Table of Contents

Use case 01: Add Employee 21

Use case 02: Add Employee 23

Use case 03: Request Retirement 25

Use case 04: Transfer Between Departments 27

Use case 05: Assign New Recruit 29

Use case 05: Promote Personnel 31

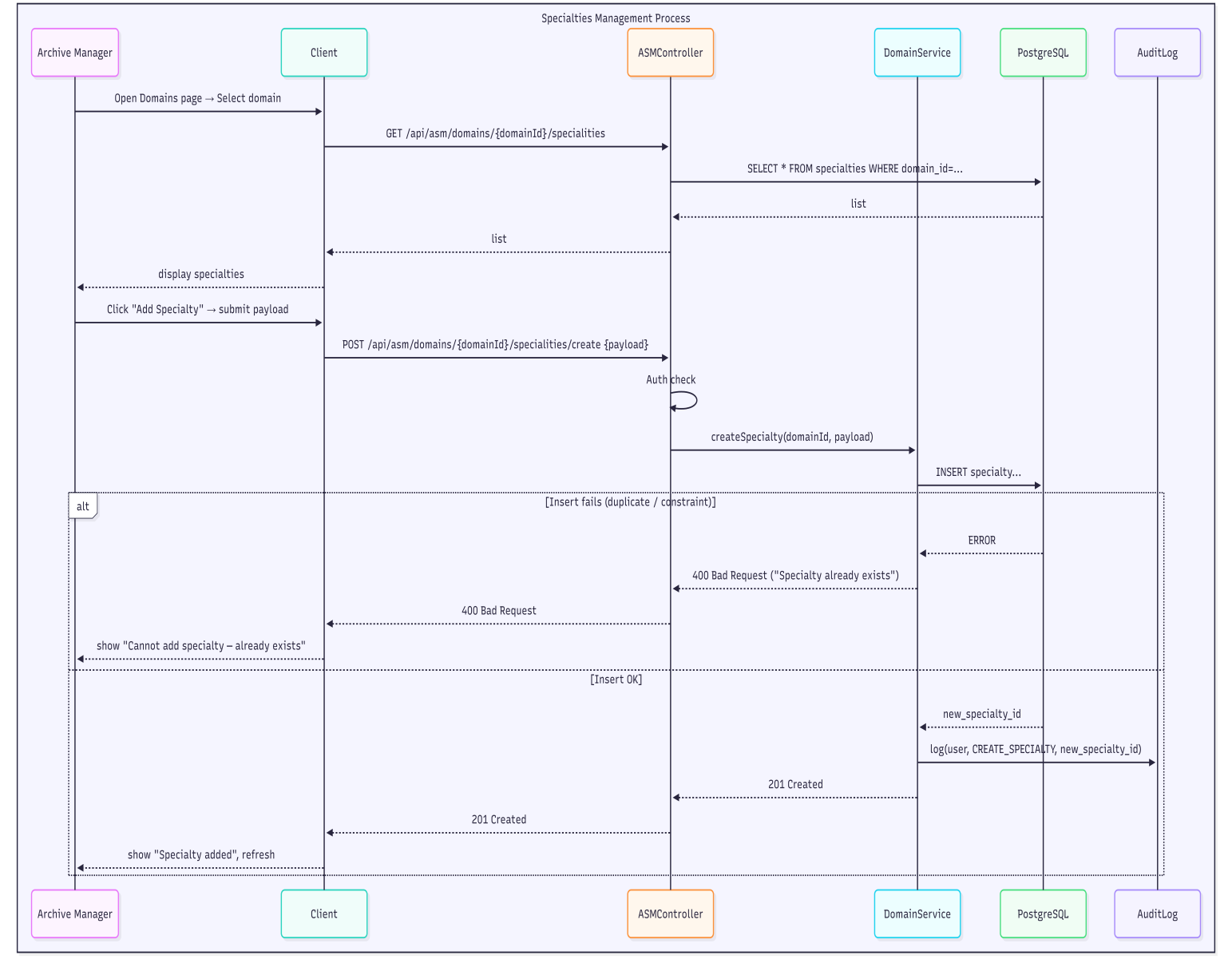
Use case 01: Generate Work Certificate 33

Use case 02: Extract a List of Names (Filtered Export) 35

Use case 03 : Manage Bodies (CRUD) 38

Use case 01: Manage Grades (CRUD) 41

Use case 02: Manage Domains & Specialties (CRUD) 45

 46

Use case 03: Validate Retirement Request 47

Use case 04: Modify Archived Employee 49

Use case 05: Consult Retiree File & Generate Certificate 51

Backups 86

Maintenance 86

# List of Figures

# .MINISTRY OF HIGHER EDUCATION AND

SCIENTIFIC RESEARCH

ABDELHAMID IBN BADIS UNIVERSITY - MOSTAGANEM

Faculty of Exact Sciences and Computer Science

Department of Mathematics and Computer Science

Sector: Computer science

PROJECT REPORT

Option: Computer Systems

Subject: Software engineering

Subject No. 01

THEME :

Development of a Human Resources Application

Students:

* + - “Hattab Amir” Gr: 01.
    - “Bourahla Mohamed Saif Elislam” Gr: 01.
    - “Bounader Mohamed Aness” Gr: 01.
    - “Menshchikov Maxim” Gr: 01.
    - “Zouatine Zakaria” Gr: 01.

Academic Year 2025/2026

**Summary**

This project focuses on the design and development of a **Desktop Human Resources Management System** for a Public Local Health Establishment (EPSP). The primary objective is to digitize and streamline the complex administrative management of the institution's medical, paramedical, and administrative staff.

To ensure the solution meets the specific operational needs of the establishment, our approach relied on a detailed field analysis and direct consultation with current hospital staff. This allowed us to identify the critical bottlenecks in the current manual processes, particularly regarding career progression and document retrieval.

From a technical perspective, we utilized the **Rational Unified Process (RUP)** to structure the development lifecycle and the **Unified Modeling Language (UML)** to model the system's architecture. The resulting **desktop application operates on a client-server architecture**, structured around three key functional areas:

1. **Personnel Management:** Enabling the Manager to handle recruitment, assignments, and the automated calculation of promotions based on complex legal statutes.
2. **Administrative Services:** Allowing the HR Agent to instantly generate work certificates and extract filtered staff lists.
3. **Archives & Configuration:** Permitting the Archive Manager to securely consult retiree files and manage the hierarchical structure of medical specialties.

This application aims to replace paper-based workflows, thereby reducing errors, securing data within the local network, and significantly improving the efficiency of the EPSP’s human resources department.

**List of figures**

|  |  |  |
| --- | --- | --- |
| ***Figure N*** | ***Title*** | ***Page N*** |
| 01 | Static Context Diagram |  |
| 02 | Pckage Diagram |  |
| 03 | Personnel Manager Use Case |  |
| 04 | Agent Use Case |  |
| 05 | Archive Service Manager Use Case |  |
| 06 | Add Employee Sequence Diagram |  |
| 07 | Modify Employee Sequence Diagram |  |
| 08 | Request Retirement Sequence Diagram |  |
| 09 | Department Transfer Sequence Diagram |  |
| 10 | Assign New Recruit Sequence Diagram |  |
| 11 | Promote Personnel Sequence Diagram |  |
| 12 | Generate Work Certificate Sequence Diagram |  |
| 13 | Extract a List Sequence Diagram |  |
| 14 | Manage Body (CRUD) Sequence Diagram |  |
| 15 | Manage Grades (CRUD) Sequence Diagram |  |
| 16 | Manage Domains & Specialties (CRUD) Sequence Diagram |  |
| 17 | Validate Retirement Request Sequence Diagram |  |
| 18 | Modify Archeived Employees Sequence Diagram |  |
| 19 | Consult Retiree File & Generate Certificate Sequence Diagram |  |
| 20 | Overall Class Diagram |  |
| 21 | Overall Object Diagram |  |
| 22 | Component Diagram |  |
| 23 | Deployment Diagram |  |
| 24 | Test of Add Employee Sequence Diagram |  |
| 25 | Test of Modify Employee Sequence Diagram |  |
| 26 | Test of Retirement Validation Sequence Diagram |  |
| 27 | Test of Retirement Certificate Generating Sequence Diagram |  |
| 28 | Authentication System Sequence Diagram |  |
| 29 | Login Page (ScreenShot) |  |
| 30 | Add Employee [PM] (ScreenShot) |  |
| 31 | Employee Page [PM] (ScreenShot) |  |
| 32 | Add Body [Agent] (ScreenShot) |  |
| 33 | Body Detail Page [Agent] (ScreenShot) |  |
| 34 | Employee Page [Agent] (ScreenShot) |  |
| 35 | Retired Employee Page [ASM] (ScreenShot) |  |
| 36 | Modify Retiree [ASM] (ScreenShot) |  |
| 37 | Domains Page [ASM] (ScreenShot) |  |
| 38 | Request Retirement Page [ASM] (ScreenShot) |  |
| 39 |  |  |
| 40 |  |  |
| 41 |  |  |

**General Introduction**

In a sector as demanding as public healthcare, efficiency, precision, and smooth coordination are essential not only in medical practice but also in administrative management. Local Public Health Establishments (EPSP) are responsible for overseeing a wide and diverse workforce—doctors, nurses, paramedics, and administrative staff—distributed across multiple polyclinics and healthcare rooms. Yet many of these institutions still rely on manual processes to manage personnel files, track career progression, handle assignments, and issue administrative certificates. These outdated methods often result in delays, redundant work, data inconsistencies, and a general lack of visibility over the employee lifecycle.  
  
Today, digital transformation has become a necessary response to these challenges. By integrating modern technological tools, health institutions can automate repetitive HR tasks, centralize personnel information, reduce human error, and ensure that legal and administrative procedures are consistently applied. Information systems are no longer simple support tools; they have become catalysts for organizational improvement. They streamline workflows, strengthen internal communication, and free administrative staff from a large part of the manual workload. Thanks to these advances, HR managers can focus more on decision-making and less on paperwork, ultimately improving the overall performance and responsiveness of the institution.  
  
To better understand the real needs of EPSP administrative teams, we engaged in discussions with personnel managers, agents, and archive service staff. These exchanges allowed us to observe the difficulties they face daily: scattered information, difficulty retrieving documents, inconsistent career tracking, and the absence of a unified system to manage promotions, assignments, and retirement procedures. Although some HR software exists in the healthcare field, none fully matches the specific operational expectations of EPSP structures or covers the complete employee lifecycle in a clear and reliable manner.  
  
In response to these real-world challenges, we designed a desktop application dedicated to HR management within EPSP institutions. Our solution aims to digitalize and centralize personnel records, automate the generation of essential administrative documents, and ensure compliance with career-management rules. By simplifying everyday tasks and reducing administrative bottlenecks, the application provides a practical, efficient, and tailored tool that supports managers in maintaining accurate records, improving coordination between services, and enhancing the quality of administrative workflows.

**Structure of the Report**

* **Chapter 1: Specification**  
  Where we answer the question "What must the system do?"
* **Chapter 2: Modeling**  
  Where we answer the question "How will the system work?"
* **Chapter 3: Implementation**  
  Where we answer the question "How did we build it?"

**Chapter 1: Specification**

**1.1.1 Introduction**

In this chapter, we introduce the context in which the Human Resources department of an EPSP operates and outline the main challenges it faces. We highlight the limitations of the current manual processes, which often lead to delays, inconsistencies, and increased administrative workload. Finally, we present the proposed solution: a dedicated Human Resources Management Application, and we summarize the specifications established in accordance with the IEEE 830 standard, which will guide the system's modeling and development.

**1.1.2 Context of the HR Department in EPSP**

The Human Resources department of an EPSP is responsible for managing personnel files, assignments, promotions, and administrative documents. It ensures that employee information is updated, classified, and accessible according to regulatory requirements. The department also oversees retirements and maintains archives for administrative follow-up. Given the number of staff and the variety of operations involved, HR activities require accuracy, coordination, and reliable information handling.

**1.1.3 Problems and Limitations of the Current System**

The department currently relies on manual and paper-based processes, which present several issues:

* Delays in updating and retrieving personnel information
* Frequent inconsistencies due to manual data entry
* Difficult tracking of assignments and promotion eligibility
* Time-consuming certificate generation
* Lack of traceability for modifications
* Increased administrative workload

These limitations reduce efficiency and highlight the need for a centralized, automated HR management system.

**1.1.4 Proposed Solution**

To address these limitations, we propose the development of a dedicated Human Resources Management Application for the EPSP. The system will centralize personnel information, automate assignments and promotions, and streamline certificate generation. It will also ensure data consistency, improve traceability, and reduce administrative workload through secure digital processes. This solution aims to modernize HR operations and provide a reliable foundation for efficient decision-making.

**1.2.1 System Engineering Logic (SEL) – IEEE 830 Specifications**

This section establishes the functional and non-functional requirements of the Human Resources Management Application. It defines the actors involved, their interactions with the system, and the operational constraints to be respected. These specifications constitute the formal reference for the system's modeling, design, and implementation phases.

**1.2.2 Actors**

* **HR Director:** Validates retiree-related modifications and approves password reset requests.
* **Personnel Manager:** Performs CRUD operations on personnel records and manages employee information, and promotions.
* **Employee/Agent:** Views basic personal information and requests certificates or employee lists.
* **Archive Service Manager:** Manages retiree files and handles requests for certificates and archived documents.

**1.2.3 Functional Requirements**

**FR-01: Personnel Record Management (CRUD)**

**Description:**   
The system shall allow the Personnel Manager to create, read, update, and delete (request retirement) personnel records.

**Inputs:** Personnel data fields (identity, specialty, domain, grade, etc.).

**Outputs:** Confirmation of successful operation or error messages.

**FR-02: Assignment and Department Change Management**

**Description:**  
The system shall allow the Personnel Manager to assign employees to departments, modify their departmental assignment.

**Inputs:** Employee ID, new department name.

**Outputs:** Updated department employees and employee assignment.

**FR-03: Promotion Management**

**Description:**   
The system shall allow submission, validation, and processing of promotion requests.

**Inputs:** Promotion, personnel ID, next grade or rank.

**Outputs:** Updated personnel record.

**FR-04: Certificate Generation**

**Description:**   
The system shall generate work, employment, and retirement certificates.

**Inputs:** Personnel ID or search criteria.

**Outputs:** PDF or printable certificate.

**FR-05: Reporting and Extraction (Agent)**

**Description:**   
The system shall generate lists based on grade, domain, specialty, or employment status.

**Inputs:** Filter criteria.

**Outputs:** Displayed or exported list.

**FR-06: Body Management (CRUD + Employee Management)**

**Description:**  
The system shall allow authorized users to create, modify, and delete bodies. It shall also allow assigning employees to a body, assigning grades within that body, and transferring employees between bodies.

**Inputs:** Body data, employee ID, grade.

**Outputs:** Updated body structure or employee affiliation.

**FR-07: Grade–Body Assignment Management (Archive Manager)**

**Description:**  
The system shall allow the Archive Manager to add grades to a body and configure grade hierarchy.

**Inputs:** Grade information, body ID.

**Outputs:** Updated body/grade configuration.

**FR-08: Specialties and Domains Management (Archive Manager)**

**Description:**  
The system shall allow the Archive Manager to add, modify, and delete specialties and domains, and associate specialties with domains.

**Inputs:** Domain/specialty names and codes.

**Outputs:** Updated domain–specialty structure.

**FR-09: Retiree and Archive Management**

**Description:**  
The system shall allow the Archive Manager to archive personnel records, consult archived files, and generate certificates for retirees.

**Inputs:** Personnel ID, retirement date.

**Outputs:** Archived record or certificate.

**FR-10: User Access and Authentication**

**Description:**  
The system shall authenticate users and provide role-based access control according to their actor category.

**Inputs:** Username, password.

**Outputs:** Access granted or denied.

**FR-11: Password Reset Management (Director Approval)**

**Description:**  
The system shall allow HR actors to reset their password. A password reset shall only be completed after entering the PIN provided by the HR Director as approval.

**Inputs:** User account type, new password, director PIN.

**Outputs:** User account details updated, confirmation message.

**1.2.4 Non-Functional Requirements**

**NFR-01: Security**

**Description:**  
The system shall ensure secure access and data protection.

**Requirements:**

* NFR-01.1: All users shall authenticate before accessing the system.
* NFR-01.2: Access rights shall be enforced according to user roles (RBAC).
* NFR-01.3: Passwords shall be stored using a secure hashing method.
* NFR-01.4: The system shall be operated only on a trusted network, and sensitive operations shall require additional validation.

**NFR-02: Performance**

**Description:**  
The system shall provide fast response times during normal usage.

**Requirements:**

* NFR-02.1: Searches and filters shall return results within 3 seconds.
* NFR-02.2: Certificate generation shall not exceed 5 seconds.
* NFR-02.3: Interface screens shall load within 2 seconds.

**NFR-03: Maintainability**

**Description:**  
The system shall be easy to update, correct, and extend.

**Requirements:**

* NFR-03.1: The system shall use a modular architectural structure (UI, business logic)
* NFR-03.2: Code shall be documented and follow consistent naming and formatting conventions.
* NFR-03.3: Changes to one module shall not require changes to unrelated modules (low coupling).

**NFR-04: Usability**

**Description:**  
The interface shall be easy to use for administrative staff.

**Requirements:**

* NFR-04.1: All forms shall include clear validation and error messages.
* NFR-04.2: Navigation shall be simple and organized by user role.
* NFR-04.3: The system shall display clear error messages for invalid inputs or failed operations, indicating the issue and how to correct it.

**NFR-05: Reliability**

**Description:**  
The system shall operate consistently and preserve data integrity.

**Requirements:**

* NFR-05.1: The system shall prevent conflicting updates using transactional logic.
* NFR-05.2: The system shall restore the last consistent state after unexpected failure.

**NFR-06: Availability**

**Description:**  
The system shall remain accessible during HR operational hours.

**Requirements:**

* NFR-06.1: Availability during office hours shall be at least 99%.
* NFR-06.2: Maintenance shall be scheduled outside normal usage periods.

**NFR-07: Language Requirements for Body Management**

**Description:**  
The system shall support multilingual data entry for bodies, grades, domains, and specialties.

**Requirements:**

* NFR-07.1: Bodies shall include French and Arabic labels.
* NFR-07.2: Grades, domains, and specialties shall support bilingual names.
* NFR-07.3: The interface shall correctly display multilingual text for these elements.

**NFR-08: Scalability**

**Description:**  
The system shall support future growth in HR data and system features.

**Requirements:**

* NFR-08.1: The system shall handle increasing volumes of personnel, assignments, grades, bodies, domains, and specialties.
* NFR-08.2: The system shall support adding new features or modifications without conflicts or unexpected errors.

**1.3 Conclusion**

This chapter outlined the operational context of the HR department within an EPSP and examined the main limitations of the current manual processes. These constraints clearly demonstrate the necessity of implementing a dedicated Human Resources Management Application capable of centralizing information, reducing inconsistencies, and enhancing overall administrative efficiency. The functional and non-functional requirements specified in accordance with the IEEE 830 standard establish a solid and comprehensive foundation for the system's architecture. They also ensure that the objectives of the solution are well-defined and aligned with user needs. This groundwork will guide the detailed modeling and design activities presented in the next chapter.

**Chapter 2: Modeling**

**2.1 Introduction**

This chapter introduces the modeling approach used in the design of the Human Resources Management Application. Building on the specifications established in the previous chapter, we adopt the Rational Unified Process (RUP) to guide the system's analysis and design phases. RUP offers an iterative, use-case–driven methodology that ensures the system aligns with user needs while maintaining architectural clarity, modularity, and scalability.

In the sections that follow, we outline the fundamental principles of RUP, explain the rationale behind its selection, and develop the different models required for the application, including the use case model, analysis model, design model, implementation model, deployment model, and test model.

**2.2.1 Definition of the RUP**

The Rational Unified Process (RUP) is a structured software development methodology used in our project that emphasizes iterative development, effective risk management, and continuous validation of requirements. It organizes the project lifecycle into four distinct phases:

* **Inception:** Defines the project scope, objectives, initial risks, and key functional requirements.
* **Elaboration:** Establishes the system's architecture, refines critical functionalities, and addresses major technical risks.
* **Construction:** Implements, tests, and integrates the application components according to the established architecture.
* **Transition:** Deploys the system to end users, performs final optimizations, and resolves issues identified during acceptance testing.

RUP is fundamentally use-case driven, meaning that all development activities originate from analyzing how actors interact with the system. It is also architecture-centric, ensuring that a robust and scalable structure is established early in the process, guiding subsequent design and implementation efforts.

**2.2.2 Introduction to UML**

As part of RUP, system modeling is performed using the Unified Modeling Language (UML), a standardized notation used to represent both structural and behavioral aspects of the system. UML diagrams such as use case diagrams, class diagrams, sequence diagrams, and deployment diagrams provide clear and consistent models that support the analysis, design, implementation, and testing phases throughout the project. UML therefore serves as an essential tool for ensuring clarity, coherence, and traceability across all stages of system development.

**2.3 Why Use the RUP**

The Rational Unified Process was chosen for this project because it offers a structured and flexible approach that aligns well with the complexity of Human Resources management in an EPSP. The system involves multiple actors, various business rules, and a significant amount of interconnected data, which makes a clear and iterative methodology essential.

RUP's use-case driven approach ensures that the system is designed directly based on how users interact with it, allowing us to accurately capture functional needs and reduce misunderstandings. Its architecture-centric nature ensures that the core system structure is defined early, reducing technical risks and supporting future scalability.

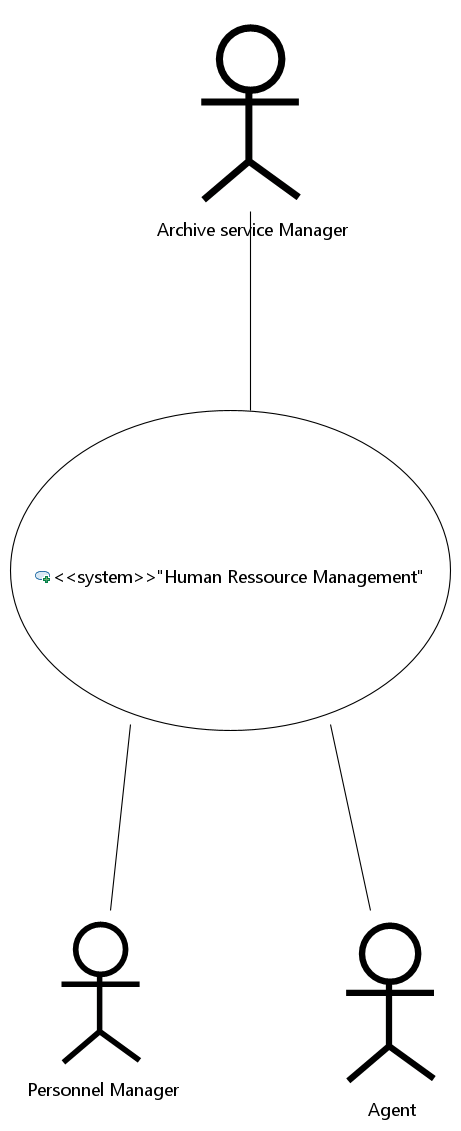
Additionally, the iterative development cycles of RUP allow progressive refinement of requirements and design models, making it easier to detect and correct issues early in the process. This is particularly important in HR systems, where reliability, accuracy, and data integrity are critical.

For these reasons, RUP provides an appropriate framework for modeling, designing, and ultimately developing a robust and maintainable Human Resources Management Application.

**2.4 Modeling**

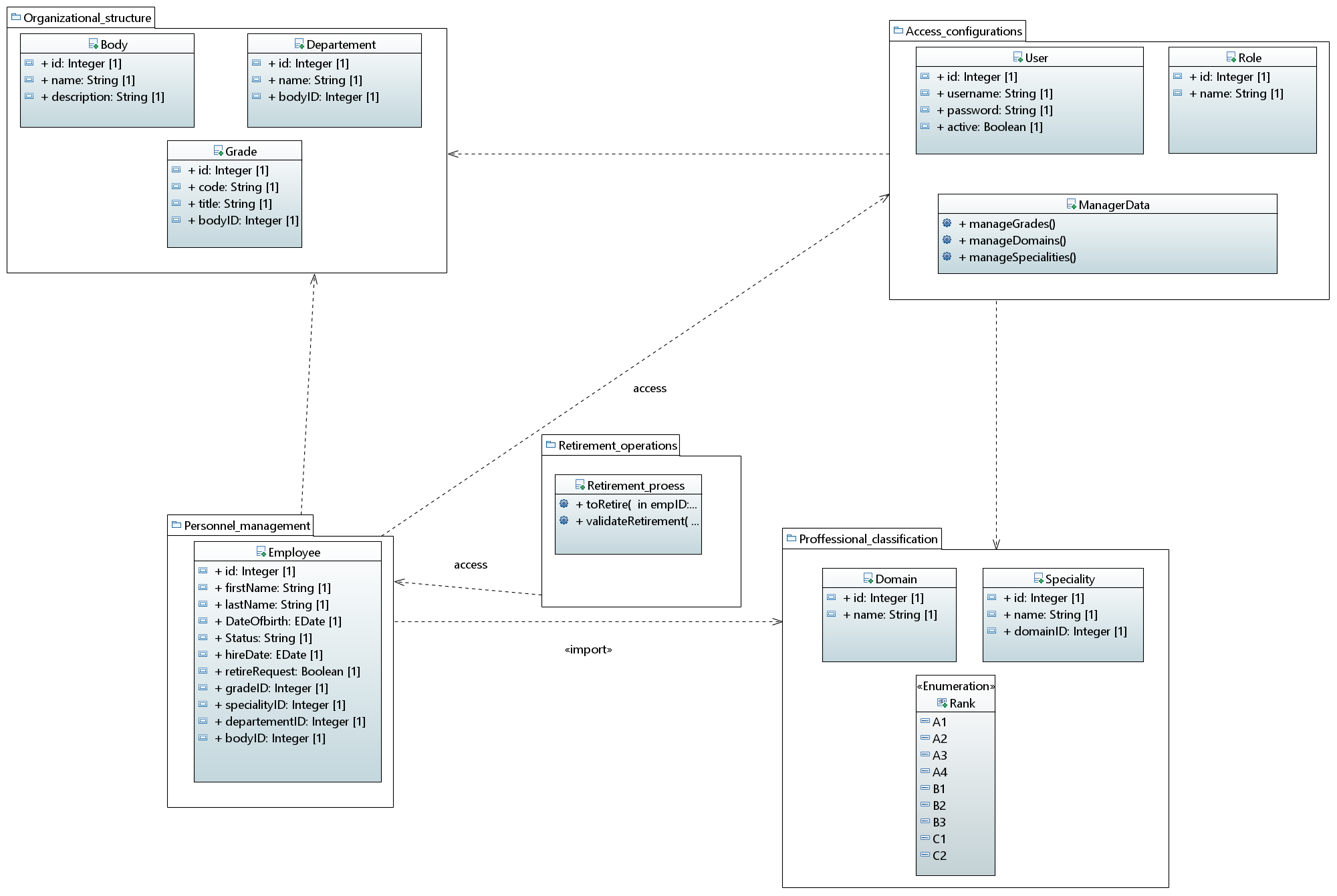
**2.4.1 Use Case Model**

**2.4.1.1 Static Context Diagram**

**** *Figure 01: Static Context Diagram*

**2.4.1.2 Package Diagram**

*Figure 02: Package Diagram*

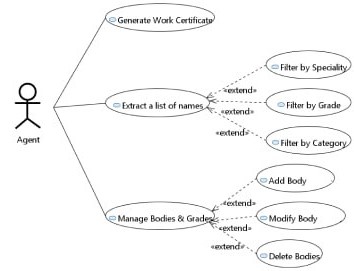


**2.4.1.3 Use Case Diagram**

**1) Personnel Manager  
**

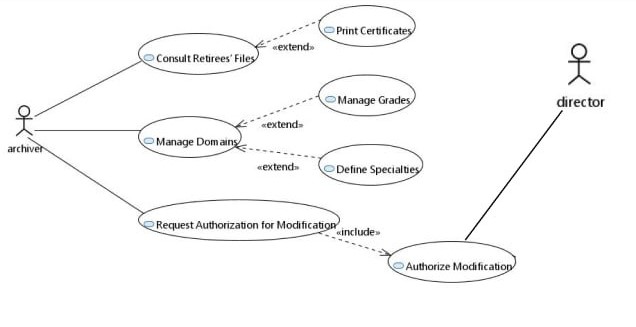
*Figure 03: Personnel Manager Use case*

1. **Agent**



*Figure 04: Agent Use case*

1. **Archive Service Manager**



*Figure 05: Archive Manager Service Use case*

**2.4.1.4 Text Description**

**2.4.1.4.1 Textual descriptions and sequence diagrams — Package: Personnel Manager**

### Use case 01: ****Add Employee****

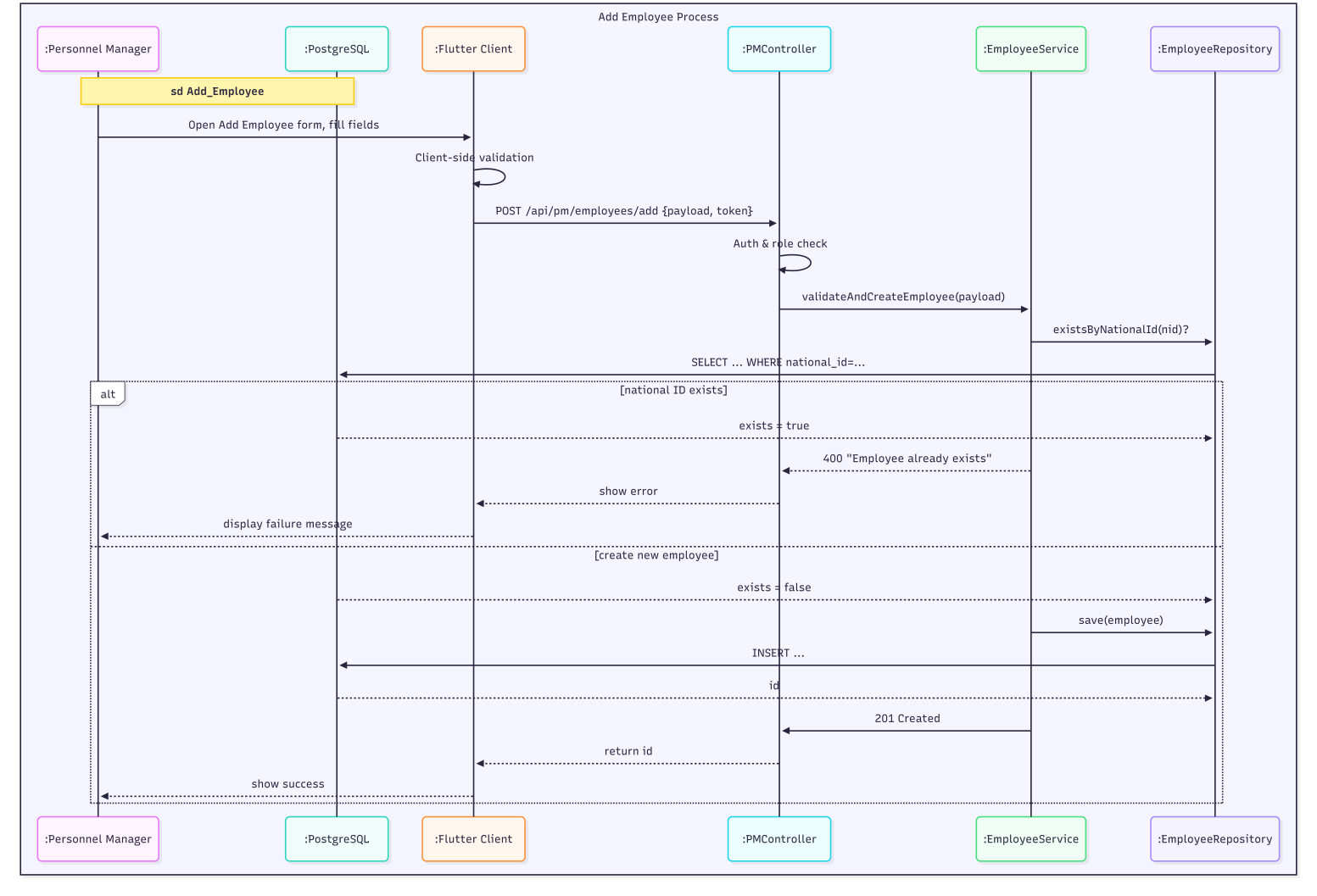
* **Objective:** Allow the Personnel Manager to create a new employee record in the system.
* **Primary actor:** Personnel Manager (PM)
* **Precondition:** PM is authenticated and has ROLE\_PM; required reference data (department, body, grade) exist.
* **Postcondition:** New employee record saved and visible in the employee list.

**Scenario nominal — Add Employee**

1. PM opens the Add Employee form.  
   1.1. The system displays the form with mandatory fields (first name, last name, national ID, department, body, grade, start date).
2. PM fills the form and clicks **Save**.  
   2.1. Client validates mandatory fields.
3. Client sends POST /api/pm/employees/add with payload and Bearer token.
4. Backend verifies authentication and role, validates payload.
5. EmployeeService checks business rules (unique national ID).
6. EmployeeRepository inserts the employee record.
7. System returns 201 Created and displays success message.
8. PM sees the new employee in the list.

**Scenarios alternatifs**

* **A1 — Mandatory field empty:** If a required field is empty, client shows “field required” and prevents submission. PM fills field and resumes at step 2.
* **A2 — Duplicate national ID:** If national ID already exists, backend returns 409 Conflict with message “employee exists”. PM may cancel or correct ID.
* **A3 — Server error:** If DB error occurs, backend returns 500. Client shows “operation failed” and logs error.

****

*Figure 06: Add Employee Sequence Diagram*

### Use case 02: ****Add Employee****

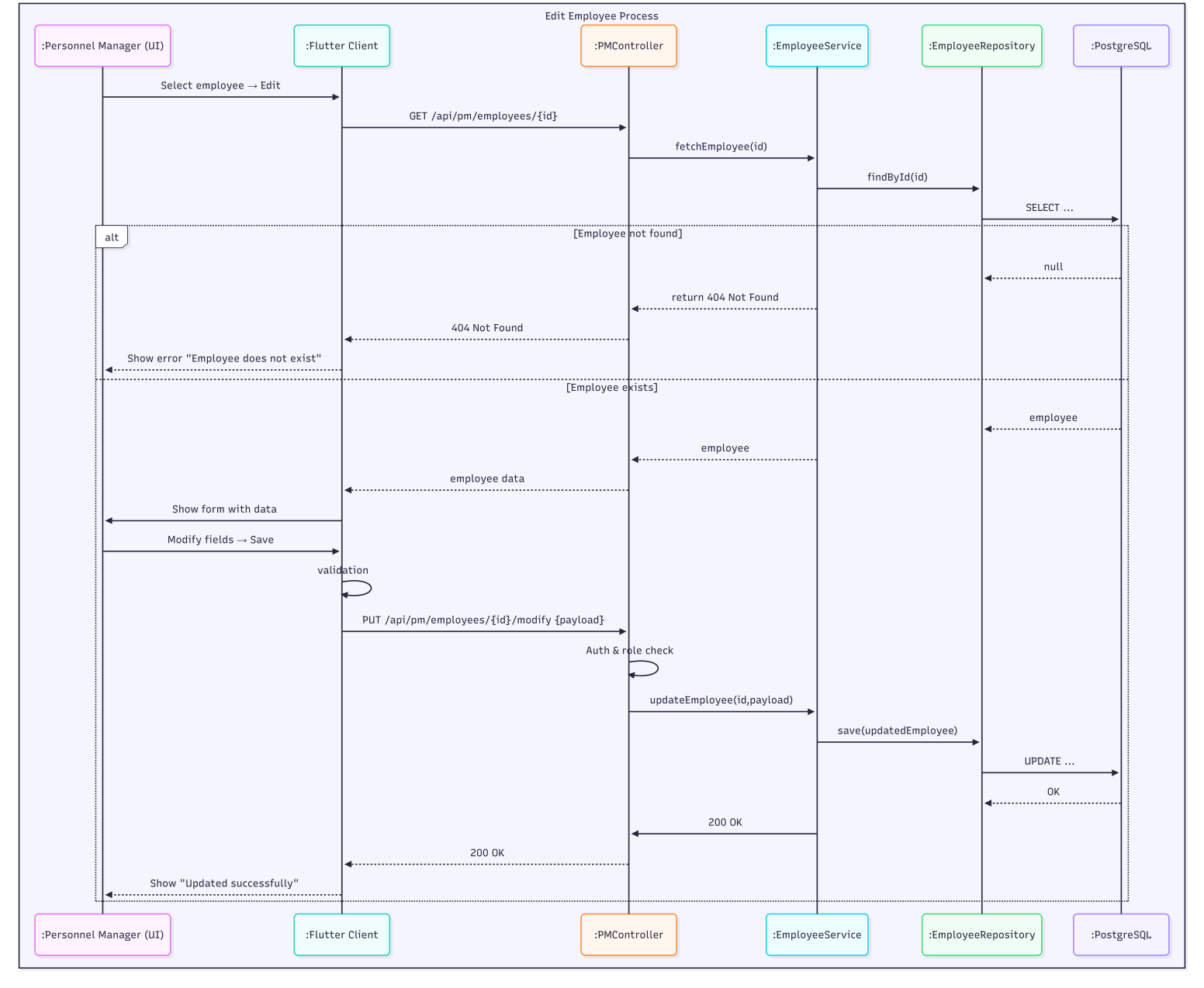
* **Objective:** Allow the Personnel Manager to update existing employee details.
* **Primary actor:** Personnel Manager (PM)
* **Precondition:** PM authenticated; employee exists and is not archived.
* **Postcondition:** Employee record updated with new values and audit logged.

**Scenario nominal — Modify Employee**

1. PM selects an employee from the list and clicks.  
   1.1. System loads and displays the employee form populated with current data.
2. PM changes fields and clicks **Save**.  
   2.1. Client validates fields.
3. Client sends PUT /api/pm/employees/{id}/modify with payload.
4. Backend authenticates and validates payload; Service checks business rules.
5. Service updates the entity via repository.
6. System logs audit and returns 200 OK.
7. PM sees confirmation and updated record.

**Scenarios alternatifs**

* **A1 — Concurrent update conflict:** If record changed concurrently, backend returns 409 Conflict; PM can refresh and reapply changes.
* **A2 — Invalid data:** Backend returns 400 Bad Request; client displays validation errors.
* **A3 — Cancel edit:** PM cancels -> system discards changes and returns to list.



*Figure 07: Modify Employee Sequence Diagram*

### Use case 03: ****Request Retirement****

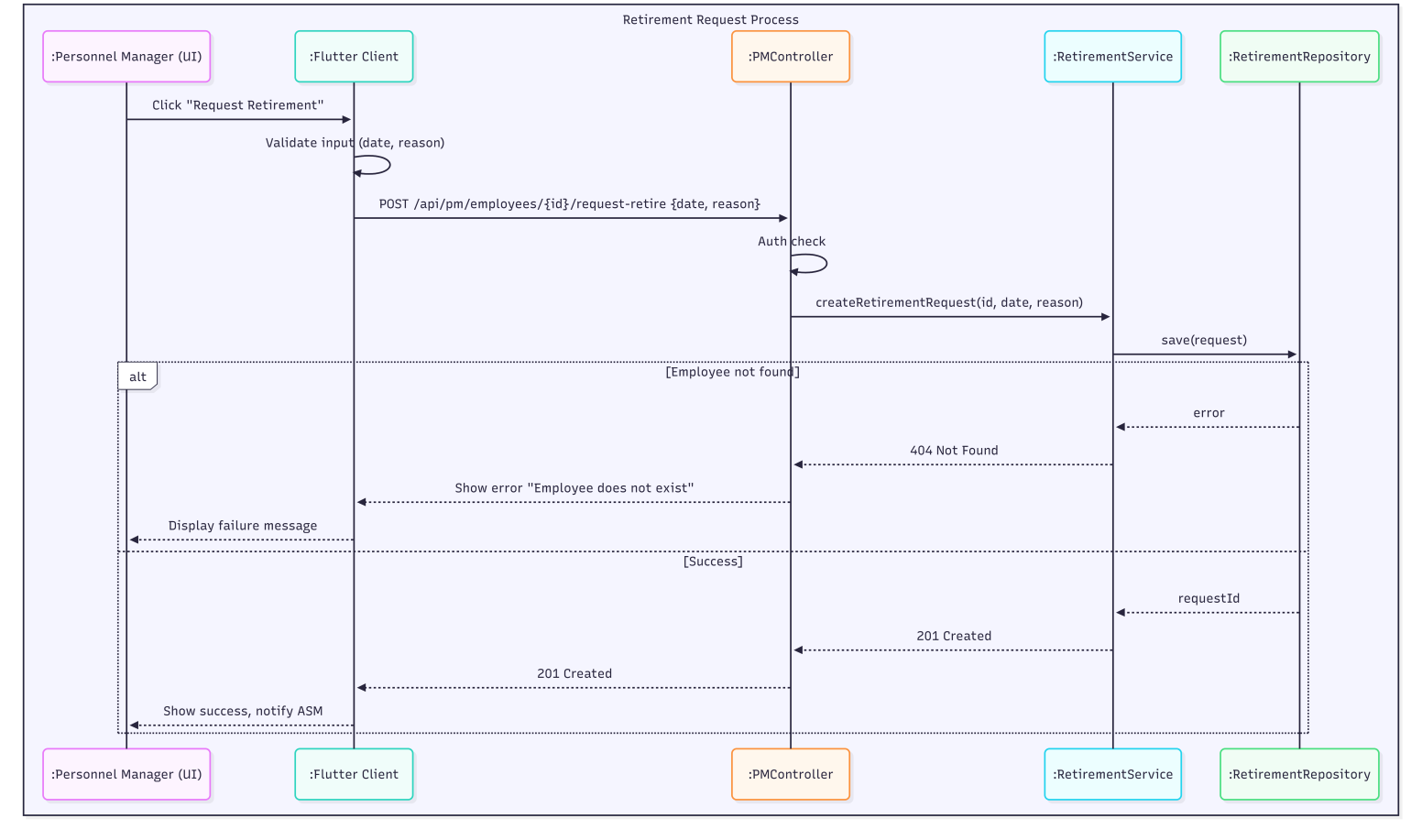
* **Objective:** Allow PM to create a retirement request for an employee (moves to retireRequests).
* **Primary actor:** Personnel Manager (PM)
* **Precondition:** PM authenticated; employee eligible or flagged for retirement.
* **Postcondition:** Retirement request created and visible to Archive Manager.

**Scenario nominal — Request Retirement**

1. PM selects employee and clicks **Request Retirement**.
2. System asks for retirement date and justification.
3. PM submits form.
4. Client sends POST /api/pm/employees/{id}/request-retire (or similar).
5. Backend authenticates and stores retirement request.
6. System returns 201 Created and notifies Archive Manager queue.

**Scenarios alternatifs**

* **A1 — Missing date/justification:** Client shows error; PM corrects.
* **A2 — Not eligible:** Service returns 400 with message; PM cannot create request.
* **A3 — Cancel:** PM cancels -> no request created.



*Figure 08: Request Retirement Sequence Diagram*

### Use case 04: ****Transfer Between Departments****

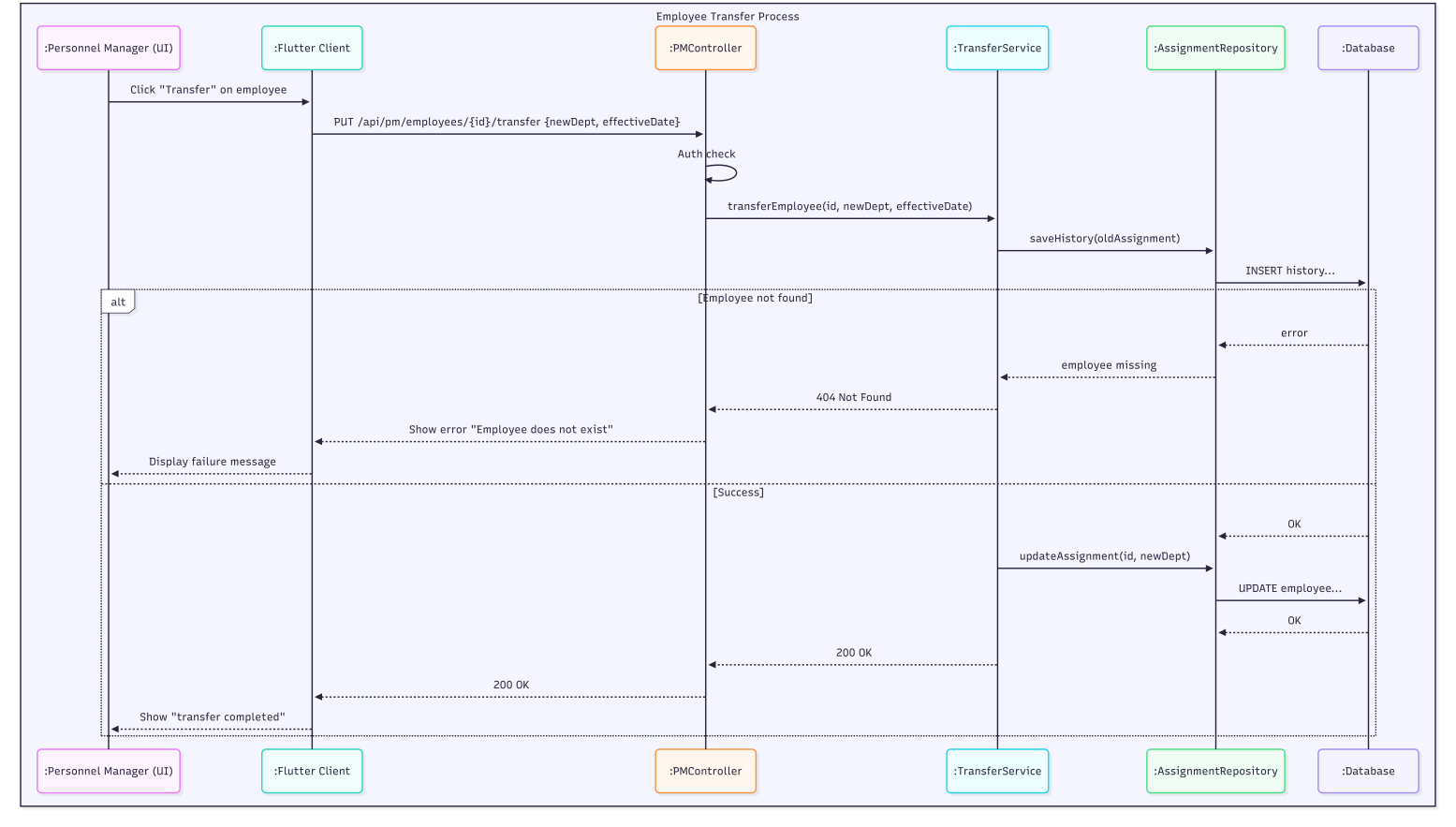
* **Objective:** Transfer an employee from one department to another with history tracking.
* **Primary actor:** Personnel Manager (PM)
* **Precondition:** PM authenticated; employee exists; target department available.
* **Postcondition:** Employee department updated; previous assignment stored in history.

**Scenario nominal — Transfer**

1. PM selects employee and clicks **Transfer**.
2. System shows current assignment and list of departments.
3. PM selects new department and effective date.
4. PM confirms transfer.
5. Client calls PUT /api/pm/employees/{id}/transfer.
6. Backend validates, updates employee record, inserts history entry, returns 200 OK.

**Scenarios alternatifs**

* **A1 — Conflict with active tasks:** System warns, PM may confirm with override or schedule transfer.
* **A2 — Invalid date:** System rejects and asks for valid effective date.



*Figure 09: Department Transfer Sequence Diagram*

### Use case 05: ****Assign New Recruit****

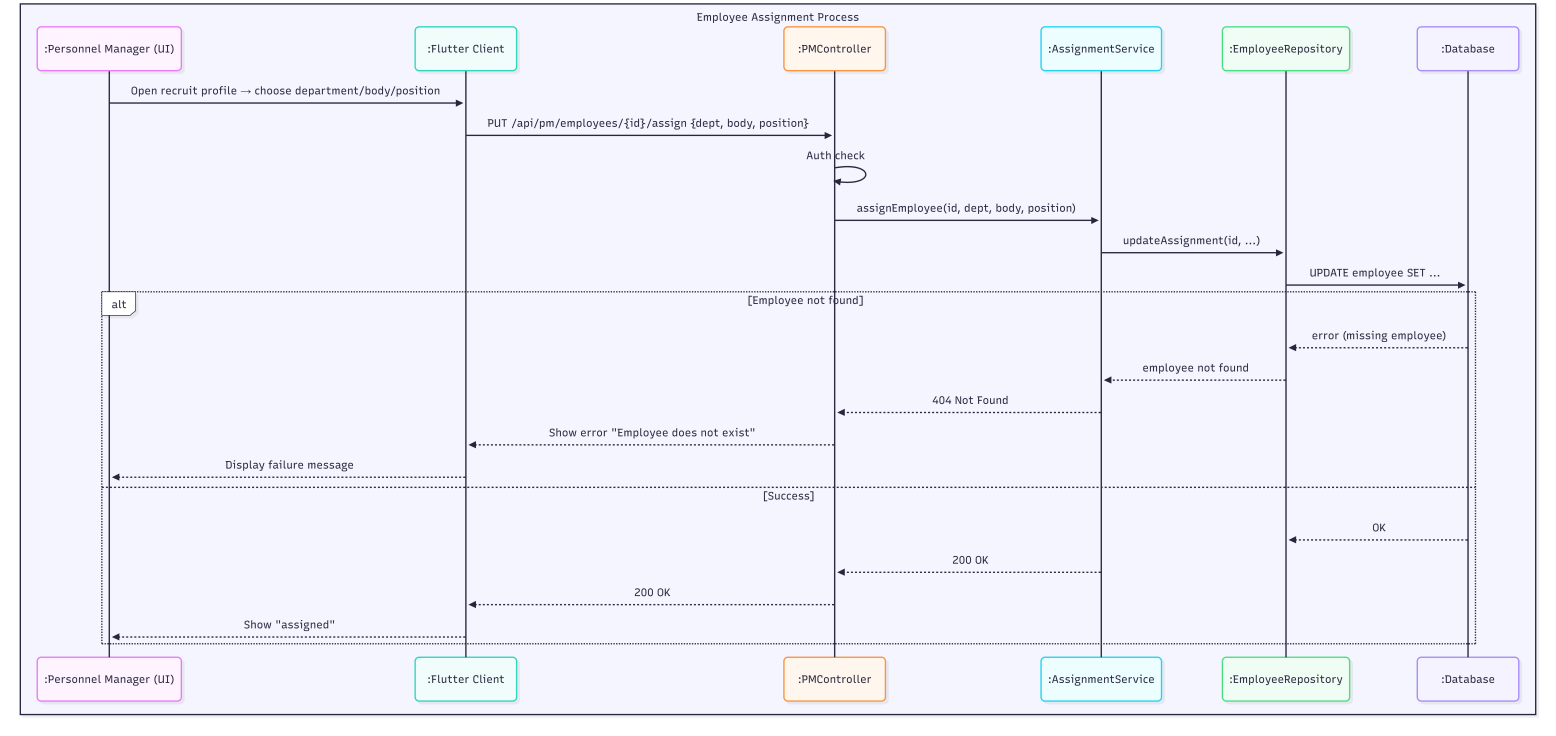
* **Objective:** Assign a newly recruited employee to a department/body and initial role.
* **Primary actor:** Personnel Manager (PM)
* **Precondition:** PM authenticated; recruit record exists or is being created.
* **Postcondition:** Employee assigned to department/body and record updated.

**Scenario nominal — Assign New Recruit**

1. PM opens a recruit profile (new or existing).
2. PM selects department, body, job position, and start date.
3. PM clicks **Assign**.
4. Client validates and sends PUT /api/pm/employees/{id}/assign.
5. Backend updates employee assignment and logs action.
6. System returns 200 OK and updates list.

**Scenarios alternatifs**

* **A1 — Department invalid:** System returns 400; PM selects valid department.
* **A2 — Employee already assigned:** System shows current assignment and asks to confirm transfer instead.



*Figure 10: Assign New Recruit Sequence Diagram*

### Use case 05: ****Promote Personnel****

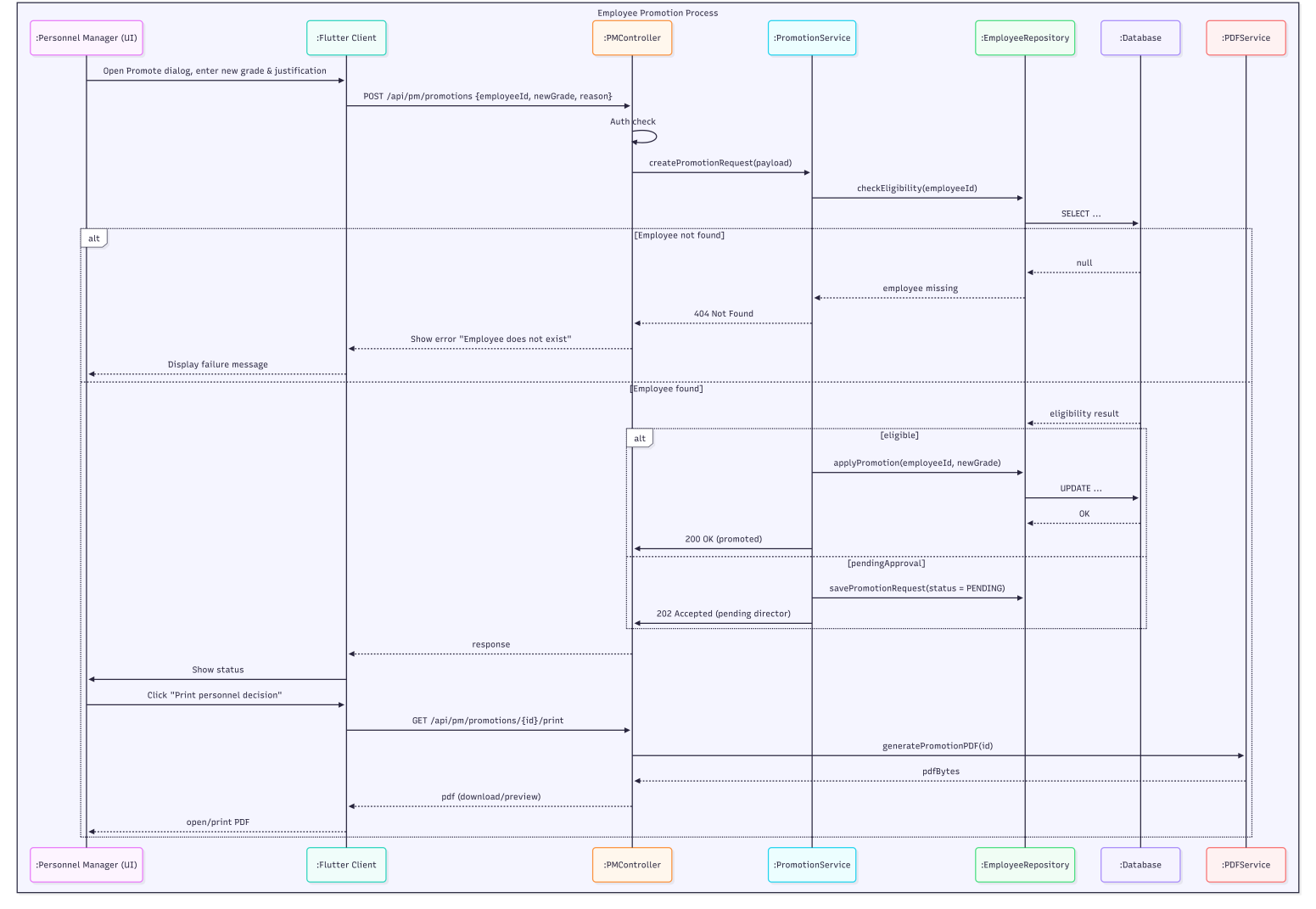
* **Objective:** Submit and process a promotion request, optionally printing the personnel decision.
* **Primary actor:** Personnel Manager (PM)
* **Precondition:** PM authenticated; employee meets promotion eligibility rules or request is supported by justification.
* **Postcondition:** Promotion recorded if validated; decision printable.

**Scenario nominal — Promote Personnel**

1. PM selects employee and opens **Promotion** dialog.
2. PM enters requested new grade/step and justification, then submits.
3. Client validates and sends POST /api/pm/promotions (or similar).
4. Backend validates eligibility (seniority, grade rules).
5. If valid, Service creates promotion request/changes status; system may require director validation (outside PM scope).
6. On approval, Service updates employee grade and logs decision.
7. PM can click **Print personnel decision** (included use case) — client requests PDF generation and prints.

**Scenarios alternatifs**

* **A1 — Not eligible:** Backend returns 400 with reason; PM may cancel or attach more documentation.
* **A2 — Requires Director authorization:** Request created with status PENDING\_DIRECTOR\_APPROVAL.
* **A3 — Cancel before submit:** PM cancels -> no change.



*Figure 11: Promote Personnel Sequence Diagram*

**2.4.1.4.2 Textual descriptions and sequence diagrams — Package: Agent**

### Use case 01: ****Generate Work Certificate****

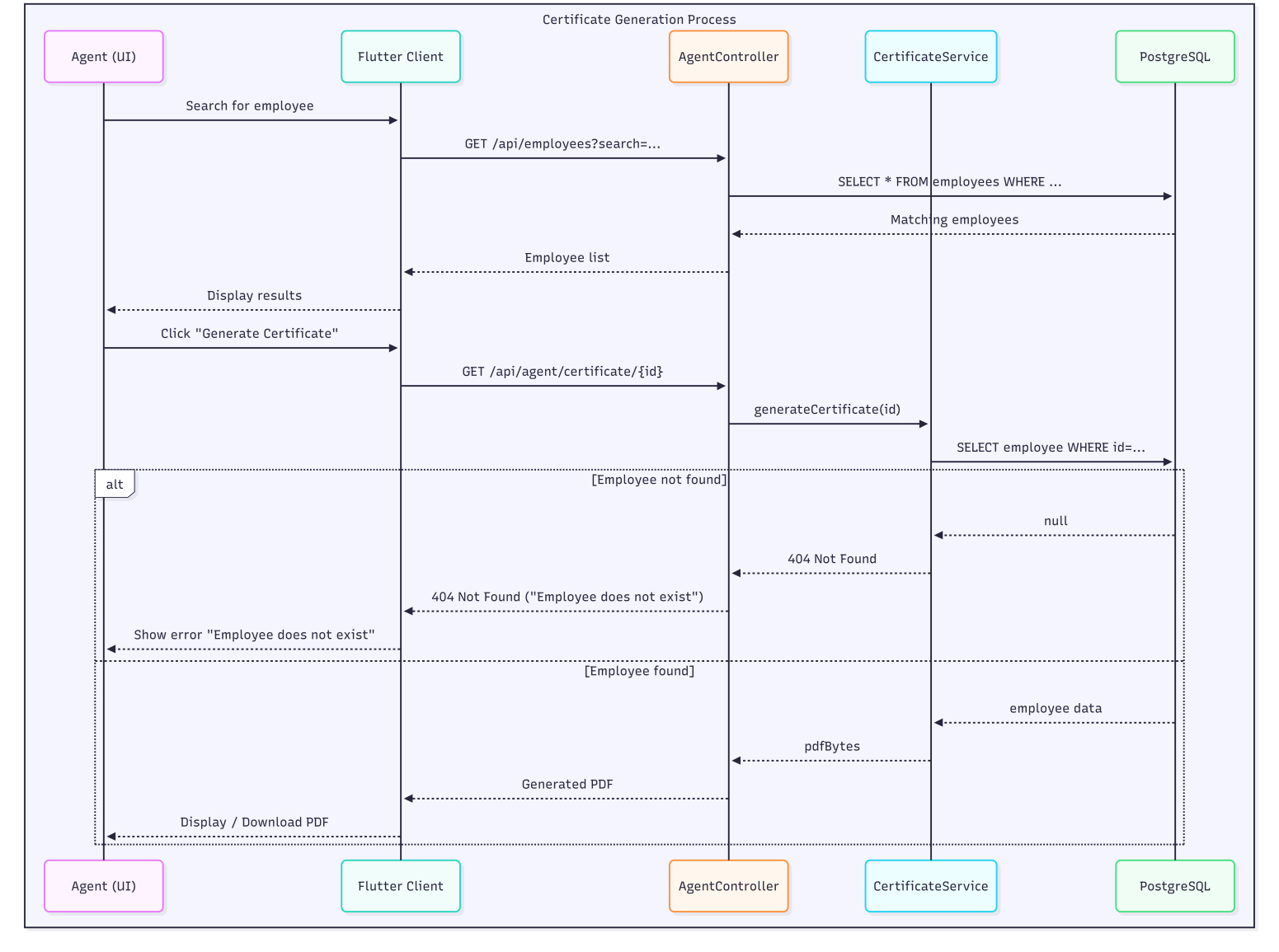
* **Objective:** Allow the Agent to instantly generate an employee’s work certificate in PDF format.
* **Primary Actor:** Agent
* **Precondition:** The Agent is authenticated; the selected employee exists in the database.
* **Postcondition:** A PDF work certificate is generated and ready for download or printing.

**Scenario nominal — Generate Work Certificate**

1. The system displays all employee list.
2. The Agent enters the employee’s name or ID if he has specific employee.
3. The system shows matching results.
4. The Agent selects an employee from the list.
5. The Agent clicks **Generate Certificate.**
6. The system builds the certificate template.
7. The system generates the PDF document.
8. The system displays a button to download or print the certificate.

**Scénarios alternatifs**

• **A1 — No employee found:** At step 2.1, if no matches, system shows “No matching employee”; Agent revises search (return to step 2).  
• **A2 — Missing required data:** At step 4.1, if fields missing (e.g., employment dates), system shows “Insufficient data to generate certificate”; Agent requests data correction from PM (stop or return to step 1 after fix).  
• **A3 — Server error:** If backend fails (PDF generation error or DB error), backend returns 500; client shows “operation failed” and logs error.



*Figure 12: Generate Work Certificate Sequence Diagram*

### 

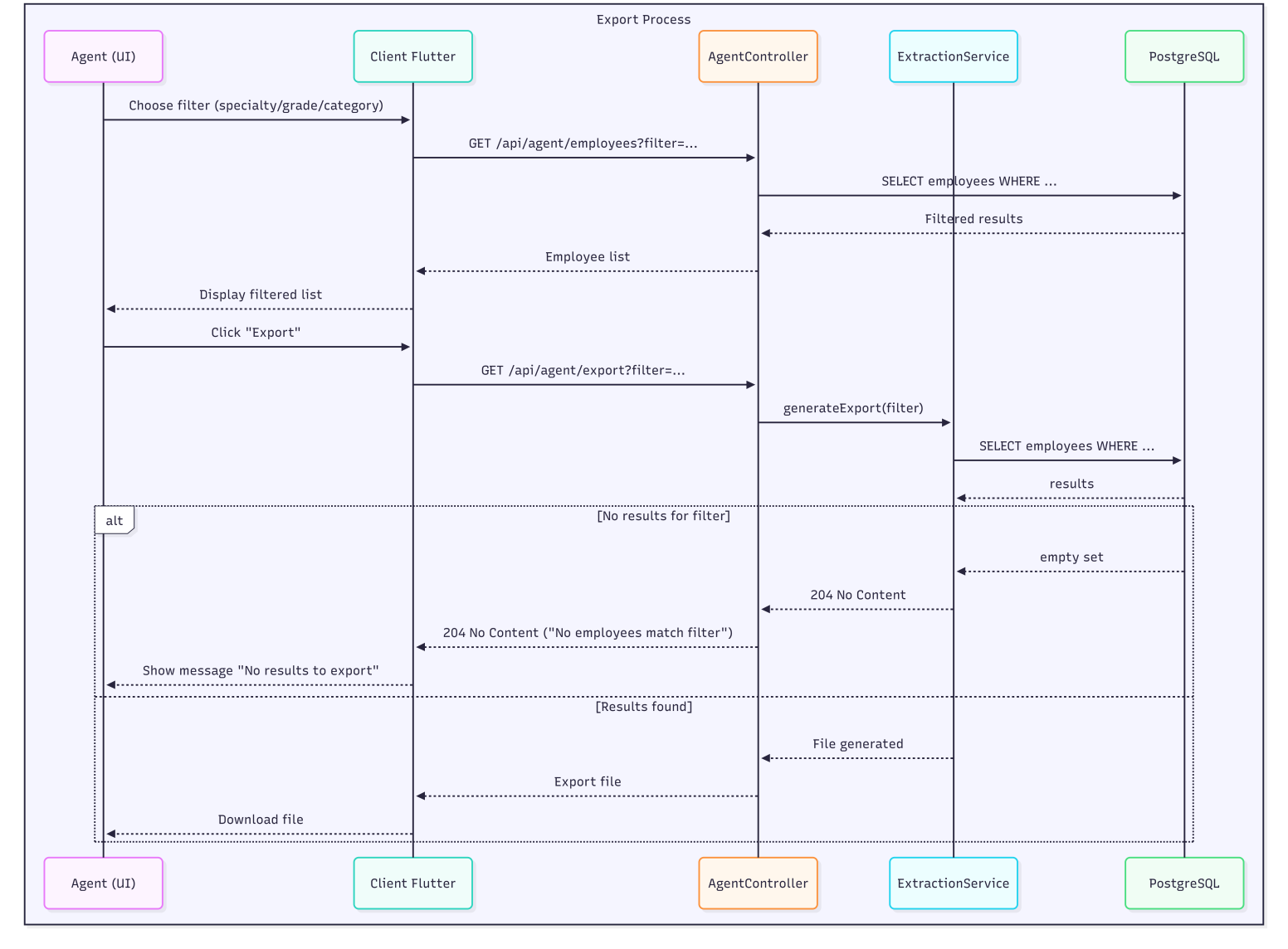
### Use case 02: Extract a List of Names (Filtered Export)

• **Objective:** Allow the Agent to extract and export a filtered list of employees (PDF/CSV) by specialty, grade, or category.  
• **Primary actor:** Agent  
• **Precondition:** Agent is authenticated; personnel data exist.  
• **Postcondition:** Filtered list displayed and export file (PDF/CSV) generated.

**Scenario nominal — Extract a List of Names**

1. Agent opens the List Extraction interface.  
   1.1. The system displays filter controls (specialty, grade, category, employment status).
2. Agent selects one or more filters and clicks **Search**.  
   2.1. Client validates filter values.
3. Client sends GET /api/agent/employees?filter=... with token.
4. Backend applies filters and returns the filtered employee list.
5. System displays the filtered list to the Agent.
6. Agent clicks **Export** and chooses format (PDF or CSV).  
   6.1. Client requests GET /api/agent/export?filter=... to backend.
7. Backend generates export file and returns file bytes.
8. Client prompts Agent to download the file.
9. Agent downloads the export and confirms completion.

**Scenarios alternatifs**  
• **A1 — No results found:** At step 4, if no employees match, system shows “No results found”; Agent modifies filters (return to step 2).  
• **A2 — Invalid filter value:** At step 2.1, client shows validation error (e.g., invalid date range); Agent corrects filters and retries.  
• **A3 — Export generation error:** At step 7, if export fails (timeout or server), backend returns 500; client shows “export failed” and logs the attempt.



*Figure 13: Extract a List Sequence Diagram*

### Use case 03 : Manage Bodies (CRUD)

• **Objective:** Allow the Agent to create, modify or delete bodies (corps). The Agent manages bodies only; grade CRUD is the responsibility of the Archive Manager.  
• **Primary actor:** Agent  
• **Precondition:** Agent is authenticated and authorized to manage bodies; required bilingual labels and body code fields are available.  
• **Postcondition:** Body is created, modified, or marked deleted/archived and visible in the bodies list.

**Scenario nominal — Add Body**

1. Agent opens Bodies Management.  
   1.1. System displays existing bodies and a control to add a new body.
2. Agent clicks **Add Body**.  
   2.1. System displays Add Body form (French/Arabic name, code, optional description).
3. Agent fills the form and clicks **Save**.  
   3.1. Client validates required fields (both language labels present).
4. Client sends POST /api/agent/bodies/create with payload and token.
5. Backend verifies auth/role and saves the new body.
6. System returns 201 Created and shows success message.
7. Agent sees the new body in the list.

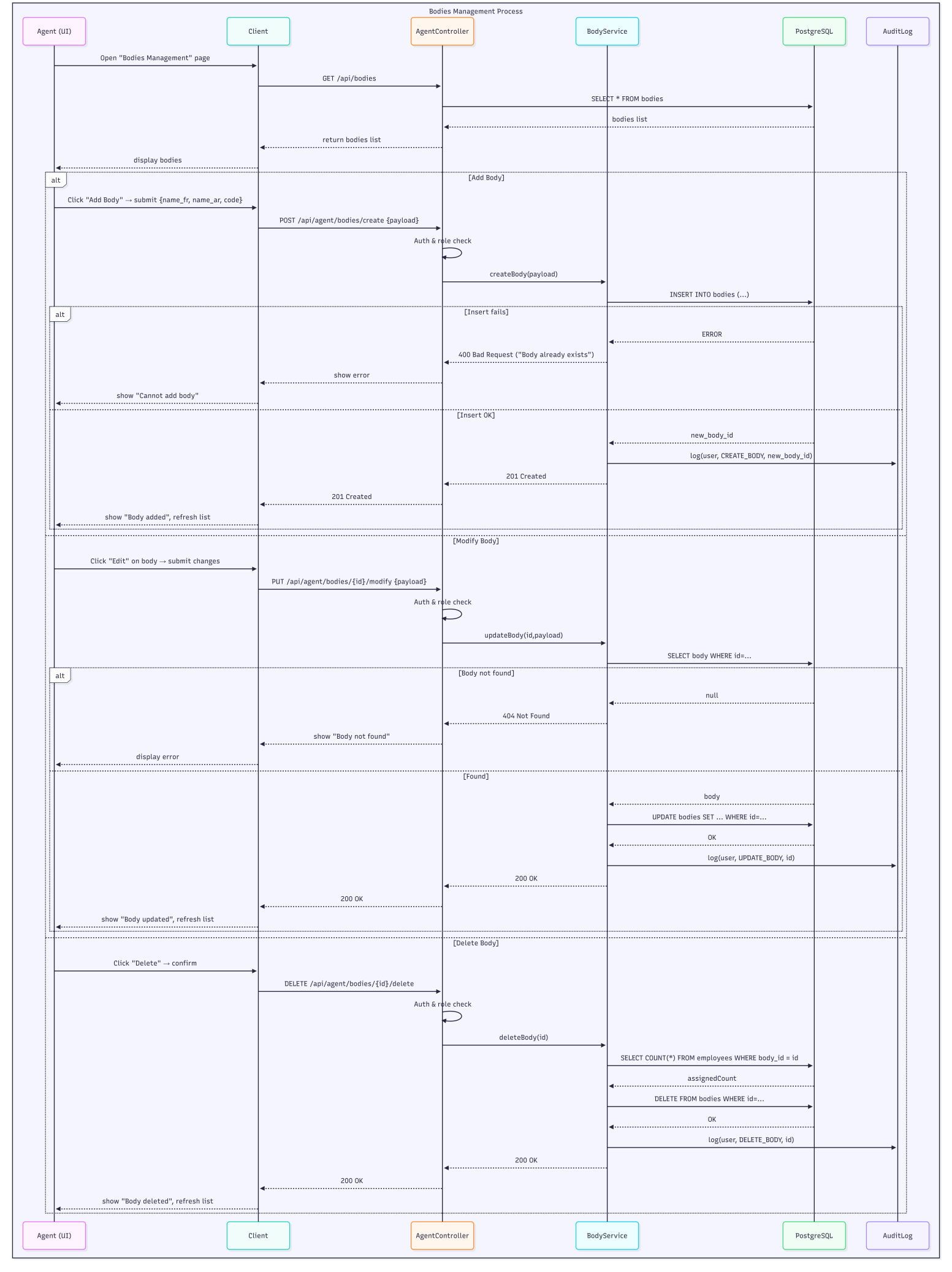
**Scenario nominal — Modify Body**

1. Agent selects a body and clicks **Edit**.  
   1.1. System displays the form populated with current data.
2. Agent updates fields and clicks **Save**.  
   2.1. Client validates inputs.
3. Client sends PUT /api/agent/bodies/{id}/modify with payload.
4. Backend verifies auth and updates the body record.
5. System returns 200 OK and UI refreshes with updated body.

**Scenario nominal — Delete Body**

1. Agent selects a body and clicks **Delete**.  
   1.1. System requests confirmation.
2. Agent confirms.
3. Client sends DELETE /api/agent/bodies/{id}/delete with token.
4. Backend checks referential constraints (no active employees assigned, or marks body as archived).
5. If allowed, backend deletes/archives the body and returns 200 OK.
6. System updates UI and shows success.

**Scenarios alternatifs**  
• **A1 — Missing bilingual labels:** At step 3.1, if either language label is missing, client shows “field required”; Agent fills and resumes.  
• **A2 — Duplicate body code/name:** Backend returns 409 Conflict with “body exists”; Agent changes code/name or cancels.  
• **A3 — Constraint on delete (body in use):** At step 4 (delete), backend returns 409 Conflict or 400 with message “body assigned to employees”; delete aborted; Agent may request reassignment of employees before retry.  
• **A4 — Server error:** Backend returns 500; client shows “operation failed” and logs error.



*Figure 14: Manage Body (CRUD) Sequence Diagram*

**2.4.1.4.3 Textual descriptions and sequence diagrams — Package: Agent**

### Use case 01: ****Manage Grades (CRUD)****

**Objective:** Allow the Archive Manager to create, modify and delete grades under a given body (corps).  
• **Primary actor:** Archive Manager (ASM)  
• **Precondition:** ASM is authenticated with ROLE\_ASM; the body exists.  
• **Postcondition:** Grade is created, updated, or removed; changes visible to the system and usable by Agents and PMs.

**Scenario nominal — Create Grade**

1. ASM navigates to Body → Grades for a chosen body.  
   1.1. System displays existing grades for that body.
2. ASM clicks **Add Grade**.  
   2.1. System shows the Add Grade form (grade code, bilingual name, level, description).
3. ASM fills the form and clicks **Save**.  
   3.1. Client validates mandatory fields.
4. Client sends POST /api/asm/bodies/{bodyId}/grades/create with payload and token.
5. Backend verifies auth/role and saves the new grade under the body.
6. System returns 201 Created and shows success.
7. ASM sees the new grade in the body’s grade list.

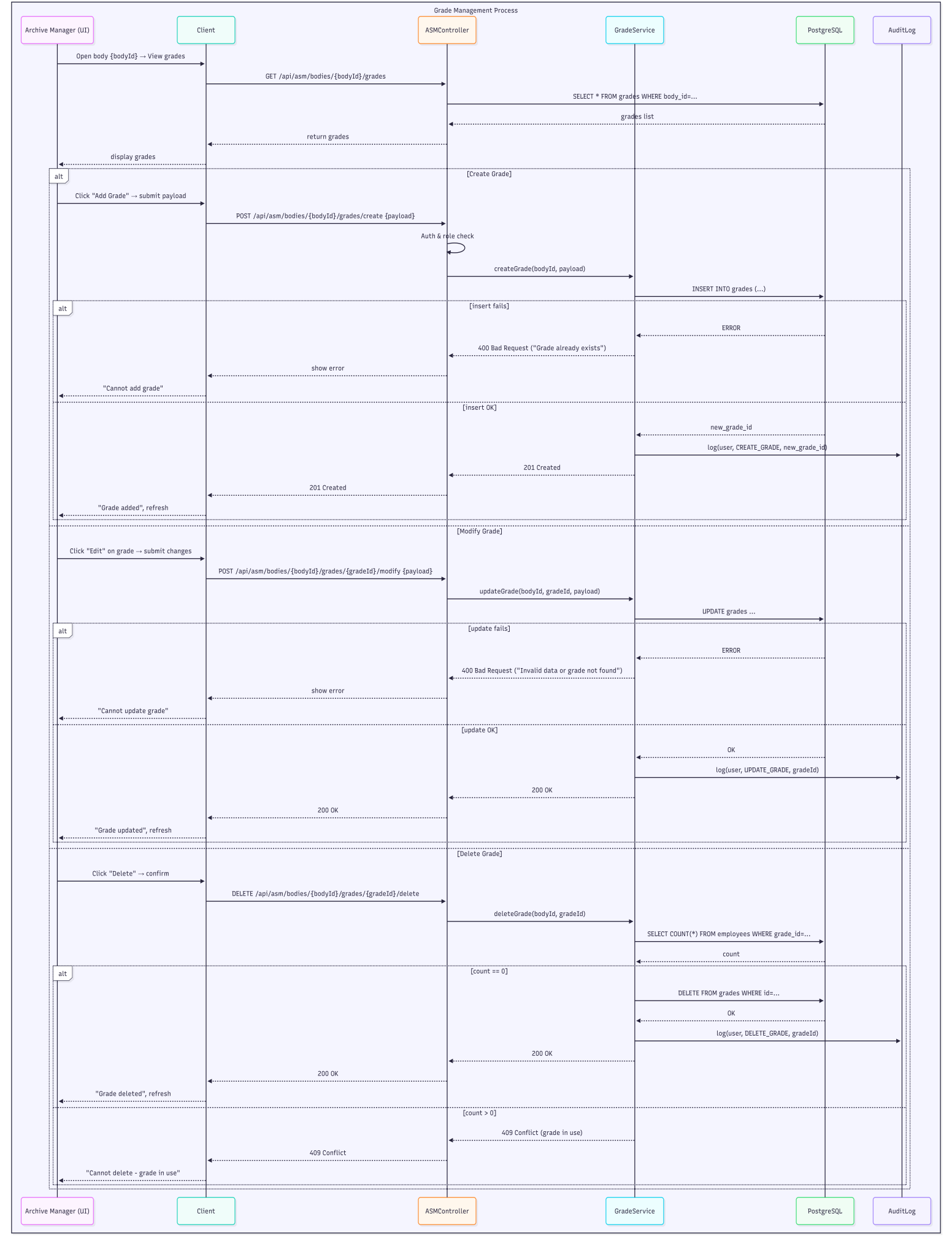
**Scenario nominal — Modify Grade**

1. ASM selects an existing grade and clicks **Edit**.  
   1.1. System displays grade form populated with data.
2. ASM modifies fields and clicks **Save**.  
   2.1. Client validates inputs.
3. Client sends POST /api/asm/bodies/{bodyId}/grades/{gradeId}/modify.
4. Backend updates grade and returns 200 OK.
5. ASM sees updated grade.

**Scenario nominal — Delete Grade**

1. ASM selects grade and clicks **Delete**.  
   1.1. System requests confirmation.
2. ASM confirms.
3. Client sends DELETE /api/asm/bodies/{bodyId}/grades/{gradeId}/delete.
4. Backend checks constraints (no active references preventing deletion).
5. If allowed, backend deletes or archives the grade and returns 200 OK.
6. UI refreshes; ASM sees updated list.

**Scenarios alternatifs**  
• **A1 — Missing required field:** client shows validation errors; ASM corrects.  
• **A2 — Duplicate grade code:** backend returns 409 Conflict; ASM changes code.  
• **A3 — Constraint on delete (in use):** backend returns 409 Conflict with message “grade in use”; ASM must reassign employees or archive instead.  
• **A4 — Server error:** backend returns 500; client shows “operation failed” and logs error.

*****Figure 15: Manage Grades (CRUD) Sequence Diagram*

### Use case 02: ****Manage Domains & Specialties (CRUD)****

• **Objective:** Allow ASM to create, edit and remove Domains and associated Specialties (medical specialties).  
• **Primary actor:** Archive Manager (ASM)  
• **Precondition:** ASM authenticated; domain codes and specialty codes available.  
• **Postcondition:** Domains and specialties are updated and reflected in the system for PM/Agents to use.

**Scenario nominal — Create Domain / Specialty**

1. ASM opens Domains & Specialties management.  
   1.1. System lists existing domains and specialties.
2. ASM clicks **Add Domain** or chooses a domain and clicks **Add Specialty.**
3. ASM fills required fields (code, bilingual labels, description) and clicks **Save**.
4. Client validates input and calls POST /api/asm/domains/create or POST /api/asm/domains/{domainId}/specialities/create.
5. Backend saves entity and returns 201 Created.
6. ASM sees new domain/specialty.

**Scrnario nominal — Modify / Delete Domain or Specialty**

1. ASM selects domain/specialty and clicks **Edit** or **Delete**.
2. For edit: modify fields -> PUT/POST to update endpoint -> backend returns 200 OK.
3. For delete: confirm -> DELETE /api/asm/domains/{domainId}/delete or DELETE /api/asm/specialities/{id}/delete -> backend checks constraints (specialty not used by employees) -> delete/archive or return 409 Conflict.
4. UI refreshes.

**Scenarios alternatifs**  
• **A1 — Missing fields**: client validation error.  
• **A2 — Duplicate code**: backend 409 Conflict.  
• **A3 — Delete blocked because in use**: backend 409 Conflict with explanation.  
• **A4 — Server error**: backend 500.

### C:\Users\local user\Downloads\Mermaid Chart - Create complex, visual diagrams with text.-2025-12-11-204241.png

*Figure 16: Manage Domains & Specialties (CRUD) Sequence Diagram*

### Use case 03: ****Validate Retirement Request****

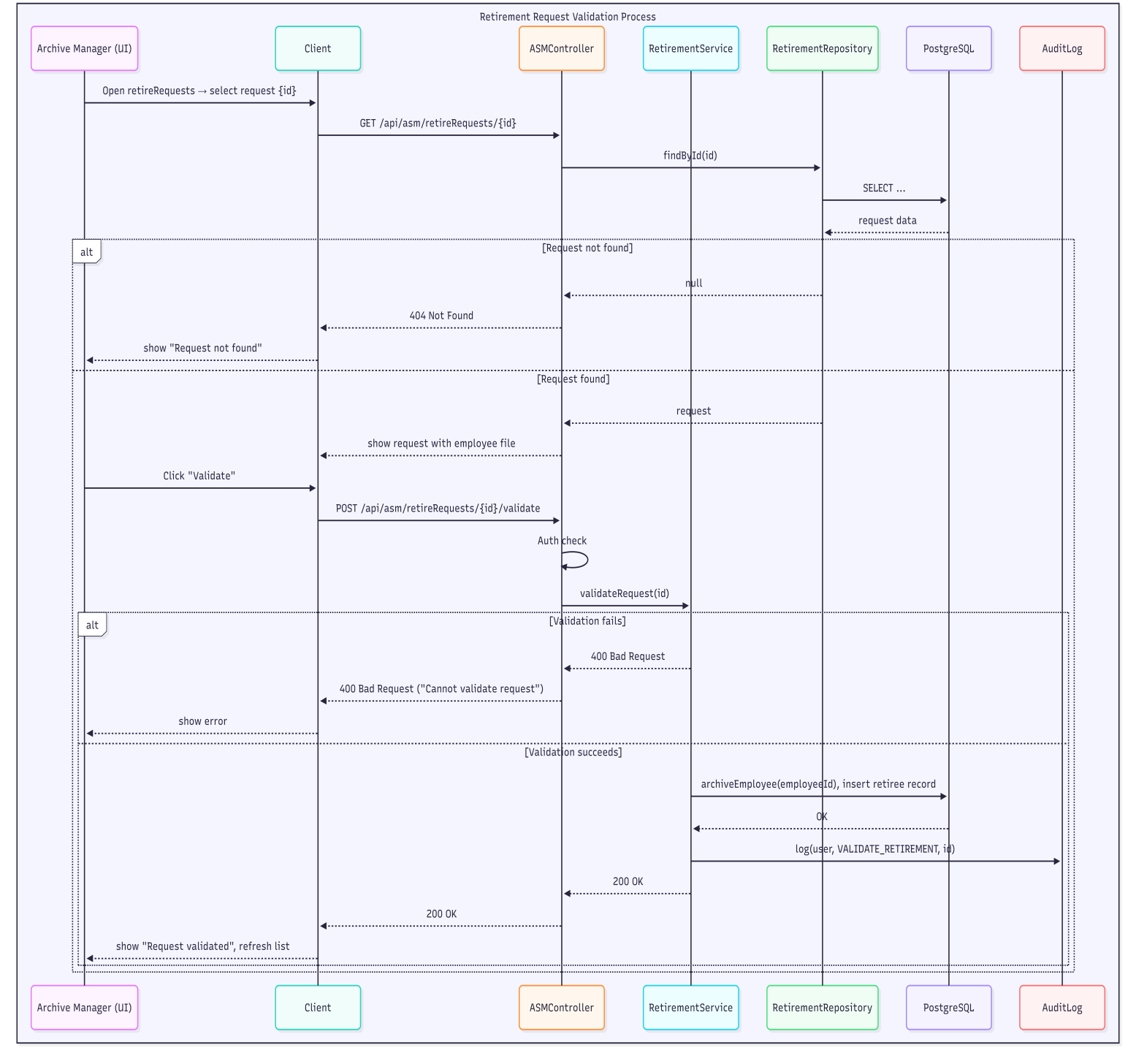
• **Objective:** Allow ASM to review and validate retirement requests submitted by the Personnel Manager; upon validation the employee record is archived and a retirement certificate can be generated.  
• **Primary actor:** Archive Manager (ASM)  
• **Precondition:** ASM authenticated; retirement request exists in the retireRequests queue.  
• **Postcondition:** Retirement request status set to VALIDATED (or REJECTED); if validated the employee is moved to archived state and archive record created.

**Scenario nominal — Validate Retirement Request**

1. ASM opens Retirement Requests list.  
   1.1. System displays pending requests with details (employee, requested date, justification).
2. ASM selects a request and views full employee file and verification checklist.
3. ASM clicks **Validate** (or **Reject**) after manual verification.  
   3.1. If **Validate**, system may request Director PIN for confirmation (depending on policy).
4. If Director PIN required: ASM inputs Director PIN -> backend verifies PIN.
5. Client sends POST /api/asm/retireRequests/{id}/validate with token and optional directorPin.
6. Backend verifies auth, performs checks, marks request VALIDATED, archives employee record, generates retiree record and optionally a retirement certificate.
7. System returns 200 OK and notifies PM and Agent about archive.
8. ASM sees validated request removed from pending list.

**Scenarios alternatifs**  
• **A1 — Missing evidence / inconsistent data:** ASM chooses **Reject** and records reason; system returns 200 OK with status REJECTED.  
• **A2 — Director PIN invalid:** If PIN is required but incorrect, backend returns 403 Forbidden; ASM may retry or escalate.  
• **A3 — Already archived / duplicate request:** backend returns 409 Conflict; ASM views archive entry.  
• **A4 — Server error:** backend 500.

### 

**** *Figure 17: Validate Retirement Request Sequence Diagram*

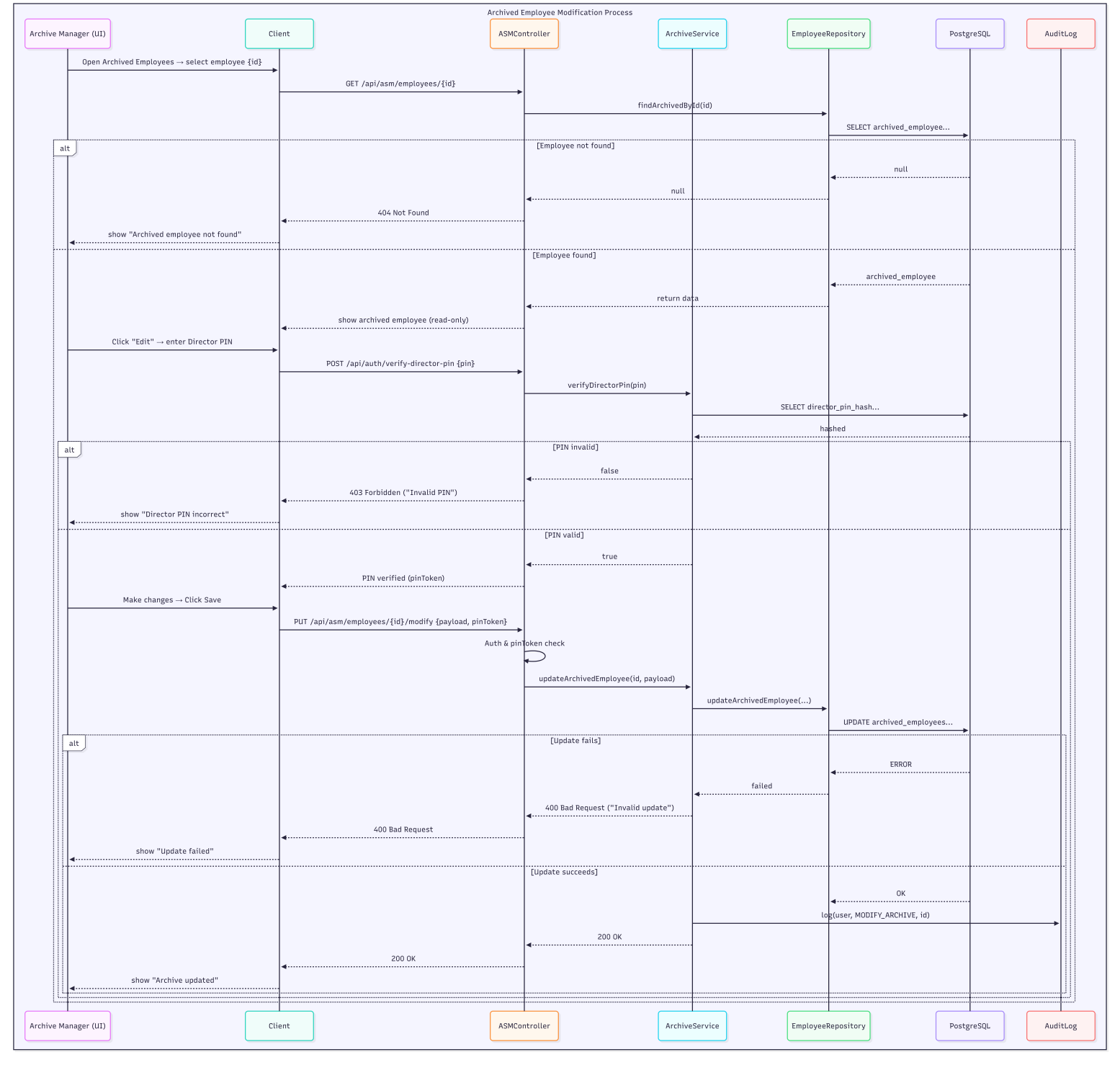
### Use case 04: ****Modify Archived Employee****

• **Objective:** Allow ASM to correct or update archived employee records (for example to fix missing fields or correct retirement date). This is a sensitive operation and requires Director PIN.  
• **Primary actor:** Archive Manager (ASM)  
• **Precondition:** ASM authenticated; employee is archived; Director PIN must be provided.  
• **Postcondition:** Archived employee record updated and audit trail preserved.

**Scenario nominal — Modify Archived Employee**

1. ASM opens Archived Employees list and selects a record.  
   1.1. System displays archived employee data (read-only until edit mode).
2. ASM clicks **Edit** and inputs Director PIN when prompted.  
   2.1. Client sends PIN to backend for verification (do not store PIN).
3. If PIN valid, client enables editable fields. ASM updates values and clicks **Save**.
4. Client sends PUT /api/asm/employees/{id}/modify with payload and (optionally) directorPin verification token.
5. Backend re-checks auth and PIN verification, applies updates to archive record, and appends an audit entry.
6. System returns 200 OK and shows confirmation.

**Scenarios alternatifs**  
• **A1 — PIN invalid:** Backend returns 403; edit disabled. ASM may retry with correct PIN or escalate.  
• **A2 — Concurrent archive lock:** If another admin is editing, backend returns 409 Conflict; ASM waits or retries.  
• **A3 — Forbidden field:** Some fields may be immutable; backend returns 400 for attempted changes to immutable fields.  
• **A4 — Server error:** backend 500.

*****Figure 18: Modify Archived Employee Sequence Diagram*

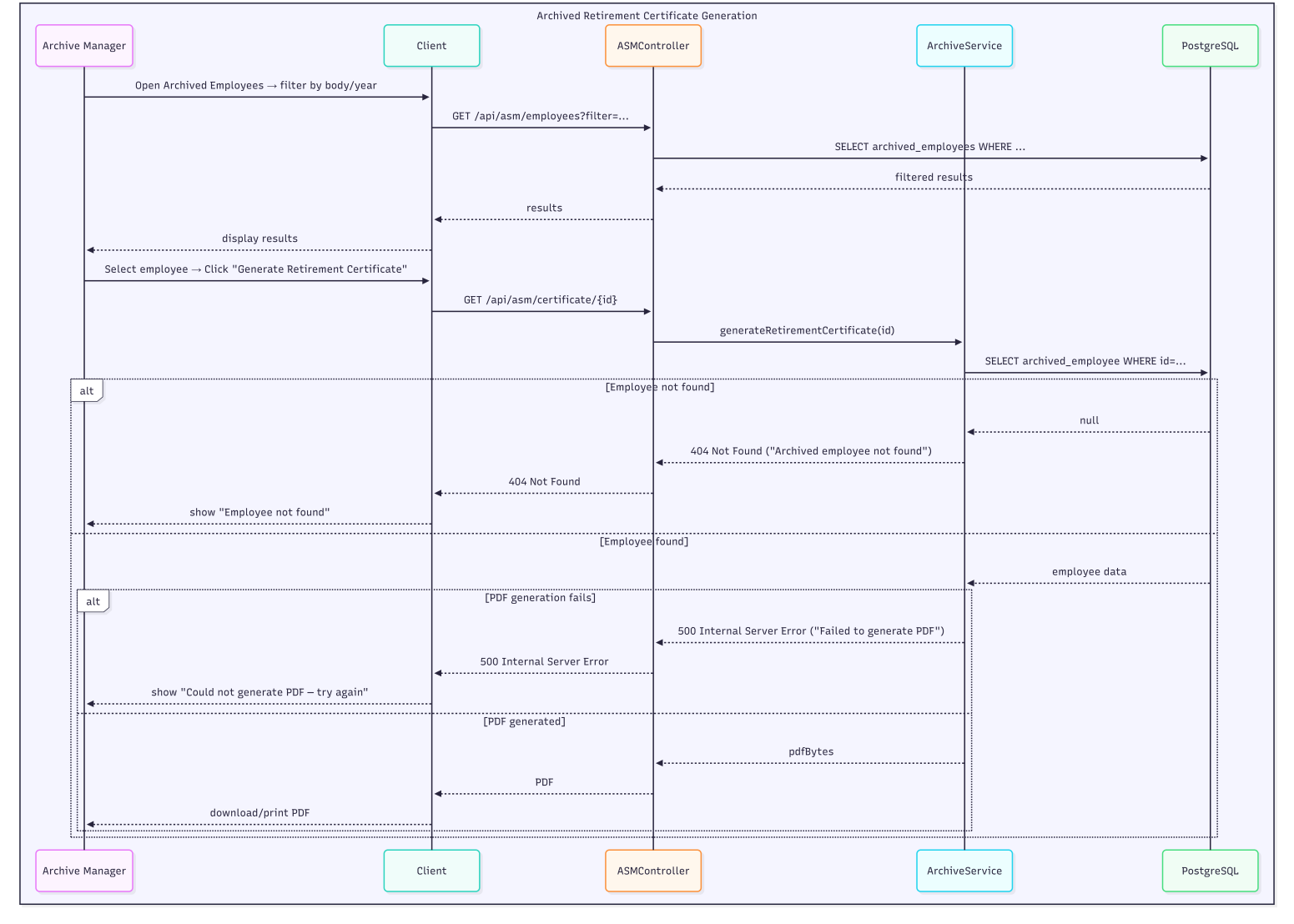
### Use case 05: Consult Retiree File & Generate Certificate

• **Objective:** Allow ASM to consult archived employee files and generate retiree certificates or reports.  
• **Primary actor:** Archive Manager (ASM)  
• **Precondition:** ASM authenticated; archive contains retiree records.  
• **Postcondition:** ASM views/archive entries and can generate retiree certificates or export lists.

**Scenario nominal — Consult Retiree File**

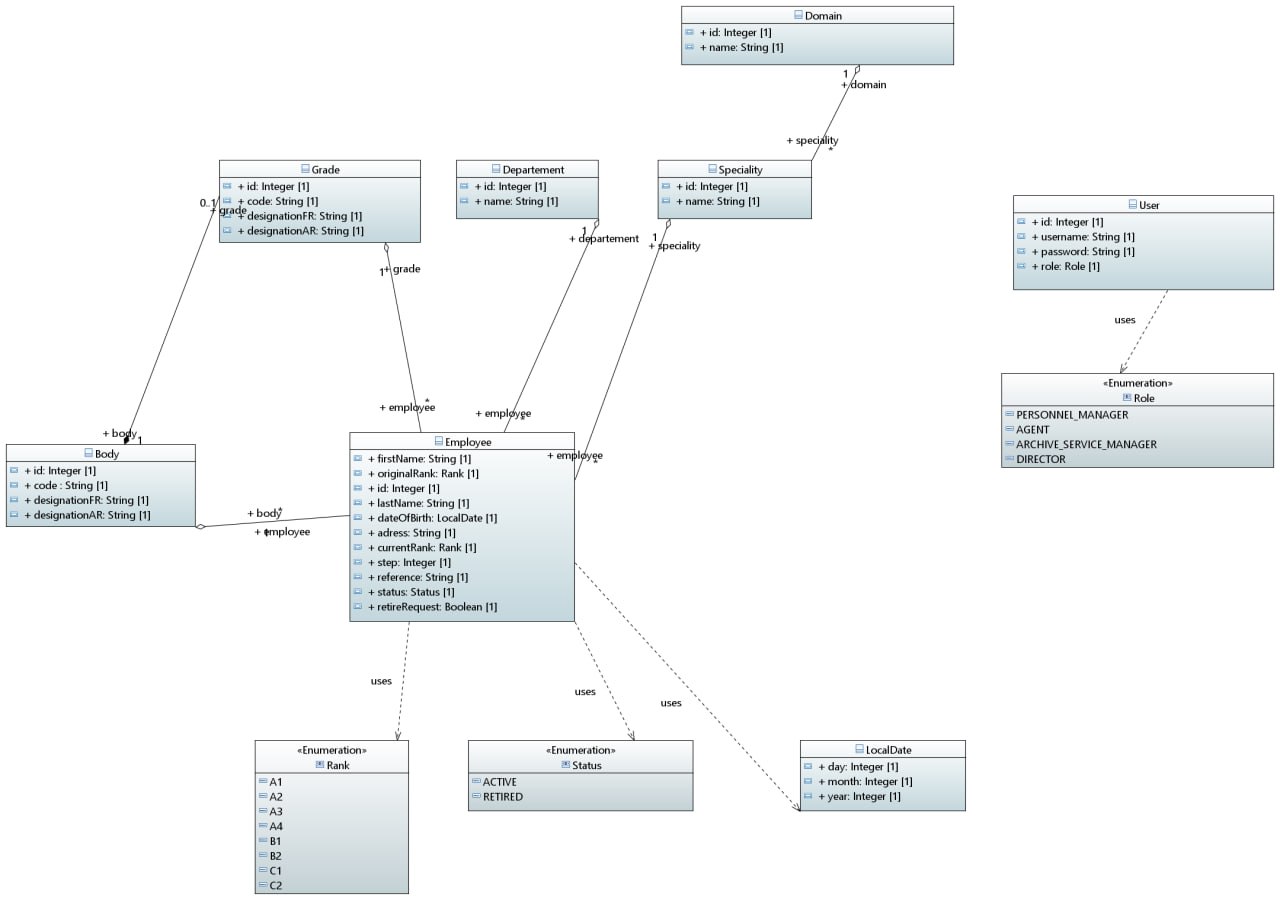
1. ASM opens Archived Employees or Retirees module.  
   1.1. System displays a searchable list of archived employees.
2. ASM searches or filters retiree list.
3. System returns filtered list.
4. ASM selects a record and the system displays full retired file and document options (Generate certificate).
5. ASM clicks **Generate Retirement Certificate** -> GET /api/asm/certificate/{id} -> backend generates PDF -> client offers download/print.

**Scenarios alternatifs**  
• **A1 — No archive results:** system shows “No records found” and suggests adjusting filters.  
• **A2 — Export error:** backend returns 500 -> client shows error.  
• **A3 — Access restriction:** if ASM role lacks permission for certain archive subsets, backend returns 403.

*****Figure 19: Consult Retiree File & Generate Certificate Sequence Diagram*

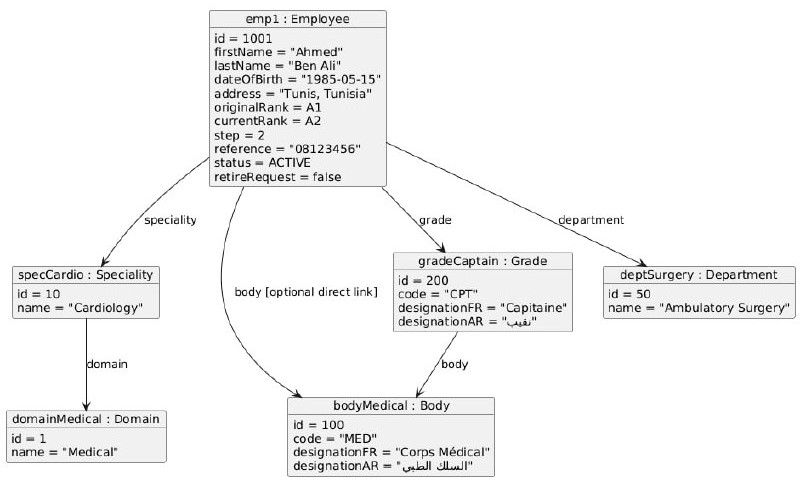
**2.4.2 Analysis Model**

**2.4.2.1 Overall Class Diagram:**

****

*Figure 20: Overall Class Diagram*

**2.4.2.2 Overall Object Diagram:**

****

*Figure 21: Overall Object Diagram*

**2.4.3 Design Model**

**2.4.3.1 Generating Tables:**

**- Sql File:**-- bodies

CREATE TABLE body (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

code VARCHAR(255),

designation\_ar VARCHAR(255),

designation\_fr VARCHAR(255)

);

-- departments

CREATE TABLE department (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

-- domains

CREATE TABLE domain (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

-- grades (belongs to a body)

CREATE TABLE grade (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

body\_id BIGINT NOT NULL REFERENCES body(id),

code VARCHAR(255),

designation\_ar VARCHAR(255),

designation\_fr VARCHAR(255)

);

-- specialities (belongs to a domain)

CREATE TABLE speciality (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

domain\_id BIGINT NOT NULL REFERENCES domain(id),

name VARCHAR(255) NOT NULL

);

-- employees

CREATE TABLE employee (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

first\_name VARCHAR(255),

last\_name VARCHAR(255),

date\_of\_birth DATE,

address VARCHAR(255),

reference VARCHAR(255),

status VARCHAR(255) CHECK (status IN ('ACTIVE','RETIRED')) DEFAULT 'ACTIVE',

retire\_request BOOLEAN NOT NULL DEFAULT FALSE,

step INTEGER NOT NULL DEFAULT 0,

body\_id BIGINT REFERENCES body(id),

department\_id BIGINT REFERENCES department(id),

grade\_id BIGINT REFERENCES grade(id),

speciality\_id BIGINT REFERENCES speciality(id),

current\_rank VARCHAR(255) CHECK (current\_rank IN ('A1','A2','A3','A4','B1','B2','B3','C1','C2')),

original\_rank VARCHAR(255) CHECK (original\_rank IN ('A1','A2','A3','A4','B1','B2','B3','C1','C2'))

);

-- authentication / users table

CREATE TABLE users (

id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

username VARCHAR(255) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL, -- store strong hash

role VARCHAR(255) CHECK (role IN ('PERSONAL\_MANAGER','AGENT','ARCHIVE\_SERVICE\_MANAGER','DIRECTOR')) NOT NULL

);

**2.4.3.1 Important Algorithms:**

**1. Authentication**

**Algorithm:** Login

**Goal:** Authenticate a user and return their role.

**Input:**

* username: String
* password: String

**Output:** Role (Enum) or Exception

**algo**

**Algorithm Login(username, password)**

**Begin**

**user ← FindUserByUsername(username)**

**If user is NULL Then**

**Throw InvalidUsernameOrPasswordException**

**End If**

**If user.password ≠ password Then**

**Throw InvalidUsernameOrPasswordException**

**End If**

**Return user.role**

**End**

**```**

**2. Retirement Operations**

**Algorithm: ValidateRetirementRequest**

**Goal:** Finalize the retirement process for an employee.

**Input:**

* employeeId`: Long

**Output:** VOID or Exception

**algo**

**Algorithm ValidateRetirement(employeeId)**

**Begin**

**employee ← FindEmployeeById(employeeId)**

**If employee is NULL Then**

**Throw EmployeeNotFoundException**

**End If**

**employee.retireRequest ← False**

**employee.status ← RETIRED**

**Save(employee)**

**End**

**```**

**3. Modify retiree**

**Algorithm: ModifyEmployeeByASM**

**Goal:** Allow ASM (Admin) to override employee data with Director's authorization.

**Input:**

* employeeId: Long
* directorsCode: String
* data: EmployeeDTO (Contains new DepartmentId, SpecialityId, etc.)

**Output:** VOID or Exception

**algo**

**Algorithm ModifyEmployeeByASM(employeeId, directorsCode, data)**

**Begin**

**// 1. Authorize**

**director ← FindUserByRole(DIRECTOR)**

**If director.password ≠ directorsCode Then**

**Throw AccessDeniedException**

**End If**

**// 2. Fetch & Validate Target**

**employee ← FindEmployeeById(employeeId)**

**If employee is NULL Then**

**Throw EmployeeNotFoundException**

**End If**

**If employee.status ≠ ACTIVE Then**

**Throw AccessDeniedException**

**End If**

**// 3. Validate Dependencies**

**department ← FindDepartmentById(data.departmentId)**

**If department is NULL Then**

**Throw DepartmentNotFoundException**

**End If**

**speciality ← FindSpecialityById(data.specialityId)**

**If speciality is NULL Then**

**Throw SpecialtyNotFoundException**

**End If**

**// 4. Update Fields**

**employee.firstName ← data.firstName**

**employee.lastName ← data.lastName**

**employee.dateOfBirth ← data.dateOfBirth**

**employee.address ← data.address**

**employee.originalRank ← data.originalRank**

**employee.currentRank ← data.originalRank**

**employee.step ← data.step**

**employee.reference ← data.reference**

**employee.retireRequest ← data.retireRequest**

**// 5. Update Relations**

**employee.department ← department**

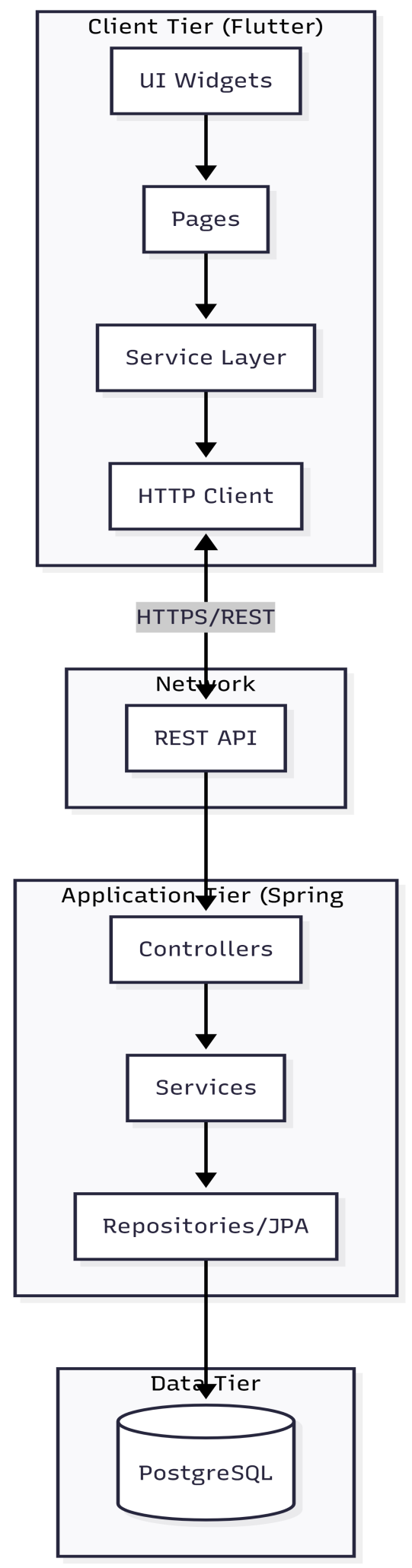
**employee.speciality ← speciality**

**Save(employee)**

**End**

**```**

**2.4.4 Implementation Model**

**** *Figure 22: Component Diagram*

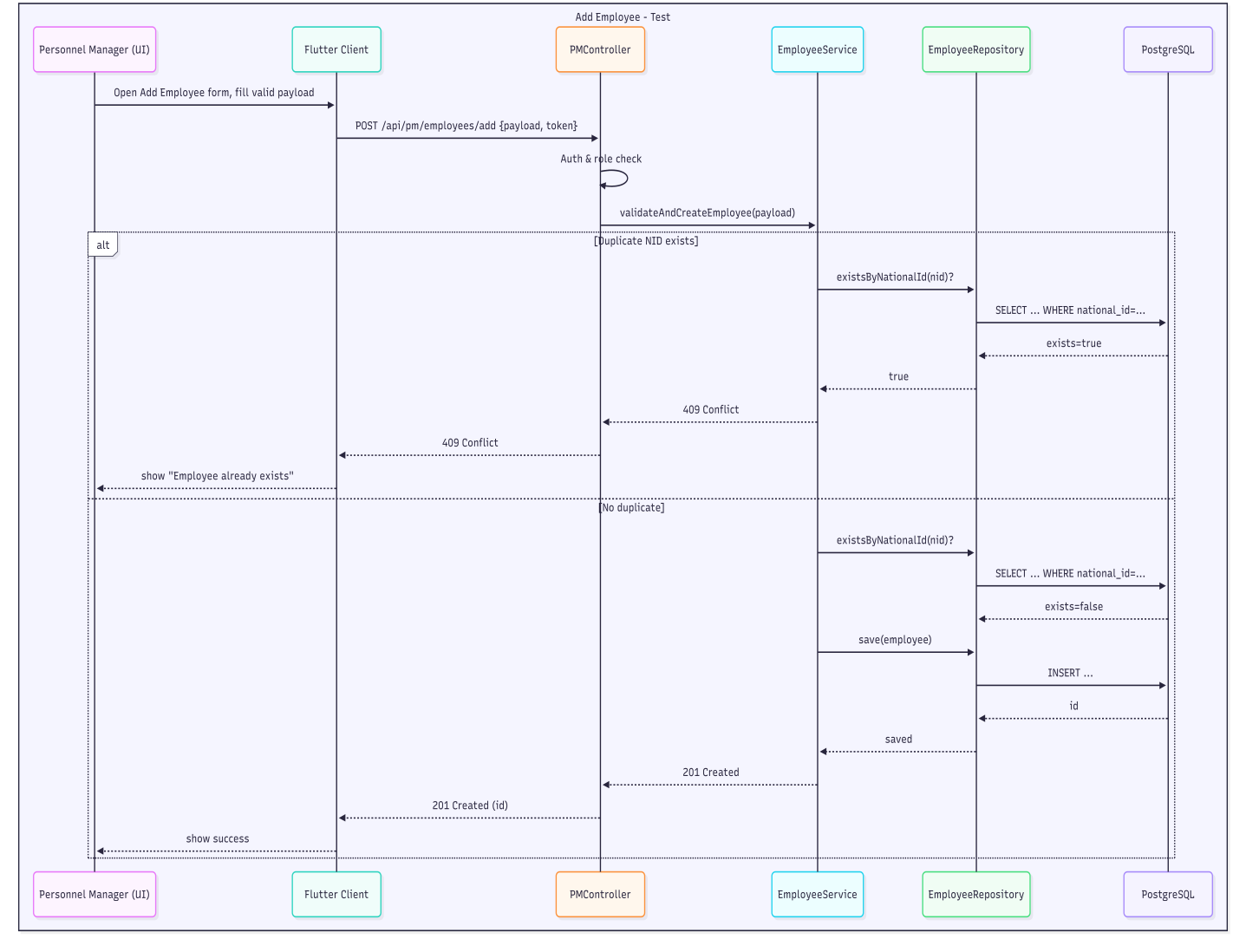
**2.4.5 Deployment Model**

 *Figure 23: Deployment Diagram*

**2.4.6 Test Model**

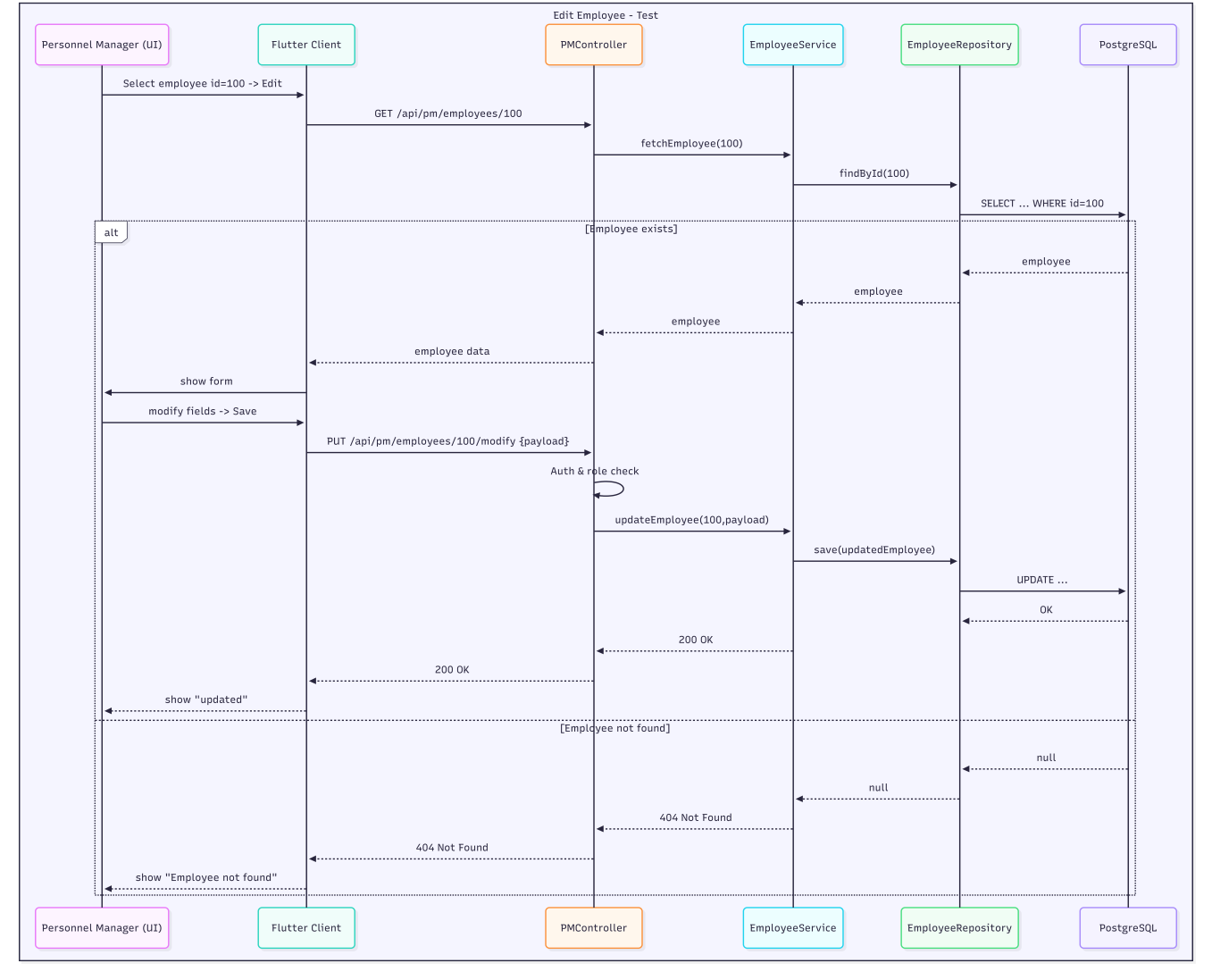
**2.4.6.1 Test Sequence Diagrams:**

**1) Test 1 — Add Employee :**

****

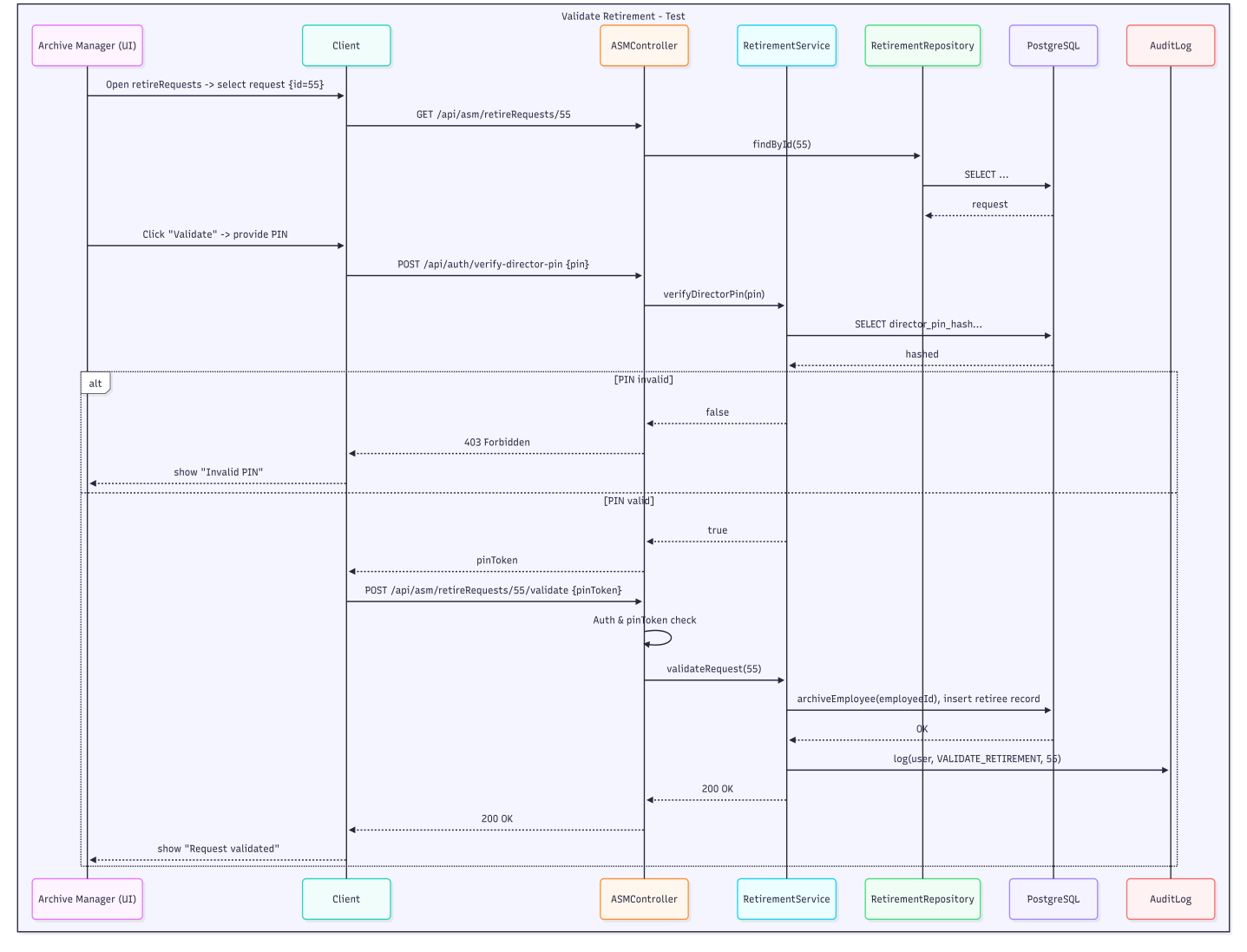
*Figure 24: Test of Add Employee Sequence Diagram*

**2) Test 2 — Modify Employee :**

****

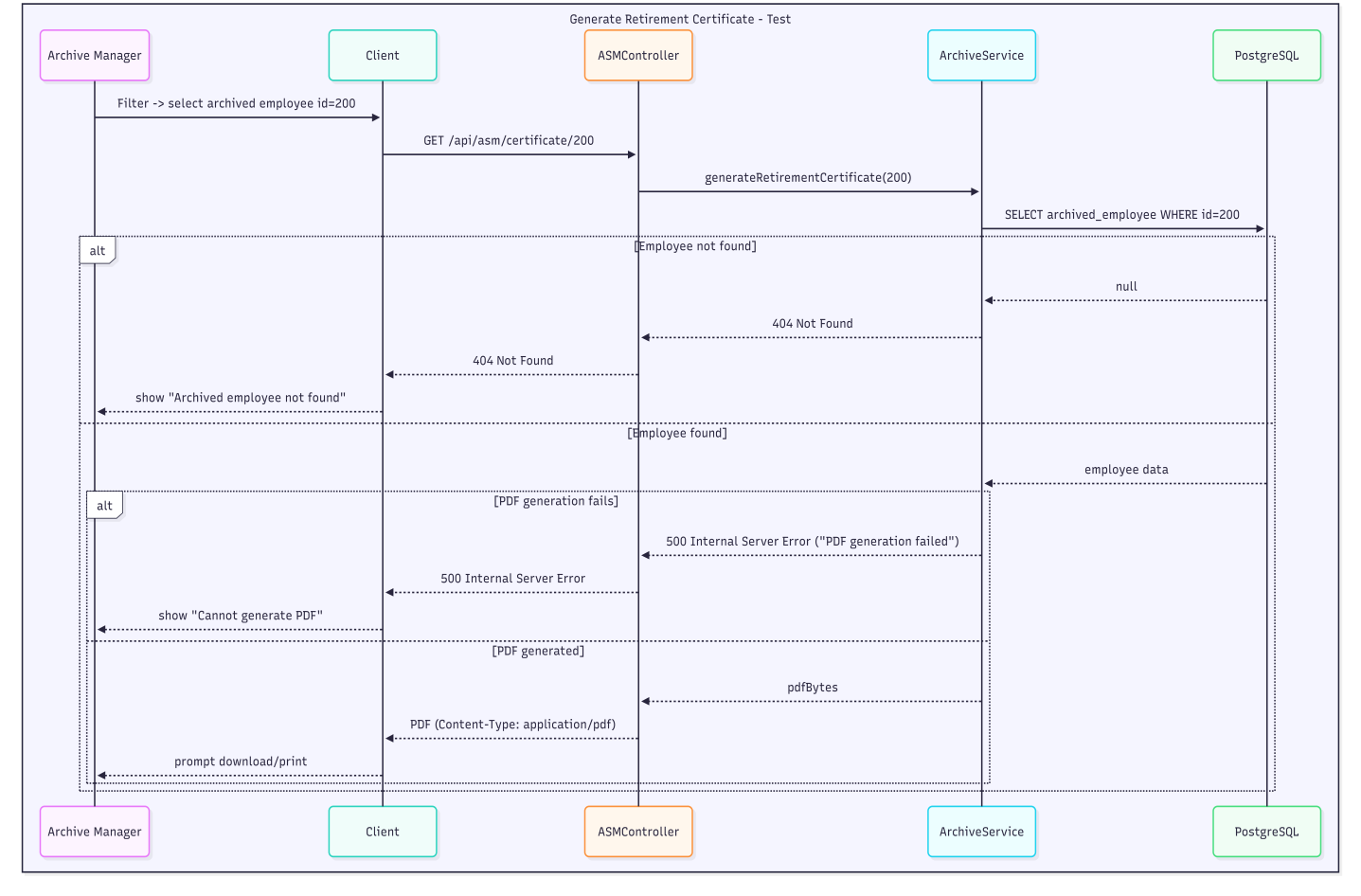
*Figure 25: Test of Modify Employee Sequence Diagram*

**3) Test 3 — Validate Retirement:**

****

*Figure 26: Test of Retirement Validation Sequence Diagram*

**4) Test 4 — Generate Retirement Certificate :**

****

*Figure 27: Test of Retirement Certificate Generating Sequence Diagram*

**2.4.7 Conclusion**

The development of this personnel management system showed how essential it is to combine clear architecture, well-structured models, and thorough testing to build software that is reliable and easy to maintain. By working through each layer—from the database and interfaces to implementation, deployment, and testing—the system gained a coherent structure that supports good performance, scalability, and straightforward operation.

The tiered Client–Application–Data architecture helps keep responsibilities separate, making the system easier to extend in the future. Well-defined REST interfaces ensure smooth communication between the Flutter client and the backend, while security features like JWT authentication, Director PIN verification, and upcoming inactivity timeouts improve overall robustness.

Testing through detailed sequence diagrams confirmed that key processes—such as adding employees, updating profiles, validating retirements, and generating certificates—work correctly in both normal and exceptional cases. Although some limitations remain, such as platform-specific printing differences and the simple Director PIN mechanism, the system is built with future enhancements in mind, including stronger authentication and better cross-platform support.

Overall, this work provides a solid and modern foundation for an HR management solution that is modular, secure, and ready to evolve as organizational needs grow.

**Chapter 3: IMPLEMENTATION**

**3.1 Introduction**

This chapter presents the technical realization of the Human Resources Management System (HRMS). It describes the choice of application type, the technologies selected for development, the justification behind these choices, and the security mechanisms integrated into the system. The HRMS is implemented as a desktop application designed to operate within the EPSP’s internal network, ensuring reliability, ease of deployment, and full support for the operational requirements of the Human Resources department. **3.2 Choice of Application Type**

We have developed a hybrid application using Flutter, which offers the following advantages over a traditional website:

| **Criteria** | **Application (Flutter)** | **Traditional Website** |
| --- | --- | --- |
| **Performance** | Native compilation provides smooth 60fps UI | Limited by browser rendering engine |
| **Platform Reach** | Single codebase deploys to Windows, Android, Web, macOS, iOS | Requires separate mobile adaptation |
| **Native Features** | Direct access to file system for PDF generation and printing | Limited browser sandbox access |
| **User Experience** | Rich, responsive interfaces with platform-specific design | Constrained by web standards |
| **Security** | Compiled binary code harder to reverse engineer | JavaScript source visible to users |

The choice of a Flutter application is justified by the administrative nature of the system, which requires:

* **Desktop deployment** for office use on Windows workstations
* **PDF generation and printing** capabilities for official documents
* **Rich data entry forms** with real-time validation
* **Role-based access** from a single unified interface  
  1. **Technologies Used for Production**

**3.3.1 Frontend Technology Stack**

| **Technology** | **Version** | **Purpose** |
| --- | --- | --- |
| **Flutter SDK** | ^3.8.1 | Cross-platform UI framework |
| **Dart** | ^3.0 | Programming language |
| **go\_router** | ^17.0.0 | Declarative routing and navigation |
| **http** | ^1.2.0 | HTTP client for REST API communication |
| **pdf** | ^3.11.1 | PDF document generation |
| **printing** | ^5.13.3 | Cross-platform print functionality |
| **path\_provider** | ^2.1.4 | Access to device file system directories |
| **popover** | ^0.3.1 | Popup menu components |

**3.3.2 Backend Technology Stack**

| **Technology** | **Version** | **Purpose** |
| --- | --- | --- |
| **Spring Boot** | 3.0+ | Java/Kotlin web framework |
| **Kotlin** | 1.8+ | Backend programming language |
| **Gradle (Kotlin DSL)** | Latest | Build automation tool |
| **Spring Data JPA** | Included | Object-Relational Mapping |
| **Hibernate Validator** | Included | Bean validation framework |
| **PostgreSQL** | 15+ | Production relational database |

**3.3.3 Infrastructure & DevOps**

| **Component** | **Technology** | **Purpose** |
| --- | --- | --- |
| **Backend Hosting** | Render.com | Cloud platform for Spring Boot API |
| **Database Hosting** | Aiven Hosting | Managed database service |
| **Version Control** | Git / GitHub | Source code management |
| **IDE** | Visual Studio Code | Development environment |
| **API Testing** | Postman / Browser DevTools | Endpoint verification |

* + 1. **API Endpoints Structure**The backend exposes RESTful endpoints organized by module:

**Base URL: https://hr-server-3s0m.onrender.com**

**EP-01 AUTHENTICATION (/api/auth)**

**Endpoints:**

* POST /login - User authentication
* POST /reset-password - Password reset (requires Director's Code)

**EP-02 EMPLOYEE MANAGEMENT (/api/pm)**

**Endpoints:**

* GET /employees List all active employees
* GET /employees/{id} Get employee details by ID
* GET /test/allemployees List all employees (including retired)
* GET /employees/toRetire List employees eligible for retirement
* POST /employees/add Create new employee record
* PUT /employees/{id}/modify Modify employee details

**EP-03 ARCHIVE SERVICE MANAGER (/api/asm)**

**Endpoints:**

* GET /employees - List archived employees
* GET /retireRequests - List all retirement requests
* POST /retireRequests/{id}/validate - Validate a retirement request
* PUT /employees/{id}/modify - Modify archived employee (with Director's Code)
* POST /domains/create - Create a new domain
* POST /domains/{domainId}/specialities/create - Create a specialty under a domain
* DELETE /domains/{domainId}/delete - Delete a domain
* GET /bodies/{bodyId}/grades - List grades of a given body
* POST /bodies/{bodyId}/grades/create - Create a grade under a body
* POST /bodies/{bodyId}/grades/{gradeId}/modify - Modify grade details
* DELETE /bodies/{bodyId}/grades/{gradeId}/delete - Delete a grade

**EP-04 BODY (CORPS) MANAGEMENT (/api/agent)**

**Endpoints:**

* POST /bodies/create - Create new body
* PUT /bodies/{id}/modify - Modify body details
* DELETE /bodies/{id}/delete - Delete body

**EP-05 COMMON REFERENCE DATA (/api)**

**Endpoints:**

* GET /departments - List all departments
* GET /bodies - List all bodies (corps)
* GET /domains - List all domains
* GET /specialties - List all specialties  
  1. **Technology Justification**

**3.4.1 Why Flutter for Frontend?**

| **Justification** | **Explanation** |
| --- | --- |
| **Single Codebase, Multi-Platform** | The HR system runs on Windows desktops (primary), Android tablets (field use), and web browsers—Flutter compiles from one Dart codebase to all platforms |
| **Rich UI Components** | Material Design widgets provide polished, consistent interfaces for complex data entry forms required by HR functions |
| **Hot Reload** | Rapid development iteration with instant UI updates during development |
| **PDF Generation** | The pdf and printing packages enable native document creation for employee records, reports, and certificates |
| **Type Safety** | Dart's static typing catches errors at compile time, essential for data-sensitive HR applications |
| **Performance** | Compiled to native ARM/x64 code, avoiding JavaScript bridge overhead |

**3.4.2 Why Spring Boot (Kotlin) for Backend?**

| **Justification** | **Explanation** |
| --- | --- |
| **Enterprise-Grade Framework** | **Spring Boot provides robust dependency injection, security, and data access layers ideal for government administration systems** |
| **Kotlin Advantages** | **Null safety, concise syntax, and modern language features reduce boilerplate and potential runtime errors** |
| **Spring Data JPA** | **Simplifies database operations with repository patterns and automatic query generation** |
| **Scalability** | **Stateless REST architecture allows horizontal scaling on cloud platforms** |
| **Security Integration** | **Spring Security provides out-of-the-box authentication and authorization mechanisms** |

**3.4.3 Why PostgreSQL for Database?**

| **Justification** | **Explanation** |
| --- | --- |
| **Relational Model** | **Complex relationships (Employee → Department → Body) map naturally to relational tables** |
| **JSON Support** | **PostgreSQL's JSONB type offers flexibility for storing semi-structured data** |
| **Open Source** | **No licensing costs for government deployments** |

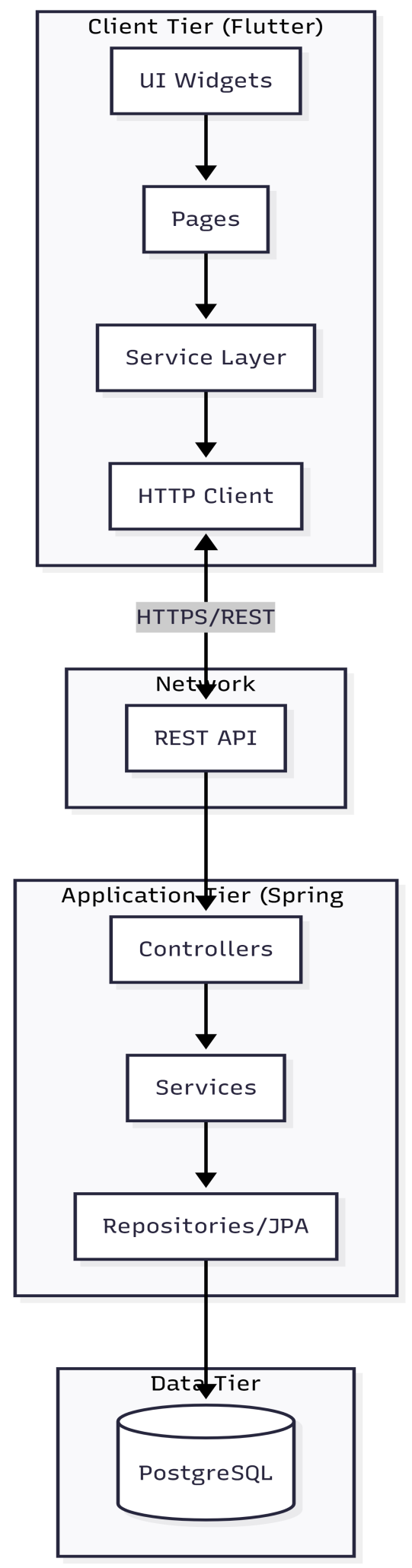
**3.4.4 Why Render.com for Hosting?**

| **Justification** | **Explanation** |
| --- | --- |
| **Free Tier Availability** | Suitable for academic projects and prototyping |
| **Automatic Deployments** | Git-based continuous deployment from repository |
| **Managed PostgreSQL** | Database hosting with automatic backups |
| **HTTPS by Default** | All endpoints secured with TLS certificates |
| **Zero Configuration Scaling** | Simple vertical scaling when needed |

* 1. **Additional Featured**

**3.5.1 Software Architecture**

**3.5.1.1 Client-Server Architecture Diagram**

****

*Figure 22: Implementation Diagram(Client-Server)*

**3.5.1.2 Frontend Architecture** The Flutter client follows a Service-Oriented Architecture:

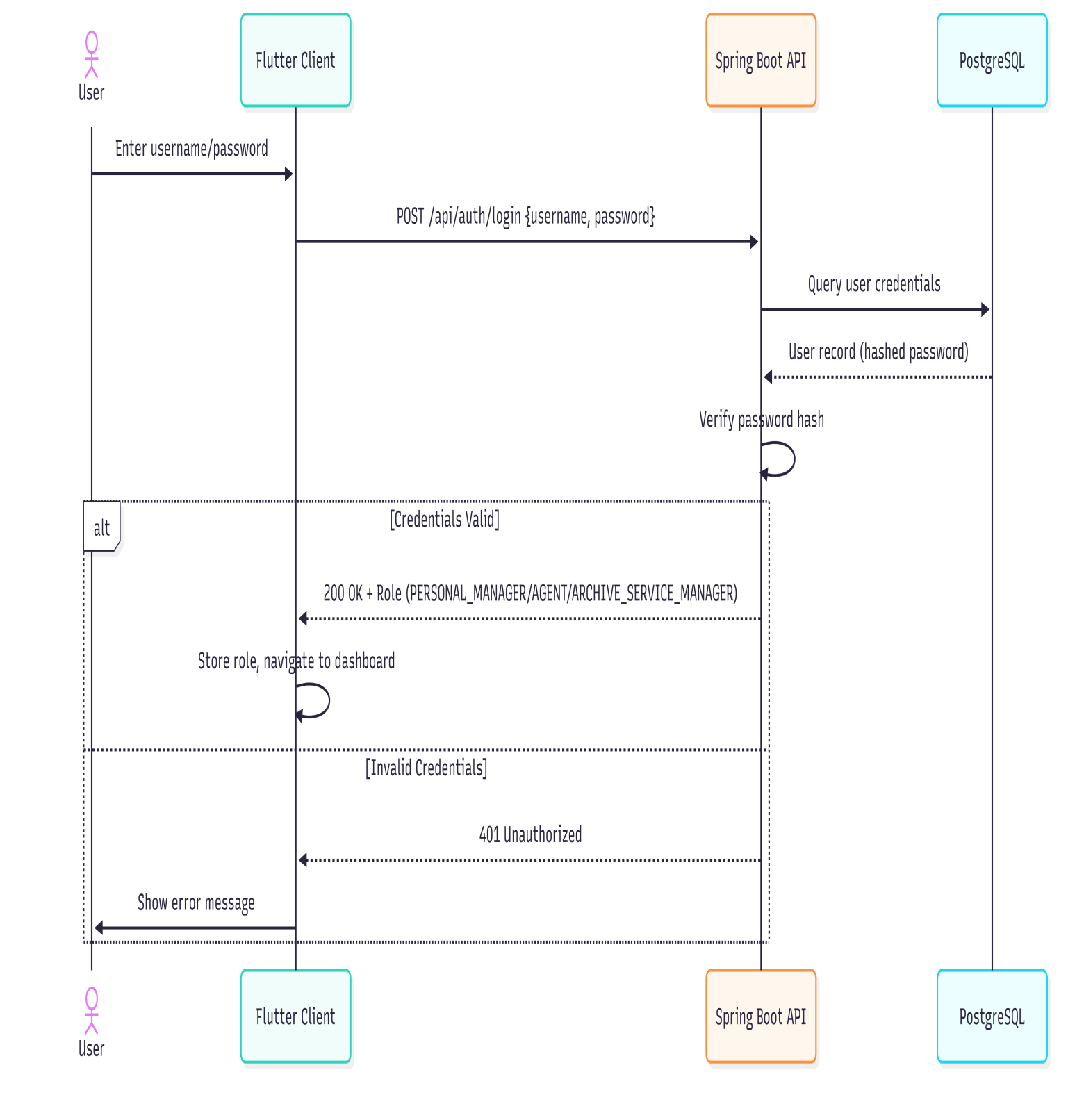
| **Layer** | **Components** | **Responsibility** |
| --- | --- | --- |
| **Presentation** | AuthinticationDialog, SideBar, EmployeeList | UI rendering and user interaction |
| **Pages** | Loginpage, Homepage | Route-level containers |
| **Services** | AuthService, EmployeeService, PdfService | Business logic and API communication |
| **Models** | Employee, Department, Body, Grade | Data structure definitions |

**3.5.1.3 Backend Architecture** The Spring Boot Backend implements a Layered Architecture:

| **Layer** | **Components** | **Responsibility** |
| --- | --- | --- |
| **Controller** | AuthController, PMController, ASMController | HTTP request handling |
| **Service** | AuthService, EmployeeService | Business logic implementation |
| **Repository** | EmployeeRepository, UserRepository | Database access via JPA |
| **DTO** | EmployeeDTOPM, LoginRequest, ASMModifyEmployeeRequest | API contract definitions |
| **Entity** | Employee, User | Database table mappings |

**3.5.2 Security Mechanisms**

**3.5.2.1 Authentication System**

****

*Figure 28: Authentication System Sequence Diagram*

**3.5.2.2 Password Security**

| **Mechanism** | **Implementation** | **Purpose** |
| --- | --- | --- |
| **Password Hashing** | BCrypt algorithm via Spring Security | Passwords are never stored in plaintext; BCrypt includes salt and is computationally expensive to prevent brute-force attacks |
| **Password Reset** | Director's code verification via /api/auth/reset-password | Password changes require manager-level authorization |
| **Input Trimming** | username.trim() and password.trim() in AuthService | Prevents accidental whitespace issues |

**3.5.2.3 Role Based Access Control**

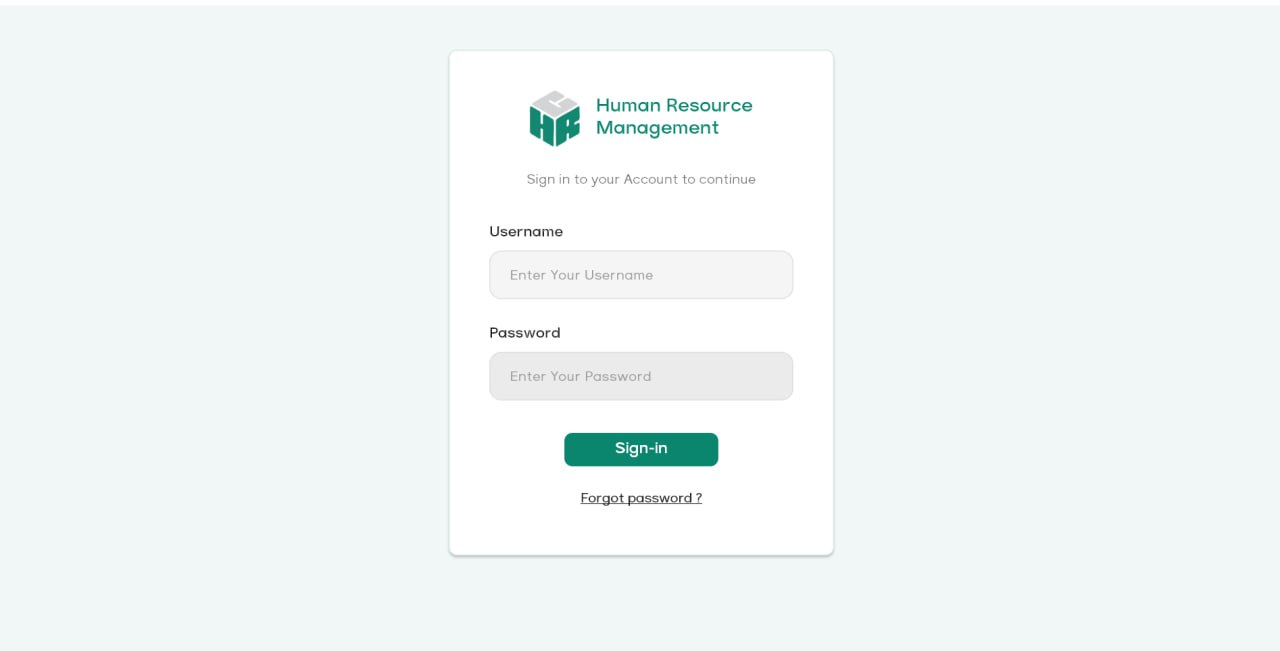
The system enforces access based on three user roles:

| **Role** | **Code** | **Permissions** |
| --- | --- | --- |
| **Personnel Manager (PM)** | PERSONAL\_MANAGER | Full employee CRUD, view all departments |
| **Agent** | AGENT | Read-only access to employee lists, generate reports, manage bodis |
| **Archive Service Manager (ASM)** | ARCHIVE\_SERVICE\_MANAGER | Manage retirement requests, modify archived records (requires Director's Code) manage domains, specialities and grades |

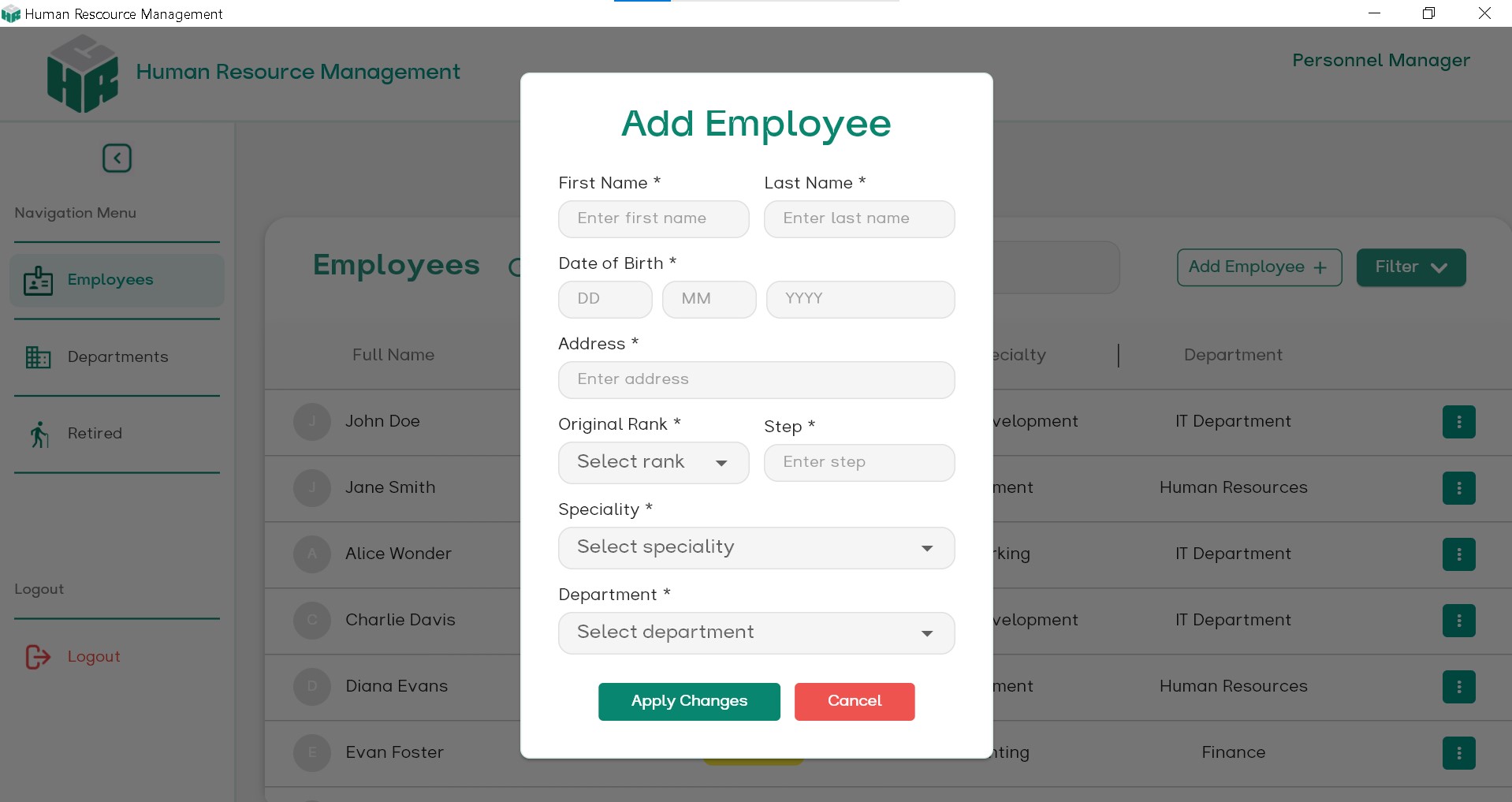
**3.5.2.4 Input Validation**

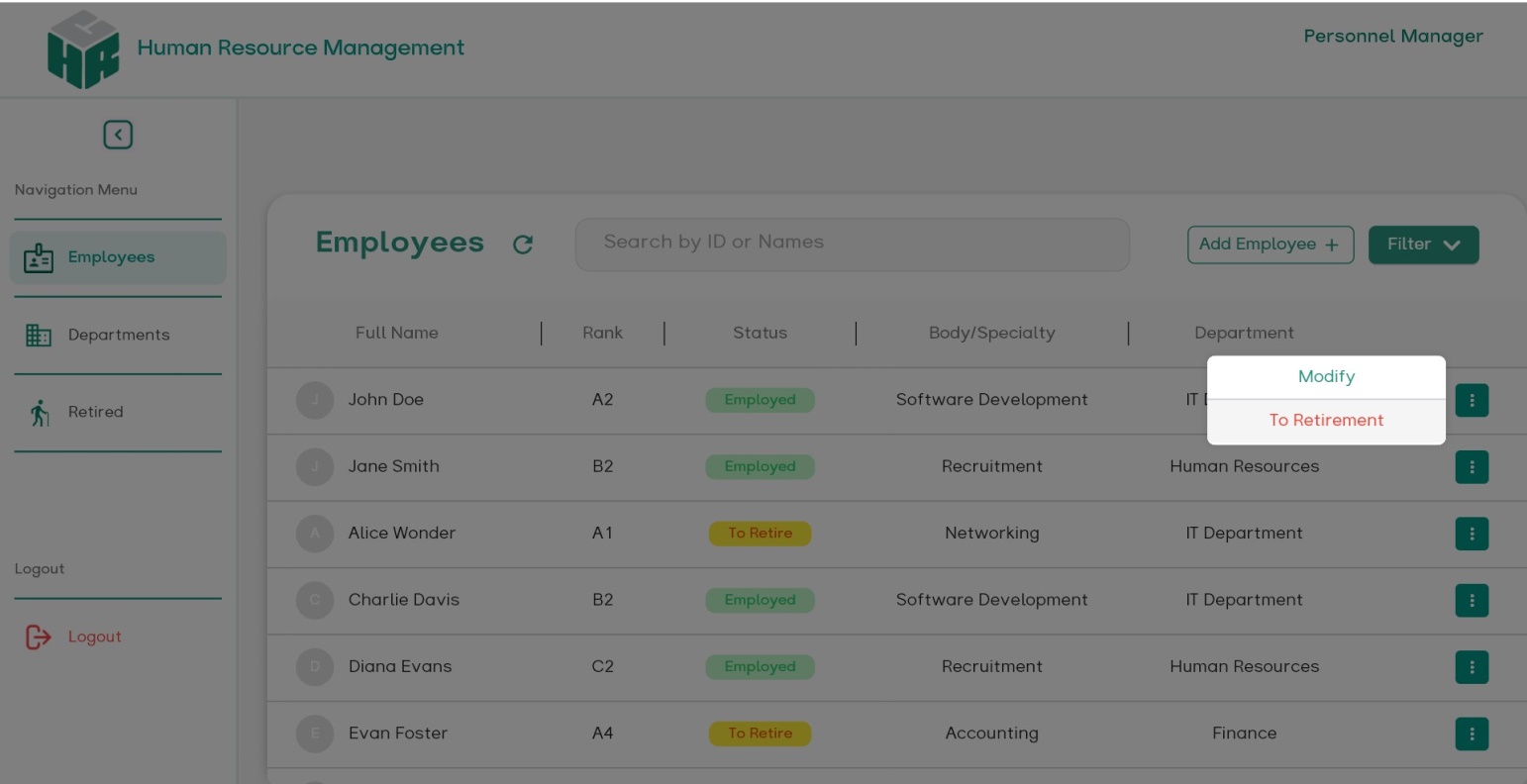
| **Layer** | **Mechanism** | **Example** |
| --- | --- | --- |
| **Frontend** | Form validation via TextFormField.validator | Required fields, format checks before submission |
| **Backend** | Bean Validation (@Valid, @NotNull, @Size) | Entity-level constraints enforced by Hibernate Validator |
| **Database** | PostgreSQL constraints | NOT NULL, CHECK, foreign key constraints |

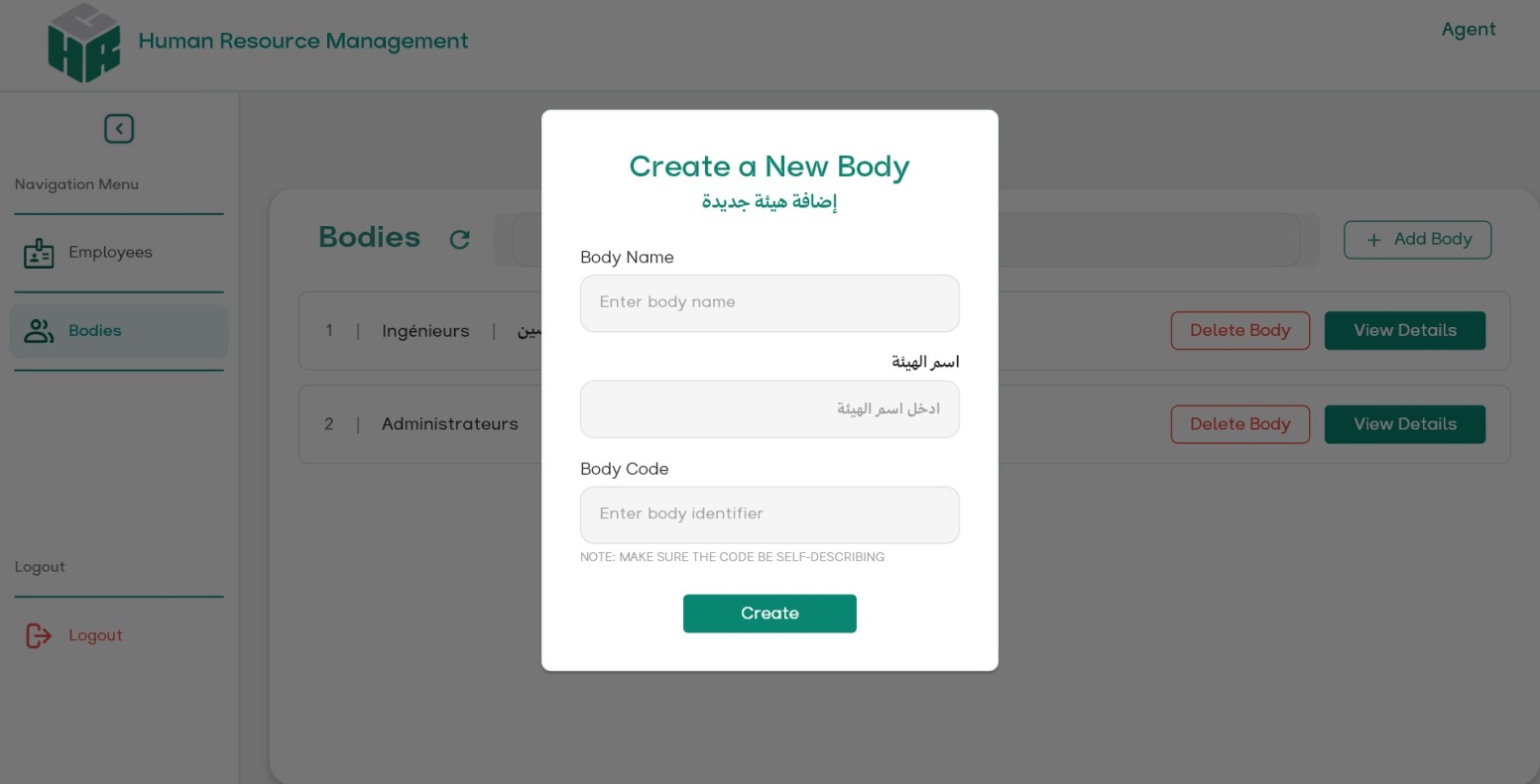
**3.5.3 Screenshots:**

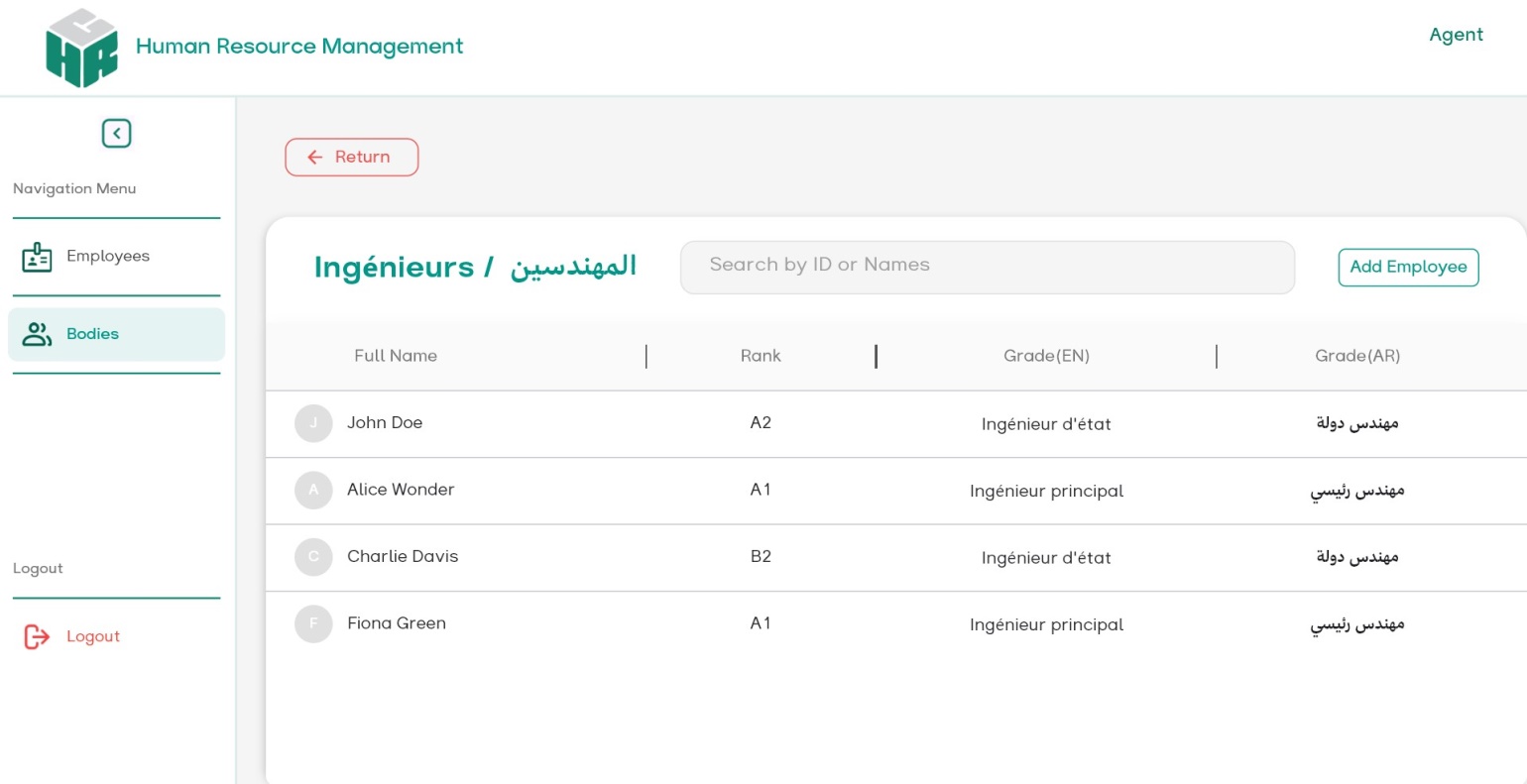
*****Figure 29: Login Page*

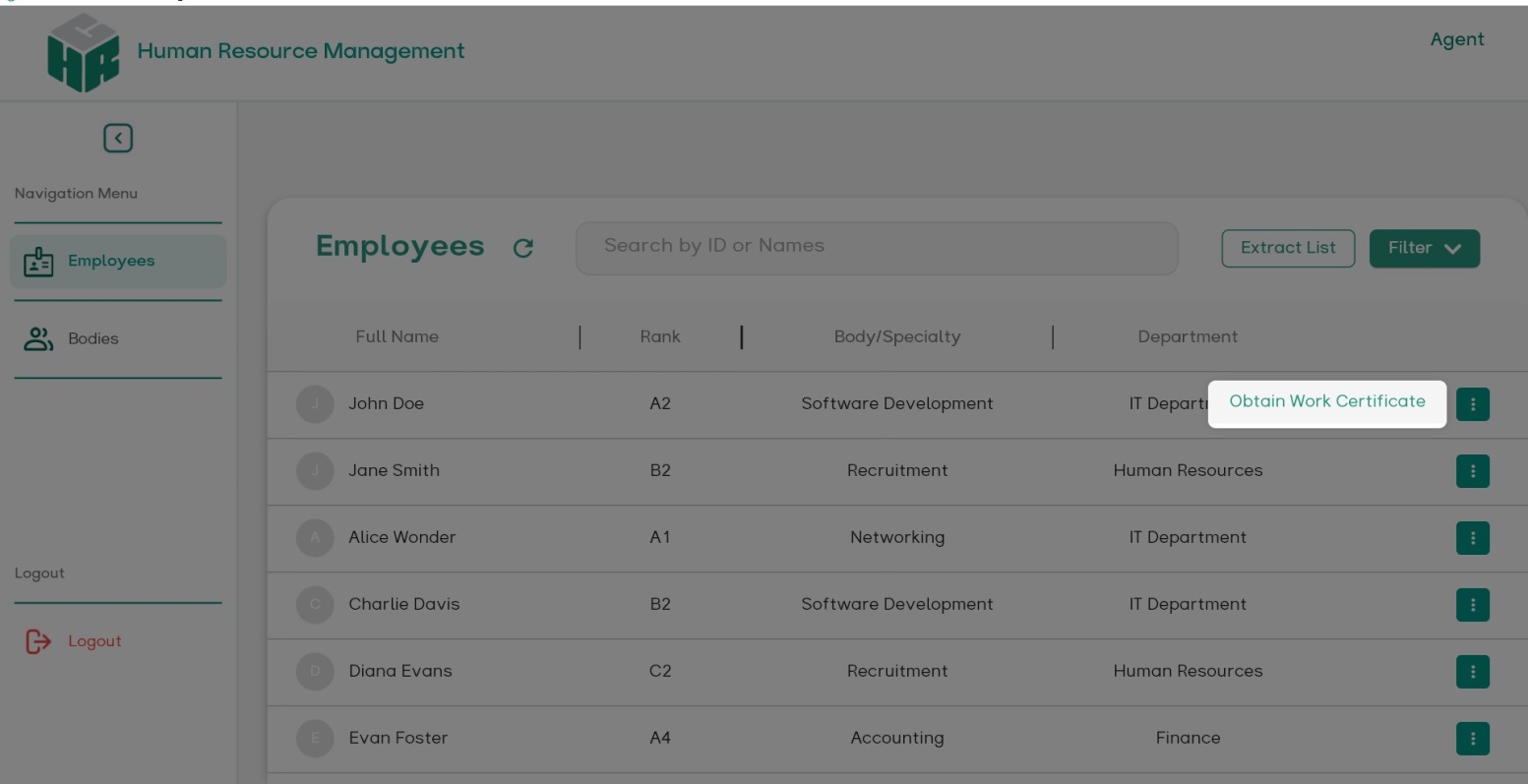
*Figure 30: Add Employee [PM]*

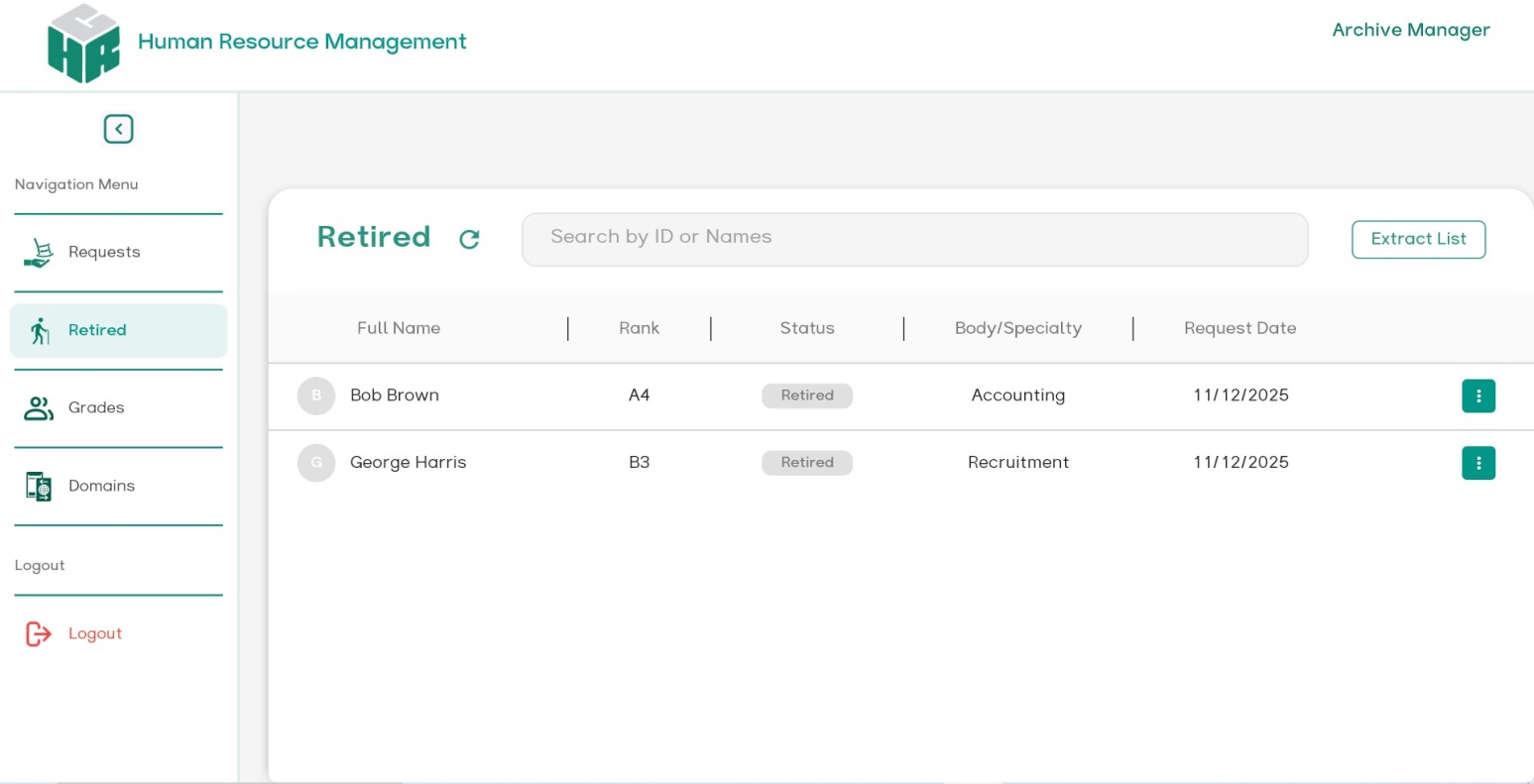


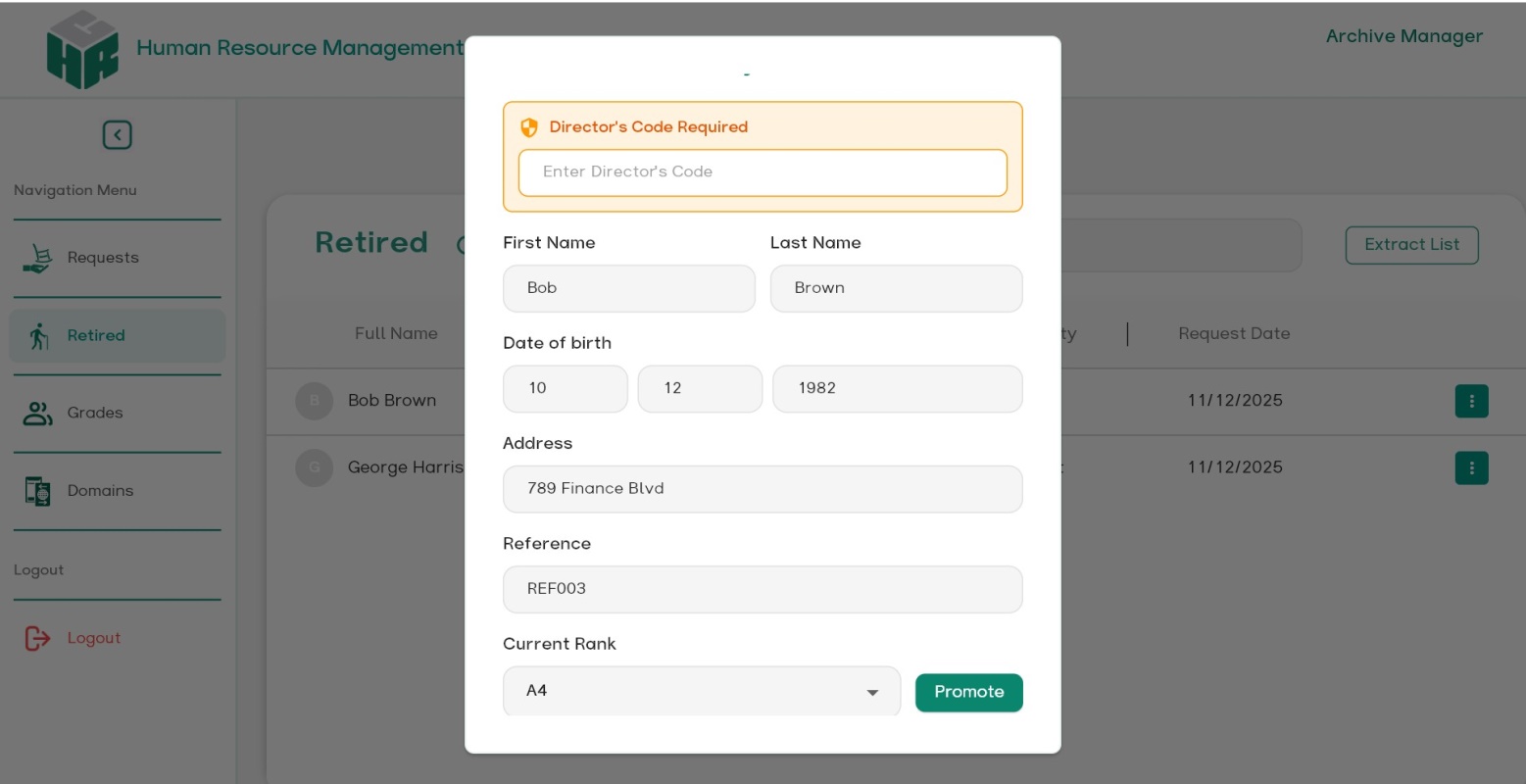
**** *Figure 31: Employee Page [PM]*

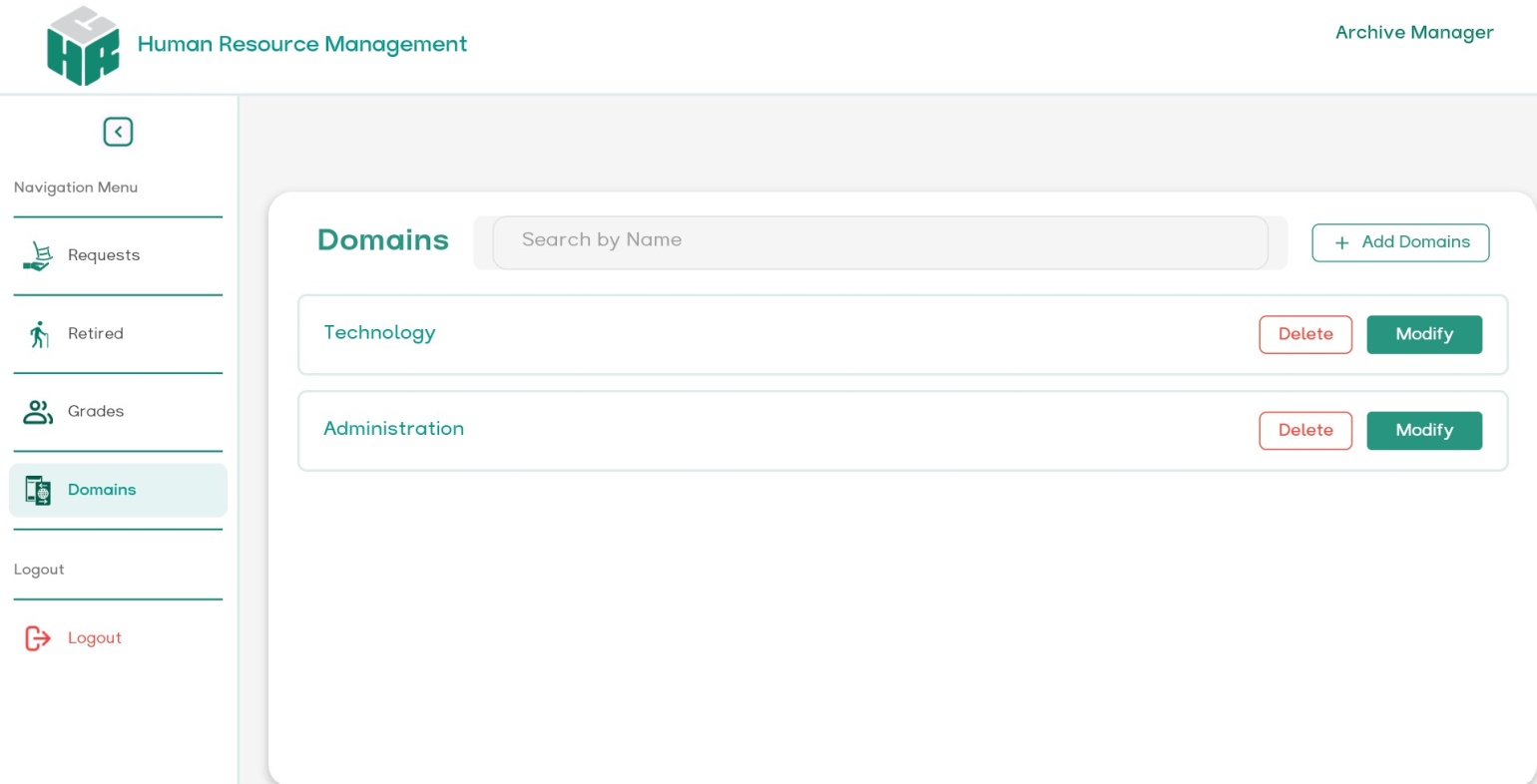
*****Figure 32: Add Body [Agent]*

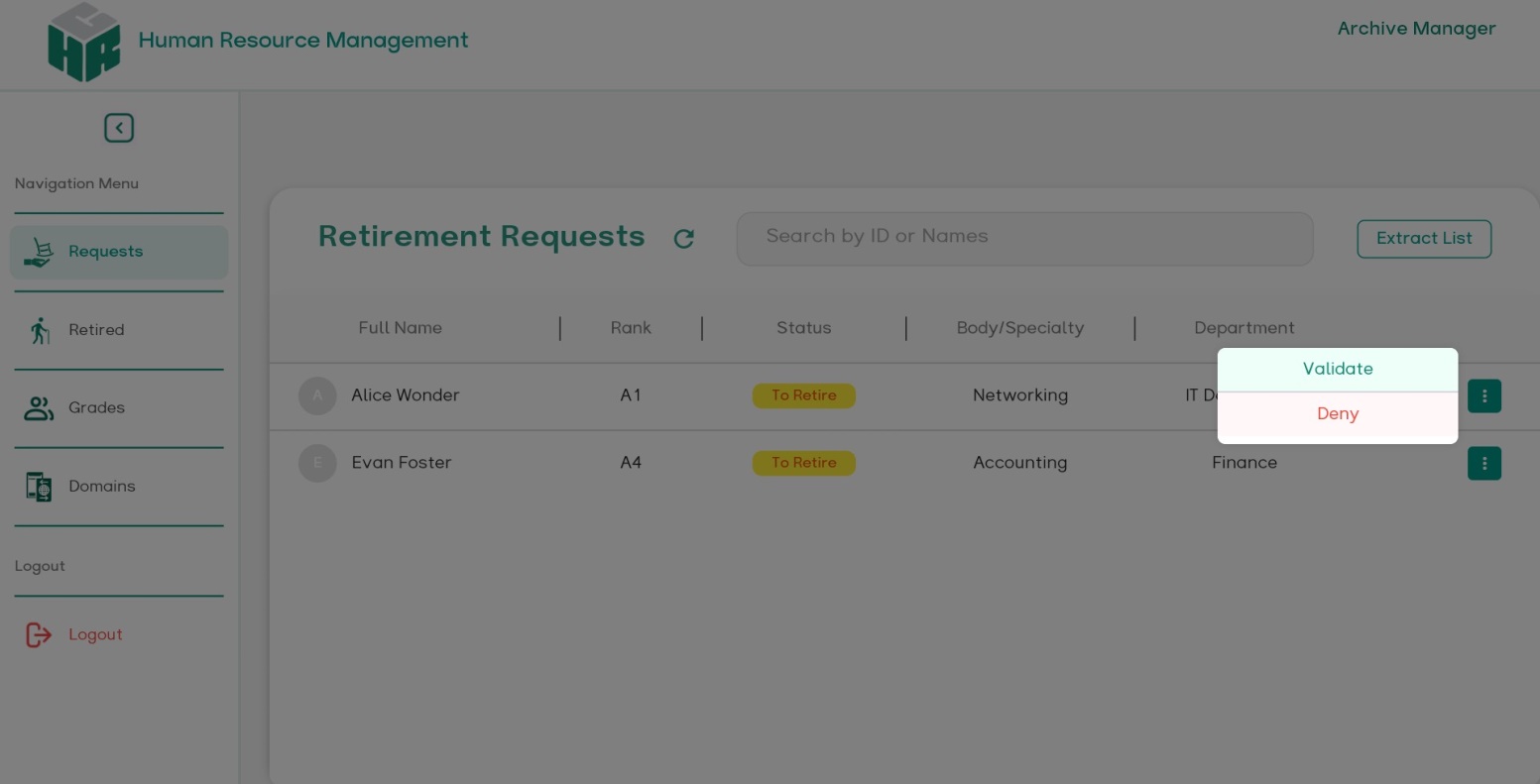
**** *Figure 33: Body Detail Page [Agent]*

*****Figure 34: Employee Page [Agent]*

**** *Figure 35: Retired Employee Page [ASM]*

*****Figure 36: Modify Retiree [ASM]*

**** *Figure 37: Domain Page [ASM]*

*****Figure 38: Request Retirment page [ASM]*

**3.5.4 Backup & Maintenance:**

### Backups

* Daily DB backups (Aiven or managed provider).

### Maintenance

* Schedule maintenance windows outside office hours.
* Apply security patches monthly; test on staging before production.

**3.5.5 Limitations & Known Issues**

* Current implementation targets desktop primarily; mobile/web is supported but some native features (printing) have platform-specific differences.
* Director PIN handling is simple (shared PIN); in future replace with multi-factor Director approval (e.g., Director login + OTP) to improve security.

**3.6 Conclusion**

The implementation of the Human Resources Management System (HRMS) results in a robust full-stack solution that meets the functional and non-functional requirements defined earlier. The Flutter client provides a unified codebase deployable across Windows, web, and Mobile devices, offering responsive interfaces and native PDF generation essential for HR workflows. On the server side, Spring Boot with Kotlin ensures a scalable and type-safe architecture, enriched with enterprise-grade security and seamless PostgreSQL integration.

Security mechanism; including BCrypt password hashing, role-based access control, Director PIN validation, and HTTPS encryption, guarantee the protection of sensitive HR data. Cloud deployment on Render.com simplifies operations through automated builds, managed PostgreSQL hosting, and TLS certificates. Overall, this architecture achieves an effective balance between development productivity and operational reliability, making the system well-suited for EPSP administrative environments.

**General Conclusion**

The development of the Human Resources Management System (HRMS) presented in this work has demonstrated the effectiveness of applying structured software engineering methods to real administrative needs. Through field analysis and consultation with EPSP HR staff, we identified the major limitations of the existing manual processes. These findings guided a clear set of specifications that led to the creation of a centralized, secure, and efficient digital platform. The implemented system successfully supports employee management, assignments, promotions, certificate generation, and retirement procedures, all while ensuring data integrity and traceability.

Technically, the combination of Flutter for the client interface and Spring Boot with Kotlin for the backend proved to be a powerful and scalable solution. PostgreSQL provided robust data persistence, and security mechanisms—such as role-based access control, password hashing, Director authorization, and HTTPS—ensured compliance with the sensitivity of HR information. The resulting application significantly improves workflow reliability and reduces the administrative burden on EPSP personnel.

Although the system fulfills its primary objectives, several perspectives remain open for future work. Enhancements could include advanced reporting dashboards, automated notifications for key HR events, stronger authentication mechanisms such as multi-factor verification, and deployment strategies enabling multi-site synchronization. Additional modules—for leave management, training history, or payroll integration—could further increase the system’s value and operational scope.

In summary, this project provides a solid and extensible foundation for digital transformation within EPSP institutions and demonstrates that well-designed software can play a decisive role in improving the efficiency and quality of public health administration.