Autonomous Institute, Affiliated to VTU

PYTHON AAT Report on

Hospital Lab Test Entry System

Submitted in partial fulfillment of the requirements for AAT

Bachelor of
Engineering in
Computer Science and Engineering

Submitted by:

M Afnaan Ali (1WN24CS148)

Likhith R (1BM24CS147)

Likith Krishnan (1WA24CS155)

Madhav V Nair (1BF24CS159)

Department of Computer Science and Engineering
B.M.S College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560
019 2024-2025

B.M.S COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **M Afnaan Ali**, **Likhith R** , **Likith Krishnan**, **Madhav V Nair** students of 2nd Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this AAT Project entitled "**Hospital Lab Test Entry System**" has been carried out in Department of CSE, BMS College of Engineering, Bangalore during the academic semester Oct 2024 – Jan 2025. We also declare that to the best of our knowledge and belief, the AAT Project report is not from part of any other report by any other students.

Student Name **Student Signature**

1. **M Afnaan Ali**
2. **Likhith R**
3. **Likith Krishnan**
4. **Madhav V Nair**

BMS COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the AAT Project titled **"Hospital Lab Test Entry System"** ***has been carried out by*** **M Afnaan Ali (1WN24CS148) ,Madhav V Nair (1BF24CS159) , Likith Krishnan (1WA24CS155) , Likhith R (1BM24CS147)*** *of CV section during the academic year 2024-2025.*

Signature of the Faculty in Charge

Table of Contents

Introduction

In today's fast-paced healthcare environment, accurate and timely data management plays a critical role in delivering effective patient care. Laboratories form the backbone of diagnostic services in hospitals, and maintaining precise records of tests, patient information, and results is essential for diagnosis, treatment planning, and follow-up care. However, traditional methods of manual data entry are often time-consuming, error-prone, and inefficient, especially when handling a large volume of patients and diverse test types.

To address these challenges, the Hospital Lab Test Entry System was developed as a simple yet powerful application designed to digitize and streamline the lab test recording process. Built using Python and the Tkinter GUI library, this system offers a clean, interactive interface where users can easily enter and manage patient information such as name, age, gender, and ID. It allows the selection of lab tests from a dropdown menu, including tests like CBC, Blood Glucose, Lipid Profile, and Urinalysis. Based on the selected test, the system dynamically generates relevant input fields, making data entry specific and accurate. Once entered, the data is stored in a structured JSON format for easy retrieval and management. Users can view previously saved patient test results in a scrollable and stylized table format, organized either by test type or patient ID. The interface ensures that all data entries are validated and neatly recorded, significantly reducing the chances of errors while enhancing workflow efficiency.

Overall, the Hospital Lab Test Entry System improves hospital record-keeping and serves as a foundational step toward more advanced digital healthcare solutions. It has the potential to be expanded in the future to include features such as automated report generation, integration with doctor feedback systems, or connections to hospital-wide management platforms.

Hardware Requirements

Component	Specification
Processor	Minimum: Intel Core i3 or equivalent
	Recommended: Intel Core i5 or higher
RAM	Minimum: 4 GB
	Recommended: 8 GB or more
Hard Disk	Minimum: 100 MB free space (for code & data)
Display	1024x768 resolution or higher
Input Devices	Keyboard and Mouse
Operating System	Windows 7/8/10/11, macOS, or Linux

Software Requirements

Component	Specification
Programming Language	Python 3.7 or above
GUI Framework	Tkinter (comes built-in with Python)
Text Editor/IDE	IDLE / VS Code / PyCharm / Sublime Text
Libraries Used	tkinter, json, os, random, datetime
Storage	JSON file (lab_test_data.json) for saving patient data

Technologies Used:

Technology / Library	Purpose
Python 3.x	Core programming language used to build the application
Tkinter (tkinter)	Python's standard GUI library used to create the graphical interface
JSON (json)	Used to save and load patient data persistently in a structured format
OS Module (os)	Checks if the data file exists before loading it
Random Module (random)	Generates a unique random Patient ID
Datetime Module (datetime)	Automatically logs sample received/tested time with timestamps

Optional Supporting Tools:

Tool / Feature	Purpose
Text Editor / IDE	Writing and running the code (e.g., VS Code, PyCharm, IDLE)
Git/GitHub	For generating professional project reports
Windows/Linux/Mac	The system the GUI app runs on

Design

1. Start Screen (Main Menu)

- Displays three main options:
    1. **New Patient Entry**
    2. **View Test Records**
    3. **Search by ID or Name**
- The window is centered, styled, and user-friendly.

2. New Patient Entry Flow

Step-by-Step:

1. User enters:
    - Patient **Name**
    - **Gender** (checkbox or input)
    - **Age** (selected from a dropdown: 8 to 95)
2. User selects a **Lab Test** from dropdown:
    - CBC, Blood Glucose, Lipid Profile, or Urinalysis
3. Based on the test, dynamic input fields appear:
    - e.g., Fasting? (Yes/No), Time of Collection, etc.
4. User enters a **Purpose** for the test
5. System:
    - Generates a **Patient ID** randomly
    - Records **Sample Received Time** and **Sample Tested Time**
    - Saves all data to lab_test_data.json
6. Displays : success message

3. View Test Records Flow

Step-by-Step:

1. User selects a **test type** from dropdown
2. System filters and displays **all saved patients** who took that test
3. Patient details are shown in a **horizontal card layout**:
    - Name, Age, Gender, Patient ID
    - Test Info
    - Purpose
    - Sample timestamps
    - Test-specific inputs

4. Search by ID or Name Flow

Step-by-Step:

1. User can either:
    - Type a name or ID
    - Select from a **dropdown** showing all patient names with IDs
2. On clicking Search:
    - The system searches through saved records
    - If a match is found → displays full test details (similar to View mode)
    - If not → shows "No match found" message

5. Data Handling

- All data is stored in lab_test_data.json
- JSON format is a list of patient records
- On launch, data is loaded and used to populate dropdowns and views

6. Exit or Return

- Users can return to the main screen at any time using "⬅ Back to Menu"
- The app runs continuously until closed by the user

Source Code

check4.py > ...

```python
import tkinter as tk
from tkinter import messagebox
import json
import os
import random
from datetime import datetime

# Lab tests and their required fields
lab_tests = {
    "CBC": ["Blood Sample Collected? (Yes/No)"],
    "Blood Glucose": ["Fasting? (Yes/No)", "Time of Last Meal"],
    "Lipid Profile": ["Fasting Hours", "Dietary Habits"],
    "Urinalysis": ["Time of Collection", "Fluid Intake (ml)"]

}

data_file = "lab_test_data.json"

# Load saved data
if os.path.exists(data_file):
    with open(data_file, "r") as f:
        try:
            saved_data = json.load(f)
            if not isinstance(saved_data, list):
                saved_data = []
        except json.JSONDecodeError:
            saved_data = []
else:
    saved_data = []

root = tk.Tk()
root.title("\U0001F3E5 Hospital Lab Test Entry System")
root.config(bg="#f0f4f7")
window_width = 600
window_height = 600
screen_width = root.winfo_screenwidth()
```

```python
screen_height = root.winfo_screenheight()
x = (screen_width // 2) - (window_width // 2)
y = (screen_height // 2) - (window_height // 2)
root.geometry(f"{window_width}x{window_height}+{x}+{y}")

# Big button template
def big_button(parent, text, command, color):
    return tk.Button(parent, text=text, font=("Arial", 12, "bold"), width=30, height=2, bg=color, command=command)

# Main Menu
def start_screen():
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text="\U0001F3E5 Welcome to the Lab Test System", font=("Arial", 18, "bold"), bg="#f0f4f7").pack(pady=30)
    tk.Label(root, text="Please choose an option:", font=("Arial", 12), bg="#f0f4f7").pack(pady=10)

    big_button(root, "➕ New Patient Entry", patient_entry_screen, "#b6e2d3").pack(pady=10)
    big_button(root, "📄 View Test Records", select_test_to_view, "#b3d9ff").pack(pady=10)

    big_button(root, "🔎 Search by ID or Name", search_by_id_or_name_screen, "#f5d27a").pack(pady=15)

# Patient Entry

def patient_entry_screen():
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text="👩‍⚕️ Enter Patient Information", font=("Arial", 16, "bold"), bg="#f0f4f7").pack(pady=15)

    form_frame = tk.Frame(root, bg="#f0f4f7")
    form_frame.pack(pady=10)

    label_font = ("Arial", 12)
    entry_font = ("Arial", 12)

```

```
61  def patient_entry_screen():
72
73      tk.Label(form_frame, text="Name:", font=label_font, bg="#f0f4f7").grid(row=0, column=0, sticky="e", pady=5, padx=10)
74      name_entry = tk.Entry(form_frame, width=30, font=entry_font)
75      name_entry.grid(row=0, column=1, pady=5)
76
77      tk.Label(form_frame, text="Age:", font=label_font, bg="#f0f4f7").grid(row=1, column=0, sticky="e", pady=5, padx=10)
78      age_var = tk.StringVar()
79      age_entry = tk.Spinbox(form_frame, from_=8, to=95, width=28, font=entry_font, textvariable=age_var)
80
81      age_entry.grid(row=1, column=1, pady=5)
82
83      tk.Label(form_frame, text="Gender:", font=label_font, bg="#f0f4f7").grid(row=2, column=0, sticky="e", pady=5, padx=10)
84      gender_var = tk.StringVar(value="Male")  # default value
85
86      tk.Radiobutton(form_frame, text="Male", variable=gender_var, value="Male", bg="#f0f4f7").grid(row=2, column=1, sticky="w")
87      tk.Radiobutton(form_frame, text="Other", variable=gender_var, value="Female", bg="#f0f4f7").grid(row=2, column=1, sticky="e")
88      tk.Radiobutton(form_frame, text="Female", variable=gender_var, value="Other", bg="#f0f4f7").grid(row=2, column=1)
89
90
91      tk.Label(form_frame, text="Purpose of Test:", font=label_font, bg="#f0f4f7").grid(row=3, column=0, sticky="e", pady=5, padx=10)
92      purpose_entry = tk.Entry(form_frame, width=30, font=entry_font)
93      purpose_entry.grid(row=3, column=1, pady=5)
94
95      tk.Label(root, text="🧪 Select Lab Test", font=("Arial", 12), bg="#f0f4f7").pack(pady=10)
96      test_var = tk.StringVar()
97      test_var.set("Choose a test")
98      test_menu = tk.OptionMenu(root, test_var, *lab_tests.keys(), command=lambda _: show_test_inputs())
99      test_menu.config(font=("Arial", 11), width=25)
100     test_menu.pack()
101
102
103     dynamic_frame = tk.Frame(root, bg="#f0f4f7")
104     dynamic_frame.pack(pady=10)
105
106     back_btn = big_button(root, "⬅ Back to Menu", start_screen, "#e0e0e0")
```

```
check4.py > ...
61  def patient_entry_screen():
106     back_btn = big_button(root, "⬅ Back to Menu", start_screen, "#e0e0e0")
107     back_btn.pack(pady=20)
108
109     def show_test_inputs():
110         for widget in dynamic_frame.winfo_children():
111             widget.destroy()
112
113         test = test_var.get()
114         if test not in lab_tests:
115             return
116
117         tk.Label(dynamic_frame, text="📝 Test Requirements:", font=("Arial", 12, "bold"), bg="#f0f4f7").pack()
118
119         input_entries = []
120         for field in lab_tests[test]:
121             tk.Label(dynamic_frame, text=field, bg="#f0f4f7", font=label_font).pack()
122             entry = tk.Entry(dynamic_frame, width=40, font=entry_font)
123             entry.pack(pady=3)
124             input_entries.append(entry)
125
126         def save_patient_data():
127             name = name_entry.get().strip()
128             age = age_var.get().strip()
129             gender = gender_var.get().strip()
130             purpose = purpose_entry.get().strip()
131
132             if not name or not age or not gender or not purpose:
133                 messagebox.showerror("Error", "Please fill in all patient and purpose details.")
134                 return
135
136             input_vals = []
137             for entry in input_entries:
138                 val = entry.get().strip()
139                 if not val:
```

check4.py > ...

```python
def select_test_to_view():
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text="📋 View Test Records", font=("Arial", 16, "bold"), bg="#f0f4f7").pack(pady=20)
    test_var = tk.StringVar()
    test_var.set("Choose a test")
    tk.OptionMenu(root, test_var, *lab_tests.keys()).pack(pady=10)

    big_button(root, "🔍 View Records", lambda: view_saved_data(test_var.get()), "#cde0f0").pack(pady=5)
    big_button(root, "⬅ Back to Menu", start_screen, "#e0e0e0").pack(pady=5)

# View Saved Data

def view_saved_data(selected_test):
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text=f"📁 Records for {selected_test}", font=("Arial", 14, "bold"), bg="#f0f4f7").pack(pady=10)

    entries = [entry for entry in saved_data if entry["Test"] == selected_test]
    if not entries:
        tk.Label(root, text="❌ No records found.", fg="red", bg="#f0f4f7").pack(pady=20)
    else:
        frame = tk.Frame(root, bg="#ffffff", relief="sunken", borderwidth=1)
        frame.pack(pady=10, fill="both", expand=True)

        canvas = tk.Canvas(frame, bg="#ffffff")
        scrollbar = tk.Scrollbar(frame, orient="vertical", command=canvas.yview)
        scroll_frame = tk.Frame(canvas, bg="#ffffff")

        scroll_frame.bind("<Configure>", lambda e: canvas.configure(scrollregion=canvas.bbox("all")))
        canvas.create_window((0, 0), window=scroll_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        for record in entries:
```

check4.py > ...

```python
def view_saved_data(selected_test):
        for record in entries:
            user = record["User"]
            inputs = record["Inputs"]
            patient_id = f"PID{record['PatientID']}"
            entry_created = record.get("SampleReceivedTime", "N/A")
            sample_tested = record.get("SampleTestedTime", "Not tested")

            card = tk.Frame(scroll_frame, bg="#ffffff", bd=1, relief="solid", padx=10, pady=8)
            card.pack(fill="x", pady=6, padx=6)

            tk.Label(card, text=f"Patient ID: {patient_id}", font=("Arial", 10, "bold"), bg="#ffffff").pack(anchor="w")
            tk.Label(card, text=f"Name: {user['Name']}", bg="#ffffff").pack(anchor="w")
            tk.Label(card, text=f"Age: {user['Age']}", bg="#ffffff").pack(anchor="w")
            tk.Label(card, text=f"Gender: {user['Gender']}", bg="#ffffff").pack(anchor="w")
            tk.Label(card, text="Test Inputs:", font=("Arial", 10, "bold"), bg="#ffffff").pack(anchor="w")
            for key, val in inputs.items():
                tk.Label(card, text=f"• {key}: {val}", bg="#ffffff").pack(anchor="w")

            tk.Label(card, text="Sample Status:", font=("Arial", 10, "bold"), bg="#ffffff").pack(anchor="w")
            tk.Label(card, text=f"Received: Yes", bg="#ffffff").pack(anchor="w")
            tk.Label(card, text=f"Tested: {'No' if sample_tested == 'Not tested' else 'Yes'}", bg="#ffffff").pack(anchor="w")

            tk.Label(card, text="Timestamps:", font=("Arial", 10, "bold"), bg="#ffffff").pack(anchor="w")
            tk.Label(card, text=f"• Entry Created: {entry_created}", bg="#ffffff").pack(anchor="w")
            tk.Label(card, text=f"• Sample Received: {entry_created}", bg="#ffffff").pack(anchor="w")
            tk.Label(card, text=f"• Sample Tested: {sample_tested}", bg="#ffffff").pack(anchor="w")

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

    big_button(root, "⬅ Back to Menu", start_screen, "#e0e0e0").pack(pady=10)
    scrollbar.pack(side="right", fill="y")

# Search by ID or Name
```

check4.py > ...

```python
def search_by_id_or_name_screen():
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text="🔎 Search Patient Records", font=("Arial", 16, "bold"), bg="#f0f4f7").pack(pady=10)

    tk.Label(root, text="Type Name/ID or Select Patient Below:", font=("Arial", 12), bg="#f0f4f7").pack(pady=5)

    search_entry = tk.Entry(root, font=("Arial", 12), width=40)
    search_entry.pack(pady=5)

    tk.Label(root, text="Available Patients:", font=("Arial", 12, "bold"), bg="#f0f4f7").pack(pady=5)

    options = [f"{record['User']['Name']} (ID: {record['PatientID']})" for record in saved_data]
    search_var = tk.StringVar()
    if options:
        search_var.set(options[0])
    else:
        search_var.set("No Patients")

    option_menu = tk.OptionMenu(root, search_var, *options)
    option_menu.config(font=("Arial", 12), width=15, height=2)
    option_menu.pack(pady=10)

    result_frame = tk.Frame(root, bg="#ffffff", relief="sunken", borderwidth=1)
    result_frame.pack(pady=10, fill="both", expand=True)

    def display_result(record):
        for widget in result_frame.winfo_children():
            widget.destroy()

        user = record['User']
        inputs = record['Inputs']
        tk.Label(result_frame, text=f"🆔 Patient ID: {record['PatientID']}", bg="#ffffff", font=("Arial", 10, "bold")).pack(anchor="w",
        tk.Label(result_frame, text=f"👤 {user['Name']} | Age: {user['Age']} | Gender: {user['Gender']}", bg="#ffffff").pack(anchor="w")
        tk.Label(result_frame, text=f"💉 Test: {record['Test']}", bg="#ffffff").pack(anchor="w")
```

check4.py > ...

```python
def search_by_id_or_name_screen():
    def display_result(record):
        tk.Label(result_frame, text=f"👤 {user['Name']} | Age: {user['Age']} | Gender: {user['Gender']}", bg="#ffffff").pack(anchor="w")
        tk.Label(result_frame, text=f"💉 Test: {record['Test']}", bg="#ffffff").pack(anchor="w")
        tk.Label(result_frame, text=f"📌 Purpose: {record['Purpose']}", bg="#ffffff").pack(anchor="w")
        tk.Label(result_frame, text=f"📬 Sample Received: {record['SampleReceivedTime']}", bg="#ffffff").pack(anchor="w")
        tk.Label(result_frame, text=f"🔬 Sample Tested: {record['SampleTestedTime']}", bg="#ffffff").pack(anchor="w")
        for key, val in inputs.items():
            tk.Label(result_frame, text=f"   - {key}: {val}", bg="#ffffff").pack(anchor="w")
        tk.Label(result_frame, text="-"*60, bg="#ffffff").pack(pady=3)

    def perform_search():
        query = search_entry.get().strip()
        if not query:
            query = search_var.get()

        matches = []
        for record in saved_data:
            name_match = query.lower() in record['User']['Name'].lower()
            id_match = query in str(record['PatientID'])
            full_match = f"{record['User']['Name']} (ID: {record['PatientID']})" == query
            if name_match or id_match or full_match:
                matches.append(record)

        if matches:
            display_result(matches[0])
        else:
            messagebox.showinfo("No Match", "No matching patient found.")

    big_button(root, "🔍 Search", perform_search, "#e6f2ff").pack(pady=5)
    big_button(root, "⬅ Back to Menu", start_screen, "#e0e0e0").pack(pady=10)

# Start the app
start_screen()
root.mainloop()
```

Customization & Extensibility

How the system can be enhanced or modified to support additional features and workflows in the future.

1. Add/Edit/Delete Patient Records

- *Current*: Patients can be added.
- *Customizable*:
    - Add an **Edit Patient** option to update existing records.
    - Add a **Delete** option with confirmation to remove records safely.
    - Add **Admin privileges** for data control.

2. Advanced Search Features

- Add:
    - Filter by **date range**
    - Search by **age group**, **gender**, or **test purpose**
    - View **all records** with pagination or tabular view

3. Add More Lab Tests

- Lab test dropdown is dynamic.
- To support more tests:
    - Just extend the lab_tests dictionary:

Code

```
lab_tests["Thyroid Panel"] = ["TSH", "T3", "T4"]
```

    - The UI will automatically create the needed fields.

4. Switch to Database

- Currently using a JSON file (lab_test_data.json)
- Replace with:
  - **SQLite** for local structured storage
  - **Firebase/Firestore** for cloud-based real-time data
  - **PostgreSQL/MySQL** for multi-user hospital-wide usage

5. Export Reports

- Add options to:
  - Export patient record as **PDF**
  - Generate **CSV summaries** of tests
  - Email results directly from the app

6. Theme and Accessibility

- Add:
  - Light/Dark mode toggle
  - Font size adjustments
  - Language switcher (e.g., English/Hindi/Tamil)

7. Cross-Platform Packaging

- Convert to standalone app:
  - Use **PyInstaller** to package into .exe for Windows
  - Use **Kivy** or **PyQt** for better mobile/tablet support

8. Test Result Integration

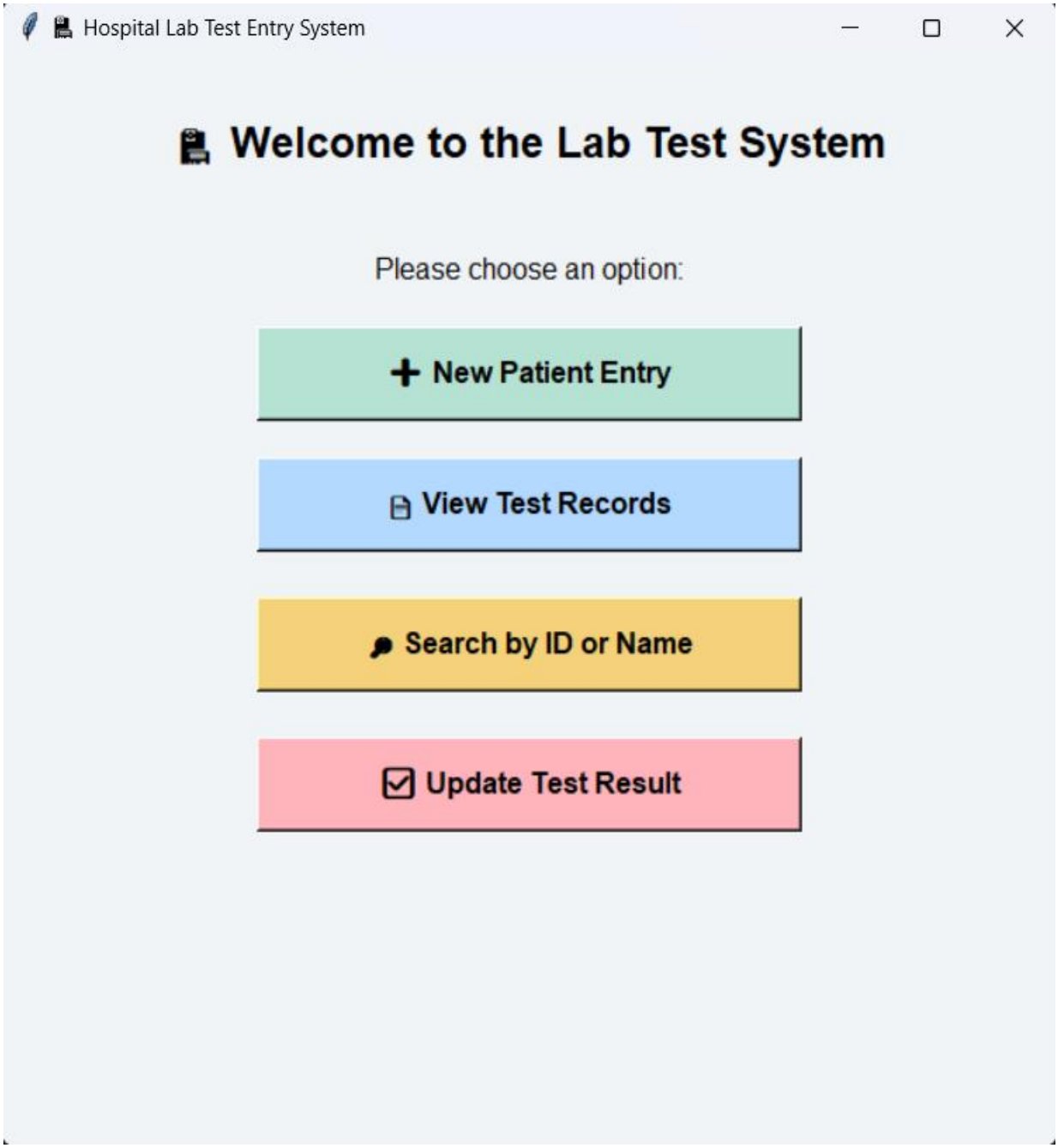
- Add:
    - Fields to input lab result values
    - Auto-flagging of abnormal ranges
    - Graphical charting using matplotlib or recharts (if using web tech)

9. AI Integration (Future Scope)

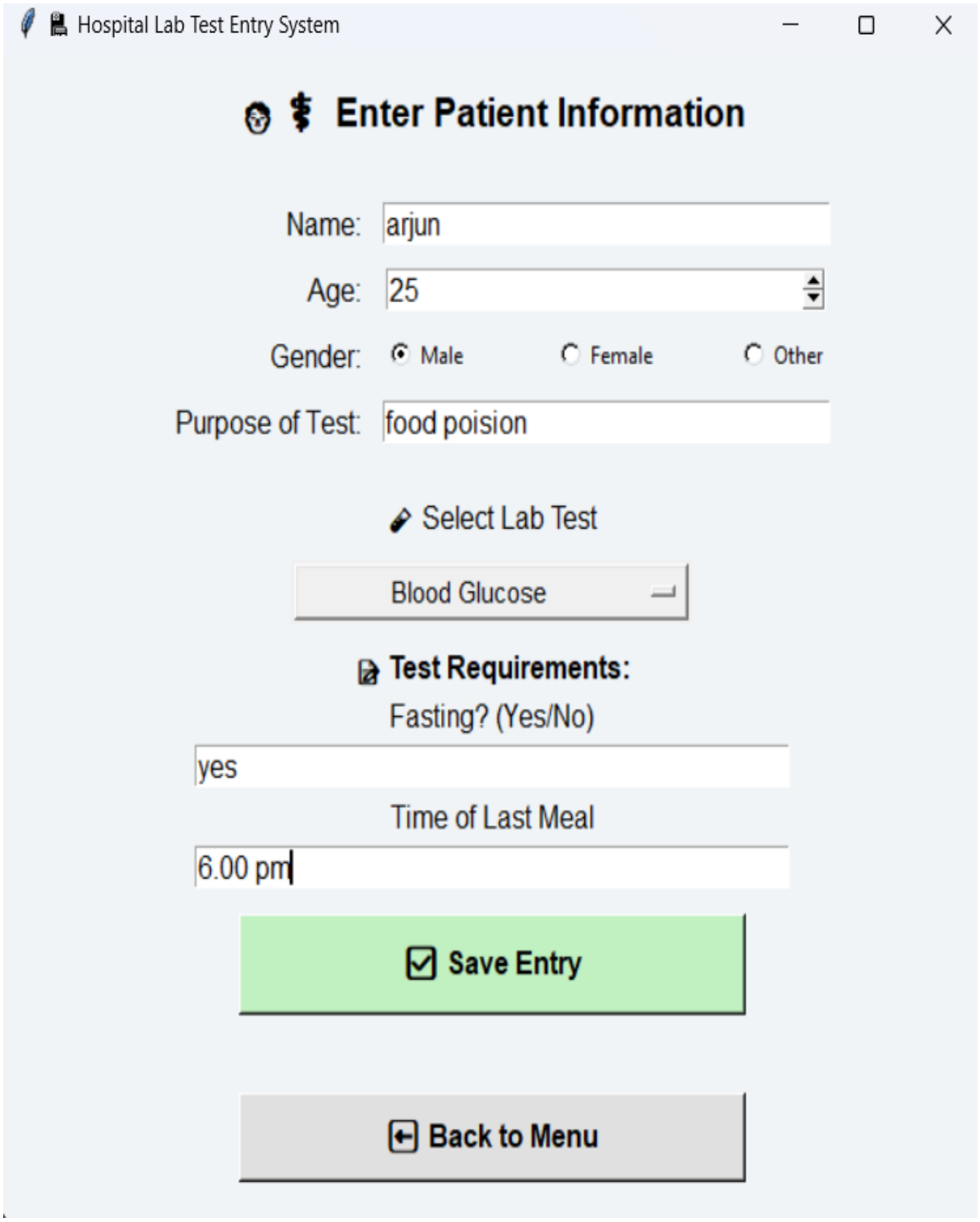
- Predictive features using machine learning:
    - Risk prediction based on test inputs
    - Smart test recommendations based on patient history

Results

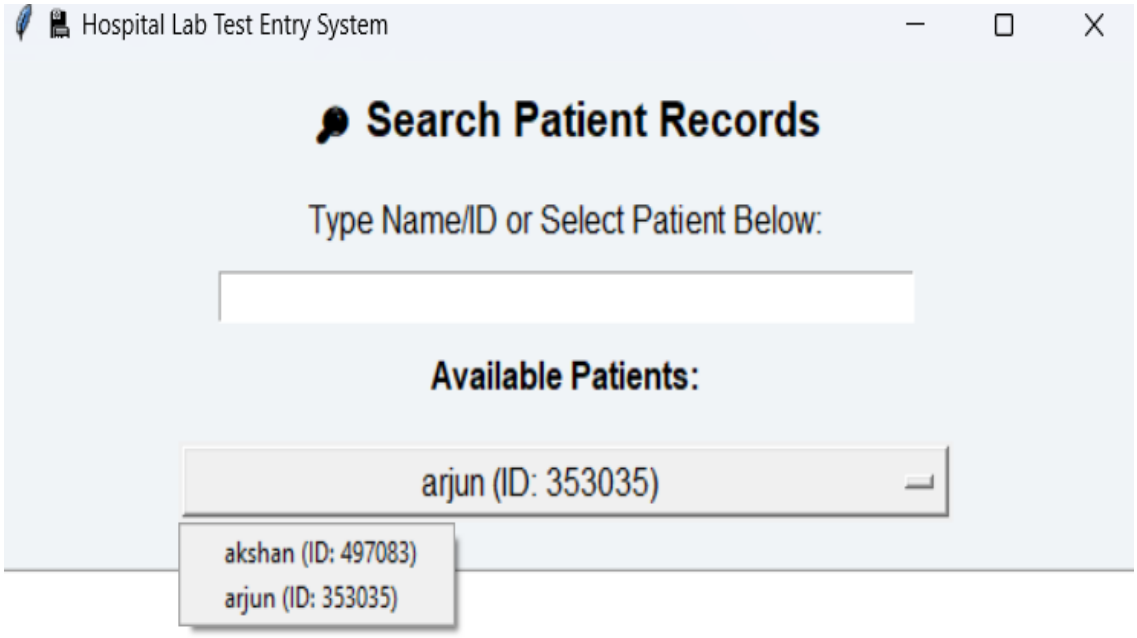
1. Main Menu Screen

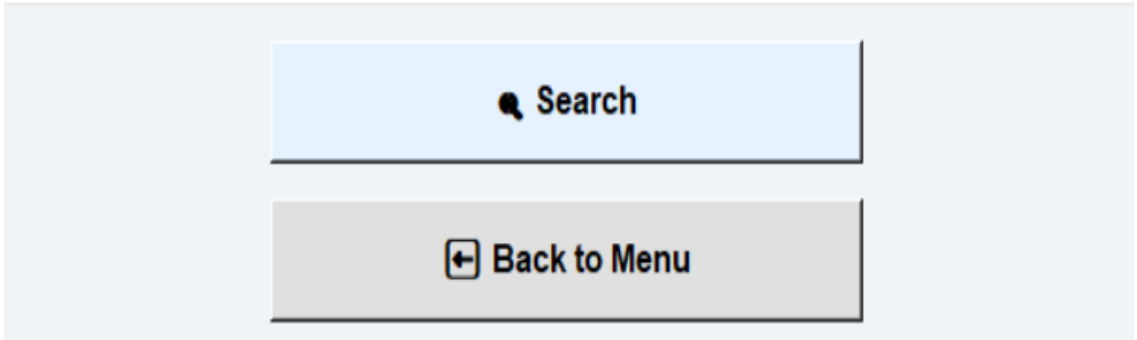


2.New Patient Entry

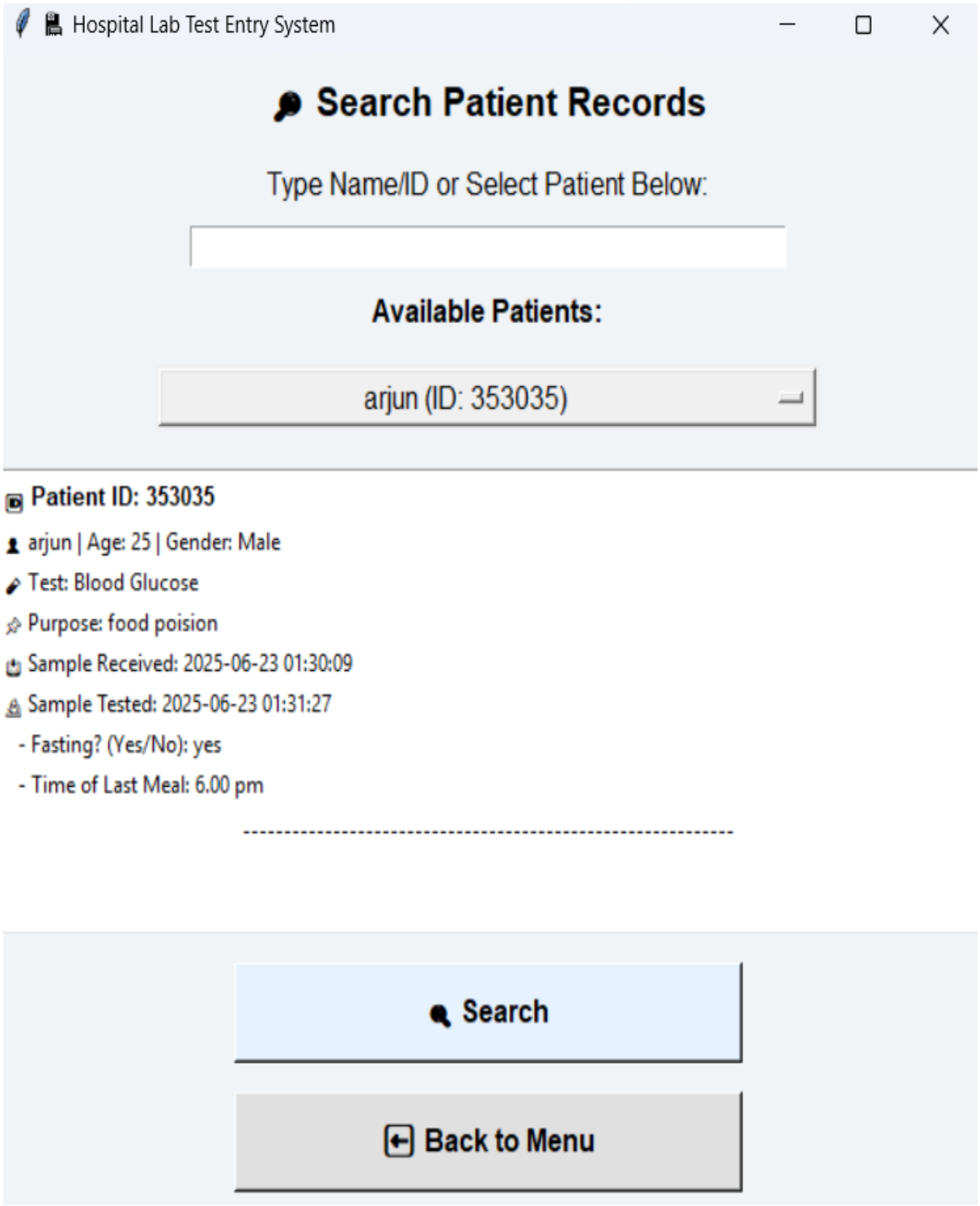


3.View Test Records – Dropdown





4.Patient Search Result



References

1. www.python.org
2. Tkinter Python Tutorial by Edureka on youtube
3. Tkinter module Document on Pip website
4. www.geeksforgeeks.org
5. JSON module Documentaion
6. www.google.com