Acknowledgment

We are deeply grateful to all of those who have helped and inspired us from time to time during this project, influencing us with different ideas in making this project unique and putting these ideas, well above the level of simplicity and into something concrete.

Abstract

Electronic voting has been employed in many forms with essential advantages over paper-based systems, such as enhanced efficiency and fewer errors. Yet widespread adoption of such systems remains a difficulty, owing to their uncertain efficiency against potential flaws and their reliance on a central authority.

However, distributed ledger technology, such as Blockchain, is an exciting innovation in the area of information technology, it is a state-of-the-art technology based on cryptography that promises to improve the overall resilience of e-voting systems.

This project aims to leverage the features of blockchain technology, such as cryptographic foundations, decentralization, transparency, and consensus mechanisms, to achieve an effective scheme for implementing an intelligent, secured, and trustworthy e-voting system.

We propose a new blockchain-based electronic decentralized voting system that overcomes the issues we observed. In brief, this documentation evaluates distributed ledger technologies by conducting an appropriate review in order to fine-tune the process of political election decisions and deploying a blockchain-based application that enhances security and lowers the cost of holding countrywide elections.

Table of Contents

List of Figures

List of Tables

1. Introduction

Progress in computerizing traditional paper-based systems over the last decades has been astonishing. Our societies wouldn't function in the same way without large professional digital systems. Nowadays national utilities, infrastructure, and communication systems, all depend on complex and mostly reliable computer systems. All these technological advancements and how the information is easily accessible within the reach of our hands, encourage us to develop new software systems to aid us in our daily lives and makes our life more computerized.

One of the most critical systems that plays a key part in our civilized societies where there has always been a daunting challenge to digitize it, is the voting system.

The objective of voting is to allow voters to exercise their right in expressing their choices and opinions regarding general discussions and debates, or specific issues such as pieces of legislation, citizen initiatives, constitutional amendments, or to choose their political representatives. Voting is an essential tool for any democratic system, it's the most significant factor which makes governments for the people and by the people. It is a very crucial matter in terms of choosing the right leader in an election. A good leader can bring prosperity to a country and can lead the country in the right direction.

However, elections are surrounded with various problems as ballot forgery, coercion and relying on manual votes count by humans which increases time, cost, and error, also a lot of security vulnerabilities that threatens the integrity of the election's result. Nowadays it has become very usual for some forces to indulge in rigging elections which may eventually lead to a result contrary to the actual verdict given by the people.

1.1 Problem Statement

Electronic voting (E-Voting) is voting that uses electronic means to either aid or take care of casting and counting ballots. Depending on the particular implementation, e-voting may use standalone electronic voting machines (EVM) or computers connected to the Internet. It may encompass a range of Internet services, from basic transmission of tabulated results to full-function online voting through common connectable household devices.

Electronic voting technology can include punched cards, optical scan voting systems and specialized voting kiosks including self-contained direct-recording electronic voting systems (DRE). It can also involve remote voting thru transmission of ballots and votes via telephones, private computer networks, or the Internet (I-Voting).

The degree of automation may be limited to marking a paper ballot, or may be a comprehensive system of vote input, vote recording, data encryption and transmission to servers, and consolidation and tabulation of election results.

A worthy e-voting system must perform most of these tasks while complying with a set of standards established by regulatory bodies, and must also be capable to deal successfully with strong requirements associated with security, accuracy, integrity, swiftness, privacy, auditability, accessibility, cost-effectiveness, scalability and ecological sustainability.

Hence, developing an electronic voting system that manages to handle these problems has been a challenge for a very long time. In order to provide an efficient solution, we needed to come up with an innovative approach to tackle these problems such as using the blockchain technology accompanied by AI assistance.

Traditional Voting	**Electronic Voting**	**Blockchain-Based E-Voting**
Paper-based	Electronic system	Electronic system
Physical & locally deployed	Centralized system	Distributed system
Forged entries, count problems	Vulnerable and mutable	Immutable and secured
Polling agents and ballots required in polling stations	E-voting machines and web connectivity	Network infrastructure and web connectivity
High operating costs	Low operating costs	Low operating costs
Huge political influence	Low political influence	No political influence
Lacking transparency	More transparency	Greater transparency
Lengthy procedures, delay in the output results	The output results are considerably faster	The output results are shown on runtime

Table (1) - Different Types of Voting

1.2 Objectives

- Automating the voting process leading to reduced cost and time.
- Abolishing the vulnerable centralized obsolete model.
- Providing transparent and verifiable voting procedures.
- Ensuring voter’s identity and preventing fraud.
- Eliminating the human error by reducing involvement.
- Providing secure network with immutable database.

1.3 Scope and Impact

The introduced system will be developed as a distributed network application that’s going to be used by the governing authority as administrators and by the population as end users. This system will have wide coverage. It will facilitate the voting process a lot for both sides, as it will provide easier, faster, and more secure engagement to users.

• Individual Level:

The user will be able to vote in a totally private and entirely secure manner, assured that his vote has been registered as he desired without any possible manipulation.

• Population Level:

The success of the voting system is reliant on achieving high percentage of participants, and on having comprehensive, high-quality information on the characteristics of the elections. At population level, a key aspect of the system is to have an overview on the voting status in a population at a given time period and geographical area.

Technological advancements will encourage people more into shifting for a computerized implementation which will enable the system to achieve greater public acceptance and a wider population involvement. Generally, the role of individuals is mutating as individuals increasingly demand more advanced, secure, and easier ways to be able to perform the traditional procedures which will lead to a more robust and trustworthy system.

2. Related Work

In recent decades, several researchers have been researching on blockchain systems. Satoshi Nakamoto, an anonymous entity, created the first cryptocurrency, Bitcoin, in 2008, paving the way for blockchain technology. The Bitcoin blockchain technology is built on the Proof of Work consensus method and uses a public distributed ledger, immutable transactions, and secure and restricted encryption.

Every transaction on the blockchain is cryptographically signed and publicly certified, ensuring that the data stored on the blockchain cannot be changed, hence the term "immutable transactions." Data can be written, but not updated or deleted, when a new block is formed in the blockchain structure.

The hash of the preceding block is contained in each block, which is formed by adding the nonce (number only once) value to generate the hash according to specified requirements. This improves the difficulty of the hash while also providing immutability.

The ICT academic community's perspective on blockchain security and privacy has altered as the technology's popularity has grown in recent years. To offer a distributed and immutable ledger containing transactions, blockchain relies on cryptographic proofs, digital signatures, and peer-to-peer networking. These characteristics entice scientists to use blockchain to create distributed applications.

Blockchain technology is being used in a wide range of areas, including cryptocurrencies, healthcare, and supply chain management. Blockchain is widely employed in the healthcare business to preserve and improve the security and accuracy of health-related records.

Medicine and the healthcare business have benefited a lot from blockchain technology. Several researchers have proposed data management strategies, medicinal precisions, biological sciences, and brain research using the blockchain.

Previously, researchers contributed to the elimination of transaction malleability in order to control and reduce the tampering of blocks and transactions in Bitcoin. In e-voting systems for fair elections, transaction tempering and malleability is a serious concern.

Practitioners look at the limitations of blockchain technology and strive to implement globally verifiable blockchain-based e-voting systems. Secure designs, visual cryptography, and consensus techniques are all used to improve e-voting systems.

3. Development Model

The system is developed using a managed and understood development process. We planned the development process accurately having clear ideas of what will be produced and when it will be completed. Of course, the specific process that we used depends on the type of software that we are developing.

A voting system is a critical system as any malfunction in the system, failure, mishandling of information or any sort of disturbance could be threatening and endangers the legitimacy of the elections as their results depends on the integrity of the system, so any time delay or any wrong output of information would have tragic consequences. It is developed on a very large scale as this system will cover the entire population of the country. So due to such crucial characteristics, the system required a lot of up-front analysis before implementation.

Formal traditional methods can be cost-effective in the development of critical software systems because the costs of system failure are very high and so additional cost in the development process is justified. The system must gain regulatory approval before it's used, it is a very difficult process to convince a regulator that a system is secure. The use of a formal intensive specifications and associated correctness is vital to assure the regulator of the promised security standards to gain his trust to endorse the system and give it the green light.

It certainly needed a plan-driven approach for development, with the requirements carefully analyzed. Requirements are clearly defined with no ambiguities or omissions, also future changes on the system would be very restricted. There is a need for extensive safety and security analysis of the software specification and design. Security problems in the specification and design would be very expensive to correct at the implementation stage.

Waterfall model is therefore the appropriate approach to use, with formal transitions between different development stages, as we took the fundamental process activities of specification, development, validation, and evolution and represented them as separate process phases.

3.1 Phases

1. Requirement’s analysis and definition:

The system’s services, constraints, and goals are established by consultation with accrediting authorities. They are then defined in detail and serve as a system specification.

2. System and software design:

Allocating the requirements to either hardware or software systems. Establishing an overall system architecture. Identifying and describing the fundamental software system abstractions and their relationships.

3. Implementation and unit testing:

The software design is realized as a set of program units. Unit testing involves verifying that each unit conforms to its specification.

4. Integration and system testing:

The individual program units are integrated and tested as a complete system to ensure that the system requirements have been clearly met. After testing, the software system is delivered to the authority.

5. Operation and maintenance:

This is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors that were not discovered in earlier stages of project, improving the implementation of system units, and enhancing the system's services as new requirements are introduced.

Also, a hybrid development model could have been adopted by combining the waterfall model with a reuse-based approach if this system would be integrated with other governmental legacy systems but preferably this system was developed from scratch and will be deployed independently due to high security risks from any possible vulnerabilities associated with other systems and to achieve greater reliability and maintainability.

3.2 Challenges

1. Heterogeneity:

The system is required to operate on distributed systems across networks that include different types of computers and mobile devices. It may interact with older operating systems written in different programming languages. The challenge here was to develop and build a dependable software that is flexible enough to cope with this heterogeneity.

2. Domain and social change:

Political field and society are evolving incredibly quickly as new laws are introduced and new technologies become available. Requirements may change and the need to update the existing software may arise. The system needed to be produced while carefully anticipating any future possible changes so that the cost required for software update is reduced.

3. Security and trust:

The system is intertwined with one of the most critical aspects of our lives, which is the elections, so it is essential that people can trust this software. We had to make sure that malicious users cannot successfully attack our software and that the privacy of users' information is securely maintained.

4. System scale:

The software had to be developed across a very wide range users, it needed to cover the entire population, so we had to ensure that the system is able to handle such a huge userbase safely and effectively.

5. Dependability and performance:

Software should behave as expected, without failures, and should be available for use when it is required. We ensured that the system is safe in its operation as far as possible and to perform effectively and efficiently without wasting any allocated resources.

4. System Specifications

4.1 System Overview

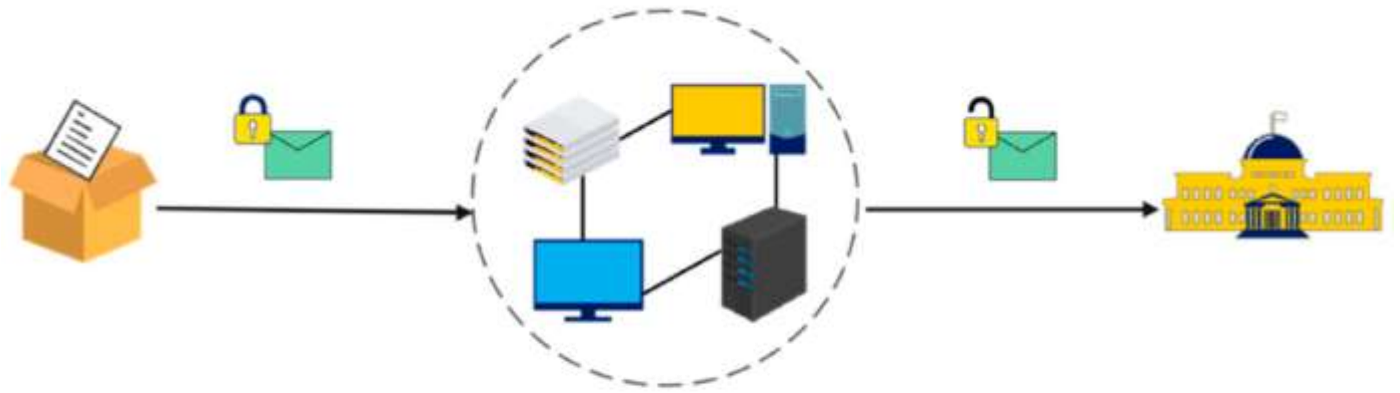


Fig. (1) - System Process

The blockchain is the base that our project is built upon. We have developed a permissioned blockchain, where access on the blockchain is only granted to authorized set of participant nodes. The nodes are a set of distributed Electronic Voting Machines (EVM) hosting on it the Voting Application and the Blockchain Client that grants access to the network.

Electronic voting machines are computing devices running dedicated proprietary software.

Administrators will use the Voting Application on the EVMs to setup and customize elections.

Voters will be able to cast their votes on the EVMs using the Voting Application after scanning their credentials using OCR and verifying their identity using advanced AI Facial Recognition.

Facial Recognition software has been developed using sophisticated Deep Learning algorithms to integrate various Neural Network models to achieve the maximum accuracy possible.

Nodes will register the votes as transactions on the blockchain, which will be broadcasted over the network to the rest of the nodes, to acquire validation by predetermined consensus protocol.

Network packets and communication channels are secured with several layers of encryption.

A website and a mobile application were developed as a supportive platform to guide voters through the elections with instructions, news and display dynamic live statistical results.

4.2 User Requirements

1. Guide the voters through the election process.

2. Only allow eligible individuals to vote in an election.

3. Ensure the authenticity of the voter’s identity and provide anonymity.

4. Ensure that every vote was registered and counted correctly.

5. Prevent any voter from voting more than once.

6. Prohibit traceability of a vote to a voter’s identifying credentials.

7. Deny control from any third party to tamper with any vote.

4.3 System Requirements

1. Guide the voters through the election process.

   I. Provide biographical information about the election’s candidates.

   II. Provide geographical locations of voting booths specified for each voter.

   III. Provide live dynamic statistical results during the elections.

   IV. Inform the voters with the election schedule.

2. Only allow eligible individuals to vote in an election.

   I. Adopt the database of authorized voters from the civil logs.

   II. Prevent unqualified users from voting.

3. Ensure the authenticity of the voter’s identity and provide anonymity.

   I. Request voters to provide their official identifying credentials.

   II. Scan the voter’s credentials and extract their information.

   III. Identify voter’s identity using AI Facial Recognition.

   IV. Match the scanned face with the ID’s picture.

   V. Encrypt the voter’s credentials securely.

   VI. Disqualify the voter if no match found.

4. Ensure that every vote was registered and counted correctly.

   I. Hold the transaction until complete vote transfer.

   II. Confirm to the voter successful registration of his vote.

   III. Provide the voter with his unique transaction ID.

5. Prevent any voter from voting more than once.

   I. Create a synchronized distributed database to store voters list.

   II. Search for a match in the voters list.

   III. Deny the individual from voting if a match was found.

6. Prohibit traceability of a vote to a voter’s identifying credentials.

   I. Separate the vote from the verifying credentials.

   II. Store the voting credentials solely in a dedicated encrypted database.

7. Deny control from any third party to tamper with any vote.

   I. Deploy it independently as a standalone system alienated from other systems.

   II. The system can’t be interrupted or altered during the runtime.

4.4 Non-Functional Requirements
(System Constraints)

4.4.1 Product Requirements

1. Security:

   I. ID: Any voter who make use of the system need to hold an ID.

   II. Modifications: Any modifications on the database records can only be made by the authorized set of nodes only.

   III. Protection: System should have robust security firewalls against malicious attacks.

2. Availability:

   I. Accessibility: The system should be continuously online all the time to serve voters.

   II. Fault tolerance: System must rely on backup generators in case of power outage.

   III. Delay time: System’s delay time can’t exceed 1 minute.

3. Reliability:

   I. Integrity: The system’s data should be regularly checked to maintain its validity.

4. Efficiency:

   I. Resources utilization: System should not make wasteful use of hardware resources.

5. Performance:

   I. Response time: System must provide acknowledgments as soon as a request is sent.

   II. Capacity: The system network needs to be robust and have the ability to handle high traffic without any congestions.

   III. Responsiveness: The UI needs to be highly optimized with smooth transitions.

   IV. Computing power: The system must have the sufficient computing power to achieve the minimal processing time possible.

6. Usability:

I. Ease: The system should be user friendly to all users with various experience levels.

II. Help: Interactive instructions are implemented to guide users with voice assistance.

7. Maintainability:

I. Upgradeability: Software should be written in such an optimized way that it can evolve to allow installation of future updates.

II. Maintenance: Periodic maintenance shall be performed on the system daily.

4.4.2 Organizational Requirements

1. Operational:

I. The system shall be implemented as a client-server peer-to-peer system with election information held on distributed nodes maintained by the authority.

II. The system shall run on Linux operating system for additional security.

III. The system should be power efficient to decrease operational costs.

2. Development:

I. The system shall be developed and integrating all its components using a general purpose object-oriented programming language (C#).

II. Supporting website and mobile application should be developed using a cross platform language to provide the same consistent experience on every device.

4.4.3 Domain Requirements

1. The system must adhere to the authority's civil rules and regulations.
2. The system must conform to the standards and provisions for secure information systems as set out by the authorities.
3. The system must be transparent and conform with the ethical standards.
4. System shall be required to report statistical information to other systems.

5. System Design

Our software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components. We did not arrive at a finished design immediately, but we have developed the design in stages. Details have been added consistently through development, with constant backtracking to modify earlier designs. New information about the design was constantly being generated, and that affected previous design decisions therefore design rework was inevitable. This chapter is an abstract of the design process showing the inputs to the design process, system activities, and the process outputs. The design models are both interleaved and interdependent.

5.1 System Architecture

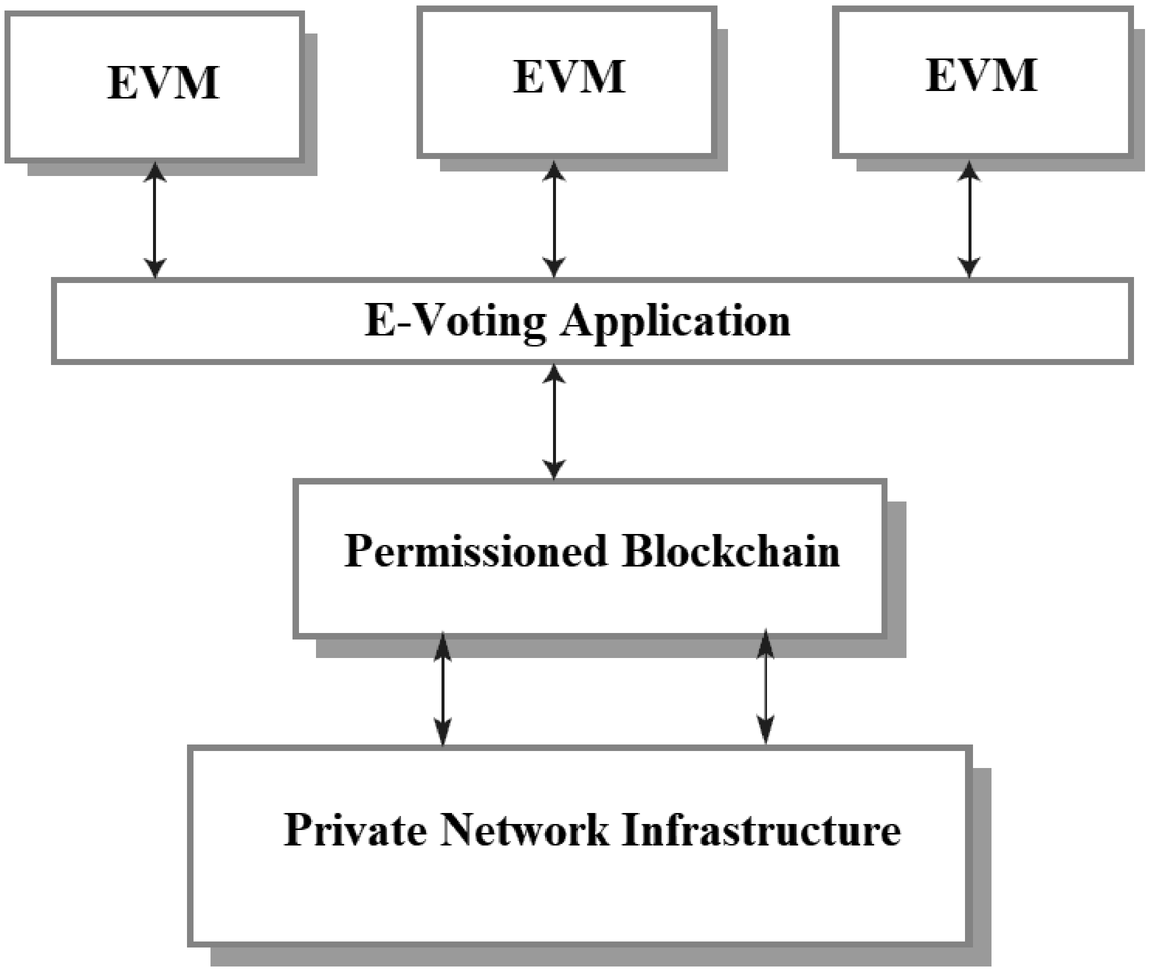


Fig. (2) - External System's Architecture

Fig. (2) identifies the overall structure of the system, the principal components (subsystems or modules), and how they are distributed and organized as a whole.

5.2 Internal System Structure

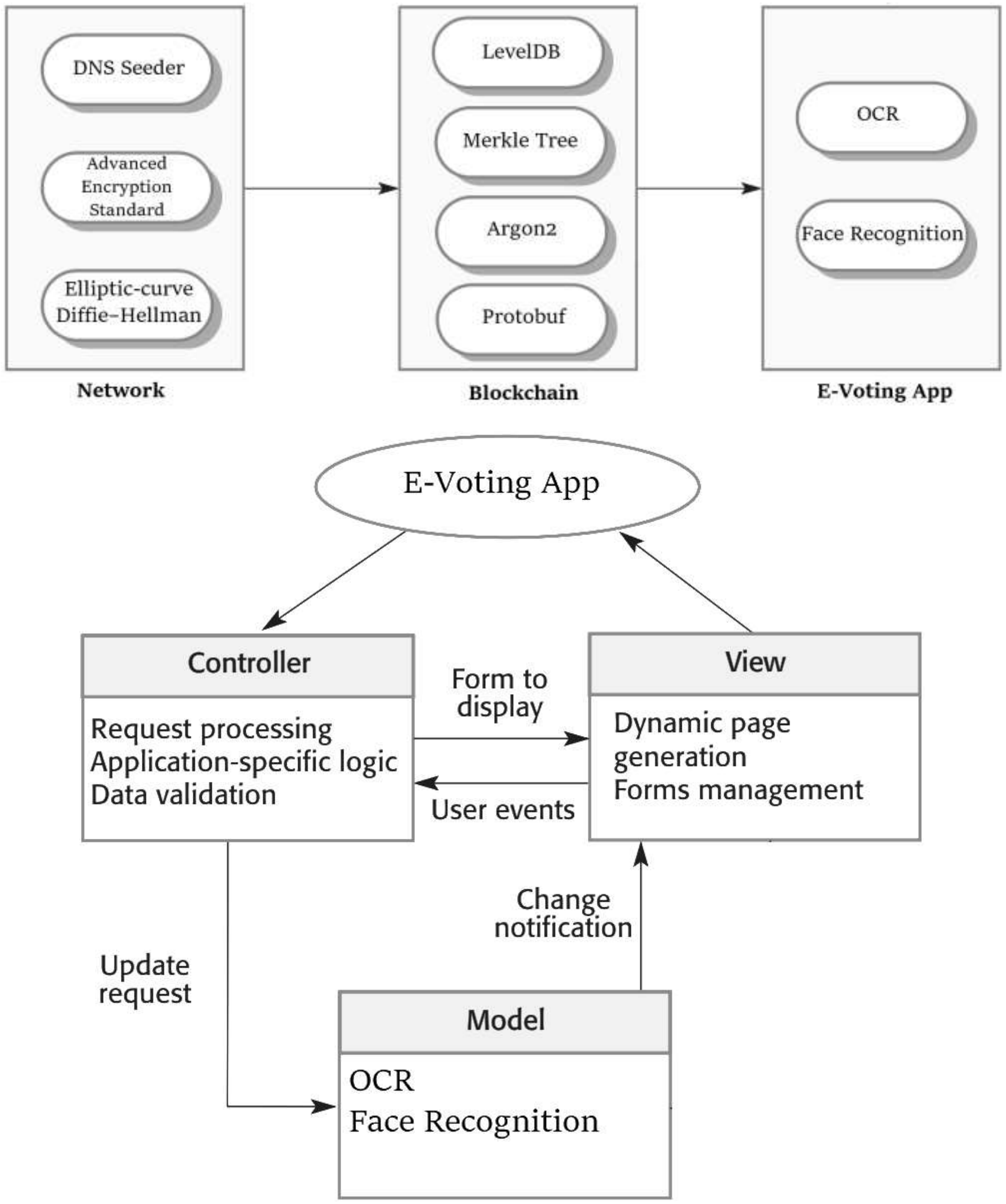


Fig. (3) - Internal System's Structure

Fig. (3) shows the internal structure of the system, illustrating the system's components and the interactions between them during the runtime.

5.3 Use Case Model

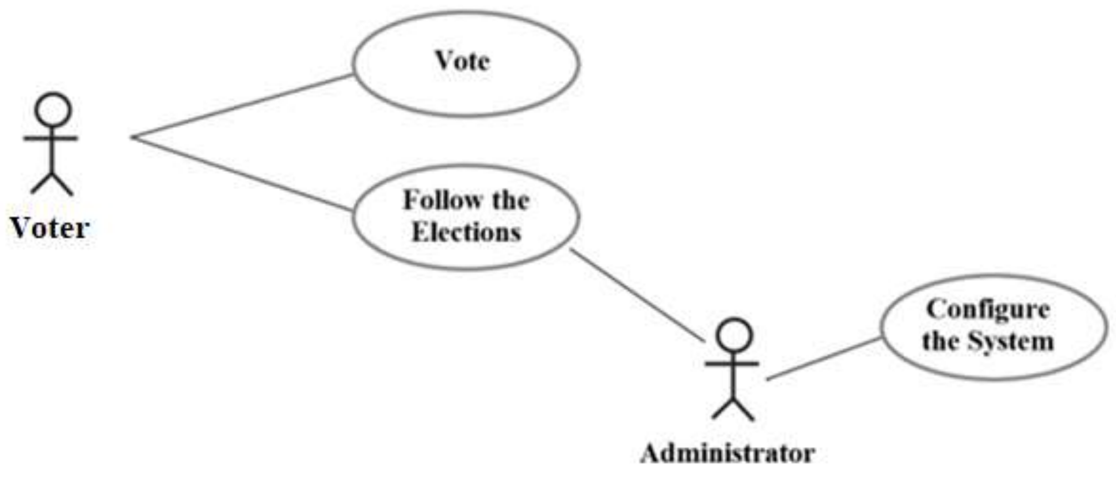


Fig. (4) - Different Use Cases Involving System Actors

Fig. (4) models the interactions between the system and the external agents. It's a simple description of what is expected from the system. Each use case represents a discrete task that involves external interaction with the system.

Use Case	Vote
Actors	Voter
Description	The user approaches the electronic voting machine (EVM) to initiate the voting process by inserting his identifying credentials to be scanned using optical character recognition (OCR). If the voter is eligible, a face recognition system captures a photo of the user to authenticate his identity. The user is then prompted to select one of the available candidates. Upon selection the vote is then encrypted and registered on the blockchain, and the voter's ID is blocked from voting again.
Data	Voter's Credentials, Voter's Picture, Voter's Selection
Stimulus	Voting process has been initiated by the user.
Response	Confirmation that the vote has been registered successfully.
Comments	Voter must have a valid ID card at the time of voting that matches his true identity that has been verified biometrically through face recognition.

Table (2) - Tabular description of the Vote use case

5.4 Sequence Diagram

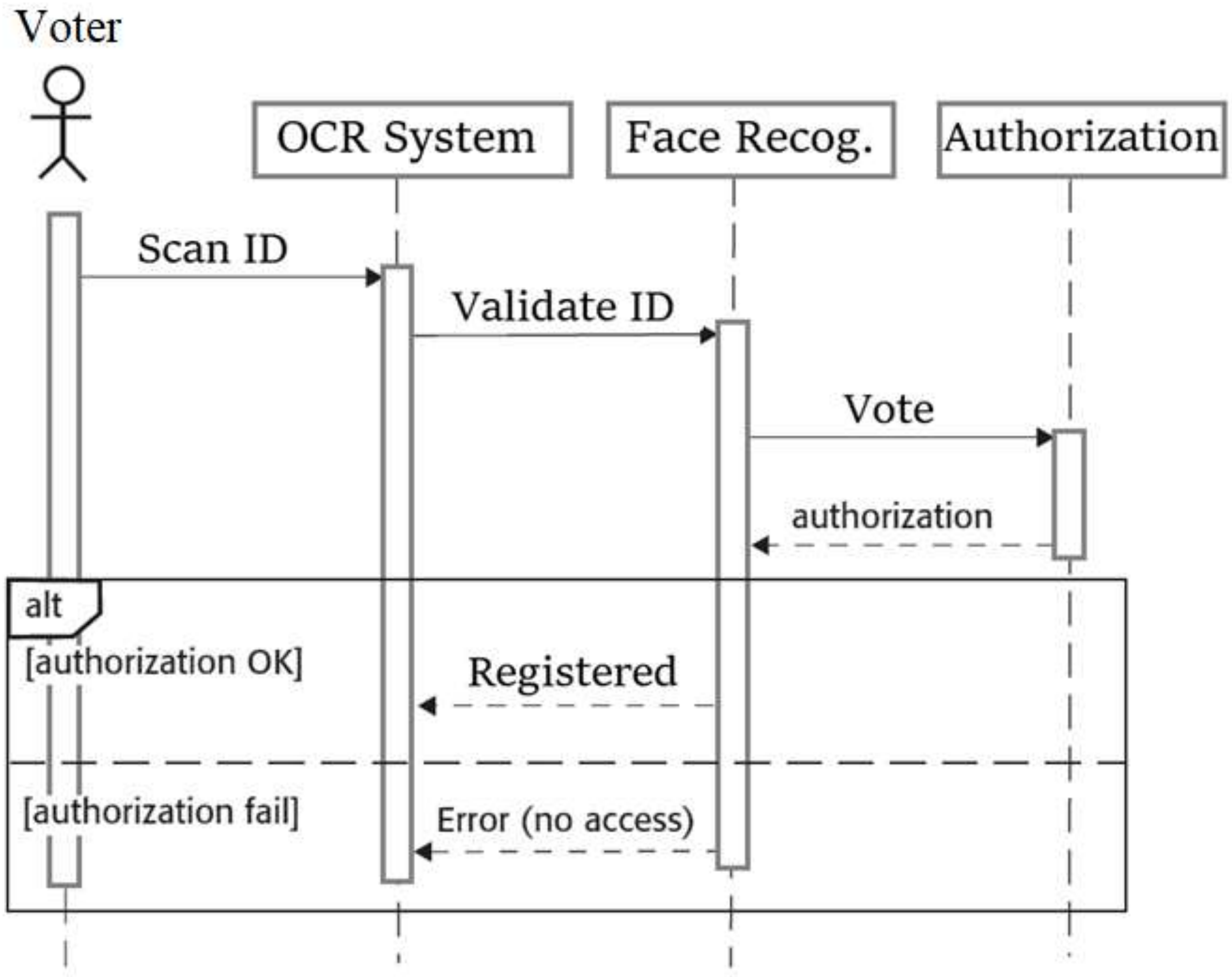


Fig. (5) - Sequence Diagram for the Voting Process

Fig. (5) models the sequence of interactions between the internal system components and modules that takes place during the voting process, that's triggered by the Citizen.

1. The voter triggers the voting process by invoking the Scan ID method through the OCR model, supplying the system with the voter's credentials.

2. The system calls the Face Recognition model to return a photo of the voter's face, supplying the system with an identifier to allow security checking.

3. The system checks if the voter's identity matches the photo.

4. If it matches, the user is then authorized to select one of the available candidates and the vote is registered successfully, if not, then no access is granted.

5.5 Process Model

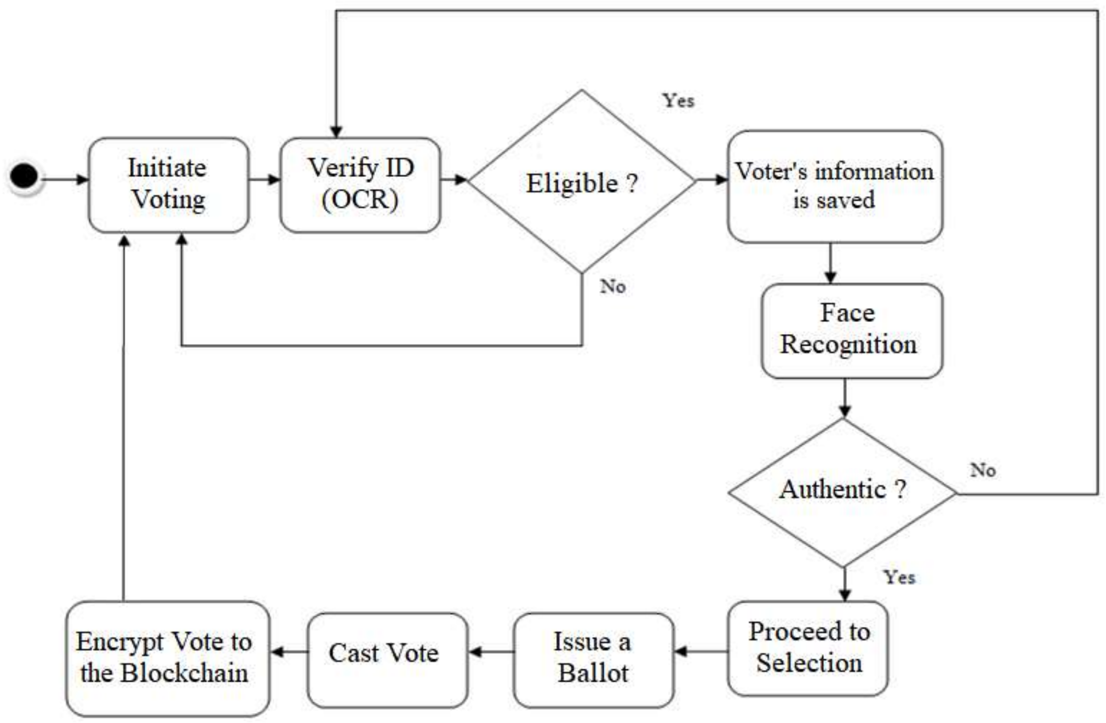


Fig. (6) - System's Process Activity Model

Fig. (6) is a Unified Modeling Language activity model that dynamically illustrates how the software flows and transforms through a sequence of events that drives the voting process.

5.6 Class Diagram

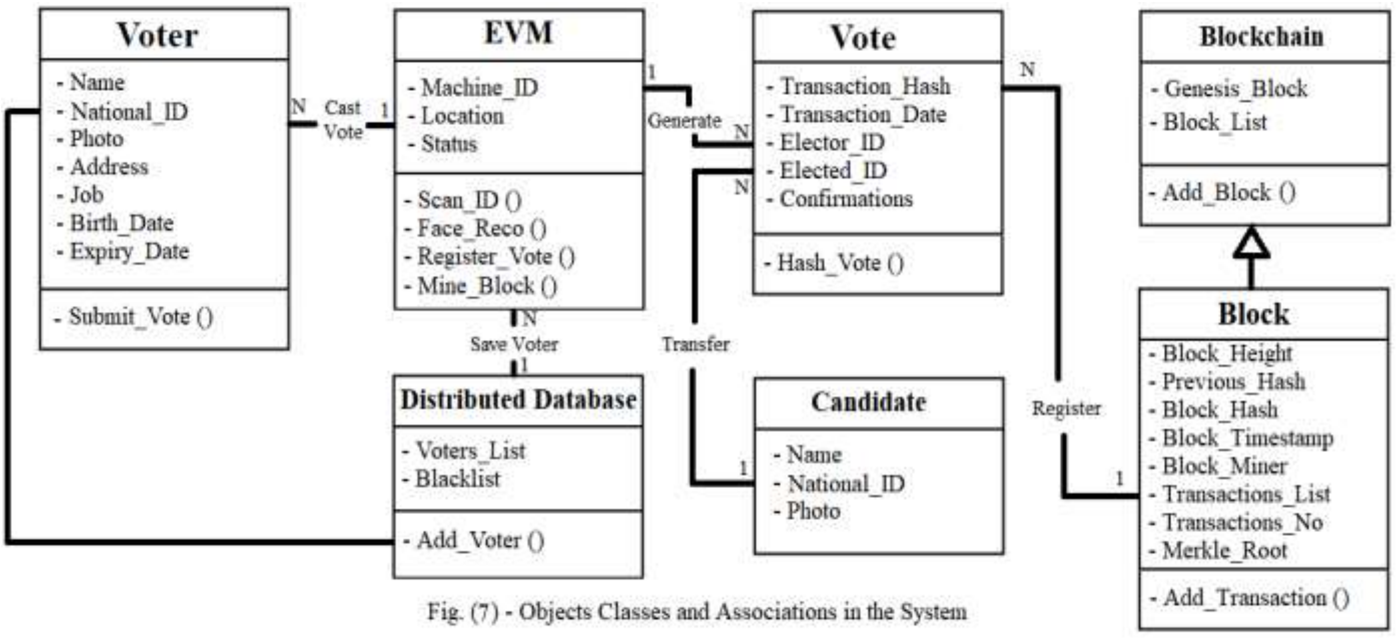


Fig. (7) - Objects Classes and Associations in the System

Fig. (7) shows the class structure in the system and their associations displaying the system's organization in terms of components that make up the system and relationships in between.

6. System Implementation

The implementation stage of the system development is the process of developing an executable operational system for delivery to the authority that’s ready for deployment.

6.1 Programming Language (C#)

C# is our choice, it’s a general-purpose, multi-paradigm programming language. C# encompasses static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines.

The C# programming language was designed by Anders Hejlsberg from Microsoft in 2000 and was later approved as an international standard by Ecma (ECMA-334) in 2002 and ISO (ISO/IEC 23270) in 2003. Microsoft introduced C# along with .NET Framework and Visual Studio, both of which were closed-source. At the time, Microsoft had no open-source products. Four years later, in 2004, a free and open-source project called Mono began, providing a cross-platform compiler and runtime environment for the C# programming language. A decade later, Microsoft released Visual Studio Code (code editor), Roslyn (compiler), and the unified .NET platform (software framework), all of which support C# and are free, open-source, and cross-platform. As of 2021, the most recent version of the language is C# 10.0, which was released in 2021 in .NET 6.0.[5]

- **Why C# ?**

Of all the programming languages, C# is perhaps one of the best. This multi-paradigm language is versatile, fairly easy to learn and object-oriented. With so many different programming languages to choose from, we thought C# would be the proper language and the right fit for us in our project also due to the development team's previous experience with it.

C# language is considered topnotch as it’s easy to learn, performs well, and can be used for any type of task, application, website, or platform. It is used to develop professional, dynamic websites on the .NET platform, or open-source software. It can create a fully functional website applications and services, as this language is object-oriented, it is utilized to develop websites that are incredibly efficient, easily scalable and a breeze to maintain.

6.2 Software Framework (.NET)

.NET is a free and open-source, managed computer software framework for Windows, Linux, and macOS operating systems. It is a cross-platform successor to .NET Framework. The project is primarily developed by Microsoft employees by way of the .NET Foundation and released under the MIT License.

- **Why .NET ?**

Since .NET is a cross-platform development framework for building many different types of applications, we were able to use multiple languages, editors, and libraries to build our project for web, mobile and desktop.

6.2.1 .NET Advantages

1. One consistent API

.NET Standard is a base set of APIs that are common to all .NET implementations. Each implementation can also expose additional APIs that are specific to the OS it runs on.

2. Libraries

To extend functionality, Microsoft maintains a healthy package ecosystem built on .NET. NuGet is a package manager built specifically for .NET that contains over 90,000 packages.

3. Active community and open source

Because .NET is open source, one can join the thousands of developers and companies already contributing to the .NET platform and get quick answers to questions with an active community of developers on Stack Overflow.

4. Productivity

.NET helps to develop high quality applications faster. Modern language constructs like generics, LINQ and asynchronous programming make development more productive.

5. Performance

.NET is fast, applications provide better response times and require less compute power.

6.3 Development Environment

6.3.1 Microsoft Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that expand the functionality at almost every level including adding support for source control systems (like Subversion and Git) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Azure DevOps client: Team Explorer).

6.3.2 Git

Git is a software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).

As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server. Git is free and open-source software distributed under the GPL-2.0-only license.

GitHub is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features. It provides access control and collaboration features such as bug tracking, feature requests, task management, continuous integration, & wikis for every project.

6.4 Network

We tried to ensure maximum efficiency for our network performance, taking in consideration the nature of our application, we decided that our project's network infrastructure should be built around a distributed Peer-to-Peer structure.

6.4.1 Peer-to-Peer

Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. They form a peer-to-peer network of nodes.

Without the need for central coordination by servers or stable hosts, peers are both suppliers and consumers of resources, in contrast to the traditional client–server model in which the consumption and supply of resources is divided.

Blockchains are typically managed by a peer-to-peer network for use as a publicly distributed ledger, where nodes collectively adhere to a protocol to communicate and validate new blocks.

- **Why P2P ?**

1. Fault Tolerance

A fault tolerant network is one that limits the number of affected devices during a failure. It is built to withstand any failure or malfunction that might occur to any node in the network.

2. Scalability

A scalable network expands quickly to support new users. It does this without degrading the performance of services that are being accessed by existing users.

3. Decentralized

Unlike a client-server network, which can fail if the central server malfunctions (SPOF), a P2P has no such drawback as the workload is distributed all over the entire network.

4. High Performance

A P2P network can improve its performance when more clients join it. This is because each client in a P2P network is also a server that contributes resources to the network.

6.4.2 Encryption

Network packets and communication channels are secured with several layers of encryption.

6.4.2.1 Advanced Encryption Standard (AES)

AES is used in our project to encrypt every transmitted network packet in the system.

The Advanced Encryption Standard (AES), also known by its original name Rijndael is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.

It is a symmetric block cipher chosen by the U.S. government to protect classified information.

AES is implemented in software and hardware throughout the world to encrypt sensitive data. It is essential for government computer security, cybersecurity, and electronic data protection.

NIST started development of AES in '97 when it announced the need for an alternative to the Data Encryption Standard (DES), which started to become vulnerable to brute-force attacks.

AES Encryption algorithm is a symmetric block cipher algorithm with a block/chunk size of 128 bits. It converts these individual blocks using keys of 128, 192, and 256 bits. Once it encrypts these blocks, it joins them together to form the ciphertext.

It is based on a substitution-permutation network, also known as an SP network. It consists of a series of linked operations, including replacing inputs with specific outputs (substitutions) and others involving bit shuffling (permutations).

What are the features of AES ?

- SP Network: It works on an SP network structure rather than a Feistel cipher structure, as seen in the case of the DES algorithm.
- Key Expansion: It takes a single key up during the first stage, which is later expanded to multiple keys used in individual rounds.
- Byte Data: The AES encryption algorithm does operations on byte data instead of bit data. So, it treats the 128-bit block size as 16 bytes during the encryption procedure.
- Key Length: The number of rounds to be carried out depends on the length of the key being used to encrypt data.

- **How AES encryption works ?**

AES includes three block ciphers, each cipher encrypts and decrypts data in blocks of 128 bits using cryptographic keys of 128, 192 and 256 bits, respectively.

Symmetric, also known as secret key, ciphers use the same key for encrypting and decrypting. The sender and the receiver must both know and use the same secret key.

There are 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. A round consists of several processing steps that include substitution, transposition and mixing of the input plaintext to transform it into the final output of ciphertext.

The AES encryption algorithm defines numerous transformations that are to be performed on data stored in an array. The first step of the cipher is to put the data into an array, after which the cipher transformations are repeated over multiple encryption rounds.

The first transformation in the AES encryption cipher is substitution of data using a substitution table. The second transformation shifts data rows. The third mixes columns. The last transformation is performed on each column using a different part of the encryption key. Longer keys need more rounds to complete.

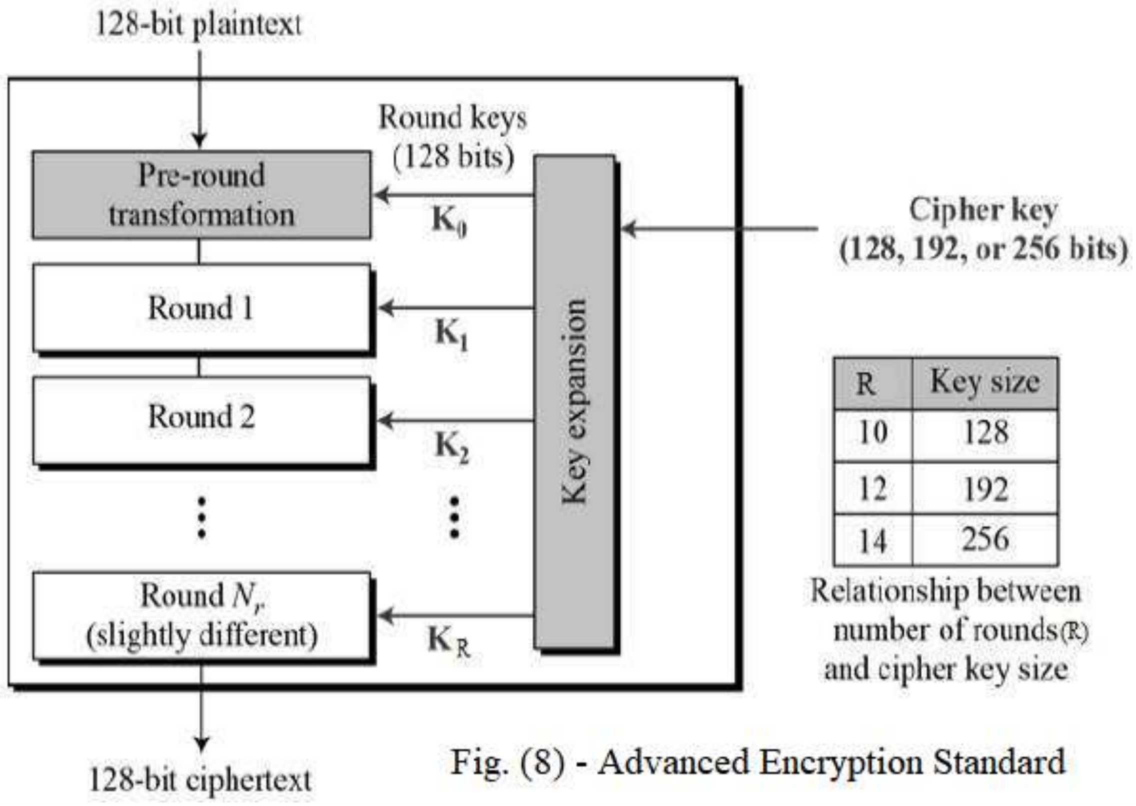


Fig. (8) - Advanced Encryption Standard

6.4.2.2 Elliptic-Curve Diffie-Hellman (ECDH)

ECDH is used to secure the transmission of AES encryption key between the network nodes.

Elliptic-curve Diffie-Hellman is a key agreement protocol that allows two parties, each having an elliptic-curve public-private key pair, to establish a shared secret over an insecure channel. This shared secret may be directly used as a key, or to derive another key. The key, or the derived key, can then be used to encrypt subsequent communications using a symmetric-key cipher. It is a variant of the Diffie-Hellman protocol using elliptic-curve cryptography.

Elliptic-curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. ECC allows smaller keys compared to non-EC cryptography (based on plain Galois fields) to provide equivalent security.

Elliptic curves are applicable for key agreement, digital signatures, pseudo-random generators and other tasks. Indirectly, they can be used for encryption by combining the key agreement with a symmetric encryption scheme.

ECDH derives a shared secret value from a secret key owned by an Entity A and a public key owned by an Entity B, when the keys share the same elliptic curve domain parameters. Entity A can be either the initiator of a key-agreement transaction, or the responder in a scheme. If two entities both correctly perform the same operations with corresponding keys as inputs, the same shared secret value is produced.

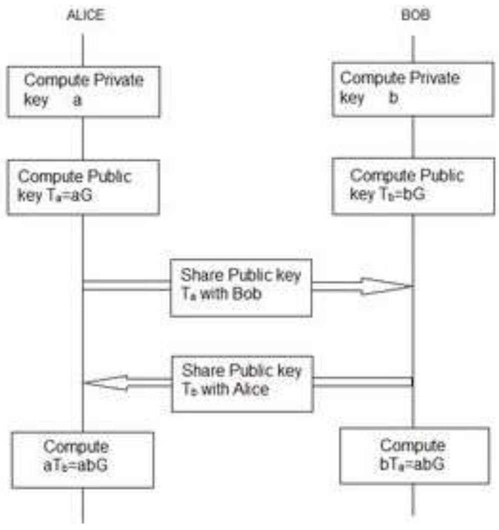

ALICE
BOB
Compute Private key a
Compute Private key b
Compute Public key T_a=aG
Compute Public key T_b=bG
Share Public key T_a with Bob
Share Public key T_b with Alice
Compute aT_b=abG
Compute bT_a=abG

Fig. (9) - ECDH Key Exchange

6.4.3 DNS Seeder

DNS Seeder is a blockchain network crawler that maintains a list of IP addresses of the most reliable nodes on the network and shares those node IPs via DNS request to anyone requiring an entry point into the decentralized network.

Blockchain is a decentralized network, meaning there is no central node that we can rely on and that it will be always active. Nodes go up and down. When we start a new local node, it won’t know the IPs of the current active nodes. In order to discover them, it queries one or more DNS names (DNS Seeders). The seeder responses with several IPs of nodes that are active. Now on a local node tries to connect to them. If for some reason this process fails and no IPs were returned, there is a fallback option.

IPs of the current active nodes are hardcoded. If the local node could not connect to the network via DNS it will try to connect to those hardcoded IPs. If it could not connect to anything then we could manually type IPs of nodes that we know are active.

At the initial start the DNS Seeder doesn’t know any IPs of active nodes. That’s why we hardcoded IPs in the code. The Seeder tries to connect to the hardcoded nodes. Upon respond the Seeder adds them to the list of active nodes. On their behalf, the responding nodes will be asked for the IPs of other nodes that they are connected to, and the Seeder will try to connect to them, too. This is a continuous process. Nodes that are in the list are pinged after a certain time period, if they don’t respond they are removed from the list.

6.4.3.1 Features

- Regularly revisits known nodes to check their availability.
- Keep a running list of all nodes in the network.
- Bans/Unlists nodes after enough failures, or bad behavior.
- Keeps statistics over different time windows, to base decisions on.
- Very low memory (a few tens of megabytes) and CPU requirements.
- Force connections to IPv4 or IPv6 only if desired.
- Customizable options via a configuration file.

6.5 Blockchain

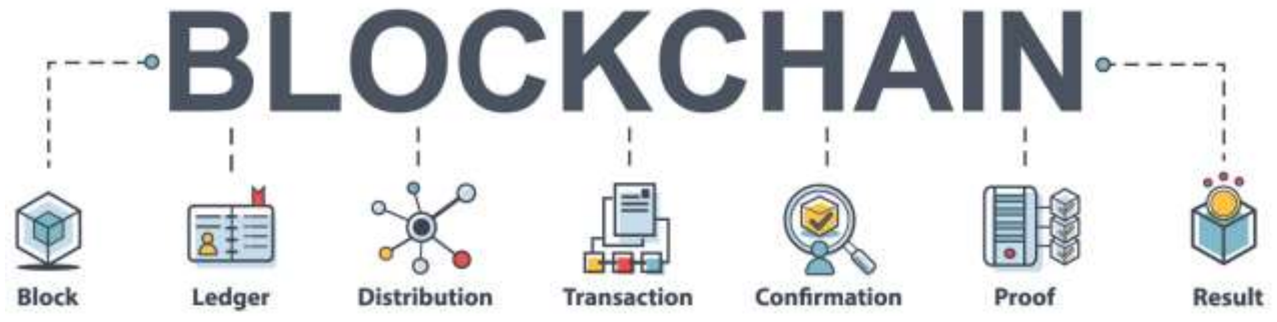


Fig. (10) - Blockchain

Blockchain is essentially a fully decentralized distributed database of records for transactions or digital events, that are linked using cryptography and shared among participating parties connected within a network where each participant holds the same synchronized copy of data.

It is basically a growing chain of blocks where blocks are connected to form a chain that holds data or information regarding any event. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data represented as a Merkle tree. Hash act as a digital fingerprint that uniquely identifies a block and is determined by its contents.

As each block contains information about the block previous to it, they form a chain, with each additional block reinforcing it. Therefore, blockchains are resistant to modification of their data because once recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks and revoking the validity of the chain.

Each transaction or activity within the blockchain is verified by consensus of the majority of the participants, without the approval of the majority of the network, an activity cannot be taken into consideration. Once some data has been inserted into a blockchain, it becomes very difficult to change it due to having immutability configuration.

Blockchains are typically managed as a peer-to-peer network, where nodes collectively adhere to a specific protocol to communicate and validate new blocks, eliminating single point of failure, preventing third party involvement, and discarding all the intended fraudulent activities, therefore, problems such as security, cost, trust and other related issues are resolved.

These technological features operate through advanced cryptography, providing a security level greater than any previously known database. Blockchain is therefore considered by many, including us, to be the ideal tool, to securely modernize the democratic voting process.

6.5.1 Blockchain Structure

The blockchain data structure is an ordered, back-linked list of encrypted blocks of transactions. In our case the transactions are the casted votes in our system.

A block is a data container that aggregates transactions for inclusion in the blockchain. It records the recent transactions not yet validated by the network. Once the data are validated, the block is closed. Then, a new block is created for new transactions to be entered & validated.

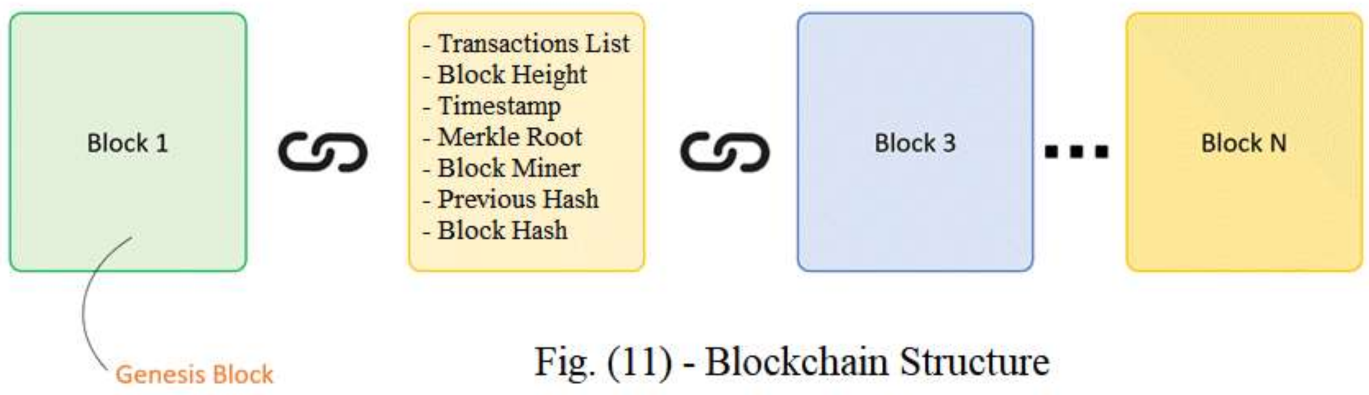


Fig. (11) - Blockchain Structure

Genesis Block is the first block created, which forms the foundation of the entire blockchain.

6.5.1.1 Metadata

1. Block Height: Refers to the position of the block in the blockchain, measured by how many confirmed blocks precede it. The current block height of a blockchain is an indication of its current size or time in existence.

2. Timestamp: It's the time of block generation set by the node which mined the block.

6.5.2 Merkle Tree

Blockchain is comprised of various blocks that are linked with one another. A hash tree, or the Merkle tree, encodes the blockchain data in an efficient and secure manner. It enables the quick verification of blockchain data, as well as quick movement of large amounts of data from one computer node to the other on the peer-to-peer blockchain network.

Every transaction occurring on the blockchain network has a hash associated with it. However, these hashes are not stored in a sequential order on the block, rather in the form of a tree-like structure such that each hash is linked to its parent following a parent-child tree-like relation.

Since there are numerous transactions stored on a particular block, all the transaction hashes in the block are also hashed, which results in a Merkle root.

Merkle root is the hash of all the hashes of all the transactions that are part of a block in a blockchain network. It’s a simple mathematical way to verify the data on a Merkle tree. Merkle roots are used in blockchain to make sure data blocks passed between peers on a peer-to-peer network are whole, undamaged, and unaltered.

For example, consider a seven-transaction block. At the lowest level (called the leaf-level), there will be four transaction hashes. At the level one above the leaf-level, there will be two transaction hashes, each of which will connect to two hashes that are below them at the leaf level. At the top (level two), there will be the last transaction hash called the root, and it will connect to the two hashes below it (at level one).

Effectively, we get an upside-down binary tree, with each node of the tree connecting to only two nodes below it (hence the name "binary tree"). It has one root hash at the top, which connects to two hashes at level one, each of which again connects to the two hashes at level three (leaf-level), and the structure continues depending upon the number of transactions.

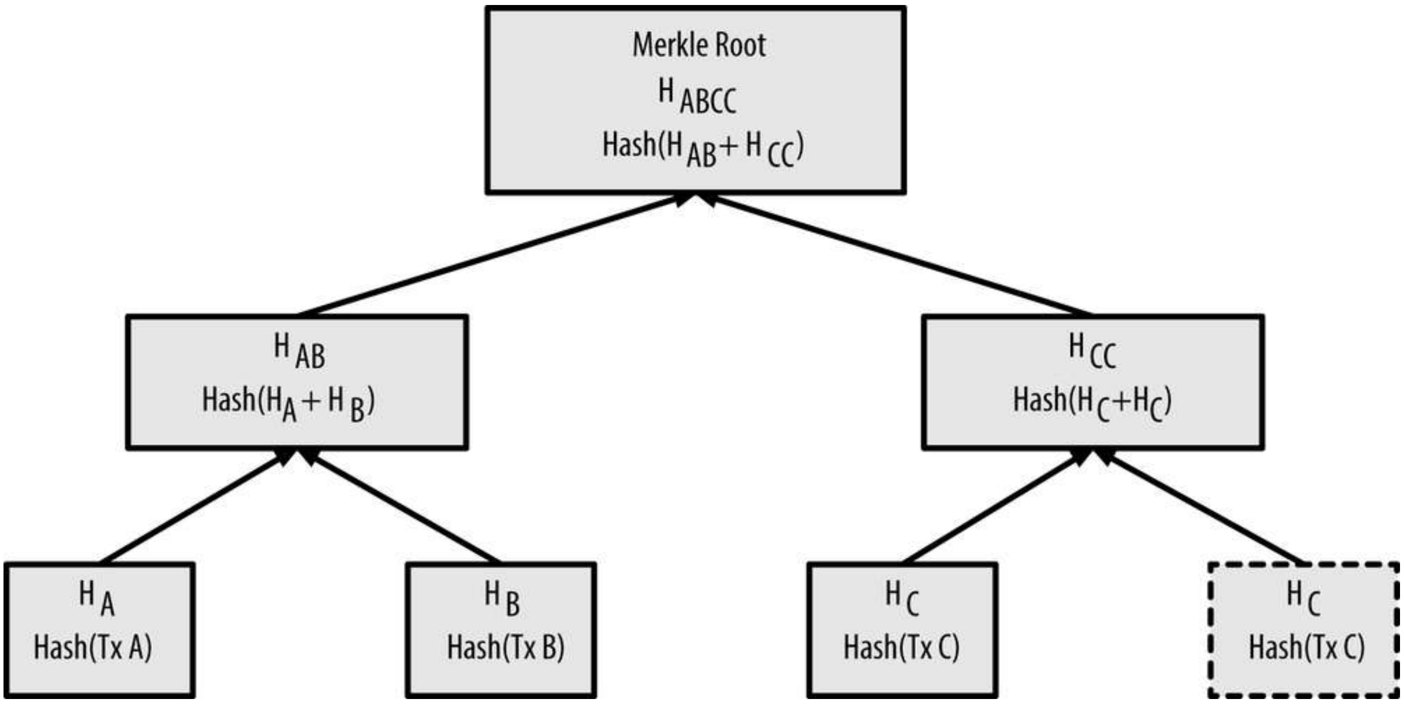


Fig. (12) - Merkle Tree

6.5.3 Hash

A hash is a mathematical function that converts an input of arbitrary length into an encrypted output of a fixed length. Thus, regardless of the original amount of data or file size involved, its unique hash will always be the same size. Moreover, hashes cannot be used to "reverse-engineer" the input from the hashed output, since hash functions are "one-way" (like a meat grinder; we can't put the ground beef back into a steak). Still, if we use such a function on the same data, its hash will be identical, so we can validate that the data is the same (i.e., unaltered) if we already know its hash.

Hash meets the encrypted demands needed to solve for a blockchain computation. Hashes are of a fixed length since it makes it nearly impossible to guess the length of the hash if someone was trying to crack the blockchain. The same data will always produce the same hashed value. A hash is developed based on the information present in the block.

- **How hashing works ?**

Typical hash functions take inputs of variable lengths to return outputs of a fixed length. A cryptographic hash function combines the message-passing capabilities of hash functions with security properties.

Hashes are used data structures in computing systems for tasks, such as checking the integrity of messages and authenticating information. While they are considered cryptographically "weak" because they can be solved in polynomial time, they are not easily decipherable.

Cryptographic hash functions add security features to typical hash functions, making it more difficult to detect the contents of a message or information about recipients and senders.

In particular, cryptographic hash functions exhibit these three properties:

1. Collision-Free: This means that no two input hashes should map to the same output hash.

2. Hidden: It should be difficult to guess the input value for a hash function from its output.

3. Puzzle-Friendly: It should be difficult to select an input that provides a predefined output.

Because of the features of a hash, they are used extensively in online security - from protecting passwords to detecting data breaches to checking the integrity of a downloaded file.

6.5.3.1 Argon2

Argon2id is our used cryptographic hashing function implemented in the blockchain.

Argon2 is a is modern ASIC-resistant and GPU-resistant secure key derivation function that was selected as the winner of the Password Hashing Competition.

Argon2 has 6 input parameters: password, salt, memory cost (the memory usage of the algorithm), time cost (the execution time of the algorithm and the number of iterations), parallelism factor (the number of parallel threads), hash length.

It was designed by Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich from the University of Luxembourg. The reference implementation of Argon2 is released under a Creative Commons CC0 license or the Apache License 2.0, and provides three related versions:

1. Argon2d maximizes resistance to GPU cracking attacks. It accesses the memory array in a password dependent order, which reduces the possibility of time–memory trade-off (TMTO) attacks but introduces possible side-channel attacks.
2. Argon2i is optimized to resist side-channel attacks. It accesses the memory array in a password independent order.
3. Argon2id is a hybrid version. It follows the Argon2i approach for the first half pass over memory and the Argon2d approach for subsequent passes. The RFC recommends using Argon2id if the difference between the types is not known, or if side-channel attacks are considered to be a viable threat.

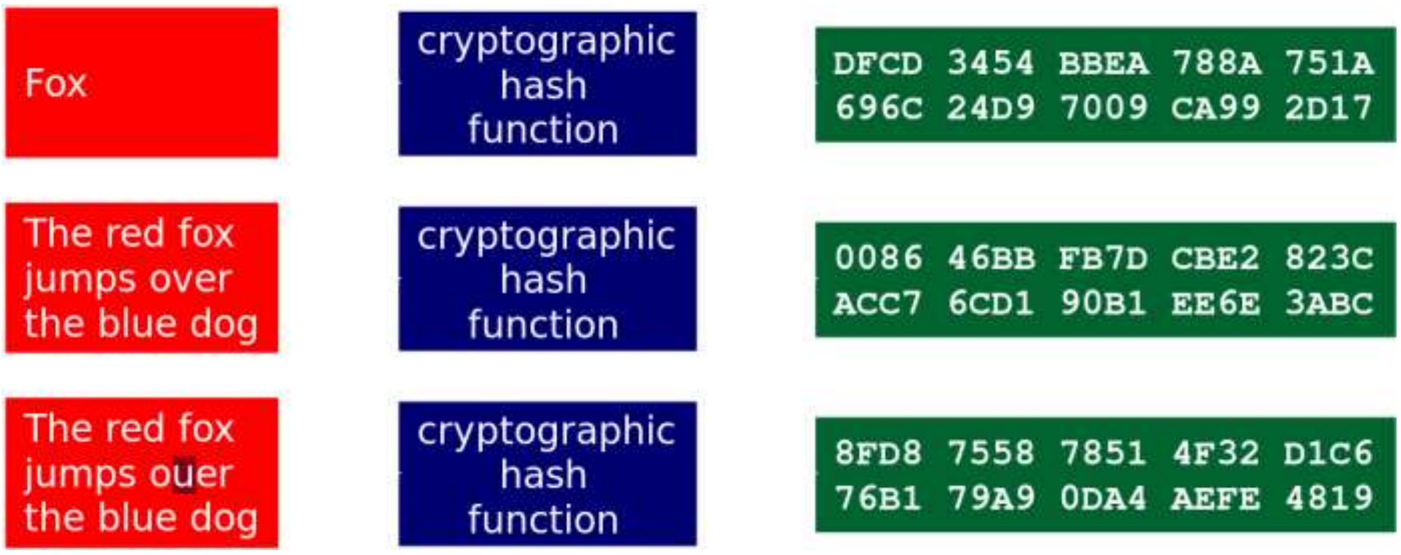

Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819

Fig. (13) - Cryptographic Hashing

6.5.3.2 Hashing in Blockchain

A block also has a hash. Hash can be understood as a fingerprint which is unique to each block. It identifies a block and all of its contents, and it's always unique, just like a fingerprint. So once a block is created, any change inside the block will cause the hash to change.

Consider the following example, where we have a chain of 3 blocks. The 1st block has no predecessor. Hence, it does not contain a hash of a previous block. Block 2 contains the hash of block 1. While block 3 contains the hash of block 2. Hence, all blocks contain the hashes of previous blocks. This is the technique that makes a blockchain so secure.

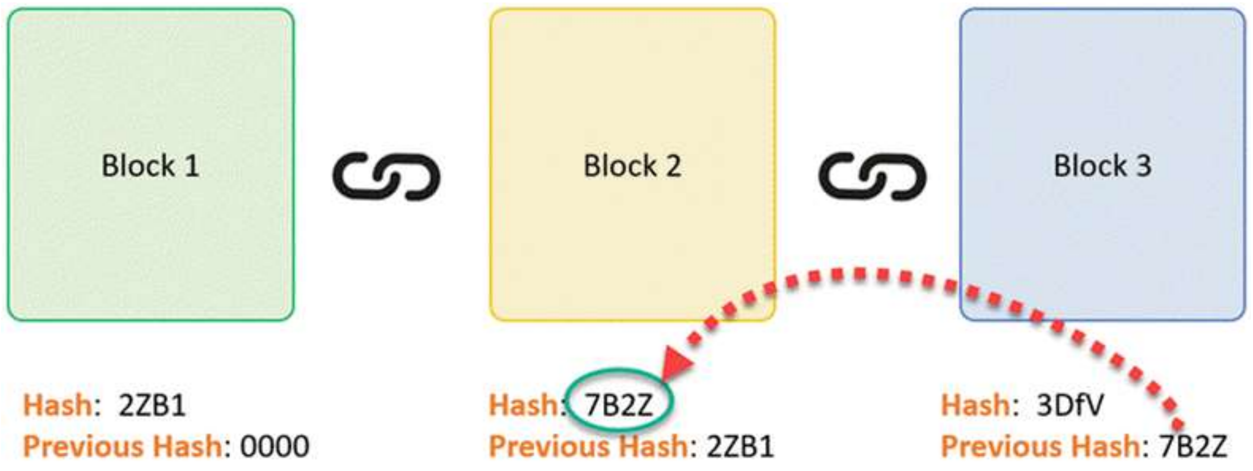


Fig. (14) - Previous Hash

Assume an attacker can change the data present in block 2. Correspondingly, the hash of the block also changes. But block 3 still contains the old hash of block 2. This makes block 3, and all succeeding blocks invalid as they do not have the correct hash of the previous block.

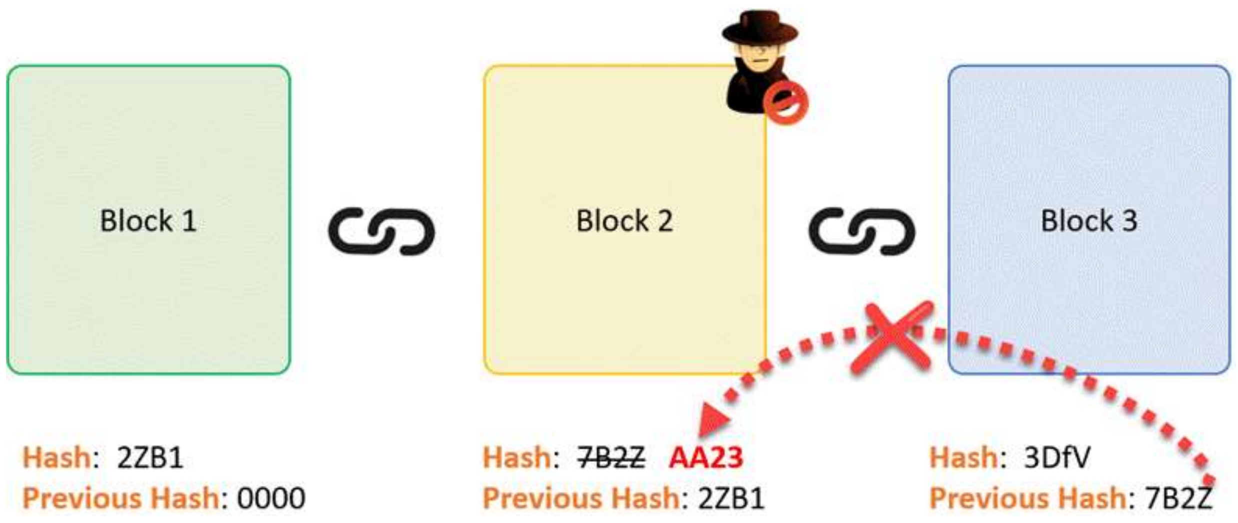
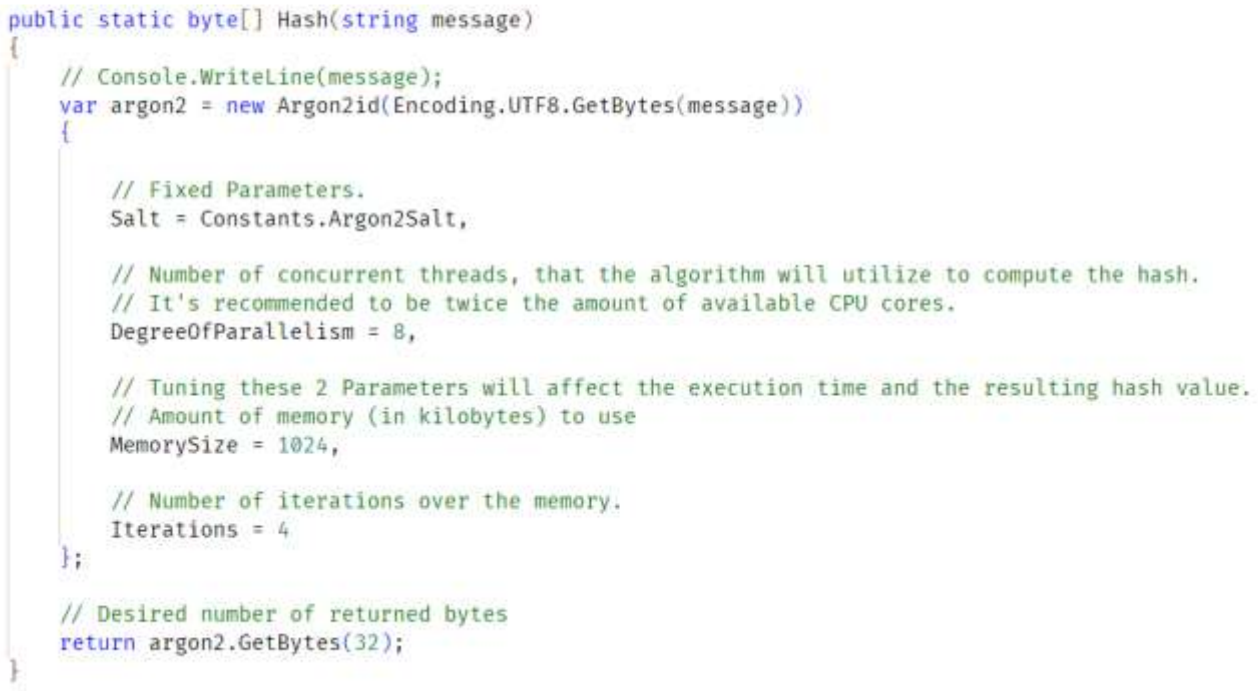


Fig. (15) - Invalid Blockchain

6.5.3.3 Hash Implementation

Argon2 hashing function is used to generate the hash values for any kind of data that must be verified and kept unchanged in the blockchain.



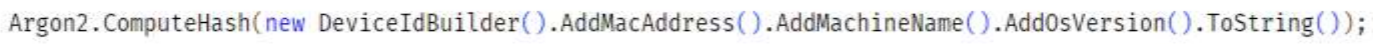
```
public static byte[] Hash(string message)
{
    // Console.WriteLine(message);
    var argon2 = new Argon2id(Encoding.UTF8.GetBytes(message))
    {

        // Fixed Parameters.
        Salt = Constants.Argon2Salt,

        // Number of concurrent threads, that the algorithm will utilize to compute the hash.
        // It's recommended to be twice the amount of available CPU cores.
        DegreeOfParallelism = 8,

        // Tuning these 2 Parameters will affect the execution time and the resulting hash value.
        // Amount of memory (in kilobytes) to use
        MemorySize = 1024,

        // Number of iterations over the memory.
        Iterations = 4
    };

    // Desired number of returned bytes
    return argon2.GetBytes(32);
}
```

1. Generating Hash of a Transaction:

Transactions data consist of the following information for describing a vote:

1. An identifier of the voting machine which initiates the transaction on the voter's behalf to prevent traceability of his identity.

This ID is generated based on some information about the voting machine (i.e., MAC address), then we hash it. In this way we can uniquely identify any voting machine.



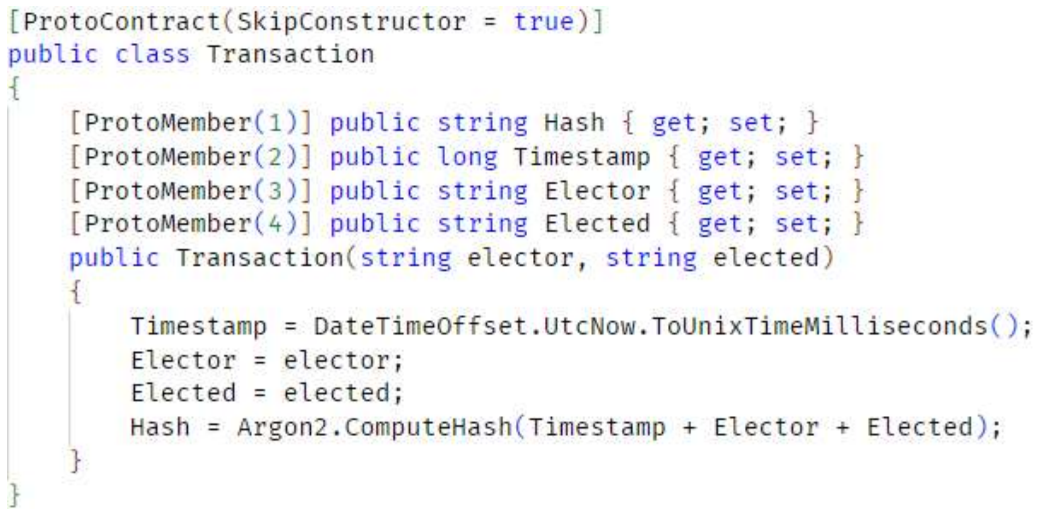
```
Argon2.ComputeHash(new DeviceIdBuilder().AddMacAddress().AddMachineName().AddOsVersion().ToString());
```

2. An identifier for the candidate that is to receive the vote.

The hash value of the candidate is generated by hashing his national identity number.

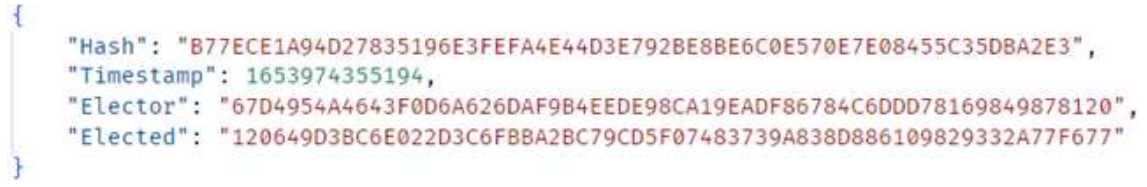
3. The timestamp of the transaction.

All these information are combined and hashed to generate the transaction hash.



```
[ProtoContract(SkipConstructor = true)]
public class Transaction
{
    [ProtoMember(1)] public string Hash { get; set; }
    [ProtoMember(2)] public long Timestamp { get; set; }
    [ProtoMember(3)] public string Elector { get; set; }
    [ProtoMember(4)] public string Elected { get; set; }
    public Transaction(string elector, string elected)
    {
        Timestamp = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
        Elector = elector;
        Elected = elected;
        Hash = Argon2.ComputeHash(Timestamp + Elector + Elected);
    }
}
```

The output is:



```
{
    "Hash": "B77ECE1A94D27835196E3FEFA4E44D3E792BE8BE6C0E570E7E08455C35DBA2E3",
    "Timestamp": 1653974355194,
    "Elector": "67D4954A4643F0D6A626DAF9B4EEDE98CA19EADF86784C6DDD78169849878120",
    "Elected": "120649D3BC6E022D3C6FBBA2BC79CD5F07483739A838D886109829332A77F677"
}
```

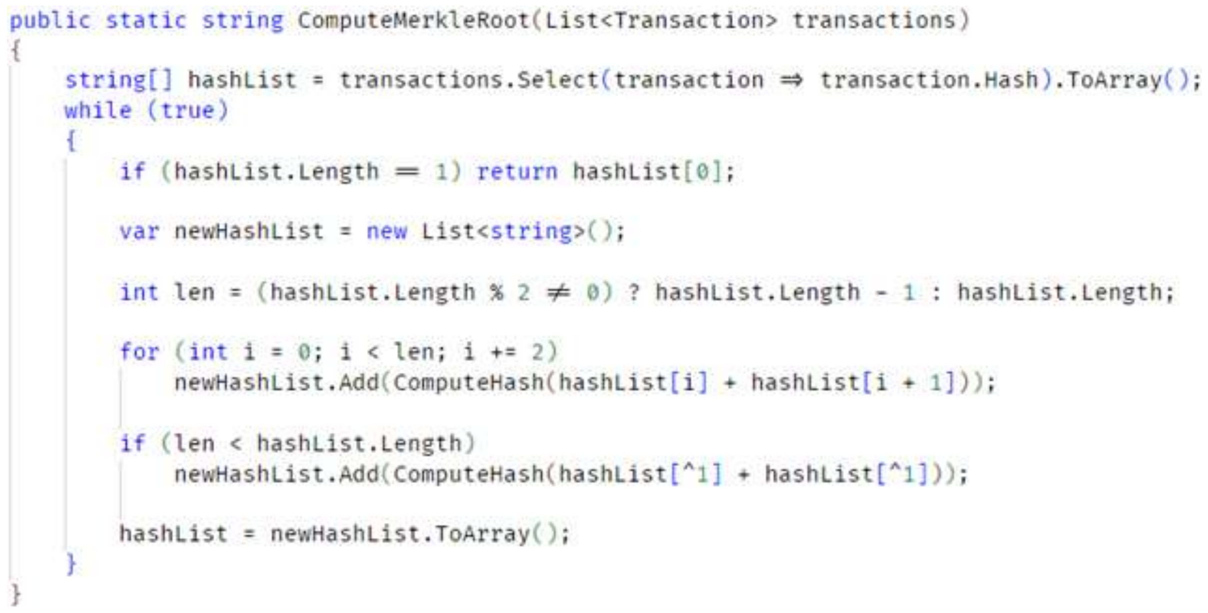
2. Generating Hash of a Block:

Every block header consists of the following information:

1. The Merkle root of all hashes of all transactions data to be added in the block.
2. A hash reference to the previous block.
3. The timestamp when the block was created.

The steps to be performed to add new block to the blockchain are:

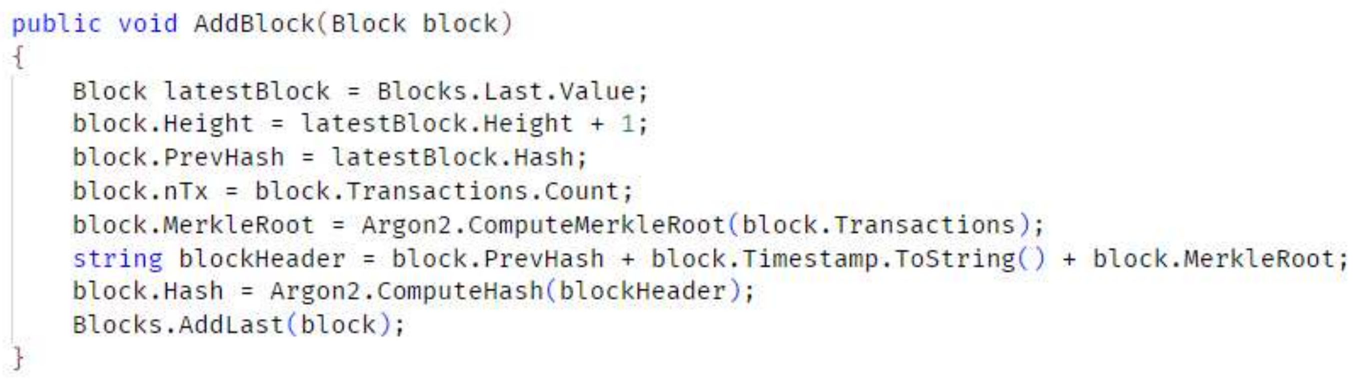
1. Create a Merkle root of all hashes of all transactions to be included in the block.



```
public static string ComputeMerkleRoot(List<Transaction> transactions)
{
    string[] hashList = transactions.Select(transaction => transaction.Hash).ToArray();
    while (true)
    {
        if (hashList.Length == 1) return hashList[0];

        var newHashList = new List<string>();

        int len = (hashList.Length % 2 != 0) ? hashList.Length - 1 : hashList.Length;

        for (int i = 0; i < len; i += 2)
            newHashList.Add(ComputeHash(hashList[i] + hashList[i + 1]));

        if (len < hashList.Length)
            newHashList.Add(ComputeHash(hashList[^1] + hashList[^1]));

        hashList = newHashList.ToArray();
    }
}
```

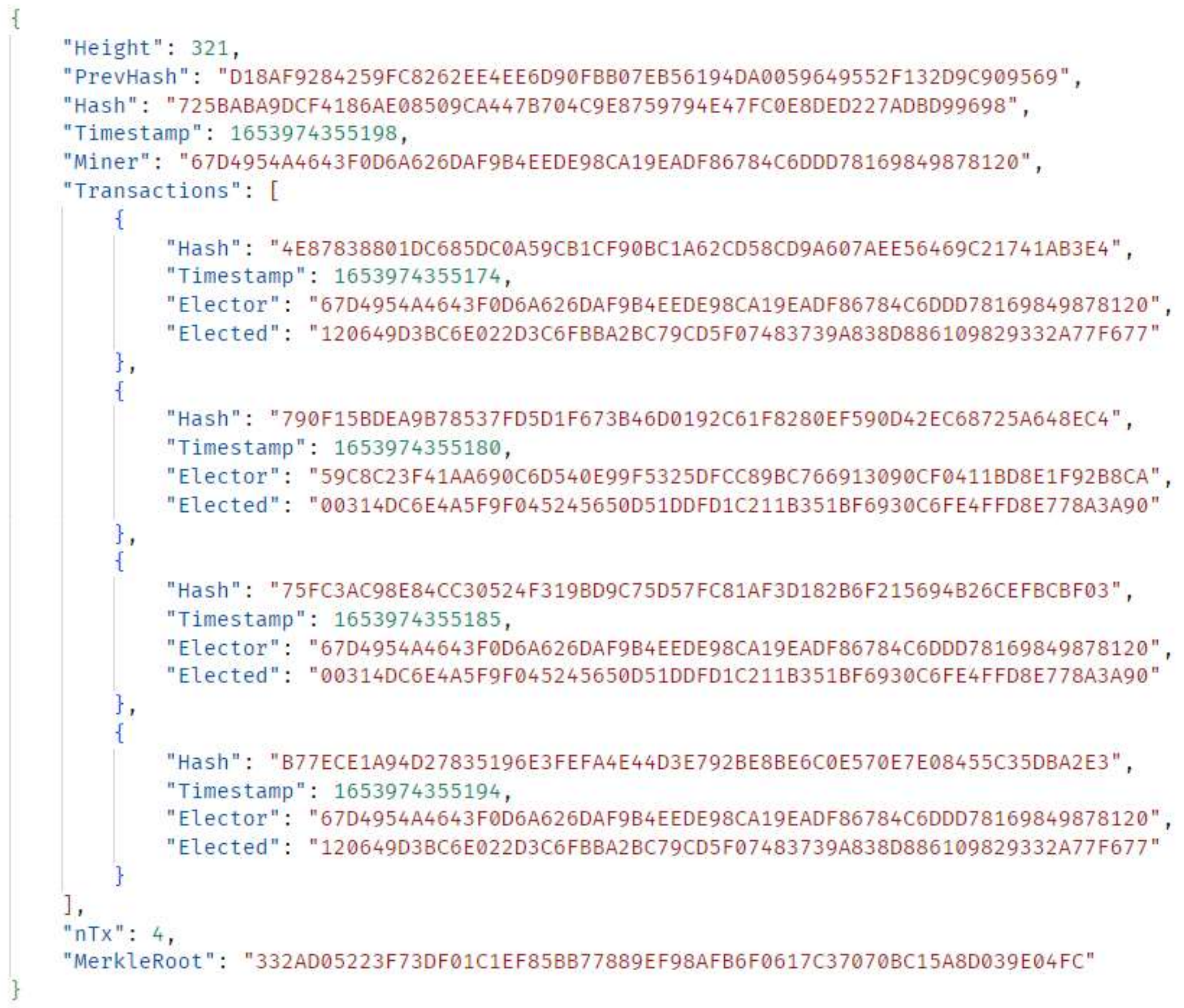
2. Create a block header that contains a hash reference to its preceding block and the Merkle root and also the timestamp.

3. Create the hash of the new block by hashing the block's header.



```
public void AddBlock(Block block)
{
    Block latestBlock = Blocks.Last.Value;
    block.Height = latestBlock.Height + 1;
    block.PrevHash = latestBlock.Hash;
    block.nTx = block.Transactions.Count;
    block.MerkleRoot = Argon2.ComputeMerkleRoot(block.Transactions);
    string blockHeader = block.PrevHash + block.Timestamp.ToString() + block.MerkleRoot;
    block.Hash = Argon2.ComputeHash(blockHeader);
    Blocks.AddLast(block);
}
```

The output is:



```
{
    "Height": 321,
    "PrevHash": "D18AF9284259FC8262EE4EE6D90FBB07EB56194DA0059649552F132D9C909569",
    "Hash": "725BABA9DCF4186AE08509CA447B704C9E8759794E47FC0E8DED227ADBD99698",
    "Timestamp": 1653974355198,
    "Miner": "67D4954A4643F0D6A626DAF9B4EEDE98CA19EADF86784C6DDD78169849878120",
    "Transactions": [
        {
            "Hash": "4E87838801DC685DC0A59CB1CF90BC1A62CD58CD9A607AEE56469C21741AB3E4",
            "Timestamp": 1653974355174,
            "Elector": "67D4954A4643F0D6A626DAF9B4EEDE98CA19EADF86784C6DDD78169849878120",
            "Elected": "120649D3BC6E022D3C6FBBA2BC79CD5F07483739A838D886109829332A77F677"
        },
        {
            "Hash": "790F15BDEA9B78537FD5D1F673B46D0192C61F8280EF590D42EC68725A648EC4",
            "Timestamp": 1653974355180,
            "Elector": "59C8C23F41AA690C6D540E99F5325DFCC89BC766913090CF0411BD8E1F92B8CA",
            "Elected": "00314DC6E4A5F9F045245650D51DDFD1C211B351BF6930C6FE4FFD8E778A3A90"
        },
        {
            "Hash": "75FC3AC98E84CC30524F319BD9C75D57FC81AF3D182B6F215694B26CEFBCBF03",
            "Timestamp": 1653974355185,
            "Elector": "67D4954A4643F0D6A626DAF9B4EEDE98CA19EADF86784C6DDD78169849878120",
            "Elected": "00314DC6E4A5F9F045245650D51DDFD1C211B351BF6930C6FE4FFD8E778A3A90"
        },
        {
            "Hash": "B77ECE1A94D27835196E3FEFA4E44D3E792BE8BE6C0E570E7E08455C35DBA2E3",
            "Timestamp": 1653974355194,
            "Elector": "67D4954A4643F0D6A626DAF9B4EEDE98CA19EADF86784C6DDD78169849878120",
            "Elected": "120649D3BC6E022D3C6FBBA2BC79CD5F07483739A838D886109829332A77F677"
        }
    ],
    "nTx": 4,
    "MerkleRoot": "332AD05223F73DF01C1EF85BB77889EF98AFB6F0617C37070BC15A8D039E04FC"
}
```

6.5.4 Transaction Process

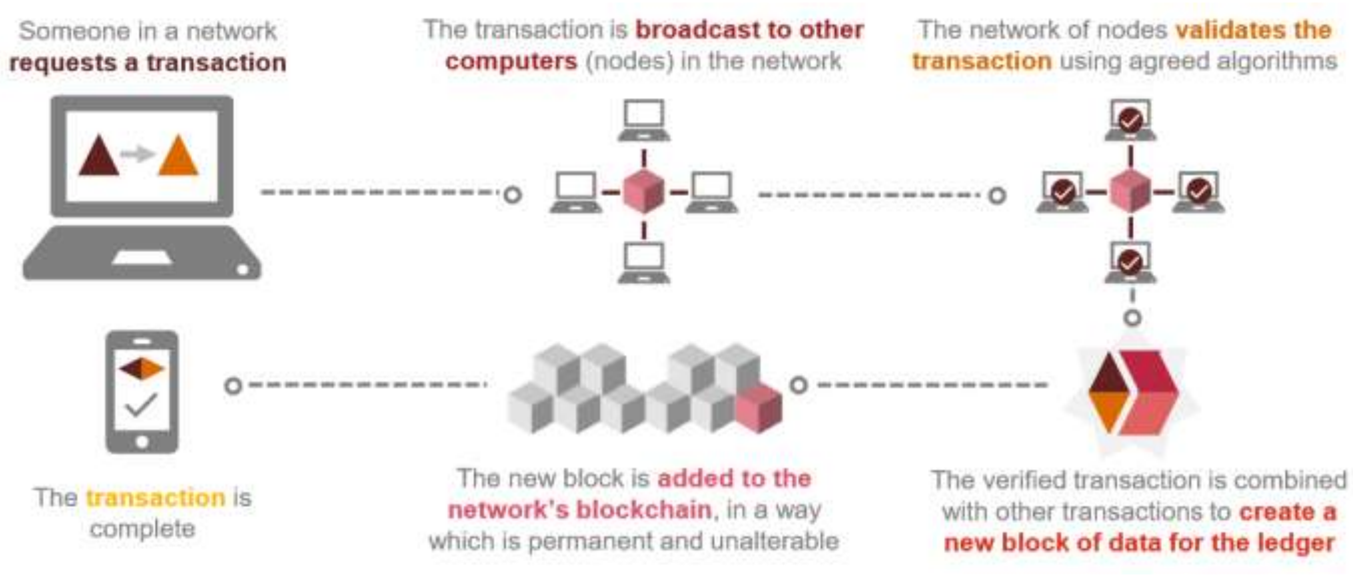

Someone in a network requests a transaction
The transaction is broadcast to other computers (nodes) in the network
The network of nodes validates the transaction using agreed algorithms
The verified transaction is combined with other transactions to create a new block of data for the ledger
The new block is added to the network's blockchain, in a way which is permanent and unalterable
The transaction is complete

Fig. (16) - Transaction Process

A transaction in our system is a vote sent from the machine address to the candidate address.

- Step 1 - A voter requests a transaction. The transaction is generated by the machine.
- Step 2 - Requested transaction is broadcasted to a P2P network with the help of nodes.
- Step 3 - The network nodes validate the transaction via a consensus algorithm.
- Step 4 - Verified transaction is combined with other transactions to create a new block.
- Step 5 - Block is mined and added to blockchain permanently in an unalterable way.

6.5.5 Transaction Pool

Transaction pool is the place which contains all of the unconfirmed transactions. Transaction pool is stored on the node's memory and its contents can be accessed, observed in real time.

Every node running the blockchain client software holds a complete copy of the Blockchain, at the same time, each node contains its own unique set of unconfirmed transactions, means that it’s impossible for a single node to contain the entirety of the transaction pool.

Continuously, transactions will be thrown away to the Transaction pool. Valid transactions are the transactions which are included into the block in the blockchain.

6.5.6 Mining (RNG)

Blockchain mining is the process of creating a new block in the blockchain by the contributing nodes. It involves the network nodes competing to have the right to form the block in the blockchain by doing some sort of computation.

A new block is mined and added to the blockchain every 5 minutes, at the time, every node generates a random number, the generated random numbers from the nodes are then sorted, and the node that has generated the lowest random number earns the right to mine the block.

6.5.7 Consensus Mechanism

There is no central authority present to validate and verify the transactions, yet every transaction in the Blockchain is considered to be completely secured and verified. This is possible only because of the presence of the consensus protocol which is a core part of any Blockchain network. A consensus algorithm is a procedure through which all the peers of the Blockchain network reach a common agreement about the present state of the distributed ledger. In this way, consensus algorithms achieve reliability in the Blockchain network and establish trust between unknown peers in a distributed computing environment.

The consensus protocol makes sure that every new block that is added to the Blockchain is the one and only version of the truth that is agreed upon by all the nodes in the Blockchain.

Let's see what happens when any node creates a new block. This new block is sent to all the nodes on the network. Each node needs to verify the block to make sure that it hasn't been altered. After complete checking, each node adds this block to their blockchain copy.

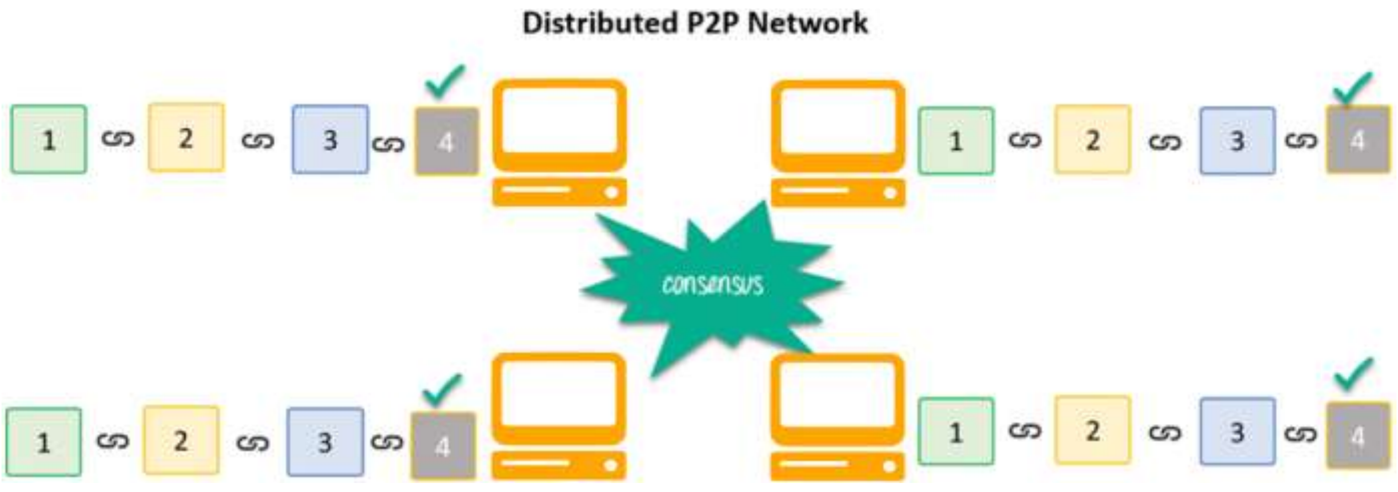


Fig. (17) - Consensus Protocol

All these nodes in this network create a consensus. They agree about what blocks are valid and which are not. Nodes in the network will reject any blocks that are tampered with.

So, to successfully tamper with a blockchain

- You will need to tamper with all the subsequent blocks on the chain.
- Recompute the hash value for each block.
- Take control of greater than 50% of the peer-to-peer network.

After doing all these, the tampered block becomes accepted by the majority resulting in a fork.

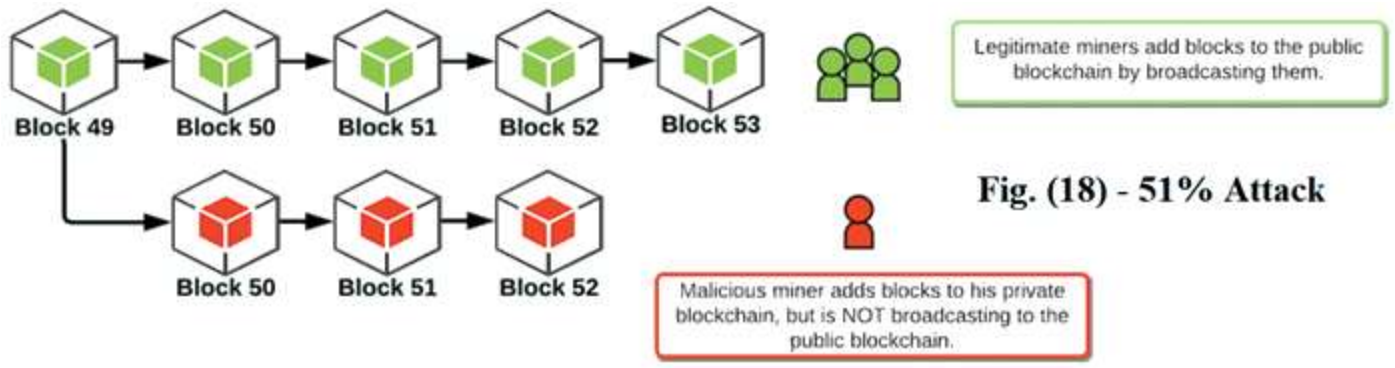

Block 49
Block 50
Block 51
Block 52
Block 53
Legitimate miners add blocks to the public blockchain by broadcasting them.
Block 50
Block 51
Block 52
Malicious miner adds blocks to his private blockchain, but is NOT broadcasting to the public blockchain.

Fig. (18) - 51% Attack

This cascade effect ensures that once a block has many generations following it, it cannot be changed without forcing a recalculation of all subsequent blocks. Because such a recalculation would require enormous computation, the existence of a long chain of blocks makes the blockchain's deep history immutable, which is a key feature of system's security.

6.5.8 Blockchain Variants

- Public: It's a permissionless blockchain, ledgers are visible to everyone on the internet. It allows anyone to verify and add a block of transactions to the blockchain. Public networks have incentives for people to join and are free for use. Anyone can use a public blockchain.

- Private: The private blockchain is run by a single organization/authority. It allows only specific participants to verify and add transaction blocks. However, anyone on the internet is generally allowed to view it, which is the case in our system.

- Consortium: In this Blockchain variant, only a group of organizations can verify and add transactions. Here, the ledger can be open or restricted to select groups. Consortium blockchain is used cross-organizations. It is only controlled by pre-authorized nodes.

6.5.9 Why do we need Blockchain ?

1. Resilience: The blockchain can still be operated by the majority of the network uncompromised nodes in the event of a massive attack against the system.

2. Time reduction: Blockchain can play a vital role by allowing quicker settlement of transactions as it does not need a lengthy process of verification and settlement because a single version of agreed-upon data of the shared ledger is available between all nodes.

3. Reliability: Blockchain certifies and verifies the identities of the interested parties. This removes double records, reduces rates, and accelerates transactions.

4. Unchangeable transactions: By registering transactions in chronological order, Blockchain certifies the inalterability of all operations, which means when any new block has been added to the chain of ledgers, it cannot be removed nor modified.

5. Fraud prevention: The concepts of shared information and consensus prevent possible losses due to fraud or embezzlement. Thus, blockchain doesn't even require monitoring.

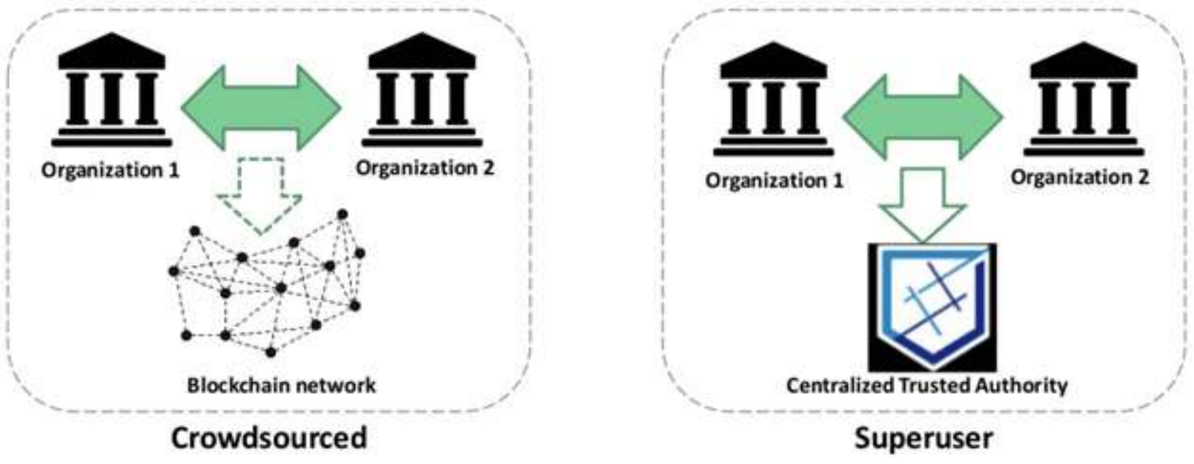
6. Security: Attacking a traditional database is the bringing down of a specific target. With the help of Distributed Ledger Technology, each party holds a copy of the original chain, so the system remains operative, even if a large number of other nodes fall.

7. Transparency: Changes to the blockchains are publicly viewable to everyone. This offers greater transparency and credibility where all transactions are immutable.

8. Collaboration: Allows parties to transact directly with each other without the need for mediating third parties or central authorities.

9. Consensus: There are standards and rules on how every node exchanges the blockchain information. This method ensures that all transactions are validated, and all valid transactions are added one by one.

6.5.10 Blockchain Vs. Shared Database



Parameters	**Blockchain**	**Shared Database**
Operations	Insert	Create/ Read/ Update and Delete
Replication	Full replication on every peer	Master-Slave Multi-Master
Consensus	Most of the peers agree on the outcome of transactions.	Distributed transactions which held in two phases commit and Paxos.
Validation	Global rules are enforced on the whole blockchain system.	Offers only local integrity constraints.
Disintermediation	It is allowed with blockchain.	Not allowed.
Confidentiality	Fully confidential.	Not totally confidential.
Robustness	Fully robust technology.	Not entirely robust.

Fig. (19) - Blockchain Vs. Shared Database

6.6 LevelDB

Google’s LevelDB database is used for storing the system's blockchain metadata.

LevelDB is a fast open-source on-disk key-value storage library written at Google that provides an ordered mapping from string keys to string values based on a comparator function.

LevelDB stores keys and values in arbitrary byte arrays, and data is sorted by key. It supports batching writes, forward and backward iteration, and compression of the data via Google's Snappy compression library.

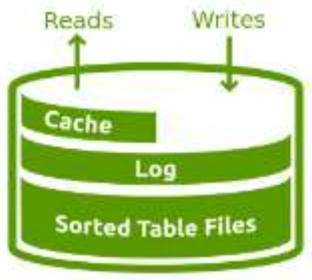


Fig. (20) - LevelDB

6.6.1 Properties

- Keys and values are arbitrary byte arrays.
- Data stored is sorted by key.
- Callers can provide a custom comparison function to override the sort order.
- The basic operations are **Put(**key,value**)**, **Get(**key**)**, **Delete(**key**)**.
- Multiple changes can be made in one atomic batch.
- Users can create a transient snapshot to get a consistent view of data.
- Forward and backward iteration is supported over the data.
- Data is automatically compressed using the Snappy compression library.
- External activity (file system operations etc.) is relayed through a virtual interface so users can customize the operating system interactions.

This is not a SQL database. It does not have a relational data model, it does not support SQL queries, and it has no support for indexes. Only a single process (possibly multi-threaded) can access a particular database at a time. There is no client-server support built-in to the library. An application that needs such support will have to wrap their own server around the library.

Google has provided benchmarks comparing LevelDB's performance to SQLite and Kyoto Cabinet in different scenarios. LevelDB outperforms both SQLite and Kyoto Cabinet in write operations and sequential-ordered read operations. LevelDB also excels at batch writes.

6.6.2 Data Storage

It temporarily keeps the data into MemTable and periodically transfers data from MemTable to SSTable (Sorted String Table). SSTable is immutable so, the data are immutable. LevelDB compacts to minimize invalid data in each level, then generates a new block at the next level.

In LevelDB, immutable data are stored in a disk, and this can be shared by different clustered nodes. Overall, there are a total of 7 levels to keep data as well as at most two in-memory tables. Firstly, the system buffer writes the data into an in-memory table called MemTable when this MemTable becomes full it flushes data to disk. Every 7 levels contain multiple tables called as SSTable. In LevelDb, the down-level maintains a larger capacity than the upper level. When the upper level becomes full, the system pushes the data to the down level, down level reads and writes data into multiple SSTable.

6.6.3 Features

1. Checkpoints Blocking

When operation logging file exceeds over the limit, it will do checkpoints. Data will be flushed to the disk. And compaction scheme will be called. So, data will go down levels. Aside from that, LevelDB will generate new logging file and MemTable for new use.

2. Compression: Naïve (Record-Level)

3 .Concurrency Control: Two-Phase Locking (Deadlock Prevention)

LevelDB only allow one process to open at a time. The OS will use the locking scheme to prevent concurrent access. Within one process, LevelDB can be accessed by multiple threads. For multi-writers, it will only allow the first writer to write to database and other writers will be blocked. For read-write conflicts, readers retrieve data from immutable which is separated from writing process. The updated version will come into effect in compaction process.

4. Indexes: Skip List Log-Structured Merge Tree

It uses skip list in MemTable. Aside from that, LSM-tree is one type of write-optimized B-tree variants consisting of key-value pairs. The LSM-tree is a persistent key-value store optimized for insertions and deletions. LevelDB is an open source LSM-tree implementation.

5. Isolation Levels: Snapshot Isolation

It saves the state of database at a given point and supports reference to it. Users can retrieve data from specific snapshot at the time the snapshot was created.

6. Logging: Logical Logging

Before every insertion, update or delete, system need to add the message to log. In case of node's failure, uncommitted messages can be retrieved and redo operation again for recovery.

7. Query Interface: Custom API

Keys and values in LevelDB are byte arrays with arbitrary length. It supports basic operations like Put(), Get(), Delete(). It also supports Batch operations: Batch(). The whole process of operations will run together and return result in a single Batch operation. However, it does not support SQL queries because this is not a SQL type database.

8. Storage Architecture: Disk-Oriented

It puts temporarily accessed data into MemTable and periodically moves data from MemTable into an immutable SSTable. Aside from that, it adopts compaction to reduce the invalid data in each level and then generates one new block at the next level. It uses MMap or native read SysCall to read SSTable for querying. The OS caches SSTable for LevelDB. LevelDB supports using snappy to compress the value, to avoid uncompressed data for each query.

9. Storage Model: N-ary Storage Model (Row/Record)

SSTable uses NSM to arrange data. It contains a set of arbitrary, sorted key-value pairs. At the end of the block, it provides the start offset and key value for each block. So, bloom filter can be used to search for target block.

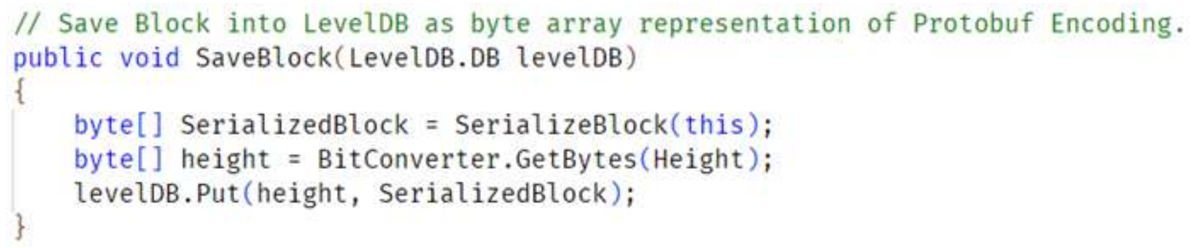
10. System Architecture: Embedded

In LevelDB immutable are stored on the disk which can be shared by different cluster nodes. There are totally 7 levels plus at most two in-memory tables. The procedure can be described as firstly the system buffers write operations in an in-memory table called MEMTable and flushes data to disk when it becomes full. On the disk, tables are organized into levels.

6.6.4 Usage

LevelDB is used as a key-value storage of our blockchain, where the key is the block height, and the value is the block's metadata followed by the list of transactions' metadata.

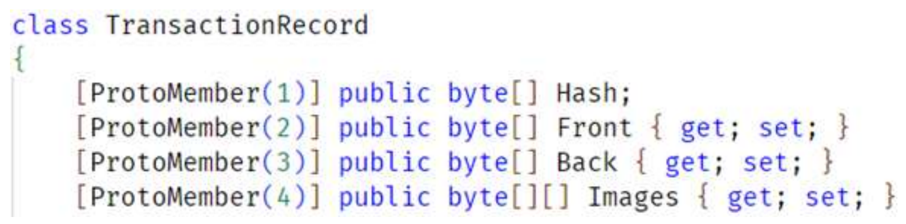
6.6.4.1 Storing Blocks



```
// Save Block into LevelDB as byte array representation of Protobuf Encoding.
public void SaveBlock(LevelDB.DB levelDB)
{
    byte[] SerializedBlock = SerializeBlock(this);
    byte[] height = BitConverter.GetBytes(Height);
    levelDB.Put(height, SerializedBlock);
}
```

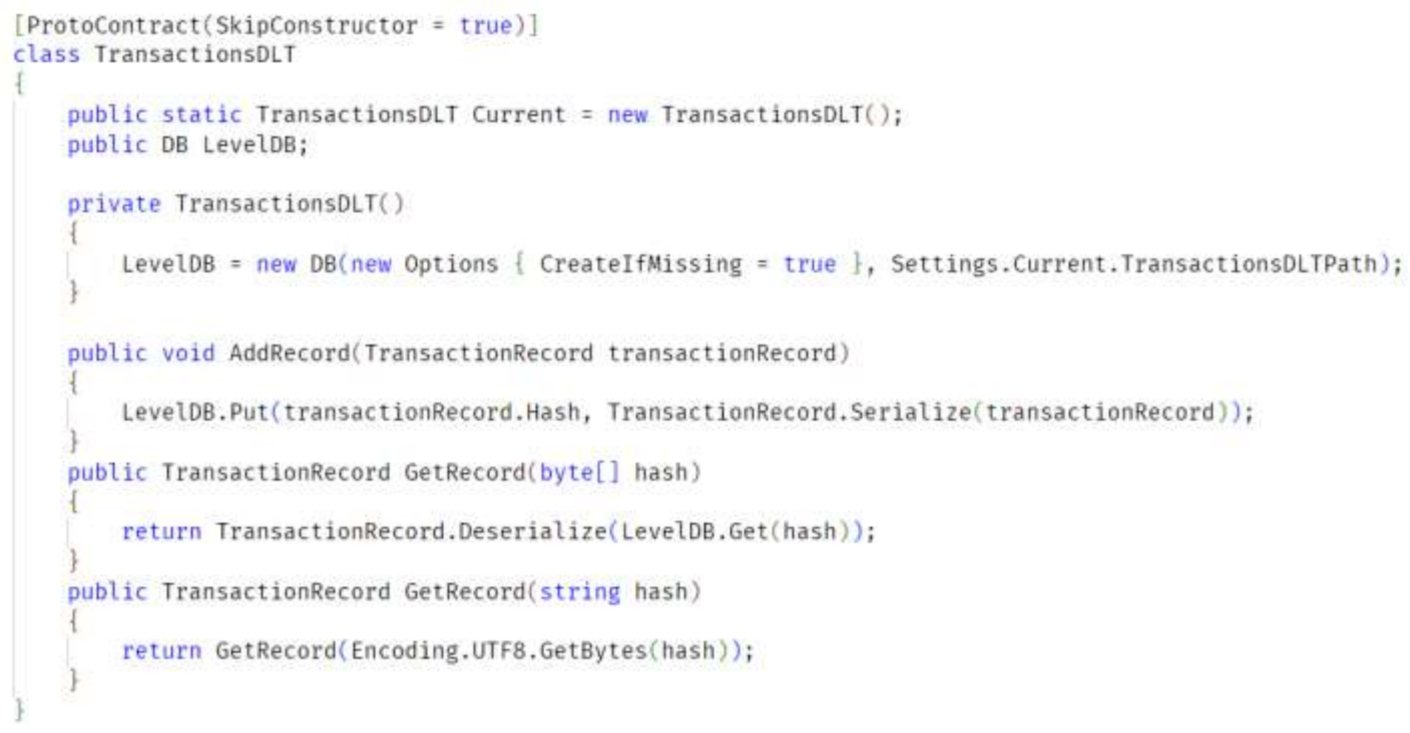
6.6.4.2 Storing Transactions Records

Additionally, LevelDB is utilized to store the transaction record where the key is the transaction's hash, and the value is byte images of front and back side of the identification card of the voter along with bytes frames captured live during the voting process.



```
class TransactionRecord
{
    [ProtoMember(1)] public byte[] Hash;
    [ProtoMember(2)] public byte[] Front { get; set; }
    [ProtoMember(3)] public byte[] Back { get; set; }
    [ProtoMember(4)] public byte[][] Images { get; set; }
```

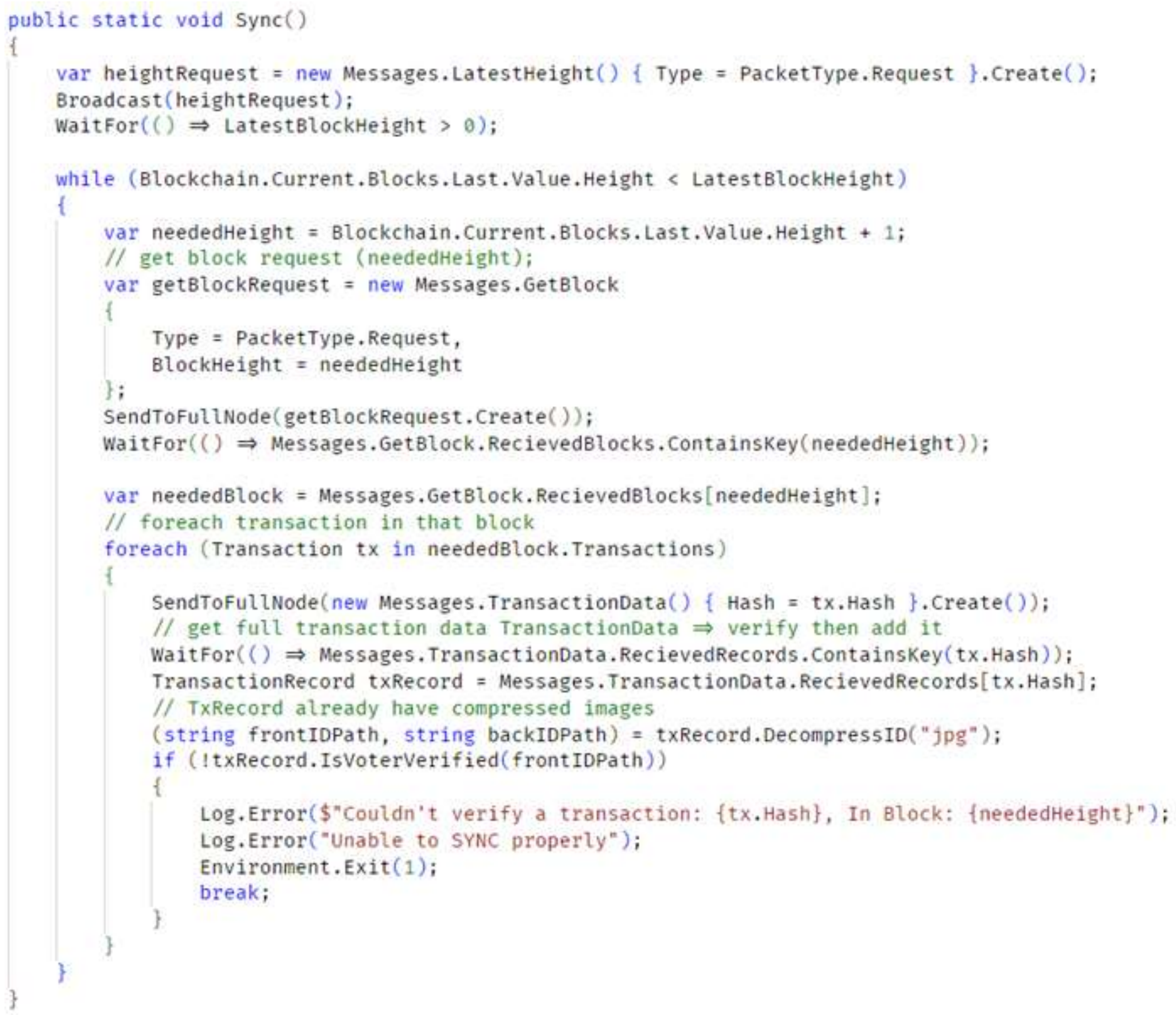
Each transaction record gets stored in transactionsDLT (a LevelDB instance which only exists on a full node). A full node is a node that keeps track of history of all the transactions.



```
[ProtoContract(SkipConstructor = true)]
class TransactionsDLT
{
    public static TransactionsDLT Current = new TransactionsDLT();
    public DB LevelDB;

    private TransactionsDLT()
    {
        LevelDB = new DB(new Options { CreateIfMissing = true }, Settings.Current.TransactionsDLTPath);
    }

    public void AddRecord(TransactionRecord transactionRecord)
    {
        LevelDB.Put(transactionRecord.Hash, TransactionRecord.Serialize(transactionRecord));
    }
    public TransactionRecord GetRecord(byte[] hash)
    {
        return TransactionRecord.Deserialize(LevelDB.Get(hash));
    }
    public TransactionRecord GetRecord(string hash)
    {
        return GetRecord(Encoding.UTF8.GetBytes(hash));
    }
}
```

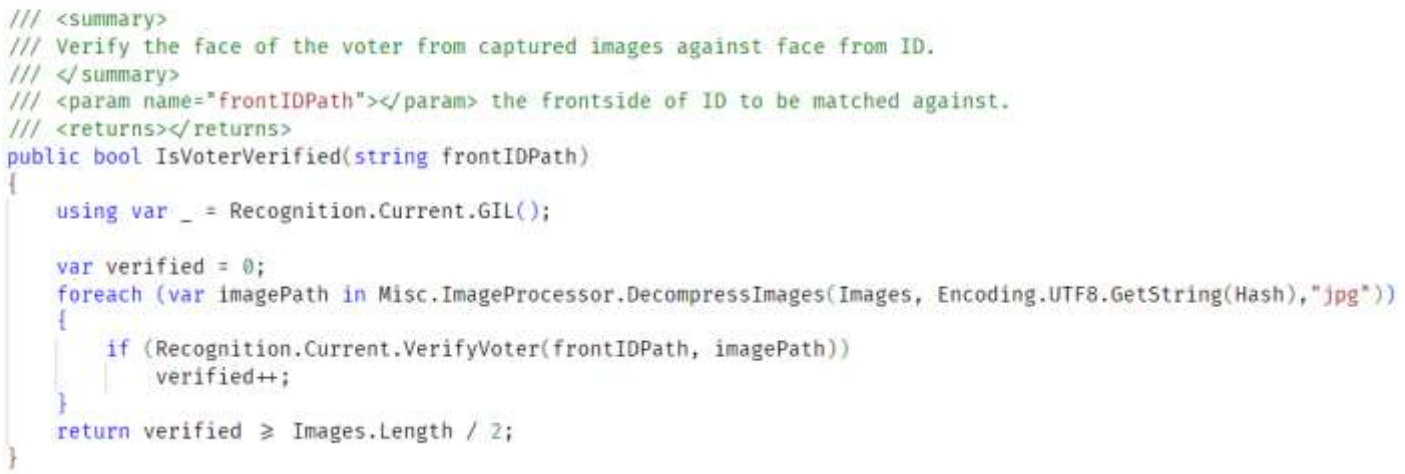
6.6.4.3 Synchronization & Verification

An out of sync node or a new one to the network have to sync it’s blockchain copy and need to query the transaction records data from a full node.



```
public static void Sync()
{
    var heightRequest = new Messages.LatestHeight() { Type = PacketType.Request }.Create();
    Broadcast(heightRequest);
    WaitFor(() ⇒ LatestBlockHeight > 0);

    while (Blockchain.Current.Blocks.Last.Value.Height < LatestBlockHeight)
    {
        var neededHeight = Blockchain.Current.Blocks.Last.Value.Height + 1;
        // get block request (neededHeight);
        var getBlockRequest = new Messages.GetBlock
        {
            Type = PacketType.Request,
            BlockHeight = neededHeight
        };
        SendToFullNode(getBlockRequest.Create());
        WaitFor(() ⇒ Messages.GetBlock.RecievedBlocks.ContainsKey(neededHeight));

        var neededBlock = Messages.GetBlock.RecievedBlocks[neededHeight];
        // foreach transaction in that block
        foreach (Transaction tx in neededBlock.Transactions)
        {
            SendToFullNode(new Messages.TransactionData() { Hash = tx.Hash }.Create());
            // get full transaction data TransactionData ⇒ verify then add it
            WaitFor(() ⇒ Messages.TransactionData.RecievedRecords.ContainsKey(tx.Hash));
            TransactionRecord txRecord = Messages.TransactionData.RecievedRecords[tx.Hash];
            // TxRecord already have compressed images
            (string frontIDPath, string backIDPath) = txRecord.DecompressID("jpg");
            if (!txRecord.IsVoterVerified(frontIDPath))
            {
                Log.Error($"Couldn't verify a transaction: {tx.Hash}, In Block: {neededHeight}");
                Log.Error("Unable to SYNC properly");
                Environment.Exit(1);
                break;
            }
        }
    }
}
```

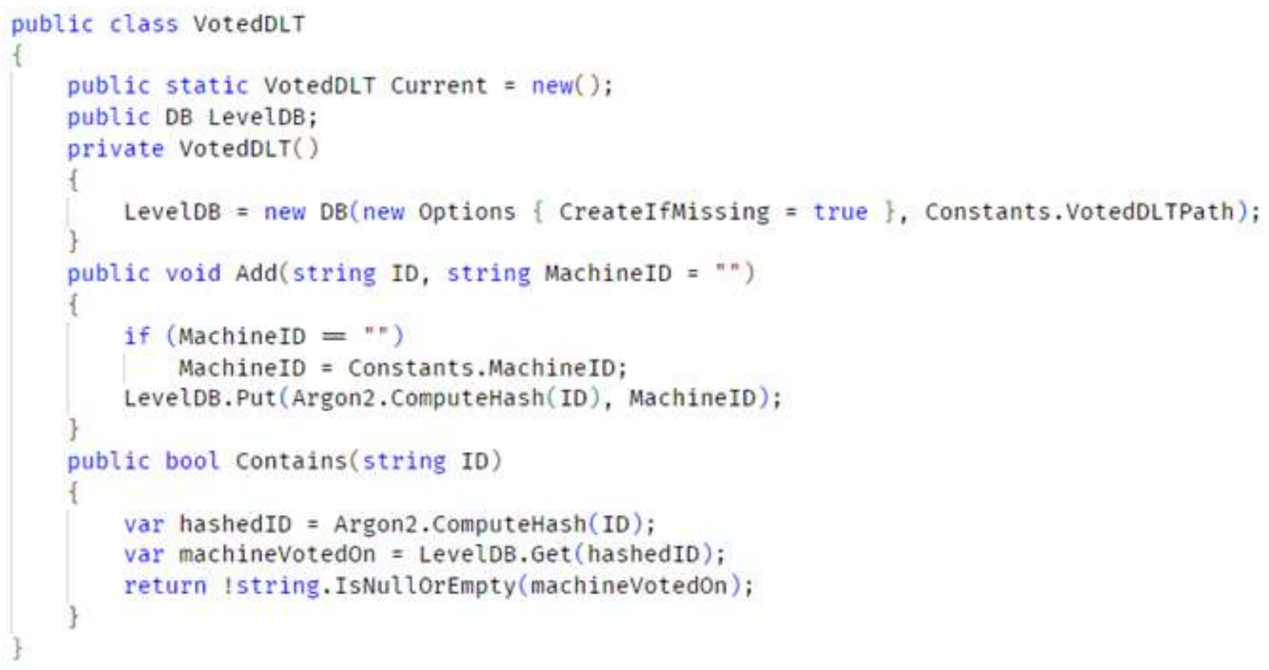
After the node gets the transactions data, it needs to verify it.



```
/// <summary>
/// Verify the face of the voter from captured images against face from ID.
/// </summary>
/// <param name="frontIDPath"></param> the frontside of ID to be matched against.
/// <returns></returns>
public bool IsVoterVerified(string frontIDPath)
{
    using var _ = Recognition.Current.GIL();

    var verified = 0;
    foreach (var imagePath in Misc.ImageProcessor.DecompressImages(Images, Encoding.UTF8.GetString(Hash),"jpg"))
    {
        if (Recognition.Current.VerifyVoter(frontIDPath, imagePath))
            verified++;
    }
    return verified ≥ Images.Length / 2;
}
```

6.6.4.4 Storing Hashes of the Already Voted

LevelDB is also used to store a list of the hashes of identification numbers for the already voted, to detect frauds and prevent double voting.



```
public class VotedDLT
{
    public static VotedDLT Current = new();
    public DB LevelDB;
    private VotedDLT()
    {
        LevelDB = new DB(new Options { CreateIfMissing = true }, Constants.VotedDLTPath);
    }
    public void Add(string ID, string MachineID = "")
    {
        if (MachineID == "")
            MachineID = Constants.MachineID;
        LevelDB.Put(Argon2.ComputeHash(ID), MachineID);
    }
    public bool Contains(string ID)
    {
        var hashedID = Argon2.ComputeHash(ID);
        var machineVotedOn = LevelDB.Get(hashedID);
        return !string.IsNullOrEmpty(machineVotedOn);
    }
}
```

6.7 Protocol Buffers (Protobuf)

Protobuf is Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data like XML, but smaller, faster, and simpler. We define how we want our data to be structured once, then we can use special generated source code to easily write and read our structured data to and from a variety of data streams and using a variety of languages.

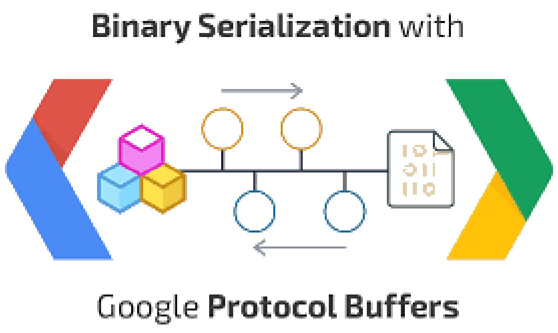


Fig. (21) - Protobuf

Protobuf provides a mechanism for serializing structured data in a forward compatible and backward compatible way like JSON, except it's faster & generates native language binding.

Protocol buffers are a combination of the definition language (created in .proto files), the code that the proto compiler generates to interface with data, language-specific runtime libraries, and the serialization format for data that is written to a file or sent across a network connection.

6.7.1 Addressed Problem

Protocol buffers provide a serialization format for packets of typed, structured data that are up to a few megabytes in size. The format is suitable for both ephemeral network traffic and long-term data storage. Protocol buffers can be extended with new information without invalidating existing data or requiring code to be updated.

Protocol buffers are the most commonly-used data format at Google. They are used extensively in inter-server communications as well as for archival storage of data on disk. Protocol buffer messages and services are described by engineer-authored .proto files.

6.7.2 Mechanism

The proto compiler is invoked at build time on .proto files to generate code in various programming languages to manipulate the corresponding protocol buffer. Each generated class contains simple accessors for each field and methods to serialize and parse the whole structure to and from raw bytes.

Because protocol buffers are used extensively across all manner of services at Google and data within them may persist for some time, maintaining backwards compatibility is crucial. Protocol buffers allow for the seamless support of changes, including the addition of new fields and the deletion of existing fields, to any protocol buffer without breaking existing services.

6.7.3 Benefits

Protocol buffers are ideal for any situation in which we need to serialize structured, record-like, typed data in a language-neutral, platform-neutral, extensible manner. They are most often used for defining communications protocols (together with gRPC) and for data storage.

Some of the advantages of using protocol buffers include:

- Compact Data Storage
- Fast Parsing
- Availability in Many Programming Languages
- Optimized Functionality through Automatically-Generated Classes

6.7.3.1 Cross-language Compatibility

The same messages can be read by code written in any supported programming language. We can have a Java program on one platform capture data from one software system, serialize it based on a .proto definition, and then extract specific values from that serialized data in a separate Python application running on another platform.

6.7.3.2 Cross-Project Support

We can use protocol buffers across projects by defining message types in .proto files that reside outside of a specific project’s code base. If we're defining message types or enums that we anticipate will be widely used outside of our immediate team, we can put them in their own file with no dependencies.

6.7.3.3 Updating Proto Definitions Without Updating Code

It’s standard for software products to be backward compatible, but it is less common to be forward compatible. If some simple practices are followed when updating .proto definitions, old code will read new messages without issues, ignoring newly added fields. To the old code, fields that were deleted will have their default value, and deleted repeated fields will be empty.

6.7.4 How do Protocol Buffers work ?

The following diagram shows how we use protocol buffers to work with our data.

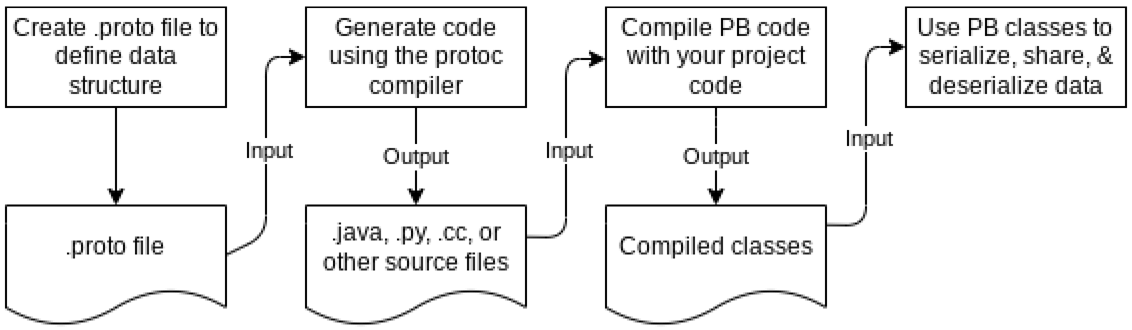

Create .proto file to define data structure
.proto file
Input
Generate code using the protoc compiler
Output
.java, .py, .cc, or other source files
Input
Compile PB code with your project code
Output
Compiled classes
Input
Use PB classes to serialize, share, & deserialize data

Fig. (22) - Protocol Buffers Workflow

The code generated by protocol buffers provides utility methods to retrieve data from files and streams, extract individual values from the data, check if data exists, serialize data back to a file or stream, and other useful functions.

Compiling this .proto file creates a Builder class that we can use to create new instances, we can then deserialize data using the methods protocol buffers creates in other languages.

Canonically, messages are serialized into a binary wire format which is compact, forward and backward-compatible, but not self-describing (that is, there is no way to tell the names, meaning, or full datatypes of fields without an external specification). There is no defined way to include or refer to such an external specification (schema) within a Protocol Buffers file. The officially supported implementation includes an ASCII serialization format, but this format - though self-describing - loses the forward and backward-compatibility behavior, and is thus not a good choice for applications other than debugging.

6.7.5 Definition Syntax

When defining .proto files, we can specify that a field is either optional or repeated (proto2 and proto3) or singular (proto3). The option to set a field to required is absent in proto3 and strongly discouraged in proto2.

After setting the optionality/repeatability of a field, we specify the data type. Protocol buffers support the usual primitive data types, such as integers, booleans, and floats.

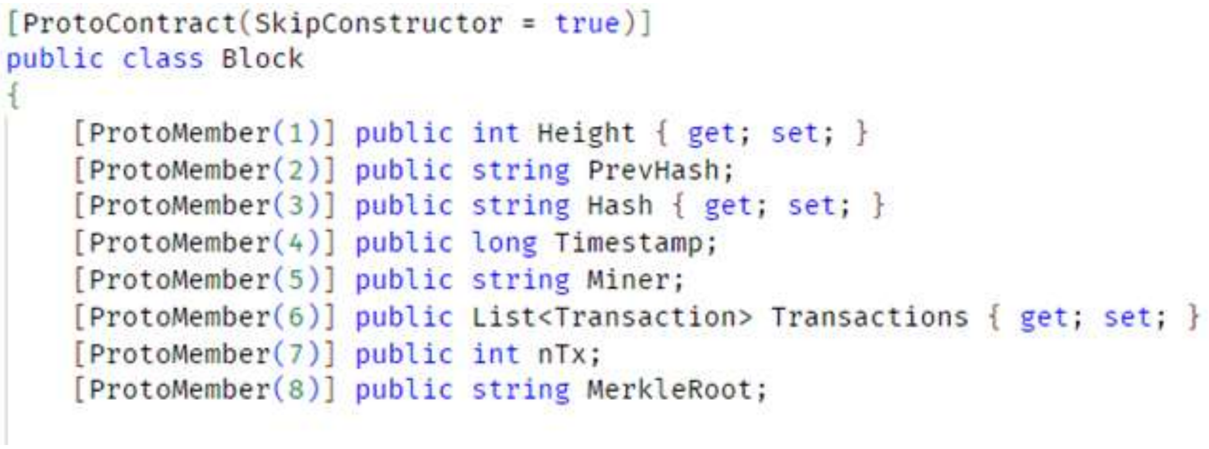
6.7.6 Usage

Protocol Buffer is used as:

- A binary serialization/exchange format.
- A storage format.

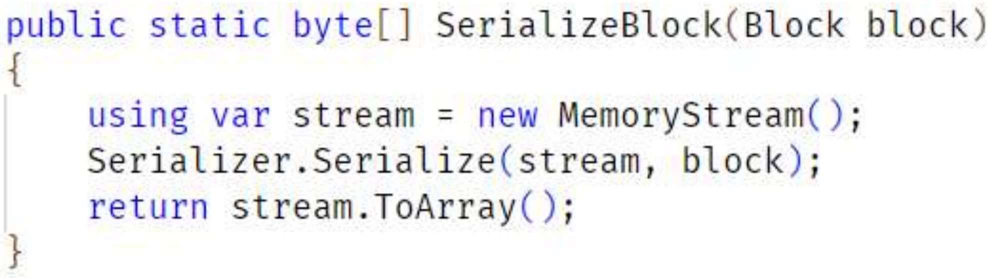
6.7.6.1 Serializing a Block

First, the structure of the block to be serialized is specified.



```
[ProtoContract(SkipConstructor = true)]
public class Block
{
    [ProtoMember(1)] public int Height { get; set; }
    [ProtoMember(2)] public string PrevHash;
    [ProtoMember(3)] public string Hash { get; set; }
    [ProtoMember(4)] public long Timestamp;
    [ProtoMember(5)] public string Miner;
    [ProtoMember(6)] public List<Transaction> Transactions { get; set; }
    [ProtoMember(7)] public int nTx;
    [ProtoMember(8)] public string MerkleRoot;
```

Protocol Buffer is then used to serialize the block's structure.

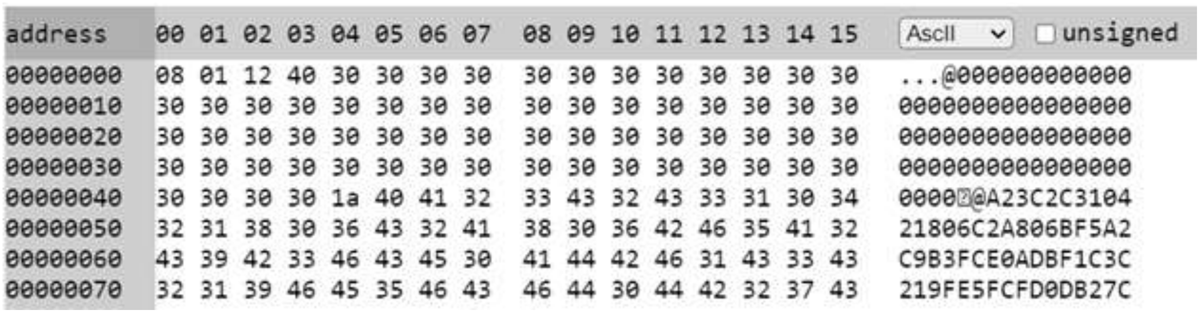


```
public static byte[] SerializeBlock(Block block)
{
    using var stream = new MemoryStream();
    Serializer.Serialize(stream, block);
    return stream.ToArray();
}
```

The output is a byte array ready for transmission and storage.

6.7.6.2 Storing a Block

This is how the Protobuf format is stored on the desk on LevelDB as a byte array.

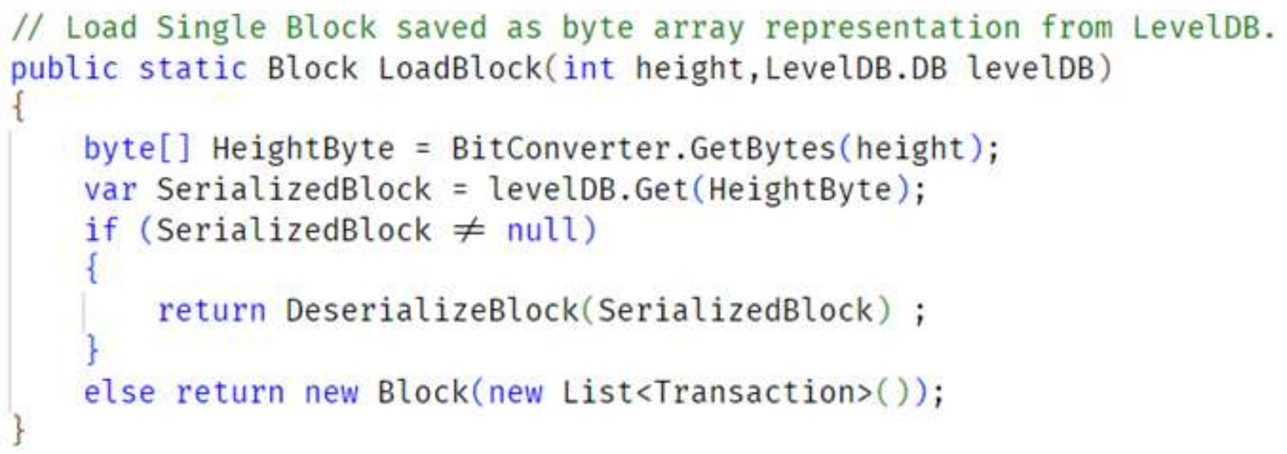


```
address   00 01 02 03 04 05 06 07  08 09 10 11 12 13 14 15   AscII  unsigned
00000000  08 01 12 40 30 30 30 30  30 30 30 30 30 30 30 30   ...@000000000000
00000010  30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30   0000000000000000
00000020  30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30   0000000000000000
00000030  30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30   0000000000000000
00000040  30 30 30 30 1a 40 41 32  33 43 32 43 33 31 30 34   0000▯@A23C2C3104
00000050  32 31 38 30 36 43 32 41  38 30 36 42 46 35 41 32   21806C2A806BF5A2
00000060  43 39 42 33 46 43 45 30  41 44 42 46 31 43 33 43   C9B3FCE0ADBF1C3C
00000070  32 31 39 46 45 35 46 43  46 44 30 44 42 32 37 43   219FE5FCFD0DB27C
```

Table (3) - Block Storage

Storing the block in this format saves space, secures the data, and speed up the process of sending this block to any node as we just need to deserialize it.

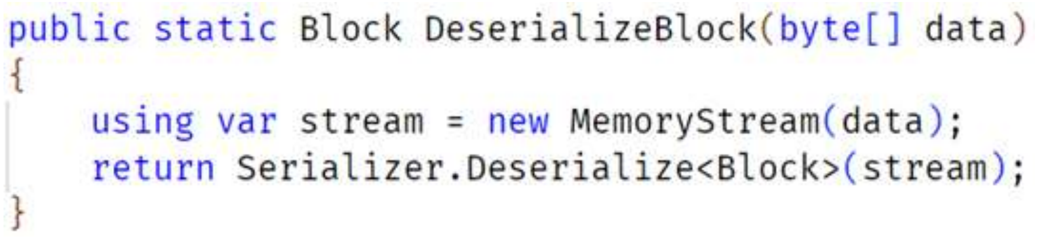
When this block is needed, we can query it from LevelDB by its height.



```
// Load Single Block saved as byte array representation from LevelDB.
public static Block LoadBlock(int height,LevelDB.DB levelDB)
{
    byte[] HeightByte = BitConverter.GetBytes(height);
    var SerializedBlock = levelDB.Get(HeightByte);
    if (SerializedBlock ≠ null)
    {
        return DeserializeBlock(SerializedBlock) ;
    }
    else return new Block(new List<Transaction>());
}
```

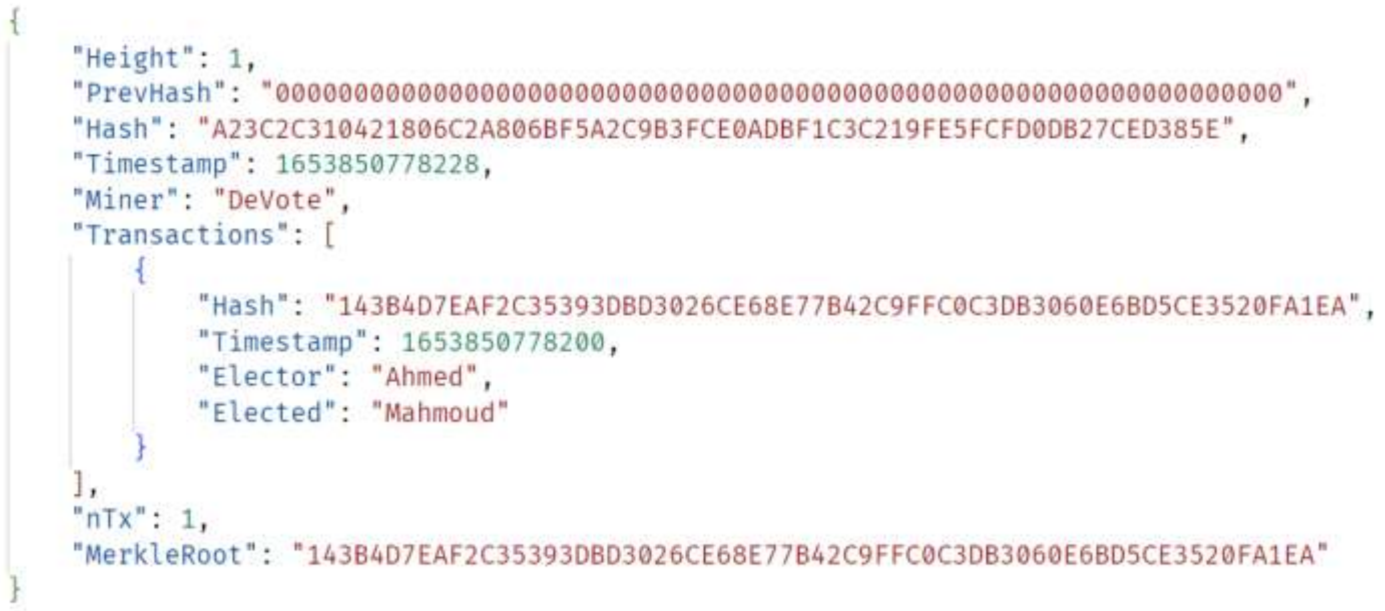
6.7.6.3 Deserializing a Block

After loading the serialized block from LevelDB, it gets deserialized to its original structure.



```
public static Block DeserializeBlock(byte[] data)
{
    using var stream = new MemoryStream(data);
    return Serializer.Deserialize<Block>(stream);
}
```

The output is



```
{
    "Height": 1,
    "PrevHash": "0000000000000000000000000000000000000000000000000000000000000000",
    "Hash": "A23C2C310421806C2A806BF5A2C9B3FCE0ADBF1C3C219FE5FCFD0DB27CED385E",
    "Timestamp": 1653850778228,
    "Miner": "DeVote",
    "Transactions": [
        {
            "Hash": "143B4D7EAF2C35393DBD3026CE68E77B42C9FFC0C3DB3060E6BD5CE3520FA1EA",
            "Timestamp": 1653850778200,
            "Elector": "Ahmed",
            "Elected": "Mahmoud"
        }
    ],
    "nTx": 1,
    "MerkleRoot": "143B4D7EAF2C35393DBD3026CE68E77B42C9FFC0C3DB3060E6BD5CE3520FA1EA"
}
```

6.8 Blockchain Explorer

Blockchain explorer is an online blockchain search engine and browser that can show the details of all transactions that have ever happened on the blockchain network.

A blockchain explorer can be used to find any specific transaction or view the recent history of the blockchain, generally it can be used to extract virtually any data related to transactions.

A blockchain explorer is a piece of software that uses an API and blockchain node to draw various data from a blockchain and then uses a database to arrange the searched data and to present the data to the user in a searchable format.

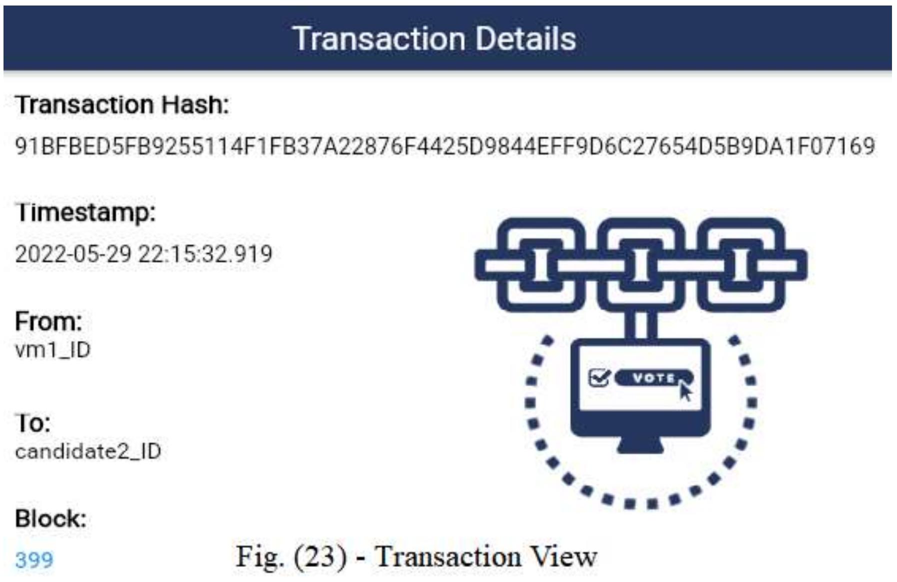
User’s inputs are searchable terms on the explorer which are then searched through an organized table on the database. The explorer will already have organized data from a blockchain into the table format.

A blockchain explorer will allow users to view blockchain activity, search and explore data about recently mined blocks or recently carried out transactions on the blockchain. Ideally, they allow viewing a live feed of blocks as they are mined, as well as the data related to them.

Why the need for Blockchain Explorer ?

- Transparency

Blockchain Explorer allows the voter to validate the authenticity of his vote by checking the result of his vote using the provided Transaction ID/Hash, ensuring that his vote has been sent out to the desired candidate. It acts as public evidence of the system’s integrity and credibility.


Transaction Details

Transaction Hash:
91BFBED5FB9255114F1FB37A22876F4425D9844EFF9D6C27654D5B9DA1F07169

Timestamp:
2022-05-29 22:15:32.919

From:
vm1_ID

To:
candidate2_ID

Block:
399

Fig. (23) - Transaction View

6.8.1 BC Explorer Components

1. Relational Database:

A relational database allows the storage of data components in a table such that they are related to each other. This allows the simplification of the management of data. For instance, instead of having a large block table containing all details for each block, the block table entries can be linked to an entry in a header table.

2. Structured Query Language (SQL or Sequel):

This is a protocol and format for giving a query or searching a database. For instance, the software can create a table in the database, insert new records on the table, search the term, create a new table of results, and then present the results on a web page to the person searching.

3. Application Programming Interface (API):

This is the protocol that allows users to communicate with machines through software. They define the criteria, format, and interactions for sending and receiving responses by the software being used by the user.

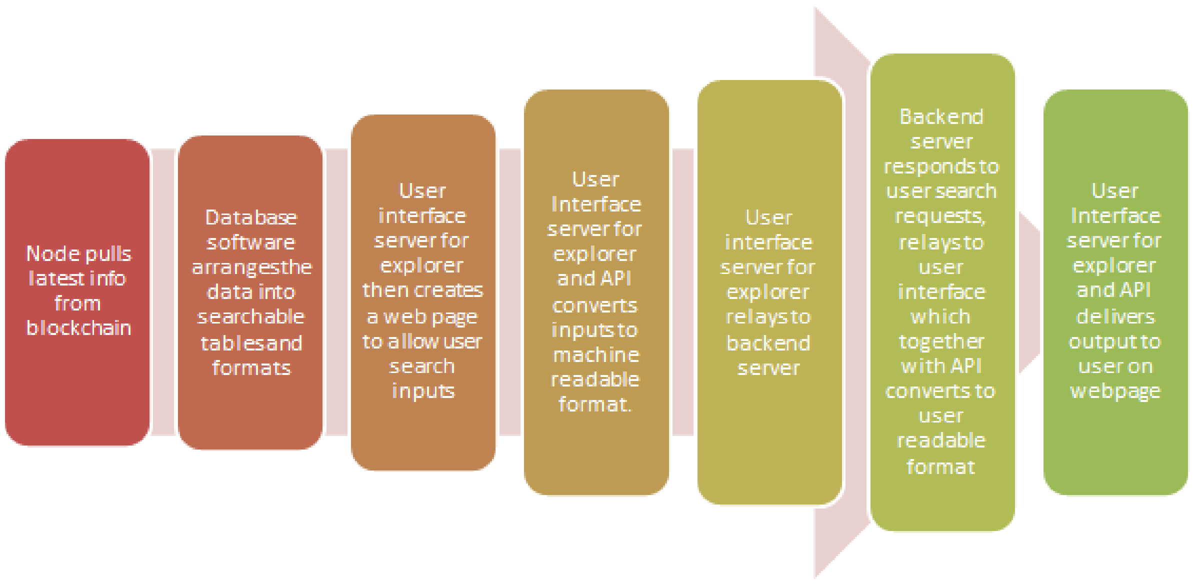

Node pulls latest info from blockchain
Database software arranges the data into searchable tables and formats
User interface server for explorer then creates a web page to allow user search inputs
User Interface server for explorer and API converts inputs to machine readable format.
User interface server for explorer relays to backend server
Backend server responds to user search requests, relays to user interface which together with API converts to user readable format
User Interface server for explorer and API delivers output to user on webpage

Fig. (24) - BC Explorer Process

6.8.2 How Does BC Explorer Work ?

Blockchain explorer work by using a database that holds all blockchain in a searchable format and tables. An explorer will, therefore, work with a node interface to first extract all the data in the blockchain. Once it derives the data, it then stores it in easily searchable tables.

It will gather the latest transactions and blocks and arrange them according to the defined searchable categories – for instance, transaction IDs.

Explorer also provides an interface to the user to allow them to search for the information. In terms of the technology, an explorer may use a relational database, SQL database, and an API.

We are already familiar with the fact that each blockchain comprises many distributed nodes. Each node that can directly read data on a blockchain, grabs details of the latest transaction and mined block and other data. This is then sent to the database, where the data is arranged in the form of searchable tables.

This makes the explorer fast to use. Most blockchains use 24 tables including block, address, transaction, etc. Each row has a unique ID or key, for instance, a unique identifier for addresses used on the blockchain. Others create a unique key.

The UI server for explorer then creates a web page that allows it to interact with a user by way of the latter input of searchable terms. It also provides an API to interface with other computers. These are sent to the backend server in a server readable format and the back-end server then responds to the user interface server for the search terms. The UI and API then sends the web pages as HTML to the browser to allow the reading of responses by the user.

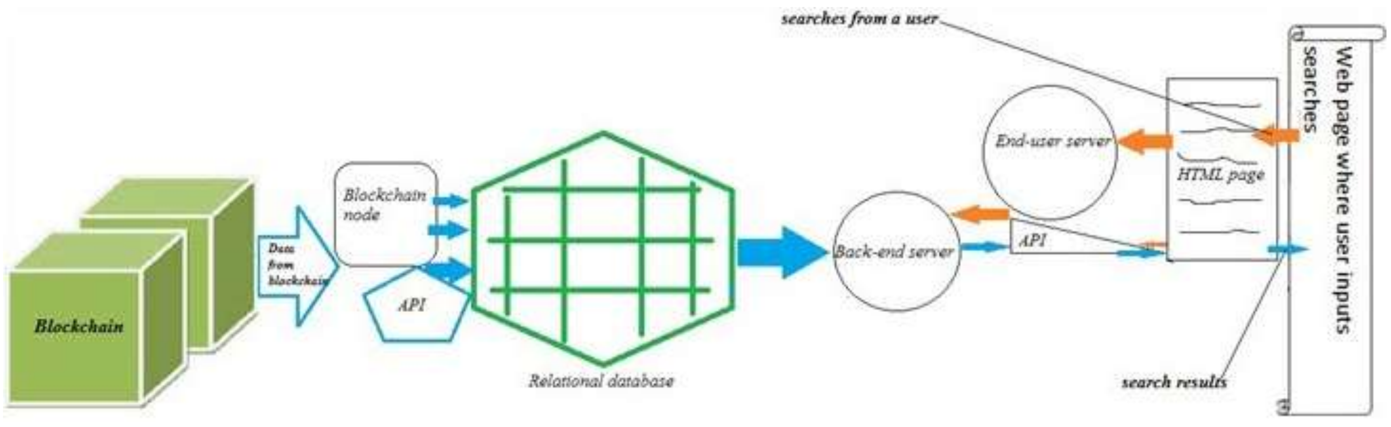


Fig. (25) - Blockchain Explorer Architecture

6.9 Backend Environment

6.9.1 Programming Language (JavaScript)

JavaScript often abbreviated JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. Over 97% of websites use JavaScript on the client side for web page behavior, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on users' devices.

JavaScript is a high-level, often just-in-time compiled language that conforms to the ECMAScript standard. It has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

The ECMAScript standard does not include any input/output (I/O), such as networking, storage, or graphics facilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O. JavaScript engines were originally used only in web browsers but are now core components of some servers and a variety of applications. The most popular runtime system for this usage is Node.js.

6.9.1.1 JavaScript Object Notation (JSON)

JSON is an open standard file format and a lightweight data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values). It is a common data format with diverse uses in electronic data interchange, including that of web applications with servers.

JSON is a language-independent data format. It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data. JSON filenames use the extension .json.

Douglas Crockford originally specified the JSON format in the early 2000s. He and Chip Morningstar sent the first JSON message in April 2001.

6.9.2 Runtime (Node.js)

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting - running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications.

The Node.js distributed development project was previously governed by the Node.js Foundation and has now merged with the JS Foundation to form the OpenJS Foundation, which is facilitated by the Linux Foundation's Collaborative Projects program.

Corporate users of Node.js software include GoDaddy, Groupon, IBM, LinkedIn, Microsoft, Netflix, PayPal, Rakuten, SAP, Walmart, Yahoo!, and Amazon Web Services.

Node.js allows the creation of Web servers and networking tools using JavaScript and a collection of "modules" that handle various core functionalities. Modules are provided for file system I/O, networking (DNS, HTTP, TCP, TLS/SSL, or UDP), binary data (buffers), cryptography functions, data streams, and other core functions. Node.js's modules use an API designed to reduce the complexity of writing server applications.

JavaScript is the only language that Node.js supports natively, but many compile-to-JS languages are available. As a result, Node.js applications can be written in CoffeeScript, Dart, TypeScript, ClojureScript and others.

Node.js is primarily used to build network programs such as Web servers. The most significant difference between Node.js and PHP is that most functions in PHP block until completion (commands execute only after previous ones finish), while Node.js functions are non-blocking (commands execute concurrently or parallel, and use callbacks to signal completion or failure).

6.9.2.1 Advantages

- Great performance! Node was designed to optimize throughput and scalability in web applications and is a good solution for many common web-development problems.
- Code is written in plain old JavaScript, which means that less time is spent dealing with context shift between languages when writing both client-side and server-side code.
- JavaScript is a relatively new programming language and benefits from improvements in language design when compared to other traditional web-server languages.
- Many other new and popular languages compile/convert into JavaScript.
- The node package manager (NPM) provides access to hundreds of thousands of reusable packages. It also has best-in-class dependency resolution and can also be used to automate most of the build toolchain.
- Node.js is portable. It is available on Microsoft Windows, macOS, Linux, FreeBSD, and OpenBSD. Furthermore, it is well-supported by many web hosting providers, that often provide specific infrastructure and documentation for hosting Node sites.
- It has a very active third-party ecosystem and developer community, with lots of help.

6.9.3 Package Manager (NPM)

Node Package Manager is a package manager for the JavaScript programming language maintained by npm, Inc., npm is the default package manager for the JavaScript runtime environment Node.js. It consists of a command line client, also called npm, and an online database of public and paid-for private packages, called the npm registry. The registry is accessed via the client, and the available packages can be browsed and searched via the npm website. The package manager and the registry are managed by npm, Inc.

npm is written entirely in JavaScript and was developed by Isaac Z. Schlueter as a result of having "seen module packaging done terribly" and with inspiration from other similar projects such as PEAR (PHP) and CPAN (Perl).

6.9.3.1 Description

npm is included as a recommended feature in the Node.js installer. npm consists of a command line client that interacts with a remote registry. It allows users to consume and distribute JavaScript modules that are available in the registry. Packages in the registry are in CommonJS format and include a metadata file in JSON format. Over 1.3 million packages are available in the main npm registry.

The registry does not have any vetting process for submission, which means that packages found there can potentially be low quality, insecure, or malicious. Instead, npm relies on user reports to take down packages if they violate policies by being low quality, insecure, or malicious. npm exposes statistics including number of downloads and number of depending packages to assist developers in judging the quality of packages.

In npm version 6, the audit feature was introduced to help developers identify and fix security vulnerabilities in installed packages. The source of security vulnerabilities were taken from reports found on the Node Security Platform (NSP) and has been integrated with npm since npm's acquisition of NSP.

6.9.3.2 Usage

npm can manage packages that are local dependencies of a particular project, as well as globally installed JavaScript tools. When used as a dependency manager for a local project, npm can install, in one command, all the dependencies of a project through the package.json file. In the package.json file, each dependency can specify a range of valid versions using the semantic versioning scheme, allowing developers to auto-update their packages while at the same time avoiding unwanted breaking changes.

npm also provides version-bumping tools for developers to tag their packages with a particular version. npm also provides the package-lock.json file which has the entry of the exact version used by the project after evaluating semantic versioning in package.json.

6.9.3.3 Registry

Internally npm relies on the NoSQL Couch DB to manage publicly available data.

6.9.4 Web App Framework (Express)

Express.js, or simply Express, is a back-end web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

The original author, TJ Holowaychuk, described it as a Sinatra-inspired server, meaning that it is relatively minimal with many features available as plugins. Express is the back-end component of popular development stacks like the MEAN, MERN or MEVN stack, together with the MongoDB database software and a JavaScript front-end framework or library.

6.9.4.1 Usages

1. Write handlers for requests with different HTTP verbs at different URL paths (routes).
2. Integrate with view rendering engines to generate responses by inserting data into templates.
3. Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
4. Add additional request processing at any point within the request handling pipeline.

6.9.4.2 Benefits

- Robust Routing
- Concentrate on High-Performance
- HTTP Helpers (Redirection and Caching)
- Content Negotiation

While Express itself is fairly minimalist, developers have created compatible middleware packages to address almost any web development problem. There are libraries to work with cookies, sessions, user logins, URL parameters, POST data, security headers, and many more.

Express provides methods to specify what function is called for a particular HTTP verb (GET, POST, SET, etc.) and URL pattern ("Route"), and methods to specify what template ("view") engine is used, where template files are located, and what template to use to render a response.

Express middleware can be used to add support for cookies, sessions, and users, getting POST/GET parameters, etc. Any database mechanism supported by Node can be used (Express does not define any database-related behavior).

6.9.5 Rest API

Application Programming Interface is a set of definitions and protocols for building and integrating application software. API is a mediator that helps communication and request fulfillment, it allows the interaction with a computer or a system to retrieve information or perform a function. It provides a way of organization to share resources and information while maintaining security, control, and authentication - determining who gets access to what.

REST stands for "Representational State Transfer" and refers to a software architectural style. It is based on six principles that describe how networked resources are defined and addressed on the web. When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP. JSON is the most generally popular file format to use because, it’s language-agnostic, as well as readable by both humans and machines.

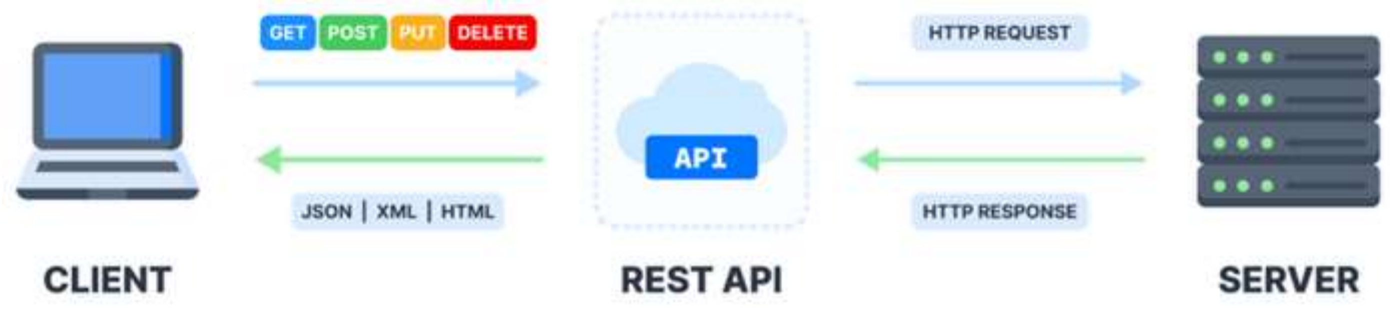


Fig. (26) - REST API Model

6.9.5.1 The Six Principles of REST

1. Client-Server Architecture

The principle behind the client-server architecture is the separation of problems. Dividing the user interface from data storage improves the portability of that interface across multiple platforms. It also has the advantage that different components can be developed independently.

2. Statelessness

Statelessness means that the communication between client and server always contains all the information needed to execute the request. There is no session state on the server, it is kept entirely on the client. If access to a resource requires authentication, the client must authenticate itself on each request.

3. Caching

The client, server, and any intermediate components can cache all resources to improve performance. The information can be classified as cacheable or non-cacheable.

4. Uniform Interface

Components of a RESTful API have to follow the same rules to communicate with each other. It also makes it easier to understand interactions between the various components of a system.

5. Layered System

Individual components cannot see beyond the immediate level they interact with. It means that a client that connects to an intermediate component such as a proxy does not know what is behind it. Thus, components can be easily exchanged or expanded independently of each other.

6. Code-on-Demand

Additional code can be downloaded to extend client functionality. However, this is optional because the client may not be able to download or execute this code.

6.9.5.2 Advantages of REST

The complete separation of the user interface from server and data storage offers some advantages for the development of an API. For example, it improves the portability of the interface to other types of platforms, increases project scalability and allows different components to be developed independently. Developers can easily migrate to other servers or make changes to the database, provided the data is sent correctly from each request. The separation thus increases overall flexibility in development.

A REST API is always independent of the type of platform or languages used, it adapts to the type of syntax or platform used. This provides great freedom when changing or testing new environments within a development. You can use PHP, Java, Python or Node.js servers with a REST API. Only responses to requests must be in the language used for information exchange, usually XML or JSON.

6.9.6 Database (SQLite)

SQLite is a C-language in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - a database can be freely copied between 32-bit and 64-bit systems or between big-endian and little-endian architectures. These features make SQLite a popular choice as an Application File Format. SQLite database files are a recommended storage format by the US Library of Congress.

SQLite is a compact library. With all features enabled, the library size can be less than 750Kbs, depending on the target platform and compiler optimization settings. 64-bit code is larger and some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger. There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments. Depending on how it is used, SQLite can be faster than direct filesystem I/O. SQLite is carefully tested prior to every release and has a reputation for being very reliable. It responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or failures.

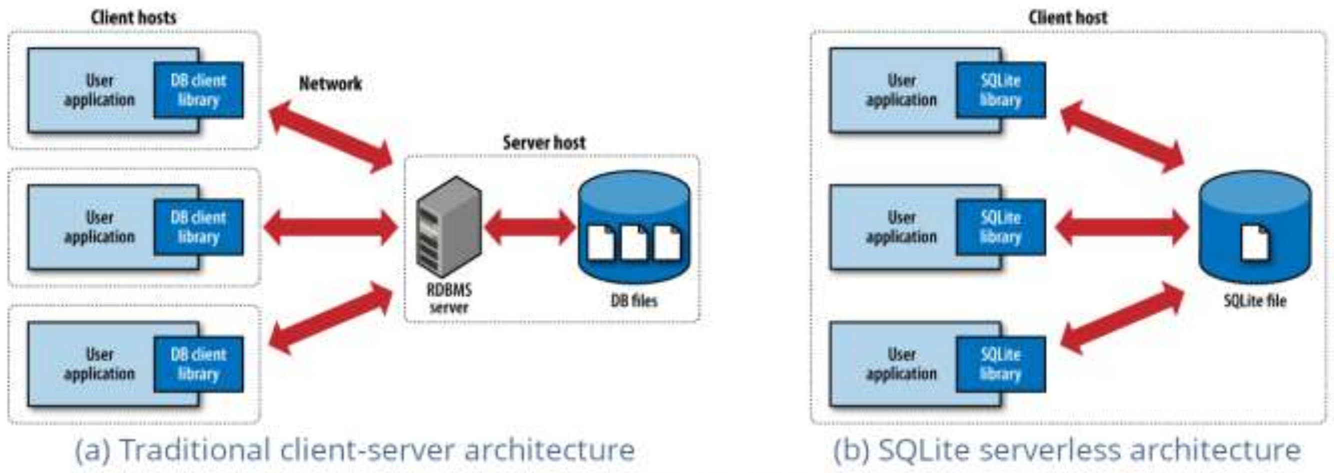


Fig. (27) - SQLite

6.9.7 Backend Operations

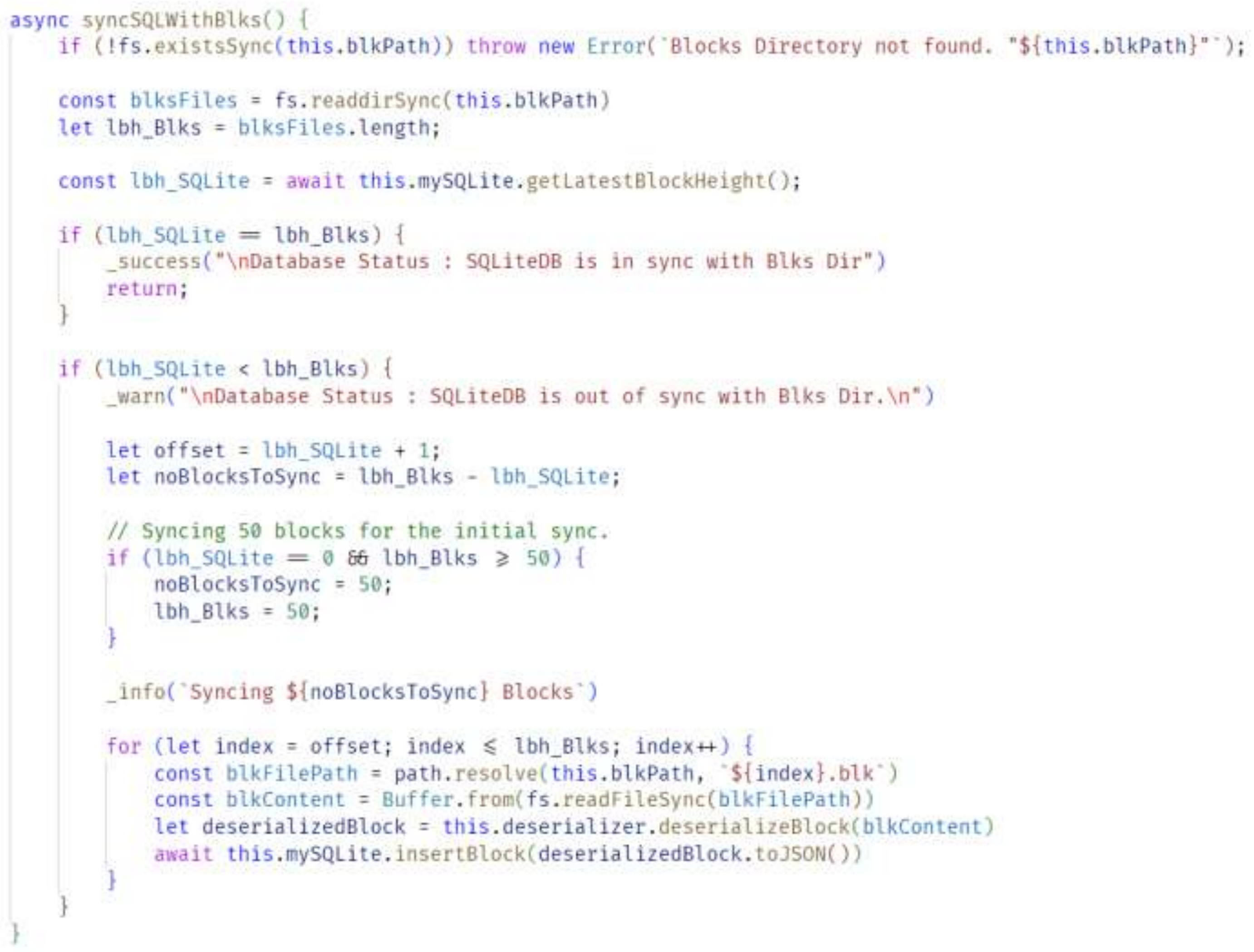
6.9.7.1 Extracting the Blockchain from a Node

The network's node and the backend server can't be running and accessing LevelDB at the same time because it can only be held open by a single process at a time. So, blocks had to be saved as .blk files for the backend to access and read them from relational database.

Steps to be performed to extract/insert blocks to the SQLite database:

- Extract the latest mined blocks from .blks files directory.
- Deserialize each block using block.proto file structure.
- Insert each block along with its transactions in SQLite database.

These steps are performed by the "syncSQLWithBlks" method which gets the height of the latest block inserted in SQLite blocks table and check if it's in or out of sync with blks files.



```
async syncSQLWithBlks() {
    if (!fs.existsSync(this.blkPath)) throw new Error(`Blocks Directory not found. "${this.blkPath}"`);

    const blksFiles = fs.readdirSync(this.blkPath)
    let lbh_Blks = blksFiles.length;

    const lbh_SQLite = await this.mySQLite.getLatestBlockHeight();

    if (lbh_SQLite == lbh_Blks) {
        _success("\nDatabase Status : SQLiteDB is in sync with Blks Dir")
        return;
    }

    if (lbh_SQLite < lbh_Blks) {
        _warn("\nDatabase Status : SQLiteDB is out of sync with Blks Dir.\n")

        let offset = lbh_SQLite + 1;
        let noBlocksToSync = lbh_Blks - lbh_SQLite;

        // Syncing 50 blocks for the initial sync.
        if (lbh_SQLite == 0 && lbh_Blks ≥ 50) {
            noBlocksToSync = 50;
            lbh_Blks = 50;
        }

        _info(`Syncing ${noBlocksToSync} Blocks`)

        for (let index = offset; index ≤ lbh_Blks; index++) {
            const blkFilePath = path.resolve(this.blkPath, `${index}.blk`)
            const blkContent = Buffer.from(fs.readFileSync(blkFilePath))
            let deserializedBlock = this.deserializer.deserializeBlock(blkContent)
            await this.mySQLite.insertBlock(deserializedBlock.toJSON())
        }
    }
}
```

6.9.7.2 Storing Blockchain on SQLite

The blockchain is stored and arranged in a structured tables having the following schema.

```sql
CREATE TABLE  Blocks (
    Height          INTEGER UNIQUE,
    PrevHash        TEXT,
    Timestamp       INTEGER,
    MerkleRoot      TEXT,
    Hash            TEXT PRIMARY KEY,
    Miner           TEXT,
    nTx             INTEGER
);

CREATE TABLE Transactions (
    Date            INTEGER,
    Hash            TEXT PRIMARY KEY,
    Elector         TEXT,
    Elected         TEXT,
    Confirmations   INTEGER NOT NULL,
    BlockHeight     INTEGER,
    FOREIGN KEY (BlockHeight) REFERENCES Blocks (Height)
);
```

Each block is inserted separately from its transactions with one-to-many relationship. This way, it’s quick and easy to query blocks and transaction using different query parameters such as block height, block hash and transaction hash.

```
// GET /blocks/block-height/50
{
    "result": {
      "Height": 50,
      "PrevHash": "065081F83496B0AD857B7497FEC3DE28C7DCD42784DAF836AEB0E9EF74677991",
      "Timestamp": 1654347604553,
      "MerkleRoot": "C84314555A23E055387AA9C623312C37CDB451117D59C03B1EBC281DA904B14E",
      "Hash": "AFDA6438A9537F09AB81E7CAA85F259F2CCD3D85BE3873F5EA57836F3D52EA9A",
      "Miner": "FEB60777C2472EE9F49EC3B33289D69AE626EAC88CF999B0AE3C021B7F02D797",
      "nTx": 10
    }
  }
```

```
// GET /transactions/tx-hash/7711CB8C8AAC55D3714D54645E95080698BB7CCAA1F9075E1380601C7EAAA559
{
    "result": {
      "Date": 1654347603960,
      "Hash": "7711CB8C8AAC55D3714D54645E95080698BB7CCAA1F9075E1380601C7EAAA559",
      "Elector": "FEB60777C2472EE9F49EC3B33289D69AE626EAC88CF999B0AE3C021B7F02D797",
      "Elected": "A962DED29707CCA0011DDEB4A664BF0167054654CF9D34772CBD447A06CF6424",
      "BlockHeight": 50
    }
  }
```

6.9.7.3 Counting Votes

Each candidate is inserted in Candidates table and is identified by his ID number.

```
CREATE TABLE Candidates (
    ID                  TEXT PRIMARY KEY,
    Name                Text NOT NULL,
    NoVotes             INTEGER NOT NULL,
    Color               INTEGER NOT NULL
);
```

Each voting machine is inserted in VMachines table and is identified by its ID, along with the governorate where it's placed.

```
CREATE TABLE VMachines (
    ID                  TEXT PRIMARY KEY,
    Lat                 INTEGER NOT NULL,
    Lng                 INTEGER NOT NULL,
    Address             TEXT NOT NULL,
    Governorate         Text NOT NULL
);
```

There is also a list of governorates where the running elections cover, each governorate has a list of IDs for the voting machines.

```
CREATE TABLE Governorates (
    ID                  INTEGER PRIMARY KEY AUTOINCREMENT,
    ArabicName          Text NOT NULL,
    EnglishName         Text NOT NULL,
    IDsOfVMs            Text,
    Votes               Text NOT NULL,
    Color               INTEGER NOT NULL
);
```

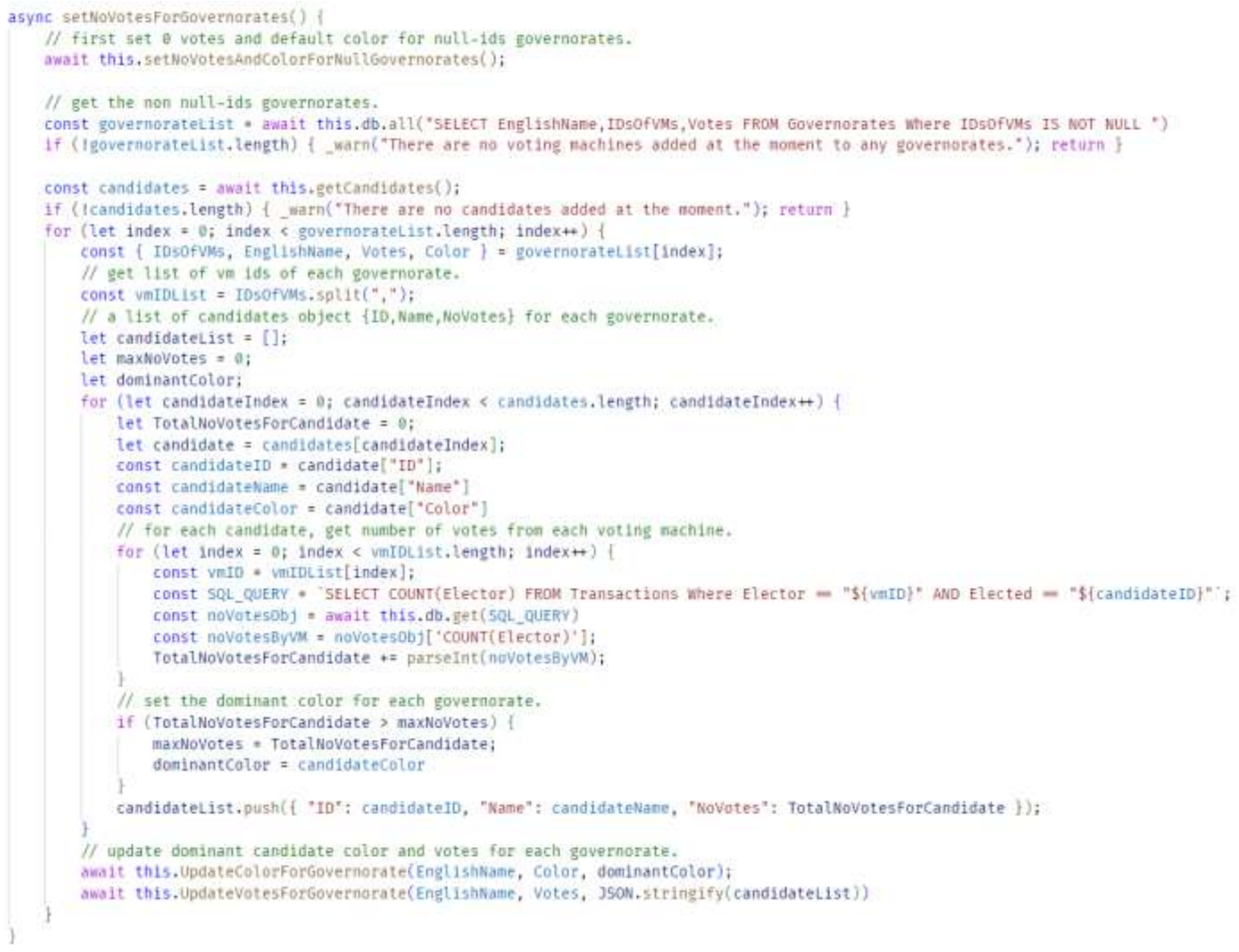
The key here is that the voting machine is voting on behalf of the actual voter. Meaning that, the elector's ID for each transaction corresponds to the ID of the voting machine.

So, we can easily count each candidate's votes per governorate by looping through the list of voting machines that each governorate has.

Steps to be performed to count each governorate’s votes:

- For each entry of a new voting machine, add its ID to its governorate list of machines.
- Get the list of governorates.
- Get the list of candidates and loop through the list of IDs of the voting machines of each governorate.
- For each candidate, get the number of votes per governorate by getting the number of transactions where "Elector" is equal to the ID of the voting machine and "Elected" is equal to the ID of the candidate.

These steps are performed by the “setNoVotesForGovernorates” method.



```
async setNoVotesForGovernorates() {
    // first set 0 votes and default color for null-ids governorates.
    await this.setNoVotesAndColorForNullGovernorates();

    // get the non null-ids governorates.
    const governorateList = await this.db.all("SELECT EnglishName,IDsOfVMs,Votes FROM Governorates Where IDsOfVMs IS NOT NULL ")
    if (!governorateList.length) { _warn("There are no voting machines added at the moment to any governorates."); return }

    const candidates = await this.getCandidates();
    if (!candidates.length) { _warn("There are no candidates added at the moment."); return }
    for (let index = 0; index < governorateList.length; index++) {
        const { IDsOfVMs, EnglishName, Votes, Color } = governorateList[index];
        // get list of vm ids of each governorate.
        const vmIDList = IDsOfVMs.split(",");
        // a list of candidates object {ID,Name,NoVotes} for each governorate.
        let candidateList = [];
        let maxNoVotes = 0;
        let dominantColor;
        for (let candidateIndex = 0; candidateIndex < candidates.length; candidateIndex++) {
            let TotalNoVotesForCandidate = 0;
            let candidate = candidates[candidateIndex];
            const candidateID = candidate["ID"];
            const candidateName = candidate["Name"]
            const candidateColor = candidate["Color"]
            // for each candidate, get number of votes from each voting machine.
            for (let index = 0; index < vmIDList.length; index++) {
                const vmID = vmIDList[index];
                const SQL_QUERY = `SELECT COUNT(Elector) FROM Transactions Where Elector == "${vmID}" AND Elected == "${candidateID}"`;
                const noVotesObj = await this.db.get(SQL_QUERY)
                const noVotesByVM = noVotesObj['COUNT(Elector)'];
                TotalNoVotesForCandidate += parseInt(noVotesByVM);
            }
            // set the dominant color for each governorate.
            if (TotalNoVotesForCandidate > maxNoVotes) {
                maxNoVotes = TotalNoVotesForCandidate;
                dominantColor = candidateColor
            }
            candidateList.push({ "ID": candidateID, "Name": candidateName, "NoVotes": TotalNoVotesForCandidate });
        }
        // update dominant candidate color and votes for each governorate.
        await this.UpdateColorForGovernorate(EnglishName, Color, dominantColor);
        await this.UpdateVotesForGovernorate(EnglishName, Votes, JSON.stringify(candidateList))
    }
}
```

6.9.8 Load Balancing

6.9.8.1 Using a Reverse Proxy (Nginx) as an HTTP Load Balancer

Load balancing across multiple application instances is a commonly used technique for optimizing resource utilization, maximizing throughput, reducing latency, and ensuring a fault-tolerant configuration. It is possible to use Nginx as a very efficient HTTP load balancer to distribute traffic to several application servers and to improve performance, scalability, and reliability of web applications with Nginx.

6.9.8.2 Load Balancing Methods

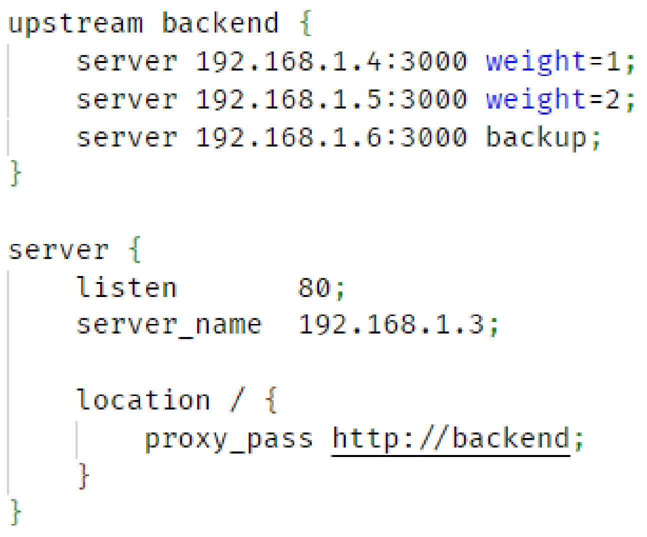
The following load balancing mechanisms (or methods) are supported in Nginx:

• round-robin: Requests to the application servers are distributed in a round-robin fashion.

• least-connected: Next request is assigned to the server with the least no. of active connections.

• ip-hash: A hash-function is used to determine which server should be selected for the next request (based on the client's IP address).

Nginx HTTP module is used to balance the load over HTTP servers using the upstream block:



```
upstream backend {
    server 192.168.1.4:3000 weight=1;
    server 192.168.1.5:3000 weight=2;
    server 192.168.1.6:3000 backup;
}

server {
    listen       80;
    server_name  192.168.1.3;

    location / {
        proxy_pass http://backend;
    }
}
```

In this configuration we have multiple running instances of the explorer's backend server, and the load is distributed over them. As the load increases, another instance of the server can be brought online and added. Also, this configuration balances load across the first two servers on port 80, and defines the last one as a backup, which is used when the primary two are unavailable. The weight parameter instructs Nginx to pass twice as many requests to the second server, and the weight parameter defaults to 1.

6.10 Deep Learning Models

6.10.1 Libraries

6.10.1.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source image processing and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for developers to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDAand OpenCL interfaces are being actively developed. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

6.10.1.2 TensorFlow

TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++.

TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. It supports production prediction at scale, with the same models used for training.

TensorFlow allows developers to create dataflow graphs - structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection is a multidimensional data array, or a tensor.

TensorFlow applications can be run on any target device that's convenient whether It's a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

6.10.1.3 Dlib

Dlib is a modern general-purpose cross-platform software library written in the programming language C++ containing machine learning algorithms and tools for creating complex deep learning models to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open-source license (released under Boost Software License) allows us to use it in any application, free of charge.

Since development began in 2002, Dlib has grown to include a wide variety of tools. As of 2016, it contains software components for dealing with networking, threads, graphical user interfaces, data structures, linear algebra, machine learning, image processing, data mining, XML and text parsing, numerical optimization, Bayesian networks, and many other tasks. In recent years, much of the development has been focused on creating a broad set of statistical machine learning tools and in 2009 Dlib was published in the Journal of Machine Learning Research. Since then it has been used in a wide range of domains.

6.10.2 Speech Synthesizer Model (TTS)

Speech synthesis is the artificial production of human speech. A computer system used for this purpose is called a speech synthesizer and can be implemented in software and hardware products. A text-to-speech (TTS) system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech.

The quality of a speech synthesizer is judged by its similarity to the human voice and by its ability to be understood clearly. An intelligible text-to-speech program allows people with visual impairments or reading disabilities to listen to written words on computers.

TTS model was developed to automatically read any generated messages to aid users through the voting process. The model was trained with the help of TensorFlow library using an RNN.

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed or undirected graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. Recurrent neural networks are theoretically Turing complete and can run arbitrary programs to process arbitrary sequences of inputs.

6.10.3 Optical Character Recognition Model (OCR)

Optical character recognition (OCR) is the electronic conversion of images of typed, handwritten, or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo or from subtitle text superimposed on an image.

Widely used as a form of data entry from printed paper data records – whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation – it is a common method of digitizing printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence, and computer vision.

OCR mode was developed for automatic data entry to facilitate & accelerate the voting process and reduce any possible errors resulting from incorrect insertions or users' lack of experience. The model was built with the help of OpenCV library and using a CNN. Various custom adjustments were made on the model to enhance the recognition of Arabic letters and numbers.

A convolutional neural network (CNN) is a class of artificial neural network (ANN), most commonly applied to analyze visual imagery. CNNs are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivariant responses known as feature maps. Counter-intuitively, most convolutional neural networks are only equivariant, as opposed to invariant, to translation. They have applications in image and video recognition, recommender systems, image classification, image segmentation, image analysis, natural language processing, brain–computer interfaces, & financial time series.

6.10.3.1 The Process

1. The user is asked to enter his ID card for scanning, guided through graphical instructions and generated voice commands using the text to speech model.
2. The card gets detected using its dimensions and the object's contour.
3. The image gets segmented, and the card is cropped to isolate it from background.
4. The card is checked to be an authentic national ID.

   4.1 Objects are detected via their coordinates for extraction.

   4.2 Check the front for the presence of a unique identifier.

   4.3 Check the back for the presence of the barcode.
5. ID number gets verified through a prestored database.
6. ID number is utilized for age verification and transaction processing.
7. Expiry date is checked to ensure card validity.
8. Job title is checked to confirm eligibility.
9. A copy of the front is saved for later processing.

6.10.4 Face Recognition Model

Face recognition technology is capable of matching human faces from digital images or video frames against a database of faces, typically employed to authenticate users through ID verification services, by pinpointing and measuring facial features from the given image.

Because computerized facial recognition involves the measurement of a human's physiological characteristics, facial recognition systems are categorized as **biometrics**. Although the accuracy of facial recognition systems as a biometric technology is lower than iris recognition and fingerprint recognition, it is adopted due to its contactless process.

Face recognition model was developed to effectively verify voter’s identity eliminating any human errors leading to any frauds. The model was built with the help of Dlib library and trained by a CNN reinforced by augmented data and fine-tuning achieving > 95 % accuracy.

- Data augmentation are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data.

- Fine-tuning is applying transfer learning by re-training model on a specific custom dataset (ID Photos) to adjust the weights very precisely in order to fit with certain observations.

6.10.4.1 The Process

1. Open a live stream from the camera.
2. Ask the voter to remove any worn head accessories.
3. Perform face detection and ensure that the voter is alone.
4. Capture multiple frames from the live feed.
5. Initiate Anti-Spoofing algorithm to prevent impersonation.

   5.1 Ask the user to change poses by following specific set of moves.

   5.2 Check color depth and facial expressions.
6. Crop the face from the captured frames.
7. Extract face features into a set of feature vectors.
8. Match the results against the saved ID photo.

6.11 Embedding Python in .NET

The voting process involves extracting the voter's info from his ID card using OCR then verifying voter's identity biometrically via face recognition, and their functionalities are written in Python. So, to empower our voting application and the blockchain client - which are written in C# - with these functionalities, we needed a wrapper to integrate python code in C#.

A wrapper in programming is a package that allows codes written in a certain programming language to be embedded into programs written in a different language to reach a fully integrated environment with no compatibility issues.

6.11.1Types of Wrappers

- Compiling Wrapper: which compiles python code into Intermediate Language (IL), that runs on the .Net runtime. (IronPython)

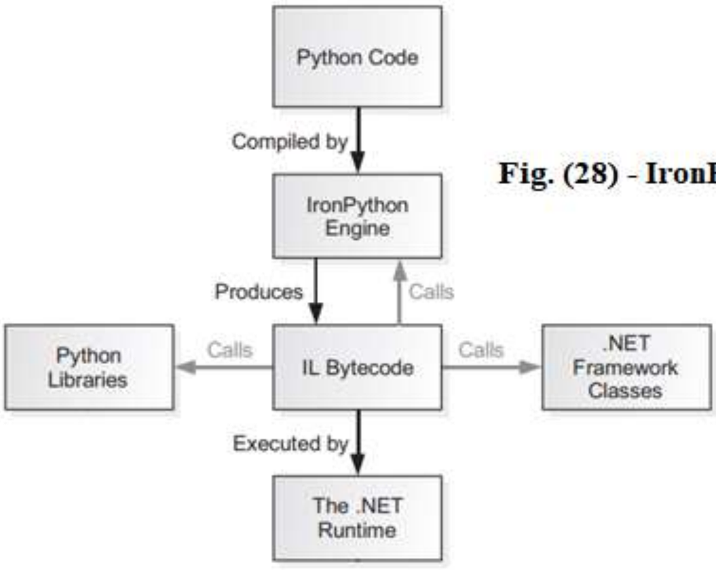


Fig. (28) - IronPython

- Embedding Wrapper: which embeds and integrates Python Interpreter in .Net. (Python.Net)

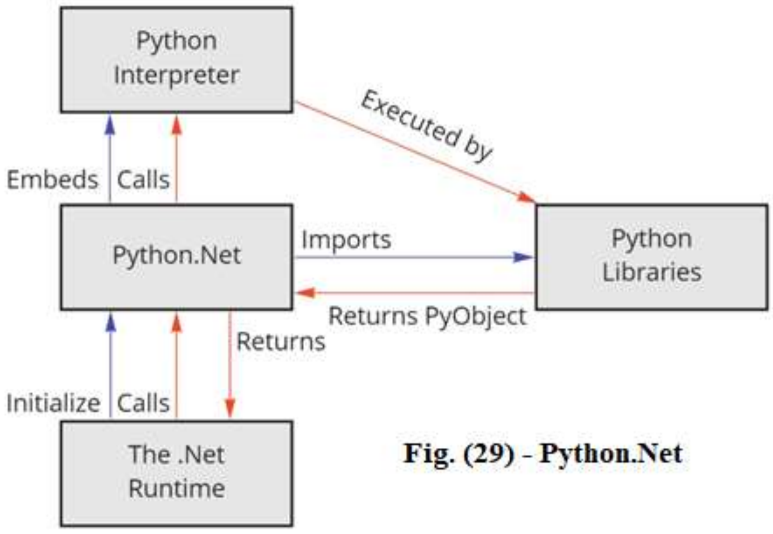


Fig. (29) - Python.Net

A compiling wrapper couldn't be used as it can only compile pure python modules & it doesn't support C extension modules (.pyd) such as OpenCV, so we used Python.Net, an embedding wrapper that maintains native execution speed for native & non-native Python code.

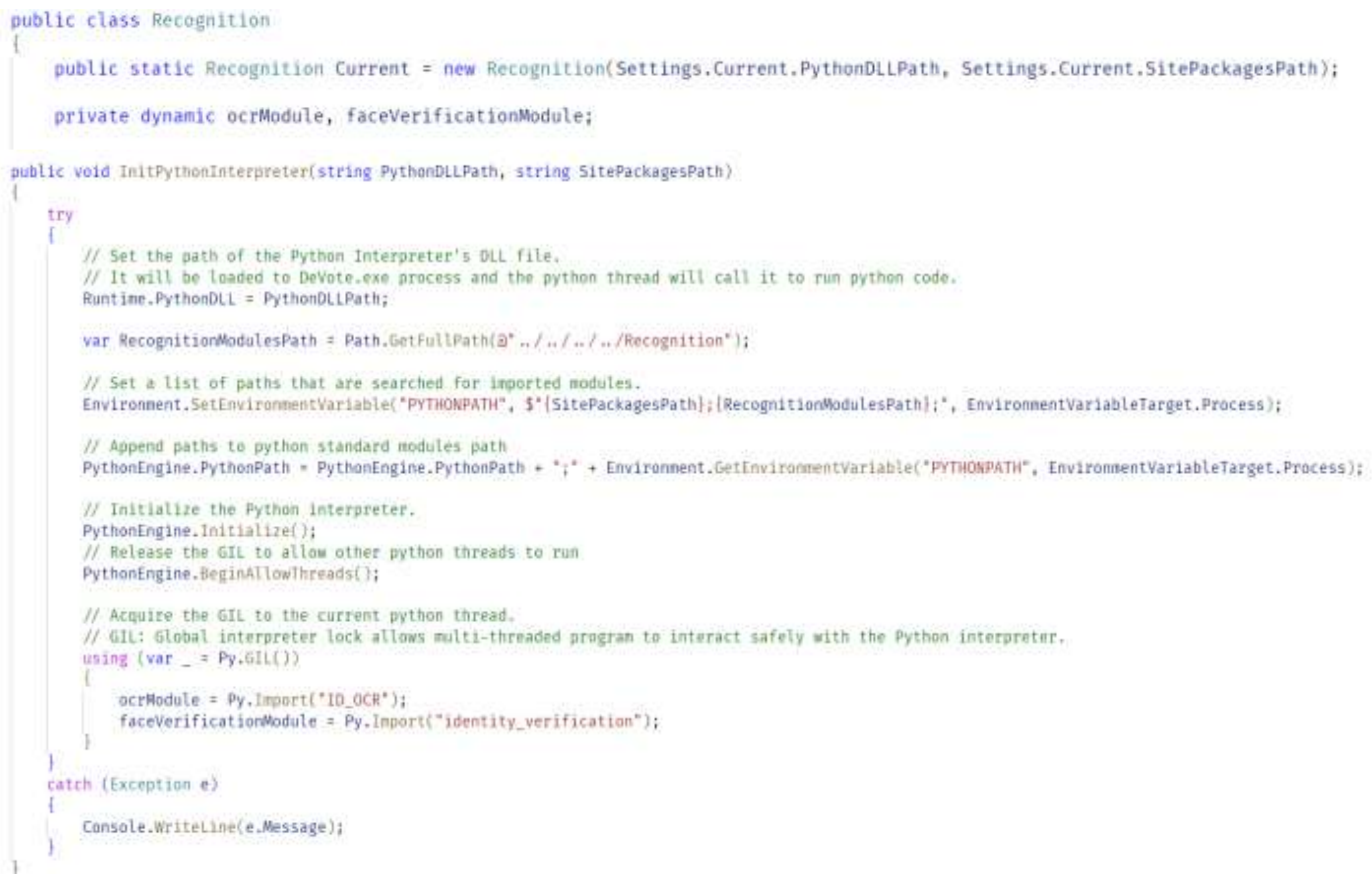
6.11.2 Python.Net

Python.NET is a package that gives Python programmers nearly seamless integration with the .NET Common Language Runtime (CLR) and provides a powerful application scripting tool for .NET developers. It allows Python code to interact with the CLR, and may also be used to embed Python into a .NET application.

6.11.3 Embedding Python Interpreter

To embed Python in our application we needed to:

- Install Python interpreter.
- Reference Python interpreter.dll to Python.Runtime.dll of wrapper.
- Reference paths of the OCR, face recognition and dependencies modules to PythonEngine.PythonPath of wrapper.
- Call PythonEngine.Initialize() to initialize Python interpreter.
- Call PythonEngine.ImportModule() to import OCR and face recognition modules.

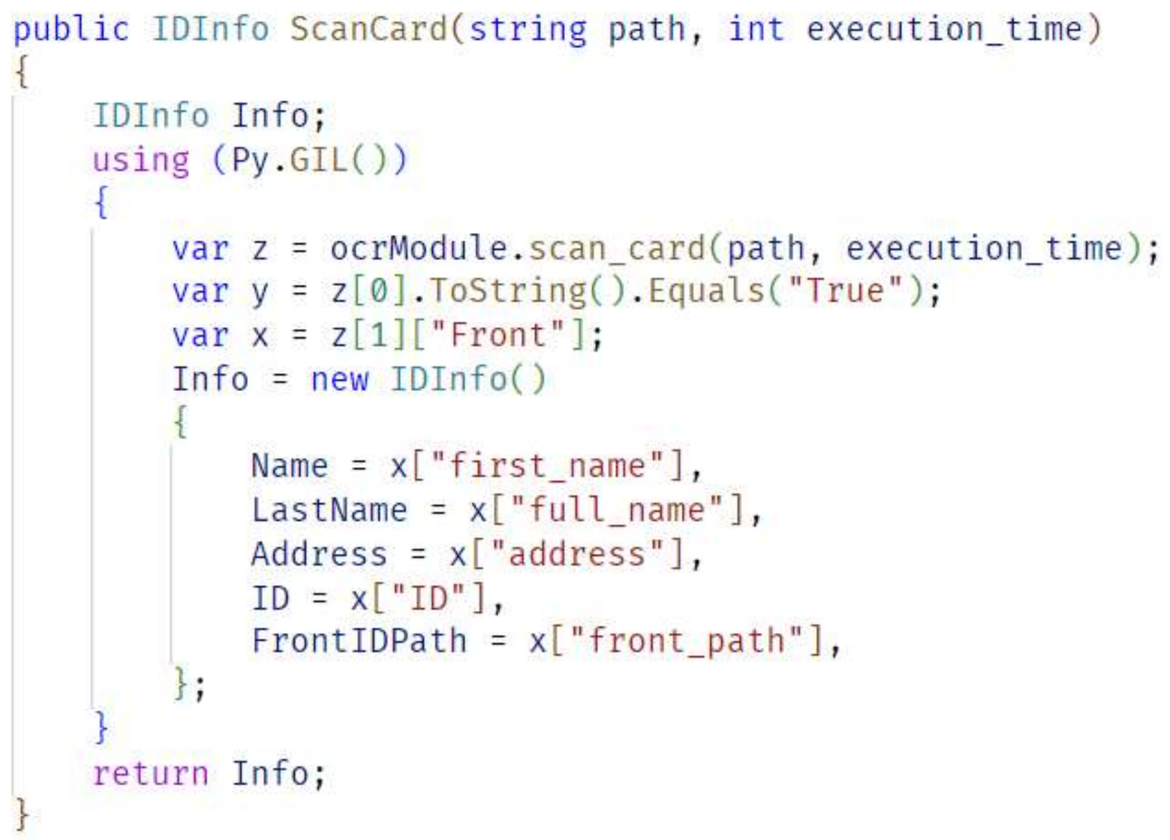


```
public class Recognition
{
    public static Recognition Current = new Recognition(Settings.Current.PythonDLLPath, Settings.Current.SitePackagesPath);

    private dynamic ocrModule, faceVerificationModule;

public void InitPythonInterpreter(string PythonDLLPath, string SitePackagesPath)
{
    try
    {
        // Set the path of the Python Interpreter's DLL file.
        // It will be loaded to DeVote.exe process and the python thread will call it to run python code.
        Runtime.PythonDLL = PythonDLLPath;

        var RecognitionModulesPath = Path.GetFullPath(@"../../../../Recognition");

        // Set a list of paths that are searched for imported modules.
        Environment.SetEnvironmentVariable("PYTHONPATH", $"{SitePackagesPath};{RecognitionModulesPath};", EnvironmentVariableTarget.Process);

        // Append paths to python standard modules path
        PythonEngine.PythonPath = PythonEngine.PythonPath + ";" + Environment.GetEnvironmentVariable("PYTHONPATH", EnvironmentVariableTarget.Process);

        // Initialize the Python interpreter.
        PythonEngine.Initialize();
        // Release the GIL to allow other python threads to run
        PythonEngine.BeginAllowThreads();

        // Acquire the GIL to the current python thread.
        // GIL: Global interpreter lock allows multi-threaded program to interact safely with the Python interpreter.
        using (var _ = Py.GIL())
        {
            ocrModule = Py.Import("ID_OCR");
            faceVerificationModule = Py.Import("identity_verification");
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

6.11.4 Calling Python Functionalities through the Wrapper

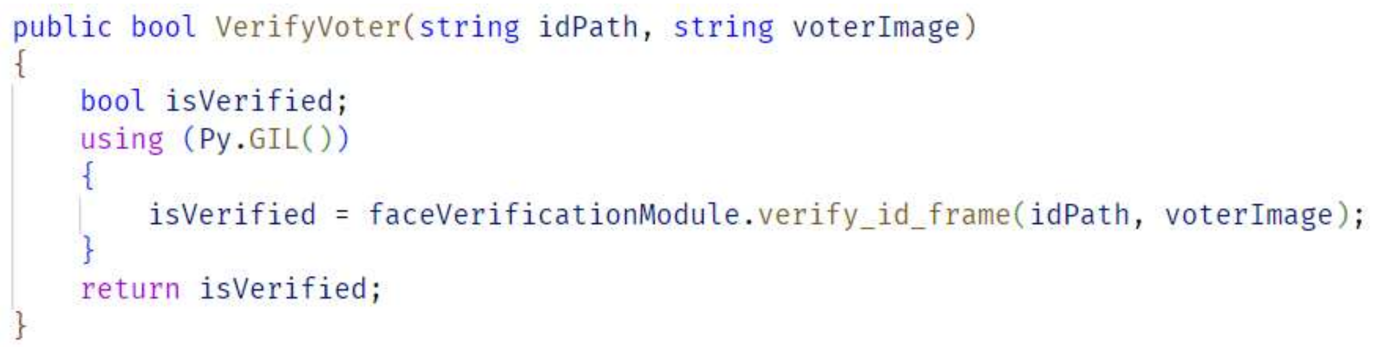
Now our voting application and blockchain client can call the installed python interpreter to execute any functions inside the OCR and face recognition modules.

We can extract the voter's information by calling the **ScanCard** method which wraps the **scan_card** function which is written in Python inside the OCR module.



```
public IDInfo ScanCard(string path, int execution_time)
{
    IDInfo Info;
    using (Py.GIL())
    {
        var z = ocrModule.scan_card(path, execution_time);
        var y = z[0].ToString().Equals("True");
        var x = z[1]["Front"];
        Info = new IDInfo()
        {
            Name = x["first_name"],
            LastName = x["full_name"],
            Address = x["address"],
            ID = x["ID"],
            FrontIDPath = x["front_path"],
        };
    }
    return Info;
}
```

We can verify the voter biometrically by calling **VerifyVoter** method which wraps **verift_id_frame** function which is written in Python inside the face recognition module.



```
public bool VerifyVoter(string idPath, string voterImage)
{
    bool isVerified;
    using (Py.GIL())
    {
        isVerified = faceVerificationModule.verify_id_frame(idPath, voterImage);
    }
    return isVerified;
}
```

6.12 User Interface (UI)

6.12.1 E-Voting App (Desktop)

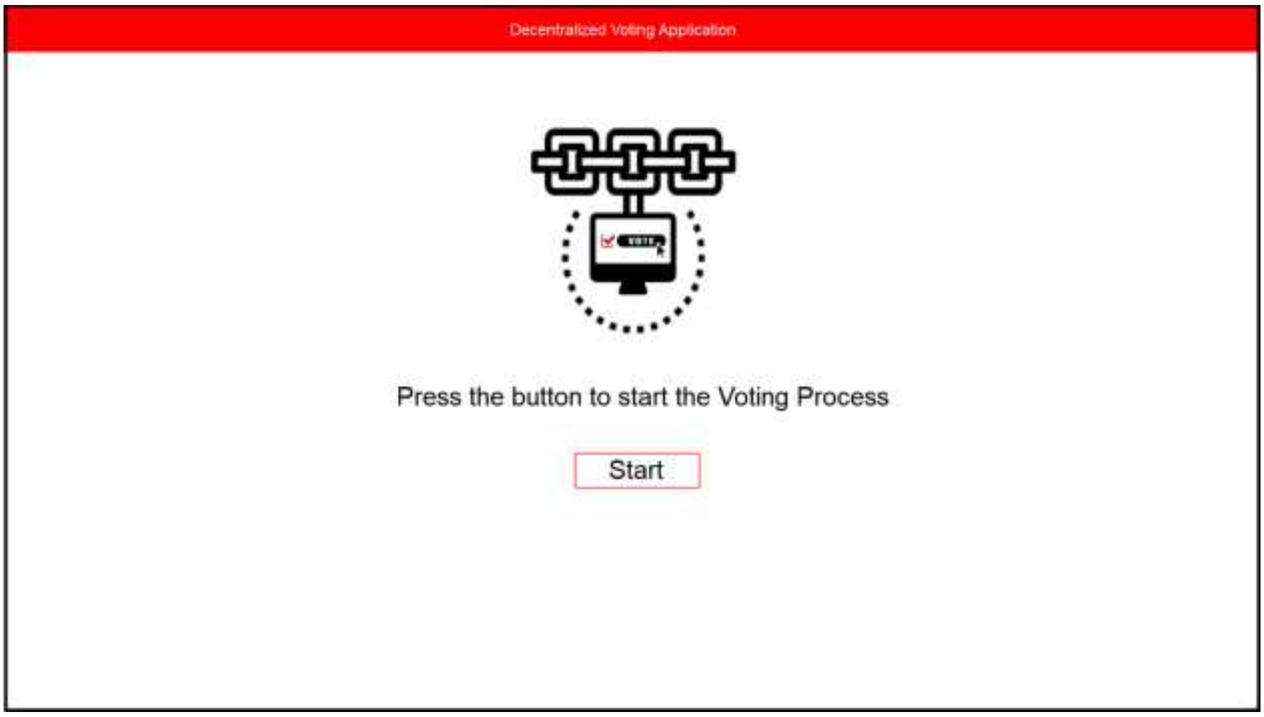

Decentralized Voting Application
VOTE
Press the button to start the Voting Process
Start

Fig. (30) - Welcome Screen

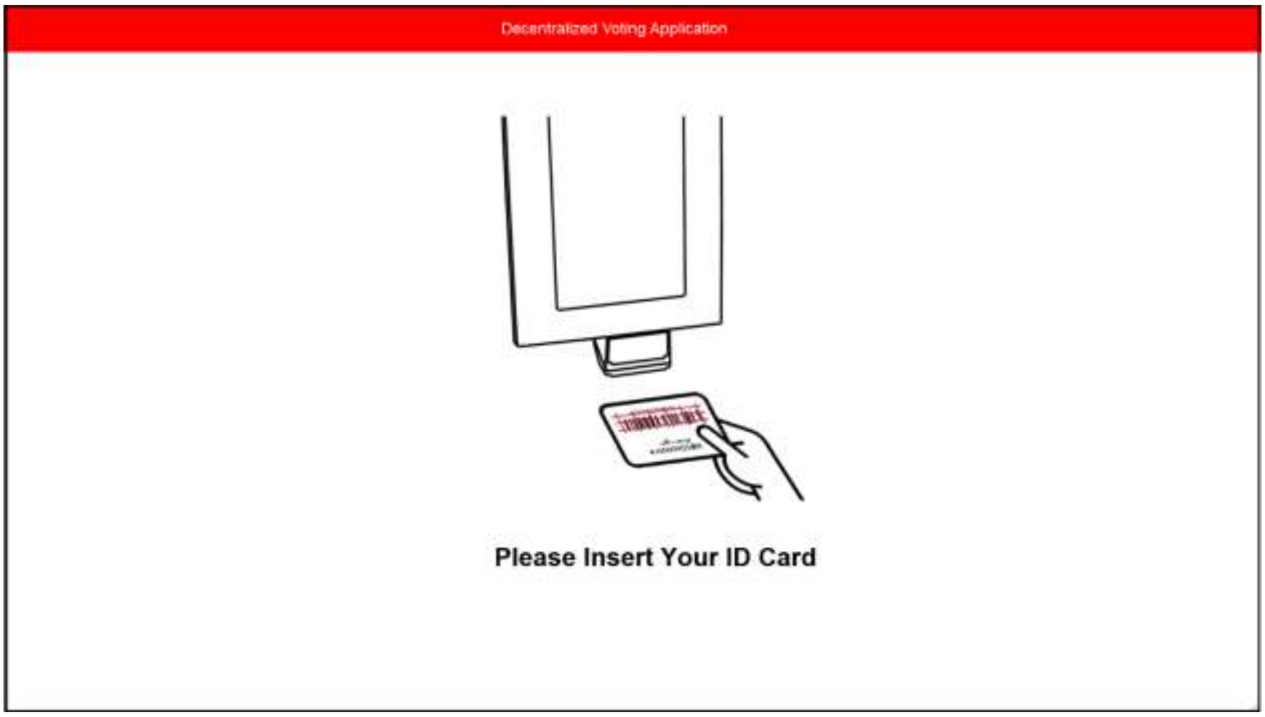
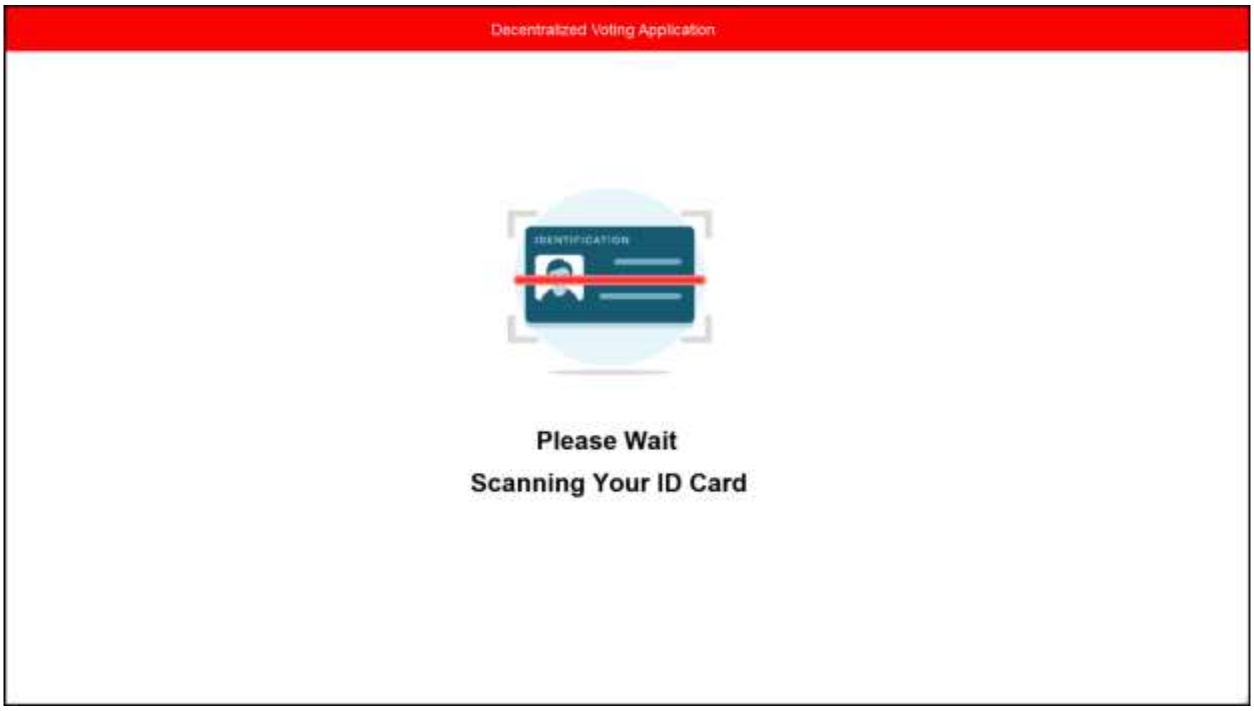


Fig. (31) - ID Card Insertion

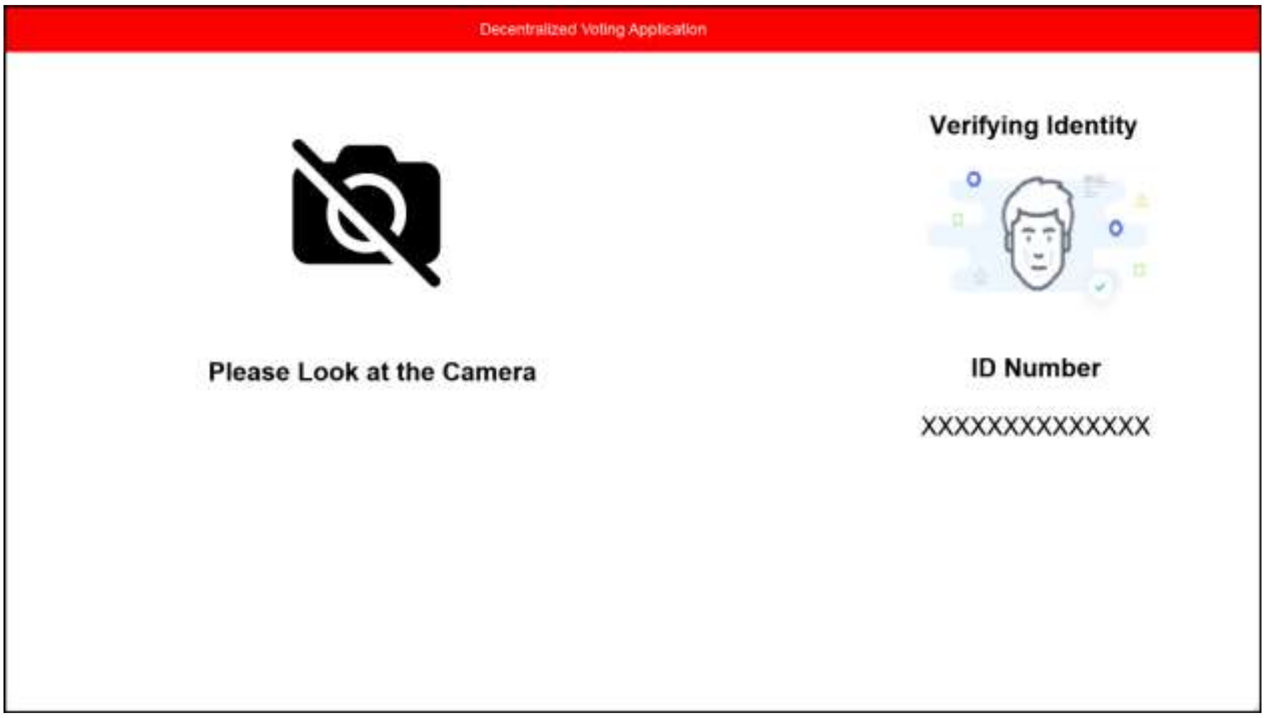
In Fig. (31) the voter is prompted to insert his ID card into the machine’s scanner.



Please Wait
Scanning Your ID Card

Fig. (32) - ID Card Scanning

In Fig. (32) - the machine extracts the voter's information from his ID card using OCR.



Please Look at the Camera

Verifying Identity

ID Number

XXXXXXXXXXXXXX

Fig. (33) - Identity Verification

In Fig. (33) the machine verifies the voter's identity using face recognition.

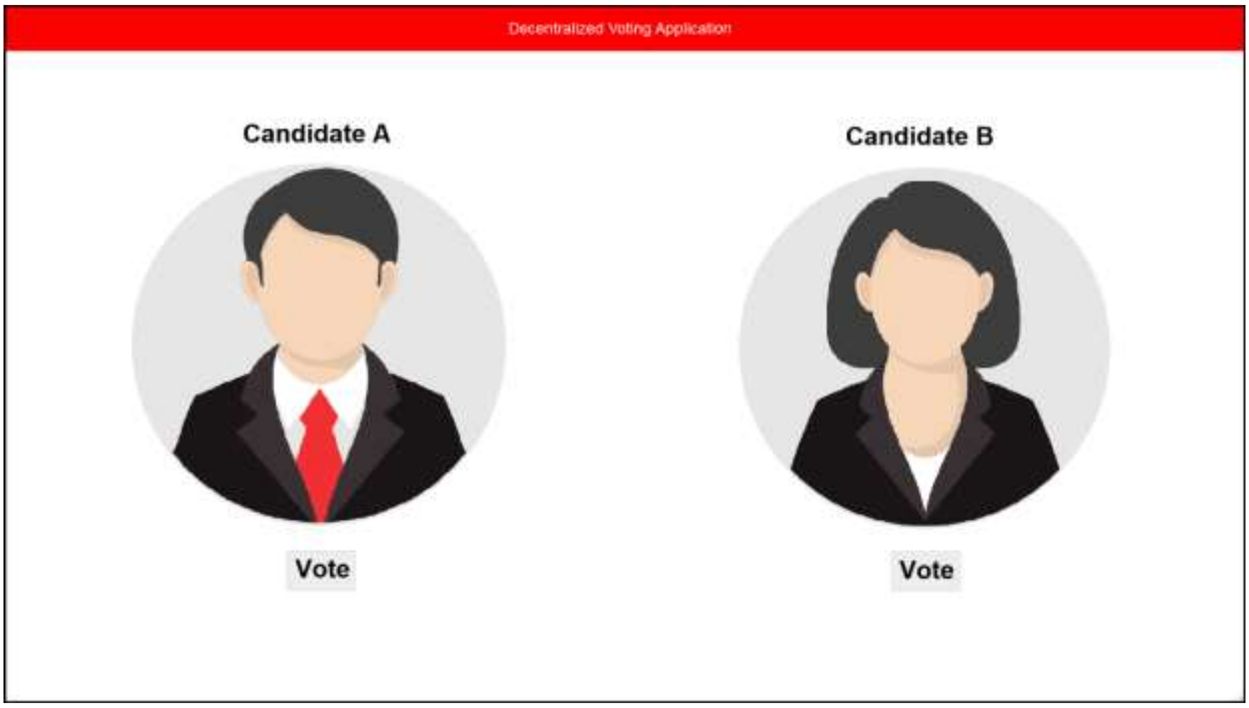
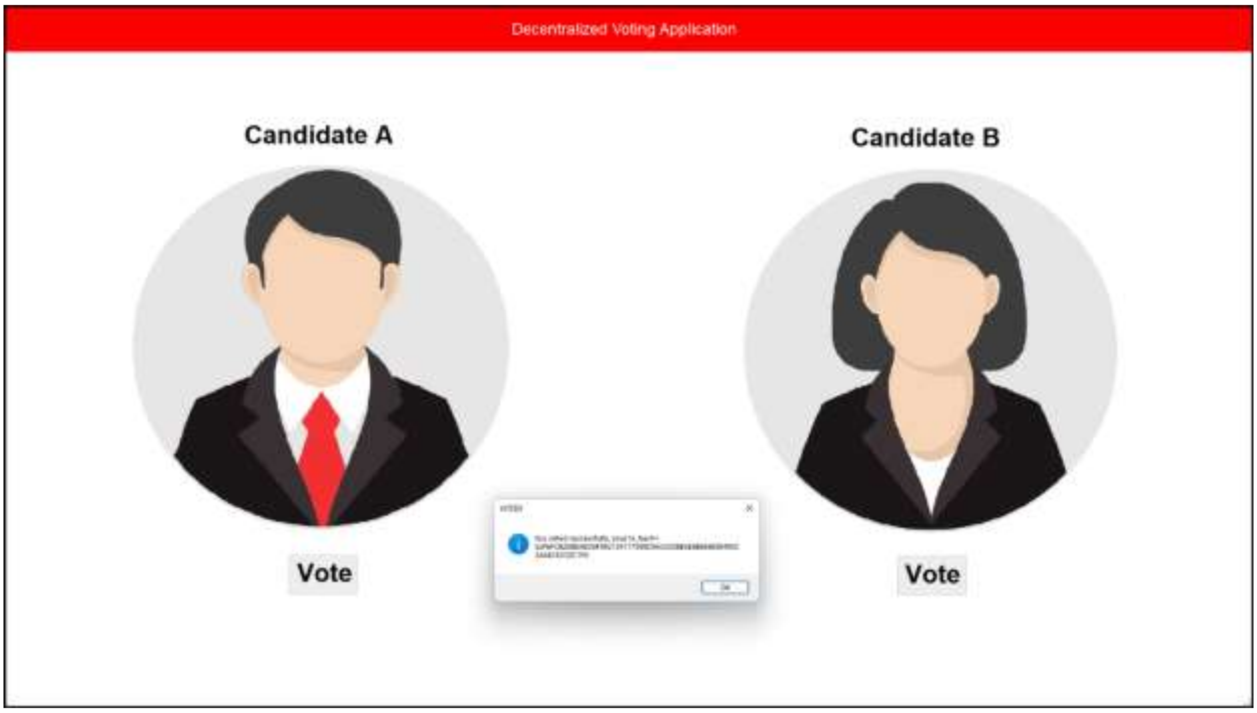


Fig. (34) - Vote Submission

In Fig. (34) the voter is prompted to make his selection from the available candidates.



You voted successfully, your tx hash= [illegible]

Fig. (35) - Vote Confirmation

In Fig. (34) the system confirms successful vote registration and provides the transaction ID.

6.12.2 BC Explorer App (Flutter)

A Flutter web application was developed as a supporting platform to host the blockchain explorer and to guide voters through the elections with instructions, news and dynamic live statistical results.

Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, macOS, Windows, Google Fuchsia, and the web from a single codebase.

Flutter was utilized due to its cross-platform nature that provides a consistent performance over every device and for being lightweight granting a smooth and responsive experience.

6.12.2.1 Used Packages

1. SyncFusion Flutter Maps

   Flutter Maps is a powerful data visualization widget that displays statistical information for a geographical area. Its rich feature set that includes tile rendering from OpenStreetMap, Azure Maps, Bing Maps, Google Maps, and other tile providers. It has marker support and shape layers with features like selection, legends, labels, markers, tooltips, bubbles, color mapping.

2. TimeAgo

   TimeAgo is a dart library that converts a date into a humanized text. Instead of showing a date 2020-12-12 18:30 with TimeAgo you can display something like "now", "an hour ago", "~1y".

3. Animated Text Kit

   A flutter package which contains a collection of some cool and awesome text animations.

4. Google Maps Flutter

   A Flutter plugin that provides a Google Maps widget.

5. SyncFusion Flutter Charts

   The Flutter Charts package is a data visualization library written natively in Dart for creating beautiful, animated, and high-performance charts, which are used to craft high-quality mobile app user interfaces using Flutter.

6. Shimmer

A package that provides an easy way to add shimmer effect in a Flutter project.

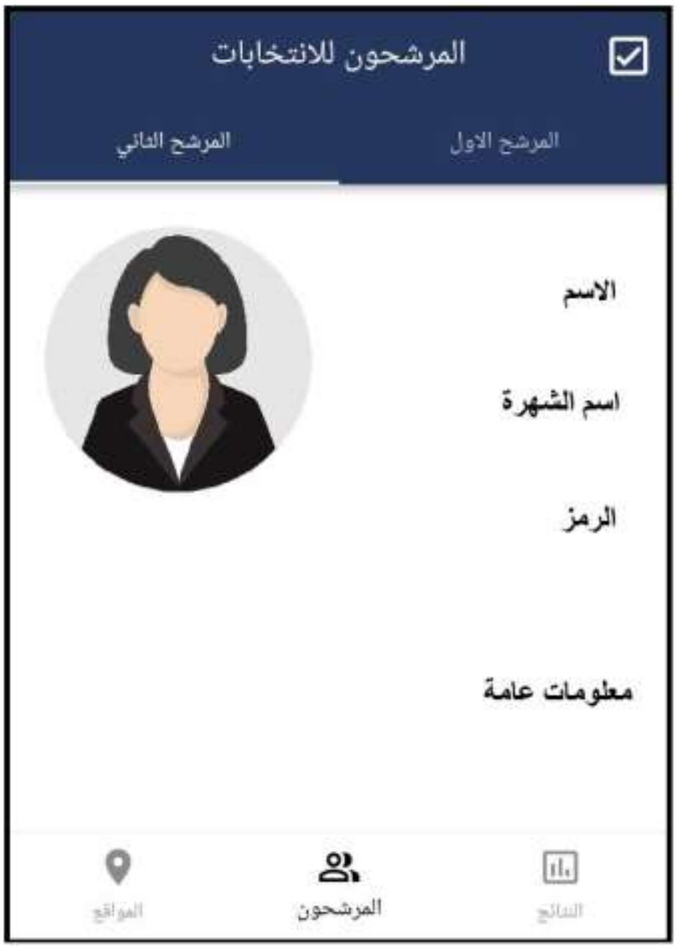
7. HTTP

This package contains a set of high-level functions and classes that make it easy to consume HTTP resources. It's multi-platform, and supports mobile, desktop, and the browser.

8. Linked Scroll Controller

This package provides a way to set up a set of scrollable widgets whose scrolling is synchronized. The set can be stable across the lifetime of the containing screen, or can change dynamically (for example, a vertically scrolling ListView.builder() whose items are Scrollables that scroll horizontally in unison).

6.12.2.2 Mobile App



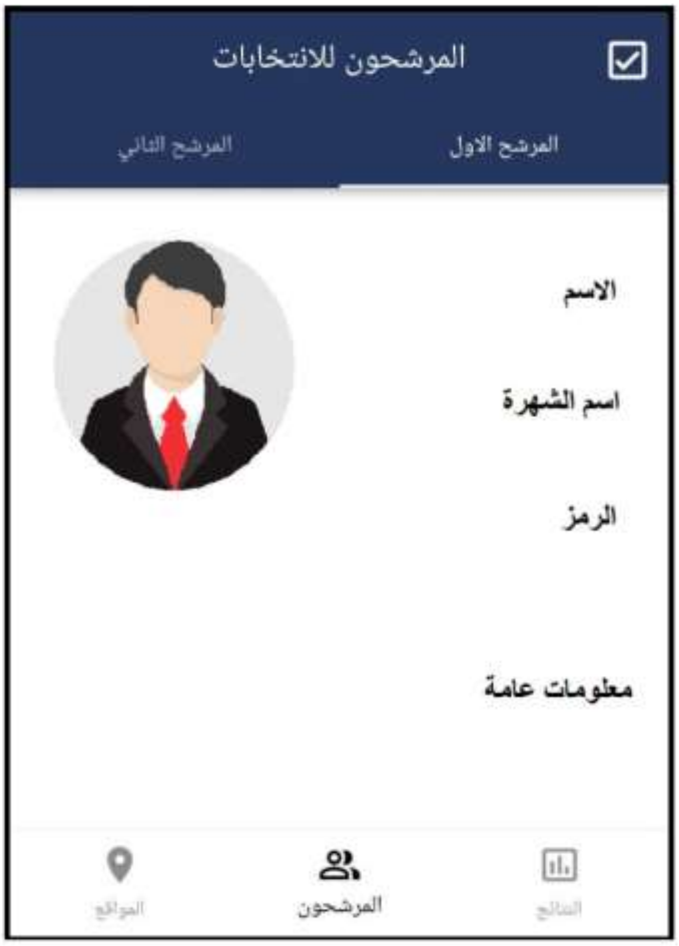


Fig. (36) - Candidates Information

In Fig. (36) the app displays information about each candidate running for the elections.

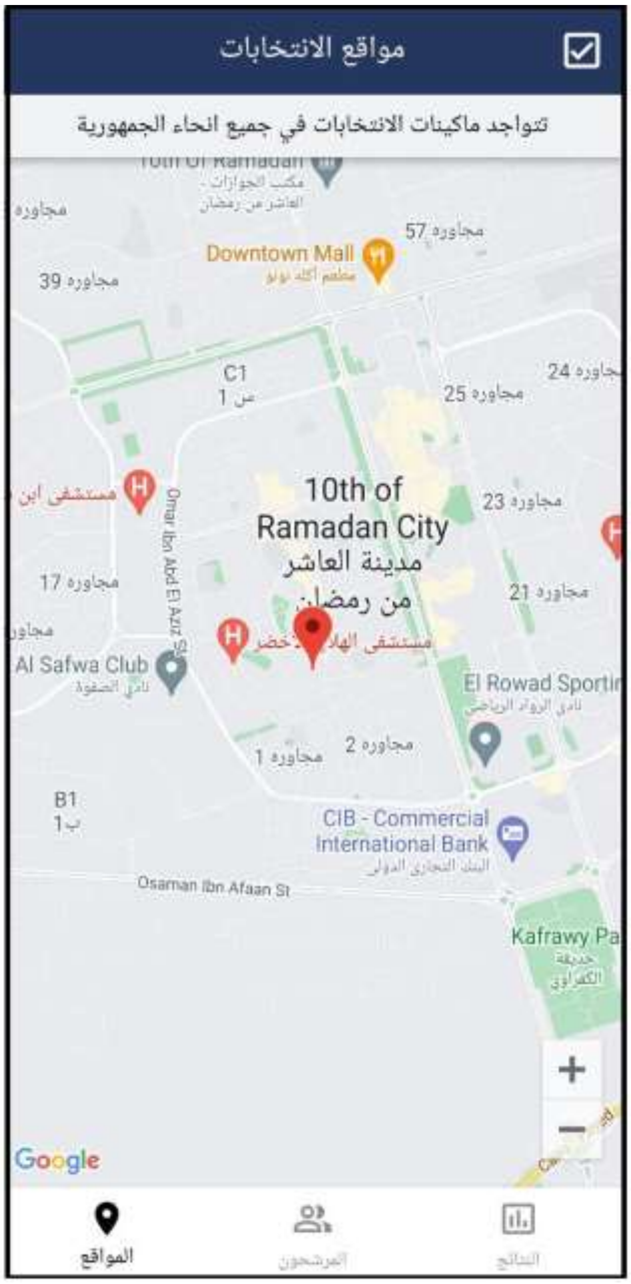


Fig. (37) - Voting Booths Locations

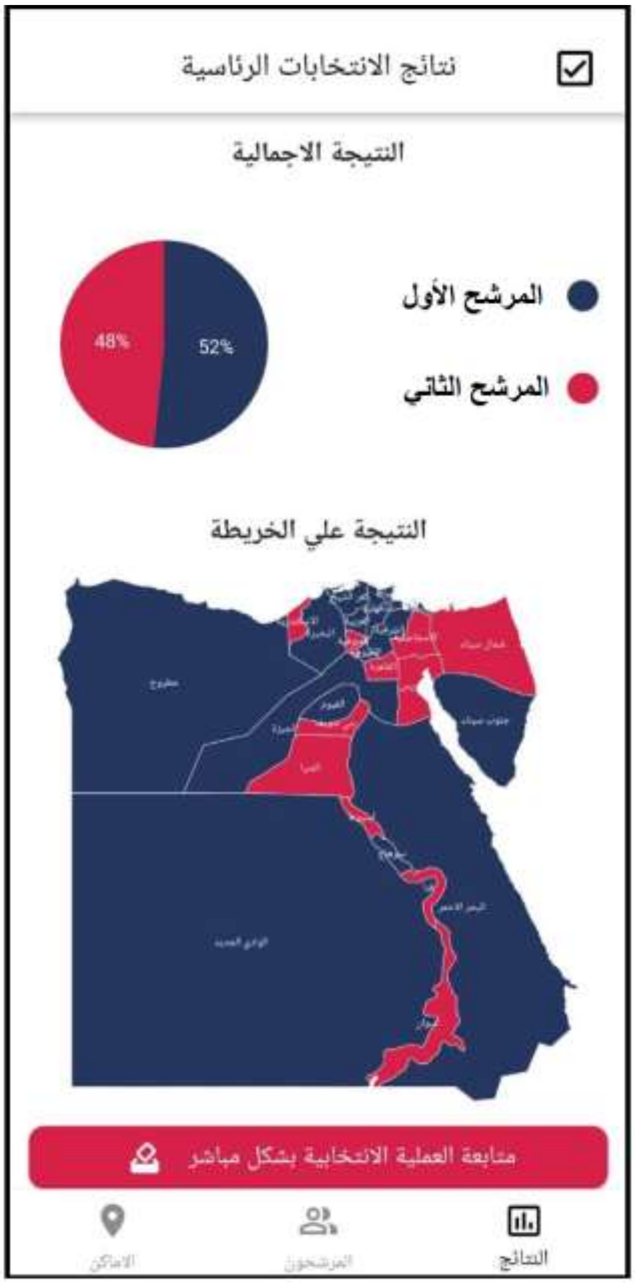
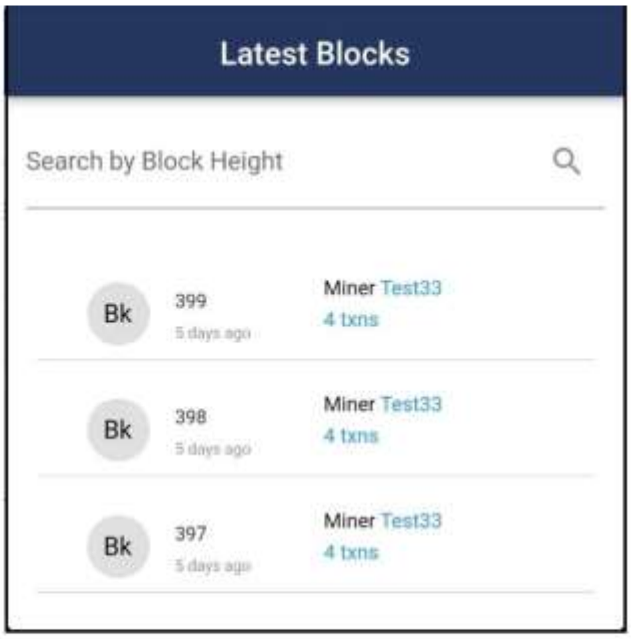


Fig. (38) - Elections Results

In Fig. (37) the app displays the available locations of the voting booths on a geographic map, guiding the voter by showing the ones that are closest to him depending on his location at the time of use.

In Fig. (38) the app displays live results of the running elections. The map is dived into governorates each one is represented with the color of the winning candidate in this governorate. The chart shows the total statistical result based on the wining ration of each candidate. At the bottom of the page there’s a button that allows the user to access the Blockchain Explorer to see more details.



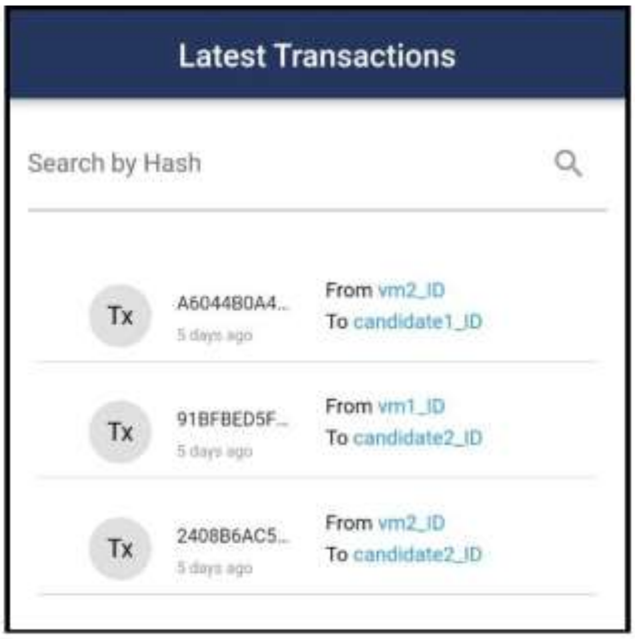
Latest Blocks

Search by Block Height

Bk 399 5 days ago Miner Test33 4 txns

Bk 398 5 days ago Miner Test33 4 txns

Bk 397 5 days ago Miner Test33 4 txns



Latest Transactions

Search by Hash

Tx A6044B0A4... 5 days ago From vm2_ID To candidate1_ID

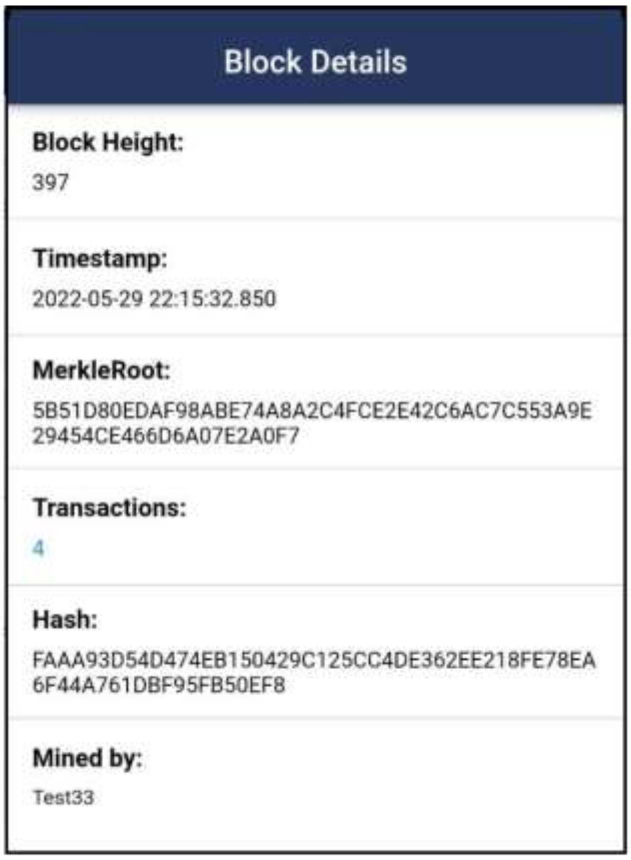
Tx 91BFBED5F... 5 days ago From vm1_ID To candidate2_ID

Tx 2408B6AC5... 5 days ago From vm2_ID To candidate2_ID

Fig. (39) - Blocks List

Fig. (40) - Transactions List

In Fig. (39) and (40) the app displays the list of blocks and the transactions in the corresponding block providing the user with real-time updates. The user can search for his own transaction using Hash.



Block Details

Block Height:
397

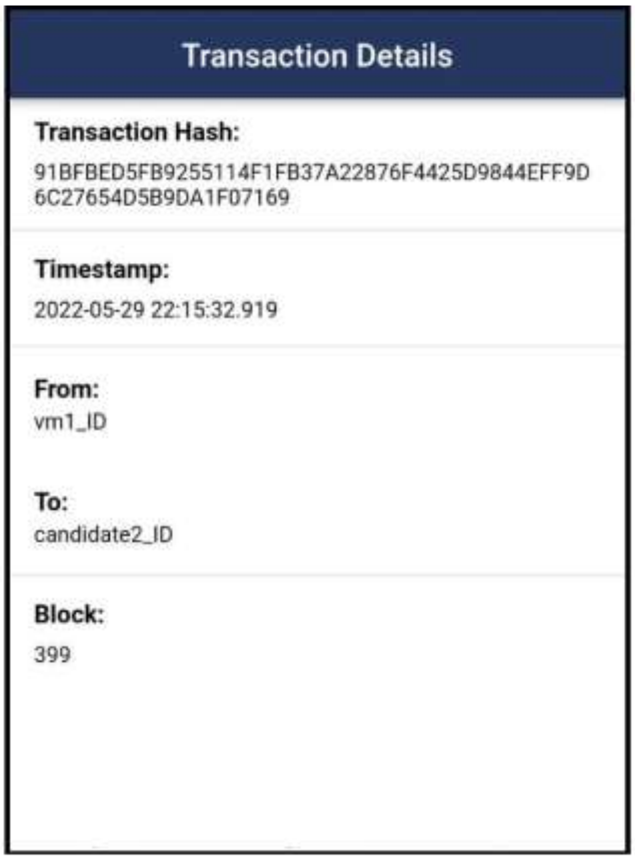
Timestamp:
2022-05-29 22:15:32.850

MerkleRoot:
5B51D80EDAF98ABE74A8A2C4FCE2E42C6AC7C553A9E29454CE466D6A07E2A0F7

Transactions:
4

Hash:
FAAA93D54D474EB150429C125CC4DE362EE218FE78EA6F44A761DBF95FB50EF8

Mined by:
Test33



Transaction Details

Transaction Hash:
91BFBED5FB9255114F1FB37A22876F4425D9844EFF9D6C27654D5B9DA1F07169

Timestamp:
2022-05-29 22:15:32.919

From:
vm1_ID

To:
candidate2_ID

Block:
399

Fig. (41) - Block Details

Fig. (42) - Transaction Detail

In Fig. (41) and (42) the app displays the details of the selected block or the transaction, where the user can validate his vote using his transaction hash to ensure transparency.

Conclusion

Predicting the future of electronic voting is not an easy task, especially when considering a new technology that is the object of active research and further development. Obviously public permissionless blockchains have a limited commercial or governmental use due to their questionable integrity & lack of privacy. On the other hand, private permissioned blockchains are those that attract the most attention from the business side or the governmental authorities.

To sum up, in this document we proposed the use of a permissioned blockchain as the base of a secure decentralized e-voting system supported by deep learning authentication techniques. We found that even if this is still far from being a mainstream approach, due to voting itself being an inherently difficult problem to tackle and the accepted solutions that are currently being used do not solely rely on technical procedures, to settle the conflicting security requirements, instead are built on layers of trust in institutions, but the use of the blockchain establishes a foundation for a technical solution that could revolutionize the e-voting systems.

Agreement is reached on that electronic voting is such a critical application and should be independent and E2E verifiable. Most experts propose using a paper trail for verifiability since the cryptographic solutions are somewhat immature, but there are huge gains that can’t be denied from using the blockchain technology to reinforce the security of the voting systems. The blockchain solves a major concern of the voting problem - how to secure the votes during the elections and ensure its immutability, however, this is only the tip of the iceberg. In order to be able to convince the authorities to endorse the system, as well as the public, a voting protocol must satisfy strict security requirements, for which the blockchain does help a lot.

Electronic voting is still in its infancy and should be approached with caution as it can help attackers cheat at a large scale with ease. However, as all human activities are going to be conducted electronically in the long run, it is inevitable that voting follows suit. Cryptography and other techniques will evolve to support the security requirements. At the same time the public should gradually get accustomed to the new processes by participating in small scale elections where the stakes are not as high as in national governmental elections. This means that the introduction of electronic voting should be bottom up and gradual, providing security guarantees that are theoretically rigorous, yet easily conceivable by the average voter.