HOSPITAL MANAGEMENT SYSTEM

A MINI-PROJECT REPORT

Submitted by

PRANATHEE H 241901078

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING AND CYBER SECURITY

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI NOVEMBER 2025

BONAFIDE CERTIFICATE

Certified that this project **"HOSPITAL MANAGEMENT SYSTEM"** is the bonafide work of **PRANATHEE H** who carried out the project work under my supervision.

SIGNATURE
Ms. M.FOWZIA SIHANA
ASSISTANT PROFESSOR(SS)

Department of Computer Science and Engineering (Cybersecurity),
Rajalakshmi Engineering College
Chennai

This mini project report is submitted for the viva voce examination to be held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

The project titled **"Hospital & Patient Management System"** is a lightweight, database-driven desktop application developed using **Python Tkinter** for the graphical user interface and **SQLite** for backend data storage. The system is designed to streamline the management of patient records within a healthcare or clinical environment, reducing manual paperwork and improving the accuracy and accessibility of patient information.

The application provides a structured and user-friendly interface for entering and maintaining patient details such as name, age, gender, contact information, diagnosis, address, and admission date. It supports complete **CRUD operations**—including adding new patients, updating existing records, deleting entries, and retrieving patient data from the database. A built-in **search mechanism** allows users to quickly locate patients using criteria such as name, phone number, or ID, while a **sorting feature** enables efficient organization of displayed records.

To enhance usability, the system includes an interactive **Tkinter Treeview table** for viewing patient data in a clean, tabulated format. Users can also **export patient lists to CSV**, allowing easy sharing and backup of records. Input validation is integrated throughout the system to ensure data integrity, such as validating age limits, phone number formats, and date structures.

Overall, the Hospital & Patient Management System provides a reliable, efficient, and accessible approach to managing patient records. Its combination of a simple modern interface, persistent data storage, and essential management features makes it suitable for small hospitals, clinics, or educational demonstrations of database-based GUI applications.

ACKNOWLEDGEMENT

We express our sincere thanks to our beloved and honorable chairman **MR. S. MEGANATHAN** and the chairperson **DR. M. THANGAM MEGANATHAN** for their timely support and encouragement.

We are greatly indebted to our respected and honorable principal **Dr. S.N. MURUGESAN** for his able support and guidance.

No words of gratitude will suffice for the unquestioning support extended to us by our Head Of The Department **Mr. BENEDICT J N** for being ever supporting force during our project work

We also extend our sincere and hearty thanks to our internal guide **Ms. FOWZIA SIHANA,** for her valuable guidance and motivation during the completion of this project.

Our sincere thanks to our family members, friends and other staff members of computer science engineering and cybersecurity.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The **Hospital & Patient Management System** is a simple software tool that helps hospitals manage patient records digitally. It replaces manual paper-based methods by storing patient details such as age, gender, phone number, diagnosis, and admission date. The system is built using **Python Tkinter** for the interface and **SQLite** for the database, making it easy to use and reliable for daily hospital operations.

1.2 SCOPE OF THE WORK

This system focuses on providing an efficient way to add, update, search, and delete patient records. It displays all data in a clear table and allows users to sort or search for specific patients. The application also supports exporting patient data to **CSV files**, making it useful for reports and documentation. It is suitable for small hospitals, clinics, and medical offices.

1.3 PROBLEM STATEMENT

Managing patient records manually can lead to mistakes, lost files, and slow processing. Hospitals often struggle to keep information organized and updated. This project solves these problems by offering a simple digital system that stores all patient records safely and makes them easy to access whenever needed.

1.4 AIM AND OBJECTIVES OF THE PROJECT

The main aim of this project is to make hospital record management faster and more accurate. The objectives are:

- To store patient information in an organized way.
- To simplify updating and retrieving patient records.
- To improve accuracy by reducing manual errors.
- To allow exporting patient data for reporting.
- To make record-keeping quicker and more efficient.

CHAPTER 2

SYSTEM SPECIFICATIONS

2.1 HARDWARE SPECIFICATIONS

Processor	:	Intel i5
Memory Size	:	8GB (Minimum)
HDD	:	1 TB (Minimum)

2.2 SOFTWARE SPECIFICATIONS

Operating System	:	WINDOWS 10
Front – End	:	Python
Back - End	:	SQLite
Language	:	python,SQLite

CHAPTER 3

MODULE DESCRIPTION

3.1 Patient Management Module

This module allows the user to add, update, or delete patient records. It captures details such as name, age, gender, phone number, diagnosis, and admission date. The system ensures that required fields are properly validated before storing the data in the database.

3.2 Database Management Module

This module handles all communication with the SQLite database. It creates the patient table if it does not exist and manages the insert, update, delete, and search operations. Patient information such as ID, contact details, and diagnosis is stored securely and retrieved efficiently when needed.

3.3 Search & Sorting Module

The user can search for patients using their **name, phone number, or ID**, allowing quick access to records. The module also provides sorting features, enabling the user to arrange the patient list by ID, name, age, gender, phone, or admission date with a single click on the column headers.

3.4 Form Validation Module

This module checks all input fields before saving patient data. It ensures that
The name is not empty, the age is valid, the phone number contains digits only,
and the admission date follows the correct format. These validations help prevent
incorrect entries and maintain accurate records.

3.5 Export Module

This module allows exporting patient records into a CSV file. The exported file includes:

- Patient ID and basic details
- Contact information
- Admission date
- Diagnosis summary
- All data currently displayed in the table view

The export feature helps hospitals generate reports for documentation, backup,
andadministrative purposes.

CHAPTER 4

IMPLEMENTATION AND WORKING

4.1 Tools Used

- **Python:** For application logic and GUI
- **SQLite3:** For database management
- **Tkinter:** For graphical interface
- **CSV module:** For exporting data

4.2 Working Procedure

1. The user launches the application.
2. Patient details are entered.
3. The user can add, update, delete, or search for patient records.
4. The system displays all records in a tab with sorting and selection features.
5. Results can be viewed and exported as CSV reports.

4.3 Data Validation Process

The system validates user input before storing it in the database. It checks for required fields, ensures age and phone number formats are correct, and verifies that the admission date follows the proper format. This helps maintain accuracy and prevents invalid entries.

CHAPTER 5

CODING

```
import sqlite3
import tkinter as tk
from tkinter import ttk, messagebox
import csv

# ---------------------- Database Setup ----------------------
conn = sqlite3.connect("hospital.db")
cur = conn.cursor()

cur.execute("""
CREATE TABLE IF NOT EXISTS patient(
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT, age INTEGER, gender TEXT, phone TEXT, diagnosis TEXT,
        admit TEXT
)""")

# ---------------------- Tkinter UI ----------------------
root = tk.Tk()
root.title("Hospital Management System (Sample)")
root.geometry("700x500")

tabs = ttk.Notebook(root)
tabs.pack(fill="both", expand=True)
```

```
# ---------------------- Tab 1: Add Patient ----------------------
tab1 = ttk.Frame(tabs)
tabs.add(tab1, text="Patient Entry")

ttk.Label(tab1, text="Name").grid(row=0, column=0, padx=5, pady=5)
ttk.Label(tab1, text="Age").grid(row=1, column=0, padx=5, pady=5)
ttk.Label(tab1, text="Gender").grid(row=2, column=0, padx=5, pady=5)
ttk.Label(tab1, text="Phone").grid(row=3, column=0, padx=5, pady=5)
ttk.Label(tab1, text="Diagnosis").grid(row=4, column=0, padx=5, pady=5)
ttk.Label(tab1, text="Admit Date").grid(row=5, column=0, padx=5, pady=5)

ent_name = ttk.Entry(tab1)
ent_age = ttk.Entry(tab1)
ent_gender = ttk.Combobox(tab1, values=["Male","Female","Other"])
ent_phone = ttk.Entry(tab1)
ent_diag = ttk.Entry(tab1)
ent_date = ttk.Entry(tab1)

ent_name.grid(row=0,column=1)
ent_age.grid(row=1,column=1)
ent_gender.grid(row=2,column=1)
ent_phone.grid(row=3,column=1)
ent_diag.grid(row=4,column=1)
ent_date.grid(row=5,column=1)

def add_patient():
cur.execute("""
INSERT INTO patient(name,age,gender,phone,diagnosis,admit)


VALUES(?,?,?,?,?,?)""",(ent_name.get(),ent_age.get(),ent_gender.get(),ent_phone.get(),ent_
diag.get(),ent_date.get()))
conn.commit()
messagebox.showinfo("Success","Patient Added")
ttk.Button(tab1, text="Add Patient", command=add_patient).grid(row=6, column=1,
pady=10)
```

```
# ---------------------- Tab 2: Patient Records ----------------------
tab2 = ttk.Frame(tabs)
tabs.add(tab2, text="Records")

cols = ("id","name","age","gender","phone","diagnosis","admit")
tree = ttk.Treeview(tab2, columns=cols, show="headings")
for c in cols: tree.heading(c, text=c.upper())
tree.pack(fill="both", expand=True, pady=10)

search_entry = ttk.Entry(tab2)
search_entry.pack(pady=5)

def load_data():
    tree.delete(*tree.get_children())
    txt = f"%{search_entry.get()}%"
    cur.execute("""
    SELECT * FROM patient WHERE name LIKE ? OR phone LIKE ? OR CAST(id    AS
TEXT) LIKE ?""",(txt,txt,txt))
            for row in cur.fetchall(): tree.insert("", "end", values=row)
      ttk.Button(tab2, text="Search", command=load_data).pack()

  # ---------------------- Tab 3: Update/Delete ----------------------
  tab3 = ttk.Frame(tabs)
  tabs.add(tab3, text="Modify")

  ttk.Label(tab3, text="Patient ID").grid(row=0,column=0,padx=5,pady=5)
  ttk.Label(tab3, text="New Phone").grid(row=1,column=0,padx=5,pady=5)
  ttk.Label(tab3, text="New Diagnosis").grid(row=2,column=0,padx=5,pady=5)
  ent_uid = ttk.Entry(tab3)
  ent_uphone = ttk.Entry(tab3)
  ent_udiag = ttk.Entry(tab3)
  ent_uid.grid(row=0,column=1)
  ent_uphone.grid(row=1,column=1)
  ent_udiag.grid(row=2,column=1)

  def update_patient():
      cur.execute("""
      UPDATE patient SET phone=?, diagnosis=? WHERE id=?
      """,(ent_uphone.get(), ent_udiag.get(), ent_uid.get()))
      conn.commit()
      messagebox.showinfo("Updated","Record Updated")

  def delete_patient():
      cur.execute("DELETE FROM patient WHERE id=?", (ent_uid.get(),))
      conn.commit()
      messagebox.showinfo("Deleted","Record Deleted")
```

```
ttk.Button(tab3,text="Update",command=update_patient).grid(row=3,column=1,pady=8)
ttk.Button(tab3,text="Delete",command=delete_patient).grid(row=4,column=1,pady=8)
# ---------------------- Tab 4: Export ----------------------
tab4 = ttk.Frame(tabs)
tabs.add(tab4, text="Export")

def export_csv():
    cur.execute("SELECT * FROM patient")
    rows = cur.fetchall()
    with open("patients.csv","w",newline="") as f:
        w = csv.writer(f)
        w.writerow(cols)
        w.writerows(rows)
    messagebox.showinfo("Exported","CSV File Saved")

ttk.Button(tab4,text="Export to CSV",command=export_csv).pack(pady=20)

root.mainloop()
```

CHAPTER 6

SCREEN SHOTS

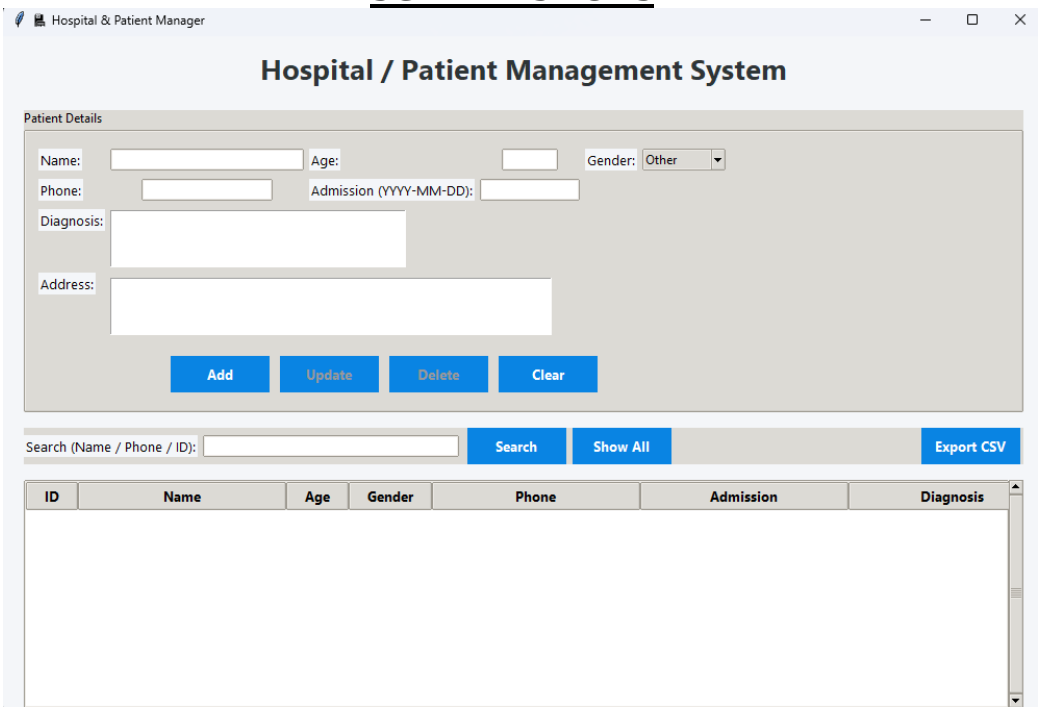

Fig 6.1 Introduction page

Patient details:

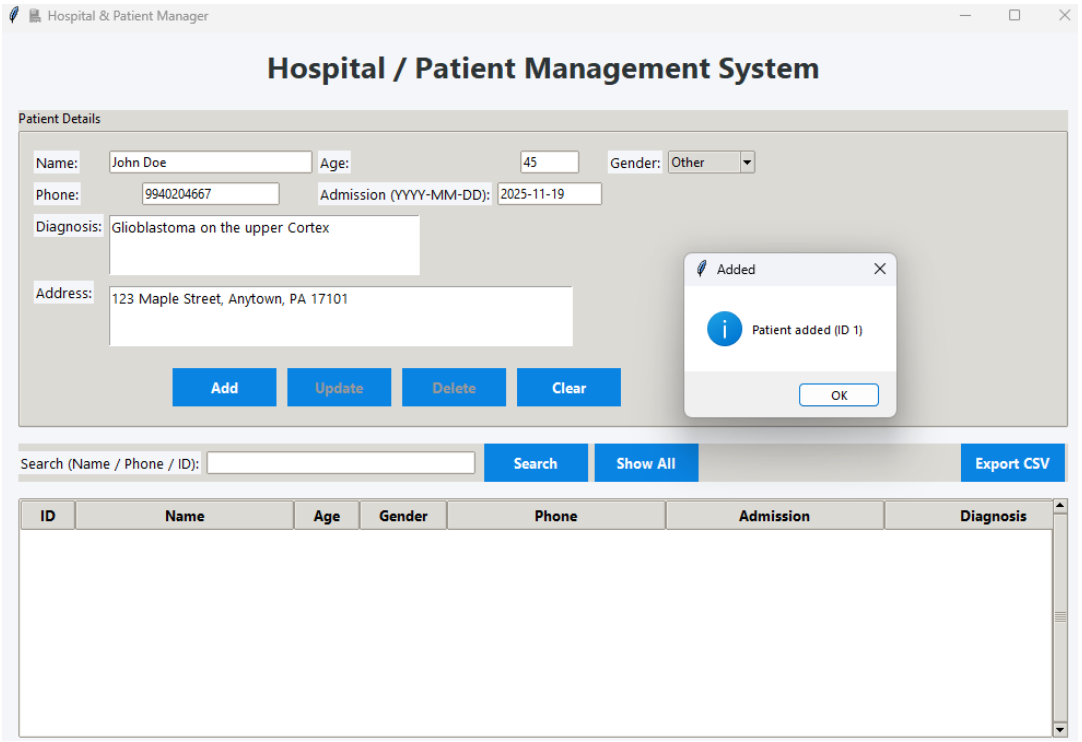

Fig 6.2 (a) Adding Patient details

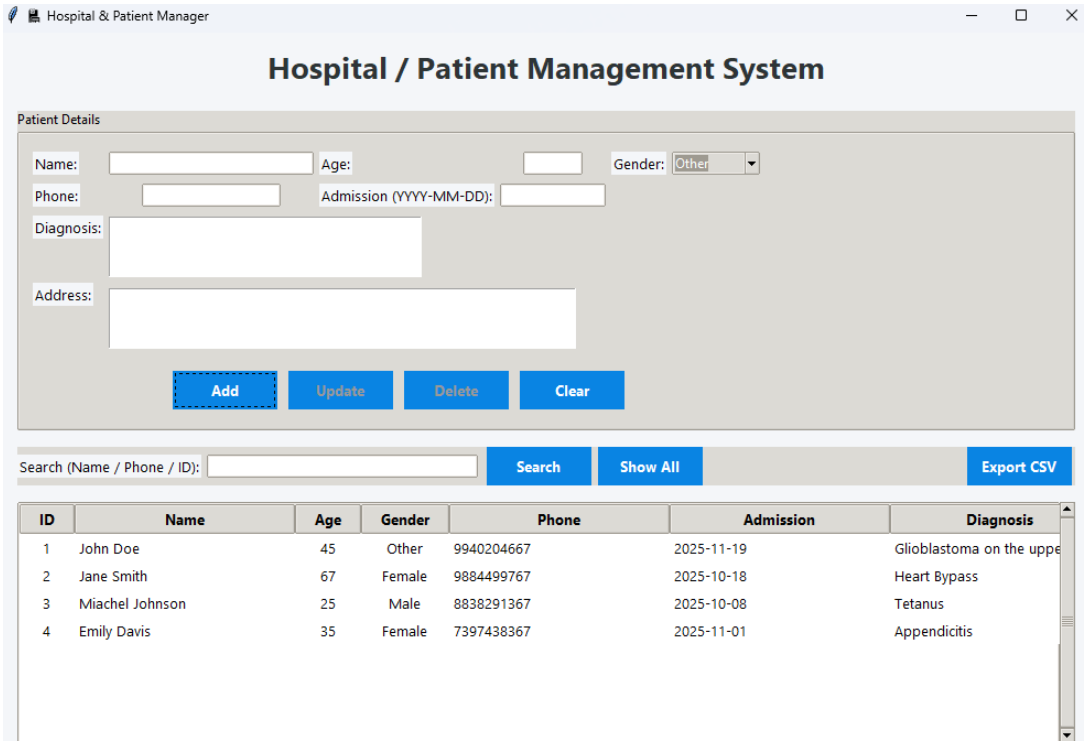

Fig 6.2 (b) Added Patient details

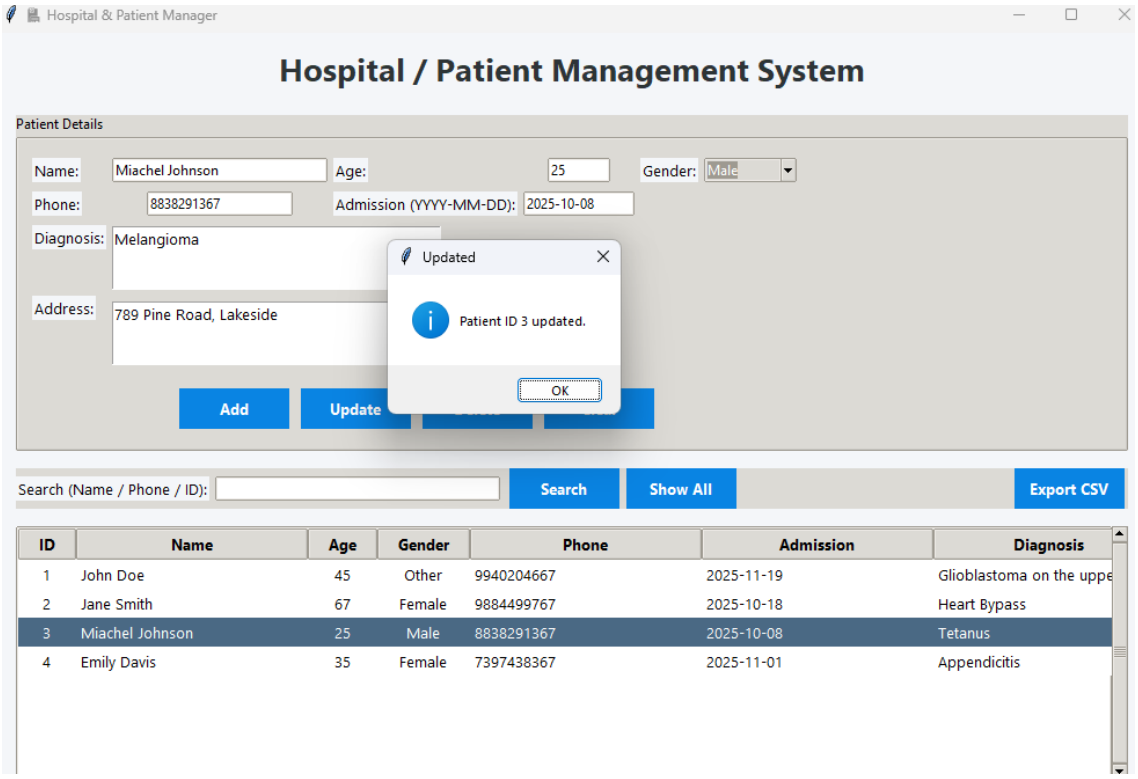

Fig 6.2(c) Updation in Patient Details

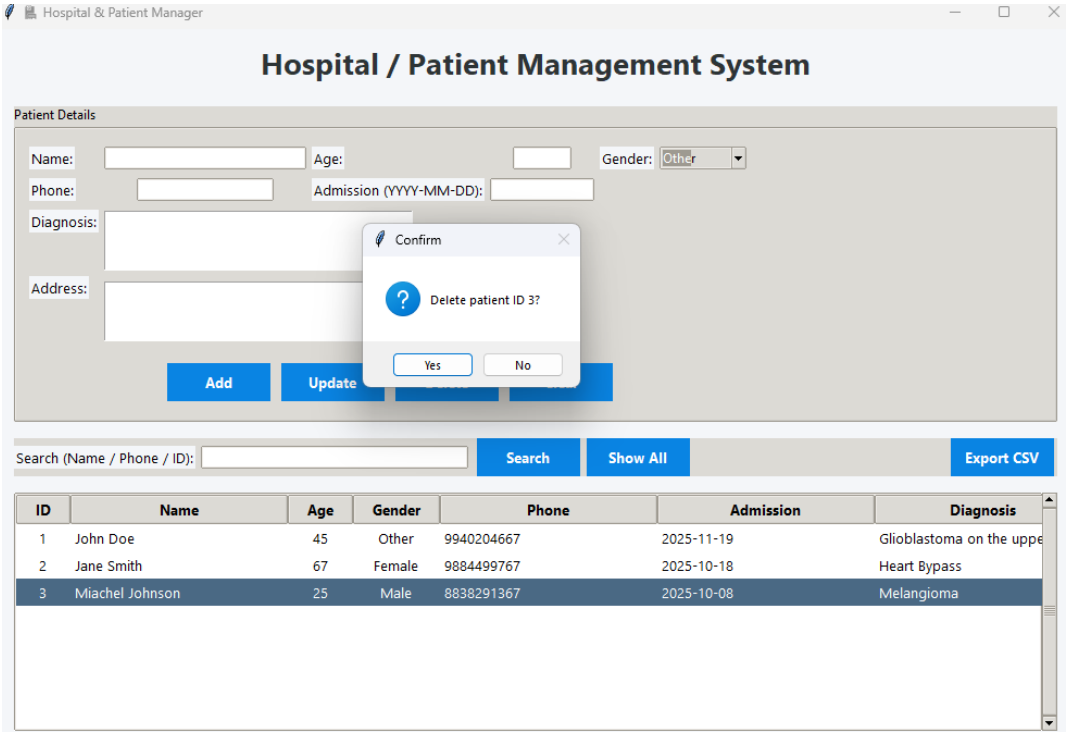

Fig 6.2 (d) Deletion in Patient Details

Exporting the result details as csv file:

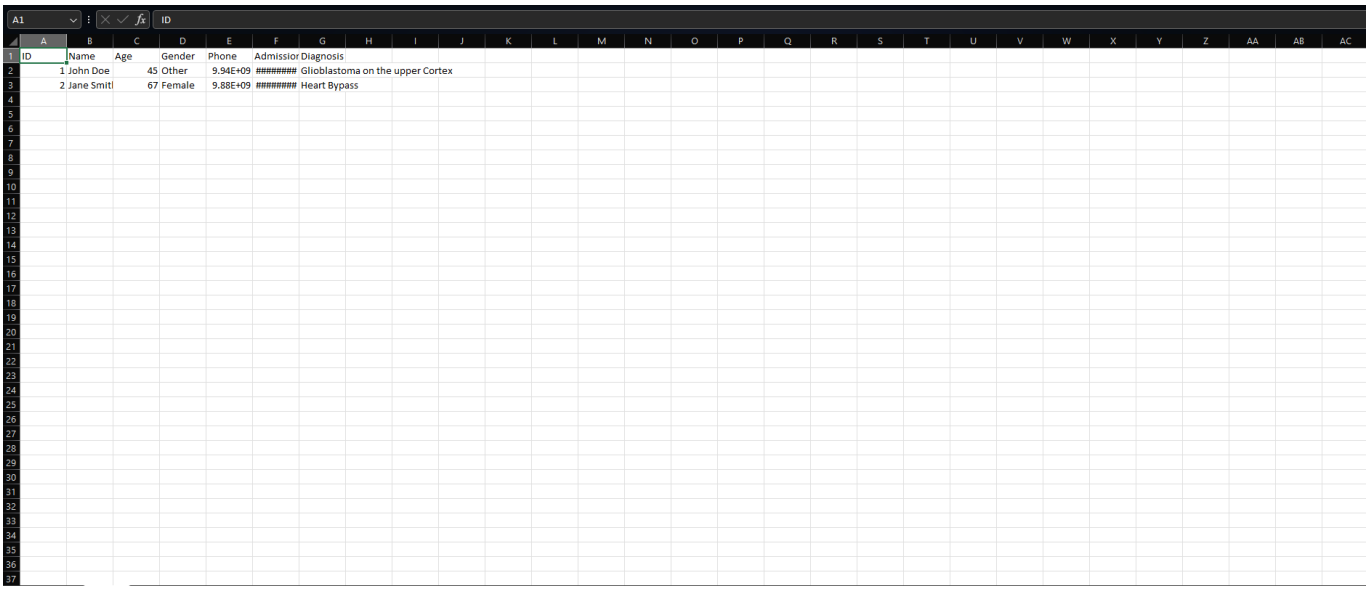

Fig 6.3 Exporting csv

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

The Hospital & Patient Management System successfully automates essential hospital operations such as patient record management, data entry, updating, searching, and report generation. By integrating a lightweight SQLite database with a clean Tkinter interface, the system ensures reliable storage, easy navigation, and quick access to patient information. The solution reduces manual errors, speeds up administrative workflows, and maintains consistency in record-keeping across all patients.

The system is designed with scalability and usability in mind, making it suitable for extension into more advanced hospital management tools. Features such as automated CSV exports, printable patient reports, appointment scheduling, and analytics dashboards can be integrated in the future. Overall, this project provides a strong foundation for a modern, efficient, and user-friendly hospital record management system.

REFERENCES

1. Python Official Documentation — https://docs.python.org/3/

2. SQLite Documentation — https://www.sqlite.org/docs.html

3. TkinterGUIGuide — https://docs.python.org/3/library/tkinter.html

4. W3SchoolsPythonTutorials—https://www.w3schools.com/python