VITyarthi

Build Your Own Project
Course Project Report

Project Title: *Small Clinic Management System*

Submitted By: *Nehal Sonal*

Registration Number: *25BHI10068*

Program: *B. Tech*

Course Code: *CSE1021*

Course Name: *Introduction to problem solving and programming*

Faculty: SIVABALAN.KR

Institution: *Vellore Institute of Technology, Bhopal*

Academic Year: *2025-29*

Introduction

Managing patient information efficiently is essential for small clinics, where the workload is basically maintaining records, updating medical history, scheduling appointments, and retrieving patient details quickly. Many available clinic management systems are either very complex or designed for large-scale hospitals which work on heavy duty, making it difficult for small clinics or sole doctors to use.

To address this, the Small Clinic Management System was developed as a simple and lightweight Python-based command-line program. The main objective of this project is to provide an easy to use system that allows clinic staff to add new patients on a regular basis , view their records, edit information if anything gets input incorrectly, record medical history, schedule appointments, and manage all data from one place. The program stores information in a text file, making sure of the fact that the records remain available even after restarting the system.

During the development of this project, Python concepts such as modules , loops, conditional statements , manipulation of string , and file handling were used. For this problem statement the system design includes requirement analysis and workflow structure based on software engineering practices. Overall, this project offers understanding for the building of real-world utility applications by focusing on a real world problem statement while strengthening skills in Python programming .

Problem Statement

As we know Small clinics often struggle to maintain patient information in an organized and easily accessible manner as they do not have a lot of employed people. Using paper-based records can lead to lost data, difficulty in recovering patient history, and inefficiency in managing appointments. Many available digital management systems are overly complex, require advanced setup and features to operate which is an unlike need for small clinics.

Therefore, there is a need for a simple, easy to operate system that can run on any computer without complex installation and perform essential clinic operations.

The objective of this project is to build a Python-based Small Clinic Management System that allows users to add patient records, view details, update information, manage medical history, schedule appointments, view its statistics and maintain all data in a straightforward command-line interface. This program stores all patient details securely in a text file, ensuring that records remain protected and can be easily accessed .

Objectives

- To develop a python based command line based clinic management system that is easy to use.
- To provide essential functionalities such as adding patient details , editing their records, deleting entries once they leave , and viewing patient details by referring to its id .
- Maintenance of all patient data, the medical history for reference, and appointments using a text file based storage system for reference.
- To implement the project using inbuilt modules of Python .
- To understand and apply core Python concepts such as functions, loops, conditional statements, and file handling.

Functional Requirements

Module 1: Patient Registration

- Add a new patient with basic details such as patient ID, name, age, and gender.

Module 2: Patient detail view by Id

- Reviewing patient details by entering their Id for future references.

Module 3: Searching all patients

- For referring to details of all patients and knowing which of them are still under treatment or checked out this feature is helpful.

Module 4: Patient record through name

- Search patients by name if in case the Id is misplaced or forgotten.

Module 5: Patient Information Management (combined for options 5,6,7,8 in the code)

- Edit an existing patient's details
- Delete a patient record
- Add medical history notes
- Add or update appointment information

Module 6: System Utilities (combined feature for 9 and 10 in the code)

- Clear all patient records
- View basic clinic statistics (total patients, male/female count)

Non-Functional Requirements

- **Usability:** Helps the clinic staff as it provides a simple and easy to navigate command line menu.
- **Performance:** Ensures instant reading and writing of patient records which is very suitable for small to medium-sized clinics.
- **Maintainability:** Uses a modular programming approach structure that makes the system easy to update or expand.
- **Reliability:** Maintains accuracy of data through structured text file storage and controlling of file operations.
- **Error Handling:** Validation of user input and handling of incorrect or missing data to avoid system crashes.

System Architecture

The system is divided into three main components:

1. **User Interface (CLI)**

- ➢ Displaying the main menu options
- ➢ Takes user inputs such as patient ID, name, age, history notes, or appointment details.
- ➢ Shows outputs for the input taken patient records, search results, and calculated statistics of the clinic.

2. **Clinic Management Logic**

- ➢ Handling all core operations such as adding, editing, deleting, and viewing patient information
- ➢ Managing actions like addition medical history, scheduling of appointments, and searching by name.
- ➢ Uses modular Python functions to keep operations organized.

3. **Storage Module**

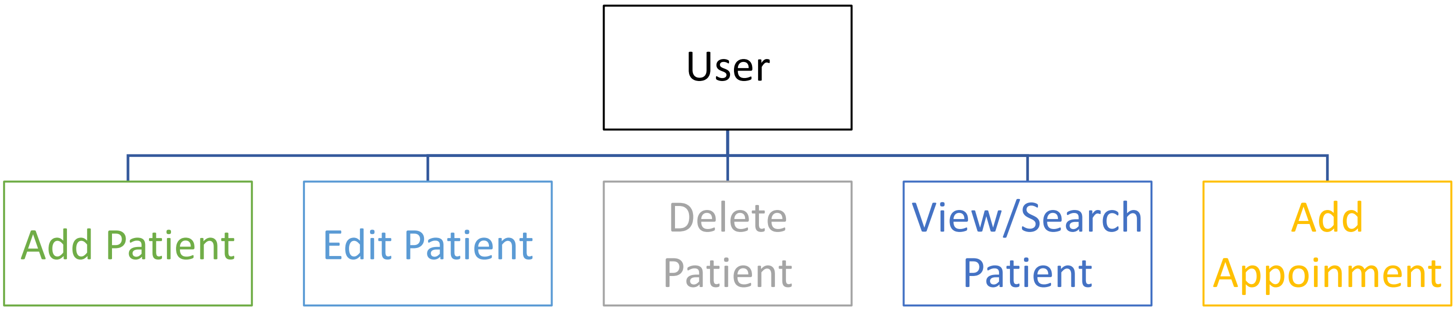
- Reads patient data from a text file (patients.txt)
- Writes new or updated patient records back to the file
- Maintains persistence so data remains even after the program is closed

4. **Architecture Flow:**

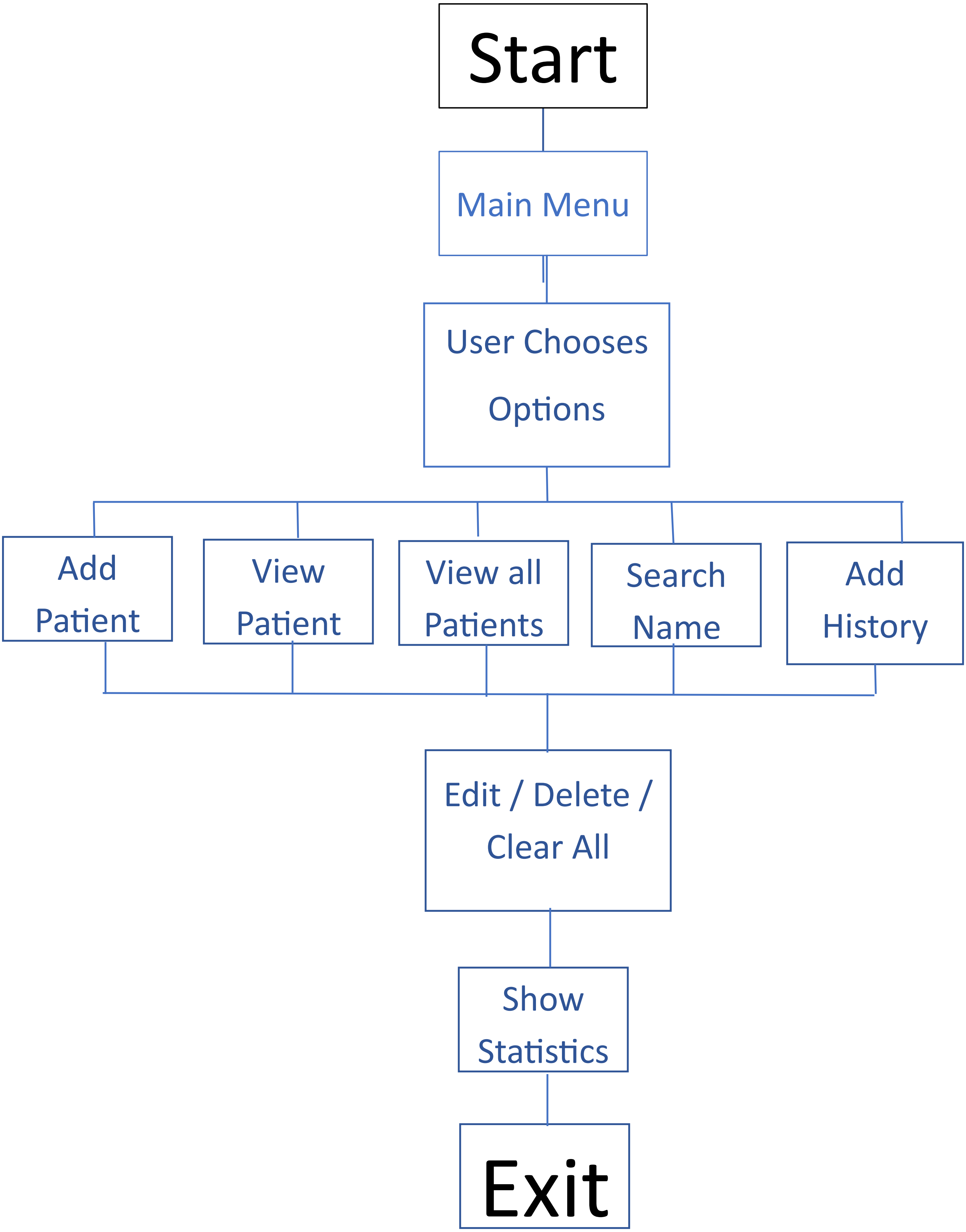
User → CLI → Clinic Logic → File Storage → Output to User

Design Diagrams

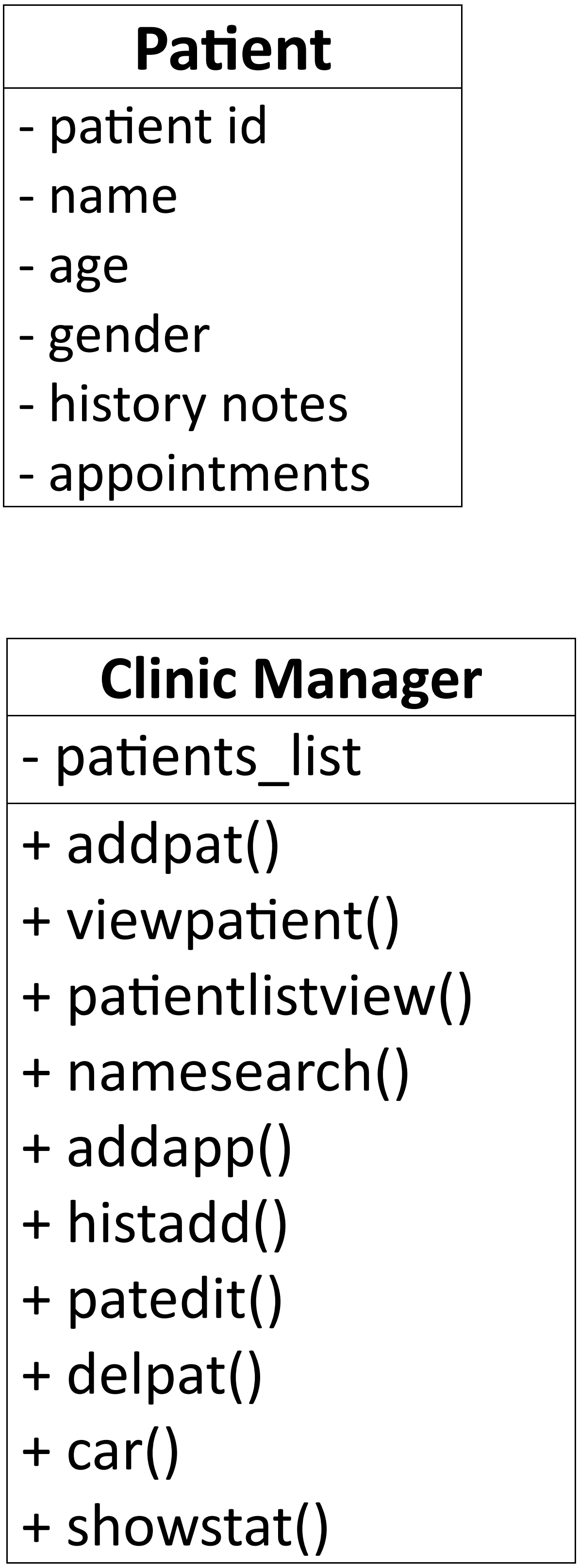
1. Use Case Diagram



2. Workflow Diagram



3. Class Diagram



Patient

- patient id
- name
- age
- gender
- history notes
- appointments

Clinic Manager

- patients_list

+ addpat()
+ viewpatient()
+ patientlistview()
+ namesearch()
+ addapp()
+ histadd()
+ patedit()
+ delpat()
+ car()
+ showstat()

Storage
➢ readal() ➢ writeal() ➢ ensurefe()

Algorithm

Step 1: Launch the Program

1.1 If patients.txt does not already exist, create it by calling ensurefe().

1.2 Show the main menu in an endless loop.

Step 2: Obtain User Preference

2.1 Ask the user to "enter choice."

2.2 Read the user's input.

Step 3: Execute the Choice-Based Operation

If option = 1 → Include the patient

3.1 Get the patient's information (name, age, gender, and ID).

3.2 Set appointment = "None" and history = "None" at initialization.

3.3 Add a single line containing all the information to patients.txt.

3.4 Show a message of success.

View Patient by ID if option = 2.

Choice = 2 → View Patient by ID

3.5 Read every line in the file.

3.6 Look for a patient ID that matches.

3.7 Show details if found; otherwise, display "not found."

Choice = 3 → View All Patients

3.8 Go through every line.

3.9 Divide the data and show the most basic information for each line.

If option = 4 → Look up the patient's name

3.10 Go through every line.

3.11 Change stored names and search terms to lowercase.

3.12 Display records containing the keyword.

If option = 5 → Make an appointment

3.13 Go through every line.
3.14 Look for an ID that matches.
3.15 Use "||" to add a new appointment as date-reason.
3.16 Write all of the data back into the file.

If option = 6 → Include a History Note

3.17 Go through every line.
3.18 Look for an ID that matches.
3.19 Use "||" to append new history text.
3.20 Rewrite every piece of information to the file.

If option = 7 → Modify Patient Information

3.21 Go through every line.
3.22 Look for an ID that matches.
3.23 Let the user change their gender, age, or name (or leave them as they are).
3.24 Update the data.

If option = 8 → Delete Patient

3.25 Go through every line.
3.26 Remove line with the provided ID.
3.27 Rewrite updated list to the file.

If choice = 9 → Clear All Records

3.28 Ask for confirmation.
3.29 If confirmed, write an empty list to the file.

If choice = 10 → Show Statistics

3.30 Count total patients, male count, female count.
3.31 Display results.

If choice = 11 → Exit Program

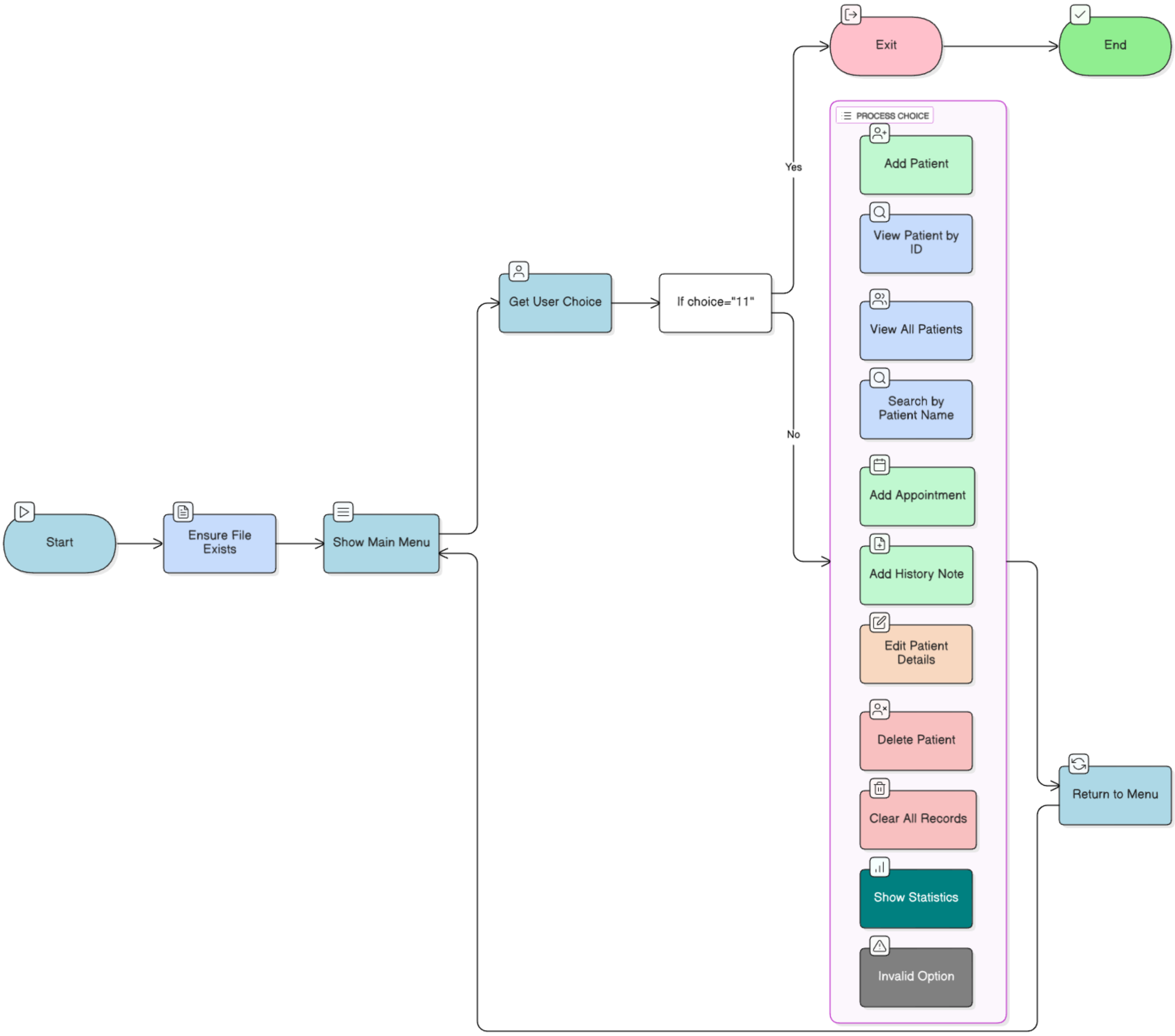
3.32 Break the loop and terminate.

Step 4: Handle Invalid Input

4.1 If choice does not match any option, display error message and show menu again.

Step 5: End Program

Flowchart


Start
Ensure File Exists
Show Main Menu
Get User Choice
If choice="11"
Yes
No
Exit
End
PROCESS CHOICE
Add Patient
View Patient by ID
View All Patients
Search by Patient Name
Add Appointment
Add History Note
Edit Patient Details
Delete Patient
Clear All Records
Show Statistics
Invalid Option
Return to Menu

Design Decisions & Rationale

- **Text-based file storage:** A text file (.txt) with records was chosen because it avoids the complexity of databases and is easy to read.

- **Modular function based structure:** The program is divided into clear functions such as addpat(), patsearch(), editpat(), etc. This improves readability and makes the code easy to maintain or expand.

- **CLI-based design:** **A** Command-Line Interface (CLI) is used to keep the application lightweight and easy to run. input() and print() are the major interactions .

- **Manual patient IDs instead of UUID:** The system uses user-entered patient IDs rather than auto-generated UUIDs to keep the workflow simple and allow the clinic to follow its existing ID format.

Implementation Details

Technologies Used

- **Python 3** — Core programming language used for all logic and CLI.
- **Text file storage (patients.txt)** — Data is stored as pipe-separated records instead of JSON.
- **Basic File Handling** — open(), read(), write() for storing and updating records.
- **Procedural Programming** — The system uses functions rather than classes or OOP.

Modules Developed

File Handling Module

- ensurefe()
- readal()
- writeal()

  Handles creation, loading, and saving of data.

Patient Management Module

- ➢ addpat()
- ➢ patedit()
- ➢ delpat()
- ➢ viewpatient()
- ➢ patientlistview()

  Core operations for managing patient records.

Search & Update Module

- ➢ namesearch()
- ➢ addhis()
- ➢ addapp()

  Enables searching and updating patient-related details.

Statistics Module

- ➢ showstat()

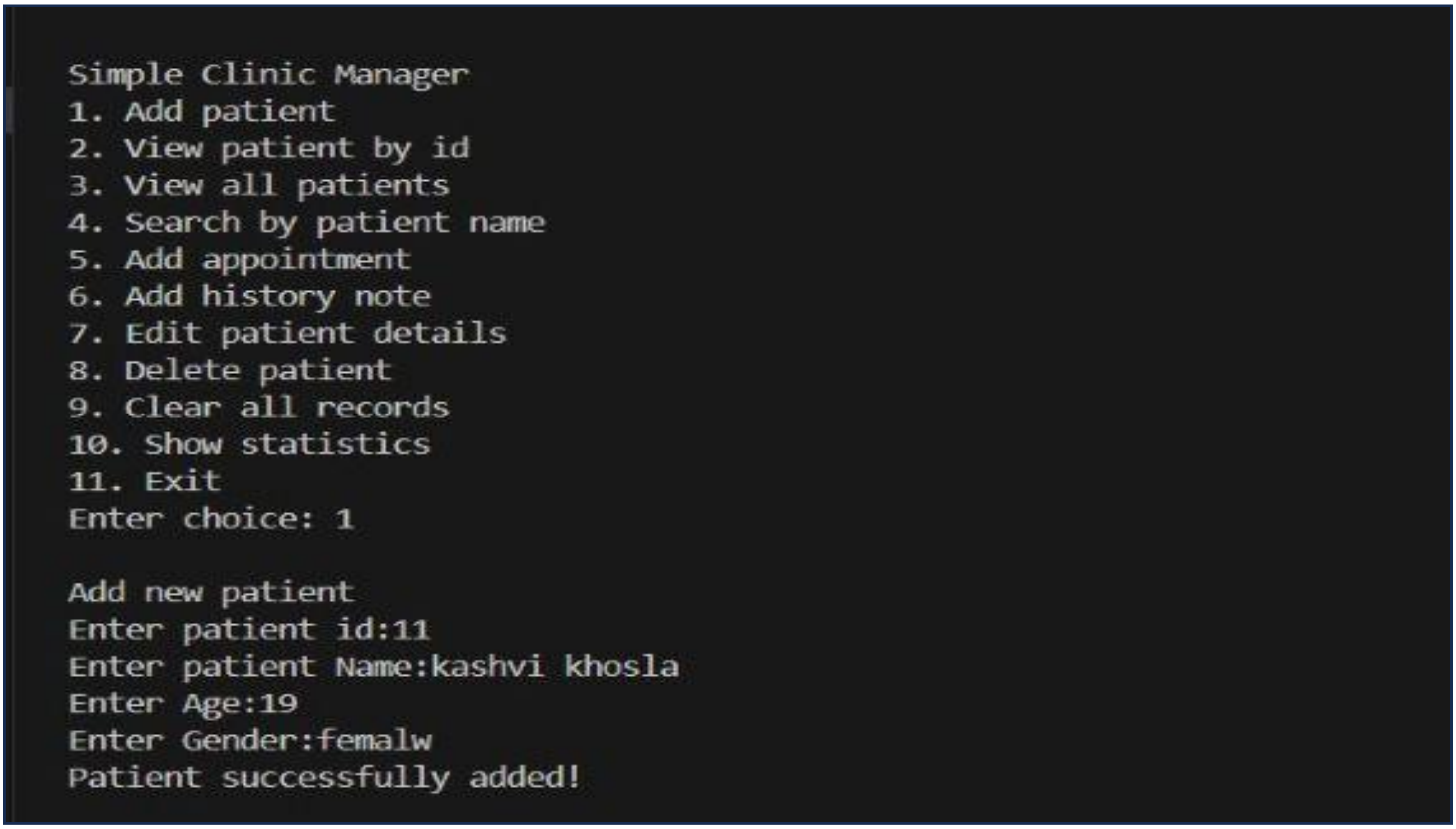
  Displays basic analytics such as total male/female patients.

Main CLI Module

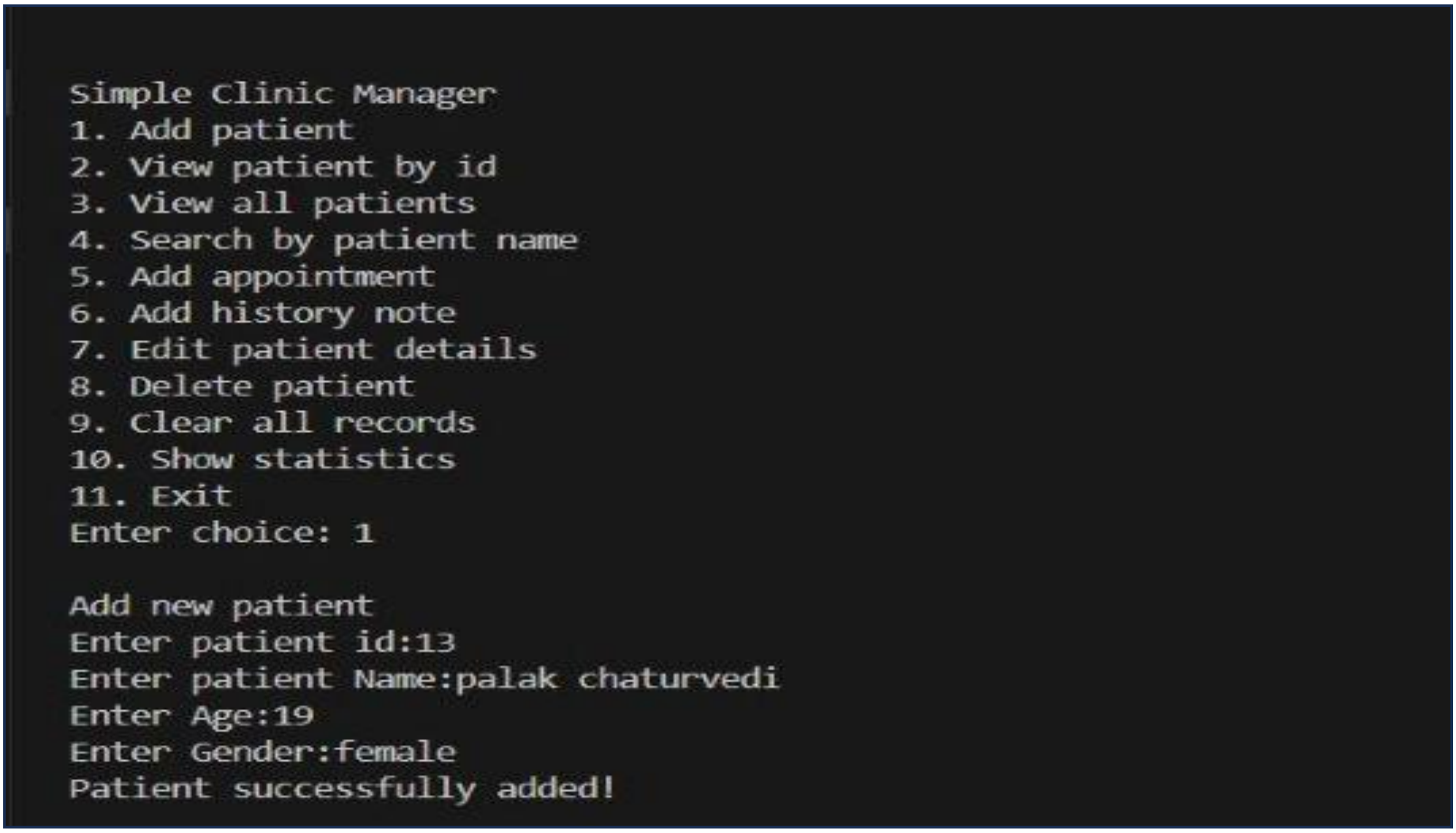
- ➢ main()

  Provides the menu system and user interaction loop.

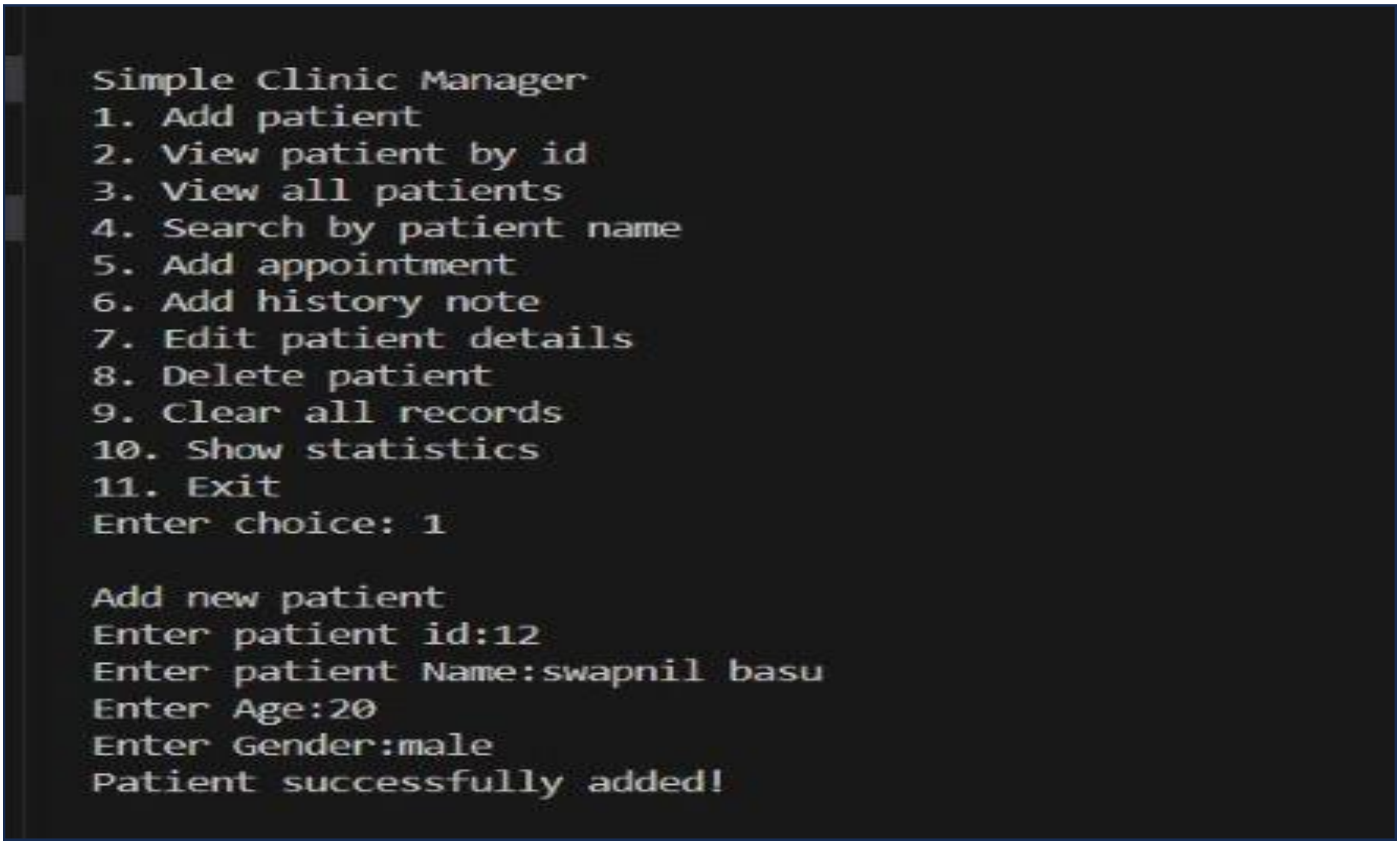
Screenshots / Output



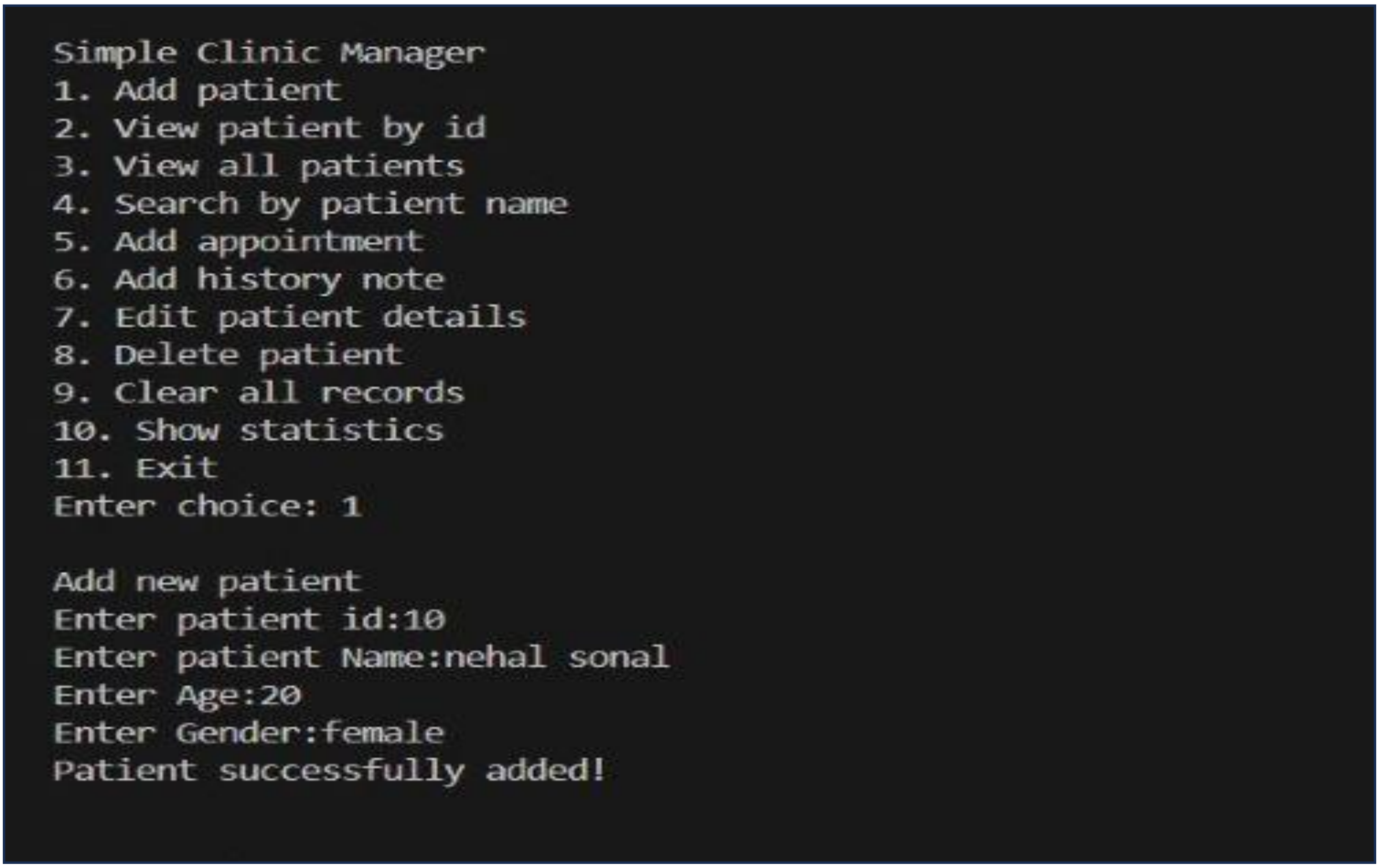
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 1

Add new patient
Enter patient id:11
Enter patient Name:kashvi khosla
Enter Age:19
Enter Gender:femalw
Patient successfully added!
```



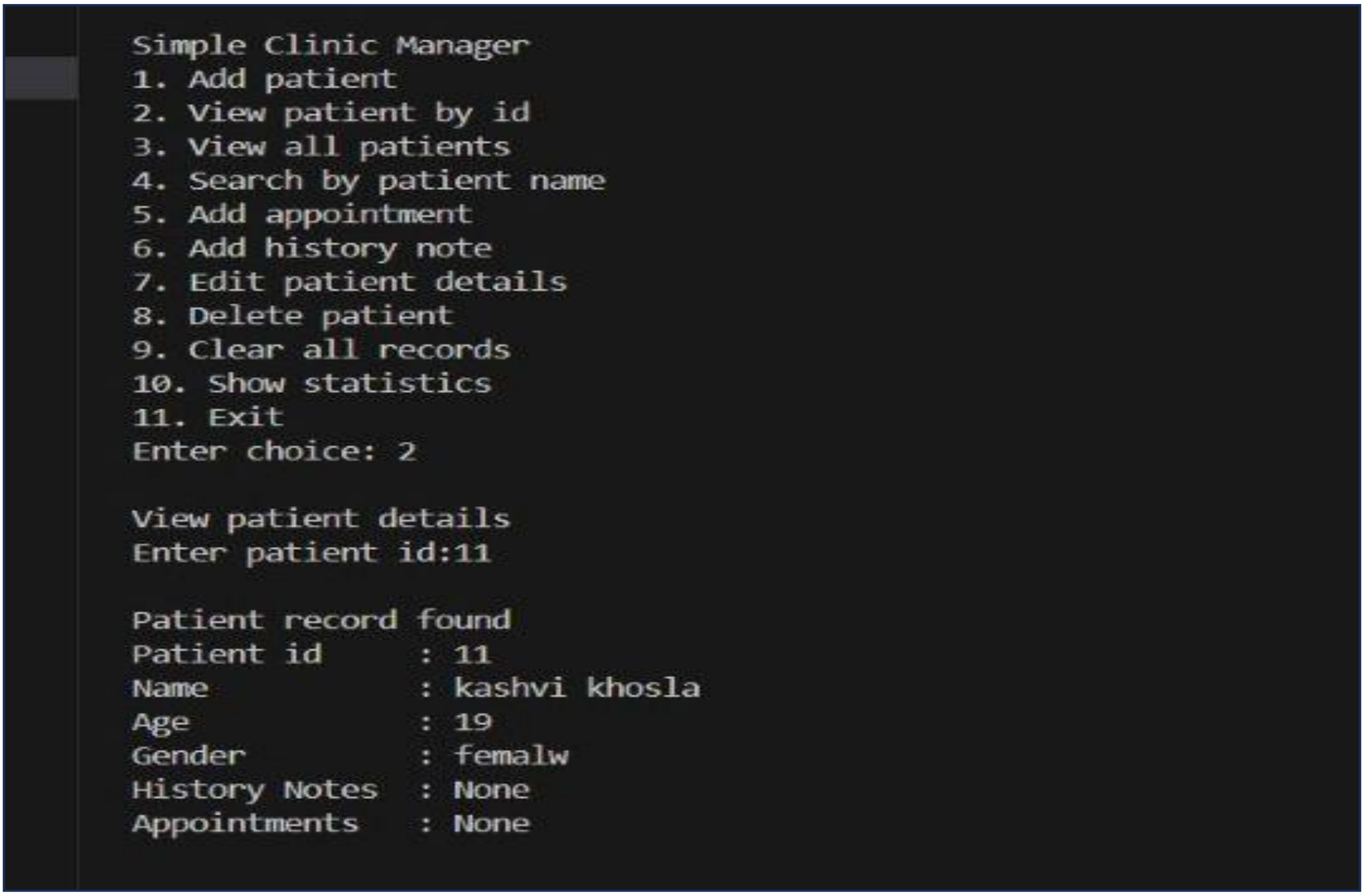
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 1

Add new patient
Enter patient id:13
Enter patient Name:palak chaturvedi
Enter Age:19
Enter Gender:female
Patient successfully added!
```



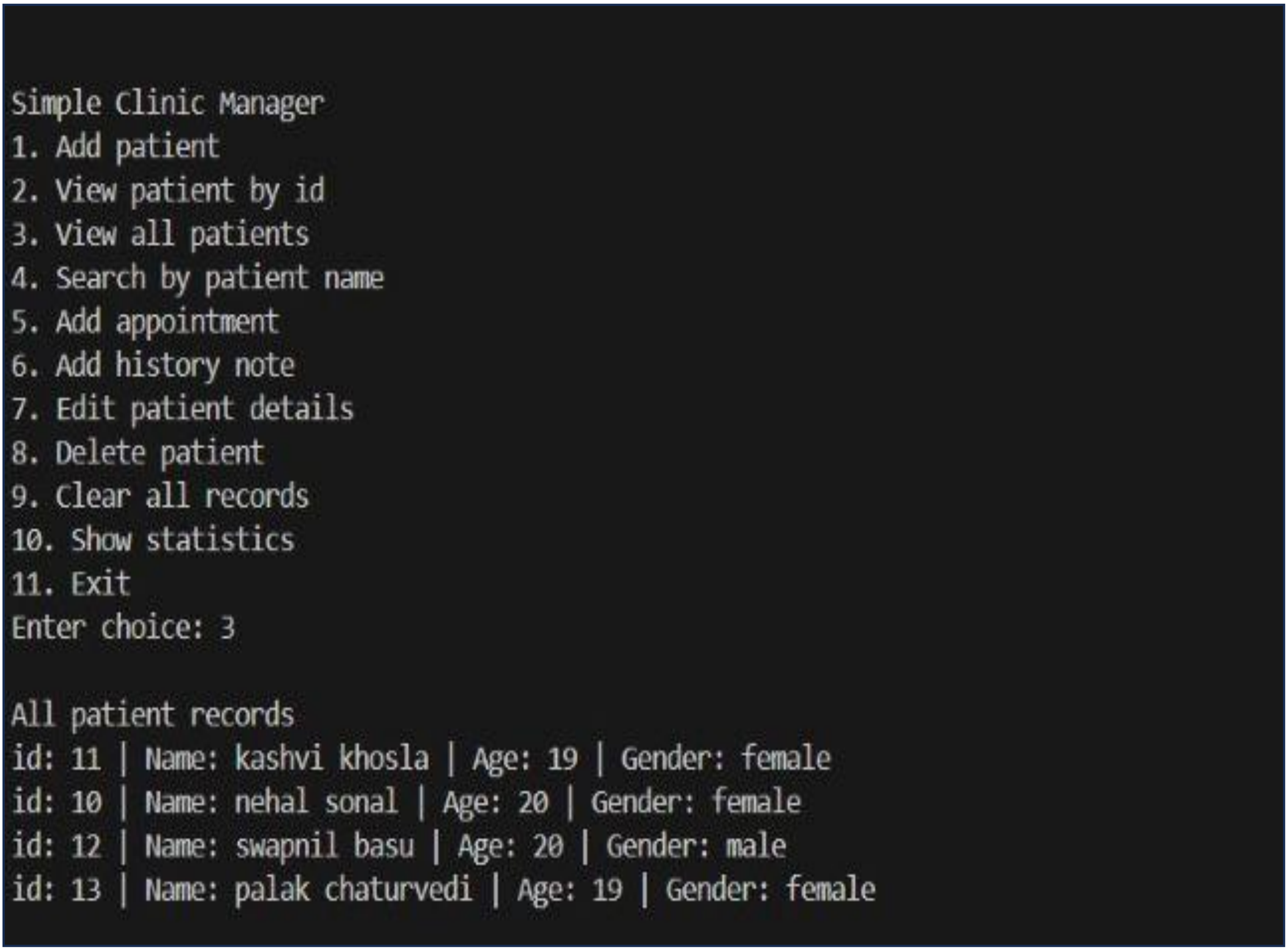
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 1

Add new patient
Enter patient id:12
Enter patient Name:swapnil basu
Enter Age:20
Enter Gender:male
Patient successfully added!
```



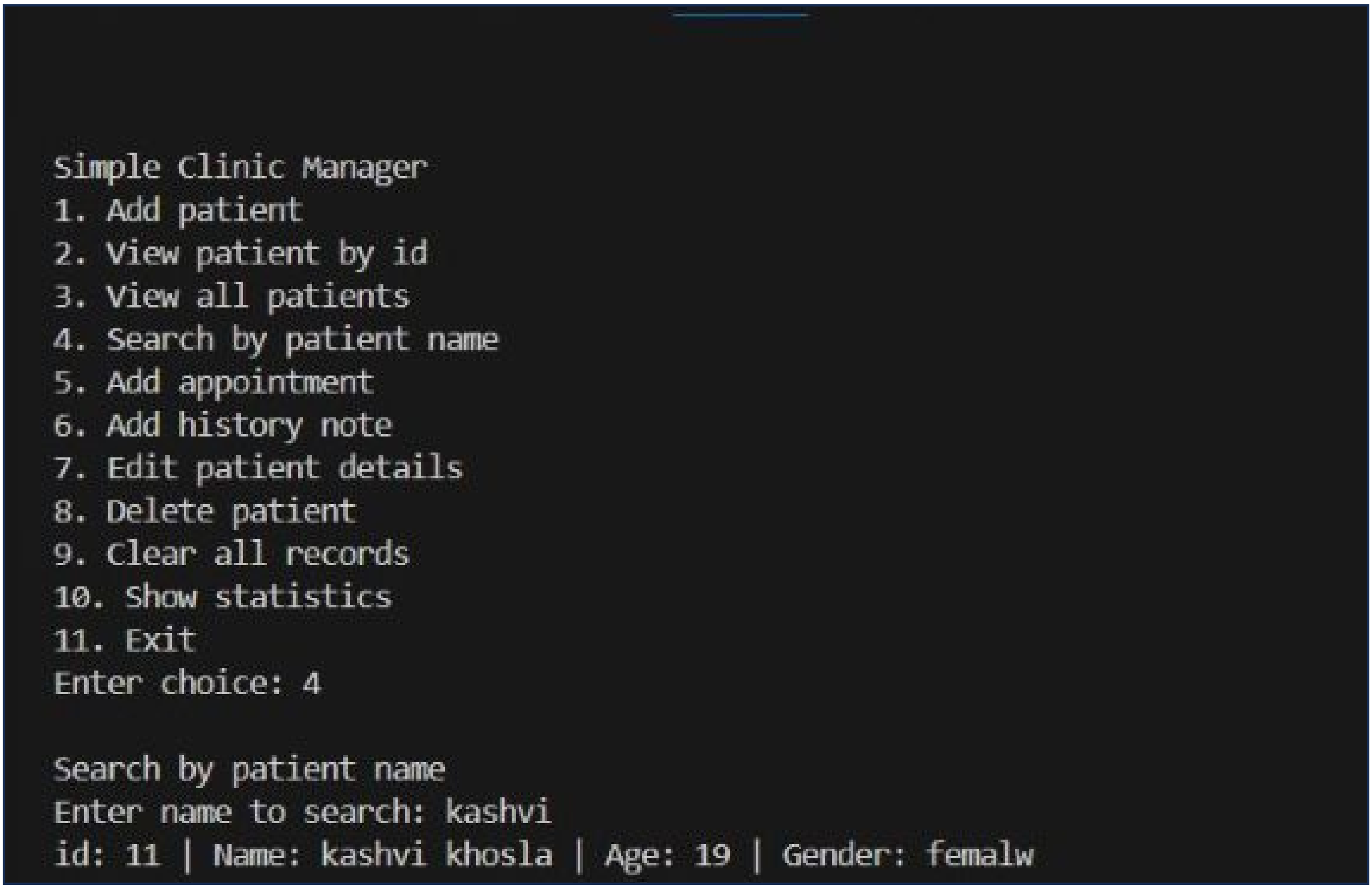
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 1

Add new patient
Enter patient id:10
Enter patient Name:nehal sonal
Enter Age:20
Enter Gender:female
Patient successfully added!
```



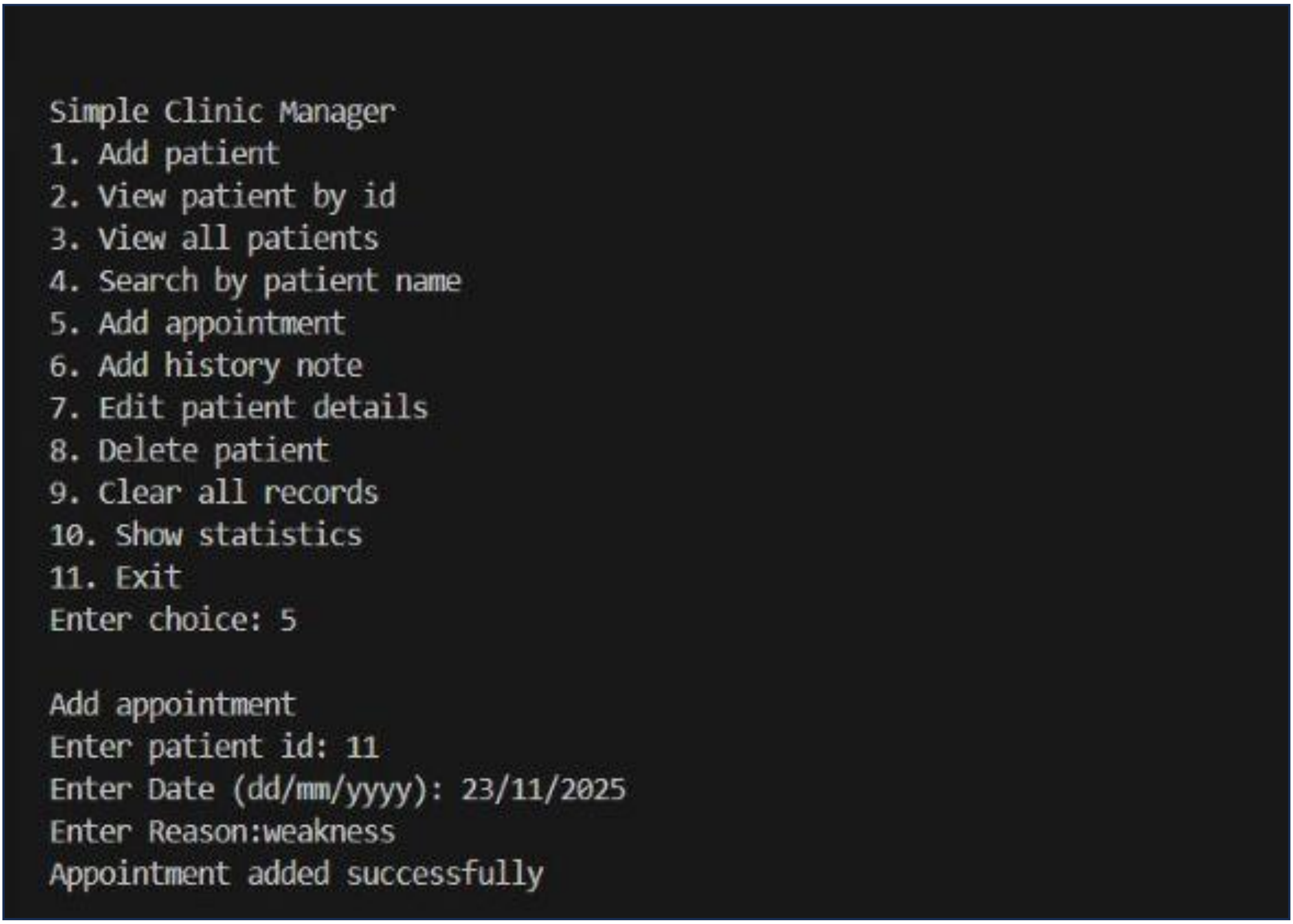
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 2

View patient details
Enter patient id:11

Patient record found
Patient id     : 11
Name           : kashvi khosla
Age            : 19
Gender         : femalw
History Notes  : None
Appointments   : None
```



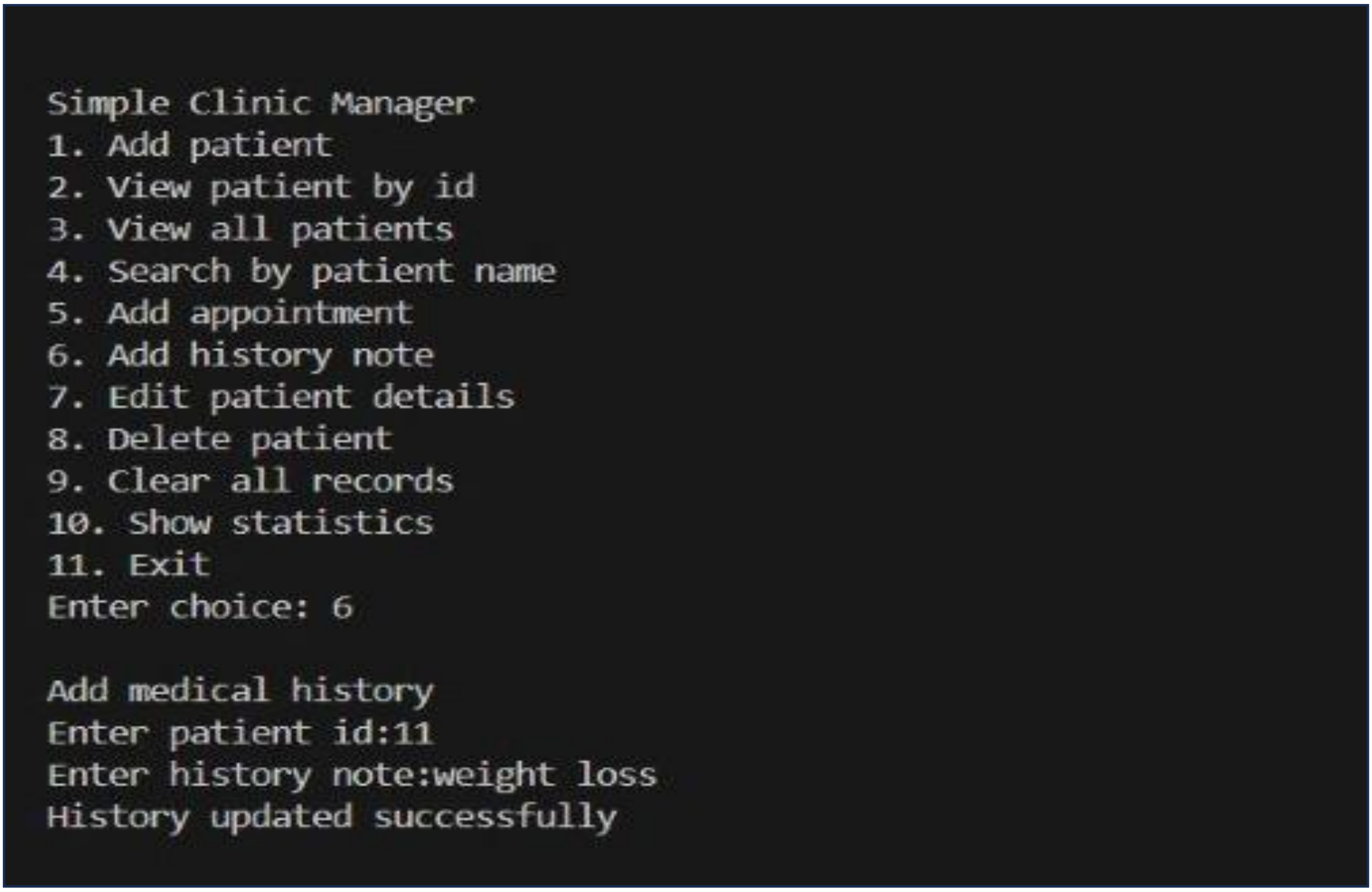
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 3

All patient records
id: 11 | Name: kashvi khosla | Age: 19 | Gender: female
id: 10 | Name: nehal sonal | Age: 20 | Gender: female
id: 12 | Name: swapnil basu | Age: 20 | Gender: male
id: 13 | Name: palak chaturvedi | Age: 19 | Gender: female
```



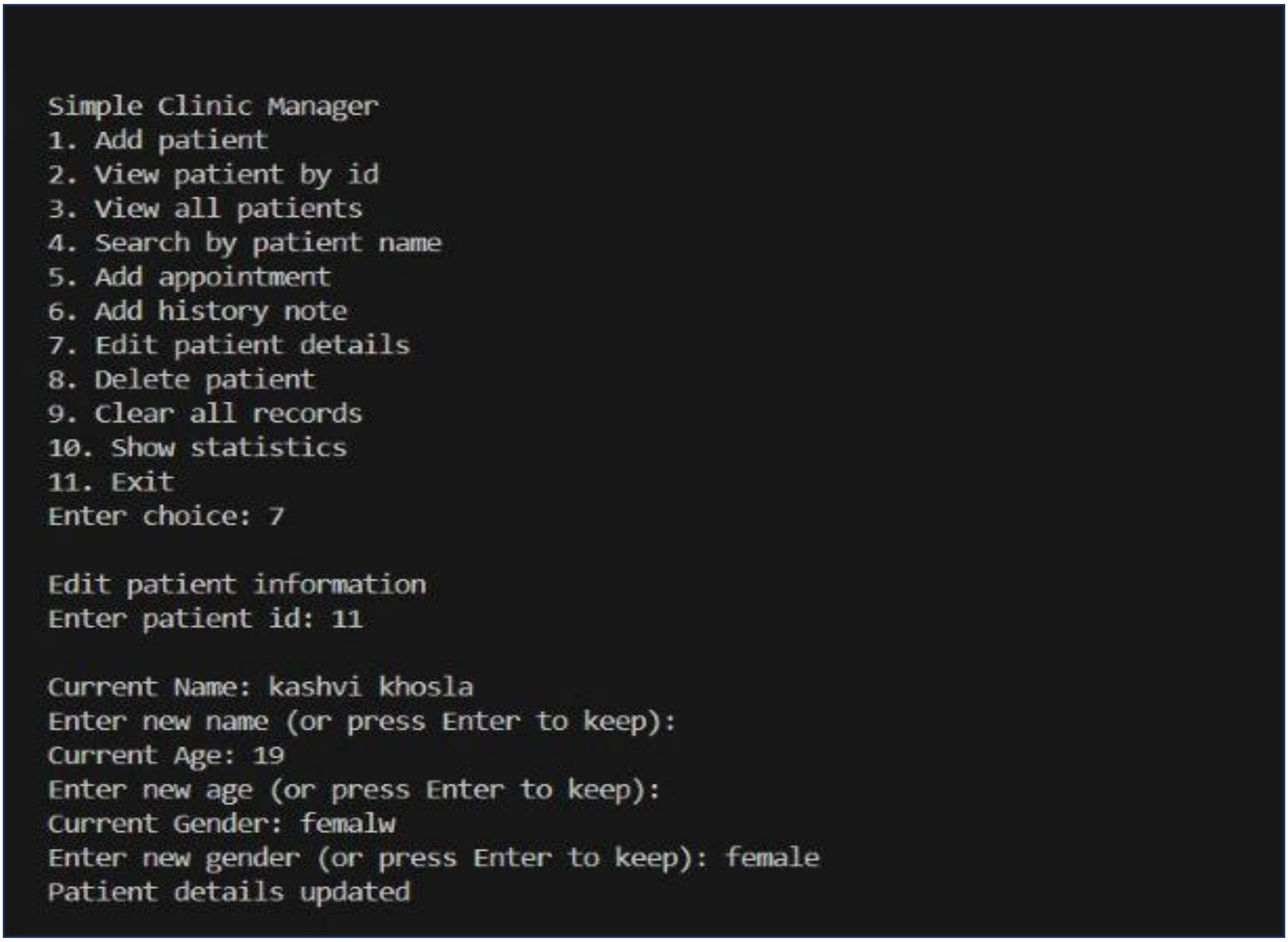
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 4

Search by patient name
Enter name to search: kashvi
id: 11 | Name: kashvi khosla | Age: 19 | Gender: femalw
```



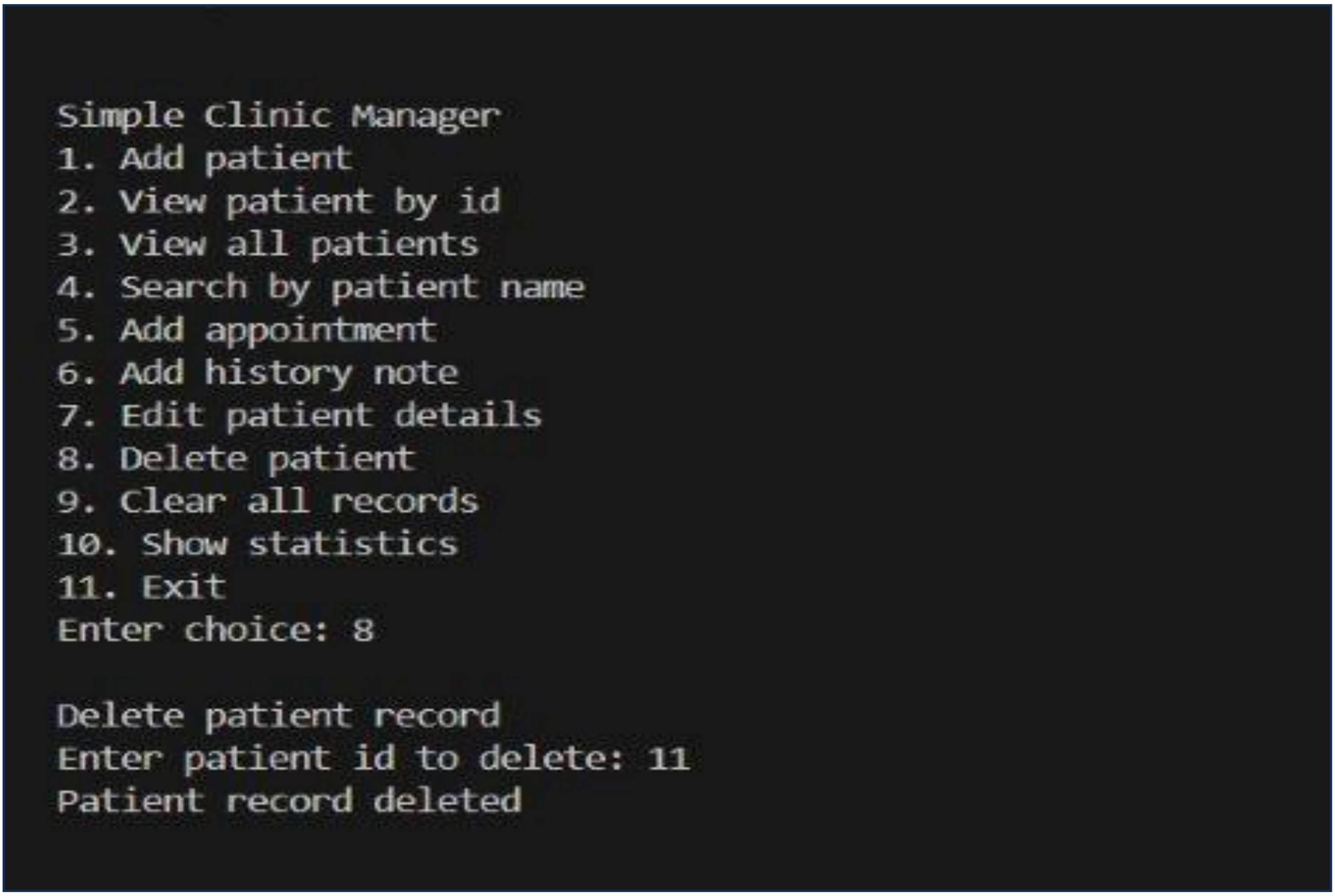
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 5

Add appointment
Enter patient id: 11
Enter Date (dd/mm/yyyy): 23/11/2025
Enter Reason:weakness
Appointment added successfully
```



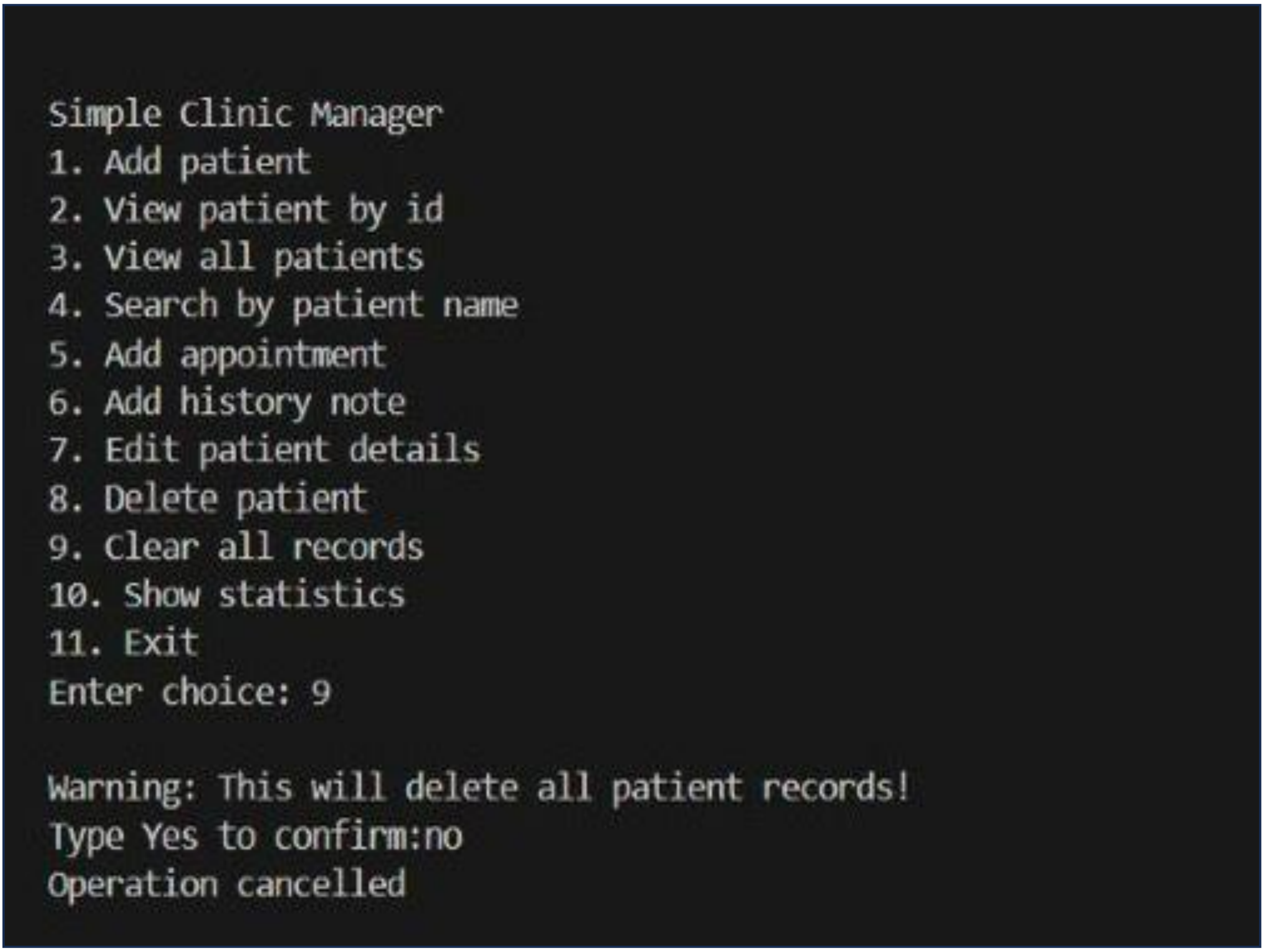
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 6

Add medical history
Enter patient id:11
Enter history note:weight loss
History updated successfully
```



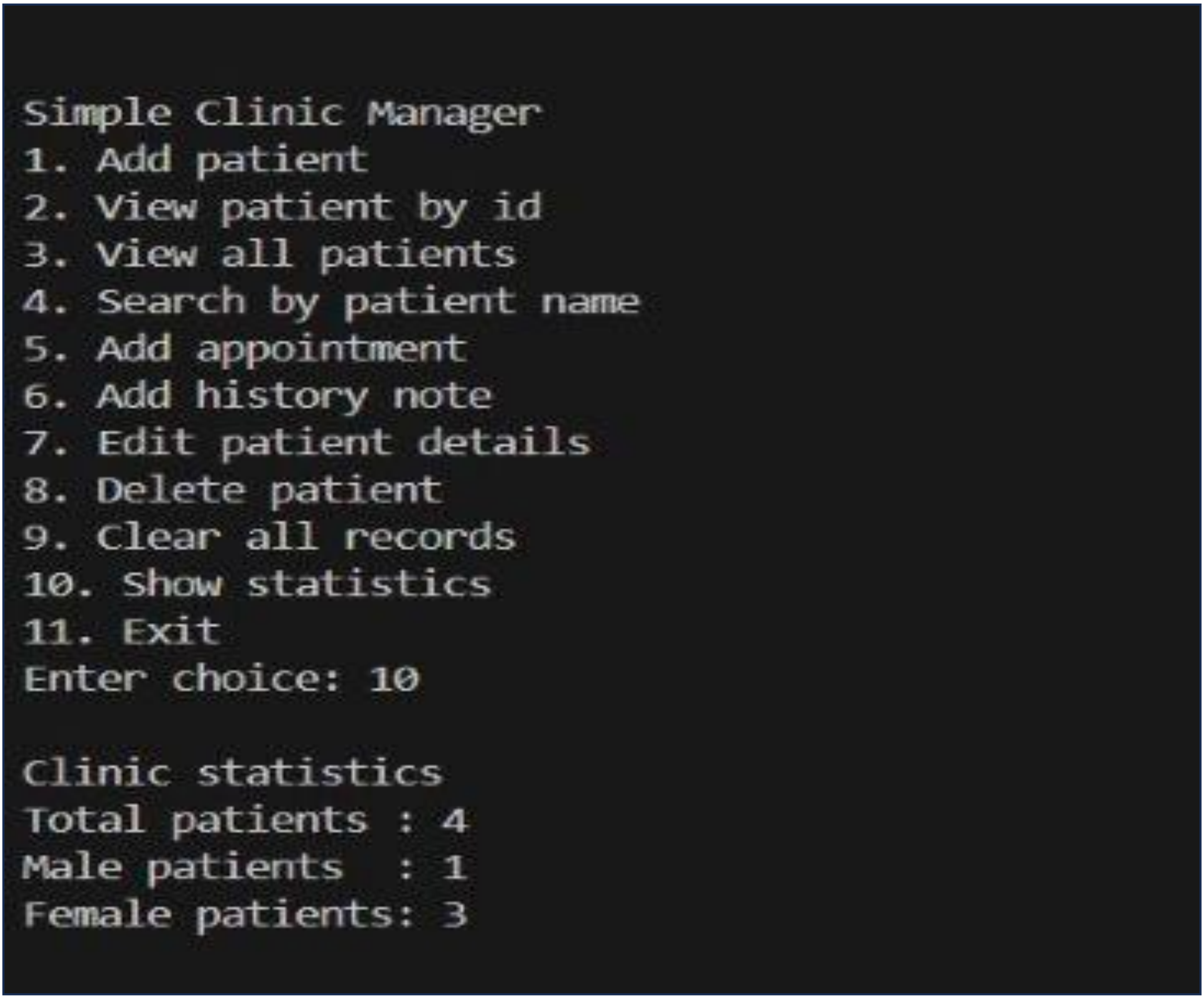
```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 7

Edit patient information
Enter patient id: 11

Current Name: kashvi khosla
Enter new name (or press Enter to keep):
Current Age: 19
Enter new age (or press Enter to keep):
Current Gender: femalw
Enter new gender (or press Enter to keep): female
Patient details updated
```



```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 8

Delete patient record
Enter patient id to delete: 11
Patient record deleted
```



```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 9

Warning: This will delete all patient records!
Type Yes to confirm:no
Operation cancelled
```



```
Simple Clinic Manager
1. Add patient
2. View patient by id
3. View all patients
4. Search by patient name
5. Add appointment
6. Add history note
7. Edit patient details
8. Delete patient
9. Clear all records
10. Show statistics
11. Exit
Enter choice: 10

Clinic statistics
Total patients : 4
Male patients  : 1
Female patients: 3
```

Testing Approach

- To make sure that functions like adding patients, viewing records, searching by name, editing details, and deleting records functioned properly, each option was tested separately.
- To make sure the program handled errors without crashing, invalid inputs were tested, such as entering an incorrect patient ID, typing an empty name, or providing incorrect menu choices.
- To verify the accuracy of the data displayed, several patient records were added and then viewed using various functions (view by ID, view all, search by name).
- To ensure that updated data was saved correctly and that deleting one record did not impact other records, editing and deleting operations were tested.
- To verify that all patient data was accurately saved in the patients.txt file, program persistence was examined by restarting the application several times.

Challenges Faced

Although working on this Clinic Manager project gave me invaluable practical experience, there were a number of difficulties.

Managing inaccurate or unexpected user input was one of the main challenges. Occasionally, users typed values in the incorrect format, left fields empty, or entered invalid patient IDs. Careful input validation was necessary to ensure that the program did not crash and instead displayed unambiguous error messages.

Managing data within a plain text file presented another difficulty. Updating a single field required reading every line, changing the correct record, and then rewriting everything back to the file because patient information was stored as pipe-separated strings. To prevent data corruption or unintentionally overwriting records, this procedure required accuracy.

There were additional difficulties when working without OOP or separate modules. It took more work to keep the code organized and prevent duplication because the program is written using procedural functions. To maintain dependability, it was crucial to make sure that every function interacted with the text file format consistently. Despite these challenges, overcoming them strengthened knowledge of file handling, error control, and structured program design while also enhancing system stability.

Learnings & Key Takeaways

This Small Clinic Management System project provided a number of significant conceptual and technical insights.
Gaining a deeper understanding of Python file handling, particularly how to safely read, update, and rewrite records stored in a plain text file, was one important lesson learned. Working with structured text-based storage systems became more comfortable when data was managed in a straightforward "pipe-separated" format.
The significance of clean, procedural program design was another important lesson learned. Keeping the code organized through distinct functions was crucial because the project does not use classes or modules. This enhanced comprehension of code reuse, function flow, and preserving clarity in a non-OOP program.

Additionally, the project improved user interaction skills via a Command-Line Interface (CLI).

Creating user-friendly menus, providing step-by-step instructions, and managing incorrect input all demonstrated the value of considering the viewpoint of the user. One important practical lesson was making sure the program doesn't crash even when the user enters unexpected values. Lastly, planning the workflow, creating documentation, and creating diagrams prior to coding demonstrated how crucial it is to properly structure a project. As a result, later confusion was lessened and the development proceeded more smoothly.

Overall, the project strengthened practical problem-solving skills and offered solid foundational knowledge of file handling, Python, and creating basic management systems.

Future Enhancements

While adding patients, viewing records, updating details, and scheduling appointments are all handled by the Small Clinic Management System's current version, there are a number of enhancements that could make the system more robust and user-friendly in the future.
Adding a graphical user interface would be a significant improvement since it would facilitate navigation for clinic employees who might not be familiar with command-line inputs.
Moving from a simple text file to a more structured storage method like JSON or even a lightweight database like SQLite would be another beneficial upgrade that would enable faster searching and more secure data handling.

The precision and speed of information retrieval could be enhanced by features like advanced search, sorting by name or age, appointment reminders, and patient history filters.
In addition to preventing unwanted access, a staff login system would enable various clinic employees

to handle records on their own.

Long-term, the system would be more useful for actual clinical use if features like printable reports, automatic appointment notifications, and patient data exporting were added.

References

- **Python Official Documentation** — for understanding input handling, file operations, and basic program structure
- **GeeksforGeeks — Python File Handling** — helped in implementing reading, writing, and updating the text file
- **W3Schools — Python Basics** — useful for refreshing fundamental Python concepts used in the project
- **Ai tools (chat gpt, gemini) for reference**

THANK YOU