Dev Ops Dynasty

CSc3350 Software Development
Software Design Document
Group Team Project Spring 2026

Cornelius Moore
Armando Galvan
Victor Doga
Kadie Jenkins
Luis Garcia

Date: 04/26/2026

TABLE OF CONTENTS

Software Design Document

1. INTRODUCTION

1.1. Software Purpose

[Identify the purpose of system being built and its intended audience.]

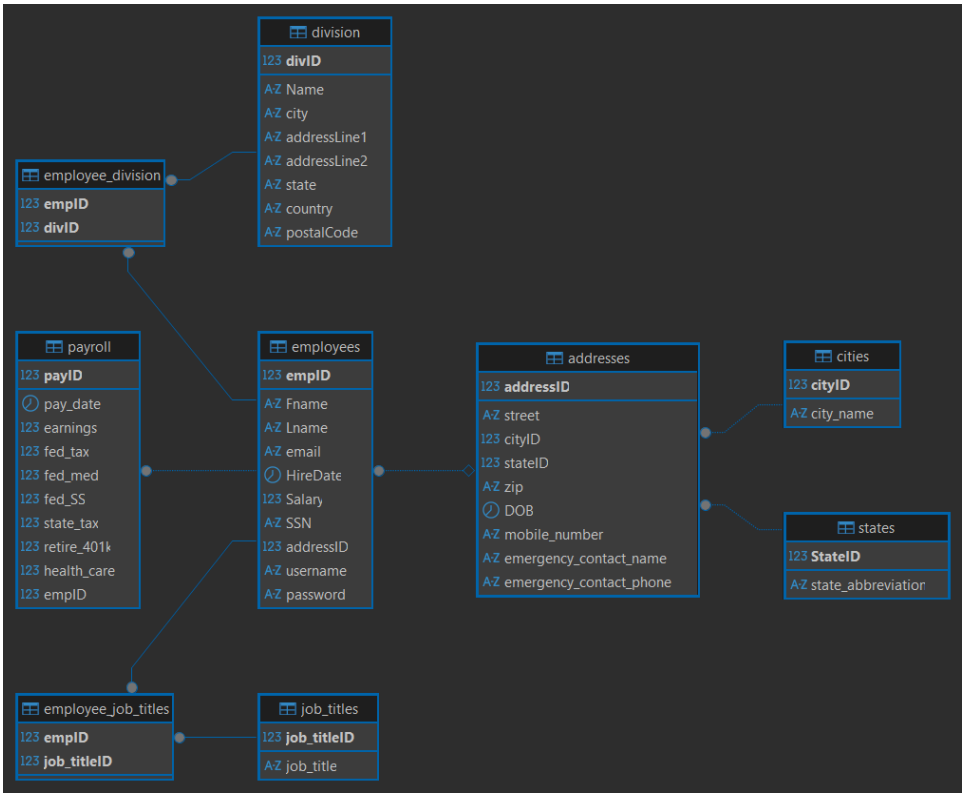
The Employee Management System for Company Z is designed to replace their current manual, insecure process (HR Admin using MySQL root account) with a secure, role-based Java application. The system serves two primary audiences:

1. HR Admin: Needs full CRUD access to employee data, salary management, and reporting capabilities.
2. General Employee: Need secure, read only access to view their own personal data and pay history.

The System must handle the company's planned growth from 100 to 300 employees over the next 24 months.

2. DATABASE SCHEMA DIAGRAM

[Use an updated schema diagram from dBeaver or SQL Workbench from #1 in deliverable items.]



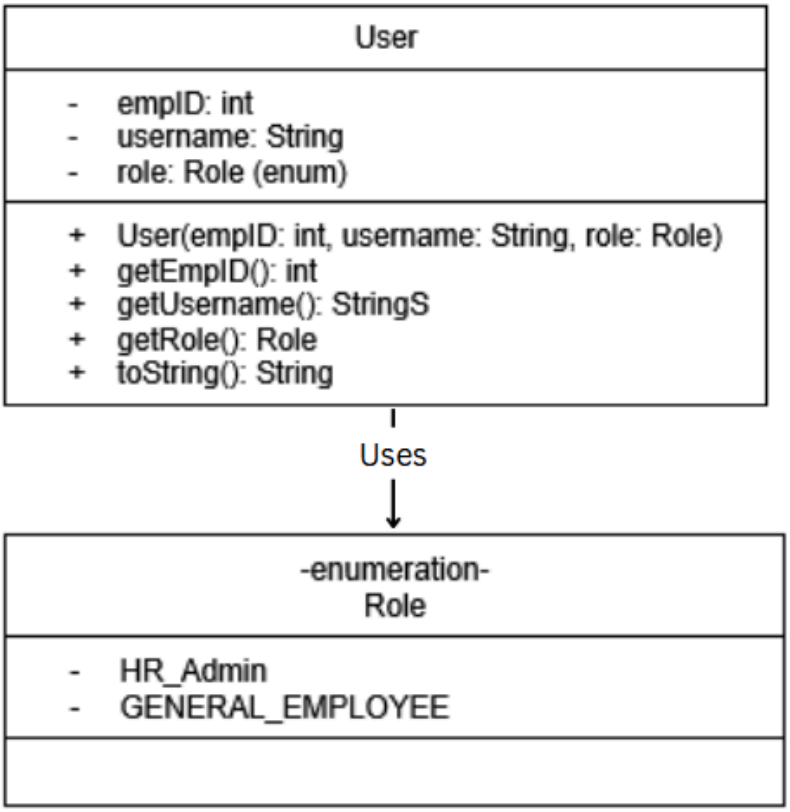
3. JAVA CLASS DIAGRAMS

[Classes, interfaces, abstract classes showing details and relationships]

Employee.java

Employee
- empID: Int - fname: String - lname: String - email: String - hireDate: Date - salary: double - SSN: String - addressID: int - username: String - password: String
+ Employee(empID, addressID, fname, lname, email, hireDate, SSN, salary, username, password) + getEmpID(): int + getFname(): String + getLname(): String + getEmail(): String + getHireDate(): Date + getSalary(): double + getSSN(): String + getAddressID(): int + getUsername(): String + getPassword(): String + setEmpID(empID: int): void + setFname(fname: String): void + setLname(lname: String): void + setEmail(email: String): void + setHireDate(hireDate: Date): void + setSalary(salary: double): void + setSSN(SSN: String): void + setAddressID(addressID: int): void + setUsername(username: String): void + setPassword(password: String): void + toString(): String

User.java



DataAccessLayer.java

DataAccessLayer
- Conn: Connection
+ connect(): void + disconnect(): void + findEmployeeById(empID: int): Employee + findEmployeeByName(name: String): List<Employee> + insertEmployee(emp: Employee): void + updateEmployee(empID: int, data: Employee): void + deleteEmployee(empID: int): void + updateSalaryByPercentage(percent: double, min: double, max: double): void + getPayHistory(empID: int): List<String> + getNewHiresByDateRange(start: Date, end: Date): List<Employee> + getMonthlyTotalPayByTitle(job_title_id: int, month: int, year: int): double + getMonthlyTotalPayByDivision(div_ID: int,

month: int, year: int): double + validateLogin(username: String, password: String): User + findOrCreateState(stateAbbr: String): int + findOrCreateCity(cityName: String): int + insertAddress(street: String, cityID: int, stateID: int, zip: String, dob: Date, mobile: String, emergName: String, emergPhone: String): int

AuthenticationController.java

AuthenticationController
- dal: DataAccessLayer - currentUser: User
+ AuthenticationController(dal: DataAccessLayer) + login(username: String, password: String): boolean + logout(): void + getCurrentUser(): User + getCurrentUserRole(): String + isLoggedIn(): boolean + isHRAdmin(): boolean + isGeneralEmployee(): boolean + main(args: String[]): void

SearchController.java

SearchController
- auth: AuthenticationController - dal: DataAccessLayer
+ SearchController(auth: AuthenticationController, dal: DataAccessLayer) + searchByEmpID(empID: int): Employee + searchByName(name: String): List<Employee> + searchByDOB(dob: Date): List<Employee> + searchBySSN(ssn: String): Employee + hideSensitiveData(emp: Employee): void + main(args: String[]): void

EmployeeManagementController.java

EmployeeManagementController
- auth: AuthenticationController - dal: DataAccessLayer
+ EmployeeManagementController(auth: AuthenticationController, dal: DataAccessLayer) + isHRAdmin(): boolean + addEmployee(emp: Employee): boolean + updateEmployee(empID: int, newData: Employee): boolean + updateSalaryByPercentage(percent: double, min: double, max: double): boolean + deleteEmployee(empID: int): boolean + main(args: String[]): void

ReportGenerator.java

ReportGenerator
- dal: DataAccessLayer - auth: AuthenticationController
+ ReportGenerator(dal: DataAccessLayer, auth: AuthenticationController) + getMyPayHistory(): List<String> + getPayHistory(empID: int): List<String> + getTotalPayByJobTitle(jobTitleId: int, month: int, year: int): double + getTotalPayByDivision(divID: int, month: int, year: int): double + getNewHiresByDateRange(start: Date, end: Date): List<Employee> + main(args: String[]): void

UserInterface.java

UserInterface
- auth: AuthenticationController - search: SearchController - empMgmt: EmployeeManagementController - reports: ReportGenerator - dal: DataAccessLayer - sc: Scanner
+ UserInterface(auth: AuthenticationController,

search: SearchController, empMgmt: EmployeeManagementController, reports: ReportGenerator, dal: DataAccessLayer) + start(): void + loginFlow(): void + employeeMenu(): void + hrMenu(): void + createEmployeeInput(): Employee + updateEmployeeFlow(): void + updateEmployeeFlow(empID: int): void + deleteEmployeeFlow(): void + viewPayHistoryMenu(): void + getIntInput(): int + getDoubleInput(): double

main.java

main
+ main(args: String[]): void

4. PROGRAMMING TASKS

[10 minimum from User Story; use 5 from #2 in deliverable items]

Employee.java (no dependencies)

Create Employee.java as a data container matching the employees table in the database.

Fields:

- empID (int, Primary Key)
- fname, lname, email (String)
- HireDate (Date)
- Salary (Double)
- SSN (String)
- addressID (int, foreign key)
- username, password (Added, String for authentication)

Additional tasks/requirements:

- Constructor with all fields
- Getters and setters for all fields
- toString() method for debugging

User.java (no dependencies)

Create a class with fields for logged in users. User inner enum for role.

Fields:
- empID (int)
- username (String)
- role (role enum)

Inner Enum:

```
public enum Role {
        HR_ADMIN,
        GENERAL_EMPLOYEE
}
```

DataAccessLayer.java (depends on Employee.java)

Implement the Data Access Layer (DAL) as the single point of contact with the MySQL database. All SQL queries are executed here. Converts database rows to Employee objects.

Components Involved:
- DataAccessLayer - manages all database operations
- EmployeeDatabase - external MySQL database
- Employee.java - data container for results
- User.java - for login validation

Input:
- Database connection parameters (url, user, password) from config.properties file
- Various query parameters depending on method/function

Process
- Load database credentials from config.properties file (with fallback to localhost)
- Establish MySQL connection using JDBC
- Execute CRUD operations using PreparedStatement
- Convert ResultSet rows to Employee objects
- Handle errors gracefully with try-catch blocks
- Close connection when done

Output:
- Success: Employee objects, lists of employees, pay history, or update counts
- Failure: null, empty list, or error message printed to console

Dependencies
- MySQL JDBC driver (mysql-connector-j-9.1.0.jar)
- config.properties file with db.url, db.user, db.password
- Employee.java and User.java model classes
- employees table with proper columns (empID, Fname, Lname, email, HireDate, Salary, SSN, addressID, username, password)

AuthenticationController.java (depends on AuthenticationController, User.java)

Implement the Authentication Controller to manage user login, sessions, and permission checks. This component serves as the single source of truth for the current user and their role throughout the application.

Components Involved:
- AuthenticationController - manages login/logout and session state
- DataAccessLayer - validates credentials against database
- User.java - data container for logged-in user information

Input:
- username (String) - user's email or login name
- password (String) - plain text password (for testing purposes)

Process:
- User submits credentials through UserInterface
- AuthenticationController calls DataAccessLayer.validateLogin(username, password)
- DataAccessLayer queries employees table for matching credentials
- If valid, create User object with empID, username, and role
- Store User object as currentUser (session)
- Return true/false indicating login success

Output:
- Success: User object created, session stored, returns true
- Failure: Returns false, error message displayed
- Role methods return appropriate role or null

Dependencies:
- DataAccessLayer must have validateLogin() method
- User.java must exist with Role enum
- Database must have employees table with username and password columns

SearchController.java (depends on AuthenticationController, DataAccessLayer)

Implement the Search Controller to process all employee search requests. Results should be formatted differently based on user role (HR Admin sees full data, General Employee sees limited data with sensitive information hidden).

Components Involved:
- SearchController - handles search logic
- AuthenticationController - provides current user role
- DataAccessLayer - retrieves data from database
- Employee.java - data container for results

Input:
- empID (int) - unique identifier for an employee
- name (String) - first or last name (partial matches allowed)
- dob (Date) - date of birth (stub method)

- ssn (String) - social security number (HR Admin only)

Process:
- Receive search request from UserInterface
- Call appropriate DataAccessLayer method to retrieve data
- If employee not found, print "No employee found with ID: [empID]"
- Check user role via AuthenticationController.isHRAdmin()
- If role is GENERAL_EMPLOYEE, hide sensitive data (SSN, salary, password)
- If role is HR Admin, return full employee data
- For SSN search, restrict access to HR Admin only

Output:
- Success: Employee object or list of employees returned
- Failure: null or empty list, error message printed

Dependencies:
- AuthenticationController must have isHRAdmin() method
- DataAccessLayer must have findEmployeeById() and findEmployeeByName() methods
- Employee.java must have setSSN(), setSalary(), setPassword() methods for hiding data

EmployeeManagementController.java (depends on AuthenticationController, DataAccessLayer)

Implement the Employee Management Controller to handle all CRUD operations (add, update, delete) for employee records. These operations should only be accessible to users with HR Admin role.

Components Involved:
- EmployeeManagementController - handles add/update/delete operations
- AuthenticationController - verifies user has HR Admin role
- DataAccessLayer - performs database operations
- Employee.java - data container for employee information

Input:
- Employee object containing employee data
- empID (int) - unique identifier for target employee
- percent (double), min (double), max (double) - for salary updates

Process:
- Receive request from UserInterface
- Check if current user has HR Admin role via AuthenticationController
- If not HR Admin, return false and display "Access denied"
- If HR Admin, call appropriate DataAccessLayer method
- Return true if operation completed

Output:
- Success: true, operation completed, confirmation message shown
- Failure: false, access denied message or error message

Dependencies:
- AuthenticationController must have isHRAdmin() method
- DataAccessLayer must have insertEmployee(), updateEmployee(), deleteEmployee(), updateSalaryByPercentage()
- Employee.java must be fully implemented

ReportGenerator.java (depends on AuthenticationController, DataAccessLayer)

Implement the Report Generator to generate pay statements, job title totals, division totals, and new hire reports. All reports except getMyPayHistory() require HR Admin privileges.

Components Involved:
- ReportGenerator - handles all report generation
- AuthenticationController - verifies user role for access control
- DataAccessLayer - retrieves data from database
- Employee.java - data container for employee information

Input:
- Various parameters depending on report type (empID, jobTitleId, divID, month, year, date range)

Process:
- Receive report request from UserInterface
- For getMyPayHistory(): uses current logged-in user's empID, no role check
- For all other reports: verify user has HR Admin role via AuthenticationController
- If not HR Admin, print "Access denied" and return empty list or 0.0
- If authorized, call appropriate DataAccessLayer method
- Return formatted results

Output:
- Success: List of pay history strings, list of employees, or double total
- Failure: Empty list, 0.0, or access denied message

Dependencies:
- AuthenticationController must have isHRAdmin() and getCurrentUser() methods
- DataAccessLayer must have getPayHistory(), getMonthlyTotalPayByTitle(), getMonthlyTotalPayByDivision(), getNewHiresByDateRange()
- Employee.java must be fully implemented

UserInterface.java (depends on ALL controllers)

Implement the User Interface as the main interaction layer for the Employee Management System. Handle all user input/output with different menus for HR Admin and General Employee roles.

Components Involved:
- UserInterface - handles all console-based user interaction
- AuthenticationController - for login/logout and role verification

- SearchController - for employee search operations
- EmployeeManagementController - for CRUD operations (HR Admin only)
- ReportGenerator - for generating all reports
- DataAccessLayer - for address lookup operations (cities, states)

Input:
- User menu selections (integers)
- Text input for search criteria, employee data, report parameters
- Authentication credentials (username, password)

Process:
- Display main menu (Login/Exit)
- After successful login, display role-appropriate menu
- For General Employee: search by ID/name, view own pay history, logout
- For HR Admin: full CRUD operations, salary updates, all reports, delete with confirmation
- Validate numeric inputs to prevent crashes
- Collect address information when adding new employees
- Call appropriate controller methods based on user selection
- Display results or error messages

Output:
- Success: Employee data, pay history, report results, confirmation messages
- Failure: Error messages, "Access denied" for unauthorized actions

Dependencies:
- All controllers must be fully implemented (AuthenticationController, SearchController, EmployeeManagementController, ReportGenerator)
- DataAccessLayer must have findOrCreateCity(), findOrCreateState(), insertAddress() methods
- Employee.java must be fully implemented

Main.java (depends on UserInterface)

Implement the main entry point for the Employee Management System. Create instances of all controllers and the DataAccessLayer, then start the UserInterface.

Components Involved:
- main - entry point class
- DataAccessLayer - database connection management
- AuthenticationController - user login/session management
- SearchController - employee search operations
- EmployeeManagementController - CRUD operations
- ReportGenerator - report generation
- UserInterface - console interaction

Input:
- None (command line arguments not used)

Process:

- Create DataAccessLayer instance and establish database connection
- Create AuthenticationController instance (depends on DAL)
- Create SearchController instance (depends on Auth and DAL)
- Create EmployeeManagementController instance (depends on Auth and DAL)
- Create ReportGenerator instance (depends on DAL and Auth)
- Create UserInterface instance (depends on all controllers and DAL)
- Call ui.start() to begin the application
- After UI exits, disconnect from database

Output:
- Success: Application starts, main menu displayed
- Failure: Database connection error message

Dependencies:
- All controllers must be fully implemented
- DataAccessLayer must have connect() and disconnect() methods
- UserInterface must have start() method

Database Setup and Sample Data Setup

Create the complete database schema with all required tables (employees, payroll, employee_job_titles, job_titles, employee_division, division, addresses, cities, states) with proper primary/foreign key connections. Populate the database with sample data for testing across the team using an online shared database (Aiven).

Components Involved:
- MySQL database (hosted on Aiven for team access)
- setup.sql - table creation script with foreign key constraints
- loadData.sql - sample data population script

Input:
- None (run scripts once on shared database)

Process:
- Disable foreign key checks before dropping tables
- Drop all existing tables in correct order (child tables first)
- Re-enable foreign key checks
- Create tables in order: states, cities, addresses, employees, job_titles, division, employee_job_titles, employee_division, payroll
- Define PRIMARY KEY constraints with AUTO_INCREMENT where applicable
- Define FOREIGN KEY constraints with ON DELETE CASCADE where appropriate
- Insert sample data: states (12 records), cities (16 records), addresses (12 records), employees (12 records), job titles (12 records), divisions (4 records), employee-job title mappings (12 records), employee-division mappings (12 records), payroll records (36 records - 3 months per employee)

Output:
- Success: All tables created, foreign keys verified, sample data inserted
- Failure: Error message if constraints fail or data integrity issues

Dependencies:

- MySQL server running (Aiven online database)
- User must have CREATE, INSERT, and ALTER privileges
- Config.properties file with database connection details

5. TEST CASES

[cleaned up from #2 in deliverable items]

Tests by file:

- Underlined tests are from deliverable #2 assignments. Included new tests we implemented in our files.

AuthenticationController.java

Test 1: Initial State (No User Logged In)

Action: Check Authentication state before any login attempt.

Expected Result: No user should be logged in; all role methods should return false/null

Pass: isLoggedIn() returns false, isHRAdmin() returns false, getCurrentUser() returns null

Fail: Any method returns true or non-null user before login

Test 2: Failed Login Attempt

Action: Attempt login with invalid username and password (wronguser@companyz.com / wrongpassword)

Expected Result: Login should fail; no session should be created

Pass: login() returns false, isLoggedIn() returns false

Fail: login() returns true or isLoggedIn() becomes true

Test 3: Successful Login (HR Admin Role)

Action: Login with HR Admin credentials (john.doe@companyz.com / password123)

Expected Result: Login should succeed; user should have HR_ADMIN role

Pass: login() returns true, getCurrentUserRole() returns "HR_ADMIN", isHRAdmin() returns true, isGeneralEmployee() returns false

Fail: login() returns false, wrong role returned, or role methods return incorrect values

Test 4: Logout

Action: Call logout() after successful login

Expected Result: Session should be cleared; no user should be logged in

Pass: isLoggedIn() returns false, getCurrentUser() returns null

Fail: isLoggedIn() still returns true or getCurrentUser() returns non-null

Test 5: Successful Login (General Employee Role)

Action: Login with General Employee credentials (jane.smith@companyz.com / password123)

Expected Result: Login should succeed; user should have GENERAL_EMPLOYEE role

Pass: login() returns true, getCurrentUserRole() returns "GENERAL_EMPLOYEE", isGeneralEmployee() returns true, isHRAdmin() returns false

Fail: login() returns false, wrong role returned, or role methods return incorrect values

Test 6: Role Checking Methods

Action: After successful login as General Employee, call role checking methods

Expected Result: Methods should return correct role values based on logged-in user

Pass: getCurrentUserRole() returns "GENERAL_EMPLOYEE", isHRAdmin() returns false, isGeneralEmployee() returns true

Fail: Any method returns incorrect role value

SearchController.java

Test 1: searchByEmpID as HR Admin

Action: HR Admin searches for existing empID (1)

Expected Result: Employee returned with full data (SSN visible, salary visible)

Pass: Non-null Employee, getSSN() returns non-null, getSalary() returns > 0

Fail: Employee null, sensitive data hidden, or wrong employee

Test 2: searchByEmpID as General Employee

Action: General Employee searches for existing empID (1)

Expected Result: Employee returned with sensitive data hidden (SSN null, salary 0, password null)

Pass: Non-null Employee, getSSN() returns null, getSalary() returns 0.0

Fail: Employee null or sensitive data still visible

Test 3: searchByEmpID for Non-existent Employee

Action: User searches for empID that does not exist (9999)

Expected Result: Error message printed, null returned

Pass: "No employee found with ID: 9999" message printed, null returned

Fail: Non-null employee returned or no error message

Test 4: searchByName as HR Admin

Action: HR Admin searches by name "doe"

Expected Result: List of employees with matching names, full data visible

Pass: Non-empty list, SSN and salary visible for each employee

Fail: Empty list when data exists or sensitive data hidden

Test 5: searchByName as General Employee

Action: General Employee searches by name "smith"

Expected Result: List of employees with matching names, sensitive data hidden

Pass: Non-empty list, SSN null, salary 0.0 for each employee

Fail: Empty list or sensitive data visible

Test 6: searchBySSN as HR Admin

Action: HR Admin searches by SSN (stub method)

Expected Result: Currently returns null (stub)

Pass: Returns null

Fail: Throws exception or returns non-null without implementation

Test 7: searchBySSN as General Employee

Action: General Employee attempts to search by SSN

Expected Result: Access denied, null returned

Pass: "Access denied: SSN search is restricted to HR Admins" message printed, null returned

Fail: Non-null returned or no access denied message

EmployeeManagementController.java

Test 1: addEmployee as General Employee (Access Denied)

Action: General Employee attempts to add new employee

Expected Result: Access denied, employee not added

Pass: addEmployee() returns false, "Access denied" message printed

Fail: addEmployee() returns true or employee added

Test 2: addEmployee as HR Admin

Action: HR Admin adds new employee with valid data

Expected Result: Employee inserted into database

Pass: addEmployee() returns true, "Employee inserted successfully" message printed

Fail: addEmployee() returns false or employee not added

Test 3: updateEmployee as General Employee (Access Denied)

Action: General Employee attempts to update existing employee

Expected Result: Access denied, employee not updated

Pass: updateEmployee() returns false, "Access denied" message printed

Fail: updateEmployee() returns true or employee updated

Test 4: updateEmployee as HR Admin

Action: HR Admin updates existing employee with valid data

Expected Result: Employee record updated

Pass: updateEmployee() returns true, "Employee updated successfully" message printed

Fail: updateEmployee() returns false or employee not updated

Test 5: deleteEmployee as General Employee (Access Denied)

Action: General Employee attempts to delete employee

Expected Result: Access denied, employee not deleted

Pass: deleteEmployee() returns false, "Access denied" message printed

Fail: deleteEmployee() returns true or employee deleted

Test 6: deleteEmployee as HR Admin (Non-existent ID)

Action: HR Admin attempts to delete employee with non-existent empID (9999)

Expected Result: Operation attempted safely, no actual deletion

Pass: deleteEmployee() returns true (authorized, operation attempted)

Fail: Exception thrown or unexpected behavior

Test 7: updateSalaryByPercentage as General Employee (Access Denied)

Action: General Employee attempts to update salaries by percentage

Expected Result: Access denied, salaries unchanged, returns false

Pass: updateSalaryByPercentage() returns false, "Access denied. HR Admin privileges required" message printed

Fail: updateSalaryByPercentage() returns true or salaries changed

Test 8: updateSalaryByPercentage with Negative Percentage (Validation Error)

Action: HR Admin attempts to update salaries with negative percentage (-5)

Expected Result: Validation error, salaries unchanged, returns false

Pass: updateSalaryByPercentage() returns false, "Please enter valid positive numerical input" message printed

Fail: updateSalaryByPercentage() returns true or salaries changed

Test 9: updateSalaryByPercentage as HR Admin with Valid Inputs

Action: HR Admin updates salaries by 5% for employees with salary less than 50000

Expected Result: Eligible employees receive salary increase, returns true

Pass: updateSalaryByPercentage() returns true, confirmation message with count printed

Fail: updateSalaryByPercentage() returns false or no salaries updated

---

ReportGenerator.java

Test 1: getMyPayHistory as HR Admin

Action: HR Admin requests their own pay history

Expected Result: List of pay stubs returned for empID 1

Pass: Non-empty list returned, records printed correctly

Fail: Empty list returned or exception thrown

Test 2: getMyPayHistory as General Employee

Action: General Employee requests their own pay history

Expected Result: List of pay stubs returned for empID 2 (no role restriction)

Pass: Non-empty list returned, records printed correctly

Fail: Empty list returned or access denied message

Test 3: getPayHistory as HR Admin

Action: HR Admin requests pay history for another employee (empID 2)

Expected Result: List of pay stubs returned for empID 2

Pass: Non-empty list returned, records printed correctly

Fail: Empty list returned or access denied

Test 4: getPayHistory as General Employee

Action: General Employee requests pay history for another employee (empID 1)

Expected Result: Access denied, empty list returned

Pass: "Access denied" message printed, empty list returned (size 0)

Fail: Non-empty list returned or no access denied message

Test 5: getTotalPayByJobTitle as HR Admin

Action: HR Admin requests total pay by job title for January 2024

Expected Result: Sum of earnings returned as double

Pass: Positive double value returned

Fail: 0.0 returned when data exists or exception thrown

Test 6: getTotalPayByJobTitle as General Employee

Action: General Employee requests total pay by job title

Expected Result: Access denied, 0.0 returned

Pass: "Access denied" message printed, 0.0 returned

Fail: Positive value returned or no access denied message

Test 7: getTotalPayByDivision as HR Admin

Action: HR Admin requests total pay by division for January 2024

Expected Result: Sum of earnings returned as double

Pass: Positive double value returned

Fail: 0.0 returned when data exists or exception thrown

Test 8: getTotalPayByDivision as General Employee

Action: General Employee requests total pay by division

Expected Result: Access denied, 0.0 returned

Pass: "Access denied" message printed, 0.0 returned

Fail: Positive value returned or no access denied message

Test 9: getNewHiresByDateRange as HR Admin

Action: HR Admin requests new hires between 2020-01-01 and 2025-12-31

Expected Result: List of employees hired within date range

Pass: Non-empty list returned with correct employees

Fail: Empty list when data exists or wrong employees returned

Test 10: getNewHiresByDateRange as General Employee

Action: General Employee requests new hires by date range

Expected Result: Access denied, empty list returned

Pass: "Access denied" message printed, empty list returned (size 0)

Fail: Non-empty list returned or no access denied message

---

6. SEQUENCE DIAGRAMS

[cleaned up from #3 in deliverable items]

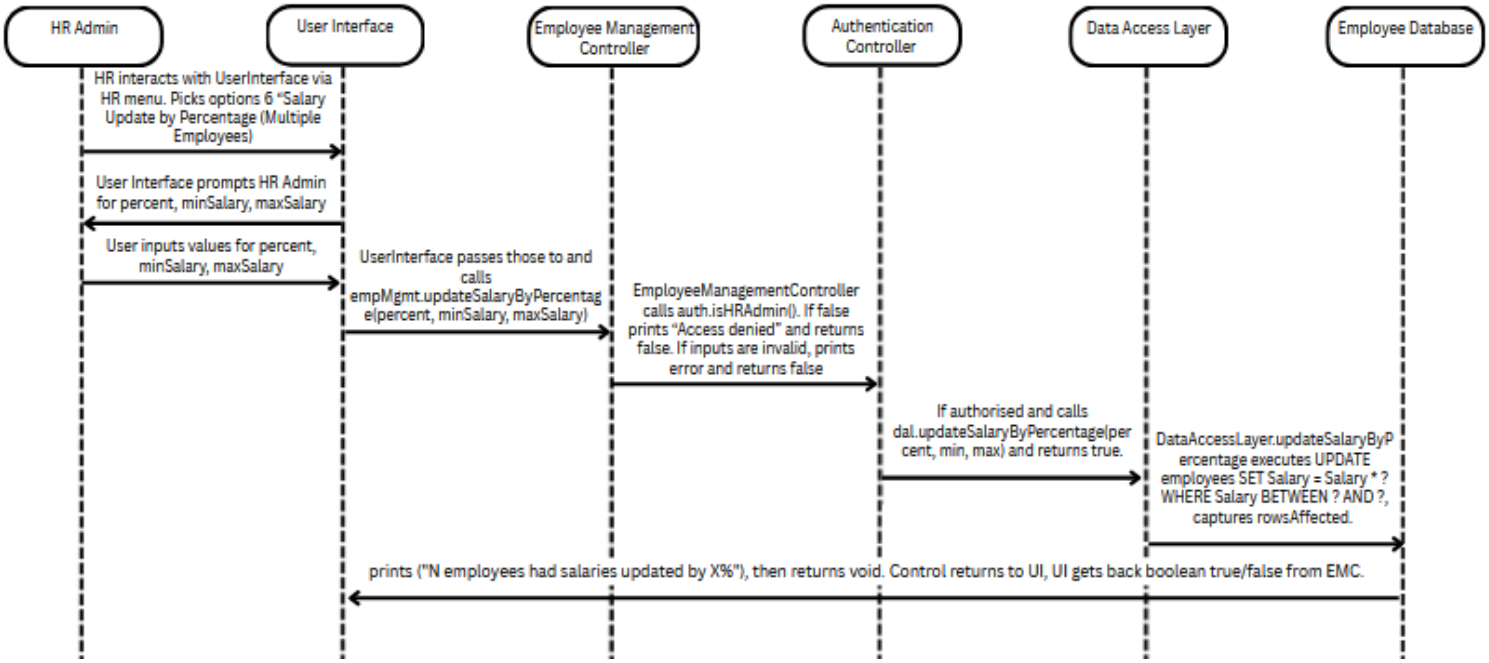
Updated Version based on whats implemented

a. Increase salary by % for all employees with their salaries in a specified range (what are inputs?)
NOTE: don't worry about if there are no changed salaries in the specified range, that would indicate no updates

Listing actions sequentially before drawing:

- HR interacts with UserInterface via HR menu. Picks options 6 "Salary Update by Percentage (Multiple Employees)
- User Interface prompts HR Admin for percent, minSalary, maxSalary

- User inputs values for percent, minSalary, maxSalary
- UserInterface passes those to and calls empMgmt.updateSalaryByPercentage(percent, minSalary, maxSalary)
- EmployeeManagementController calls auth.isHRAdmin(). If false prints “Access denied” and returns false. If inputs are invalid, prints error and returns false
- If authorised and calls dal.updateSalaryByPercentage(percent, min, max) and returns true.
- DataAccessLayer.updateSalaryByPercentage executes UPDATE employees SET Salary = Salary * ? WHERE Salary BETWEEN ? AND ?, captures rowsAffected.
- prints ("N employees had salaries updated by X%"), then returns void. Control returns to UI, UI gets back boolean true/false from EMC.

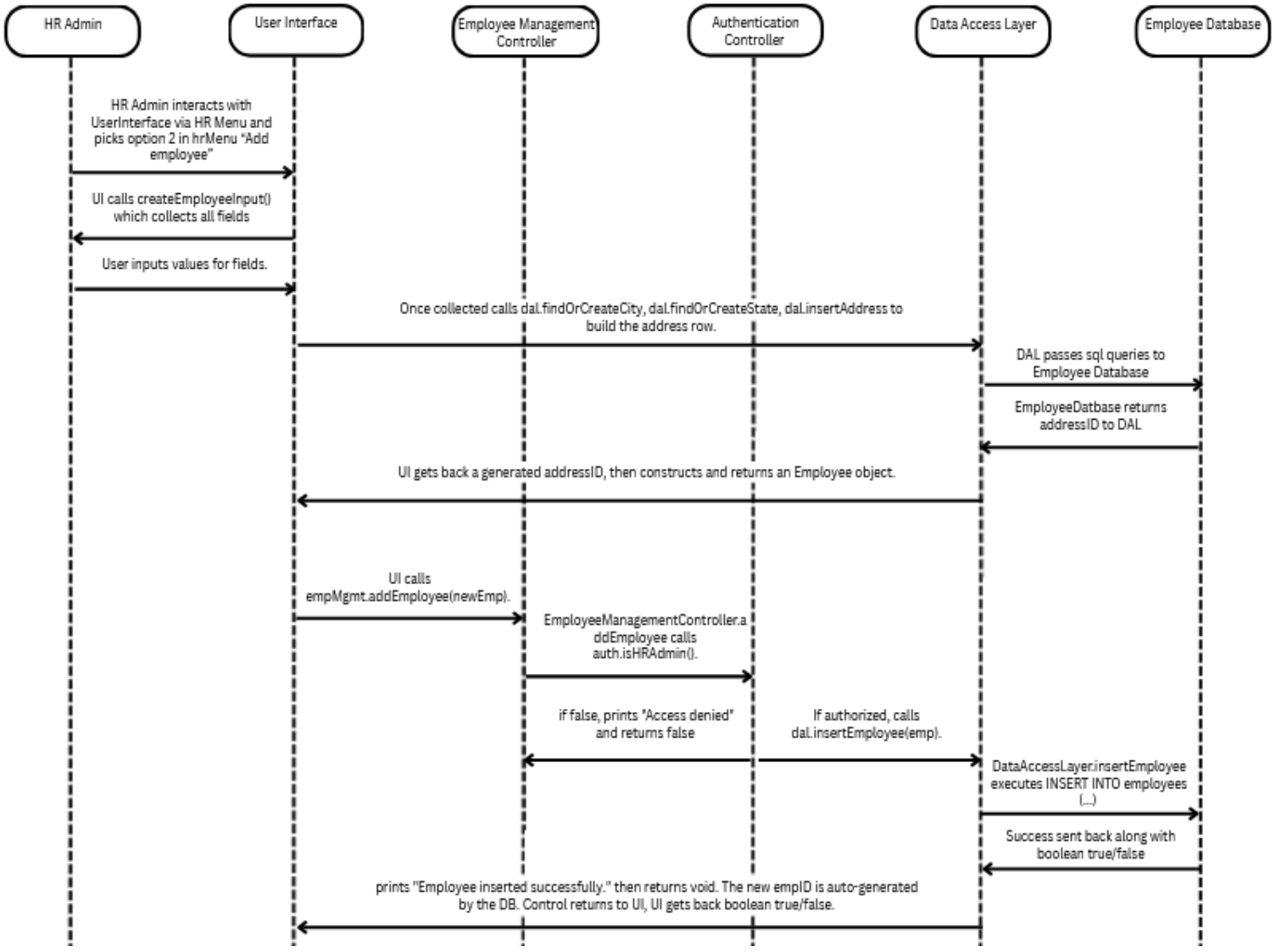


b. Add new employee to employeeData database (NO SEARCHING, empID is autogenerated for uniqueness)

Listing actions sequentially before drawing:

- HR Admin interacts with UserInterface via HR Menu and picks option 2 in hrMenu “Add employee”
- UI calls createEmployeeInput() which collects all fields
- User inputs values for fields.
- Once collected calls dal.findOrCreateCity, dal.findOrCreateState, dal.insertAddress to build the address row.
- DAL passes sql queries to Employee Database
- EmployeeDatbase returns addressID to DAL

- UI gets back a generated addressID, then constructs and returns an Employee object. UI calls empMgmt.addEmployee(newEmp).
- EmployeeManagementController.addEmployee calls auth.isHRAdmin(), if false, prints "Access denied" and returns false.
- If authorized, calls dal.insertEmployee(emp).
- DataAccessLayer.insertEmployee executes INSERT INTO employees (...)
- Success sent back along with boolean true/false
- prints "Employee inserted successfully." then returns void. The new empID is auto-generated by the DB. Control returns to UI, UI gets back boolean true/false.



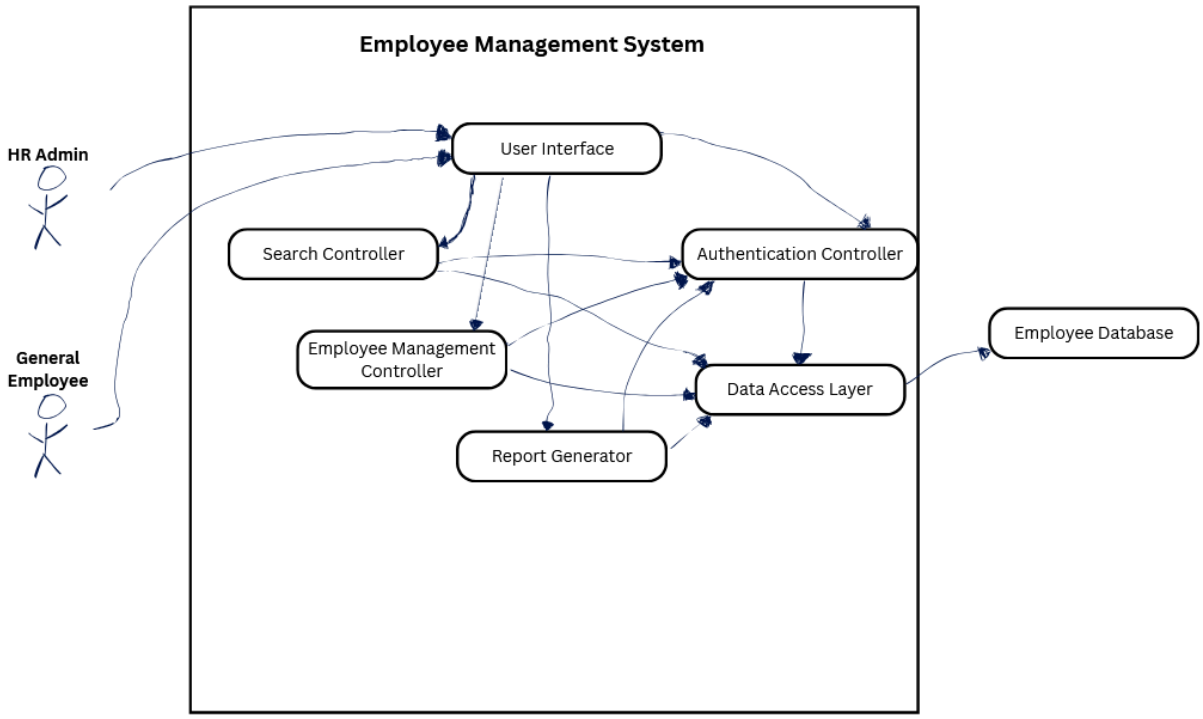
7. APPENDIX

7.1. Definitions and Acronyms

CRUD - Create, Read, Update, Delete
DAO - Data Access Object
DAL - Data Access Layer
DBMS - Database Management System
FK - Foreign Key
GUI - Graphical User Interface
HR- Human Resources
JDBC - Java Database Connectivity
PK - Primary Key
SDD - Software Design Document
SQL - Structured Query Language
UML - Unified Modeling Language
UX - User Experience

7.2. Additional UML diagrams

Original UML Case Diagram



7.3. Reflection

[give a short paragraph of your impression/opinion of the project: importance, understandability, instruction quality, clarity of directions, timeline.]

Note: If you review the SDD and provide feedback before submission, I will update this reflection accordingly. I've already communicated that I wanted everyone to review this document.

Software Design Document

Armando:

This project was both educational and challenging. The most valuable technical lesson I learned was the importance of a clear architecture before writing code. Our component design (controllers, DAO, model, view) proved essential because it allowed work to be divided, even if the division ended up being unequal in practice.

Team coordination was the biggest challenge. Several team members had limited availability due to work and other commitments, which made synchronous collaboration very difficult. As a result, I had to take on a disproportionate amount of the work, including the majority of the SDD, most of the planning and communication, and code reviews, adding test cases, missing imports, fields, and methods to verify everything was working properly. A couple of members, mainly Kadie and occasionally Luis, were helpful when available. Otherwise, progress depended heavily on individual initiative rather than true team collaboration.

If I could do something differently, I would establish clearer expectations for availability and contribution at the start of the project. I would also push harder for a shared online database earlier, that decision, once made, saved us significant integration headaches despite the coordination difficulties.

Despite the challenges, the system meets all core requirements. The experience reinforced that while good architecture makes implementation possible, clear communication and accountability make team projects successful. Next time, I would advocate for a more structured contribution tracking system from day one.