbe logo, navigation links for 'My Listings', 'Browse', 'Map', 'Trends', and 'Admin Panel', a search bar with placeholder 'Search rentals ...', and two buttons: 'LOGIN' and a purple 'GET STARTED' button. Below the header, a large registration form is displayed with a light gray background. The form is titled 'Register' and contains four input fields: 'Name', 'Email', and 'Password', followed by a 'Role' dropdown menu set to 'Renter'. At the bottom of the form are two buttons: a purple 'REGISTER' button and a smaller 'ADMIN LOGIN' link.

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)

A screenshot of the BashaLagbe login form. The form is titled 'Login' and contains two input fields: 'Email' and 'Password'. Below the password field is a 'Forgot Password?' link. At the bottom of the form are two buttons: a purple 'LOGIN' button on the left and a blue 'REGISTER' button on the right.

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)

The screenshot shows the BashaLagbe dashboard for a logged-in user named 'Asiful islam'. The top navigation bar includes links for 'My Listings', 'Browse', 'Map', 'Trends', 'Admin Panel', a search bar, and a 'Property Owner' profile icon. Below the navigation is a section titled 'My Listings' with a heading 'Add Listing'. A filter bar allows selecting 'All', 'Available', or 'Rented' status and 'All types', 'Apartment', 'Room', 'Sublet', 'Commercial', or 'Hostel' type. A search bar for 'Search listings...' is also present. Two property listings are displayed as cards:

- Luxury Flat in Gulshan**: Located at 11/6/A, nai, Tolarbag, Gulshan, Dhaka, Dhaka. Price: ₢50000. Type: Apartment. Action buttons: Edit, Delete.
- Affordable Flat in Mirpur**: Located at Tolarbag, Dhanmondi, Dhaka, Dhaka. Price: ₢13232. Type: Apartment. Action buttons: Edit, Delete.

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)

The screenshot shows the BashaLagbe browse interface. The top navigation bar includes links for 'My Listings', 'Browse', 'Map', 'Trends', 'Admin Panel', a search bar, and a 'Property Owner' profile icon for 'Asiful islam'. Below the navigation is a section titled 'Browse Rentals' with an advanced search interface. The search interface includes dropdowns for 'All types', 'Division', 'District', 'Subdistrict / Upazila', 'Area', and 'Min price', along with input fields for 'Max price', 'Min rooms', 'Max rooms', 'Min washrooms', 'Max washrooms', 'Min persons', 'Max persons', 'Min corridor/balcony', 'Max corridor/balcony', 'Min service charge', 'Max service charge', and a dropdown for sorting by 'Newest'. A 'Search' button and a 'Reset' button are at the bottom of the search interface. Below the search interface are four property cards:

- Family house in Dhanmondi**: Located at Tolarbag, Dhanmondi, Dhaka, Dhaka. Price: ₢12000. Type: Apartment.
- Low cost home in Badda**: Located at Mujib val er bari, Dhanmondi, Dhaka, Dhaka. Price: ₢20000. Type: Apartment.
- Luxury Flat in Gulshan**: Located at Tolarbag, Gulshan, Dhaka, Dhaka. Price: ₢50000. Type: Apartment.
- Affordable Flat in Mirpur**: Located at Tolarbag, Dhanmondi, Dhaka, Dhaka. Price: ₢13232. Type: Apartment.

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

The screenshot shows a detailed view of a property listing. At the top, there's a navigation bar with links like 'My Listings', 'Browse', 'Map', 'Trends', 'Admin Panel', and a search bar. Below the navigation is a breadcrumb trail: 'BACK / Browse / 3 Bedroom Flat Tolet / Rent from Au...'. The main content area features a large image of a room with a sink and a door, labeled '1 / 5'. Below this is a thumbnail strip with five small images. To the right of the image is a 'Status' section showing 'Available' with a green dot, 'AVAILABLE FROM 1/10/2025', and creation/update dates. There's also a 'Landlord Contact' section with name 'Mahir Tajwar Rahman' and phone '01315351504', and a red 'REPORT LISTING' button. Below the main image is a summary box for a '3 Bedroom Flat Tolet / Rent from August for Family in Savar, Dhaka' located at '11/6, Banasree, Kotwali, Dhaka, Dhaka'. It lists the price as '৳20,000 (Negotiable)', deposit as '৳12000', service charge as '৳1500', and includes icons for bed, bath, and parking. At the bottom left is a 'Property Facts' section.

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:

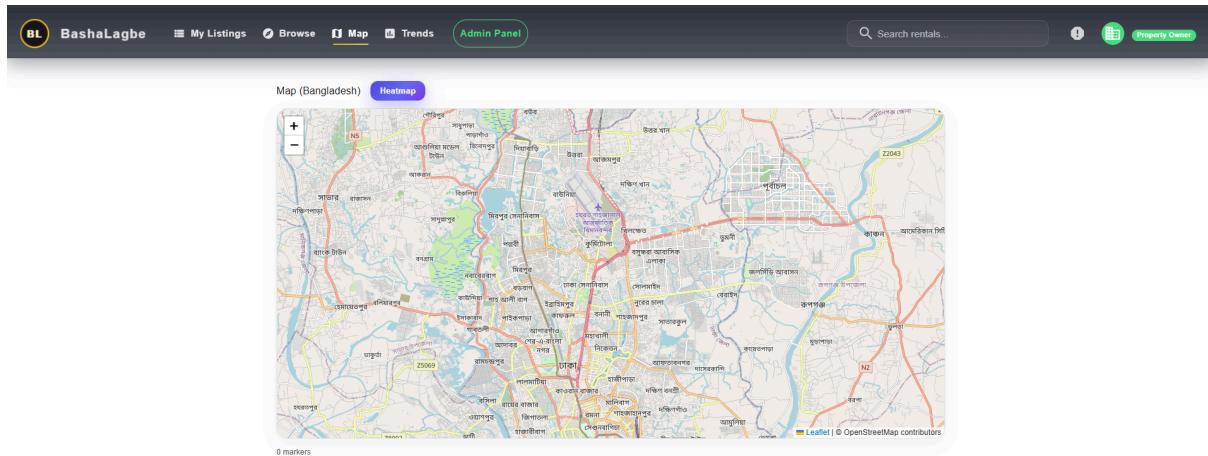
- Title and Address prominently displayed
- Pricing details with monthly rent and additional charges
- Property Facts grid with icons: Type, Rooms, Bathrooms, etc.
- Description with "Read more" expansion
- Interactive map showing exact location

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

Add Listing

BASIC ADDRESS MEDIA PRICING DETAILS CONTACT

Title * Apartment

Available From * mm/dd/yyyy

Rooms *

Bathrooms

Balcony

Person Count

Features (comma separated)

Description

SAVE CANCEL

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

The screenshot shows the 'Edit Listing' interface. At the top, there are tabs for BASIC, ADDRESS, MEDIA, PRICING, DETAILS, and CONTACT. The BASIC tab is selected. The form fields include:

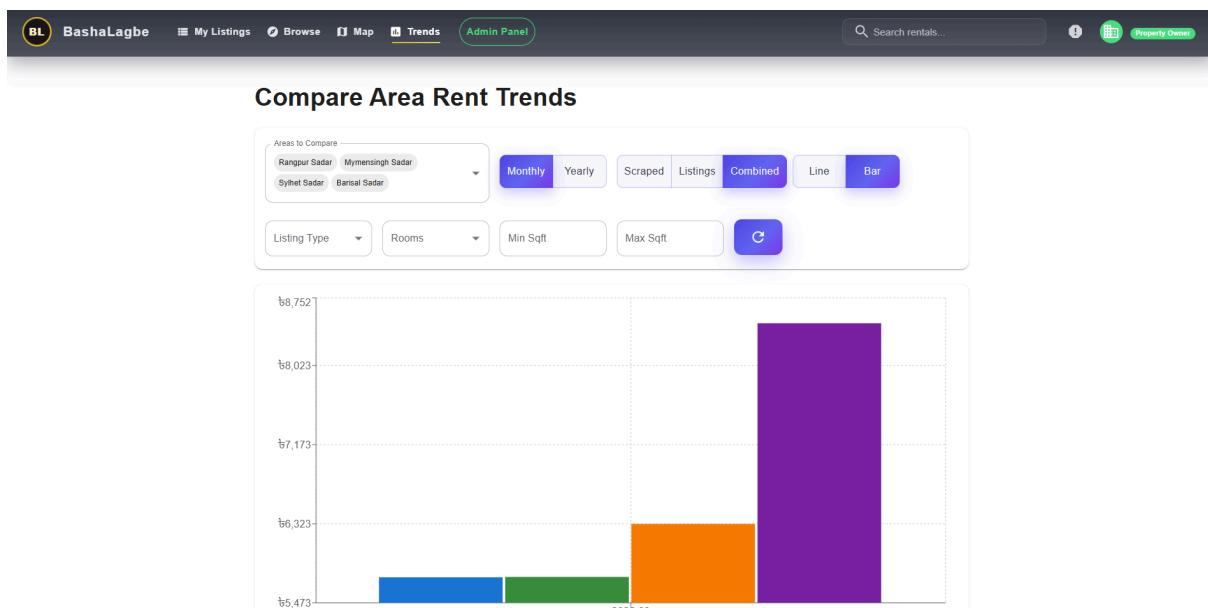
- Title: Luxury Flat in Gulshan
- Listing Type: Apartment
- Available From: 12/02/2001
- Rooms: 1
- Bathrooms: 1
- Balcony: 2
- Person Count: 4
- Features: water
- Description: best home

At the bottom, there are 'SAVE' and 'CANCEL' buttons.

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

The screenshot shows the BashaLagbe website's user profile management interface. At the top, there is a navigation bar with links for 'My Listings', 'Browse', 'Map', 'Trends', and 'Admin Panel'. A search bar is also present. On the right side of the header, there is a user profile icon for 'Mahir Tajwar Rahman' with the role 'Property Owner'. Below the header, the main content area is titled 'My Profile'. It features a circular placeholder for the user's profile picture, which is currently empty. To the right of this placeholder, the user's name 'Mahir Tajwar Rahman' and role 'Property Owner' are displayed. The 'Account Information' section contains three input fields: 'Name' (Mahir Tajwar Rahman), 'Email' (mahir19800@gmail.com), and 'Role' (Property Owner). Below these fields, a note states 'Account Type: Property Owner' followed by a descriptive message: 'As a property owner, you can create and manage rental listings.' At the bottom right of the profile page, there is a 'Save Changes' button.

- 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

The screenshot shows the 'Submit a Report' page. At the top, there's a navigation bar with links for 'My Listings', 'Browse', 'Map', 'Trends', 'Admin Panel', and a search bar. On the right, there are user icons for 'Property Owner' and 'Asifur Islam'. Below the navigation, the page title is 'Home / Submit Report'.

The main form area has a heading 'Submit a Report' with a purple icon. It includes a note: 'Help us maintain a safe and trustworthy community by reporting inappropriate content or behavior.' There are two radio buttons for 'What would you like to report?': 'Listing' (selected) and 'User'. A text input field for 'Listing Title' contains the text '3 Bedroom Flat Tolet / Rent from August for Family in Savar, Dhaka'. A dropdown menu for 'Reason for Report' is open. A text area for 'Additional Details (Optional)' is empty. Below the form, there's a section for 'Add Proof (Optional)' with a note about file uploads and a 'UPLOAD FILES' button.

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

The screenshot shows the 'My Reports' page. The navigation bar and user icons are identical to the previous screenshot. The page title is 'Home / My Reports'.

The main content area has a heading 'My Reports' with a purple icon. It includes a note: 'Here you can view all the reports you've submitted for both listings and users. Our moderation team reviews each report carefully and takes appropriate action when necessary.' A 'SUBMIT NEW REPORT' button is visible. Below this, a table lists submitted reports:

Type	Target	Reason	Status	Submitted	Admin Action	Actions
Listing	Mofij er basha ID: 68aab5946283f13549b7d433	Offensive Content	Valid	9/7/2025 10:26:28 PM	User Warned	<button>VIEW</button>

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

The screenshot displays the Lighthouse performance audit report for the URL <https://bashalagbe-7se3.onrender.com/listing/68c4015cc7824ce25ec7d33d>. The overall score is 96, with 100s in Accessibility, Best Practices, and SEO, and 96 in Performance.

Performance Metrics:

- First Contentful Paint: 0.6 s
- Total Blocking Time: 90 ms
- Speed Index: 1.3 s
- Largest Contentful Paint: 1.1 s
- Cumulative Layout Shift: 0

Insights:

- Network dependency tree
- Render blocking requests
- LCP breakdown

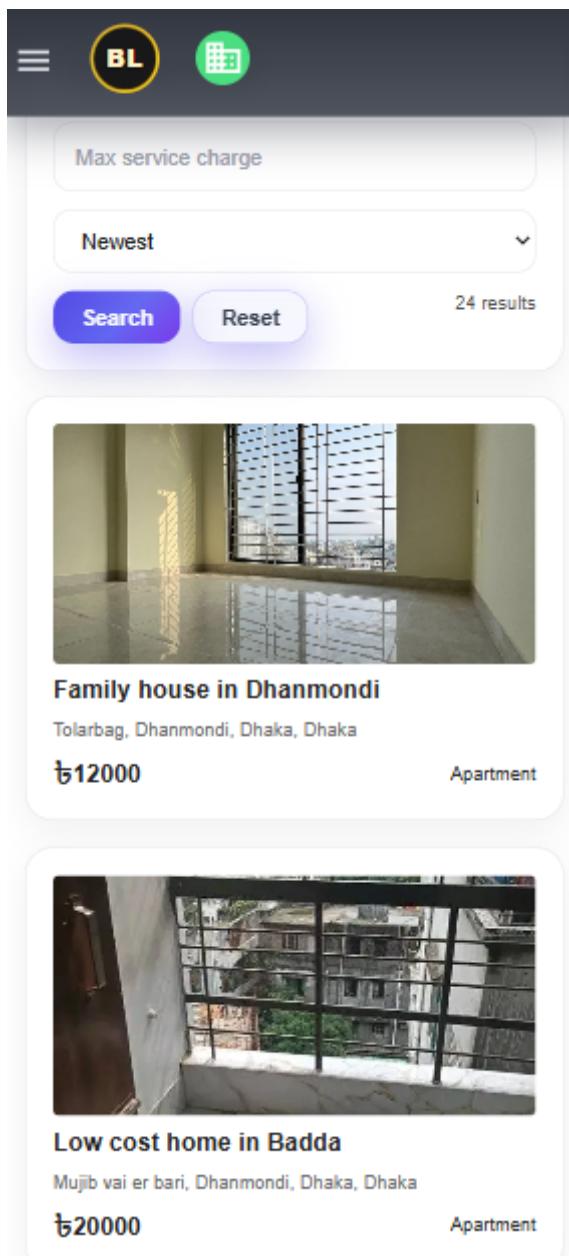
Diagnostics:

- Reduce unused JavaScript — Est savings of 234 KiB
- Reduce unused CSS — Est savings of 11 KiB
- Avoid long main-thread tasks — 2 long tasks found

More information about the performance of your application. These numbers don't directly affect the Performance score.

PASSED AUDITS (27) Show

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 by area, 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:	
<p>1. User Registration & Login Users can register and log in using email/password or third-party OAuth providers (e.g., Google).</p>	
<p>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.</p>	
<p>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.</p>	
<p>4. Listing Comparison Tool Users can select and compare 2–3 listings side-by-side to assess features like rent, size, amenities, and location.</p>	

Group member - 04	
Name: Fardin Kamran	Student ID: 21101023
Functional Requirements which are developed by this member:	
<p>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.</p>	
<p>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.</p>	
<p>3. Wishlist Users can save listings to a personal wishlist, organize them into folders, and get alerts for price changes or availability updates.</p>	

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.