COLLEGE CODE : 9216

COLLEGE NAME : **Anna University Regional Campus Madurai**

NAME : Mohan Thulasi G

DEPARTMENT : CSE

NM ID : 9C663630720C21C9148420B4096FD70D

ROLL NO : 921623104706

DATE : 31/10/2025

COMPLETED **PHASE 5** IN THE PROJECT NAMED AS **STUDENT**

**GRADING SYSTEM**

**SUBMITTED BY,**

NAME : Mohan Thulasi G

MOBILE NO : 8637415705

**Phase 5:** STUDENT GRADING SYSTEM

# Final Demo Walkthrough:

1. Introduction (1–2 minutes)

Start with a short introduction:

"Good [morning/afternoon]. Today, I'll be presenting my project — the Stud
This system helps manage student data, calculate grades, and store results

Mention key goals:

Automate grading process

Reduce manual calculation errors

Make student result management easier

2. Project Setup (Show the Environment)

Explain the tools you used:

Frontend: HTML / CSS / React / Console UI

Backend: Node.js / Java / C#

Database: Local JSON / MySQL / MongoDB

Example:

"The backend runs on Node.js with Express, and I've used local storage for
Here's how I start the server using node app.js."

3. Core Features Demo (Main Section)
(a) Add New Student

Show a form or console input for entering:

Name

Roll Number

Subject Marks

Click Submit / Add Student

“This feature lets us add a student’s record into the system.”

(b) Calculate Grade Automatically

Show how grades are assigned automatically:

Example logic:

90–100 → A

80–89 → B

70–79 → C

Below 40 → Fail

“After submitting, the system automatically calculates the total, average, and grade.”

(c) View All Student Records

Show a table or list:

| Roll No | Name | Marks | Total | Grade |
|---|---|---|---|---|
| 101 | Ravi | 85 | 425 | B |
| 102 | Meena | 92 | 460 | A |

“This table shows all stored records fetched from the database or local storage.”

(d) Edit or Delete a Record

Demonstrate updating marks or deleting a student.

“If a student’s marks were entered incorrectly, we can easily edit them or remove the record.”

(e) Search or Filter

Search by Roll Number or Name

“This makes it easier to find a student in large lists.”

4. Data Storage

Explain how data is saved:

If using Local Storage: show how data persists after refresh

If using Database: open database or terminal to show stored records

“All data is stored safely, and the system can retrieve it anytime.”

5. Testing / Error Handling

Show what happens if:

User leaves a field empty

Marks entered > 100 or < 0

“I’ve added validation checks to avoid invalid data entry.”

6. Extra Features (Optional but impressive)

You can mention or show:

Login System (Admin / Teacher)

Export to PDF / Excel

Graphical Analysis (using charts)

Responsive Web Design

7. Conclusion (Wrap-up)

“In conclusion, my Student Grading System helps automate result management, improves accuracy, and saves time.
It can be extended further to support multiple classes or integrate online result publishing.”

# Project Report

## 1. Title of the Project

Student Grading System

## 2. Introduction

The Student Grading System is a software application designed to automate the process of evaluating and grading students based on their academic performance.
This project eliminates manual calculation errors and makes the management of student records efficient and reliable.

The system allows teachers or administrators to add student details, input marks, calculate total and average scores, assign grades automatically, and display the results in a structured format.

## 3. Objectives of the Project

To design an automated system for calculating student grades.

To store, manage, and retrieve student performance records efficiently.

To reduce human errors in mark calculation and grade assignment.

To provide a user-friendly interface for teachers and administrators.

To generate reports that summarize student performance.

## 4. Scope of the Project

The system is designed to handle:

Multiple student records

Automatic calculation of total, average, and grade

Searching, editing, and deleting of student information

Storage of student data in local storage or a database

Future versions can include:

User authentication (Admin/Teacher login)

Cloud database integration

Report card PDF export

Graphical analysis of student performance

## 5. Problem Statement

In traditional grading systems, teachers manually calculate total marks and assign grades to students, which is time-consuming and error-prone.
The need for an automated grading system arises to ensure accuracy, save time, and simplify result management.

## 6. System Requirements
### Hardware Requirements

Processor: Intel i3 or above

RAM: Minimum 4 GB

Storage: 100 MB or more

Display: Standard Monitor (1024×768 resolution)

### Software Requirements

Operating System: Windows / Linux / macOS

Development Tool: VS Code / Eclipse / IntelliJ / Node.js

Programming Language: (Specify your language – Java / C# / Python / Node.js / Web)

Database: (Local Storage / MySQL / MongoDB / File System)

## 7. System Design
### Modules

Student Module – Input student details and marks.

Grading Module – Calculate total, average, and assign grade.

Display Module – Show student records in table format.

Data Storage Module – Save and retrieve data from storage or database.

Admin Module (Optional) – Manage user accounts and access control.

Data Flow Diagram (Conceptual)

```
  +------------------+
  |  Admin/Teacher   |
  +------------------+
           |
           v
 +------------------------+
 | Enter Student Details  |
 +------------------------+
           |
           v
 +------------------------+
 |  Calculate Grades      |
 +------------------------+
           |
           v
 +------------------------+
 |  Display / Save Data   |
 +------------------------+
```

8. Implementation Details
Algorithm / Logic

Input student name, roll number, and marks for all subjects.

Compute total and average marks.

Assign grade based on average:

90–100 → A

80–89 → B

70–79 → C

60–69 → D

Below 40 → Fail

Store the data in a database or local storage.

Display all records in a table format.

Sample Output Table

| Roll No | Name | Marks (Avg) | Total | Grade |
|---|---|---|---|---|
| 101 | Ravi | 92 | 460 | A |
| 102 | Meena | 78 | 390 | C |
| 103 | Arjun | 85 | 425 | B |

## 9. Testing

The project was tested using sample data to verify:

Correct calculation of totals and averages

Accurate grade assignment

Validation of input data (marks not exceeding 100)

Proper record storage and retrieval

All test cases produced the expected results.

## 10. Advantages

Reduces manual workload for teachers

Minimizes calculation errors

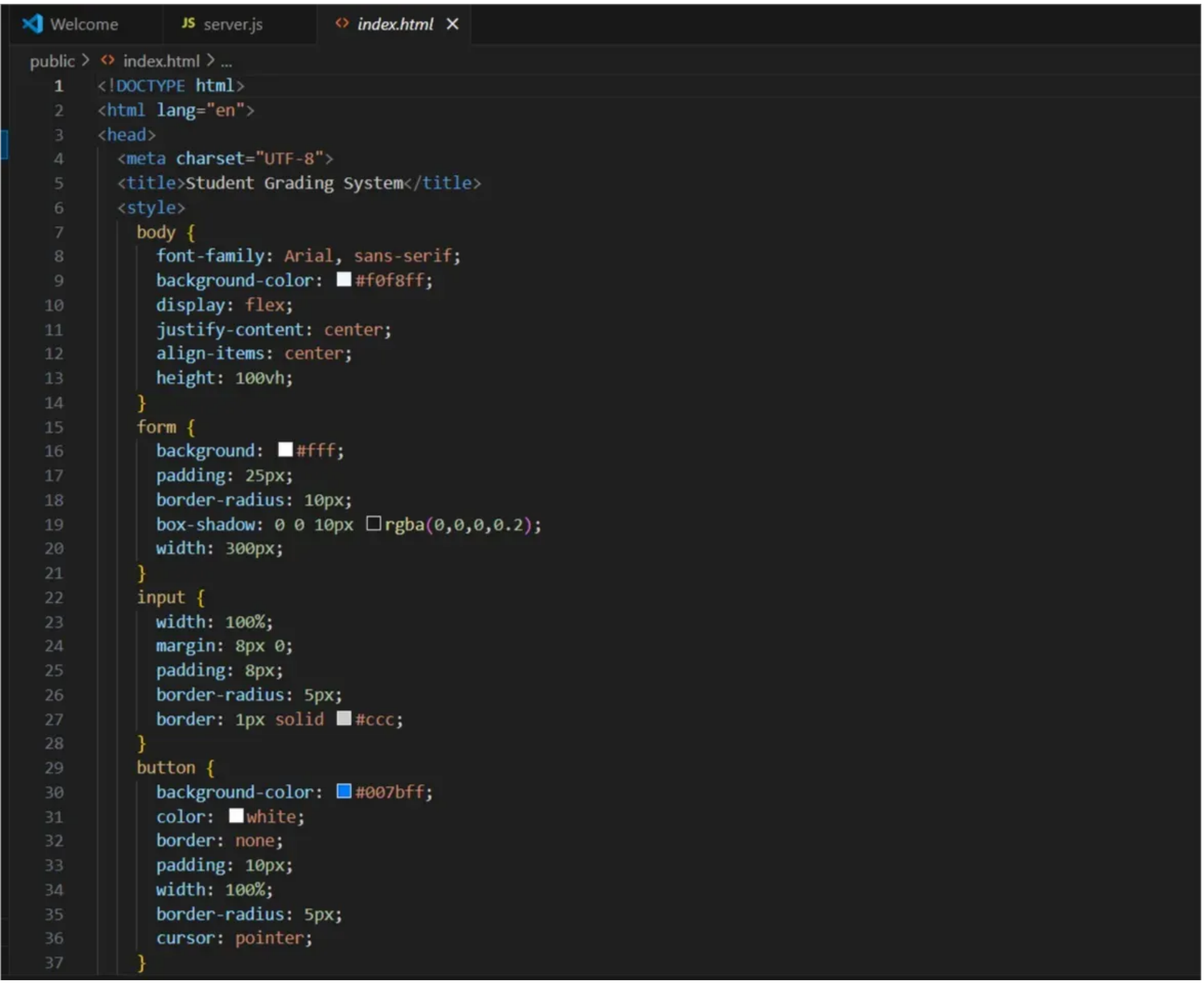
Easy access and update of student data

Fast and accurate report generation

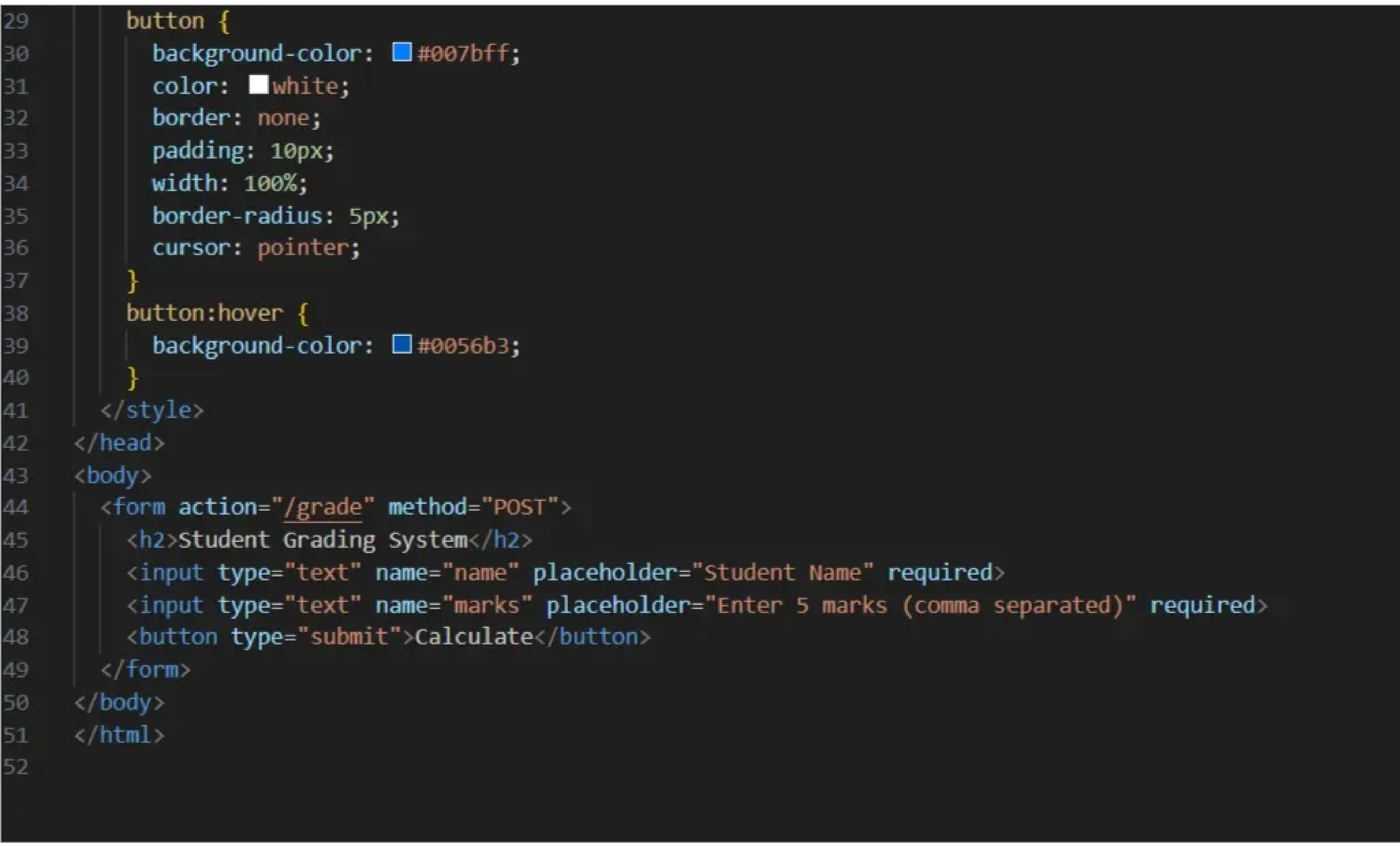
Simple and user-friendly interface

## 11. Limitations

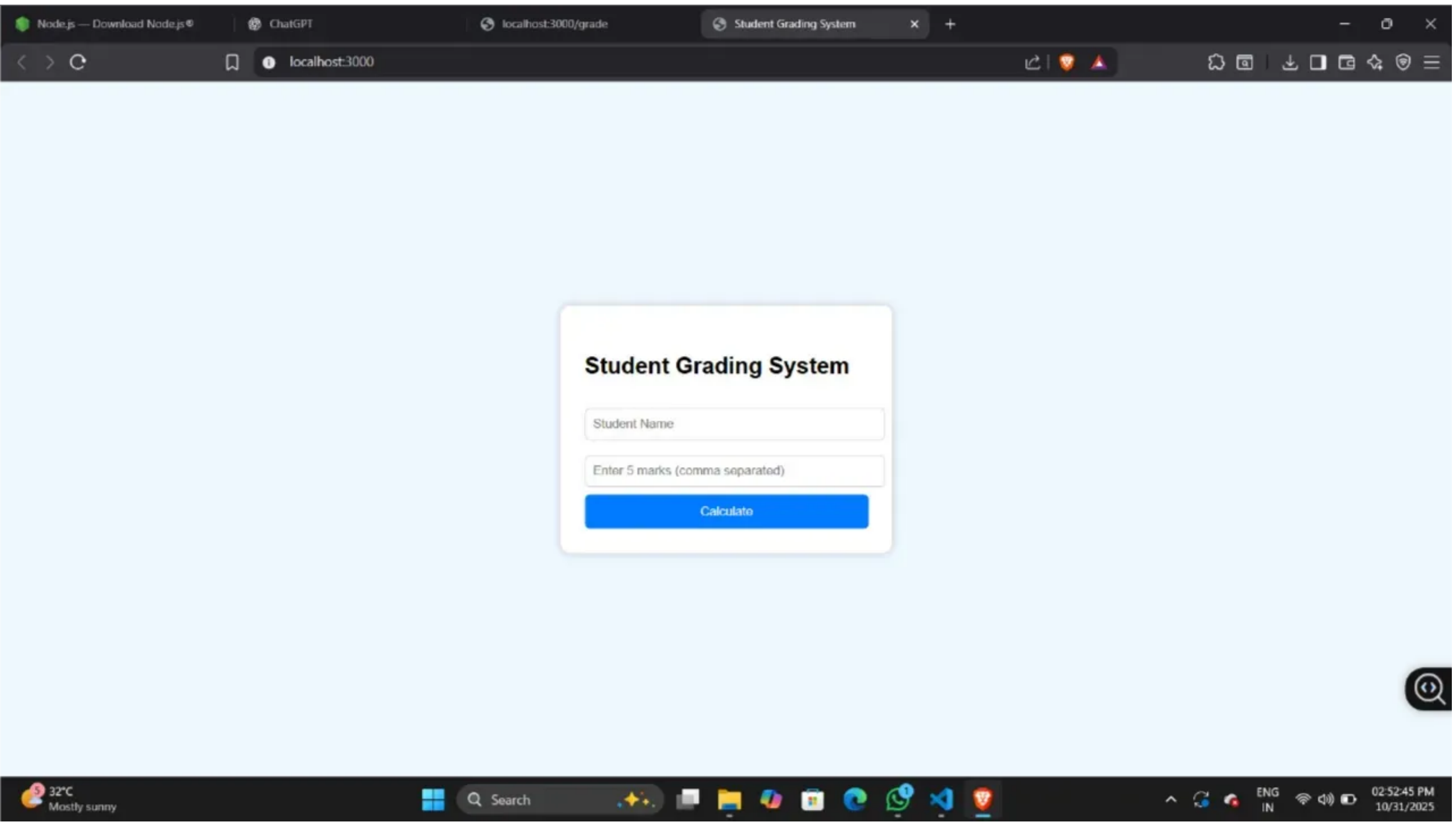
Currently limited to a single class or batch

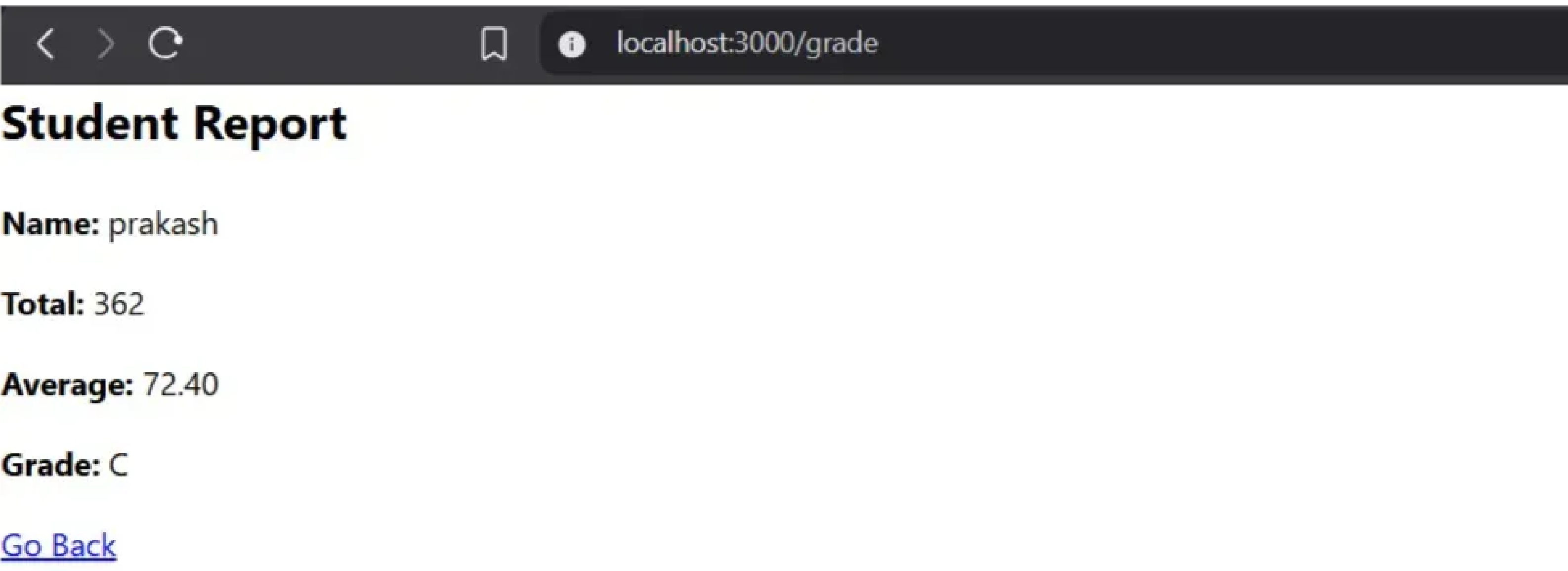


```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Student Grading System</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f0f8ff;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
    }
    form {
      background: #fff;
      padding: 25px;
      border-radius: 10px;
      box-shadow: 0 0 10px rgba(0,0,0,0.2);
      width: 300px;
    }
    input {
      width: 100%;
      margin: 8px 0;
      padding: 8px;
      border-radius: 5px;
      border: 1px solid #ccc;
    }
    button {
      background-color: #007bff;
      color: white;
      border: none;
      padding: 10px;
      width: 100%;
      border-radius: 5px;
      cursor: pointer;
    }
```



```
    button {
      background-color: #007bff;
      color: white;
      border: none;
      padding: 10px;
      width: 100%;
      border-radius: 5px;
      cursor: pointer;
    }
    button:hover {
      background-color: #0056b3;
    }
  </style>
</head>
<body>
  <form action="/grade" method="POST">
    <h2>Student Grading System</h2>
    <input type="text" name="name" placeholder="Student Name" required>
    <input type="text" name="marks" placeholder="Enter 5 marks (comma separated)" required>
    <button type="submit">Calculate</button>
  </form>
</body>
</html>
```


Node.js — Download Node.js®
ChatGPT
localhost:3000/grade
Student Grading System
localhost:3000
Student Grading System
Student Name
Enter 5 marks (comma separated)
Calculate
32°C
Mostly sunny
Search
ENG
IN
02:52:45 PM
10/31/2025


localhost:3000/grade

Student Report

**Name:** prakash

**Total:** 362

**Average:** 72.40

**Grade:** C

Go Back

Developing and testing all features within the given academic deadline was challenging.

Solution:
The project was divided into phases:

Basic functionality (add/view students)

Grading logic

Data storage

Testing & presentation prep

This modular approach helped complete the project on time.

10. Challenge: Future Scalability

Problem:
The initial version was limited to a single class or small number of students.

Solution:
The system was designed with scalability in mind — data structures and modular code allow easy expansion to multiple classes, subjects, or users in future versions.

No authentication system (optional for expansion)

No advanced analytics or graphical reports

## 12. Future Enhancements

Integration with cloud-based storage

Implementation of student login portal

Automatic email of report cards

Graph-based performance analysis

Multi-class and multi-subject support

## 13. Conclusion

The Student Grading System effectively automates the evaluation process, reducing human effort and improving accuracy.
It provides a solid foundation for a school or college to digitize grading and result management processes.
Future versions can extend the system with analytics and cloud features.

## 14. References

Java / Node.js / C# Documentation

W3Schools Tutorials

GeeksforGeeks – File Handling and Database Tutorials

Stack Overflow – Developer Community

## Challenges and Solutions

1. Challenge: Manual Grade Calculation Logic

Problem:
Creating an accurate grading algorithm was challenging — especially defining grade boundaries and ensuring correct grade assignment for edge values (e.g., exactly 90 or 100 marks).

Solution:
A clear conditional logic was implemented:

```
if (average >= 90) → A
else if (average >= 80) → B
else if (average >= 70) → C
else if (average >= 60) → D
else → Fail
```

This ensured that grades were assigned consistently for all mark ranges.

2. Challenge: Handling Invalid Input Data

Problem:
Users could enter invalid data, such as negative marks or marks above 100, leading to incorrect grade calculation.

Solution:
Input validation checks were added:

Marks must be between 0 and 100

Required fields cannot be empty

Error messages appear for invalid entries

This improved the accuracy and reliability of the system.

3. Challenge: Data Storage and Retrieval

Problem:
Managing persistent storage for student records was difficult at first, especially deciding between local files, local storage, or databases.

Solution:
A simple and efficient data storage solution was implemented:

For small projects → Local JSON or text file storage

For advanced versions → Database (MySQL / MongoDB) integration

This allowed smooth data saving, retrieval, and updates without performance issues.

4. Challenge: Displaying Student Records Clearly

Problem:
Displaying all student details (marks, totals, and grades) in a readable and organized format was initially complex.

Solution:
A structured table layout was created for the output screen.
Each student's data is displayed in a row showing roll number, name, total, and grade.
This improved clarity and made the system more user-friendly.

5. Challenge: Editing or Deleting Records

Problem:
Once data was saved, editing or deleting specific student records was difficult.

Solution:
Unique identifiers (like roll numbers) were used to locate records quickly.
Functions were added to update or remove data entries without affecting others.

6. Challenge: Maintaining System Accuracy

Problem:
When multiple students were entered, some calculations became inconsistent due to incorrect data handling or overwriting.

Solution:
Data structures (like arrays or objects) were used carefully to store student data separately.
Each student’s record was processed individually, ensuring accurate calculations.

7. Challenge: User Interface Design

Problem:
Designing an intuitive and simple interface that works well for teachers with minimal training.

Solution:
A clean interface was developed using HTML/CSS or console text menus (depending on project type), keeping navigation simple with clear options like:

Add Student

View Records

Calculate Grades

Exit

8. Challenge: Testing and Debugging

Problem:
Unexpected errors occurred during testing — such as incorrect averages or missing records.

Solution:
Comprehensive testing was done using:

Valid and invalid test cases

Edge case inputs (e.g., marks = 0 or 100)
Debugging tools (like console logs or print statements) helped identify and fix logical errors efficiently.

9. Challenge: Time Constraints

Problem:

## Version Control (GITHUB)

https://github.com/mohang765432-lang/Student-grading-system-full/tree/main

https://github.com/mohang765432-lang/Student-grading-system-phase-5

## Team members

1.K.Prakashraj

2.G.Mohan Thulasi

3.A.Loganathan