PROJECT TITLE: Student Management System

TEAM NO: 21

UNDER THE GUIDANCE OF: Anupama

INTRODUCTION:

This mini-project titled "Student Management System" was developed as part of the coursework for Mini Project Submission at GCEM.

It aims to streamline student data management using Python's tkinter library.

PROBLEM STATEMENT:

Student Management System - A program to manage student records.

Students should be able to add, update, delete, and view records.

OBJECTIVES:

1. To design a simple and user-friendly interface for managing student data.
2. To provide functionalities such as adding, updating, searching, and deleting student records.
3. To store student records persistently using a text file as a database.
4. To allow real-time updates and minimize manual effort.

FEATURES:

1. Add Student: Allows users to add a new student record.
2. Fetch Data: Fetches details of a student based on the USN (Unique Student Number).

3. Save Updates: Enables modification and saving of student details.

4. Delete Student: Deletes a specific student record based on the USN.

5. Search Student: Searches for a student's details by USN and displays them.

6. Input Validations: Ensures all fields are mandatory to prevent incomplete data entry.

TOOLS AND TECHNOLOGIES USED:

1. Programming Language: Python

2. Libraries: tkinter, messagebox, simpledialog, ttk (for the GUI)

3. Data Storage: Text file-based storage.

USER INTERFACE (UI) DESIGN:

The UI for the project includes input fields for USN, Name, Gender, CGPA, Semester, Date of Birth, and contact

information (both student and parent). It has buttons for each functionality: Add, Fetch, Save, Search, and Delete.

CODE EXPLANATION:

1. The main class, `StudentManagementSystem`, initializes the tkinter interface and manages all functionalities.

2. Data is read and written to a text file, "students.txt", which serves as a lightweight database.

3. Input validation ensures that no fields are left empty, avoiding data inconsistency.

IMPLEMENTATION:

The project was implemented using the Model-View-Controller (MVC) pattern, with the tkinter library handling the

view (UI), functions serving as controllers, and the text file representing the model

CHALLENGES FACED:

1. Handling file I/O efficiently.
2. Ensuring input validation for fields such as email, phone numbers, and CGPA.
3. Designing an intuitive and error-free interface.

RESULTS AND DISCUSSION:

The project successfully meets its objectives by providing a functional interface for student data management.

It is a scalable solution for small-scale institutions and can be enhanced further.

FUTURE ENHANCEMENTS:

1. Replace the text file with a robust database like SQLite or MySQL.
2. Add advanced search features (e.g., by name or semester).
3. Implement data encryption for sensitive information.
4. Create a web-based or mobile version for broader accessibility.

CONCLUSION:

The Student Management System serves as a practical tool for managing student records. It showcases the team's

understanding of GUI design, file handling, and basic software development principles.

DETAILED DESCRIPTION OF THE FEATURES:

1. Add Student:

- This feature enables the user to add a new student record to the system. The user must fill in all required fields, including USN, Name, Gender, CGPA, Semester, Date of Birth, and contact details.
- If any field is left blank, an error message is displayed to ensure data integrity.

Implementation:

- Input is collected using tkinter's Entry and Combobox widgets.
- The collected data is written to a text file in a comma-separated format.
- The program ensures that duplicate entries are avoided.

Challenges:

- Validating inputs like CGPA (ensuring it's within 0-10) and mobile numbers (10-digit format).
- Handling file write operations to prevent overwriting existing data.

2. Fetch Data:

- The "Fetch Data" option allows users to retrieve an existing student's details by entering their USN.
- Once the USN is provided, the program searches through the text file and populates the input fields with the corresponding data.

Implementation:

- Reads the text file line by line to find a matching USN.
- If found, the data is parsed and displayed in the UI.
- If the USN is not found, an error message is shown.

Applications:

- Useful for quickly retrieving and viewing a student's information.
- Acts as a prerequisite step for updating or deleting a student's record.

3. Save Updates:

- After fetching a student's data, the user can modify any field and save the updated details.
- This feature ensures that changes are reflected in the database file.

Implementation:

- The text file is read line by line, and the matching record is replaced with the updated data.
- The updated file is then written back to the disk.

Use Cases:

- Correcting errors in student records (e.g., updating contact details or semester).

4. Delete Student:

- This feature allows the user to delete a student's record from the system.
- The user is prompted to enter the USN of the student they wish to delete.
- Once confirmed, the record is removed from the database.

Challenges:

- Ensuring the deletion process does not affect other records.
- Providing a confirmation dialog to prevent accidental deletions.

5. Search Student:

- Users can search for a student's details using their USN.
- This feature is similar to "Fetch Data" but provides a summary in a pop-up dialog box rather than populating the input fields.

Benefits:

- Quick access to student information without the need for detailed editing.

FILE HANDLING IN THE PROJECT:

The project uses a text file (`students.txt`) to store all student data. Each record is stored on a new line in

a comma-separated format (CSV). The fields include USN, Name, Gender, CGPA, Semester, DOB, Mobile numbers, and Email.

Advantages of Using a Text File:

- Simplicity: No need for external database dependencies.
- Portability: The file can be easily transferred and backed up.

Limitations of Using a Text File:

- Scalability: As the number of records grows, searching and updating data may become slow.
- Security: The data is stored in plain text, making it vulnerable to unauthorized access.

FUTURE ENHANCEMENTS IN DETAIL:

1. Database Integration:

Replacing the text file with a relational database like SQLite or MySQL would improve scalability and performance. It would also enable more complex queries and data analysis.

2. Web and Mobile Versions:

Developing a web-based version using frameworks like Django or Flask would allow multi-user access. A mobile app version could further enhance accessibility.

3. Data Security:

Implementing encryption techniques to secure sensitive information like email addresses and phone numbers.

4. Additional Features:

- Batch-wise record management.
- Automatic generation of reports (e.g., performance analysis by semester).
- Integration with other systems, such as attendance tracking.

5. UI Enhancements:

- Adding themes and customization options for better user experience.
- Implementing a dashboard to display summary statistics (e.g., average CGPA, total students).

LEARNING OUTCOMES:

Through this project, the team gained hands-on experience in:

- Python programming and GUI development.
- File handling and data storage techniques.
- Problem-solving and debugging.
- Collaboration and teamwork during the development process.

CODE SNIPPETS:

Below are examples of some key functions:

#Add Student:

```python
def add_student(self):
    data = (
        self.roll_no_var.get(),
        self.name_var.get(),
        self.gender_var.get(),
        self.cgpa_var.get(),
        self.semester_var.get(),
        self.dob_var.get(),
        self.mobile_var.get(),
        self.pmobile_var.get(),
        self.email_var.get(),
    )
    if any(not field for field in data):
        messagebox.showerror("Error", "All fields are required!")
        return
    with open(self.file_name, "a") as file:
        file.write(",".join(data) + "\n")
    messagebox.showinfo("Success", f"Student {data[0]} added successfully.")
```

#Fetch Student:

```python
def fetch_student(self):
    roll_no = self.roll_no_var.get()
    if not roll_no:
        messagebox.showerror("Error", "Enter a USN to fetch!")
        return
    with open(self.file_name, "r") as file:
        for line in file:
            if line.startswith(roll_no + ","):
                data = line.strip().split(",")
                self.name_var.set(data[1])
                self.gender_var.set(data[2])
                # ... populate other fields ...
                messagebox.showinfo("Success", "Student details fetched.")
```

```
                return
    messagebox.showerror("Error", "USN not found!")
```

APPENDICES:

1. Complete source code (refer to the attached files).
2. Screenshots of the application interface.

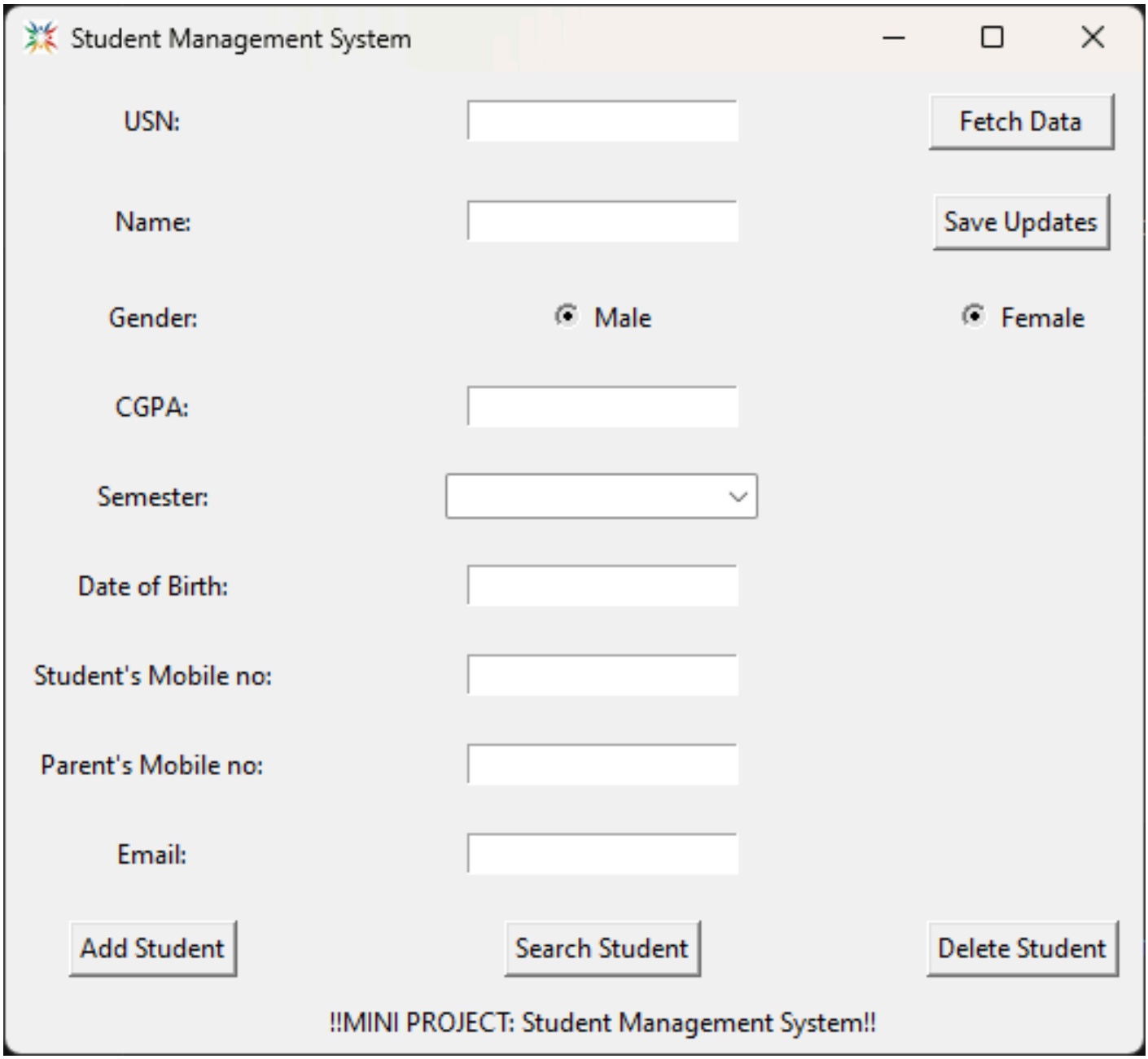
ACKNOWLEDGMENTS

We would like to thank our guide, Anupama ma'am, for their guidance and support throughout this project.

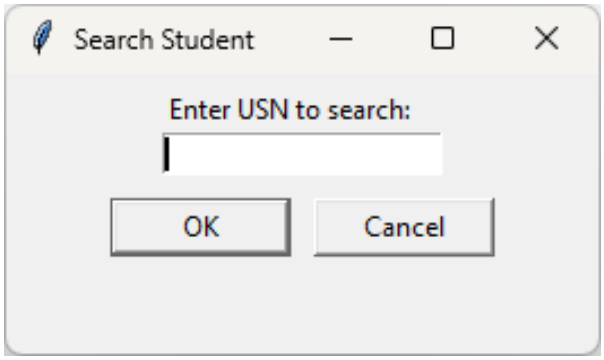
We also extend our heartfelt gratitude to Dr. Manoj Challa Sir for providing us the opportunity to participate in this endeavor, create this app, and for offering a platform to showcase our work

We would also like to acknowledge our peers and the GCEM faculty for providing valuable feedback.

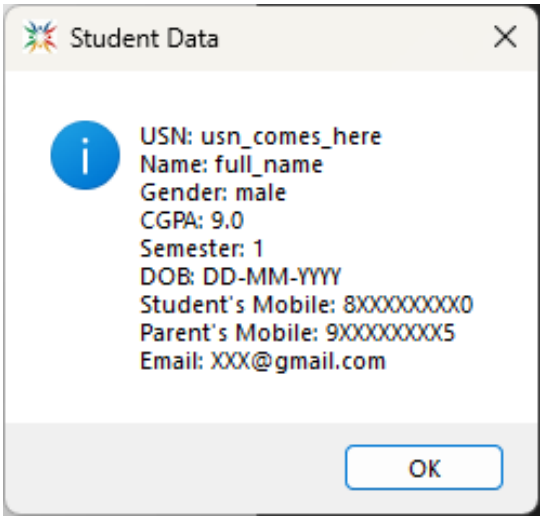
SCREENSHOTS OF THE APPLICATION INTERFACE



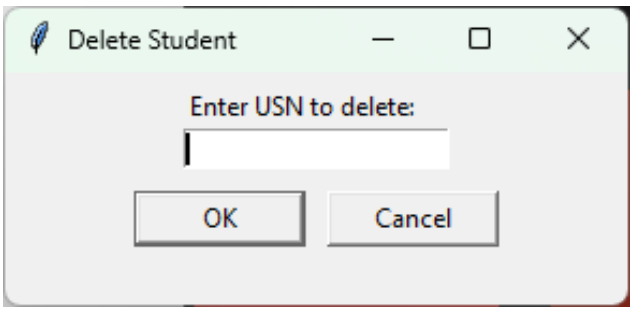
Search option: input



Search option: output



Delete option



--- END OF REPORT ---