INTRODUCTION TO PROBLEM SOLVING AND PROGRAMMING

PROJECT

NAME::GOPI VARSHINI

REGISTRATION ID:25MIM10249

INTRODUCTION

The Student Grade Calculator is a software application designed to simplify the process of calculating, tracking, and managing student academic performance. In traditional educational settings, calculating final grades often involves manual input, weighting of different assignments (e.g., homework, quizzes, exams), and complex formulas. This project aims to automate and streamline this process, reducing calculation errors and providing students and educators with an immediate and accurate view of academic standing. The application provides a user-friendly interface for inputting assignment scores and weights, and immediately calculates the overall grade, often translating it into a corresponding letter grade.

FUNCTIONAL REQUIREMENTS

Functional requirements define what the system must do.

- FR1: Score Input: The system must allow users (students/instructors) to input scores for various graded components (e.g., Homework, Quizzes, Midterm, Final Exam).
- FR2: Weighting Input: The system must allow users to assign a percentage weight to each graded component (e.g., Homework: 20%, Final Exam: 40%).
- FR3: Weighted Grade Calculation: The system must correctly calculate the weighted score for each component and sum them to determine the overall numerical grade
- $$\text{Overall Grade} = \sum_{i=1}^{n} (\text{Score}_i \times \text{Weight}_i)$$
- FR4: Letter Grade Conversion: The system must convert the final numerical grade into a corresponding letter grade (e.g., $90-100\% \to \text{A}$, $80-89\% \to \text{B}$) based on a customizable grading scale.
- FR5: Data Persistence (Optional/Advanced): The system should allow the user to save and load different grading schemes or student records.
- FR6: Error Handling: The system must validate input, ensuring that scores are within a valid range (e.g., 0 to Max Score) and that the sum of all weights equals 100%.

5. Non-functional Requirements

Non-functional requirements specify criteria that can be used to judge the operation of a system.

- NFR1: Usability: The user interface (UI) must be intuitive and easy to navigate, allowing a user to input data and view results with minimal training.
- NFR2: Performance: The grade calculation must be performed instantaneously (e.g., within 0.5 seconds) after all necessary inputs are provided.
- NFR3: Reliability: The calculated grades must be 100% accurate based on the provided formula and inputs.

NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements specify criteria that can be used to judge the operation of a system.
NFR1: Usability: The user interface (UI) must be intuitive and easy to navigate, allowing a user to input data and view results with minimal training.
NFR2: Performance: The grade calculation must be performed instantaneously (e.g., within 0.5 seconds) after all necessary inputs are provided.
NFR3: Reliability: The calculated grades must be 100% accurate based on the provided formula and inputs.
NFR4: Security (If multi-user): If the system stores data, it must implement basic measures to ensure data privacy and integrity.
NFR5: Maintainability: The code must be well-structured, commented, and follow coding standards to allow for future updates and debugging.

TESTING APPROACH

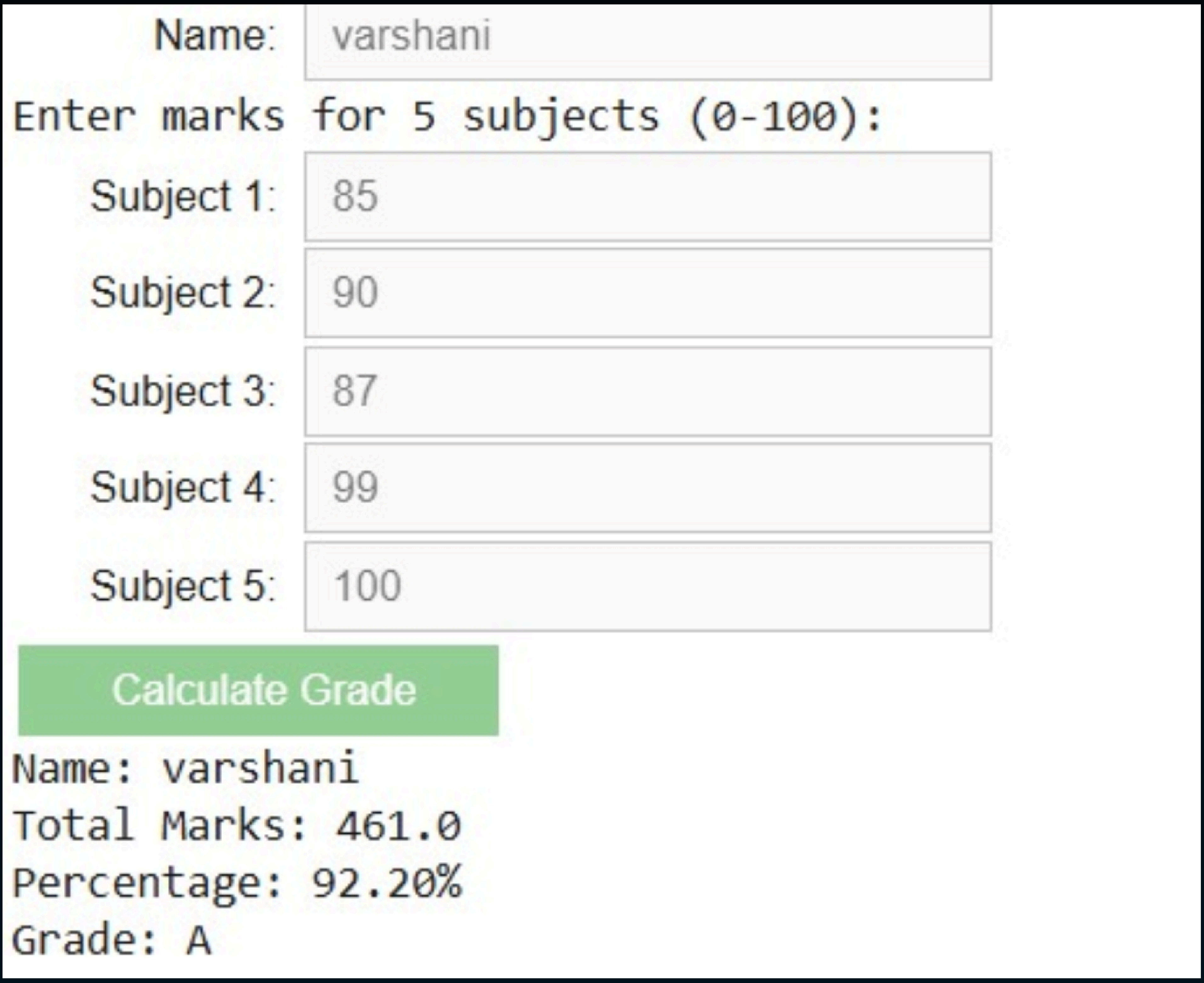
The project utilized a Unit and Integration Testing approach to ensure the reliability and accuracy of the core calculation logic.

- Unit Testing: Individual functions, such as calculate_weighted_score and convert_to_letter_grade, were tested in isolation using mock data. Key tests included edge cases like:
    - Inputting a 0% weight for a component.
    - Testing scores at the boundary of letter grades (e.g., 89.9% and 90.0%).
    - Testing negative or overly large score inputs to verify error handling.
-

TEST APPROACH

- Integration Testing: The flow from data input $\to$ calculation $\to$ result display was tested end-to-end to ensure components work together correctly. For example, a user scenario was tested where a user inputs scores and weights, and the final output is verified against a manually calculated result.
-
-
- User Acceptance Testing (UAT): A small set of target users (e.g., classmates or a teacher) reviewed the UI and provided feedback on usability, which led to refinements in the data entry flow.

OUTPUT SCREENSHOT



Name:	varshani

Enter marks for 5 subjects (0-100):

Subject 1:	85
Subject 2:	90
Subject 3:	87
Subject 4:	99
Subject 5:	100

```
Name: varshani
Total Marks: 461.0
Percentage: 92.20%
Grade: A
```

PROBLEMS FACED

Challenge 1: Handling Non-100% Weights: Ensuring the system correctly handles a scenario where the user inputs weights that do not sum up to 100% (e.g., by automatically normalizing them or prompting an error).

Challenge 2: Floating-Point Precision: Dealing with minor inaccuracies in standard computer arithmetic ($0.1 + 0.2 \ne 0.3$) when calculating and comparing grade boundaries. This was resolved by using appropriate rounding functions at the final output stage.

Challenge 3: Complex Grading Schemes: Initial design did not account for "drop-the-lowest-score" or extra credit, requiring a modification to the data structure to accommodate these advanced features.

FUTURE ENHANCEMENTS

Your paragraph textThe current application provides a solid foundation, but the following features are planned for future versions:

- Persistence and User Accounts: Implement a database or file system to save student/course data and potentially restrict access using basic user authentication.
- "What-If" Analysis: Allow users to input hypothetical scores for future assignments to see their potential final grade (e.g., "What grade do I need on the final exam to get an A?").
- Visualization: Incorporate charts or graphs to visually represent the grade breakdown and progress over time.
- Mobile Responsiveness: Develop a mobile-friendly interface for on-the-go access.

REFERENCES

1. Ipywidgets from IPython.display

2. VS CODE COMPILER