# STUDENT MARKS MANAGEMENT SYSTEM

Course Project Report

## 1. Cover Page

Title: Student Marks Management System

Student Name Dev Vishwakarma

Registration Number: 25BAI10448

Faculty: RAMRAJ

Institution: VIT Bhopal University

## 2. Introduction

Managing student marks is a common task in schools and colleges. In many cases, marks are still written in notebooks or stored in loose spreadsheets. This makes it difficult to quickly update marks, search for a particular student or calculate the overall performance of the class.

This project aims to build a small Python console application that can handle these basic tasks in a simple and structured way. The system allows the user to add, update, delete, search and view student records. For each student, it calculates the total marks, percentage and grade automatically based on the entered marks.

The project mainly focuses on using core Python concepts such as variables, lists, dictionaries, conditional statements, loops and functions. It also shows how to split a program into multiple modules to make the code cleaner and easier to maintain.

## 3. Problem Statement

Manual maintenance of student marks is time-consuming and error-prone. Updating marks or finding the topper often involves going through multiple pages or files. There is no quick way to get basic statistics like class average or highest score without doing calculations by hand.

To address this, the project provides a command-line based “Student Marks Management System” which stores basic details for each student (roll number, name and marks in three subjects) in memory. The system offers options to manage the records and generate simple reports.

## 4. Functional Requirements

The main functional requirements of the system are:

1. **FR1 – Add Student**

   - The user can add a new student by entering roll number, name and marks in three subjects.

   - The system should calculate total marks, percentage and grade automatically.

2. **FR2 – Update Student Marks**

   - The user can select an existing student using their roll number and update the marks.

   - After updating, the total, percentage and grade should be recalculated.

3. **FR3 – Delete Student**

   - The user can delete a student record using the roll number.

- The record should be removed from the in-memory list.

4. **FR4 – View All Students**

- The system should display all student records in a clear format, showing roll number, name, individual subject marks, total, percentage and grade.

5. **FR5 – Search Student by Roll Number**

- The user can search for a single student using roll number and view full details if the record exists.

6. **FR6 – View Class Statistics**

- The system should show basic statistics such as:
  - Total number of students
  - Average percentage of the class
  - Topper’s roll number, name and percentage

These functional requirements are grouped into three main modules: **Student Management**, **Marks Processing** and **Reports & Statistics**.

## 5. Non-Functional Requirements

1. **Usability**

- The system should provide a simple text-based menu so that any user with basic computer knowledge can operate it without training.

2. **Performance**

- Operations like add, update, delete and search should give results immediately for typical class sizes (10–100 students), as data is kept in memory.

3. **Reliability and Error Handling**

  - The program should handle invalid inputs (such as non-numeric marks) and display proper error messages instead of crashing.

  - If a roll number does not exist during search, update or delete, the system should clearly inform the user.

4. **Maintainability**

  - The code should be modular, with separate files for operations, reports and data storage. This makes future changes easier.

5. **Scalability (Basic Level)**

  - Although the current implementation uses in-memory storage, the design should be simple to extend later to use files or a database.

## 6. System Architecture

The system follows a simple modular architecture using four main Python files:

- **main.py** – Handles the main menu, user input and overall program flow.

- **operations.py** – Contains functions to add, update and delete student records.

- **reports.py** – Contains functions for viewing all students, searching by roll number and computing class statistics.

- **data_store.py** – Stores the global `students` list that holds all student records in memory.

High-Level Architecture Description

User → interacts with the menu in **main.py**.

Based on the option selected, **main.py** calls functions from **operations.py** or **reports.py**.

Both these modules read and modify the `students` list defined in **data_store.py**.

(Architecture diagram can show User → main.py → {operations.py, reports.py} → data_store.py.)

## 7. Design Diagrams

### 7.1 Use Case Diagram

Actor: User (teacher / class representative).

Use Cases:

- Add Student

- Update Student

- Delete Student

- View All Students

- Search Student

- View Class Statistics

The use case diagram shows the User connected to each of these use cases.

### 7.2 Workflow / Activity Diagram

A possible workflow:

1. Start program

2. Display main menu

3. User chooses an option

4. If Add/Update/Delete/View/Search/Statistics, the corresponding function is executed

5. After completion, the system returns to the main menu

6. If Exit is chosen, the program terminates

The activity diagram can show decisions based on the menu choice and loops back to the menu until exit.

### 7.3 Sequence Diagram (Example: Add Student)

Objects: User, main.py, operations.py, data_store.py

1. User selects "Add Student" from the menu.

2. main.py calls `add_student()` in operations.py.

3. operations.py prompts the User for roll number, name and marks.

4. operations.py calculates total, percentage and grade.

5. operations.py appends a new record (dictionary) to the `students` list in data_store.py.

6. Control returns to main.py and a success message is displayed.

### 7.4 Component / Module Diagram

Components:

- main.py
- operations.py
- reports.py
- data_store.py

main.py depends on operations.py and reports.py. Both these modules depend on data_store.py for accessing the student list.

### 7.5 ER Diagram

Logical entity:

STUDENT

- roll (Primary Key)
- name
- m1
- m2
- m3
- total
- percent
- grade

The ER diagram can show a single STUDENT entity as, in a real database, this table would store all rows.

## 8. Design Decisions & Rationale

1. **Console-Based Interface**

- A simple command-line interface was chosen instead of a GUI to keep the focus on core Python concepts and logic rather than UI design.

2. **In-Memory Storage Using Lists and Dictionaries**

- For a small academic project, using lists and dictionaries is sufficient and easier to implement than integrating a full database.

- Each student is represented as a dictionary which makes the code readable.

3. **Fixed Three Subjects**

  - To keep the system simple, the number of subjects is fixed to three. This also simplifies the calculation logic.

4. **Modular Code Structure**

  - Separating program logic into main, operations, reports and data_store improves readability and makes testing easier.

## 9. Implementation Details

- **Programming Language:** Python 3

- **IDE / Editor:** VS Code (or any text editor)

Key Concepts Used:

- Variables and basic data types (int, float, str)

- Lists to store multiple student records

- Dictionaries to store details of each student (roll, name, marks, total, percent, grade)

- Functions for modularity (e.g., `add_student()`, `update_student()`, `show_class_statistics()`)

- Conditional statements (`if`, `elif`, `else`) for grade calculation and menu choices

- Loops (`for`, `while`) to traverse records and show menus

The main loop in `main.py` repeatedly displays the menu until the user chooses to exit. Each option calls the related function from operations.py or reports.py. The `students` list in data_store.py is shared across these modules by importing it.

## 10. Screenshots / Results

(These should be captured from the running program and inserted into the final PDF.)

Suggested screenshots:

- Main menu screen
- Adding a new student
- Viewing all students
- Searching for a student by roll number
- Displaying class statistics

Each screenshot can be labelled and described briefly below it.

## 11. Testing Approach

The system was tested manually using the following methods:

1. **Functional Testing**

   - Added several students with different marks and verified that total, percentage and grade were correct.
   - Updated marks for existing students and checked that the values were updated properly.
   - Deleted students and ensured they no longer appeared in the list.
   - Used the search function with valid and invalid roll numbers.

2. **Input Validation Testing**

   - Tried entering letters or blank values instead of numeric marks to see if the system handled them gracefully.
   - Tried using duplicate roll numbers to confirm that the system reports the conflict.

3. **Boundary Testing**

  - Checked marks at boundaries such as 0, 50, 60, 70, 80, 90 and 100 to ensure correct grade assignment.

4. **Statistics Testing**

  - After entering multiple students, verified that the average percentage and topper details were calculated correctly.

## 12. Challenges Faced

- Designing a simple but meaningful grading logic.
- Ensuring that the program does not crash when the user enters invalid input.
- Deciding how to structure the project into multiple modules instead of writing everything in a single file.
- Keeping the code readable while passing data between functions and modules.

## 13. Learnings & Key Takeaways

- Gained hands-on practice with Python lists, dictionaries, functions and loops.
- Understood how to design and implement a menu-driven console application.
- Learned the importance of modular programming and separating logic into different files.
- Realised how basic validation and error handling improves the robustness of a program.
- Understood how to convert simple requirements into a working solution with clear architecture and design.

## 14. Future Enhancements

- Add file or database storage so that marks are saved even after the program exits.
- Allow the user to configure the number of subjects instead of fixing it to three.
- Provide a graphical user interface using Tkinter or develop a small web-based front end.
- Add login and authentication so that only authorised users can access or modify records.
- Export reports to CSV or PDF for printing and sharing.

## 15. References

- Python official documentation: https://docs.python.org/
- Course lecture notes and lab materials
- Basic programming and Python tutorials (books / class resources)