

Bucharest University of Economic Studies

The Faculty of Economic Cybernetics, Statistics and Informatics

IT&C Security Master

DISSERTATION THESIS

Coordinator

Ph. D. Cristian TOMA

Graduate

Mihail Rareș NEDELCU

Bucharest

2025



Bucharest University of Economic Studies

The Faculty of Economic Cybernetics, Statistics and Informatics

IT&C Security Master

E-VOTING APP BASED ON BLOCKCHAIN

DISSERTATION THESIS

Coordinator

Ph. D. Cristian TOMA

Graduate

Mihail Rareș NEDELCU

Bucharest

2025

Statement regarding the originality of the content

I hereby declare that the results presented in this paper are entirely the result of my own creation unless reference is made to the results of the other authors. I confirm that any material used from other sources (magazines, books, articles, and Internet sites) is clearly referenced in the paper and is indicated in the bibliographic reference list.

Table of Contents

[1. Introduction 5](#_Toc194142656)

# Introduction

Since we were kids, we all wanted to grow and take part to the voting process. It was such a mature action, it seemed like it involved a lot of knowledge and that its power was immense. Now, that we’ve grown, we can clearly see that a vote has immense pressure on us, voters, but also on those who are in charge of the entire organization of the process. Entire institutions must be organized in a way that is as efficient as possible and allows for a seamless voting process. Each voter must be identified and verified for their right to vote, then receive their ballot and stamp so they can go to the booth to choose their favorites. It is a complicated process that requires a lot of organizational skills, time, and human resources.

In contemporary democracies, the public trust in electoral outcomes decreases and people start to really question the existing infrastructure and its credibility. Even though we live in such a digitalized world, the vote counting process has lagged behind. We rely on a lot of people to put in a considerable effort to provide the population with statistics following their votes.

This dissertation investigates the design and implementation of a voter authentication architecture, focusing on how emerging technologies can strengthen democratic processes by mitigating identity fraud end enhancing trust in digital voting.

Blockchain technology has emerged as a viable option for transparent and safe electronic voting systems. E-voting systems can enhance voter anonymity, reduce fraud and manipulation and boost election process trust by utilizing blockchain technology’s decentralization, immutability and transparency. Furthermore, compared to traditional voting methods, blockchain-based electronic voting systems can save money and time.

Conventional voting methods sometimes depend on centralized organizations, which can create weaknesses like election fraud or result manipulation. A potential remedy for the shortcomings of conventional and other e-voting methods is provided by the decentralized and unchangeable characteristics of blockchain technology. It can provide a transparent and impenetrable platform for electronic voting. By combining consensus protocols and cryptographic techniques, blockchain-based electronic voting systems offer safe, verifiable and auditable voting processes.

Traditional paper-based voting procedures are still prone to fraud, human error and inefficiency in many electoral systems, which is why governments and corporate groups are looking into digital alternatives. However, there has been doubt about the reliability of these digital endeavors, especially in the wake of high-profile data integrity problems and cybersecurity attacks. This mistrust emphasizes how important it is to have an electronic voting system that strengthens security measures while also guaranteeing user accessibility. A hopeful remedy for this lack of confidence is blockchain technology, which consists of a series of blocks that use consensus algorithms to permanently record every transaction.

The distributed and append-only features of blockchain are what make it so effective in the electoral setting. Vote tampering is reduced since votes recorded on a blockchain are nearly impossible to change after the fact. Alongside this immutability, blockchain-based systems frequently use cryptographic methods to ensure the secrecy and authenticity of voter data, including hashing and public/private key encryption. However, there are still issues regarding the most effective way to confirm voter IDs prior to allowing people to vote on the blockchain. Thus, authentication becomes a crucial element that guarantees every vote is cast by a legitimate, registered vote. E-voting systems may guarantee that only authorized voters participate in the election process by utilizing sophisticated identity verification and biometric matching techniques.

The urgent necessity to balance the potential of blockchain’s security features with the real-world difficulty of certifying a frequently sizable and diverse electorate is what motivates this research. By automating voter verification, blockchain technology combined with trustworthy authentication can lower administrative expenses, increase public trust in electronic voting and lessen the possibility of fraudulent ballots or duplicate voting. Furthermore, by allowing independent auditors and election officials to confirm results using cryptographic proofs rather that proprietary, opaque software, such a system can promote transparency and traceability.

The inherent benefits of blockchain technology for safe data processing support the choice to base electronic voting on it. Because blockchain is a ledger, the votes that are recorded are protected from tampering, making it impossible for bad actors to change, remove, or falsify records without being discovered. Because it provides an auditable ballot trail that is consistent throughout the network of participating nodes, this feature is essential for maintaining election integrity.

Importantly, the importance of strong authentication techniques is also emphasized in this research. Blockchain can offer consensus-driven validation and maintain data integrity, but it is unable to independently verify a voter’s identity. The validity of blockchain’s unchangeable record is rendered irrelevant if an unauthorized user manages to access the system; the ledger will still record an invalid vote. In order to bridge this gap, the study looks into how blockchain technology can be integrated with biometric or multi-factor authentication systems, providing a comprehensive defense against impersonation and unwanted access. The suggested solution aims to strike a balance between user-friendliness and strict security techniques, such as facial matching and government-issued ID card analysis.

The dissertation uses interdisciplinary insights from identity verification, distributed computing and cryptography in choosing this strategy. The foundation of an electronic voting application might theoretically be other technologies, but blockchain is the only one that combines distributed consensus, transparency and cryptographic security in a way that satisfies the fundamental needs of a democratic election. The strategy aims to provide a reliable system where stakeholders may verify the procedure and the outcomes without depending on the internal records of a central authority when combined with stringent authentication.

This dissertation’s focus is on a thorough analysis of the efficacy and security of a blockchain-based electronic voting system that uses strong user authentication. Although this study’s foundation is informed by earlier research on blockchain applications and digital identity verification, the current study focuses on a single area: maintaining vote integrity in a safe online setting. Thus, the following primary areas are examined in this dissertation.

It begins by examining the theoretical underpinnings and real-world applications of blockchain technology in e-voting contexts, with a focus on the system’s capacity to uphold integrity, transparency and auditability. Second, it discusses how sophisticated user authentication techniques, especially those that use biometric information, might confirm voters’ identities prior to granting them access to the blockchain, reducing the possibility of multiple votes or impersonation by the same person. The study looks into the cryptographic safeguards for identity data as well as the effects incorporating such safeguards into an election process has on user experience.

The dissertation also assesses the system’s performance under normal election loads, emphasizing the ways in which network latency, blockchain throughput and cryptographic calculations affect the viability of widespread deployments. A key component of this work includes security issues, such as handling anonymized data and resilience of denial-of-service assaults. The study evaluates how well a blockchain-based electronic voting system with strict authentication procedures functions in real-world operational scenarios by putting these factors into practice in test or simulated situations.

While nothing that real-world adoption also depends on policy, legal frameworks and public acceptance, the dissertation focuses on conceptual and technical validations of blockchain security and authentication efficacy in defining its limitations. Despite note being the main focus, these social and legal aspects are acknowledged as having a significant impact on future scalability and useful implementation. The ultimate goal of this work is to clarify how e-voting may advance the goal of safe, transparent and reliable elections in the digital age by examining the complexities of blockchain protocols and cutting-edge authentication methods.

1. Used Technologies

Modern electronic voting and digital identity systems demand a combination of robustness, security, and scalability that no single technology can provide on its own. **Blockchain technology** has emerged as a foundation for such systems due to its decentralized trust model and tamper-evident ledger. By design, a blockchain maintains an immutable record of transactions distributed across many nodes, making large-scale manipulations of data (such as votes or identity records) extremely difficult​ [1] [pmc.ncbi.nlm.nih.gov](https://pmc.ncbi.nlm.nih.gov/articles/PMC8434614/#:~:text=online%20voting%20solutions%20are%20viewed,replacement%20for%20traditional%20electronic%20voting).

This chapter explores the **integrated architecture** of a blockchain-based system, focusing on the key technologies that work in concert to enable secure identity verification and e-voting. We introduce the fundamentals of blockchain ledgers, emphasizing smart contracts and transactions as programmable, self-executing agreements on the ledger. We then examine **Hyperledger Fabric**, a permissioned blockchain framework well-suited for enterprise and governmental applications, highlighting how it manages identities and transactions in a restricted network. Next, we discuss the role of traditional databases like **MySQL** and in-memory stores like **Redis** within decentralized applications – explaining how off-chain data management complements on-chain data for performance and scalability. To secure user interactions, we review authentication and authorization standards (**JWT** and **OAuth2**) that can be integrated with blockchain systems to ensure only authorized participants can submit transactions or votes. We also consider the deployment infrastructure, specifically **NGINX** as a web server and load balancer, which provides a reliable interface and distribution layer for blockchain-based services. Finally, we explore AI-powered tools – **Tesseract OCR** and **Amazon Rekognition** – used for identity verification and document digitization, and explain their critical role in verifying voter identities and documents before linking that information to blockchain records.

In the remainder of this chapter, each of these components is discussed in detail. Section 2.1 explains blockchain fundamentals, including how transactions are processed and how smart contracts function. Section 2.2 covers Hyperledger Fabric as a permissioned blockchain and its architectural elements. Section 2.3 examines the use of Redis and MySQL for state management in blockchain applications. Section 2.4 describes JWT and OAuth2 for securing user authentication and authorization in a blockchain ecosystem. Section 2.5 highlights the role of NGINX in serving and scaling blockchain-based web services. Section 2.6 presents Tesseract and Amazon Rekognition for identity verification and document processing, illustrating how they integrate into a blockchain-backed system. Together, these sections provide a comprehensive view of a modern blockchain-based system architecture, setting the stage for analyzing how such an architecture can deliver a secure e-voting solution with strong identity verification.

2.1 Blockchain Fundamentals: Transactions and Smart Contracts

Blockchain technology is essentially a distributed ledger that records transactions in a verifiable and permanent way across a peer-to-peer network. The concept was introduced with Bitcoin in 2008, demonstrating how a decentralized ledger could support a digital currency without a central authority [2] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=process%20forms%20the%20ledger%20by,a%20set%20of%20known%2C%20iden%02tified).

A transaction in a blockchain context is an operation proposed by a user (such as transferring a coin or casting a vote) that is cryptographically signed and broadcast to the network. Once transactions are validated by the network’s consensus mechanism, they are grouped into blocks. Each block is linked to the previous one via a cryptographic hash, forming a chronological blockchain. This chain of hashed blocks ensures **immutability**: once a block (and its transactions) is confirmed and added, it cannot be altered retroactively without breaking the cryptographic links​ [3] [pmc.ncbi.nlm.nih.gov](https://pmc.ncbi.nlm.nih.gov/articles/PMC8434614/#:~:text=so%20a%20blockchain%20by%20itself,the%20issue%20of%20voter%20privacy). This property is invaluable for e-voting and identity systems because it provides a tamper-evident record of all actions (votes cast, identities registered, etc.), thereby enhancing transparency and trust in the system.

Early blockchains like Bitcoin were limited in functionality, essentially supporting simple transactions (e.g., payment transfers) with basic scripting. The next major advance was the introduction of **smart contracts**, most prominently by the Ethereum platform in 2015​[4] [pmc.ncbi.nlm.nih.gov](https://pmc.ncbi.nlm.nih.gov/articles/PMC8434614/#:~:text=match%20at%20L275%20introduced%20much,The). Smart contracts are self-executing programs stored on the blockchain that run automatically when predefined conditions are met. The concept was first described by Nick Szabo in the 1990s as “a set of promises, specified in digital form, including protocols within which the parties perform on these promises” [5] [pmc.ncbi.nlm.nih.gov](https://pmc.ncbi.nlm.nih.gov/articles/PMC8434614/#:~:text=match%20at%20L275%20introduced%20much,The) [6] [scholarship.law.georgetown.edu](https://scholarship.law.georgetown.edu/cgi/viewcontent.cgi?article=3451&context=facpub#:~:text=,%E2%80%9D%20Szabo). In a blockchain context, a smart contract’s code is deployed to the ledger and typically defines rules and procedures (for example, an election smart contract might define how votes can be submitted and tallied). When a user wants to invoke a smart contract’s function, they create a transaction that calls that function, embedding any required parameters. This transaction is then propagated through the network and, upon consensus agreement, each network node executes the contract code to ensure the outcomes are the same everywhere (this is part of achieving **distributed consensus** on the new state)​ [7] [pmc.ncbi.nlm.nih.gov](https://pmc.ncbi.nlm.nih.gov/articles/PMC8434614/#:~:text=parties%20perform%20on%20these%20promises%E2%80%9D,The). The result of the execution (e.g., a vote being recorded or a credential being issued) is recorded on the ledger as part of a new block.

Smart contracts greatly expand the capabilities of blockchain beyond simple transfers. They enable **programmable transactions** – logic that can enforce complex workflows or business rules automatically. For instance, in an e-voting scenario, a smart contract can be programmed to allow a vote to be cast only if certain conditions are satisfied (such as the voter being registered and not having voted before). Once a vote is cast via the contract, the contract can instantly count or store it, and perhaps even prevent any further vote from that same voter address, thereby automating eligibility and tallies. Because the contract’s rules are transparent and immutable on the blockchain, stakeholders can trust that the election is executed exactly as coded, without relying on a central election authority. This trust through transparency is a key benefit of blockchain in elections​ [8] [pmc.ncbi.nlm.nih.gov](https://pmc.ncbi.nlm.nih.gov/articles/PMC8434614/#:~:text=Blockchain%20technology%20came%20into%20the,This%20study). Similarly, for identity verification, a smart contract could require that a digital identity token is present (or a user’s identity has been verified by an authorized party) before allowing certain actions, ensuring that only verified individuals can, say, register for a vote or access a service.

Another fundamental aspect of blockchain operations is the **transaction lifecycle**. Using Ethereum as an example of a public blockchain: a user’s client (wallet) crafts a transaction (such as invoking a function of a smart contract), signs it with the user’s private key (ensuring authenticity), and broadcasts it to the network. Miners (or validators) collect pending transactions and execute the smart contract code to verify the outcome (this is part of transaction validation). In Ethereum’s case, this execution occurs during mining and each transaction’s computations consume “gas” to incentivize and limit work done on-chain​ [9] [ieeexplore.ieee.org](https://ieeexplore.ieee.org/iel8/6287639/6514899/10695085.pdf#:~:text=Smart%20contracts%20are%20one%20of,Understanding%20the) [10] [ieeexplore.ieee.org](https://ieeexplore.ieee.org/iel8/6287639/10380310/10695085.pdf#:~:text=on%20blockchain%20fundamentals%20,blockchain%20smart%20contracts%2C%27%27%20IEEE). A consensus protocol (like Proof of Work in original Ethereum or Proof of Stake in modern Ethereum) then decides which proposed block of transactions is accepted. Once consensus is reached, the block is appended to the chain and the transaction is considered confirmed. Every full node in the network updates its copy of the ledger and the world state (the current values of all accounts and contract storage) to reflect the results of the transactions in that block. Through this process, blockchains achieve **eventual consistency** across all nodes without a central coordinator.

Immutability and distributed consensus come at a cost: public blockchain networks often face performance and scalability limitations (e.g., limited transactions per second and latency) and all data is transparently visible, which can conflict with privacy needs. These challenges have spurred the development of specialized blockchain frameworks and hybrid architectures (discussed later) to better serve use cases like e-voting. Nonetheless, the core features of blockchains – decentralized trust, immutable audit trails, and smart contracts – provide an excellent foundation for building secure voting and identity systems. In fact, researchers have noted that blockchain’s end-to-end verifiability and elimination of single points of failure can help solve many issues in electronic voting systems​ [11] [pmc.ncbi.nlm.nih.gov](https://pmc.ncbi.nlm.nih.gov/articles/PMC8434614/#:~:text=Blockchain%20technology%20came%20into%20the,This%20study). In summary, a blockchain can act as the **secure backbone** of a modern system, where each transaction (be it a vote cast or an identity attestation) is indelibly recorded, and smart contracts ensure the system’s rules are enforced uniformly and transparently across all participants.

2.2 Hyperledger Fabric as a Permissioned Blockchain System

While public blockchains like Bitcoin and Ethereum allow anyone to participate, many e-voting and identity applications involve a consortium of known entities (e.g. government agencies, election commissions, or universities) rather than the general public. **Hyperledger Fabric** is a prominent permissioned blockchain framework designed for such private or consortium scenarios. Fabric is an open-source project under the Linux Foundation’s Hyperledger umbrella, targeting enterprise use cases that require fine-grained access control, high transaction throughput, and the flexibility to integrate with existing infrastructure without a cryptocurrency​ [12] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=distributed%20applications%20written%20in%20standard%2C,way%20blockchains%20cope%20with%20non%02determinism) [13] [pseudoyu.com](https://www.pseudoyu.com/en/2021/03/20/blockchain_hyperledger_fabric_structure/#:~:text=%E6%88%91%E4%BB%AC%E7%9D%80%E9%87%8D%E6%9D%A5%E8%AE%B2%E8%AE%B2%E5%85%B6%E4%B8%AD%E5%BA%94%E7%94%A8%E6%9C%80%E5%B9%BF%E6%B3%9B%E7%9A%84%20,%E6%A0%87%EF%BC%88%E4%B8%9A%E5%8A%A1%E9%9C%80%E6%B1%82%EF%BC%89%E4%BD%86%E5%BD%BC%E6%AD%A4%E4%B8%8D%E5%AE%8C%E5%85%A8%E4%BF%A1%E6%81%AF%E7%9A%84%E5%AE%9E%E4%BD%93%E4%B9%8B%E9%97%B4%E7%9A%84%E4%B8%9A%E5%8A%A1%E6%8F%90%E4%BE%9B%E4%BA%86%E4%BF%9D%E6%8A%A4%EF%BC%8C%E4%BE%8B%E5%A6%82%E8%B7%A8%E5%A2%83%E7%94%B5%E5%95%86%E3%80%81%E8%B5%84%E9%87%91%E4%BA%A4%E6%98%93%E3%80%81%E6%BA%AF%E6%BA%90%E7%AD%89%E3%80%82). In a permissioned network, all participants are authenticated and identified – there are no anonymous miners. This model aligns well with e-voting, where only authorized voters and officials should participate, and with identity systems where users and authorities have verified identities.

**Architecture and Components:** Hyperledger Fabric’s architecture is notably modular and differs from public blockchain designs in its transaction flow. Instead of the classic “order-execute” model (where transactions are ordered into a block and then every node executes them), Fabric employs an **execute-order-validate** approach​ [14] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=Figure%202%3A%20Execute,cannot%20be%20chosen%20or%20modified) [15] [pseudoyu.com](https://www.pseudoyu.com/en/2021/03/20/blockchain_hyperledger_fabric_structure/#:~:text=%E8%80%8C%20%60Fabric%20%60%E9%87%87%E7%94%A8%E4%BA%86%20%60Execute%20,Update%20State%20%60%E6%9E%B6%E6%9E%84%E3%80%82). The Fabric transaction flow has three phases: execution (endorsement), ordering, and validation. In the execution phase, transactions are first simulated on a subset of nodes called endorsing peers **before** they are ordered. A client submits a transaction proposal (for example, “record vote for candidate X”) to endorsing peers that have the relevant smart contract (called chaincode in Fabric) installed. Each endorser executes the chaincode transaction against its current ledger state, but in a sandboxed way – the results are not yet committed, they are just captured as a read-set and write-set (the state elements read and the tentative updates)​ [16] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=The%20endorsers%20simulate%20the%20proposal%2C,key%20have%20monotonically%20increas%02ing%20version) [17] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=As%20a%20result%20of%20the,client%20collects%20endorsements%20until%20they). The endorsers then cryptographically sign these results (an endorsement). If the endorsing peers all agree on the outcome (e.g., all simulate that the vote is valid and would increment candidate X’s vote count), the client collects these endorsements and proceeds to the ordering phase.

In the ordering phase, a dedicated service (the ordering service or orderers) takes endorsed transactions from clients network-wide and globally orders them into blocks, using a consensus protocol. Notably, Fabric’s default consensus is crash-fault tolerant (Solo, Kafka, or Raft in newer versions) and can also plugin Byzantine fault tolerant algorithms if needed​ [18] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=In%20a%20public%20or%20permissionless,on%20the%20iden%02tities%20of%20the). Because membership is permissioned (i.e., participants are known), Fabric can leverage classic consensus (like Paxos/Raft or BFT protocols) without the heavy overhead of Proof of Work. The ordered blocks of transactions are then disseminated to all peers.

Finally, in the validation phase, each peer (including non-endorsing peers that didn’t simulate initially) takes the block of transactions and validates them before committing. Validation involves checking that each transaction’s endorsements meet the predefined endorsement policy and that there have been no conflicting updates (a read-write conflict check)​ [19] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=%E2%80%A2%20An%20endorsement%20policy%20that,cannot%20be%20chosen%20or%20modified) [20] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=the%20endorsement%20policy%20configured%20for,for%20all%20transactions%20in%20the). For example, a vote transaction might require endorsements from N of M endorsers as per chaincode policy; peers will verify those signatures. They also ensure the read-set version numbers still match the current state (ensuring no other transaction has modified a value in the meantime)​ [22] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=writeset%2C%20consisting%20of%20the%20state,particular%2C%20this%20requires%20all%20endorsers). Any transaction failing these checks is marked invalid and is not applied to the ledger state, though it remains recorded in the block for audit. All valid transactions are then committed to the ledger, updating the world state in each peer’s database. This validate-commit design allows Fabric to achieve high throughput by executing transactions in parallel (during endorsement) and only ordering relatively small transaction results (rather than full transaction execution on chain)​ [23] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=Figure%202%3A%20Execute,cannot%20be%20chosen%20or%20modified) [24] [pseudoyu.com](https://www.pseudoyu.com/en/2021/03/20/blockchain_hyperledger_fabric_structure/#:~:text=%E8%80%8C%20%60Fabric%20%60%E9%87%87%E7%94%A8%E4%BA%86%20%60Execute%20,Update%20State%20%60%E6%9E%B6%E6%9E%84%E3%80%82). The trade-off is a more complex protocol, but it suits enterprise needs for performance and consistency under a permissioned trust model.

The major components of a Fabric network include: **Peers**, **Orderers**, and **Membership Services**. Peers are the nodes that maintain the ledger and state; they come in two special types: endorsing peers (which execute chaincode for simulation) and committing peers (which validate and commit transactions; in practice, all peers commit, and some are also endorsers). Orderers (also called the ordering service nodes) collectively form the consensus layer that orders transactions into blocks. The **Membership Service Provider (MSP)** is the component that handles identity management – it issues and validates the cryptographic certificates for participants (organizations, peers, clients). Identities in Fabric are typically backed by a Certificate Authority (Fabric CA or an external CA), and every peer and user possesses a digital certificate. This ties each action on the network to a known identity (e.g., a specific voter or official)​ [25] [researchgate.net](https://www.researchgate.net/figure/Hyperledger-Fabric-architecture_fig7_358452291#:~:text=Compared%20with%20blockchain%20technology%2C%20the,this%20system%20through%20the) [26] [pseudoyu.com](https://www.pseudoyu.com/en/2021/03/20/blockchain_hyperledger_fabric_structure/#:~:text=%E6%AD%A3%E5%A6%82%E4%B8%8A%E8%BF%B0%E8%81%94%E7%9B%9F%E9%93%BE%E7%89%B9%E6%80%A7%E4%B8%AD%E6%89%80%E8%BF%B0%EF%BC%8C).

**Integration with Identity Management:** Fabric’s design explicitly accommodates integration with existing identity systems. Identities are modular – the MSP can plug into organizational directories or external PKI infrastructure. The Fabric whitepaper notes that Fabric implements a “portable notion of membership” that can integrate with industry-standard identity management systems​ [27] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=domain,decisions%2C%20its%20most%20prominent%20implementation). In practice, this means an organization could map Fabric identities to real-world identities (for instance, linking a blockchain user certificate to a government-issued ID or an OAuth2 federation identity). This feature is highly relevant for e-voting: one can ensure that each on-chain user corresponds to an actual eligible voter by issuing Fabric credentials only to verified individuals. Because participants are permissioned, one can enforce one person, one identity on the blockchain. As an example, a voter might register through a traditional registration system (verifying their real identity) and then be given credentials (a digital certificate) to interact with the Fabric network. The **permissioned model** also means that only authorized nodes (run by the election organizers or stakeholders) can host a peer or orderer, limiting access to sensitive transactions and enhancing privacy. The use of channels in Fabric (isolated sub-ledgers that a subset of peers can see) can further compartmentalize data; for instance, one channel could hold voting transactions visible only to election officials, while another channel could publish an anonymized tally for public verification.

Fabric does not use a built-in cryptocurrency for incentives; instead, it relies on the consortium governance for operation. This is advantageous in voting systems as it avoids the complexity and volatility associated with cryptocurrencies and allows focus on transaction logic and policy. The lack of mining and the use of efficient consensus protocols give Fabric much higher throughput – thousands of transactions per second in certain configurations​ [28] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=aspects%2C%20as%20well%20as%20its,%E2%88%97Work%20done%20at%20IBM) – which is beneficial when an election system needs to record a large number of votes in a short time.

**Chaincode (Smart Contracts in Fabric):** In Fabric, smart contracts are referred to as chaincode and can be written in general-purpose programming languages (like Go, Java, or Node.js). This flexibility allows developers to implement complex logic (for example, vote tallying, or verification flows) using familiar languages and libraries. Chaincode is installed on peers and invoked by network transactions. For e-voting, a chaincode might implement functions such as RegisterVoter, CastVote, and CountVotes. Because Fabric’s chaincode execution is endorsed by peers from specific organizations, one can also enforce that, say, a vote casting transaction must be endorsed by a peer from the election commission and perhaps observers from independent organizations (this could be part of the endorsement policy) before it is considered valid. Such a multi-signatory endorsement mechanism provides an extra layer of trust – multiple parties witness and approve the transaction.

**Relevance to Secure E-Voting:** Hyperledger Fabric’s features align well with the requirements of secure online voting. The permissioned and identity-aware infrastructure means only authenticated voters can interact with the voting smart contract, and any malicious behavior by nodes can be traced to a known organization. Fine-grained access control can be implemented at the chaincode level to segregate roles (voters vs. administrators). Fabric’s confidentiality features (channels and private data collections) can protect sensitive information – for instance, a private data collection could be used to store the mapping of voter IDs to votes (to verify who has voted) accessible only to authorized auditors, while the actual votes could be recorded in anonymized form on a public channel for transparency. Additionally, Fabric’s high throughput and finality (no probabilistic confirmation as in PoW chains) ensure that even a nationwide election’s transactions can be processed in a timely manner with deterministic results.

2.3 Off-Chain Data Management with Redis and MySQL in Decentralized Apps

Blockchain ledgers, by design, store data in a distributed, append-only manner, which guarantees integrity but is not always efficient for all data types and queries. In complex applications like e-voting or digital identity platforms, we often need to handle large volumes of data (such as high-resolution ID document images or analytics on voting results) and support query patterns that blockchains are not optimized for (such as relational queries across many records). This is where traditional databases and caching systems become crucial. **MySQL**, a relational database management system, and **Redis**, an in-memory data store, are two technologies commonly integrated into blockchain-based architectures to manage off-chain data and improve performance.

**MySQL for Persistent and Relational Data:** MySQL is a widely-used SQL database known for ensuring ACID (Atomicity, Consistency, Isolation, Durability) transactions and supporting complex queries with indexes and joins. In a blockchain-based system, MySQL can serve as a system of record for certain kinds of data that do not need to reside on the ledger or that require heavy read/query operations. For example, an e-voting application might use MySQL to maintain a registry of voters with detailed profiles (name, address, precinct, etc.), or to store ballots in a structured form for fast retrieval by administrators when displaying results. Storing such data directly on a blockchain could be impractical due to size or privacy (one might not want to put personal information on a public ledger). Instead, a hash or reference of the data can be stored on-chain for integrity, while the full data lives in MySQL. This pattern – often called off-chain storage – leverages the strengths of both systems: the blockchain holds the fingerprint of data to ensure it hasn’t been tampered with, and the SQL database holds the actual data and can execute rich queries on it​ [29] [sciencedirect.com](https://www.sciencedirect.com/science/article/pii/S2096720922000574#:~:text=Azure%20Blockchain%20Workbench%20delivers%20data,The) [30] [link.springer.com](https://link.springer.com/article/10.1007/s41019-023-00212-z#:~:text=the%20difference%20is%20that%20data,the%20integrity%20of%20retrieved%20data).

Another scenario is using MySQL as a mirror or companion to the blockchain data to facilitate reporting and user interface features. For instance, although all vote transactions are recorded on-chain, an application might extract those transactions (perhaps via an event listener or periodic batch job) and insert them into a MySQL database table that is indexed by candidate, region, timestamp, etc. This would allow the application to run complex SQL queries (like retrieving vote counts per region, or finding all voters who have not yet voted) very quickly, something that could be slow or impossible to do directly on the blockchain. Microsoft’s Azure Blockchain Workbench, for example, used this approach – delivering ledger data to an off-chain SQL database and providing a REST API for querying it​ [31] [sciencedirect.com](https://www.sciencedirect.com/science/article/pii/S2096720922000574#:~:text=Databases%20fit%20for%20blockchain%20technology%3A,The). In doing so, it abstracted the complexities of direct chain interaction and provided developers a familiar interface to query blockchain-originated data using SQL.

**Redis for Caching and Fast Data Access:** Redis is an open-source in-memory key–value data store, often used for caching, real-time analytics, and pub/sub messaging. Being in-memory, Redis is extremely fast (read/write operations typically take sub-millisecond time). In a blockchain system context, Redis is commonly employed to cache frequently accessed data and reduce the load on both the blockchain network and any backing databases. For example, consider an identity verification system where after a user is verified on-chain, a flag needs to be checked frequently by a web application to grant access. Rather than querying the blockchain for each page load or API call (which could be relatively slow, on the order of seconds), the system could store the verification status in Redis keyed by the user’s ID. The web application can then check Redis (which is a local network call and very fast) to confirm the user’s verified status. This dramatically improves response times while still maintaining security (the Redis cache can be updated automatically when on-chain events occur, ensuring consistency).

Redis can also act as a message broker between the blockchain layer and other components. Many blockchain clients emit events for new blocks or specific transactions. A service could listen for these events (for example, a vote cast event from a smart contract) and then publish a message to a Redis channel. Other parts of the application (perhaps a notification service or a real-time dashboard) could subscribe to these Redis channels to get immediate updates (e.g., update the vote tally live on a website). This decoupling via Redis Pub/Sub allows a scalable, event-driven architecture where heavy on-chain operations do not directly block user-facing services. It’s worth noting that Hyperledger Fabric itself uses an internal state database (either LevelDB or CouchDB) to store world state for quick lookup​ [32] [arxiv.org](https://arxiv.org/pdf/1801.10228#:~:text=state,Given%20the%20appropriate%20permis%02sion%2C%20a); however, application developers often set up additional caches like Redis at the application layer for domain-specific needs.

**Hybrid Blockchain-Database Workflows:** The combination of on-chain and off-chain storage leads to hybrid workflows. Consider voter registration: a user submits their personal details and documents. The system stores the full details in MySQL (for administrative use), but also writes a registration transaction to the blockchain (perhaps containing a hash of the user's data or a pointer to it). The blockchain transaction serves as an immutable receipt of the registration, timestamped and signed, which can be independently verified. The MySQL record holds the data for quick access by election officials. If a discrepancy ever arises (for instance, someone questions if a voter was registered at a certain time), the blockchain record can be checked – the hash there should match the data in MySQL, proving the MySQL data wasn’t altered, thanks to blockchain’s integrity guarantee​ [33] [link.springer.com](https://link.springer.com/article/10.1007/s41019-023-00212-z#:~:text=match%20at%20L973%20the%20difference,the%20integrity%20of%20retrieved%20data). This approach addresses a common challenge: blockchains alone are not efficient for large data or complex queries, but databases alone lack decentralized trust. Together, they create a balance between **trust and efficiency**​ [34] [link.springer.com](https://link.springer.com/article/10.1007/s41019-023-00212-z#:~:text=The%20success%20of%20blockchain%20technology,offs.%20Then%2C%20by) [35] [link.springer.com](https://link.springer.com/article/10.1007/s41019-023-00212-z#:~:text=management%20systems%20and%20develop%20fusion,meet%20various%20requirements%20in%20practice).

**Data Consistency and Integrity:** One important consideration when integrating off-chain databases is consistency with the blockchain. Since there are two sources of truth (on-chain and off-chain), mechanisms must ensure they do not diverge. Techniques include: using the blockchain as the trigger for database updates (only update the SQL/Redis when a new block/transaction is confirmed), and storing verification hashes on-chain as mentioned. Some advanced solutions use cryptographic techniques or protocols (like **Baseline Protocol**, which uses zero-knowledge proofs) to continuously verify that off-chain records stay in sync with on-chain commitments​ [36] [learn.microsoft.com](https://learn.microsoft.com/en-us/answers/questions/39303/how-to-detect-changes-between-blockchain-data-and#:~:text=%40Alejandro%20Ben%C3%ADtez%20Arag%C3%B3n). In our context, a straightforward approach suffices: for any piece of critical data stored in MySQL, we maintain a corresponding hash or ID on the blockchain. For example, each identity document could be hashed, and that hash stored in a smart contract when the identity is verified. Later, anyone can recompute the hash of the document and compare it to the on-chain value to ensure the document hasn’t been swapped or altered in the database. This way, MySQL can be freely used for its strengths (storing and querying rich data), while the blockchain anchors the data to a tamper-proof record​ [37] [link.springer.com](https://link.springer.com/article/10.1007/s41019-023-00212-z#:~:text=match%20at%20L973%20the%20difference,the%20integrity%20of%20retrieved%20data).

| **Aspect** | **MySQL (Relational DB)** | **Redis (In-Memory Store)** |
| --- | --- | --- |
| Data Model | Relational (tables with rows and columns, SQL queries) | Key–value pairs (strings, hashes, lists, sets, etc.) |
| Storage & Persistence | On-disk storage (durable), with transaction commit logs | In-memory (with optional disk persistence snapshots) |
| Strengths | Complex queries (JOINs, aggregations), ACID transactions, large data storage, structured schemas  [sciencepubco.com](https://www.sciencepubco.com/index.php/ijet/article/download/11759/4580#:~:text=,Glykantzis%20V%2C%20%E2%80%9COn%20the%20security) | Extremely fast read/write, pub/sub messaging, ephemeral caching, simple data access patterns |
| Usage in Blockchain Apps | Storing persistent off-chain data: user profiles, verified documents, vote results for reporting. Ensuring data integrity by comparing with on-chain hashes. Also used to index blockchain data for search (e.g., voter list)​  [sciencedirect.com](https://www.sciencedirect.com/science/article/pii/S2096720922000574#:~:text=Azure%20Blockchain%20Workbench%20delivers%20data,The)  . | Caching frequently accessed data: authentication tokens, verification flags, recent blocks or votes to reduce chain lookups. Queuing and broadcasting events (new vote notifications) to other services for real-time updates. Session store for web applications (stateless backend with state in Redis). |
| Role in Identity/E-Voting | Holds authoritative voter and identity information (e.g., a voter registry table), which can be cross-checked against blockchain records for audits. Facilitates batch analytics (e.g., how many voters verified, turnout statistics by region). | Speeds up verification checks (e.g., “has this voter already voted?” cached as a key). Stores short-lived data like OTP codes, login sessions during voting. Distributes events like “vote cast” to trigger real-time UI updates. |

Table 2.1: Comparison of MySQL and Redis roles in a blockchain-based system.

In practice, a well-architected blockchain solution will use MySQL and Redis together to provide a smooth user experience. For instance, when a voter logs into a web portal to vote, the system may authenticate them and then fetch their voter record from MySQL (to display their details or ballot options). That page might also show whether they have been verified or have voted, which could be a quick lookup to Redis (populated from on-chain data). When the voter submits a vote, a blockchain transaction is created to record it, and simultaneously an entry in Redis could mark that this voter’s vote is “pending” or cast. Once the blockchain confirms the vote (perhaps detectable via an event listener), the MySQL database might be updated with a flag for that voter and Redis cache invalidated or updated. Through this interplay, the user gets immediate feedback (thanks to Redis), the administrators have structured records to query (thanks to MySQL), and the source of truth remains the blockchain (thanks to the immutable transaction stored there).

In summary, **off-chain databases** like MySQL complement blockchains by handling data that is impractical or inefficient to put on-chain, while **caching layers** like Redis ensure that applications remain responsive at scale. Together, they address the performance and usability gaps of blockchain, all the while preserving security by tying back into the blockchain’s verifiable ledger. This kind of hybrid architecture is increasingly recognized as a best practice for enterprise blockchain applications​ [38] [link.springer.com](https://link.springer.com/article/10.1007/s41019-023-00212-z#:~:text=in%20the%20data%20management%20field,of%20each%20type%20of%20fusion) [39] [link.springer.com](https://link.springer.com/article/10.1007/s41019-023-00212-z#:~:text=Finally%2C%20we%20outline%20the%20unsolved,meet%20various%20requirements%20in%20practice), and it is especially pertinent in e-voting systems where large amounts of auxiliary data (voter info, audit logs, user sessions) need management alongside the core blockchain ledger of votes.

2.4 Secure Authentication and Authorization: JWT and OAuth2 Integration

Any system that involves user interaction – be it casting a vote or accessing personal identity data – must ensure that only authorized individuals can perform certain actions. In a blockchain-based application, the blockchain itself might enforce some rules (via smart contracts as discussed), but there is still a need to authenticate users to the application and manage their sessions and permissions off-chain. Two widespread technologies for web security are **JSON Web Tokens (JWT)** and **OAuth2**. They are often used in tandem to provide a secure authentication and authorization infrastructure. In our context, they can be leveraged to authenticate voters or administrators and gate access to the blockchain functions (for example, only allow a user to invoke the “cast vote” transaction if they have a valid session token confirming their identity).

**OAuth2 Overview:** OAuth 2.0 is an authorization framework that allows users to grant a third-party application limited access to their resources on another service without exposing credentials. It is widely used for “login with X” functionality (e.g., “Login with Google”) and for API authorization in microservice architectures. OAuth2 defines several flows (authorization code, implicit, resource owner password, client credentials) to accommodate different scenarios (web-server apps, single-page apps, machine-to-machine, etc.)​ [40] [dl.acm.org](https://dl.acm.org/doi/10.1145/3474624.3477068#:~:text=Nick%20Szabo,Ethereum%3A%20A%20secure) [41] [pmc.ncbi.nlm.nih.gov](https://pmc.ncbi.nlm.nih.gov/articles/PMC8434614/#:~:text=Electoral%20integrity%20is%20essential%20not,through%20which%20judges%20judge%20who). The key idea is the separation of roles: a Resource Owner (user), an Authorization Server (who authenticates the user and issues tokens), a Client (the application requesting access), and a Resource Server (the API or service with protected data). In an e-voting scenario, one could imagine an OAuth2-compliant service where the election authority runs an Authorization Server. When a voter tries to access the voting DApp, they could be redirected to this server to authenticate (perhaps using multi-factor auth or government eID). Upon success, the voter gets an access token – often a JWT – that the DApp can use to authorize blockchain transactions on the voter’s behalf.

**JSON Web Tokens (JWT):** JWT is a compact, URL-safe means of representing claims to be transferred between two parties. A JWT is a string consisting of three parts (header, payload, signature) separated by dots, e.g., xxxxx.yyyyy.zzzzz. The header typically declares the token type and signing algorithm (e.g., HS256 or RS256). The payload contains claims – statements about the user or token such as user ID, roles, expiration time (exp), issuer (iss), etc., in JSON format. The signature is generated by signing the header and payload (after encoding) with a secret key or private key. JWTs are self-contained: when a server receives a JWT from a client, it can verify the signature (with the secret or public key) and trust the claims inside without needing to query a database for session info​ [42] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=enable%20online%20user%20identity%20verification,With%20Amazon) [43] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=Amazon%20Rekognition%20makes%20it%20easy,With%20a). This makes JWTs very popular for stateless authentication – the server doesn’t store session state; the token itself carries the authentication state and can be validated quickly.

In our system, JWTs could be used to represent the authenticated session of a user who has logged in. For example, once a voter logs in through the OAuth2 process, the application may receive a JWT identifying that voter (with a claim like sub: voter12345 and perhaps scope: vote). The application will include this JWT in subsequent requests made by the user (e.g., a request to the backend service that interfaces with the blockchain). The backend service can verify the JWT’s signature (to ensure it was indeed issued by the trusted auth server and not tampered with) and then allow the operation – for instance, constructing a transaction to record a vote – if the token is valid and has not expired. The use of standard JWT and OAuth2 means we can also integrate with existing identity providers. For instance, if there is a national ID system or university single sign-on, we could plug that in via OAuth2, and then use the obtained JWT in our blockchain app.

**Authorization and Smart Contracts:** While OAuth2/JWT handle user login and API access control off-chain, how do they relate to on-chain permissions? One approach is indirect: the application ensures only the rightful user can invoke certain smart contract functions by requiring a valid token before it calls the blockchain on behalf of that user. In a permissioned blockchain like Fabric, each user might have their own blockchain credential, so the application could even map a JWT-authenticated user to invoking the chaincode with that user’s certificate. For example, Hyperledger Fabric can integrate with an OAuth2 identity – one could set up the Membership Service to accept an OAuth2 token or use an external IdP to issue the X.509 certificates. Although Fabric doesn’t natively consume JWTs, one could create a middleware where a JWT representing a voter is exchanged for a Fabric enrollment certificate for that voter (perhaps using an intermediate registration step). Research prototypes and some enterprise systems have proposed mapping OAuth-issued identities to blockchain identities for unified access control​ [43] [researchgate.net](https://www.researchgate.net/figure/Hyperledger-Fabric-architecture_fig7_358452291#:~:text=Compared%20with%20blockchain%20technology%2C%20the,this%20system%20through%20the). In a simpler way, our system might run a single blockchain client on the server that submits transactions, but only does so if the requesting user presented a valid JWT. In that case, the link between off-chain auth and on-chain action is enforced by the application logic.

**Roles and Permissions:** JWTs can encapsulate user roles or scopes (e.g., “admin”, “voter”). Our voting smart contracts may have certain functions only an admin can call (like startElection or auditResults). The application can include the user’s role in the JWT and check it before allowing those contract calls. Alternatively, in Fabric, channel access control can specify which organization’s members can invoke which chaincode functions. In either case, OAuth2 and JWT provide the initial trust that the user is who they claim to be and has the roles assigned by the system. After that, either the off-chain code or the on-chain code (or both) can enforce authorization rules. For instance, a chaincode function could read the invoking user's identity (Fabric passes along the certificate info) and ensure it matches the expected role for that operation. But to even reach that point, the user would need to have logged in and obtained credentials via OAuth2.

**Securing the Process:** Using OAuth2 and JWT in a blockchain context should follow security best practices: tokens should be signed with strong algorithms (RS256 or HS256 with a strong secret)​ [44] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=detect,hat%2C%20and%20confirming%20that%20the), use short expiration times to limit damage if stolen, and be transmitted over HTTPS to prevent eavesdropping. We might also employ refresh tokens (a long-lived token stored securely, used to get new short-lived JWTs) so that users don’t have to login frequently but still limit the window if an access token leaks. For e-voting, security is paramount, so multi-factor authentication could be integrated into the OAuth2 login step (for example, requiring an SMS code or biometric verification through an IdP). Once the user obtains the JWT, the subsequent interactions with the blockchain can be considered under the user’s identity.

It’s worth highlighting that blockchain itself provides **cryptographic authentication** at the protocol level – users have private keys for signing transactions. In public blockchains, simply possessing the private key is often the only authentication needed to perform actions tied to that key’s address. In permissioned blockchains like Fabric, each user has a digital certificate and private key. In our system, however, we want to bind those blockchain identities to real-world identities and typical login flows. OAuth2 and JWT serve as that bridge. One could imagine that upon a successful OAuth2 login, the system uses the user’s known identity to look up or generate a blockchain key pair for them (or the user may have one stored securely). If using Fabric, the user’s X.509 certificate might be issued at registration time and mapped to their OAuth2 identity. Then, when performing a transaction, the application could sign it with the user’s private key (retrieved securely, perhaps after the user re-authenticates or uses a secure enclave) and submit it to the network. This way, the on-chain transaction is still signed by the user’s blockchain identity, and on-chain access control (like Fabric’s MSP or chaincode ACLs) can verify it, **in addition** to the off-chain JWT checks. Such an approach layers security: the user must have a valid web session and their blockchain credential to act.

**Benefit of Standards:** By using JWT/OAuth2, our blockchain system can integrate with existing identity ecosystems. For example, if a government has an OAuth2-based citizen login portal, we could use that for voter authentication – the voter would log in via the government portal, which issues a JWT that our e-voting app trusts (perhaps the JWT is signed by the government's public key). This avoids creating yet another set of credentials for users and leverages well-tested infrastructure. Moreover, OAuth2 scopes could be used to allow or deny certain actions dynamically. If, for instance, a particular election is only open to users of a certain group (say region or age), the Authorization Server can include that eligibility in the token or simply not issue a token with the “voting” scope to ineligible users.

2.5 NGINX as a Web Server and Load Balancer in Blockchain Deployments

Blockchain applications, despite their novel backends, still need to interface with users through conventional web protocols. In an e-voting scenario, thousands or millions of users might access a web application to register, verify identity, and cast votes, all of which ultimately trigger blockchain transactions in the backend. Ensuring this web layer is scalable, secure, and reliable is critical. **NGINX** is a high-performance HTTP server and reverse proxy server that is commonly employed to meet these needs. It can serve static content (like web pages or results dashboards), forward API requests to backend services (like the application server that interacts with the blockchain), and balance load across multiple server instances. Essentially, NGINX sits at the front of the system’s architecture as the entry point for all HTTP(S) requests, acting as the traffic cop and gatekeeper.

**Web Serving and Reverse Proxy:** NGINX has an event-driven, asynchronous architecture known for its ability to handle a large number of concurrent connections efficiently. It is often used to serve static files (HTML, CSS, JS, images) directly to users, which is useful if our voting application front-end is a single-page application or a static site. More importantly, NGINX is typically configured as a **reverse proxy** in front of application servers. In our system, we might have a set of backend services – for example, a web service that provides RESTful endpoints for voting (allowing a user to submit a vote, or check their status), an API for administrators to tally results, and perhaps microservices for handling identity verification or sending notifications. NGINX can be configured to route incoming requests to the appropriate service based on the URL or domain. For instance, a request to https://vote.example.com/api/vote could be forwarded to the vote handling service, while https://vote.example.com/admin might go to an admin service. By centralizing routing in NGINX, we can change backend implementations or scale them without changing the external interface to users.

**Load Balancing:** One of the strengths of NGINX is its built-in load balancing capabilities. It can distribute incoming requests among multiple server instances using strategies like round-robin, least connections, or IP-hash, thereby increasing the throughput and reliability of the system​ [45] nginx.org [46] [developer.hashicorp.com](https://developer.hashicorp.com/nomad/tutorials/load-balancing/load-balancing-nginx#:~:text=Set%20up%20load%20balancing%20with,Balancing%20Strategies%20for%20Consul). In a high-traffic scenario like a general election, we can deploy multiple application servers (perhaps containerized or on cloud VMs) that all connect to the blockchain network. NGINX will spread users’ requests across these servers so that no single one becomes a bottleneck. If one server instance fails or becomes unresponsive, NGINX can detect that (health checks) and stop sending traffic its way, effectively isolating the failure and providing high availability. This is crucial for e-voting where downtime could disenfranchise users or erode trust. By horizontally scaling the middle-layer servers and using NGINX to balance them, the system can handle spikes in user activity (for example, many people voting just before a deadline) smoothly.

Moreover, NGINX can proxy connections to the nodes of the blockchain network itself. In some deployments, client applications might interact with a blockchain node through an API (e.g., REST or gRPC). Running multiple blockchain nodes behind an NGINX load balancer can help distribute query load (such as many users checking the public election results stored on-chain). It also abstracts the cluster of nodes behind a single endpoint. For example, users could always connect to https://fabric.example.com which is served by NGINX; NGINX then forwards requests to one of several Fabric REST gateways or nodes. This way, if we need to take nodes down for maintenance or add new nodes, users need not change anything on their side.

**TLS Termination and Security:** NGINX is often used as a TLS/SSL terminator – it handles the HTTPS encryption, decrypting incoming requests and passing them to backend services over the local network. This offloads the computational work of encryption from the application servers and centralizes certificate management in one place. Our voting website will definitely use HTTPS to protect user credentials and other data in transit. NGINX can be configured with strong TLS settings (disabling older protocols, using modern ciphers) and features like HTTP/2, which improves performance for content loading. It can also enforce web security headers (like Content Security Policy, HSTS) to harden the web interface against attacks.

As a reverse proxy, NGINX can provide an additional layer of defense: it can implement basic web application firewall (WAF) rules, rate limiting, and filtering of requests. For instance, if someone tries to flood the system with requests (a simple DoS attack), NGINX can rate-limit requests from that IP. If a vulnerability in the web API were to be discovered, NGINX could be used to block certain paths or patterns as a quick mitigation. Additionally, NGINX supports authentication at the proxy level; however, in our architecture, authentication is handled by the application (via OAuth2/JWT), so NGINX would mainly ensure that the OAuth2 flows (like redirects) and API calls are properly forwarded to the right services.

**Static Content and Client App**: In a likely setup, the e-voting user interface might be a single-page application (SPA) that is downloaded to the user’s browser. NGINX can efficiently serve this static bundle. Once loaded, the SPA will communicate via AJAX/Fetch API to our backend (to do things like get the list of candidates or submit a vote). All those calls go to api.example.com or similar, which NGINX handles and proxies to our internal API endpoints. This separation (static front-end vs API) can also be done by NGINX through different virtual host configurations or path-based routing.

**Integration with Docker/Kubernetes:** Modern deployments often use containers and orchestration (like Kubernetes). NGINX (or its variant NGINX Plus or the open-source NGINX Ingress Controller) is commonly used in Kubernetes as an Ingress controller to manage external access to services. If our system is containerized (each component – web app, identity verifier, blockchain node – is a container), we could use NGINX as the glue that ties the container network to the outside world. It would make our system more portable and easier to manage. Kubernetes could also handle some of the load balancing, but NGINX provides additional control and features.

**Performance**: NGINX’s efficiency ensures that the addition of a reverse proxy does not become a new bottleneck. It’s known to handle on the order of tens of thousands of requests per second on modest hardware​ [47] [f5.com](https://www.f5.com/content/dam/f5/corp/global/pdf/ebooks/High-Performance-Caching-NGINX-Plus.pdf#:~:text=%5BPDF%5D%20High,of%20the%20top%20100%2C000%20websites), which is likely to exceed what our backends or blockchain can do, meaning it will comfortably handle the traffic overhead. By using techniques like content caching (for static assets or even API responses if appropriate), NGINX can further reduce load on the application servers. For example, while votes must always hit the backend to create a transaction, something like fetching the current vote tally could be cached for a short interval (say updated every few seconds) at the NGINX layer, so if hundreds of users refresh the results page simultaneously, they get a cached response rather than each triggering a new database or blockchain query. This improves perceived speed and reduces backend strain. Of course, caching must be used carefully in voting to avoid serving stale or inconsistent data – likely we’d cache only non-sensitive or infrequently changing data.

**Monitoring and Logging:** NGINX also centralizes logging of requests, which is useful for audit and monitoring. We could aggregate logs to see how many vote submissions are coming in and detect anomalies (like an unusual surge from a single IP). These logs might even feed into a real-time monitoring dashboard to watch the progress of voting or identify any errors.

In summary, **NGINX provides the interface between users and the blockchain application services**, ensuring that the web platform is scalable, fast, and secure. It is a proven component in web architecture: indeed, more than half of the top 100,000 websites use NGINX as their web server or proxy​ [48] [f5.com](https://www.f5.com/content/dam/f5/corp/global/pdf/ebooks/High-Performance-Caching-NGINX-Plus.pdf#:~:text=%5BPDF%5D%20High,of%20the%20top%20100%2C000%20websites), attesting to its performance and reliability. In our blockchain-based system, NGINX does not directly interact with the blockchain, but it is an indispensable part of the system architecture. It manages network traffic so that the underlying services (identity verification service, voting transaction service, results service, etc.) can do their jobs without being overwhelmed or directly exposed to the internet. By incorporating NGINX, we adhere to a modular design: the blockchain handles state and trust, the databases handle data storage, the OAuth2 server handles identity, and NGINX ties it all together into one coherent service endpoint for users. This layered approach (sometimes called the **multi-tier architecture**) is critical for complex systems like e-voting, because it allows each component to be optimized and scaled independently (for example, scaling web servers vs. scaling blockchain nodes) and adds defense in depth from a security perspective.

2.6 AI-Based Identity Verification: Tesseract OCR and Amazon Rekognition in Blockchain Systems

A secure e-voting system must not only protect votes but also ensure that **each vote corresponds to a real, verified voter**. Similarly, any blockchain-based identity management solution needs a reliable way to link digital identities to real-world identities. This is where **identity verification technologies** come into play. Two such technologies are **Tesseract** and **Amazon Rekognition**, which can be used to verify documents and faces, respectively. Tesseract is an open-source Optical Character Recognition (OCR) engine that extracts text from images (for example, reading the name and ID number from a photo of a driver’s license)​ [49] [research.google.com](https://research.google.com/pubs/archive/33418.pdf#:~:text=1,and%20algorithms%20can%20be%20revealed) [50] [research.google.com](https://research.google.com/pubs/archive/33418.pdf#:~:text=released%20Tesseract%20for%20open%20source,Tesseract). Amazon Rekognition is a cloud-based image analysis service that, among other capabilities, can detect and compare human faces in images​ [51] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=Amazon%20Rekognition%20makes%20it%20easy,With%20a) [52] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=detect,hat%2C%20and%20confirming%20that%20the). By integrating these tools into a blockchain workflow, we can automate the verification of voter identities or customer identities (in a Know-Your-Customer scenario) and then record the verification results on the blockchain for audit and trust.

**Tesseract OCR for Document Digitization:** Tesseract has a long history, originally developed by Hewlett-Packard in the 1980s and later released as open source with support from Google​ [53] [research.google.com](https://research.google.com/pubs/archive/33418.pdf#:~:text=1,and%20algorithms%20can%20be%20revealed). It is capable of recognizing printed text in many languages and even basic forms of handwriting. In our system, Tesseract can be used to process images of identification documents. For instance, when a user registers to vote, they might upload a scan or photo of their government-issued ID. Tesseract can automatically extract the textual fields from this ID – such as the person’s name, date of birth, and ID number​ [54] [ocrsolutions.com](https://ocrsolutions.com/decentralized-kyc-how-blockchain-and-ocr-revolutionize-identity-verification/?srsltid=AfmBOoq7ODXzLS9HiqadCjAP6Rrzj9garXT8pL8ryB9ggmzLDeQtSEX3#:~:text=Decentralized%20KYC%20blockchain%20and%20OCR,how%20industries%20approach%20client%2Fcustomer%20identification). These extracted details can then be checked against the information the user provided or against authoritative databases (e.g., checking that the ID number is valid and matches the name in a government database). By automating this data extraction, we greatly speed up the identity verification process and reduce manual data entry errors.

Once extracted, the data can be fed into the blockchain system in a couple of ways. One approach is to store a cryptographic hash of the ID’s textual data or even the entire image on the blockchain as part of the user’s identity record. This provides a verifiable link – later on, if needed, one can prove that the document presented is the same one that was originally verified by recomputing the hash and comparing it with the on-chain value. The actual text data extracted (name, etc.) might be stored off-chain in a database (MySQL) for quick lookup, with the blockchain storing just a reference or confirmation. Alternatively, if privacy regulations allow, some of the extracted data (like an official ID number or a yes/no flag for verification) could be recorded in a smart contract representing the user’s identity status. For example, a smart contract could map a voter’s unique ID to a boolean that indicates “identity verified” or “not verified.” Tesseract’s role is primarily in the initial registration phase – turning physical identity evidence into digital data that can then be reasoned about by our system.

Tesseract’s accuracy has improved greatly, especially after integration of LSTM neural network models in recent versions, making it suitable for well-formatted documents like passports and ID cards. Nonetheless, OCR is not perfect – there can be errors due to image quality or uncommon fonts. Our system can account for this by implementing confidence thresholds and manual fallback: if Tesseract is not very confident in the result (it does provide confidence scores per word), we can flag that registration for manual review by an official. But in most cases, a clear photo of a government-issued ID yields highly accurate text extraction. By eliminating manual data transcription, we both speed up verification and remove a potential human point of failure or collusion.

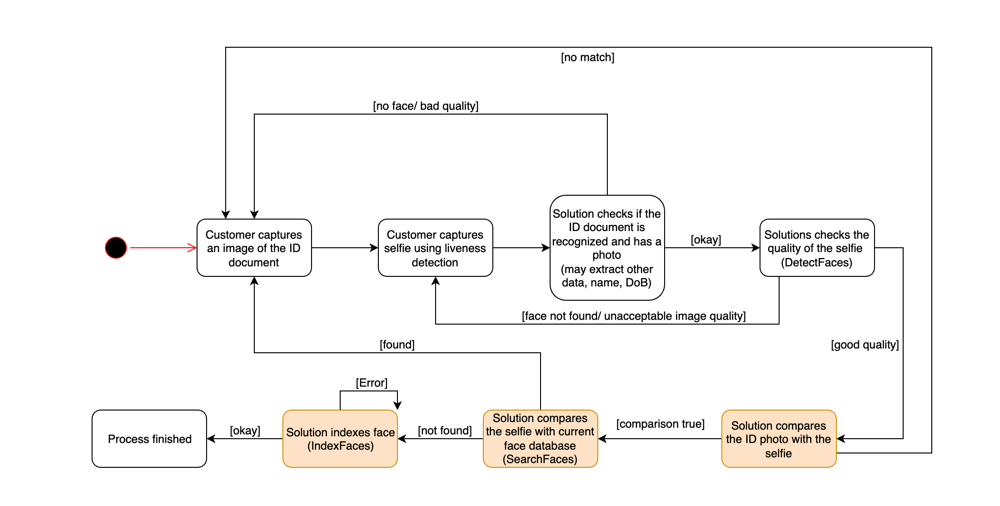
**Amazon Rekognition for Face Verification:** Amazon Rekognition provides a suite of computer vision capabilities, but the most relevant to identity verification are its face detection and face comparison features​ [55] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=match%20at%20L281%20detect,hat%2C%20and%20confirming%20that%20the) [56] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=Amazon%20Rekognition%20makes%20it%20easy,With%20a). Face detection will find and extract face regions from an image (and can assess attributes like if the face is frontal, if the eyes are open, if the image quality is sufficient). Face comparison can take two face images and return a similarity score – essentially, how likely it is that they are the same person. In the context of verifying someone’s identity, a common approach is to perform a liveness check and face match: The user is asked to take a selfie (possibly a selfie video for liveness, meaning the person is present and not just a static photo) and also submit a photo of their ID card. Rekognition can compare the face in the selfie with the face on the ID card. If the faces match with high confidence​ [57] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=1,database%20of%20existing%20user%20faces) [58] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=2,database%20of%20existing%20user%20faces), it gives strong evidence that the person submitting the ID is indeed the person pictured on that ID. Rekognition also has liveness detection features (recently added) to ensure the selfie is of a live person and not a photograph or screen replay​ [59] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=1,database%20of%20existing%20user%20faces).

In our integrated system, this would work as follows: a user, say a new voter, uses our application to take a photo of their ID and a selfie video. The application sends these images to the Rekognition API. Rekognition first checks the selfie for quality and liveness (for example, it might require the user to blink or turn their head, and it can detect if the image is a flat photo). It detects the face on the ID card and the face in the selfie and computes a similarity score. If the score is above a chosen threshold (e.g., 99% confidence)​ [60] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=match%20at%20L287%20searches%20for,If%20the), and liveness is confirmed, our system can mark the user as verified. This entire process might take only a few seconds. Behind the scenes, the user’s data (like their name from OCR, plus this face match confirmation) can be assembled, and a transaction can be sent to the blockchain recording “User X has been verified at time Y by method Z”. On a permissioned blockchain, this transaction could be issued by an identity verifier component (perhaps run by the election authority) and co-signed by the verifying service’s key. Storing the verification result on-chain ensures that it cannot be altered or deleted later – essential for audit trails. If later someone claims a fraudulent registration, we have the on-chain evidence of what was verified (including potentially references to the images or their hashes).

Rekognition’s use can extend beyond initial verification. For example, during voting, one could use facial recognition for login – a voter might log in by taking a selfie and the system checks that it matches the face from their registration record. This would be a form of biometric authentication. It’s powerful but also raises privacy concerns. A balance must be struck between security and privacy; in a government-run e-voting system, biometric verification might only be used at registration to build trust in the voter’s digital identity, and not for every login (instead, using passwords or tokens for routine login).

It is also notable that Rekognition can perform identification against a database of faces. If we stored all verified voters’ face encodings in an Rekognition Collection, the system could detect duplicate registrations by checking if the same face appears already under a different name (to prevent someone from registering twice with different IDs)​ [61] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=the%20user%20face,database%20of%20existing%20user%20faces) [62] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=,You%20can%20use). This is a powerful feature to enforce one-person-one-account. However, maintaining such biometric databases must be handled carefully and in compliance with regulations.

**Integration with Blockchain:** After Tesseract and Rekognition have done their jobs (extracting data and validating the person), the blockchain can be updated with a verifiable claim. For instance, a smart contract for identity could have a function VerifyIdentity(userID, attributes, verifier) that is called by an authorized verifier agent when someone is verified. The contract could emit an event or simply store a flag. Later, the voting smart contract can require that flag to be true for a given user before accepting a vote cast from that user. This links the off-chain verification with on-chain permission. In a more decentralized scenario, one could use the concept of a **self-sovereign identity**: issue the user a cryptographic credential (perhaps a signed JSON token or a DID Document) that attests to their verified attributes, and store a hash or registry of such credentials on-chain. The user then proves possession of that credential when voting (this edges into advanced territory of decentralized identity (DID) and verifiable credentials, which are complementary standards that could be layered on our system). Regardless of the approach, the contribution of OCR and face recognition is to provide a high-assurance input that the blockchain alone cannot provide – linking a blockchain address or identity to a flesh-and-blood human and their official documents.

Figure 2.1: An example identity verification workflow integrating OCR and face recognition.

First, a user submits an image of their ID document and a selfie with liveness detection. The system (solution) checks the quality of the images and uses OCR to extract text from the ID and face recognition to compare the selfie with the ID’s photo. If the document data is extracted successfully and the face matches with high confidence, the user’s face is also checked against existing records to prevent duplicates. Upon success (“okay”), the solution indexes the user’s face in a database (for future reference) and a blockchain transaction is created to mark the user as verified. If any step fails (e.g., no face found, poor image quality, or face mismatch), the process is halted with an error, and the user may be asked to retry or provide alternate verification.​ [63] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=The%20following%20figure%20shows%20a,steps%20in%20this%20process%20are) [64] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=The%20following%20reference%20architecture%20shows,services%2C%20to%20implement%20identity%20verification)

The figure above illustrates the decision flow: starting from capturing images, through checks and comparisons, ending in either a verified identity or an error. This kind of flow can be implemented in a combination of front-end (for capturing images), back-end (calling Tesseract and Rekognition, doing checks), and blockchain (recording the result). Amazon Rekognition’s API calls used in such a flow include DetectFaces (for quality and attributes), CompareFaces (for matching selfie to ID photo)​ [65] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=detect,hat%2C%20and%20confirming%20that%20the), and optionally SearchFacesByImage (to search the face in a collection of known faces to avoid duplicate enrollments)​ [66] [aws.amazon.com](https://aws.amazon.com/blogs/machine-learning/identity-verification-using-amazon-rekognition/#:~:text=searches%20for%20an%20existing%20face,If%20the). Tesseract would run on the ID image to get text fields which the system might validate (for example, checking the ID’s expiration date is in the future).

**Security and Privacy Considerations:** It’s worth noting that handling images and biometric data requires strong protections. Communications with Rekognition (a cloud service) must be encrypted and comply with data protection laws. One might not want to send raw ID images to the cloud in some jurisdictions; an alternative is to use an on-premise or open-source face recognition solution (there are libraries like OpenCV or FaceNet that could be used, though Rekognition offers a pre-trained convenient solution). Assuming Rekognition is acceptable, AWS assures that images can be not stored after processing if we don’t explicitly save them (though they might keep some metadata). In our design, we should delete or secure these images immediately after use. The blockchain should also avoid storing any image or biometric directly (we store only hashes or flags) to protect user privacy and minimize the sensitive data on-chain (which is typically immutable and public/long-lived).

**Benefits to E-voting:** By incorporating Tesseract and Rekognition, we automate what would otherwise be a very labor-intensive process of verifying voter identities. In traditional systems, this might require people to show up in person or officials to manually inspect documents. Automation not only scales better (imagine verifying millions of voters within days using cloud compute power) but can potentially reduce human bias or error. It also enhances accessibility – people could register from home by taking pictures with a smartphone. Once verified, those users obtain a trusted digital identity that the blockchain recognizes, enabling online voting with confidence that each voter is unique and legitimate. This synergy of AI and blockchain essentially creates a **trusted bridge between the physical and digital realms**: AI verifies the physical identity, and blockchain provides the digital trust and record-keeping.

Beyond e-voting, this combination is applicable in blockchain-based **KYC (Know Your Customer)** processes for finance, **self-sovereign identity** frameworks, and digital onboarding for services. Many startups and enterprises are exploring storing verified credentials on blockchains such that once a user is verified by one institution, another can rely on that verification. Tools like Tesseract and Rekognition are often behind that initial verification step, after which a blockchain token or certificate can represent the verified status. Our system, by design, can issue such a token on-chain (for example, a voter NFT or simply a verifiable claim entry) once Rekognition/Tesseract checks pass and an authorized authority confirms it.