TROY UNIVERSITY

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY





FINAL PROJECT

COURSE: OBJECT-ORIENTED PROGRAMMING

Apollo Resident Management

Group: **[GROUP]**

Supervisor: **Assoc. Prof. Dr. Cao Tuan Dung**

List of students:

No.	Full Name	Student ID
1	[STUDENT_1]	[ID_1]
2	[STUDENT_2]	[ID_2]
3	[STUDENT_3]	[ID_3]
4	[STUDENT_4]	[ID_4]
5	[STUDENT_5]	[ID_5]

HANOI, January 2026

Troy University Resources

TABLE OF CONTENTS

TABLE OF CONTENTS 2

LIST OF FIGURES 4

LIST OF TABLES 6

PREFACE 8

GROUP MEMBER TASK ASSIGNMENT 9

1 PROBLEM ANALYSIS 10

1.1 Problem Requirements Description . . . 10
1.2 Problem Investigation . . . 10
1.3 Identify Basic Business Information for the Problem . . . 11
1.4 Develop the Business Process Diagram and Functional Decomposition . . . 11
1.5 Simple Project Plan . . . 15

2 PROBLEM REQUIREMENTS SPECIFICATION 17

2.1 General Introduction . . . 17
2.1.1 Actors (System Stakeholders) . . . 17
2.1.2 Use Cases . . . 17
2.1.3 Table of Actors . . . 19
2.1.4 Use Case Identification by Actor . . . 19
2.2 Use Case Diagram . . . 23
2.2.1 Overall Use Case Diagram . . . 23
2.2.2 Level 2 Decomposed Use Case Diagram . . . 25
2.3 Use Case Specifications . . . 27
2.4 Non-Functional Requirements . . . 45
2.4.1 Functionality (General Functional Requirements) . . . 45
2.4.2 Usability . . . 45
2.4.3 Performance (Efficiency) . . . 45
2.4.4 Reliability (Availability and Fault Tolerance) . . . 46
2.4.5 Security (Safety and Access Control) . . . 46
2.4.6 Maintainability . . . 46
2.4.7 Technical Constraints (Database, Technology, Environment) . . . 46

3 REQUIREMENTS ANALYSIS 47

3.1 Identify Analysis Classes . . . 47
3.2 Build Sequence Diagrams . . . 51
3.3 Build Analysis Class Diagram . . . 58
3.4 Build Entity Relationship Diagram (ERD) . . . 59

4 SYSTEM DESIGN 63
4.1 Architecture Design . . . 63
4.1.1 Selected Software Architecture . . . 63
4.1.2 Applying MVC Architecture in the Project . . . 63
4.1.3 UML Package Diagram cho dự án của bạn . . . 65
4.2 Database Design . . . 66
4.2.1 Detailed Specifications of Selected Database Tables . . . 68

5 DEMONSTRATION SYSTEM DEVELOPMENT 71
5.1 Libraries and Tools Used . . . 71
5.2 Demonstration System Results . . . 71
5.2.1 System Packaging Structure . . . 71
5.2.2 Packaged Product 1: Backend (Server) . . . 71
5.2.3 Packaged Product 2: Frontend (Client) . . . 72
5.2.4 Packaged Product 3: Database . . . 72
5.3 Demonstration of System Functions Interface . . . 73
5.3.1 Login . . . 73
5.3.2 Account Registration . . . 74
5.3.3 Online Public Services . . . 75
5.3.4 News / Announcements . . . 76
5.3.5 Admin Dashboard . . . 77

CONCLUSION AND FUTURE DEVELOPMENT 78

REFERENCES 80

Troy University Resources

LIST OF FIGURES

1-1 Input–Process–Output (IPO) Diagram 11
1-2 Activity Diagram with Swimlanes for Resident Management System 12
1-3 Business Function Diagram (BFD) – Core Business Functions 12
1-4 Business Function Diagram (BFD) – Supporting and Admin Functions 13
2-1 Overall Use Case Diagram 24
2-2 Decomposed Use Case: UC-01 Register New Resident 25
2-3 Decomposed Use Case: UC-03 Change Residency Status 25
2-4 Decomposed Use Case: UC-08 Generate Invoice 26
2-5 Activity Diagram for “Register New Resident” 28
2-6 Activity Diagram for “Change Residency Status” 31
2-7 Activity Diagram for “Generate Invoice” 37
2-8 Activity Diagram for “Upload/Download File” 43
3-1 Initial decomposition of use case “UC-01: Register New Resident” 47
3-2 Initial decomposition of use case “UC-02: Update Resident Profile” 47
3-3 Initial decomposition of use case “UC-03: Change Residency Status” 47
3-4 Initial decomposition of use case “UC-04: Manage Household” 48
3-5 Initial decomposition of use case “UC-05: View Resident Information” 48
3-6 Initial decomposition of use case “UC-06: Manage Fee Definitions” 48
3-7 Initial decomposition of use case “UC-07: Calculate Service Fees” 48
3-8 Initial decomposition of use case “UC-08: Generate Invoice” 49
3-9 Initial decomposition of use case “UC-09: Mark Invoice as Paid” 49
3-10 Initial decomposition of use case “UC-10: View Invoices and Payments” 49
3-11 Initial decomposition of use case “UC-11: Submit Feedback” 50
3-12 Initial decomposition of use case “UC-12: Manage Feedback” 50
3-13 Initial decomposition of use case “UC-13: Upload / Download File” 50
3-14 Initial decomposition of use case “UC-14: Login / Logout” 50
3-15 Initial decomposition of use case “UC-15: Manage Roles and Permissions” 51
3-16 Sequence Diagram for use case “UC-01: Register New Resident” 51
3-17 Sequence Diagram for use case “UC-02: Update Resident Profile” 52
3-18 Sequence Diagram for use case “UC-03: Change Residency Status” 52
3-19 Sequence Diagram for use case “UC-04: Manage Household” 52
3-20 Sequence Diagram for use case “UC-05: View Resident Information” 53
3-21 Sequence Diagram for use case “UC-06: Manage Fee Definitions” 53
3-22 Sequence Diagram for use case “UC-07: Calculate Service Fees” 53
3-23 Sequence Diagram for use case “UC-08: Generate Invoice” 54
3-24 Sequence Diagram for use case “UC-09: Mark Invoice as Paid” 54
3-25 Sequence Diagram for use case “UC-10: View Invoices and Payments” 54
3-26 Sequence Diagram for use case “UC-11: Submit Feedback” 55
3-27 Sequence Diagram for use case “UC-12: Manage Feedback” 55

3-28 Sequence Diagram for use case “UC-13: Upload / Download File” 56
3-29 Sequence Diagram for use case “UC-14: Login / Logout” 56
3-30 Sequence Diagram for use case “UC-15: Manage Roles and Permissions” 57
3-31 Analysis Class Diagram 58
3-32 ERD for Resident and Household Management System 59
3-33 ERD for Fee and Invoice Management 60
3-34 ERD for Feedback and Attachments 60
3-35 ERD for Users and Authorization 61
3-36 ERD for File Management and File Linking with Other Functions 62
4-1 UML Package Diagram 65
4-2 ERD for User, Role, Permission, Feedback and Fee 67

Troy University Resources

LIST OF TABLES

1-1 Description of Functions in the BFD 14
1-2 Project Work Plan 15
1-3 Risk Management Table 16
2-1 Table of Actors 19
2-2 Use Case List for Resident Management System 21
2-3 Use Case Specification UC001 “Register New Resident” 27
2-4 Input data for personal information 27
2-5 Use Case Specification UC-02 “Update Resident Profile” 29
2-6 Input data for updating a resident profile 29
2-7 Use Case Specification UC-03 “Change Residency Status” 30
2-8 Input data for changing residency status 30
2-9 Use Case Specification UC-04 “Manage Household” 31
2-10 Input data for managing households 32
2-11 Use Case Specification UC-05 “View / Search Resident Information” 33
2-12 Input data for searching and viewing resident information 33
2-13 Use Case Specification UC-06 “Manage Fee Definitions” 34
2-14 Input data for managing service fees 34
2-15 Use Case Specification UC-07 “Calculate Service Fees” 35
2-16 Input data for calculating service fees 35
2-17 Use Case Specification UC-07 “Calculate Service Fees” 36
2-18 Input data for generating an invoice 37
2-19 Use Case Specification UC-09 “Mark Invoice as Paid” 38
2-20 Input data for confirming invoice payment 38
2-21 Use Case Specification UC-10 “View Invoices and Payments” 39
2-22 Input data for viewing invoices and payments 39
2-23 Use Case Specification UC-11 “Submit Feedback” 40
2-24 Input data for submitting feedback 40
2-25 Use Case Specification UC-12 “Manage Feedback” 41
2-26 Input data for managing feedback 41
2-27 Use Case Specification UC-13 “Upload / Download File” 42
2-28 Input data for file operations 42
2-29 Use Case Specification UC-14 “Login / Logout” 43
2-30 Input data for authentication 44
2-31 Use Case Specification UC-15 “Manage Roles and Permissions” 44
2-32 Input data for managing roles and permissions 44
4-1 Specification of Table “users” 68
4-2 Specification of Table “roles” 68
4-3 Specification of Table “permissions” 69
4-4 Specification of Table “fees” 69

4-5 Specification of Table “feedback” 70
5-1 List of Libraries and Tools Used 71
5-2 System Statistics 73

Troy University Resources

PREFACE

In the context of rapid digital transformation, electronic administrative management systems are playing an increasingly important role in improving management efficiency, enhancing information transparency, and reducing manual administrative procedures. In particular, within residential areas, apartment buildings, and modern urban complexes, the management of residents, households, service fees, and related invoices requires a system that is highly organized, accurate, and easily scalable. Motivated by these practical needs, our team selected the topic "Resident Management System", inspired by the electronic administrative management application VNeID, with the aim of developing a centralized, systematic, and efficient solution for resident management.

This topic not only has practical significance in modeling and addressing common resident management problems, but also holds strong academic value by demonstrating and applying the principles of Object-Oriented Programming (OOP) to a real-world problem domain. Through the development of this system, the team had the opportunity to explore and apply key OOP concepts such as Domain Modeling, Encapsulation, Abstraction, Inheritance, and Polymorphism, thereby enhancing software design thinking and the ability to build systems that are well-structured, maintainable, and extensible.

In terms of content, the project focuses on developing a web-based resident management system in which core entities such as Resident, Household, Invoice, InvoiceItem, and FeePolicy are modeled as objects with both state and behavior, rather than as simple data structures. The system supports the management of resident information, households, and service-related invoices through flexible fee calculation policies implemented using the Strategy Pattern. The approach to the project involved analyzing real-world business requirements, transforming them into a domain model, and then implementing the system using modern technologies such as Next.js, Spring Boot, Spring Data JPA, REST APIs, and MySQL, combined with DevOps tools to support system deployment and completion.

Throughout the development process, the team progressively refined the system, from architectural design and object modeling to feature implementation, integration, and testing. The final outcome is a resident management system with a clear structure that effectively demonstrates OOP principles and provides strong potential for future scalability, thereby fulfilling both the academic objectives and the practical requirements of the project.

GROUP MEMBER TASK ASSIGNMENT

Full Name	Summary of Work Done	Evaluation Based on Task Division
[STUDENT_4]	Designed overall OOP architecture and backend core. Built Entity–Service–Repository–Controller structure, applied OOP principles (inheritance, encapsulation, polymorphism) and design patterns like Strategy for FeePolicy. Explained DTO usage and separation of business logic.	10 / 10
[STUDENT_5]	Designed main entities and class structure: User, Role, Permission, Profile, Fee, Feedback, etc. Built class diagram and defined relationships (association, aggregation, composition). Clarified Entity vs DTO and Enum vs Entity.	9 / 10
[STUDENT_1]	Implemented service layer and business logic: UserService, ProfileService, FeeService, WantedService. Explained processing flow (Controller → Service → Repository) and applied SOLID principles.	9.5 / 10
[STUDENT_3]	Implemented security and authentication: JWT, role–permission system, SecurityConfig, filters, decoder. Explained authorization logic and separation of security layer.	9 / 10
[STUDENT_2]	Built frontend modules (Login, Register, Profile, News, Services). Mapped API request/response with backend, used DTOs, and maintained API contract between frontend and backend.	9 / 10

Troy University Resources

CHAPTER 1. PROBLEM ANALYSIS

1.1. Problem Requirements Description

In many residential areas such as apartment buildings, dormitories, and urban housing complexes, resident-related information is often managed using fragmented or manual methods. Data about residents, households, service fees, and invoices is frequently stored in separate systems or handled through paperwork, which leads to inefficiencies, data inconsistency, and difficulties in tracking changes over time. Managing resident status changes such as new registrations, temporary residence, or moving out also becomes complex and error-prone without a centralized system.

In addition, residential management involves not only storing information but also enforcing business rules and workflows. For example, residents belong to households, households are associated with apartments, and service fees must be calculated according to different policies such as fixed fees, water usage, or parking services. Treating these elements as simple data records makes it difficult to maintain correctness, extend functionality, or adapt the system to new requirements.

Therefore, the problem addressed in this project is the design and implementation of a Resident Management System that can model real-world residential management in a structured and maintainable way. The system must support centralized management of residents, households, service fees, and invoices, while clearly defining the relationships, states, and behaviors of each entity. At the same time, the system should be flexible enough to accommodate different fee calculation strategies and future extensions, reflecting both practical management needs and sound software engineering principles.

1.2. Problem Investigation

To clearly understand the business requirements of the resident management problem, the project team conducted a survey of real-world administrative documents and related materials used in residential and population management. The main purpose of this survey was to identify essential information, workflows, and constraints that should be reflected in the system design.

Specifically, the team examined common administrative forms such as household registration records, temporary residence registration forms, and temporary absence declarations. In addition, the structure and data presentation of the electronic administrative management application VNeID were reviewed as a reference. From these materials, the team identified core resident information, including personal details, residency status, household relationships, and changes in residency over time.

Through this analysis, it was observed that a resident is not merely a static data record, but an entity with multiple states and business behaviors, such as new registration, temporary residence, and moving out. Furthermore, a household functions as a management unit that groups multiple residents together, reflecting real-life family or cohabitation relationships.

For service fees and billing management, since no standardized public administrative forms were available, the team discussed and designed their own conceptual forms based on typical operations in residential buildings and apartment complexes. These forms include invoice information, lists of service fees, payment deadlines, and invoice status. As a result, the system requirements were defined to support multiple fee calculation methods, such as flat fees, water usage–based fees, and parking fees.

The outcomes of this problem analysis served as the foundation for defining system requirements and developing the domain model, which guided subsequent stages including object modeling, system design, and implementation.

1.3. Identify Basic Business Information for the Problem

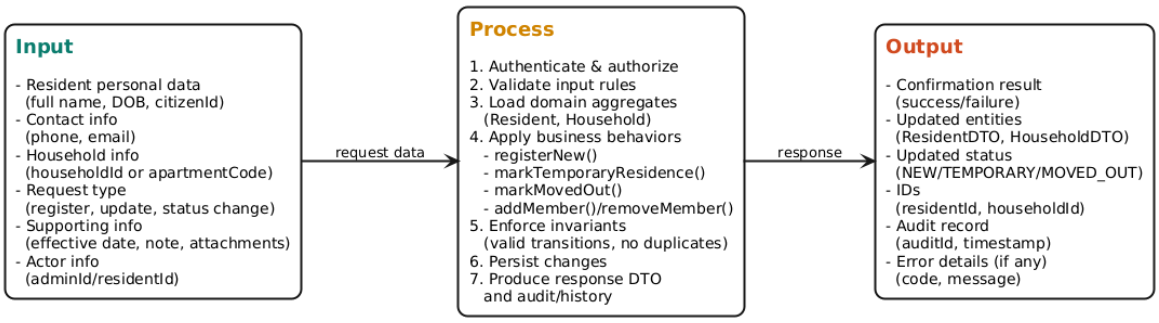


Figure 1-1: Input–Process–Output (IPO) Diagram

1.4. Develop the Business Process Diagram and Functional Decomposition

The workflow begins in the Building Admin lane, where the administrator logs into the system, creates a household if necessary, enters resident information, and submits a request to register a new resident. The process then moves to the System lane, where the system authenticates and authorizes the administrator, validates the input data, and checks for duplicate citizen identification. If a duplicate is found, the system rejects the request and returns an error, and the admin must correct the data. If the data is valid, the system creates a new Resident entity, sets its initial status, stores it in the database, and links it to the appropriate household.

After the resident is successfully registered, the system automatically generates initial service fees, creates an invoice with invoice items, and sends a notification. The workflow then continues in the Resident lane, where the resident logs in, views household and invoice details, selects a payment method, and submits the payment. Finally, the system validates the payment, marks the invoice as paid, stores the payment timestamp and receipt, and returns the confirmation to the resident. The diagram clearly shows the sequence of actions, decision points, and interactions among different actors, providing a comprehensive view of the end-to-end business process from resident registration to fee payment.

Troy University Resources

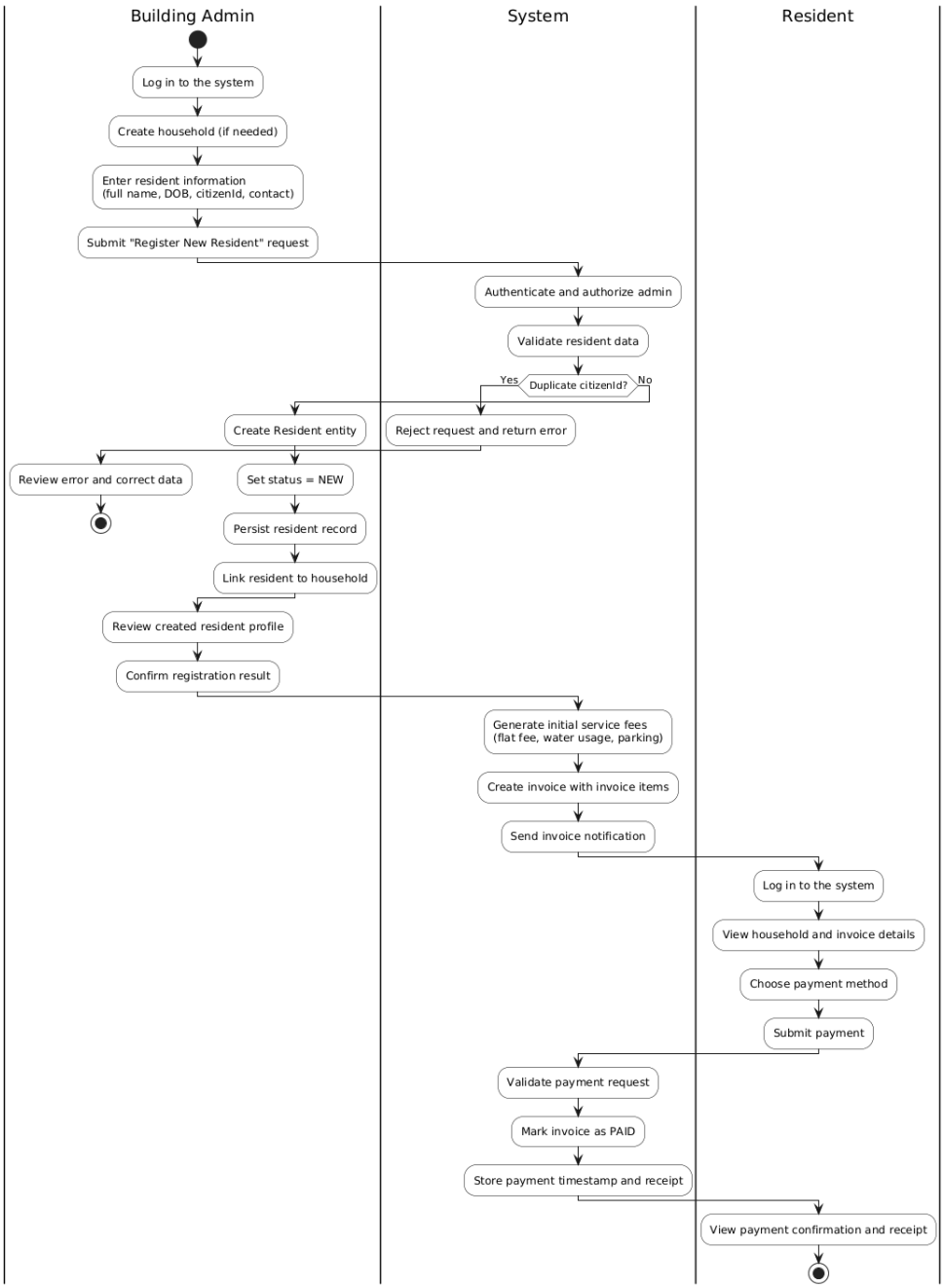


Figure 1-2: Activity Diagram with Swimlanes for Resident Management System

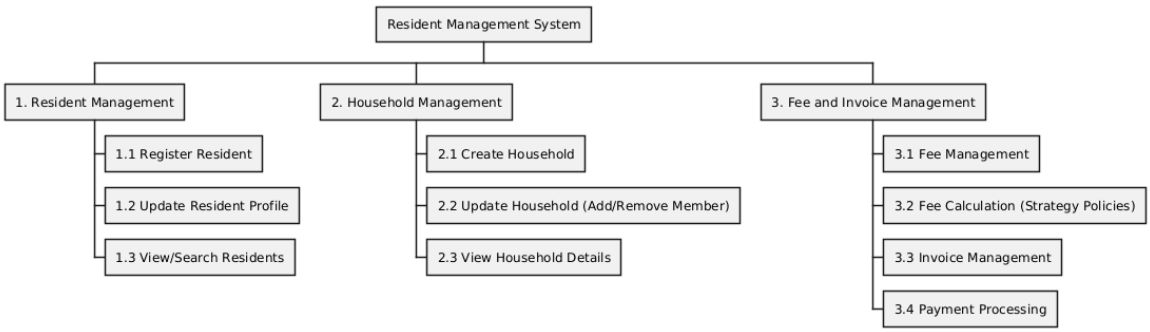


Figure 1-3: Business Function Diagram (BFD) – Core Business Functions

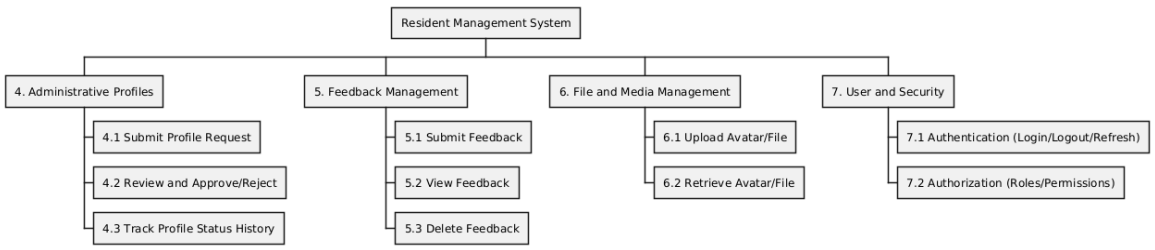


Figure 1-4: Business Function Diagram (BFD) – Supporting and Admin Functions

Troy University Resources

Table 1-1: Description of Functions in the BFD

Function Name	Description	Feasibility Assessment (People, Time, Technology, Environment)
Resident Management	Manage resident data, including registration, update, and status changes.	People: Team has both FE and BE skills. Time: Moderate, fits course timeline. Technology: Supported by Spring Boot, JPA, REST. Environment: Suitable for web-based deployment.
Household Management	Manage households and relationships between residents.	People: Team can handle domain modeling. Time: Low–medium complexity. Technology: Easy with relational DB and ORM. Environment: Works well in centralized system.
Fee Management	Create, update, and delete service fees.	People: Requires backend logic and admin UI. Time: Medium. Technology: JPA + REST supports CRUD well. Environment: Common in residential systems.
Fee Calculation (Strategy)	Calculate fees using different policies (flat, water, parking).	People: Requires good OOP understanding. Time: Medium. Technology: Java supports Strategy Pattern well. Environment: Fits modular system design.
Invoice Management	Generate invoices and manage invoice items and status.	People: Team can divide FE/BE tasks. Time: Medium–high. Technology: Supported by database and REST APIs. Environment: Needed in real residential management.
Payment Processing	Mark invoices as paid and store payment history.	People: Needs careful implementation. Time: Medium. Technology: Simple status update, no real payment gateway. Environment: Simulated environment is sufficient.
Administrative Profiles	Manage administrative requests and their status.	People: Requires workflow design. Time: Medium. Technology: Spring + REST handles well. Environment: Fits e-government style systems.
Feedback Management	Allow users to send feedback and admins to manage it.	People: Easy for both FE and BE. Time: Low. Technology: CRUD + file upload. Environment: Common in web apps.
File & Media Management	Upload and retrieve avatars and attachments.	People: Team has FE/BE experience. Time: Low–medium. Technology: Multipart upload supported by Spring. Environment: Works in cloud or local server.
Authentication	Handle login, token refresh, and logout.	People: Needs security knowledge. Time: Medium. Technology: Spring Security, JWT available. Environment: Web-based system supports it well.
Authorization	Manage roles and permissions.	People: Requires understanding of access control. Time: Medium. Technology: Spring Security supports RBAC. Environment: Necessary for admin vs user separation.

1.5. Simple Project Plan

Table 1-2: Project Work Plan

Task	Time (hours)	Number of People
1. Analysis and Overall Design	30	3
1.1 Survey business requirements and documents	8	3
1.2 Analyze the problem and identify business groups	6	3
1.3 Design overall system architecture	10	2
1.4 Create detailed task allocation plan	6	3
2. OOP and Data Modeling Design	40	2
2.1 Design entities, enums, and data relationships	15	1
2.2 Build class diagram and analyze OOP relationships	10	1
2.3 Design DTOs and mappers	8	1
2.4 Review model with business requirements	7	2
3. Backend Core Development	70	3
3.1 Build Controller–Service–Repository architecture	12	1
3.2 Implement entities and repositories	15	1
3.3 Develop service layer (User, Profile, Fee, etc.)	20	1
3.4 Apply OOP principles and Design Patterns (FeePolicy Strategy)	13	1
3.5 Backend testing and bug fixing	10	2
4. Security and Authorization	35	1
4.1 Design JWT authentication	10	1
4.2 Implement Role–Permission model	10	1
4.3 Configure Spring Security	10	1
4.4 Security testing and permission validation	5	1
5. Frontend and API Contract	50	1
5.1 Design main user interface	12	1
5.2 Build Login, Profile, Service pages	18	1
5.3 Connect APIs and handle data	12	1
5.4 UI testing and user flow validation	8	1
6. System Integration	20	3
6.1 Integrate frontend and backend	10	3
6.2 End-to-end business testing	10	3
7. Report Writing and Defense Preparation	35	5
7.1 Write technical report	15	3
7.2 Write OOP and design pattern sections	8	2
7.3 Prepare presentation slides	6	3
7.4 Practice presentation and Q&A	6	5

Troy University Resources

Table 1-3: Risk Management Table

Task / Activity	Hazard	Risk	Level	Strategy	Mitigation
Requirement analysis	Unclear requirements	Wrong functionality	High	Risk mitigation	Team meetings, document requirements, confirm with lecturer
System design	Incomplete design	Hard to maintain and extend	Medium	Risk mitigation	Design review, draw diagrams before coding
Backend development	Logic errors	System works incorrectly	High	Risk mitigation	Test each function, cross-check among members
Frontend development	Poor usability	Users dislike the system	Medium	Risk mitigation	Early UI testing, improve based on feedback
Security	Wrong authorization	Data leakage	High	Risk elimination	Apply Spring Security, carefully test permissions
FE–BE integration	API mismatch	System does not run	High	Risk mitigation	Agree on API contract, integration testing
Time management	Schedule delay	Cannot meet deadline	High	Risk mitigation	Create plan, track progress weekly
Human resources	Members are busy	Lack of manpower	Medium	Risk transfer	Assign backup tasks to other members
Testing	Not enough test time	Many bugs in demo	Medium	Risk mitigation	Reserve dedicated testing time
Report writing	Writing in a rush	Low-quality report	Low	Risk mitigation	Write gradually following project progress

CHAPTER 2. PROBLEM REQUIREMENTS SPECIFICATION

2.1. General Introduction

2.1.1. Actors (System Stakeholders)

2.1.1.1 Primary actors (direct users)

- Building Admin (Residence/Apartment Administrator)

Uses the system to manage residents, households, service fees, invoices, administrative profiles, and feedback. The building admin is responsible for daily operational tasks and data accuracy.

- Resident (Citizen/Household Member)

Uses the system to view personal and household information, check fees and invoices, submit feedback, and update limited personal profile information.

2.1.1.2 Supporting and administrative actors

- System Admin (System Administrator)

Maintains and administers the system. Responsibilities include managing user accounts, roles and permissions, system configurations, and monitoring overall system health and security.

2.1.1.3 External systems and services

- File Storage Service (Local or Cloud Storage)

Stores uploaded avatars and file attachments. The system interacts with this service to upload, retrieve, and manage files.

- Database (MySQL)

Stores all persistent data such as residents, households, fees, invoices, profiles, and feedback. In UML diagrams, it can be shown as an external system to emphasize system integration.

2.1.2. Use Cases

2.1.2.1 Authentication and Authorization

- Log in
- Log out
- Refresh token
- Manage roles
- Manage permissions
- Assign roles and permissions to users

2.1.2.2 Resident Management

- Register new resident
- Update resident profile
- Change residency status (new, temporary, moved out)
- View resident details
- Search and filter residents

Troy University Resources

2.1.2.3 Household Management

- Create household
- Update household members (add or remove)
- View household details

2.1.2.4 Fee and Invoice Management

- Manage fee categories (create, update, delete)
- Calculate service fees (flat, water, parking policies)
- Generate invoice
- View invoices
- Mark invoice as paid or confirm payment
- View payment history

2.1.2.5 Administrative Profiles (Requests and Records)

- Submit administrative profile or request
- Approve or reject profile or request
- View profile history

2.1.2.6 Feedback and Support

- Submit feedback (with optional attachment)
- View feedback list
- Delete feedback

2.1.2.7 File and Media Management

- Upload avatar or attachment
- Download or view file

2.1.2.8 Online Status (Optional)

- Update online status (ping)
- Check user online status

2.1.3. Table of Actors

Table 2-1: Table of Actors

No.	Actor Name	Actor Description
1	Building Admin	Manages residents, households, service fees, invoices, administrative profiles, and feedback. Provides most operational data to the system and receives management results.
2	Resident	Views personal and household information, views invoices and fees, submits feedback, and performs limited profile updates. Receives results of system functions related to personal data.
3	System Admin	Maintains and administers the system, manages user accounts, roles, permissions, and system configuration. Ensures stable system operation.
4	File Storage Service	External service that stores uploaded files such as avatars and attachments and provides them to the system when requested.

Relationships Between Actors

- System Admin has higher privileges than Building Admin and can manage system-wide settings, roles, and permissions.
- Building Admin manages data related to Resident and households and provides information to the system on behalf of residents.
- Resident is dependent on Building Admin for most administrative operations (registration, status changes).
- The system communicates with the File Storage Service to store and retrieve uploaded files for both Building Admin and Resident.

2.1.4. Use Case Identification by Actor

- **Building Admin**

The Building Admin is the main operational user of the system. This actor uses the system to manage all resident-related and administrative activities. The Building Admin can register new residents, update resident information, and change residency status such as new registration, temporary residence, or moved out. The admin also creates and manages households, adds or removes household members, and views household details.

In addition, the Building Admin manages service fees and invoices by creating fee definitions, generating invoices, updating invoice status, and confirming payments. The admin is also responsible for handling administrative profile requests by approving or rejecting them and managing feedback submitted by residents. All of these actions create, update, or delete core data in the system.

- **Resident**

The Resident is the end user who mainly consumes information provided by the system. This actor can log in to view personal profile information, household details, and relationships with other members. Residents can view service fees, invoices, payment status, and payment history.

Residents can also interact with the system by submitting feedback or support requests and, in some cases, updating limited personal contact information. These actions create new feedback records and may update a small part of resident data. Residents receive notifications and confirmations from the system about invoices, payments, and feedback processing.

- **System Admin**

Troy University Resources

The System Admin is responsible for maintaining and controlling the system at a technical and security level. This actor manages user accounts, roles, and permissions, defines access rules, and configures security policies. The System Admin does not manage daily resident data but controls who is allowed to perform which actions in the system.

Through these use cases, the System Admin creates and updates authorization data and monitors system status, logs, and security-related information.

- **File Storage Service (External Actor)**

The File Storage Service is an external system that interacts with the Resident Management System to store and retrieve files such as avatars and attachments. It supports use cases like uploading files, downloading files, and returning file status.

Although it does not initiate business actions like human actors, it participates in use cases related to file handling and provides responses that affect how the system completes upload and download operations.

Table 2-2: Use Case List for Resident Management System

No.	Use Case Code	Use Case Name	Use Case Description	Interacting Actor	Complexity
1	UC-01	Register New Resident	Create a new resident profile, assign initial residency status, and link to a household.	Building Admin	High
2	UC-02	Update Resident Profile	Edit resident personal information and contact details.	Building Admin	Medium
3	UC-03	Change Residency Status	Change resident status to New, Temporary, or Moved Out with business rules.	Building Admin	High
4	UC-04	Manage Household	Create household and add or remove household members.	Building Admin	Medium
5	UC-05	View Resident Information	Search, filter, and view resident and household details.	Building Admin, Resident	Low
6	UC-06	Manage Fee Definitions	Create, update, and delete service fee types.	Building Admin	Medium
7	UC-07	Calculate Service Fees	Apply different fee policies (flat, water, parking).	System (triggered by Admin)	High
8	UC-08	Generate Invoice	Create invoice and invoice items for households.	Building Admin	High
9	UC-09	Mark Invoice as Paid	Confirm payment and update invoice status.	Building Admin	Medium
10	UC-10	View Invoices and Payments	View invoices, payment status, and history.	Building Admin, Resident	Low
11	UC-11	Submit Feedback	Send feedback with optional attachment.	Resident	Low
12	UC-12	Manage Feedback	View and delete feedback.	Building Admin	Low
13	UC-13	Upload / Download File	Upload avatar or attachment and retrieve file.	Building Admin, Resident, File Storage Service	Medium
14	UC-14	Login / Logout	Authenticate users and end sessions.	All Users	Medium
15	UC-15	Manage Roles and Permissions	Create roles, permissions, and assign them to users.	System Admin	High

Relationships Between Actors and Use Cases

Building Admin → Use Cases

- UC-01 Register New Resident
- UC-02 Update Resident Profile
- UC-03 Change Residency Status
- UC-04 Manage Household
- UC-05 View/Search Resident Information
- UC-06 Manage Fee Definitions
- UC-08 Generate Invoice
- UC-09 Mark Invoice as Paid
- UC-10 View Invoices and Payments
- UC-12 Manage Feedback
- UC-13 Upload/Download File
- UC-14 Login/Logout

Resident → Use Cases

- UC-05 View/Search Resident Information (read-only scope)

Troy University Resources

- UC-10 View Invoices and Payments
- UC-11 Submit Feedback
- UC-13 Upload/Download File (attachment/avatar)
- UC-14 Login/Logout

System Admin → Use Cases

- UC-15 Manage Roles and Permissions
- UC-14 Login/Logout
- File Storage Service (External Actor) → Use Cases
- UC-13 Upload/Download File (system calls storage service for file operations)

Relationships Between Actors (Actor–Actor)

- System Admin and Building Admin are separate roles; System Admin has higher administrative scope (system-wide security and access control), while Building Admin focuses on daily operations.
- Building Admin acts on behalf of Residents for administrative operations (registration, status updates, household changes). Residents do not perform these operations directly.
- Resident depends on Building Admin for changes that require administrative approval (e.g., residency status and household membership in many real scenarios).

Relationships Between Use Cases (Use Case–Use Case)

Include relationships (mandatory sub-functions)

- UC-01 Register New Resident includes UC-14 Login/Logout (authentication required)
- UC-02 Update Resident Profile includes UC-14 Login/Logout
- UC-03 Change Residency Status includes UC-14 Login/Logout
- UC-04 Manage Household includes UC-14 Login/Logout
- UC-06 Manage Fee Definitions includes UC-14 Login/Logout
- UC-08 Generate Invoice includes UC-14 Login/Logout
- UC-09 Mark Invoice as Paid includes UC-14 Login/Logout
- UC-13 Upload/Download File includes UC-14 Login/Logout
- UC-15 Manage Roles and Permissions includes UC-14 Login/Logout

Include relationships (business logic dependency)

- UC-08 Generate Invoice includes UC-07 Calculate Service Fees
  (invoice generation must calculate fees first)
- UC-10 View Invoices and Payments includes “Retrieve Invoice Details” (internal sub-step, optional to name)

Extend relationships (optional/conditional behavior)

- UC-11 Submit Feedback extends UC-13 Upload/Download File
  (attachment upload happens only if the resident includes a file)
- UC-01 Register New Resident extends UC-04 Manage Household
  (only when a new household must be created or membership updated during registration)

Generalization relationships (optional)

- UC-14 Login/Logout can be generalized under a parent use case like Authenticate User (optional)
- UC-13 Upload/Download File can be generalized into Upload File and Download File (optional)

2.2. Use Case Diagram

2.2.1. Overall Use Case Diagram

The overall Use Case Diagram presents a high-level view of how different actors interact with the Resident Management System. It identifies the main types of users, including Resident, Building Admin, and System Admin, as well as an external File Storage Service, and shows the major functions that the system provides to each of them.

Through this diagram, we can see that the Building Admin is responsible for most operational use cases, such as registering residents, managing households, defining fees, generating invoices, and handling feedback. The Resident mainly uses the system to view personal and household information, check invoices and payment status, and submit feedback. The System Admin focuses on system-level tasks such as managing roles and permissions and maintaining security. The File Storage Service supports file-related use cases by storing and retrieving uploaded files.

The diagram also shows relationships between use cases, such as include and extend. Most business functions include the Login/Logout use case, meaning authentication is required before performing any operation. The Generate Invoice use case includes the Calculate Service Fees use case, because fees must be calculated before an invoice can be created. The Submit Feedback use case may extend Upload/Download File when the user attaches a file.

Overall, the Use Case Diagram provides a clear and concise overview of system scope, user roles, and main functionalities. It serves as a foundation for more detailed analysis, such as writing use case specifications and designing system architecture in later stages.

Troy University Resources

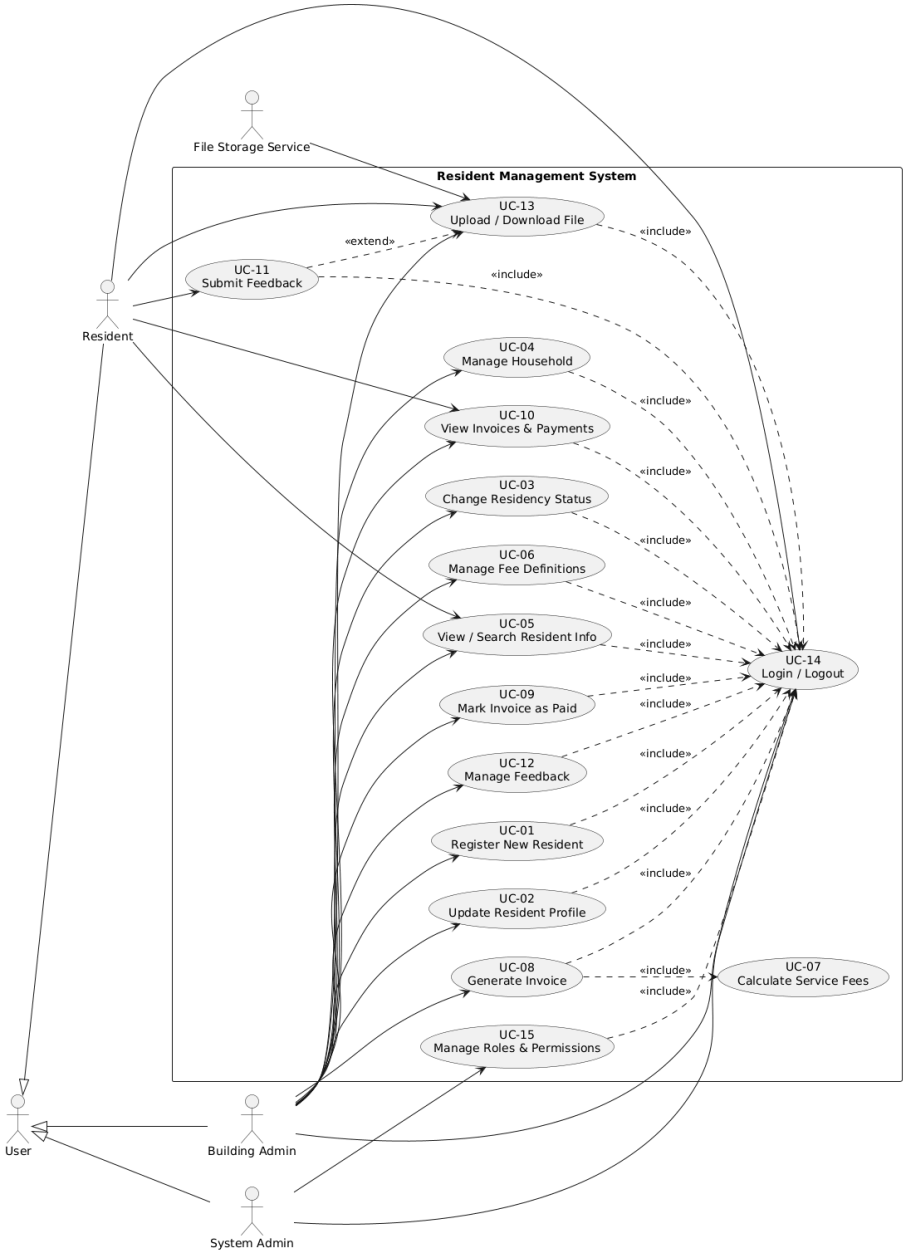


Figure 2-1: Overall Use Case Diagram

2.2.2. Level 2 Decomposed Use Case Diagram

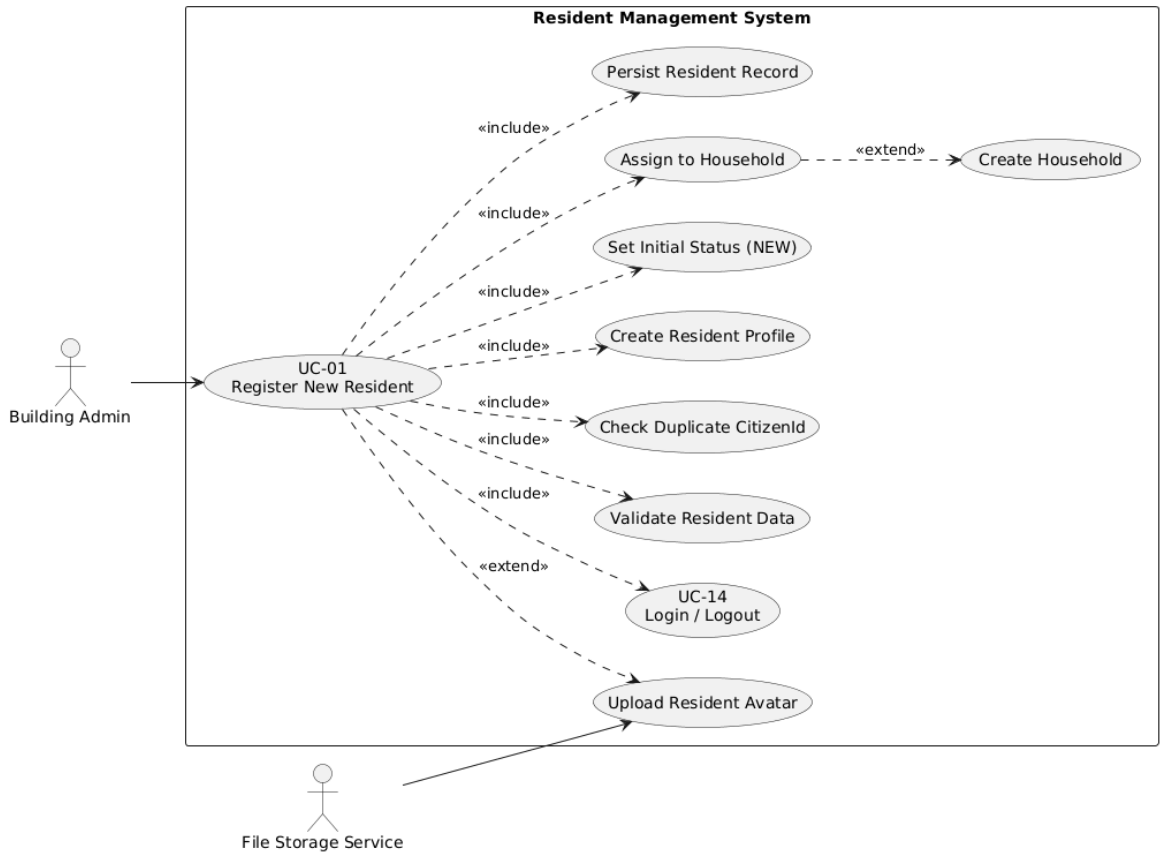


Figure 2-2: Decomposed Use Case: UC-01 Register New Resident

This decomposed use case shows the internal steps required to register a new resident. After authentication, the system validates resident information, checks for duplicate citizen identification, creates the resident profile, sets the initial status (NEW), and persists the record. The resident can then be assigned to an existing household, and in cases where a household does not exist yet, the process may extend to creating a new household. Avatar upload is modeled as an optional extension because it only happens when the admin provides an image, and it interacts with the external File Storage Service.

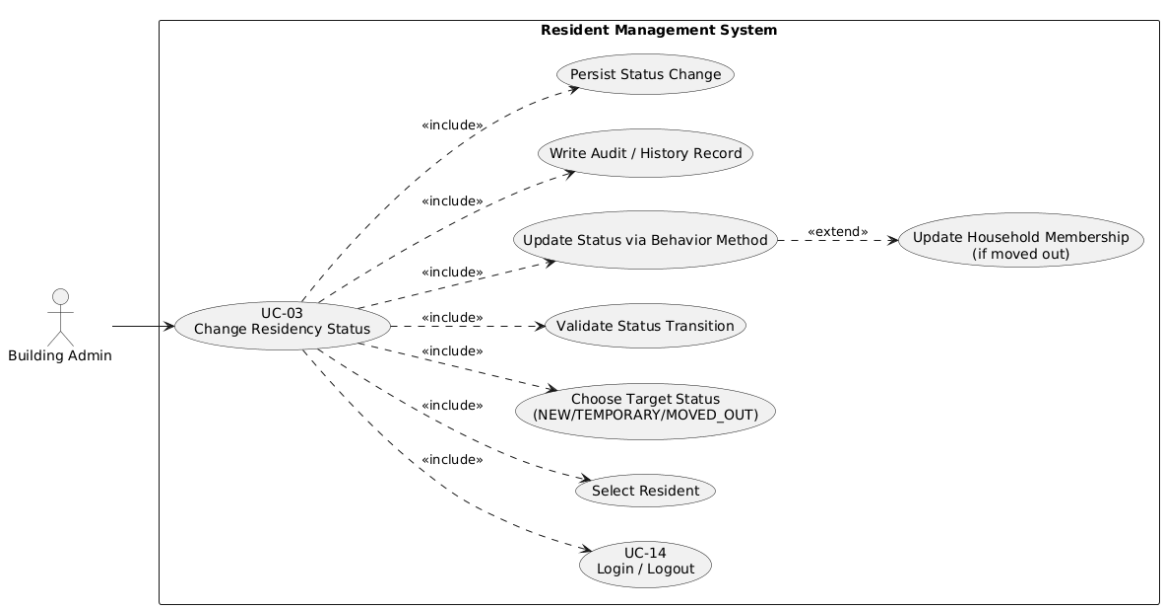


Figure 2-3: Decomposed Use Case: UC-03 Change Residency Status

Troy University Resources

This diagram breaks down the workflow for changing a resident's residency status. The Building Admin authenticates, selects the target resident, and chooses the desired status (NEW, TEMPORARY, or MOVED_OUT). The system then validates whether the transition is allowed based on business rules, updates the status through the resident's behavior methods, and persists the change. An audit/history record is always created to keep track of status changes, while updating household membership is modeled as an extension because it is only required in specific cases such as when a resident moves out.

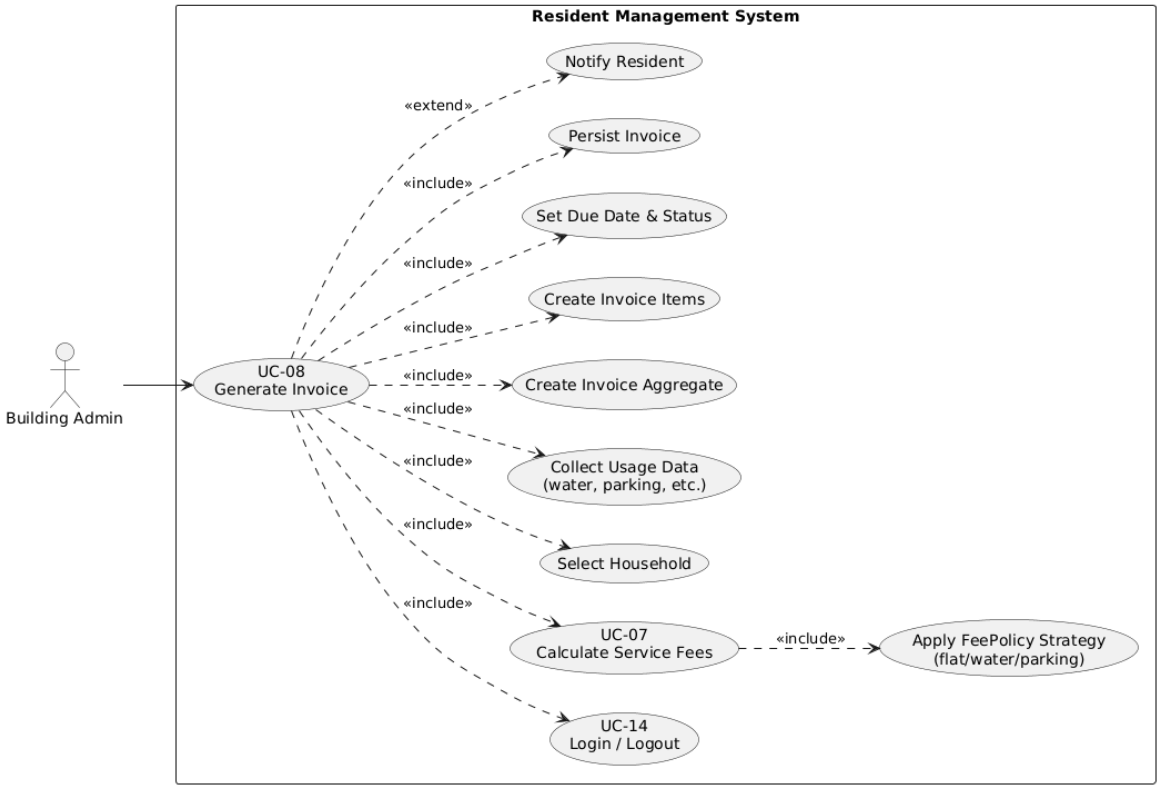


Figure 2-4: Decomposed Use Case: UC-08 Generate Invoice

This decomposed use case describes the invoice generation process for a household. After authentication, the Building Admin selects the household and provides required usage data (such as water usage or parking usage). The system calculates service fees by applying the FeePolicy strategy (flat fee, water usage fee, parking fee), then creates an invoice aggregate and its invoice items, sets the due date and initial status, and persists the invoice. Resident notification is modeled as an extension because it may depend on system configuration or whether notifications are enabled.

2.3. Use Case Specifications

- ***Use Case Specification UC-01 "Register New Resident"***

Table 2-3: Use Case Specification UC001 "Register New Resident"

Use Case Code	UC-01	**Use Case Name**	Register New Resident
Actor	Building Admin		
Description	Creates a new resident profile, assigns initial status, and optionally links the resident to a household.		
Preconditions	Building Admin is logged in and authorized. Citizen ID (or equivalent identifier) must be available.		

Main Flow (Success)

No.	Performed by	Action
1	Building Admin	Selects "Register New Resident".
2	System	Displays resident registration form.
3	Building Admin	Enters resident information (full name, DOB, citizenId, contact).
4	Building Admin	Selects household (existing) or chooses "Create new household" (optional).
5	Building Admin	Submits registration request.
6	System	Validates required fields and data format.
7	System	Checks duplicate citizenId.
8	System	Creates Resident entity and sets status = NEW.
9	System	Links resident to household (existing or newly created).
10	System	Persists resident record and returns confirmation with residentId.

Alternative Flows

No.	Performed by	Action
6a	System	Shows error: Missing or invalid fields.
7a	System	Shows error: citizenId already exists (duplicate resident).
9a	System	If "Create new household" selected, system creates household first, then links resident.
10a	System	Shows error: Database/service failure, registration not completed.

Postconditions	A new resident is stored in the system with status NEW. Resident is linked to a household if provided/created.

- ***Input data for personal information includes the following fields:***

Table 2-4: Input data for personal information

No.	Data Field	Description	Required	Valid Condition	Example
1	Full Name	Resident's full legal name	Yes	Only letters and spaces, 2–50 characters	Nguyen Van A
2	Date of Birth	Resident's birth date	Yes	Valid date, not in the future	2000-05-12
3	Citizen ID	National ID/Identification number	Yes	Unique, 9–12 digits	012345678901
4	Gender	Resident's gender	No	Male/Female/Other	Male
5	Phone Number	Contact phone number	Yes	9–11 digits, numeric	0989123456
6	Email	Contact email address	No	Valid email format	a.nguyen@gmail.com

Troy University Resources

No.	Data Field	Description	Required	Valid Condition	Example
7	Address	Permanent or current address	No	Max 255 characters	12 Tran Phu, Hanoi
8	Initial Status	Initial residency status	Yes	Must be NEW	NEW
9	Household ID	Existing household identifier	No	Must exist in system	HH-102
10	Avatar	Profile image file	No	JPG/PNG, max 5MB	avatar.png

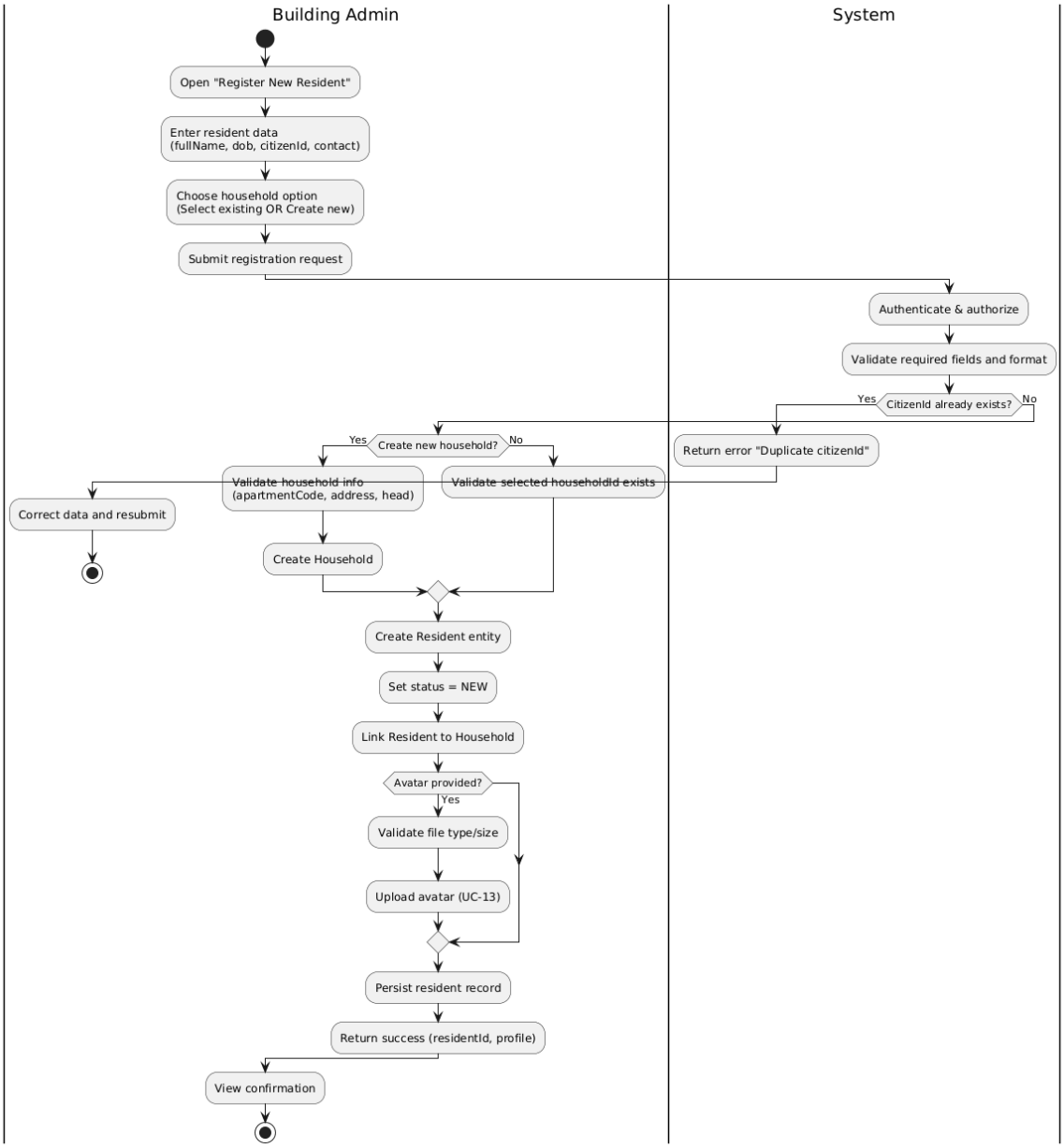


Figure 2-5: Activity Diagram for “Register New Resident”

- ***Use Case Specification UC-02 "Update Resident Profile"***

Table 2-5: Use Case Specification UC-02 "Update Resident Profile"

Use Case Code	UC-02	**Use Case Name**	Update Resident Profile
Actor	Building Admin (primary), Resident (optional, limited fields if allowed)		
Description	Updates a resident's profile information (for example name, contact details, date of birth), while enforcing validation rules and access control.		
Preconditions	Actor is logged in and authorized. Target resident exists in the system.		

Main Flow (Success)

No.	Performed by	Action
1	Actor	Selects "Update Resident Profile".
2	System	Displays resident search/selection screen.
3	Actor	Selects a resident by residentId or search.
4	System	Displays current resident profile information.
5	Actor	Edits allowed fields (for example phone, email, address info).
6	Actor	Submits update request.
7	System	Validates input fields (format, required fields, constraints).
8	System	Applies updates through domain/service methods (no direct field edits).
9	System	Persists updated resident information.
10	System	Returns updated profile confirmation.

Alternative Flows

No.	Performed by	Action
3a	System	Shows error: Resident not found.
5a	System	Shows error: Field not editable (permission or rule).
7a	System	Shows error: Invalid input format (email/phone/etc.).
9a	System	Shows error: Update failed due to system/database error.

Postconditions	Resident profile data is updated in the system. Change is stored and available for later viewing.

- ***Input data for updating a resident profile includes the following fields:***

Table 2-6: Input data for updating a resident profile

No.	Data Field	Description	Required?	Valid Condition	Example
1	Resident ID	Identifier of resident to update	Yes	Must exist in system	R-1001
2	Full Name	Updated full name	No	Letters and spaces, 2–50 chars	Nguyen Van B
3	Phone Number	Updated contact number	No	9–11 digits	0912345678
4	Email	Updated email	No	Valid email format	b.nguyen@gmail.com
5	Address	Updated address	No	Max 255 characters	45 Le Loi, HCM
6	Avatar	New avatar image	No	JPG/PNG, max 5MB	new_avatar.jpg

Troy University Resources

- ***Use Case Specification UC-03 "Change Residency Status"***

Table 2-7: Use Case Specification UC-03 "Change Residency Status"

Use Case Code	UC-03	**Use Case Name**	Change Residency Status
Actor	Building Admin		
Description	Updates a resident's residency status (NEW, TEMPORARY, MOVED_OUT) following business rules and stores history/audit.		
Preconditions	Building Admin is logged in and authorized. Target resident exists.		

Main Flow (Success)

No.	Performed by	Action
1	Building Admin	Selects "Change Residency Status".
2	System	Displays resident selection/search.
3	Building Admin	Selects a resident.
4	System	Displays current status and allowed transitions.
5	Building Admin	Chooses target status and provides effective date (and note if needed).
6	Building Admin	Submits the request.
7	System	Validates transition rules (invariants).
8	System	Applies status change via behavior method (example: markTemporaryResidence(), markMovedOut()).
9	System	Writes audit/history record.
10	System	Persists changes and returns confirmation.

Alternative Flows

No.	Performed by	Action
3a	System	Shows error: Resident not found.
7a	System	Shows error: Invalid transition (not allowed by rules).
10a	System	Shows error: Persisting failed, no update applied.

Postconditions	Resident status is updated if valid. Audit/history record is stored.

- ***Input data for changing residency status includes the following fields:***

Table 2-8: Input data for changing residency status

No.	Data Field	Description	Required?	Valid Condition	Example
1	Resident ID	Target resident identifier	Yes	Must exist	R-1001
2	Current Status	Resident's current status	Yes	Must match system value	NEW
3	New Status	Status to change to	Yes	NEW / TEMPORARY / MOVED_OUT	TEMPORARY
4	Effective Date	Date status becomes active	Yes	Valid date, ≥ today	2026-01-15
5	Note	Reason or description	No	Max 255 characters	Temporary stay
6	Admin ID	Admin performing action	Yes	Valid admin account	A-01

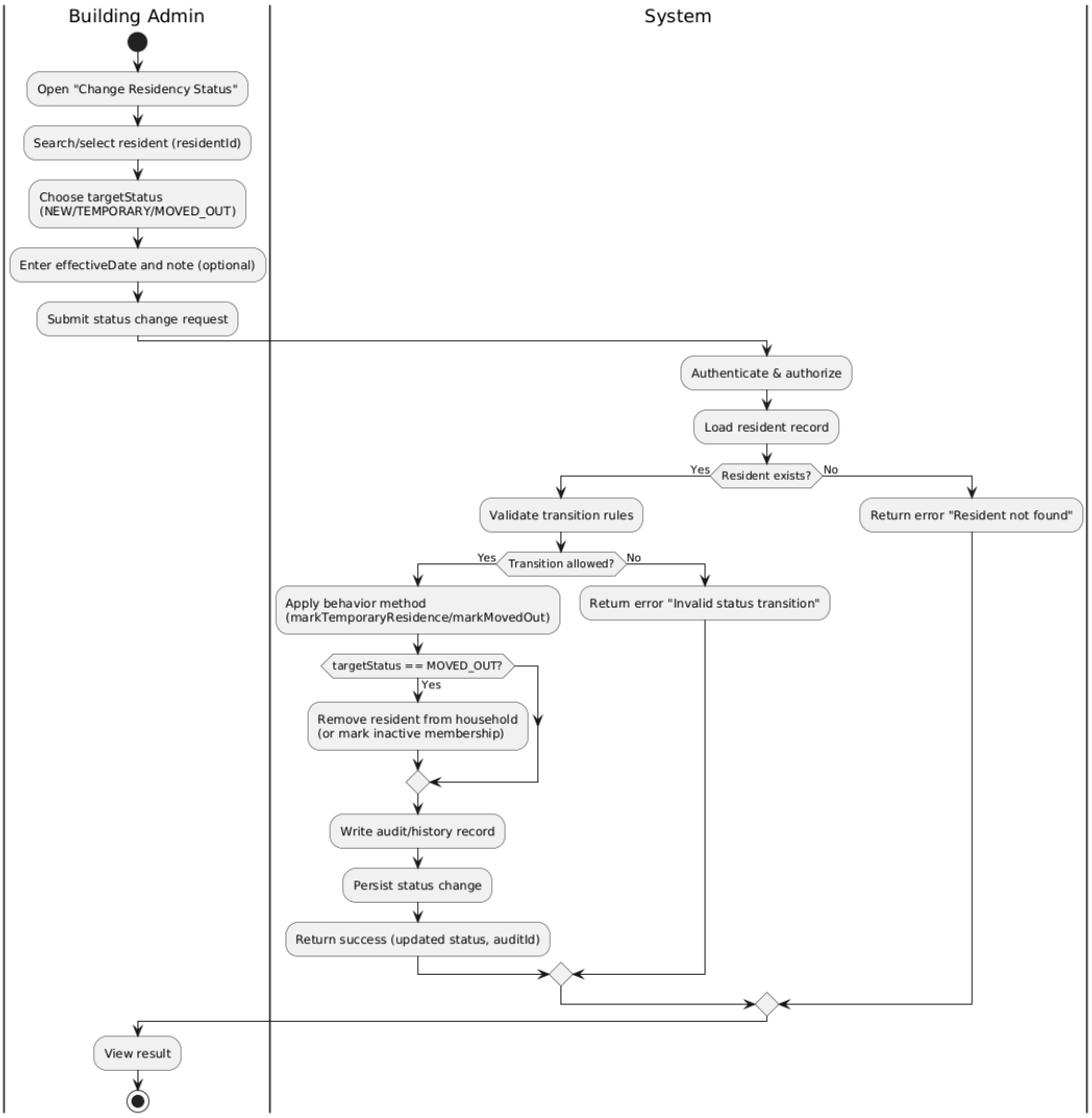


Figure 2-6: Activity Diagram for “Change Residency Status”

- ***Use Case Specification UC-04 “Manage Household”***

Table 2-9: Use Case Specification UC-04 “Manage Household”

Use Case Code	UC-04	**Use Case Name**	Manage House-hold
Actor	Building Admin		
Description	Creates a household and manages its members by adding or removing resi-dents.		
Preconditions	Building Admin is logged in and authorized.		

Troy University Resources

Use Case Code	UC-04	**Use Case Name**	Manage Household

Main Flow (Success)

No.	Performed by	Action
1	Building Admin	Selects “Household Management”.
2	System	Displays household list and actions (create, update, view).
3	Building Admin	Chooses “Create Household” or selects an existing household to update.
4	Building Admin	Enters/updates household info (address/apartment code).
5	Building Admin	Adds or removes household members (residentIds).
6	System	Validates household constraints (no duplicates, valid members).
7	System	Persists household changes and returns updated household details.

Alternative Flows

No.	Performed by	Action
5a	System	Shows error: Resident cannot be added (invalid status or already assigned).
6a	System	Shows error: Constraint violation (duplicates, invalid household).
7a	System	Shows error: Persisting failed.

Postconditions Household is created/updated with correct membership list.

- ***Input data for managing households includes the following fields:***

Table 2-10: Input data for managing households

No	Data Field	Description	Required?	Valid Condition	Example
1	Household ID	Identifier of household (for update)	No (Yes if update)	Must exist when updating	HH-102
2	Apartment Code	Apartment or house code	Yes	1–20 characters, unique	A12-05
3	Address	Household address	Yes	Max 255 characters	12 Nguyen Trai, Hanoi
4	Head Resident ID	Main resident of household	Yes	Must exist, status not MOVED_OUT	R-1001
5	Member List	List of resident IDs	No	Each ID must exist and be valid	R-1001, R-1002
6	Note	Additional description	No	Max 255 characters	Family of 4

- ***Use Case Specification UC-05 "View / Search Resident Information"***

Table 2-11: Use Case Specification UC-05 "View / Search Resident Information"

Use Case Code	UC-05	**Use Case Name**	View / Search Resident Information
Actor	Building Admin, Resident		
Description	Allows users to search, filter, and view resident and household information based on their access level.		
Preconditions	Actor is logged in. Actor has permission to view the requested information.		

Main Flow (Success)

No.	Performed by	Action
1	Actor	Opens "Residents" or "Households" view.
2	System	Displays search/filter options (name, citizenId, status, household, etc.).
3	Actor	Enters search keywords and/or filters.
4	Actor	Submits search request.
5	System	Validates request and access permissions.
6	System	Retrieves matching records from database.
7	System	Displays result list (paging/sorting if available).
8	Actor	Selects a record to view details.
9	System	Displays detailed resident/household information.

Alternative Flows

No.	Performed by	Action
5a	System	Shows error: Not authorized to view requested data.
6a	System	Shows empty results message if no records match.
9a	System	Shows error: Record not found (deleted or invalid id).

Postconditions	No data is modified. Actor receives the requested information (list and/or detail view).

- ***Input data for searching and viewing resident information includes the following fields:***

Table 2-12: Input data for searching and viewing resident information

No.	Data Field	Description	Required?	Valid Condition	Example
1	Keyword	Search by name or ID	No	Max 50 characters	Nguyen
2	Citizen ID	Filter by citizen ID	No	9–12 digits	012345678901
3	Status	Filter by residency status	No	NEW / TEMPORARY / MOVED_OUT	NEW
4	Household ID	Filter by household	No	Must exist	HH-102
5	Page Number	Result page index	No	≥ 1	1
6	Page Size	Records per page	No	1–100	10
7	Sort Field	Field to sort by	No	Valid column name	fullName

Troy University Resources

- ***Use Case Specification UC-06 “Manage Fee Definitions”***

Table 2-13: Use Case Specification UC-06 “Manage Fee Definitions”

Use Case Code	UC-06	**Use Case Name**	Manage Fee Definitions
Actor	Building Admin		
Description	Allows the admin to create, update, delete, and view service fee definitions that will be used later in fee calculation and invoice generation.		
Preconditions	Building Admin is logged in and authorized. Fee management permissions are granted.		

Main Flow (Success)

No.	Performed by	Action
1	Building Admin	Opens “Fee Management”.
2	System	Displays fee list and actions (create/update/delete).
3	Building Admin	Chooses an action (create/update/delete).
4	Building Admin	Enters or edits fee details (fee name, type, unit, base price, policy).
5	Building Admin	Submits the request.
6	System	Validates fee information (required fields, value range).
7	System	Persists changes to the fee definitions.
8	System	Returns confirmation and updated fee list.

Alternative Flows

No.	Performed by	Action
4a	System	Shows error: Missing or invalid fee data (negative price, invalid type).
7a	System	Shows error: Cannot delete fee in use by invoices (if applicable).
7b	System	Shows error: Create/update/delete failed due to system/database error.

Postconditions	Fee definitions are created/updated/deleted as requested. Updated fee configurations are available for fee calculation (UC-07) and invoice generation (UC-08).

- ***Input data for managing service fees includes the following fields:***

Table 2-14: Input data for managing service fees

No.	Data Field	Description	Required?	Valid Condition	Example
1	Fee ID	Identifier of fee (for update/delete)	No (Yes if update/delete)	Must exist	F-01
2	Fee Name	Name of service fee	Yes	2–50 characters	Water Fee
3	Fee Type	Type of fee	Yes	FLAT / WATER / PARKING	WATER
4	Unit Price	Price per unit	Yes	≥ 0	5000
5	Unit Name	Unit description	No	Max 20 characters	m3
6	Policy Type	Fee policy strategy	Yes	FlatFee / WaterUsage / Parking	WaterUsage
7	Description	Fee description	No	Max 255 characters	Monthly water usage

- ***Use Case Specification UC-07 "Calculate Service Fees"***

Table 2-15: Use Case Specification UC-07 "Calculate Service Fees"

Use Case Code	UC-07	**Use Case Name**	Calculate Service Fees
Actor	System (triggered by Building Admin via invoice generation), Building Admin (initiator)		
Description	Calculates service fees for a household or billing period by applying different fee policies (Strategy Pattern), such as flat fee, water usage fee, and parking fee.		
Preconditions	Building Admin is logged in and authorized (if triggered from UI). Fee definitions and fee policies are configured. Household and billing context (period, usage data) are available.		
Main Flow (Success)	**No.** / **Performed by** / **Action** 1 / Building Admin / Starts invoice generation or fee calculation for a household. 2 / System / Loads fee definitions and applicable FeePolicy strategies. 3 / System / Collects required inputs (billing period, water usage, parking quantity, etc.). 4 / System / Validates input values (non-negative, required fields). 5 / System / Applies fee calculation per policy (FlatFee/WaterUsage/Parking). 6 / System / Produces calculated fee items (name, unit price, quantity, subtotal). 7 / System / Returns calculated fee list for invoice creation.		
Alternative Flows	**No.** / **Performed by** / **Action** 2a / System / Shows error: Missing fee configuration or policy not found. 4a / System / Shows error: Invalid usage data (missing or negative value). 5a / System / Shows error: Calculation failed due to system error.		
Postconditions	A set of calculated fee items is produced (not necessarily persisted by this use case alone). Results can be consumed by UC-08 Generate Invoice.		

- ***Input data for calculating service fees includes the following fields:***

Table 2-16: Input data for calculating service fees

No.	Data Field	Description	Required?	Valid Condition	Example
1	Household ID	Target household	Yes	Must exist	HH-102
2	Billing Period	Month/year of billing	Yes	MM-YYYY format	01-2026
3	Water Usage	Water consumption	No (Yes if WATER fee exists)	≥ 0	15
4	Parking Count	Number of vehicles	No (Yes if PARKING fee exists)	≥ 0, integer	2
5	Flat Fee Enabled	Apply flat fee or not	No	true / false	true
6	Calculation Date	Date of calculation	Yes	Valid date	2026-01-31
7	Admin ID	Admin triggering calculation	Yes	Valid admin account	A-01

Troy University Resources

- ***Use Case Specification UC-07 “Calculate Service Fees”***

Table 2-17: Use Case Specification UC-07 “Calculate Service Fees”

Use Case Code	UC-08	**Use Case Name**	Generate Invoice
Actor	Building Admin (primary), System (fee calculation)		
Description	Generates an invoice for a household by calculating fees using FeePolicy strategies and creating invoice items.		
Preconditions	Building Admin is logged in and authorized. Household exists. Fee definitions/policies are configured (flat/water/parking).		

Main Flow (Success)

No.	Performed by	Action
1	Building Admin	Selects “Generate Invoice”.
2	System	Displays household selection and billing period form.
3	Building Admin	Selects household and billing period.
4	Building Admin	Enters usage data if needed (water usage, parking count).
5	Building Admin	Submits invoice generation request.
6	System	Validates inputs and retrieves applicable fee policies.
7	System	Calculates each fee item using FeePolicy (Strategy Pattern).
8	System	Creates Invoice aggregate root and InvoiceItem list.
9	System	Sets due date and initial status (UNPAID).
10	System	Persists invoice and returns invoiceId and summary.

Alternative Flows

No.	Performed by	Action
3a	System	Shows error: Household not found.
6a	System	Shows error: Missing fee configuration or invalid usage data.
10a	System	Shows error: Invoice creation failed (transaction rollback).

Postconditions	A new invoice is created and stored with invoice items and status UNPAID.

- ***Input data for generating an invoice includes the following fields:***

Table 2-18: Input data for generating an invoice

No.	Data Field	Description	Required?	Valid Condition	Example
1	Household ID	Household to be billed	Yes	Must exist	HH-102
2	Billing Period	Month and year of invoice	Yes	MM-YYYY format	01-2026
3	Usage Data	Usage values for services	Yes (if required by fee type)	≥ 0	Water: 15 m3
4	Fee Policy Set	Set of applied fee policies	Yes	Must be configured	WaterUsage, Parking
5	Due Date	Invoice due date	Yes	≥ invoice date	2026-02-10
6	Created By	Admin creating invoice	Yes	Valid admin ID	A-01
7	Note	Additional invoice note	No	Max 255 characters	January services
8	Notification Flag	Send notification or not	No	true / false	true

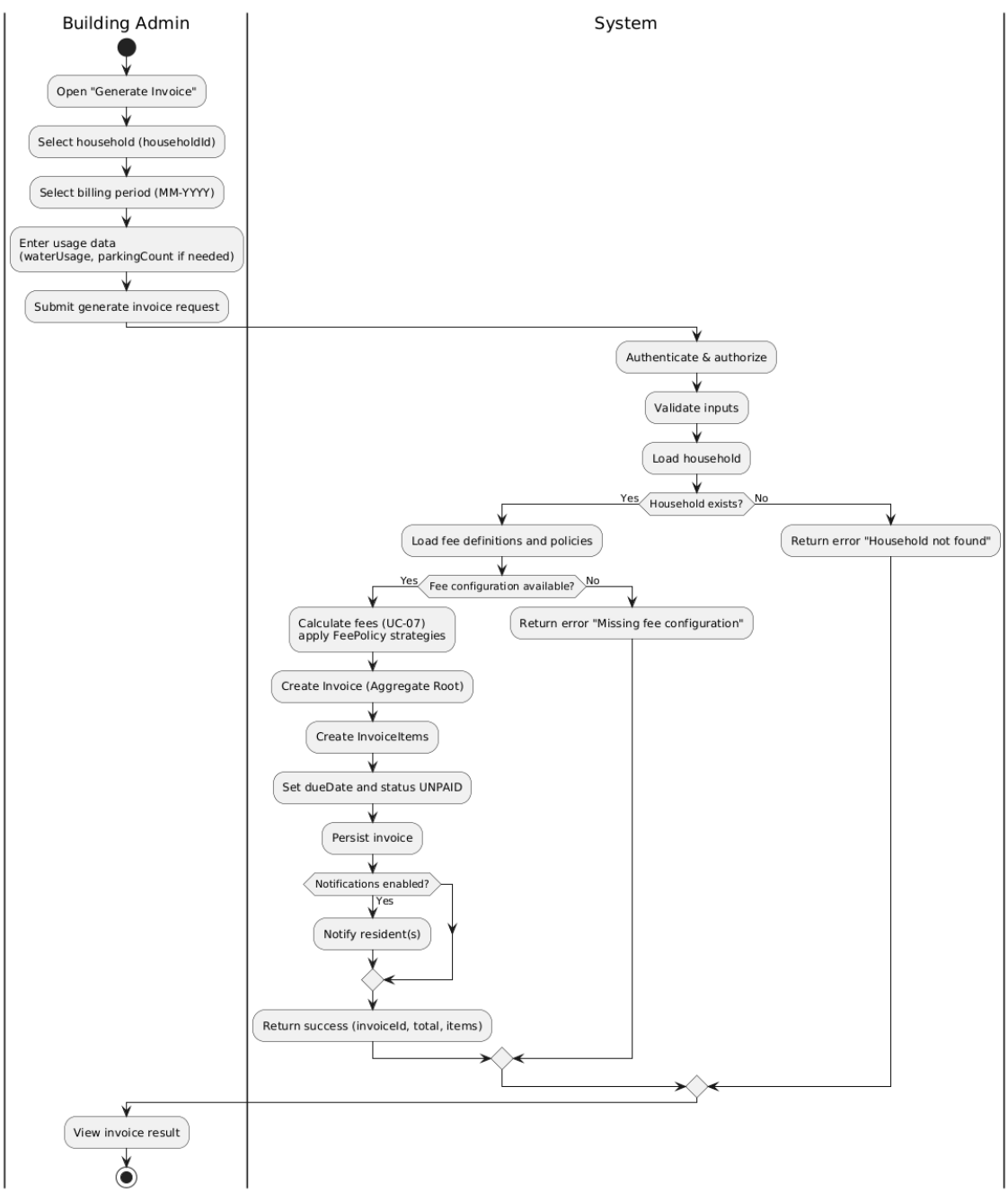


Figure 2-7: Activity Diagram for “Generate Invoice”

Troy University Resources

- ***Use Case Specification UC-07 “Calculate Service Fees”***

Table 2-19: Use Case Specification UC-09 “Mark Invoice as Paid”

Use Case Code	UC-09	**Use Case Name**	Mark Invoice as Paid
Actor	Building Admin		
Description	Confirms that an invoice has been paid and updates invoice status, payment timestamp, and optional payment note.		
Preconditions	Building Admin is logged in and authorized. Invoice exists and is currently UNPAID (or not PAID).		

Main Flow (Success)

No.	Performed by	Action
1	Building Admin	Opens invoice list and selects an invoice.
2	System	Displays invoice details and current payment status.
3	Building Admin	Chooses “Mark as Paid” and enters payment info (date, note, method if tracked).
4	Building Admin	Submits payment confirmation.
5	System	Validates invoice status (must be payable) and required fields.
6	System	Updates invoice status to PAID and stores payment timestamp.
7	System	Persists changes and returns confirmation/receipt.

Alternative Flows

No.	Performed by	Action
1a	System	Shows error: Invoice not found.
5a	System	Shows error: Invoice already PAID or not eligible for payment update.
7a	System	Shows error: Update failed due to system/database error.

Postconditions	Invoice status becomes PAID. Payment information is stored and visible in invoice history.

- ***Input data for confirming invoice payment includes the following fields:***

Table 2-20: Input data for confirming invoice payment

No.	Data Field	Description	Required?	Valid Condition	Example
1	Invoice ID	Identifier of invoice	Yes	Must exist	INV-2026-01-01
2	Payment Date	Date of payment	Yes	Valid date	2026-02-05
3	Payment Method	Method of payment	No	Cash/Transfer/Other	Transfer
4	Payment Note	Additional note	No	Max 255 characters	Paid via bank
5	Admin ID	Admin confirming payment	Yes	Valid admin account	A-01

- ***Use Case Specification UC-10 "View Invoices and Payments"***

Table 2-21: Use Case Specification UC-10 "View Invoices and Payments"

Use Case Code	UC-10	**Use Case Name**	View Invoices and Payments
Actor	Resident, Building Admin		
Description	Allows users to view invoices, fee breakdown (invoice items), payment status, and payment history based on access level.		
Preconditions	Actor is logged in. Actor has permission to view requested invoices (Resident: own household; Admin: all).		

Main Flow (Success)

No.	Performed by	Action
1	Actor	Opens "Invoices" page/menu.
2	System	Displays invoice list with filters (period, status, household).
3	Actor	Applies filters/search and submits request.
4	System	Validates access permission and query parameters.
5	System	Retrieves invoice list from database (with paging/sorting if available).
6	System	Displays invoice list summary (total, status, due date).
7	Actor	Selects an invoice to view details.
8	System	Displays invoice details, invoice items, and payment status/history.

Alternative Flows

No.	Performed by	Action
4a	System	Shows error: Not authorized to view these invoices.
5a	System	Shows empty result if no invoices match filters.
8a	System	Shows error: Invoice details not found (invalid id).

Postconditions	No system data is modified. Actor receives invoice and payment information.

- ***Input data for viewing invoices and payments includes the following fields:***

Table 2-22: Input data for viewing invoices and payments

No.	Data Field	Description	Required?	Valid Condition	Example
1	Actor ID	User requesting data	Yes	Must exist	R-1001
2	Role	Role of actor	Yes	RESIDENT / ADMIN	RESIDENT
3	Household ID	Filter by household	No	Must exist	HH-102
4	Billing Period	Filter by period	No	MM-YYYY	01-2026
5	Status	Filter by invoice status	No	PAID / UNPAID	UNPAID
6	Page Number	Page index	No	≥ 1	1
7	Page Size	Records per page	No	1–100	10
8	Sort Field	Sort by column	No	Valid field name	dueDate

Troy University Resources

- ***Use Case Specification UC-11 “Submit Feedback”***

Table 2-23: Use Case Specification UC-11 “Submit Feedback”

Use Case Code	UC-11	**Use Case Name**	Submit Feedback
Actor	Resident		
Description	Allows a resident to submit feedback or a support request, optionally with an attachment.		
Preconditions	Resident is logged in.		

Main Flow (Success)

No.	Performed by	Action
1	Resident	Selects “Submit Feedback”.
2	System	Displays feedback form.
3	Resident	Enters subject/content and optional attachment.
4	Resident	Submits feedback.
5	System	Validates required fields.
6	System	Uploads attachment if provided.
7	System	Stores feedback record and returns confirmation.

Alternative Flows

No.	Performed by	Action
5a	System	Shows error: Missing required fields.
6a	System	Shows error: Attachment upload failed or invalid file type.

Postconditions	Feedback record is stored. Attachment reference is saved if uploaded.

- ***Input data for submitting feedback includes the following fields:***

Table 2-24: Input data for submitting feedback

No.	Data Field	Description	Required?	Valid Condition	Example
1	Resident ID	Resident submitting feedback	Yes	Must exist	R-1001
2	Subject	Feedback subject	Yes	5–100 characters	Water issue
3	Content	Feedback content	Yes	10–1000 characters	Water pressure is low
4	Attachment	Optional file	No	JPG/PNG/PDF, ≤ 5MB	photo.jpg
5	Submit Date	Date of submission	Yes	Valid date	2026-01-20
6	Status	Initial feedback status	Yes	NEW	NEW

- ***Use Case Specification UC-12 "Manage Feedback"***

Table 2-25: Use Case Specification UC-12 "Manage Feedback"

Use Case Code	UC-12	**Use Case Name**	Manage Feedback
Actor	Building Admin		
Description	Allows the admin to view, review, and delete feedback submitted by residents.		
Preconditions	Building Admin is logged in and authorized. At least one feedback record exists.		

Main Flow (Success)

No.	Performed by	Action
1	Building Admin	Opens "Feedback Management".
2	System	Displays feedback list with filters (date, status, resident).
3	Building Admin	Selects a feedback item.
4	System	Displays feedback details and attachments (if any).
5	Building Admin	Chooses an action (mark reviewed, delete).
6	Building Admin	Confirms the action.
7	System	Updates feedback status or deletes record.
8	System	Persists changes and refreshes feedback list.

Alternative Flows

No.	Performed by	Action
3a	System	Shows error: Feedback not found.
5a	System	Shows error: Action not permitted.
7a	System	Shows error: Update/delete failed.

Postconditions	Feedback is updated (reviewed) or removed from the system.

- ***Input data for managing feedback includes the following fields:***

Table 2-26: Input data for managing feedback

No.	Data Field	Description	Required?	Valid Condition	Example
1	Feedback ID	Identifier of feedback	Yes	Must exist	FB-102
2	Action Type	Action to perform	Yes	VIEW / MARK_REVIEWED / DELETE	MARK_REVIEWED
3	Admin ID	Admin performing action	Yes	Valid admin account	A-01
4	Note	Admin note (optional)	No	Max 255 characters	Reviewed
5	Action Date	Date of action	Yes	Valid date	2026-01-22
1	Feedback ID	Identifier of feedback	Yes	Must exist	FB-102
2	Action Type	Action to perform	Yes	VIEW / MARK_REVIEWED / DELETE	MARK_REVIEWED
3	Admin ID	Admin performing action	Yes	Valid admin account	A-01
4	Note	Admin note (optional)	No	Max 255 characters	Reviewed
5	Action Date	Date of action	Yes	Valid date	2026-01-22

Troy University Resources

- ***Use Case Specification UC-13* “Upload / Download File”**

Table 2-27: Use Case Specification UC-13 “Upload / Download File”

Use Case Code	UC-13	**Use Case Name**	Upload / Download File
Actor	Resident, Building Admin, File Storage Service		
Description	Handles uploading and retrieving files such as avatars and attachments.		
Preconditions	Actor is logged in. File type and size are within allowed limits.		

Main Flow (Success)

No.	Performed by	Action
1	Actor	Chooses “Upload File” or “Download File”.
2	System	Displays file selection or file list.
3	Actor	Selects file (for upload) or selects file to download.
4	Actor	Submits request.
5	System	Validates file type, size, and permissions.
6	System	Sends file to File Storage Service (upload) or requests file (download).
7	File Storage Service	Stores or retrieves file and returns status.
8	System	Saves or reads file reference and returns result to actor.

Alternative Flows

No.	Performed by	Action
5a	System	Shows error: Invalid file type or size too large.
6a	System	Shows error: Storage service unavailable.
7a	System	Shows error: File not found in storage.

Postconditions	On upload: File is stored and linked to a system record. On download: File is delivered to the actor.

- ***Input data for file operations includes the following fields:***

Table 2-28: Input data for file operations

No.	Data Field	Description	Required?	Valid Condition	Example
1	File	File to upload	Yes (for upload)	Allowed type, ≤ 5MB	avatar.png
2	File ID	Identifier of file (for download)	Yes (for download)	Must exist	FILE-778
3	File Type	Category of file	Yes	AVATAR / ATTACHMENT	AVATAR
4	Owner ID	Owner of file	Yes	Must exist	R-1001
5	Request Type	Upload or Download	Yes	UPLOAD / DOWNLOAD	UPLOAD
6	Upload Date	Date of upload	Yes (for upload)	Valid date	2026-01-21

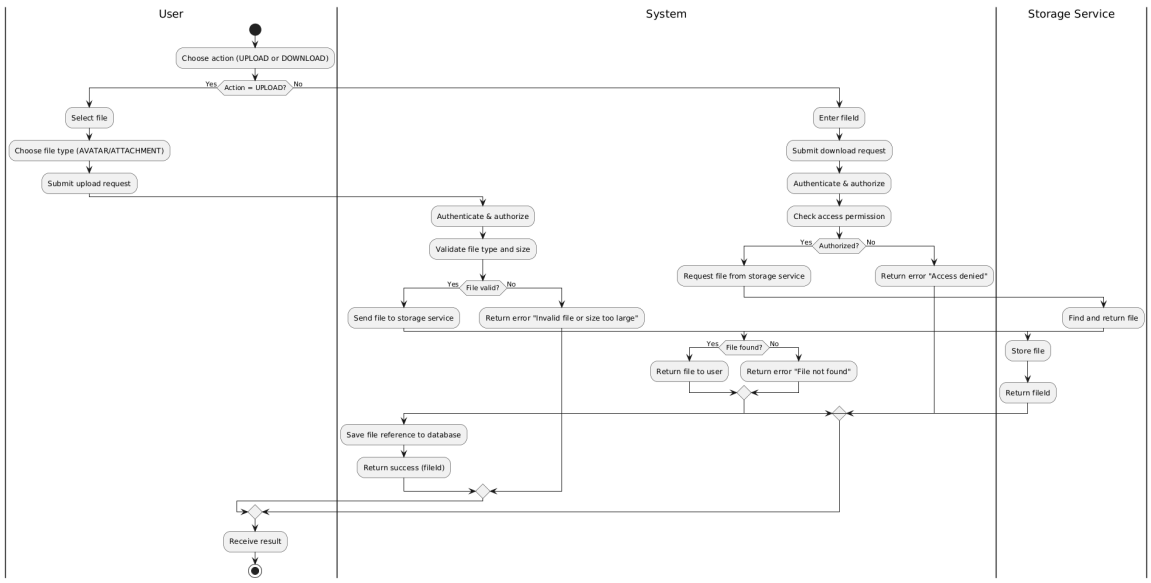


Figure 2-8: Activity Diagram for “Upload/Download File”

- ***Use Case Specification UC-14 “Login / Logout”***

Table 2-29: Use Case Specification UC-14 “Login / Logout”

Use Case Code	UC-14	**Use Case Name**	Login / Logout
Actor	Resident, Building Admin, System Admin		
Description	Allows a user with an existing account to access the system and end the session when finished.		
Preconditions	User has a valid account (username/password). System is available.		

Main Flow (Success)

No.	Performed by	Action
1	User	Selects “Login”.
2	System	Displays login form.
3	User	Enters username and password.
4	User	Submits login request.
5	System	Validates required fields.
6	System	Verifies credentials.
7	System	Generates access token (JWT) and returns user role(s).
8	System	Redirects user to the main page/dashboard.
9	User	Selects “Logout” when finished.
10	System	Invalidates session token (or marks it as logged out) and returns logout confirmation.

Alternative Flows

No.	Performed by	Action
5a	System	Shows error: Missing required fields.
6a	System	Shows error: Invalid username and/or password.
7a	System	Shows error: Account disabled or not permitted.

Postconditions	On success: User is authenticated and can access authorized features. On logout: User session ends and protected resources are no longer accessible.

- ***Input data for authentication includes the following fields:***

Troy University Resources

Table 2-30: Input data for authentication

No.	Data Field	Description	Required?	Valid Condition	Example
1	Username	Account username	Yes	4–30 characters	admin01
2	Password	Account password	Yes	≥ 8 chars, mixed types	Abc@1234
3	Action	Login or Logout	Yes	LOGIN / LOGOUT	LOGIN
4	Device Info	Client device info	No	Max 100 characters	Chrome-Win10
5	IP Address	Client IP	No	Valid IPv4/IPv6	192.168.1.10

- ***Use Case Specification UC-15 "Manage Roles and Permissions"***

Table 2-31: Use Case Specification UC-15 "Manage Roles and Permissions"

Use Case Code	UC-15	**Use Case Name**	Manage Roles and Permissions
Actor	System Admin		
Description	Allows the system administrator to define roles, permissions, and assign them to users.		
Preconditions	System Admin is logged in and authorized. Security module is active.		

Main Flow (Success)

No.	Performed by	Action
1	System Admin	Opens "Role & Permission Management".
2	System	Displays role list and permission list.
3	System Admin	Chooses to create, update, or delete a role.
4	System Admin	Enters or edits role name and assigned permissions.
5	System Admin	Submits changes.
6	System	Validates role data and permission references.
7	System	Persists role/permission changes.
8	System	Returns confirmation and updated lists.

Alternative Flows

No.	Performed by	Action
4a	System	Shows error: Missing or invalid role data.
7a	System	Shows error: Cannot delete role in use.
7b	System	Shows error: Save failed due to system/database error.

Postconditions	Roles and permissions are updated. New access rules are effective for subsequent logins/sessions.

- ***Input data for managing roles and permissions includes the following fields:***

Table 2-32: Input data for managing roles and permissions

No.	Data Field	Description	Required?	Valid Condition	Example
1	Role ID	Role identifier (for update/delete)	No (Yes if update/delete)	Must exist	ROLE-ADMIN
2	Role Name	Name of role	Yes (for create)	3–30 characters, unique	BUILDING_ ADMIN

No.	Data Field	Description	Required?	Valid Condition	Example
3	Permission List	Permissions assigned	Yes	Must exist in system	CREATE_ RESIDENT, VIEW_ INVOICE
4	Action Type	Create/Update/Delete	Yes	CREATE / UPDATE / DELETE	CREATE
5	Admin ID	System admin performing action	Yes	Valid system admin	SA-01
6	Description	Role description	No	Max 255 characters	Admin role

2.4. Non-Functional Requirements

2.4.1. Functionality (General Functional Requirements)

- The system must support multiple users working concurrently (Residents and Admins).
- All use cases that require access to the database must implement database error handling. If a connection/query/transaction fails, the system must show a clear message indicating the issue is database-related (not a user input error).
- Any use case performed by Building Admin or System Admin must require login with the correct role. Unauthorized access must be rejected.
- Common validation rules must be applied consistently across the system:
  - Unique citizenId for residents.
  - No duplicate resident membership within a household.
  - Status transitions must follow defined business rules (NEW → TEMPORARY → MOVED_ OUT, etc.).
- Standard response format for APIs:
  - Return success/failure flag, message, and relevant IDs (residentId, householdId, invoiceId).
  - Return detailed error information (error code and message) when failure occurs.

2.4.2. Usability

- The user interface should be designed to be easy to learn and operate, with clear navigation for both Residents and Admins.
- Error messages must be specific and actionable, indicating:
  - what is wrong,
  - which field caused the error,
  - and how to correct it (example: “Citizen ID must be 12 digits”).
- Common actions must require minimal steps (for example: searching residents, viewing invoices).
- The system should provide consistent UI patterns across pages (forms, tables, buttons, and alerts).

2.4.3. Performance (Efficiency)

- The system should handle at least 200 concurrent users without noticeable degradation under typical course/demo conditions.
- For common operations (search, list, view details), the system should return results within:
  - ≤ 2 seconds for most requests,
  - ≤ 5 seconds for invoice generation (fee calculation + persistence).
- File upload/download operations should complete within ≤ 10 seconds for files up to 5MB, depending on network conditions.

Troy University Resources

2.4.4. Reliability (Availability and Fault Tolerance)

- The system should be available for use during business hours and demo sessions, with minimal downtime.
- All critical operations (register resident, change status, generate invoice, mark invoice paid) must be executed using transaction management to avoid partial updates.
- The system must log server-side errors to support debugging and recovery.

2.4.5. Security (Safety and Access Control)

- Authentication must be implemented using secure token-based login (JWT).
- Authorization must be role-based:
  - Residents can access only their personal/household data.
  - Building Admin can manage resident/household/fees/invoices.
  - System Admin can manage roles and permissions.
- Sensitive data (passwords) must be stored using hashing (never plain text).
- The system must validate all inputs to prevent common web issues (invalid formats, injection risks).

2.4.6. Maintainability

- The codebase should follow layered architecture (Controller–Service–Repository) and clear separation of concerns.
- Business rules should be centralized in domain/service logic rather than scattered in controllers.
- Fee calculation must be extensible using the Strategy Pattern so new fee types can be added without changing existing logic heavily.
- The system should support basic unit/integration testing for critical business flows.

2.4.7. Technical Constraints (Database, Technology, Environment)

- Database must be a relational DBMS: MySQL.
- Backend: Spring Boot, Spring Data JPA (Hibernate), Spring Security, REST API.
- Frontend: Next.js (TypeScript), Tailwind CSS.
- Deployment/DevOps: Docker, Nginx, GitHub Actions, and cloud environment (for example AWS ECS) if used in the project.
- The system must provide a web-based interface and support modern browsers (Chrome/Edge).

CHAPTER 3. REQUIREMENTS ANALYSIS

3.1. Identify Analysis Classes

1) Result of the initial decomposition of use case “UC-01: Register New Resident”

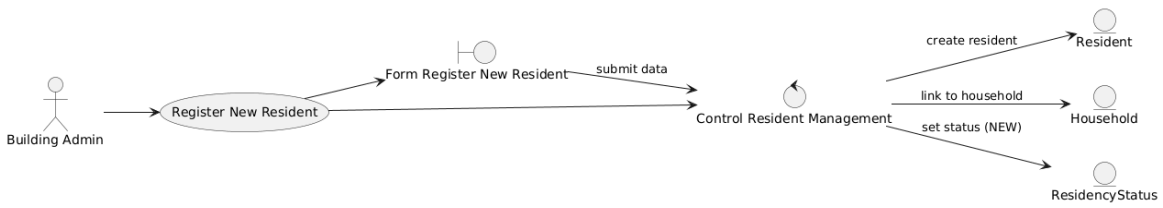


Figure 3-1: Initial decomposition of use case “UC-01: Register New Resident”

- Short description (UC-01):
  - Boundary: Form Register New Resident – data input interface.
  - Control: Control Resident Management – coordinates the process.
  - Entity: Resident, Household, ResidencyStatus.
  - Flow: Admin enters data → Control creates Resident → assigns initial status → links to Household.

2) Result of the initial decomposition of use case “UC-02: Update Resident Profile”

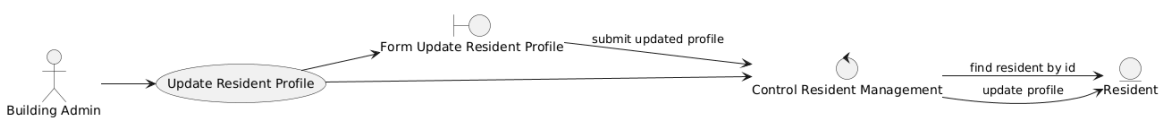


Figure 3-2: Initial decomposition of use case “UC-02: Update Resident Profile”

- Short description (UC-02):
  - Boundary: Form Update Resident Profile – profile editing interface.
  - Control: Control Resident Management – manages the update flow.
  - Entity: Resident.
  - Flow: Admin submits new data → Control loads Resident → updates information.

3) Result of the initial decomposition of use case “UC-03: Change Residency Status”

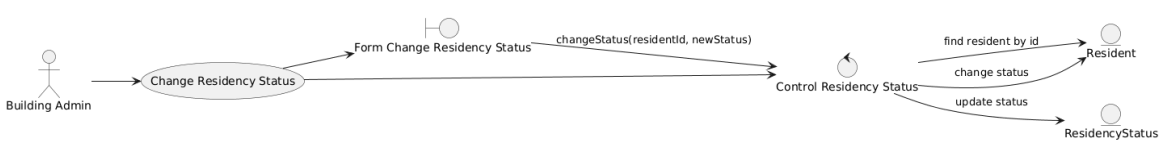
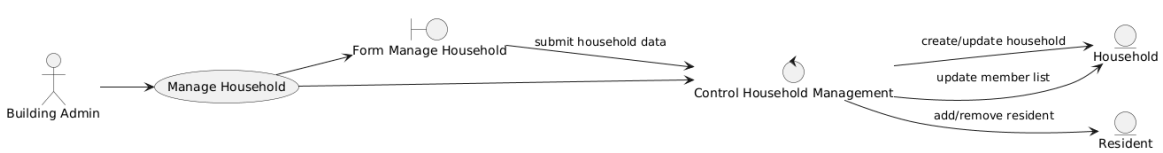


Figure 3-3: Initial decomposition of use case “UC-03: Change Residency Status”

- Short description (UC-03):
  - Boundary: Form Change Residency Status – status selection interface.
  - Control: Control Residency Status – coordinates status changes.
  - Entity: Resident, ResidencyStatus.
  - Flow: Admin selects new status → Control loads Resident → updates status.

4) Result of the initial decomposition of use case “UC-04: Manage Household”



Troy University Resources

Figure 3-4: Initial decomposition of use case “UC-04: Manage Household”

- Short description (UC-04):
  - Boundary: Form Manage Household for creating or editing households.
  - Control: Control Household Management coordinates operations.
  - Entity: Household and Resident.
  - Flow: Admin enters data → Control creates or updates Household → adds or removes Residents.

5) Result of the initial decomposition of use case “UC-05: View Resident Information”

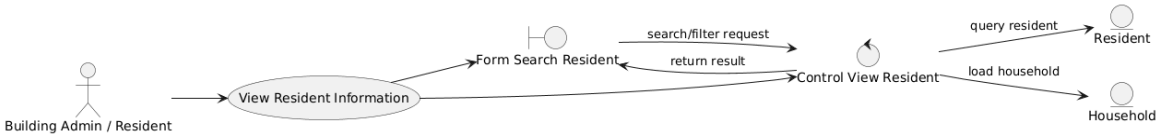


Figure 3-5: Initial decomposition of use case “UC-05: View Resident Information”

- Short description (UC-05):
  - Boundary: Form Search Resident for searching and filtering.
  - Control: Control View Resident handles queries.
  - Entity: Resident and Household.
  - Flow: User searches → Control queries data → returns information.

6) Result of the initial decomposition of use case “UC-06: Manage Fee Definitions”

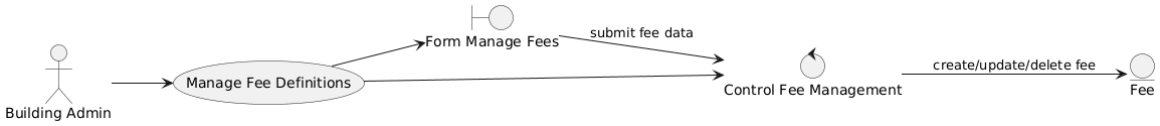


Figure 3-6: Initial decomposition of use case “UC-06: Manage Fee Definitions”

- Short description (UC-06):
  - Boundary: Form Manage Fees for defining service fees.
  - Control: Control Fee Management coordinates fee operations.
  - Entity: Fee.
  - Flow: Admin submits fee data → Control creates, updates, or deletes Fee.

7) Result of the initial decomposition of use case “UC-07: Calculate Service Fees”

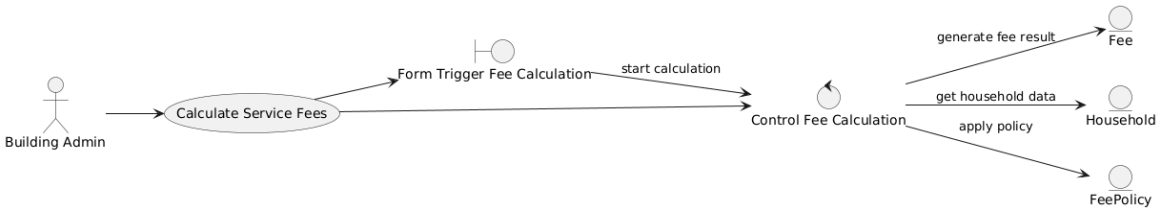


Figure 3-7: Initial decomposition of use case “UC-07: Calculate Service Fees”

- Short description (UC-07):
  - Boundary: Form Trigger Fee Calculation.
  - Control: Control Fee Calculation coordinates fee computation.
  - Entity: Fee, Household, FeePolicy.

– Flow: Admin triggers calculation → Control loads Household → applies FeePolicy → generates Fee results.

8) Result of the initial decomposition of use case “UC-08: Generate Invoice”

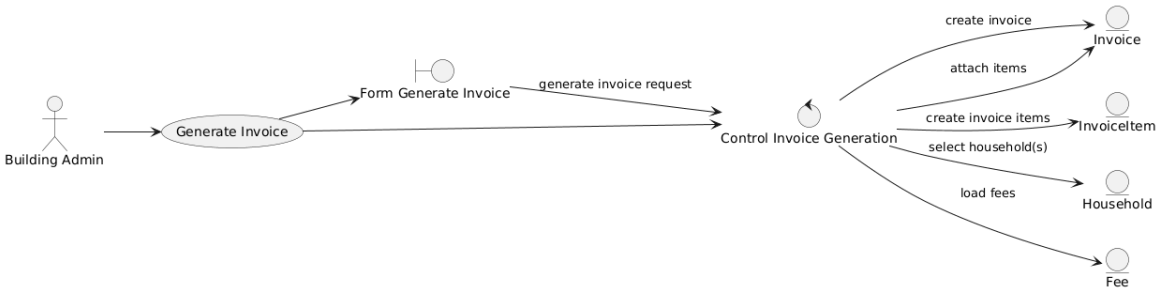


Figure 3-8: Initial decomposition of use case “UC-08: Generate Invoice”

- Short description (UC-08):
  - Boundary: Form Generate Invoice to start invoice creation.
  - Control: Control Invoice Generation manages the workflow.
  - Entity: Household, Fee, Invoice, InvoiceItem.
  - Flow: Admin requests invoice → Control loads households and fees → creates Invoice and InvoiceItems.

9) Result of the initial decomposition of use case “UC-09: Mark Invoice as Paid”

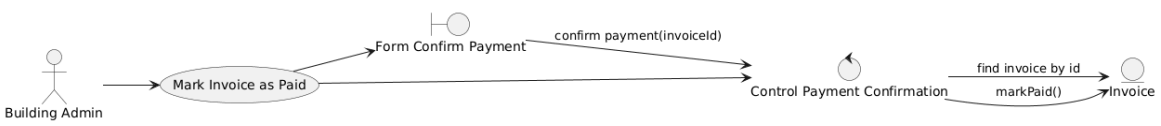


Figure 3-9: Initial decomposition of use case “UC-09: Mark Invoice as Paid”

- Short description (UC-09):
  - Boundary: Form Confirm Payment.
  - Control: Control Payment Confirmation coordinates payment confirmation.
  - Entity: Invoice.
  - Flow: Admin confirms payment → Control loads Invoice → marks it as paid.

10) Result of the initial decomposition of use case “UC-10: View Invoices and Payments”

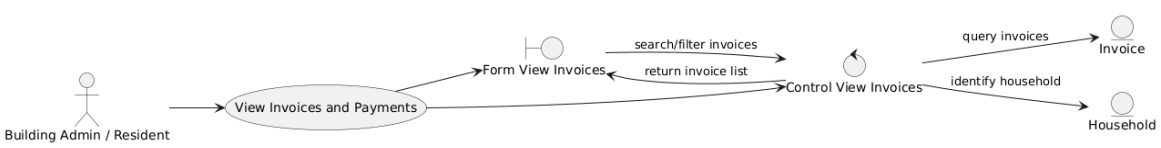


Figure 3-10: Initial decomposition of use case “UC-10: View Invoices and Payments”

- Short description (UC-10):
  - Boundary: Form View Invoices.
  - Control: Control View Invoices handles queries and filtering.
  - Entity: Household and Invoice.
  - Flow: User searches invoices → Control queries Invoice data → returns results.

11) Result of the initial decomposition of use case “UC-11: Submit Feedback”

Troy University Resources

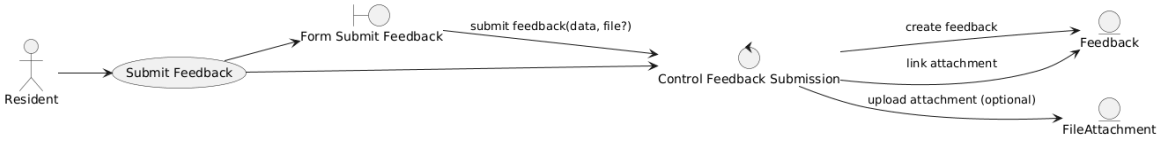


Figure 3-11: Initial decomposition of use case “UC-11: Submit Feedback”

- Short description (UC-11):
  - Boundary: Form Submit Feedback for entering feedback and optional file.
  - Control: Control Feedback Submission coordinates creation and upload.
  - Entity: Feedback and optional FileAttachment.
  - Flow: Resident submits feedback → Control creates Feedback → uploads and links attachment if provided.

12) Result of the initial decomposition of use case “UC-12: Manage Feedback”

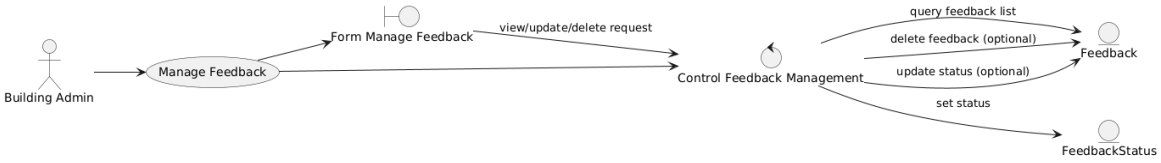


Figure 3-12: Initial decomposition of use case “UC-12: Manage Feedback”

- Short description (UC-12):
  - Boundary: Form Manage Feedback to view and manage feedback.
  - Control: Control Feedback Management coordinates actions.
  - Entity: Feedback and FeedbackStatus.
  - Flow: Admin opens management form → Control loads feedback → updates status or deletes feedback.

13) Result of the initial decomposition of use case “UC-13: Upload / Download File”

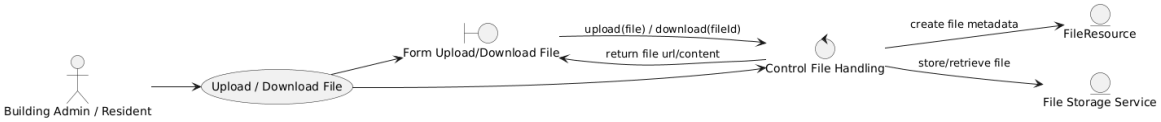


Figure 3-13: Initial decomposition of use case “UC-13: Upload / Download File”

- Short description (UC-13):
  - Boundary: Form Upload/Download File.
  - Control: Control File Handling coordinates file operations.
  - Entity: FileResource and external File Storage Service.
  - Flow: User requests upload/download → Control stores or retrieves via File Storage → returns URL/content.

14) Result of the initial decomposition of use case “UC-14: Login / Logout”

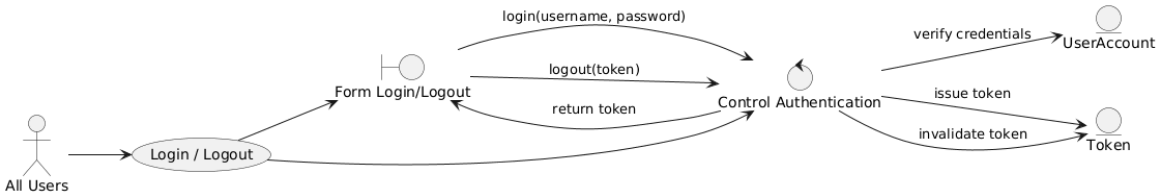


Figure 3-14: Initial decomposition of use case “UC-14: Login / Logout”

- Short description (UC-14):
  - Boundary: Login/Logout form.
  - Control: Control Authentication handles authentication flow.
  - Entity: UserAccount and Token.
  - Flow: User logs in → Control verifies → issues token; user logs out → token invalidated.

15) Result of the initial decomposition of use case “UC-15: Manage Roles and Permissions”

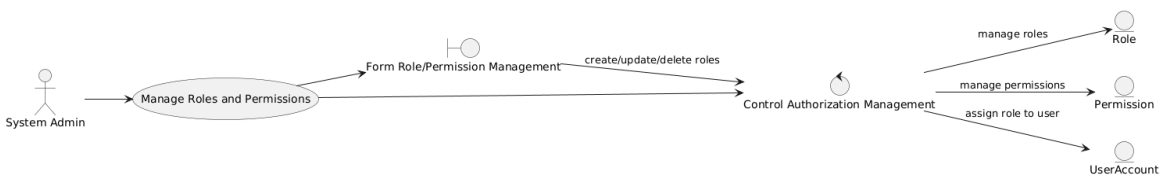


Figure 3-15: Initial decomposition of use case “UC-15: Manage Roles and Permissions”

- Short description (UC-15):
  - Boundary: Form Role/Permission Management.
  - Control: Control Authorization Management coordinates admin actions.
  - Entity: Role, Permission, UserAccount.
  - Flow: Admin manages roles/permissions → assigns roles to users.

3.2. Build Sequence Diagrams

- Sequence Diagram for use case “UC-01: Register New Resident”

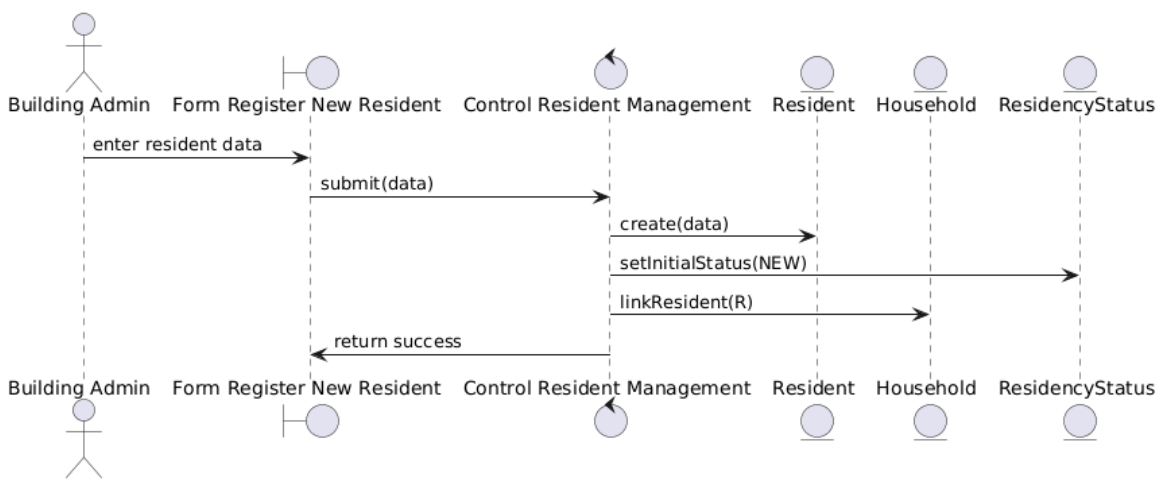


Figure 3-16: Sequence Diagram for use case “UC-01: Register New Resident”

- Sequence Diagram for use case “UC-02: Update Resident Profile”

Troy University Resources

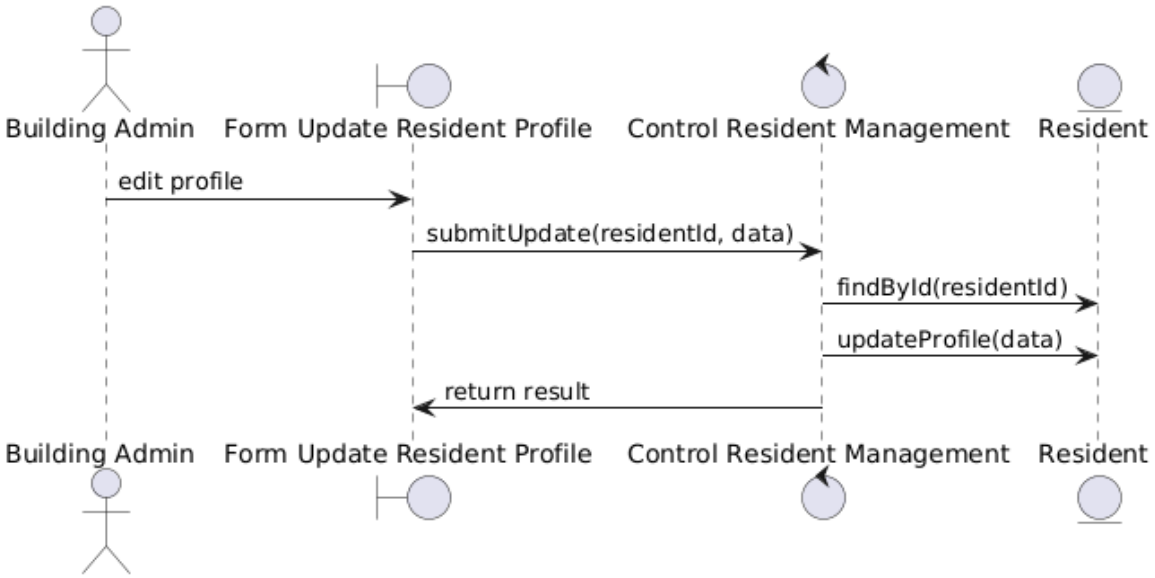


Figure 3-17: Sequence Diagram for use case “UC-02: Update Resident Profile”

- Sequence Diagram for use case “UC-03: Change Residency Status”

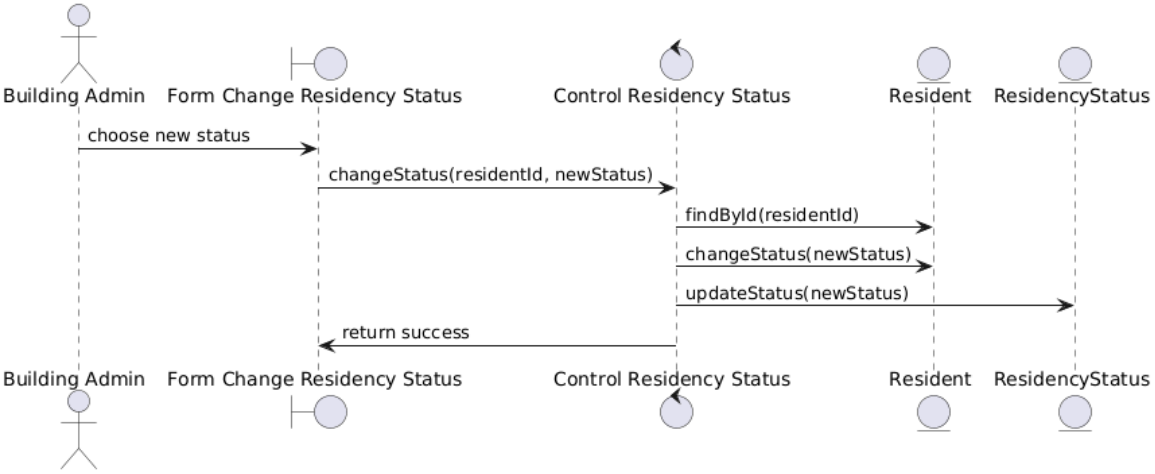


Figure 3-18: Sequence Diagram for use case “UC-03: Change Residency Status”

- Sequence Diagram for use case “UC-04: Manage Household”

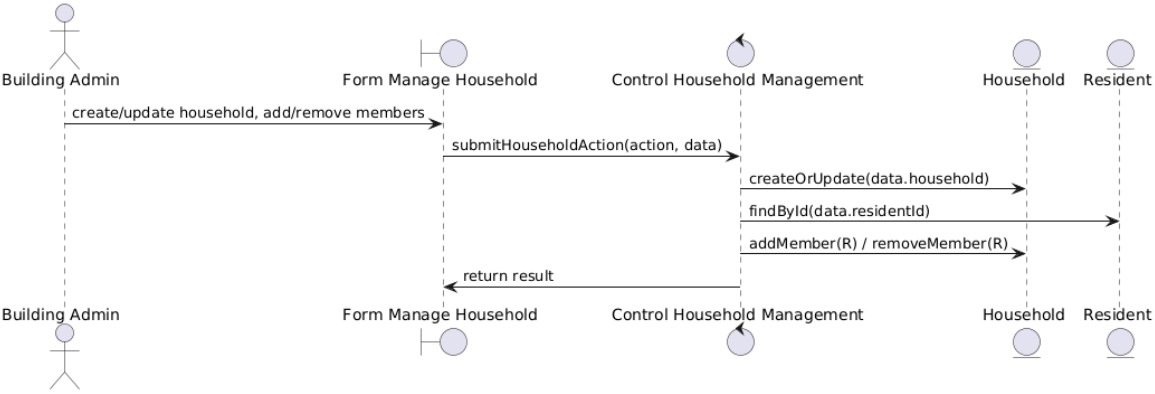


Figure 3-19: Sequence Diagram for use case “UC-04: Manage Household”

- Sequence Diagram for use case “UC-05: View Resident Information”

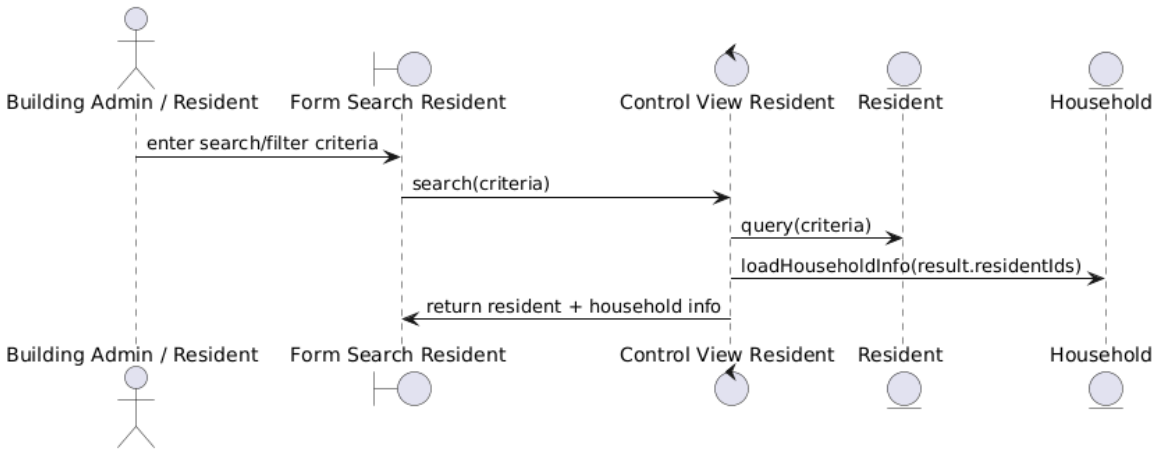


Figure 3-20: Sequence Diagram for use case “UC-05: View Resident Information”

- Sequence Diagram for use case “UC-06: Manage Fee Definitions”

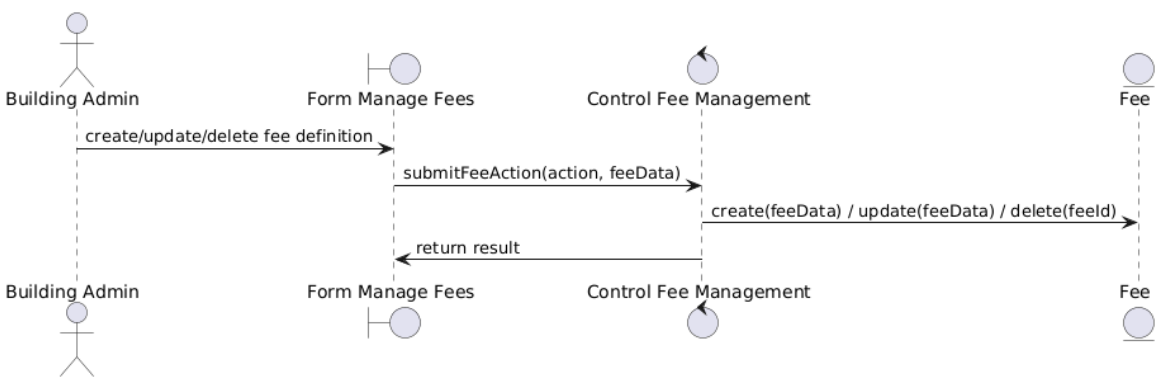


Figure 3-21: Sequence Diagram for use case “UC-06: Manage Fee Definitions”

- Sequence Diagram for use case “UC-07: Calculate Service Fees”

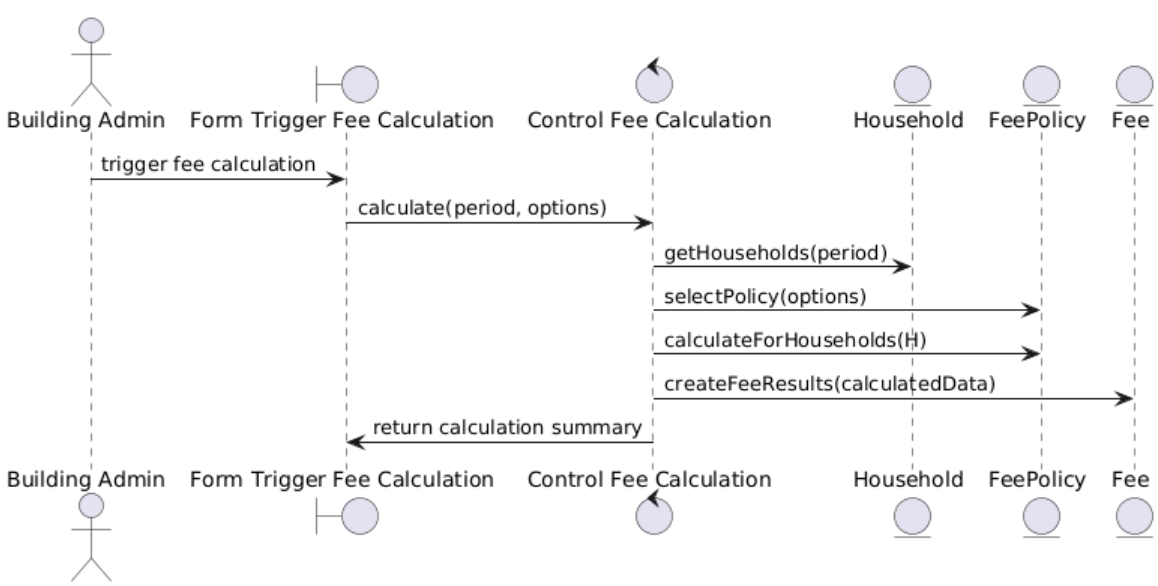


Figure 3-22: Sequence Diagram for use case “UC-07: Calculate Service Fees”

- Sequence Diagram for use case “UC-08: Generate Invoice”

Troy University Resources

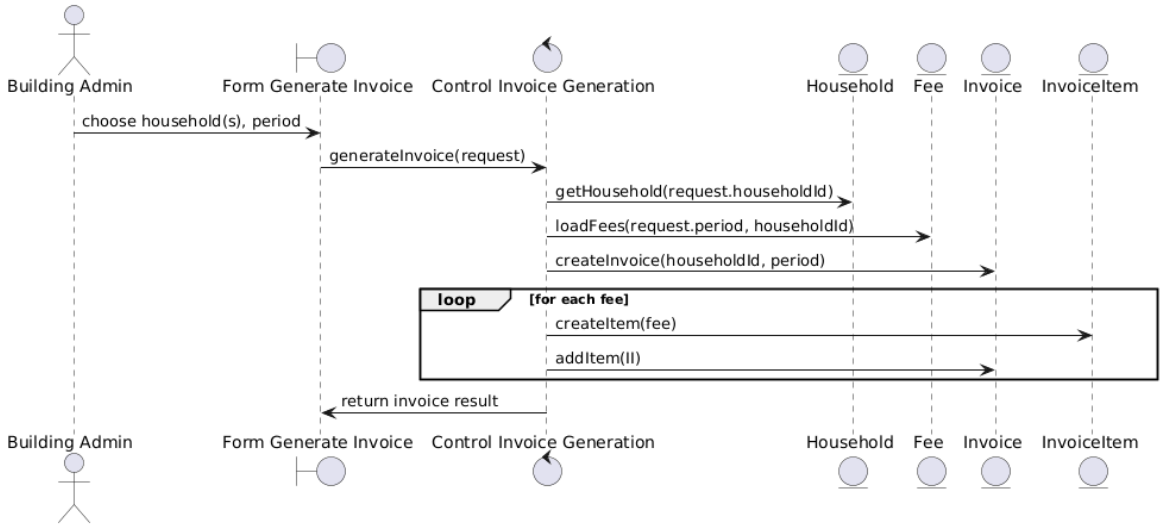


Figure 3-23: Sequence Diagram for use case “UC-08: Generate Invoice”

- Sequence Diagram for use case “UC-09: Mark Invoice as Paid”

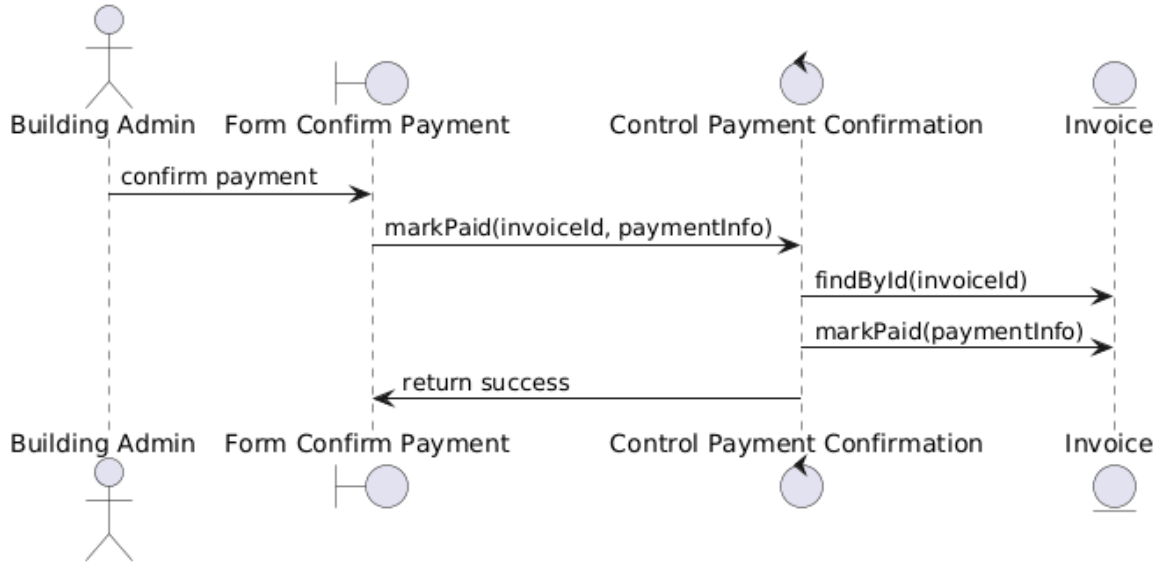


Figure 3-24: Sequence Diagram for use case “UC-09: Mark Invoice as Paid”

- Sequence Diagram for use case “UC-10: View Invoices and Payments”

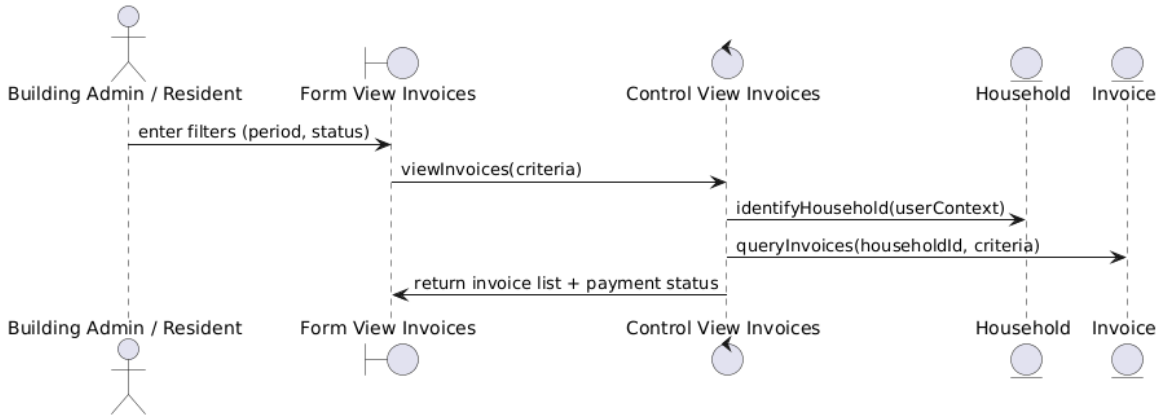


Figure 3-25: Sequence Diagram for use case “UC-10: View Invoices and Payments”

- Sequence Diagram for use case “UC-11: Submit Feedback”

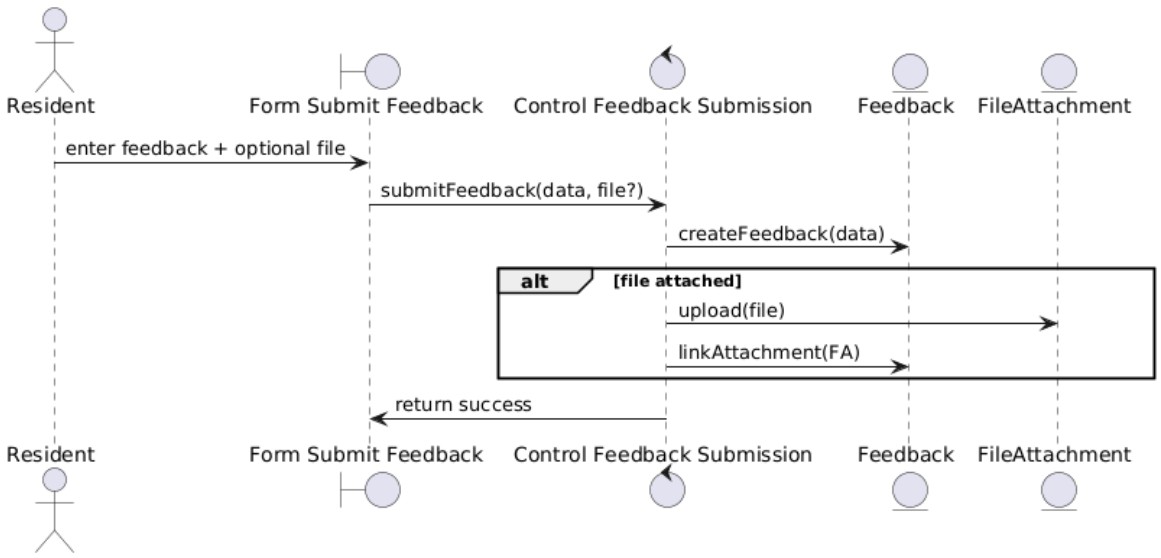


Figure 3-26: Sequence Diagram for use case “UC-11: Submit Feedback”

- Sequence Diagram for use case “UC-12: Manage Feedback”

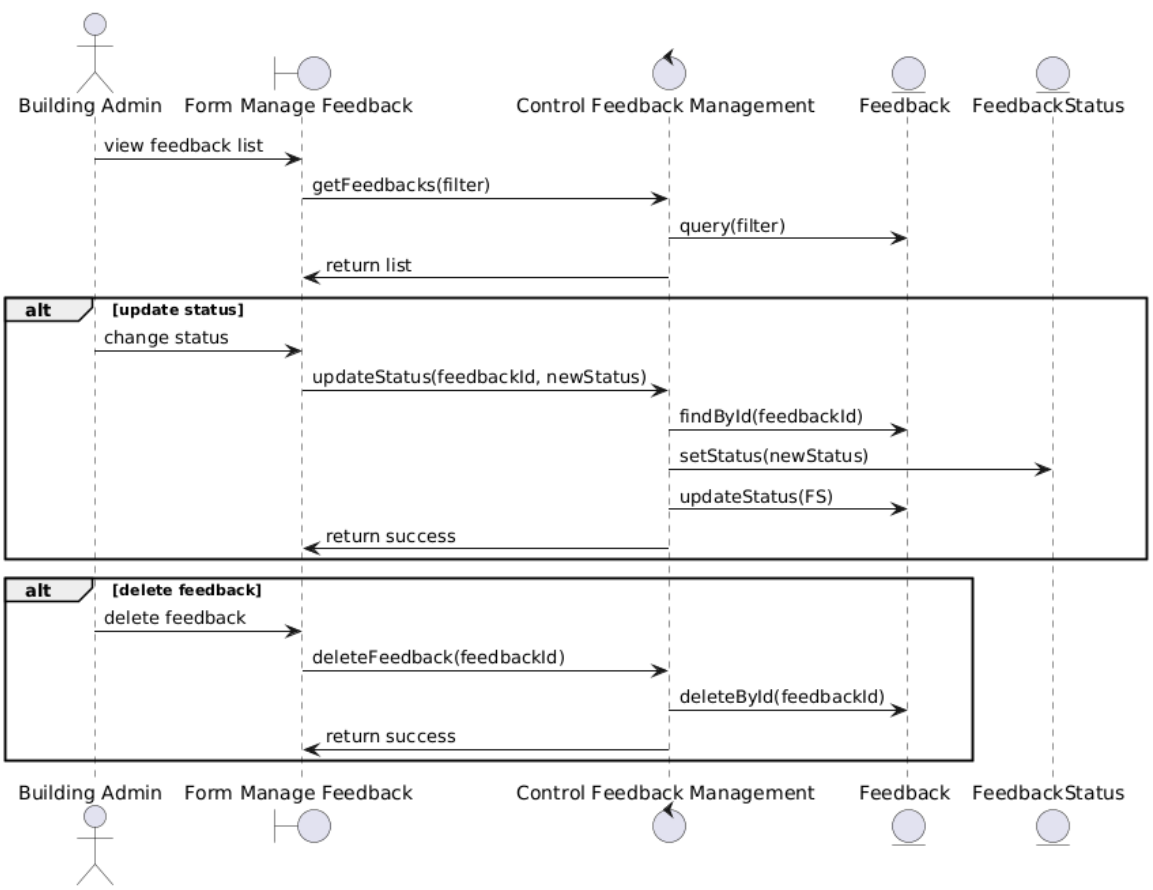


Figure 3-27: Sequence Diagram for use case “UC-12: Manage Feedback”

- Sequence Diagram for use case “UC-13: Upload / Download File”

Troy University Resources

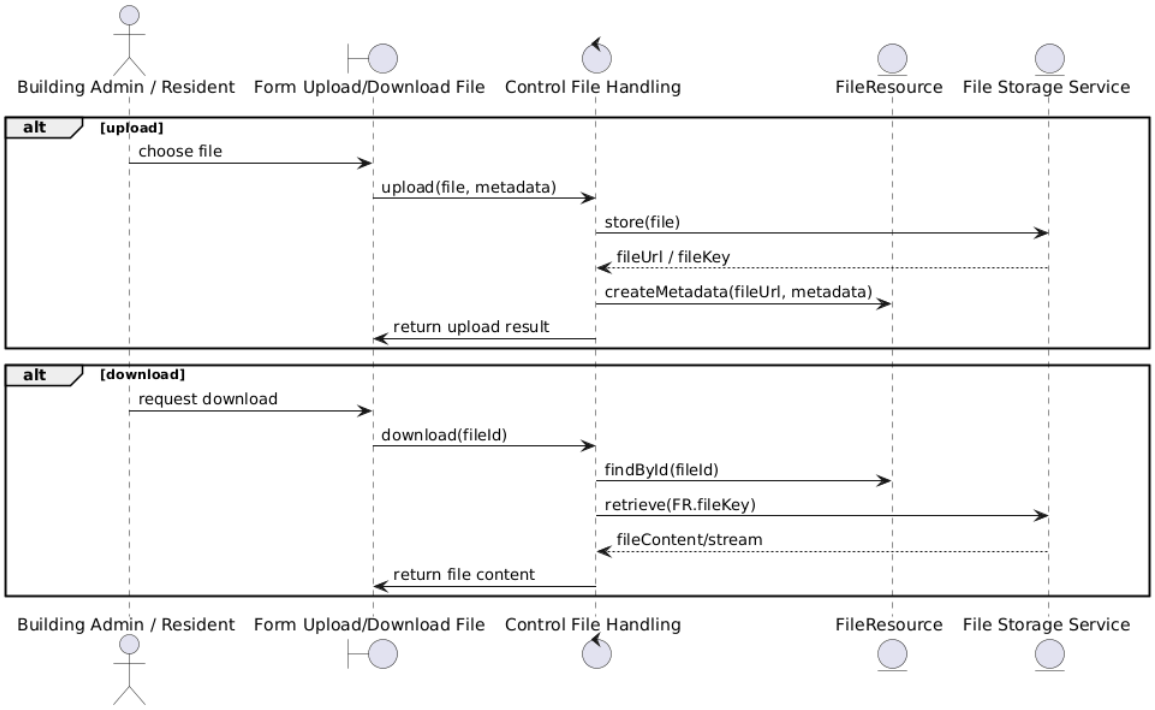


Figure 3-28: Sequence Diagram for use case “UC-13: Upload / Download File”

- Sequence Diagram for use case “UC-14: Login / Logout”

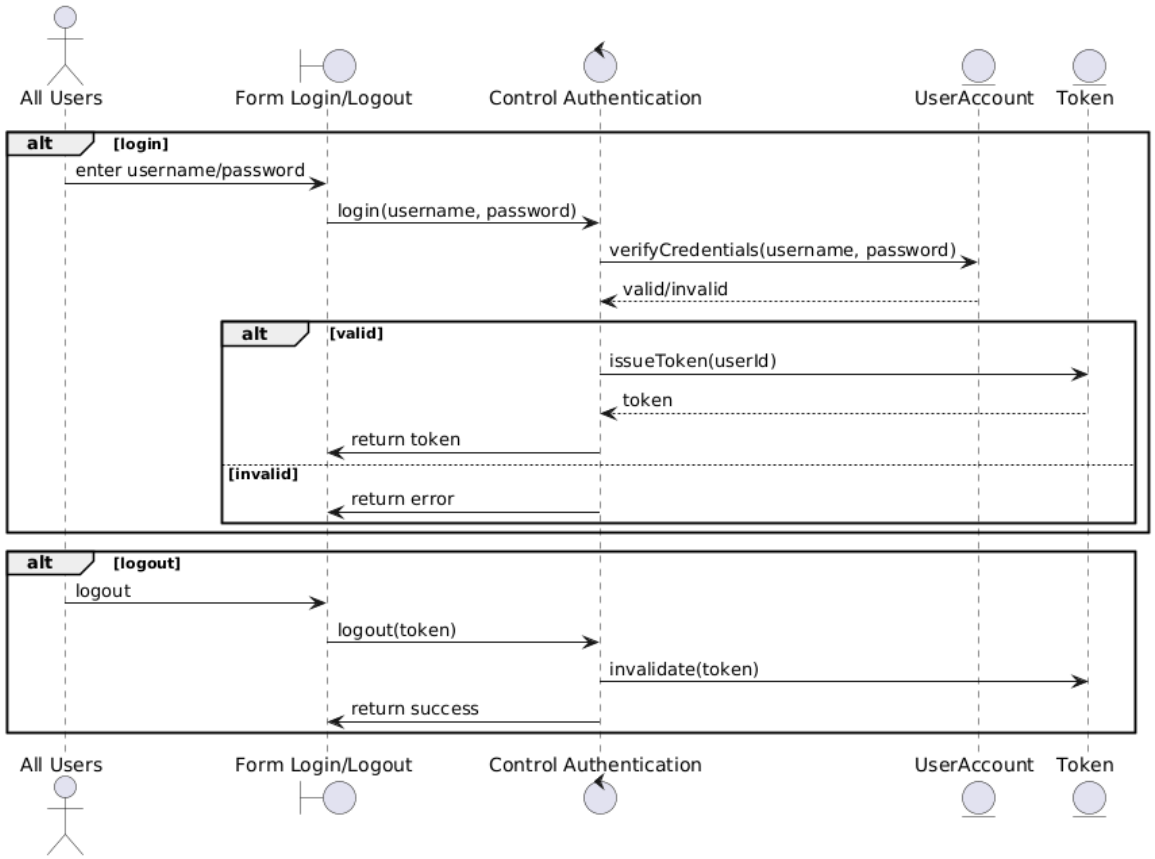


Figure 3-29: Sequence Diagram for use case “UC-14: Login / Logout”

- Sequence Diagram for use case “UC-15: Manage Roles and Permissions”

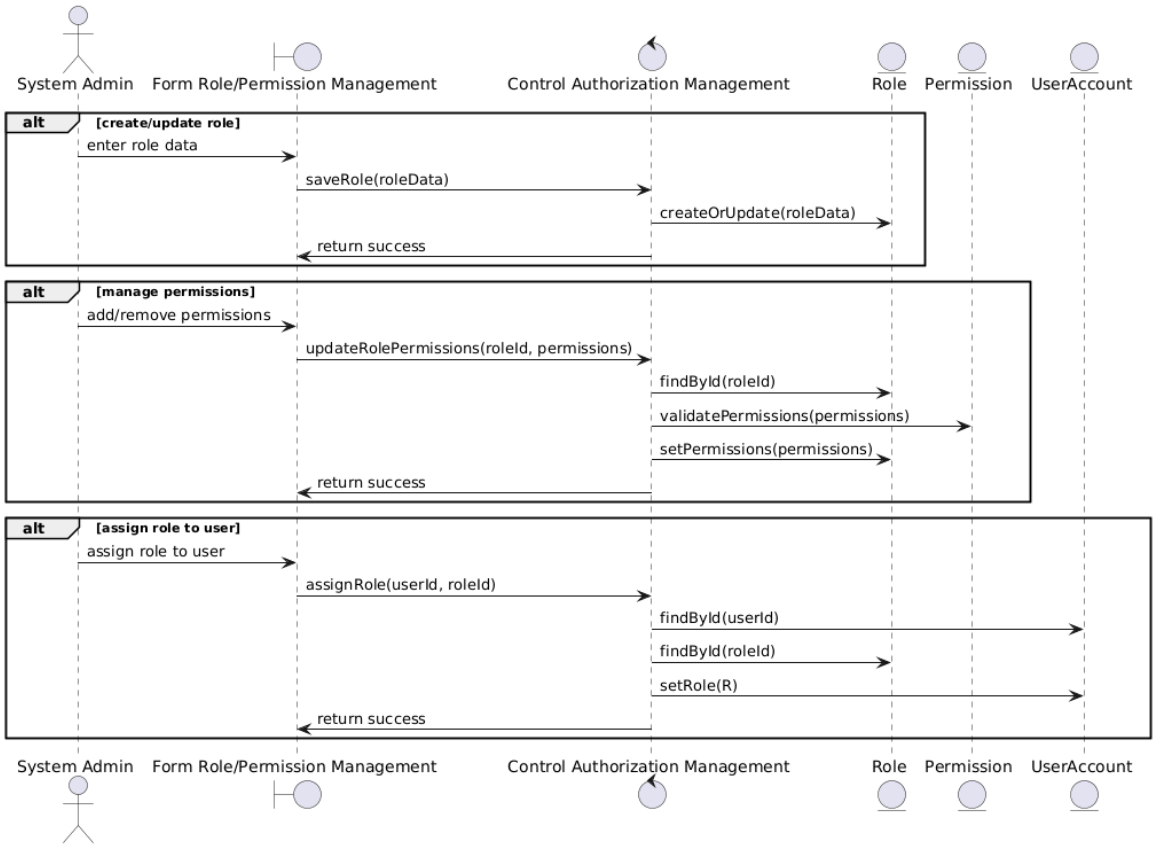


Figure 3-30: Sequence Diagram for use case “UC-15: Manage Roles and Permissions”

Troy University Resources

3.3. Build Analysis Class Diagram

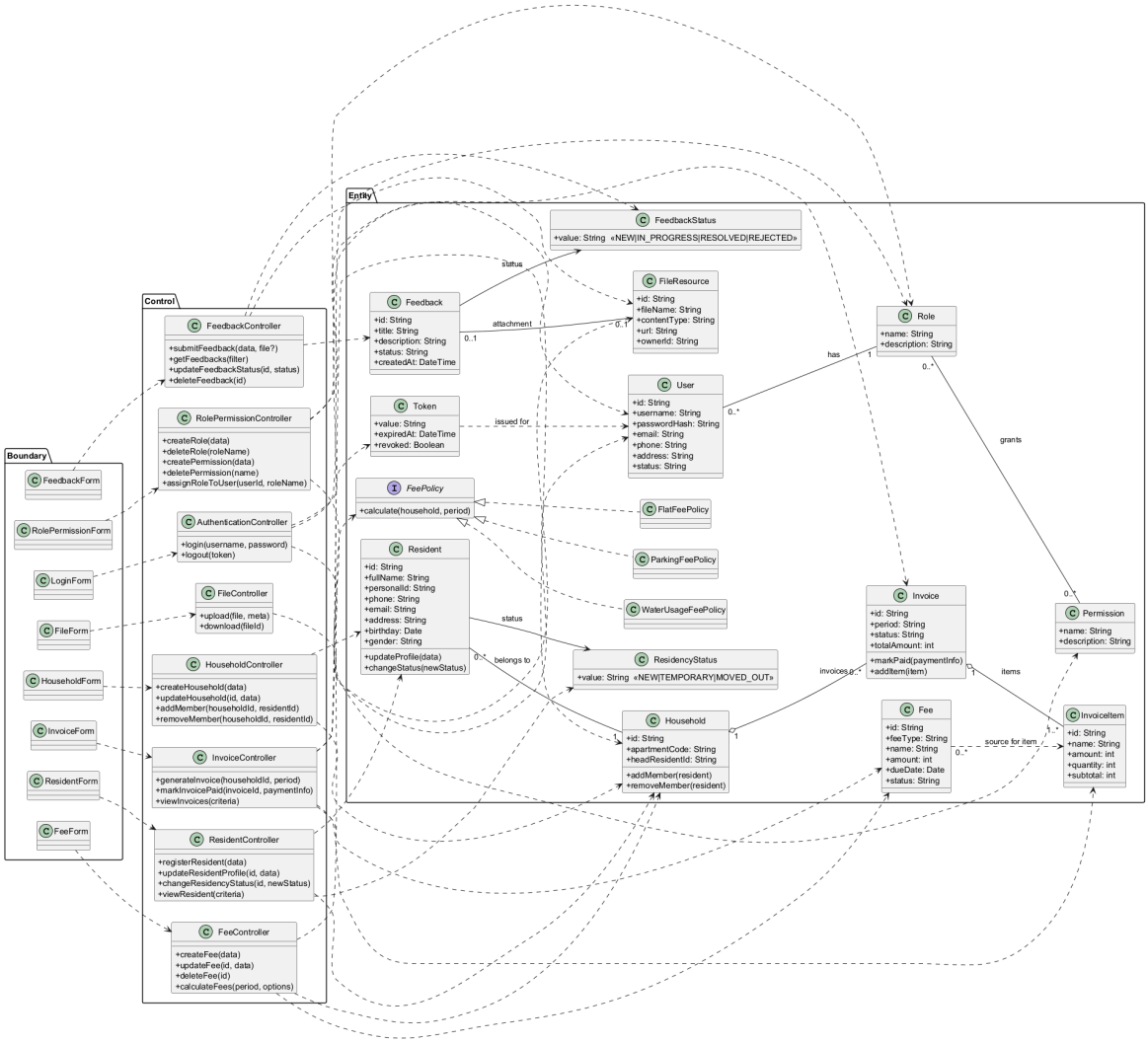


Figure 3-31: Analysis Class Diagram

3.4. Build Entity Relationship Diagram (ERD)

Functional Group 1: Resident and Household Management

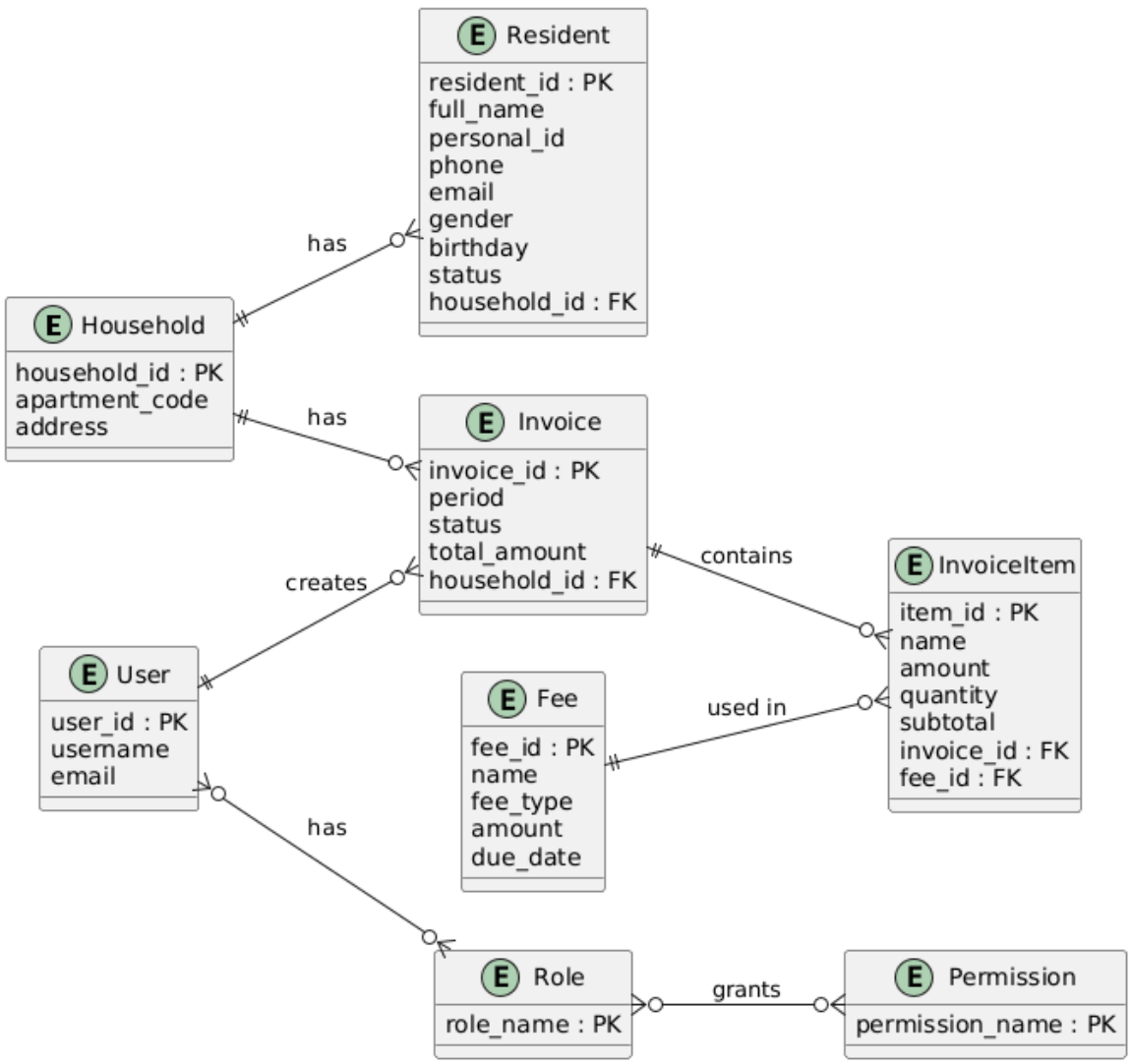


Figure 3-32: ERD for Resident and Household Management System

Relationships description

• Household – Resident (1–N):

One Household can have many Residents.

Each Resident belongs to exactly one Household at a given time.

• Foreign key:

The field household_id in the Resident table is a foreign key referencing Household.household_id. This relationship allows the system to track which household a resident belongs to and to display the member list of a household.

• Business meaning:

When adding a new resident or updating household registration, the system only needs to assign Resident.household_id. When a resident moves out, the system can update the status (for example, MOVED_OUT) and still keep the historical relationship based on the design.

Troy University Resources

Functional Group 2: Fee and Invoice Management (Fee, Invoice)

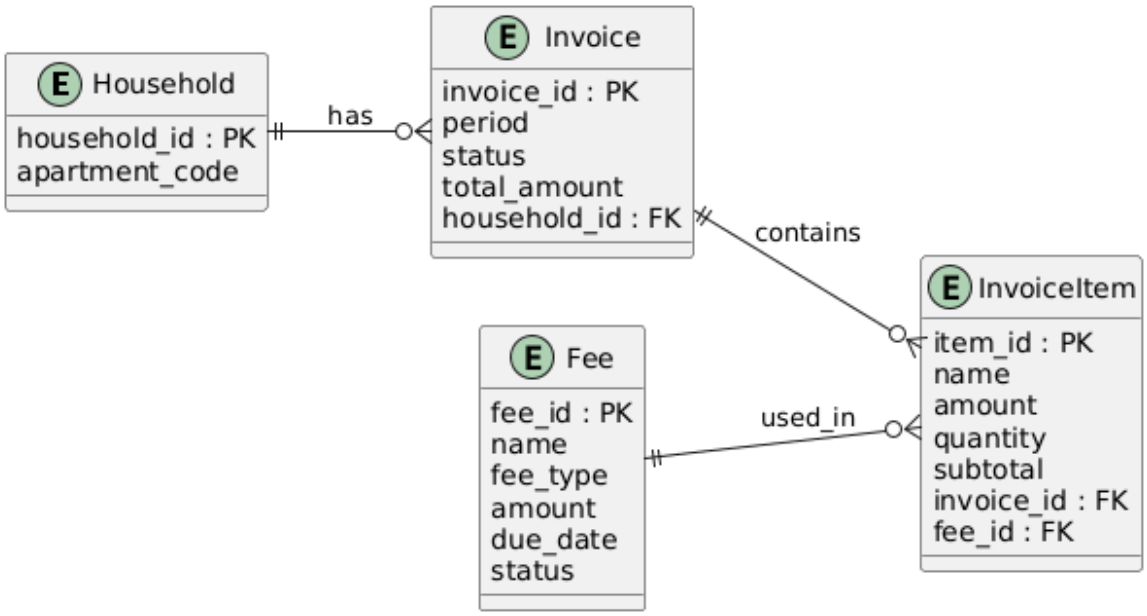


Figure 3-33: ERD for Fee and Invoice Management

Relationship description

• Household – Invoice (1–N):

One Household can generate many Invoices over different periods. Each Invoice belongs to exactly one Household.

• Invoice – InvoiceItem (1–N):

One Invoice consists of many InvoiceItems. Each InvoiceItem belongs to exactly one Invoice.

• Fee – InvoiceItem (1–N):

One Fee can appear in many InvoiceItems across different invoices. Each InvoiceItem references exactly one Fee.

Functional Group 3: Feedback and Attachments (Feedback, File)

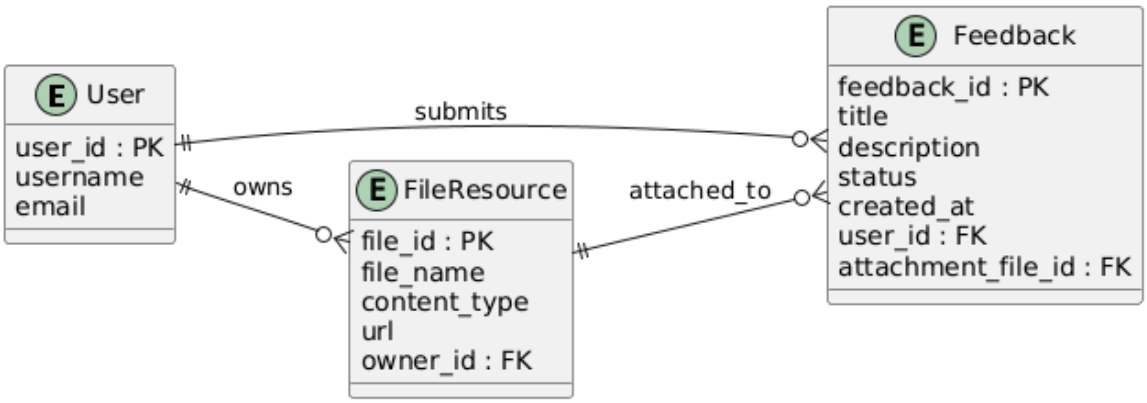


Figure 3-34: ERD for Feedback and Attachments

Relationship description

• User – Feedback (1–N):

One User can submit many Feedback entries, but each Feedback belongs to exactly one User.

• User – FileResource (1–N):

One User can own many FileResources (avatar or attachments). Each FileResource belongs to exactly one User.

• FileResource – Feedback (1–N or 0..1 depending on business rules):

One Feedback can attach one file or none. One FileResource can be reused as an attachment for many Feedback entries if the system allows reuse. If reuse is not allowed, the relationship can be considered 1–0..1 depending on implementation logic.

Functional Group 4: Users and Authorization (User, Role, Permission)

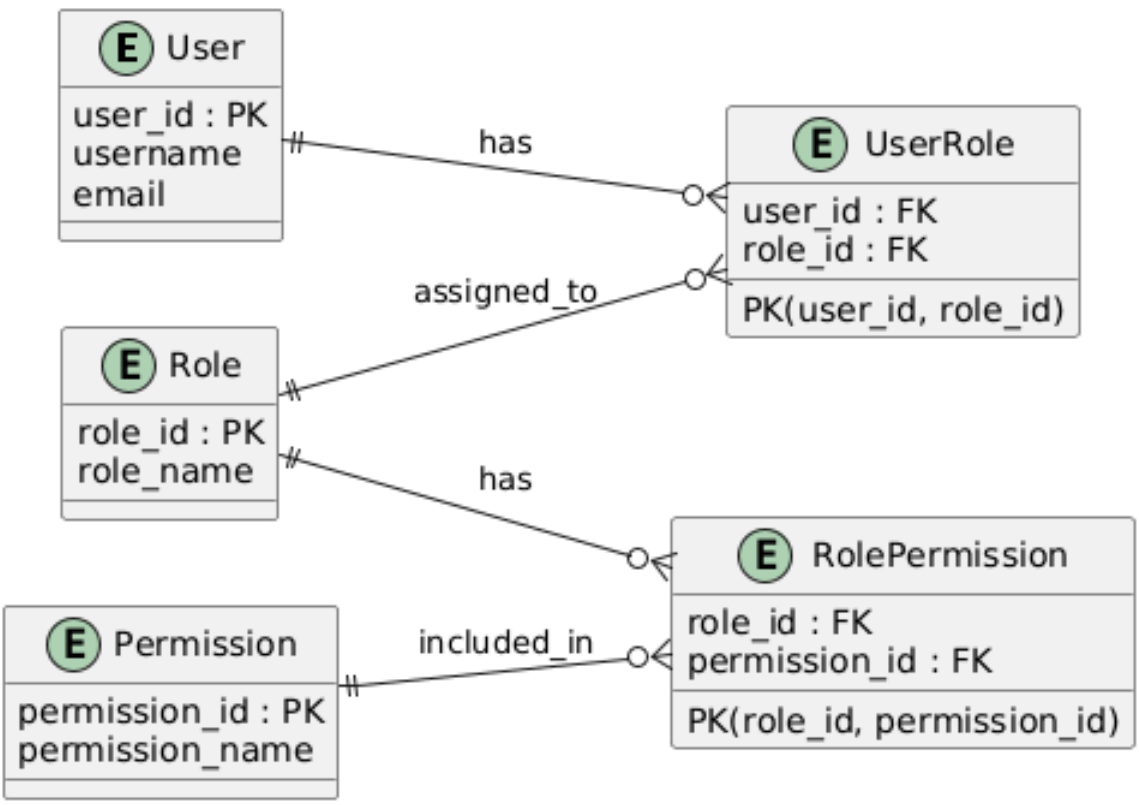


Figure 3-35: ERD for Users and Authorization

Relationship description

• User – Role (N–N) through UserRole:

One User can have many Roles, and one Role can be assigned to many Users. The UserRole table stores this relationship.

• Role – Permission (N–N) through RolePermission:

One Role can have many Permissions, and one Permission can belong to many Roles. The RolePermission table stores this relationship.

• Composite primary keys:

UserRole uses (user_id, role_id) as a composite primary key.

RolePermission uses (role_id, permission_id) as a composite primary key to avoid duplicate links.

Troy University Resources

Functional Group 5: File Management and File Linking with Other Functions

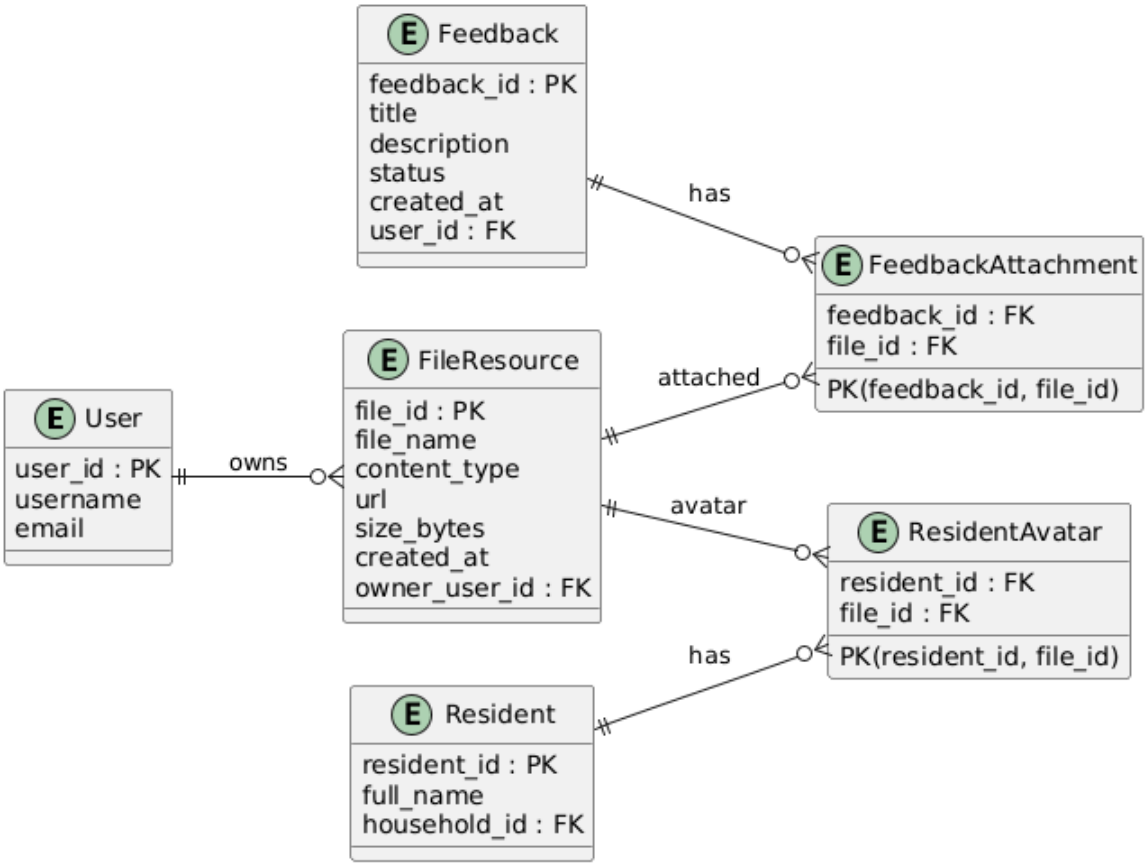


Figure 3-36: ERD for File Management and File Linking with Other Functions

Describe relationship

• User – FileResource (1–N):

One User can upload many files (FileResource). Each FileResource belongs to exactly one User as the owner. This relationship supports access control and tracking of who uploaded the file.

• Feedback – FileResource (N–N) through FeedbackAttachment:

One Feedback can attach many files, and one file can be attached to many Feedback entries if reuse is allowed. The intermediate table FeedbackAttachment stores this relationship and avoids file data duplication.

• Resident – FileResource (N–N) through ResidentAvatar:

Each Resident can have an avatar (image file). The design uses the ResidentAvatar table for flexibility:

CHAPTER 4. SYSTEM DESIGN

4.1. Architecture Design

4.1.1. Selected Software Architecture

The system uses:

- 3-tier architecture combined with MVC and REST
- Client–Server model

Layered structure:

1. Presentation Layer
   - Frontend: Next.js (TypeScript, Tailwind)
   - Backend: Spring Controllers (REST API)
2. Application/Service Layer
   - Service classes handle business logic
3. Data Layer
   - Entity and Repository (Spring Data JPA)
   - MySQL database

This architecture is suitable because:

- It separates UI from business logic
- It is easy to extend modules (user, fee, feedback, role, etc.)
- It fits well with Spring Boot and Next.js

4.1.2. Applying MVC Architecture in the Project

View (V) On the frontend:

- app/ in Next.js:
  - login, register, profile, services, admin, etc.
- components/: forms, tables, modals, buttons
- utils/: API calls, token storage

Responsibilities:

- Display user interface
- Send REST requests
- Receive and render responses

Controller (C) On the backend:

Package:

- com.team.ResidentManagement.controller

Examples:

- AuthenticationController
- UserController
- FeeController
- FeedbackController

Troy University Resources

- FileController
- RoleController
- PermissionController

Responsibilities:

- Receive requests from frontend
- Call corresponding services
- Return ApiResponse

Model (M) Includes:

- Entity layer:
  entity/
  User, Role, Permission, Fee, Feedback, Wanted, InvalidatedToken, etc.
- Service layer:
  service/
  UserService, FeeService, FeedbackService, AuthenticationService, etc.
- Repository layer:
  repository/
  UserRepository, FeeRepository, FeedbackRepository, etc.
- DTO layer:
  dto/request, dto/response

4.1.3. UML Package Diagram cho dự án của bạn

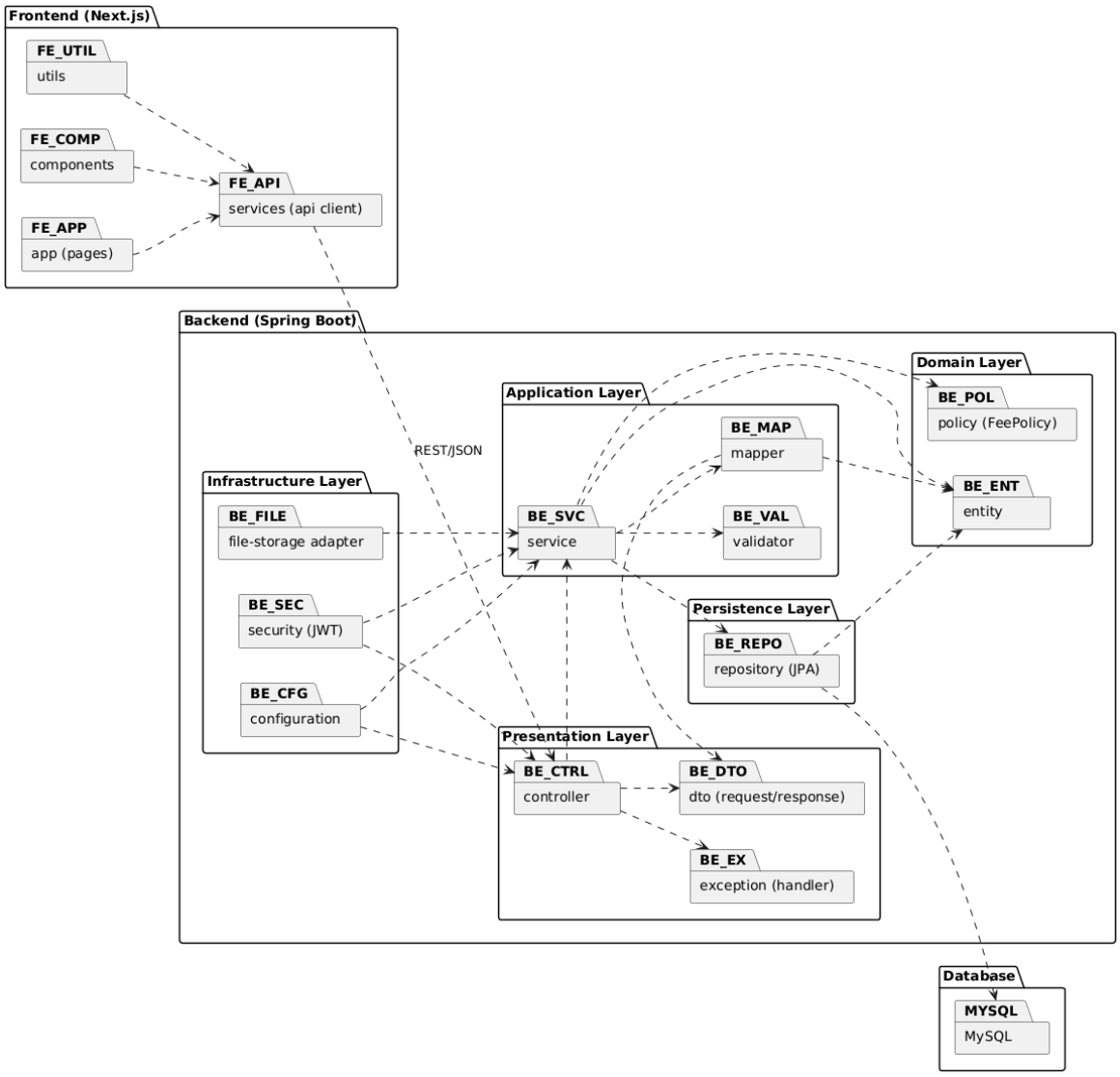


Figure 4-1: UML Package Diagram

Frontend

- app: functional pages
- components: reusable UI components
- utils/services: API calls and token handling

Backend

- controller: receives REST requests
- dto: intermediate data for requests and responses
- service: handles business logic
- entity: data models
- repository: database access
- configuration: security, CORS, JWT, Redis
- mapper: converts between Entity and DTO
- exception: centralized error handling
- validator: data validation

Troy University Resources

- Database
- MySQL: stores all system data

Design Rules Compliance

- Lower layers do not depend on upper layers

  (entities do not call controllers)

- No layer skipping

  (controller → service → repository)

- No circular dependencies
- Each package has a clear responsibility

4.2. Database Design

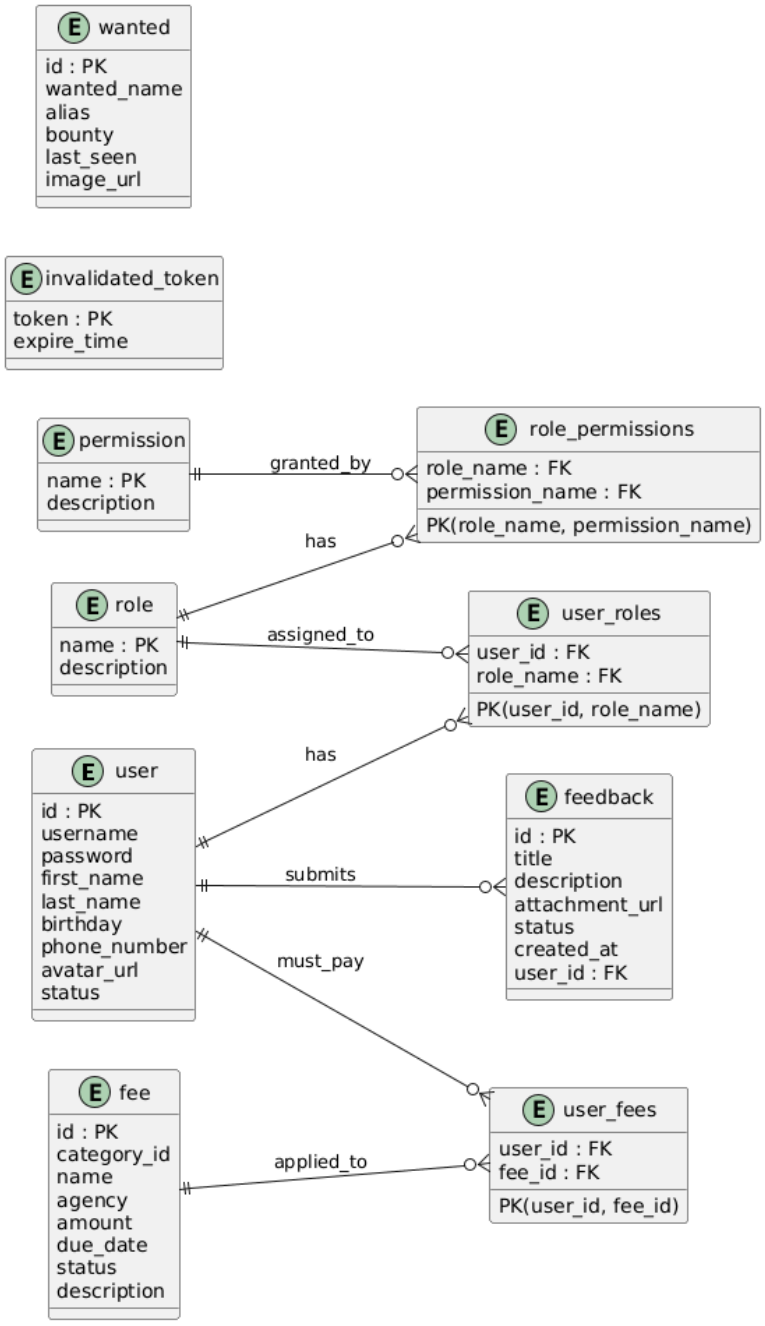


Figure 4-2: ERD for User, Role, Permission, Feedback and Fee

Database Design

- The database design of the system is organized into several functional groups: user and authorization management, fee management, feedback management, logout token management, and an independent business module.

1) User and Personal Information Group

- The user table stores system user information, including username, password, full name, date of birth, phone number, avatar, and account status.

2) Authorization Group (Role, Permission)

- The role table stores role information such as ADMIN, RESIDENT, and SYSTEM_ADMIN.
- The permission table stores detailed operation rights, for example VIEW_INVOICE and MANAGE_FEE.
- Because one user can have multiple roles and one role can be assigned to many users, the system uses the user_roles table to represent the many-to-many relationship between user and role.
- Similarly, since one role can have many permissions and one permission can belong to many roles, the system uses the role_permissions table to represent the many-to-many relationship between role and permission.

3) Fee Management Group

- The fee table stores information about fees that residents must pay, including fee name, collecting agency, amount, due date, status, and description.
- Because one user may have to pay many fees and one fee can apply to many users, the system uses the user_fees table to link user with fee in a many-to-many relationship.

4) Feedback Group

- The feedback table stores feedback information such as title, content, status, created date, and attachment file path if any.
- Each feedback belongs to exactly one user, so feedback.user_id is a foreign key referencing user.id. This forms a one-to-many relationship from user to feedback.

5) Logout Security Group (JWT Invalidation)

- The invalidated_token table stores tokens that have been revoked when users log out.
- This prevents revoked tokens from being reused before they naturally expire, supporting the system's security requirements.

6) Independent Business Table (Wanted)

- The wanted table stores data of wanted persons used for lookup and display features, if this module is included.
- This table is independent and does not require foreign key relationships with other tables.

Relationships Between Data Entities

- user and role have a many-to-many relationship, implemented by user_roles.
- role and permission have a many-to-many relationship, implemented by role_permissions.
- user and fee have a many-to-many relationship, implemented by user_fees.

Troy University Resources

- user and feedback have a one-to-many relationship: one user can submit many feedback entries, and each feedback belongs to exactly one user.

4.2.1. Detailed Specifications of Selected Database Tables

- The “users” table stores information about application users, including login data, personal details, account status, and links to roles, feedback, and assigned fees.

Table 4-1: Specification of Table “users”

Field Name	Data Type	Size	Integrity Constraints	Format	Notes
id	VARCHAR	255	PK, NOT NULL	UUID	Primary key, auto-generated
username	VARCHAR	255	UNIQUE, NULL	String	Login name
password	VARCHAR	255	NULL	Encrypted string	Hashed password
gender	VARCHAR	50	NULL	String	Male/Female/Other
first_name	VARCHAR	255	NULL	String	First name
last_name	VARCHAR	255	NULL	String	Last name
birthday	DATE		NULL	YYYY-MM-DD	Date of birth
email	VARCHAR	255	NULL	Email	User email
address	VARCHAR	255	NULL	String	Address
personal_id	VARCHAR	255	NULL	ID/CCCD	National ID
phone_number	VARCHAR	20	NULL	Digits	Phone number
avatar_url	VARCHAR	255	NULL	URL	Avatar image
status	VARCHAR	50	NULL	String	Account status

Relations:

- N–N with role via user_roles
- N–N with fee via user_fees
- 1–N with feedback

- The “roles” table stores system roles such as administrator, resident, and system admin, which are used to classify users by permission level.

Table 4-2: Specification of Table “roles”

Field Name	Data Type	Size	Integrity Constraints	Format	Notes
name	VARCHAR	255	PK, NOT NULL	String	Role name (ADMIN, USER, etc.)
description	VARCHAR	255	NULL	String	Role description

Relations:

- N–N with user via user_roles
- N–N with permission via role_permissions

- The “permissions” table stores the list of detailed system permissions, where each permission represents a specific action that a user is allowed to perform

Table 4-3: Specification of Table “permissions”

Field Name	Data Type	Size	Integrity Constraints	Format	Notes
name	VARCHAR	255	PK, NOT NULL	String	Permission code
description	VARCHAR	255	NULL	String	Permission meaning

Relations:

- N–N with role via role_permissions

• The “fees” table stores information about service fees that residents need to pay, including fee name, fee type, amount, due date, and payment status.

Table 4-4: Specification of Table “fees”

Field Name	Data Type	Size	Integrity Constraints	Format	Notes
id	VARCHAR	255	PK, NOT NULL	UUID	Primary key
fee_type	VARCHAR	50	NULL	String	Type of fee
category_id	VARCHAR	255	NULL	String	Category code
name	VARCHAR	255	NULL	String	Fee name
agency	VARCHAR	255	NULL	String	Collecting agency
amount	INT		NULL	Number	Fee amount
due_date	DATETIME		NULL	YYYY-MM-DD HH:mm:ss	Due date
status	VARCHAR	50	NULL	String	Payment status
description	VARCHAR	255	NULL	String	Fee description

Relations:

- N–N with user via user_fees
- The “feedback” table stores user feedback and complaints, including title, content, processing status, and sender information.

Troy University Resources

Table 4-5: Specification of Table “feedback”

Field Name	Data Type	Size	Integrity Constraints	Format	Notes
id	VARCHAR	255	PK, NOT NULL	UUID	Primary key
name	VARCHAR	255	NULL	String	Sender name
email	VARCHAR	255	NULL	Email	Sender email
phone	VARCHAR	20	NULL	Digits	Sender phone
address	VARCHAR	255	NULL	String	Sender address
title	VARCHAR	255	NULL	String	Feedback title
description	TEXT/VARCHAR		NULL	Text	Feedback content
attachment_url	VARCHAR	255	NULL	URL	Attached file
status	VARCHAR	50	NULL	Enum string	Feedback status
created_at	DATE		NULL	YYYY-MM-DD	Creation date
user_id	VARCHAR	255	FK, NULL	UUID	References user.id

Relations:

- user 1–N feedback (one user can submit many feedbacks)

CHAPTER 5. DEMONSTRATION SYSTEM DEVELOPMENT

5.1. Libraries and Tools Used

Table 5-1: List of Libraries and Tools Used

Purpose	Tool / Technology	Version Used	Official URL
Frontend Framework	Next.js	14.x	https://nextjs.org
Programming Language (Frontend)	TypeScript	5.x	https://www.typescriptlang.org
CSS Framework	Tailwind CSS	3.x	https://tailwindcss.com
Backend Framework	Spring Boot	3.x	https://spring.io/projects/spring-boot
Programming Language (Backend)	Java	17 (LTS)	https://www.oracle.com/java
ORM Framework	Hibernate (JPA)	6.x	https://hibernate.org
Security Framework	Spring Security	6.x	https://spring.io/projects/spring-security
API Style	RESTful API	—	https://restfulapi.net
Database	MySQL	8.0	https://www.mysql.com
Containerization	Docker	24.x	https://www.docker.com
Cloud Platform	AWS (ECS/EKS)	—	https://aws.amazon.com
Reverse Proxy / Load Balancer	Nginx	1.24	https://nginx.org
CI/CD	GitHub Actions	—	https://github.com/features/actions
Infrastructure as Code	Terraform	1.6	https://www.terraform.io
Kubernetes Package Manager	Helm	3.14	https://helm.sh
IDE (Backend)	IntelliJ IDEA / Eclipse	2023.x / Oxygen	https://www.jetbrains.com/idea / https://www.eclipse.org
IDE (Frontend)	Visual Studio Code	1.86	https://code.visualstudio.com
Version Control	Git	2.43	https://git-scm.com
Testing (Backend)	JUnit	5.x	https://junit.org
API Testing	Postman	10.x	https://www.postman.com
Browser Testing	Google Chrome	120+	https://www.google.com/chrome

5.2. Demonstration System Results

5.2.1. System Packaging Structure

The application is developed as a management system consisting of three main packaged components: Backend (Server), Frontend (Client), and Database.

5.2.2. Packaged Product 1: Backend (Server)

- **Components:**
  Includes REST APIs built with Spring Boot, handling business logic for functions such as user management, authorization, fee management, invoicing, feedback, authentication, and role-based

Troy University Resources

access control using JWT. The backend interacts with the MySQL database through Spring Data JPA (Hibernate) and integrates Spring Security to secure the system.

- **Role and Purpose:**
  The backend is the core of the system, responsible for receiving and processing all requests from the frontend. It validates data, executes business logic, ensures data integrity and security, and provides APIs for the frontend to communicate and exchange data.
- **Main Technologies:**
  Spring Boot, Spring Data JPA (Hibernate), Spring Security, JWT, MySQL, REST API.

5.2.3. Packaged Product 2: Frontend (Client)

- **Components:**
  The user interface is built with Next.js using TypeScript and Tailwind CSS. The frontend includes functional pages such as login, resident management, fee management, invoice viewing, feedback submission, and role management. UI components include data tables, input forms, dashboard pages, and reusable UI elements.
- **Role and Purpose:**
  The frontend is the direct interaction layer for users, allowing them to work with the system in an intuitive and convenient way. It sends requests to the backend via REST APIs and displays processing results to users.
- **Main Technologies:**
  Next.js, TypeScript, Tailwind CSS.

5.2.4. Packaged Product 3: Database

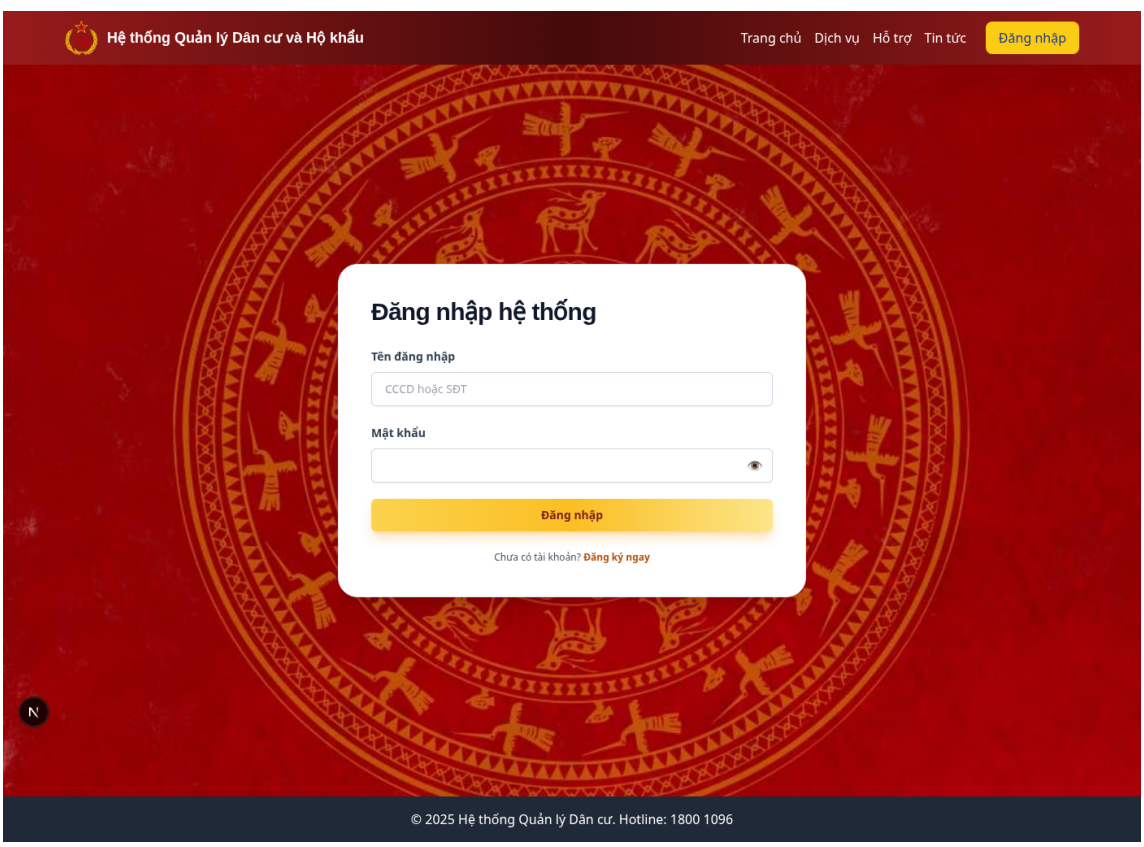
- **Components:**
  The MySQL database stores all system data such as users, roles, permissions, fees, feedback, logout tokens, and intermediate tables.
- **Role and Purpose:**
  It is the long-term data storage layer, ensuring consistency, integrity, and fast data retrieval for the system.
- **Main Technologies:**
  MySQL.

Table 5-2: System Statistics

Statistics Item	Value / Size	Unit	Notes
Total source code size	749	MB	Includes frontend, backend, configuration files, and environment files
Package size: Backend	115	MB	Size after installing all dependencies
Package size: Frontend	415	MB	Size after installing all dependencies
Number of backend API endpoints	46	APIs	Total number of routes in controller files

5.3. Demonstration of System Functions Interface

5.3.1. Login

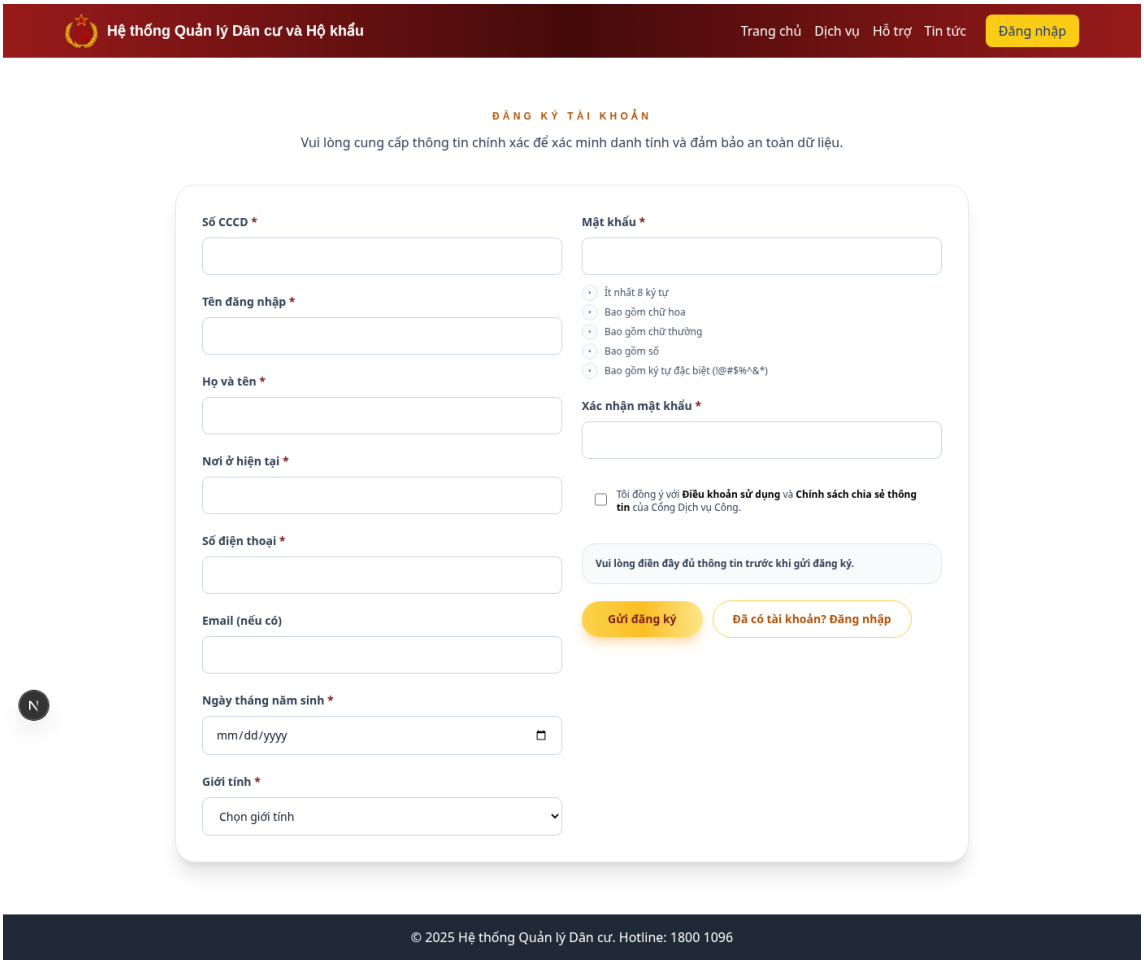


Function description:

- Login form with username (Citizen ID number or phone number) and password.
- Button to show/hide password (eye icon).
- Login button performs authentication.
- “Register now” link for users who do not have an account.

Troy University Resources

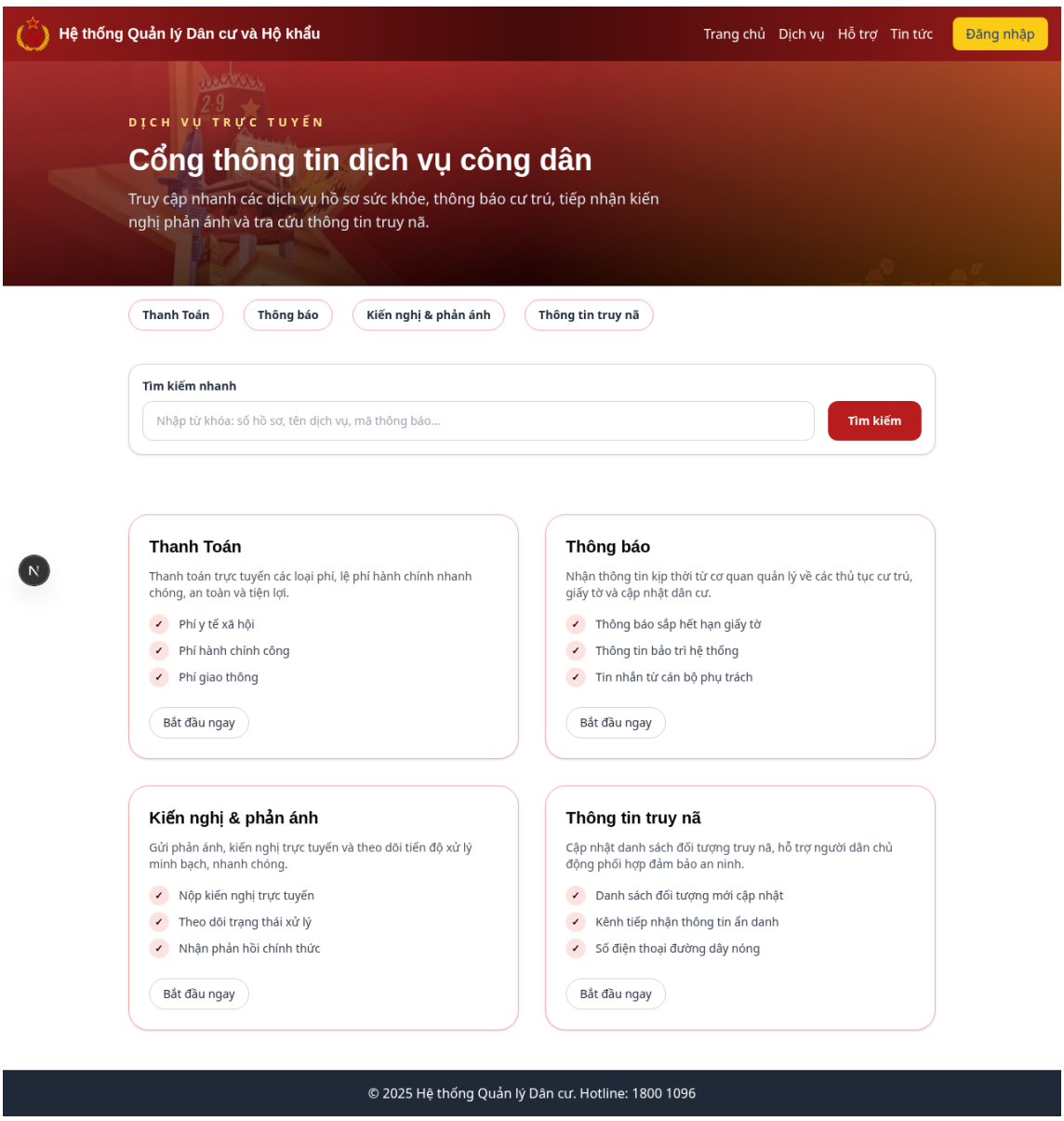
5.3.2. Account Registration



Function description:

- Registration form with full personal information: Citizen ID, username, full name, address, phone number, email, date of birth, gender.
- Password setup with strength requirements (uppercase, lowercase, number, special character).
- Checkbox to agree to terms and policies.
- Submit button to create a new account.
- Link to switch to the login page if the user already has an account.

5.3.3. Online Public Services



Function description:

- Page that aggregates citizen services.
- Quick filter tabs: Payment and Complaints & Feedback.
- Quick search box to look up by keyword (application number, service name, etc.).
- Service cards show short descriptions and a “Start now” button.

Troy University Resources

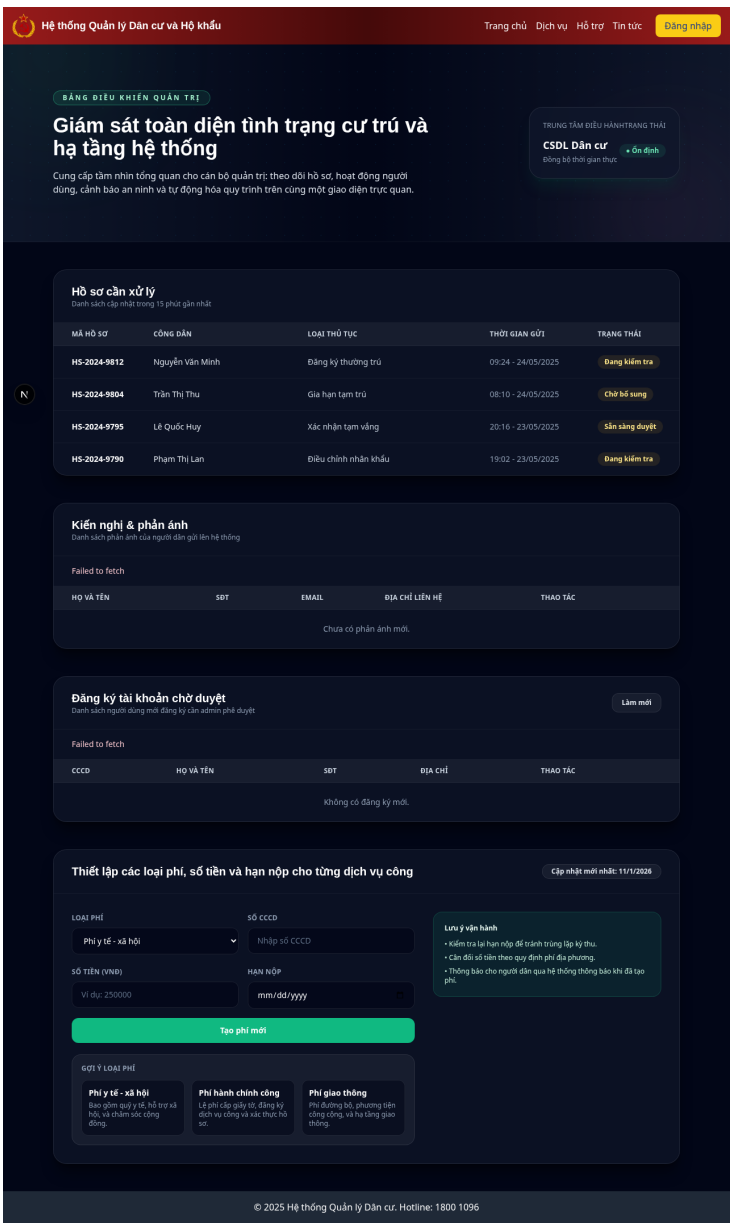
5.3.4. News / Announcements



Function description:

- Page that updates daily announcements from authorities.
- Featured news cards with title, description, and “View details” link.
- Subscribe button so users can follow new information.
- Notification channels: Email, SMS, Mobile app.

5.3.5. Admin Dashboard



Function description:

- Overview monitoring of residency status and system infrastructure.
- “Pending Applications” table showing application ID, citizen, procedure type, submission time, status.
- Complaints & Feedback area with list of reports and processing status.
- Pending account registrations area.
- Form to set service fees, amounts, and deadlines.

Troy University Resources

CONCLUSION AND FUTURE DEVELOPMENT

1. Conclusion

- After a serious process of research, analysis, and development, the team has successfully completed the main objectives of the project “Resident Management System”, inspired by the VNeID administrative management model. The system focuses on core resident–community management problems, including resident registration, household management, service fee calculation, invoice generation, payment confirmation, feedback handling, and role-based access control.
- From a product perspective, the system provides a centralized and structured way to manage resident data and administrative information. Manual and scattered management processes are replaced by a digital platform where data is stored consistently, searched quickly, and updated safely. Core business workflows such as registering residents, changing residency status, generating invoices, and confirming payments have been clearly defined and automated, helping administrators reduce repetitive manual work and minimize errors.
- Beyond technical functions, the system also contributes to transparency and efficiency in residential management. By allowing residents to view invoices, payment status, and submit feedback directly, the system creates a more open and interactive environment between administrators and residents. The user interface is designed to be simple, clear, and easy to use, even for users who are not highly familiar with technology.
- From an academic perspective, this project is a practical demonstration of applying knowledge from software engineering and object-oriented programming courses. The team followed a complete development process, from requirement analysis and use case modeling to UML diagrams (Use Case, Activity, Business Function Diagram) and detailed use case specifications. Through this process, members improved their skills in teamwork, problem analysis, system design, and project organization.

2. Limitations and Future Development

- Although the current version (v1.0) works well for a small to medium residential environment, the team recognizes that there are still limitations when considering large-scale real-world deployment. In the future, the system can be improved and expanded in the following main directions:
- First: Digital Payment Integration
  In the context of a cashless economy, the system should integrate popular digital payment gateways such as Momo, VNPay, ZaloPay, or banking APIs. Each invoice can be attached with a dynamic QR code so that residents can pay instantly by scanning. The system can then automatically reconcile payments without manual confirmation.
- Second: Scalability and Security Enhancement
  To support large residential chains or urban areas, the system should be upgraded to a multi-tenant architecture, allowing multiple buildings or projects to be managed on the same platform. The role-based access control (RBAC) model can also be refined to clearly separate permissions among system administrators, building managers, accountants, staff, and residents. In addition, stronger security mechanisms such as audit logging, encryption, and monitoring should be added.
- Third: Smart and AI-Driven Features
  In the long term, the system can be enhanced with intelligent features:
- Personalized reminders: Using data analysis to send smart payment reminders at suitable times through email, SMS, or app notifications.
- Financial forecasting: Applying machine learning to analyze historical data and predict cash flow, helping administrators detect risks early.
- Automation and support: Integrating OCR to digitize paper documents automatically and deploying an AI chatbot to answer common questions and support residents 24/7.

3. Final Remarks

- With the above orientations, the team hopes that the Resident Management System will not remain only as a course project, but can become the foundation for a practical and scalable solution in the future. This project is a small but meaningful step toward digital transformation in residential and community management, contributing to building smarter, more transparent, and more efficient living environments in the era of Industry 4.0.

Troy University Resources

REFERENCES

[1] Bun Technical Documentation.

[2] ElysiaJS Technical Documentation.

[3] Next.js Technical Documentation.

[4] Drizzle ORM Technical Documentation.

[5] MySQL Official Documentation.

[6] JWT (JSON Web Token) Specification and Usage Guide.

[7] RESTful API Design Guidelines.

[8] Tailwind CSS Documentation.

[9] TypeScript Official Handbook.

[10] Git and GitHub User Guide for Version Control.