**DBMS MINI PROJECT**

**PROJECT REPORT**

**DATABASE MANANGEMENT SYSTEM FOR A BLOOD BANK**

**Submitted by:**

**1.A.Anand 205002008**

**2.Prithvikiran 205002069**

**3.Sahithi M.R 205002076**

INDEX

| Sno | Content |
| --- | --- |
| 1 | Problem statement and Description |
| 2 | Requirement Analysis |
| 3 | Entitiy Relationship Diagram |
| 4 | Database Schema |
| 5 | Normalisation |
| 6 | Coding and Implementation |
| 7 | Results |
| 8 | Appendix |
| 9 | Bibliography |

**1.Problem Statement :**

To create a database management system for a blood bank.

**2. Requirement Analysis of the application:**

The user will initially be given two options to login- donor login or patient login, the next frame will direct the user to a page where the user will be prompted for login credentials for both cases .

For new users an icon will be available to sign up, where the user details are collected and donor id or patient id is generated .

The users will be redirected to the original page once signed up.

The donor once logged in with correct credentials, a prompt to donate blood is displayed and user is free to confirm his/her consent to donate blood.

Later the application checks the eligibility for the same using SYSDATE , where last donation must be before 4months or 120 days.

On the other end, if a patient is logged in with correct credentials .

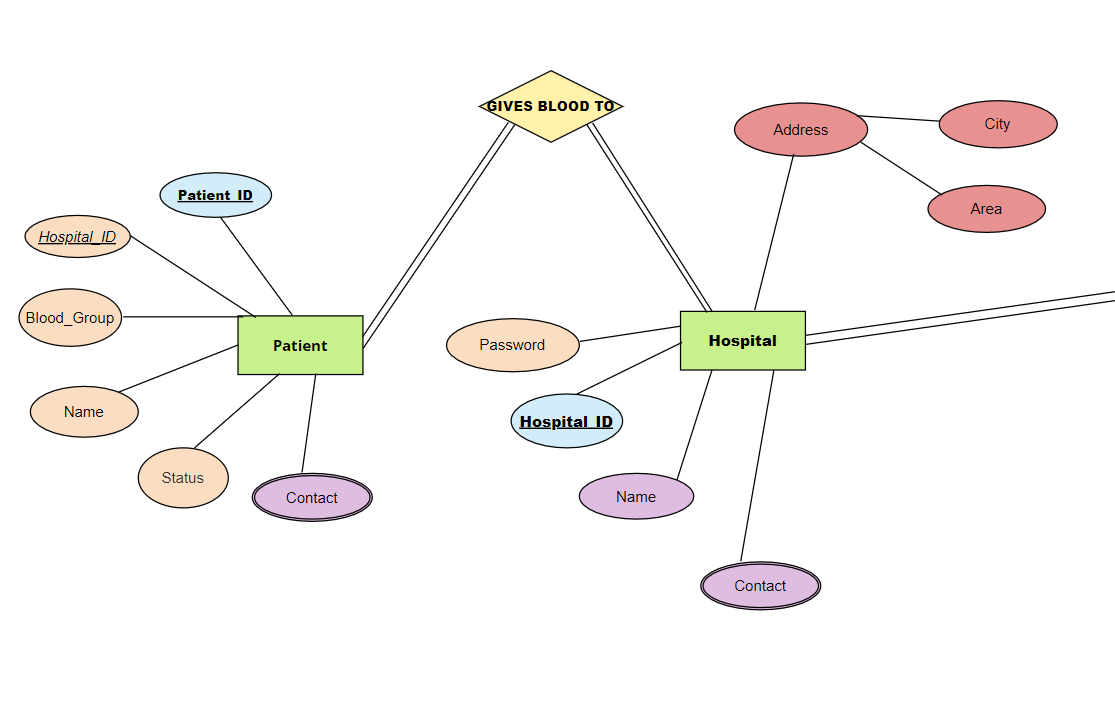
The patients are prompted for the blood type, quantity once submitted the availability of blood is checked and returned.

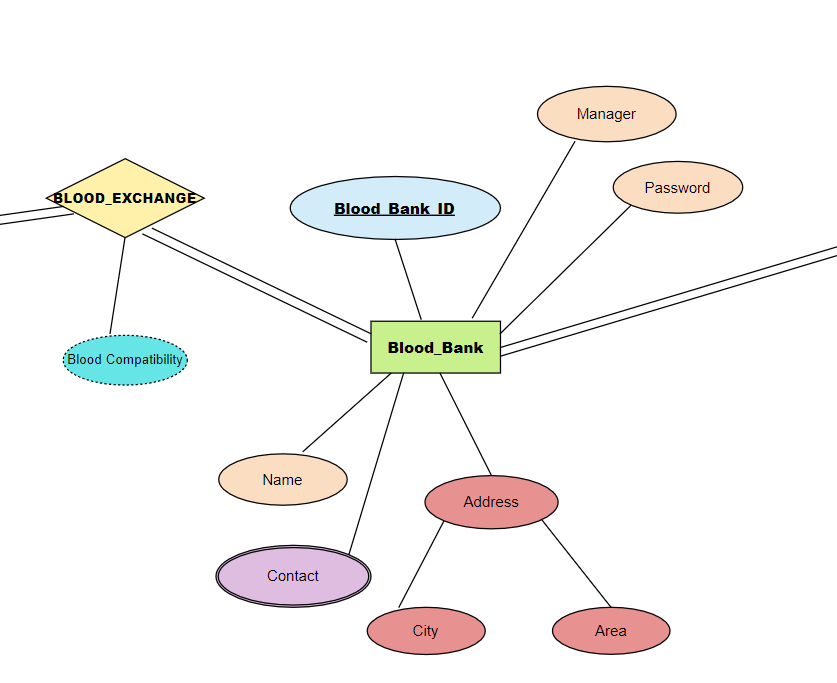
**Challenges :**

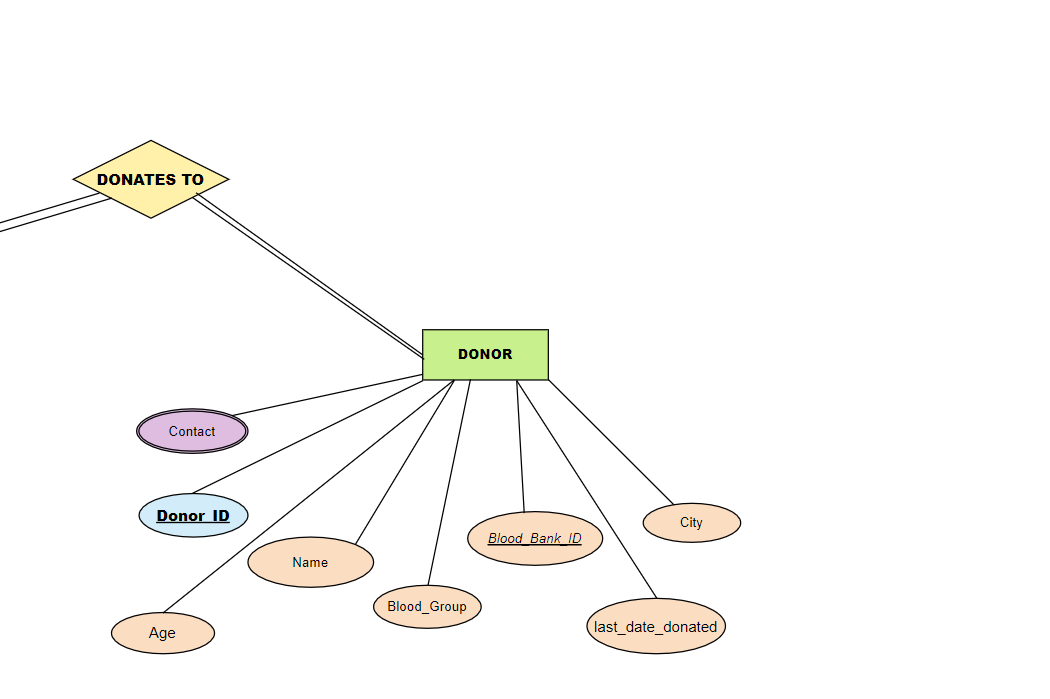
To enable an interactive user interface to facilitate optimum user experience.

To enable connection and fetch data from the SQL database.

**3. ENTITY RELATIONSHIP DIAGRAM**

****

****

****

**4. Database schema**

**Hospital table**

| Hospital\_id | Number | Primary key |
| --- | --- | --- |
| Name | Varchar |  |
| Address | Varchar | Unique |
| Contact no | Varchar | Unique |
| Password | Varchar | Unique |

**Blood bank table**

| Blood\_Bank\_ID | Number | Primary key |
| --- | --- | --- |
| Name | Varchar |  |
| Address | Varchar | Unique |
| Manager | Varchar |  |
| Contact | Varchar | Unique |
| Password | Varchar |  |

**Patient table**

| Patient\_id | Number | Primary key |
| --- | --- | --- |
| Name | Varchar |  |
| Bloodgroup | Varchar |  |
| City | Varchar |  |
| Contact\_number | Number | Unique |
| Status | Number |  |
| Hospital\_ID | Number |  |

**Stock table**

| S\_ID | Number | Primary Key |
| --- | --- | --- |
| Blood\_bank\_ID | Number |  |
| A\_POS | Number |  |
| A\_NEG | Number |  |
| B\_POS | Number |  |
| B\_NEG | Number |  |
| O\_POS | Number |  |
| O\_NEG | Number |  |
| AB\_POS | Number |  |
| AB\_NEG | Number |  |

**Donor table**

| Donor\_ID | Number | Primary key |
| --- | --- | --- |
| Name | Varchar |  |
| Blood\_group | Varchar |  |
| City | Varchar |  |
| Contact | Number | Unique |
| Blood\_Bank\_ID | Number | Foreign key |
| Age | Number |  |
| last\_date\_donated | Date |  |

**5. Normalisation**

**Hospital table**

| Hospital\_id | Number | Primary key |
| --- | --- | --- |
| Name | Varchar |  |
| Address | Varchar | Unique |
| Contact no | Varchar | Unique |
| Password | Varchar | Unique |

1NF:

All the attributes in the hospital relation have atomic values which can be uniquely identified using one prime attribute. Therefore, it is in 1st Normalized Form.

2NF:

Since, the relation already satisfies 1st Normal Form and does not have any partial

dependencies as everything only depends on the prime attribute.

3NF:

Since, the relation already satisfies 1st Normal Form and 2nd Normal Form and does not have any transitive dependencies i.e., all attribute only depends on prime attribute and not on any other attribute, hence it is in 3rd Normal Form.

**Donor table**

| Donor\_ID | Number | Primary key |
| --- | --- | --- |
| Name | Varchar |  |
| Blood\_group | Varchar |  |
| City | Varchar |  |
| Contact | Number | Unique |
| Blood\_Bank\_ID | Number | Foreign key |
| Age | Number |  |
| last\_date\_donated | Date |  |

1NF:

All the attributes in the donor relation, have atomic values which can be uniquely identified using one prime attribute. Therefore, it is in 1st Normalized Form.

2NF:

Since the relation already satisfies 1st Normal Form and does not have any partial

dependencies as everything only depends on the prime attribute.

3NF:

Since, the relation already satisfies 1st Normal Form and 2nd Normal Form and does not have any transitive dependencies i.e., all attribute only depends on prime attribute and not on any other attribute, hence it is in 3rd Normal Form.

**Stock table**

| S\_ID | Number | Primary Key |
| --- | --- | --- |
| Blood\_bank\_ID | Number |  |
| A\_POS | Number |  |
| A\_NEG | Number |  |
| B\_POS | Number |  |
| B\_NEG | Number |  |
| O\_POS | Number |  |
| O\_NEG | Number |  |
| AB\_POS | Number |  |
| AB\_NEG | Number |  |

1NF:

All the attributes in the stock relation have atomic values which can be uniquely identified using one prime attribute. Therefore, it is in 1st Normalized Form.

2NF:

Since, the relation already satisfies 1st Normal Form and does not have any partial

dependencies as everything only depends on the prime attribute.

3NF:

Since, the relation already satisfies 1st Normal Form and 2nd Normal Form and does not have any transitive dependencies i.e., all attribute only depends on prime attribute and not on any other attribute, hence it is in 3rd Normal Form.

**Patient table**

| Patient\_id | Number | Primary key |
| --- | --- | --- |
| Name | Varchar |  |
| Bloodgroup | Varchar |  |
| City | Varchar |  |
| Contact\_number | Number | Unique |
| Status | Number |  |
| Hospital\_ID | Number |  |

1NF:

All the attributes in the patient relation have atomic values which can be uniquely identified using one prime attribute. Therefore, it is in 1st Normalized Form

2NF:

Since the relation already satisfies 1st Normal Form and does not have any partial

dependencies as everything only depends on the prime attribute.

3NF:

Since the relation already satisfies 1st Normal Form and 2nd Normal Form and does not have any transitive dependencies i.e. all attribute only depends on prime attribute and not on any other attribute, hence it is in 3rd Normal Form.

**Blood bank table**

| Blood\_Bank\_ID | Number | Primary key |
| --- | --- | --- |
| Name | Varchar |  |
| Address | Varchar | Unique |
| Manager | Varchar |  |
| Contact | Varchar | Unique |
| Password | Varchar |  |

1NF:

All the attributes in the blood bank relation have atomic values which can be uniquely identified using one prime attribute. Therefore, it is in 1st Normalized Form.

2NF:

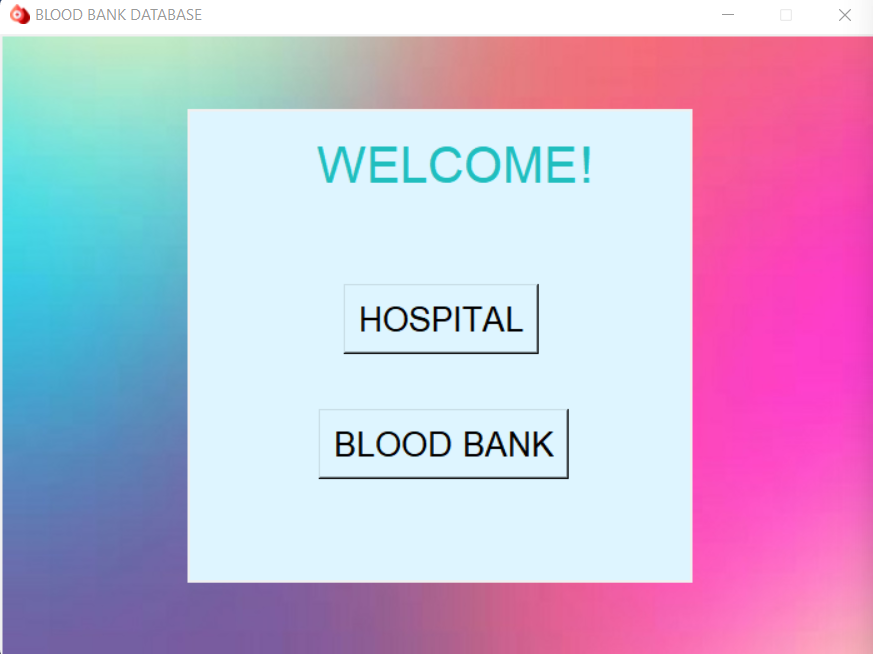
Since, the relation already satisfies 1st Normal Form and does not have any partial

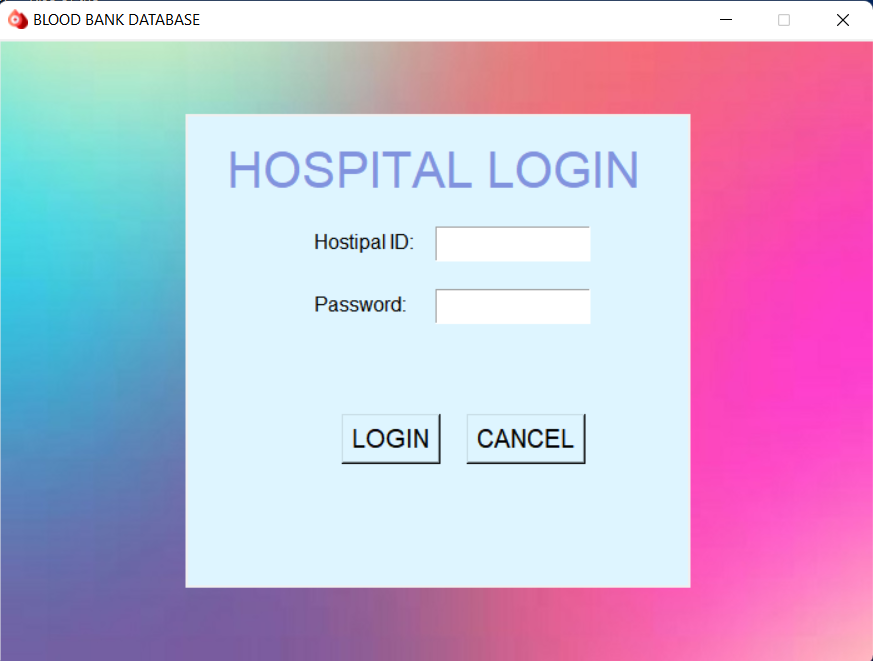
dependencies as everything only depends on the prime attribute.

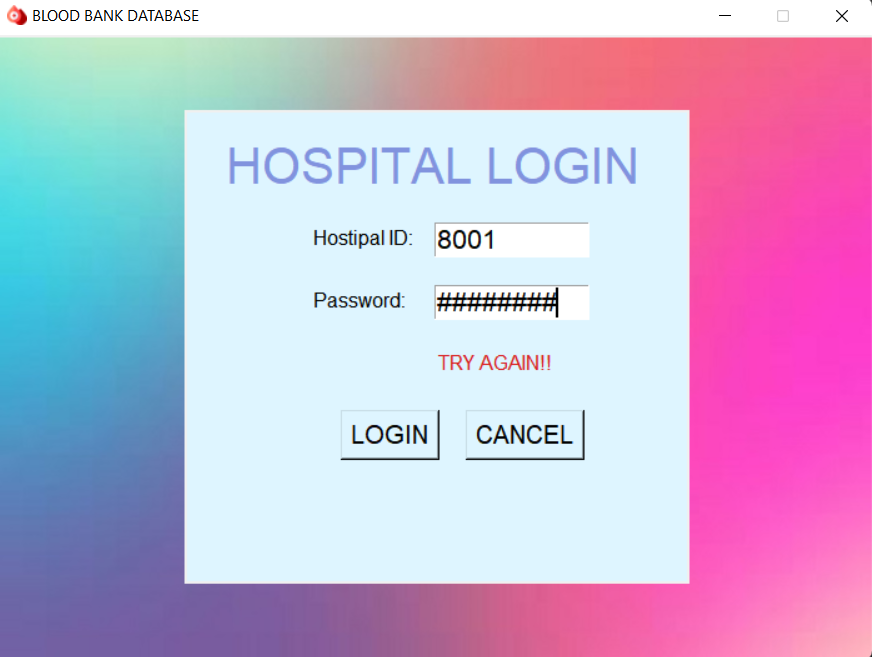
3NF:

Since the relation already satisfies 1st Normal Form and 2nd Normal Form and does not have any transitive dependencies i.e. all attribute only depends on prime attribute and not on any other attribute, hence it is in 3rd Normal Form.

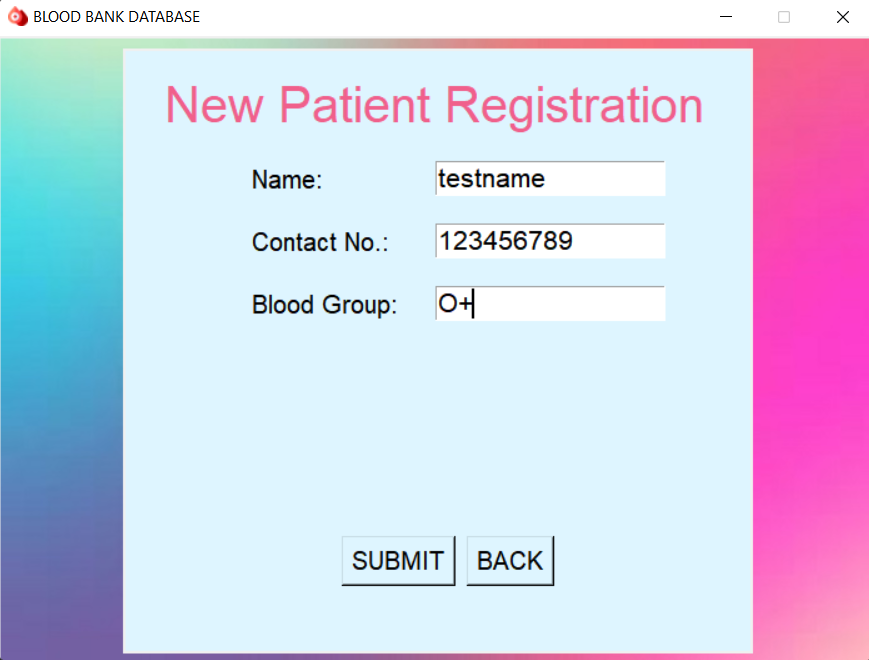
**6. Results**

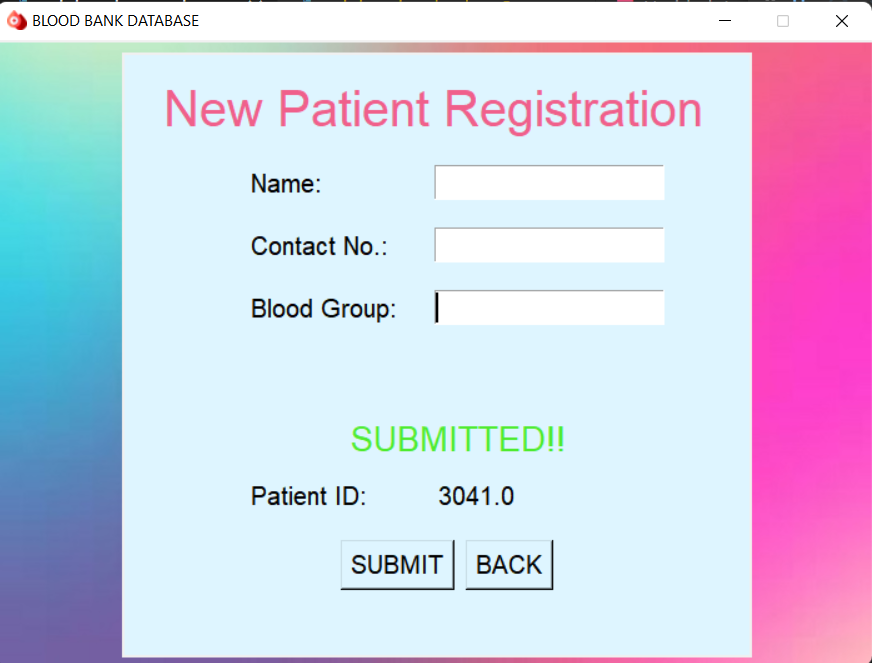
****

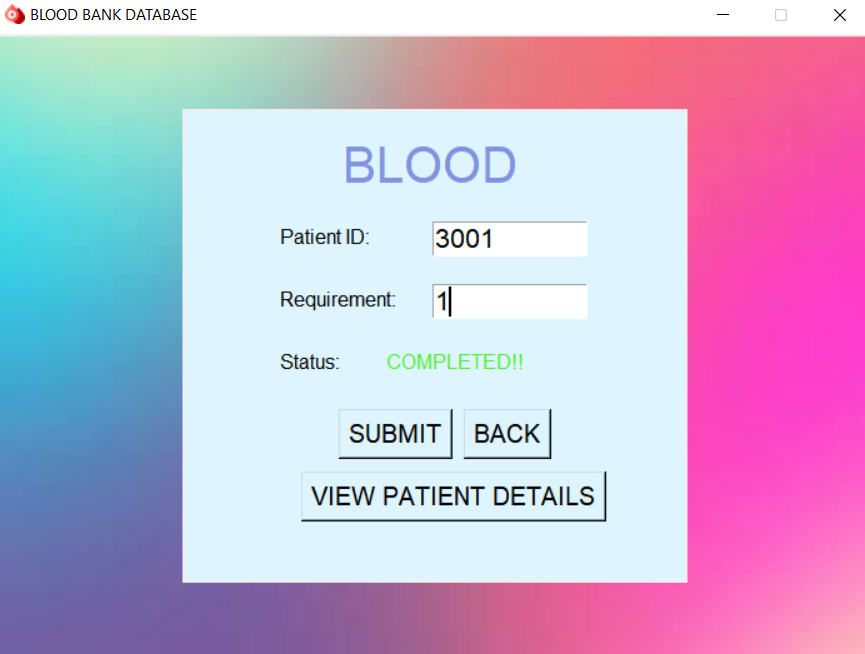
****

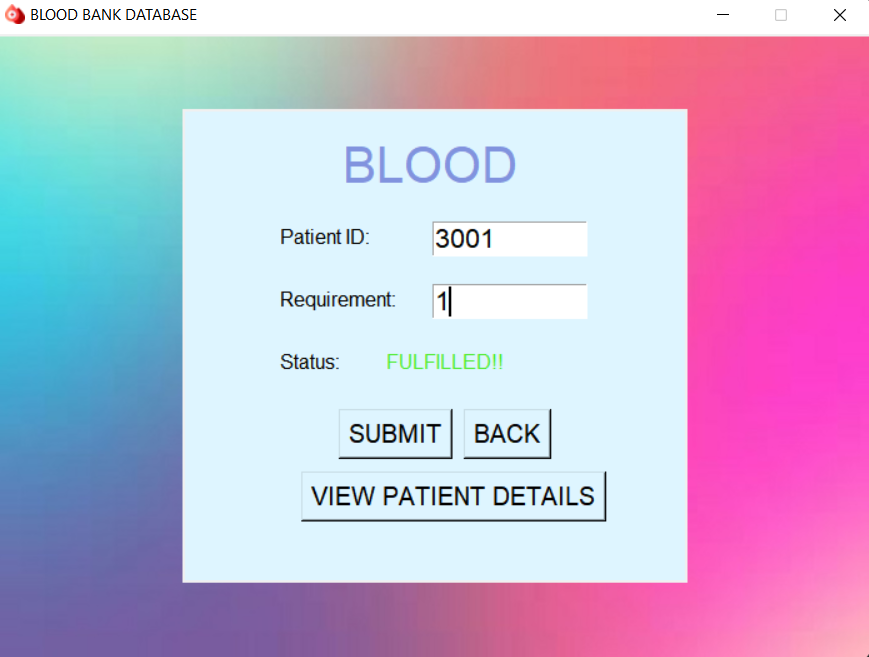
****

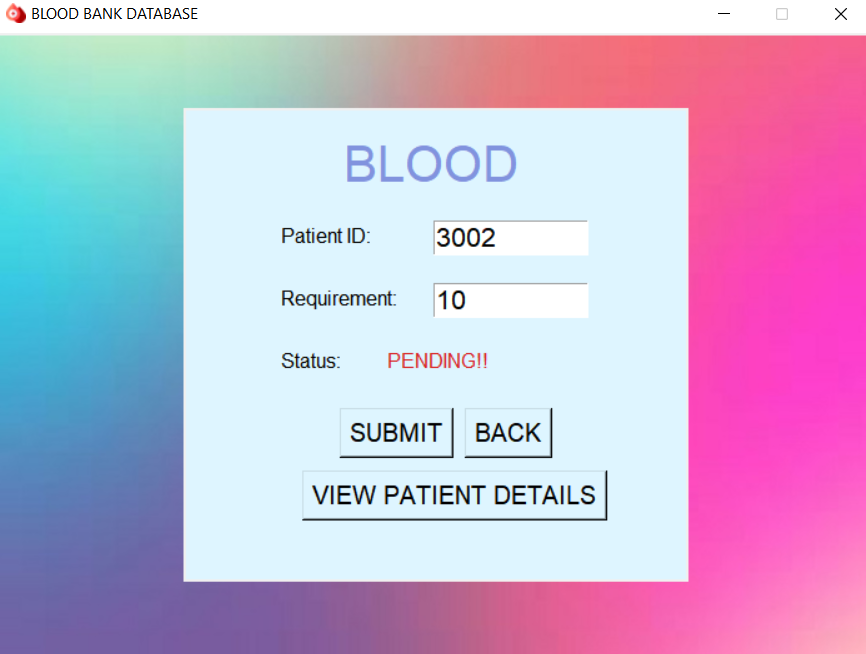
****

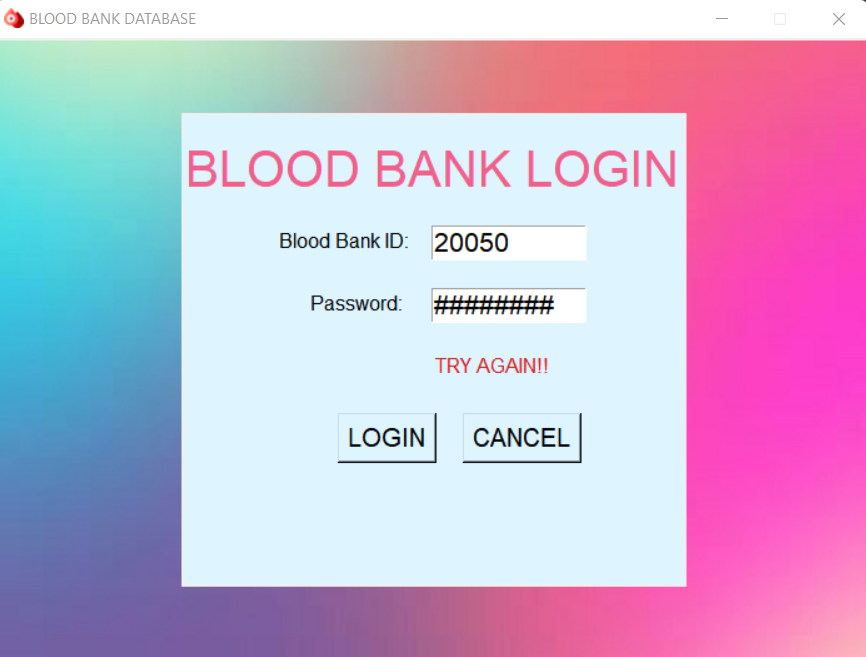


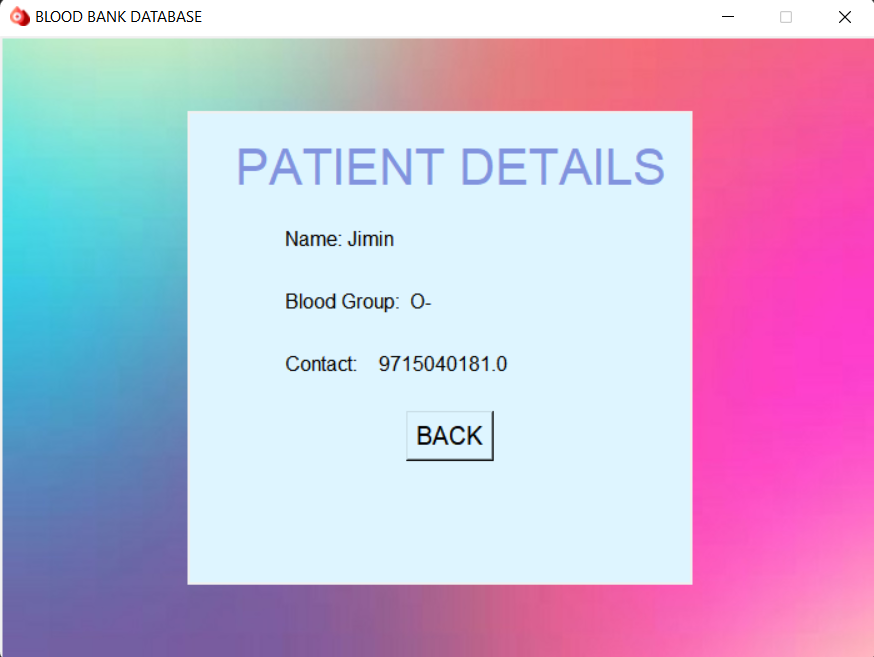


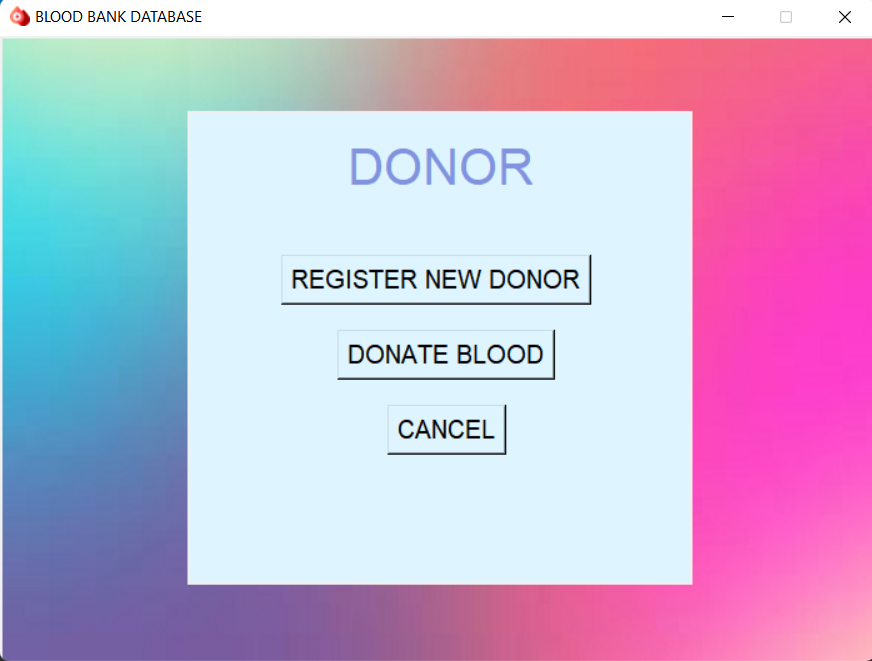


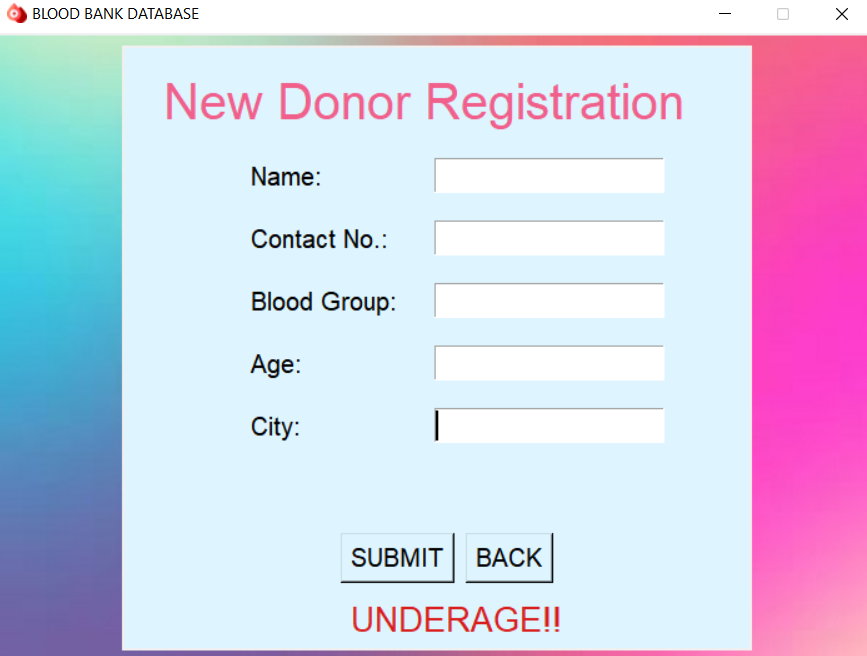


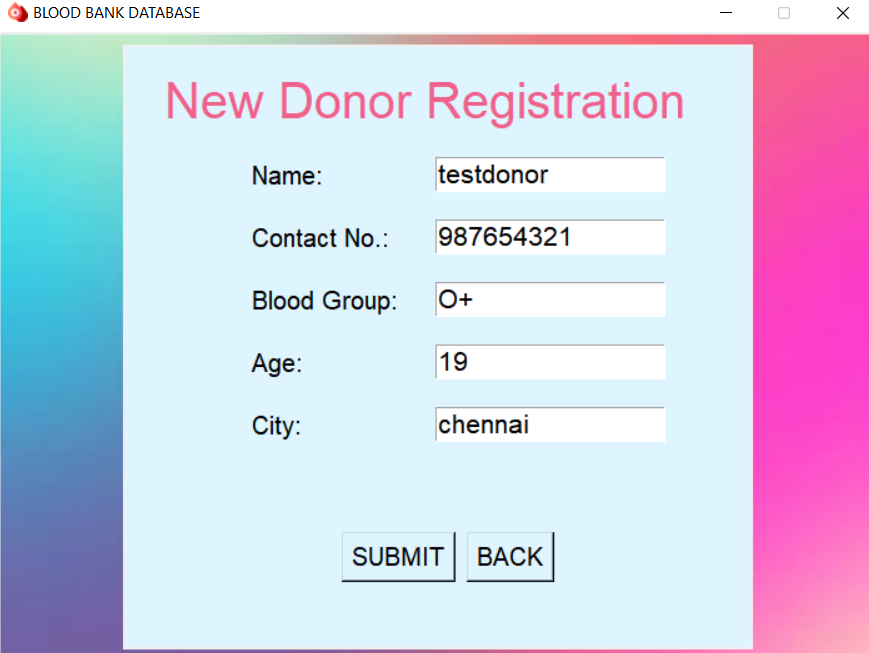


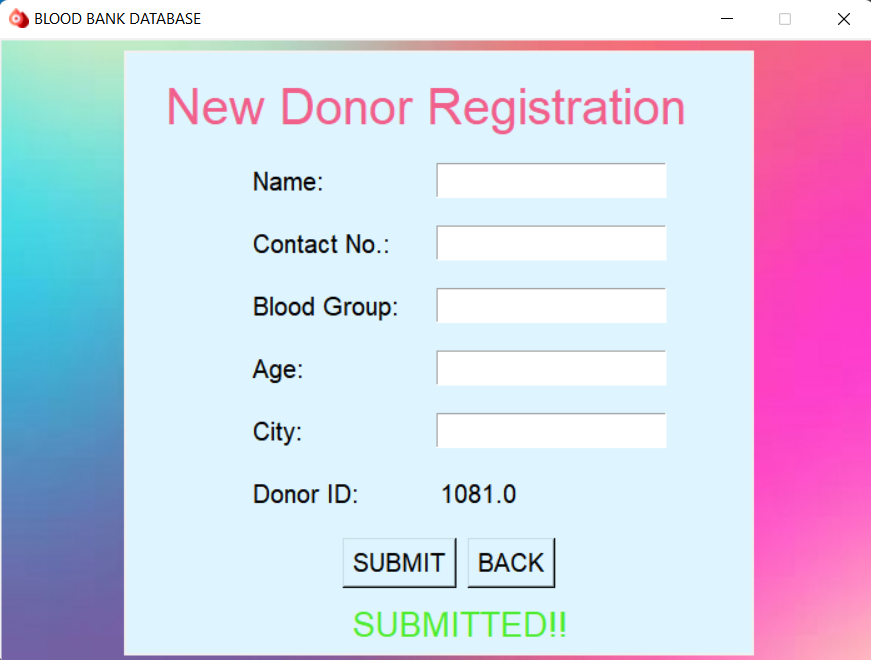


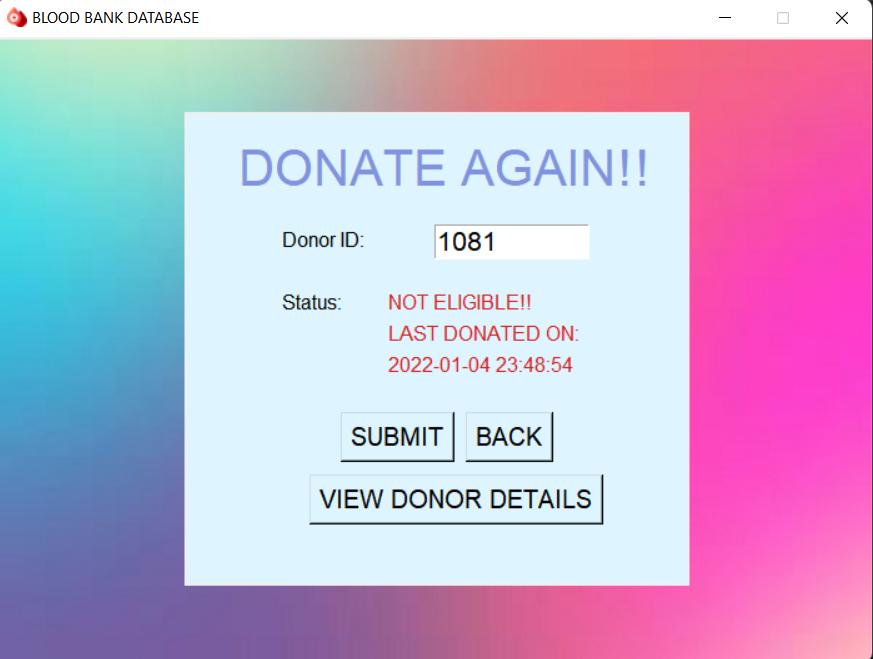


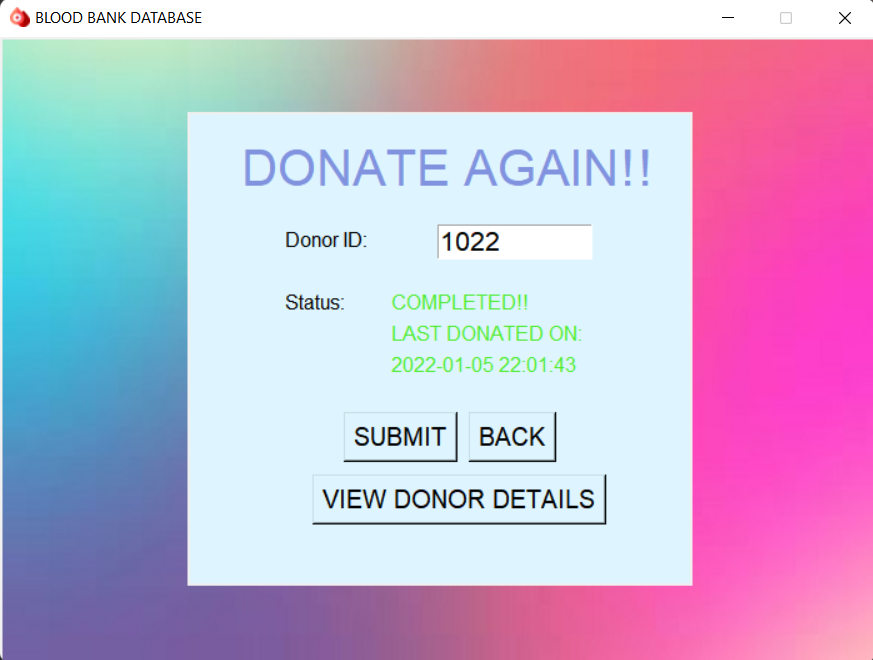


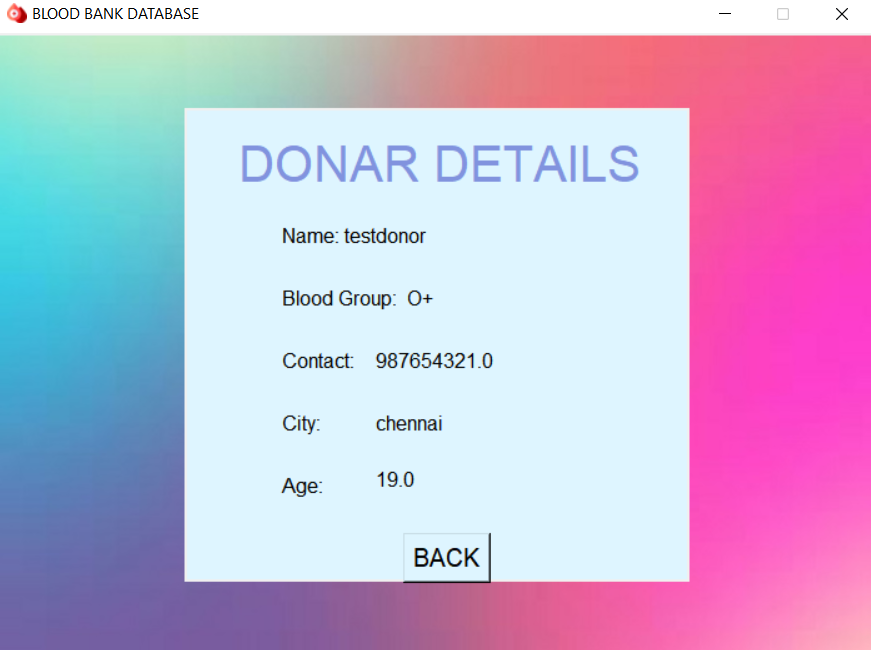








****

****

**8. APPENDIX: Report on Python GUI Tkinter.**

Image

Tkinter is an inbuilt Python module used to create simple GUI apps. It is the most commonly used module for GUI apps in the Python.

The tkinter package (“Tk interface”) is the standard Python interface to the Tcl/Tk GUI toolkit. Both Tk and tkinter are available on most Unix platforms, including macOS, as well as on Windows systems.

Running python –m tkinter from the command line should open a window demonstrating a simple Tk interface, letting you know that tkinter is properly installed on your system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.

Tkinter supports a range of Tcl/Tk versions, built either with or without thread support. The official Python binary release bundles Tcl/Tk 8.6 threaded. See the source code for  
the \_tkinter module for more information about supported versions.

Tkinter is not a thin wrapper, but adds a fair amount of its own logic to make the experience more pythonic. This documentation will concentrate on these additions and changes, and refer to the official Tcl/Tk documentation for details that are unchanged.

Python has a lot of GUI frameworks, but Tkinter is the only framework that’s built into the Python standard library. Tkinter has several strengths. It’s cross-platform, so the same code works on Windows, macOS, and Linux. Visual elements are rendered using native operating system elements, so applications built with Tkinter look like they belong on the platform where they’re run.

Although Tkinter is considered the de-facto Python GUI framework, it’s not without criticism. One notable criticism is that GUIs built with Tkinter look outdated. If you want a shiny, modern interface, then Tkinter may not be what you’re looking for.

However, Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where a modern sheen is unnecessary, and the top priority is to build something that’s functional and cross-platform quickly.

Anything that happens in a user interface is an event. We say that an event is fired whenever the user does something – for example, clicks on a button or types a keyboard shortcut. Some events could also be triggered by occurrences which are not controlled by the user – for example, a background task might complete, or a network connection might be established or lost.

Our application needs to monitor, or listen for, all the events that we find interesting, and respond to them in some way if they occur. To do this, we usually associate certain functions with particular events. We call a function which performs an action in response to an event- an event handler- we bind handles to events.

**Bibliography:**

https://python-textbok.readthedocs.io/en/1.0/Introduction\_to\_GUI\_Programming.html

https://www.tutorialspoint.com/python/python\_gui\_programming.htm

<https://www.devart.com/odbc/oracle/>

https://stackoverflow.com/