**SRM UNIVERSITY, DELHI-NCR, SONEPAT, HARYANA**

**Plot No. 39, Rajiv Gandhi Education City, Sonepat, Haryana 131029**

**MAY 2025**

***LIVE PROJECT REPORT***

*Submitted in Partial fulfilment of the requirements for the award of the degree of*

# Bachelor of Technology in Computer Science and Engineering


**Supervisor:**

Dr. Priyanka Maan

Assistant Professor

**Submitted by:**

Javed Rain (10322210010)

Jai Kishan (10322210023)

Tanshul Deshwal (10322210047)



**SRM UNIVERSITY, DELHI-NCR, SONEPAT, HARYANA**

**Plot No. 39, Rajiv Gandhi Education City, Sonepat, Haryana**

**131029 MAY 2025**

# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

# PROJECT APPROVAL PAGE

**B. Tech CSE [Core]**

GROUP No: 8 SESSION: 2024 - 2025

**To be filled in by the student: -**

Student Name(s):Javed Rain, Jai Kishan, Tanshu Deshwal

Enrolment Number:103222100 - 10, 23, 47

Project Title: Blood Donation Application

Signature:

**To be filled in by the supervisor: -**

This is to certify that the project submitted and presented by the above-mentioned student(s) is

COMPLETE ACCEPTED

and the draft of the graduation project report has been corrected for all content flaws, typing errors, and language mistakes.

Supervisor name and Signature: Date:

Examiner's Name and Signature: Date:

HOD's Signature: Date:

**For Departmental Office use only**

Date received:

Signatures:

# CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the project entitled "**Blood Donation App**" in partial fulfilment of the requirement for the award of the degree of B.Tech – CSE(Core) and submitted in the Department of Computer Science & Engineering of SRM University, Delhi-NCR, Sonipat, Haryana, (India) is an authentic record of our own work carried out under the supervision of **Dr. Priyanka Maan** as Report File in Sixth semester during the academic year 2024-2025. The matter presented in this project has not been submitted for the award of any other degree of this or any other Institute / University.


Javed Rain

Registration No.:10322210010

Jai Kishan

Registration No.:10322210023

Tanshul Deshwal

Registration No.:10322210047

# CERTIFICATE

This is to certify that the "**Blood Donation App**" project is a Bonafide work completed as a Live Project (Course Code:21CS0304) in partial fulfilment for the award of the degree of Bachelor of Technology in CSE(Core) under the guidance of **Dr. Priyanka Maan**. This project is submitted by Javed Rain(10322210010), Jai Kishan(10322210023), Tanshul Deshwal(10322210047) during the academic Semester VI of year 2024-2025 of SRM University, Delhi-NCR, Sonipat, Haryana.

**Dr. Priyanka Maan**

**Assistant Professor**

# ACKNOWLEDGEMENT

Most importantly, we would like to express our immense gratitude to our supervisor **Dr. Priyanka Maan**, for her patient guidance, encouragement, training, and advice she has provided throughout the time. The knowledge we have gained throughout this time will stay with us for years to come. We have been extremely lucky to have such a supervisor who cared for our work, and responded to our questions and queries so promptly. It was a great privilege to get to collaborate with her and work under her guidance. Though the following project and report is an individual work, we would never be able to explore the depths of the topic, without the help, support, guidance & efforts of our able supervisor. Her infectious enthusiasm and unlimited zeal have been major driving forces throughout the work.

We would like to extend our indebtedness to the **Project Committee Members**, Department of Computer Science and Engineering/ Department of Computer Science, SRM University Delhi-NCR, Sonepat as we have benefited so much from the constructive criticism and suggestions given to us during the project.

We feel compelled to articulate our thankfulness to **Prof. M. Mohan**, HOD, Department of Computer Science and Engineering, SRM University for his encouragement which was a source of inspiration.

Lastly, we are indebted to all the teaching and non-teaching staff members of the university for helping us directly or indirectly by all means throughout the course of our study and project work.

# Table of Contents

# Chapter 1

# INTRODUCTION

## 1.1 Overview

The Blood Donation Application is a desktop-based software system designed to automate and streamline the management of blood donations and blood bank inventories. It addresses inefficiencies found in manual systems and supports critical healthcare operations by ensuring reliable access to blood units.

## 1.2 Objective

- The primary goals of the project include:
- Developing a user-friendly GUI using Python Tkinter.
- Implementing MySQL-based secure data storage.
- Enabling functionalities like donor registration, inventory tracking, and blood request management.
- Providing basic reporting and administrative capabilities.

## 1.3 Motivation

The motivation behind this project lies in resolving the drawbacks of traditional blood bank management systems, such as manual data entry errors, delayed access to critical donor or blood stock information, and difficulty in responding to emergencies.

## 1.4 Problem Statement

Existing manual or outdated digital systems are prone to data inaccuracies, inefficiencies in managing inventory, and weak emergency response capabilities. This project proposes an integrated desktop solution that enables accurate, timely, and secure blood management operations.

## 1.5 Technologies Used

- **Frontend**: Python 3.8+ with Tkinter
- **Backend**: MySQL (via mysql-connector-python)
- **IDE**: PyCharm, VS Code
- **Tools**: MySQL Workbench, manual testing methods
- **Language**: Python
- **Libraries**: tkinter, mysql.connector

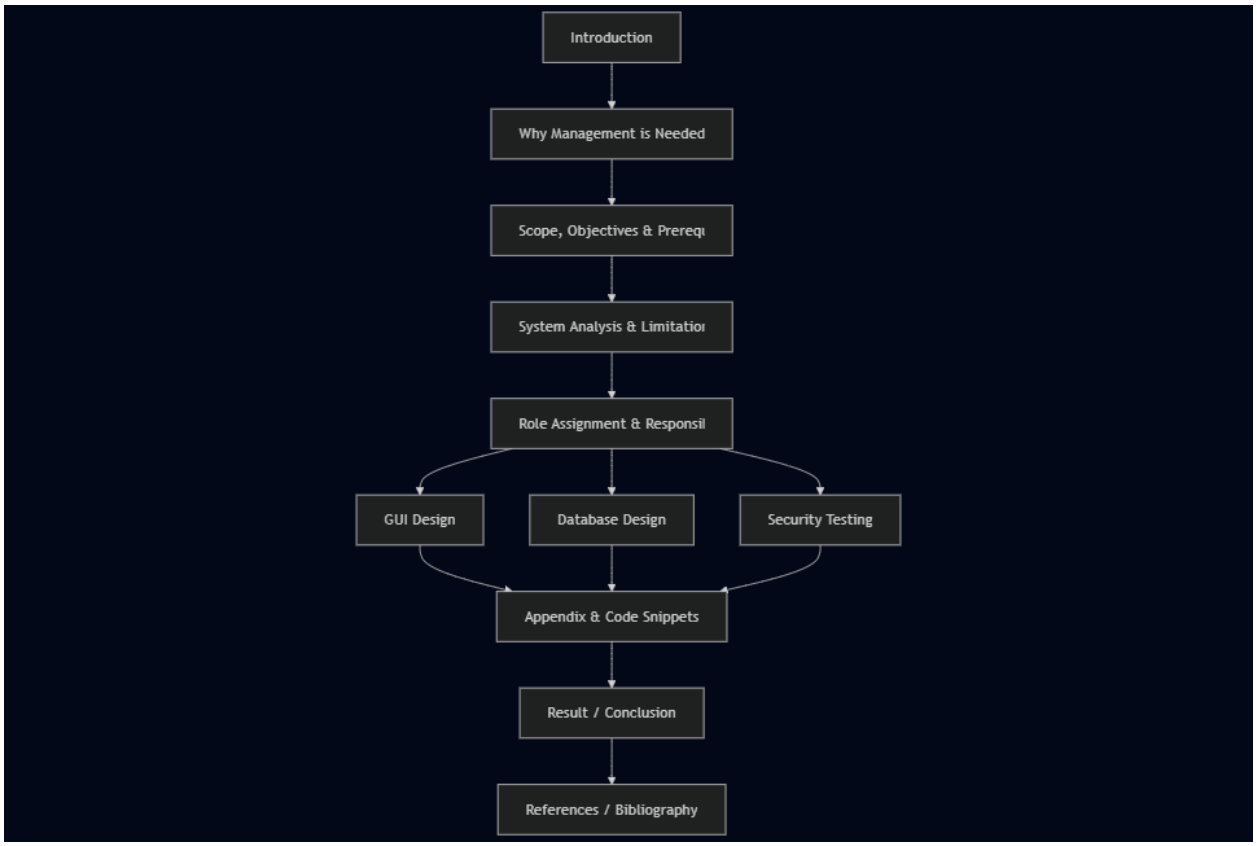
## 1.6 **Methodology**

- Requirements Analysis
- System Design (GUI and database schema)
- Implementation (Coding GUI, backend, and CRUD functions)
- Manual Testing and Security Validation
- Documentation and Reporting

## 1.7 **Project Highlights**

- Admin Login System
- Donor registration and search
- Blood inventory management with expiry tracking
- Request and donation tracking
- Basic security implementation (password validation, input sanitization)
- Manual CRUD operations with MySQL
- Tkinter GUI with validation and message dialogs

## 1.8 **Report Roadmap**




Chapter:2

# 2. PROJECT DESCRIPTION

## 2.1 **Problem Statement**

Current systems for managing blood donation processes—whether paper-based or relying on basic digital tools—suffer from inefficiencies such as inaccurate data, difficulties in locating eligible donors, and poor inventory tracking. These limitations hinder timely response to emergencies and can compromise both donor and recipient safety.

## 2.2 **Vision and Solution Overview**

The Blood Donation App is envisioned as a secure, efficient, and user-friendly desktop application that centralizes donor records, blood inventory management, and request tracking. It simplifies administrative tasks, reduces manual errors, and ensures quick access to critical data—all aimed at improving healthcare delivery and saving lives.

## 2.3 Key features

- Admin login for secure access.
- Donor registration, update, search, and deletion.
- Inventory management (add/view/update/delete blood units).
- Donation tracking (linking donors to donations).
- Request management and fulfillment tracking.
- Simple reporting features (donor list, blood stock levels).
- GUI built using Tkinter for intuitive usability.
- Data persistence and integrity via MySQL backend.
-

## 2.4 Functional Modules Overview

- Authentication Module: Validates admin credentials.
- Donor Management Module: Handles CRUD operations on donor records.
- Blood Inventory Module: Manages inventory additions, updates, and stock viewing.
- Donation Logging Module: Links donations to donors and tracks history.
- Request Handling Module: Logs and tracks blood unit requests.
- Reporting Module: Generates basic reports for operational use.

## 2.5 Architecture and Workflow

The architecture follows a layered approach:

- Presentation Layer: GUI developed in Tkinter
- Business Logic Layer: Python code handling workflows and validations
- Data Layer: MySQL database storing structured data
- Workflow

Example:

1)Admin logs in

2)Registers new donor → stored in donor table

3)Adds blood unit → reflected in inventory table

4)Logs donation → updates donor history and stock

5)Handles blood request → checks stock and marks fulfilment

## 2.6 Data Models Overview:

- Key database tables include:
- users (admin credentials)
- donors (donor profile, blood group, last donation date)
- blood_stock (blood group, quantity, expiry date)
- donations (donor ID, donation date, quantity)
- requests (blood group, quantity, status)
- Data integrity is maintained through use of primary keys, foreign keys, and input validation mechanisms.

## 2.7 Security Considerations:

- Secure Admin Login: Only authenticated users can access functionality.
- Input Validation: Forms sanitize and validate user input to avoid injection or logical errors.
- Parameterized Queries: Used in MySQL connector to prevent SQL injection.
- Password Storage: Recommendation to use hashing (implementation under progress).
- Access Control: All functionality gated behind login session.
- Manual Security Testing: Conducted for error handling, data visibility, and validation cases.

Chapter 3

# 3. TECHNOLOGY STACK

## 3.1 Frontend Technology:

- Language: Python 3.8+
- GUI Framework: Tkinter

### Description:

Tkinter is Python's standard GUI library, used to create user interfaces including forms, menus, buttons, labels, tables (Listbox/Treeview), and message boxes. It offers simple integration, cross-platform compatibility, and rapid development for desktop environments.

- Features Used:
- Entry widgets for data input
- Buttons and event handling for operations
- Listboxes and treeviews for displaying donor and inventory records
- Pop-ups for feedback and validation

## 3.2 Backend Technologies:

- Language: Python
- Database: MySQL (Community Edition)
- DB Connectivity: mysql-connector-python

### Description:

MySQL serves as the core relational database, storing structured data for donors, inventory, users, and transactions. Python scripts perform CRUD operations using parameterized queries for security and integrity.

- Tools & Utilities:
- MySQL Workbench: Database design and visualization.
- Python IDEs: VS Code, PyCharm.
- SQL: For queries, joins, and indexing.

Chapter 4

# 4. CORE FEATURES

## 4.1 Authentication & Authorization

Admin Login:
The system includes a secure login mechanism restricted to authorized administrators. Users must enter valid credentials (username and password) to access the system.

Access Control:
All critical features (donor management, inventory, requests) are gated behind the login, ensuring that unauthorized users cannot access or manipulate data.

Validation:
Input validation is enforced at the login stage. Passwords are currently stored in plaintext but the report notes future implementation of hashing for enhanced security.

## 4.2 Admin Panel

Dashboard Access:
After successful login, the admin is directed to a central dashboard where all modules (Donor, Blood Inventory, Requests, Reports) are accessible via GUI.

Core Functionalities:

- Add, update, delete, and search for donors
- Manage blood inventory (add/view/update/delete units)
- Log new donations and update donor histories
- Process and track blood requests
- View system reports such as donor lists and stock levels

## 4.3 Profile Management:

Donor Profiles:
Each donor record includes full name, blood group, contact details, address, birthdate, and donation history. Admins can update these records as needed.

Search & Filter:
The system allows profile searches by name, blood group, or location (if address details are complete), aiding quick access during emergencies.

## 4.4 Data Integrity:

The backend ensures that donor records are unique and current, reducing data duplication and enabling reliable filtering.

Chapter 5

# 5. SYSTEM DESIGN AND ARCHITECTURE

## 5.1 High-Level Architecture Diagram:

- Below is a description of the architecture. Let me know if you'd like this turned into a visual diagram.
- GUI Layer (Tkinter).
- Handles user interactions: login, donor entry, stock view, reports.
- Application Logic (Python Scripts).
- Manages event handling, form validations, and CRUD operations.
- Database Layer (MySQL).

Stores donor data, blood stock, donations, and requests.

Example (can be illustrated with a flowchart):
User → GUI (Tkinter) → App Logic (Python) → MySQL Database.

## 5.2 Component Breakdown

| Component | Description |
|---|---|
| Login System | Handles admin authentication and access control. |
| Donor Management | Supports add/update/delete/search operations for donors. |
| Inventory Manager | Manages blood unit tracking (entry, status, expiry). |
| Donation Tracker | Logs donations and updates donor history. |
| Request Logger | Captures and fulfills blood unit requests. |
| Report Generator | Provides donor lists, inventory summaries, etc. |

## 5.3 Data Flow Overview:

Admin logs in → credentials validated → access granted.

Admin adds donor → data pushed to donor table in DB.

Admin adds blood unit → stored in inventory with expiry.

A donation is logged → donor's last donation date updated.

Request is made → available stock verified and marked.

## 5.4 Role Management and Middleware:

Roles:

- Admin: Full control over all modules.

Middleware/Logic Layer:

- Handled via event-driven logic in Python functions.
- Validates all inputs before database interaction.
- Confirms CRUD operations via message boxes.

## 5.5 Security Considerations:

- Authentication: Admin login required to access features.
- SQL Security: Parameterized queries used to prevent SQL injection.
- Input Validation: Ensures safe and sanitized form data.
- Error Handling: Displays user-friendly messages without revealing internal stack traces.
- Data Storage: Passwords are currently plaintext; hashing recommended in future.

## 5.6 Scalability Aspects:

- MySQL Backend: Can handle moderate data volumes efficiently.
- Modular Codebase: Python scripts can be expanded with minimal coupling.

## 5.7 Limitations:

- Currently a single-user, desktop-based app.

- No cloud or multi-device access.

# Chapter 6

## 6. Project Scope , Objectives and Pre-Requisties:

The "Blood Donation Application" is designed as a desktop software solution to facilitate the management of blood donor information and blood bank inventory. The scope of this project encompasses the following core functionalities.

## 6.1 Scope:

### 6.1.1 User Authentication:

- Admin login functionality to access and manage the system.

### 6.1.2 Donor Management:

- Registration of new blood donors with details such as name, contact information, address, blood group, date of birth, and last donation date.
- Viewing a list of registered donors.
- Searching for donors based on criteria like blood group, name, or location (if address fields are detailed enough).
- Updating existing donor information.
- Deleting donor records (with appropriate confirmations).

### 6.1.3 Blood Inventory Management:

- Adding new blood units to the inventory, specifying blood group, collection date, and expiry date.
- Viewing the current blood stock, categorized by blood group.
- Updating the status or quantity of blood units (e.g., marking units as dispatched or expired).
- Searching for available blood units by blood group.

### 6.1.4 Donation Tracking (Basic):

- Logging new blood donations and linking them to registered donors.
- Updating the donor's last donation date upon a new donation.

### 6.1.5 Request Management (Basic):

- Logging requests for blood units, specifying blood group and quantity needed.
- Viewing pending blood requests.
- Marking requests as fulfilled or closed.

### 6.1.6 Graphical User Interface (GUI):

- Development of an intuitive and user-friendly interface using Python's Tkinter library for all the above functionalities.

### 6.1.7 Database Management:

- Design and implementation of a relational database using MySQL to store all application data securely and efficiently.
- Ensuring data integrity and consistency.

### 6.1.8 Reporting (Basic):

- Generating simple reports like a list of donors by blood group or current stock levels.

## 6.2. **Objectives:**

The primary objectives for the development of this Blood Donation Application are:

**6.2.1 To develop a functional and efficient desktop application** for managing blood donor information and blood bank inventory.

**6.2.2 To design and implement an intuitive Graphical User Interface (GUI)** using Python's Tkinter library that is easy for blood bank staff or administrators to learn and use.

**6.2.3 To create a robust and normalized relational database** using MySQL to securely store, manage, and retrieve data related to donors, blood units, donations, and requests with high integrity.

**6.2.4 To implement seamless and reliable connectivity** between the Python application (front-end and logic) and the MySQL database (back-end).

**6.2.5 To provide core functionalities** including secure user login, donor registration and management, blood stock tracking, basic donation recording, and simple blood request logging.

**6.2.6 To enable efficient searching and retrieval** of donor information and blood availability based on various criteria like blood group.

**6.2.7 To generate basic reports** that can aid in the day-to-day operations of a blood bank or donation center.

**6.2.8 To ensure data accuracy and consistency** through appropriate validation and database design.

**6.2.9 To produce a well-documented project** including system design, database schema, and user guidance for future maintenance and potential enhancements.

## 6.3 Prerequisites:

To successfully undertake and complete the development of the Blood Donation Application, the following prerequisites are identified:

### 6.3.1 Software Prerequisites:

- **Python:** A recent stable version of Python (e.g., Python 3.8 or higher) installed on the development machine.
- **Tkinter Library:** This is usually included with standard Python installations. Ensure it's available and functional.
- **MySQL Server:** A running instance of MySQL Community Server (or a similar compatible version) to host the application's database.
- **MySQL Client/Workbench:** A tool like MySQL Workbench, phpMyAdmin (if using a web server stack), or a command-line client to design, create, and manage the MySQL database and tables.
- **Python-MySQL Connector:** A Python library to enable communication between Python scripts and the MySQL database. The mysql.connector-python library is a common choice and needs to be installed (e.g., via pip install mysql-connector-python).
- **Integrated Development Environment (IDE) or Text Editor:** A suitable IDE like PyCharm, VS Code, Spyder, or a capable text editor for writing and managing Python code.

### 6.3.2 Hardware Prerequisites:

- A personal computer or laptop with sufficient processing power, memory (RAM), and storage to run the Python interpreter, IDE, MySQL server, and the application itself smoothly.
- Standard input/output devices (keyboard, mouse, display).

### 6.3.3 Knowledge Prerequisites:

- **Python Programming:** Solid understanding of Python programming fundamentals, including data types, control structures (loops, conditionals), functions, modules, and object-oriented programming (OOP) concepts (classes, objects).
- **Tkinter GUI Development:** Basic to intermediate knowledge of Tkinter for creating GUI elements (windows, frames, widgets like labels, buttons, entry fields, listboxes/treeviews), event handling, and layout management (pack, grid, place).
- **Relational Database Concepts:** Understanding of relational database theory, including tables, primary keys, foreign keys, relationships, and normalization.

- **SQL (Structured Query Language):** Proficiency in writing SQL queries for MySQL, including CREATE TABLE, INSERT, SELECT, UPDATE, DELETE statements, and JOIN operations.
- **MySQL Database Management:** Basic familiarity with creating databases and tables in MySQL, managing users, and ensuring data integrity.
- **Software Development Lifecycle (SDLC):** A basic understanding of the phases involved in software development (requirements, design, implementation, testing, deployment).

Chapter 7

# 7. System Analysis and Limitations:

## 7.1 System Analysis:

### 7.1.1 Existing System and its Problems (Problem Analysis):

Currently, many smaller blood donation centers, clinics, or even some larger institutions might rely on:

- **Manual Systems:** Paper-based registers, logbooks, and card files for donor information, blood inventory, and transaction records. This often leads to:
    - **Inefficiency:** Slow data retrieval, difficulty in searching for specific blood types or donors.
    - **Errors:** Prone to human errors in data entry, calculations, and record maintenance.
    - **Data Redundancy:** Duplicate entries and inconsistent information.
    - **Poor Inventory Control:** Difficulty in tracking blood unit expiry dates, leading to wastage or shortages.
    - **Limited Reporting:** Generating reports is time-consuming and often lacks comprehensive insights.
    - **Security Risks:** Physical records are susceptible to loss, damage, or unauthorized access.
- **Outdated or Disparate Digital Systems:** May use basic spreadsheets or standalone, non-integrated software that doesn't offer comprehensive functionality or real-time updates. This can result in data silos and lack of a unified view.

These inefficiencies can critically impact the ability to respond quickly to urgent blood needs, manage donor relations effectively, and ensure the overall safety and quality of the blood supply chain.

### 7.1.2 Proposed System:

The proposed "Blood Donation Application" is a desktop software solution developed using Python with a Tkinter GUI and a MySQL database backend. It aims to digitize and streamline the core processes of blood donation management.

- **Purpose:** To provide an efficient, reliable, and user-friendly system for managing donor information, tracking blood inventory, recording donations and requests, and generating basic operational reports.
- **Functionalities (High-Level):** The system will allow administrators/staff to perform tasks such as user authentication, donor registration and management, blood unit addition and tracking, inventory viewing, donation logging, basic request management, and report generation.

- **Benefits:**
    - **Improved Efficiency:** Faster data entry, retrieval, and searching.
    - **Enhanced Accuracy:** Reduced manual errors through validated data input and consistent storage.
    - **Better Inventory Management:** Real-time view of blood stock, easier tracking of expiry dates, minimizing wastage.
    - **Streamlined Donor Management:** Centralized donor database for quick access to donor history and contact information.
    - **Informed Decision Making:** Access to basic reports for operational insights.
    - **Improved Data Security:** Centralized database with potential for password protection and backup.

## 7.2 Limitations:

While the proposed system aims to provide significant improvements, it will have certain limitations, especially given its nature as a desktop application developed with specific technologies:

1. **Standalone Desktop Application:**
    - **Limited Accessibility:** The application can only be accessed from the specific computer(s) where it is installed. It does not offer remote access via the web or mobile devices.

- **Data Sharing Constraints:** Real-time data sharing across multiple locations or with other institutions is not inherently supported.

2. **Single-User Focus (Typically):**
    - Unless specifically designed for multi-user concurrent access (which adds complexity with Tkinter and local database connections), it's often best suited for a single administrator or a few users operating non-concurrently on the same machine or a shared database with careful management.
3. **Scalability:**
    - While MySQL can scale, a Tkinter desktop application is generally not designed for very large-scale enterprise-level operations with hundreds of concurrent users or massive datasets compared to web-based enterprise solutions.

4. **Manual Data Entry:**
    - The system relies on manual input for all data. There's no integration for automated data capture from lab equipment or external systems.

5. **Basic Reporting:**
    - Reporting capabilities will be limited to predefined simple reports. Complex ad-hoc querying or advanced business intelligence features are not in scope.
6. **Security (Local System Dependent):**
    - While application-level login can be implemented, overall data security heavily depends on the security of the underlying operating system and physical security of the machine hosting the database and application. Advanced threat protection might be limited.
7. **No Real-time Inter-Bank Connectivity:**
    - The system operates in isolation and cannot connect with other blood banks or a central national registry in real-time.
8. **Maintenance and Updates:**
    - Updates and maintenance require deploying new versions of the application to each machine where it is installed.
9. **Platform Dependency (Minor):**
    - While Python and Tkinter are cross-platform, slight differences in UI rendering or behavior might occur across different operating systems. Packaging for different OS also needs consideration.
10. **No Automated Backup/Recovery (unless specifically implemented):**

- Database backup and recovery procedures would likely be manual or reliant on external MySQL tools unless specific scripts/features are built into the application.

# Chapter 7

## 7.1 Role Assignment and Responsibilities:

Effective project execution relies on a clear division of tasks and responsibilities among team members. For the development of the Blood Donation Application, the roles have been assigned as follows to ensure specialized focus and efficient progress:

### 7.1.1 Javed Rain: GUI Developer

- **Primary Role:** Responsible for the design, development, and implementation of the Graphical User Interface (GUI) for the Blood Donation Application.
- **Key Responsibilities:**
  - Designing an intuitive and user-friendly interface using Python's Tkinter library.
  - Translating functional requirements into interactive visual elements such as windows, forms, buttons, labels, and data display areas.
  - Ensuring the GUI is responsive and provides clear feedback to user actions.
  - Implementing event handling for user interactions within the GUI (e.g., button clicks, data entry).
  - Collaborating with Person B (Database Management) to ensure seamless data flow between the front-end interface and the back-end database.
  - Conducting usability testing on the GUI components to ensure ease of use.
  - Iteratively refining the GUI based on feedback and testing outcomes.

### 7.1.2 Jai Kishan: Database Management

- **Primary Role:** Responsible for all aspects of database design, implementation, and management using MySQL.
- **Key Responsibilities:**
  - Designing the logical and physical database schema for the Blood Donation Application, including defining tables, columns, data types, primary keys, foreign keys, and relationships.
  - Creating and configuring the MySQL database and its tables.

- Implementing data validation rules and constraints within the database to ensure data integrity and consistency.
- Developing and optimizing SQL queries for Create, Read, Update, and Delete (CRUD) operations required by the application.
- Managing the connectivity between the Python application and the MySQL database, primarily using the mysql.connector-python library.
- Ensuring efficient data storage and retrieval.
- Planning for basic database backup and recovery strategies (if within scope).
- Collaborating with Person A (GUI Developer) to provide necessary data access points and with Person C (Project Manager & Security Tester) regarding data security aspects.

### 7.1.3 Tanshul Deshwal : Project Manager & Security Tester

- **Primary Roles:**
  - **Project Manager:** Overseeing the overall project lifecycle, ensuring tasks are completed on time and objectives are met.
  - **Security Tester:** Responsible for evaluating and testing the security aspects of the application.
- **Key Responsibilities (as Project Manager):**
  - Defining the project scope, objectives, and deliverables in collaboration with the team.
  - Developing the project plan, including timelines and milestones.
  - Coordinating tasks and responsibilities among team members (Person A and Person B).
  - Monitoring project progress against the plan and addressing any deviations.
  - Facilitating communication and collaboration within the team.
  - Managing project documentation and ensuring all deliverables are met.
  - Identifying and mitigating project risks.

- **Key Responsibilities (as Security Tester):**
  - Identifying potential security vulnerabilities in the application, particularly concerning data access, user authentication, and data protection within the MySQL database.
  - Testing the implemented security measures, such as the admin login functionality and data handling processes.
  - Evaluating the application for common security flaws relevant to desktop applications (e.g., insecure storage of credentials if applicable, input validation to prevent basic injection attacks).

- Recommending security best practices and improvements to Person A and Person B.
- Documenting security testing procedures and findings.

# Chapter 8

## 8. GUI Design:

The Graphical User Interface (GUI) for the Blood Donation Application has been designed with a primary focus on simplicity, intuitiveness, and ease of use for administrators and blood bank staff. Python's Tkinter library was utilized to create all visual elements, ensuring a consistent and functional user experience for this desktop application. The design aims to streamline workflows, minimize the learning curve, and allow users to perform necessary tasks efficiently.

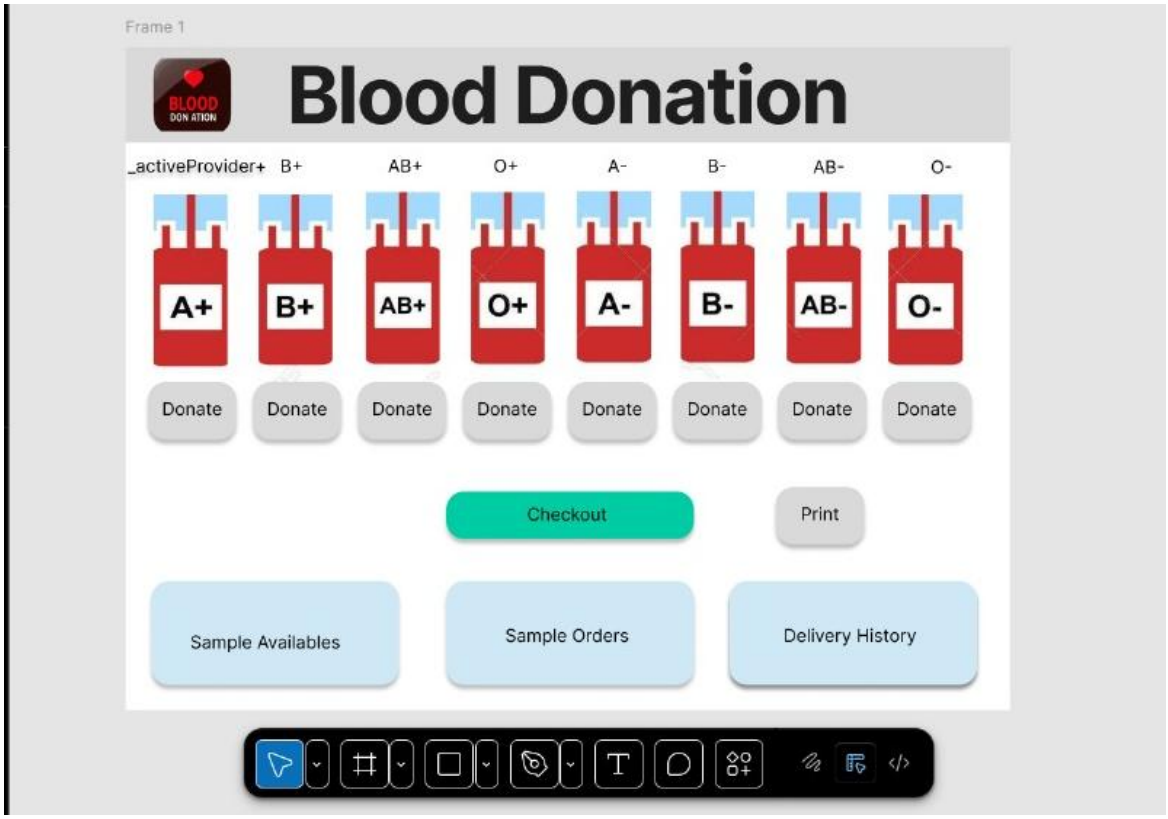
## 8.1 Overall Design Philosophy:

- **Clarity:** All labels, buttons, and input fields are clearly named to avoid ambiguity.
- **Consistency:** A consistent layout, color scheme (if any), and widget style are maintained across different screens to provide a familiar environment for the user.
- **Efficiency:** Forms and data displays are organized to allow for quick data entry and retrieval.
- **Feedback:** The system provides appropriate visual feedback for user actions, such as confirmation messages upon successful operations or error messages when issues arise.
- **Navigation:** Clear navigational pathways are provided, typically through a main menu or dashboard, to allow users to easily move between different modules of the application.

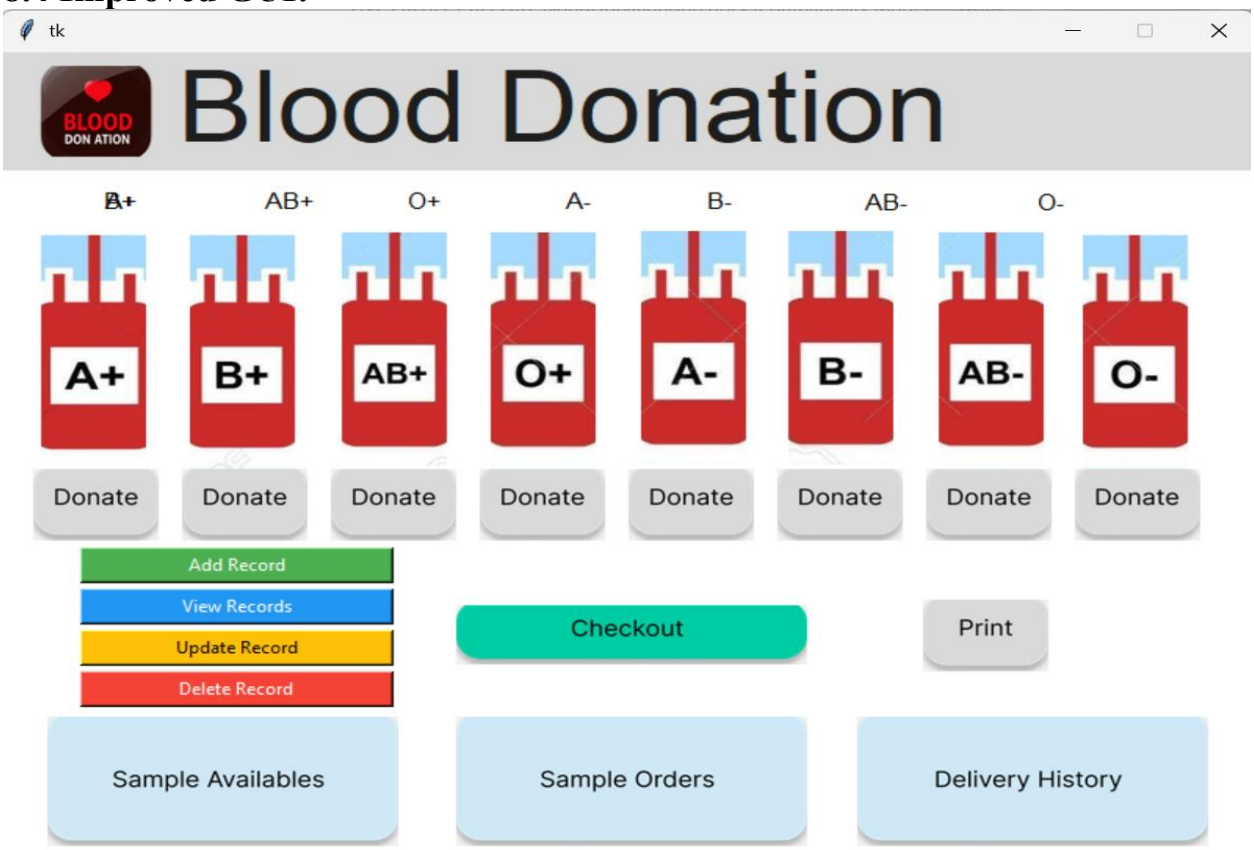
## 8.2 Common UI Elements and Conventions:

- **Navigation Bar/Menu:** A consistent menu or set of navigation buttons might be present on most screens (post-login) to allow easy switching between modules.
- **Buttons:** Standardized button styles (e.g., "Save," "Cancel," "Search," "Add," "Delete") are used. Destructive actions like "Delete" will typically include a confirmation dialog.
- **Feedback Messages:** A dedicated area (e.g., a status bar or pop-up dialogs) is used to display success messages, error alerts, or warnings to the user.
- **Data Display:** Tables or listboxes are used for displaying multiple records, often with sortable columns or search/filter capabilities.
- **Forms:** Data entry forms are organized logically with clear labels for each field. Input validation is implemented to guide the user and prevent incorrect data entry where possible.

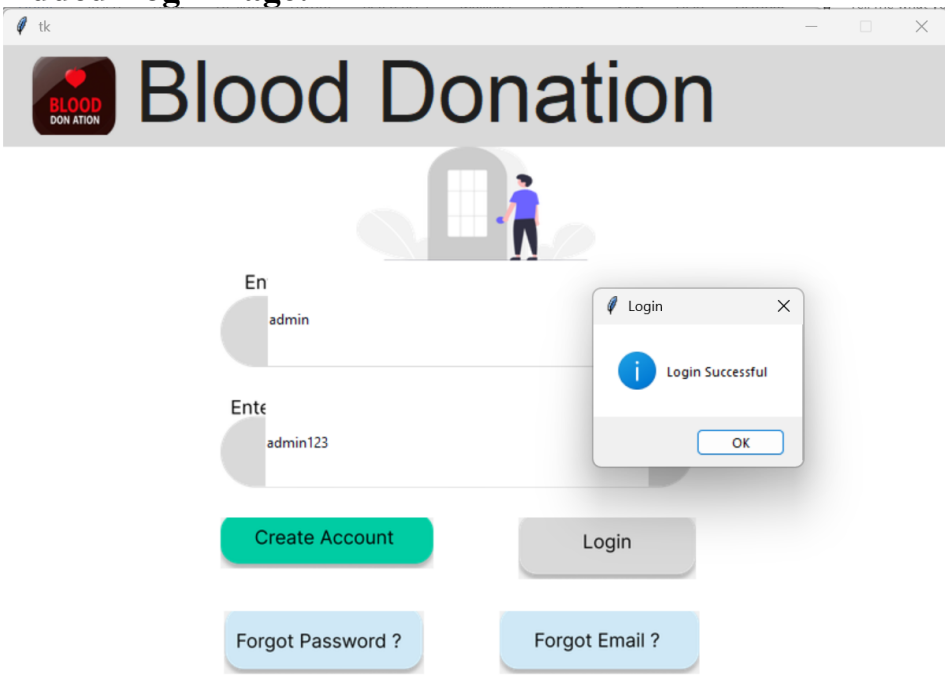
## 8.3 Prototype of GUI:



## 8.4 Improved GUI:




## Added Login Page:



Chapter 9

## 9. Database Design:

The database is a critical component of the Blood Donation Application, serving as the central repository for all data related to donors, blood inventory, user administration, donations, blood requests, and external blood orders. This design utilizes MySQL, a relational database management system (RDBMS), to ensure data integrity, consistency, and efficient retrieval. The design aims to minimize redundancy and support the application's functional requirements.

## 9.1 Design Goals

- **Data Integrity:** Ensuring accuracy and consistency of data through constraints like primary keys, foreign keys, and data type restrictions.
- **Minimize Redundancy:** Normalizing data to reduce repetition and improve storage efficiency.
- **Efficiency:** Structuring data to allow for quick querying and retrieval, supporting responsive application performance.
- **Scalability:** Providing a foundation that can be reasonably scaled if the application's scope expands in the future (within the context of a desktop application).
- **Support for Application Functionalities:** Directly enabling all CRUD operations and reporting features outlined for the application.

| | Standard | Standard | Standard | Standard |
|---|---|---|---|---|
| 1 | Column Name | Data Type | Constraints | Description |
| 2 | bid | INT | PRIMARY KEY | Unique identifier for the order |
| 3 | Provider | VARCHAR(255) | NOT NULL | Name of the provider |
| 4 | bType | ENUM('A+','A-','B+','B-','O+','O-','AB+','AB-') | NOT NULL | Blood type |
| 5 | orderDate | DATE | NOT NULL | Date when the order was placed |
| 6 | deliveryDate | DATE | NOT NULL | Date when the order was delivered |
| 7 | senderAddress | VARCHAR(255) | NOT NULL | Address of the sender |
| 8 | receiverAddress | VARCHAR(255) | NOT NULL | Address of the receiver |
| 9 | senderContact | VARCHAR(20) | NOT NULL | Contact information for the sender |
| 10 | receiverContact | VARCHAR(20) | NOT NULL | Contact information for the receiver |

## 9.2 CREATE Operation

**Function Name:** Algorithm_CreateOrder

**Purpose:** To add a new blood order record to the collection.

**Inputs:**

- **provider**_data**:** The provider's name.
- **bType**_data: The blood type.
- **orderDate**_data: The date the order was placed.

- **deliveryDate**_data: The expected or actual delivery date.
- **senderAddress**_data: The sender's address.
- **receiverAddress**_data: The receiver's address.
- **senderContact**_data**:** The sender's contact information.
- **receiverContact_**data: The receiver's contact information.

### Output:

- new_bid: The unique identifier (bid) assigned to the newly created order, or an error status if creation failed.

## 9.3 READ Operation

### 2.1. Read a Specific Order by BID

**Function Name:** Algorithm_ReadOrderByBID

**Purpose:** To retrieve a specific blood order record from the collection using its unique bid.

**Input:**

- target_bid: The bid of the order to retrieve.

**Output:**

- The order record if found, or a "Not Found" status.

### 9.3.1 Read All Orders

**Function Name:** Algorithm_ReadAllOrders

**Purpose:** To retrieve all blood order records from the collection.

**Input:** None

**Output:**

- A list containing all blood order records, or an empty list if the collection is empty.

## 9.4 UPDATE Operation:

**Function Name:** Algorithm_UpdateOrder

**Purpose:** To modify the details of an existing blood order record in the collection.

**Inputs:**

- target_bid: The bid of the order to update.
- new_provider_data (Optional): New provider's name.
- new_bType_data (Optional): New blood type.
- new_orderDate_data (Optional): New order date.
- new_deliveryDate_data (Optional): New delivery date.
- new_senderAddress_data (Optional): New sender's address.
- new_receiverAddress_data (Optional): New receiver's address.
- new_senderContact_data (Optional): New sender's contact.
- new_receiverContact_data (Optional): New receiver's contact. *(Note: Inputs for fields to be updated are optional; only provided fields will be updated.)*

### Output:

- "Success" status if the update was successful.
- "Not Found" status if the order with target_bid does not exist.
- "Invalid Input" status if new data fails validation.

## 9.5 DELETE Operation:

**Function Name:** Algorithm_DeleteOrder

**Purpose:** To remove a specific blood order record from the collection.

**Input:**

- target_bid: The bid of the order to delete.

**Output:**

- "Success" status if the deletion was successful.
- "Not Found" status if the order with target_bid does not exist.

Chapter 10

# 10. Security Testing:

## 10.1 Introduction:

Security testing is a critical phase in the development of the Blood Donation Application to ensure the confidentiality, integrity, and availability of the application and its data. The primary goal of security testing for this project is to identify and mitigate potential vulnerabilities, particularly focusing on unauthorized access, data protection, and ensuring the application behaves securely under various conditions. The testing approach primarily involved manual testing, focused code reviews for security aspects, and simulating common attack vectors relevant to a desktop application environment. This was overseen by Person C, designated as the Project Manager and Security Tester.

## 10.2 Scope of Security Testing

The scope of security testing encompassed the following key areas and modules of the Blood Donation Application:

- **User Authentication:** The admin login mechanism, including credential validation and password storage.
- **Data Entry and Input Validation:** All forms where users input data (e.g., Donor Registration, Blood Unit Addition, Blood Order Creation) to prevent invalid data entry and potential injection flaws.
- **Database Interaction:** Review of how CRUD operations interact with the MySQL database, focusing on the use of parameterized queries.
- **Access Control:** Ensuring that application functionalities are accessible only after successful authentication.
- **Error Handling:** Reviewing how the application handles errors and exceptions to prevent information leakage.
- **Data Protection:** Reviewing the handling of sensitive donor information and admin credentials.

## 10.3 Types of Security Tests Performed

- **Authentication Testing:**
    - **Credential Validation:** Testing with valid and invalid username/password combinations.
    - **Password Security:** Verifying that passwords are not stored in plain text in the database (confirming implementation of hashing).
    - **Brute-Force Considerations:** While full-scale automated brute-force testing was not performed, the importance of strong password policies for the admin account was noted.

- **Authorization Testing:**
    - **Access Control Checks:** Attempting to access administrative functionalities directly without logging in (e.g., by trying to invoke internal functions or open windows directly, if possible within the Tkinter framework).
    - **Session Integrity:** Ensuring that after logout, previously accessed functionalities are no longer available without re-authentication.
- **Input Validation Testing:**
    - **Data Type and Format Testing:** Inputting incorrect data types (e.g., text in numeric fields), out-of-range values, and improperly formatted dates into forms.
    - **Boundary Value Analysis:** Testing inputs at the edges of valid ranges.
    - **Special Character Handling:** Inputting common special characters (e.g., ', ;, <, >) into input fields to check for unexpected behavior or basic SQL injection vulnerabilities (though mysql.connector with parameterized queries inherently mitigates most direct SQLi).
- **Data Security Review:**
    - **Code Review:** Reviewing Python code responsible for handling sensitive data to ensure it's not unnecessarily exposed or logged insecurely.
    - **Database Schema Review:** Confirming that sensitive fields in the MySQL database are appropriately defined.
- **Error Handling Review:**
    - Triggering various error conditions (e.g., database connection failure, invalid input) to examine the error messages displayed to the user.
    - Ensuring error messages are user-friendly and do not reveal sensitive system information or stack traces that could be exploited.

### 10.4 Tools to be used :-

- Manual Code Review (Python and SQL query construction).
- Manual functional testing by simulating user actions and malicious inputs.
- MySQL Workbench (or similar) for inspecting database structure and data.

### 10.5 Actual Testing : To be done after finishing Project.

### 10.6 Limitations of Testing

- The security testing was primarily manual and based on common vulnerabilities relevant to desktop applications. It did not involve automated penetration testing tools or exhaustive testing for all possible attack vectors.
- Testing focused on the application layer and its interaction with the database. Security of the underlying operating system, network (if the database is hosted on a separate server),

and physical security are outside the scope of this application-specific testing but are crucial for overall system security.
- Time and resource constraints limited the depth of some security test scenarios.

# Chapter 11

## 11.Future Scope

The Blood Donation application presents several avenues for future development and enhancement to broaden its impact and efficiency:

- Expanded Platform Availability: Development of dedicated mobile applications (iOS/Android) and a comprehensive web portal for integration with government healthcare systems and hospital networks, ensuring wider accessibility and official operational support.

- AI-Powered Blood Analysis: Integration of Machine Learning (ML) capabilities to analyze blood samples via advanced sensors. This feature would enable early detection of diseases or identification of specific patterns, significantly enhancing blood safety and screening processes.

- Streamlined Logistics with API Integration: Implementation of an online delivery system facilitated by API integrations with third-party logistics and transport services. This would optimize the efficient and timely delivery of blood to hospitals and recipients.

# Chapter 12

## 12.1 Project Recap

This project focused on developing a fundamental Blood Donation application, primarily a login interface, using Python's Tkinter for the graphical user interface and MySQL for database management. The application aims to provide a basic framework for managing user and administrator access to a blood donation system.

## 12.2 Outcomes and Deliverables

The key outcome of this phase of the project is a functional login module capable of authenticating users against a MySQL database. This includes a user-friendly graphical interface, secure handling of credentials (with future plans for advanced hashing), and the ability to transition to a main application upon successful login.

## 12.3 Learning and Skills Gained

- GUI Development: Utilizing Tkinter for creating interactive and visually appealing desktop applications.

- Database Integration: Connecting Python applications to MySQL databases using

mysql.connector for data storage and retrieval.

- Basic Authentication: Implementing a login mechanism for user verification.
- Project Structure: Understanding the organization of application files and asset management.
- Error Handling: Implementing try-except blocks for database connections and module imports.

## 12.4 Challenges Faced

- Module Dependencies: Resolving ModuleNotFoundError by ensuring all necessary Python libraries (like mysql-connector-python) were correctly installed.
- Asset Path Management: Addressing issues with hardcoded file paths to ensure the application's portability across different environments.
- Security Considerations (Ongoing): Identifying and planning for the crucial implementation of secure password hashing practices to protect user data, which is paramount for any real-world application.

## 12.5 Final Words

We have created a Blood donation application with minimal features to handle blood donations from users. Further enhancements, including robust account creation, adherence to secure MySQL practices, and comprehensive bug fixes, will be incorporated in future releases.

# Chapter 13

## 13.References

**REFERENCES / BIBLIOGRAPHY**

The following resources were consulted during the research, design, and development phases of the Blood Donation Application:

**1. Official Documentation:**


1. **Python Software Foundation.** *Python 3 Documentation.* Available at: https://docs.python.org/3/ (Accessed: May 2025).

2. **Python Software Foundation.** *Tkinter — Python interface to Tcl/Tk.* Available at: https://docs.python.org/3/library/tkinter.html (Accessed: May 2025).
3. **Oracle Corporation.** *MySQL Documentation.* Available at: https://dev.mysql.com/doc/ (Accessed: May 2025).
4. **Oracle Corporation.** *MySQL Connector/Python Developer Guide.* Available at: https://dev.mysql.com/doc/connector-python/en/ (Accessed: May 2025).

**2. Books:**

5. Lutz, Mark. *Learning Python.* 5th ed., O'Reilly Media, 2013.

6. Summerfield, Mark. *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming.* Prentice Hall, 2007.

7. Welling, Luke, and Laura Thomson. *PHP and MySQL Web Development.* 5th ed., Addison-Wesley Professional, 2016.

8. DuBois, Paul. *MySQL.* 5th ed. (Developer's Library), Addison-Wesley Professional, 2018.

9. Pressman, Roger S., and Bruce R. Maxim. *Software Engineering: A Practitioner's Approach.* 9th ed., McGraw-Hill Education, 2019.

**3. Online Tutorials and Resources:**

10. *GeeksforGeeks - Python Programming.* Available at: https://www.geeksforgeeks.org/python-programming-language/ (Accessed: May 2025).
11. *Real Python - Python Tutorials.* Available at: https://realpython.com/ (Accessed: May 2025).
12. *TutorialsPoint - Python Tkinter.* Available at: https://www.tutorialspoint.com/python/python_gui_programming.htm 1 (Accessed: May 2025).

**4. Ai and other tools**

**13. ChatGPT, Gemini**

**14.Mermaid flowchart**

**15.Python IDLE**

**16.VS code**

**18.Terminal**

Chapter 14

## 14. Appendix & Code Snippets

### 14.1 For Main Application Interface:

***# Not for Login***

***#GUI interface to use buttons , canvas and database linking with gui***

```
import threading

from pathlib import Path

# from tkinter import *

# Explicit imports to satisfy Flake8

from tkinter import Tk, Canvas, Entry, Text, Button, PhotoImage

OUTPUT_PATH = Path(_file_).parent

ASSETS_PATH = OUTPUT_PATH /
Path(r"C:\Users\gaura\OneDrive\Documents\LiveProject6Sem\BloodDonationApp_0.0.2\Blood
DonationApp_0.0.2\build\assets\frame0")

import mysql.connector

from tkinter import messagebox, simpledialog

try:

    mycon = mysql.connector.connect(
```

```
            host="localhost",
            user="root",
            password="10JavedRain10",
            database="BloodDonation"
        )
        cursor = mycon.cursor()
except Exception as e:
        messagebox.showerror("Database Error", str(e))
class BloodManager:
    def _init_(self, b_type=None):
        self.b_type = b_type
        self.connection = mysql.connector.connect(
            host="localhost",
            user="admin",
            password="10JavedRain10",  # or your password
            database="BloodDonation"
        )
        self.cursor = self.connection.cursor()



    def add_record(self):
        try:
            b_id = simpledialog.askstring("Input", "Blood ID:")
```

```
            provider = simpledialog.askstring("Input", "Provider Name:")
            price = int(simpledialog.askstring("Input", "Price:"))
            btype = self.b_type if self.b_type else simpledialog.askstring("Input", "Blood Type (A+, O-, etc):")
            qty = int(simpledialog.askstring("Input", "Quantity:"))
            per_litre = int(simpledialog.askstring("Input", "Per Litre Price:"))
            self.cursor.execute(
                "INSERT INTO bloodData (b_id, b_providerName, b_price, b_type, b_qnt, b_perLitre) VALUES (%s, %s, %s, %s, %s, %s)",
                (b_id, provider, price, btype, qty, per_litre)
            )
            self.connection.commit()
            messagebox.showinfo("Success", "✅ Record Added")
        except Exception as e:
            messagebox.showerror("Error", str(e))
    def view_records(self):
        try:
            self.cursor.execute("SELECT * FROM bloodData")
            records = self.cursor.fetchall()
            record_str = "\n".join(str(row) for row in records)
            messagebox.showinfo("All Records", record_str)
        except Exception as e:
            messagebox.showerror("Error", str(e))
    def update_record(self):
```

```
        try:
            b_id = simpledialog.askstring("Input", "Enter Blood ID to update:")
            new_qty = int(simpledialog.askstring("Input", "Enter new quantity:"))
            self.cursor.execute("UPDATE bloodData SET b_qnt=%s WHERE b_id=%s", (new_qty,
b_id))
            self.connection.commit()
            messagebox.showinfo("Success", "✅ Record Updated")
        except Exception as e:
            messagebox.showerror("Error", str(e))
    def delete_record(self):
        try:
            b_id = simpledialog.askstring("Input", "Enter Blood ID to delete:")
            self.cursor.execute("DELETE FROM bloodData WHERE b_id=%s", (b_id,))
            self.connection.commit()
            messagebox.showinfo("Deleted", "🗑 Record Deleted")
        except Exception as e:
            messagebox.showerror("Error", str(e))
from tkinter import messagebox, simpledialog
def checkout_total():
 try:
   con = mysql.connector.connect(
   host="localhost",
   user="root",
   password="10JavedRain10",
```

```
    database="BloodDonation"
    )
    cursor = con.cursor()
    cursor.execute("SELECT SUM(b_price * b_qnt) FROM bloodData")
  except Exception as e:
    messagebox.showerror("Error", str(e))
def add_record():
    try:
        b_id = simpledialog.askstring("Input", "Blood ID:")
        name = simpledialog.askstring("Input", "Provider Name:")
        price = int(simpledialog.askstring("Input", "Price:"))
        btype = simpledialog.askstring("Input", "Blood Type (A+, O-, etc):")
        qty = int(simpledialog.askstring("Input", "Quantity:"))
        per_litre = int(simpledialog.askstring("Input", "Per Litre:"))
        mycon = mysql.connector.connect(host="localhost", user="root",
password="10JavedRain10", database="BloodDonation")
        cursor = mycon.cursor()
        cursor.execute("INSERT INTO bloodData VALUES (%s, %s, %s, %s, %s, %s)",
                       (b_id, name, price, btype, qty, per_litre))
        mycon.commit()
        messagebox.showinfo("Success", "✅ Record Added")
    except Exception as e:
        messagebox.showerror("Error", str(e))
def view_records():
```

```
    try:
        mycon = mysql.connector.connect(host="localhost", user="root", password="10JavedRain10", database="BloodDonation")
        cursor = mycon.cursor()
        cursor.execute("SELECT * FROM bloodData")
        records = cursor.fetchall()
        record_str = "\n".join(str(row) for row in records)
        messagebox.showinfo("All Records", record_str)
    except Exception as e:
        messagebox.showerror("Error", str(e))
def update_record():
    try:
        b_id = simpledialog.askstring("Input", "Enter Blood ID to update:")
        new_qty = int(simpledialog.askstring("Input", "Enter new quantity:"))

        mycon = mysql.connector.connect(host="localhost", user="root", password="10JavedRain10", database="BloodDonation")
        cursor = mycon.cursor()
        cursor.execute("UPDATE bloodData SET b_qnt=%s WHERE b_id=%s", (new_qty, b_id))
        mycon.commit()
        messagebox.showinfo("Success", "✅ Record Updated")
    except Exception as e:
        messagebox.showerror("Error", str(e))
def delete_record():
```

```
    try:
        b_id = simpledialog.askstring("Input", "Enter Blood ID to delete:")
        mycon = mysql.connector.connect(host="localhost", user="root",
password="10JavedRain10", database="BloodDonation")
        cursor = mycon.cursor()
        cursor.execute("DELETE FROM bloodData WHERE b_id=%s", (b_id,))
        mycon.commit()
        messagebox.showinfo("Deleted", "🗑 Record Deleted")
    except Exception as e:
        messagebox.showerror("Error", str(e))
def relative_to_assets(path: str) -> Path:
    return ASSETS_PATH / Path(path)
window = Tk()
window.geometry("800x600")
window.configure(bg = "#FFFFFF")
canvas = Canvas(
    window,
    bg = "#FFFFFF",
    height = 600,
    width = 800,
    bd = 0,
    highlightthickness = 0,
    relief = "ridge"
)
```

```
canvas.place(x = 0, y = 0)
button_image_1 = PhotoImage(
    file=relative_to_assets("button_1.png"))
button_1 = Button(
    image=button_image_1,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("Donated for O- !"),
    relief="flat"
)
button_1.place(
    x=685.0,
    y=303.0,
    width=80.0,
    height=52.0
)
canvas.create_rectangle(
    0.0,
    0.0,
    800.0,
    86.0,
    fill="#D9D9D9",
    outline="")
```

```
canvas.create_text(
    112.0,
    0.0,
    anchor="nw",
    text="Blood Donation",
    fill="#1E1E1E",
    font=("Inter Bold", 72 * -1)
)
image_image_1 = PhotoImage(
    file=relative_to_assets("image_1.png"))
image_1 = canvas.create_image(
    58.0,
    218.0,
    image=image_image_1
)
image_image_2 = PhotoImage(
    file=relative_to_assets("image_2.png"))
image_2 = canvas.create_image(
    441.0,
    215.0,
    image=image_image_2
)
image_image_3 = PhotoImage(
```

```
    file=relative_to_assets("image_3.png"))
image_3 = canvas.create_image(
    723.0,
    218.0,
    image=image_image_3
)
canvas.create_text(
    58.0,
    99.0,
    anchor="nw",
    text="  A+\n",
    fill="#000000",
    font=("Inter", 16 * -1)
)
canvas.create_text(
    360.0,
    99.0,
    anchor="nw",
    text="A-",
    fill="#000000",
    font=("Inter", 16 * -1)
)
canvas.create_text(
```

```
    661.0,
    100.0,
    anchor="nw",
    text="O-\n",
    fill="#000000",
    font=("Inter", 16 * -1)
)
button_image_2 = PhotoImage(
    file=relative_to_assets("button_2.png"))
'''
button_2 = Button(
    image=button_image_2,
    borderwidth=0,
    highlightthickness=0,
    command=handle_add_A,
    relief="flat"
)
'''
def handle_add_A():
    def db_task():
        manager = BloodManager("A+")
        manager.add_record()
        print("✅ Successfully added record for blood type: A+")
```

```
    threading.Thread(target=db_task).start()

button_2 = Button(

    image=button_image_2,

    borderwidth=0,

    highlightthickness=0,

    command=lambda: handle_add_A,#//(BloodManager("A+").add_record(), print(" ✅ Added record for A+")),#//,window, text="Donate A+", bg="#2196F3", fg="white"),#print("button_2 clicked"),

    relief="flat",

)

button_2.place(

    x=20.0,

    y=303.0,

    width=80.0,

    height=52.0

)

button_image_3 = PhotoImage(

    file=relative_to_assets("button_3.png"))

button_3 = Button(

    image=button_image_3,

    borderwidth=0,

    highlightthickness=0,

    command=lambda: print("Donated for A- !"),

    relief="flat"
```

```
)
button_3.place(
    x=400.0,
    y=303.0,
    width=80.0,
    height=52.0
)
button_image_4 = PhotoImage(
    file=relative_to_assets("button_4.png"))
button_4 = Button(
    image=button_image_4,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("button_4 clicked"),
    relief="flat"
)
button_4.place(
    x=546.0,
    y=483.0,
    width=224.0,
    height=94.0
)
button_image_5 = PhotoImage(
```

```
    file=relative_to_assets("button_5.png"))
button_5 = Button(
    image=button_image_5,
    borderwidth=0,
    highlightthickness=0,
    command=lambda:print("Delivery History"),
    relief="flat"
)
button_5.place(
    x=546.0,
    y=483.0,
    width=224.0,
    height=94.0
)
button_image_6 = PhotoImage(
    file=relative_to_assets("button_6.png"))
button_6 = Button(
    image=button_image_6,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("SAMPLE orders"),
    relief="flat"
)
```

```
button_6.place(
    x=290.0,
    y=483.0,
    width=224.0,
    height=94.0
)
button_image_7 = PhotoImage(
    file=relative_to_assets("button_7.png"))
button_7 = Button(
    image=button_image_7,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: checkout_total(),
    relief="flat"
)
button_7.place(
    x=290.0,
    y=402.0,
    width=224.0,
    height=43.0
)
button_image_8 = PhotoImage(
    file=relative_to_assets("button_8.png"))
```

```
button_8 = Button(
    image=button_image_8,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("Donated for B+ !"),
    relief="flat"
)
button_8.place(
    x=115.0,
    y=303.0,
    width=80.0,
    height=52.0
)
button_image_9 = PhotoImage(
    file=relative_to_assets("button_9.png"))
button_9 = Button(
    image=button_image_9,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("Donated for O+ !"),
    relief="flat"
)
button_9.place(
```

```
    x=305.0,
    y=303.0,
    width=80.0,
    height=52.0
)
button_image_10 = PhotoImage(
    file=relative_to_assets("button_10.png"))
button_10 = Button(
    image=button_image_10,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("Donated for AB- !"),
    relief="flat"
)
button_10.place(
    x=590.0,
    y=303.0,
    width=80.0,
    height=52.0
)
image_image_4 = PhotoImage(
    file=relative_to_assets("image_4.png"))
image_4 = canvas.create_image(
```

```
    153.0,
    218.0,
    image=image_image_4
)
canvas.create_text(
    65.0,
    99.0,
    anchor="nw",
    text="B+",
    fill="#000000",
    font=("Inter", 16 * -1)
)
image_image_5 = PhotoImage(
    file=relative_to_assets("image_5.png"))
image_5 = canvas.create_image(
    345.0,
    216.0,
    image=image_image_5
)

canvas.create_text(
    259.0,
    99.0,
```

```
    anchor="nw",
    text="O+",
    fill="#000000",
    font=("Inter", 16 * -1)
)
image_image_6 = PhotoImage(
    file=relative_to_assets("image_6.png"))
image_6 = canvas.create_image(
    631.0,
    216.0,
    image=image_image_6
)
canvas.create_text(
    551.0,
    100.0,
    anchor="nw",
    text="AB-",
    fill="#000000",
    font=("Inter", 16 * -1)
)
import datetime
from tkinter import messagebox
def print_receipt():
```

```
    try:
        # Example static data — replace with actual input or DB query
        donor_name = "user "#entry_1.get()  # From your donor name entry
        blood_type = "A+"          # Hardcoded or from dropdown
        quantity = 2               # Quantity in Litres
        price = 1000               # Total Price
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        receipt = f"""
        Date: {timestamp}
        Donor: {donor_name}
        Blood Type: {blood_type}
        Quantity: {quantity}L
        Total Price: ₹{price}
        Thank you for your donation! 💉
        """
        print(receipt)  # Log to terminal
        messagebox.showinfo("Receipt", receipt)
        with open("donation_receipt.txt", "w") as f:
            f.write(receipt)

    except Exception as e:
        messagebox.showerror("Error", f"Failed to print receipt: {e}")
button_image_11 = PhotoImage(
```

```
    file=relative_to_assets("button_11.png"))
button_11 = Button(
    image=button_image_11,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print_receipt(),#print("button_11 clicked"),
    relief="flat"
)
button_11.place(
    x=588.0,
    y=398.0,
    width=80.0,
    height=52.0
)
image_image_7 = PhotoImage(
    file=relative_to_assets("image_7.png"))
image_7 = canvas.create_image(
    60.0,
    43.0,
    image=image_image_7
)
button_image_12 = PhotoImage(
    file=relative_to_assets("button_12.png"))
```

```
button_12 = Button(
    image=button_image_12,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("Donated for AB+ !"),
    relief="flat"
)
button_12.place(
    x=210.0,
    y=303.0,
    width=80.0,
    height=52.0
)
image_image_8 = PhotoImage(
    file=relative_to_assets("image_8.png"))
image_8 = canvas.create_image(
    250.0,
    216.0,
    image=image_image_8
)
canvas.create_text(
    168.0,
    99.0,
```

```
    anchor="nw",
    text="AB+",
    fill="#000000",
    font=("Inter", 16 * -1)
)
button_image_13 = PhotoImage(
    file=relative_to_assets("button_13.png"))
button_13 = Button(
    image=button_image_13,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("Donated for B- !"),
    relief="flat"
)
button_13.place(
    x=495.0,
    y=303.0,
    width=80.0,
    height=52.0
)image_image_9 = PhotoImage(
    file=relative_to_assets("image_9.png"))
image_9 = canvas.create_image(
    535.0,
```

```
    215.0,
    image=image_image_9
)
canvas.create_text(
    450.0,
    99.0,
    anchor="nw",
    text="B-",
    fill="#000000",
    font=("Inter", 16 * -1)
)
button_image_14 = PhotoImage(
    file=relative_to_assets("button_14.png"))
button_14 = Button(
    image=button_image_14,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("SAMPLE availables"),
    relief="flat"
)
button_14.place(
    x=29.0,
    y=483.0,
```

```
    width=224.0,
    height=94.0
)
Button(window, text="Add Record", command=add_record, bg="#4CAF50",
fg="white").place(x=50, y=360, width=200)
Button(window, text="View Records", command=view_records, bg="#2196F3",
fg="white").place(x=50, y=390, width=200)
Button(window, text="Update Record", command=update_record, bg="#FFC107",
fg="black").place(x=50, y=420, width=200)
Button(window, text="Delete Record", command=delete_record, bg="#F44336",
fg="white").place(x=50, y=450, width=200)
window.resizable(False, False)
window.mainloop()
```

## 14.2 For Login Interface:

```
#App Runner
#This provides the login interface
#import mysql.connector
from tkinter import messagebox
import subprocess
import sys
def login():
    username = entry_1.get()
    password = entry_2.get()
    try:
```

```
        mycon = mysql.connector.connect(
            host="localhost",
            user="admin",
            password="10JavedRain10",
            database="BloodDonation"
        )
        cursor = mycon.cursor()
        cursor.execute("SELECT * FROM users WHERE username=%s AND
password=%s", (username, password))
        if cursor.fetchone():
            messagebox.showinfo("Login", "Login Successful")
            mycon.close()
            window.destroy()
            subprocess.Popen([sys.executable, "gui.py"])
        else:
            messagebox.showerror("Login Failed", "Invalid credentials")
    except Exception as e:
        messagebox.showerror("Database Error", str(e))


# Add this button to your login window//[DONE]
#login_btn = Button(window, text="Login", command=login, bg="#4CAF50",
fg="white")
```

```
#login_btn.place(x=310, y=400, width=100, height=40)

# This file was generated by the Tkinter Designer by Parth Jadhav

# https://github.com/ParthJadhav/Tkinter-Designer

from pathlib import Path

# from tkinter import *

# Explicit imports to satisfy Flake8

from tkinter import Tk, Canvas, Entry, Text, Button, PhotoImage

OUTPUT_PATH = Path(_file_).parent

ASSETS_PATH = OUTPUT_PATH /
Path(r"C:\Users\gaura\OneDrive\Documents\LiveProject6Sem\BloodDonationApp
_0.0.2\BloodDonationApp_0.0.2\build\assets\frame1")

#frame1

def relative_to_assets(path: str) -> Path:

    return ASSETS_PATH / Path(path)

window = Tk()

window.geometry("800x600")

window.configure(bg = "#FFFFFF")

canvas = Canvas(

    window,

    bg = "#FFFFFF",

    height = 600,

    width = 800,
```

```
    bd = 0,
    highlightthickness = 0,
    relief = "ridge"
)
canvas.place(x = 0, y = 0)
canvas.create_rectangle(
    0.0,
    0.0,
    800.0,
    86.0,
    fill="#D9D9D9",
    outline="")
canvas.create_text(
    112.0,
    0.0,
    anchor="nw",
    text="Blood Donation",
    fill="#1E1E1E",
    font=("Inter Bold", 72 * -1)
)
button_image_1 = PhotoImage(
```

```
    file=relative_to_assets("button_1.png"))
button_1 = Button(
    image=button_image_1,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("button_1 clicked"),
    relief="flat"
)
button_1.place(
    x=187.0,
    y=478.0,
    width=169.0,
    height=53.0
)
button_image_2 = PhotoImage(
    file=relative_to_assets("button_2.png"))
button_2 = Button(
    image=button_image_2,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("button_2 clicked"),
```

```
    relief="flat"
)
button_2.place(
    x=420.0,
    y=478.0,
    width=169.0,
    height=53.0
)
button_image_3 = PhotoImage(
    file=relative_to_assets("button_3.png"))
button_3 = Button(
    image=button_image_3,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: print("Create Account (To be implemented)"),
    relief="flat"
)
button_3.place(
    x=184.0,
    y=399.0,
    width=180.0,
```

```
    height=43.0
)
#Login Button
button_image_4 = PhotoImage(
    file=relative_to_assets("button_4.png"))
button_4 = Button(
    image=button_image_4,
    borderwidth=0,
    highlightthickness=0,
    command=lambda: login(), #login_btn = Button(window, text="Login",
command=login, bg="#4CAF50", fg="white"),#print("button_4 clicked"),
    relief="flat"
)
button_4.place(
    x=436.0,
    y=399.0,
    width=150.0,
    height=52.0
)
image_image_1 = PhotoImage(
    file=relative_to_assets("image_1.png"))
image_1 = canvas.create_image(
```

```
    60.0,
    43.0,
    image=image_image_1
)
entry_image_1 = PhotoImage(
    file=relative_to_assets("entry_1.png"))
entry_bg_1 = canvas.create_image(
    384.0,
    233.0,
    image=entry_image_1
)
entry_1 = Entry(
    bd=0,
    bg="#FFFFFF",
    fg="#000716",
    highlightthickness=0
)
entry_1.place(
    x=224.0,
    y=193.0,
    width=320.0,
```

```
    height=78.0
)
entry_image_2 = PhotoImage(
    file=relative_to_assets("entry_2.png"))
entry_bg_2 = canvas.create_image(
    384.0,
    337.0,
    image=entry_image_2
)
entry_2 = Entry(
    bd=0,
    bg="#FFFFFF",
    fg="#000716",
    highlightthickness=0
)
entry_2.place(
    x=222.0,
    y=299.0,
    width=324.0,
    height=74.0
)
```

```
image_image_2 = PhotoImage(
    file=relative_to_assets("image_2.png"))
image_2 = canvas.create_image(
    400.0,
    134.0,
    image=image_image_2
)
window.resizable(False, False)
window.mainloop()
```