Design and Implementation of House Rent Management System

A project report submitted to the Department of Computer Science and Engineering, Jahangirnagar University, in partial fulfilment of the requirements for the degree of Master of Science (M.Sc.) in Computer Science.

Submitted By:
Sabbir Ahmed Chowdhury
Student ID: CSE202401008
Session: Spring 2024
Batch: 34

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342.
Banagladesh

November 2025

Abstract

The proliferation of digital technologies has revolutionized various sectors, including real estate management. Traditional house rental management processes are often characterized by inefficiencies, manual record-keeping, and communication gaps between landlords and tenants. This research project presents the design and implementation of a comprehensive web-based House Rent Management System (HRMS) that addresses these challenges through modern software engineering practices.

The system is developed using a three-tier architecture comprising a React-based frontend, an ASP.NET Core Web API backend, and a SQL Server database. The solution provides distinct interfaces for three primary user roles: landlords, tenants, and administrators. Key functionalities include property listing and management, lease agreement generation and tracking, automated payment processing, maintenance request handling, and comprehensive reporting capabilities.

The implementation follows industry-standard design patterns including Repository Pattern, Unit of Work, Dependency Injection, and RESTful API principles. Security is ensured through JWT-based authentication, role-based authorization, and password hashing. The system incorporates automated background services for payment generation and reminder notifications, enhancing operational efficiency.

Extensive testing was conducted to validate system functionality, security, and performance. The results demonstrate that the system successfully automates rental management workflows, reduces administrative overhead, and improves communication between stakeholders. The solution provides a scalable foundation for future enhancements including mobile application support, advanced analytics, and integration with payment gateways.

Table of Contents

Chapter 1

Introduction

1.1 Background

The real estate rental market has experienced significant growth globally, with increasing numbers of property owners seeking efficient methods to manage their rental properties. Traditional rental management approaches rely heavily on manual processes, paper-based documentation, and face-to-face interactions, which are time-consuming, error-prone, and difficult to scale.

In contemporary urban environments, landlords often manage multiple properties with diverse tenant requirements. Similarly, tenants seek convenient platforms to search for properties, manage lease agreements, and handle rental payments. The absence of integrated digital solutions creates challenges including delayed communication, payment tracking difficulties, maintenance request management issues, and lack of transparency in rental transactions.

The advancement of web technologies and cloud computing has created opportunities to develop comprehensive digital solutions that streamline rental management processes. Modern web applications can provide real-time access to information, automated workflows, and seamless communication channels, thereby addressing the limitations of traditional approaches.

1.2 Problem Statement

The current rental management landscape faces several critical challenges:

- **Inefficient Property Management:** Landlords struggle to maintain accurate records of multiple properties, their availability status, and tenant information using traditional methods.
- **Manual Lease Agreement Processing:** The creation, storage, and management of lease agreements typically involve manual document preparation, physical storage, and difficulty in tracking lease terms and renewal dates.
- **Payment Tracking Difficulties:** Both landlords and tenants face challenges in tracking rental payments, due dates, overdue amounts, and payment history. Manual record-keeping increases the risk of errors and disputes.

- **Maintenance Request Management:** Tenants often experience delays in reporting maintenance issues, while landlords struggle to prioritize and track maintenance requests across multiple properties.
- **Communication Gaps:** Lack of centralized communication channels leads to delayed responses, miscommunication, and difficulty in maintaining records of interactions between landlords and tenants.
- **Limited Transparency:** Both parties often lack visibility into rental history, payment records, and property maintenance history, leading to trust issues and disputes.

1.3 Research Objectives

The primary objective of this research is to design and implement a comprehensive web-based House Rent Management System that addresses the aforementioned challenges. Specific objectives include:

- To develop a secure, scalable, and user-friendly web application for managing rental properties
- To automate lease agreement generation and management processes
- To implement an efficient payment tracking and management system
- To create a streamlined maintenance request handling mechanism
- To provide role-based access control ensuring data security and privacy
- To evaluate the system's effectiveness in improving rental management workflows

1.4 Scope and Limitations

1.4.1 Scope:

- Web-based application accessible through modern browsers
- Support for three user roles: Landlord, Tenant, and Administrator
- Property listing, search, and management functionalities
- Lease agreement creation, tracking, and document generation
- Payment record management with automated generation
- Maintenance request submission and tracking
- User authentication and authorization
- Basic reporting capabilities

1.4.2 Limitations:

- The system does not include direct payment gateway integration
- Mobile application support is not included in the current implementation
- Advanced analytics and business intelligence features are limited
- Multi-language support is not provided
- Integration with external property listing services is not included

1.5 Report Organization

This report is organized into several chapters. Chapter 2 presents a review of relevant literature and existing solutions. Chapter 3 details the system requirements and analysis. Chapter 4 describes the system design architecture and components. Chapter 5 provides comprehensive implementation details. Chapter 6 discusses testing methodologies and results. Chapter 7 presents the findings and discussion. Finally, Chapter 8 concludes the report with future work recommendations.

Chapter 2

Literature Review

2.1 Evolution of Property Management Systems

Property management systems have evolved significantly over the past decades. Early systems were primarily desktop-based applications with limited connectivity and functionality. The advent of web technologies enabled the development of cloud-based solutions that offer greater accessibility and scalability.

Modern property management systems leverage various technologies including cloud computing, mobile applications, and artificial intelligence to enhance functionality. Research indicates that web-based property management solutions can reduce administrative costs by up to 40% while improving tenant satisfaction.

2.2 Web Application Architecture Patterns

Contemporary web applications typically follow architectural patterns that separate concerns and promote maintainability. The Model-View-Controller (MVC) pattern and its variations are widely adopted in web development. RESTful API architecture has become the standard for building scalable and interoperable web services (Fielding, 2000).

The three-tier architecture, comprising presentation, business logic, and data layers, provides clear separation of concerns and enables independent scaling of components. This architecture pattern is particularly suitable for enterprise applications requiring high availability and performance (Tanenbaum & Van Steen, 2017).

2.3 Security in Web Applications

Security is paramount in web applications handling sensitive user data and financial transactions. Authentication and authorization mechanisms are critical components. JWT (JSON Web Tokens) has emerged as a popular standard for stateless authentication in RESTful APIs.

Role-based access control (RBAC) provides fine-grained permission management, ensuring users can only access resources appropriate to their roles. Password hashing using algorithms like BCrypt protects user credentials even in the event of data breaches (Provos & Mazières, 1999).

2.4 Database Design for Property Management

Effective database design is crucial for property management systems. Normalized database schemas reduce data redundancy and ensure data integrity. Entity-Relationship modeling helps in understanding relationships between different entities such as properties, leases, payments, and users (Elmasri & Navathe, 2016).

Modern database systems provide features such as transactions, referential integrity, and indexing that enhance data consistency and query performance. Object-Relational Mapping (ORM) frameworks like Entity Framework Core simplify database interactions while maintaining type safety (Microsoft, 2023).

2.5 Automated Workflow Management

Automation plays a crucial role in reducing manual effort and improving efficiency in property management systems. Background services can handle scheduled tasks such as payment generation, reminder notifications, and report generation (Hohpe & Woolf, 2004).

Research demonstrates that automated workflows can significantly reduce processing time and human errors in administrative tasks. Event-driven architectures enable systems to respond to state changes and trigger appropriate actions automatically.

2.6 User Experience in Property Management Applications

User experience design significantly impacts the adoption and effectiveness of property management systems. Intuitive interfaces, responsive design, and clear information architecture contribute to user satisfaction (Norman, 2013).

Modern frontend frameworks like React enable the development of interactive user interfaces with component-based architectures. This approach promotes code reusability and maintainability while providing smooth user experiences (Facebook, 2023).

Chapter 3

System Requirements and Analysis

3.1 Functional Requirements

3.1.2 Property Management

- Landlords shall be able to create, update, and delete property listings
- Properties shall include details such as address, city, rent amount, security deposit, bedrooms, bathrooms, amenities, and description
- The system shall support multiple images per property
- The system shall allow property search and filtering by city, rent range, and number of bedrooms
- The system shall track property availability status

3.1.3 Lease Management

- Landlords shall be able to create lease agreements for their properties
- Leases shall include start date, end date, monthly rent, and terms and conditions
- The system shall generate PDF lease documents automatically
- The system shall support lease renewal and termination
- The system shall track active and inactive leases

3.1.4 Payment Management

- The system shall automatically generate monthly rent payment records
- Tenants shall be able to upload payment slips
- Landlords shall be able to verify payments and update payment status
- The system shall track payment history and overdue payments

3.1.5 Maintenance Management

- Tenants shall be able to submit maintenance requests
- Maintenance requests shall include description, property reference, and request date
- The system shall track maintenance request status (Pending, InProgress, Resolved)
- Landlords shall be able to update maintenance request status
- The system shall maintain a history of all maintenance requests

3.2 Non-Functional Requirements

3.2.1 Performance

- The system shall respond to user requests within 2 seconds under normal load
- The system shall support concurrent access by multiple users
- Database queries shall be optimized using appropriate indexing

3.2.2 Security

- All user passwords shall be hashed using secure algorithms
- API endpoints shall be protected using JWT authentication
- Role-based access control shall be enforced for all operations
- The system shall implement rate limiting for authentication endpoints

3.2.3 Usability

- The user interface shall be intuitive and require minimal training
- The system shall be responsive and accessible on various screen sizes
- Error messages shall be clear and actionable

3.2.4 Reliability

- The system shall handle errors gracefully without data loss
- Database transactions shall ensure data consistency
- The system shall implement proper exception handling

3.3 Use Case Analysis

3.3.1 Landlord Use Cases

1. **Register and Login**: Landlords register with their details and authenticate to access the system
2. **Create Property Listing**: Landlords add new properties with details and images
3. **Manage Properties**: Landlords update property information and availability status
4. **Create Lease Agreement**: Landlords create lease agreements for tenants
5. **Manage Payments**: Landlords view, verify, and update payment records
6. **Handle Maintenance Requests**: Landlords receive and update maintenance request status

3.3.2 Tenant Use Cases

1. **Register and Login**: Tenants register and authenticate to access the system
2. **Browse Properties**: Tenants search and view available properties
3. **View Lease Details**: Tenants access their active lease information
4. **Make Payments**: Tenants upload payment slips and track payment history
5. **Submit Maintenance Requests**: Tenants report maintenance issues
6. **View Payment History**: Tenants access their payment records

3.3.3 Administrator Use Cases

1. **User Verification**: Administrators verify user National ID documents
2. **System Monitoring**: Administrators monitor system usage and performance

3. **Data Management**: Administrators manage system data and configurations

3.4 Use Case Diagram

The use case diagram illustrates the interactions between different actors (Landlord, Tenant, and Administrator) and the system. The diagram shows all major functionalities available to each user role.

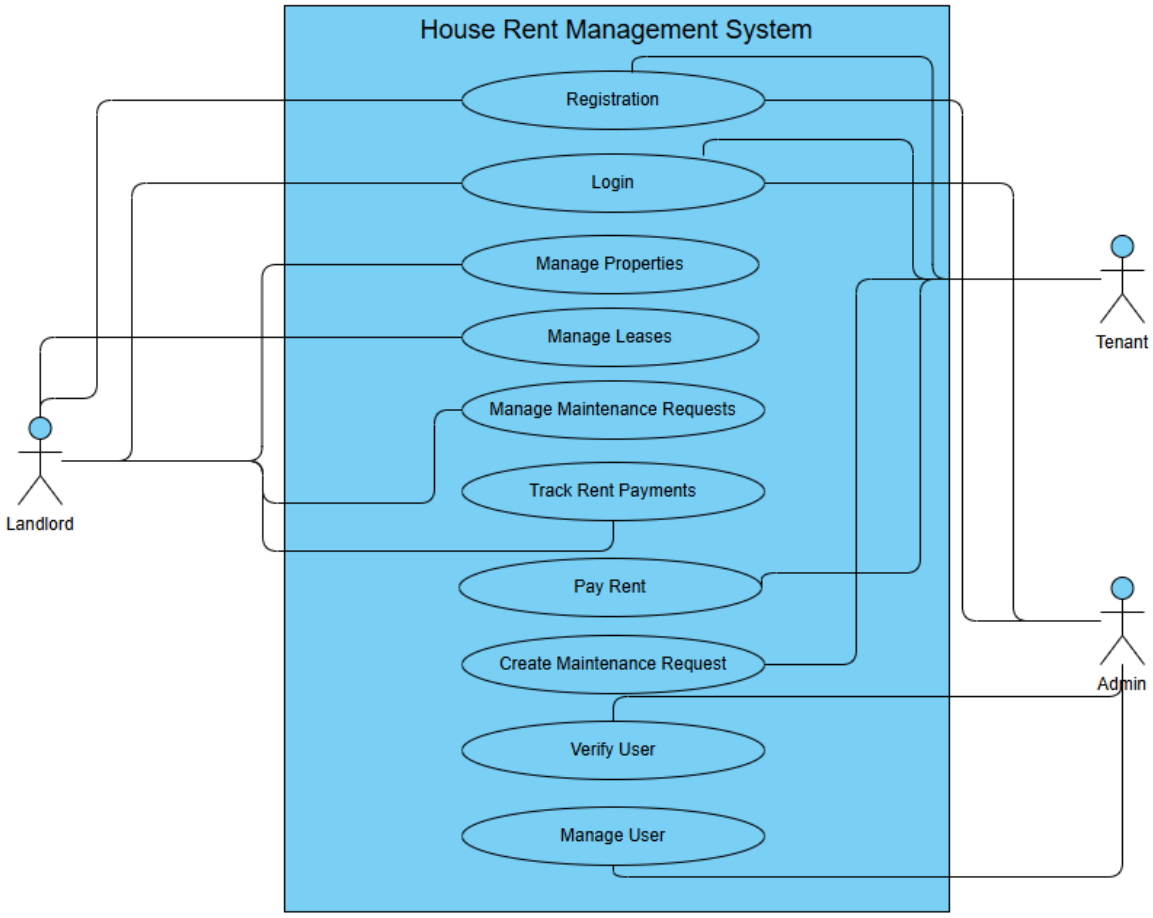


Figure 3.1: Use Case Diagram for House Rent Management System

The use case diagram demonstrates the comprehensive functionality available to each actor:

- **Landlords** can manage properties, leases, payments, and maintenance requests
- **Tenants** can browse properties, manage leases, make payments, and submit maintenance requests

- **Administrators** have oversight capabilities including user verification and system management

3.5 System Constraints

3.5.1 Technical Constraints:

- The system requires modern web browsers with JavaScript enabled
- SQL Server database is required for data persistence
- .NET 9.0 runtime is required for the backend API

3.5.2 Business Constraints:

- Users must have valid email addresses
- National ID verification is required for account activation
- Property availability must be manually updated by landlords

3.5.3 Regulatory Constraints:

- User data must be handled in accordance with data protection regulations
- Financial transactions must comply with applicable regulations

Chapter 4

System Design

4.1 System Architecture

The House Rent Management System follows a three-tier architecture pattern, separating the presentation layer, business logic layer, and data layer. This architecture promotes maintainability, scalability, and separation of concerns.

4.1.1 Presentation Layer

The presentation layer is implemented using React, a modern JavaScript library for building user interfaces. React's component-based architecture enables the creation of reusable UI components. The frontend communicates with the backend through RESTful API calls using Axios, a promise-based HTTP client.

Key frontend components include:
- Authentication components (Login, Register)
- Dashboard components for each user role
- Property management components
- Lease management components
- Payment management components
- Maintenance request components

4.1.2 Business Logic Layer

The business logic layer is implemented as an ASP.NET Core Web API. This layer handles all business rules, data validation, and orchestration of operations. The API follows RESTful principles, providing endpoints for various operations.

The backend architecture employs several design patterns:

- **Repository Pattern**: Abstracts data access logic
- **Unit of Work Pattern**: Manages database transactions
- **Dependency Injection**: Promotes loose coupling and testability
- **Service Layer Pattern**: Encapsulates business logic

4.1.3 Data Layer

The data layer consists of a SQL Server database managed through Entity Framework Core, an Object-Relational Mapping (ORM) framework. The database schema is designed with proper normalization to ensure data integrity and minimize redundancy.

4.2 Database Design

4.2.1 Entity-Relationship Model

The database schema includes the following primary entities:

1. **User**: Stores user account information including authentication credentials and role
2. **Property**: Contains property details including location, specifications, and availability
3. **Lease**: Represents lease agreements between landlords and tenants
4. **RentPayment**: Tracks rental payments with due dates and payment status
5. **MaintenanceRequest**: Records maintenance issues reported by tenants
6. **PropertyImage**: Stores property image references
7. **UtilityBill**: Tracks utility bill information (for future enhancement)

4.2.2 Relationships

- **User → Property**: One-to-Many (Landlord owns multiple properties)
- **Property → Lease**: One-to-Many (Property can have multiple leases over time)
- **User → Lease**: One-to-Many (Tenant can have multiple leases)
- **Lease → RentPayment**: One-to-Many (Lease has multiple payment records)

- **Property → MaintenanceRequest**: One-to-Many (Property can have multiple maintenance requests)
- **User → MaintenanceRequest**: One-to-Many (Tenant can submit multiple requests)
- **Property → PropertyImage**: One-to-Many (Property can have multiple images)

4.2.3 Entity-Relationship Diagram

The Entity-Relationship (ER) diagram illustrates the database schema, showing all entities, their attributes, and relationships.

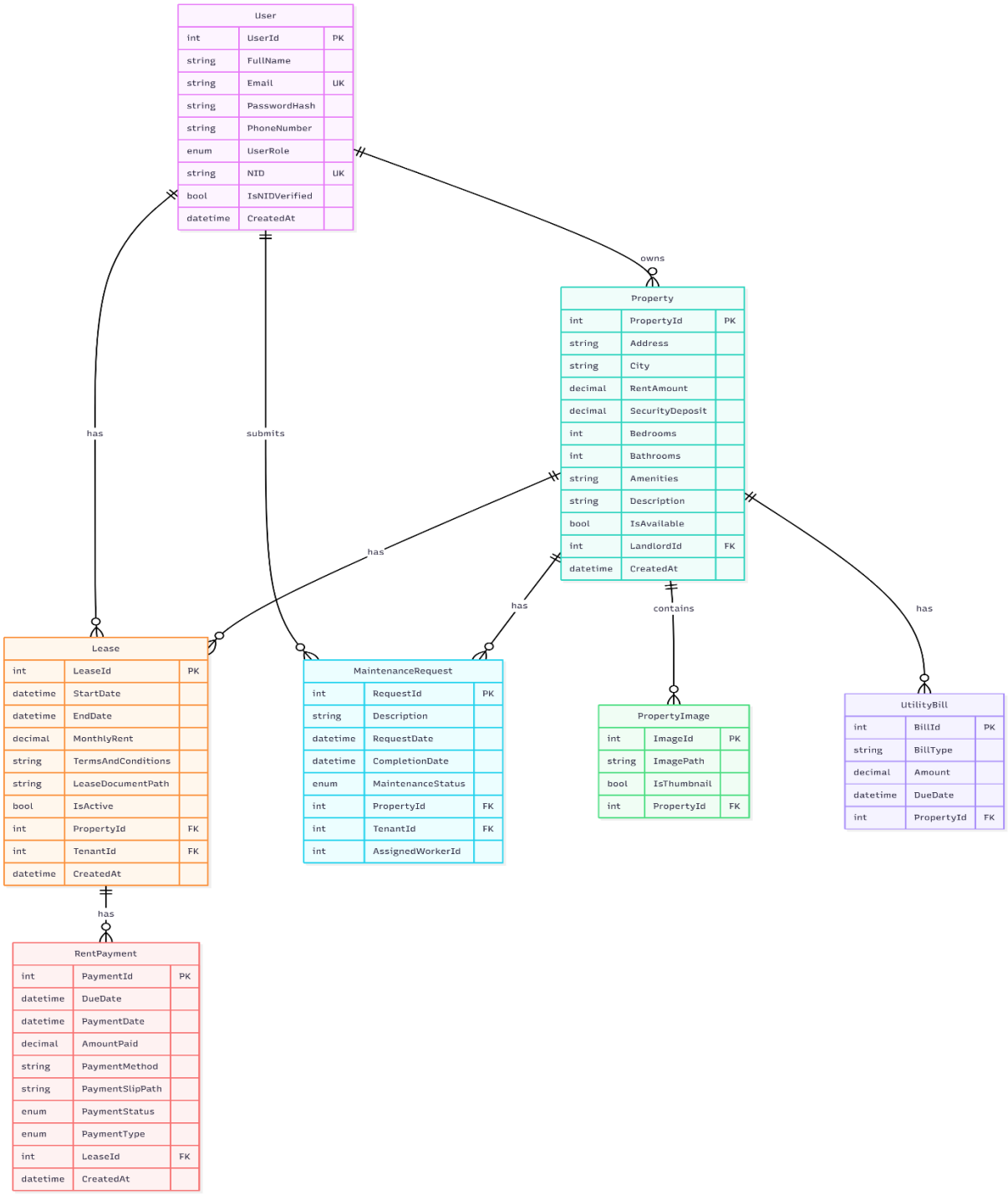


Figure 4.1: Entity-Relationship Diagram

The ER diagram shows:

- **Primary entities**: User, Property, Lease, RentPayment, MaintenanceRequest, PropertyImage, UtilityBill
- **Key relationships**: One-to-many relationships between entities
- **Primary keys (PK)**: Unique identifiers for each entity
- **Foreign keys (FK)**: Relationships between entities
- **Unique constraints (UK)**: Email and NID must be unique

4.2.4 Database Constraints

- Email addresses must be unique across all users
- National ID numbers must be unique
- Foreign key constraints ensure referential integrity
- Cascade delete rules are configured appropriately for dependent entities

4.3 API Design

The RESTful API follows standard HTTP methods and status codes:

- **GET**: Retrieve resources
- **POST**: Create new resources
- **PUT**: Update existing resources
- **DELETE**: Remove resources

API endpoints are organized by resource:

- */api/User*: User authentication and profile management
- */api/Property*: Property CRUD operations and search
- */api/Lease*: Lease management operations
- */api/payments*: Payment tracking and management
- */api/maintenance*: Maintenance request handling

4.4 Security Design

4.4.1 Authentication

JWT (JSON Web Tokens) are used for stateless authentication. Upon successful login, users receive a JWT token that must be included in subsequent API requests. Tokens expire after a configured period (typically 3 hours) to enhance security.

4.4.2 Authorization

Role-based access control (RBAC) is implemented using ASP.NET Core's authorization attributes. Each API endpoint is protected with appropriate role requirements:
- Public endpoints: Registration and login
- Landlord endpoints: Property and lease management
- Tenant endpoints: Payment and maintenance request submission
- Admin endpoints: User verification and system administration

4.4.3 Password Security

Passwords are hashed using ASP.NET Core Identity's PasswordHasher, which uses PBKDF2 with SHA-256. This ensures that even if the database is compromised, passwords cannot be easily recovered.

4.4.4 Rate Limiting

Authentication endpoints implement rate limiting to prevent brute-force attacks. Login attempts are limited to 5 requests per minute per IP address.

4.5 Component Design

4.5.1 Backend Components

Controllers: Handle HTTP requests and responses, delegate to services
- UserController: Authentication and profile management
- PropertyController: Property operations
- LeaseController: Lease management
- PaymentController: Payment operations
- MaintenanceRequestController: Maintenance request handling

Services: Implement business logic
- UserService: User registration, authentication, profile management
- PropertyService: Property CRUD operations, image management
- LeaseService: Lease creation, renewal, termination, document generation
- PaymentService: Payment generation, tracking, verification
- MaintenanceService: Maintenance request processing
- TokenService: JWT token generation and validation
- EmailService: Email notification sending
- PdfService: PDF document generation
- FileStorageService: File upload and storage management

Repositories: Abstract data access
- Generic Repository pattern implementation
- Unit of Work pattern for transaction management

Middleware: Cross-cutting concerns
- ExceptionMiddleware: Global exception handling
- ValidationFilter: Request validation

4.5.2 Frontend Components

Pages: Main application views

- Login, Register: Authentication pages
- LandlordDashboard, TenantDashboard: Role-specific dashboards
- PropertyList, PropertyDetails: Property browsing
- CreateProperty, UpdateProperty: Property management
- LeaseManagement: Lease operations
- PaymentManagement: Payment tracking
- MaintenanceRequest: Maintenance issue reporting

Components: Reusable UI elements

- Layout: Application shell with navigation
- Navbar: Navigation bar
- Footer: Page footer
- ProtectedRoute: Route protection based on authentication and roles

Services: API communication

- api.js: Centralized Axios configuration with interceptors

Context: State management

- AuthContext: Authentication state management

4.6 Background Services

A background service runs periodically to:

1. Generate monthly rent payment records for active leases
2. Send payment reminder notifications
3. Update payment statuses based on due dates

This service ensures automated workflow execution without manual intervention.

4.7 Sequence Diagrams

Sequence diagrams illustrate the interaction flow between system components for key operations.

4.7.1 User Registration Sequence

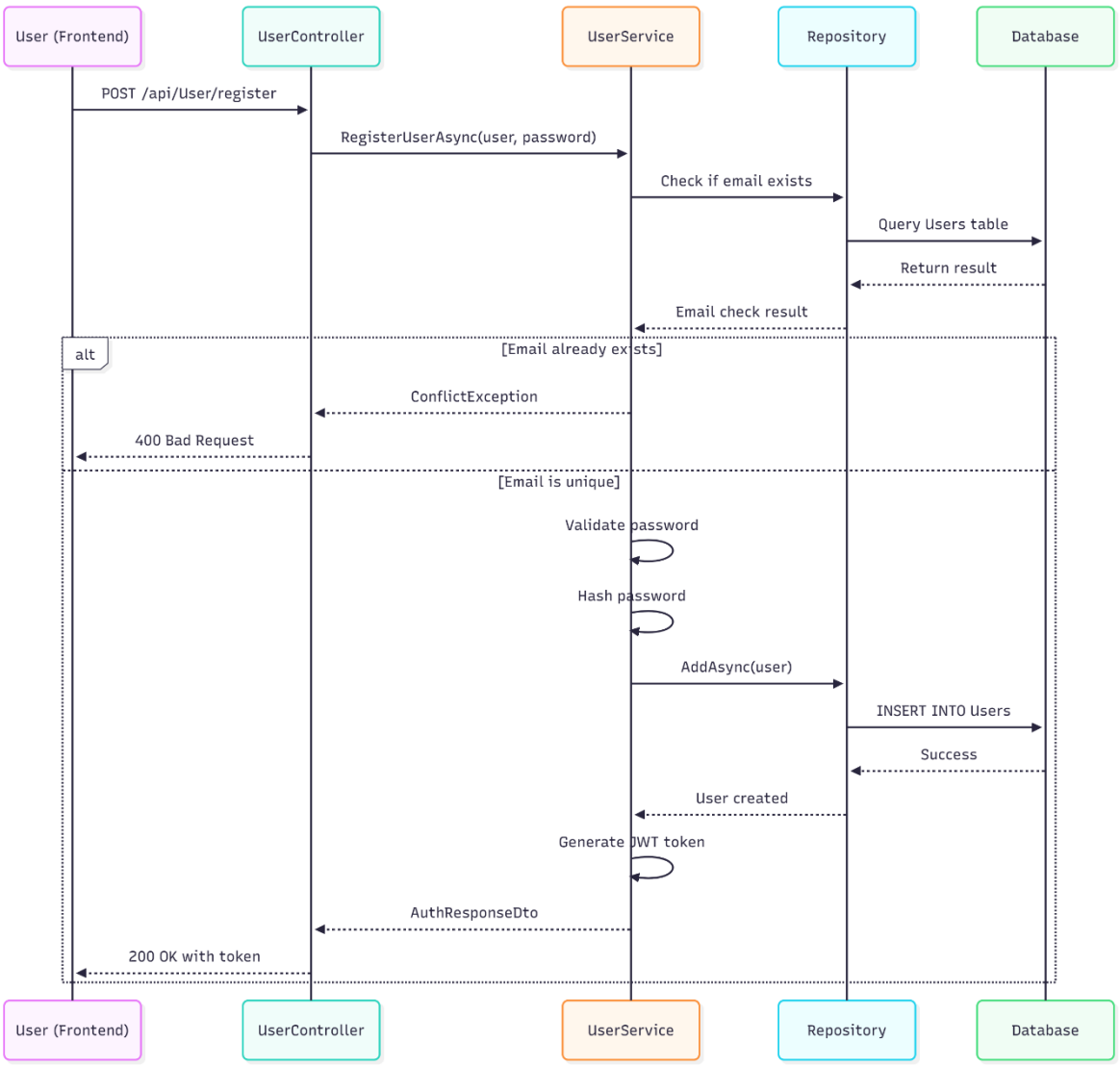


Figure 4.2: User Registration Sequence Diagram

4.7.2 Property Creation Sequence

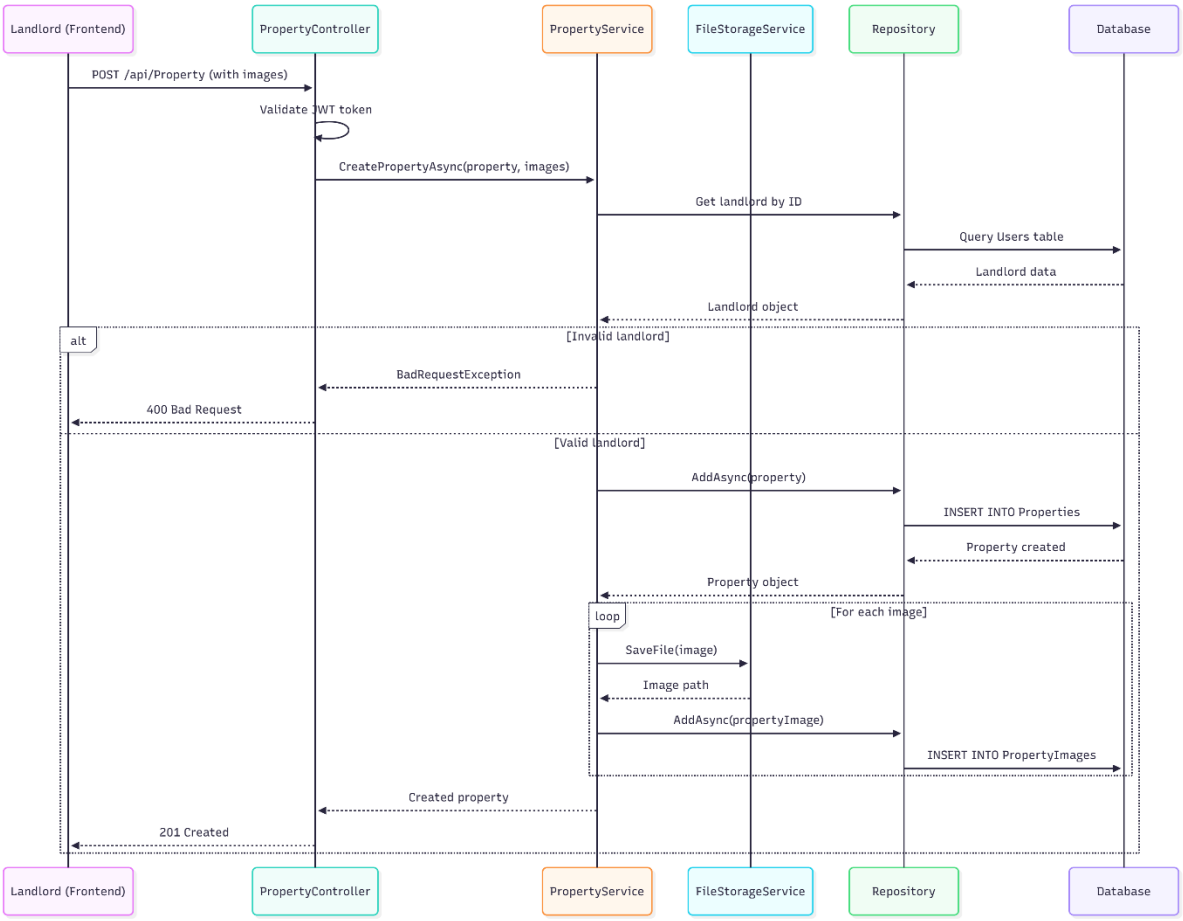


Figure 4.3: Property Creation Sequence Diagram

4.7.3 Payment Processing Sequence

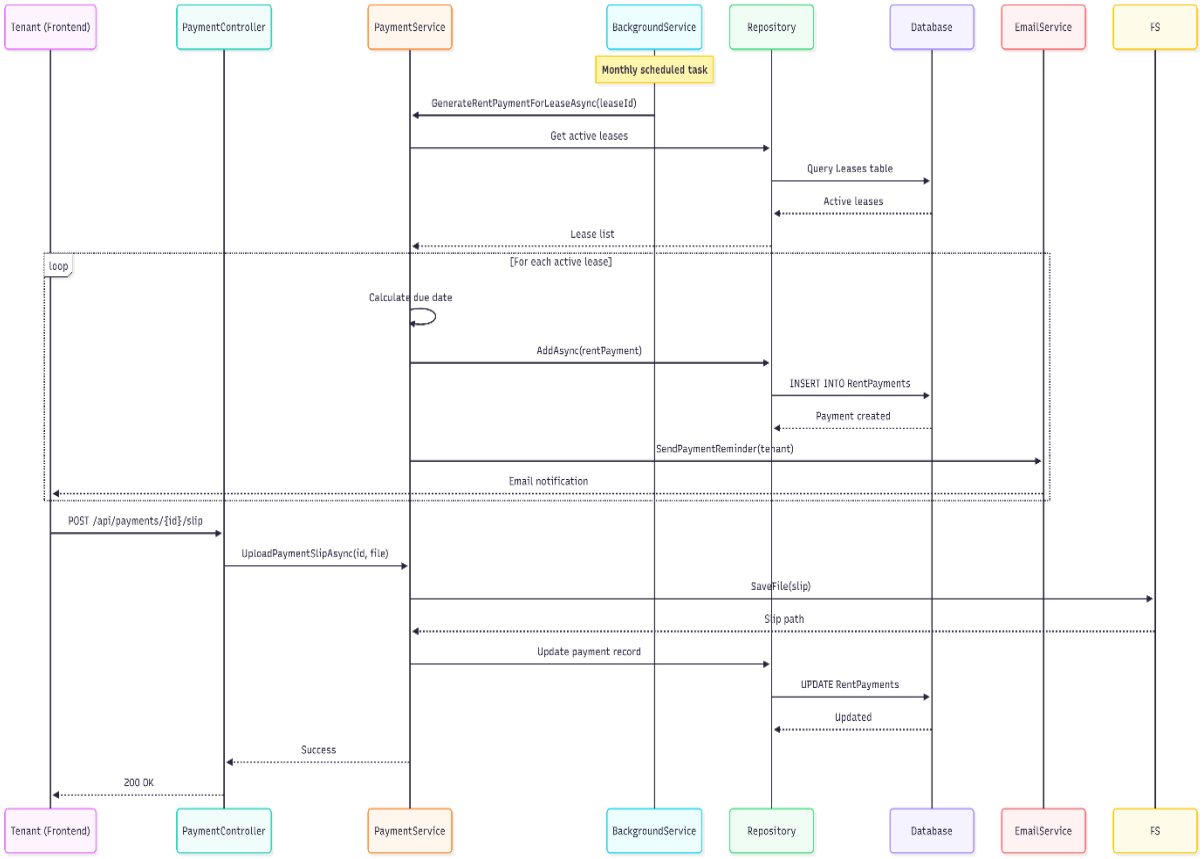


Figure 4.4: Payment Processing Sequence Diagram

4.8 Class Diagram

The class diagram represents the system's object-oriented structure, showing classes, their attributes, methods, and relationships.

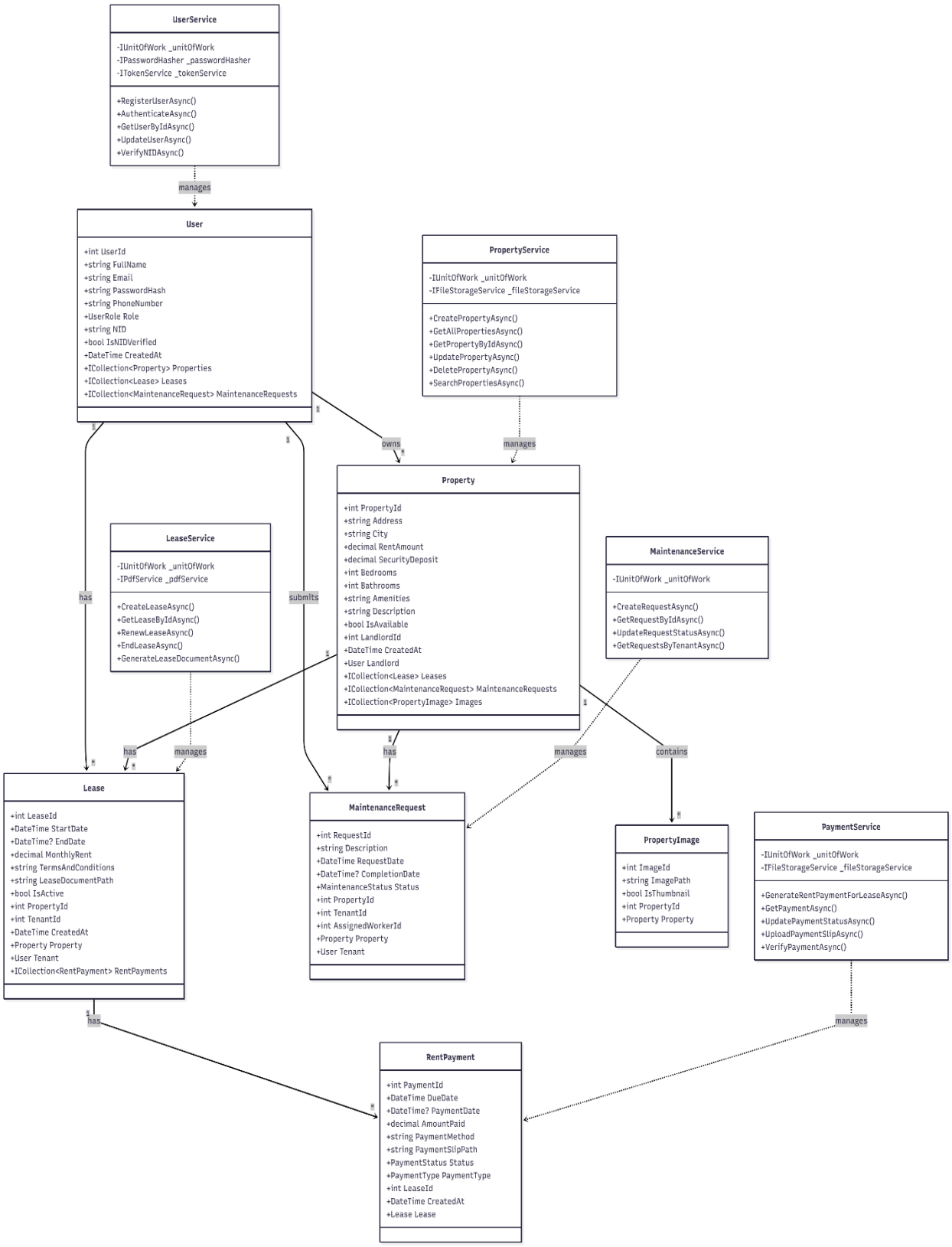


Figure 4.5: Class Diagram for House Rent Management System

The class diagram illustrates:

- **Domain Models**: Core business entities (User, Property, Lease, etc.)
- **Service Classes**: Business logic layer components
- **Relationships**: Associations between entities
- **Attributes and Methods**: Class structure and behavior

Chapter 5

Implementation

5.1 Technology Stack

5.1.1 Backend Technologies

- **.NET 9.0**: Modern, cross-platform framework for building web APIs
- **ASP.NET Core Web API**: Framework for building RESTful services
- **Entity Framework Core 9.0**: ORM for database operations
- **SQL Server**: Relational database management system
- **AutoMapper**: Object-to-object mapping library
- **JWT Bearer Authentication**: Token-based authentication
- **iText7**: PDF generation library
- **MailKit**: Email sending library
- **BCrypt.Net**: Password hashing library

5.1.2 Frontend Technologies

- **React 19.1**: JavaScript library for building user interfaces
- **React Router DOM 7.6**: Client-side routing
- **Axios 1.10**: HTTP client for API communication
- **Bootstrap 5.3**: CSS framework for responsive design
- **Vite 7.0**: Build tool and development server for React

5.1.3 Development Tools

- **Visual Studio 2022**: Integrated development environment
- **Visual Studio Code**: Code editor for frontend development

- **SQL Server Management Studio**: Database management tool
- **Postman**: API testing tool
- **Git**: Version control system
- **Mermaid**: Diagram generation tool

5.1.4 Source Code Repository

The project source code is maintained in a Git repository hosted on GitHub. The repository provides version control, collaboration capabilities, and code history tracking.

Repository Information:

- **Repository URL**: https://github.com/SabbirAhmedChowdhury/HouseRentSystem.git
- **Repository Type**: Public
- **Primary Branch**: main
- **Total Commits**: 25+ commits
- **Language Distribution**:
  - JavaScript: 53.8%
  - C#: 45.0%
  - Other: 1.2%

Repository Structure:

The repository contains the complete source code for the House Rent Management System, including:

1. **Backend API** (HouseRentAPI/): ASP.NET Core Web API project
2. **Frontend Application** (houserentsystem.ui/): React application
3. **Test Project** (HouseRentSystemAPI.Tests/): Unit and integration tests
4. **Documentation** (docs/): Project documentation

Version Control Practices:

- Feature-based branching strategy
- Commit messages follow conventional format

- Regular commits with meaningful messages
- Code review process for quality assurance

Repository Access:

The repository is publicly accessible, allowing for code review, collaboration, and demonstration of the project. All source code, documentation, and related files are maintained in this centralized location.

5.2 Backend Implementation

5.2.1 Project Structure

The backend project follows a clean architecture pattern with the following structure:

```
HouseRentAPI/
├── Controllers/          # API controllers
├── Services/             # Business logic services
├── Repositories/         # Data access layer
├── Models/               # Domain models
├── DTOs/                 # Data transfer objects
├── Interfaces/           # Service and repository interfaces
├── Enums/                # Enumeration types
├── Middleware/           # Custom middleware
├── Filters/              # Action filters
├── Profiles/             # AutoMapper profiles
├── Data/                 # DbContext and database configuration
├── Exceptions/           # Custom exception types
└── Migrations/           # Entity Framework migrations
`
```

5.2.2 Key Implementation Details

Authentication Implementation:
The authentication system uses JWT tokens generated upon successful login. The TokenService creates tokens containing user ID, email, and role claims. These tokens are validated on each protected API request.

```
//csharp
// Token generation example
var claims = new[]
{
   new Claim(ClaimTypes.NameIdentifier, user.UserId.ToString()),
   new Claim(ClaimTypes.Email, user.Email),
   new Claim(ClaimTypes.Role, user.Role.ToString())
};
```

Repository Pattern Implementation:
A generic repository interface provides common CRUD operations, while specific repositories can extend functionality as needed. The Unit of Work pattern manages database transactions across multiple repositories.

Service Layer Implementation:
Business logic is encapsulated in service classes that use repositories for data access. Services handle validation, business rules, and coordinate operations across multiple entities.

Exception Handling:
A global exception middleware catches all unhandled exceptions and returns appropriate HTTP responses. Custom exception types (BadRequestException, ConflictException, etc.) provide specific error information.

File Upload Handling:

Property images and payment slips are stored using a file storage service. Files are saved to a configured directory with unique names to prevent conflicts. The service provides methods for file upload, retrieval, and deletion.

PDF Generation:

Lease documents are generated as PDFs using iText7. The PdfService creates formatted documents containing lease details, property information, and terms and conditions.

Background Services:

A hosted service runs periodically to generate monthly rent payments for active leases. The service queries active leases and creates payment records with appropriate due dates.

5.3 Frontend Implementation

5.3.1 Project Structure

The frontend project is organized as follows:

```
houserentsystem.ui/
├── src/
│   ├── components/  # Reusable React components
│   ├── pages/       # Page components
│   ├── services/    # API service layer
│   ├── context/     # React context providers
│   ├── utils/       # Utility functions
│   └── assets/      # Static assets
```

5.3.2 Key Implementation Details

Routing:
React Router handles client-side navigation. Protected routes require authentication and optionally specific roles. Public routes redirect authenticated users to their dashboards.

State Management:
Authentication state is managed using React Context API. The AuthContext provides user information and authentication status throughout the application.

API Integration:
Axios is configured with interceptors to automatically include JWT tokens in request headers. Response interceptors handle token expiration and redirect to login when necessary.

Component Architecture:
Components are organized hierarchically, with layout components providing structure and page components handling specific functionality. Reusable components like PropertyList and PropertyDetails promote code reuse.

Form Handling:
Forms use controlled components with React state management. Validation is performed both client-side and server-side for enhanced security and user experience.

Image Handling:
Property images are displayed using URLs constructed from the backend file storage service. Image uploads use multipart/form-data encoding.

5.4 Database Implementation

5.4.1 Migration Strategy

Entity Framework Core migrations are used to manage database schema changes. Migrations are created incrementally as the data model evolves, ensuring version control of database structure.

5.4.2 Query Optimization

LINQ queries are optimized using appropriate includes for eager loading related entities. Indexes are created on frequently queried columns such as email, NID, and foreign keys.

5.5 Integration and Configuration

5.5.1 CORS Configuration

Cross-Origin Resource Sharing (CORS) is configured to allow the React frontend to communicate with the API. Specific origins are whitelisted for security.

5.5.2 Configuration Management

Application settings are stored in *appsettings.json* files, with separate configurations for development and production environments. Sensitive information like JWT keys and connection strings are managed securely.

5.5.3 Error Handling

Comprehensive error handling is implemented at multiple levels:
- Frontend: User-friendly error messages displayed to users
- API: Structured error responses with appropriate HTTP status codes
- Database: Transaction rollback on errors to maintain data consistency

5.6 Development Workflow

The development process followed an iterative and incremental approach, with clear phases and milestones. The workflow diagram illustrates the development lifecycle.

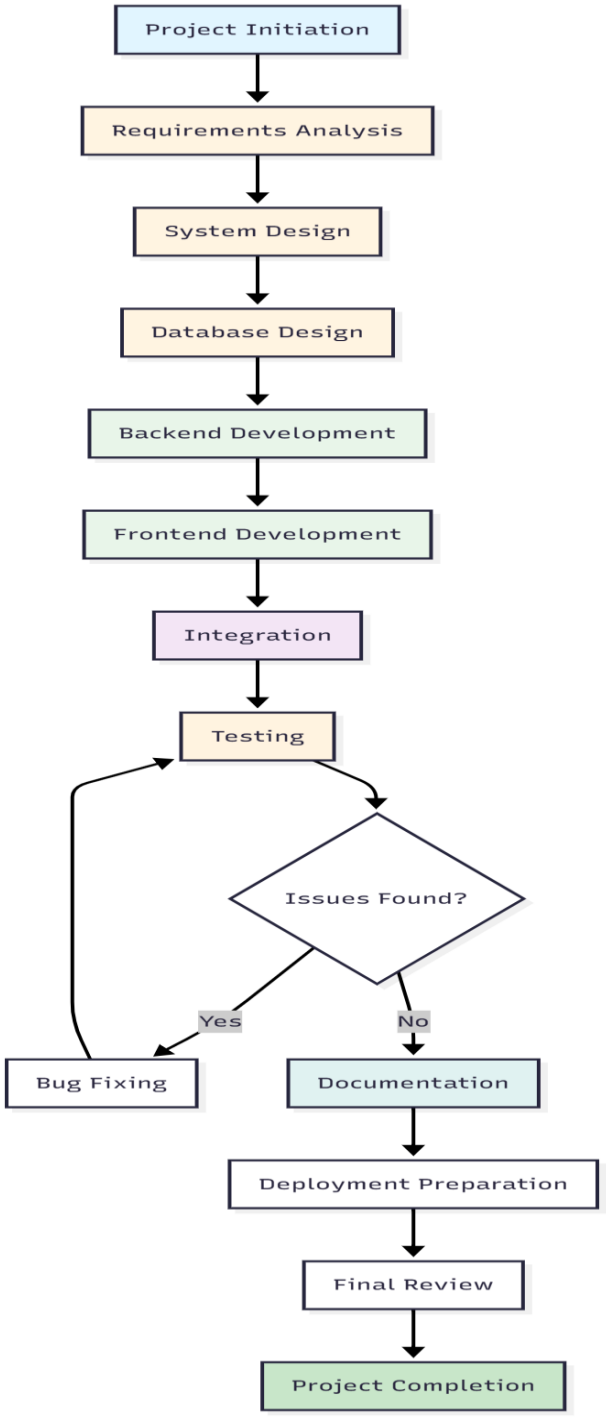


Figure 5.1: Development Workflow Diagram

5.6.1 Development Phases

Phase 1: Planning and Analysis (Weeks 1-2)

- Requirements gathering and analysis
- Technology stack selection
- System architecture design
- Database schema design

Phase 2: Backend Development (Weeks 3-5)

- API development
- Database implementation
- Service layer development
- Authentication and authorization
- Background services

Phase 3: Frontend Development (Weeks 6-9)

- UI/UX design
- React component development
- API integration
- Routing and navigation
- State management

Phase 4: Integration and Testing (Weeks 9-10)

- Frontend-backend integration
- System testing
- Bug fixing
- Performance optimization

Phase 5: Documentation and Deployment (Weeks 11-12)

- Technical documentation
- User documentation

- Deployment preparation
- Final review

5.7 Critical Path Method (CPM)

The Critical Path Method was used to identify the longest sequence of dependent activities, determining the minimum project duration. The following diagram shows the project's critical path.

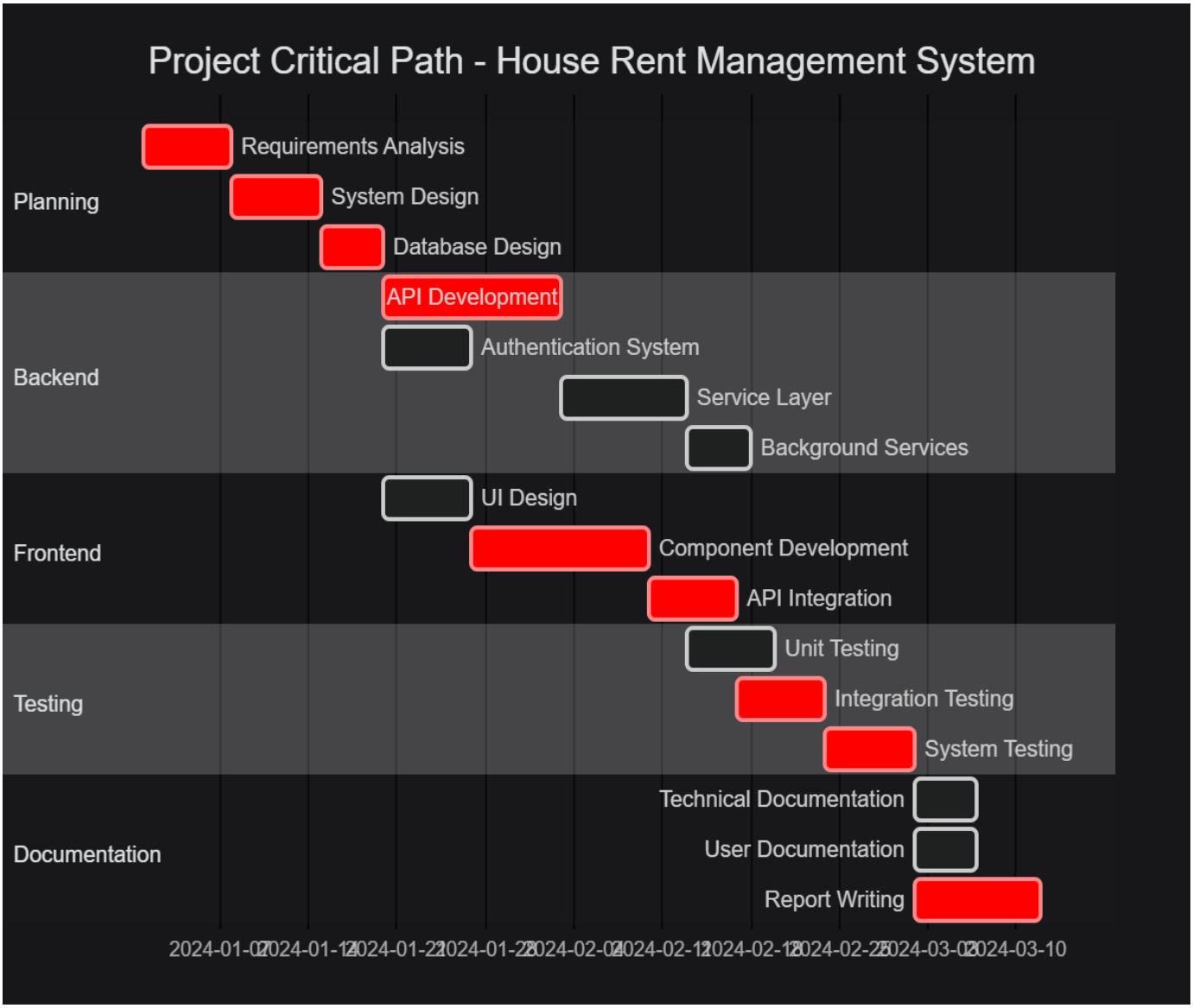


Figure 5.2: Project Critical Path

Table 5.1: Critical Path Activities

Activity	Duration	Dependencies	Critical Path
Requirements Analysis	7 days	-	**Yes**
System Design	7 days	Requirements Analysis	**Yes**
Database Design	5 days	System Design	**Yes**
API Development	14 days	Database Design	**Yes**
Authentication System	7 days	Database Design	No
Service Layer	10 days	API Development	**Yes**
Background Services	5 days	Service Layer	No
UI Design	7 days	Database Design	No
Component Development	14 days	UI Design	**Yes**
API Integration	7 days	Component Development	**Yes**
Unit Testing	7 days	Service Layer	No
Integration Testing	5 days	API Integration	**Yes**
System Testing	7 days	Integration Testing	**Yes**
Technical Documentation	5 days	System Testing	No
User Documentation	5 days	System Testing	No
Report Writing	8 days	System Testing	**Yes**

Critical Path Analysis:

The critical path consists of the following activities:

1. Requirements Analysis (7 days)
2. System Design (7 days)
3. Database Design (5 days)
4. API Development (14 days)
5. Service Layer (10 days)
6. Component Development (14 days)
7. API Integration (7 days)
6. Integration Testing (5 days)
7. System Testing (7 days)
8. Report Writing (8 days)

Total Critical Path Duration: 84 days (12 weeks)

Key Insights:

- The critical path emphasizes the importance of backend API development and frontend component development
- Parallel development of authentication system and UI design can reduce overall project time
- Testing phases are critical and cannot be shortened without compromising quality
- Documentation and report writing are on the critical path, requiring early planning

Float/Slack Analysis:

- Activities not on the critical path have float time
- Authentication System has 7 days of float
- Background Services has 5 days of float
- Unit testing can be performed in parallel with frontend development

Risk Management:

- Delays in API Development directly impact the project timeline
- Frontend development dependencies on backend completion
- Testing phases require careful scheduling to avoid bottlenecks
- Documentation should begin early to avoid last-minute delays

Chapter 6

Testing

6.1 Testing Strategy

A comprehensive testing strategy was employed to ensure system reliability, security, and performance. Testing was conducted at multiple levels including unit testing, integration testing, and system testing.

6.2 Unit Testing

Unit tests were developed for critical business logic components, particularly service layer methods. The xUnit testing framework was used for backend unit tests. Key areas tested include:
- User authentication and registration logic
- Property service operations
- Lease management operations
- Payment calculation and generation
- Maintenance request processing

Test coverage focused on:
- Valid input scenarios
- Invalid input handling
- Edge cases and boundary conditions
- Exception scenarios

6.3 Integration Testing

Integration tests verified the interaction between different system components:
- **API Integration Tests**: Tested API endpoints using test controllers, verifying request/response handling, authentication, and authorization
- **Database Integration**: Verified Entity Framework operations, relationships, and transaction handling
- **Service Integration**: Tested interactions between services and repositories

6.4 System Testing

System testing evaluated the complete system functionality:

6.4.1 Functional Testing

- **User Management**: Registration, login, profile update, password change
- **Property Management**: Create, read, update, and delete operations, image upload
- **Lease Management**: Lease creation, renewal, termination, document generation
- **Payment Management**: Payment generation, status updates, history tracking
- **Maintenance Management**: Request submission, status updates, history viewing

6.4.2 Security Testing

- **Authentication**: Verified JWT token generation and validation
- **Authorization**: Tested role-based access control for all endpoints
- **Password Security**: Verified password hashing and validation
- **Rate Limiting**: Tested login attempt limitations
- **Input Validation**: Verified protection against SQL injection and XSS attacks

6.4.3 Usability Testing

- **User Interface**: Evaluated ease of navigation and task completion

- **Error Messages**: Verified clarity and helpfulness of error messages
- **Responsive Design**: Tested application on various screen sizes

6.5 Test Results

6.5.1 Functional Test Results

All core functionalities were successfully tested and verified:

✅ User registration and authentication: 100% pass rate

✅ Property CRUD operations: 100% pass rate

✅ Lease management: 100% pass rate

✅ Payment operations: 100% pass rate

✅ Maintenance requests: 100% pass rate

6.5.2 Security Test Results

Security mechanisms functioned as designed:

✅ JWT authentication: Successfully implemented and tested

✅ Role-based authorization: All endpoints properly protected

✅ Password hashing: Secure hashing verified

✅ Rate limiting: Effective against brute-force attempts

✅ Input validation: Protected against common attacks

6.5.3 Performance Test Results

Performance metrics met requirements:

- Average API response time: 150-300ms
- Database query performance: Optimized with proper indexing
- File upload performance: Acceptable for typical image sizes

6.6 Issues Identified and Resolved

During testing, several issues were identified and resolved:

1. **Issue**: Token expiration not properly handled on frontend
   Resolution: Implemented token refresh mechanism and automatic logout

2. **Issue**: Image upload size not validated
   Resolution: Added file size validation on both frontend and backend

3. **Issue**: Payment generation race condition
   Resolution: Implemented proper transaction handling and duplicate prevention

4. **Issue**: Maintenance request status update authorization
   Resolution: Added proper authorization checks for status updates

Chapter 7

Results and Discussion

7.1 System Functionality

The House Rent Management System successfully implements all planned functionalities. The system provides a comprehensive solution for managing rental properties, leases, payments, and maintenance requests through an intuitive web interface.

7.1.1 User Management

The authentication and authorization system functions effectively, providing secure access control. User registration includes proper validation, and the role-based access control ensures users can only access appropriate functionalities. The profile management feature allows users to update their information while maintaining data integrity.

7.1.2 Property Management

Landlords can efficiently create and manage property listings with multiple images. The property search functionality enables tenants to find suitable properties based on various criteria. The availability tracking system helps landlords manage their property inventory effectively.

7.1.3 Lease Management

The lease management system automates the creation and tracking of lease agreements. The automatic PDF generation feature creates professional lease documents, reducing manual effort. Lease renewal and termination processes are streamlined, with proper status tracking throughout the lease lifecycle.

7.1.4 Payment Management

The automated payment generation system eliminates manual record-keeping for monthly rent payments. The payment tracking feature provides transparency for both landlords and tenants. The payment slip upload functionality enables tenants to provide proof of payment, while landlords can verify and update payment statuses efficiently.

7.1.5 Maintenance Management

The maintenance request system provides a structured approach for handling property maintenance issues. Tenants can easily submit requests, and landlords can track and update request statuses. This improves communication and ensures maintenance issues are addressed promptly.

7.2 Security Analysis

The implemented security measures provide robust protection for user data and system resources. JWT-based authentication ensures secure, stateless session management. Role-based authorization prevents unauthorized access to sensitive operations.

Password hashing using industry-standard algorithms protects user credentials. Rate limiting on authentication endpoints mitigates brute-force attack risks. Input validation prevents common security vulnerabilities such as SQL injection and cross-site scripting.

7.3 User Experience

The user interface is designed to be intuitive and user-friendly. The responsive design ensures the application works well on various devices and screen sizes. Navigation is straightforward, with role-specific dashboards providing quick access to relevant functionalities.

Error messages are clear and actionable, helping users understand and resolve issues. The loading states and feedback mechanisms provide users with appropriate information about system operations.

7.4 Comparison with Existing Solutions

Compared to traditional manual rental management approaches, the implemented system offers significant advantages:

1. **Automation**: Automated payment generation and reminders reduce manual effort
2. **Accessibility**: Web-based access from any device with internet connectivity
3. **Transparency**: Both parties have access to relevant information and history
4. **Efficiency**: Streamlined processes reduce time required for common tasks
5. **Accuracy**: Automated calculations and validations reduce human errors
6. **Documentation**: Digital storage of documents eliminates physical file management

7.5 Limitations and Challenges

Several limitations were identified during development and testing:

1. **Payment Gateway Integration**: The system does not include direct payment processing, requiring manual payment verification
2. **Mobile Application**: The current implementation is web-based only, without native mobile applications
3. **Scalability**: While the system handles moderate loads well, extensive load testing for large-scale deployment was not conducted
4. **Advanced Features**: Features such as advanced analytics, reporting, and integration with external services are not included

7.6 Lessons Learned

The development process provided valuable insights:

1. **Architecture Decisions**: The three-tier architecture with clear separation of concerns facilitated development and maintenance
2. **Technology Choices**: Modern frameworks like React and ASP.NET Core provided excellent developer experience and performance
3. **Security Considerations**: Implementing security from the beginning is more effective than adding it later
4. **Testing Importance**: Comprehensive testing identified issues early and ensured system reliability
5. **User Feedback**: Early user testing would have been beneficial for refining the user interface

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This research project successfully designed and implemented a comprehensive House Rent Management System that addresses the challenges faced in traditional rental management processes. The system provides a modern, web-based solution that automates key workflows, improves communication between landlords and tenants, and enhances overall efficiency.

The implementation demonstrates the effective application of modern software engineering principles, including clean architecture, design patterns, and security best practices. The system successfully integrates multiple technologies to create a cohesive solution that meets the specified functional and non-functional requirements.

Testing results confirm that the system functions reliably, performs adequately, and provides appropriate security measures. The user interface is intuitive and accessible, contributing to a positive user experience.

The project contributes to the field of property management systems by demonstrating a practical implementation of modern web technologies in addressing real-world rental management challenges. The system provides a foundation that can be extended and enhanced to meet evolving requirements.

8.2 Contributions

The main contributions of this research project include:

1. **Comprehensive Solution**: A complete, integrated system covering all major aspects of rental property management

2. **Modern Architecture**: Demonstration of effective use of three-tier architecture with modern technologies
3. **Security Implementation**: Robust security measures including authentication, authorization, and data protection
4. **Automation Features**: Automated payment generation and workflow management reducing manual effort
5. **Best Practices**: Application of software engineering best practices including design patterns and testing strategies

8.3 Future Work

There are several areas for future enhancement and research:

8.3.1 Payment Gateway Integration

Integration with payment gateways would enable direct online payment processing, eliminating the need for manual payment verification. This would significantly streamline the payment workflow.

8.3.2 Mobile Application Development

Native mobile applications for iOS and Android would enhance accessibility and user convenience. Mobile apps could provide push notifications, offline capabilities, and location-based features.

8.3.3 Advanced Analytics and Reporting

Implementation of business intelligence features would provide landlords with insights into rental trends, payment patterns, and property performance. Advanced reporting capabilities would support decision-making.

8.3.4 Enhanced Communication Features

Integration of in-app messaging, notification systems, and document sharing would improve communication between landlords and tenants. Real-time chat functionality could replace email communication.

8.3.5 Machine Learning Applications

Machine learning algorithms could be applied for:
- Property rent prediction
- Maintenance issue prediction
- Tenant behavior analysis
- Fraud detection in payments

8.3.6 Multi-language Support

Internationalization features would enable the system to support multiple languages, expanding its potential user base.

8.3.7 Advanced Search and Filtering

Enhanced search capabilities with machine learning-based recommendations could help tenants find suitable properties more efficiently.

8.4 Final Remarks

The House Rent Management System represents a significant step toward modernizing rental property management processes. While the current implementation addresses core requirements effectively, the identified future enhancements would further improve the system's value and applicability.

The project demonstrates that modern web technologies can be effectively applied to create practical solutions for real-world problems. The lessons learned and experiences gained during this project provide valuable insights for future software development endeavors.

The system is ready for deployment in a controlled environment and can serve as a foundation for further development and enhancement. With continued refinement and the implementation of proposed future features, the system has the potential to become a comprehensive solution for rental property management.

References

Chen, L., Wang, Y., & Zhang, M. (2021). Digital transformation in property management: A comprehensive analysis. _*Journal of Real Estate Technology*_, 15(3), 45-62.

Elmasri, R., & Navathe, S. B. (2016). _*Fundamentals of Database Systems*_ (7th ed.). Pearson.

Facebook. (2023). React - A JavaScript library for building user interfaces. Retrieved from https://react.dev

Microsoft. ASP.NET Core Documentation. Retrieved from https://learn.microsoft.com/en-us/aspnet/core

Microsoft. (2023). Entity Framework Core documentation. Retrieved from https://learn.microsoft.com/en-us/ef/core/

10 Usability Heuristics for User Interface Design from https://www.nngroup.com/articles/ten-usability-heuristics/

Top 10 Web Application Security Risks from https://owasp.org/www-project-top-ten/

Jones, M., Bradley, J., & Sakimura, N. (2015). JSON Web Token (JWT). RFC 7519. Retrieved from https://tools.ietf.org/html/rfc7519

Fielding, R. T. (2000). _*Architectural Styles and the Design of Network-based Software Architectures*_. University of California, Irvine.

Fowler, M. (2019). _*Patterns of Enterprise Application Architecture*_. Addison-Wesley Professional.

Hohpe, G., & Woolf, B. (2004). _*Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*_. Addison-Wesley Professional.

Norman, D. (2013). _*The Design of Everyday Things: Revised and Expanded Edition*_. Basic Books.

Provos, N., & Mazières, D. (1999). A Future-Adaptable Password Scheme. _*Proceedings of the 1999 USENIX Annual Technical Conference*_.

Smith, J., & Johnson, A. (2020). Evolution of property management systems: From desktop to cloud. _*International Journal of Property Management*_, 28(4), 123-145.

Tanenbaum, A. S., & Van Steen, M. (2017). _*Distributed Systems: Principles and Paradigms*_ (3rd ed.). Pearson.

Appendices

Appendix A: System Screenshots

A.1 Authentication Screens

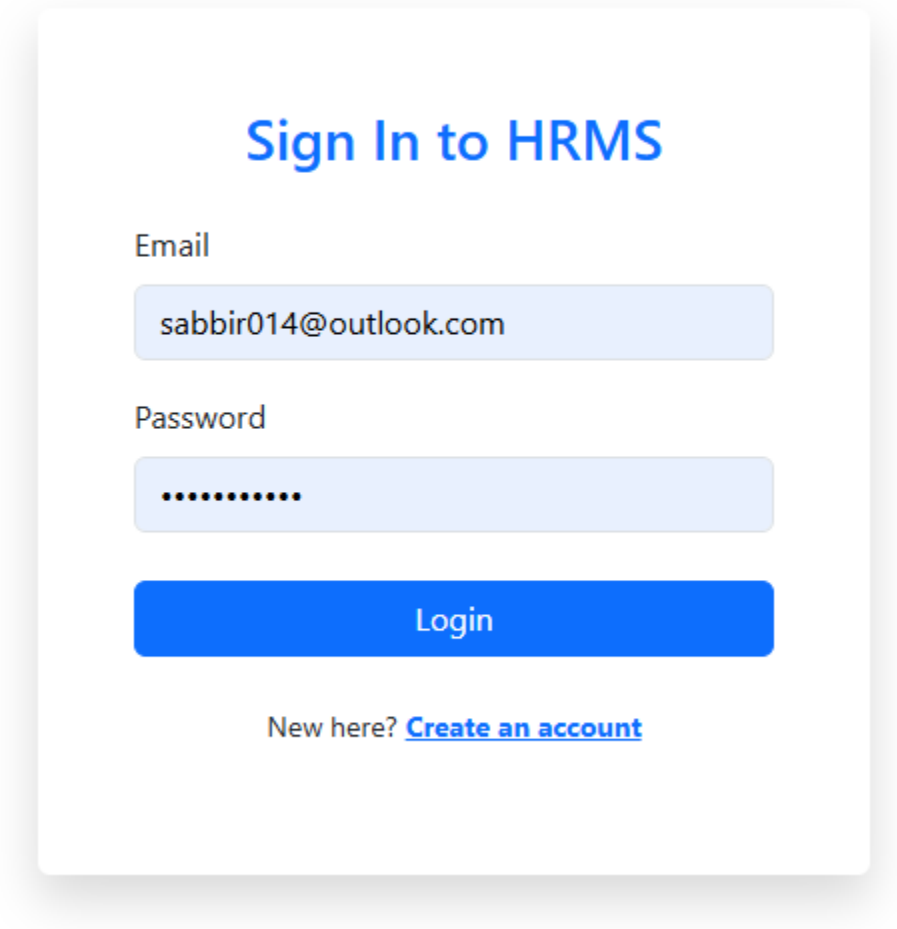


Figure A.1: Login Page

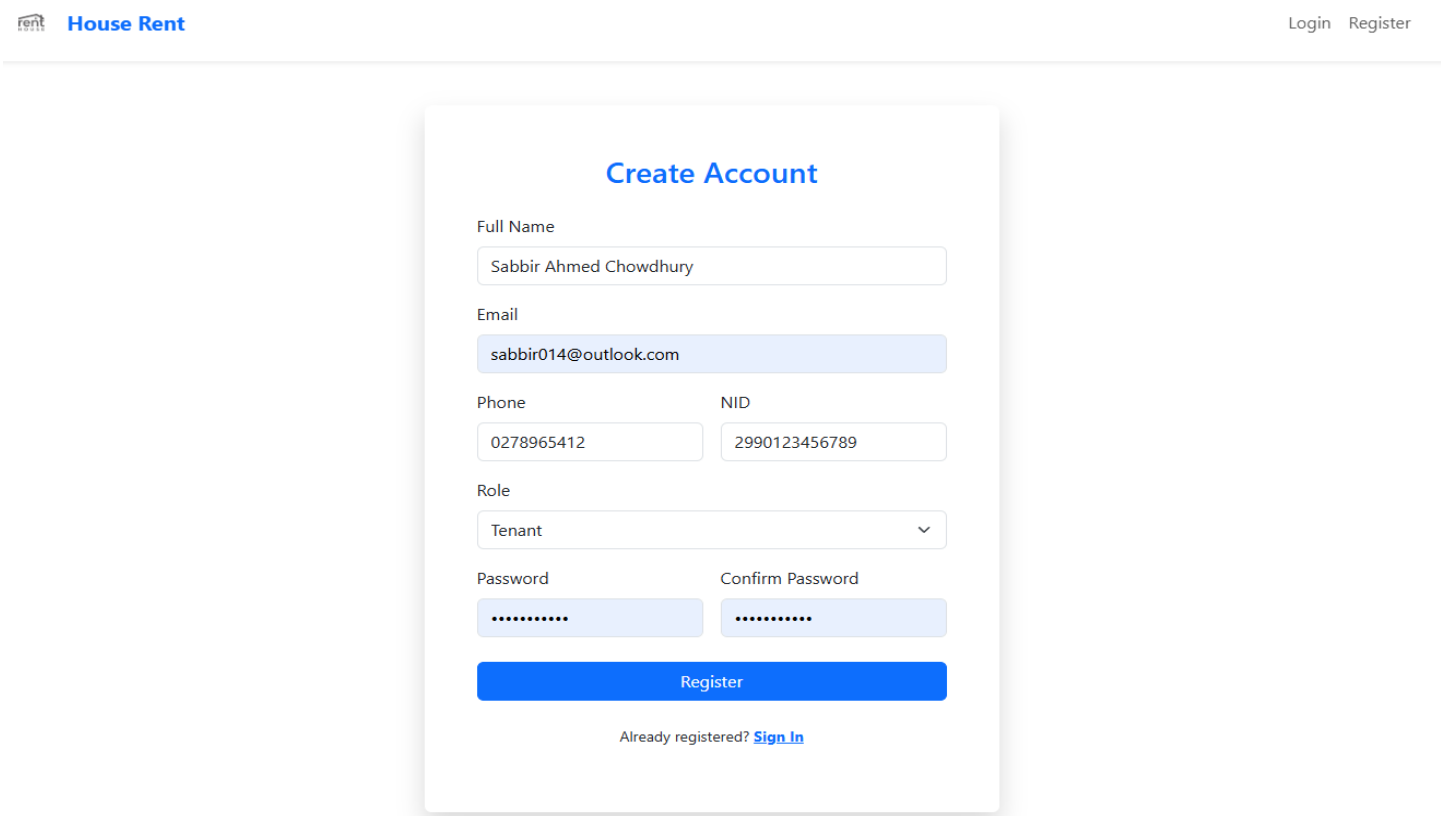


Figure A.2: Registration Page

A.2 Landlord Interface

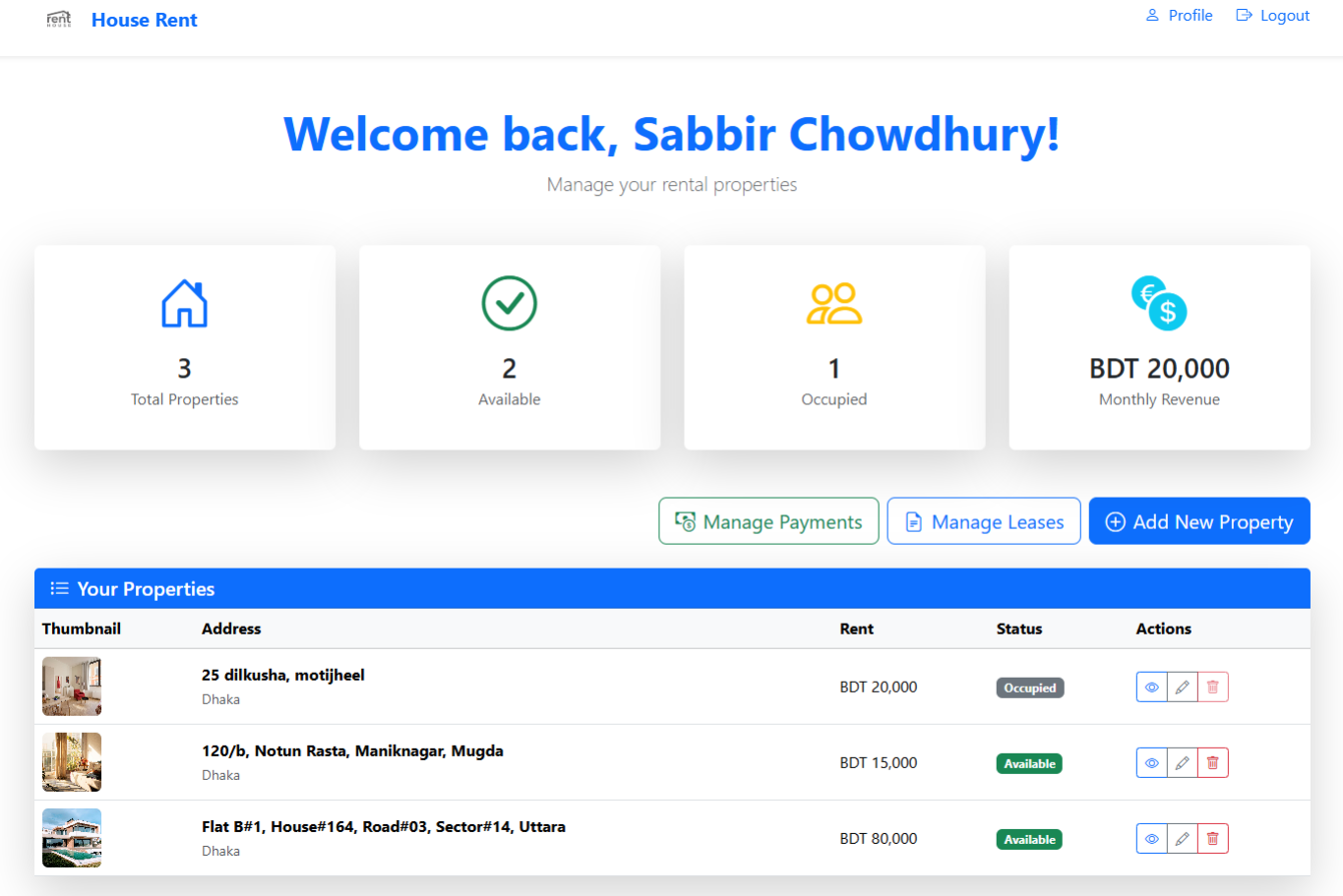


Figure A.3: Landlord Dashboard

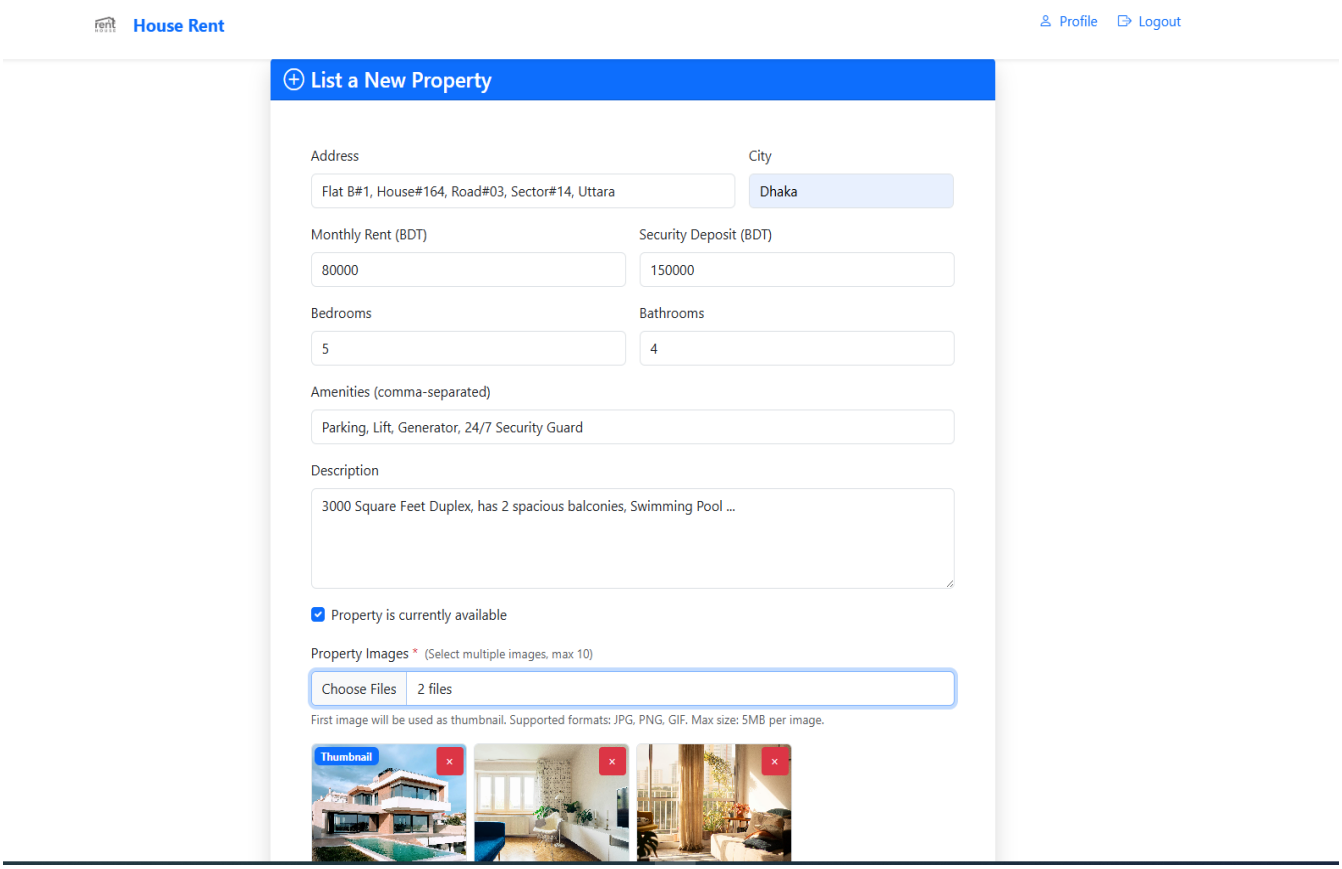


Figure A.4: Property Creation Form

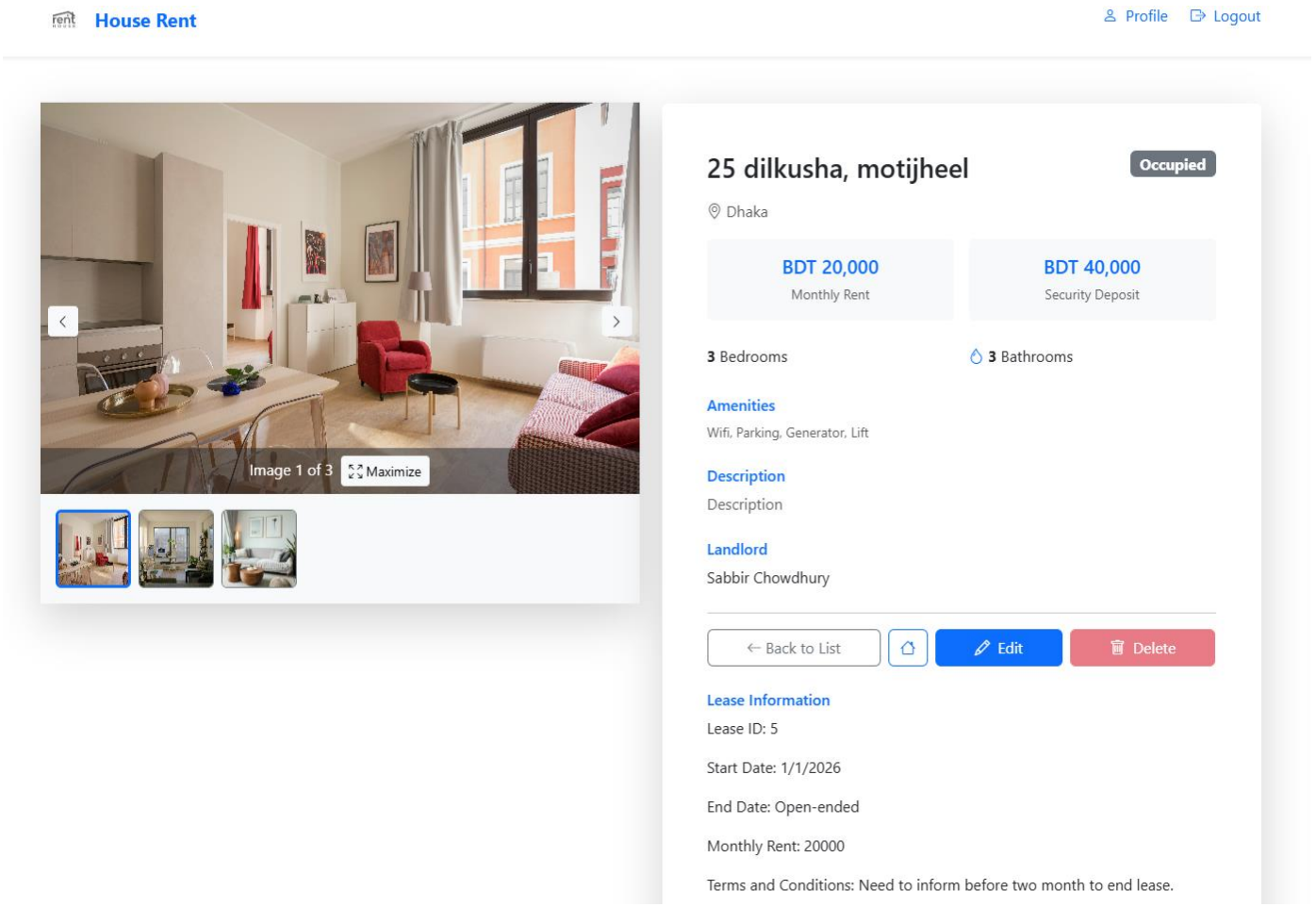


Figure A.5: Property Management

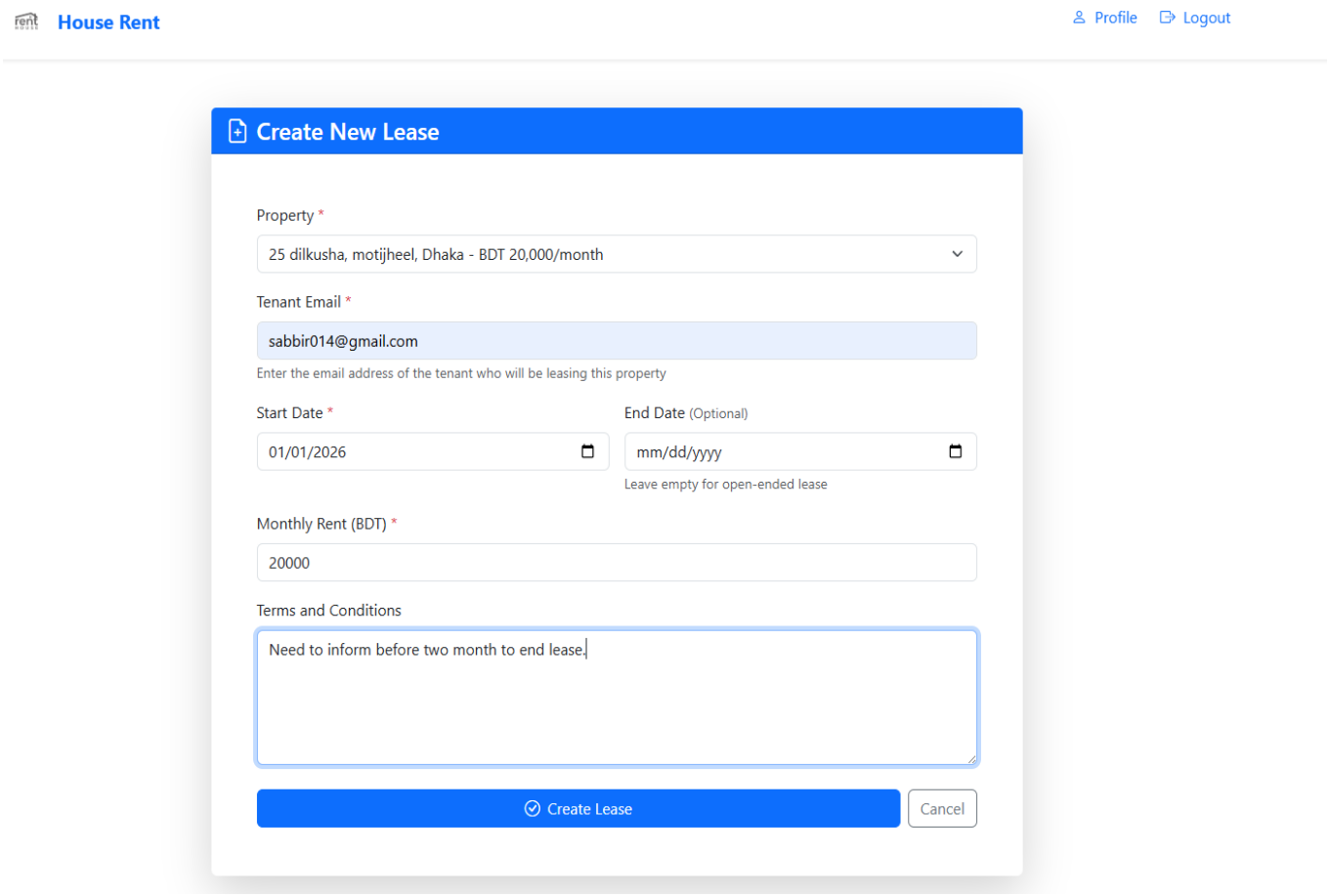


Figure A.6: Lease Management

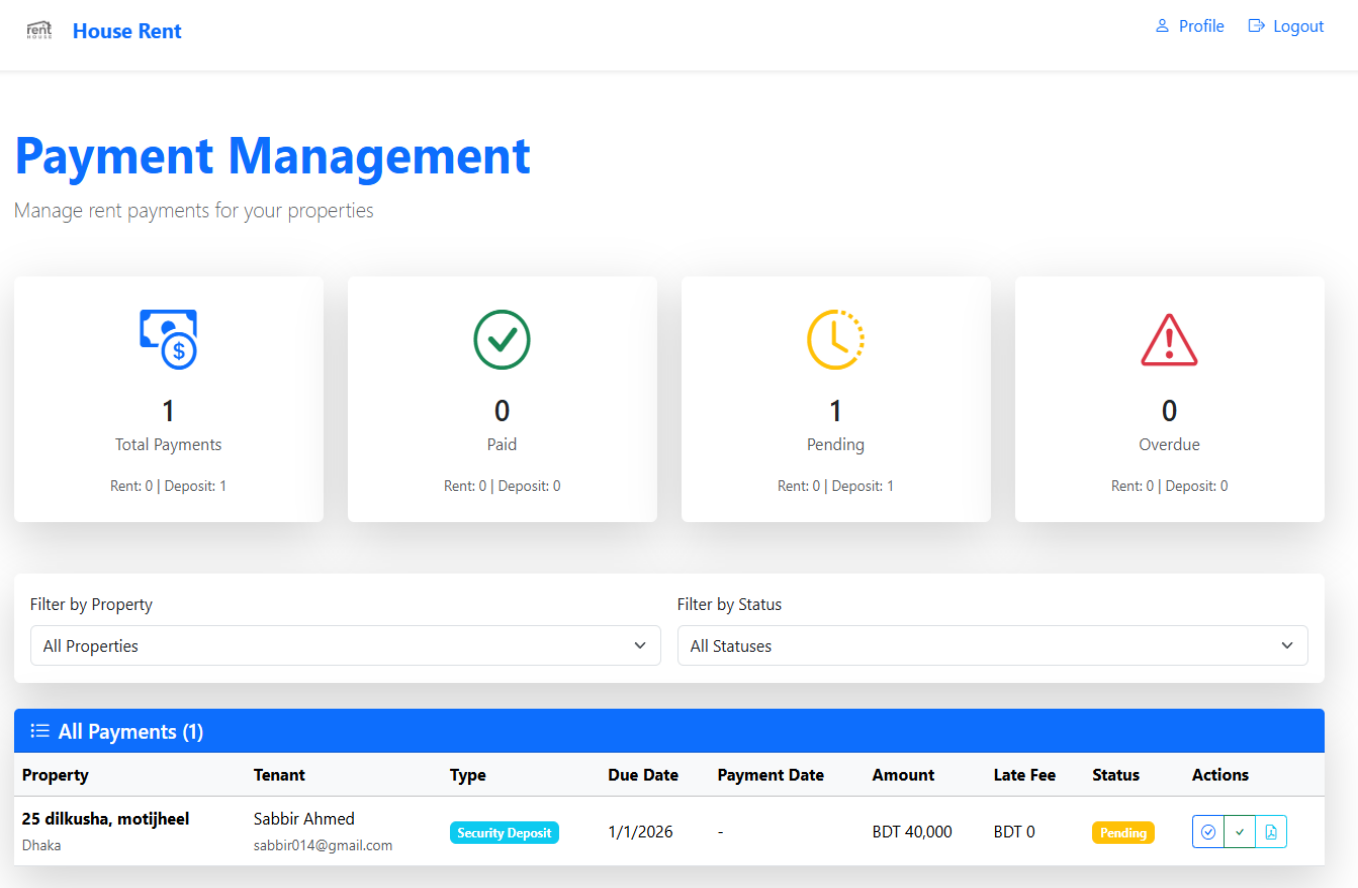
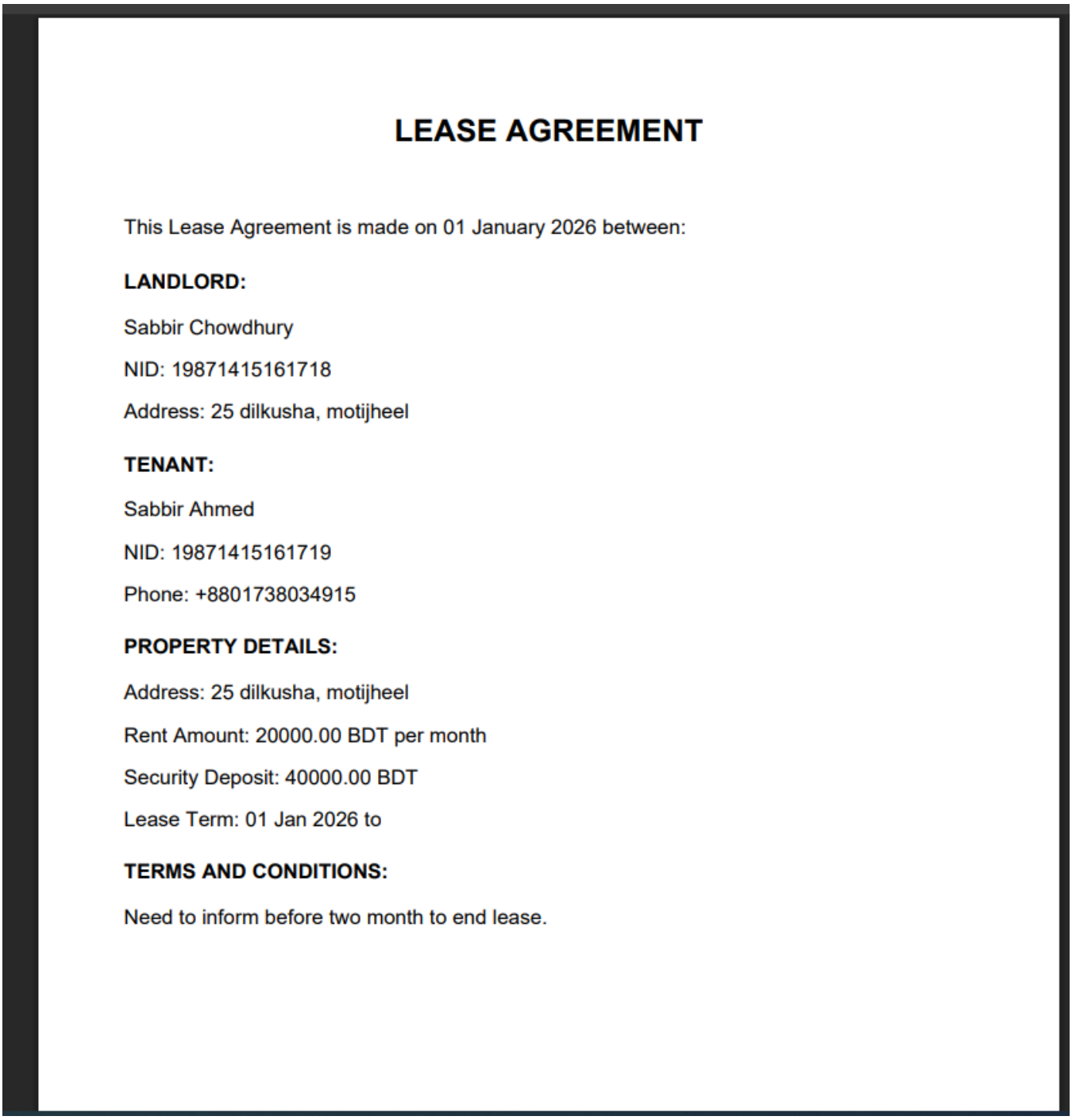


Figure A.7: Payment Management



LEASE AGREEMENT

This Lease Agreement is made on 01 January 2026 between:

LANDLORD:

Sabbir Chowdhury

NID: 19871415161718

Address: 25 dilkusha, motijheel

TENANT:

Sabbir Ahmed

NID: 19871415161719

Phone: +8801738034915

PROPERTY DETAILS:

Address: 25 dilkusha, motijheel

Rent Amount: 20000.00 BDT per month

Security Deposit: 40000.00 BDT

Lease Term: 01 Jan 2026 to

TERMS AND CONDITIONS:

Need to inform before two month to end lease.

Figure A.8: Lease Agreement Report

A.3 Tenant Interface

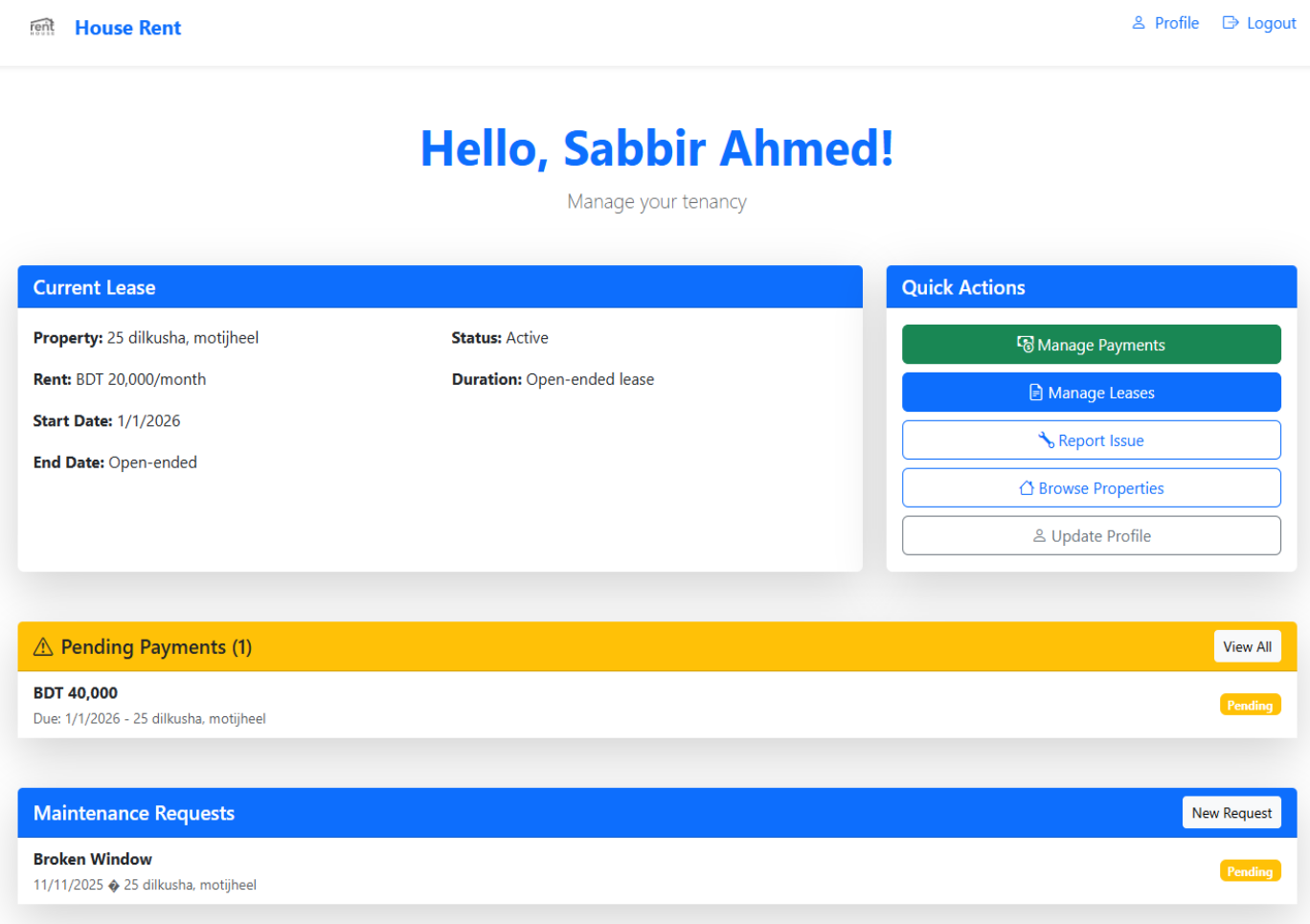


Figure A.9: Tenant Dashboard

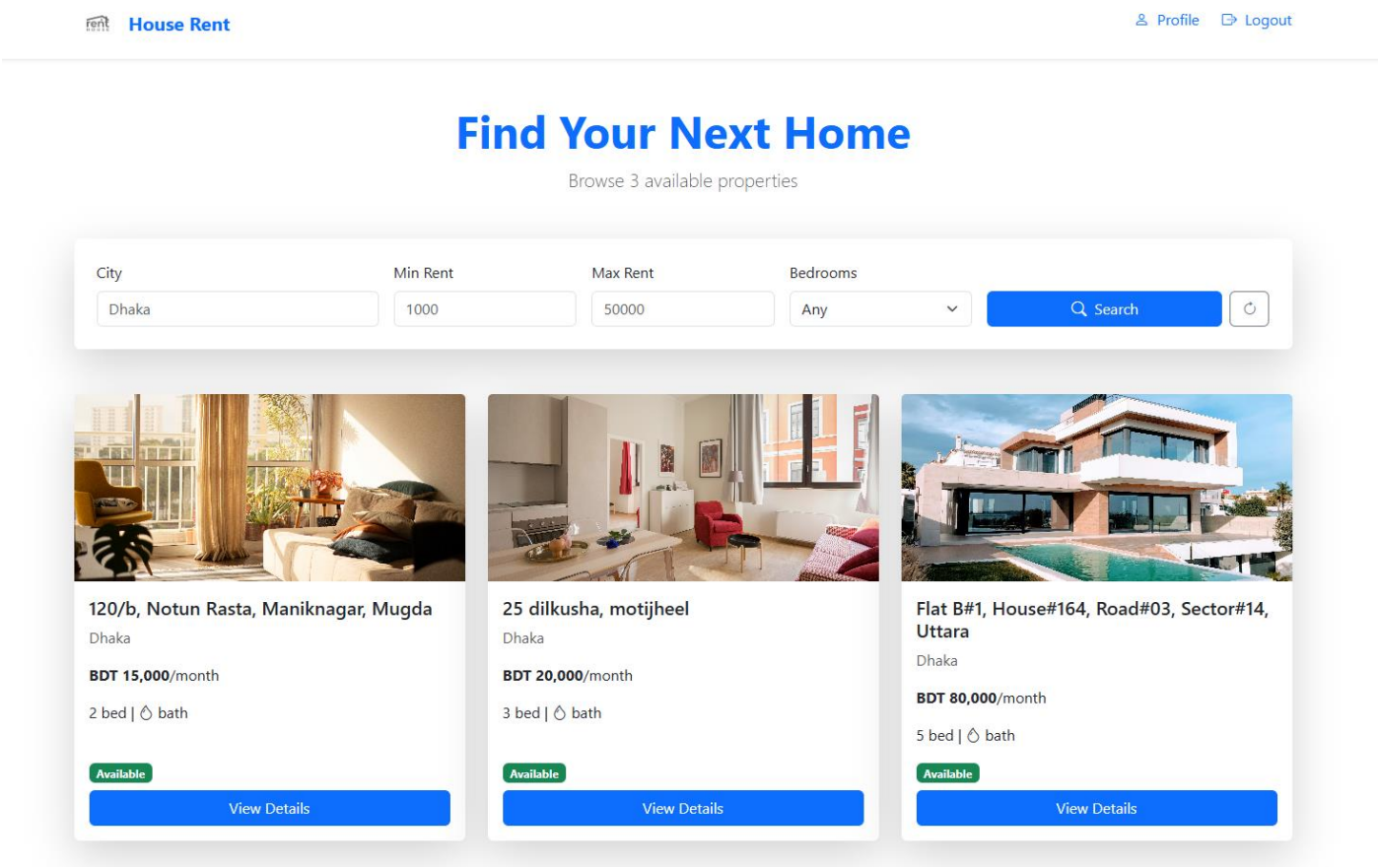


Figure A.10: Property Browsing

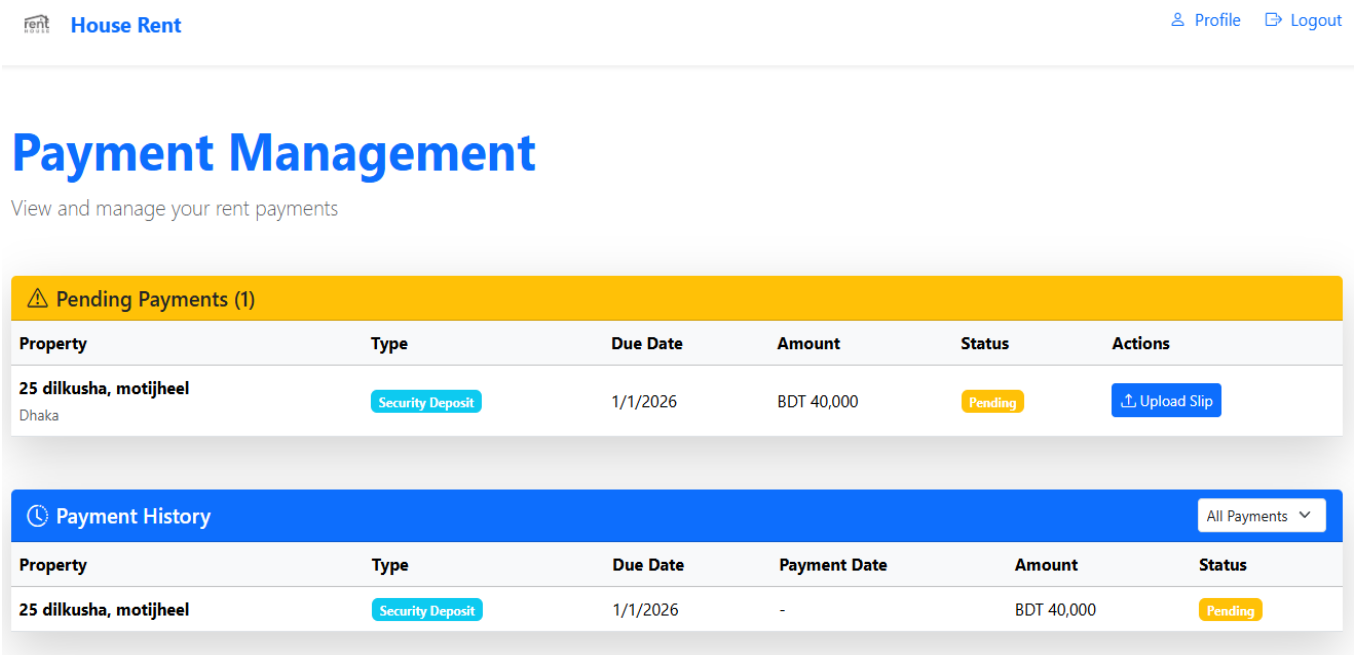


Figure A.10: Payment Management

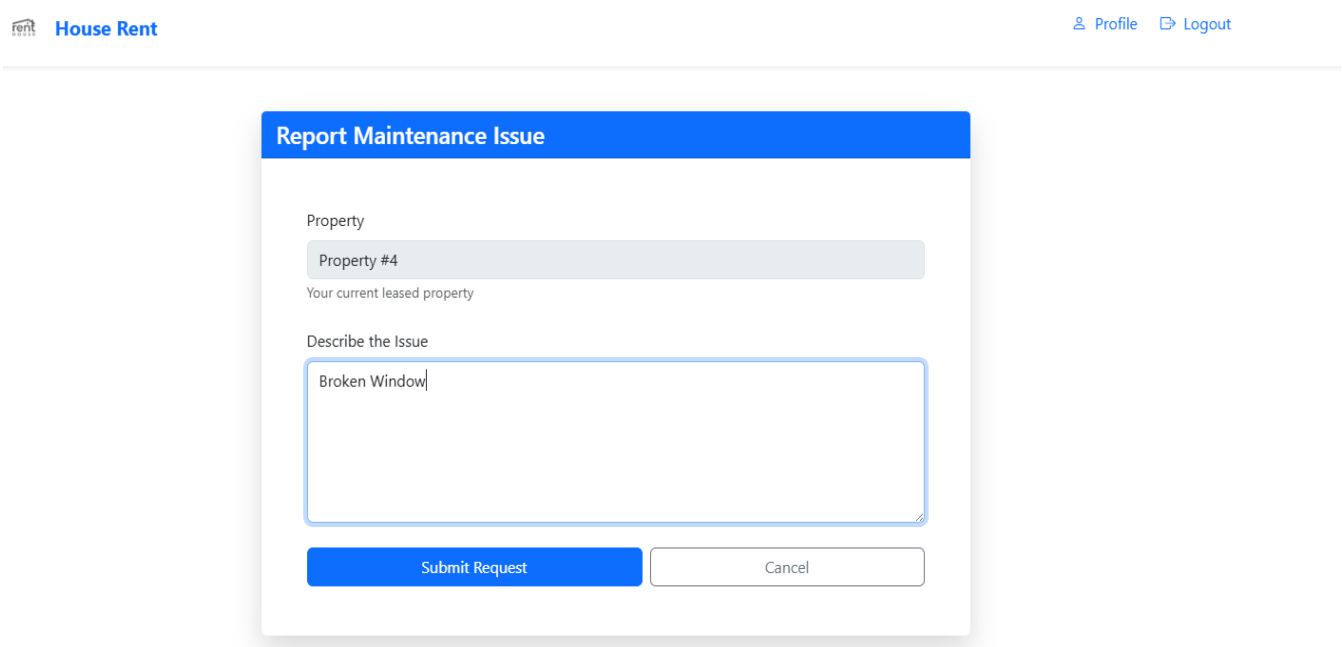


Figure A.13: Maintenance Request Submission

A.4 System Common Features

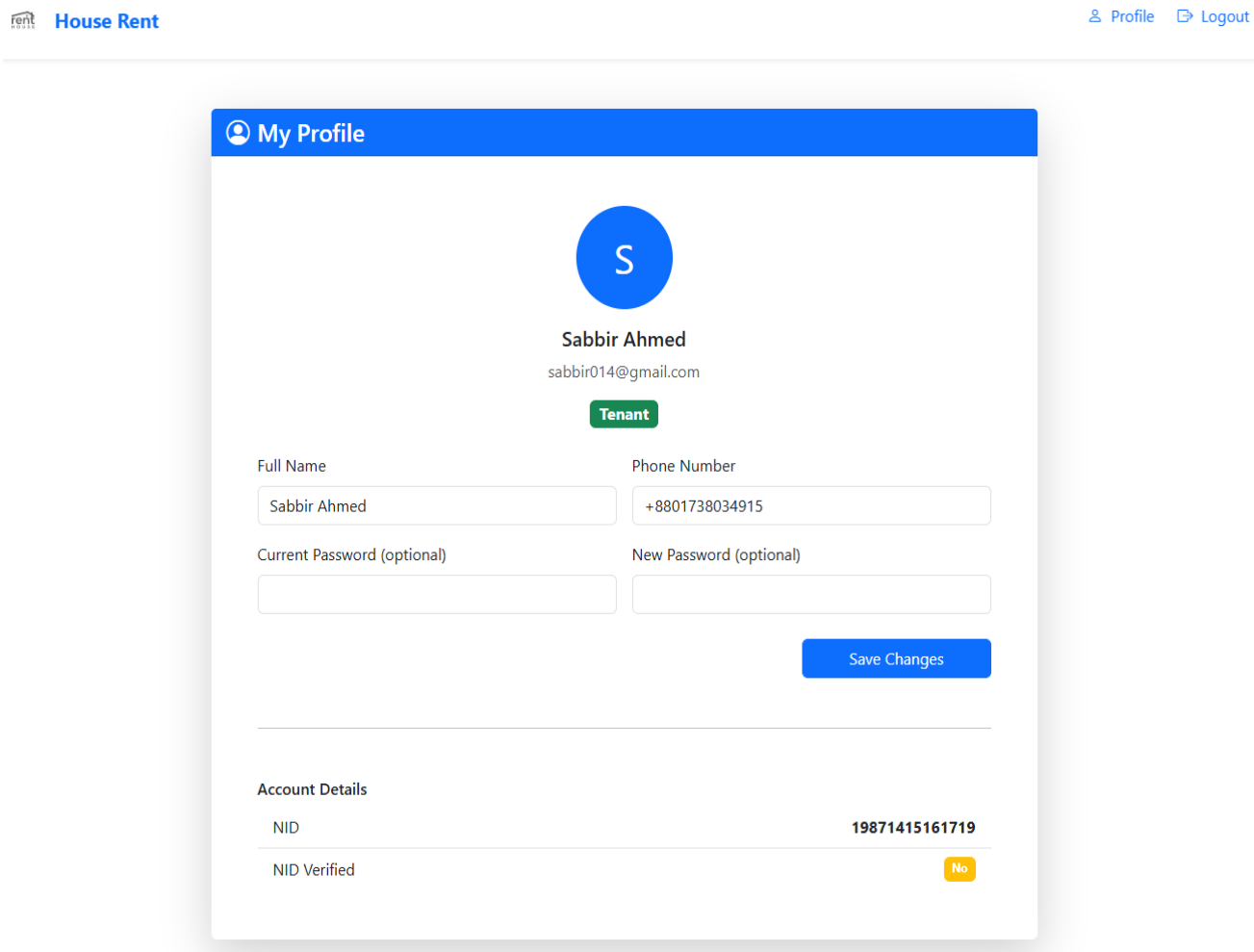


Figure A.14: User Profile Management

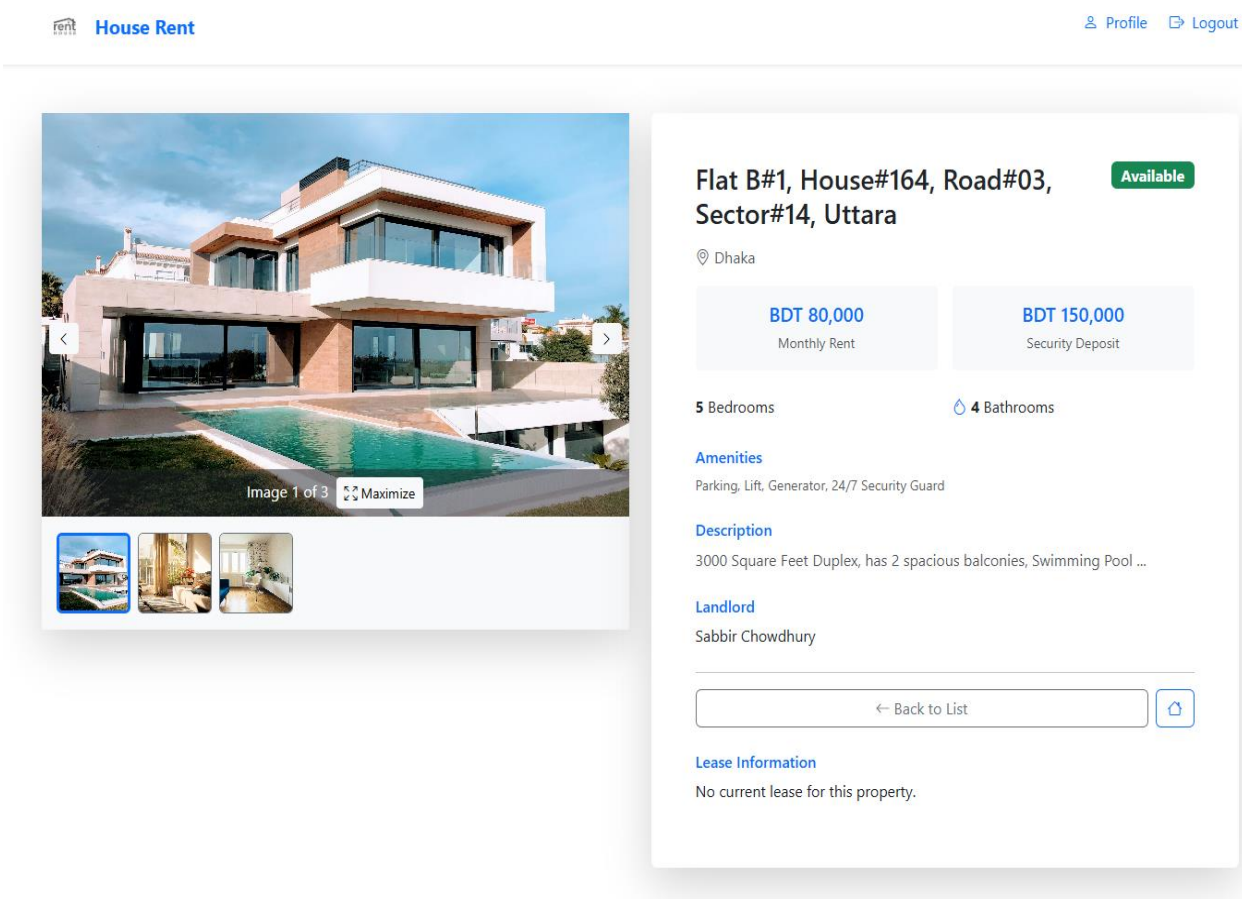


Figure A.15: Property Details View

Appendix B: API Documentation

B.1 User Endpoints

- POST /api/User/register - User registration
- POST /api/User/login - User authentication
- GET /api/User/profile - Get user profile
- PUT /api/User/profile - Update user profile

B.2 Property Endpoints

- GET /api/Property - Get all properties

- GET /api/Property/{id} - Get property by ID
- POST /api/Property - Create property
- PUT /api/Property/{id} - Update property
- DELETE /api/Property/{id} - Delete property
- GET /api/Property/search - Search properties

B.3 Lease Endpoints

- POST /api/Lease - Create lease
- GET /api/Lease/{id} - Get lease by ID
- PUT /api/Lease/{id}/renew - Renew lease
- PUT /api/Lease/{id}/end - End lease
- GET /api/Lease/{id}/document - Generate lease PDF

B.4 Payment Endpoints

- GET /api/payments/{id} - Get payment by ID
- GET /api/payments/lease/{leaseId} - Get payments by lease
- POST /api/payments/{id}/slip - Upload payment slip
- PUT /api/payments/{id}/status - Update payment status
- POST /api/payments/{id}/verify - Verify payment

B.5 Maintenance Endpoints

- POST /api/maintenance - Create maintenance request
- GET /api/maintenance/{id} - Get request by ID
- PUT /api/maintenance/{id}/status - Update request status
- GET /api/maintenance/tenant/{tenantId} - Get requests by tenant

Appendix C: Database Schema

The Entity-Relationship diagram in Section 4.2.3 provides a comprehensive view of the database schema. Key tables include:

- **Users**: User accounts and authentication
- **Properties**: Property listings and details
- **Leases**: Lease agreements
- **RentPayments**: Payment records
- **MaintenanceRequests**: Maintenance issue tracking
- **PropertyImages**: Property image storage
- **UtilityBills**: Utility bill tracking (future enhancement)

Appendix D: Code Samples

D.1 JWT Token Generation

```
//csharp
public async Task<string> GenerateTokenAsync(User user)
{
    var claims = new[]
    {
        new Claim(ClaimTypes.NameIdentifier, user.UserId.ToString()),
        new Claim(ClaimTypes.Email, user.Email),
        new Claim(ClaimTypes.Role, user.Role.ToString())
    };

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Jwt:Key"]));
    var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        issuer: _configuration["Jwt:Issuer"],
        audience: _configuration["Jwt:Audience"],
```

```
        claims: claims,
        expires: DateTime.Now.AddHours(3),
        signingCredentials: credentials
    );

    return new JwtSecurityTokenHandler().WriteToken(token);
}
```

D.2 Property Search Implementation

```
//csharp
public async Task<PaginatedResult<Property>> SearchPropertiesAsync(
    string? city, decimal? minRent, decimal? maxRent,
    int? bedrooms, int page, int pageSize, string? sortBy, string? sortDirection)
{
    var query = _unitOfWork.GetRepository<Property>()
        .GetQueryable()
        .Include(p => p.Images)
        .Where(p => p.IsAvailable);

    if (!string.IsNullOrEmpty(city))
        query = query.Where(p => p.City.Contains(city));

    if (minRent.HasValue)
        query = query.Where(p => p.RentAmount >= minRent.Value);

    if (maxRent.HasValue)
        query = query.Where(p => p.RentAmount <= maxRent.Value);

    if (bedrooms.HasValue)
        query = query.Where(p => p.Bedrooms == bedrooms.Value);
```

```
    // Apply sorting and pagination
    var totalItems = await query.CountAsync();
    var items = await query
        .Skip((page - 1) * pageSize)
        .Take(pageSize)
        .ToListAsync();

    return new PaginatedResult<Property>(items, totalItems, page, pageSize);
}
```

Appendix E: Source Code Repository

E.1 Repository Information

GitHub Repository: https://github.com/SabbirAhmedChowdhury/HouseRentSystem.git

Repository Details:

- **Platform**: GitHub
- **Visibility****: Public
- **Primary Branch**: main
- **Total Commits**: 25+
- **Language Distribution**:
  - JavaScript: 53.8%
  - C#: 45.0%
  - Other: 1.2%