Real Estate Management System

August 17, 2025

1 Problem Description and Requirements

1.1 Problem Statement

Real estate management involves intricate processes including property sales, rentals, owner and client management, billing, and maintenance tracking. Many current systems suffer from fragmented functionalities, leading to inefficiencies, manual errors, and a disjointed user experience. This web-based solution seeks to offer an integrated platform, optimizing workflows for property managers, owners, and tenants while adapting to modern real estate demands.

1.2 Key Business Requirements

- **Property Management:** Oversee properties for sale or rent, capturing details such as type (sale/rent), location, price, availability, and multimedia assets (photos, videos, virtual tours).
- **Owner and Client Management:** Maintain comprehensive profiles for owners and clients (buyers/tenants) with contact details, preferences, transaction history, and feedback ratings.
- **Billing for Rentals:** Automate invoice creation, dispatch payment reminders via email/SMS, track statuses, and support diverse payment methods (e.g., credit card, bank transfer, digital wallets).
- **Repair Management:** Record repair requests with timestamps, assign tasks to maintenance teams, provide status updates, and include estimated completion timelines and cost estimates.
- **Financial Tracking:** Track rental income, maintenance costs, and generate exportable financial reports (PDF, CSV, Excel) with real-time currency conversion.
- **User Authentication and Authorization:** Ensure secure access with role-based permissions (admin, owner, tenant, maintenance) and multi-factor authentication (MFA).
- **Responsive UI:** Deliver a seamless interface across desktops, tablets, and mobiles with adaptive layouts and accessibility features (e.g., screen reader support).

- **Performance and Scalability:** Handle high concurrency and large datasets with optimized queries, caching, and cloud-based scaling (e.g., AWS Elastic Beanstalk).
- **Data Security:** Encrypt data at rest and in transit (AES-256, TLS 1.3), conduct quarterly security audits, and comply with GDPR, CCPA, and local data protection laws.

1.3 Quality Attributes

- **Usability:** User-friendly design with interactive tutorials, multilingual interfaces (e.g., English, Spanish, Mandarin), and customizable dashboards.
- **Reliability:** Target 99.95% uptime with automated daily backups, failover systems, and comprehensive error logging.
- **Scalability:** Support thousands of users and properties with auto-scaling groups and distributed databases.
- **Security:** Implement zero-trust architecture, regular penetration testing, and data anonymization for analytics.
- **Maintainability:** Adopt a microservices architecture with CI/CD pipelines, detailed API documentation, and 80%+ test coverage.

1.4 Suggested Additional Features

- **AI-Powered Suggestions:** Utilize machine learning to recommend properties based on client behavior and predict maintenance needs with historical data analysis.
- **Automated Notifications:** Send proactive alerts for due payments, repair completions, or market trends via email, SMS, or push notifications.
- **Analytics Dashboard:** Offer interactive visualizations of income trends, vacancy rates, and ROI, with drill-down capabilities.
- **Document Management:** Provide secure storage for contracts and leases with e-signature integration and audit trails.
- **Chat Support:** Embed a 24/7 AI chatbot with human escalation options for tenant-owner interactions.

2 Domain & Data Modeling

2.1 ER Diagram

The ER diagram should feature core entities:

- **Property**, **Person (Owner/Client)**, **Invoice**, **RepairRequest**, and **Payment**.
- **Relationships:**

  - Property-Person: One-to-many (one person can own multiple properties).
  - Property-Person: Many-to-many via **Lease** for tenants.
  - Invoice-Payment: One-to-many.

- **Tools:** Utilize Lucidchart or Draw.io for diagram creation and integration.

2.2 Principal Domain Concepts

2.2.1 Actors

- **Admin:** Manages system settings, user accounts, and performance metrics.
- **Owner:** Inherits from Person; manages property listings and financials.
- **Client (Tenant/Buyer):** Inherits from Person; engages in leasing or purchasing.
- **Maintenance Team:** Handles repair workflows.

2.2.2 Processes

- Property listing, editing, and status updates.
- Lease creation, extension, and termination with legal compliance checks.
- Automated billing, payment reconciliation, and late fee application.
- Repair request lifecycle from submission to closure.
- Real-time financial reporting and forecasting.

2.2.3 Data Objects

- **Person:** ID, name, email, phone, contact_preference, created_at.
- **Owner:** Extends Person; properties_owned (array), financial_summary.
- **Client:** Extends Person; preferences, lease_history (array), transaction_log.
- **Property:** ID, type (sale/rent), address, price, status, media_urls.
- **Invoice:** ID, amount, due_date, status, client_id.
- **RepairRequest:** ID, property_id, description, status, assigned_team, estimated_cost.
- **Payment:** ID, invoice_id, amount, date, method.

2.3 Technology-Oriented Schema

- **PostgreSQL** for relational data with inheritance.
- **MongoDB** for unstructured content.

2.3.1 PostgreSQL Schema

Listing 1: PostgreSQL Schema

```sql
CREATE TABLE Person (
    person_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(20),
    contact_preference VARCHAR(50) CHECK (contact_preference IN (
        'email', 'phone', 'both')),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Owners (
    CHECK (true)
) INHERITS (Person);

ALTER TABLE Owners ADD COLUMN properties_owned INT[] NOT NULL
   DEFAULT '{}';
ALTER TABLE Owners ADD COLUMN financial_summary JSONB;

CREATE TABLE Clients (
    CHECK (true)
) INHERITS (Person);

ALTER TABLE Clients ADD COLUMN preferences JSONB;
ALTER TABLE Clients ADD COLUMN lease_history INT[] NOT NULL
   DEFAULT '{}';
ALTER TABLE Clients ADD COLUMN transaction_log JSONB;

CREATE TABLE Properties (
    property_id SERIAL PRIMARY KEY,
    owner_id INT REFERENCES Owners(person_id),
    type VARCHAR(20) CHECK (type IN ('sale', 'rent')),
    address VARCHAR(200),
    price DECIMAL(10, 2),
    status VARCHAR(20) CHECK (status IN ('available', 'occupied',
        'sold')),
    media_urls TEXT[],
    INDEX idx_property_status (status)
);

CREATE TABLE Leases (
    lease_id SERIAL PRIMARY KEY,
    property_id INT REFERENCES Properties(property_id),
    client_id INT REFERENCES Clients(person_id),
    start_date DATE,
    end_date DATE,
    monthly_rent DECIMAL(10, 2)
);

```

```sql
CREATE TABLE Invoices (
    invoice_id SERIAL PRIMARY KEY,
    lease_id INT REFERENCES Leases(lease_id),
    amount DECIMAL(10, 2),
    due_date DATE,
    status VARCHAR(20) CHECK (status IN ('pending', 'paid', '
        overdue')),
    INDEX idx_invoice_status (status)
);

CREATE TABLE RepairRequests (
    repair_id SERIAL PRIMARY KEY,
    property_id INT REFERENCES Properties(property_id),
    description TEXT,
    status VARCHAR(20) CHECK (status IN ('open', 'in_progress', '
        closed')),
    assigned_team VARCHAR(100),
    estimated_cost DECIMAL(10, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE Payments (
    payment_id SERIAL PRIMARY KEY,
    invoice_id INT REFERENCES Invoices(invoice_id),
    amount DECIMAL(10, 2),
    payment_date TIMESTAMP,
    method VARCHAR(50)
);
```

2.3.2 MongoDB Schema (for documents)

Listing 2: MongoDB Schema

```
{
  "property_id": Number,
  "documents": [
    {
      "doc_id": String,
      "type": String,
      "url": String,
      "uploaded_at": ISODate,
      "version": Number
    }
  ],
  "images": [
    {
      "image_id": String,
      "url": String,
      "description": String,
      "caption": String
    }
  ]
}
```

2.3.3 Sample React Component

Listing 3: Sample React Component

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

interface Property {
  id: number;
  address: string;
  price: number;
  type: 'sale' | 'rent';
  status: 'available' | 'occupied' | 'sold';
  media_urls: string[];
}

const PropertyList: React.FC = () => {
  const [properties, setProperties] = useState<Property[]>([]);

  useEffect(() => {
    axios
      .get('/api/properties')
      .then((response) => setProperties(response.data))
      .catch((error) => console.error('Error fetching properties
         :', error));
  }, []);

  return (
    <div className="container mx-auto p-4">
      {properties.map((property) => (
        <div key={property.id} className="border p-4 mb-4 rounded
            -lg shadow hover:shadow-lg transition">
          <h3 className="text-xl font-bold">{property.address}</
             h3>
          <p>Price: ${property.price.toLocaleString()}</p>
          <p>Type: {property.type}</p>
          <p>Status: {property.status}</p>
          {property.media_urls[0] && <img src={property.
             media_urls[0]} alt={property.address} className="mt
             -2 w-full h-32 object-cover rounded" />}
        </div>
      ))}
    </div>
  );
};

export default PropertyList;
```

3 Front-End Design

3.1 Framework Choice

- **React** with **TypeScript** for robust component development.
- **Tailwind CSS** with dark mode support for responsive styling.

3.2 Mock-ups/Wireframes

- **Home Page:** Search bar with filters, map integration, and featured listings.
- **Property Details:** Gallery view, detailed specs, and application button.
- **Dashboard:**
  - **Owner:** Property overview, income charts, repair logs.
  - **Tenant:** Lease summary, payment history, repair request form.
  - **Admin:** User roles, system health, analytics.
- **Responsive Design:** Mobile-first with swipe gestures and voice command support.
- **Tools:** Figma for wireframes, including a multi-device preview.

3.3 Improvements Made

1. **Clarity and Structure:** Streamlined sections with actionable headings and prioritized requirements.
2. **Technical Enhancements:** Added `created_at` to `Person`, `media_urls` to `Properties`, and `estimated_cost` to `RepairRequests` for better tracking.
3. **Usability:** Included multilingual support, voice commands, and dark mode in UI design.
4. **Security:** Upgraded to zero-trust and quarterly audits, reflecting 2025 security standards.
5. **Scalability:** Incorporated auto-scaling and distributed databases for modern cloud demands.
6. **Front-End:** Enhanced React component with image previews and hover effects for better engagement.
7. **Data Modeling:** Expanded MongoDB schema with `version` and `caption` for document and image management.