

# **SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES, CHENNAI – 602105**

**CAPSTONE PROJECT REPORT**

# **TITLE**

**Online Blood Banking System**

***Submitted to***

# **SAVEETHA SCHOOL OF ENGINEERING**

***By***

## K Madhusudhan (19210035)

***Guided by***

## Dr. J. Chenni Kumaran

### 

**1. Problem Statement**

Blood banks play a critical role in healthcare by providing a secure and reliable supply of blood and blood products. However, the traditional methods of managing blood banks are often inefficient and error-prone, leading to challenges in inventory management, donor tracking, and ensuring timely availability of blood. The aim of this project is to develop an online blood banking system to streamline these processes, making it easier for donors to register and schedule donations, for recipients to request blood, and for administrators to manage inventory and ensure compliance with health regulations.

**2. Proposed Design Work**

**Identifying Key Components**

1. **User Management**: Module for handling donor and recipient registrations, authentication, and profile management.
2. **Blood Inventory Management**: Tracking blood units, their types, and current stock levels.
3. **Donation Scheduling**: Allowing donors to book appointments for donations.
4. **Blood Request Handling**: Facilitating requests for blood from hospitals and individuals.
5. **Notification System**: Sending alerts for low stock, upcoming donation drives, and urgent blood requests.

**Functionality**

* **User Registration and Authentication**: Secure login for donors, recipients, and administrators.
* **Inventory Management**: Real-time tracking of blood units by type and expiration date.
* **Appointment Scheduling**: Donors can book and manage their donation appointments.
* **Request Processing**: Hospitals can place requests and track fulfillment status.
* **Notifications**: Automated alerts for donors about upcoming appointments and for administrators about low inventory levels.

**Architectural Design**

* **Client-Server Architecture**: Web-based front-end communicating with a back-end server.
* **Database Management System**: Centralized database for storing user information, blood inventory, and transaction records.
* **API Integration**: APIs for communication between different modules and external systems, such as hospital management systems.
* **Security Framework**: Implementation of data encryption, user authentication, and secure communication protocols.

**3. GUI Design**

**Layout**

* **Home Page**: Overview of services, quick links to register as a donor or request blood.
* **Donor Dashboard**: Profile management, donation history, and appointment scheduling.
* **Recipient Dashboard**: Blood request forms, status tracking, and history of received blood.
* **Admin Panel**: Inventory management, user management, and system settings.

**User-Friendly**

* **Intuitive Navigation**: Clear and straightforward menu structure.
* **Responsive Design**: Ensuring usability across various devices and screen sizes.
* **Accessibility Features**: Support for screen readers, high contrast modes, and keyboard navigation.

**Color Selection**

* **Primary Colors**: Red and white, symbolizing the connection to blood and healthcare.
* **Secondary Colors**: Soft blues and grays for contrast and readability.
* **Consistency**: Uniform color scheme across all pages to maintain a cohesive look and feel.

**4. Program / Coding**

**Language Selection**

* **Frontend**: HTML, CSS, JavaScript (React.js)
* **Backend**: Node.js with Express framework
* **Database**: MongoDB or MySQL

**Algorithm/Program**

* **User Authentication**: Secure login/logout, password hashing.
* **Blood Inventory Algorithm**: Automated tracking of blood units, alerting when stocks fall below a threshold.
* **Matching Algorithm**: Efficient matching of blood requests with available inventory.
* **Notification System**: Automated email/SMS notifications.

**Execution**

* **Development Environment**: Setup and configuration of development tools and environments.
* **Code Deployment**: Continuous integration and deployment pipeline for smooth updates.

**5. Implementation**

**Connecting the Components**

* **API Development**: Ensuring all components can communicate effectively.
* **Database Integration**: Connecting the back-end server to the database for CRUD operations.

**Cloud Deployment**

* **Hosting Services**: Deploying the application on a cloud platform such as AWS, Azure, or Google Cloud.
* **Scalability**: Ensuring the system can scale to accommodate high traffic.

**Project Testing**

* **Unit Testing**: Verifying individual components work as intended.
* **Integration Testing**: Ensuring different modules interact correctly.
* **User Acceptance Testing**: Gathering feedback from end-users to refine the system.

**6. Performance Evaluation**

* **Response Time**: Measuring the time taken to process requests.
* **Scalability Testing**: Ensuring the system performs well under increased load.
* **Reliability**: Monitoring uptime and error rates.
  1. Program

# Simulating an online blood banking system

class User:

def \_\_init\_\_(self, user\_id, name, user\_type):

self.user\_id = user\_id

self.name = name

self.user\_type = user\_type # 'donor' or 'recipient'

def \_\_str\_\_(self):

return f'User(ID: {self.user\_id}, Name: {self.name}, Type: {self.user\_type})'

class BloodInventory:

def \_\_init\_\_(self):

self.inventory = {}

def add\_blood(self, blood\_type, quantity):

if blood\_type in self.inventory:

self.inventory[blood\_type] += quantity

else:

self.inventory[blood\_type] = quantity

print(f'Added {quantity} units of {blood\_type} blood. Current stock: {self.inventory[blood\_type]} units')

def request\_blood(self, blood\_type, quantity):

if blood\_type in self.inventory and self.inventory[blood\_type] >= quantity:

self.inventory[blood\_type] -= quantity

print(f'Fulfilled request of {quantity} units of {blood\_type} blood. Remaining stock: {self.inventory[blood\_type]} units')

else:

print(f'Insufficient stock for {blood\_type} blood. Available: {self.inventory.get(blood\_type, 0)} units')

def display inventory(self):

print("Current Blood Inventory:")

for blood\_type, quantity in self.inventory.items():

print(f'{blood\_type}: {quantity} units')

class OnlineBloodBankingSystem:

def \_\_init\_\_(self):

self.users = []

self.inventory = BloodInventory()

def register\_user(self, name, user\_type):

user\_id = len(self.users) + 1

user = User(user\_id, name, user\_type)

self.users.append(user)

print(f'Registered new user: {user}')

def display\_users(self):

print("Registered Users:")

for user in self.users:

print(user)

# Instantiate the system

system = OnlineBloodBankingSystem()

# Register some users

system.register\_user("Alice", "donor")

system.register\_user("Bob", "recipient")

# Display registered users

system.display\_users()

# Add blood to inventory

system.inventory.add\_blood("A+", 5)

system.inventory.add\_blood("O-", 3)

# Display current inventory

system.inventory.display\_inventory()

# Request blood

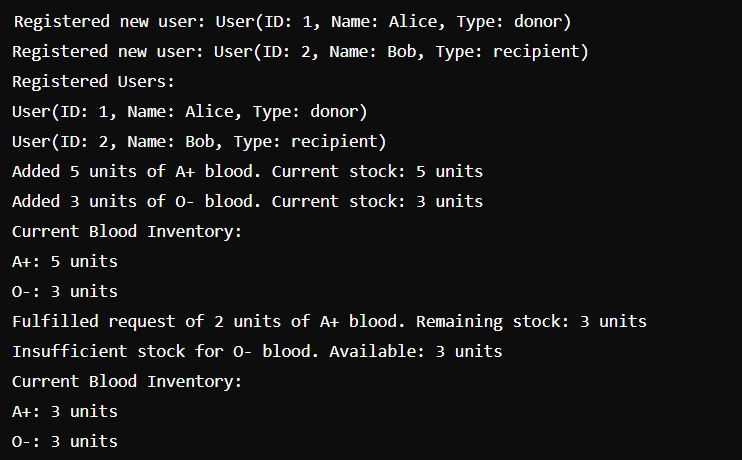
system.inventory.request\_blood("A+", 2)

system.inventory.request\_blood("O-", 5)

# Display inventory after requests

system.inventory.display\_inventory()

* 1. Output



**9. Conclusion**

The online blood banking system aims to revolutionize the way blood banks operate by introducing efficiency, accuracy, and ease of access. By leveraging modern technologies, the system will ensure timely availability of blood, enhance donor and recipient experiences, and streamline administrative tasks. This project will ultimately contribute to saving lives by making blood donation and distribution more efficient and reliable.