BLOOD BANK MANAGEMENT SYSTEM

## A PROJECT REPORT

*Submitted by*

# CH.S.K. GOWTHAM

# [Reg No: RA2211027010149]

# S.SAI CHARANI

# [Reg No: RA2211027010186]

*Under the Guidance of*

# DR. SUTHANTHIRA DEVI

Assistant Professor, Department of Data Science and Business Systems

*In partial fulfilment of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

# with a specialization in BIG DATA ANALYTICS



**FACULTY OF ENGINEERING AND**

**TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM UNIVERSITY OF SCIENCE AND**

**TECHNOLOGY**

**KATTANKULATHUR**

**MAY 2024**



### **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603 203**

**BONAFIDE CERTIFICATE**

Certified that this B.Tech project report titled “**BLOOD BANK MANAGEMENT SYSTEM**” is the bonafide work of **Mr.C.H.S.K.Gowtham [Reg. No : RA2211027010149] and Ms. S.SaiCharani [Reg. No.RA2211027010186]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. Suthanthira Devi

Assistant Professor

Department of Data Science and Business Systems

Dr. Lakshmi M

**HEAD OF THE DEPARTMENT**

Department of Data

Science and Business Systems

**Date:**

**ABSTRACT**

The "Blood Bank Management System " is a user-centric platform designed to optimize the blood donation process. The system begins with a secure recipient login, allowing users to efficiently search for specific blood types within their vicinity. By integrating real-time blood inventory tracking and location-based services, the system provides recipient with instant information on the availability of the desired blood type in nearby areas. This Blood Bank Management System addresses the need for a streamlined, accessible, and community-driven approach to blood donation. By leveraging technology, real-time data, and user-friendly interfaces, the system aims to bridge the gap between donors and recipients, making the blood donation process more efficient, transparent, and responsive to urgent needs.

# PROBLEMSTATEMENT

# In the realm of blood banking, antiquated manual processes persist, impeding operational efficiency and exacerbating challenges across donor registration, inventory management, and donor engagement. The absence of digital infrastructure within blood banks results in disparate data sources, leading to inaccuracies in donor records and cumbersome inventory tracking processes. Moreover, limited avenues for donor communication hinder effective engagement strategies, impacting donor retention and the overall blood supply chain. To navigate these pressing challenges and ensure the uninterrupted availability of life-saving blood products, there is an imperative for a modernized blood bank management system. Such a system would revolutionize donor registration processes, implement robust inventory management capabilities for real-time monitoring, and facilitate seamless communication channels to foster meaningful connections with donors. By leveraging automation and digitalization, this system aims to transform the blood donation landscape, optimizing resource utilization and ensuring a sustainable supply of blood products for those in need

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **Chapter No** | **Chapter Name** | **Page No** |
|  | Problem understanding, Identification of Entity and Relationships, Construction of DB using ER Model for the project | **2-3** |
|  | Design of Relational Schemas, Creation of Database Tables for the project. | **4-6** |
|  | Complex queries based on the concepts of constraints, sets, joins, views, Triggers and Cursors. | **7-9** |
|  | Analyzing the pitfalls, identifying the dependencies, and applying normalizations | **10-12** |
|  | Implementation of concurrency control and recovery mechanisms | **13-20** |
|  | Code for the project | **21-23** |
|  | Result and Discussion (Screen shots of the implementation with front end. | **24-26** |
|  | Attach the Real Time project certificate / Online course certificate | **27** |

**CHAPTER 1**

* 1. **Problem understanding:**

"Blood banks play a crucial role in ensuring a stable supply of safe blood for transfusions in healthcare facilities. However, many blood banks face challenges in efficiently managing their operations, including donor registration, inventory tracking, blood testing, and transfusion management. Existing manual or outdated systems often result in inefficiencies, errors, and delays, compromising the timely availability and safety of blood products. Moreover, ensuring compliance with regulatory standards and maintaining data security pose additional challenges. Therefore, there is a pressing need for a comprehensive and automated Blood Bank Management System (BBMS) that can address these issues effectively, streamline processes, enhance blood safety, and ensure regulatory compliance."

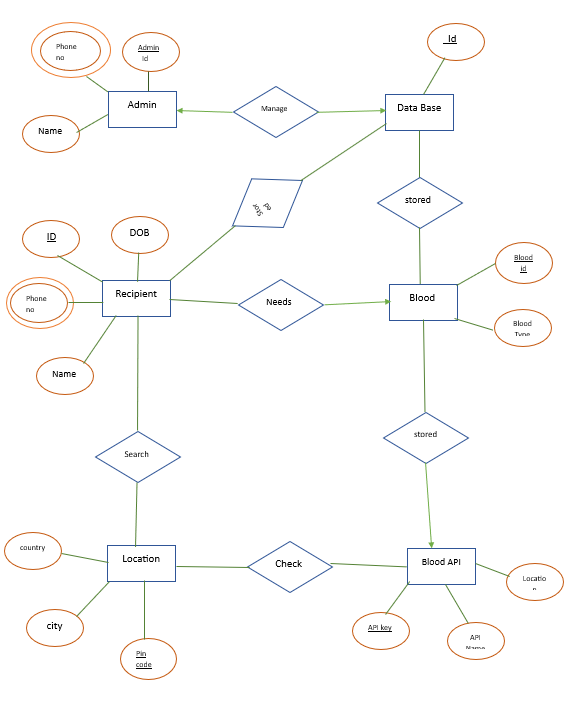
* 1. **Identification of entities and Relationships**

Entities:

* Blood Bank
* Admin
* Recipient
* Blood
* Location
* Blood API

Relationships:

* Admin manages blood bank
* Admin stores data in database
* Blood bank stores blood
* Recipient needs blood from blood bank
* Blood Bank checks blood with Blood API
* Blood API uses location API
  1. **Construction of DB using ER Model for the project**

****

* 1. **ARCHITECTURE DIAGRAM**

A diagram of a bank

Description automatically generated

**CHAPTER 2**

**2.1) Design of Relational Schemas:**

**A diagram of a database

Description automatically generated**

**2.2) Schemas**

* Admin(Admin\_id,First Name,Last Name,Address,Phone\_no,Blood\_id)
* Database(Database\_id,Database Type)
* Receipent(Id,First Name,Last Name,Phn\_no,DOB,Age,Blood\_id)
* Blood(Blood\_id,Blood Type,Blood\_units)
* Location(City,Country,Pincode)
* Donor(Donor\_id,Name,Phn\_no,DOB)

**2.3) Creation of Database Tables for the project.**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**CHAPTER 3**

**3.1) Complex queries based on the concepts of constraints:**Select blood\_type from blood where blood\_units=(Select max(blood\_units) from blood);  
Select \*from blood where blood\_units=(select max(blood\_units) from blood);

**A screenshot of a computer

Description automatically generated**

**3.2) Sets**

**A screenshot of a computer

Description automatically generated**

**3.3) Joins:**select recipient.name,blood.blood\_type from recipient inner join on blood.blood\_id = recipient.blood\_id;  
select recipient.name,blood.blood\_type from recipient left join on blood.blood\_id=recipient.blood\_id;

**A screenshot of a computer

Description automatically generated**

**3.4) Views**

Create view blood\_details as select recipient.name,blood\_id,blood\_type,location.city from recipient,blood,location where recipient.blood\_id=blood.blood\_id;

**A screenshot of a computer

Description automatically generated**

**3.5) Triggers and Cursors**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**CHAPTER 4**

**4.1) Unnormalized Table**

**A screenshot of a computer

Description automatically generated**

**1NF**

**A screenshot of a computer

Description automatically generated**

**2NF**

**A screenshot of a computer

Description automatically generated**

**3NF**

**A screenshot of a computer

Description automatically generated**

**BCNF**

**A screenshot of a computer

Description automatically generated**

**CHAPTER 5**

**TRANSACTION CONTROL & CONCURRENCY CONTROL**

**Satisfying ACID Properties:**

**ATOMICITY:**

**Scenario:**

A new recipient is being added to the recipient table, but while inserting the recipient's details, the system fails after the recipient's details are inserted, but before updating the blood bank's available blood units.

**Solution:**

Use a transaction to ensure atomicity. If any part of the transaction fails, all changes made in that transaction should be rolled back

**MySQL Query:**

# A black screen with white text Description automatically generated

**Explanation:**

* START TRANSACTION; begins a new transaction.
* INSERT INTO recipient... inserts a new recipient's details.
* UPDATE blood... decreases the available blood units.
* COMMIT; commits the transaction. If no errors occur, changes made within the transaction are saved to the database. If any part of the transaction fails, the ROLLBACK; command would undo any changes made within the transaction.

**CONSISTENCY:  
Scenario:**

A donor donates blood, and the system needs to ensure that after the donation, the total number of blood units is updated and consistent.

**Solution:**

Use transactions to ensure that the total number of blood units is consistent after a donation.

**MySQL Query:**

A black screen with a black background

Description automatically generated

**Explanation:**

* START TRANSACTION; begins a new transaction.
* UPDATE blood... increases the available blood units.
* COMMIT; commits the transaction. If no errors occur, changes made within the transaction are saved to the database. If any part of the transaction fails, the ROLLBACK; command would undo any changes made within the transaction.

**ISOLATION:  
Scenario:**

Two administrators simultaneously try to update the same recipient's details

**Solution:**

Use locking mechanisms to prevent concurrent access to the same data.

**MySQL Query:**

A black background with white text

Description automatically generated

**Explanation:**

* START TRANSACTION; begins a new transaction.
* The first UPDATE recipient... query is executed by the first admin.
* DO SLEEP(10); is a MySQL function that pauses execution for 10 seconds.
* The second UPDATE recipient... query is executed by the second admin.
* COMMIT; commits the transaction. MySQL automatically handles locking and ensures that the changes made by one transaction are isolated from the changes made by other transactions.

**DURABILITY:**

**Scenario:**

After a successful blood donation, the system crashes.

**Solution:**

Ensure that the changes made to the database are permanently saved, even if the system crashes.

**Query:**

A black screen with a black background

Description automatically generated

**Explanation:**

* START TRANSACTION; begins a new transaction.
* UPDATE blood... increases the available blood units.
* Even if the system crashes after the UPDATE statement, the changes made within the transaction will be durable. MySQL ensures durability by saving transaction logs and ensuring that committed transactions are permanently stored in the database.

**For database tables:**

**‘Admin’ Table:**

**Scenario:**

Concurrency Control and Transaction Control for the admin table

**Solution:**

To demonstrate concurrency control and transaction control for the admin table, we will perform a simple update operation within a transaction. This operation will update the phone number of an admin.

**Query:**

A computer screen with white text

Description automatically generated

**Explanation:**

* We begin a transaction using START TRANSACTION.
* We then update the phone number of the admin with admin\_id 1.
* The SELECT statement displays the updated admin table showing the changes.
* Finally, we commit the transaction using COMMIT

**‘Blood’ Table:**

**Scenario:**

Concurrency Control and Transaction Control for the blood table

**Solution:**

To demonstrate concurrency control and transaction control for the blood table, we will perform a simple update operation within a transaction. This operation will update the available units of blood.

**Query:**

A screenshot of a computer program

Description automatically generated

**Explanation:**

* We begin a transaction using START TRANSACTION.
* We then update the available units of blood for type 'A+'.
* The SELECT statement displays the updated blood table showing the changes.
* Finally, we commit the transaction using COMMIT.

**‘Recipient’ Table:**

**Scenario:**

Concurrency Control and Transaction Control for the recipient table

**Solution:**

To demonstrate concurrency control and transaction control for the recipient table, we will perform a simple delete operation within a transaction. This operation will delete a recipient from the table.

**Query:**

A screenshot of a computer

Description automatically generated

**Explanation:**

* We begin a transaction using START TRANSACTION.
* We then delete a recipient with recipient\_id 1.
* The SELECT statement displays the updated recipient table showing the changes.
* Finally, we commit the transaction using COMMIT

**‘Location’ Table:**

**Scenario:**

Concurrency Control and Transaction Control for the Location table

**Solution:**

To demonstrate concurrency control and transaction control for the Location table, we will perform a simple insert operation within a transaction. This operation will insert a new location into the table.

**Query:**

A screenshot of a computer

Description automatically generated

**Explanation:**

* We begin a transaction using START TRANSACTION.
* We then insert a new location into the Location table.
* The SELECT statement displays the updated Location table showing the changes.
* Finally, we commit the transaction using COMMIT.

**‘Donor’ Table:**

**Scenario:**

Concurrency Control and Transaction Control for the donor table

**Solution:**

To demonstrate concurrency control and transaction control for the donor table, we will perform a simple update operation within a transaction. This operation will update the address of a donor**.**

**Query:**

A screenshot of a computer

Description automatically generated

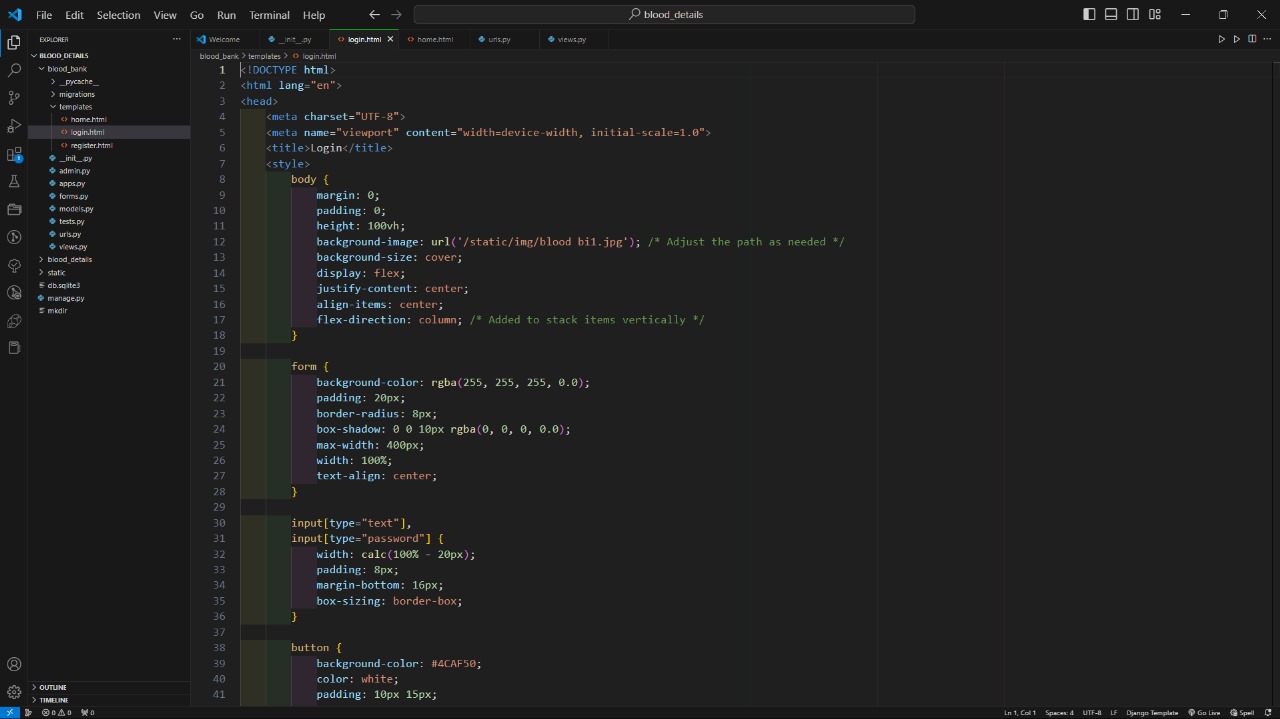
**Explanation:**

* We begin a transaction using START TRANSACTION.
* We then update the address of the donor with donor\_id 1.
* The SELECT statement displays the updated donor table showing the changes.
* Finally, we commit the transaction using COMMIT.

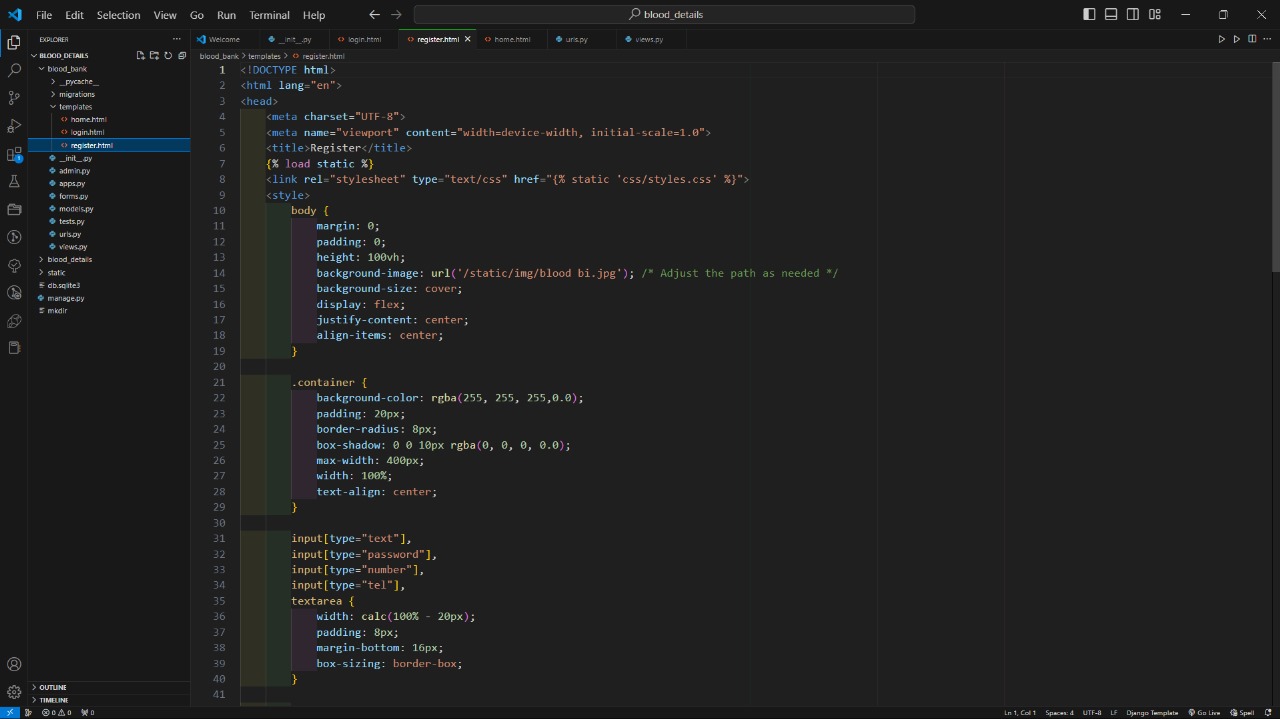
**CHAPTER 6**

**CODE SNIPPETS**

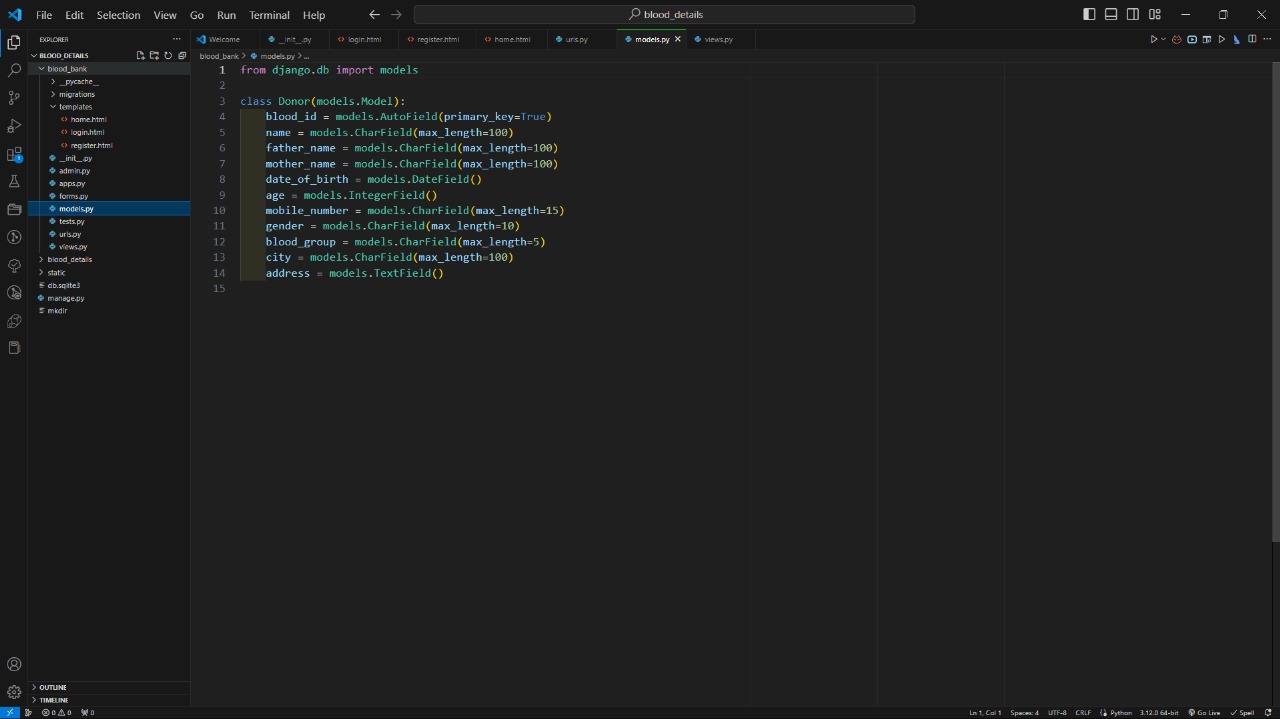
**Login.Html**

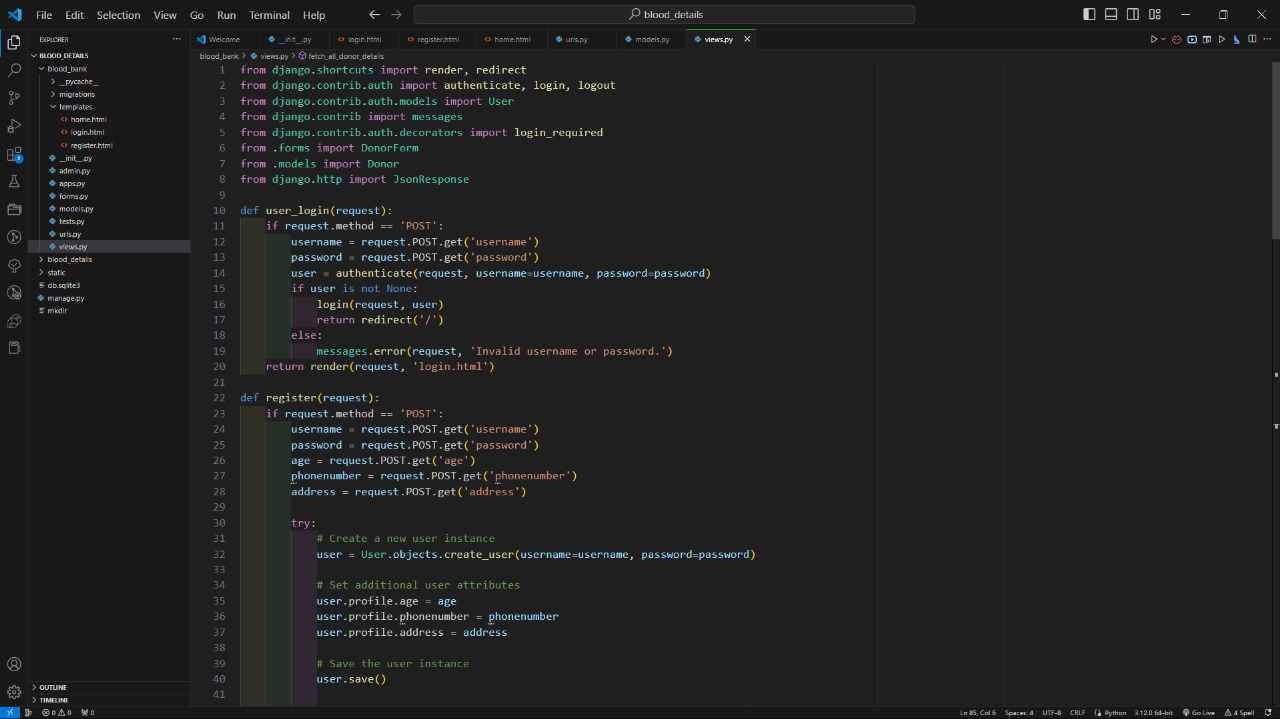


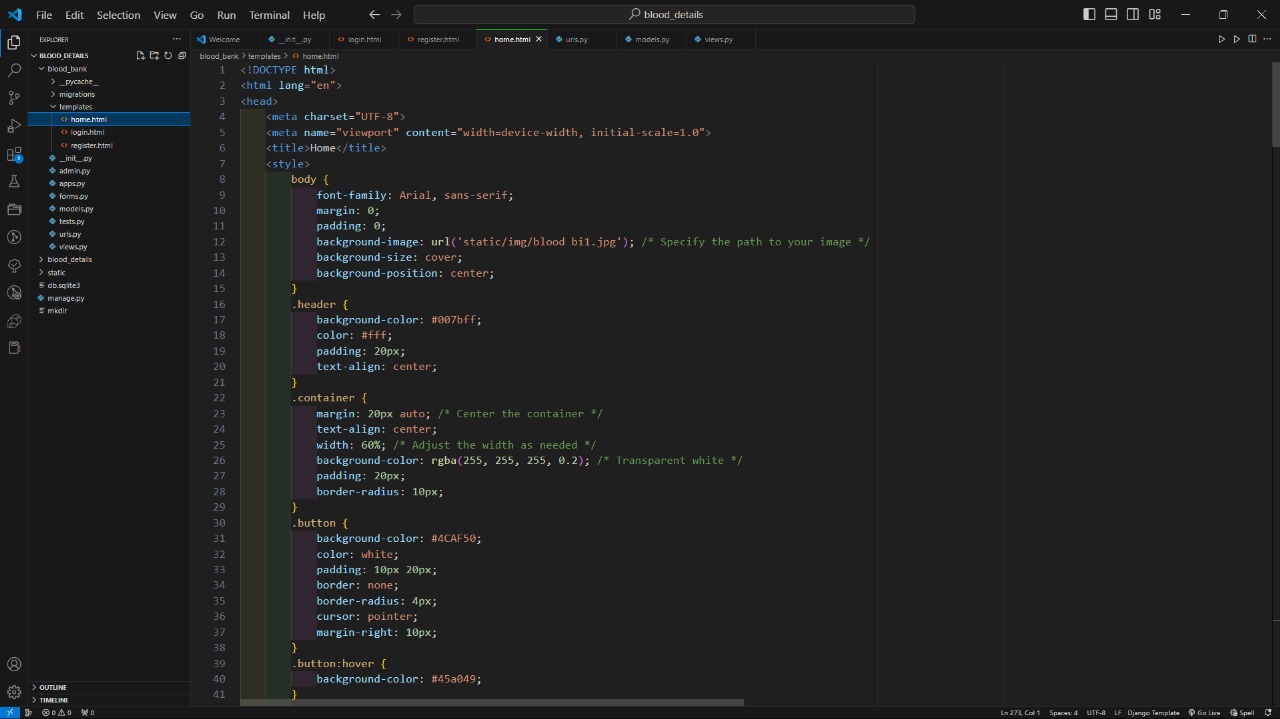
**Register.Html**



**Urls.py**

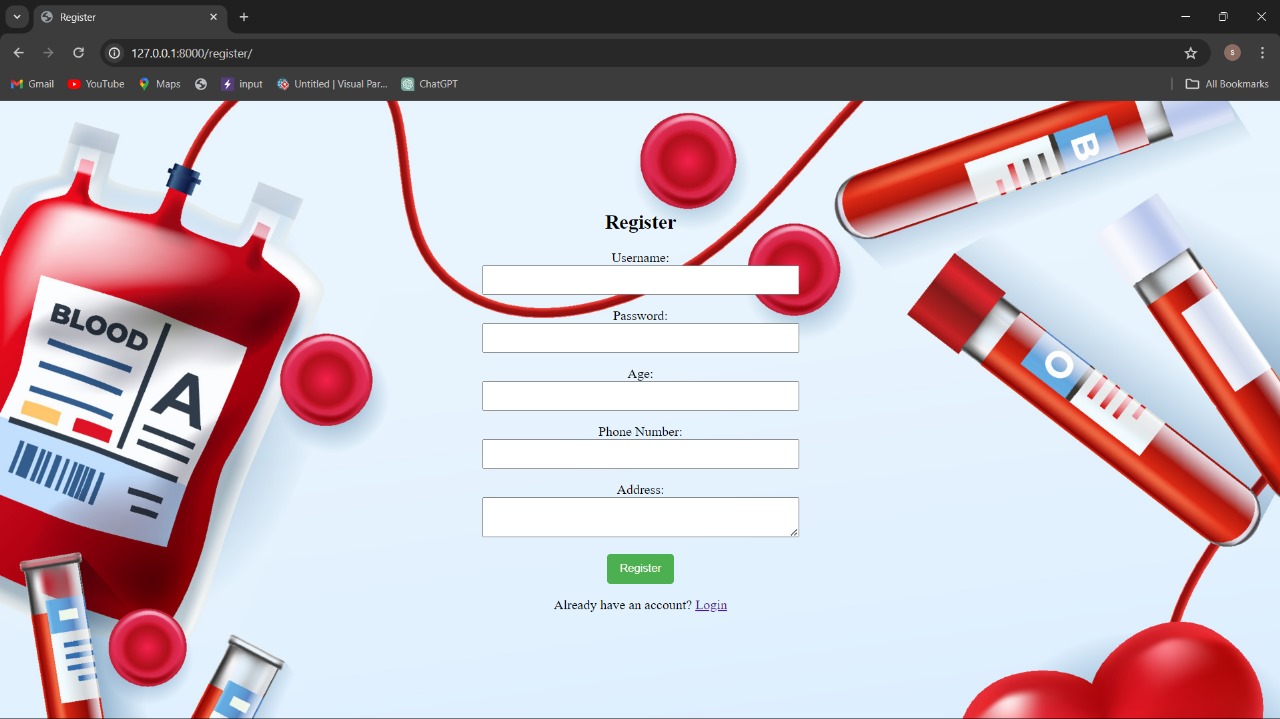
**Models.py**

**Views.py**

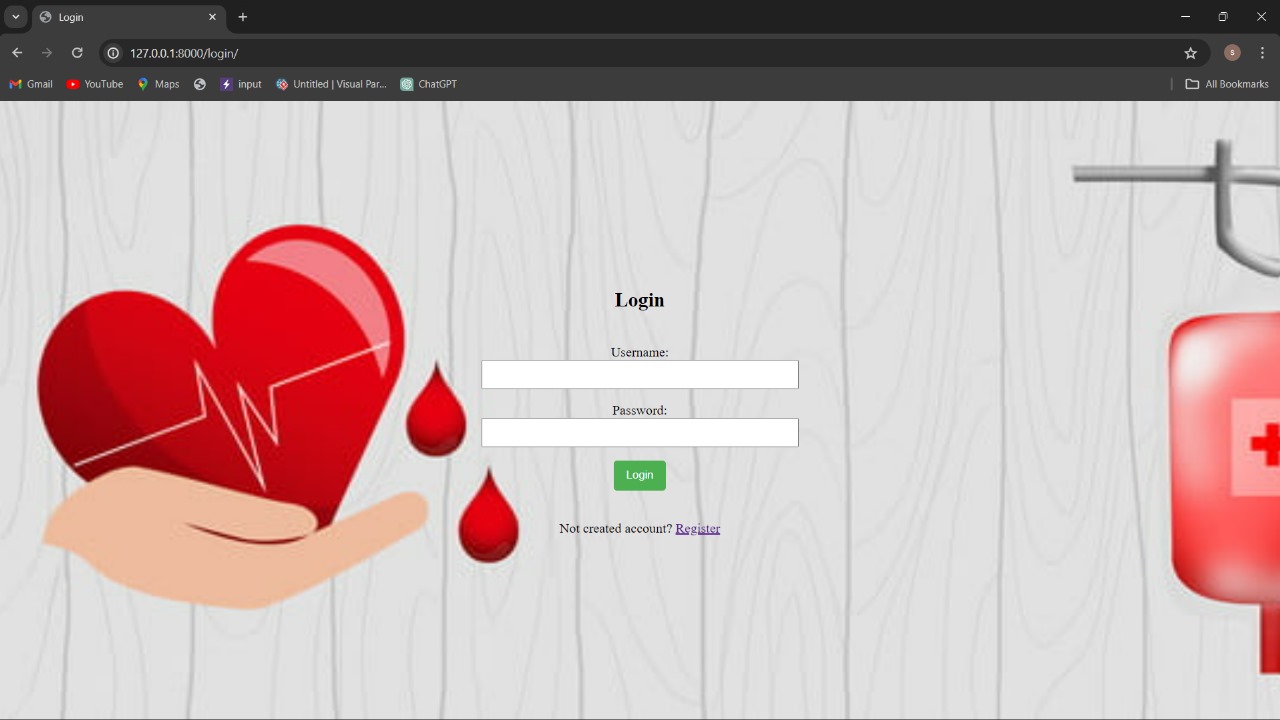
**Home.html**

**CHAPTER 7**

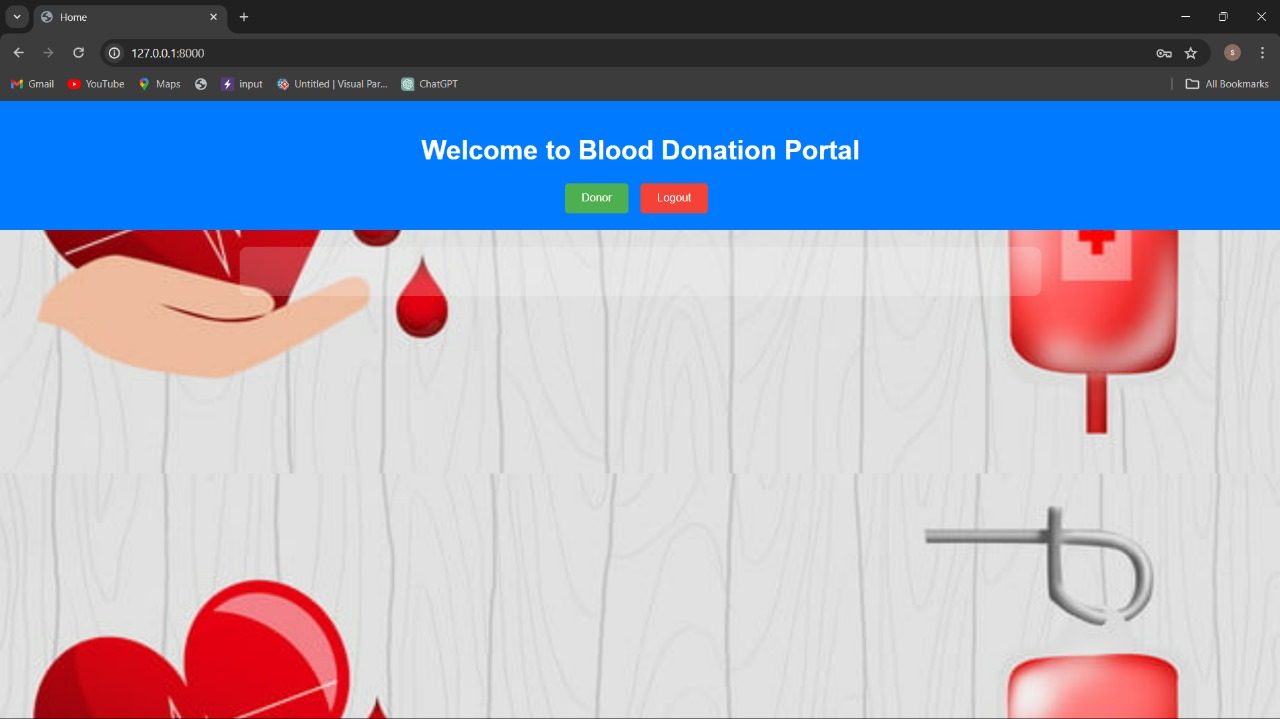
**RESULTS AND DISCUSSION**

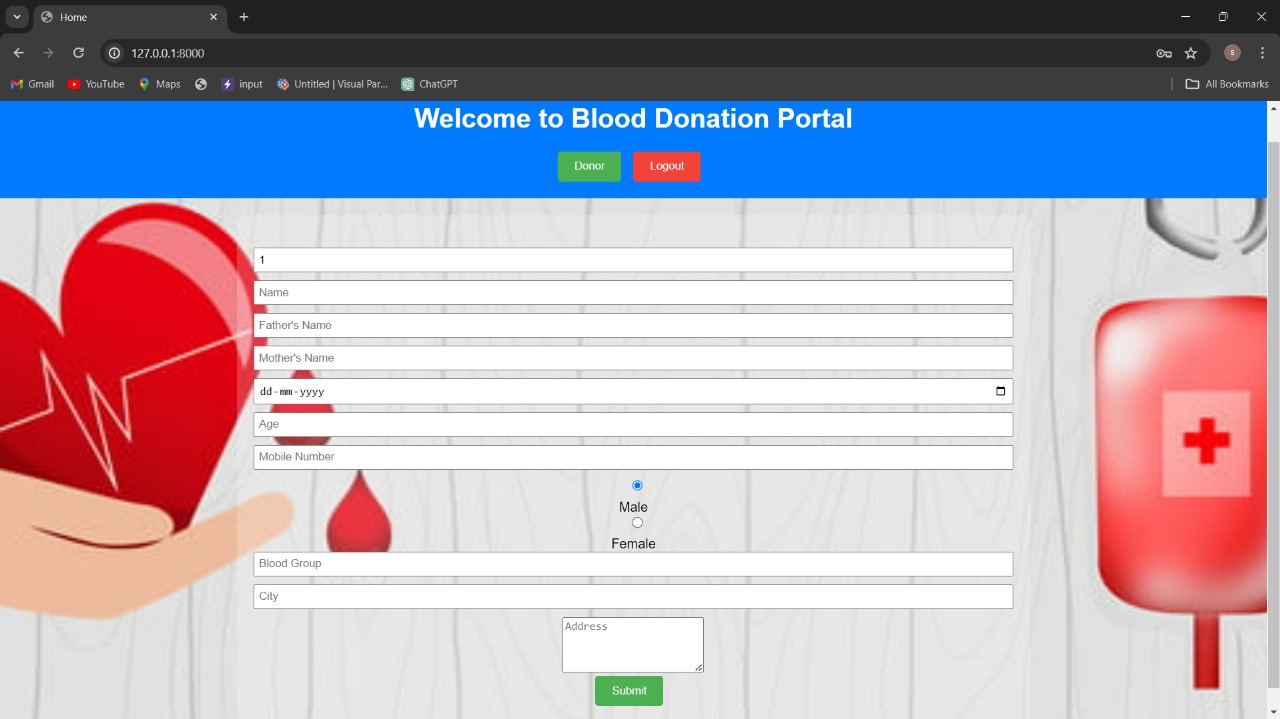
**Register Page**

**Login Page**

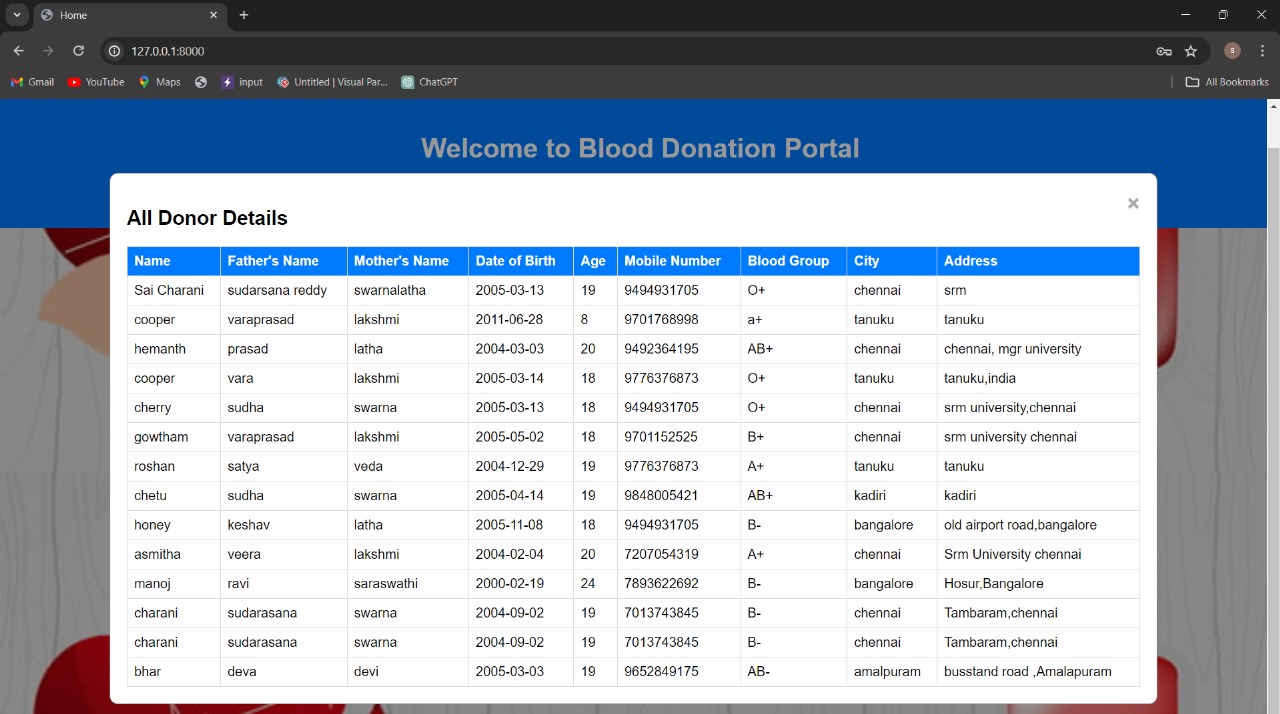


**Home Page**

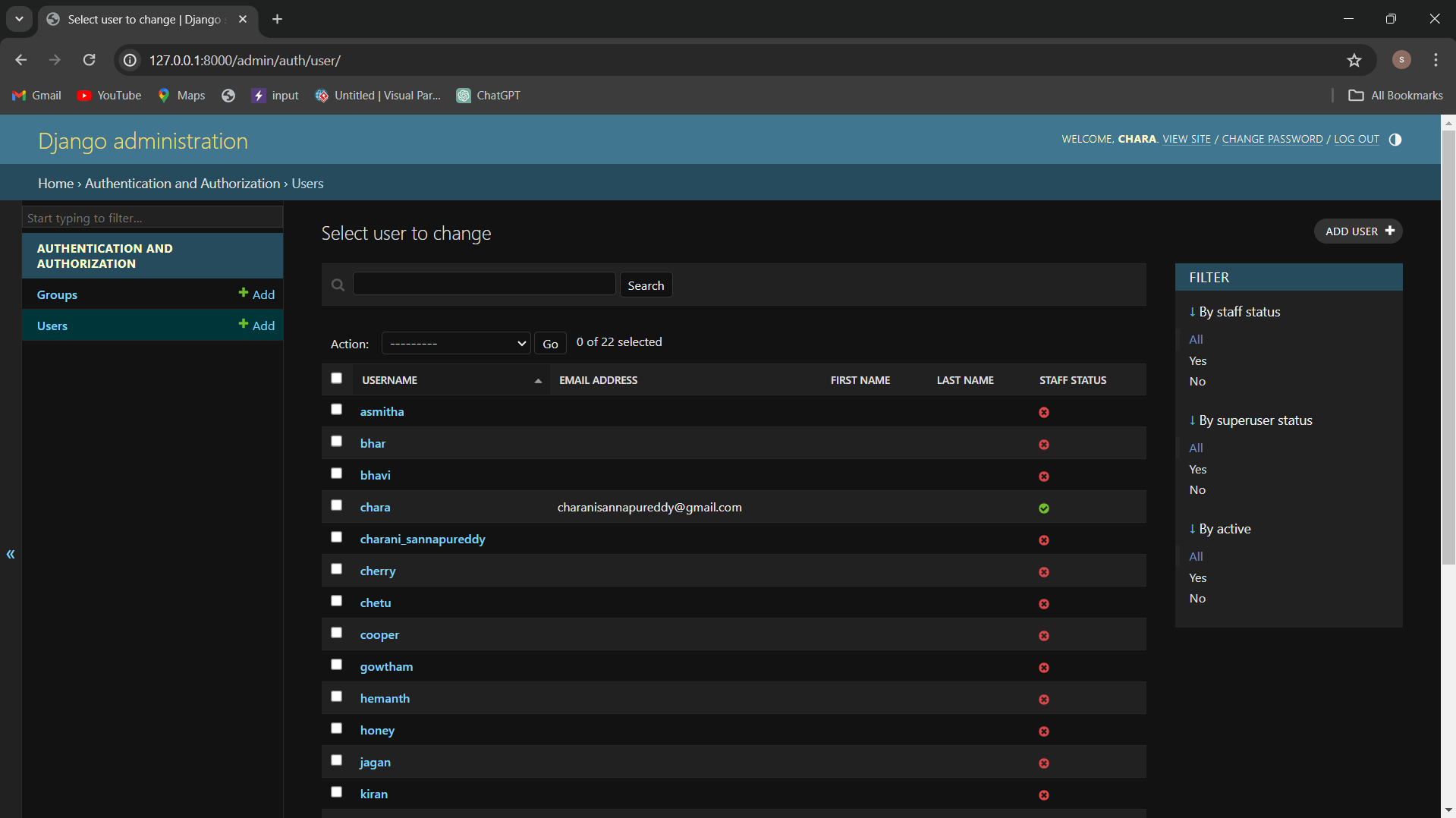


**New Donor Details**

**Add New Donor Details**



**Admin Page**

****

**CHAPTER 8**

**ONLINE CERTIFICATE**

**CH.S.K.Gowtham(RA2211027010149)**



**S.Sai Charani(RA2211027010186)**

