

Group 13

## **EmployeePulse**

Software Development Project

Name (s): Terry Andi Marala, Tobias Clements, Tytus Clements,  
Sabirin Mohamed, Syed Shuja Mukhtar, Sultan Abuhijleh

Semester: Spring 2025

Group Number: 13

Date: (04/28/2025)

## TABLE OF CONTENTS

<b>1.0</b>	<b><i>INTRODUCTION.....</i></b>	<b><i>3</i></b>
<b>1.1</b>	<b><i>Purpose.....</i></b>	<b><i>3</i></b>
<b>1.2</b>	<b><i>Scope.....</i></b>	<b><i>3</i></b>
<b>1.3</b>	<b><i>Overview.....</i></b>	<b><i>3</i></b>
<b>1.4</b>	<b><i>Reference Material.....</i></b>	<b><i>3</i></b>
<b>1.5</b>	<b><i>Definitions and Acronyms.....</i></b>	<b><i>4</i></b>
<b>2.0</b>	<b><i>SYSTEM OVERVIEW.....</i></b>	<b><i>4</i></b>
<b>3.0</b>	<b><i>SYSTEM ARCHITECTURE.....</i></b>	<b><i>4</i></b>
<b>3.1</b>	<b><i>Architectural Design.....</i></b>	<b><i>4</i></b>
<b>3.2</b>	<b><i>Decomposition Description.....</i></b>	<b><i>5</i></b>
<b>3.3</b>	<b><i>Design Rationale.....</i></b>	<b><i>5</i></b>
<b>4.0</b>	<b><i>DATA DESIGN.....</i></b>	<b><i>5</i></b>
<b>4.1</b>	<b><i>Data Description.....</i></b>	<b><i>5</i></b>
<b>4.2</b>	<b><i>Data Dictionary.....</i></b>	<b><i>5</i></b>
<b>5.0</b>	<b><i>COMPONENT DESIGN.....</i></b>	<b><i>5</i></b>
<b>6.0</b>	<b><i>HUMAN INTERFACE DESIGN.....</i></b>	<b><i>5</i></b>
<b>6.1</b>	<b><i>Overview of User Interface.....</i></b>	<b><i>5</i></b>
<b>6.2</b>	<b><i>Screen Images.....</i></b>	<b><i>6</i></b>
<b>6.3</b>	<b><i>Screen Objects and Actions.....</i></b>	<b><i>6</i></b>
<b>7.0</b>	<b><i>REQUIREMENTS MATRIX.....</i></b>	<b><i>6</i></b>
<b>8.0</b>	<b><i>APPENDICES.....</i></b>	<b><i>6</i></b>

## 1.0 INTRODUCTION

### 1.1 Purpose

This software design document (SWDD) describes the system design of *EmployeePulse*, a minimal employee management system for Company 2. It outlines the system's structure, data flow, and design rationale. The intended audience includes developers, project managers, and stakeholders involved in the project.

### 1.2 Scope

EmployeePulse aims to provide essential backend functionality to manage fewer than twenty full-time employees. The system facilitates employee information updates, salary adjustments, searches the database when imputed a SSN or Employeeid, and generates various payroll-related reports, without any user login/authorization.

### 1.3 Overview

This document includes the architectural design, data design, UI elements, component descriptions, and a mapping between requirements and system components. This is structured to present the high-level system overview, architecture, data design, component specifications, and a traceability matrix. It is based on a class-oriented Java implementation.

### 1.4 Reference Material

- Sudha Tushara Sadasivuni, In-class assignments, lecture slides

### 1.5 Definitions and Acronyms

- SSN: Social Security Number
- **EmpID**: Employee ID
- **JDBC**: Java Database Connectivity
- **SQL**: Structured Query Language
- **SWDD**: Software Design Description Document

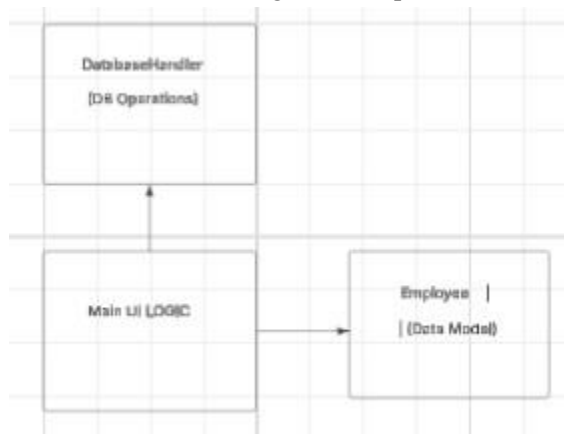
## 2.0 SYSTEM OVERVIEW

The *EmployeePulse* system is a console-based backend application for managing employee data. The data is accessed and modified through a class-based system in Java, connected to a MySQL database via JDBC. All logic is encapsulated within a few focused classes (**Employee**, **DatabaseHandler**, **Main**)

## 3.0 SYSTEM ARCHITECTURE

### 3.1 Architectural Design

[[[Develop a modular program structure and explain the relationships between the modules to achieve the complete functionality of the system. This is a high-level overview of how responsibilities of the system were partitioned and then assigned to subsystems. Identify each high level subsystem and the roles or responsibilities assigned to it. Describe how these subsystems collaborate in order to achieve the desired functionality. Don't go into too much detail about the individual subsystems. The main purpose is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together. Provide a diagram showing the major subsystems and data repositories and their interconnections. Describe the diagram if required.]]]



- **Employee:** Represents the data model for each employee.
- **DatabaseHandler:** Encapsulates all database-related operations.(database connections and queries)
- **Main:** Entry point of the system that calls other classes.(console-based user interface)

### 3.2 Decomposition Description

Provide a decomposition of the subsystems in the architectural design. Supplement with text as needed. You may choose to give a functional description or an objectoriented (OO) description. For a functional description, put a toplevel data flow diagram (DFD) and structural decomposition diagrams. For an OO description, put subsystem model, object diagrams, generalization hierarchy diagram(s) (if any), aggregation hierarchy diagram(s) (if any), interface specifications, and sequence diagrams here.

#### Main Class:

- Displays the menu.
- Captures user input.
- Calls appropriate methods in DatabaseHandler.

**DatabaseHandler Class:**

- Connects to MySQL using JDBC.
- Performs SQL operations like insert, update, select, update salary.

**Employee Class:**

- Encapsulates employee attributes like empId, name, ssn, salary, etc.
- Provides getters and setters.

**Top-Level Data Flow Diagram (DFD):**

*(Insert a diagram here, showing major processes and data flows. Example processes might be: "Manage Employee Records", "Search Employees", "Update Salaries", etc.)*

**Textual Description:**

At the top level, the system processes are:

- **Manage Employee Records:** Handles creation, modification, and deletion of employee information.
- **Search Employees:** Allows users to search employees by ID, Name, or SSN.
- **Update Salaries:** Updates employee salaries by applying percentage increases within a specified salary range.
- **Generate Pay Reports:** Generates reports showing employee payment information.

**Structural Decomposition Diagram:**

*(Insert a tree-like diagram showing decomposition, e.g., "Manage Employee Records" breaks down into "Add Employee", "Update Employee", "Delete Employee")*

**Narrative of Subsystems:**

- **Employee Management Subsystem**  
Manages all employee data operations, such as creating new entries, updating fields, and removing records.
- **Search Subsystem**  
Responsible for providing flexible search functionality using different fields (ID, Name, SSN).
- **Payroll Subsystem**  
Handles updating salaries and generating pay reports based on current salary data.

### 3.3 Design Rationale

[[[Discuss the rationale for selecting the architecture described in 3.1, including critical issues and trade/offs that were considered. You may discuss other architectures that were considered, provided that you explain why you didn't choose them.]]]

This architecture was selected to maintain a **clear separation of concerns**:

- **Employee** is a pure data model. It represents the employee as an entity. Using getters/Setters ensures that data and business logic remain separate.
- **DatabaseHandler** handles all DB logic.(inserts,updates,search, adjust salary, and show employees)
- **Main** initializes the system and provides user interaction hooks. Initializes the system, handles menu navigation, and captures user input to invoke methods from the database handler.

This makes the system modular, testable, and easy to maintain.

## 4.0 DATA DESIGN

### 4.1 Data Description

[[[Explain how the information domain of your system is transformed into data structures. Describe how the major data or system entities are stored, processed and organized. List any databases or data storage items.]]]

#### Transformation of Information Domain into Data Structures

The **Employee Management System** models real-world employee information as structured data using **object-oriented programming (OOP)** in Java and a **relational database (MySQL)**. Below is a breakdown of how the system's information domain is transformed into data structures, stored, and processed.

#### 1. Employee Entity

- Stored as a row in the employee table in MySQL.
- Each attribute (e.g., empid, name, ssn) maps to a column in the table.
- Processed via CRUD operations (INSERT, UPDATE, SELECT, etc.) through JDBC.

#### 2. Data Flow

- **Input**: User provides employee details via console (Java Scanner).
- **Storage**: Data is persisted in the database via DatabaseHandler.
- **Retrieval**: Queries fetch records for display/search.

### 4.2 Data Dictionary

[[[Alphabetically list the system entities or major data along with their types and descriptions. If you provided a functional description in Section 3.2, list all the functions and function parameters. If you provided an OO description, list the objects and its attributes, methods and method parameters.]]]

Field	Type	Description
division	String	Department or division name
empId	INT	Unique Identifier
jobTitle	String	Job role
name	String	Full Employee Name
payStatement	String	Text representation of pay history
SSN	String	Social Security Number (no dashes)
salary	Double	Current salary

## 5.0 COMPONENT DESIGN

In this section, we take a closer look at what each component does more systematically. If you gave a functional description in section 3.2, provide a summary of your algorithm for each function listed in 3.2 in procedural description language (PDL) or pseudocode. If you gave an OO description, summarize each object member function for all the objects listed in 3.2 in PDL or pseudocode. Describe any local data when necessary.

### **Class DatabaseHandler**

**Handles database connections and queries**

#### **connectToDatabase()**

- Load database driver
- Establish connection using URL, username, and password
- Return Connection object

#### **insertEmployee(employee)**

- Prepare SQL INSERT statement with employee details
- Execute statement
- Return success/failure message

#### **updateEmployee(employee)**

- Prepare SQL UPDATE statement with new employee details
- Execute update
- Return success/failure message

#### **deleteEmployee(employeeId)**

```
    Prepare SQL DELETE statement where employee_id = employeeId
    Execute delete
    Return success/failure message
searchEmployee(criteria)
    Build dynamic SQL SELECT query based on search criteria (name, SSN, ID)
    Execute query
    Return ResultSet
updateSalaryByRange(minSalary, maxSalary, percentage)
    Prepare SQL UPDATE query:
        Increase salary by percentage for employees between minSalary and maxSalary
    Execute update
    Return number of rows updated
showAllEmployees()
    Define SQL query: "SELECT * FROM employee"
    Open database connection
    Create a SQL statement
    Execute query and get result set
    WHILE there are more records in the result set DO
        Retrieve each employee's:
            - empid
            - name
            - ssn
            - job_title
            - division
            - salary
            - pay_statement
        Print the employee information in a formatted line
```

**Class Employee**  
**Represents an employee's data model.**

**getFullName()**  
Return firstName + " " + lastName

**updateSalary(newSalary)**  
Set salary = newSalary

**toString()**  
Return a formatted string with all employee field

**Class: MainUIController**  
**Manages the user interface actions and connects UI to database operations.**

**initialize()**  
Setup event listeners for buttons  
Populate initial data (if needed)



**dbhandle.AddEmployee()**

Collect form inputs  
Validate inputs  
Create Employee object  
Call DatabaseHandler.insertEmployee(employee)  
Show success/error message  
Clear form

**dbhandle.UpdateEmployee()**

Get selected employee from table  
Update fields based on user input  
Call DatabaseHandler.updateEmployee(employee)  
Show confirmation

**dbhandle.DeleteEmployee()**

Get selected employee  
Confirm deletion with user  
Call DatabaseHandler.deleteEmployee(employeeId)  
Refresh table

**dbhandle.SearchEmployee()**

Collect search term  
Call DatabaseHandler.searchEmployee(criteria)  
Display results in table

**dbhandle.UpdateSalaryByRange()**

Get salary range and percentage from form  
Call DatabaseHandler.updateSalaryByRange(minSalary, maxSalary, percentage)  
Show summary of update

**dbhandle.GenerateReport()**

Fetch all employees  
Display in report table  
Provide option to export if needed

## 6.0 HUMAN INTERFACE DESIGN

### 6.1 Overview of User Interface

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features, and the feedback information that will be displayed for the user.

The system provides a simple and intuitive Graphical User Interface (GUI) using JavaFX. The interface allows users to manage employee information easily without directly interacting with the database. Users can insert digits in-between 1-6 to search, add, update, or delete employee records and pay information through clearly labeled buttons, forms, and tables. If the the user adds or deletes any employee info a success message will be prompted.

**View All Employees**

- Action: The user selects an option to display all employees.
- Feedback:
  - The system retrieves and lists all employees, displaying key details such as ID, Name, SSN, Job Title, Division, Salary, and Pay Statement.
  - If no employees are found, a message like “No employees found” will be displayed.

**Search for an Employee**

- Action: The user can search for an employee by ID, Name, or SSN.
- Feedback:
  - If a matching employee is found, detailed information is displayed.
  - If no match is found, the user sees a message like “Employee not found.”

**Add a New Employee**

- Action: The user fills out a form (or inputs data) to create a new employee record.
- Feedback:
  - Upon successful insertion, a confirmation message like “Employee added successfully” is displayed.
  - If the insertion fails (e.g., duplicate ID or missing fields), an error message is shown.

**Update Existing Employee Information**

- Action: The user selects an employee (by ID, Name, or SSN) and modifies their information (e.g., salary, job title).
- Feedback:
  - After updating, the system confirms with a message like “Employee record updated.”
  - If no such employee exists, the user is informed accordingly.

**Delete an Employee**

- Action: The user specifies an employee ID to delete an employee record.
- Feedback:
  - On successful deletion, a message such as “Employee deleted successfully” appears.

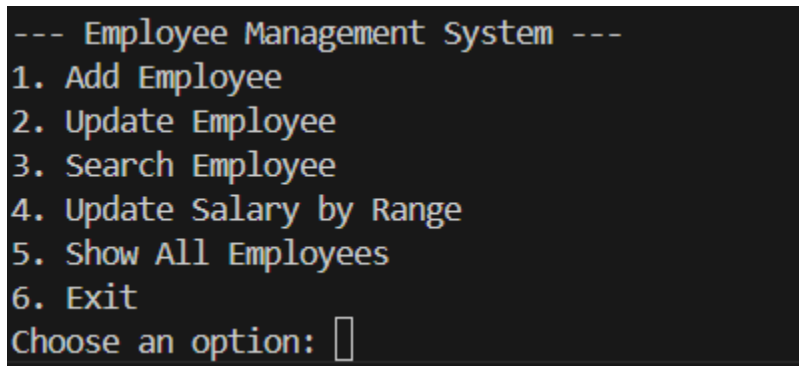
- If the ID does not exist, the system informs the user: “Employee ID not found.”

### Update Salaries by Percentage for a Salary Range

- Action: The user inputs a salary range and a percentage to increase salaries within that range.
- Feedback:
  - The system displays how many records were updated and confirms that the operation was successful.

## 6.2 Screen Images

Display screenshots showing the interface from the user’s perspective. These can be hand drawn or you can use an automated drawing tool. Just make them as accurate as possible. (Graph paper works well.)



```
--- Employee Management System ---
1. Add Employee
2. Update Employee
3. Search Employee
4. Update Salary by Range
5. Show All Employees
6. Exit
Choose an option: 
```

## 6.3 Screen Objects and Actions

A discussion of screen objects and actions associated with those objects.

1. MENU (--- Employee Management System ---)

Input Prompts: to fill in or search employee information: Prompts options 1-6

Output: Shows a success or failure message or employee information

2. Add Employee

Input Prompts: Enter ID, Name, SSN, Jobtitle, Division, Salary, PayStatment

Output Message: Employee Added successfully

3. Update Employee

Input Prompts: Employee ID to Update, New Name, New SSN, New Job Title, New

Division, New Salary, and New Pay Statement

Output Message: Employee Updated successfully

4. Search Employee

Input Prompts: SSN or Name or EmpID

Output Message: ID, Name, SSN, Title, Division, Salary, Pay Statement

5. Update Salary

Input Prompts: Enter percentage increase, Enter min salary, Enter max salary

Output Prompts:

Updated salaries for # employee(s).

6. Show All Employees

Input Prompts: N/A

Output Message: (Show all employees in table)

7. exit

Input Prompts: N/A

Output Message: Exiting

## 7.0 REQUIREMENTS MATRIX

[[[Provide a crossreference that traces components and data structures to the requirements in your software requirements specification (SWRS) document.

Use a tabular format to show which system components satisfy each of the functional requirements from the SWRS. Refer to the functional requirements by the numbers/codes that you gave them in the SWRS.

Functional Requirement	Component
Add an SSN column to the table	Employee class → <b>ssn</b> field
Search employee by name, SSN, or employee ID	<b>DatabaseHandler.searchEmployee()</b>
Update employee data	<b>DatabaseHandler.updateEmployee()</b>
Adjust the salary for the range	<b>DatabaseHandler.updateSalaryByRange()</b>
Show employee and pay information	<b>Employee.getPayStatement(), showAllEmployees()</b>
Add a new employee to the database	<b>Database.insertEmployee(Employee emp)</b>
ErrorHandler (or exception handling in)	<b>Exception handling (try-catch blocks) in DatabaseHandler methods (e.g., insertEmployee(), updateEmployee(), etc.)</b>

## 8.0 APPENDICES

*This section is optional.*

Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.