**BLOCKCHAIN-BASED MANAGEMENT OF BLOOD DONATION**

**ABSTRACT:**

Today’s a large number of blood donation management systems fall short in providing traceability, immutability, transparency, audit, privacy, and security features. Also, they are vulnerable to the single point of failure problem due to centralization. In this paper, we propose a private Ethereum blockchain-based solution to automate blood donation management in a manner that is decentralized, transparent, traceable, auditable, private, secure, and trustworthy. The proposed solution stores non-critical and large data off-chain using the decentralized storage of the InterPlanetary File System (IPFS). We present the system architecture, sequence diagrams, entity-relationship diagram, and algorithms to briefly explain the working principles of our blood donation management solution. We evaluate the performance of our solution in terms of efficiency and effectiveness through performing security analysis. We make our smart contract code publicly available on Github1 .

**INTRODUCTION:**

Blood is one of the most crucial fluids in the human body. It contributes in aiding the organs with the essential and valuable substances required for living. Since the demand for blood surpasses all other medical necessities, governments often educate their citizens on the importance of blood donation through organizing awareness programs. The number of donors in the years 2018-2019 were estimated to be 136,908 donors, contributing to a total of 216,639 donations [1], [2]. In general, every 56 days, mostly healthy individuals give blood donations [3]. The World Health Organization (WHO) estimates that the annual amount of blood donations collected is 112.5 million units which is approximately 50 million liters per year [4]. Yet, the shortage of blood donors has risen with the emergence of new diseases, raising the need to enable reliable and efficient blood donation management [5].

Patient Blood Management (PBM) is a vast and challenging task. The restrictions and gaps occurring with the current blood management system limit the efficient performance of the supply chain. Hannon [6] reported that the blood component wastage rates usually run from 1% to 5%, and that the amount of disposal is not shared or visible to clarify the reason behind it. Thus, any improvement or development is a significant factor in providing effective healthcare worldwide. Figure 1 illustrates a typical flow process of blood donation. First, donors have three options to donate blood. Option 1, through healthcare centers where blood units are transported to the nearest blood bank. Option 2, through mobile blood collection units. Option 3, directly through blood banks.

After that, separation, testing, and storage are operated on each donated blood in the blood bank. The separation process is based on separating whole blood units into components of red cells, platelets, and plasma through centrifuges. Next, testing is performed to verify the blood type and indicate any infectious diseases. When test results are established, units proper for transfusion are then labeled and stored either in refrigerators and freezer lockers or in walk-in cool and freeze rooms [7]. According to the Food and Drug Administration (FDA) and the American Association of Blood Banks (AABB) standards, red blood cells are stored in refrigerators at 6◦C and have an expiration date of 42 days. As per the FDA requirements, plasma is frozen in freezers for up to one year, and platelets are stored for up to five days at room temperature [8], [9].

Subsequently, blood units are packaged and loaded to transporters based on doctors’ orders for their patients’ treatment. Finally, after healthcare centers receive blood units, they further transfuse to patients [10]. Blood-related information might range from blood type to blood state, to the donor’s health record, when the donor donated the blood, and other related readings. Despite the benefit of interblood bank transfers, hospitals commonly fear the chance of receiving the wrong blood, or even worse, blood infected with hepatitis, HIV, or other similar diseases [11].

Several risks are associated with carrying infective donated blood in the supply chain. An epidemic in the late 80s occurred because polluted blood infected with HIV was carried out in the supply chain [12]. Nonetheless, counterfeiting and forgery of medicinal products are considered as another concern in the supply chain, where a falsified illegal copy of an original product can be replaced with the original product itself. This results in the possibility of replacing the blood with another type or attaching a false label to cover up the existing effective blood [13].

These are major obstacles in the supply chain management system which caught the attention of researchers and other interested parties. Blockchain-based technology in the blood supply chain can assist in reducing the aforementioned risks. The emerging technology possesses several solutions for the verification of the origin of the donated blood. This can be made possible by tracing the source information of the donors in a trusted manner throughout the stages of the supply chain. Several countries across the world have emphasized the importance of traceability and mandated its existence in healthcare supply chains. The Drug Supply Chain Security Act (DSCSA) in the United States, has obligated pharmaceutical industries to inherent electronic systems that will identify prescriptive drugs while being distributed across the country [14]. Similarly, China requires all stakeholders in healthcare supply chains to use a specialized IT system to be able to record the information of products whenever they are sent to or from their warehouses [15]

. Subsequently, several supply chains integrated traceability as an important part of establishing authenticity across their chains. This paper proposes a blockchain-based blood donation chain management system to address the aforementioned issues. The main contributions of this paper are as follows:

• We propose a private Ethereum blockchain-based solution to automate blood donation management processes in a manner that is fully decentralized, traceable, transparent, auditable, private, secure, and trustworthy.

• We integrate the private Ethereum blockchain with the decentralized storage of the InterPlanetary File System (IPFS) to overcome the storage limitations.

• We develop two smart contracts along with algorithms to implement functionalities and define rules regarding blood donation management.

• We evaluate the proposed blockchain-based blood donation management solution and the developed smart contracts using the security analyses. Also, we compare our proposed approach with the existing solutions.

• Our proposed blockchain-based blood donation management solution is generic and can be customized to meet the needs of other industrial applications with minimal modifications and efforts. The rest of the paper is structured as follows. Section II and III present the background and related work which is classified into two categories;

namely, blockchain and non-blockchain based blood donation management solutions. Section IV describes the proposed blockchain-based solution for blood donation management. Section V and VI present the implementation, testing, and validation details of the proposed solution. In Section VII, we conduct the security analysis of the proposed approach. Section VIII summarizes our findings along with concluding remarks.1.

**EXISITNG SYSTEM**

As per the FDA requirements, plasma is frozen in freezers for up to one year, and platelets are stored for up to five days at room temperature [8], [9]. Subsequently, blood units are packaged and loaded to transporters based on doctors’ orders for their patients’ treatment. Finally, after healthcare centers receive blood units, they further transfuse to patients [10]. Blood-related information might range from blood type to blood state, to the donor’s health record, when the donor donated the blood, and other related readings. Despite the benefit of inter blood bank transfers, hospitals commonly fear the chance of receiving the wrong blood, or even worse, blood infected with hepatitis, HIV, or other similar diseases [11]. Several risks are associated with carrying infective donated blood in the supply chain. An epidemic in the late 80s occurred because polluted blood infected with HIV was carried out in the supply chain [12]. Nonetheless, counterfeiting and forgery of medicinal products are considered as another concern in the supply chain, where a falsified illegal copy of an original product can be replaced with the original product itself. This results in the possibility of replacing the blood with another type or attaching a false label to cover up the existing effective blood [13]. These are major obstacles in the supply chain management system which caught the attention of researchers and other interested parties.

**PROPOSED SYSTEM**

This paper proposes a blockchain-based blood donation chain management system to address the aforementioned issues.

• We propose a private Ethereum blockchain-based solution to automate blood donation management processes in a manner that is fully decentralized, traceable, transparent, auditable, private, secure, and trustworthy.

• We integrate the private Ethereum blockchain with the decentralized storage of the InterPlanetary File Sytem (IPFS) to overcome the storage limitations.

• We develop two smart contracts along with algorithms to implement functionalities and define rules regarding blood donation management.

• We evaluate the proposed blockchain-based blood donation management solution and the developed smart contracts using the security analyses. Also, we compare our proposed approach with the existing solutions.

• Our proposed blockchain-based blood donation man agement solution is generic and can be customized to meet the needs of other industrial applications with minimal modifications and efforts.

**SYSTEM STUDIES:**

An important outcome of preliminary investigation is the determination that the system request is feasible. This is possible only if it is feasible within limited resource and time. The different feasibilities that have to be analyzed are

• OPERATIONAL FEASIBILITY

• ECONOMIC FEASIBILITY

• TECHNICAL FEASIBILITY

**Operational Feasibility**

Operational Feasibility deals with the study of prospects of the system to be developed. This system operationally eliminates all the tensions of the Admin and helps him in effectively tracking the project progress. This kind of automation will surely reduce the time and energy, which previously consumed in manual work. Based on the study, the system is proved to be operationally feasible.

**Economic Feasibility**

Economic Feasibility or Cost-benefit is an assessment of the economic justification for a computer based project. As hardware was installed from the beginning & for lots of purposes thus the cost on project of hardware is low. Since the system is a network based, any number of employees connected to the LAN within that organization can use this tool from at any time. The Virtual Private Network is to be developed using the existing resources of the organization. So the project is economically feasible.

**Technical Feasibility**

According to Roger S. Pressman, Technical Feasibility is the assessment of the technical resources of the organization. The organization needs IBM compatible machines with a graphical web browser connected to the Internet and Intranet. The system is developed for platform Independet environment. Java Server Pages, JavaScript, HTML, SQL server and Web Logic Server are used to develop the system. The technical feasibility has been carried out. The system is technically feasible for development and can be developed with the existing.

**INPUT DESIGN:**

Input Design plays a vital role in the life cycle of software development, it requires very careful attention of developers. The input design is to feed data to the application as accurate as possible. So inputs are supposed to be designed effectively so that the errors occurring while feeding are minimized. According to Software Engineering Concepts, the input forms or screens are designed to provide to have a validation control over the input limit, range and other related validations.

This system has input screens in almost all the modules. Error messages are developed to alert the user whenever he commits some mistakes and guides him in the right way so that invalid entries are not made. Let us see deeply about this under module design.

Input design is the process of converting the user created input into a computer-based format. The goal of the input design is to make the data entry logical and free from errors. The error is in the input are controlled by the input design. The application has been developed in user-friendly manner. The forms have been designed in such a way during the processing the cursor is placed in the position where must be entered. The user is also provided within an option to select an appropriate input from various alternatives related to the field in certain cases.

Validations are required for each data entered. Whenever a user enters an erroneous data, error message is displayed and the user can move on to the subsequent pages after completing all the entries in the current page.

**OUTPUT DESIGN:**

The Output from the computer is required to mainly create an efficient method of communication within the company primarily among the project leader and his team members, in other words, the administrator and the clients. The output of VPN is the system which allows the project leader to manage his clients in terms of creating new clients and assigning new projects to them, maintaining a record of the project validity and providing folder level access to each client on the user side depending on the projects allotted to him. After completion of a project, a new project may be assigned to the client. User authentication procedures are maintained at the initial stages itself. A new user may be created by the administrator himself or a user can himself register as a new user but the task of assigning projects and validating a new use rests with the administrator only.

The application starts running when it is executed for the first time. The server has to be started and then the internet explorer in used as the browser. The project will run on the local area network so the server machine will serve as the administrator while the other connected systems can act as the clients. The developed system is highly user friendly and can be easily understood by anyone using it even for the first time

**SOFTWARE ENVIRONMENT:**

**Java Technology**

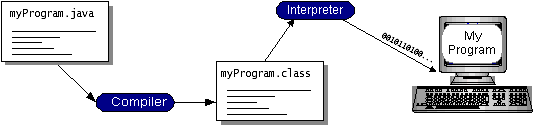
Java technology is both a programming language and a platform.

**The Java Programming Language**

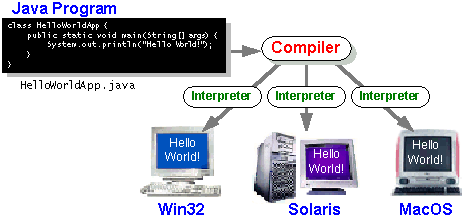
The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

* + - Simple
    - Architecture neutral
    - Object oriented
    - Portable
    - Distributed
    - High performance
    - Interpreted
    - Multithreaded
    - Robust
    - Dynamic
    - Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



You can think of Java byte codes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it’s a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make “write once, run anywhere” possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



**6.2 Java Platform**

A *platform* is the hardware or software environment in which a program runs. We’ve already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it’s a software-only platform that runs on top of other hardware-based platforms.

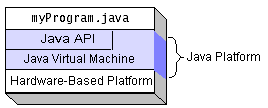
The Java platform has two components:

* The *Java Virtual Machine* (Java VM)
* The *Java Application Programming Interface* (Java API)

You’ve already been introduced to the Java VM. It’s the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section what Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that’s running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

**What Can Java Technology Do?**

The most common types of programs written in the Java programming language are *applets* and *applications*. If you’ve surfed the Web, you’re probably already familiar with applets. An apple is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

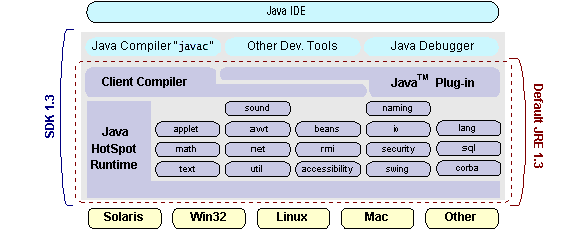
However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers are Web Servlets proxy servers, mail servers, and print servers. Another specialized program is a *servlet* A servlets can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, serviettes run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

* **The essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
* **Applets**: The set of conventions used by applets.
* **Networking**: URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.
* **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
* **Security**: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
* **Software components**: Known as JavaBeans can plug into existing component architectures.
* **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
* **Java Database Connectivity (JDBCTM)**: Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.



**How Will Java Technology Change My Life*?***

We can’t promise you fame, fortune, or even a job if you learn the Java programming language. Still, it is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

* **Get started quickly**: Although the Java programming language is a powerful object-oriented language, it’s easy to learn, especially for programmers already familiar with C or C++.
* **Write less code**: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
* **Write better code**: The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people’s tested code and introduce fewer bugs.
* **Develop programs more quickly**: Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.
* **Avoid platform dependencies with 100% Pure Java**: You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure Java Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.
* **Write once, run anywhere**: Because 100% Pure Java programs are compiled into machine-independent byte codes, they run consistently on any Java platform.
* **Distribute software more easily**: You can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded “on the fly,” without recompiling the entire program.

**ODBC**

Microsoft Open Database Connectivity (ODBC) is a standard programming interface for application developers and database systems providers. Before ODBC became a *de facto* standard for Windows programs to interface with database systems, programmers had to use proprietary languages for each database they wanted to connect to. Now, ODBC has made the choice of the database system almost irrelevant from a coding perspective, which is as it should be. Application developers have much more important things to worry about than the syntax that is needed to port their program from one database to another when business needs suddenly change.

Through the ODBC Administrator in Control Panel, you can specify the particular database that is associated with a data source that an ODBC application program is written to use. Think of an ODBC data source as a door with a name on it. Each door will lead you to a particular database. For example, the data source named Sales Figures might be a SQL Server database, whereas the Accounts Payable data source could refer to an Access database. The physical database referred to by a data source can reside anywhere on the LAN.

The ODBC system files are not installed on your system by Windows 95. Rather, they are installed when you setup a separate database application, such as SQL Server Client or Visual Basic 4.0. When the ODBC icon is installed in Control Panel, it uses a file called ODBCINST.DLL. It is also possible to administer your ODBC data sources through a stand-alone program called ODBCADM.EXE. There is a 16-bit and a 32-bit version of this program and each maintains a separate list of ODBC data sources.

From a programming perspective, the beauty of ODBC is that the application can be written to use the same set of function calls to interface with any data source, regardless of the database vendor. The source code of the application doesn’t change whether it talks to Oracle or SQL Server. We only mention these two as an example. There are ODBC drivers available for several dozen popular database systems. Even Excel spreadsheets and plain text files can be turned into data sources. The operating system uses the Registry information written by ODBC Administrator to determine which low-level ODBC drivers are needed to talk to the data source (such as the interface to Oracle or SQL Server). The loading of the ODBC drivers is transparent to the ODBC application program. In a client/server environment, the ODBC API even handles many of the network issues for the application programmer.

The advantages of this scheme are so numerous that you are probably thinking there must be some catch. The only disadvantage of ODBC is that it isn’t as efficient as talking directly to the native database interface. ODBC has had many detractors make the charge that it is too slow. Microsoft has always claimed that the critical factor in performance is the quality of the driver software that is used. In our humble opinion, this is true. The availability of good ODBC drivers has improved a great deal recently. And anyway, the criticism about performance is somewhat analogous to those who said that compilers would never match the speed of pure assembly language. Maybe not, but the compiler (or ODBC) gives you the opportunity to write cleaner programs, which means you finish sooner. Meanwhile, computers get faster every year.

**JDBC**

In an effort to set an independent database standard API for Java; Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of “plug-in” database connectivity modules, or *drivers*. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC’s framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90-day public review that ended on June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

**JDBC Goals**

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

1. **SQL Level API**

The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to “generate” JDBC code and to hide many of JDBC’s complexities from the end user.

1. **SQL Conformance**

SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.

1. **JDBC must be implemented on top of common database interfaces**  
    The JDBC SQL API must “sit” on top of other common SQL-level APIs. This goal allows JDBC to use existing ODBC-level drivers through the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.
2. **Provide a Java interface that is consistent with the rest of the Java system**

Because of Java’s acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.

1. **Keep it simple**

This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.

1. **Use strong, static typing wherever possible**

Strong typing allows for more error checking to be done at compile time; it also, appears at runtime.

1. **Keep the common cases simple**

Because more often than not, the usual SQL calls used by the programmer are simple SELECTs, INSERTs, DELETE and UPDATE, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible. Finally, we decided to proceed with the implementation using Java Networking. And for dynamically updating the cache table we go for the MS Access database.

Java has two things: a programming language and a platform.

Java is a high-level programming language that is all of the following

Simple Architecture-neutral

Object-oriented Portable

Distributed High-performance

Interpreted multithreaded

Robust Dynamic

Secure

Java is also unusual in that each Java program is both compiled and interpreted. With a compile you translate a Java program into an intermediate language called Java byte codes the platform-independent code instruction is passed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The figure illustrates how this works.

**JavaProgram**

**Compilers**

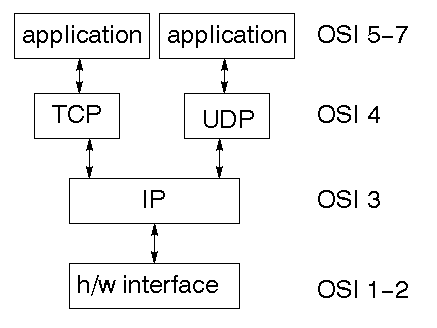
**Interpreter**

**My Program**

You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it’s a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware. Java byte codes help make “write once, run anywhere” possible. You can compile your Java program into byte codes on my platform which has a Java compiler. The byte codes can then be run in any implementation of the Java VM. For example, the same Java program can run Windows NT, Solaris, and Macintosh.

**Networking**

**TCP/IP stack :**The TCP/IP stack is shorter than the OSI one:



TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

**IP datagram’s**

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram’s must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through the Internet. It is also responsible for breaking up large datagram’s into smaller ones for transmission and reassembling them at the other end.

**UDP**

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

**TCP**

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

**Internet addresses**

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32-bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

**Network address**

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses a 16-bit network address. Class C uses 24-bit network addressing and class D uses all 32.

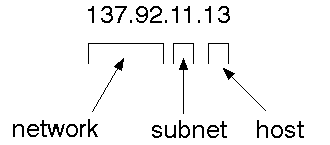
**Subnet address**

Internally, the UNIX network is divided into sub-networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

**Host address**

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

**Total address**



The 32-bit address is usually written as 4 integers separated by dots.

**Port addresses**

A service exists on a host and is identified by its port. This is a 16-bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

**Sockets**

A socket is a data structure maintained by the system to handle network connections. A socket is created using the calling socket. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with Read File and Write File functions.

#include <sys/types>

#include <sys/socket>

Int socket (int family, int type, int protocol);

Here "family" will be AF\_INET for IP communications, the protocol will be zero, and the type will depend on whether TCP or UDP is used. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

**6.3 JFree Chart**

is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. Extensive feature set includes:

A consistent and well-documented API, supporting a wide range of chart types;

A flexible design that is easy to extend, and targets both server-side and client-side applications;

Support for many output types, including Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS, and SVG);

JFree Chart is "open source" or, more specifically, [free software](http://www.gnu.org/philosophy/free-sw.html). It is distributed under the terms of the [GNU Lesser General Public License](http://www.gnu.org/licenses/lgpl.html) (LGPL), which permits use in proprietary applications.

***1.* Map Visualizations**

Charts showing values that relate to geographical areas. Some examples include (a) population density in each state of the United States, (b) income per capita for each country in Europe, and (c) life expectancy in each country of the world. The tasks in this project include:

Sourcing freely redistributable vector outlines for the countries of the world, states/provinces in particular countries (USA in particular, but also other areas);

Creating an appropriate dataset interface (plus default implementation), a rendered, and integrating this with the existing XY Plot class in JFree Chart;

Testing, documenting, testing some more, documenting some more.

***2.* Time Series Chart Interactivity**

Implement a new (to JFree Chart) feature for interactive time series charts --- to display a separate control that shows a small version of ALL the time series data, with a sliding "view" rectangle that allows you to select the subset of the time series data to display in the main chart.

**3. Dashboards**

There is currently a lot of interest in dashboard displays. Create a flexible dashboard mechanism that supports a subset of JFree Chart types (dials, pies, thermometers, bars, and lines/time series) that can be delivered easily via both Java Web Start and an applet.

**4. Property Editors**

The property editor mechanism in JFree Chart only handles a small subset of the properties that can be set for charts. Extend (or reemployment) this mechanism to provide greater end-user control over the appearance of the charts.

**J2ME (Java 2 Micro edition):-**

Sun Microsystems defines J2ME as "a highly optimized Java run-time environment targeting a wide range of consumer products, including pagers, cellular phones, screen-phones, digital set-top boxes, and car navigation systems." Announced in June 1999 at the JavaOne Developer Conference, J2ME brings the cross-platform functionality of the Java language to smaller devices, allowing mobile wireless devices to share applications. With J2ME, Sun has adapted the Java platform for consumer products that incorporate or are based on small computing devices.

**1. General J2ME architecture**



J2ME uses configurations and profiles to customize the Java Runtime Environment (JRE). As a complete JRE, J2ME is comprised of a configuration, which determines the JVM used, and a profile, which defines the application by adding domain-specific classes. The configuration defines the basic run-time environment as a set of core classes and a specific JVM that run on specific types of devices. We'll discuss configurations in detail in the profile that defines the application; specifically, it adds domain-specific classes to the J2ME configuration to define certain uses for devices. We'll cover profiles in depth in the following graphic depicts the relationship between the different virtual machines, configurations, and profiles. It also draws a parallel with the J2SE API and its Java virtual machine. While the J2SE virtual machine is generally referred to as a JVM, the J2ME virtual machines, KVM and CVM, are subsets of JVM. Both KVM and CVM can be thought of as a kind of Java virtual machine -- it's just that they are shrunken versions of the J2SE JVM and are specific to J2ME.

1. **Developing J2ME applications**

Introduction In this section, we will go over some considerations you need to keep in mind when developing applications for smaller devices. We'll take a look at the way the compiler is invoked when using J2SE to compile J2ME applications. Finally, we'll explore packaging and deployment and the role pre-verification plays in this process.

1. **Design considerations for small devices**

Developing applications for small devices requires you to keep certain strategies in mind during the design phase. It is best to strategically design an application for a small device before you begin coding. Correcting the code because you failed to consider all of the "gotchas" before developing the application can be a painful process. Here are some design strategies to consider:

\* Keep it simple. Remove unnecessary features, possibly making those features a separate, secondary application.

\* Smaller is better. This consideration should be a "no-brainer" for all developers. Smaller applications use less memory on the device and require shorter installation times. Consider packaging your Java applications as compressed Java Archive (jar) files

.

\* Minimize run-time memory use. To minimize the amount of memory used at run time, use scalar types in place of object types. Also, do not depend on the garbage collector. You should manage the memory efficiently yourself by setting object references to null when you are finished with them. Another way to reduce run-time memory is to use lazy instantiation, only allocating objects on an as-needed basis. Other ways of reducing overall and peak memory use on small devices are to release resources quickly, reuse objects, and avoid exceptions.

**4. Configurations overview**

The configuration defines the basic run-time environment as a set of core classes and a specific JVM that run on specific types of devices. Currently, two configurations exist for J2ME, though others may be defined in the future:

\* **Connected Limited Device Configuration (CLDC)**is used specifically with the KVM for 16-bit or 32-bit devices with limited amounts of memory. This is the configuration (and the virtual machine) used for developing small J2ME applications. Its size limitations make CLDC more interesting and challenging (from a development point of view) than CDC. CLDC is also the configuration that we will use for developing our drawing tool application. An example of a small wireless device running small applications is a Palm hand-held computer.

\* **Connected Device Configuration (CDC)**is used with the C virtual machine (CVM) and is used for 32-bit architectures requiring more than 2 MB of memory. An example of such a device is a Net TV box.

**5. J2ME profiles**

**What is a J2ME profile?**

As we mentioned earlier in this tutorial, a profile defines the type of device supported. The Mobile Information Device Profile (MIDP), for example, defines classes for cellular phones. It adds domain-specific classes to the J2ME configuration to define uses for similar devices. Two profiles have been defined for J2ME and are built upon CLDC: KJava and MIDP. Both KJava and MIDP are associated with CLDC and smaller devices. Profiles are built on top of configurations. Because profiles are specific to the size of the device (amount of memory) on which an application runs, certain profiles are associated with certain configurations. A skeleton profile upon which you can create your own profile, the Foundation Profile, is available for CDC.

**Profile 1: KJava**

KJava is Sun's proprietary profile and contains the KJava API. The KJava profile is built on top of the CLDC configuration. The KJava virtual machine, KVM, accepts the same byte codes and class file format as the classic J2SE virtual machine. KJava contains a Sun-specific API that runs on the Palm OS. The KJava API has a great deal in common with the J2SE Abstract Windowing Toolkit (AWT). However, because it is not a standard J2ME package, its main package is com.sun. Java We'll learn more about the KJava API later in this tutorial when we develop some sample applications.

**Profile 2: MIDP**

MIDP is geared toward mobile devices such as cellular phones and pagers. The MIDP, like KJava, is built upon CLDC and provides a standard run-time environment that allows new applications and services to be deployed dynamically on end-user devices. MIDP is a common, industry-standard profile for mobile devices that is not dependent on a specific vendor. It is a complete and supported foundation for mobile application

development. MIDP contains the following packages, the first three of which are core CLDC packages, plus three MIDP-specific packages.

\* java.lang

\* java.io

\* java.util

\* javax.microedition.io

\* javax.microedition.lcdui

\* javax.microedition.midlet

\* javax.microedition.rms

**SYSTEM DESIGN**

**UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for it to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: aMeta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

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:

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. Role of the actors in the system can be depicted.

**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.

**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.

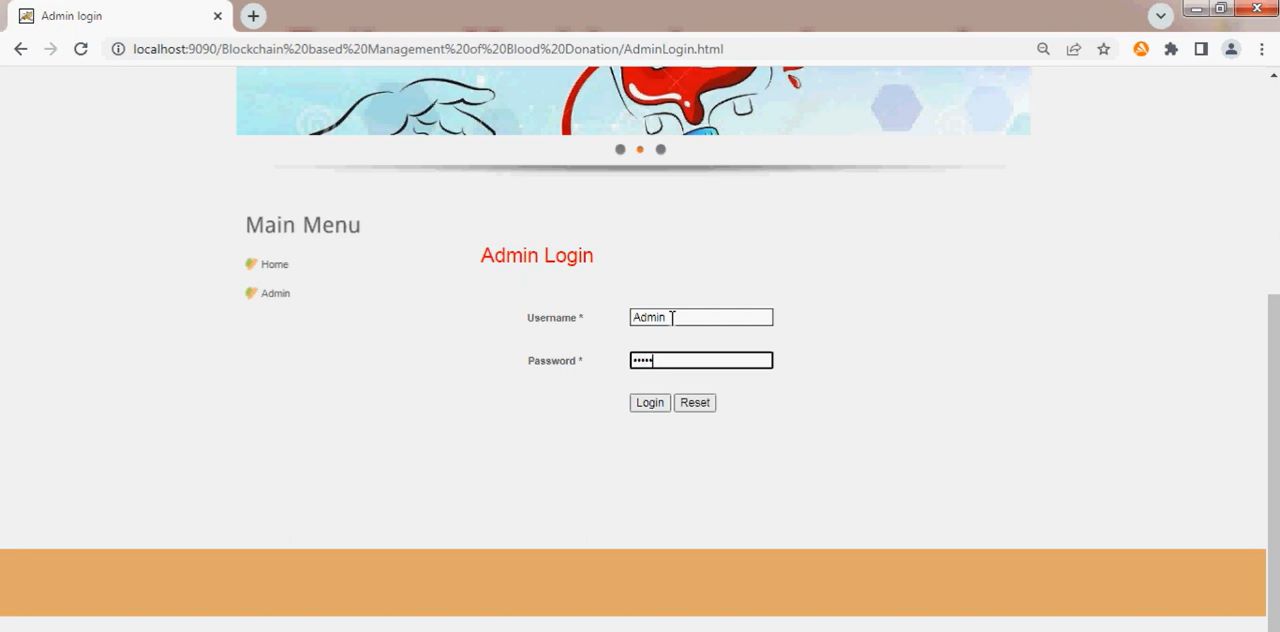
**ACTIVITY DIAGRAM:**

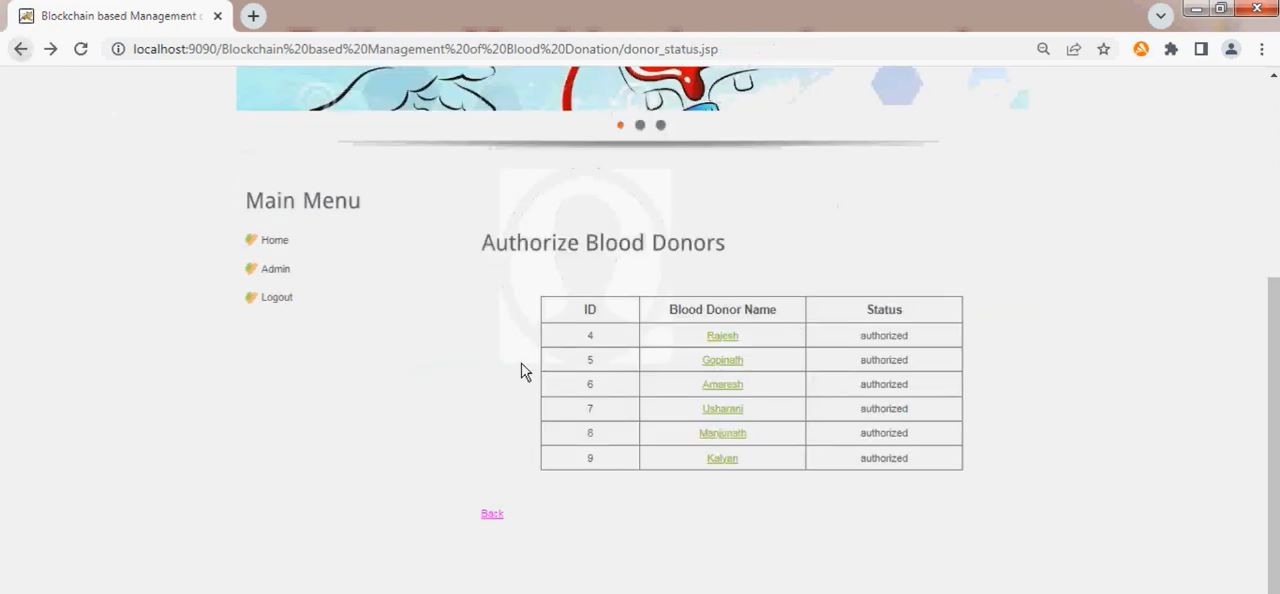
Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system.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

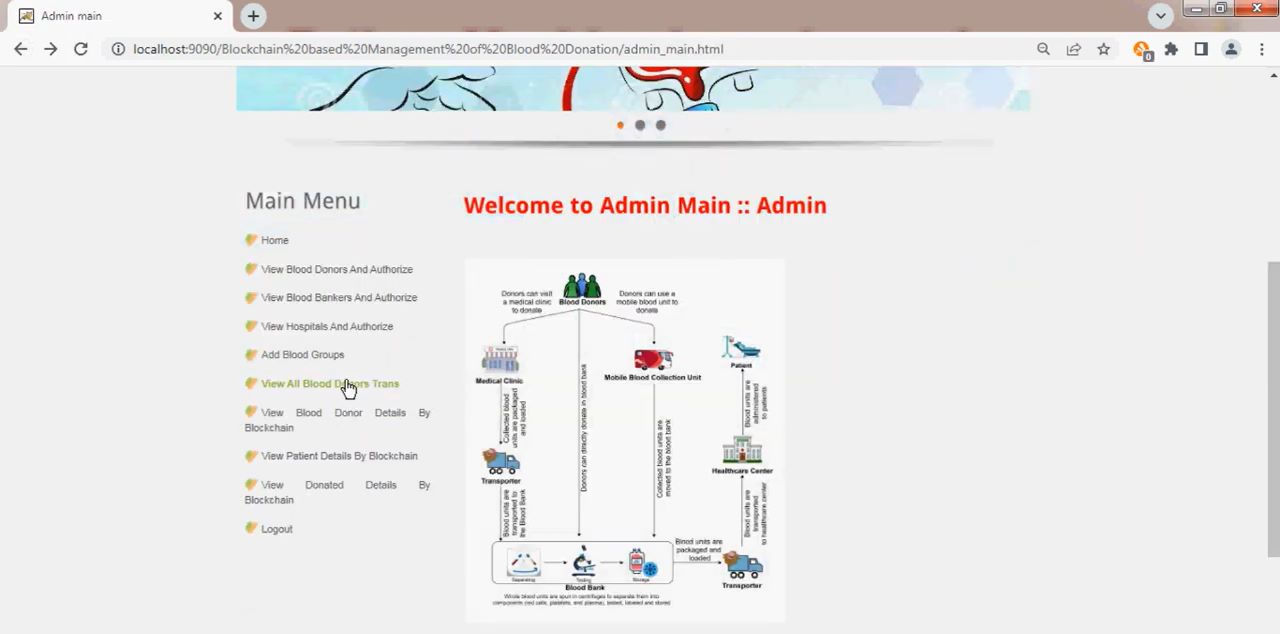
**SYSTEM TESTING:**

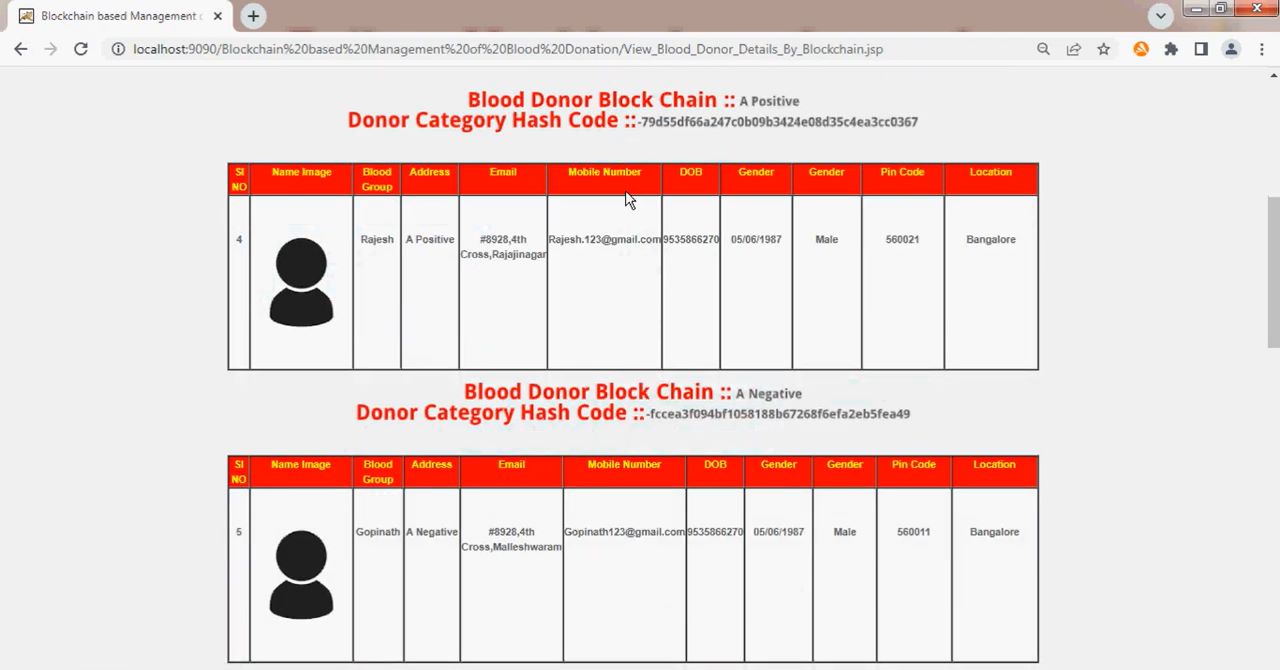
Software once validated must be combined with other system elements (e.g. Hardware, people, database). System testing verifies that all the elements are proper and that overall system function performance is achieved. It also tests to find discrepancies between the system and its original objective, current specifications, and system documentation

**RESULT:**

****

****

****

****

**CONCLUSIONS:**

In this paper, we have proposed a blockchain-based blood donation management system that traces the origin of the blood in a transparent, private, secure, trustworthy, auditable, and decentralized manner. The proposed solution employed the smart contract feature of the private Ethereum blockchain to record and log events automatically. We integrated the private Ethereum blockchain with the IPFS to deal with the limited storage issue. We tested and validated the functionality of our solution using the Remix IDE. Our developed smart contracts’ code has been made available on the Github repository. We conducted the security analysis to show that the proposed blood donation management solution is robust and secure enough against major security vulnerabilities and attacks. In addition, we compared our proposed approach with the existing solutions. In the future, we aim to deploy and test our solution on the real Ethereum network and build an end-to-end DApp. Furthermore, violation monitoring will be added to further enhance the security of the blood cold supply chain.

**REFERENCES:**

[1] Why Blood Donation Are So Important. Accessed: Jan. 20, 2021. [Online]. Available: https://www.orlandohealth.com/content-hub/why-blooddonations-are-so-important

[2] FeelGood: A Blood Donation System Based on Smart Contracts. Accessed: Mar. 20, 2021. [Online]. Available: https://www.northeastern. edu/rise/presentations/feelgood-a-blood-donation-system-based-onsmart-contracts/

[3] Dubai Health Authority. Accessed: Apr. 20, 2021. [Online]. Available: https://www.dha.gov.ae/en/SpecialtyCentres/DubaiBloodDonationCenter/ Pages/FAQ.aspx

[4] Sensire. Blood Cold Chain—The Challenges of Saving Lives. Accessed: Apr. 24, 2021]. [Online]. Available: https://www.sensire. com/blog/blood-cold-chain-the-challenges-of-saving-lives

[5] S. Kim, J. Kim, and D. Kim, ‘‘Implementation of a blood cold chain system using blockchain technology,’’ Appl. Sci., vol. 10, no. 9, p. 3330, May 2020, doi: 10.3390/app10093330.

[6] T. Hannon, ‘‘Waste not, want not,’’ Amer. J. Clin. Pathol., vol. 143, no. 3, pp. 318–319, Mar. 2015, doi: 10.1309/ajcpm8facvc0rprg.

[7] Design Guidelines Centres—Who. Accessed: Mar. 18, 2021. [Online]. Available: <https://tinyurl.com/mekcr49e>

[8] How Are Blood Component Expiration Dates Determined? Accessed: May 16, 2021. [Online]. Available: [https://stanfordbloodcenter.org/how\_ are\_blood\_c/](https://stanfordbloodcenter.org/how_%20are_blood_c/)

[9] What Happens to Donated Blood. Accessed: May 23, 2021]. [Online]. Available: https://www.redcrossblood.org/donate-blood/blood-donationprocess/what-happens-to-donated-blood.html

[10] MED Alliance Group. (Aug. 4, 2020). The Blood Supply Chain—Learning About the Points of Failure. Accessed: Feb. 12, 2021. [Online]. Available: https://www.medalliancegroup.com/the-blood-supply-chain-learningabout-the-points-of-failure/

[11] S. Lakshminarayanan, P. N. Kumar, and N. M. Dhanya, Implementation of Blockchain-Based Blood Donation Framework (IFIP Advances in Information and Communication Technology), vol. 578. Cham, Switzerland: Springer, 2020, pp. 276–290, doi: 10.1007/978-3-030-63467-4\_22.

[12] A. Farrugia, ‘‘Safety issues of plasma-derived products for treatment of inherited bleeding disorders,’’ Seminars Thrombosis Hemostasis, vol. 42, no. 5, pp. 583–588, Jul. 2016.

[13] S. Sadri, A. Shahzad, and K. Zhang, ‘‘Blockchain traceability in healthcare: Blood donation supply chain,’’ in Proc. 23rd Int. Conf. Adv. Commun. Technol. (ICACT), Feb. 2021, pp. 119–126, doi: 10.23919/icact51234.2021.9370704.

[14] U.S. Food and Drug Administration. Drug Supply Chain Security Act. Accessed: Jul. 16, 2021. [Online]. Available: https://www.fda. gov/drugs/drug-supply-chain-integrity/drug-supply-chain-security-actdscsa

[15] A. Musamih, K. Salah, R. Jayaraman, J. Arshad, M. Debe, Y. Al-Hammadi, and S. Ellahham, ‘‘A blockchain-based approach for drug traceability in healthcare supply chain,’’ IEEE Access, vol. 9, pp. 9728–9743, 2021, doi: 10.1109/access.2021.3049920.

[16] U. Hacioglu, Digital Business Strategies in Blockchain Ecosystems: Transformational Design and Future of Global Business. Cham, Switzerland: Springer, 2020, p. 74.

[17] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, ‘‘BLOCKBENCH: A framework for analyzing private blockchains,’’ in Proc. ACM Int. Conf. Manage. Data, May 2017, pp. 1085–1100.

[18] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, and Y. Zhang, ‘‘Consortium blockchain for secure energy trading in industrial Internet of Things,’’ IEEE Trans. Ind. Informat., vol. 14, no. 8, pp. 3690–3700, Aug. 2018.

[19] S. Kim and D. Kim, ‘‘Design of an innovative blood cold chain management system using blockchain technologies,’’ ICIC Exp. Lett. B, Appl., vol. 9, pp. 1067–1073, Oct. 2018, doi: 10.24507/icicelb.09.10.1067.

[20] How Blockchain is Helping Make Every Blood Donation More Effective? Accessed: Feb. 22, 2021. [Online]. Available: https://www.ey.com/en\_ ae/better-begins-with-you/how-blockchain-could-ensure-every-drop-ofblood-is-tracked-and-every-outcome-is-measured

[21] K. E. Kendall, ‘‘A decentralized information and control system for blood management,’’ J. Syst. Softw., vol. 1, pp. 299–306, Jan. 1979, doi: 10.1016/0164-1212(79)90031-1.

[22] X. Zhu, S. K. Mukhopadhyay, and H. Kurata, ‘‘A review of RFID technology and its managerial applications in different industries,’’ J. Eng. Technol. Manage., vol. 29, no. 1, pp. 152–167, 2012, doi: 10.1016/j.jengtecman.2011.09.011.

[23] R. Davis, B. Geiger, A. Gutierrez, J. Heaser, and D. Veeramani, ‘‘Tracking blood products in blood centres using radio frequency identification: A comprehensive assessment,’’ Vox Sanguinis, vol. 97, no. 1, pp. 50–60, Jul. 2009, doi: 10.1111/j.1423-0410.2009.01174.x.

[24] B Medical Systems. Accessed: Jul. 8, 2021. [Online]. Available: https://www.bmedicalsystems.com/en/wp-content/uploads/ sites/2/2020/12/BMS-Brochure-Blood-Management-Solutions-EN.pdf

[25] Problems With RFID. Accessed: Feb. 6, 2021. [Online]. Available: <http://www.technovelgy.com/ct/Technology-Article.asp?ArtNum=20>

[26] D. da Silva, J. Costa, B. Assuncao, V. Kuprych, and C. Teixeira, ‘‘Microservice-based middleware for collaborative supply chain tracing,’’ in Proc. 15th Iberian Conf. Inf. Syst. Technol. (CISTI), Jun. 2020, pp. 1–6, doi: 10.23919/CISTI49556.2020.9141036.

[27] F. Alharbi, ‘‘Progression towards an e-management centralized blood donation system in Saudi Arabia,’’ in Proc. Int. Conf. Adv. Emerg. Comput. Technol. (AECT), Feb. 2020, pp. 1–5, doi: 10.1109/AECT47998. 2020.9194178.