**CSE Department – Faculty of Engineering - MSA**

**Spring 2025**

**GSE122 GSE122i COM265 PROGRAMMING 2**

**Course Project**

**Course Instructor:Dr. Ahmed El Anany**

**Due Date 17/MAY/2025 11:59 PM on E-learning**

**Discussion inside lecture 18/May till 23/May inside lab as per lab slot**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name** | **Jana Abdelazim** | **Student ID** | **Fill ID here** |
| **Student Name** | **Sarah Reda** | **Student ID** | **244469** |
| **Student Name** | **Salma Amr** | **Student ID** | **Fill ID here** |
| **Student Name** | **Habiba Wahid** | **Student ID** | **Fill ID here** |
| **Student Name** | **Fatma Alaa** | **Student ID** | **Fill ID here** |
| **TA Name** | **Eng. Dina Magdy**  **Eng. Gehad Ehab**  **Eng. Mohamed Khaled**  **Eng. Hussien Mostafa** | **Grade: /** | |

**Project Title**

**Table of Contents**

[**Project Overview 3**](#_1qiv9zobnqqy)

[Objectives 3](#_kxya5mh10gqo)

[Roles and Responsibilities 4](#_4hrs94c0szki)

[Algorithm and external libraries 5](#_lbxclr1u8qm6)

[GUI and Database Usage 6](#_ns54jo9tktz)

[**Code explaining 7**](#_sk6m5j3846ai)

[**Output and results 8**](#_m9h15o464mtl)

[**GitHub(optional) 9**](#_u0qg68ra51oo)

[**References 10**](#_1lsyo46b7om8)

# **Project Overview:**

The **Student Management System** is a Java-based desktop application that helps manage student records, including their ID, name, subject marks, and grades. It uses a simple GUI built with Java Swing and stores data in a MySQL database. The system allows adding, searching, modifying, and deleting student records while ensuring data is saved for future use.

## **Objectives****:**

The main objectives of the Student Management System project are:

1. **Record Management**: To create, update, and manage student data including name, ID, subject-wise marks, and grades.
2. **Grade Calculation**: Automatically compute the student’s grade based on their average marks.
3. **Data Storage and Retrieval**: Store student records in a MySQL database and retrieve them when needed.
4. **User Interaction**: Provide an easy-to-use graphical interface so users can interact with the system without needing to access the database directly.
5. **Modular Design**: Ensure the system is modular by separating concerns into different classes (e.g., data storage, search, UI, etc.) for better maintainability and scalability.
6. **Persistence**: Maintain the consistency of data across sessions by using database operations like insert, update, delete, and select.

## 

## 

## 

## **Roles and Responsibilities**

This section describes the roles of each team member.

## 

## 

## **Algorithm and external libraries:**

1. **Algorithm Used:**

**Here are the key ones:**

* **Grade Calculation Algorithm (in Student.calculateGrade()):**

This method calculates a student's grade based on the average of all subject marks.

-**Algorithm Steps:**

1. Sum all subject marks.
2. Calculate average by dividing the total by the number of subjects.
3. Assign a grade based on the average:
   * >= 90 → A
   * >= 80 → B
   * >= 70 → C
   * >= 60 → D
   * < 60 → F

**➡️ This is a simple conditional grading algorithm.**

* **Search Algorithm (in SearchEngine.searchById() and searchByName()):**

Both methods use Java Streams to filter students by ID or name.

-**Algorithm Steps:**

1. Loop through the student list.
2. Compare each student's ID or name.
3. Return the first match (or null if none).

**➡️ This is a linear search algorithm, because it checks items one by one.**

* **Database CRUD Operations (in DatabaseHandler):**

These methods manage data with SQL operations:

* **Create**: CREATE TABLE IF NOT EXISTS
* **Read**: SELECT \* FROM students
* **Update**: UPDATE students SET ...
* **Delete**: DELETE FROM students WHERE id=?

**➡️ These are standard database interaction patterns — CRUD algorithms.**

* **GUI Menu Navigation (in StudentGUI.displayMainMenu()):**

A loop continuously displays a menu and takes user input.

**Algorithm Steps:**

1. Show options.
2. Wait for user to select an action.
3. Call the appropriate method.
4. Loop until "Exit" is chosen.

**➡️ This is a menu-driven algorithm.**

1. **External Libraries:**

**Java's built-in libraries**:

* **javax.swing.\***
* For GUI (Graphical User Interface) components like JOptionPane.
* Used in: StudentGUI
* **java.sql.\***
* For connecting to and interacting with a **MySQL** database.
* Used in: DatabaseHandler
* **java.util.\***
* For working with List, Map, ArrayList, HashMap, etc.
* Used in: Student, SearchEngine, StudentManagementSystem
* **java.util.stream.\* (implicitly used via .stream())**
* For functional-style data processing like filtering (e.g., students.stream().filter(...))
* Used in: SearchEngine

## **GUI and Database Usage**

**This section describes the GUI and Database Usage in the project .**

**1. Graphical User Interface (GUI):**

* **Technology Used:**
* **Java Swing** (javax.swing.\*)
* Mainly uses JOptionPane for simple input/output dialogs.
* **GUI Features:**

The GUI acts as the **front-end interface** for interacting with the system. It allows users to:

1. **Add a Student**
2. **Search for a Student**
3. **Modify a Student**
4. **Generate a Mark Sheet**
5. **Delete a Student**
6. **Exit the System**

* **How it Works:**
* The GUI starts from StudentGUI.main() using SwingUtilities.invokeLater.
* It calls displayMainMenu() to show a menu with options using JOptionPane.showInputDialog.
* **Menu Flow:**

pgsql

CopyEdit

+------------------------------------------+

| Student Management System |

+------------------------------------------+

| [1] Add Student |

| [2] Search Student |

| [3] Modify Student |

| [4] Generate Mark Sheet |

| [5] Delete Student |

| [6] Exit |

+------------------------------------------+

When a user selects an option, the corresponding method is called to perform the action using dialog prompts for data entry or display.

* **GUI Layout:**

Since it's built using JOptionPane, the layout is **dialog-based** and **non-custom**, meaning it doesn't use a visual panel-based layout like JFrame or JPanel.

Each action leads to a **dialog pop-up**, such as:

**Add Student Example:**

1. Input Dialog: "Enter Student ID"
2. Input Dialog: "Enter Student Name"

**2. Database Usage:**

* **Technology Used:**
* **MySQL** database
* **JDBC (Java Database Connectivity)** via java.sql.\*
* **Purpose of Database:**

The database stores **student records** persistently, so that data is not lost between runs.

* **Tables and Schema:**
* **Table Name: students**

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| id | VARCHAR(50) | Primary Key - Student ID |
| name | VARCHAR(100) | Student's full name |
| marks | TEXT | Subject-wise marks in plain text |
| grade | VARCHAR(5) | Student's final grade |

**SQL used to create the table:**

sql

CopyEdit

CREATE TABLE IF NOT EXISTS students (

id VARCHAR(50) PRIMARY KEY,

name VARCHAR(100),

marks TEXT,

grade VARCHAR(5)

);

* **How Database is Used:**
* **Create Table:**

When DatabaseHandler is instantiated, it ensures the table students exists.

* **Insert Data (saveDataBase()):**

Saves a new student record into the table.

* **Read Data (readDataBase()):**

Fetches all records from the database into Student objects. However, it **does not parse marks** from text back into the Map, so this could be a limitation.

* **Update Data (updateDatabase()):**

Updates name, marks, and grade for a student identified by their ID.

* **Delete Record (deleteRecord()):**

Removes the student entry with the given ID.

* **Example Database Entry (marks stored as text):**

| **id** | **name** | **marks** | **grade** |
| --- | --- | --- | --- |
| S001 | Alice Green | Math: 90, English: 85, Science: 92, | A |

## 

# **Code Explaining:**

**1. Student:**

**Purpose:** Represents a student with their ID, name, marks, and grade.

**Key Functions:**

* Stores each subject and its corresponding marks in a HashMap.
* Calculates average marks and assigns a grade (A to F).
* Converts the marks map into a string for display or saving to the database.

**Note:** The grade is only calculated when calculateGrade() is called.

**2. MarkSheet:**

**Purpose:** Responsible for generating and displaying a student's mark sheet.

**How it Works:**

* Takes a Student object.
* Prints the student's name, ID, subject-wise marks, and grade.

This class separates presentation (mark sheet output) from data logic (which is in Student).

**3. SearchEngine:**

**Purpose:** Provides methods to search for students in a list by ID or name.

**How it Works:**

* Uses Java Streams to filter the list of students.
* Returns the first matching student or null if not found.

This class encapsulates all search functionality, improving modularity and reusability.

**4. DatabaseHandler:**

**Purpose:** Manages database interaction using JDBC.

**Key Functions:**

* Creates a table students if it doesn't already exist.
* Saves a student record using SQL INSERT.
* Reads all students using SELECT.
* Updates records using UPDATE.
* Deletes a student using DELETE.

**Important Design Notes:**

* Marks are stored as plain text ("Math: 90, English: 85"), which limits how they can be queried.
* When reading from the database, marks are **not parsed back** into the HashMap, which can be improved.

**5. StudentManagementSystem:**

**Purpose:** Core controller class that connects all logic together.

**Responsibilities:**

* Maintains an in-memory list of students.
* Adds, searches, modifies, and deletes student records.
* Coordinates between the database handler and the GUI/search engine.
* Manages business logic like saving/updating to the database and updating the local list.

It's the **backbone** of the system, acting as a **service layer** between GUI/database and business logic.

**6. StudentGUI:**

**Purpose:** Provides a simple, dialog-based GUI using JOptionPane.

**GUI Layout:**

* **Main Menu:** A selection dialog with six options:
  + Add Student
  + Search Student
  + Modify Student
  + Generate Mark Sheet
  + Delete Student
  + Exit

**How It Works:**

* Each option triggers a series of input dialogs.
  + Example: "Add Student" asks for ID and name.
* Calls corresponding methods in StudentManagementSystem.

**Limitations:**

* No visual layout (no JFrame, no panels or tables).
* Input is minimal (only ID and name, no subject/marks input from the GUI).
* Output is shown via System.out or message dialogs.

**GUI Design Summary:**

**Technology:** Java Swing (JOptionPane)

**Type:** Text-based dialog pop-ups

**Flow:**

* Main menu dialog → user selects an action
* Input dialogs → get data from the user
* Output → shown via console or alert dialogs

This is a **simple user interface**, suitable for small-scale desktop apps but limited in interactivity and visuals.

**Database Design Summary:**

**Technology:** MySQL, accessed via JDBC (java.sql.\*)

**Table:** students

| **Column** | **Type** | **Description** |
| --- | --- | --- |
| id | VARCHAR(50) | Unique Student ID (Primary Key) |
| name | VARCHAR(100) | Full Name of the Student |
| marks | TEXT | String format of subject marks |
| grade | VARCHAR(5) | Final grade based on average |

**Limitations:**

* marks stored as text makes it hard to perform subject-wise queries or analytics.
* No relational structure for subjects.

# 

# Output and results

Must include the screenshots of the running program for every case with detailed explanation.

# 

# GitHub(optional)

Include link of github repo containing your project code and report and add screenshot of repo with commit logs

# References