

CSc3350 Software Development
Software Design Document
Group Team Project Fall 2025

Company Z

Team 7

Poorav Rawat
Shelden Rattray
Eden-Amari Wilson
Alaya Mccall

December 8th, 2025

TABLE OF CONTENTS

1.	<i>INTRODUCTION.....</i>	<i>4</i>
1.1.	<i>Software Purpose.....</i>	<i>4</i>
2.	<i>DATABASE SCHEMA DIAGRAM.....</i>	<i>4</i>
3.	<i>JAVA CLASS DIAGRAMS.....</i>	<i>4</i>
4.	<i>PROGRAMMING TASKS (10 minimum from User Story).....</i>	<i>4</i>
5.	<i>TEST CASES (cleaned up from #6 in Deliverables).....</i>	<i>4</i>
6.	<i>SEQUENCE DIAGRAMS (cleaned up from #7 in Deliverables).....</i>	<i>4</i>
7.	<i>APPENDIX.....</i>	<i>4</i>
7.1.	<i>Definitions and Acronyms.....</i>	<i>4</i>

1. INTRODUCTION

1.1. Software Purpose

The purpose of this software system is to develop a secure, efficient, and scalable Employee Management System (EMS) for Company Z. The organization currently maintains all employee information manually through MySQL scripts, with limited security and no user interface. The company plans to rapidly expand its workforce from 55 employees to more than 150 within the next 18 months. For this purpose, an automated, reliable, and secure system is required to support HR operations and employee access.

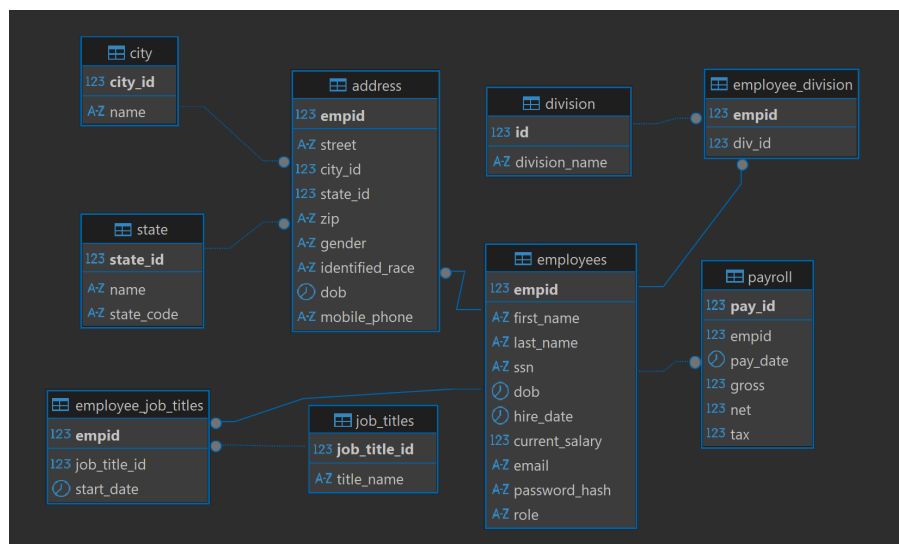
Provided Services

- Role-based access control for HR administrators and general employees.
- Centralized employee management, including searching, updating, and reporting on employee data.
- Integrated MySQL database operations, ensuring data consistency, secure storage, and proper relational design.
- A usable console that improves workflow efficiency and maintains data integrity.
- Scalability to accommodate future employee growth and additional reporting needs.

Intended Audience

- Employees at CompanyZ - Employees so that they can see their pay, salary, personal information, and other data.
- HR department at Company Z - HR administrator and management team that oversees employee operations can use this to access employee information.

2. DATABASE SCHEMA DIAGRAM



```

classDiagram
    class Company {
        +employees: List<Employee>
        +getEmployees(): List<Employee>
        +addEmployee(Employee): void
        +removeEmployee(Employee): void
        +updateEmployee(Employee): void
        +searchEmployee(String): List<Employee>
        +printEmployeeTable(): void
    }

    class App {
        +employees: List<Employee>
        +addEmployee(Employee): void
        +removeEmployee(Employee): void
        +updateEmployee(Employee): void
        +searchEmployee(String): List<Employee>
        +printEmployeeTable(): void
    }

    class AWS {
        +RESET: String
        +BLACK: String
        +RED: String
        +GREEN: String
        +YELLOW: String
        +BLUE: String
        +MAGENTA: String
        +CYAN: String
        +WHITE: String
        +HEX: String
        +BRIGHT_BLACK: String
        +BRIGHT_RED: String
        +BRIGHT_GREEN: String
        +BRIGHT_YELLOW: String
        +BRIGHT_BLUE: String
        +BRIGHT_MAGENTA: String
        +BRIGHT_CYAN: String
        +BRIGHT_WHITE: String
        +BACKGROUND_BLACK: String
        +BACKGROUND_RED: String
        +BACKGROUND_GREEN: String
        +BACKGROUND_YELLOW: String
        +BACKGROUND_BLUE: String
        +BACKGROUND_MAGENTA: String
        +BACKGROUND_CYAN: String
        +BACKGROUND_WHITE: String
        +BRIGHT_BACKGROUND_BLACK: String
        +BRIGHT_BACKGROUND_RED: String
        +BRIGHT_BACKGROUND_GREEN: String
        +BRIGHT_BACKGROUND_YELLOW: String
        +BRIGHT_BACKGROUND_BLUE: String
        +BRIGHT_BACKGROUND_MAGENTA: String
        +BRIGHT_BACKGROUND_CYAN: String
        +BRIGHT_BACKGROUND_WHITE: String
    }

    class TablePrinter {
        +TABLE_ALIGN_FORMAT: final String
        +TABLE_DECORATOR: final String
        +QUERY_DECORATOR: final String
        +TablePrinter()
        +printEmployeeTable(): void
    }

    class PayrollService {
        +payroll(): List<Payroll>
        +getPayroll(): List<Payroll>
        +getPayrollByDateRange(LocalDate, LocalDate): List<Payroll>
        +getPayrollByDateRange(LocalDate, LocalDate): List<Payroll>
        +getPayrollByDateRange(LocalDate, LocalDate): List<Payroll>
    }

    class EmployeeService {
        +employees(): List<Employee>
        +getEmployees(): List<Employee>
        +addEmployee(Employee): void
        +removeEmployee(Employee): void
        +updateEmployee(Employee): void
        +searchEmployee(String): List<Employee>
        +printEmployeeTable(): void
    }

    class ReportService {
        +getTotalPayrollByYear(int, int): Map<String, BigDecimal>
        +getTotalPayrollByMonth(int, int): Map<String, BigDecimal>
        +getTotalPayrollByQuarter(int, int): Map<String, BigDecimal>
    }

    class DatabaseManager {
        +URL: String
        +USER: String
        +PASSWORD: String
        +connection: Connection
        +getConnection(): Connection
        +closeConnection(): void
        +executeQuery(String, String): ResultSet
        +executeUpdate(String): int
        +executeQuery(String, String): ResultSet
        +executeUpdate(String): int
    }

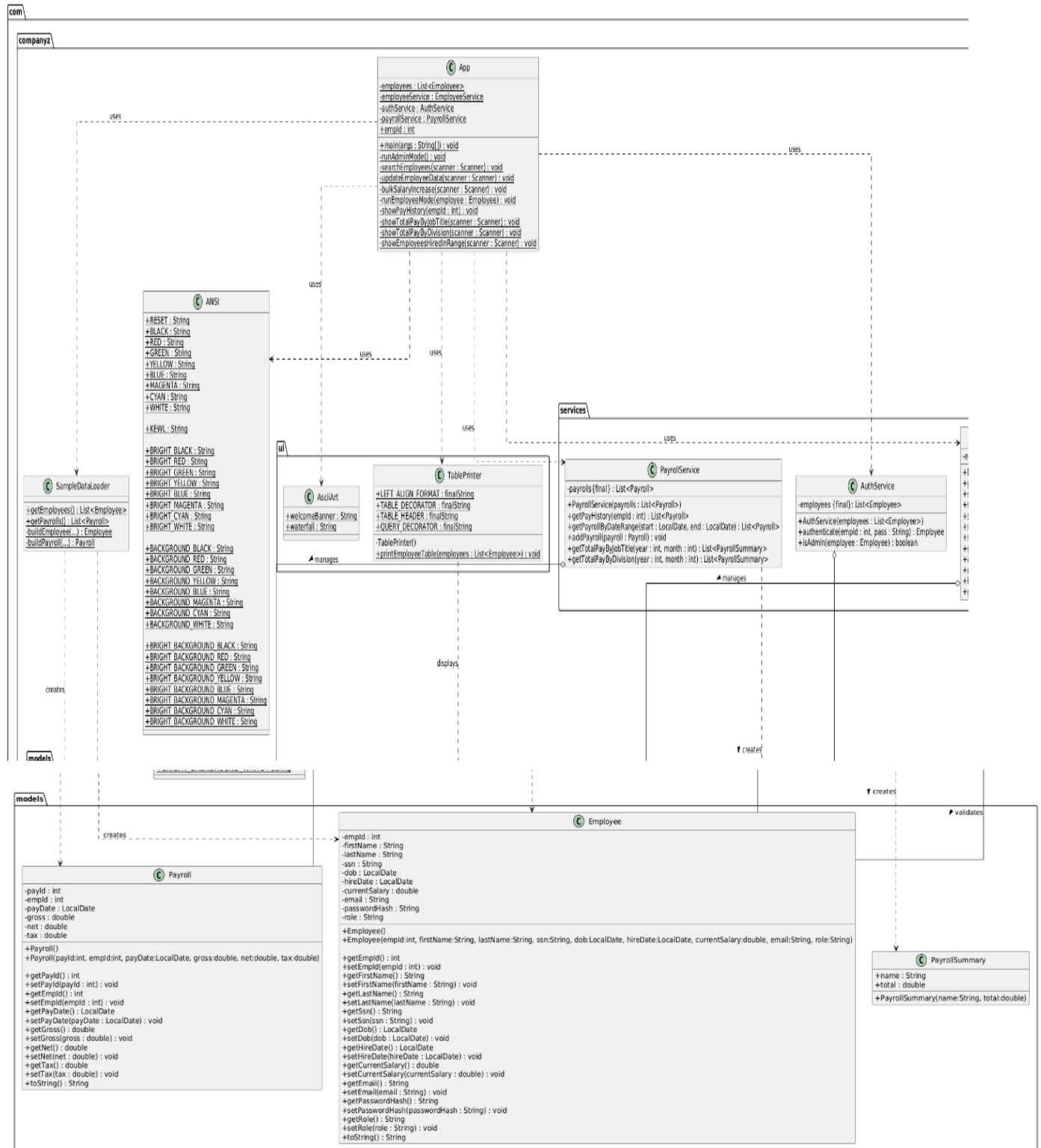
    class Payroll {
        +id: int
        +employee: int
        +payable: LocalDate
        +gross: double
        +net: double
        +tax: double
        +Payroll()
        +Payroll(int, LocalDate, LocalDate, double, double, double)
        +getPayroll(): List<Payroll>
        +getPayrollByDateRange(LocalDate, LocalDate): List<Payroll>
        +getPayrollByDateRange(LocalDate, LocalDate): List<Payroll>
        +getPayrollByDateRange(LocalDate, LocalDate): List<Payroll>
    }

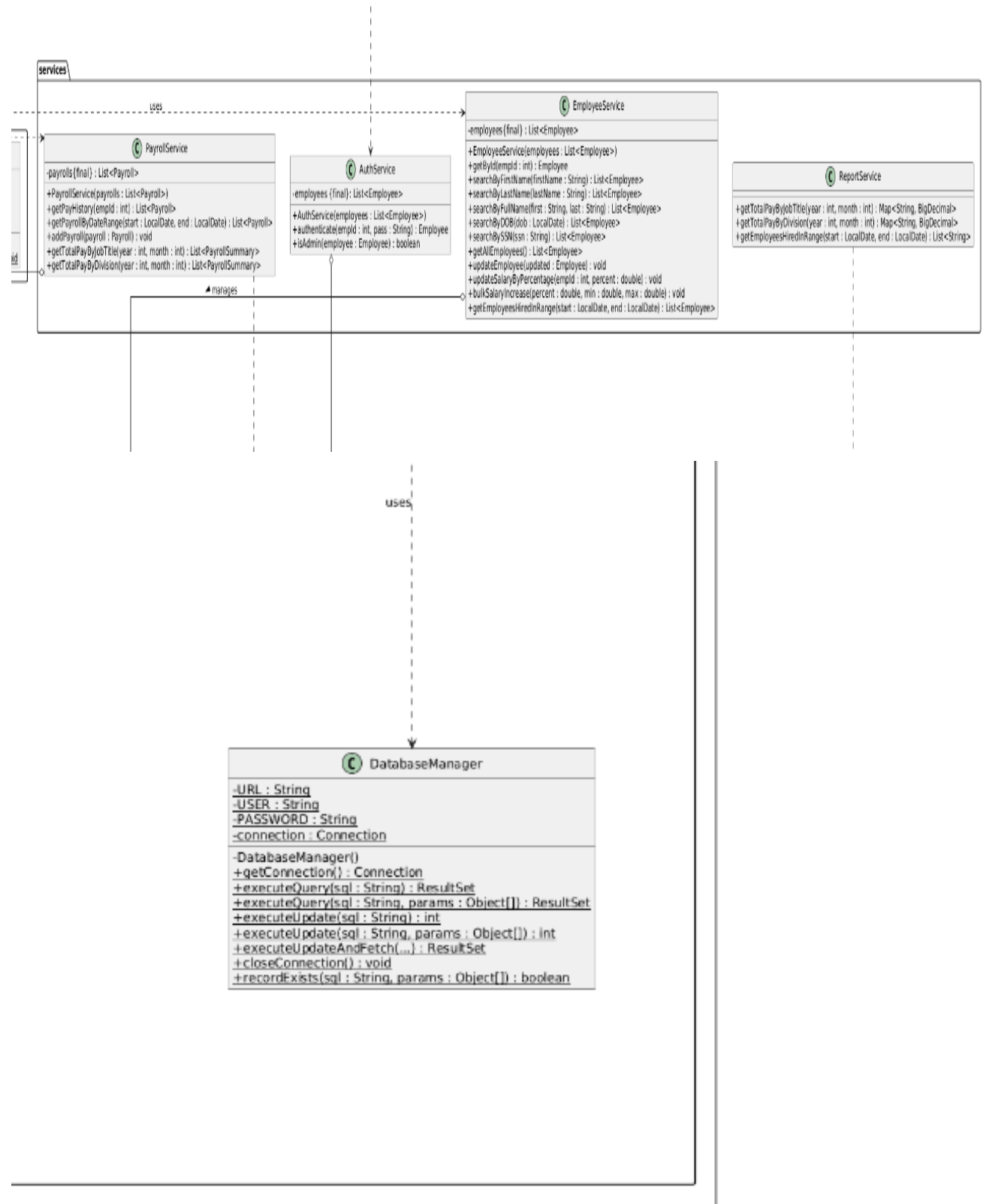
    class Employee {
        +empId: int
        +firstName: String
        +lastName: String
        +ssn: String
        +dob: LocalDate
        +hireDate: LocalDate
        +currentSalary: double
        +email: String
        +passwordHash: String
        +role: String
        +Employee()
        +Employee(int, String, String, String, String, LocalDate, LocalDate, double, String, String)
        +getEmployee(): List<Employee>
        +getEmployeeByDateRange(LocalDate, LocalDate): List<Employee>
        +getEmployeeByDateRange(LocalDate, LocalDate): List<Employee>
        +getEmployeeByDateRange(LocalDate, LocalDate): List<Employee>
    }

    class PayrollSummary {
        +name: String
        +total: double
        +PayrollSummary(String, double)
    }

    Company --> App
    App --> AWS
    App --> TablePrinter
    App --> PayrollService
    App --> EmployeeService
    App --> ReportService
    App --> DatabaseManager
    App --> Payroll
    App --> Employee
    App --> PayrollSummary
    
```

Enlarged for better viewing:





4. PROGRAMMING TASKS (10 minimum from User Story)

User Authentication

- Implement a login function that utilizes an employee's unique identifier (employee ID) – the admin ID is 1.
- Store user passwords and use them for validation.
- Return messages that describe if the provided details were valid or invalid, and what type of user was validated (employee vs admin)

HR Employee Search Handler

- Implement a search function that returns a list of *all* employees.
- Implement a (self) search function that uses employee ID, name (first, last, or full name), hire date, SSN, or DOB, and returns the results.

Employee Search Handler

- Implement a search function that provides the employee's information (using their previously provided employee ID) and returns the related results.

Database Connection Manager

- Create a module responsible for establishing, maintaining, and closing connections to the database.
- Implement methods for executing queries, handling exceptions, and ensuring secure database transactions.

Update employee's salary for an increase of a particular percentage

- Implement a function that updates an employee's salary by a specified percentage increase.
- This function should take the employee's ID and the percentage increase as inputs.
- It should connect through the Database Connection Manager, perform the update securely, and confirm success or failure.

Data Creation

- Create a table for employees, where each entry has a first name, last name, DOB, SSN, salary, hire date, job title, and division tied to it.
- Create a table for divisions, where each entry has a division ID and a division name. Each division (based on ID) should also have a table linked to it for monthly pay reports.
- Create a table for job titles, where each entry has a title ID and a title name. Each title (based on ID) should also have a table linked to it for monthly pay reports.

- Populate the employees table.
- Populate the tables for divisions and their monthly reports.
- Populate the tables for job titles and their monthly reports.

5. TEST CASES (cleaned up from #6 in Deliverables)

Start with User story (scenario) then create pass/fail test cases from each features/function

Programming task: Updating employee data

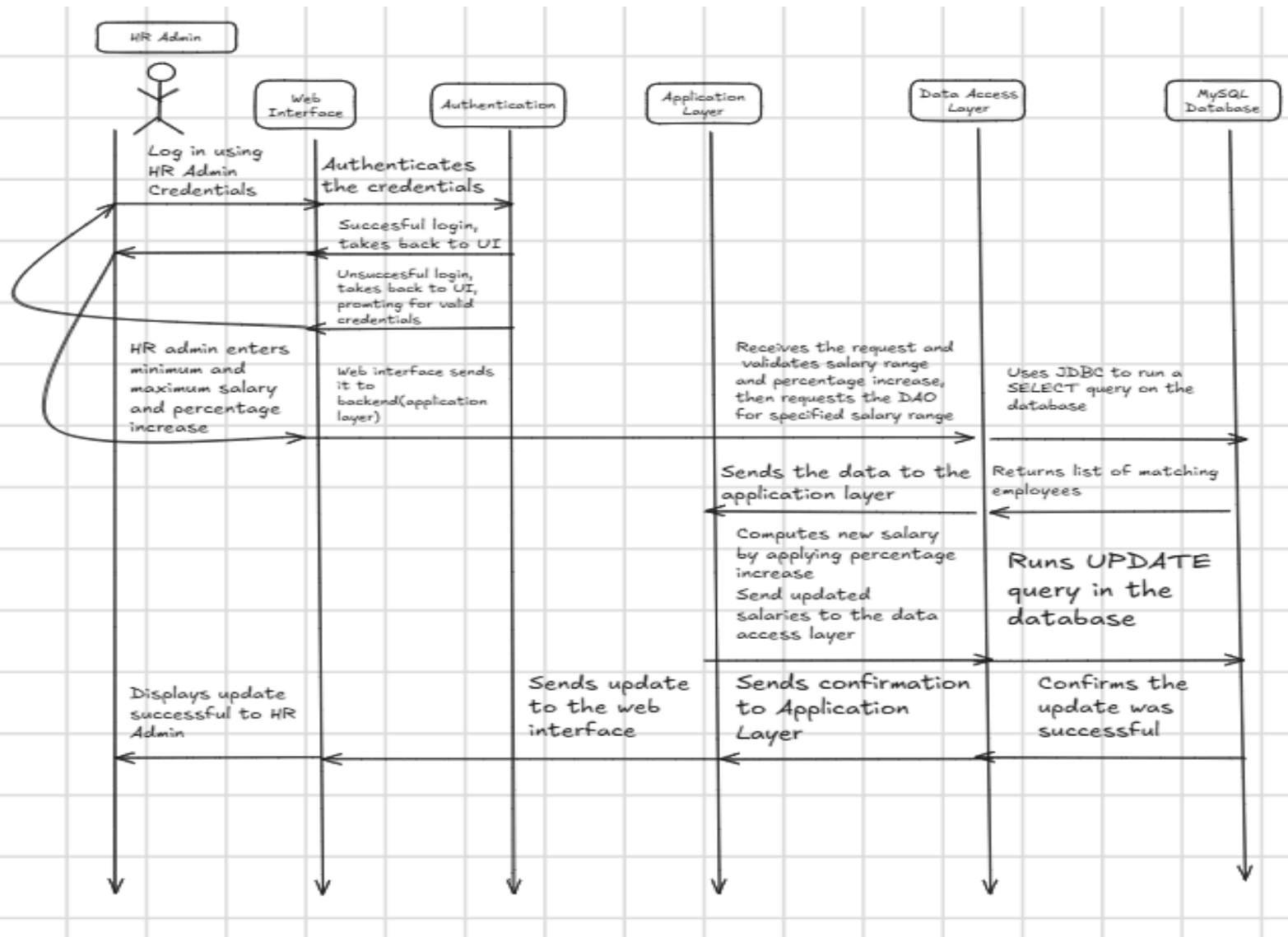
- Updating employee info using the correct inputs
- Attempting to update the information of an employee who does not exist
- Attempting to update employee information using incorrect types (for example, entering an integer when prompted to edit their first name)

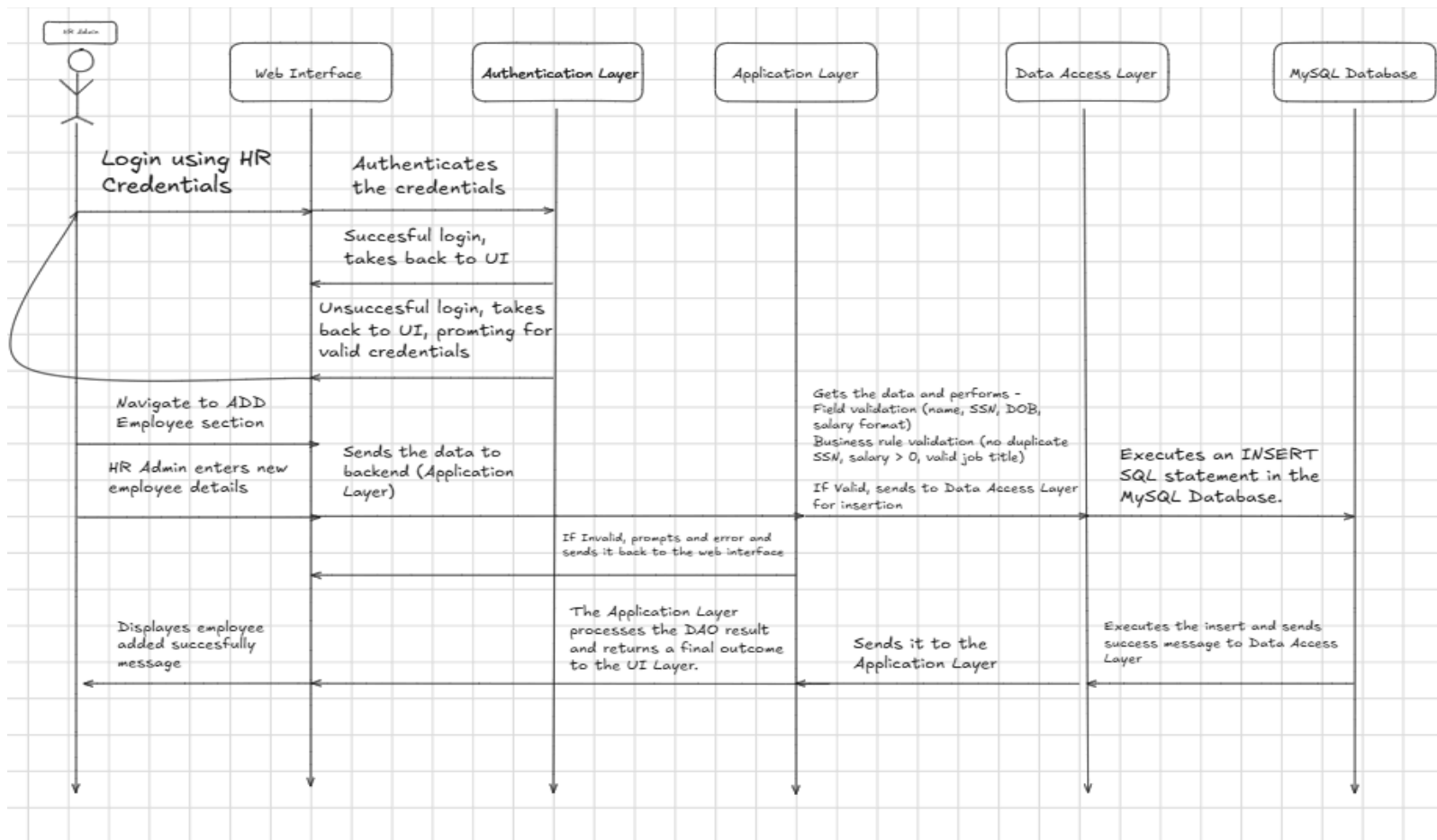
Programming task: Search for employee data

- Searching for employee info using the correct inputs
- Incorrectly entering an employee's information
- Attempting to search for the information of an employee who does not exist
- Attempting to show the information for a column that does not exist (for example, if the provided menu options are 1-9, what happens when the user enters 22?)

Programming Tasks: Update salary for all employees less than a particular amount

- Correctly entering the increase percentage and the bounds of the salary increase
- Entering a non-numeric salary value (lower bound)
- Entering a non-numeric salary value (upper bound)
- Entering a non-numeric for the percentage
- Attempting to update employees with salaries $< X$, but there are no rows in the table that match that

6. SEQUENCE DIAGRAMS (cleaned up from #7 in Deliverables)



7. APPENDIX

7.1. Definitions and Acronyms

Data Access Object (DAO) - An object that allows the program to access an underlying database system.

Schema - A model that reflects the features the development team plans to implement.

Table - A spreadsheet of rows and columns that stores data.

Terminal User Interface (TUI) - An interface that runs within a text-based terminal.

Query - A command that pulls or alters a table's data.