Marwadi University
Marwadi Chandarana Group

NAAC A+

FACULTY OF COMPUTER APPLICATIONS

A Python Project for Academic Year 2025

SCHOOL MANAGEMENT SYSTEM

NAME – KAVYA BHUVA [92300527006]

DEVANSHI SOJITRA [92300527005]

CLASS – BCA 5(A)

SUBJECT – MINI PROJECT

SUBMITTED TO – DIPAK THANKI

.

INDEX

INTRODUCTION

This Python-based School Management System is designed to facilitate efficient management of student data in a school setting. By using a CSV file (students.CVS), the system allows administrators to easily add, view, update, delete, search, and sort student records. The system validates input such as age and grade to ensure data integrity and provides error handling to manage common issues like missing files or invalid inputs. The core functionalities include adding student details, displaying all student records, updating information, deleting students, and exporting data to a text file for backup or reporting purposes.

The system also includes features like sorting students by grade and counting the total number of students, helping school staff quickly organize and assess their student population. With a user-friendly interface and simple command-line interactions, this tool offers an accessible solution for managing student information without the need for complex database systems, making it ideal for small educational institutions or personal projects.

SYNOPSIS

The School Management System is a Python-based console application designed to manage student records efficiently using a CSV file for data storage. It allows users to perform various operations such as adding new students, viewing existing records, updating and deleting student information, searching for students by ID or name, and sorting the records based on grade. The system also includes features to export the student data to a readable text file and display the total number of students currently stored. Data validation is implemented to ensure that the student's age falls within the range of 5 to 100 and that the grade is one of the accepted values (A, B, C, D, or F). The application uses Python's built-in csv, os, and datetime modules to handle file operations, check for the existence of files, and manage date formatting. It provides a user-friendly command-line interface with a menu-driven structure that continues to run until the user chooses to exit. Overall, this system serves as a simple yet effective tool for maintaining student information in educational institutions.

Project Description: School Management System

The **School Management System** is a simple yet effective Python project designed to help educational institutions manage student information in a structured and organized manner. This console-based application uses a CSV file as its data storage backend, making it lightweight, portable, and easy to use without the need for complex databases.

The system enables users to perform key operations such as **adding**, **viewing**, **updating**, **deleting**, and **searching** student records. Each record contains essential student details like ID, name, age, grade, address, phone number, email, and guardian's name. To ensure data integrity, validations are performed for age (between 5 and 100) and grade (A to F).

Additional features include **sorting students by grade**, **exporting data to a text file** (students.txt) for easy sharing or reporting, and **counting the total number of students** in the system. The application uses standard Python libraries like csv for file handling, os to check file existence, and datetime to manage timestamps (though not stored in the file currently).

Designed with simplicity in mind, the project provides a menu-driven interface that runs in a loop until the user exits. It serves as an ideal learning project for beginners in Python, especially for those looking to understand file handling, data validation, and basic CRUD (Create, Read, Update, Delete) operations in a real-world context.

Description of each module

Class: SchoolManagementSystem

This is the core class handling all the operations like adding, viewing, updating, deleting, sorting, exporting, and validating student records stored in a CSV file.

- **__init__(self, filename='students.csv')**

Purpose: Initializes the system and ensures the CSV file exists with appropriate headers.

Key Actions:

- Sets the CSV filename (default: students.csv).
- Checks if the file exists. If not, creates it with column headers.

- **add_student(self, student_id, name, age, grade, address, phone, email, guardian_name)**

Purpose: Adds a new student record to the CSV file.

Validations:

- Age must be an integer between 5 and 100.
- Grade must be one of: A, B, C, D, F.

Actions:

- Validates inputs.
- Appends the student data to students.csv.
- Displays success or error messages.

🔹 view_students(self)

Purpose: Displays all student records from the CSV file in a readable format.

Actions:

- Reads the file.
- Skips the header.
- Prints each student's details in a structured format.
- Handles empty files or file-not-found errors.

🔹 delete_student(self, student_id)

Purpose: Deletes a student based on their ID.

Actions:

- Reads all rows.
- Removes the row where row[0] == student_id.
- Writes the remaining data back to the file.
- Prints a message confirming deletion or showing an error if not found.

- **update_student(self, student_id, new_name=None, new_age=None, new_grade=None, new_address=None, new_phone=None, new_email=None, new_guardian=None)**

Purpose: Updates specific fields of a student record.

Actions:

- Searches for the student by ID.
- Replaces only the fields provided (leaves others unchanged).
- Writes the updated data back to the file.
- Prints confirmation or error if student not found.

- **search_student(self, search_term)**

Purpose: Searches for a student by ID or Name.

Actions:

- Iterates through each row.
- Compares search_term to the ID and Name fields (case-insensitive).
- Displays matching records.

Technical Information

Technologies Used:

- Python (main programming language)
- CSV (file format for storing student data)
- OS (for file operations)
- Date time (for date-related functionalities)
- Error Handling (using try-except blocks)

Development Tools:

- Python Interpreter (for running Python code)
- Code Editor / IDE: VS Code, PyCharm, or IDLE
- Version Control System: Git (with GitHub or Git Lab)
- CLI/Terminal: For running Python scripts
- Text/CSV File Editor: Notepad++

System Requirements

Hardware:

- Processor: 1 GHz (minimum), 2 GHz (recommended)
- Memory (RAM): 2 GB (minimum), 4 GB (recommended)
- Storage: 100 MB (minimum), 500 MB (recommended)

Software:

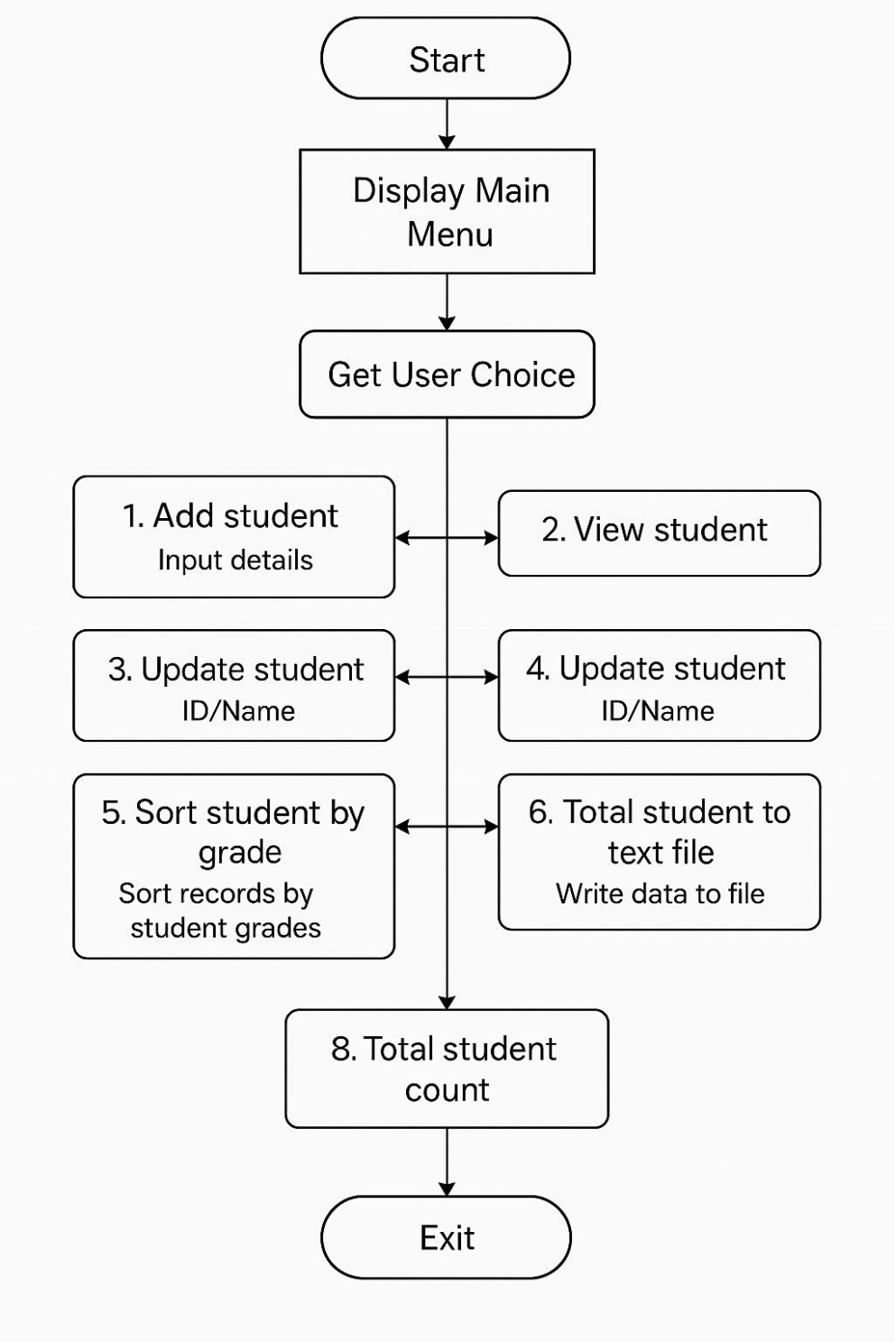
- Operating System: Windows 7+, macOS 10.10+, Linux
- Python: Python 3.6+ (recommended: Python 3.8 or higher)

Project Structure

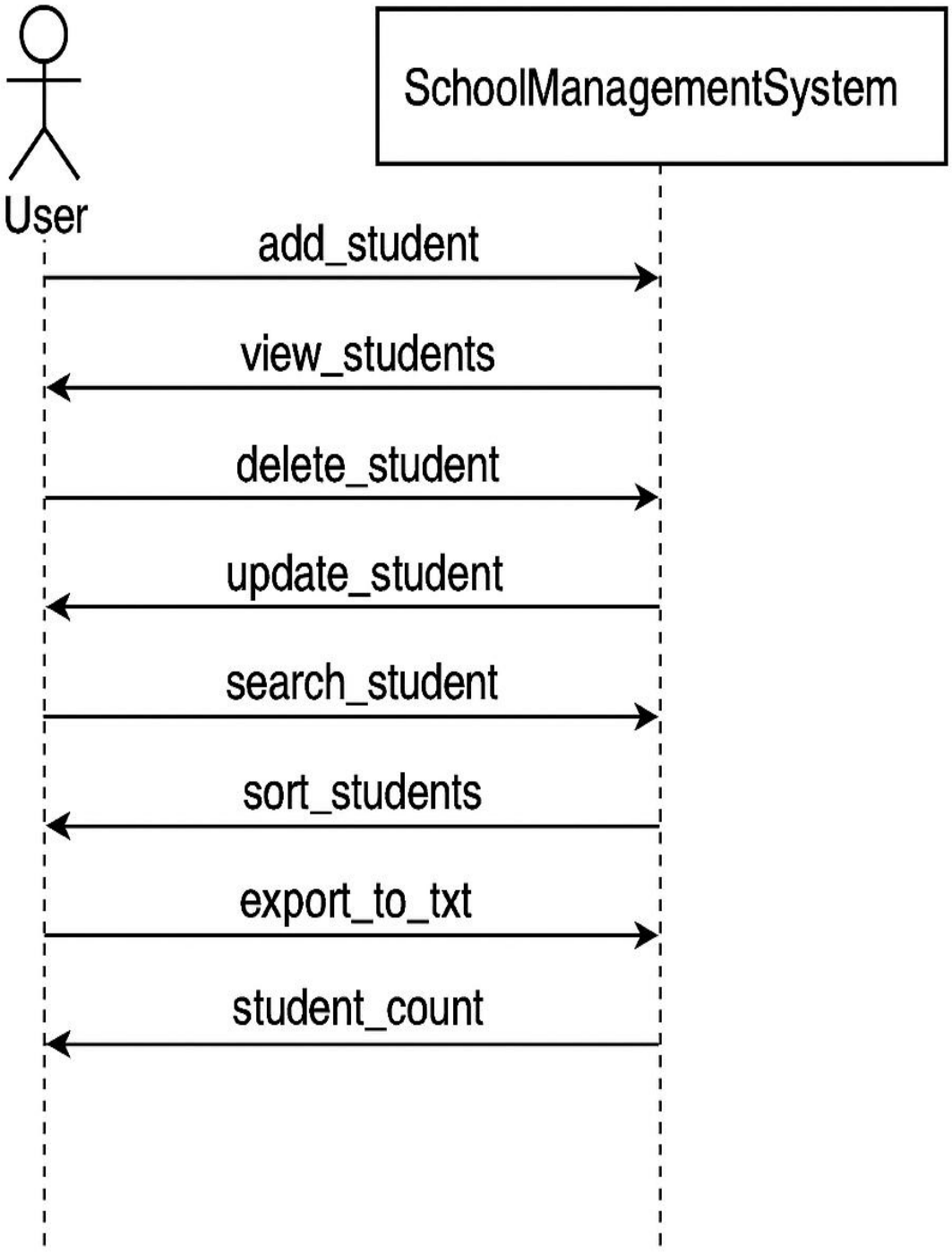
```
School Management System/
|
├── main.py                    # Main entry point for the program (the code you've written)
├── students.csv               # The CSV file where student data is stored
├── students.txt               # The text file to which student data is exported
```

- main.PY: This is the main script that contains all the logic for your School Management System, including classes and methods.
- students.CVS: This is the CSV file used for storing student data (ID, Name, Age, Grade, etc.). The program reads from and writes to this file.
- students.TXT: This is the text file that the program exports student data to when the export functionality is used.

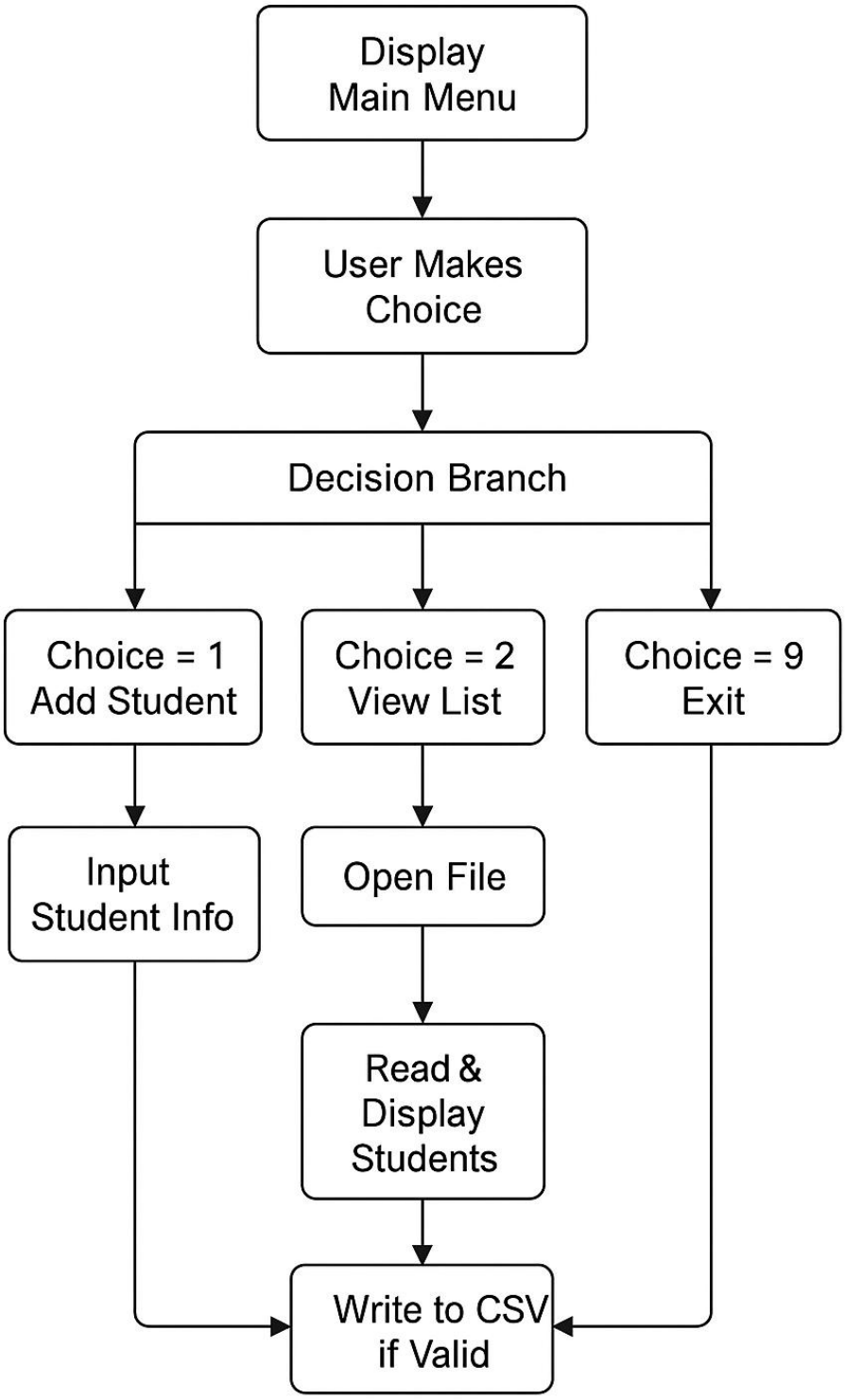
Diagram [FLOWCHAT]



Sequential Diagram



Activity Diagram



Database Structure of the School Management System

This School Management System uses a **CSV file (students.csv)** as its database to store and manage student records. Each row in the CSV file represents a single student's data, and the columns represent different attributes of that student.

Column Name	Data Type	Description
ID	String	Unique identifier for each student (e.g., "S001")
Name	String	Full name of the student
Age	Integer	Age of the student (validated to be between 5 and 100)
Grade	String	Academic grade (A, B, C, D, or F)
Address	String	Residential address of the student
Phone	String	Contact number
Email	String	Email address
Parent/Guardian Name	String	Name of the parent or guardian

Sequence Table: Add Student

Step	Initiator	Receiver	Action/Method Called	Description
1	User	Main (CLI)	Selects "Add Student"	User chooses the option from the menu
2	Main	User	Prompt for inputs	CLI prompts for ID, name, age, grade, etc.
3	User	Main (CLI)	Inputs data	User fills in student details
4	Main	SchoolManagement System	add_student()	CLI sends the data to the method
5	SMS	SMS	is_valid_age()	Internal check: Validates the student's age
6	SMS	SMS	is_valid_grade()	Internal check: Validates the student's grade

Field No.	Column Name	Description	Example
1	ID	Unique student identifier	STU101
2	Name	Full name of the student	Ananya Sharma
3	Age	Age of the student (5–100)	15
4	Grade	Academic grade (A, B, C, D, F)	A
5	Address	Home address	123 Park St, Delhi
6	Phone	Contact number	9876543210
7	Email	Email address	ananya@example.com
8	Parent/Guardian Name	Name of guardian	Mr. Rakesh Sharma

Where These Columns Are Used in

Method	Columns Accessed (by Index)
add_student	All (except reads ID duplicates unless modified)
view_students	All from 0 to 7 (8 if enrollment added)
update_student	All fields (row[1] to row[7])
delete_student	row[0] (ID)
search_student	row[0], row[1]
sort_students	row[3] (Grade)
export_to_txt	All fields
student_count	All rows (not header)

Features

- Add Student
- View Students
- Delete Student
- Update Student
- Search Student
- Sort Students by Grade
- Export Students to Text File
- Total Student Count

Screenshot

Main Menu:

```
School Management System
1. Add Student
2. View Students
3. Delete Student
4. Update Student
5. Search Student
6. Sort Students by Grade
7. Export Students to Text File
8. Total Student Count
9. Exit
Enter your choice:
```

Learning Objectives

1. Object-Oriented Programming (OOP):

• Class Definition and Object Creation: Learn how to define and work with classes (School Management System), create instances (objects), and organize functionality within a class.

• Instance Methods: Understand the purpose of instance methods (add_student, view_students, etc.) that operate on class attributes.

• Constructor Method (_init_): Learn how to use constructors to initialize an object's state and ensure that necessary setup (like file creation) occurs when an object is instantiated.

2. File Handling in Python:

• Reading and Writing CSV Files: Gain experience in reading from and writing to CSV files using Python's built-in csv module. This is critical for applications that require persistent storage.

3. Managing Lists and Iterating through Data:

• List Operations: Understand how to create, modify, and iterate through lists (e.g., storing students' records in a list and appending, removing, or updating elements).

• Search and Filter: Learn how to search for a specific student by name or ID and how to filter data based on user input.

Conclusion

In conclusion, the School Management System is a Python-based application that efficiently handles student data using CSV files. It covers essential programming concepts like file handling, object-oriented programming (OOP), data validation, and user input/output. The system allows for adding, updating, viewing, deleting, and sorting student records, providing a simple yet effective tool for small-scale school management. It demonstrates good practices in error handling, modular design, and scalability. While suitable for basic use, it can be expanded with features like database integration, a graphical user interface, and advanced reporting for larger applications.