SWDD (CRN:89536)

Group 6

Project Title: Employee Management System

Software Design Document

Names: Trajuan Smith, Ryan Semancik, Deepak Govindarajan, Tarun Gadhiraju, Deborah Maignan, Aaron Mcleed

Class: CSC 3350 Software Development

Date: (12/02/2025)

Table of Contents

1. Introduction

1.1 Purpose

This Software Design Document (SWDD) describes the architecture, system design, and detailed data structure of the Employee Management System (EMS). The purpose of this document is to translate the software requirements into a comprehensive representation of software components and interfaces, serving as the primary reference for the implementation and coding phase

1.2 Scope

The Employee Management System is a desktop application designed to manage fundamental human resource (HR) data and operations within a small-sized enterprise.

- Goals: to provide an intuitive interface for HR personnel to maintain accurate, secure, and easily accessible employee records.
- Objectives: Implement core functionality, including creating, reading, updating, and deleting (CRUD) employee records.
- Benefits: Reduce manual data entry errors, centralize employee information, and facilitate

1.3 Overviews

Section 2.0 provides a general overview of the system's functionality. Section 3.0 details the high-level system architecture and component decomposition. 4.0 describes the data design and dictionary. Subsequent sections detail the component design, human interface, and the requirements matrix.

1.4 Reference Material

- Software Requirements Specification (SWRS) - [SWRS Document number TBD]
- IEEE Std 1016: IEEE Recommended Practice for Software Design Description.

1.5 Definitions and Acronyms

- CRUD: Create, Read, Update, Delete (The four basic operations of persistent storage).
- DAO: Data Access Object. A design pattern used to abstract and encapsulate all access to the data source.
- EMS: Employee Management System.
- JDBC: Java Database Connectivity. API used to connect and interact with the MySQL database
- MVC: Model-View-Controller. An architectural pattern separating the application into three interconnected parts.

2. System Overview

The Employee Management System (EMS) is a data-centric application that acts as a secure intermediary between a user (HR personnel) and the persistent data storage (MySQL Database)

The core functionality includes:

1. Employee Management: allowing authorized users to **add** new employees, **view** existing employee details, **update** employee information ()e.g., salary, department, contact info), and **delete** records.
2. User Authentication: A simple login mechanism to ensure only authorized users can access and modify data.
3. Reporting: Basic functionality to view a list of all current employees.

The system uses Java 21 as the primary development language, Maven for dependency management, and MySQL as the relational database backend.

3. System Architecture

3.1 Architectural Design

The EMS will utilize a three-tier Architecture to provide clear separation of concerns, flexibility, and maintainability. The system is partitioned into three distinct subsystems:

1. Presentation Tier (view): the User Interface (UI) that interacts with the user
2. Application Tier (Logic/Controller): contains the business logic, processing user request, and managing data flow.
3. Data Tier (DAO/Database): responsible for persistent storage, retrieval, and management of all application data.

The flow of control starts at the Presentation Tier, moves through the Application Tier for processing, and terminates at the Data Tier for storage or retrieval. The result follows the reverse path back to the user.

3.2 Decomposition Description

The system is decomposed using an Object-Oriented (OO) approach and is expanded to explicitly include the Service Layer and the critical database utility (DatabaseInit). The key subsystems and objects are defined below:

Subsystem/Object	Type	Role and Responsibilities
com.emp_mgmt.model	Subsystem	Holds all data structures (JAVA POJOs) representing real-world entities.
Employee, Payroll, Division, JobTitle	Object (Model)	Holds data attributes for all entities. Used to transfer data between tiers.
com.emp_mgm.db	Subsystem	Handles all low-level communication with the database.
DatabaseConnectionManager	Object (Utility.Singleton)	Manages the singleton instance of the database connection. Responsible for reading .env configuration, connecting via JDBC validation, and closing connections.
DatabaseInit	Object (Utility)	Initialized the database structure and seed data by reading schema.sql and sample-data.sql files, ensuring tables are created only if they do not already exist
EmployeeDAO, PayrollDAO, DivisionDAO, JobTitleDAO, EmployeeDivisionDAO, EmployeeJobTitleDAO	Object (DAO)	Implements atomic CRUD operations and specialized queries for their respective tables using JDBC/SQL.
com.employeemgmt.service	Subsystem	The Business Logic Layer. Responsible for coordinating actions across multiple DAOs to fulfill complex business processes
Employee Services	Object (Service)	Manages all employee-related business logic. Coordinates EmployeeDAO, DivisionDAO, and JobTitleDAO to ensure data consistency across multiple tables.
ReportService	Object (Service)	Manages complex data retrieval and calculations for reports. Coordinates all DAOs to gather comprehensive employee data.

com.employeemgmt.ui	Subsystem	Contains the User Interface components
ConsoleUI	Object (View/Controller)	The main application controller. Manages the primary menu and delegates user input to the specific sub-console objects (EmployeeConsole, ReportConsole).
EmployeeConsole	Object (View/Controller)	Handles user interaction for all Employee Management functions (e.g., displaying forms, handling CRUD input).
ReportConsole	Object (View/Controller)	Handles user interaction for all Reporting functions (e.g., displaying report options, printing results).
com.emp_mgmt.controller	Subsystem	Contains the primary business logic and controls the data flow between the View and DAO.
EmployeeController	Object (Controller)	Validates user input, calls the appropriate method across all DAOs, and enforces business rules.

3.3 Design Rationale

The three-tier architecture was selected for the Employee management system due to its significant advantages in maintainability, flexibility, and testability

Key advantages:

- Separation of Concerns: Each layer (presentation, application/controller, data/DAO) has a single, well-defined responsibility. This means that a change in the database is isolated to the DAO layer and does not impact the business logic or the user interface.
- Flexibility and Scalability: By abstracting data access through the DAO layer, the business logic is not tied to MySQL's specific implementation. If the team decided to migrate the database in the future, only the com.emp_mgmt.db package would need to be rewritten, leaving the controller logic intact.
- Testability: Each component can be rigorously tested in isolation.

Critical Issues and Trade-offs
While the Three-Tier approach is robust, a trade-off was considered:

- Increased code Overhead: This architecture required more classes. This adds initial complexity compared to a monolithic application, where all logic resides in one place
- Mitigation: The team accepted this overhead because the long-term benefits of maintainability and collaboration among team members significantly outweigh the initial cost

The architecture ensures that the project remains manageable, especially as new features are added later, which is why it was chosen over a simpler, less scalable design.

4. Data Design

4.1 Data Description

The information domain is transformed into a relational database model composed of five primary tables and two mapping tables to manage relationships.

- Core Entities: The system is centered on the employee entity, which tracks basic personnel data (name, SSN, email).
- One-to-Many Relationships: the payroll table is linked to the employees table, allowing one employee to have multiple historical payroll records
- Many-to-Many Relationships:
    - The relationship between employees and divisions is managed by the employee_division table, allowing an employee to belong to multiple divisions or a division to contain multiple employees.
    - The relationship between employees and job titles is managed by the employee_job_titles table, allowing employees to hold multiple job titles or a title to be held by multiple employees.

This relationship structure ensures data integrity and flexibility for future reporting needs.

4.2 Data Dictionary

The following Data Dictionary table displays the attributes that define the system entities:

Entity	Type	Attributes	Description
employees	Table (Primary)	employee_id (PK), first_name, last_name, SSN, email	Stores basic personnel information.
payroll	Table	payroll_id (PK), employee_id (FK), amount, pay_period_start, pay_period_end	Stores individual payroll disbursements and related dates.
division	Table	division_id (PK), name	Stores organization divisions/departments
job_titles	Table	job_title_id (PK), title	Stores available job titles within the organization.
employee_division	Table (Mapping)	employee_id (FK), division_id (FK)	Maps employees to divisions for many-to-many relationship.
employee_job_titles	Table (Mapping)	employee_id (FK), job_title_id (FK)	Maps employees to job titles for many-to-many relationship.

The following Data Dictionary table displays the functions/ methods:

DAO Class	Method Name	Corresponds to SQL Operation

EmployeeDAO	createEmployee(Employee emp)
	updateEmployee(Employee emp)
	deleteEmployee(int id)
	findById(int id)
	findAll()
	findBySsn(String ssn)
	searchByName(String name)
		Insert new employee
		Update employee
		Delete employee
		Find employee by ID
		Find all employees
		Find employee by SSN
		Search employee by name

PayrollDAO	CreatePayrollRecord(Payroll rec)
	updatePayrollRecord(Payroll rec)
	deletePayrollRecord(int id)
	findById(int id)
	findAll()
	findByEmployeeId(int id)
	
	UpdateSalaryByPercentage(float percent, float min, float max)
		Insert payroll record
		Update payroll record
		Delete payroll record
		Find payroll by ID
		Find all payrolls
		Find payrolls by employee ID
		Update salary by percentage

DivisionDAO	createDivision(Division div)
	updateDivision(Division div)
	deleteDivision(int id)
	findById(int id)
	findAll()
		Insert division
		Update division
		Delete division
		Find division by ID
		Find all divisions

JobTitleDAO	createJobTitle(JobTitle title)
	updateJobTitle(Job Title title)
	deleteJobTitle(int id)
	findById(int id)
	findAll()
		Insert job title
		Update job title
		Delete job title
		Find job title by ID
		Find all job titles

EmployeeDivisionDAO	assignDivision(int empId, int divId)
	
	removeDivision(int empId, int divId)
	
	getDivisionsByEmployee(int empId)
	getEmployeeByDivision(int divId)
		Insert employee-division relationship
		Delete employee-division relationship
		Find division by employee ID
		Find employee by division ID

EmployeeJobTitleDAO	assignJobTitle(int empId, int titleId)
	
	removeJobTitle(int empId, int titleId)
	
	getJobTitlesbyEmployeeId(int empId)
	getEmployeesByJobTitleId(int titleId)
		Insert employee-job title relationship
		Delete employee-job title relationship
		Find job title by employee ID
		Find employees by jobtitle ID

5.0 Component Design

The data access objects (DAOs) in the com.emp_mgmt.db package are the primary components responsible for implementing the provided SQL algorithms. Each DAO encapsulates the persistent logic for its respective entity.

DatabaseInit Component Summary

This component is critical for setting up the environment. Its logic includes robust file reading, transaction control (COMMIT/ROLLBACK), and fail-safe checks.

Function	Pseudocode Summary (PDL)	Local Data
initialize()	GET Connection; SET AutoCommit=FALSE; TRY: IF executeSchema() fails THEN ROLLBACK; RETURN FALSE; IF executeSampleData() fails THEN ROLLBACK; RETURN FALSE; COMMIT Transaction; RETURN TRUE; CATCH Exception: ROLLBACK and log error.	Connection object, dbManager.
ExecuteSchema(Connection conn	READ SQL content from src/db/schema.sql; SPLIT content into statements; FOR each statement: IF statement is CREATE TABLE AND tableExists check passes THEN continue (skip); EXECUTE statement; CATCH: Log non-critical errors (e.g., already exists) and continue; Throw fatal errors.	Statement object, SQL content string, schema.sql path.

EmployeeService Component Summary

This component demonstrates how the business rules are managed, abstracting this complexity from the UI

Function	Pseudosode Summary (PDL)	Local Data
hireNewEmployee(Employee Data data)	BEGIN Transaction; 1. CALL EmployeeDAO.createEmployee(data); 2. GET the new employeeID; 3. CALL DivisionDAO.assignEmployee(employeeID, divisionID); 4. CALL JobTitleDAO.assignEmployee(employeeID, jobTitleID); COMMIT Transaction; CATCH Exception: ROLLBACK.	EmployeeDAO, DivisionDAO, JobTitleDAO objects.

EmployeeDAO Component Summary

This component handles basic employee searching, demonstrating the data retrieval flow.

Function	Pseudocode Summary (PLD)	Local Data
searchByName(String name)	GET Connection; PREPARE SELECT * FROM employees WHERE first_name LIKE ? OR last_name LIKE?; SET parameters with %name%; EXECUTE QUERY; WHILE ResultSet HAS ROWS: MAP row columns to Employee Model object; ADD Employee object to List; RETURN List of Employee objects.	Prepared Statement, ResultSet, List<Employee>.

PayrollDAO Component Summary

This component includes the transaction logic required for bulk financial updates, ensuring atomicity.

Function	Pseudocode Summary (PLD)	Local Data
updateSalaryByPercentage (float percent, float min, float max)	GET Connection; SET AutoCommit=FALSE; TRY: PREPARE UPDATE payroll SET amount = amount * (1 + ? / 100) WHERE amount BETWEEN ? AND ?; SET parameters (percent, min, max); EXECUTE UPDATE; COMMIT Transaction; CATCH SQLException: ROLLBACK Transaction; LOG critical error; THROW exception.	Connection object, Prepared Statement

6.0 Human Interface Design

6.1 Overview of User Interface

The Employee Management System (EMS) utilizes a common-line interface (CLI), implemented through the com.employeemgmt.ui.ConsoleUI package. The interface is menu-driven, providing a straightforward, step-by-step experience for the user (HR personnel).

The user's interaction is based on selecting numerical options from the main menu, which then navigates to sub-menus for specific tasks

System Flow from the User's Perspective:

1. Start: The application begins by invoking the DatabaseInit process.
2. Main Menu: Once initialized, the user is presented with the main menu (e.g., [1] Employee Management, [2] Reports, [3] Exit).
3. Task Selection: The user selects a task, which launches the appropriate sub-console (EmployeeConsole or ReportConsole).
4. Data Interaction: Within a sub-console, the user is prompted for necessary input (e.g., "Enter Employee Name," "Enter Employee ID to update").
5. Feedback: the system provides immediate, text-based feedback on the success or failure of an operation, including printing errors (e.g., "ERROR: Employee ID not found") or confirmation messages (e.g., "Employee record created successfully").

6.2 Screen images

Since the interface is a console, the screen image will be mockups of the terminal output:

Screen Image 1: Main Menu

```
============================================
EMPLOYEE MANAGEMENT SYSTEM (EMS) - MAIN MENU
============================================
1. Employee Management (Add, View, Update, Delete)
2. Payroll and Reporting
3. Exit Application
Enter your choice (1-3): _
```

Screen Image 2: Employee Management Sub-Menu

```
============================================
EMPLOYEE MANAGEMENT - CRUD OPERATIONS
============================================
1. Add New Employee
2. View Employee Details (by ID or SSN)
3. Update Employee Information
4. Delete Employee Record
5. Back to Main Menu
Enter your choice (1-5): _
```

6.3 Screen Objects and Actions

The interface uses basic text and numbers of interaction

Screen Object	Type	Associated Action
Menu Options (e.g., "1. Add...")	Numbered Text Input	Selecting the number calls the corresponding method in the ConsoleUI or sub-consoles, triggering a business process in the Service layer.
Text Prompt (e.g., "Enter Name:")	String Input	The Scanner object captures the user input, which is then passed to the Service layer for validation and storage.
Output Messages	Text Output	Displayed after an action to confirm success (Employee added successfully) or report failure (Error: Database connection lost).

7.0 Requirements Matrix

The Requirements matrix provides a cross-reference between the project's Functional Requirements (FRs) and the specific design components that satisfy them.

The following table uses placeholder functional requirement IDs and descriptions based on the established system scope:

FR ID	Functional Requirement Description	Satified By Component(s)
FR-1.0	The system SHALL allow the creation, retrieval, update, and deletion (CRUD) of basic employee records (Name, SSN, Email)	EmployeeDAO, EmployeeService, EmployeeConsole
FR-2.0	The system SHALL allow the definition and association of employees with Divisions and Job Titles	DivisionDAO, JobTitleDAO, EmployeeDivisionDAO, EmployeeJobTitleDAO, EmployeeService

FR-3.0	The system SHALL be initialized automatically on startup by creating the necessary database schema and loading sample data	DatabaseInit, DatabaseConnectionManager
FR-4.0	The system Shall be able to calculate and report a list of all employees in a specific Division.	ReportService, DivisionDAO, EmployeeDivisionDAO, ReportConsole
FR-5.0	The user SHALL interact with the application through a menu-driven console interface	ConsoleUI, EmployeeConsole, ReportConsole