



DLF Property Management System - Project Documentation

Done by:
A. Praneeth Ram (CSE)

Table of Contents

1. Project Overview
 2. System Architecture & Technology Stack
 3. Frontend Application (React)
 4. Backend API Service (FastAPI)
 5. Database Schema (PostgreSQL)
 6. API Endpoints
 7. User Flows
 8. File Storage with Amazon S3
 9. Deployment & Infrastructure
 10. Team Responsibilities
-

1. Project Overview

Business Problem

DLF Limited needs a centralized property ownership and document management system to handle 100,000+ properties and 50,000+ users with secure access, ownership management, and enquiry handling.

Key Requirements

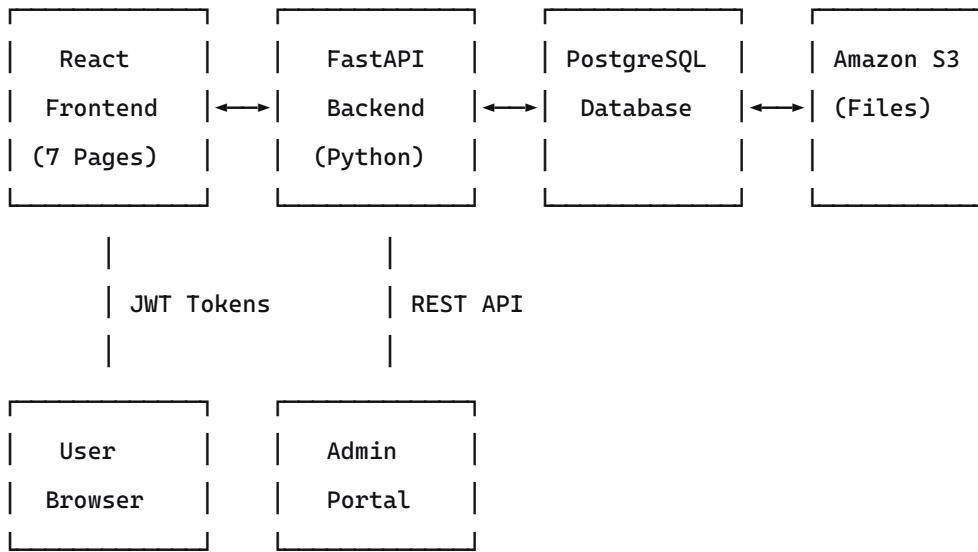
- Secure user authentication and authorization
- Property browsing and search (post-authentication)
- Secure document storage with Amazon S3
- Internal enquiry system between users and admins
- Ownership transfer functionality (admin-only)
- Scalable architecture to handle 100k+ properties efficiently.

Tech Stack

- **Frontend:** React
- **Backend:** Python FastAPI
- **Database:** PostgreSQL
- **File Storage:** Amazon S3
- **Authentication:** JWT + bcrypt
- **Deployment:** Docker

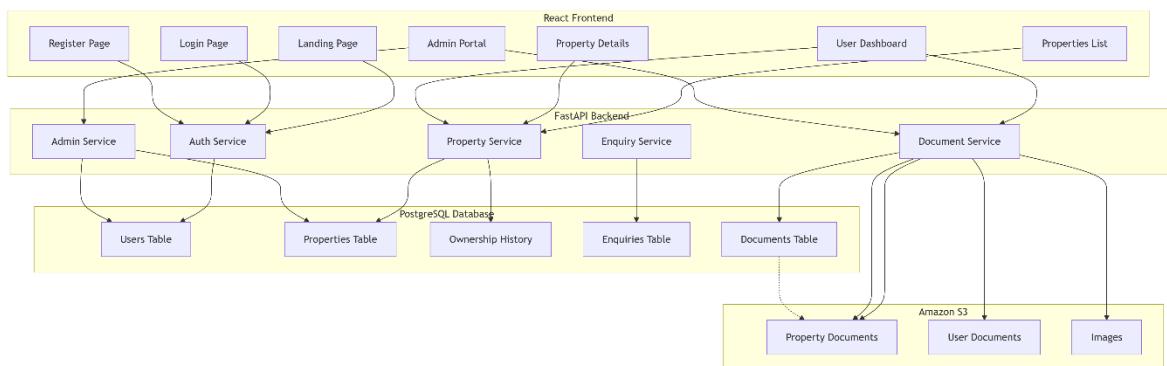
2. System Architecture

High-Level Architecture

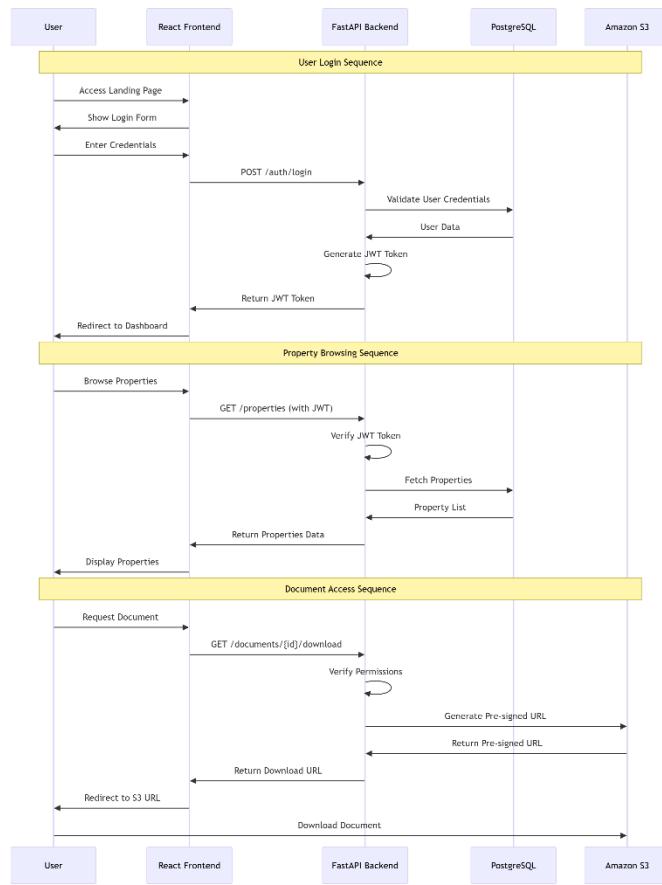


Diagrammatic Explanation of DLF project:

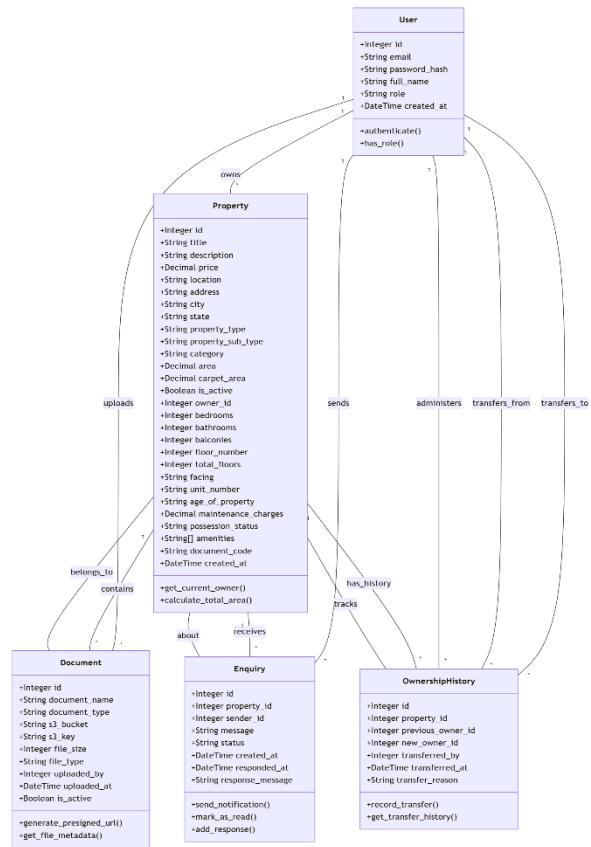
Component Diagram



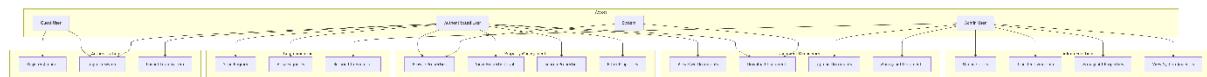
2. Sequence Diagram (User Login & Property Access)



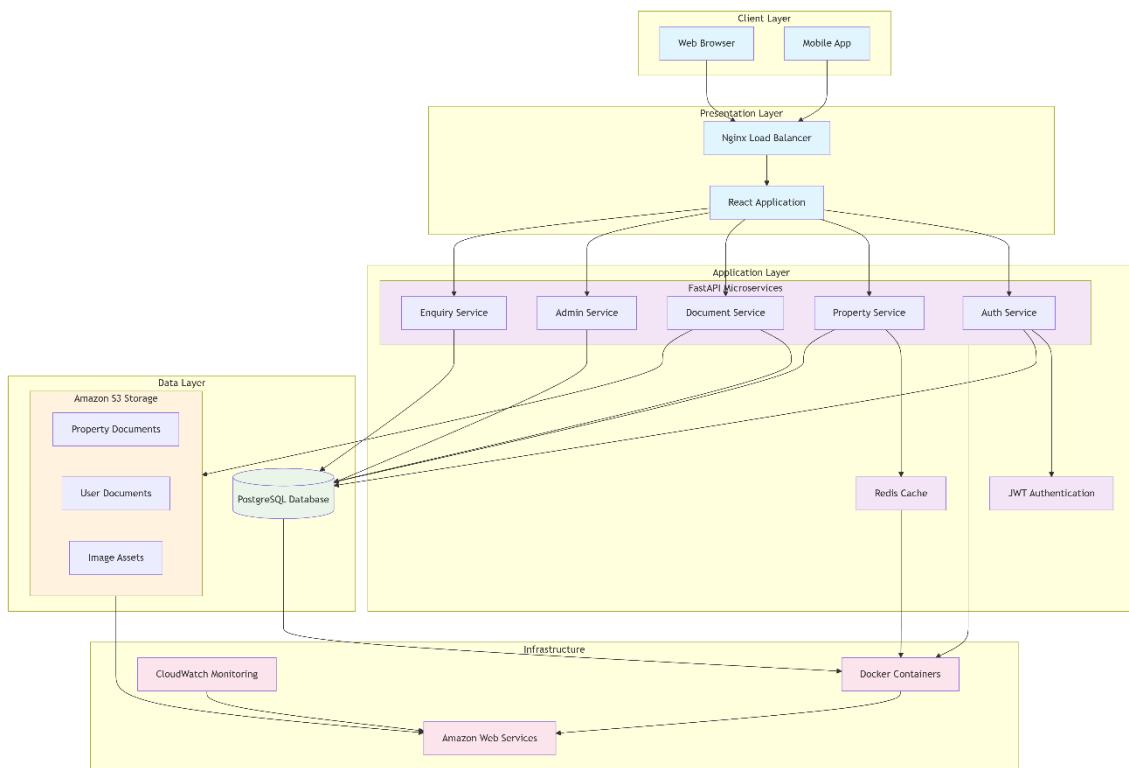
3. Class Diagram



4. Use Case Diagram



5. System Architecture Diagram



Component Responsibilities

- **Frontend (React):** Serves the user interface with 7 main routes, manages application state, and handles user interactions
- **Backend (FastAPI):** Provides RESTful API endpoints, business logic, authentication, and file handling
- **Database (PostgreSQL):** Stores all structured data including users, properties, ownership records, and enquiries
- **File Storage (Amazon S3):** Securely stores property documents, images, and user files with proper access control

⌚ 3. Frontend Application (React)

7 Page Structure

Page 1: / - Landing Page

- **Purpose:** Entry point for all users
- **Features:**
 - Hero section with project description
 - Login/Register buttons
 - No property browsing (requires login)
- **Access:** Public

Page 2: /auth/login - Login Page

- **Purpose:** User and admin authentication
- **Features:**
 - Email/password form
 - Role detection (user/admin)
 - Redirect to appropriate portal
- **Access:** Public

Page 3: /auth/register - Registration Page

- **Purpose:** New user account creation
- **Features:**
 - Registration form with validation
 - Password strength indicator
 - Terms acceptance
- **Access:** Public

Page 4: /properties - Property Listings

- **Purpose:** Browse all properties
- **Features:**
 - Search and filter 100,000+ properties
 - Pagination for performance
 - Property cards with basic info
- **Access:** Authenticated users only

Page 5: /properties/[id] - Property Details

- **Purpose:** View individual property information
- **Features:**
 - Detailed property view
 - Image gallery (served from S3)
 - "Send Enquiry" button
- **Access:** Authenticated users only

Page 6: /dashboard - User Portal

- **Purpose:** Personal management dashboard
- **Features (Tabbed Interface):**
 - My Properties: Owned properties list
 - My Documents: Access to S3-stored documents
 - My Enquiries: Sent/received messages
- **Access:** Authenticated users only

Page 7: /admin - Admin Portal

- **Purpose:** System administration
- **Features (Tabbed Interface):**
 - All Properties: Manage 100k+ properties
 - Transfer Ownership: Critical feature (2 fields only)
 - All Users: User management
 - Document Management: Upload/Manage files in S3
- **Access:** Admin users only

4. Backend API Service (FastAPI)

File Structure

```
backend/
└── app/
    ├── main.py          # FastAPI app setup
    ├── auth.py          # JWT authentication
    ├── database.py      # PostgreSQL connection
    ├── models.py         # SQLAlchemy models
    ├── schemas.py        # Pydantic schemas
    ├── crud.py           # Database operations
    ├── config.py         # Environment config
    ├── s3_client.py      # Amazon S3 file operations
    └── routes/
        ├── auth.py
        ├── properties.py
        ├── enquiries.py
        ├── documents.py
        └── admin.py
    └── scripts/
        ├── seed_data.py    # Fake data generation
        └── database_init.py # DB setup
requirements.txt
```

Key Backend Features

- **JWT Authentication:** Secure token-based authentication system
- **Role-Based Access Control:** User/Admin permission management
- **Amazon S3 Integration:** Secure file upload, download, and management
- **API Validation:** Pydantic schema validation for all requests
- **Error Handling:** Comprehensive error responses and logging

5. Database Schema (PostgreSQL)

Core Tables

- **users:** User accounts and profiles
- **properties:** Property details and information
- **ownership_history:** Property ownership tracking
- **enquiries:** User enquiry messages
- **documents:** File metadata and S3 references

Relationships

- One-to-Many: User → Properties (owned)
- One-to-Many: User → Enquiries (sent)
- One-to-Many: Property → Documents
- Many-to-Many: Property → Users (through ownership_history)

5 Table Schema

1. Users Table

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(255) NOT NULL,
    role VARCHAR(20) CHECK (role IN ('user', 'admin')) DEFAULT 'user',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. Properties Table

```
CREATE TABLE properties (
    id SERIAL PRIMARY KEY,
    title VARCHAR(500) NOT NULL,
    description TEXT,
    price DECIMAL(15,2),
    location VARCHAR(255),
    address TEXT NOT NULL,
    city VARCHAR(100) NOT NULL,
    state VARCHAR(100) NOT NULL,
    property_type VARCHAR(50),
    property_sub_type VARCHAR(50),
    category VARCHAR(50),
    area DECIMAL(10,2),
    carpet_area DECIMAL(10,2),
    is_active BOOLEAN DEFAULT true,
    owner_id INTEGER NOT NULL,
    bedrooms INTEGER,
    bathrooms INTEGER,
```

```

balconies INTEGER,
floor_number INTEGER,
total_floors INTEGER,
facing VARCHAR(50),
unit_number VARCHAR(50),
age_of_property VARCHAR(50),
maintenance_charges DECIMAL(10,2),
possession_status VARCHAR(50),
amenities TEXT[],
document_code VARCHAR(100),
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
FOREIGN KEY (owner_id) REFERENCES users(id)
);

```

3. Documents Table (DEMO Mode)

```

CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    property_id INTEGER NOT NULL,
    document_name VARCHAR(255) NOT NULL,
    document_type VARCHAR(50) CHECK (document_type IN ('deed', 'tax_receipt', 'agreement', 'plan', 'certificate')),
    s3_bucket VARCHAR(255) NOT NULL,
    s3_key VARCHAR(500) NOT NULL,
    file_size INTEGER,
    file_type VARCHAR(100),
    uploaded_by INTEGER NOT NULL,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    is_active BOOLEAN DEFAULT true,
    FOREIGN KEY (property_id) REFERENCES properties(id),
    FOREIGN KEY (uploaded_by) REFERENCES users(id)
);

```

4. Enquiries Table

```

CREATE TABLE enquiries (
    id SERIAL PRIMARY KEY,
    property_id INTEGER NOT NULL,
    sender_id INTEGER NOT NULL,
    message TEXT NOT NULL,

```

```

    status VARCHAR(20) CHECK (status IN ('unread', 'read', 'replied')) DEFAULT 'un
read',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    responded_at TIMESTAMP,
    response_message TEXT,
    FOREIGN KEY (property_id) REFERENCES properties(id),
    FOREIGN KEY (sender_id) REFERENCES users(id)
);

```

5. Ownership History Table

```

CREATE TABLE ownership_history (
    id SERIAL PRIMARY KEY,
    property_id INTEGER NOT NULL,
    previous_owner_id INTEGER NOT NULL,
    new_owner_id INTEGER NOT NULL,
    transferred_by INTEGER NOT NULL,
    transferred_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    transfer_reason TEXT,
    FOREIGN KEY (property_id) REFERENCES properties(id),
    FOREIGN KEY (previous_owner_id) REFERENCES users(id),
    FOREIGN KEY (new_owner_id) REFERENCES users(id),
    FOREIGN KEY (transferred_by) REFERENCES users(id)
);

```

🌐 6. API Endpoints

Authentication

- POST /auth/register - User registration
- POST /auth/login - User login
- POST /auth/logout - User logout

Properties

- GET /properties - List all properties (with pagination)
- GET /properties/{id} - Get property details
- POST /properties/{id}/enquiry - Send enquiry

Documents

- `GET /documents` - List user documents
- `GET /documents/{id}/download` - Generate S3 pre-signed URL
- `POST /admin/documents/upload` - Upload document to S3 (admin)

Admin

- `POST /admin/ownership/transfer` - Transfer property ownership
 - `GET /admin/users` - List all users
 - `GET /admin/properties` - Manage properties
 - `properties` - Get all properties
-



7. User Flows

User Registration Flow

1. Landing Page → Register → Email/Password → Account Created → Login → User Dashboard

Property Browsing Flow

1. Login → Properties Page → Search/Filter → Property Details → Send Enquiry

Document Access Flow

1. User Dashboard → My Documents → View List → Click Document → S3 Pre-signed URL → Download

Admin Ownership Transfer

1. Admin Login → Admin Portal → Transfer Ownership → Select Property & User → Confirm Transfer
-

8. File Storage with Amazon S3

S3 Bucket Structure

```
s3://dlf-property-documents/
└── properties/
    ├── {property_id}/
    │   ├── deeds/
    │   ├── agreements/
    │   └── images/
└── users/
    ├── {user_id}/
    │   ├── identification/
    │   └── contracts/
```

Security Features

- **IAM Roles:** Secure access credentials
- **Bucket Policies:** Restrict public access
- **Pre-signed URLs:** Time-limited download links
- **Encryption:** Server-side encryption for all files

File Operations

- **Upload:** Admin users can upload documents via secure API
- **Download:** Users get pre-signed URLs for secure file access
- **Metadata:** All file information stored in PostgreSQL for tracking

9. Deployment & Infrastructure

Docker Configuration

- **Frontend Container:** React application served via Nginx
- **Backend Container:** FastAPI application with Uvicorn
- **Database Container:** PostgreSQL with persistent volume
- **Reverse Proxy:** Nginx for routing and SSL termination

Environment Setup

- **Development:** Local Docker Compose with mock S3
- **Staging:** AWS environment with test S3 bucket
- **Production:** AWS ECS/EKS with production S3 bucket

AWS Services Integration

- **Amazon S3:** File storage and document management

- **IAM:** Secure access management for S3
- **CloudWatch:** Application monitoring and logging

10. Team Responsibilities

Frontend Team (React)

- Implement 7-page application structure
- State management and API integration
- Responsive UI/UX design

Backend Team (FastAPI)

- REST API development and documentation
- Authentication and authorization system
- PostgreSQL database design and optimization
- Amazon S3 integration and file handling

Deployment Team

- Docker containerization and orchestration
- AWS infrastructure setup (S3, ECS, etc.)

QA Team

- API endpoint testing
- User flow validation
- File upload/download testing
- Performance testing with 100k+ properties