1. INTRODUCTION

Organ failure happens when an organ is damaged due to injury or disease, which can lower the quality of life and sometimes lead to death. Organ donation is a generous act that can save lives through organ transplantation. For a transplant to be successful, the donated organ must be healthy, and the donor and recipient must be properly matched. The process should not put the donor's life at risk, especially if the donor is alive. The first successful organ transplant took place in 1954 between twin brothers. Since then, the number of transplants has grown, but the need for organs is still much greater than the number of available donors. On average, 20 people die each day waiting for a transplant, and a new person is added to the waiting list every 10 minutes. Access to this list is crucial, but factors like geography and income level can affect whether someone gets referred for a transplant. Therefore, the process of giving organs should be fair to everyone.

There are two types of organ donation: deceased donation and living donation. In both cases, the donor is carefully examined by medical experts. If the donor has died, a brain death test is done. If the donor is alive, doctors check to make sure the person is healthy enough to safely donate. All the donor’s medical details are then sent to a procurement organizer, who reviews the case and ensures everything is properly recorded in the medical system. If the donor is eligible and has given consent to donate to a stranger, the information is sent to the organ transplant organizer. This team matches donors with patients on the waiting list and creates a ranked list of who should receive the organ. A transplant surgeon then decides whether the organ is suitable for the chosen patient based on health records. If approved, the organ is removed and sent to the patient’s hospital for the transplant. However, if a living donor wants to give an organ to someone they know, the matching step is skipped, and the process goes straight to surgery.

In the past, organ donor identification began with a phone call between hospitals and procurement teams, but this was slow and only led to a small number of actual donations. Today, computer systems handle this process with instant messages. While this is faster, it raises concerns about data safety. The success of these systems depends on the ability of hospitals to protect information from hackers or dishonest employees. Trust in the system is critical. Another big issue is transparency. According to the World Health Organization, about 10% of all transplants may involve organs obtained unethically, such as through illegal trading. Because there is little transparency, some hospitals may give organs to those who pay more, ignoring patients who are higher on the waiting list. Also, many current systems are slow and outdated, and most hospitals still do not follow common standards for handling or sharing data. These issues cause delays and reduce the chances of saving lives through timely organ transplants.

2. LITERATURE SURVEY

Title: "Organ donation decentralized application using blockchain technology,''

Abstract: The proposed system is an organ donation decentralized app using blockchain technology. It would be a web application for patients to register their information-most importantly medical ID, blood type, organ type and state. The system would work on a first-in, first-out basis unless a patient is in critical condition.

Title: ''Using blockchain technology for the organ procurement and transplant network,''

Abstract: The organ donation system in the United States is centralized and difficult to audit by the general public. This centralized approach may lead to data integrity issues in the future. The Organ Procurement and Transplant Network (OPTN) was built and maintained by a non-governmental organization called the United Network for Organ Sharing (UNOS) under its proprietary UNet(SM) umbrella platform. This platform is made up of proprietary closed source software and does not provide the general public easy access to the organ transplant data for auditing. This study investigates the feasibility, challenges, and advantages of a blockchain-based OPTN.

Title: ''Use of forensic DNA testing to trace unethical organ procurement and organ trafficking practices in regions that block transparent access to their transplant data,''

Abstract: This study proposes an approach to track unethically procured organs in particular in countries or regions where investigations cannot be performed by utilizing forensic DNA methodology. Using China as an example, previous research has concluded that organs in China are in part unethically and extra-legally procured (so called "forced organ harvesting") from living prisoners of conscience without consent. Using forensic DNA-analysis, we propose building a DNA data bank from missing prisoners of conscience in China and comparing these results with DNA from donor organs in patients who received transplants in China. Biological materials collected in China will provide DNA directly or indirectly from potential victims of forced organ harvesting. Archival biopsies from transplant recipients' donor organs will provide DNA profiles of donors. Verified match between DNA profiles of transplanted organs and missing victims will establish proof of such connection, thus provides evidence despite a lack of transparency.

Title: "Decentralized and distributed system for organ/tissue donation and transplantation,''

Abstract: In today's era of digitization, many technologies have evolved that every manual work can be digitally automatized. In the digital automatizing process, security and privacy are the most important and highly demanding aspects. Blockchain offers many features that can be used in almost every sphere of life. Features like decentralization, transparency, privacy makes it an extremely useful technology. Therefore, by making use of all these features, several problems in healthcare sector can

be solved like removing complex network of third parties and lack of traceability of transactions. This paper presents a decentralized, secure and transparent organ and tissue transplant web application (also called DApp), which not only nullifies the role of any third party involved in the organ transplantation, but also is a cost-effective solution that saves the patients from high cost of transplantation. The details and Electronic Medical Record (EMR) are hashed using the IPFS (a distributed file server), which reduces the cost of upload to a great extent as shown in the results section of this paper.

Title: "A systematic review of the use of blockchain in healthcare,''

Abstract: Blockchain technology enables a decentralized and distributed environment with no need for a central authority. Transactions are simultaneously secure and trustworthy due to the use of cryptographic principles. In recent years, blockchain technology has become very trendy and penetrated different domains, mostly due to the popularity of cryptocurrencies. One field where blockchain technology has tremendous potential is healthcare, due to the need for a more patient-centric approach to healthcare systems and to connect disparate systems and increase the accuracy of electronic healthcare records (EHRs). In this systematic review, an analysis of state-of-the-art blockchain research in the field of healthcare is conducted. The aim is to reveal the potential applications of the technology and to highlight the challenges and possible directions of blockchain research in healthcare. First, background information is discussed, followed by a description of the exact methodology used in this paper. Next, an analysis of the results is given, which includes a bibliometric overview, an analysis of gathered data and its properties, and the results of a literature quality assessment. Lastly, there is a discussion of the results from the analysis. The findings indicate that blockchain technology research in healthcare is increasing and it is mostly used for data sharing, managing health records and access control. Other scenarios are very rare. Most research is aimed at presenting novel structural designs in the form of frameworks, architectures or models. Findings also show that technical details about the used blockchain elements are not given in most of the analyzed publications and that most research does not present any prototype implementation or implementation details. Often even with a prototype implementation, no details about blockchain elements are given.

Title: ''Organ's transplantation—How to improve the process ''

Abstract: The transplant of cadaveric organs must be performed in a short period of time in order to achieve satisfactory results. In Hospital S. João (HSJ), a large Portuguese hospital, during 2008 and 2009, 65 and 61 respectively potential donors were identified, but 12 and 19 of them were not validated as such in time. The number of validated donors could increase if the information workflow between donor hospitals and coordinator offices became more efficient. The goal of this work is to design and implement a multi-agent software platform to assist the information workflow between donor hospitals and coordinator offices. Through several meetings with HSJ coordinator office it was characterized a

set of basic data that would allow coordinator offices to early identify possible organs donors. This preliminary characterization provided the necessary grounds for the development of an agent-based software application allowing the storage and management of potential donors' information and optimizing the information workflow. The information workflow and the current communication processes characterization allowed the development of a multi-agent web platform, providing a way to assist the information workflow, between coordinator hospitals and their attached hospitals network. The platform also improves direct communication between coordinator offices about most relevant facts. By using this tool or a similar one the information workflow between donor hospitals and coordinator offices can become more efficient, optimizing the pre-transplantation tasks and consequently the number of successful transplants in our country.

Title: ‘‘Conceptual framework for general traceability solution: Description and bases,’’

Abstract: Purpose The purpose of this paper is to describe a proposed framework for traceability purpose. Hence, the framework provides a formal and structured way of viewing a traceability solution. This structure lays the required bases for a traceability system before starting development and deployment. Design/methodology/approach the paper examines several traceability publications, including systems and literature review. The study covers the traceability implementation phase. Therefore, this research approaches the traceability issue from three perspectives (description, engineering and executive one). The separation between aspects is essential when describing and comparing traceability systems. This distinction is also helpful when recommending solution improvements. Findings The framework identifies six traceability bases: aims, functions, specifications, data classification, processes and procedures. These can establish a basis for a general-purpose tool that can enable users to develop an efficient traceability solution. Thus, the first ontology expresses the framework domain and ensures optimal use of it. The second one represents the bases that can serve as a knowledge base to manage the product data. Research limitations/implications the suggested framework tackles the implementation of traceability. Therefore, the design emphasizes the importance of technological concerns. Some studied cases could require more research angles (i.e. economic and legislative). Thus, framework enrichment is essential for further improvements. Practical implications the framework helps users to develop a general, interoperable and scalable traceability solution. These are important to promote the generalization of traceability systems. Originality/value the framework fulfills a requirement for establishing general traceability foundations. Therefore, the guide independently operates of the product or the industry specificity. Moreover, the bases aim to bridge the gap between solution engineering and traceability requirements.

Title: ‘‘Mechanisms for online organ matching”

Abstract: Matching donations from deceased patients to patients on the waiting list account for over 85% of all kidney transplants performed in Australia. We propose a simple mechanism to perform this matching and compare this new mechanism with the more complex algorithm currently under consideration by the Organ and Tissue Authority in Australia. We perform a number of experiments using real world data provided by the Organ and Tissue Authority of Australia. We find that our simple mechanism is more efficient and fairer in practice compared to the other mechanism currently under consideration.

Title: ‘‘Kidner A worldwide decentralized matching system for kidney transplants,’’

Abstract: Individuals suffering from kidney failure today face significant challenges in order to obtain a transplant. They are placed on a waiting list and ranked by priority in hope that a kidney from a deceased donor is a transplant match. They do have another option: a living donor; someone they know, family or friend, willing to give them a kidney. These people may not be a transplant match, however there is a solution, a “Kidney Exchange” or a “Kidney Paired Donation”. In these programs, if two mismatched pairs (living donor and kidney recipient) can be grouped together so that they become transplant matches, both kidney failure patients can receive a kidney. While a great solution, these programs have a significant pitfall. They are limited to the specific registry regions participating in their program. The Kidner project was developed to help these exchange programs better detect life-saving opportunities and enable more people to access kidney transplants.

Title: ‘‘Organ donation decentralized application using blockchain technology,’’

Abstract: The proposed system is an organ donation decentralized app using blockchain technology. It would be a web application for patients to register their information-most importantly medical ID, blood type, organ type and state. The system would work on a first-in, first-out basis unless a patient is in critical condition.

3. SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

Organ donation and transplantation processes today often rely on **centralized systems** managed by hospitals or national agencies. These systems face multiple issues that hinder efficiency, transparency, and fairness in handling critical medical decisions.

Challenges in the existing systems include:

- **Lack of Data Accountability:** There's no solid mechanism to prove who accessed or modified data, leading to possible unauthorized actions.
- **No Immutability:** Centralized databases can be altered, raising concerns about data integrity.
- **Limited Auditability:** It's difficult to track historical actions and verify if proper procedures were followed.
- **Low Transparency:** Patients, families, and auditors cannot easily access real-time updates or trace the donation process.
- **No Real-Time Traceability:** The movement of donated organs from donor to recipient is hard to monitor step-by-step.
- **Erosion of Trust:** In many countries, corruption or mismanagement in the system reduces public trust in organ allocation fairness.

3.2. DISADVANTAGES OF EXISTING SYSTEM

- **Data Tampering Risks:** Without tamper-proof storage, records can be changed or deleted.
- **Opaque Matching Processes:** It's often unclear how donor-recipient matching is done, causing frustration and delays.
- **Manual Coordination:** Coordination between hospitals, transporters, and surgeons is mostly manual, leading to errors and delays.
- **Lack of Donor Verification:** There is no system to ensure that all donor information is verified and medically approved transparently.
- **No Public Trust Mechanism:** Patients and families have no way to verify if the system is being used fairly or ethically.

3.3. PROPOSED SYSTEM

The proposed blockchain-based solution for donated organ transplantation is explained in Section III. Then, it is followed by the implementation details of the proposed blockchain-based solution in Section IV and the details of testing and evaluation in Section V. The discussion and analysis of the proposed solution are given in Section VI. Finally, section VII concludes the paper by summarizing our contributions and outlining future research opportunities.

In this section, we present details of our blockchain-based organ donation and transplantation solution. Figure 2 presents an overview of the system architecture of our proposed solution. It shows that our solution uses two smart contracts (SCs); namely, organ donation and organ transplantation. The participants can access the functions and events of these smart contracts through a front-end decentralized application (DApp), which is connected by an application program interface (API). Every smart contract has unique functions that can be executed only by pre-authorized participants, who will have the ability to access data stored on the chain to review transactions, logs, and events.

The participants include doctors, hospital transplant team members, procurement organizers, organ matching organizers, a transporter and a transplant surgeon. The Organ Donation Smart Contract is responsible for creating a waiting list, accepting donors after medical test approval, and auto-matching between the donor and recipient. The Organ Transplantation Smart Contract is mostly in charge of the transplant process. It has three parts: removing an organ from a donor, getting the organ to the recipient, and putting the organ into the recipient. All the previous phases are logged and stored on the ledger for revision and verification purposes. Additionally, authorization, secrecy, and privacy are ensured by utilizing a private permissioned Ethereum blockchain.

3.4. ADVANTAGES OF PROPOSED SYSTEM

- **Decentralization:** Removes the single point of failure by distributing data across nodes.
- **Transparency:** All actions are visible to authorized parties, enhancing trust and accountability.
- **Traceability:** Each organ's journey from donor to recipient is trackable in real-time.
- **Security:** Transactions are encrypted and immutable, reducing the risk of tampering or data loss.
- **Automated Matching:** Reduces human errors and speeds up donor-recipient pairing.
- **Role-Based Access:** Enhances privacy and control—each participant has access only to necessary functions.
- **Cost-Efficient:** Reduces intermediaries and paperwork, lowering the overall cost of transplantation logistics.
- **Real-Time Updates:** System can notify relevant participants about every status change immediately.

3.5. LIMITATIONS OF PROPOSED SYSTEM

- **High Initial Setup Cost:** Deploying private Ethereum networks and infrastructure requires investment.
- **Technical Complexity:** Requires specialized skills in blockchain development and smart contracts.

- **Scalability Concerns:** The system might face performance issues if scaled to a national or global level without optimization.
- **Regulatory Acceptance:** Legal and ethical frameworks around blockchain in healthcare are still evolving and may vary by region.
- **Smart Contract Bugs:** If smart contracts are not thoroughly audited, bugs can lead to critical failures or vulnerabilities.
- **Internet Dependency:** Requires stable internet connectivity for all stakeholders to interact with the system.

4. SYSTEM STUDY

4.1. Feasibility Study

The **Organ Matching & Transplantation System using Blockchain Technology** aims to solve a critical real-world problem by improving transparency, security, and efficiency in organ donations and transplantation. This feasibility study thoroughly evaluates the **technical**, **operational**, and **economic** aspects of the proposed system to ensure its viability and sustainability.

4.1.1. TECHNICAL FEASIBILITY

- Technical feasibility determines whether the required technology, skills, and infrastructure are available to implement the system.
- **Current Technological Readiness:**
  - **Blockchain Platforms (Ethereum, Polygon, Hyperledger)**: These platforms are well-established and support smart contract functionalities that can log and verify donor-recipient matches immutably.
- **Example**: Ethereum smart contracts can match donor organs to recipients using rules (e.g., blood type, HLA compatibility, urgency) and auto-trigger alerts without human bias.
- **Smart Contracts**: They reduce human error and manipulation by enforcing logic automatically.
- **Example**: A smart contract might be written to automatically assign an organ to the highest-priority recipient on a waitlist once availability is confirmed.
- **System Architecture:**
  - **Front-End**: Technologies like **React.js or Next.js** will offer a responsive and user-friendly interface for patients, hospitals, and administrators.
- **Back-End**: A **Node.js/Express.js** server can handle hospital uploads, user roles, API communications, and real-time blockchain data fetching.
- **Secure File Storage**: **IPFS (Interplanetary File System)** enables secure, tamper-proof, decentralized storage for sensitive medical documents like donor consent, lab reports, etc.
- **Integration:**
  - **APIs for Hospital/Health System Integration**: Existing hospital systems can send patient data to the blockchain network securely using REST APIs.
- **Aadhaar/Biometric Verification**: Government APIs can authenticate the identity of donors/recipients, eliminating fake entries.
- **Developer Expertise and Tools:**
  - The tools, frameworks, and platforms are widely available.

- Open-source libraries and SDKs from blockchain providers will reduce development time and ensure code reliability.
- Availability of cloud platforms (e.g., AWS, Azure) will support scalable deployments.

Conclusion: With mature technology stacks, decentralized storage, and existing blockchain infrastructure, the system is **technically feasible** and implementable using current industry practices.

4.1.2. OPERATIONAL FEASIBILITY

Operational feasibility determines how well the system solves the identified problems and how easily it can be adopted and used.

Stakeholder Adoption:

- **Hospitals**: The system provides dashboards to manage donor and patient records, verify documentation, and coordinate logistics in real-time.
- **Government Bodies**: Regulatory authorities can monitor and audit transactions, approvals, and match-making transparently.
- **NGOs/Donor Agencies**: These can help promote, onboard, and monitor donation campaigns ethically using system data.

Benefits in Daily Operations:

- **Transparency & Accountability**: Every transaction (donor registration, match made, organ allocation) is stored on-chain and publicly verifiable.
- **Fraud Prevention**: Cases of organ trafficking, black market deals, and favoritism are reduced due to blockchain's immutable audit trail.
    - **Example**: An unauthorized recipient cannot receive a transplant unless verified through blockchain rules.
- **24/7 Access**: Since it is decentralized, the system operates even if one node/server goes down.
- **Faster Organ Allocation**: Matching logic runs in real-time, preventing organ wastage due to delay.

Legal and Ethical Compliance:

- The system supports adherence to national and international organ donation laws (e.g., THOA in India, UNOS in the USA).
- Ensures ethical practices via transparent matching and third-party audits.

Training & Adoption:

- Minimal training is required due to the intuitive user interface.
- Hospitals and organizations will be provided with digital manuals and helpdesks.

Conclusion: By addressing real-time coordination, fraud mitigation, and emergency response, the project is **operationally feasible** and aligns with the workflows of medical and administrative teams.

4.1.3. ECONOMIC FEASIBILITY

Economic feasibility assesses whether the system is affordable, and whether the long-term benefits justify the costs.

Cost Breakdown:

- **Initial Investment**:
  - Blockchain setup and smart contract development
  - Front-end and back-end development
  - Cloud infrastructure and decentralized storage (e.g., IPFS node)
  - Integration with APIs and health systems
- **Recurring Costs**:
  - Smart contract gas fees (if on public blockchain)
  - Cloud hosting and server maintenance
  - Annual security audits and updates
  - Support and training services

Cost Saving Benefits:

- **Reduced Administrative Overhead**:
  - Traditional paper-based or Excel-based matching and logging are error-prone and inefficient.
- **Less Litigation and Fraud**:
  - Transparency helps avoid legal disputes, penalties, and medical malpractice claims.
- **Faster Transplants = Lower ICU/Hospitalization Costs**:
  - Timely matching of organs shortens waiting time and reduces the cost of extended treatment.
- **Low Downtime**:
  - Decentralization reduces reliance on single server or system admin.

Funding Sources:

- **Government Grants**: Public health initiatives and digital healthcare missions.
- **CSR Funds from Tech/Healthcare Companies**: Many companies sponsor health-tech innovations.
- **NGO/Philanthropic Donations**: Foundations focusing on medical equity and accessibility.
- **Private Hospitals**: Subscription-based access to the platform.

Future Revenue/Value Opportunities:

- Premium features for international donors or private hospitals.
- Blockchain data analytics for research institutions.

- Licensing of the platform to other countries or regions.

Conclusion: The project is **economically feasible**. While the initial cost is moderate, the long-term financial, healthcare, and social benefits far outweigh it. It can even become self-sustaining or profitable via government support or tiered monetization.

5. SYSTEM REQUIREMENTS

5.1.HARDWARE REQUIREMENTS

MINIMUM (Required for Execution)		MY SYSTEM (Development)
System	Pentium IV 2.2 GHz	i3 Processor 5th Gen
Hard Disk	20 GB	500 GB
RAM	1 GB	4 GB

5.2.SOFTWARE REQUIREMENTS

Operating System	Windows 10/11
Development Software	Python 3.10
Programming Language	Python
Domain	Machine Learning
Integrated Development Environment (IDE)	Visual Studio Code
Front End Technologies	HTML5, CSS3, Java Script
Back End Technologies or Framework	Django
Database Language	SQL
Database (RDBMS)	MySQL
Database Software	WAMP or XAMPP Server
Web Server or Deployment Server	Django Application Development Server
Design/Modelling	Rational Rose

5.3.SOFTWARE ENVIRONMENT

What is Python programming language?

Python is a **high-level, general-purpose, interpreted** programming language.

1) High-level

Python is a high-level programming language that makes it easy to learn. Python doesn't require you to understand the details of the computer in order to develop programs efficiently.

2) General-purpose

Python is a general-purpose language. It means that you can use Python in various domains including:

- Web applications
- Big data applications
- Testing
- Automation
- Data science, machine learning, and AI
- Desktop software
- Mobile apps

The targeted language like SQL which can be used for querying data from relational databases.

3) Interpreted

Python is an interpreted language. To develop a Python program, you write Python code into a file called source code.

To execute the source code, you need to convert it to the machine language that the computer can understand. And the Python **interpreter** turns the source code, line by line, once at a time, into the machine code when the Python program executes.

Compiled languages like Java and C# use a **compiler** that compiles the whole source code before the program executes.

Why Python

Python increases your productivity. Python allows you to solve complex problems in less time and fewer lines of code. It's quick to make a prototype in Python.

Python becomes a solution in many areas across industries, from web applications to data science and machine learning.

Python is quite easy to learn in comparison with other programming languages. Python syntax is clear and beautiful.

Python has a large ecosystem that includes lots of libraries and frameworks.

Python is cross-platform. Python programs can run on Windows, Linux, and macOS.

Python has a huge community. Whenever you get stuck, you can get help from an active community.

Python developers are in high demand.

History of Python

- Python was created by Guido Van Rossum.
- The design began in the late 1980s and was first released in February 1991.

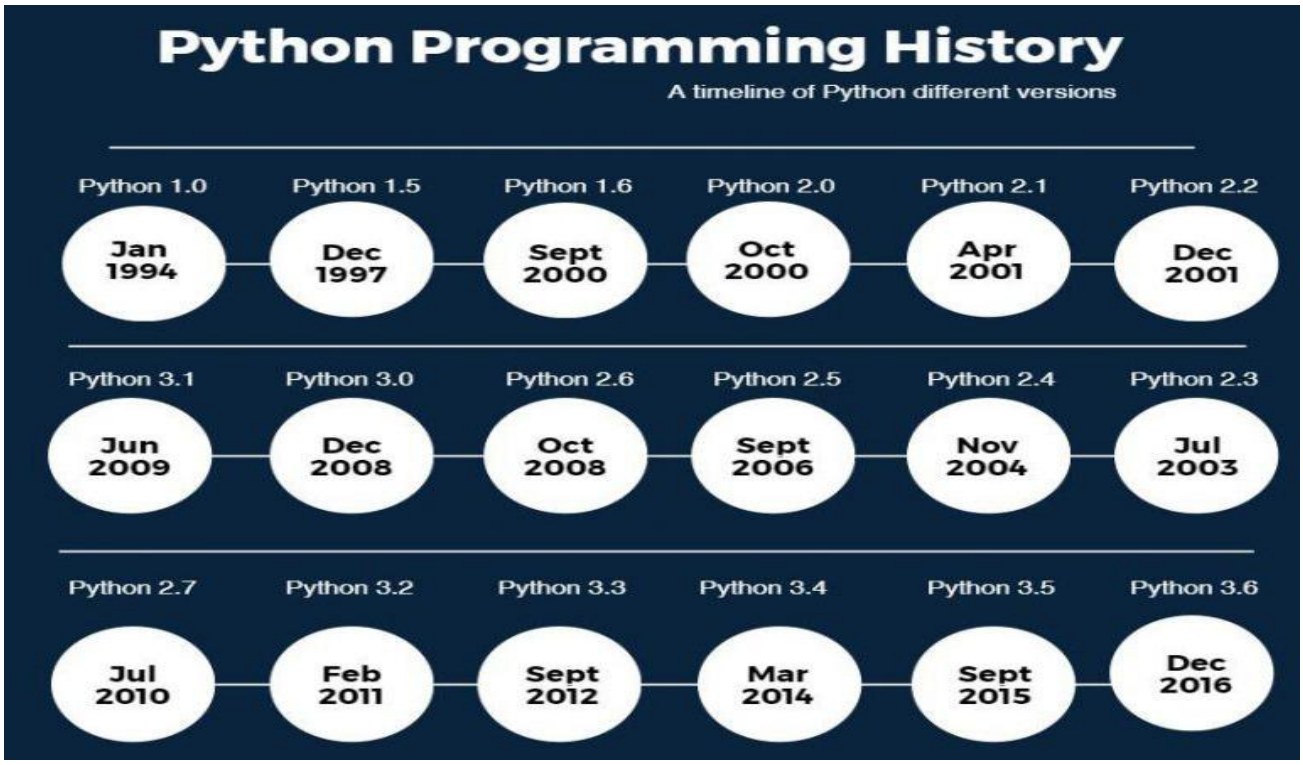
Why the name Python?

No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late 70s. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

Python Version History

Implementation started - December 1989

Internal releases – 1990

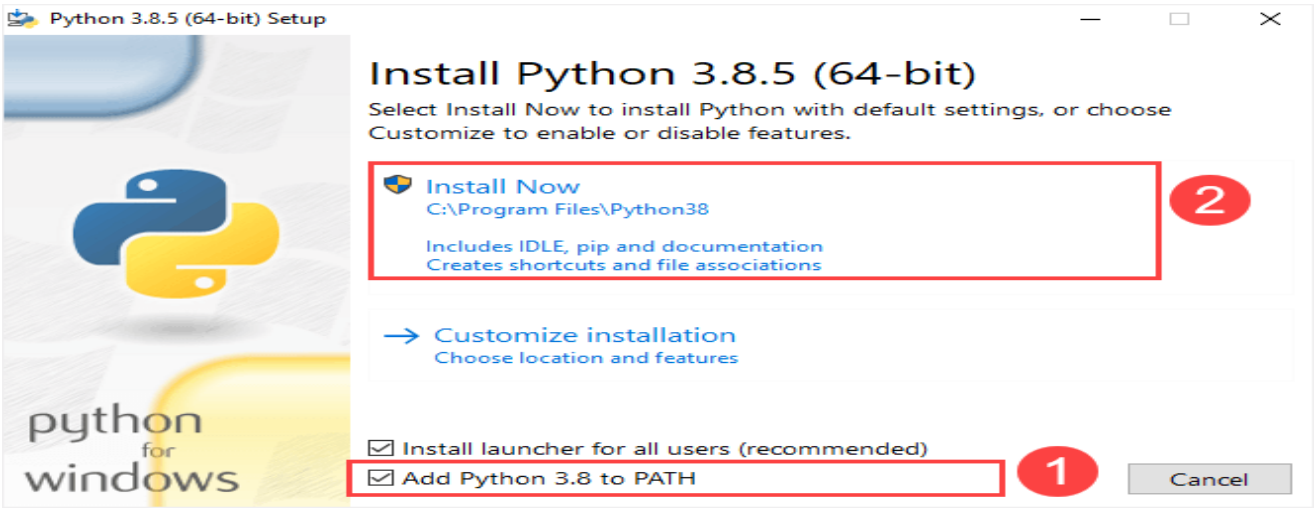


Install Python on Windows

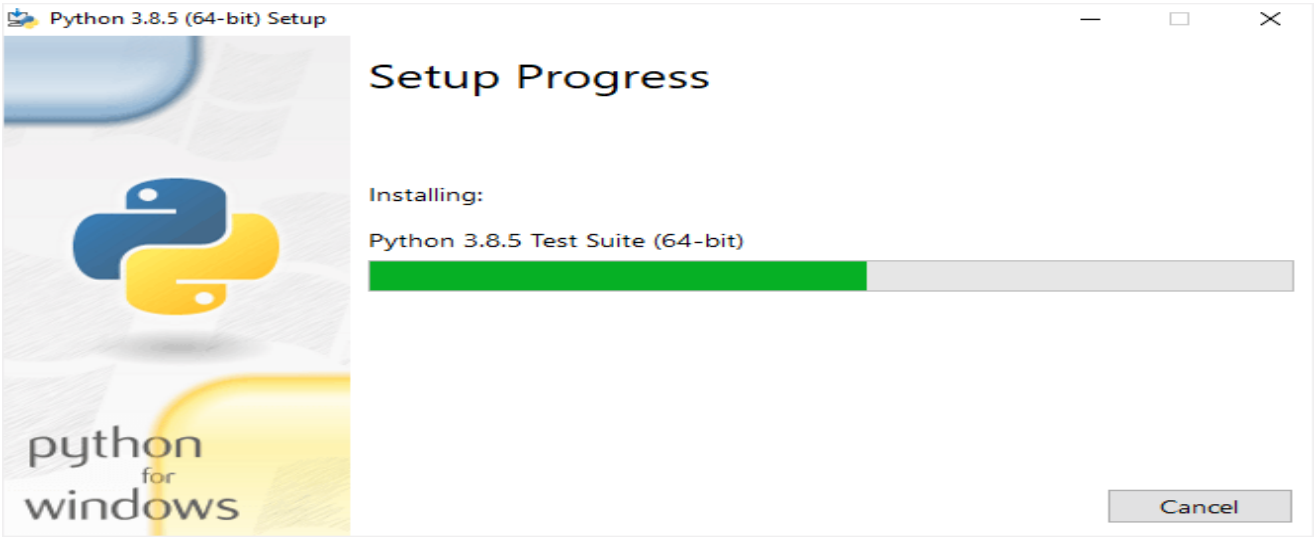
First, download the latest version of Python from the download page.

Second, double-click the installer file to launch the setup wizard.

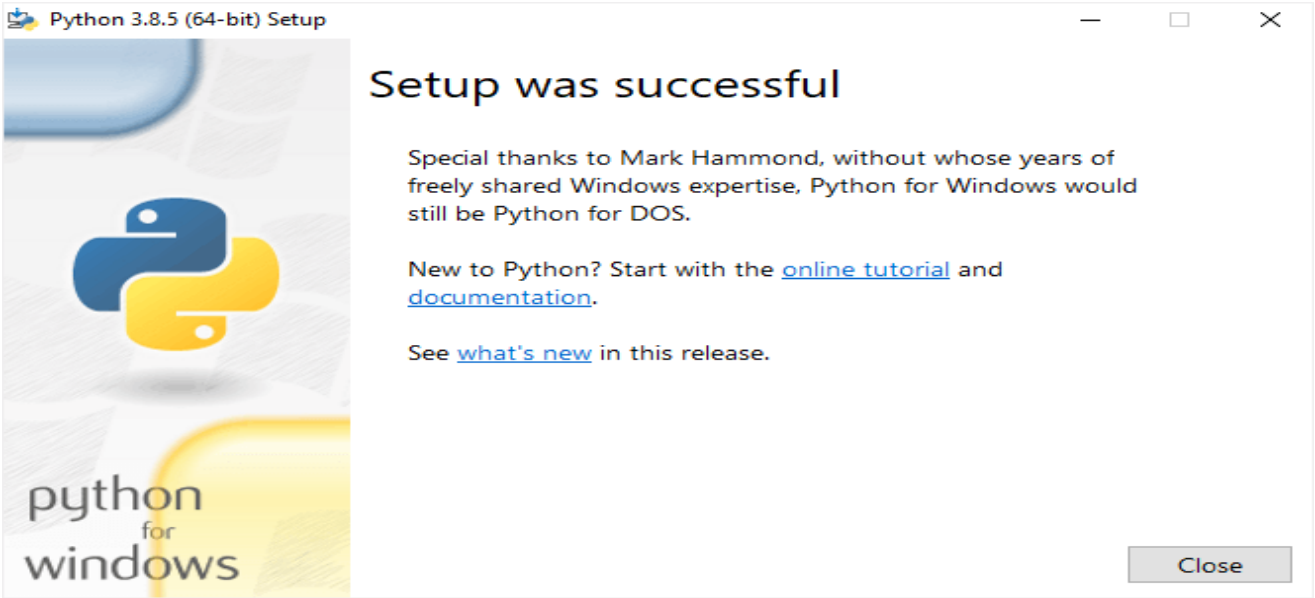
In the setup window, you need to check the **Add Python 3.8 to PATH** and click Install Now to begin the installation.



It'll take a few minutes to complete the setup.

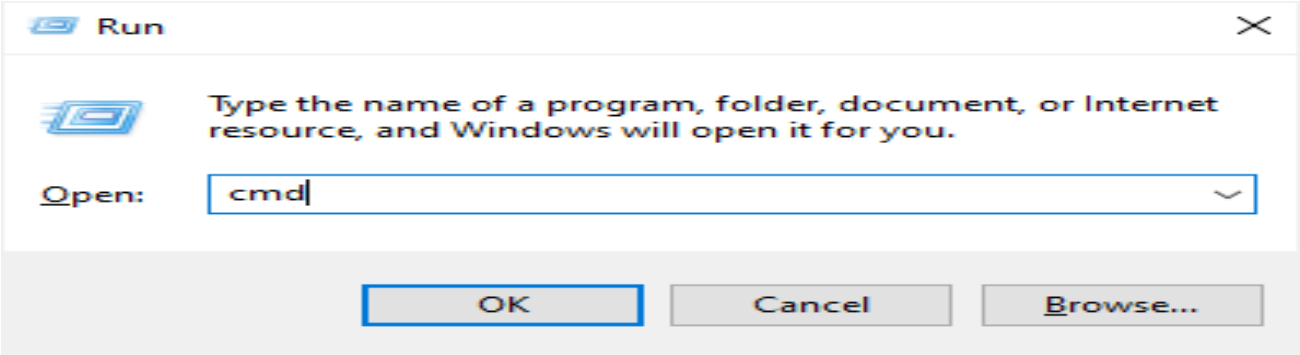


Once the setup completes, you'll see the following window:

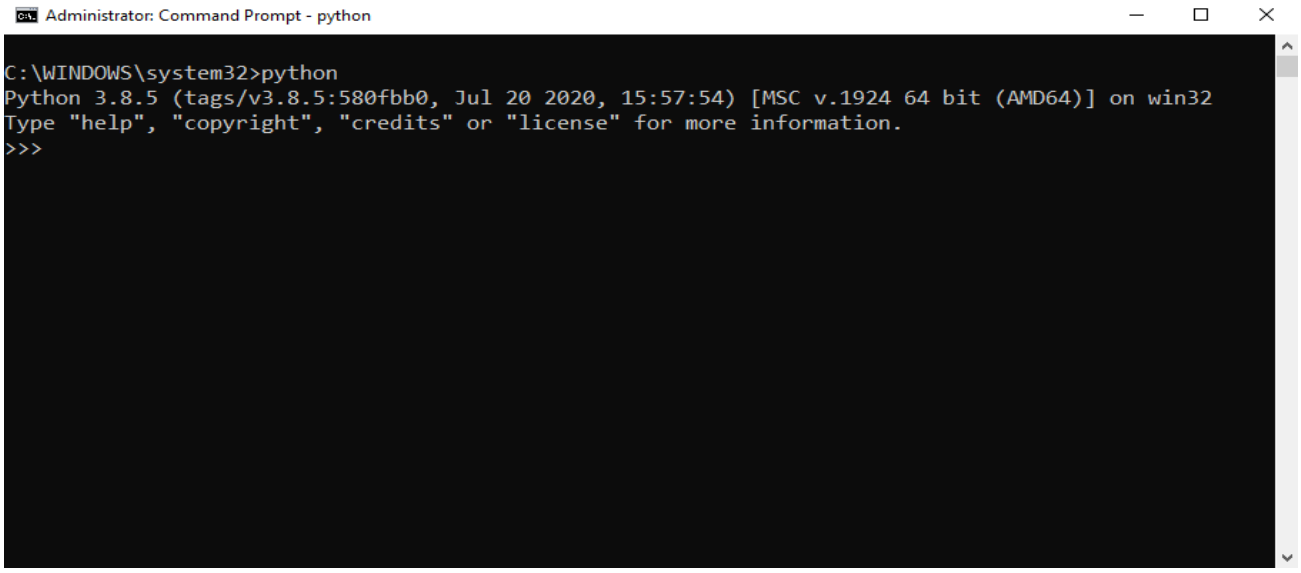


Verify the installation

To verify the installation, you open the Run window and type cmd and press Enter:



In the Command Prompt, type python command as follows:



If you see the output like the above screenshot, you've successfully installed Python on your computer.

To exit the program, you type Ctrl-Z and press Enter.

If you see the following output from the Command Prompt after typing the python command:

'python' is not recognized as an internal or external command, operable program or batch file.

Likely, you didn't check the **Add Python 3.8 to PATH** checkbox when you install Python.

Install Python on macOS

It's recommended to install Python on macOS using an official installer. Here are the steps:

- First, download a Python release for macOS.
- Second, run the installer by double-clicking the installer file.
- Third, follow the instruction on the screen and click the Next button until the installer completes.

Install Python on Linux

Before installing Python 3 on your Linux distribution, you check whether Python 3 was already installed by running the following command from the terminal:

python3 --version

If you see a response with the version of Python, then your computer already has Python 3 installed. Otherwise, you can install Python 3 using a package management system.

For example, you can install Python 3.10 on Ubuntu using apt:

sudo apt install python3.10

To install the newer version, you replace 3.10 with that version.

A quick introduction to the Visual Studio Code

Visual Studio Code is a lightweight source code editor. The Visual Studio Code is often called VS Code. The VS Code runs on your desktop. It's available for Windows, macOS, and Linux.

VS Code comes with many features such as IntelliSense, code editing, and extensions that allow you to edit Python source code effectively. The best part is that the VS Code is open-source and free.

Besides the desktop version, VS Code also has a browser version that you can use directly in your web browser without installing it.

This tutorial teaches you how to set up Visual Studio Code for a Python environment so that you can edit, run, and debug Python code.

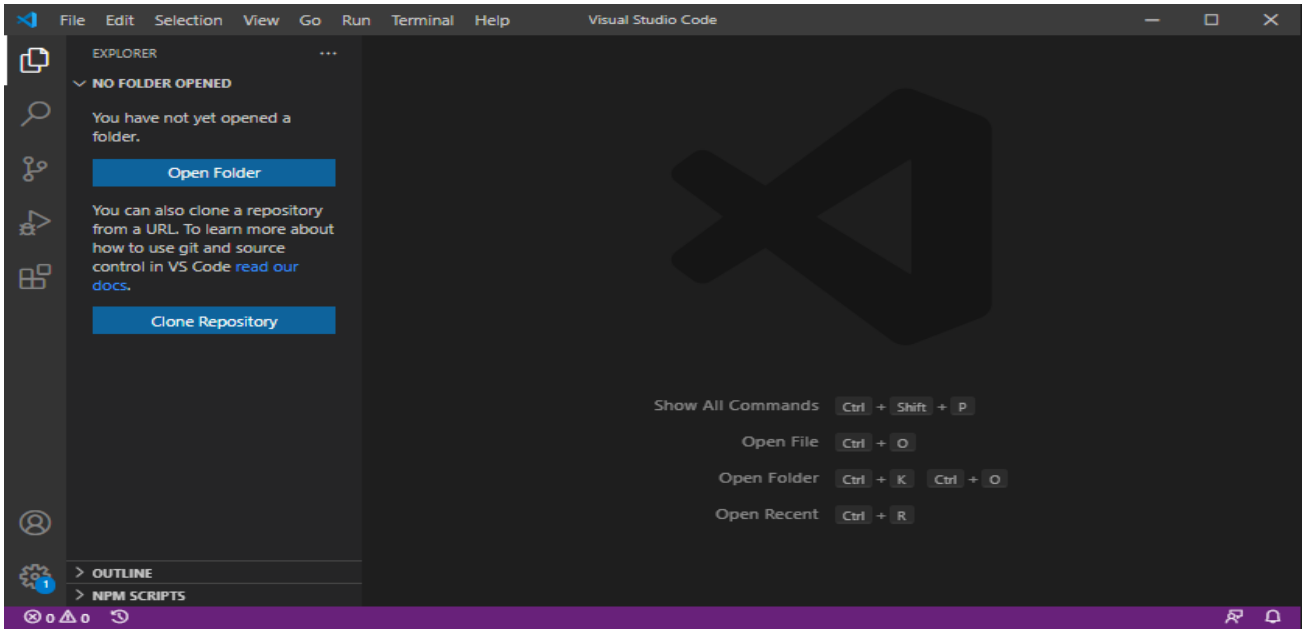
Setting up Visual Studio Code

To set up the VS Code, you follow these steps:

First, navigate to the VS Code official website and download the VS code based on your platform (Windows, macOS, or Linux).

Second, launch the setup wizard and follow the steps.

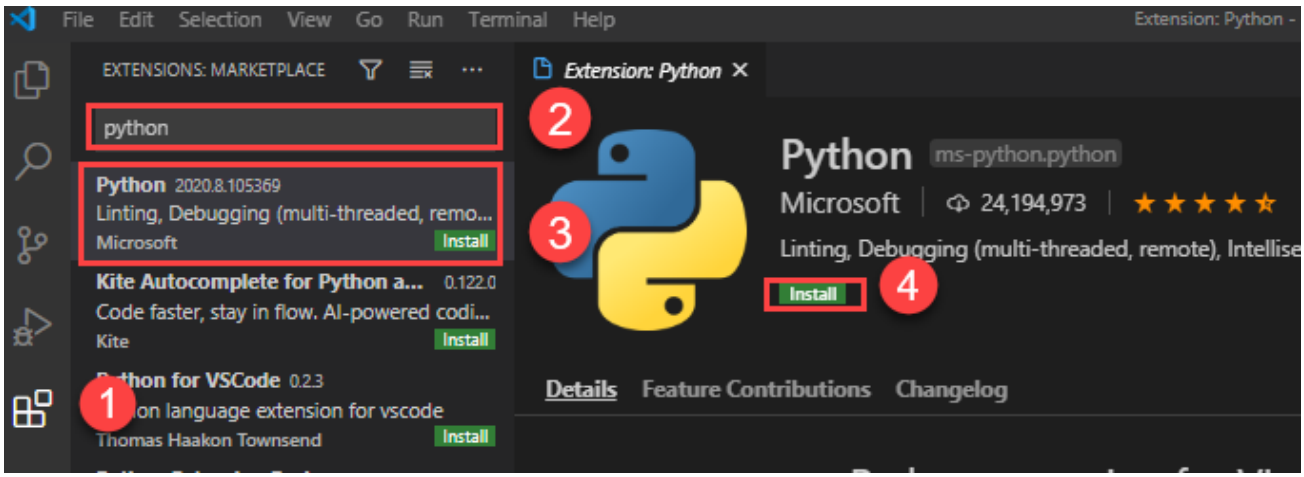
Once the installation completes, you can launch the VS code application:



Install Python Extension

To make the VS Code works with Python, you need to install the Python extension from the Visual Studio Marketplace.

The following picture illustrates the steps:



- First, click the **Extensions** tab.
- Second, type the python extension pack keyword on the search input.
- Third, click the Python extension pack. It'll show detailed information on the right pane.
- Finally, click the **Install** button to install the Python extension.

Now, you're ready to develop the first program in Python.

Creating a new Python project

First, create a new folder called helloworld.

Second, launch the VS code and open the helloworld folder.

Third, create a new app.py file and enter the following code and save the file:

```
print('Hello, World!')
```

Code language: Python (python)

The print() is a built-in function that displays a message on the screen. In this example, it'll show the message 'Hello, Word!'.

What is a function

When you sum two numbers, that's a function. And when you multiply two numbers, that's also a function.

Each function takes your inputs, applies some rules, and returns a result.

In the above example, the print() is a function. It accepts a string and shows it on the screen.

Python has many built-in functions like the print() function to use them out of the box in your program.

In addition, Python allows you to define your functions, which you'll learn how to do it later.

Executing the Python Hello World program

To execute the app.py file, you first launch the Command Prompt on Windows or Terminal on macOS or Linux.

Then, navigate to the helloworld folder.

After that, type the following command to execute the app.py file:

python app.py

Code language: Python (python)

If you use macOS or Linux, you use python3 command instead:

python3 app.py

Code language: CSS (css)

If everything is fine, you'll see the following message on the screen:

Hello, World!

Code language: Python (python)

If you use VS Code, you can also launch the Terminal within the VS code by:

- Accessing the menu **Terminal > New Terminal**
- Or using the keyboard shortcut Ctrl+Shift+`.

Typically, the backtick key (`) locates under the Esc key on the keyboard.

Python IDLE

Python IDLE is the Python Integration Development Environment (IDE) that comes with the Python distribution by default.

The Python IDLE is also known as an interactive interpreter. It has many features such as:

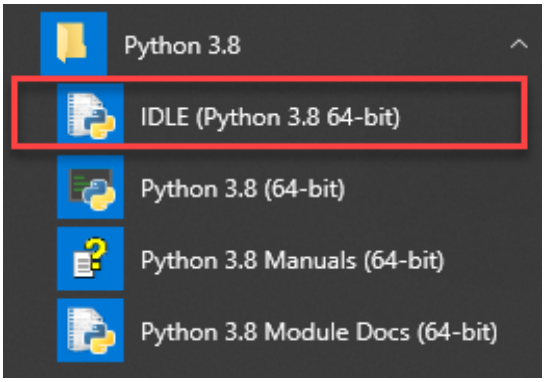
- Code editing with syntax highlighting
- Smart indenting
- And auto-completion

In short, the Python IDLE helps you experiment with Python quickly in a trial-and-error manner.

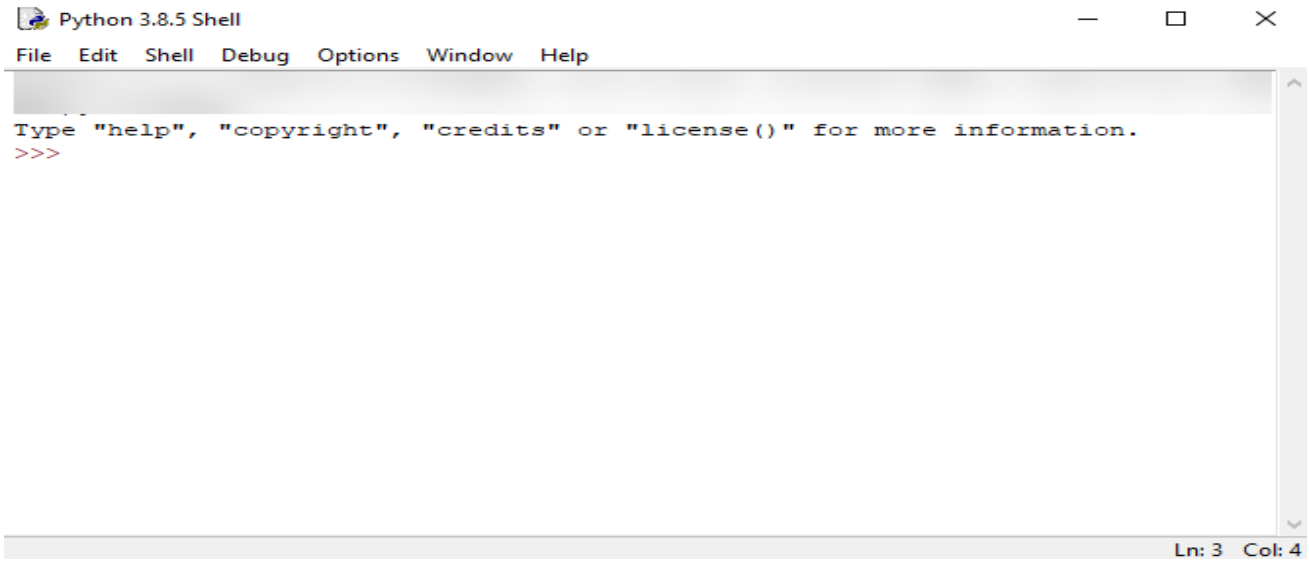
The following shows you step by step how to launch the Python IDLE and use it to execute the

Python code:

First, launch the Python IDLE program:

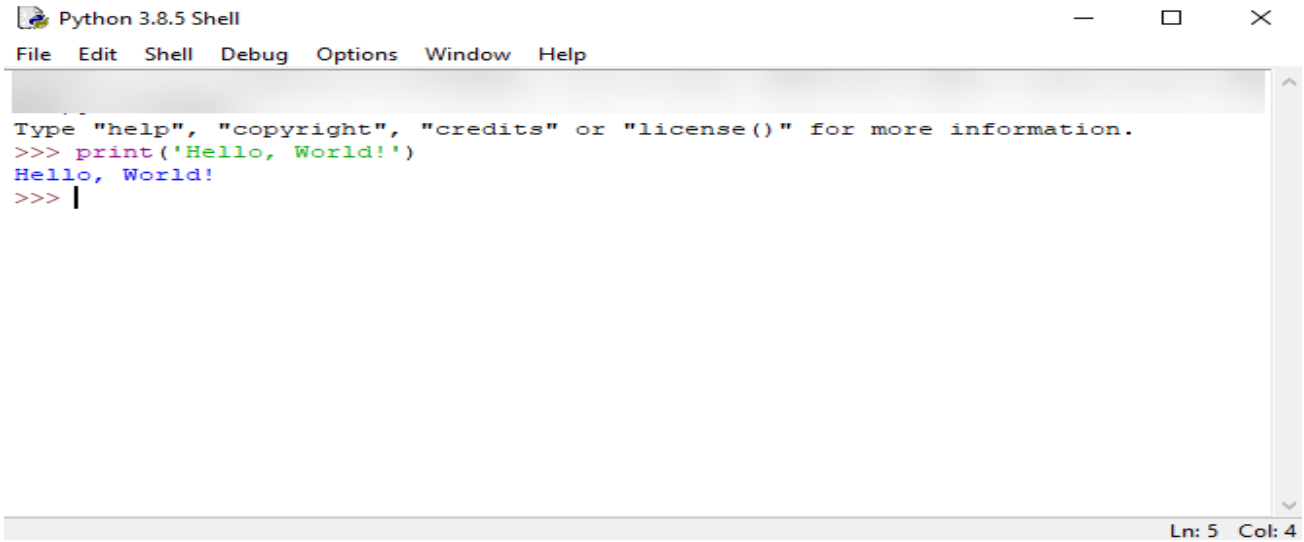


A new Python Shell window will display as follows:



Now, you can enter the Python code after the cursor >>> and press Enter to execute it.

For example, you can type the code print('Hello, World!') and press Enter, you'll see the message Hello, World! immediately on the screen:



Python Syntax

Whitespace and indentation

If you've been working in other programming languages such as Java, C#, or C/C++, you know that these languages use semicolons (;) to separate the statements.

However, Python uses whitespace and indentation to construct the code structure.

The following shows a snippet of Python code:

```
# define main function to print out something
def main():
    i = 1
    max = 10
    while (i < max):
        print(i)
        i = i + 1
# call function main
main()
```

The meaning of the code isn't important to you now. Please pay attention to the code structure instead. At the end of each line, you don't see any semicolon to terminate the statement. And the code uses indentation to format the code.

By using indentation and whitespace to organize the code, Python code gains the following advantages:

- First, you'll never miss the beginning or ending code of a block like in other programming languages such as Java or C#.
- Second, the coding style is essentially uniform. If you have to maintain another developer's code, that code looks the same as yours.
- Third, the code is more readable and clearer in comparison with other programming languages.

Comments

The comments are as important as the code because they describe why a piece of code was written.

When the Python interpreter executes the code, it ignores the comments.

In Python, a single-line comment begins with a hash (#) symbol followed by the comment. For example:

```
# This is a single line comment in Python
```

Continuation of statements

Python uses a newline character to separate statements. It places each statement on one line.

However, a long statement can span multiple lines by using the backslash (\) character.

The following example illustrates how to use the backslash (\) character to continue a statement in the second line:

```
if (a == True) and (b == False) and \ (c == True):
    print("Continuation of statements")
```

Identifiers

Identifiers are names that identify variables, functions, modules, classes, and other objects in Python. The name of an identifier needs to begin with a letter or underscore (_). The following characters can be alphanumeric or underscore.

Python identifiers are case-sensitive. For example, the counter and Counter are different identifiers.

In addition, you cannot use Python keywords for naming identifiers.

Keywords

Some words have special meanings in Python. They are called keywords.

The following shows the list of keywords in Python:

False	***class***	***finally***	***is***	***return***	***None***
continue	***for***	***lambda***	***try***	***True***	***def***
from	***nonlocal***	***while***	***and***	***del***	***global***
not	***with***	***as***	***elif***	***if***	***or***
yield	***assert***	***else***	***import***	***pass***	***break***
except	***in***	***raise***			

Python is a growing and evolving language. So, its keywords will keep increasing and changing.

Python provides a special module for listing its keywords called keyword.

To find the current keyword list, you use the following code:

```
import keyword
print(keyword. kwlist )
```

String literals

Python uses single quotes ('), double quotes ("), triple single quotes (''') and triple-double quotes (""") to denote a string literal.

The string literal needs to be surrounded with the same type of quotes. For example, if you use a single quote to start a string literal, you need to use the same single quote to end it.

The following shows some examples of string literals:

```
s = 'This is a string'
print(s)
s = "Another string using double quotes"
print(s)
s = ''' string can span
      multiple line '''
print(s)
```

6. SYSTEM ARCHITECTURE & DESIGN

6.1.SYSTEM ARCHITECTURE

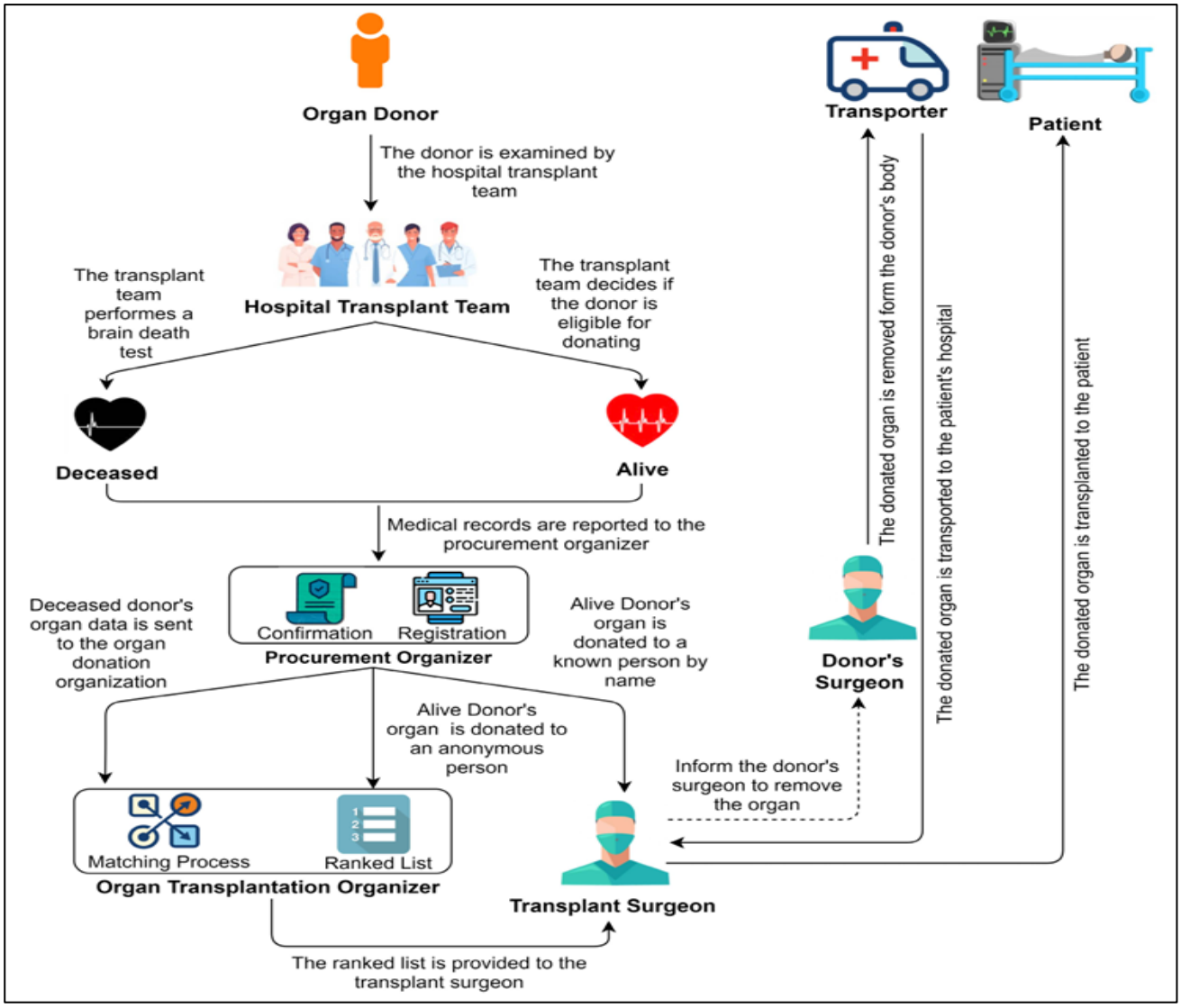


Fig.6.1 ***Organ donation and transplantation flow chart***

The image illustrates the **organ donation and transplantation process**, detailing how organs from deceased or living donors are evaluated, matched, and transplanted to recipients. Here's a summary of the key steps shown in the flowchart:

1. Organ Donor Evaluation

- The **Hospital Transplant Team** examines the donor to determine donation eligibility.

2. Donor Type

- If **Deceased**:
  - A brain death test is performed.
  - Data is sent to the **Procurement Organizer**.
  - The data is further sent to the **Organ Transplantation Organizer**.

- A **matching process** is carried out.
- A **ranked list** is created and sent to the **Transplant Surgeon**.

- If **Alive**:
  - Medical records are submitted to the **Procurement Organizer**.
  - The organ may be donated to a **known person by name** or to an **anonymous recipient**.
  - The **Transplant Surgeon** is informed.
  - The **Donor's Surgeon** is instructed to remove the organ.

3. Organ Transfer

- The **Transporter** moves the donated organ from the donor's hospital to the recipient's hospital.

4. Organ Transplantation

- The **Transplant Surgeon** transplants the organ into the **patient**.

  This system ensures proper donor evaluation, ethical procurement, and a structured organ allocation process.

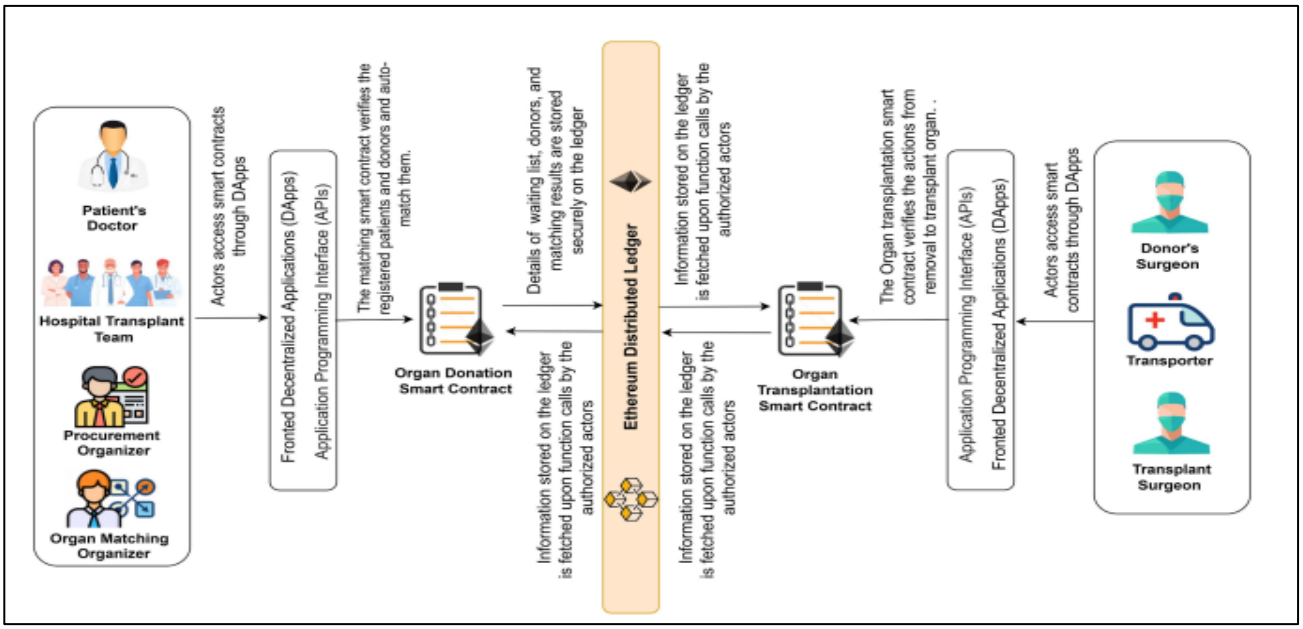


Fig.6.2 ***A High-Level System Architecture of the Proposed Blockchain-Based Solution for Organ Donation and Transplantation.***

Left Side (Donor & Matching Setup)

1. **Actors Involved:**
   - Patient's Doctor
   - Hospital Transplant Team
   - Procurement Organizer
   - Organ Matching Organizer

2. **Process:**
    - These actors **access the blockchain via DApps** and **APIs**.
    - They interact with the **Organ Donation Smart Contract**, which:
        - Verifies **registered patients and donors**.
        - Matches them using a **matching algorithm**.
        - Stores waiting lists, donor info, and match results securely on the blockchain.

Middle (Blockchain Core)

- **Ethereum Distributed Ledger**:
    - Acts as the secure and decentralized data store.
    - Only authorized actors can **read/write data via smart contracts**.
    - Ensures **immutability**, **transparency**, and **trustless coordination**.

Right Side (Organ Transplant Execution)

1. **Actors Involved:**
    - Donor's Surgeon
    - Transporter
    - Transplant Surgeon
2. **Process:**
    - They also interact with the system via **DApps/APIs**.
    - The **Organ Transplantation Smart Contract**:
        - Verifies the **details from the donation contract**.
        - Authorizes the process to **remove, transport, and transplant** the organ.

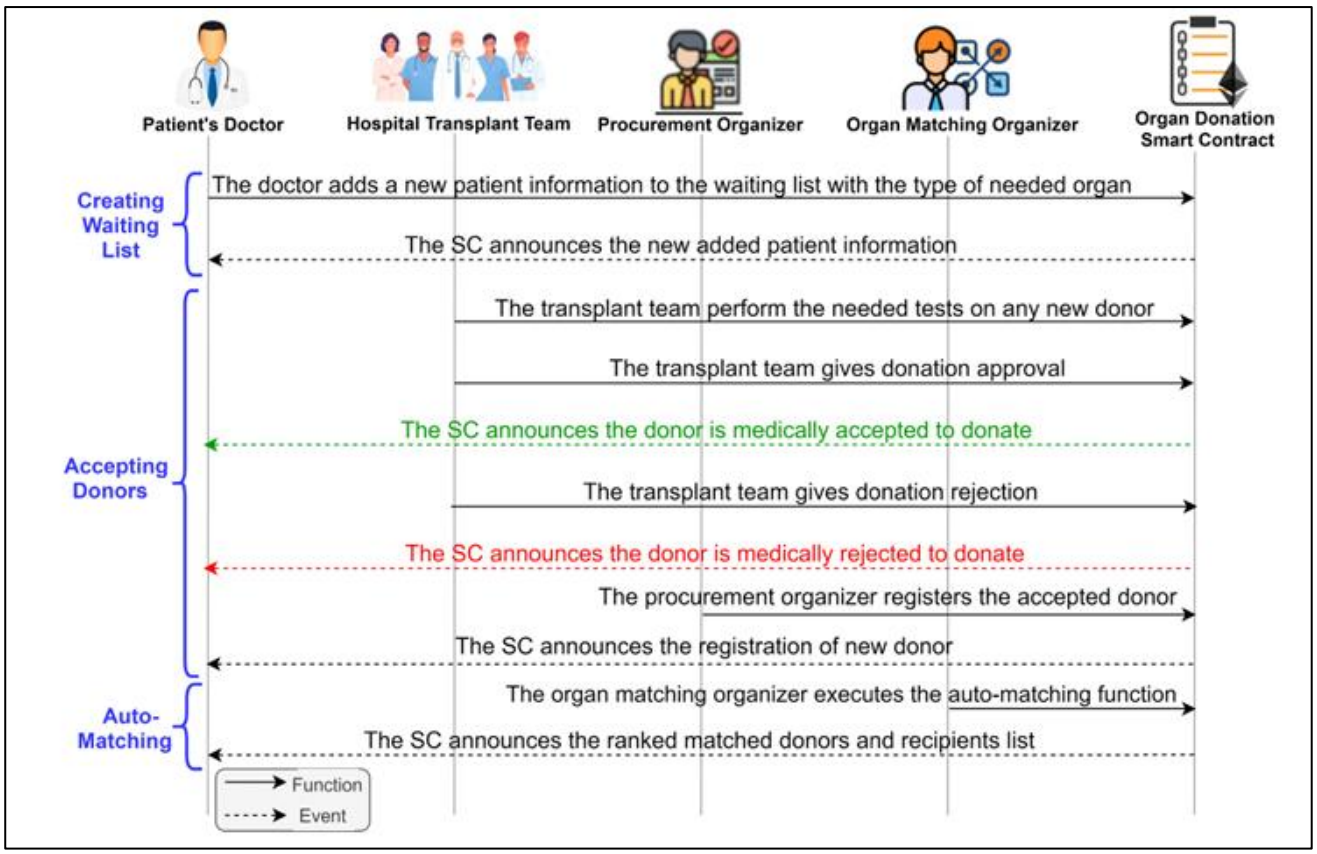


Fig.6.3 ***Sequence diagram showing interactions among the participants and organ donation smart contract***

Organ donation is a life-saving process but often suffers from inefficiencies, lack of transparency, and data tampering risks. This project proposes a blockchain-powered solution that leverages smart contracts to manage and automate the organ donation workflow—from donor registration to patient transplantation—with full traceability, decentralization, and security.

System Overview

The system comprises multiple stakeholders:

- **Organ Donor (Alive or Deceased)**
- **Hospital Transplant Team**
- **Procurement Organizer**
- **Organ Matching Organizer**
- **Patient's Doctor**
- **Transplant Surgeon & Donor's Surgeon**
- **Transporter**

The process is divided into **three major phases**, each automated using blockchain smart contracts:

1. Traditional Organ Donation Workflow (Diagram 1)

- A potential **organ donor** is examined by the **hospital transplant team**.
- If **deceased**, brain death is confirmed and the data is sent to the **organ donation organization**.
- If **alive**, the organ may be donated to a known recipient or anonymously.
- Data is sent to the **procurement organizer**, then forwarded to the **organ transplantation organizer** for matching.
- A **ranked list** of recipients is provided to the **transplant surgeon**, and the organ is extracted and delivered to the recipient via **transporters**.

2. Blockchain-Integrated Smart Contract Workflow (Diagram 2)

- All actors access the blockchain via **DApps (Decentralized Applications)** and **APIs**.
- The **Ethereum Distributed Ledger** stores:
    - Waiting list data
    - Donor eligibility
    - Matching results
- Two key smart contracts are used:
    - **Organ Donation Smart Contract**: Handles donor/patient registration and verification.
    - **Organ Transplantation Smart Contract**: Controls matching, validation, and organ removal authorization.

3. Step-by-Step Interaction Flow (Diagram 3)

Creating Waiting List:

- The **doctor** adds a patient's details (including required organ) via a DApp.
- The **smart contract (SC)** records and announces this new data.

Accepting Donors:

- **Transplant team** tests donors and decides on eligibility.
- If accepted, the **SC emits a confirmation event**.
- If rejected, a rejection event is triggered.
- The **procurement organizer** registers the eligible donor.

Auto-Matching:

- The **organ matching organizer** executes an auto-matching function.
- The SC generates and announces a **ranked list** of donor-recipient matches.

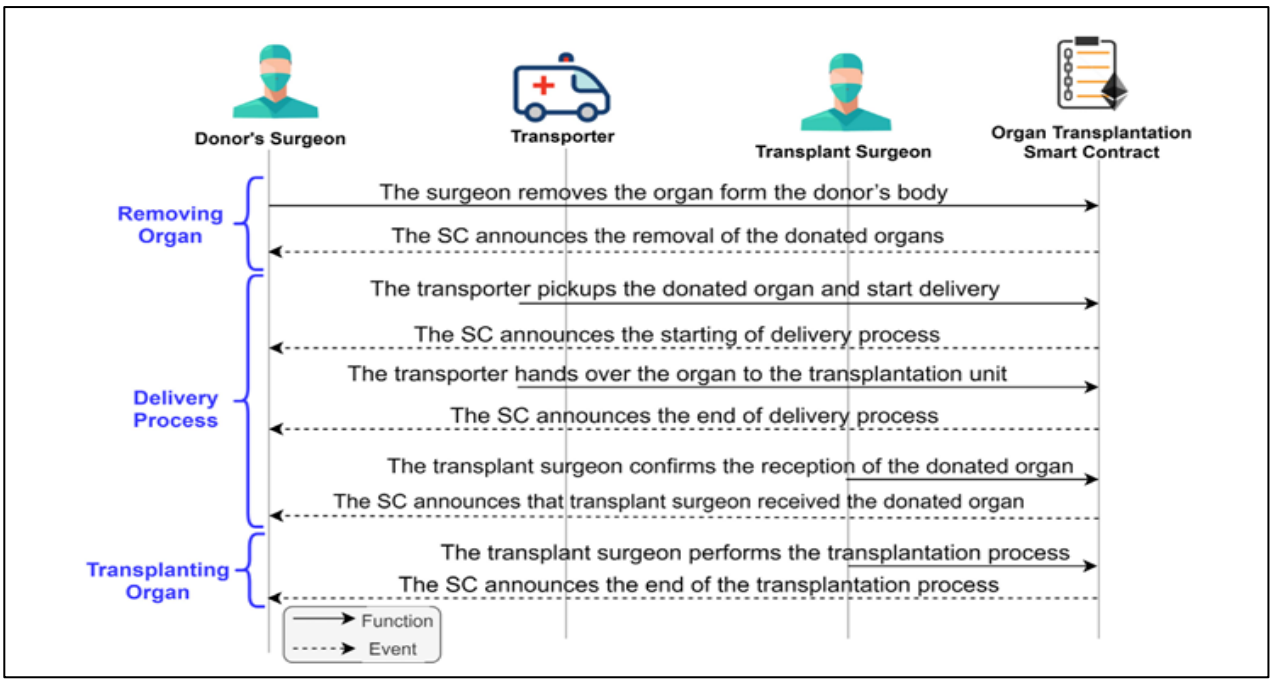


Fig.6.4 ***Sequence diagram showing interactions among the participants and organ transplantation smart contract***

Once a suitable organ donor and recipient have been matched, the next phase includes the **removal**, **delivery**, and **transplantation** of the organ. This phase is crucial and time-sensitive. It is fully monitored and logged via the **Organ Transplantation Smart Contract** to ensure transparency, accountability, and traceability.

1. Removing Organ

- The **donor's surgeon** removes the organ from the donor's body.
- A **smart contract (SC)** event is triggered to announce that the organ has been removed.
- This serves as an official timestamped record and notifies the transporter.

2. Delivery Process

- The **transporter** collects the organ and starts delivery.
- The SC logs the **start of the delivery process**, including geolocation and timestamp.
- Upon reaching the hospital, the **organ is handed over to the transplantation unit**.
- The SC confirms **end of delivery**, ensuring secure chain-of-custody.

3. Transplanting Organ

- The **transplant surgeon** receives and verifies the organ.
- A smart contract event announces that the organ has been received.
- The **surgeon then performs the transplantation surgery**.
- Upon completion, the SC logs the **end of transplantation**, completing the transaction lifecycle.

6.2.SYSTEM DESIGN

DATA FLOW DIAGRAM:

1. The DFD is also called a bubble chart. It is a simple graphical formalism used to represent a system in terms of input data, the processing carried out on this data, and the output generated by the system.
2. The data flow diagram (DFD) is an important modeling tool. It models system components, including the system process, data used by the process, external entities interacting with the system, and the information flows.
3. DFD shows how information moves through the system and how it is modified through transformations. It depicts information flow and the transformations applied from input to output.
4. DFD may represent a system at any level of abstraction. It can be partitioned into levels to show increasing information flow and functional detail.

UML DIAGRAMS:

UML (Unified Modeling Language) is a standardized general-purpose modeling language in object-oriented software engineering. Managed by the Object Management Group, UML provides a common language for creating object-oriented software models. It includes a Meta-model and a notation.

UML is essential for specifying, visualizing, constructing, and documenting software artifacts, business modeling, and other non-software systems. It incorporates best engineering practices proven effective in modeling large, complex systems.

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

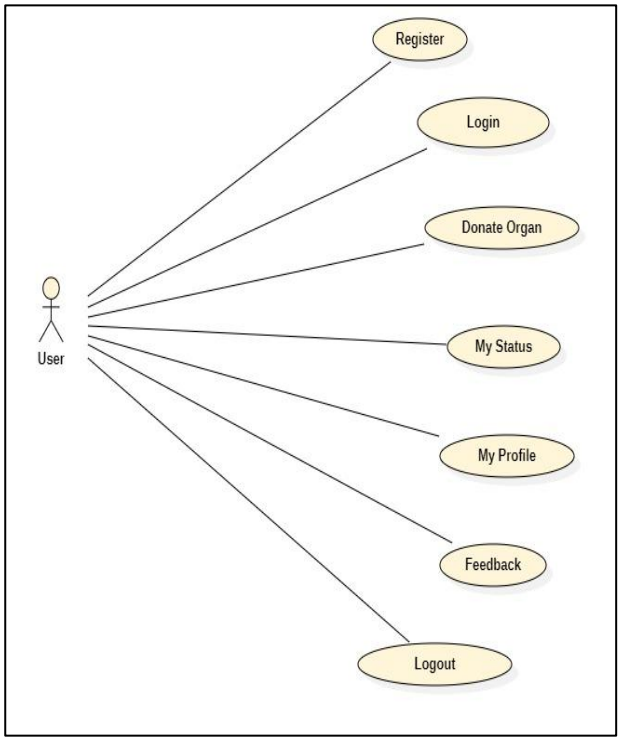


Fig.6.5 ***UseCaseDiagram(User)***

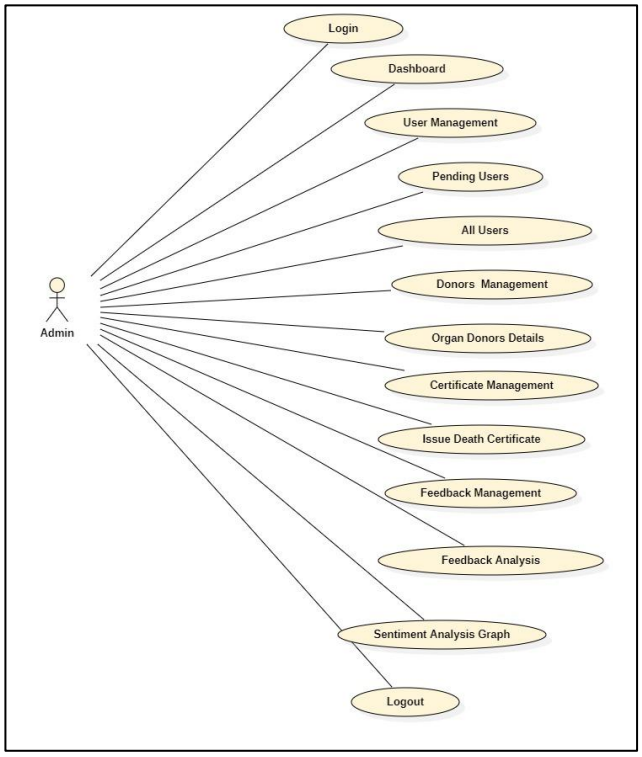


Fig.6.6 ***UseCaseDiagram(Admin)***

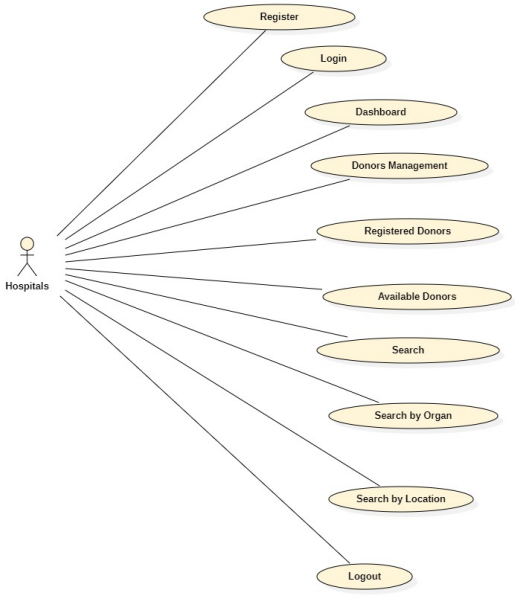


Fig.6.7 ***UseCaseDiagram(Hospital)***

CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

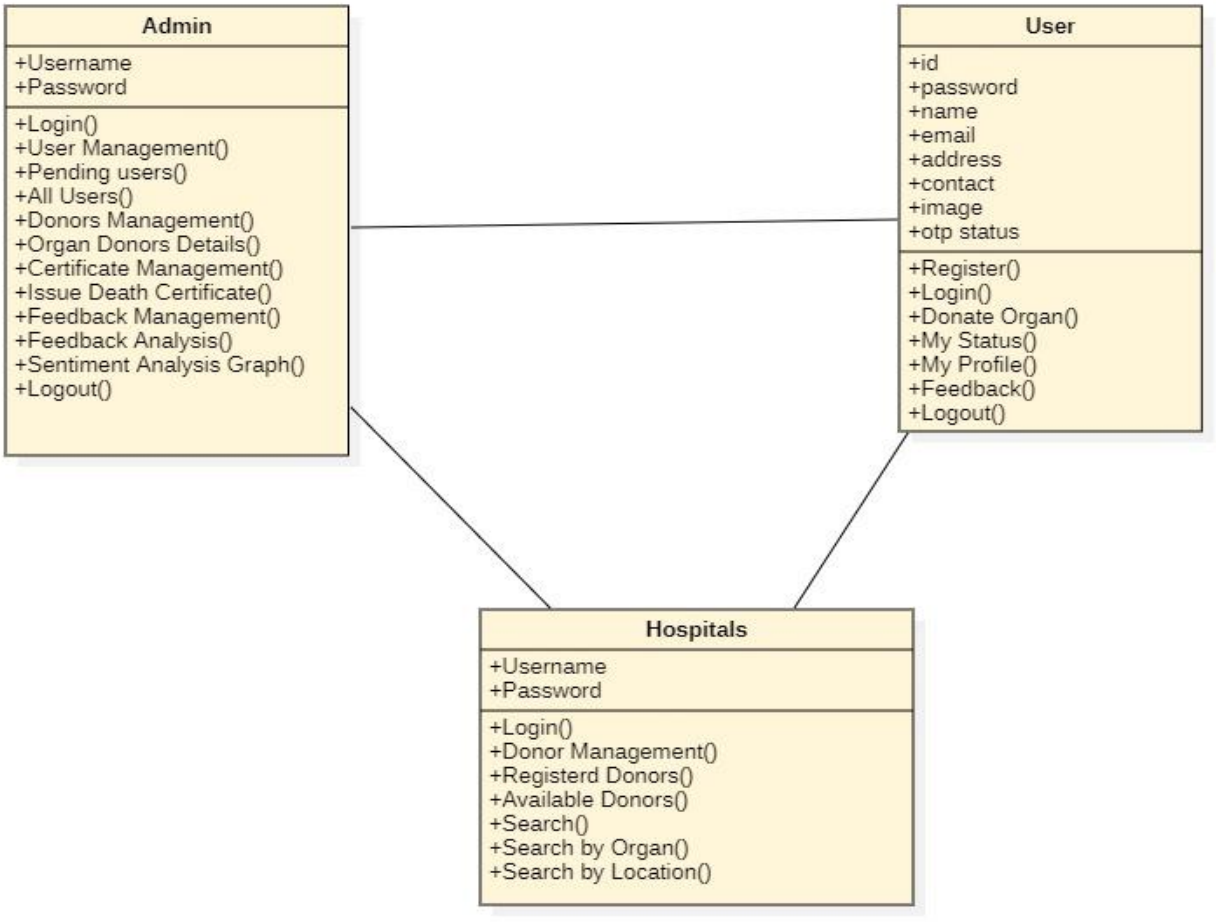


Fig.6.8 ***Class Diagram***

SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

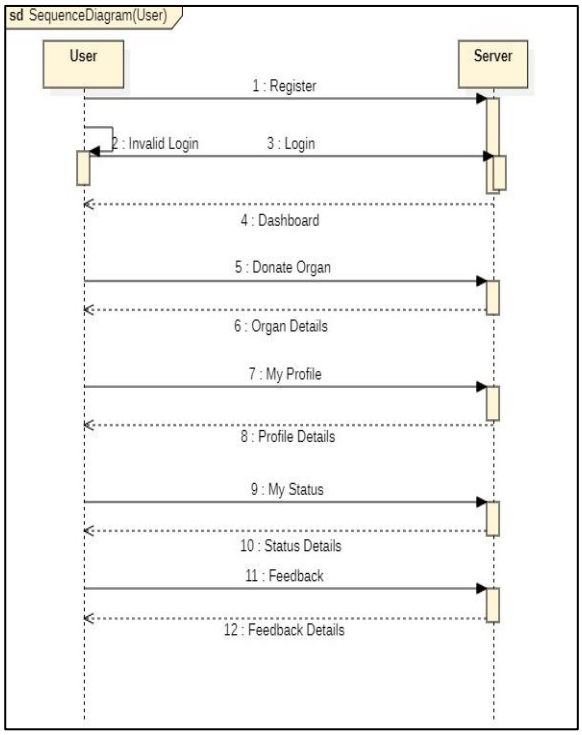


Fig.6.9 ***SequenceDiagram(User)***

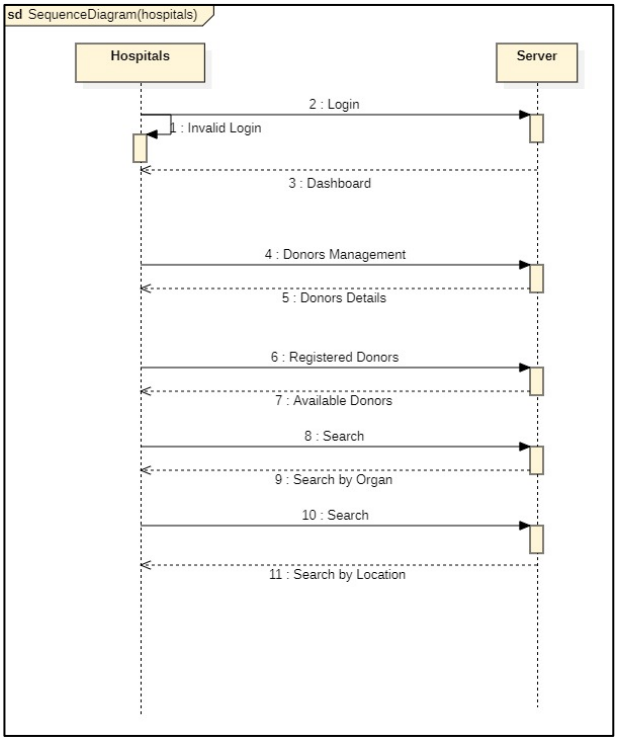


Fig.6.10 ***SequenceDiagram(Hospitals)***

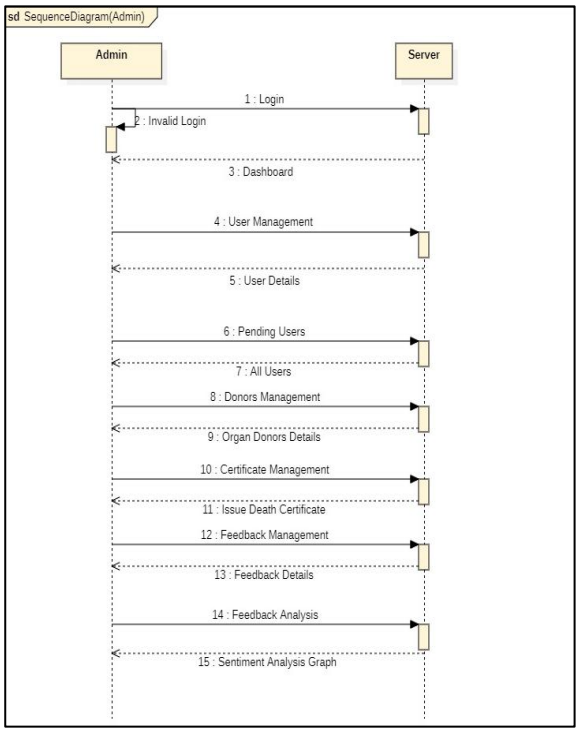


Fig.6.11 ***Sequence Diagram(Admin)***

ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

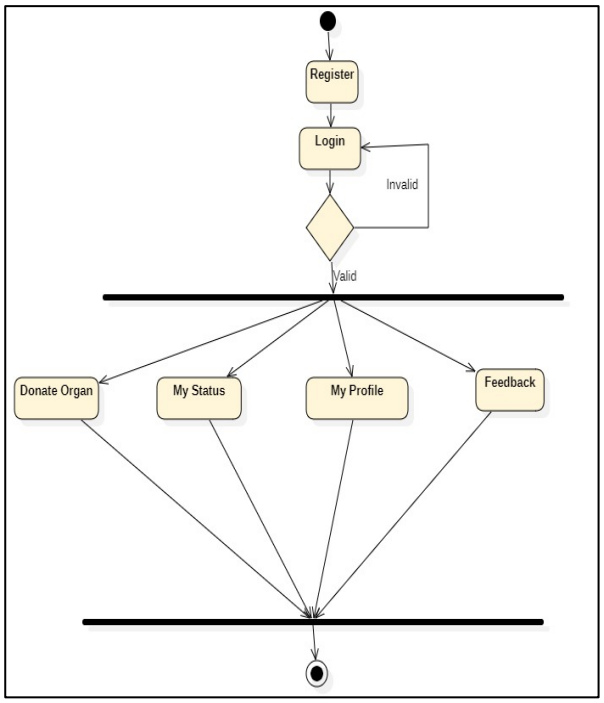


Fig6.12 ***Activity Diagram(User)***

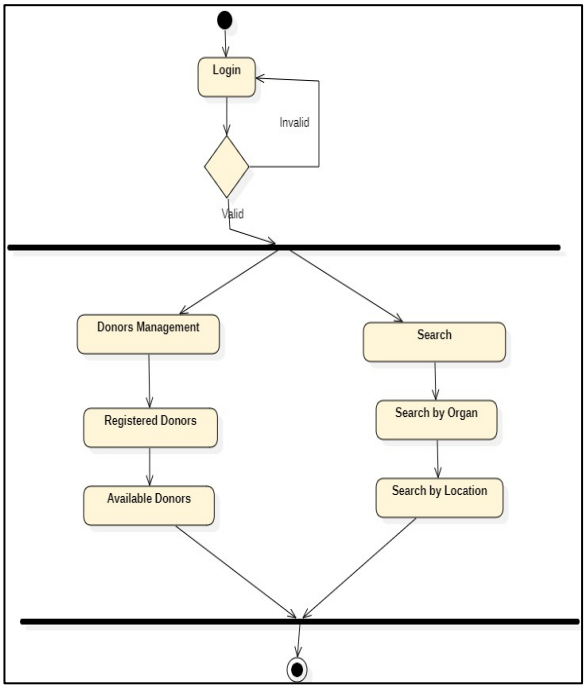


Fig.6.13 ***Activity Diagram(Hospital)***

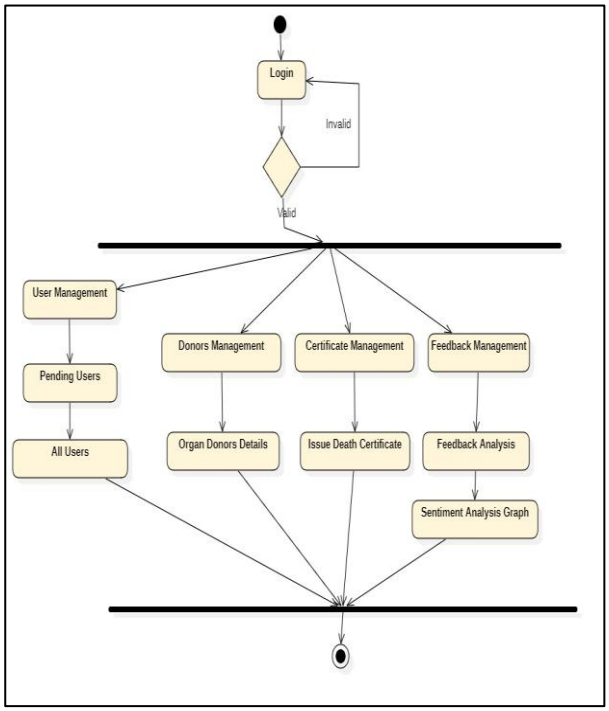


Fig.6.14 ***Activity Diagram(Admin)***

DEPLOYMENT DIAGRAM:

Deployment Diagram is a type of diagram that specifies the physical hardware on which the software system will execute. It also determines how the software is deployed on the underlying hardware. It maps software pieces of a system to the device that are going to execute it.

The deployment diagram maps the software architecture created in design to the physical system architecture that executes it. In distributed systems, it models the distribution of the software across the physical nodes. The software systems are manifested using various artifacts, and then they are mapped to the execution environment that is going to execute the software such as nodes. Many nodes are involved in the deployment diagram; hence, the relation between them is represented using communication paths.

There are two forms of a deployment diagram.

- Descriptor form
  - It contains nodes, the relationship between nodes and artifacts.
- Instance form
  - It contains node instance, the relationship between node instances and artifact instance.
  - An underlined name represents node instances.

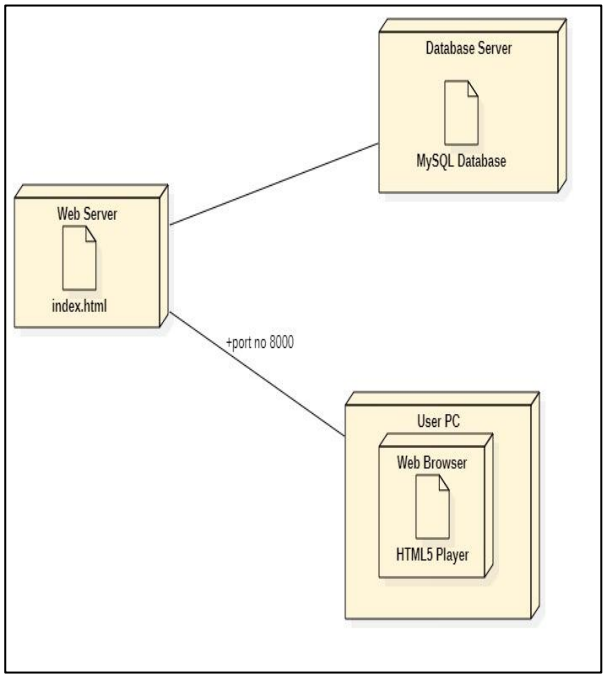


Fig.6.15 ***DeploymentDiagram1***

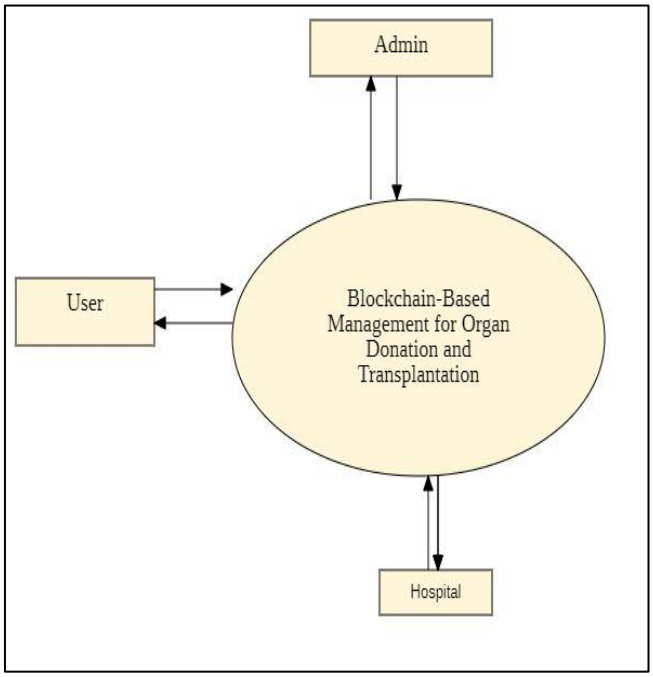
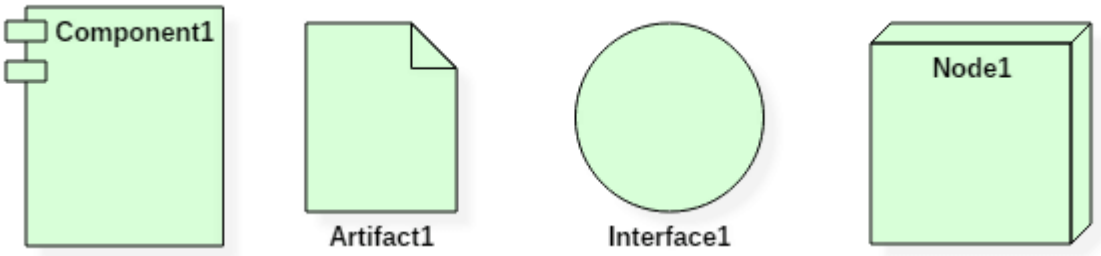


Fig.6.16 ***DeploymentDiagram2***

Purpose of a deployment diagram

Deployment diagrams are used with the sole purpose of describing how software is deployed into the hardware system. It visualizes how software interacts with the hardware to execute the complete functionality. It is used to describe software to hardware interaction and vice versa.

Deployment Diagram Symbol and notations



6.3.MODULES

A. NON-BLOCKCHAIN-BASED SOLUTIONS FOR ORGAN DONATION MANAGEMENT

In non-blockchain-based processes, various approaches and tools are utilized to come up with solutions that enhance organ donation, transplantation management, and the matching process. The authors in developed a multi-agent software platform to represent the information workflow model among donor hospitals, regulators, and recipient hospitals. This platform optimizes the pre-transplantation tasks, which can improve the process efficiency. In addition, it allows storing potential donor information and improves direct communication among all participants in the organ transplantation process. An information workflow was simulated using the developed platform, and it was estimated that the saved time might be between three to five hours.

B. BLOCKCHAIN-BASED SOLUTIONS FOR ORGAN DONATION MANAGEMENT

In a blockchain-based kidney donation system named ‘‘Kidner’’ has been proposed. It offers a kidney-pair donation module instead of the traditional kidney waiting list, which is already in use. For example, when someone wishes to donate his/her kidney to a family member but their kidney is incompatible with the person they want to donate to, the system matches the donor’s kidney to another patient who also has an inconsistent donor’s kidney.

C. PRIVATE PERMISSIONED ETHEREUM NETWORK

Private blockchains provide enhanced security and privacy where the transactions and data are not accessible to the public and only viewed by authorized entities. Enterprises can use the Ethereum blockchain to develop their own private-permissioned blockchain to improve privacy, security, and confidentiality. In general, details of donated organ transplantation are strictly confidential. These details include the patients’ health records and family histories; therefore, a private permissioned Ethereum blockchain is ideal for such an implementation.

D. BLOCKCHAIN INTEGRATION

The blockchain network is the backbone of our proposed solution. It serves as the basis for recording transactions and events permanently to ensure accountability and data provenance. The developed smart contracts must be deployed on the blockchain to ensure they are accessible at all times. However, it would not be ideal to deploy them on the main network during the testing phase.

E. PARTICIPANTS INTERACTIONS

The interaction among different participants within the matching smart contract, which can be divided into three phases.

Phase 1:- It begins with creating a waiting list, in which an authorized doctor will add a new patient to the waiting list. The doctor will record the patient's ID, age, BMI, and blood type.

Phase 2:- It is fulfilled by receiving donors who have given their consent to donate their organs. Only an authorized transplant team member will run the test approval function, and an event will be sent immediately. After that, the procurement organizer is ready to evaluate and register the donor. To make the announcement that a new donor has been registered, an event will be triggered.

Phase 3:- The auto-matching between the donor and recipient is handled by the organ transplantation organizer. The auto-matching process is done based on the age range, blood type, and BMI range obtained from the donor. Finally, a matched patient ranked list is announced.

7. IMPLEMENTATION

ADMINAPP:

views.py

```
from django.shortcuts import render,redirect
from django.contrib import messages
from userapp.models import *
from userapp.blockchainalgo import Organ_blockchain
from django.core.paginator import Paginator
# Create your views here.
def main_admin(request):
    if request.method == "POST":
        name = request.POST.get("name")
        password = request.POST.get("password")
        if name == "admin" and password == "admin":
            messages.success(request,"admin login successfully")
            return redirect("admin_dashboard")
        else:
            messages.error(request,"invalid admin name and password")
    return render(request,'main/main-admin.html')
def admin_dashboard(request):
    users = RegistrationModels.objects.all().count()
    applications = OrganapplicationModels.objects.all().count()
    pending = OrganapplicationModels.objects.filter(status = "pending").count()
    fit = OrganapplicationModels.objects.filter(status = "Fit").count()
    return render(request,'admin/admin-dashboard.html',{"users":users, "applications":applications,
"pending":pending, "fit":fit})
def admin_pendinguser(request):
    pending = OrganapplicationModels.objects.filter(status = "pending")
    paginator = Paginator(pending,4)
    pageno = request.GET.get('page')
    pending = paginator.get_page(pageno)
    return render(request,'admin/admin-pendinguser.html',{"data":pending})
def admin_alluser(request):
```

```
    pending = OrganapplicationModels.objects.all()
    paginator = Paginator(pending,4)
    pageno = request.GET.get('page')
    pending = paginator.get_page(pageno)
    return render(request,'admin/admin-alluser.html',{"user":pending})
def admin_organdonor(request):
    organdonor = OrganapplicationModels.objects.all()
    paginator = Paginator(organdonor,4)
    pageno = request.GET.get('page')
    organdonor = paginator.get_page(pageno)
    return render(request,'admin/admin-organdonor.html',{"organdonor":organdonor})
def admin_deathcertificate(request):
    deathcertificate = OrganapplicationModels.objects.filter(status = "Fit", issued_status = "Not
issued" )
    paginator = Paginator(deathcertificate,4)
    pageno = request.GET.get('page')
    deathcertificate = paginator.get_page(pageno)
    return render(request,'admin/admin-deathcertificate.html',{"deathcertificate":deathcertificate})
def admin_feedbackanalysis(request):
    feedbackanalysis = feedbackM.objects.all()
    paginator = Paginator(feedbackanalysis,4)
    pageno = request.GET.get('page')
    feedbackanalysis = paginator.get_page(pageno)
    return render(request,'admin/admin-feedbackanalysis.html',{'data':feedbackanalysis})
def admin_sentimentanalysis(request):
    fit = OrganapplicationModels.objects.filter(status = 'Fit').count()
    notissued = OrganapplicationModels.objects.filter(issued_status = 'Not issued').count()
    issued = OrganapplicationModels.objects.filter(issued_status = 'issued').count()
    unfit = OrganapplicationModels.objects.filter(status = 'Un Fit').count()
    conductingtest = OrganapplicationModels.objects.filter(status = 'Conducting Test').count()
    return render(request,'admin/admin-sentimentanalysis.html',{'f':fit,'n':notissued, 'i':issued, 'u':unfit,
'c':conductingtest})
def issued(request,organ_id):
    a = OrganapplicationModels.objects.get(organ_id = organ_id)
```

```
    a.issued_status = 'issued'
    a.save(update_fields= ['issued_status'] )
    a.save()
    return redirect('admin_deathcertificate')
def reject(request,organ_id):
    a = OrganapplicationModels.objects.get(organ_id = organ_id)
    a.issued_status = 'rejected'
    a.save(update_fields= ['issued_status'])
    a.save()
    return redirect('admin_deathcertificate')
def verify_application(request,id):
    organ = OrganapplicationModels.objects.get(pk = id)
    key = "asdfhalskdhfasdhfkajd1234asdf12341234"
    initial_block = Organ_blockchain (key,[str(organ.Name),str(organ.Phone)])
    second_block = Organ_blockchain(initial_block.block_hash,[str((organ.Disease))])
    third_block = Organ_blockchain(second_block.block_hash,[str((organ.Donate_Organ))])
    if (organ.block1 == initial_block.block_hash and organ.block2 == second_block.block_hash) and
(organ.block3 == third_block.block_hash):
        a=OrganapplicationModels.objects.get(pk=id)
        a.verification_status = 'valid'
        a.save(update_fields= ['verification_status'])
        messages.success(request,"The data is valid")
        return render(request,"admin/admin-verifyapplication.html",{'i':a})
    else:
        a=OrganapplicationModels.objects.get(pk=id)
        a.verification_status = 'invalid'
        a.save(update_fields = ['verification_status'])
        messages.info(request,"The data has been tempared")
    return render(request,"admin/admin-verifyapplication.html",{'i':a})
```

HOSPITAL APP:

views.py

```
from django.shortcuts import render,redirect
from django.contrib import messages
from userapp.models import *
from hospitalapp.models import *
from django.core.paginator import Paginator
# Create your views here.
def main_hospital(request):
    if request.method == "POST":
        name = request.POST.get("name")
        password = request.POST.get("password")
        if name == "hospital" and password == "hospital":
            messages.success(request,"hospital login successfully")
            return redirect("hospital_dasboard")
        else:
            messages.error(request,"invalid hospital name and password")
    return render(request,'main/main-hospital.html')
def hospital_dasboard(request):
    register = OrganapplicationModels.objects.all().count()
    fit = OrganapplicationModels.objects.filter(status = "Fit").count()
    test = OrganapplicationModels.objects.filter(status = "Conducting Test").count()
    unfit = OrganapplicationModels.objects.filter(status = "Un Fit").count()
    return render(request,'hospital/hospital-dashboard.html',{"register":register, "fit":fit, "test":test,
"unfit":unfit})
def hospital_registereddonors(request):
    register = OrganapplicationModels.objects.all()
    paginator = Paginator(register,4)
    pageno = request.GET.get('page')
    register = paginator.get_page(pageno)
    return render(request,'hospital/hospital-registereddonors.html',{"register":register})
def hospital_availabledonor(request):
    availabledonor = OrganapplicationModels.objects.filter(issued_status = "issued" )
```

```
    paginator = Paginator(availabledonor,4)
    pageno = request.GET.get('page')
    availabledonor = paginator.get_page(pageno)
    return render(request,'hospital/hospital-availabledonor.html',{'availabledonor':availabledonor})
def hospital_searchbylocation(request):
    location=None
    try:
        if request.method == "POST":
            name = request.POST.get("loc")
            location = OrganapplicationModels.objects.get(City=name)
    except:
        messages.info(request,"Not Found")
    return render(request,'hospital/hospital-searchbylocation.html',{'loc':location})
def hospital_searchbyorgan(request):
    organ = None
    try:
        if request.method == "POST":
            name = request.POST.get("organ")
            organ = OrganapplicationModels.objects.get(Donate_Organ = name)
    except:
        messages.info(request,"Not Found")
    return render(request,'hospital/hospital-searchbyorgan.html',{'organ':organ})
def hospital_view(request,organ_id):
    view = OrganapplicationModels.objects.get(organ_id = organ_id)
    if request.method == "POST":
        name = request.POST.get("name")
        gender = request.POST.get("gender")
        dob = request.POST.get("dob")
        donate_organ = request.POST.get("donateorgan")
        disease = request.POST.get("disease")
        email = request.POST.get("email")
        status = request.POST.get("test")
        view.Name = name
        view.Gender = gender
```

```
        view.Date_Of_Birth = dob
        view.Donate_Organ = donate_organ
        view.Disease = disease
        view.Email = email
        view.status = status
        view.save()
        HospitalregistratedModels.objects.create(
            Name = name,
            Gender = gender,
            Date_Of_Birth = dob,
            Donate_Organ = donate_organ,
            Disease =  disease,
            Email = email,
            status = status,
            )
        messages.success(request,"Application has send successfully")
    return render(request,'hospital/hospital-view.html',{"view":view})
def rejected_donors(request):
    donor = OrganapplicationModels.objects.filter(issued_status = "rejected")
    return render(request,'hospital/hospital-rejected.html',{'donor':donor})
```

USER APP:

views.py

```
from django.shortcuts import render,redirect
from userapp.models import *
from django.contrib import messages
from textblob import TextBlob
from userapp.blockchainalgo import Organ_blockchain
# Create your views here.
def main_registration(request):
    if request.method == "POST" and request.FILES['photo']:
        name = request.POST.get("name")
        father = request.POST.get("father")
        email = request.POST.get("email")
        phone = request.POST.get("phone")
        city = request.POST.get("city")
        dob = request.POST.get("date")
        photo = request.FILES["photo"]
        password = request.POST.get("password")
        gender = request.POST.get("gender")
        try:
            RegistrationModels.objects.get(email = email)
            messages.info(request,"email already existed")
            return redirect('main_registration')
        except:
            RegistrationModels.objects.create(
                name = name,
                father = father,
                email = email,
                phone = phone,
                city = city,
                dob = dob,
                photo = photo,
                password = password,
```

```
            gender = gender,
        )
        messages.success(request,"User Registered Sucessfully")
    return render(request,"main/main-registration.html")
def main_user(request):
    if request.method == "POST":
        emaill = request.POST.get("email")
        passwordd = request.POST.get("password")
        print(emaill,passwordd)
        try:
            user = RegistrationModels.objects.get(email = emaill,password = passwordd)
            request.session['user_id'] = user.user_id
            messages.success(request,'Login Sucessfully')
            return redirect("user_dashboard")
        except:
            messages.error(request,'Invalid email and password')
            return redirect("main_user")
    return render(request,"main/main-user.html")
def user_dashboard(request):
    users = RegistrationModels.objects.all().count()
    applications = OrganapplicationModels.objects.all().count()
    organ1 = request.session['user_id']
    organ = RegistrationModels.objects.get(user_id = organ1)
    organ2 = OrganapplicationModels.objects.filter(use = organ.user_id).count()
    return render(request,"user/user-dashboard.html",{"users":users, "applications":applications,
"organ2":organ2})
def user_donateorgan(request):
    user_id = request.session["user_id"]
    user_organ = RegistrationModels.objects.get(user_id = user_id)
    if request.method == "POST":
        name = request.POST.get("name")
        father = request.POST.get("father")
        email = request.POST.get("email")
        phone = request.POST.get("phone")
```

```
        city = request.POST.get("city")
        disease = request.POST.get("self")
        gender = request.POST.get("gender")
        dob = request.POST.get("date")
        organ = request.POST.getlist("value")
        try:
            OrganapplicationModels.objects.get(Email = email)
            messages.info(request,"email already exists")
            return redirect('user_donateorgan')
        except:
            blockchain = OrganapplicationModels.objects.create(
                    Name = name,
                    Father = father,
                    Email = email,
                    Phone = phone,
                    City = city,
                    Disease = disease,
                    Gender = gender,
                    Date_Of_Birth = dob,
                    Donate_Organ = organ,
                    use = user_organ
                )
            key = "asdfhalskdhfasdhfkajd1234asdf12341234"
            initial_block = Organ_blockchain (key,[str(blockchain.Name),str(blockchain.Phone)])
            blockchain.block1 = initial_block.block_hash
            second_block = Organ_blockchain(initial_block.block_hash,[str((blockchain.Disease))])
            blockchain.block2 = second_block.block_hash
            third_block =
Organ_blockchain(second_block.block_hash,[str((blockchain.Donate_Organ))])
            blockchain.block3 = third_block.block_hash
            blockchain.save()
            messages.success(request,"Organ Donation Application has send successfully")
    return render(request,"user/user-donateorgan.html",{"user_organ":user_organ})
def user_mystatus(request):
```

```
    id = request.session['user_id']
    user = RegistrationModels.objects.get(user_id=id)
    try:
        status  = OrganapplicationModels.objects.get(use = user)
    except:
        status = None
    return render(request,"user/user-mystatus.html",{"status":status})
def user_myprofile(request):
    id = request.session['user_id']
    user = RegistrationModels.objects.get(user_id = id)
    if request.method == "POST":
        name = request.POST.get('name')
        mobile = request.POST.get('mobile')
        father = request.POST.get('father')
        emails = request.POST.get('email')
        dob = request.POST.get('dob')
        gender = request.POST.get('gender')
        if len(request.FILES) !=0:
            image = request.FILES['photo']
            user.name = name
            user.phone = mobile
            user.father = father
            user.email = emails
            user.dob = dob
            user.gender = gender
            user.photo = image
            user.save()
        else:
            user.name = name
            user.phone = mobile
            user.father = father
            user.email = emails
            user.dob = dob
            user.gender = gender
```

```
            user.save()
        messages.success(request,"profile has been updated successfully")
    return render(request,"user/user-myprofile.html",{"data":user})
def user_myfeedback(request):
    user_id = request.session['user_id']
    try:
        feedback = RegistrationModels.objects.get(user_id = user_id)
        print(feedback,"tiger")
        if request.method == "POST":
            overall = request.POST.get('rating1')
            travelling = request.POST.get('rating2')
            suggestion = request.POST.get('commentText')
            analysis = TextBlob(suggestion)
            if not overall:
                messages.info(request,"Please give the overall ratings")
                return redirect(user_myfeedback)
            if not travelling:
                messages.info(request,"Please give the travelling ratings")
                return redirect(user_myfeedback)
            sentiment = ''
            if analysis.polarity >= 0.5:
                sentiment = 'VeryPositive'
            elif analysis.polarity > 0 and analysis.polarity < 0.5:
                sentiment = 'Positive'
            elif analysis.polarity < 0 and analysis.polarity >= -0.5:
                sentiment = 'Negative'
            elif analysis.polarity <= -0.5:
                sentiment = 'VeryNegative'
            else:
                sentiment = 'Neutral'
            try:
                feedbackM.objects.create(overall=overall,travelling=travelling,suggestion=suggestion,feed
back2=feedback,sentiment=sentiment)
            except:pass
```

```
            messages.success(request,"Feedback has been send successfully")
        except:
            feedback = None
        return render(request,"user/user-myfeedback.html",{'np':feedback})
    def calculater(request):
        res=''
        try:
            if request.method == "POST":
                num1 = eval(request.POST.get("num1"))
                num2 = eval(request.POST.get("num2"))
                res = request.POST.get("res")
                if res == "+":
                    res = num1+num2;
                elif res == "-":
                    res = num1-num2;
                elif res == "/":
                    res = num1/num2;
                elif res == "%":
                    res = num1%num2;
                elif res == "*":
                    res = num1*num2;
                    print(res,"tiger")
        except:
            messages.info(request,"Invalid number")
        return render(request,"user/calculater.html",{'res':res})
```

manage.py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys
def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'donate_organ.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)
if __name__ == '__main__':
    main()
```

8. SCREENSHOTS

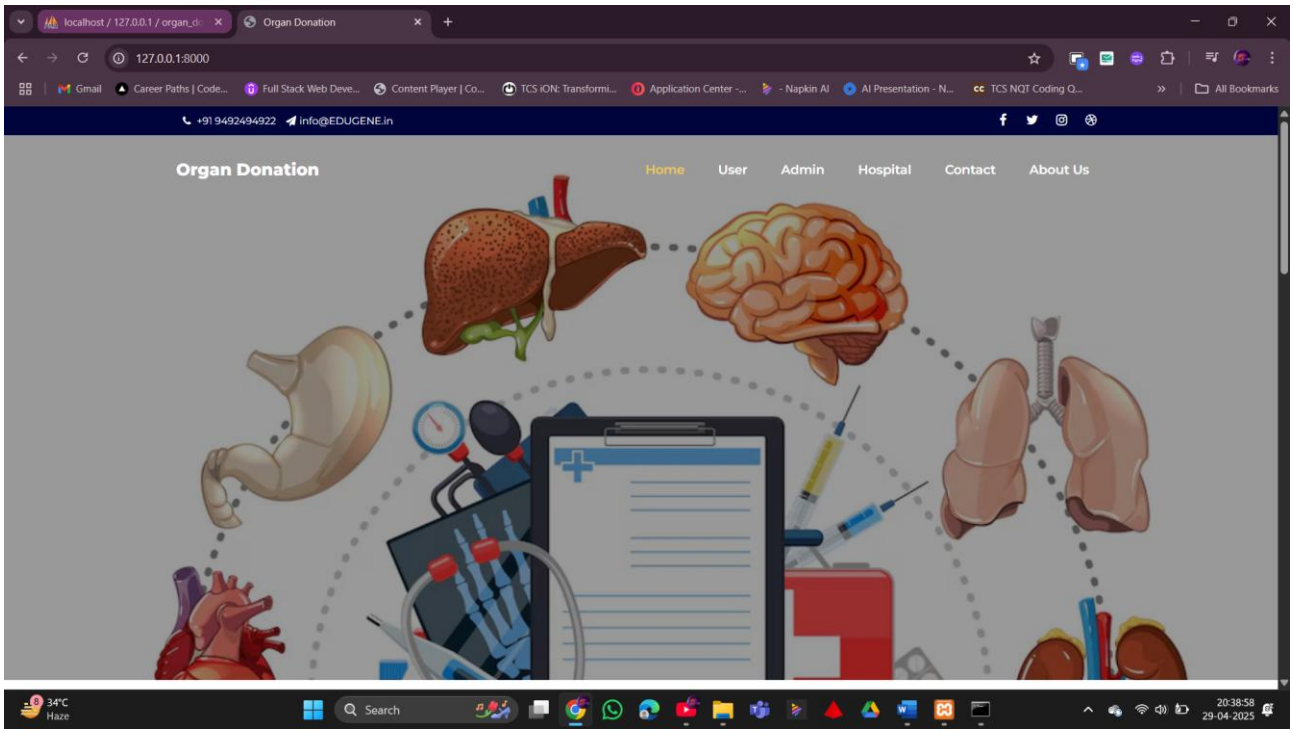


Fig.8.1 *landing page 1*

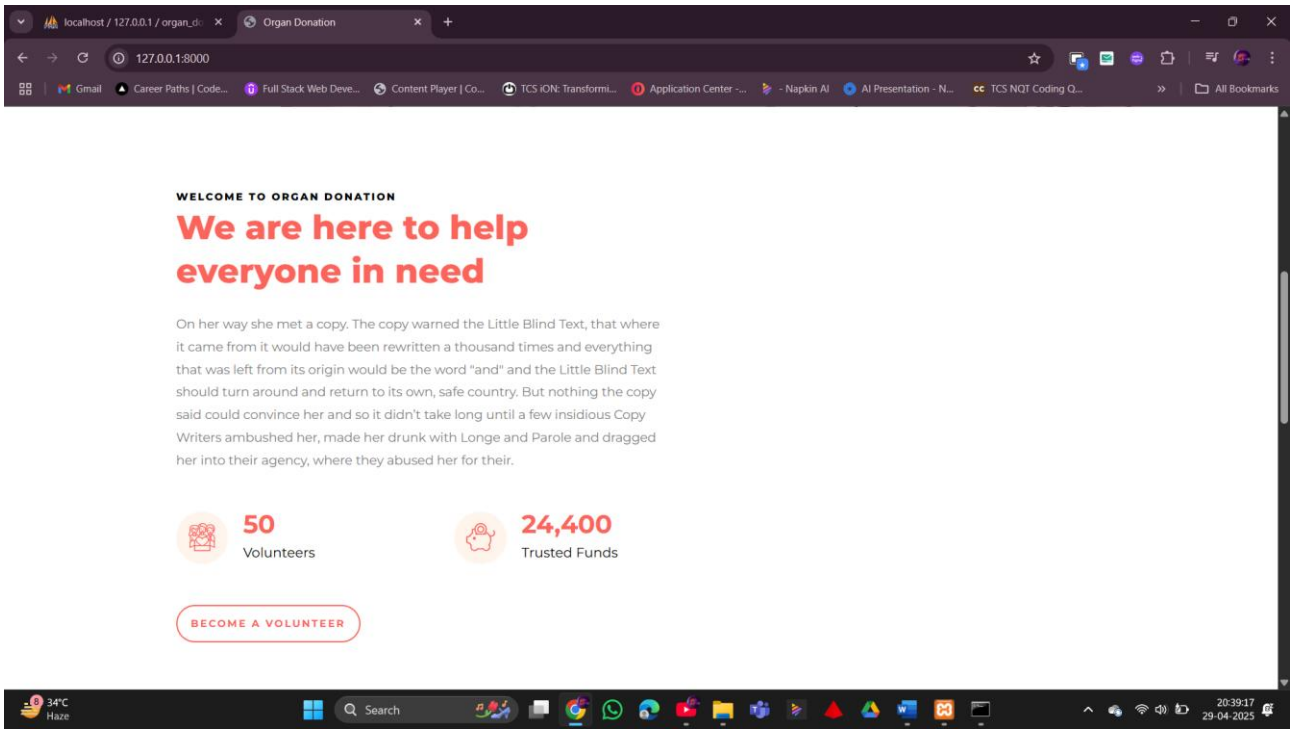


Fig.8.2 *landing page 2*

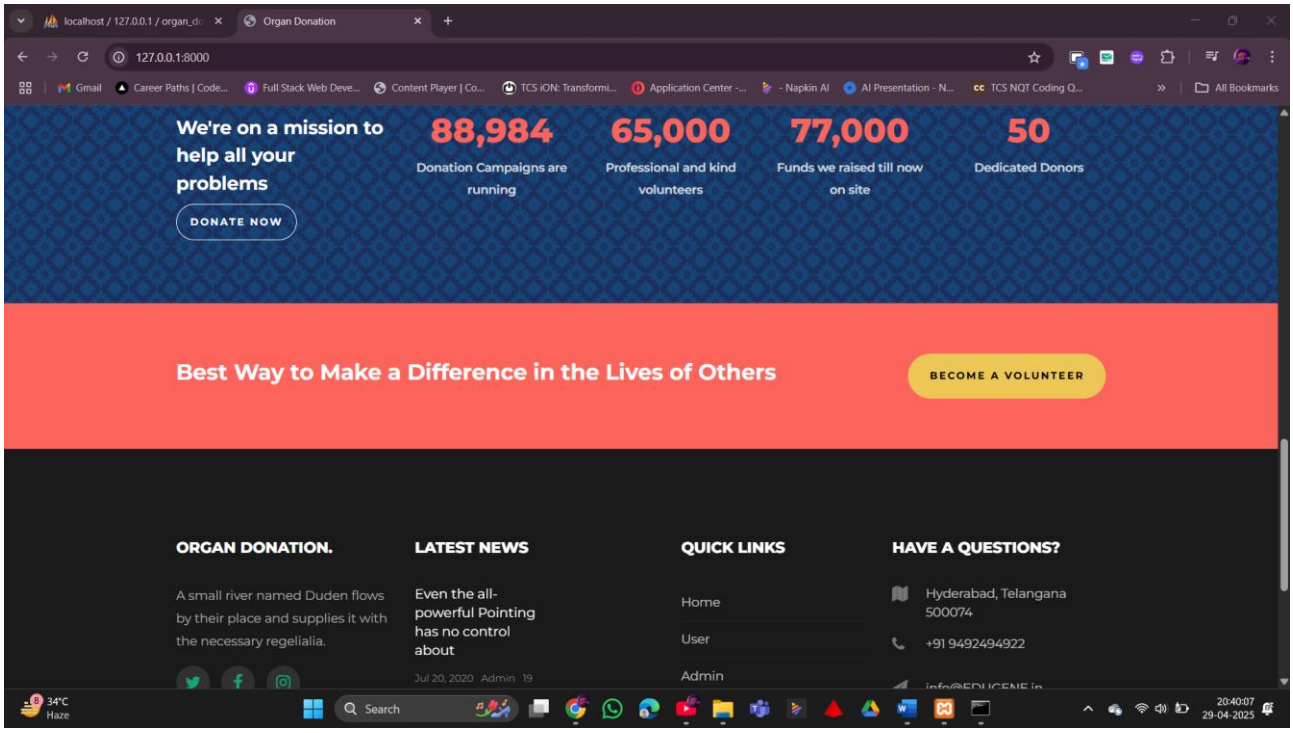


Fig.8.3 *Footer*

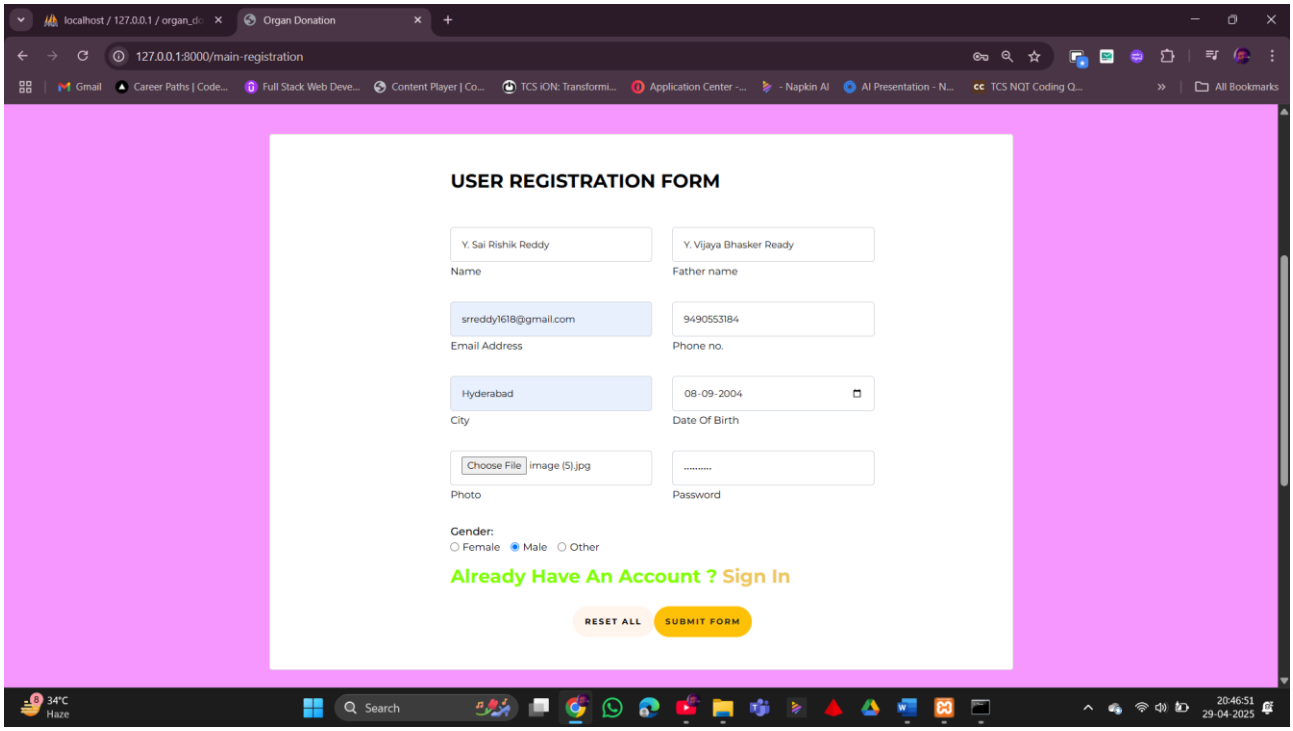


Fig.8.4 *User Registration Form*

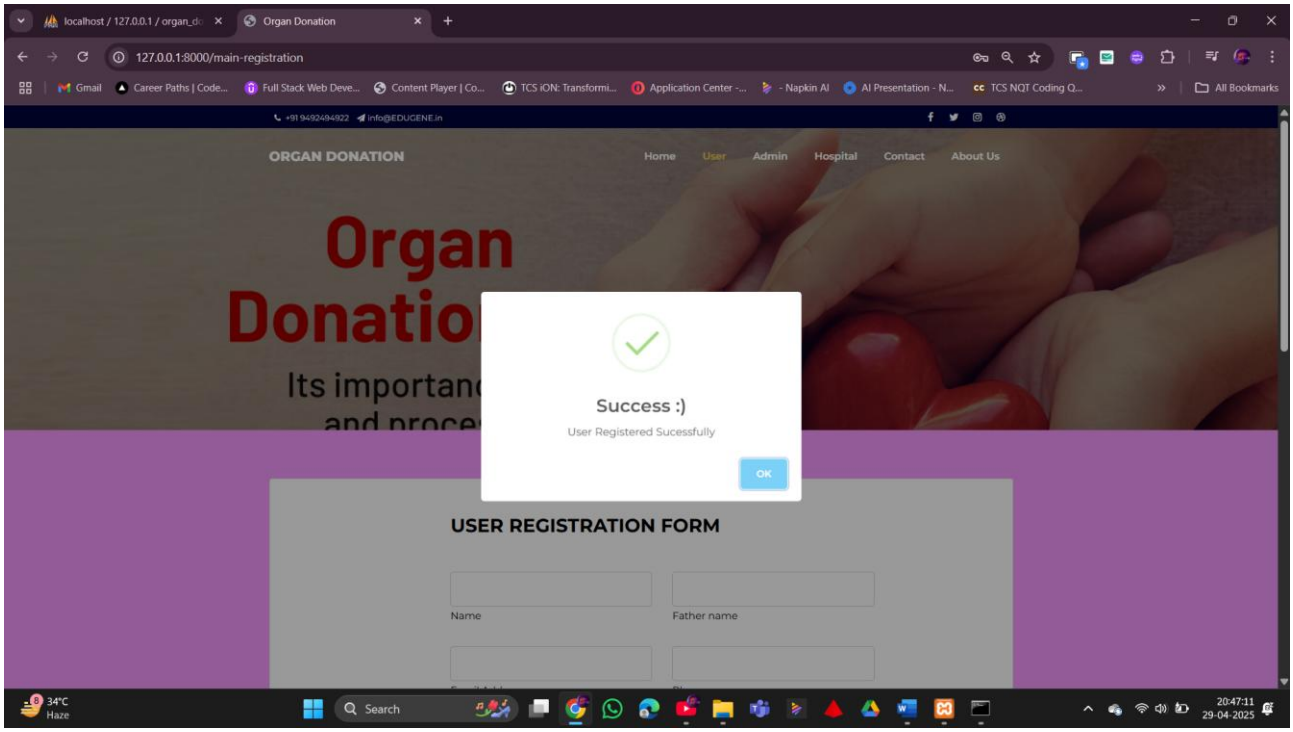


Fig.8.5 *User Registration Successful*

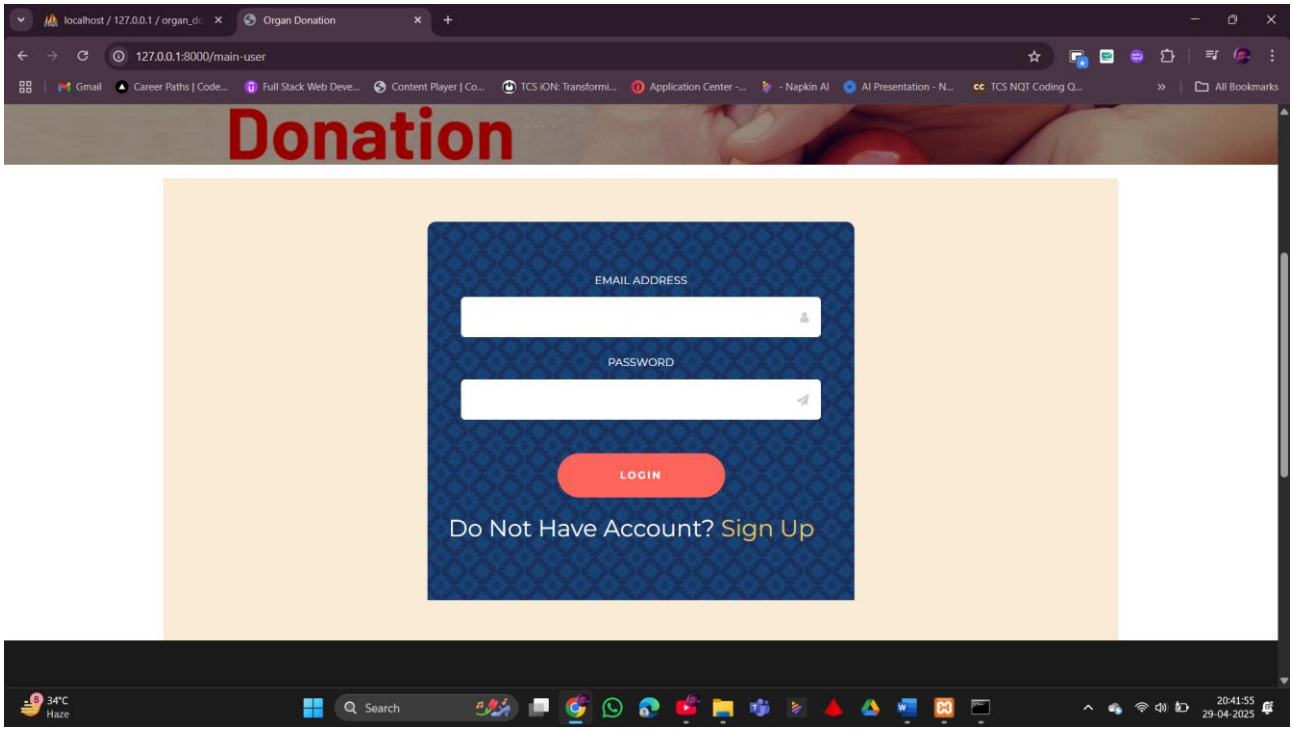


Fig.8.6 *User Login*

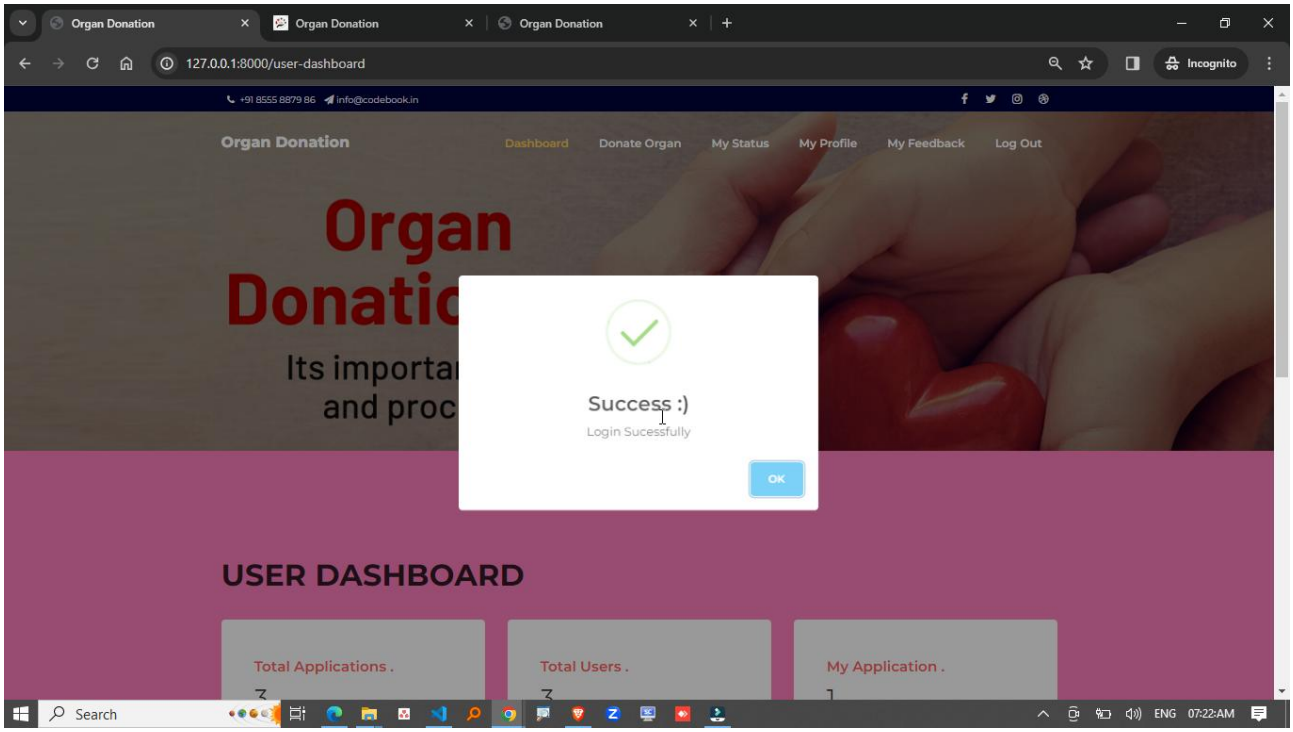


Fig.8.7 *User Login Successful*

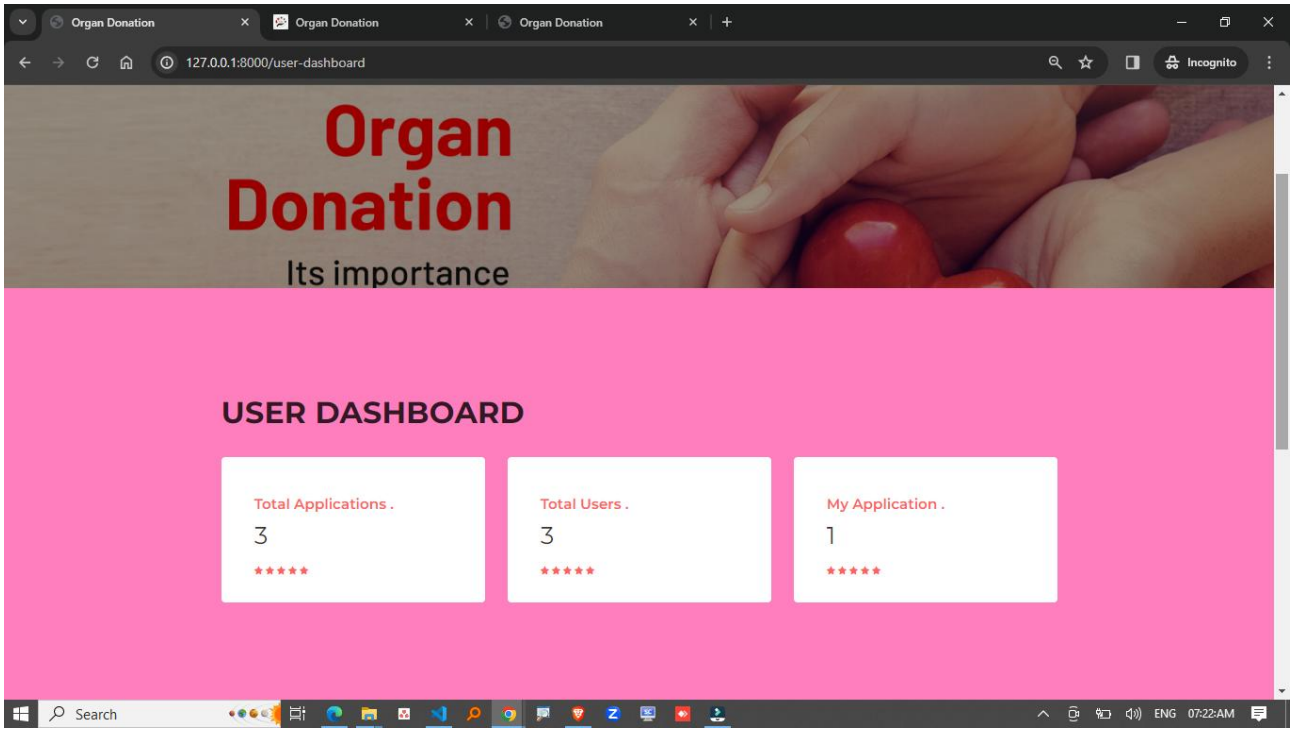


Fig.8.8 *User Dashboard*

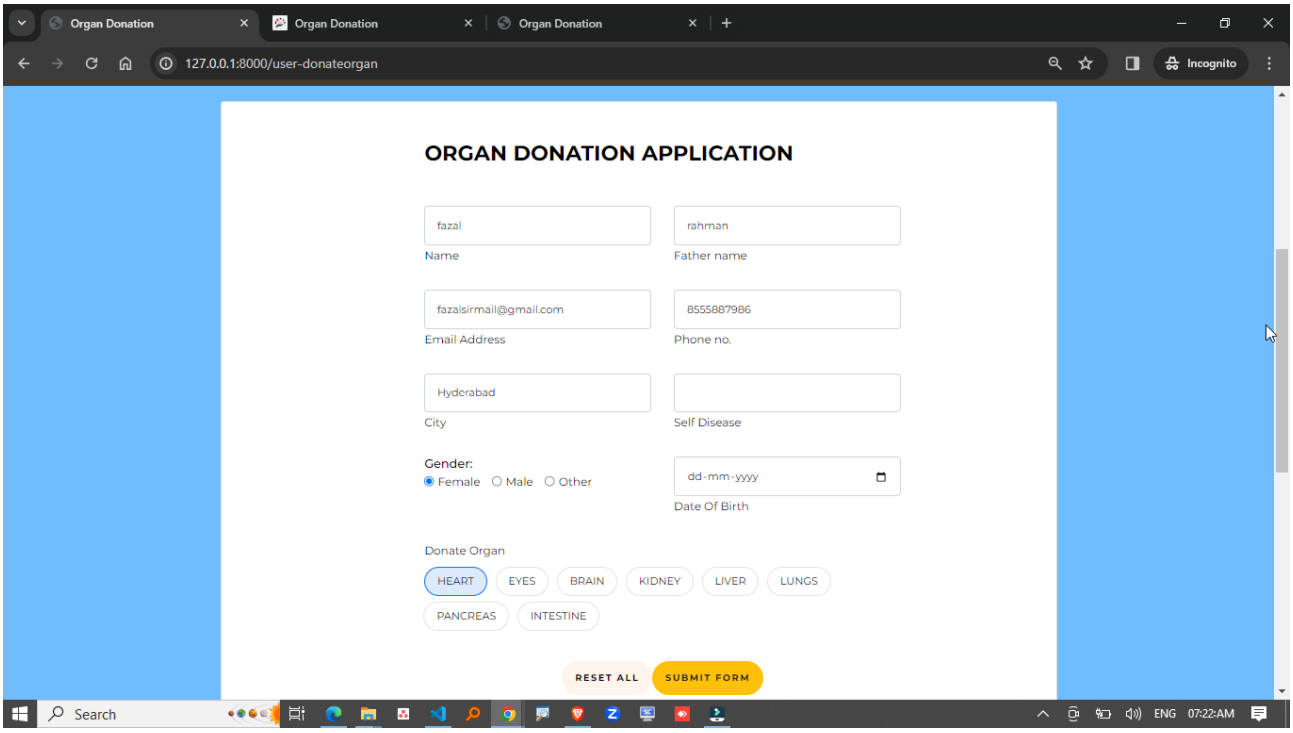


Fig.8.9 *User Organ Donation Application*

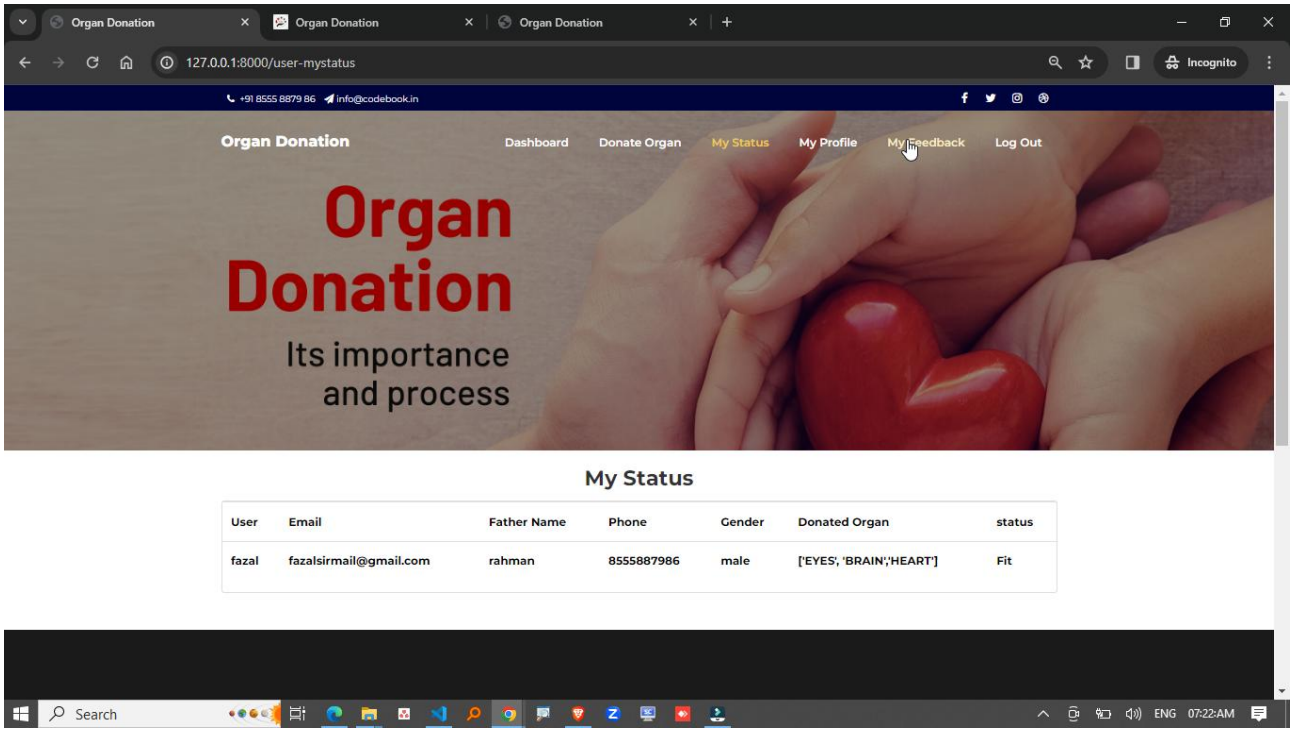


Fig.8.10 *User Doner Status*

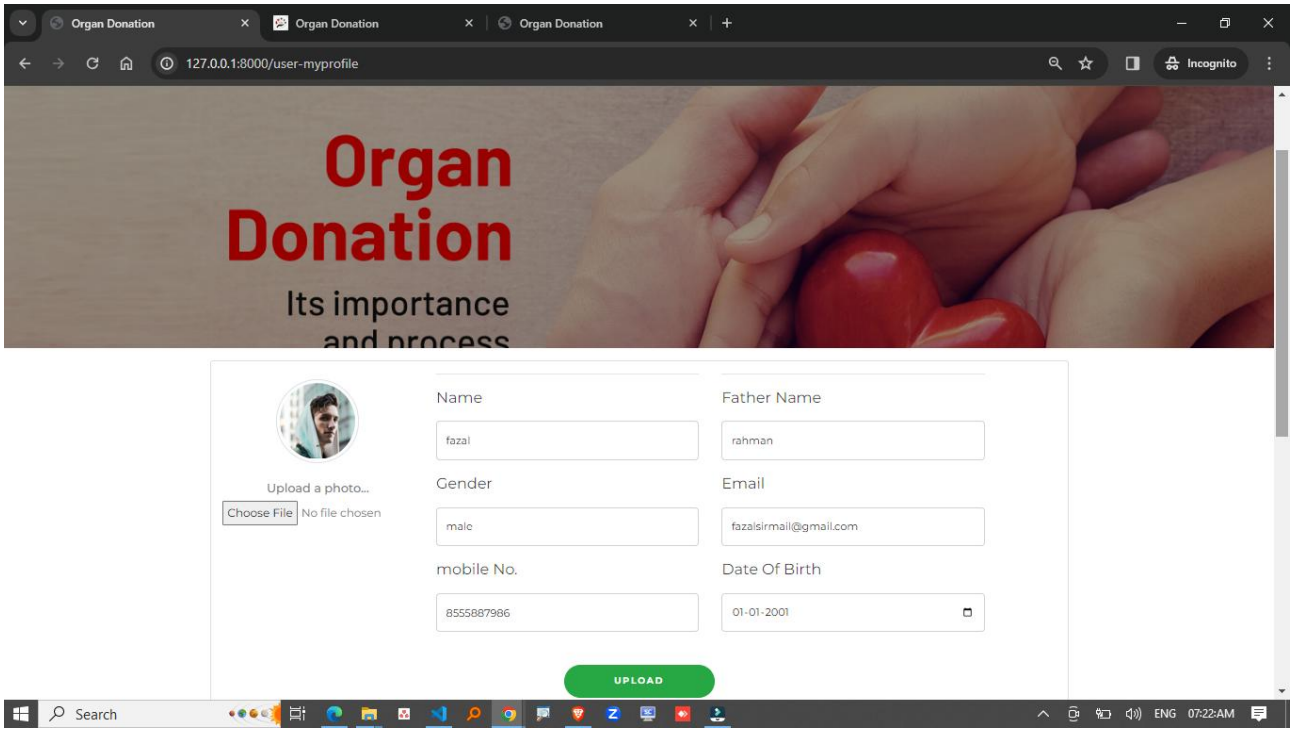


Fig.8.11 *User Profile*

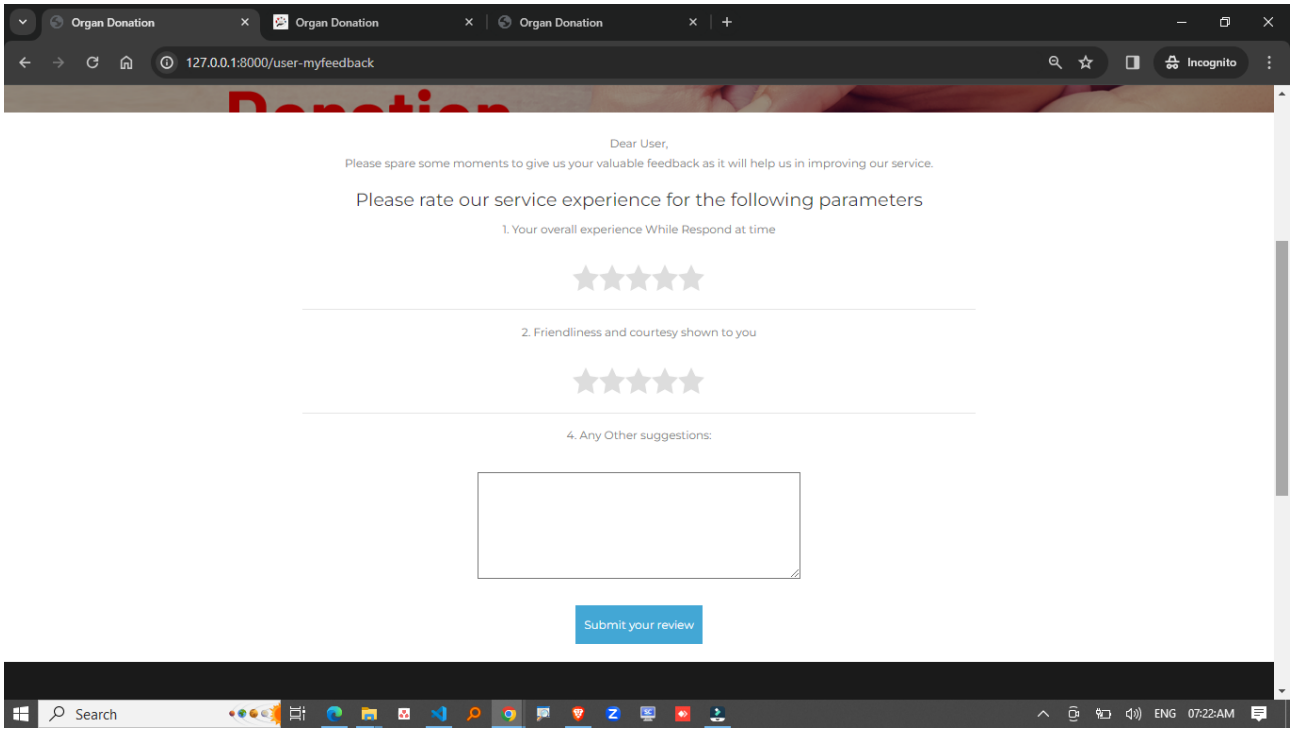


Fig.8.12 *User FeedBack*

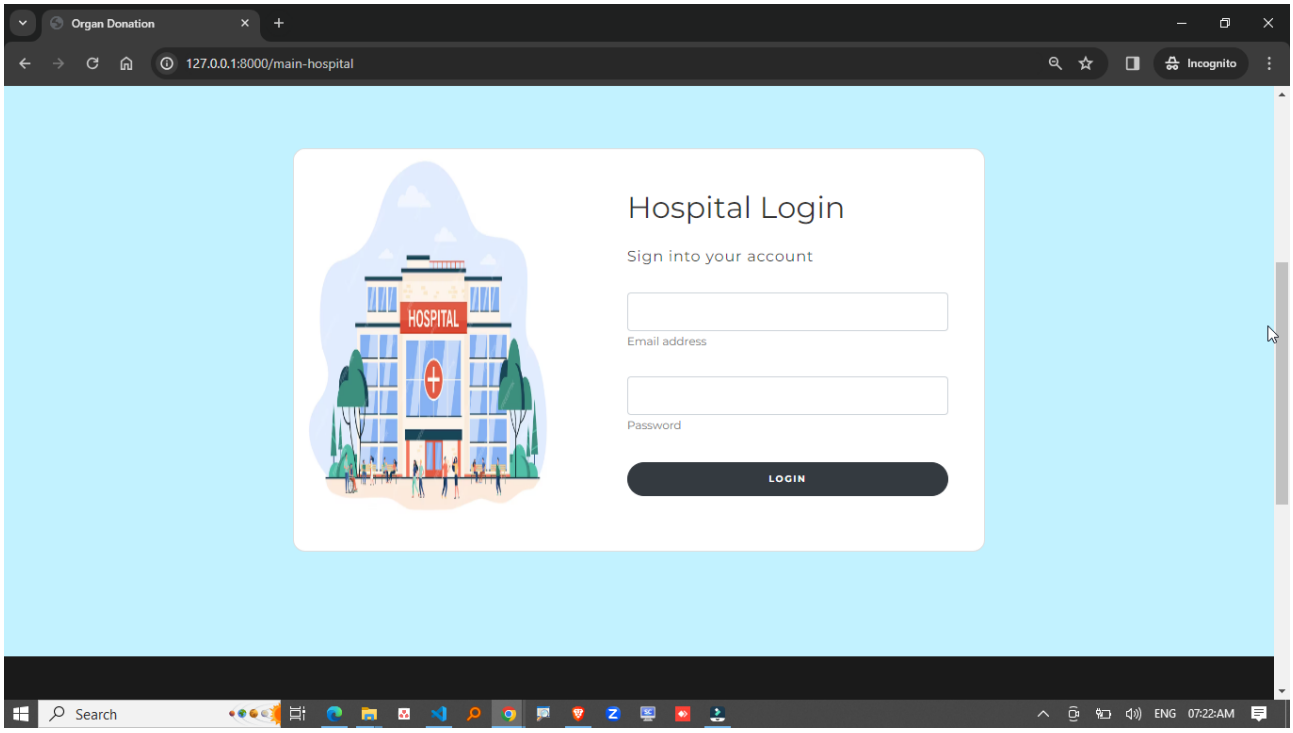


Fig.8.13 *Hospital Login*

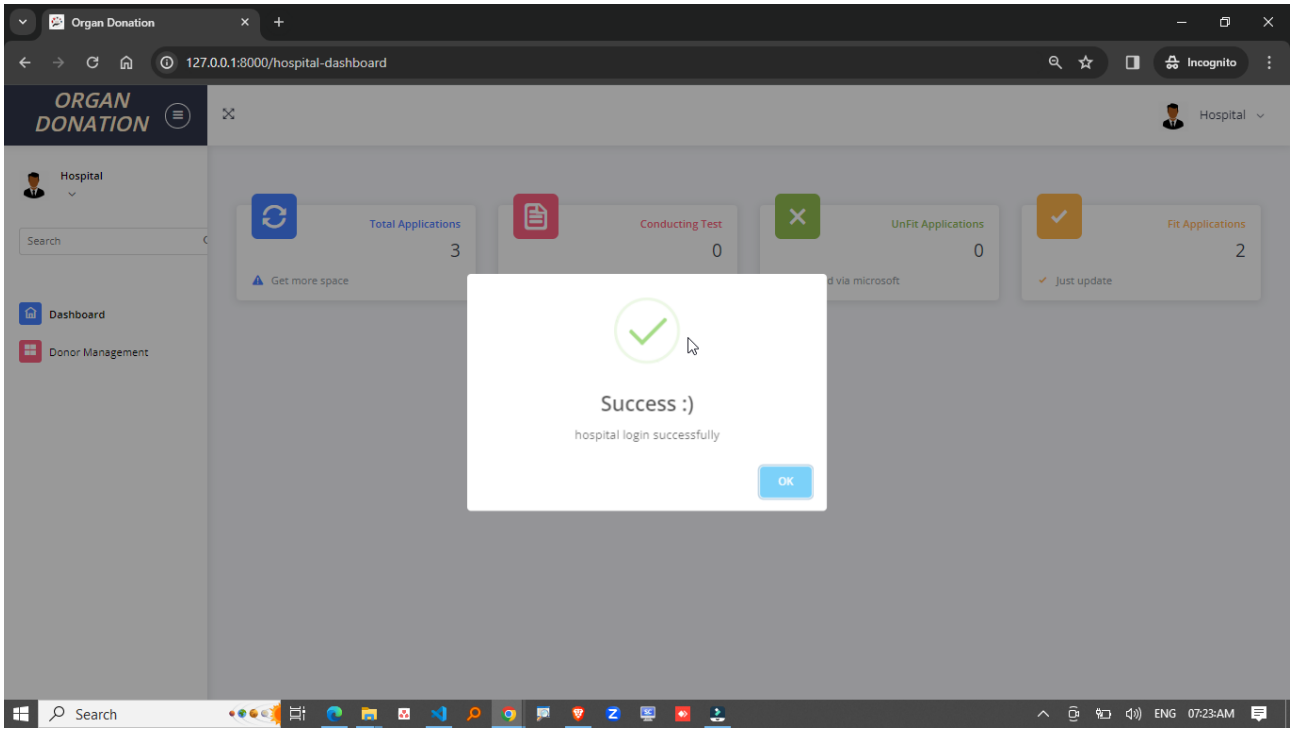


Fig.8.14 *Hospital Login Successful*

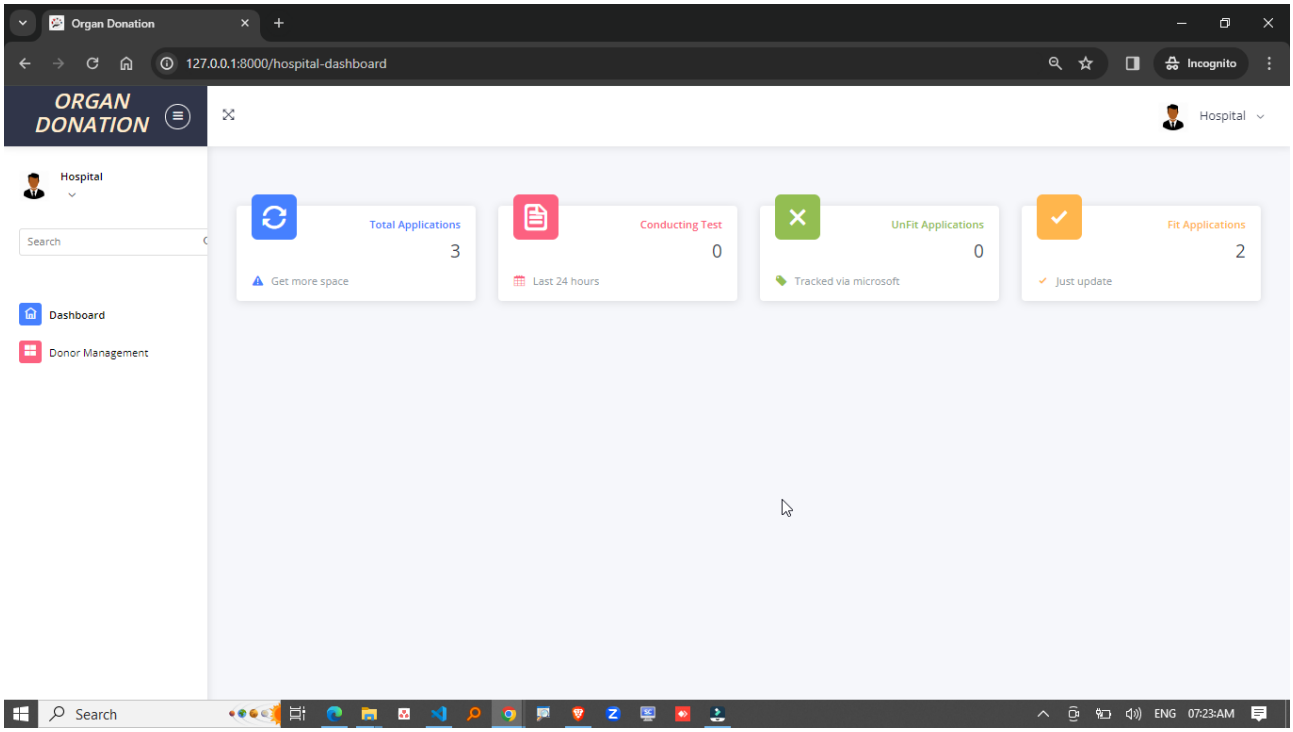


Fig.8.15 *Hospital Dashboard*

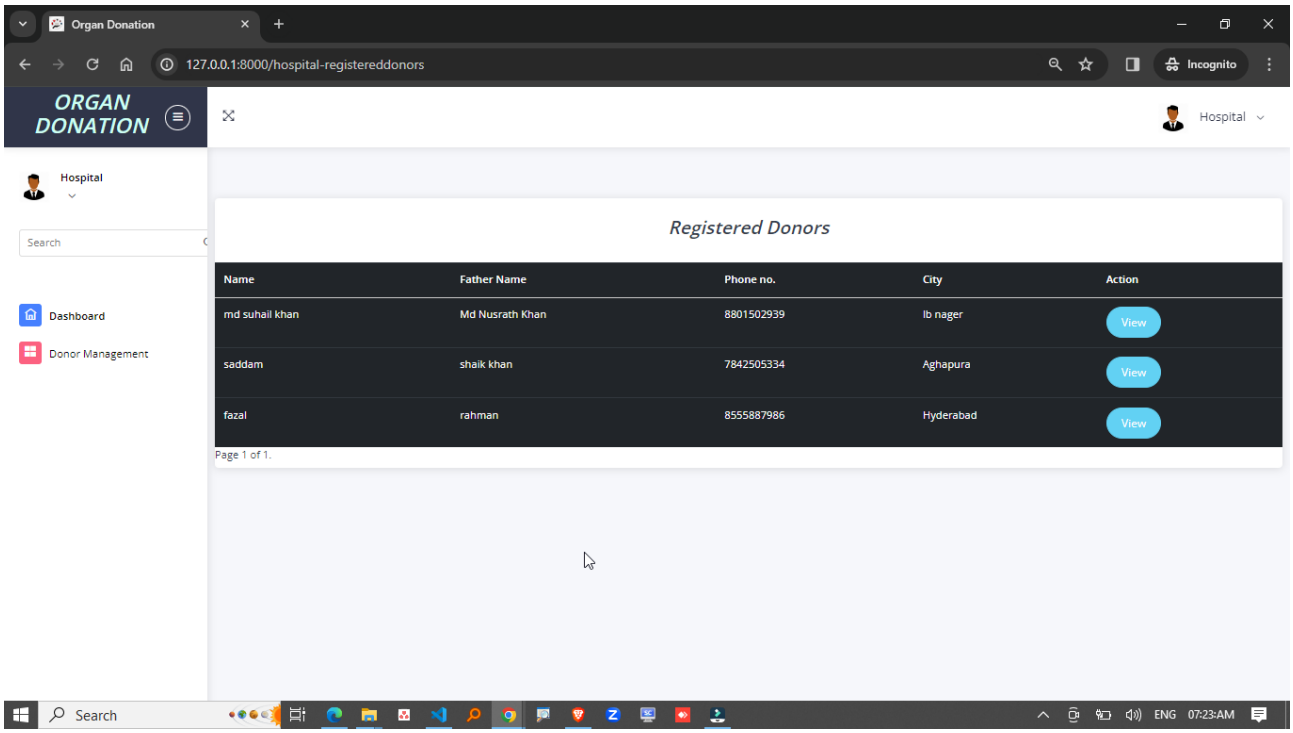


Fig.8.16 *Hospital Registered Donors*

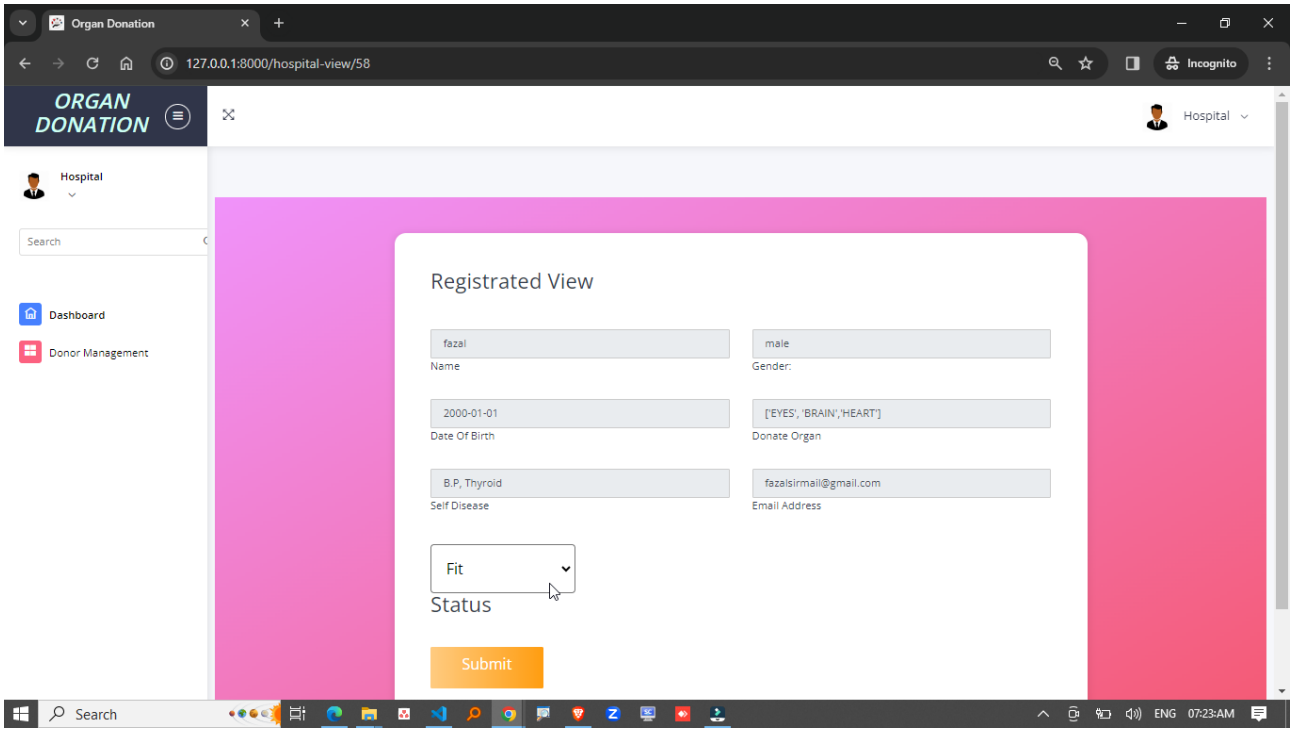


Fig.8.17 *Hospital Registrated View*

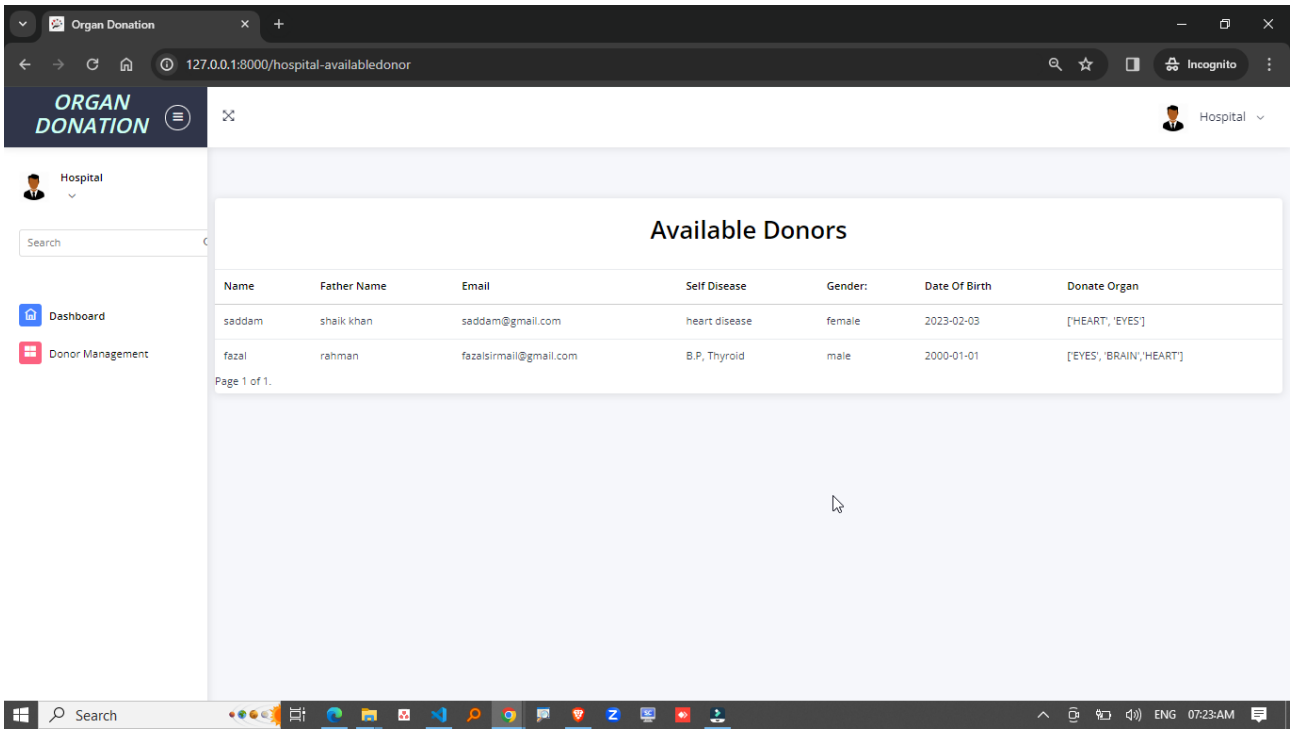


Fig.8.18 *Hospital Available Donors*

ADMIN VIEW

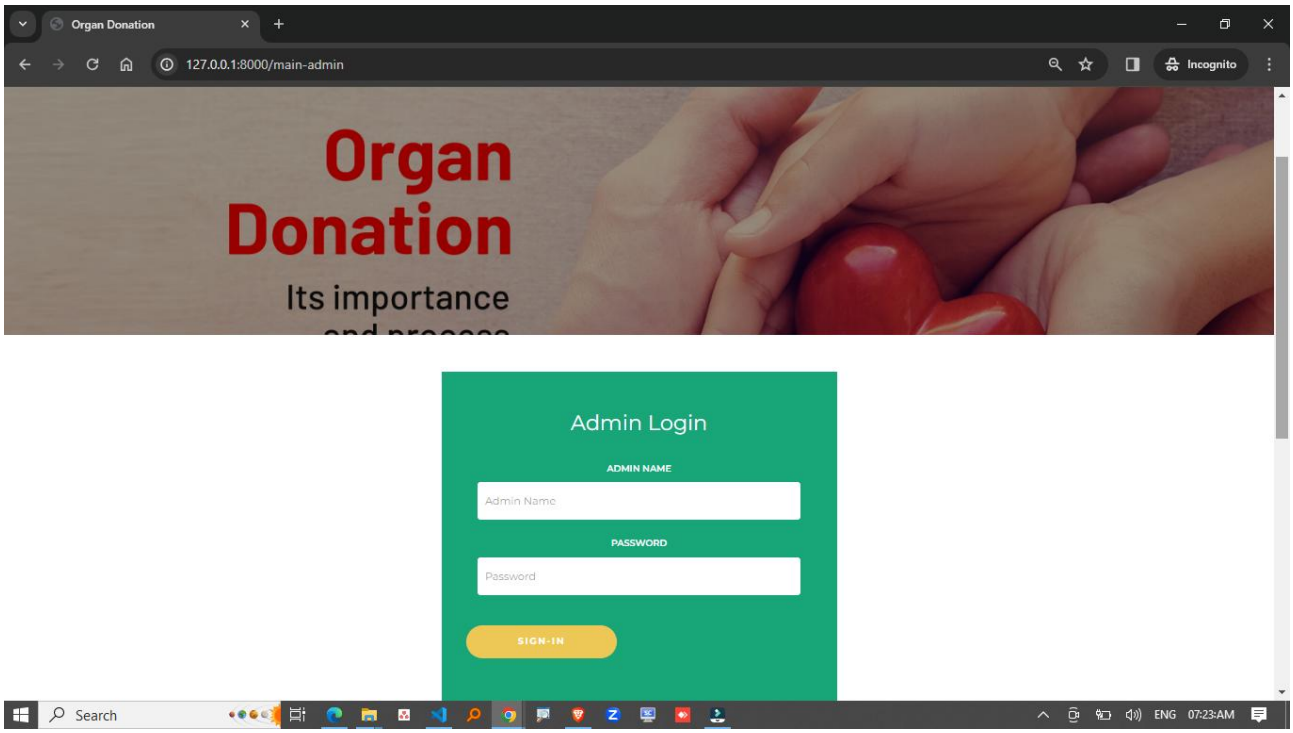


Fig.8.19 *Admin Login*

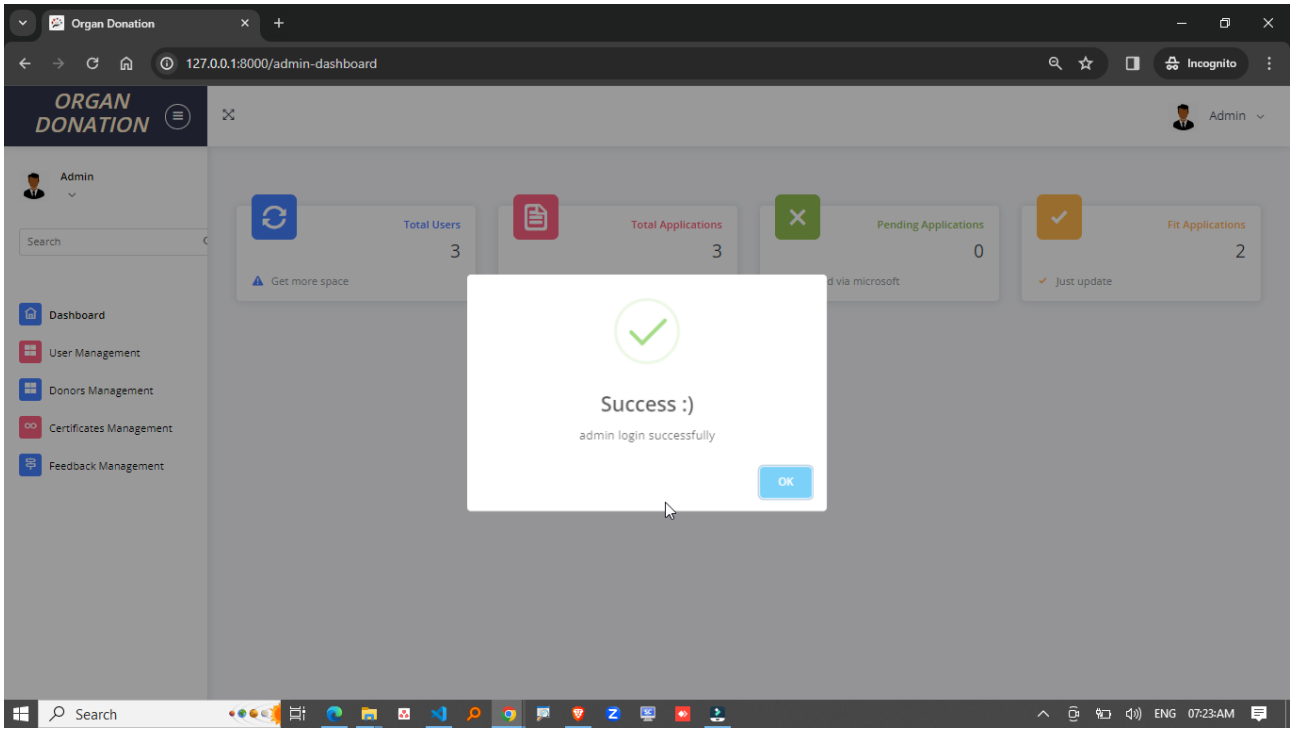


Fig.8.20 *Admin Login Successful*

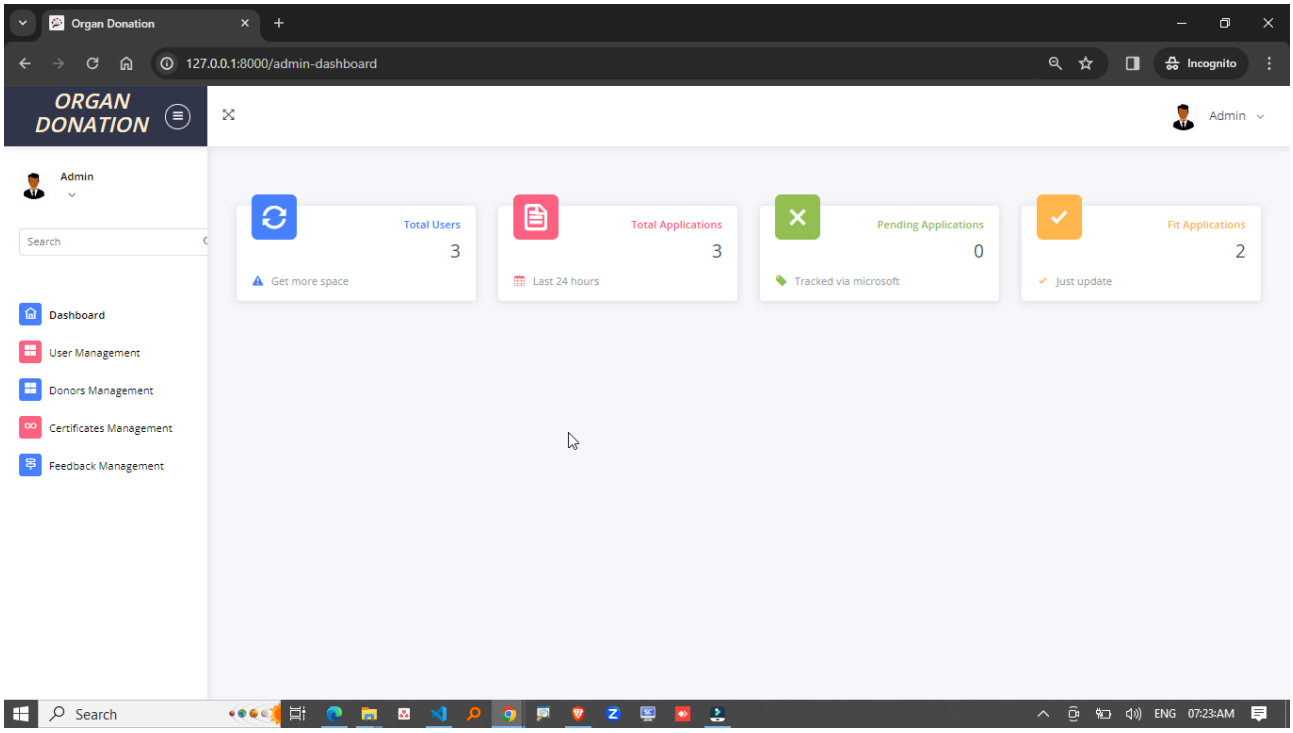


Fig.8.21 *Admin Dashboard*

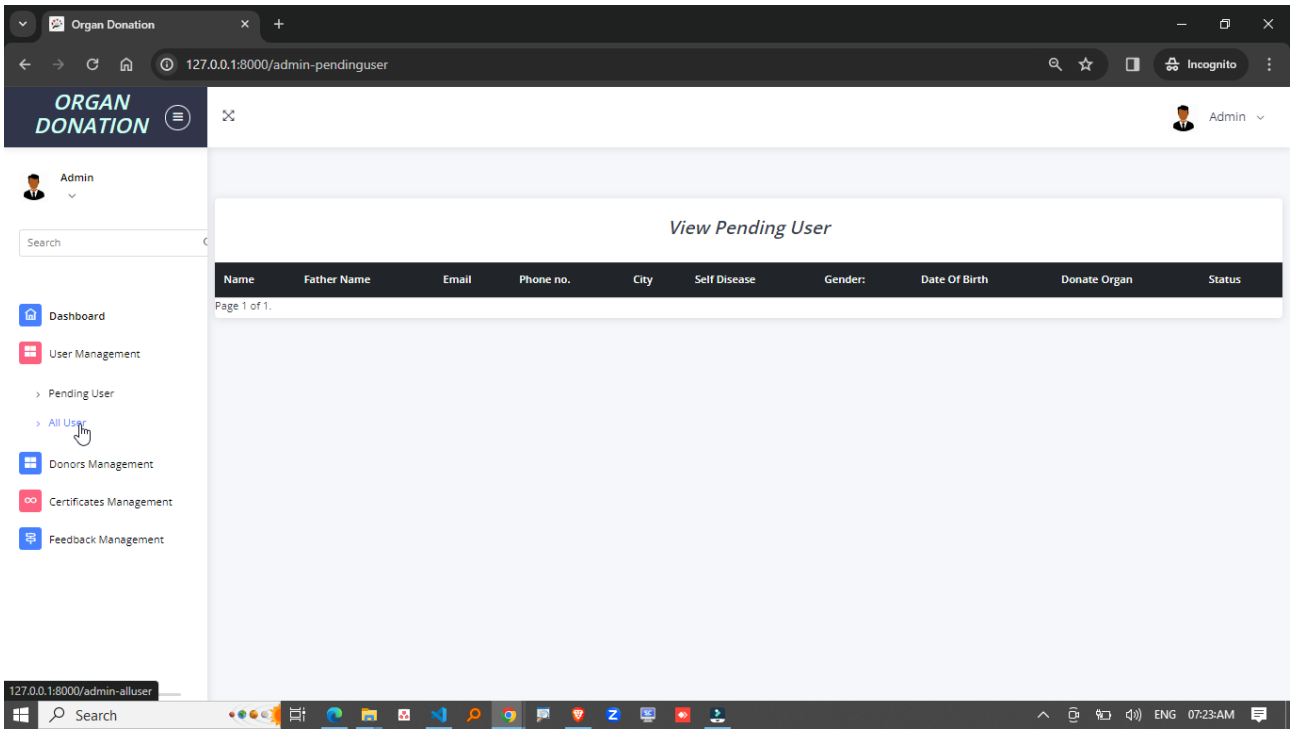


Fig.8.22 *Pending Users Dashboard*

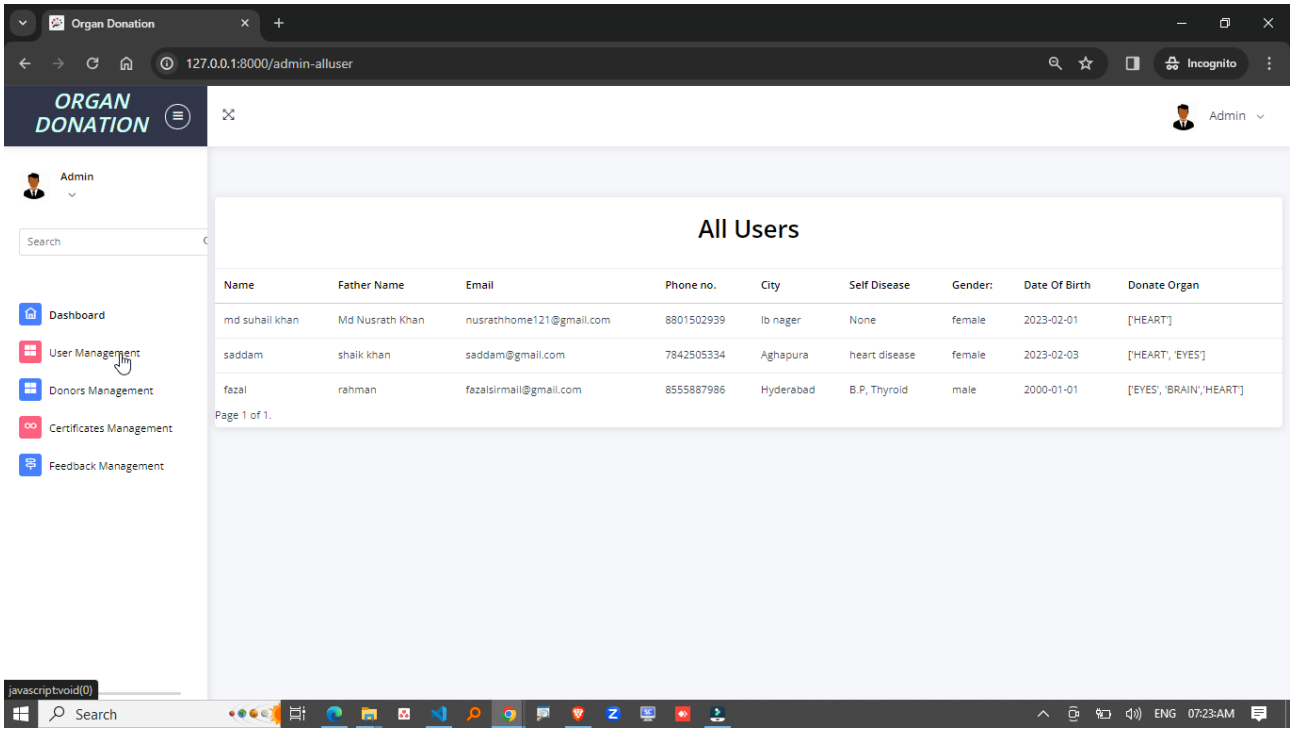


Fig.8.23 *All Users Dashboard*

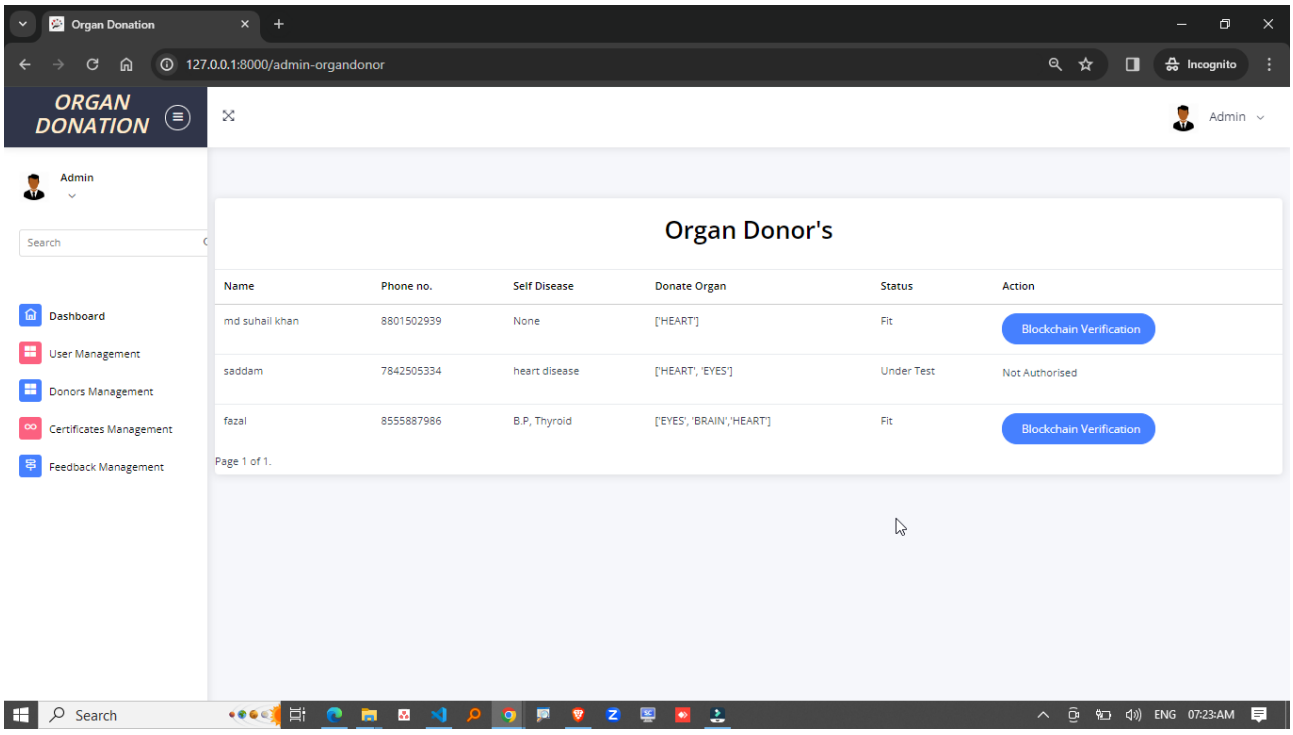


Fig.8.24 *Organ Donor's Dashboard*

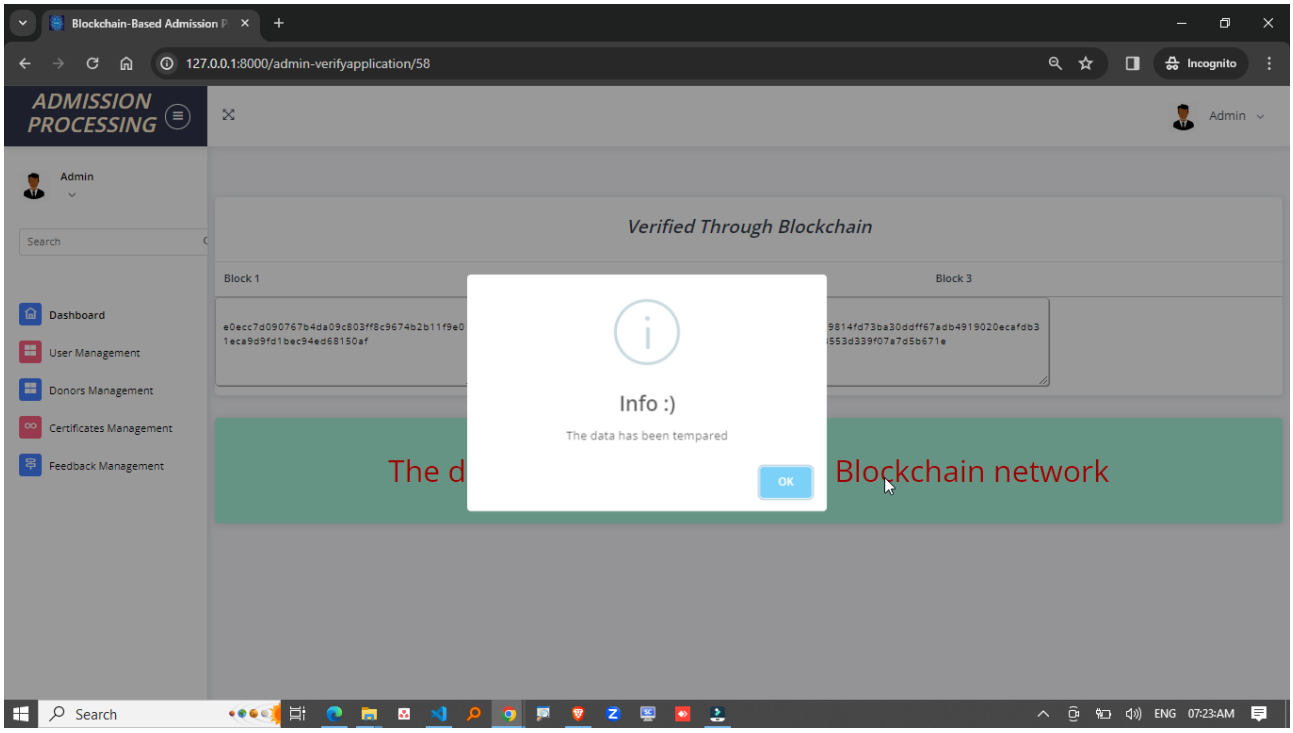


Fig.8.25 *Verify Application*

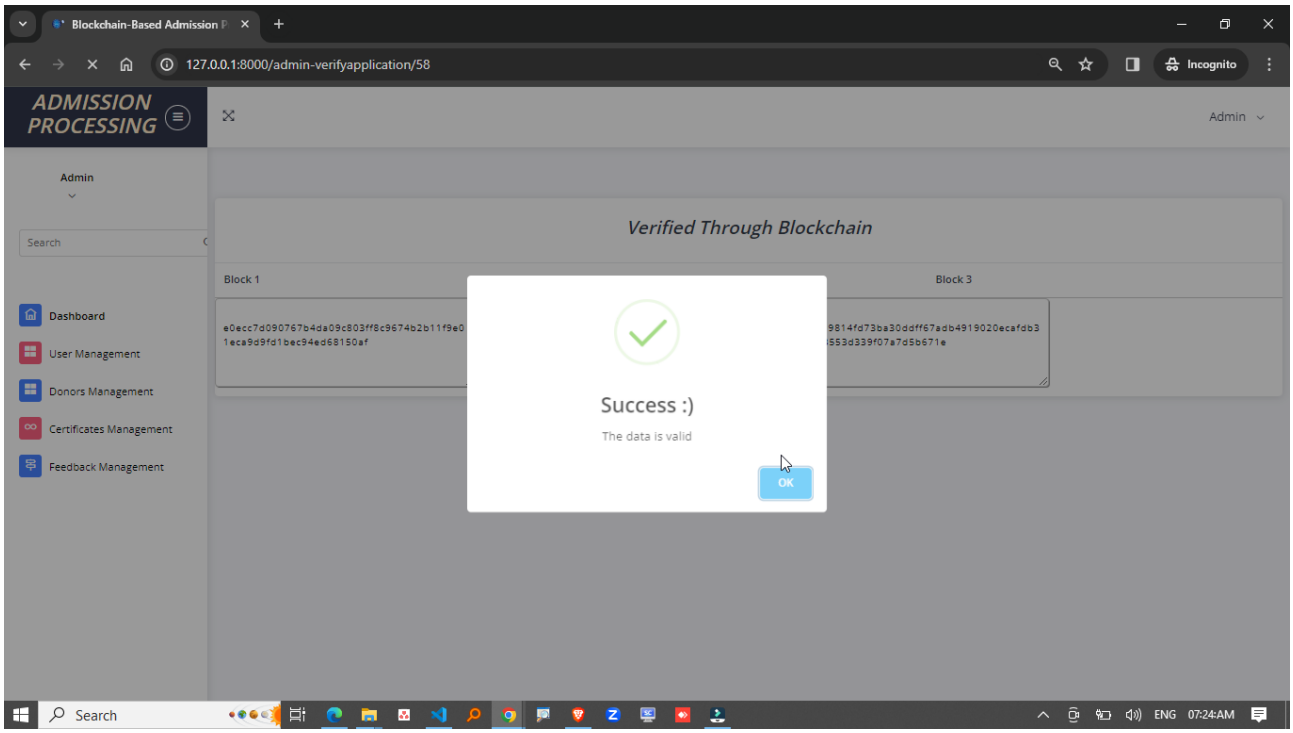


Fig.8.26 *Data is Valid*

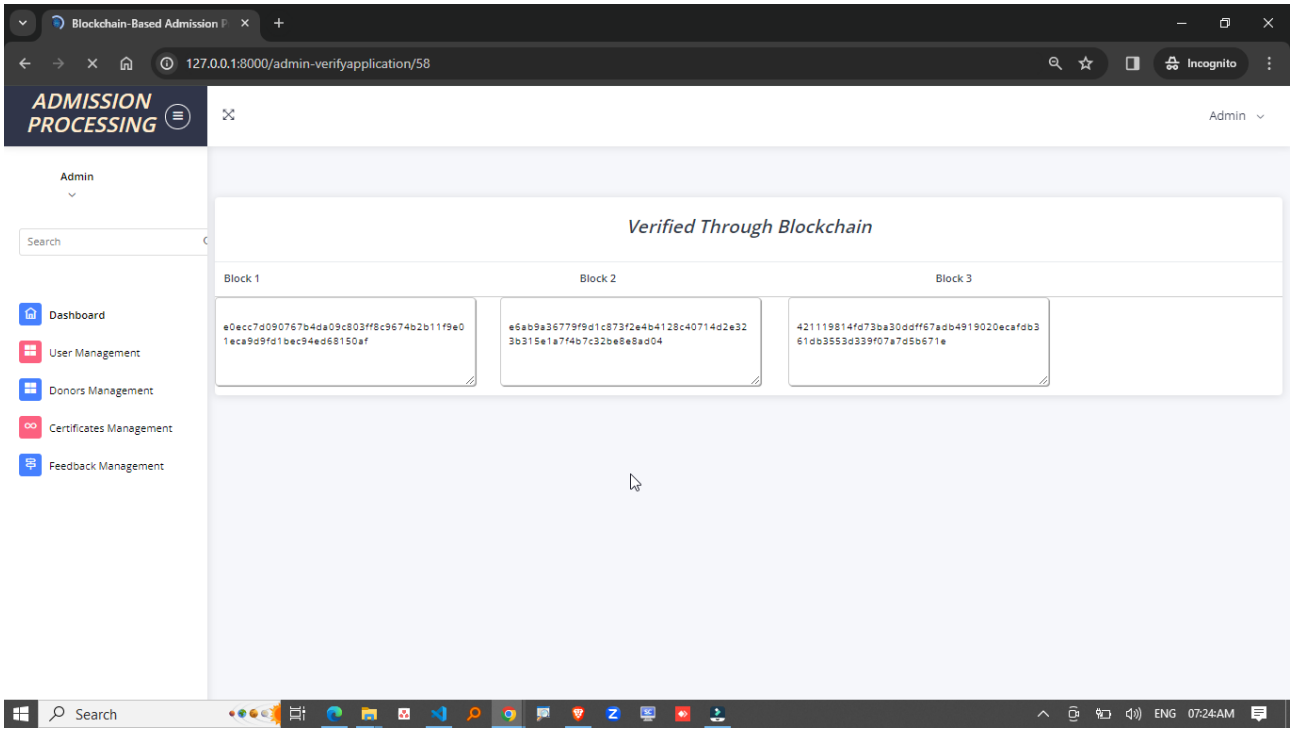


Fig.8.27 *Verified Through Blockchain*

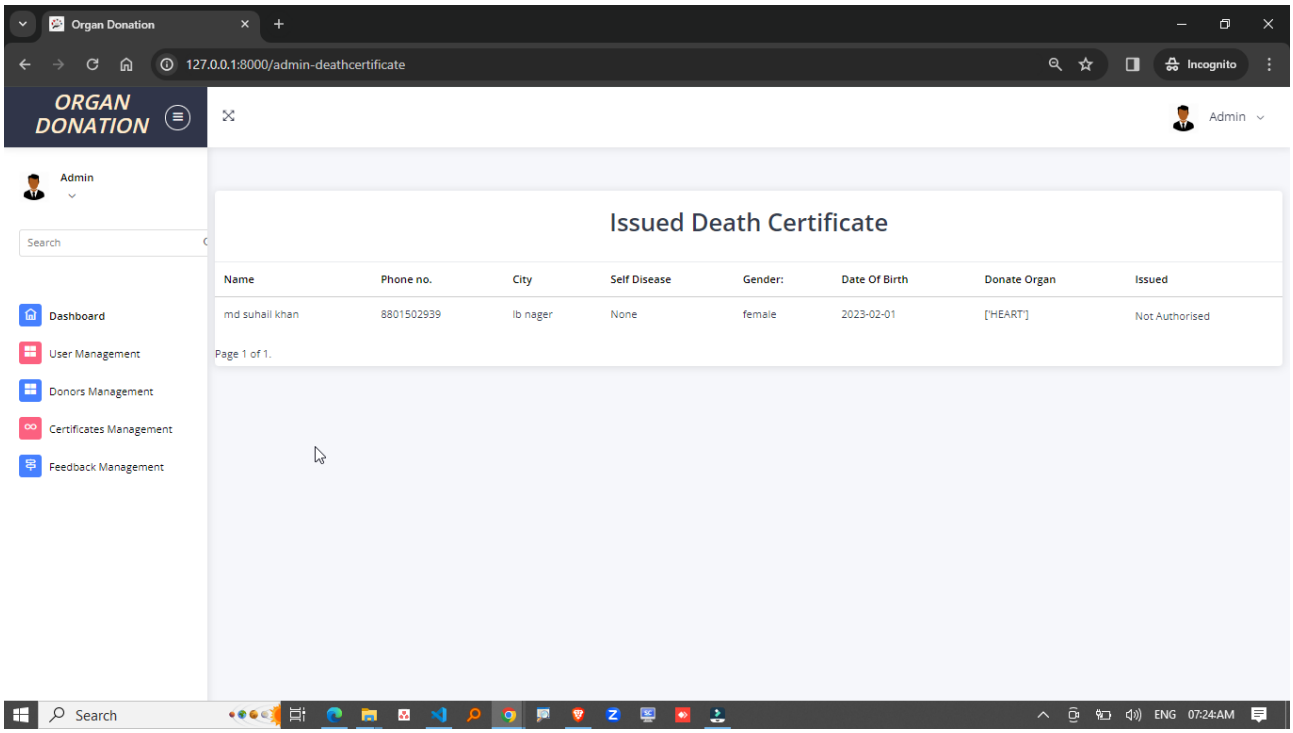


Fig.8.28 *Issued Death Certificate*

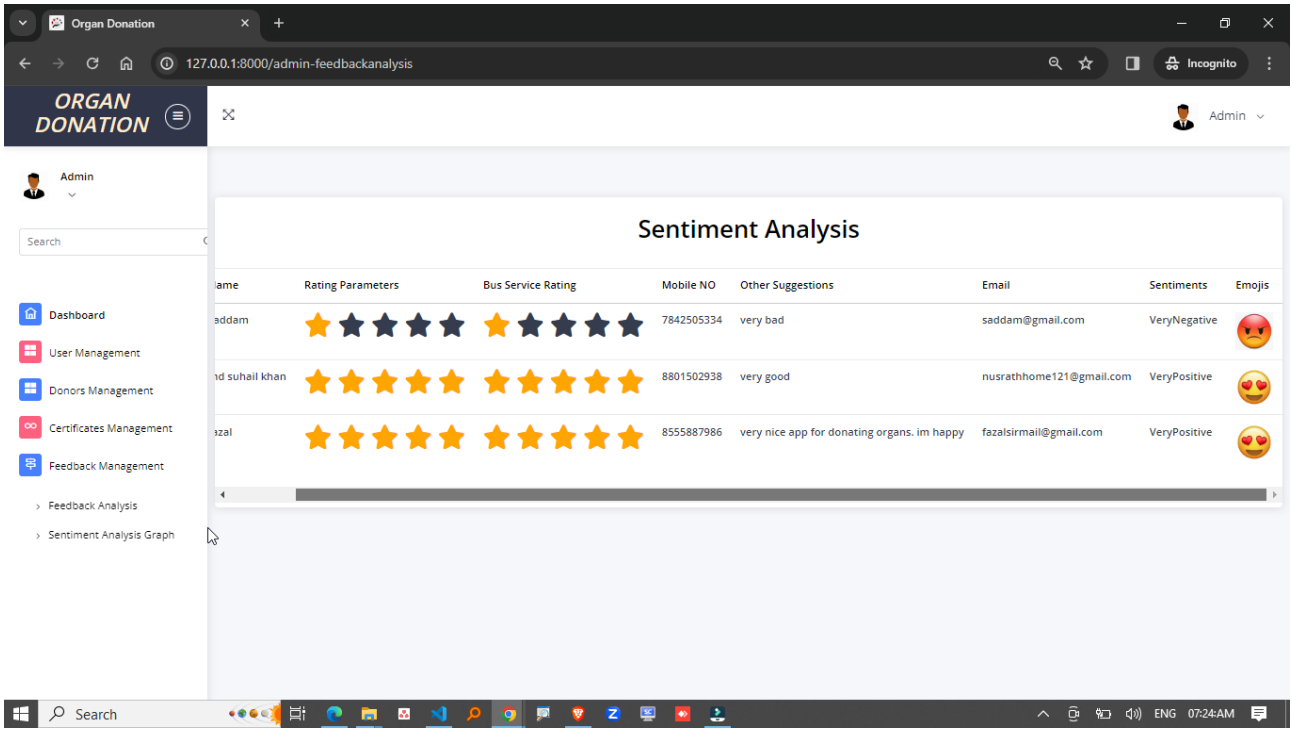


Fig.8.29 *Sentiment Analysis*

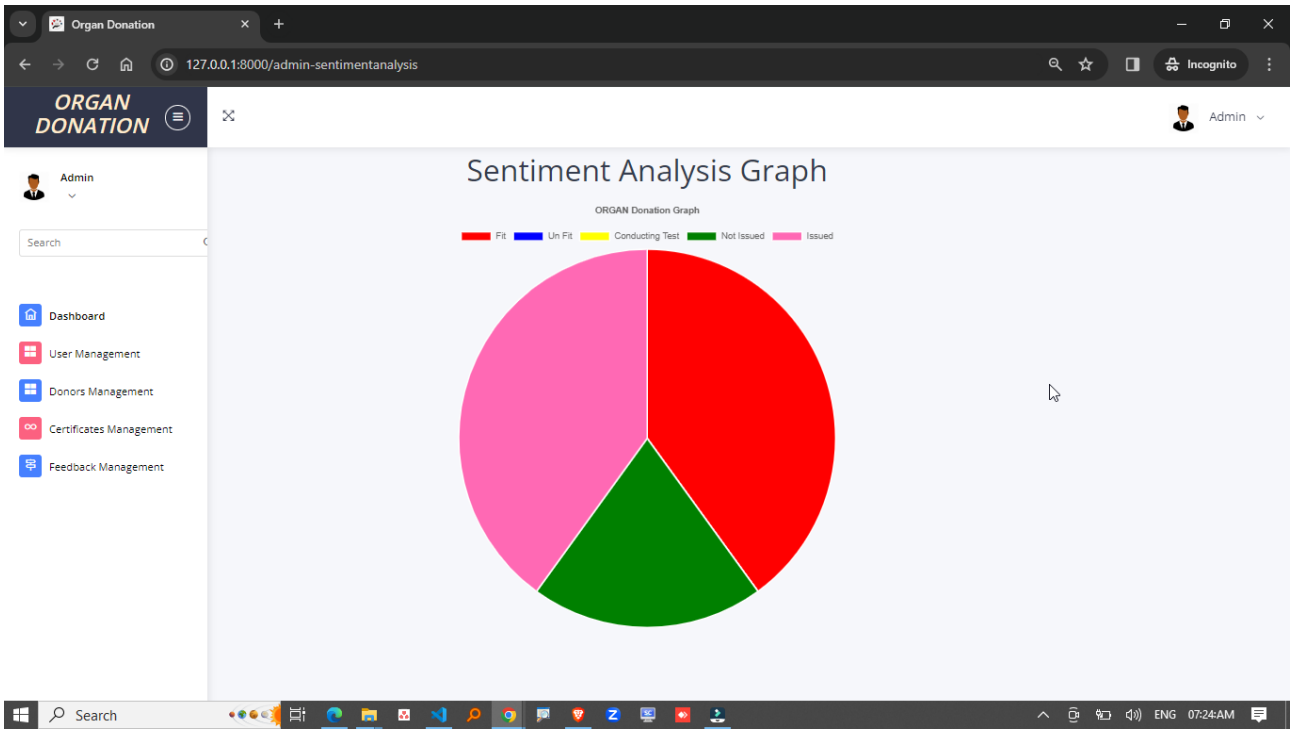


Fig.8.30 *Sentiment Analysis Graph*

9. SYSTEM TESTING

TYPES OF SOFTWARE TESTING:

DIFFERENT TESTING TYPES WITH DETAILS

We, as testers, are aware of the various types of Software Testing like Functional Testing, Non-Functional Testing, Automation Testing, Agile Testing, and their sub-types, etc.

Each type of testing has its own features, advantages, and disadvantages as well. However, in this tutorial, we have covered mostly each and every type of software testing which we usually use in our day-to-day testing life.

DIFFERENT TYPES OF SOFTWARE TESTING

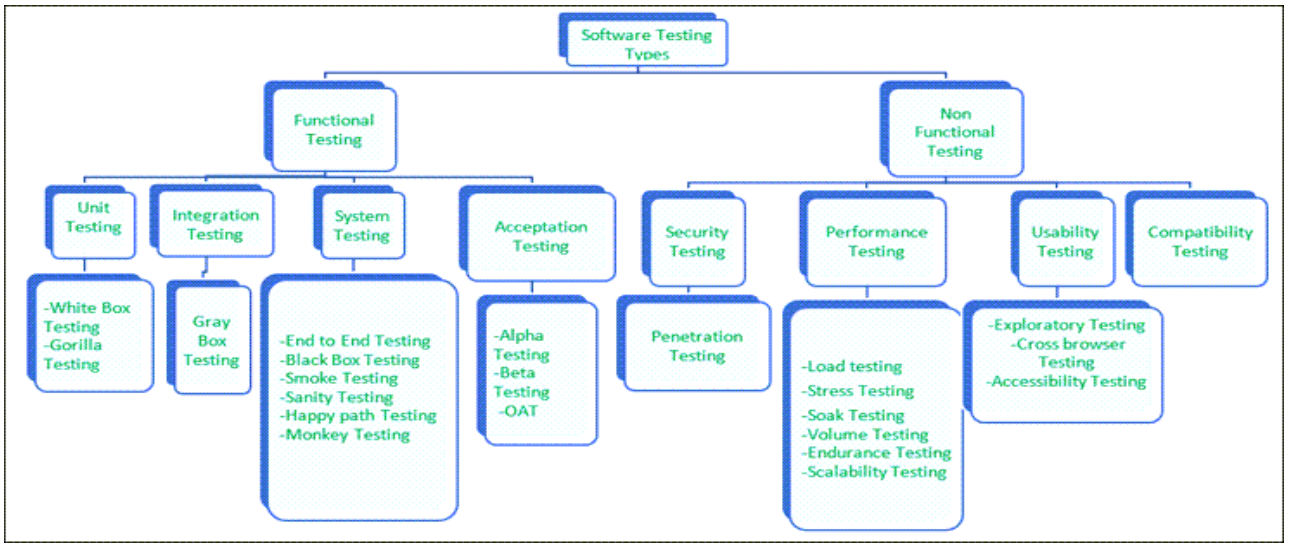


Fig.9.1 *Software Testing*

FUNCTIONAL TESTING

There are four main types of functional testing.

1. Unit Testing

Goal: Test individual components or functions in isolation to ensure they work correctly.

- **Details:**
  - Focuses on testing a single "unit" of code (usually functions or methods).
  - Helps identify bugs early in the development process.
  - Typically, automated.
- **Tools:** JUnit (Java), Mocha (JavaScript), PyTest (Python).
- **Example:**
  - Testing a function that calculates organ compatibility based on blood type.

2. Integration Testing

Goal: Verify that different components/modules of the system interact properly.

- **Details:**
  - Focuses on the interaction between different modules or services.
  - Ensures that integrated components work together as expected.
  - Tests APIs, databases, and third-party services.
- **Tools:** Postman (for API), JUnit (for Java-based systems), PyTest.
- **Example:**
  - Verifying that the patient registration API properly interacts with the database.

3. System Testing

Goal: Test the complete system in an environment that mimics production to verify the system as a whole.

- **Details:**
  - End-to-End testing of the entire application, including its components and integrations.
  - Ensures the system meets specified requirements.
  - Includes functional, non-functional, and security tests.
- **Tools:** Selenium (for web apps), TestComplete.
- **Example:**
  - Testing the entire Organ Matching & Transplantation Blockchain system, including user registration, matching algorithms, and blockchain logging.

4. Acceptance Testing

Goal: Ensure the system meets business requirements and is ready for release.

- **Details:**
  - Usually performed by the end-users or clients to validate the system meets business needs.
  - Includes functional and user experience validation.
  - Can be further divided into **Alpha** (internal testing) and **Beta** (external testing with a limited audience).
- **Tools:** User Acceptance Testing (UAT) tools.
- **Example:**
  - A hospital testing the organ transplant request system to confirm it meets all operational needs before going live.

5. Smoke Testing

Goal: Ensure that the most critical functions of the system work after a new build or update.

- **Details:**
  - Also known as "build verification testing".
  - A shallow, broad test to check basic functionality, ensuring that the build is stable enough for further detailed testing.
- **Tools:** None specifically; manual tests are commonly used.
- **Example:**
  - After deploying a new build of the organ matching system, a test is run to ensure that users can log in, and critical features (like organ listing) are accessible.

6. Sanity Testing

Goal: Ensure that a specific part or feature of the system works after changes are made.

- **Details:**
  - More focused than smoke testing, typically applied after bug fixes or minor changes to verify that the changes work and didn't introduce new issues.
- **Tools:** Manual testing is often sufficient.
- **Example:**
  - After fixing an issue with organ matching, sanity testing ensures the matching algorithm works as expected without introducing new bugs.

7. Regression Testing

Goal: Ensure that new code changes don't negatively impact the existing functionality.

- **Details:**
  - Focuses on detecting issues in existing features after code changes.
  - Typically automated to cover existing features.
- **Tools:** Selenium, TestNG, JUnit.
- **Example:**
  - After adding a new feature for organ donation alerts, regression testing checks that previous features, like donor registration, are still working.

8. Performance Testing

Goal: Evaluate how the system performs under load and stress.

- **Details:**
  - Focuses on system performance, including **load testing**, **stress testing**, **scalability**, and **stability**.
  - It ensures the system can handle the expected number of users or data transactions.

- **Tools:** JMeter, LoadRunner.
- **Example:**
  - Testing how the organ matching system performs with a high number of simultaneous patient registration and organ donation requests.

9. Load Testing

Goal: Test the system's performance under expected load conditions.

- **Details:**
  - Simulates normal and peak load conditions to measure response time and system behavior.
  - Helps identify bottlenecks or performance degradation under normal or expected traffic.
- **Tools:** Apache JMeter, LoadRunner.
- **Example:**
  - Testing how the system behaves with 1000 hospital registration requests per minute.

10. Stress Testing

Goal: Test how the system behaves under extreme conditions, beyond its expected load.

- **Details:**
  - Focuses on the system's stability and robustness by pushing it to its limits.
  - Identifies the breaking point of the system and ensures recovery from failure.
- **Tools:** JMeter, LoadRunner.
- **Example:**
  - Testing how the system behaves when 10,000 organ donation requests are sent in a short period.

11. Security Testing

Goal: Identify vulnerabilities and ensure the system is protected from attacks.

- **Details:**
  - Tests the system for security flaws, such as data leaks, unauthorized access, and hacking risks.
  - Includes tests like penetration testing, risk analysis, and authentication checks.
- **Tools:** OWASP ZAP, Burp Suite.
- **Example:**
  - Testing the organ matching system's authentication process to prevent unauthorized access to sensitive patient data.

12. Usability Testing

Goal: Ensure the system is user-friendly and intuitive.

- **Details:**
  - Focuses on the user experience, making sure the system is easy to navigate and use.
  - Includes evaluating the system's interface, workflows, and user feedback.
- **Tools:** No specific tools (manual evaluation, surveys, user feedback).
- **Example:**
  - Testing the user interface for ease of use when hospitals interact with the organ matching system.

13. Compatibility Testing

Goal: Verify that the system works across different environments (browsers, devices, OS).

- **Details:**
  - Ensures the software performs well across different configurations.
  - Can involve testing the system across different operating systems, devices, screen resolutions, and browsers.
- **Tools:** BrowserStack, CrossBrowserTesting.
- **Example:**
  - Testing the organ matching platform to ensure it works correctly on both Android and iOS devices.

14. Recovery Testing

Goal: Ensure the system can recover from failures, crashes, or data corruption.

- **Details:**
  - Simulates system failures (e.g., power loss, database corruption) to test the system's ability to recover gracefully.
  - Focuses on data integrity and ensuring that no data is lost after recovery.
- **Tools:** Custom tools for failure simulation.
- **Example:**
  - Simulating a crash during the organ matching process and testing if the data is restored accurately from backups.

15. Alpha Testing

Goal: Perform initial testing within the development team before releasing the software to a larger audience.

- **Details:**
  - Conducted by the internal team (developers, QA) before releasing to beta testers.
  - Identifies major issues early in the development process.
- **Tools:** Manual testing.
- **Example:**
  - Developers test the initial version of the organ matching system to ensure it meets basic requirements.

16. Beta Testing

Goal: Perform external testing to gather feedback from users in real-world scenarios.

- **Details:**
  - Typically conducted with a limited set of end users who are not part of the development team.
  - Provides valuable insights into user experience and potential bugs before full release.
- **Tools:** Beta testing platforms or manual testing.
- **Example:**
  - A few hospitals and medical professionals test the system with real data to ensure usability, correctness, and performance.

10. CONCLUSION

This paper presents a novel approach to managing organ donation and transplantation through a private Ethereum blockchain solution, designed to enhance decentralization, accountability, traceability, and security. By leveraging the decentralized nature of blockchain technology, our system ensures that all data related to organ donation is immutable, transparent, and securely recorded, significantly reducing the risks of fraud and improving the trustworthiness of the entire process.

The core of our solution involves the use of smart contracts, which automate the recording of key events and ensure data provenance throughout the system. These contracts enhance the efficiency of the organ matching and transplantation process by reducing human error, improving data accuracy, and providing an auditable trail of all transactions. To demonstrate the effectiveness of our solution, we developed six distinct algorithms, each tested and validated through rigorous evaluation.

We also conducted a comprehensive security analysis of the smart contracts, ensuring that they are protected against common vulnerabilities and attacks, which is crucial in a highly sensitive domain like organ donation. This thorough security assessment confirms that the system is both secure and reliable.

In comparison to existing blockchain-based solutions, our approach stands out by offering a more transparent and accountable system for organ donation. The ability to track and audit every transaction in real time offers an unmatched level of trust. Moreover, our solution is flexible and can easily be adapted to meet the specific needs of other sectors with similar challenges, showcasing its broad applicability.

Looking forward, several improvements are possible. One key enhancement is the development of a fully functional end-to-end decentralized application (DApp) to simplify user interaction and further enhance the security and transparency of the platform. Additionally, testing and deployment on a real private Ethereum network would provide valuable insights into the scalability and performance of the system in a live environment. Furthermore, integrating the **Quorum platform** could offer even greater confidentiality, as it allows transactions to be viewed only by specific participants, enhancing privacy compared to our current solution, where transactions are visible to all authorized actors in the private blockchain.

In conclusion, our blockchain-based solution for organ donation and transplantation offers a secure, transparent, and efficient framework for managing this critical process. With potential for future improvements and adaptability for other sectors, this system represents a step forward in how sensitive data can be managed in a decentralized and trustworthy manner.

11. REFERENCES

[1] L. A. Dajim, S. A. Al-Farras, B. S. Al-Shahrani, A. A. Al-Zuraib, and R. Merlin Mathew, ‘‘Organ donation decentralized application using blockchain technology,’’ in Proc. 2nd Int. Conf. Comput. Appl. Inf. Secur. (ICCAIS), May 2019, pp. 1–4, doi: 10.1109/cais.2019.8769459.

[2] A. Powell. (Mar. 18, 2019). A Transplant Makes History. Harvard Gazette. [Online]. Available: https://news.harvard.edu/gazette/story/2011/09/atransplant-makes-history/

[3] Organ Donation Facts and Info: Organ Transplants. Accessed: Apr. 18, 2021. [Online]. Available: https://my.clevelandclinic.org/health/ articles/11750-organ-donation-and-transplantation

[4] (Mar. 21, 2019). Facts and Myths About Transplant. Accessed: Apr. 21, 2021. [Online]. Available: https://www.americantransplant foundation.org/about-transplant/facts-and-myths/

[5] Organ Procurement and Transplantation Network. Accessed: Apr. 18, 2021. [Online]. Available: https://optn.transplant.hrsa.gov/resources/ethics/ethical-principles-in-the-allocation-of-humanorgans/

[6]How Donation Works. Accessed: Jan. 7, 2022. [Online]. Available: https://www.organdonor.gov/learn/process

[7] UFO Themes. (Aug. 1, 2017). Organ Donation and Transplantation in Germany. Plastic Surgery Key. [Online]. Available: https:// plasticsurgerykey.com/organ-donation-and-transplantation-in-germany/

[8] Harvard Business Review. (Dec. 13, 2021). Electronic Health Records Can Improve the Organ Donation Process. Accessed: Apr. 8, 2022. [Online]. Available: https://hbr.org/2021/12/electronic-health-records-can-improvethe-organ-donation-process

[9] U. Jain, ‘‘Using blockchain technology for the organ procurement and transplant network,’’ San Jose State Univ., San Jose, CA, USA, Tech. Rep., 2020, doi: 10.31979/etd.g45p-jtuy.

[10] M. He, A. Corson, J. Russo, and T. Trey, ‘‘Use of forensic DNA testing to trace unethical organ procurement and organ trafficking practices in regions that block transparent access to their transplant data,’’ SSRN Electron. J., 2020, doi: 10.2139/ ssrn.3659428.

[11] Livemint. The Illegal Organ Trade Thrives in India-and it isn’t Likely to End Soon. Accessed: Dec. 21, 2021. [Online]. Available: https://www.livemint.com/Politics/pxj4YasmivrvAhanv6OOCJ/Whyorgan-trafficking-thrives-in-India.html

[12] D. P. Nair. (2016). Organ is Free, Transplant Cost is Problem. [Online]. Available: https://timesofindia.indiatimes.com/life-style/ healthfitness/health-news/Organ-is-free-transplant-cost-isproblem/ articleshow/54014378.cms

[13] P. Ranjan, S. Srivastava, V. Gupta, S. Tapaswi, and N. Kumar, ‘‘Decentralised and distributed system for organ/tissue donation and transplantation,’’ in Proc. IEEE Conf. Inf. Commun. Technol., Dec. 2019, pp. 1–6, doi: 10.1109/cict48419. 2019.9066225.

[14] V. Puggioni. (Feb. 26, 2022). An Overview of the Blockchain Development Lifecycle. Cointelegraph. Accessed: Apr. 8, 2022. [Online]. Available: https://cointelegraph.com/explained/an-overview-of-the-blockchaindevelopment-lifecycle

[15] History of Blockchain. Accessed: Apr. 8, 2022. [Online]. Available: https://www.icaew.com/technical/technology/blockchain-andcryptoassets/blockchain-articles/what-is-blockchain/history

[16] M. Hölbl, M. Kompara, A. Kamišalić, and L. N. Zlatolas, ‘‘A systematic review of the use of blockchain in healthcare,’’ Symmetry, vol. 10, no. 10, p. 470, Oct. 2018, doi: 10.3390/sym10100470.

[17] V. Ferraza, G. Oliveira, P. Viera-Marques, and R. Cruz-Correia, ‘‘Organs transplantation—How to improve the process ?’’ Eur. Fed. Med. Inform., Cardiff, U.K., Tech. Rep., 2011, doi: 10.3233/978-1-60750-806-9-300.

[18] Organ Procurement and Transplantation Network. Accessed: Nov. 27, 2021. [Online]. Available: https://optn.transplant.hrsa.gov/governance/ public-comment/standardize-organ-coding-and-tracking-system/

[19] A. Bougdira, A. Ahaitouf, and I. Akharraz, ‘‘Conceptual framework for general traceability solution: Description and bases,’’ J. Model. Manage., vol. 15, no. 2, pp. 509–530, Oct. 2019.

[20] N. Mattei, A. Saffidine, and T. Walsh, ‘‘Mechanisms for online organ matching,’’ in Proc. 26th Int. Joint Conf. Artif. Intell., Aug. 2017, pp. 345–351, doi: 10.24963/ijcai.2017/49.

[21] S. Zouarhi, ‘‘Kidner—A worldwide decentralised matching system for kidney transplants,’’ J. Int. Soc. Telemed. E-Health, vol. 5, Apr. 2017, Art. no. e62. [Online]. Available: https://journals.ukzn.ac.za/index.php/JISfTeH/article/view/287

[22] Kidner Project. Accessed: Dec. 28, 2021. [Online]. Available: https://www.kidner-project.com/

[23] L. A. Dajim, S. A. Al-Farras, B. S. Al-Shahrani, A. A. Al-Zuraib, and R. M. Mathew, ‘‘Organ donation decentralized application using blockchain technology,’’ in Proc. 2nd Int. Conf. Comput. Appl. Inf. Secur. (ICCAIS), May 2019, pp. 1–4, doi: 10.1109/cais.2019.8769459.

[24] A. Soni and S. G. Kumar, ‘‘Creating organ donation system with blockchain technology,’’ Eur. J. Mol. Clin. Med., vol. 8, no. 3, pp. 2387–2395, Apr. 2021.

[25] G. Alandjani, ‘‘Blockchain based auditable medical transaction scheme for organ transplant services,’’ Tech. Rep., 2019, doi: 10.17993/3ctecno.2019.specialissue3.

12. BIBLIOGRAPHY

W3 School – (https://www.w3schools.com/python/)

Geek for Geeks – (https://www.geeksforgeeks.org/python-programming-language/learn-python-tutorial/)

Python Official Documentation – (https://docs.python.org/3/tutorial/)

Tutorials Point – (https://www.tutorialspoint.com/python/index.htm)

Real Python – (https://realpython.com/)

Django for Beginners – (https://djangoforbeginners.com/introduction/)

Guru99 – (https://www.guru99.com/django-tutorial.html)