Blockchain Based E-Tendering System with Enhanced Encryption

by

Shurfa Maliha Lorin (201014034)
Md. Mahamudul Hassan Shisir (201014011)
Md. Saiful Islam (202014003)

Project/Thesis report (CSE499) submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science in Computer Science and Engineering

Under the supervision of

Mohammed Ashikur Rahman

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIVERSITY OF LIBERAL ARTS BANGLADESH

SUMMER 2024

© *Shurfa Maliha Lorin , Md. Mahamudul Hassan Shisir , Md. Saiful Islam*
All rights reserved

DECLARATION

Project Title	**Blockchain Based E-Tendering System with Enhanced Encryption**
Authors	*Shurfa Maliha Lorin ,Md. Mahamudul Hassan Shisir,Md. Saiful Islam*
Student IDs	201014034, 201014011, 202014003
Supervisor	Mohammed Ashikur Rahman

We declare that this project/thesis report entitled *Blockchain-Based E-Tendering System with Enhanced Encryption* is the result of our own work except as cited in the references. The project/thesis report has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Shurfa Maliha Lorin
201014034

Department of Computer Science and Engineering
University of Liberal Arts Bangladesh

Md. Mahamudul Hassan Shisir
201014011

Department of Computer Science and Engineering
University of Liberal Arts Bangladesh

Md. Saiful Islam
202014003

Department of Computer Science and Engineering
University of Liberal Arts Bangladesh

Date: October 14, 2024

Department of Computer Science and Engineering
University of Liberal Arts Bangladesh
Mohammadpur, Dhaka - 1207

CERTIFICATE

This is to certify that the project/thesis report entitled **Blockchain Based E-Tendering System with Enhanced Encryption** , submitted by **Shurfa Maliha Lorin (ID :01014034),Md. Mahamudul Hassan Shisir (ID : 201014011) , Md. Saiful Islam (ID: 202014003)** are undergraduate students of the **Department of Computer Science and Engineering** has been examined. Upon recommendation by the examination committee, we hereby accord our approval of it as the presented work and submitted report fulfill the requirements for its acceptance in partial fulfillment for the degree of *Bachelor of Science in Computer Science and Engineering*.

Mohammed Ashikur Rahman, Assistant Professor
Department of Computer Science and Engineering,
University of Liberal Arts Bangladesh

Muhammad Golam Kibria, Professor
Department of Computer Science and Engineering, University of Liberal Arts Bangladesh

Nafees Mansoor, Associate Professor
Department of Computer Science and Engineering, University of Liberal Arts Bangladesh

Rubaiya Hafiz, Senior Lecturer
Department of Computer Science and Engineering, University of Liberal Arts Bangladesh

Place: Dhaka
Date: October 14, 2024

ACKNOWLEDGEMENTS

We would like to express our deep and sincere gratitude to our research supervisor, *Mohammed Ashikur Rahman* ,for giving us the opportunity to conduct research and providing invaluable guidance throughout this work. His dynamism, vision, sincerity and motivation have deeply inspired us. He has taught us the methodology to carry out the work and to present the works as clearly as possible. It was a great privilege and honor to work and study under his guidance.

We are greatly indebted to our honorable teachers of the Department of Computer Science and Engineering at the University of Liberal Arts Bangladesh who taught us during the course of our study. Without any doubt, their teaching and guidance have completely transformed us to the persons that we are today.

We are extremely thankful to our parents for their unconditional love, endless prayers and caring, and immense sacrifices for educating and preparing us for our future. We would like to say thanks to our friends and relatives for their kind support and care.

Finally, we would like to thank all the people who have supported us to complete the project work directly or indirectly.

Shurfa Maliha Lorin, Md. Mahamudul Hassan Shisir, Md. Saiful Islam
University of Liberal Arts Bangladesh
Date: October 14, 2024

Dedicated to
my loving Mother

– *Shurfa Maliha Lorin*

Dedicated to
my loving Mother

– *Md. Mahamudul Hassan Shisir*

Dedicated to
my loving Mother

– *Md. Saiful Islam*

ABSTRACT

Companies and governments frequently use the tendering process to acquire goods and services from suppliers or manufacturers. Traditionally, this procedure is susceptible to several problems, including favoritism, corruption, lack of openness, and data tampering. A possible solution for these issues is the incorporation of blockchain technology. The goal of this project is to create a decentralized, blockchain-based e-tendering system with Ethereum smart contracts. The project's key components are the creation and publication of the tender, the submission of bids, and the selection of the winners. To promote fair and open selection, smart contracts automate crucial procedures like bid evaluation and winner selection based on predetermined criteria. Although blockchain improves security, smart contract vulnerabilities and brute force attacks can still occur. To reduce these risks, countermeasures are used, such as the SHA-256 hashing technique and ChaCha20 encryption. Bid documents are protected by ChaCha20 encryption, and each bid and document is securely stored on the blockchain using SHA-256, which generates an unchangeable hash of both. Using blockchain's built-in features, this project seeks to promote trust, reduce corruption, and expedite the entire e-tendering process. A system that greatly improves the security, openness, and efficiency of tendering operations is the anticipated result.

Keywords: Blockchain, Smart contract, Ethereum, SHA-256 Hashing, Chacha20 Encryption

Contents

1 Introduction **1**

1.1 Background Literature 1

1.2 Problem statement 2

1.3 Objectives and Significance of the Study 3

1.3.1 Objective 3

1.3.2 Motivation 3

1.3.3 Significance 4

1.4 Limitations 4

2 Literature Review **5**

2.1 Literature Review 5

2.2 Performance Evaluation Criterion 11

2.3 Justification of the Adopted Performance Evaluation Criterion 13

3 Design and Development **14**

3.1 Project Management and Financial Activities 14

3.1.1 Financial Activities : 15

3.1.2 Design Concepts and Major Subsystems : 16

3.1.3 Engineering Analysis of the Design Process : 18

3.2 Adopted Tools and techniques 23

3.3 Design the Solution 25

3.4 SDLC 28

3.5 Implementation 31

3.6 Evaluate the solution : 35

4 Result analysis **38**

4.1 Investigate the final result 38

5 Conclusions **51**

5.1 Societal Impacts and Mitigation Strategies for E-Tendering System 51

5.2 Impact on Environment and Sustainability 52

5.3 Ethical and professional principles 52

5.4 Brief Summary 53

5.5 Future works 54

References **56**

List of Figures

3.1 Timeline of Blockchain Based E-Tendering System 14
3.2 Flow Diagram of Blockchain Based E-Tendering System 17
3.3 Encryption(Chacha20) 20
3.4 Hashing(SHA-256) 21
3.5 System Design of e-Tendering System 27
3.6 Methodology 30

4.1 The deployment of Smart Contract 39
4.2 Smart Contract deployment details on Etherscan 39
4.3 Creating Tenders on Smart Contracts 40
4.4 Etherscan shows Details of Tender Creation 41
4.5 Bid Submission by Bidder 1 41
4.6 Etherscan shows Bid Submission Details 42
4.7 Bid Submission by Bidder 2 42
4.8 Etherscan shows Bid Submission Details 43
4.9 View Active Tenders by Admin and Bidder 43
4.10 View All Bids by Admin and Bidder 44
4.11 View Winner And Bid Amount by Admin and Bidder 44
4.12 Administrator Login 45
4.13 Admin Panel 45
4.14 Admin creates the tender. 45
4.15 Signing Up as a Bidder 46
4.16 Login as Bidder 46
4.17 Bidder Panel 46
4.18 Bidder submit Bid 47
4.19 Selection of Document Hash 47
4.20 Hash Generated 48

4.21 Selection of File Encryption 48
4.22 Encryption of Documents 49
4.23 Documents Encrypted 49
4.24 Decryption of Documents 50
4.25 File Decrypted 50

List of Tables

2.1 Comparative Analysis (General Parameters) 9
2.2 Attacks and Countermeasures Using Blockchain Technology 10
2.3 Comparison of SHA-256 and ChaCha20 in Blockchain-based E-tendering Systems 11

3.1 Cost Analysis of an E-tendering Project Based on Blockchain 15

Chapter 1

Introduction

1.1 Background Literature

The public and private sectors use the bidding process as a basic way to acquire products and services, but it is vulnerable to several security and transparency problems. Conventional tendering methods frequently include vulnerabilities that result in corruption and inefficiency, including contractor favoritism, document tampering, bid manipulation, and a lack of transparency. To address these issues, blockchain technology which is decentralized and unchangeable has been suggested as a way to develop transparent and safe e-tendering systems. It ensures integrity and security by securely storing every bid and making data manipulation nearly impossible.

The benefits of blockchain in tendering processes have been emphasized in several areas of research. For instance, the Ethereum blockchain's smart contracts can automate and safeguard the bidding process, guaranteeing that all proposals are fairly assessed and free from the potential for data manipulation Mali et al. (2020). Their efforts strengthen the procurement process's overall integrity by highlighting the fair evaluation of all bids free from the potential for data manipulation. Similar to this, Pawar et al. (2023) investigates how blockchain technology can be used in the bidding process, outlining a methodical procedure that comprises creating the tender, submitting bids, evaluating them, and choosing the best offers. By offering a safe and open framework, this study demonstrates how blockchain can effectively address vulnerabilities in conventional tendering approaches.

To enhance the security of sensitive data, advanced encryption techniques must be integrated with blockchain technology.Marati et al. (2023) proposes utilizing advanced encryption techniques like SHA-256 to improve the security of blockchain-based tender systems to protect sensitive data.Alves Batista (2024) emphasizes that tendering systems can use encryption to protect sensitive documents. This maintains confidentiality and integrity throughout the tendering process by limiting access to and viewing of the data to authorized personnel.

According to a study by Hiremath & V (2023), blockchain technology can eliminate traditional paper-based tendering processes, which reduces corruption and provides every bid an equal chance of winning. The study also emphasizes how blockchain technology may be used more widely in other public services, emphasizing how this strategy might improve efficiency, security, and openness in a range of government sectors.

However, this study aims to examine how blockchain technology can be integrated with advanced encryption techniques like SHA-256 and ChaCha20 to improve the tendering process' security and transparency.We demonstrate how the integration of these technologies can make procurement procedures more efficient and trustworthy while maintaining the integrity of tendering systems.

1.2 Problem statement

There are significant vulnerabilities in the current e-tendering system that may facilitate data manipulation and compromise sensitive data because of security gaps. It is essential to use a blockchain-based e-tendering technology to solve this. Although some vulnerabilities are mitigated by using a blockchain-based infrastructure, persistent security threats still exist. To solve these, a proposal that emphasizes the strong implementation of ChaCha20 encryption and the SHA-256 hash function ensures greater transparency and accountability throughout the procurement process.

1.3 Objectives and Significance of the Study

This section describes the objectives and the motivations behind our project, which combines blockchain technology with advanced encryption techniques like SHA-256 and ChaCha20 in an e-tendering system. We will discuss the primary objectives of the study, specify particular goals intended to accomplish these aims and describe the reasons behind our investigation of safe and open tendering procedures. We will also talk about the importance of this research in addressing the shortcomings of traditional tendering systems and its broader implications for improving procurement activities' security, transparency, and trust.

1.3.1 Objective

- To develop a blockchain-based e-tendering system using the SHA-256 Hashing and advanced encryption technique.
- To provide an effective procurement platform.

1.3.2 Motivation

This study is motivated by the shortcomings of conventional tendering methods, which are frequently vulnerable to fraud, data manipulation, and a lack of transparency. The integrity of the tendering process has been compromised by these problems, which could result in unfair practices and corruption. Furthermore, it's critical to understand that blockchain technology is susceptible to several security risks, such as assaults that compromise data integrity, confidentiality, and the process's general credibility, as it becomes more widely used for e-tendering. A more secure and transparent system can be established by combining blockchain technology with SHA-256 hashing and ChaCha20 encryption to reduce these issues. The decentralized and immutable ledger of blockchain technology improves transparency, and hashing and encryption methods shield private data from manipulation and unwanted access. By addressing the vulnerabilities in blockchain e-tendering systems, these technologies can provide a safer and more reliable way to manage bids and tenders, according to this study.

1.3.3 Significance

The significance of this study consists of its full address to the security and transparency issues facing traditional tendering systems. Blockchain technology combined with advanced encryption methods solves issues of data manipulation, illegal access, and opaqueness. Making use of an immutable blockchain ledger ensures that every transaction is documented so that all parties involved can view the same data, thus reducing the likelihood of fraud and corruption. Because participants may be convinced that the tendering process is impartial and fair, this transparency helps to build confidence. The decentralized and unchangeable characteristics of blockchain technology, along with the strong security offered by SHA-256 and ChaCha20, guarantee that tendering procedures are reliable, transparent, and safe. The efficiency and integrity of procurement operations are enhanced by this method, which also has the potential to completely transform the way tenders are handled in a variety of industries.

1.4 Limitations

There are various barriers to the use of blockchain-based tendering systems. Blockchain infrastructure accessibility and the technical expertise required to run such systems efficiently are examples of technological barriers. Decentralized tendering procedures must be in line with current legal and compliance frameworks, which present regulatory issues. Cost issues are also important because the initial setup and ongoing operating costs of blockchain systems may be higher than those of conventional techniques. Finally, a lack of knowledge of blockchain technology and its perceived complexity may cause stakeholders used to traditional tendering procedures to oppose the move, resulting in user adoption resistance. All of these elements work together to affect blockchain's viability and adoption in e-tendering platforms.

Chapter 2

Literature Review

2.1 Literature Review

Research on integrating blockchain technology into tendering procedures has been expanding because it may be able to solve problems with efficiency, security, and transparency that come with using traditional tendering methods. Several studies and technical papers have examined how blockchain technology and advanced cryptography can revolutionize procurement processes.

Mali et al. (2020) introduces a blockchain-based e-tendering system that uses smart contracts to decentralize tender administration. Their research highlights the security and transparency benefits of blockchain technology, which records every transaction on an immutable ledger. Automating tender procedures with smart contracts significantly reduces delays and human error. This study is in support of using blockchain technology as the basis for more transparent procurement practices.

Another study Hiremath & V (2023) discusses how blockchain-based solutions might improve e-tendering's effectiveness by eliminating traditional paper-based procedures, which frequently result in corruption and inefficiency. Distributed ledger technology, which enables parties to monitor and confirm the authenticity of bids and tender papers, is essential for safe, tamper-proof transactions, according to the authors. The findings of Mali et al. (2020) are consistent with this, supporting the idea that blockchain can revolutionize procurement by promoting an open and equitable tendering process.

Similarly, research by Marati et al. (2023) discusses the use of blockchain technology in tender management to solve security and transparency concerns in conventional tendering procedures. All transactions and data, including proposals and company profiles, are safely stored in an unchangeable ledger due to their blockchain-based system. They emphasize increases in transparency and equity through the use of cryptographic techniques and the decentralization of tender data. But they also warn that to keep security, continuous audits and updates are required. To further promote the security of the system, they recommend that future work improve the security of smart contracts by encrypting private data using more advanced cryptographic algorithms like SHA-256.

H. Li & Xue (2021) suggests a blockchain-integrated electronic bidding system for the construction sector that will enhance security, trust, and transparency. A decentralized strategy is used in the system architecture to mitigate the problems of identity verification, bidding fraud, and data manipulation. To ensure fairness throughout the auction process, the study also uses big data analytics to identify and stop bidding fraud. Their research shows how blockchain technology may revolutionize electronic bidding by boosting dependability and efficiency.

A blockchain-based solution is presented by Reddy et al. (2023) to address important problems with conventional e-tendering, including security risks, contractor favoritism, and transparency concerns. To increase the security and accountability of the tendering procedure, the study makes use of blockchain's decentralized ledger to guarantee that every transaction is safely documented and unchangeable. Bid submissions and evaluations are among the critical processes that smart contracts are utilized to automate. This minimizes human participation, minimizes errors, and speeds up the process as a whole. The system suggests cryptographic encryption techniques to protect sensitive data in addition to the blockchain's built-in security measures, which strengthens the system's resistance to tampering and illegal access. This blockchain-based approach makes the tendering process more effective and equitable by removing middlemen and offering real-time transparency, which raises procurement accountability and trust.

Alves Batista (2024) investigates how blockchain technology might improve accountability and transparency in public procurement, with a particular emphasis on reducing records fraud. The study suggests using blockchain technology to stop record tampering and errors and emphasizes the significance of appropriate records management for procurement integrity. It shows how blockchain can guarantee the security and authenticity of procurement records, lowering fraud opportunities and boosting confidence in public procurement systems, by analyzing international regulations like those from the World Bank and the European Union.

Additionally, research by Pawar et al. (2023) examines how blockchain technology can be used in the tendering process, outlining a methodical procedure that involves creating the tender, bidding on it, and choosing the winning bids. Their research highlights how blockchain technology can offer a transparent and safe framework that increases stakeholder trust and reduces the hazards connected to centralized databases. This thorough approach presents blockchain as a workable alternative while addressing the security concerns of the current e-tendering platforms.

Dubey et al. (2023) proposes a blockchain-based approach to government tender distribution in order to secure and decentralize the procedure. They highlight how blockchain technology ensures security and transparency through an immutable record, whereas traditional centralized systems are susceptible to fraud and hacks. The solution reduces human interference and potential bias by automating bid screening and contract awarding with Ethereum smart contracts. In order to ensure a just and open allocation of contracts, the authors explicitly stress the importance of anonymity and trustless interactions throughout the tendering process.

Kuntamukala & Raju (2024) introduce a smart tender management system based on blockchain technology to fix the problems with current e-tendering systems. They address the problems with centralized databases, like how they can be used for fraud, changing info, and not being open to everyone. When you use blockchain, you build a decentralized, encrypted transaction system that keeps bids and tender documents safe. This approach makes sure that everything is clear by letting bidders watch what's going on in the portal. It also gets rid of centralized control, which makes the tendering process more secure as well as fair.

Bikkad (2023) propose a web-based tender management system that uses blockchain to make buying more open and safe. They stress that the immutability of blockchain and smart contracts could make the bidding process faster, get rid of middlemen, and automate the review of bids. Their method is meant to help stakeholders work together in real time, make things easier to track, and lower the risk of fraud and manipulation. The authors conclude that blockchain could change the procurement business by making it more efficient, accountable, and cost-effective.

A blockchain-based e-tendering evaluation system that makes tender processes safer for data is presented by Gong et al. (2022). Their approach keeps immutable records of bids and reviews using smart contracts and a transaction data model. This architecture ensures transparency and prevents data manipulation when tender papers are stored on the blockchain. To demonstrate how well the system functions and how the blockchain-based approach can be a more secure and efficient alternative to traditional methods of evaluating bids, they use experimental scenarios.

Gaikwad et al. (2023) presents a blockchain-based approach to make digital bidding safer and more open. Smart contracts on the Ethereum blockchain are used in the suggested system to make sure that e-tendering is open, decentralized, and safe. Their study shows how cryptographic techniques can be used to encrypt data and check its integrity. The study reveals that blockchain can get rid of security risks and make the whole bidding process more efficient by fixing vulnerabilities in current systems.

Shelar & Chandratre (2022) explores blockchain technology used in an online auction system to fix the security problems that come with regulated platforms. With blockchain and smart contracts powering their system, they make sure that sales information can't be changed and can only be accessed by authorized parties. This makes things much more secure and apparent. Without intermediaries, their model automates the buying process, which makes auctions more efficient and reliable.

Multiple study papers from 2020 to 2023 were compared in order to help with the evaluation of blockchain-based e-tendering systems.

Authors Year	Mali et al 2020	Hiremath et al 2023	Marati et al 2023	Dubey et al 2023	Li & Xue 2021	Halikar et al 2023	Thilak 2022	Kiyeng 2021	Pawar 2023	Ashwini 2021	Gong et al 2022
Security	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
Transparency	yes	yes	yes	yes	-	yes	yes	yes	yes	yes	yes
Efficiency	-	yes	yes	yes	yes	-	yes	-	-	-	-
Confidentiality	-	-	yes	-	-	-	-	-	yes	-	-
Immutability	yes	yes	yes	-	yes	yes	yes	yes	yes	yes	yes
Auditability	yes	-	-	-	yes	-	yes	yes	yes	-	-
Fairness	yes	yes	yes	yes	yes	yes	yes	yes	yes	-	-
Accountability	-	yes	-	-	-	-	-	-	-	-	-
Accessibility	-	yes	-	-	-	-	-	-	-	-	yes

Table 2.1: Comparative Analysis (General Parameters)

This comparison focuses on important factors like security, openness, effectiveness, privacy, inflexibility, auditability, equality, responsibility, and ease of access. The research shows that blockchain-based systems always support security, openness, and immutability, which are important for e-tendering systems to make sure they are trustworthy and data is kept safe. But there are some differences in terms of auditability, privacy, and efficiency. This shows that, based on the needs of the system, more improvements may need to be made in these areas. Because of the benefits this research shows, blockchain should be used to make tender processes more fair and safe.

Even though the blockchain-based e-tendering system has a strong decentralized design, there are still many methods of attack that need to be thought through to make sure it is safe and reliable. Cyberattacks like brute force, phishing, DDoS, Sybil, and even smart contract flaws can still happen to blockchain systems. These attacks can damage the system's integrity and effectiveness if they are not stopped properly.

To protect the system from possible threats, it is important to take steps such as using strong encryption algorithms, stricter authentication protocols, smart contracts that have been tried, and decentralized validation mechanisms. The blockchain-based e-tendering system can provide a higher level of security by using

Authors	Types of Attacks	CounterMeasures
Jonkman et al., 2019	Brute force attack	Strong encryption and a robust algorithm Dasgupta et al. (2019)
Dasgupta et al., 2018	Phishing Attack	Stronger authentication and strong passwords Dasgupta et al. (2019)
Konig et al., 2020	Smart Contract Vulnerability	Use trustworthy and well-established smart contract platforms and only deploy well-tested contracts. König et al. (2020)
Averin & Averina, 2019	DDoS attack	Utilizing distributed infrastructure and expanding network capacity Begum et al. (2020)
Rahman et al., 2019	Sybil attack	Stronger authentication and decentralized node validation Averin & Averina (2019)
Embong et al., 2021	Eclipse attack	Network encryption make use of reputable, well-maintained blockchains with robust network resiliency. Dasgupta et al. (2019)

Table 2.2: Attacks and Countermeasures Using Blockchain Technology

these security measures. This lowers the risk of fraud and makes sure the platform is safe and reliable.

Adding advanced cryptographic methods like SHA-256 and ChaCha20 encryption to blockchain-based e-tendering systems is an excellent way to make them secure and protect against attacks like phishing, brute force, and DDoS.A well-known cryptographic hashing algorithm called SHA-256 is very important for keeping data secure in blockchain environments. It gives each input a unique 256-bit hash value, so even the smallest change to the input will lead to a different hash output. On the other hand, ChaCha20 encryption encrypts private data like bid details and documents very quickly. Its light security keeps important data private and out of the hands of people who aren't supposed to have access to it.

Features	SHA-256 Hashing	ChaCha20 Encryption
Role in E-tendering using blockchain	It can be used in e-tendering systems to hash contracts and bid amounts, creating a safe and immutable record of the bidding procedure. Mali et al. (2020),Marati et al. (2023).	Sensitive information, including bid details and bid documents, can be encrypted using ChaCha20 in e-tendering. De Santis et al. (2017),Nir & Langley (2015).
Security Enhancement	The e-tendering system's security is improved by SHA-256's resilience to collisions and attacks, which offers an efficient way of confirming the accuracy of data that has been stored.Mali et al. (2020),Marati et al. (2023).	ChaCha20's robust encryption features help to keep sensitive data private and prevent unwanted parties from accessing it, which enhances the e-tendering system's security. De Santis et al. (2017),Nir & Langley (2015).

Table 2.3: Comparison of SHA-256 and ChaCha20 in Blockchain-based E-tendering Systems

2.2 Performance Evaluation Criterion

Efficiency and security are critical in a blockchain-based e-tendering system to guarantee that bids and tender documents are handled promptly, remain confidential and are immutable. When assessing the overall performance of the system, the effectiveness of the hashing and encryption algorithms is critical. Detailed performance evaluation criteria with an emphasis on security, hashing efficiency, and encryption are listed below.

Encryption and Hashing Efficiency : In order to ensure data security and integrity, this criterion evaluates the computational overhead, memory consumption, key management, and time efficiency associated with document encryption and hash value generation.

ChaCha20 stream cipher is appropriate for contexts with limited performance since it is quicker and more effective than other encryption, especially on devices without hardware acceleration. As a continuous data processing stream cipher, it provides strong security and resistance to timing attacks with a more straightforward and secure implementation.

Key Size: A 256-bit (32-byte) secret key is used by ChaCha20 encryption to provide robust protection against attacks.

File Size: Tender documents, which are 3 kB files, are encrypted.

Time Taken: Using ChaCha20, the system encrypts a 3 kB document in 9 milliseconds.

Time Complexity: The time complexity of ChaCha20 is O(n), where n is the number of 64-byte blocks (48 blocks for a 3 kB file). This guarantees steady scaling for bigger files. Decryption Time: The 3 kB document takes 8 milliseconds for the system to decrypt.

Hashing Performance (SHA-256) :

Hash Size: By producing a fixed 256-bit (32-byte) hash value, SHA-256 guarantees the integrity of the file regardless of its size.

File Size: The 3 kB file is used in the hashing procedure.

Time Taken: Small files, such as 3 kB, take SHA-256 1 microsecond to compute a hash.

Time Complexity: O(n), where n is the number of 512-bit (64-byte) blocks—48 blocks for a 3 kB file—is the complexity of SHA-256 hashing.

Security Evaluation in Terms of Encryption and Hashing :

The blockchain-based E-tendering system uses SHA-256 hashing and ChaCha20 encryption to protect bid papers and bid amounts. A crucial component is the system's ability to resist attacks, since hashing and encryption protect it from many threats on the internet. ChaCha20 encryption offers an effective technique while preserving strong defense against attacks thanks to its quick encryption (9 milliseconds for a 3 kB file) and decryption timings (8 milliseconds). It is more difficult to exploit timing vulnerabilities when encryption periods are shorter because they limit the window of opportunity for side-channel attacks. Furthermore, SHA-256 hashing ensures data integrity by making it simple to identify any illegal changes to bid papers through hash verification.

Key Size: ChaCha20 employs a 256-bit key, which is a lot more secure than algorithms that use smaller keys. It would take exponentially longer to break a larger key, making it more resistant to attacks.

Encryption Strength: ChaCha20 is known for its ability to resist cryptographic attacks such as linear and differential cryptanalysis. It has shown resilience to established assault techniques. Similar to this, SHA-256 offers robust protection against collision attacks, guaranteeing the preservation of bid data integrity and making any attempts at tampering instantly identifiable. The E-tendering system is completely secure because to this combination of encryption and hashing.

2.3 Justification of the Adopted Performance Evaluation Criterion

The blockchain-based E-tendering system's efficiency and security justify the performance evaluation criteria that were specified. Due to its exceptional speed and efficiency—especially in contexts with limited performance—without compromising security, ChaCha20 encryption was chosen. Its linear time complexity (O(n)) guarantees that the system's performance stays scalable and predictable even as the file size grows. A 256-bit key offers a high degree of defense against brute-force attacks, guaranteeing the confidentiality of bid documents during the bidding procedure. In the same way, the hashing mechanism (SHA-256) provides excellent resistance to collision attacks and resilient data integrity with low computational overhead, significantly improving efficiency.

Additionally, it evaluates hashing and encryption/decryption times, addressing important system performance factors including processor speed and resource usage. The evaluation ensures that the system not only meets real-time speed requirements but also provides a secure environment for crucial transactions by placing a strong emphasis on memory usage, attack resistance, and key management. The E-tendering system's comprehensive approach, which maintains a balance between speed, resource efficiency, and security, allows it to handle large bid volumes without compromising data integrity or confidentiality.

Chapter 3

Design and Development

3.1 Project Management and Financial Activities

Planning, organizing, executing out, and monitoring the project from start to finish are all included in project management. It include setting project objectives, determining tasks and deadlines, assigning resources, and making sure the project stays within financial and time limits. Achieving project goals requires team members to collaborate, communicate, and coordinate effectively throughout the project lifecycle.

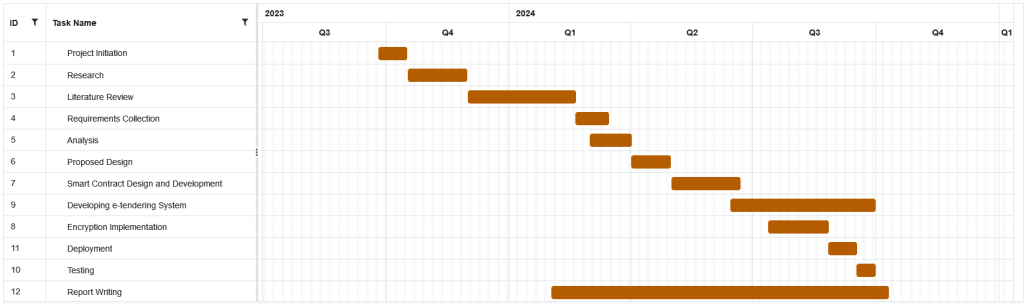


Figure 3.1: Timeline of Blockchain Based E-Tendering System

The blockchain-based e-tendering system's project management process starts with identifying any gaps in current systems that the project could fill and creating a comprehensive project proposal that details the goals, parameters, and project stakeholders. Starting in October 2023 and ending in February 2024, the research and literature review phases follow approval of the project proposal.These phases look

into blockchain technology and security protocols including ChaCha20 encryption and SHA-256 hashing.

Following the research, the requirements collection phase is conducted from February 2023 to March 2024. With a focus on blockchain integration, encryption specifications, and user roles (admin and bidder), the functional and non-functional needs for the system were identified. The design and coding stage, which includes creating smart contracts, starts in March 2024 and lasts until June 2024. This involves utilizing tools like Web3.js, MetaMask, and Remix IDE to integrate the system with blockchain. From June 2024 to July 2024, encryption and security measures are put into place. In the months that follow, deployment and testing take place. For testing, the system uses the Sepolia testnet. Final testing and report writing bring the project to a close, guaranteeing a comprehensive assessment and documentation.

3.1.1 Financial Activities :

This section discusses the overall project development cost. The entire development phase of our project takes roughly 20 weeks or five months. The following is the cost study of an e-tendering system based on blockchain:

Type of Cost	**Description**	**Cost**
Software Cost	Web3.js, Node.js, Express, MetaMask, and the Remix IDE (all open-source)	Tk. 0
Hardware Cost	Three Computers, Three Routers (Personal Technology Used)	Tk. 1,50000
Developer Salary	Five-month developer salary: Tk. 15,000/= per month	Tk. 75,000
Cost of Internet	Team's monthly internet bill (5 months at Tk 1,000 per month)	Tk. 5,000
Testing	Testing environment setup (MetaMask)	Tk. 0
Total Cost		**Tk. 2,30000**

Table 3.1: Cost Analysis of an E-tendering Project Based on Blockchain

The blockchain-based e-tendering system's financial operations, which total Tk. 230,000, are broken down into major cost categories. The project hardware costs are 2,30,000. All software tools, including Web3.js, Node.js, Express, MetaMask, and the Remix IDE, were open-source, so there were no software costs. The developer's compensation, which was Tk. 15,000 a month for five months, was the largest expense, coming to Tk. 75,000. The crew also paid Tk 1,000 a month for internet services, which added up to Tk 5,000 throughout the project. Finally, for simulation and testing, which included establishing a MetaMask testing environment resulting in no costs.

3.1.2 Design Concepts and Major Subsystems :

The blockchain-based e- tendering system offers a decentralized approach for safe, open, and effective tendering procedures. Managing bids securely, maintaining data integrity, and ensuring a fair selection process are the main issues with design. The two primary users of this system are the bidder and the tender organization (admin). Engineering analysis follows a thorough explanation of the design concepts and key subsystems.

Tender Creation Subsystem : The admin (tender organization) can create and publish new tenders with the help of this subsystem. It documents the specifics of the tender, such as the requirements, bid criteria, and deadlines. Active tenders can be seen by bidders and the admin. All tender information will be stored on the Ethereum blockchain via the smart contract, which offers transparency and immutability.

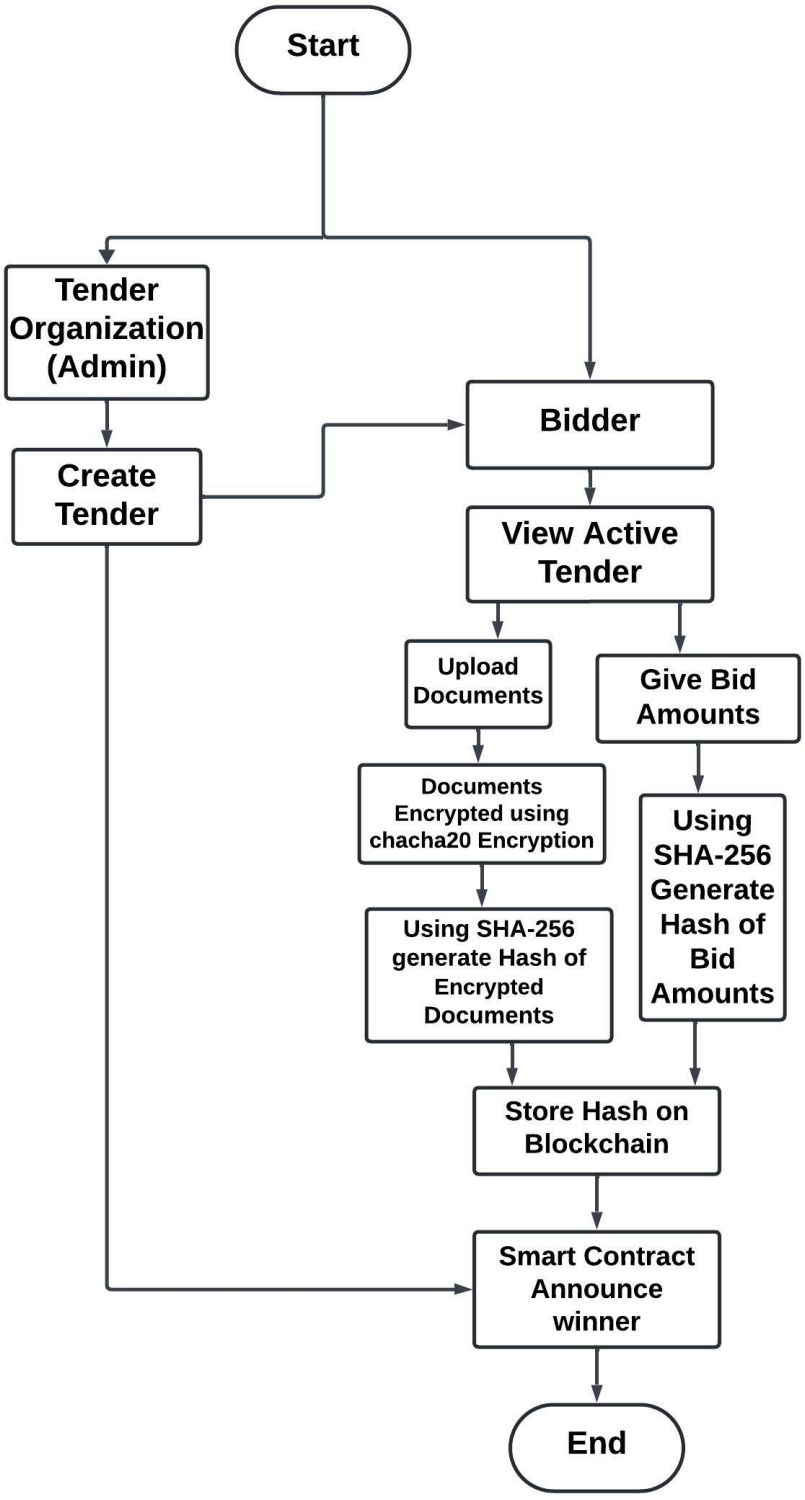


Figure 3.2: Flow Diagram of Blockchain Based E-Tendering System

Bid Submission and Encryption Subsystem : Bids are submitted by bidders together with additional documents and bid amounts. To ensure confidentiality, the bid documents are encrypted using ChaCha20 encryption. Following encryption, the bid amounts and encrypted documents' unique hash are generated using SHA-256 hashing. Data integrity is guaranteed by the blockchain's storage of the bid amount hashes and encrypted documents. The technology connects the frontend and blockchain using Web3.js. When a bid is submitted, the frontend communicates with the smart contract via Web3.js. The SHA-256 hash is stored in the smart contract. Bid papers are encrypted using secret keys that can only be decrypted by the admin, guaranteeing bid confidentiality until the end of the deadline.

Winner Selection Using Smart Contracts : The winning bid will be selected by the system based on criteria written into the smart contract when the bid submission time ended. The selection procedure is safe and transparent, and the blockchain stores all bid evaluations in an immutable way. This guarantees the winner is selected fairly. An automated smart contract mechanism that evaluates bids according to with preset standards and criteria is utilized to choose the winner.

3.1.3 Engineering Analysis of the Design Process :

The engineering research that proves the design's scalability, security, and dependability—ensuring that it will function as required—is included in this part. The demonstration emphasizes on encryption, blockchain architecture, smart contract execution, and performance to demonstrate that the design is strong enough to handle the e-tendering process without any problems.

1. Encryption and Data Integrity Analysis:

SHA-256 hashing and ChaCha20 encryption improve e-tendering system security by resolving a number of possible vulnerabilities. ChaCha20 encryption is used to protect bidders' private documents and bid amounts, making the encrypted data inaccessible even in the case of unauthorized access. This guarantees bid secrecy all the way through the bidding procedure. The SHA-256 hashing algorithm is used to encrypt documents and bid amounts, producing unique digital fingerprints (hashes) that are kept on the blockchain.

These hashes' immutability and integrity ensure that any tampered material can be quickly identified with the bid because even a small change would produce an entirely new hash. With ChaCha20's confidentiality and SHA-256's integrity and immutability guarantees, the system is highly resilient to security threats such document counterfeiting, data breaches, and manipulation. This dual-layered cryptographic technique offers a strong architectural solution for preserving security and transparency while strengthening the e-tendering system against attacks.

ChaCha20 Encryption: Using an XOR operation between the data and the generated keystream, this stream cipher creates ciphertext from the plaintext bid documents:

$$C = P \oplus K$$

Here, K is the keystream produced by the ChaCha20 algorithm, P is the plaintext (bid documents), and C is the ciphertext.

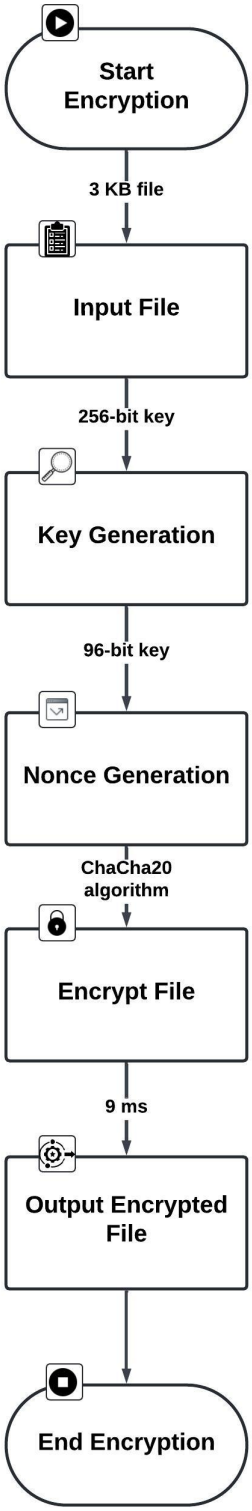


Figure 3.3: Encryption(Chacha20)

SHA-256 Hashing: To ensure data integrity, this function creates a 256-bit hash for the encrypted data:

$$H = \text{SHA-256}(C)$$

Here, H is the blockchain-stored hash, and C is the ciphertext generated during the encryption process.

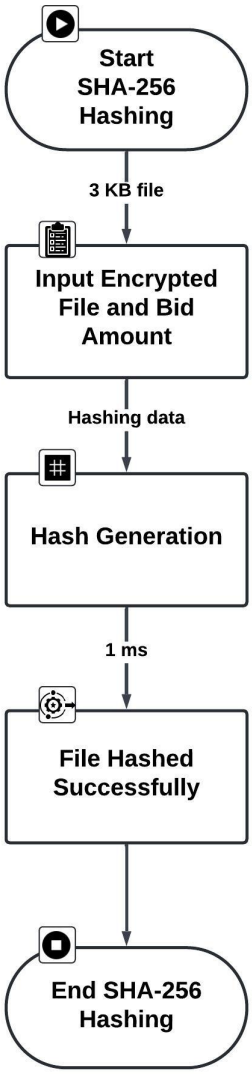


Figure 3.4: Hashing(SHA-256)

Blockchain Integration and Smart Contract Execution :

The Ethereum blockchain ensures the transparency and immutability of the creation of tenders, bid submission, and winner selection. The design uses smart contracts to automate and enforce important features, such deadlines and winner selection, without the need for human participation.

Tender Creation: When a tender is created, the following format is used to record details on the blockchain:

Ti=tenderID,description,deadline

where Ti is the transaction that contains the tender data, together with the integrity-preserving hash of the tender details.

Bid Submission: The smart contract submits the encrypted bid data and its hash for every bid:

storeBid(bidderID,hashData,encryptedData)

Winner Selection: After the deadline, the smart contract evaluates bids using predefined criteria, such the lowest bid.

selectWinner()winningBid

The smart contract ensures fairness and transparency by operating independently and executing the rules without human intervention. The blockchain ensures that all bid submissions and tender information are auditable and inaccessible. The selection process is clear and equitable, and no bid can be changed after it has been submitted due to the smart contract logic.

Results of Analysis : ChaCha20 encryption and SHA-256 hashing are used in the blockchain-based e-tendering system to guarantee security and integrity, making bid data easily verifiable and impenetrable. Immutability is guaranteed by the Ethereum blockchain, which guards against fraud and manipulation throughout the bidding process. By automating crucial processes like winner selection and deadlines, smart contracts guarantee equity and openness. Storing only hashes and metadata on-chain minimizes expenses while optimizing the system for efficiency and scalability. To ensure that the blockchain-based e-tendering system would function as needed and not malfunction, it has undergone extensive engineering examination. The solution is safe, dependable, and scalable due to the integration of smart contract automation, blockchain immutability, and encryption, which solves any problem that arises with conventional tendering methods. By adhering to these engineering principles, the design of the system may be investigated and replicated with the assurance that it will function as planned.

3.2 Adopted Tools and techniques

Several tools and technologies were used to handle the project's frontend and backend during the development of the blockchain-based e-tendering system. The tools utilized at each stage of the project are broken down below:

Software Requirements :

Solidity Remix IDE: Used to create, test, and implement smart contracts that manage the development of tenders, the submission of bids, and the automatic selection of winners.

Web3.js: Enables communication between the Ethereum blockchain and the frontend (JavaScript). It facilitates transaction sending, contract function calls, and blockchain data retrieval.

Express.js and Node.js in JavaScript: Both front-end and back-end development require JavaScript. Express.js is a web framework for managing routes and APIs, and Node.js provides the runtime for server-side programming.Backend logic and Web3.js integration for blockchain communication are all handled by Node.js.For

managing bid submissions, responding to HTTP requests, and communicating with the frontend, Express.js offers routing.

Embedded JavaScript Timing, or EJS: EJS is a templating engine that uses JavaScript to dynamically create HTML pages. Enables the creation of dynamic front-end content, including the presentation of ongoing bids, tenders, and outcomes derived from blockchain data. Easily combines with Express.js to create dynamic, server-rendered webpages.

SHA-256 hashing and ChaCha20 encryption: SHA-256 hashes are created and stored on the blockchain, and ChaCha20 secures the bid documents prior to submission. While ChaCha20 is chosen for its lightweight, fast, and secure encryption, SHA-256 is best suited for creating tamper-proof hashes.

MetaMask: A browser extension that is used to sign transactions on the Ethereum blockchain and authenticate users.

XAMPP: Utilized as a local server for managing non-blockchain data databases (such as user and tender details).XAMPP offers a user-friendly and adaptable programming environment for local server setup and database administration.

Justification for Tools Selection: For the blockchain-based e-tendering system, the techniques and tools were selected based on their scalability, security, ease of integration, and blockchain-specific characteristics. Smooth and secure interactions are ensured via Web3.js and MetaMask's seamless connection with the Ethereum network. Data integrity and bid documents are strongly secured by SHA-256 hashing and ChaCha20 encryption. In a tendering system, processing several concurrent requests requires excellent scalability and throughput, which Node.js and Express.js guarantee. With its capacity to render dynamic material server-side, EJS is perfect for presenting bid submissions and tender information in real time. A adaptive local environment to handle non-blockchain data was offered by XAMPP, while the Remix IDE was chosen for effective smart contract development. The performance, security, and functioning of the system are guaranteed by these tools.

Hardware Requirements : Our blockchain-based e-tendering system's physical architecture is described in depth under the hardware requirements. They could consist of:

- **Processor:** Intel Core i5 or AMD Ryzen 5
- **RAM :** 16 GB
- **Storage:** 512 GB SSD
- **Operating System:** Windows 11.
- **Network:** A 50 Mbps broadband connection .

3.3 Design the Solution

The blockchain-based electronic tendering system is intended to provide a safe, open, and effective platform for overseeing the whole tendering procedure. The system combines essential functional and non-functional needs to guarantee smooth functioning.The system is modular and decentralized, with a secure distributed ledger to record transactions and blockchain technology for transparency. The back end manages smart contract deployment, bid submission, and tender development, while the front end offers admin and bidder interfaces. Data integrity is verified by hashing algorithms, while bid secrecy is guaranteed by encryption.

Functional Requirements :

Tender Creation: Admin (tender organizations) should be able create tenders with all the necessary information, such as bid requirements and deadlines.

Bid Submission: Through the system, bidders can sacurely submit their bids and any supporting documentation.

Encryption and Hashing: Before being stored on the blockchain, bid documents and amounts are hashed using SHA-256 and encrypted using ChaCha20.

Automated Winner Selection: Using predefined criteria, smart contracts choose the winning bid automatically.

Non Functional Requirements :

Security: Using hashing and encryption techniques, the system must ensure the bid data's integrity, confidentiality, and integrity.

Fairness : Since no human interference may change automatic decision-making, smart contracts guarantee that bids are treated impartially and fairly.

Immutability: The blockchain secures the system from tampering by storing data that cannot be altered once committed, such as bid details and record of transactions.

Transparency: The blockchain records every operation and transaction, guaranteeing accountability and transparency.

Usability: Both admin and bidders should find the user interface easy to use, enabling seamless navigation and communication.

Performance: The system should respond instantly to the formation of tenders, the submission of bids, and the encryption of bids.

System Design :

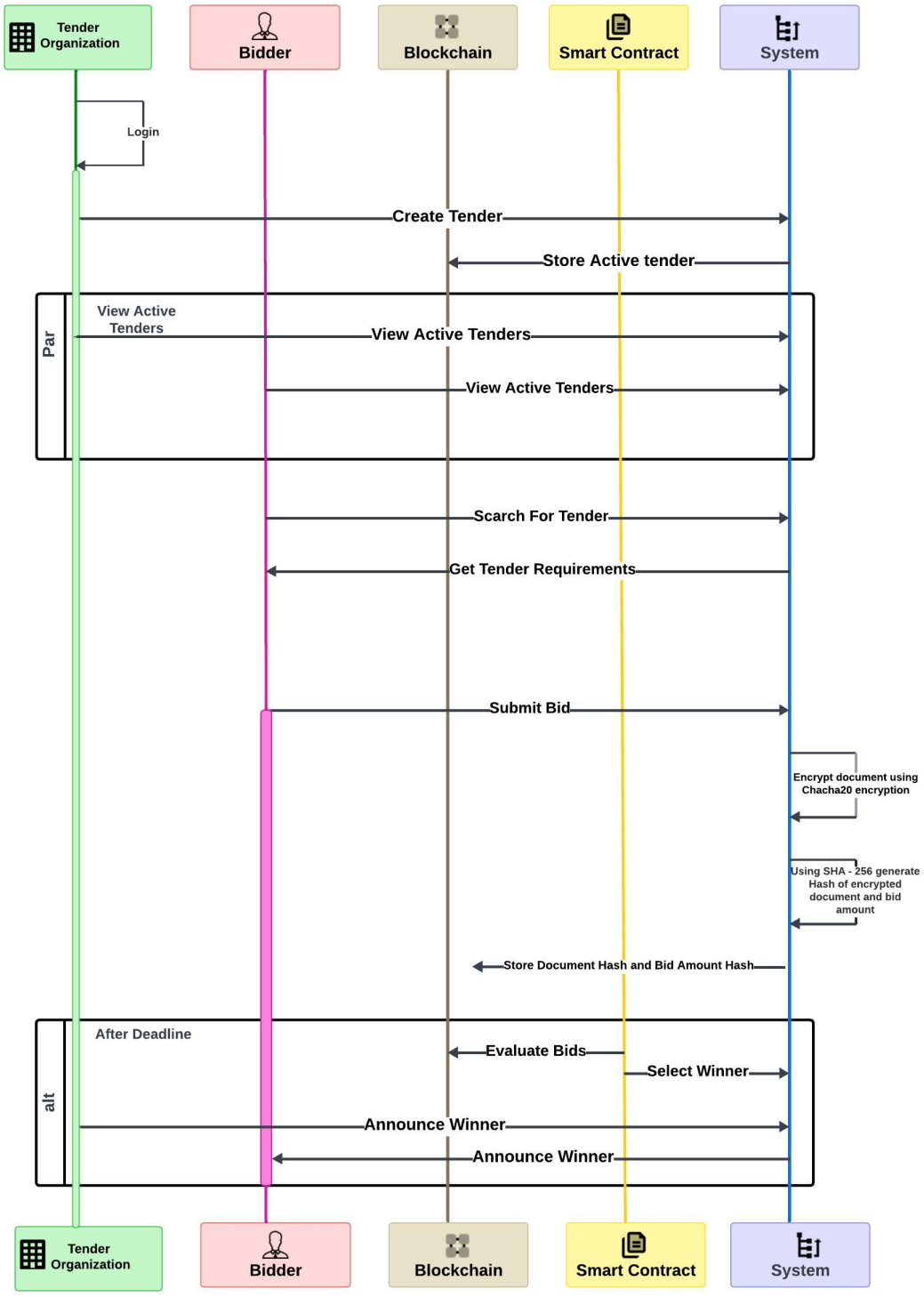


Figure 3.5: System Design of e-Tendering System

By combining several elements, including smart contracts, blockchain, and encryption techniques, the blockchain-based e-tendering system's design, as shown in the figure, ensures an efficient and safe tendering process. The bidder and the tender organization (admin) are the two main users of the system. To ensure transparency, a tender is first created by the tender organization and then stored on the blockchain. The active tenders are viewed by bidders and the admin. After looking for tenders and retrieving the necessary information, the bidders submit their offers, which are secured by ChaCha20 encryption and hashed using the SHA-256 algorithm. To ensure data integrity, the blockchain stores the hash of the bid amounts and encrypted documents.

After the deadline, a smart contract selects the winner by analyzing the submissions according to specified criteria. This procedure guarantees that the bidding process is secure and impenetrable because every interaction is stored on the blockchain. The winner is announced automatically using smart contracts, which reduces the need for human participation and guarantees fairness and transparency. Through the use of cryptographic algorithms and blockchain concepts, the system offers a secure, decentralized, and reliable e-tendering platform.

3.4 SDLC

Within the Software Development Life Cycle (SDLC), the Waterfall model serves as the framework for the approach used to create the blockchain-based e-tendering system. The necessity for exacting, clearly defined phases that guarantee security, transparency, and immutability—all essential for a blockchain-based platform—informed this decision. A thorough investigation of current blockchain-based e-tendering systems is the first step in the process. This stage of study looks at existing solutions and identifies problems including centralized control, lack of transparency, and bid manipulation. Identifying these difficulties highlights the issue with conventional e-tendering systems, which are unreliable and untrustworthy.

The requirements specification phase starts after the research is finished. Various individuals, including as bidders and tender organizations, gather both functional and non-functional needs during this phase. Creating tenders, placing bids, encrypting papers, and safely storing data on the blockchain are examples of functional needs. Security, system performance, and scalability are the main topics of non-functional requirements. These specifications serve as the foundation for the system's design, guaranteeing that all required functionalities are included and that the system functions properly and efficiently.

The next step is system design, during which the blockchain-based e-tendering system's architecture is thoroughly described. This stage entails creating the interface between the blockchain network, bidders, the tender organization, and smart contracts. Here, utilizing blockchain's decentralized structure and applying cryptographic techniques to ensure the system's security and transparency is a top priority.

After that, smart contract development starts. The entire tendering process is automated via smart contracts, which represent an essential system component. Rules and criteria for creating tenders and evaluating bids are regulated by the contracts. To ensure fairness and remove human bias, particular bid evaluation criteria are encoded into smart contracts at this step. These contracts then automatically choose the winning bid based on predetermined characteristics.

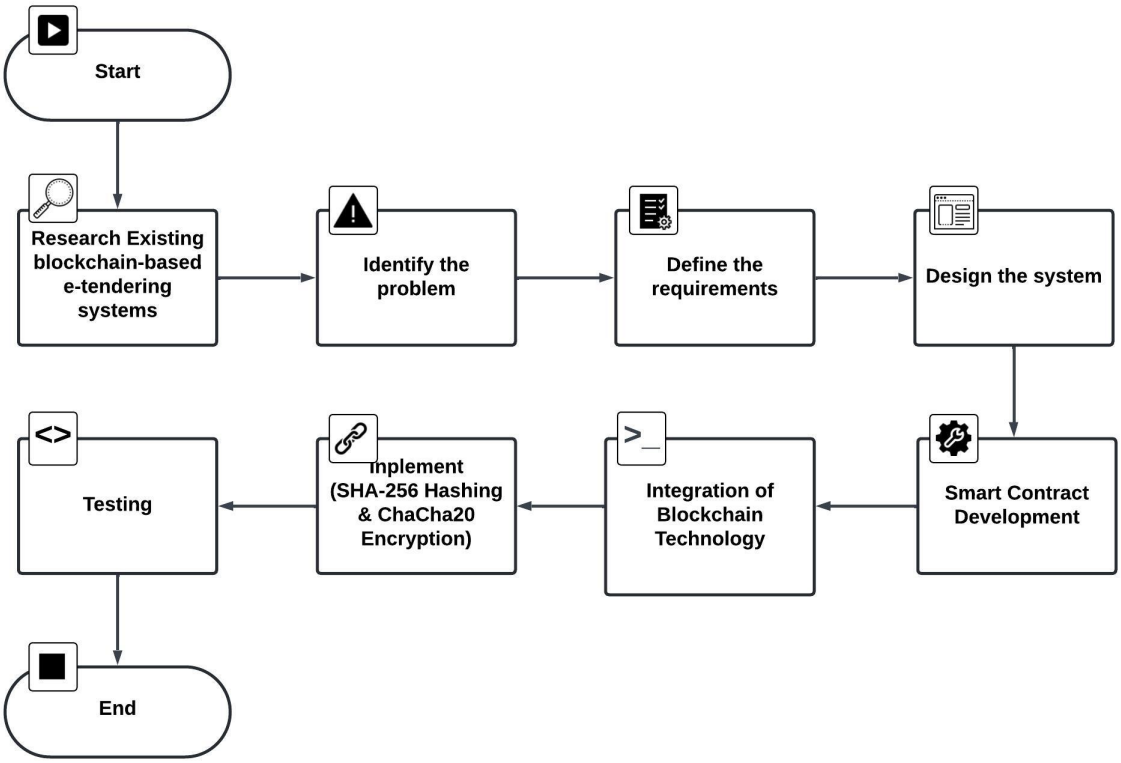


Figure 3.6: Methodology

Blockchain technology is included into the system after smart contracts are developed. With the blockchain integrated into the system, all transactions (including bid submissions and tender announcements) are documented in a decentralized ledger during this period. This maintains transparency and prevents illegal modifications. The immutability of blockchain technology is used to foster confidence between all stakeholders because every action is forever recorded and verifiable at any time.

Using encryption techniques is the next stage. The system generates a hash of the encrypted bid amounts and documents using SHA-256 hashing and encrypts bid documents using ChaCha20. The blockchain stores this hash, ensuring the confidentiality and integrity of the data. By using these cryptographic techniques, bid data is protected against manipulation and decryption without the right keys, even if it is intercepted.

The system is then put through testing. This stage includes security testing, which looks at how well the encryption, hashing, and blockchain integration work, as well as functional testing, which makes sure the system can complete activities like creating tenders and submitting bids. Additionally, usability testing is done to make sure the user interface is easy to use and accessible for bidders and the tender organization.

SHA-256 hashing and ChaCha20 encryption improve e-tendering system security by resolving a number of possible vulnerabilities. ChaCha20 encryption is used to protect bidders' private documents and bid amounts, making the encrypted data inaccessible even in the case of unauthorized access. This guarantees bid secrecy all the way through the bidding procedure. The SHA-256 hashing algorithm is used to encrypt documents and bid amounts, producing unique digital fingerprints (hashes) that are kept on the blockchain.

This page

3.5 Implementation

The blockchain-based e-tendering system's integration of smart contracts, robust cryptography, and blockchain technology is intended to ensure the tendering process's efficiency, security, and transparency. To enable seamless interaction between bidders and the tender organization (admin), the system is constructed with a combination of frontend and backend technologies and tools.

Smart Contract Development with Solidity and Remix IDE

Solidity smart contracts were used to create the fundamental logic of the system, which included creating tenders, submitting bids, and choosing winners. Remix IDE, a potent online programming environment for Ethereum smart contracts, was used to create and test these smart contracts. Important tasks like establishing bid deadlines, confirming submissions, and choosing the lowest price after the deadline are all automated by the contract.

Smart contract deployment and testing using the Sepolia testnet were part of the implementation of the blockchain-based e-tendering system. Sepolia is an Ethereum

test network that replicates the primary Ethereum network without the expenses and dangers associated with using real Ethereum (ETH). It gives developers the opportunity to create, test, and interact with smart contracts in a context that is comparable to Ethereum but makes use of a separate currency. It is appropriate for trying out decentralized apps (dApps), such as the e-tendering system. Sepolia was used to deploy and extensively test the smart contract that creates tenders, submits bids, and automatically selects winners.

Deployment of Smart Contract:

In order to deploy the smart contract, we first link the Sepolia test network to MetaMask, a browser extension that functions as an Ethereum wallet. We used a faucet to add some test Ether (ETH) to your MetaMask wallet after it was set up in order to pay transaction fees. After setting up MetaMask, the deployment procedure entails using Web3.js to connect to the Sepolia testnet.

The e-tendering system's deployment can be viewed on The Etherscan, which provides comprehensive details such as the block confirmation, transaction hash, and contract address. This makes it possible to track the smart contract's status on the blockchain transparently.

Create Tender: In a blockchain-based e-tendering system, an admin can create a new tender using the createTender function, a Solidity smart contract technique. Three factors are required: name (the tender's title), description (a thorough explanation of the tender), and deadline (the Unix timestamp indicating the bid deadline). Using the necessary statement, the function first verifies that the deadline is in the future. If not, an error message is displayed and the transaction is reverted.

After validation, the tenderCounter which records the total number of tenders generated—is raised. The following details are then included in the new tender that is added to the tender mapping: ID, name, description, deadline, active status, winner selection status, winning bidder address, winning bid amount, and winning company name. All parties participating in the e-tendering process can examine and validate the tender's pertinent information thanks to this structure, which ensures that it is captured and stored on the blockchain.

Bid Submission: In the blockchain-based e-tendering system, a bidder can place a bid for a particular tender using the submitBid function, a Solidity smart contract mechanism. Four parameters are accepted: companyName, documentHash, bidAmount, and tenderId.To prevent submissions for non-existent tenders, the function first verifies that the tender with the given tenderId exists by executing the tenderExists modifier.

Additionally, it uses the onlyBeforeDeadline modification to determine the current time before the tender's deadline, guaranteeing that bids can only be submitted. The tender, however, is still open to bid. If both requirements are met, the method generates a new instance of the Bid struct, which contains the firm name, document hash, bid amount, bidder address, and tender ID. After that, this bid is added to the bids mapping, which keeps track of all the bids related to every tender. In addition to giving bidders a simple and safe method to submit their offers by their preferences, this system guarantees that the tender's guidelines and deadlines are kept.

ViewActiveTenders : Both admin and bidders can examine all active tenders that haven't had a winner chosen yet with the viewActiveTenders function, a public view method in a Solidity smart contract. The number of active tenders is initially determined by iterating over all of the tenders (from 1 to tenderCounter) and verifying that each one is active and has not yet had a winner chosen. The active tenders are then created and stored in an array. This array is then filled in a second loop with the specifics of every active tender that satisfies the requirements. Lastly, this approach gives back the array of active tenders, giving users a simple way to see what opportunities are ready to bid.

ViewAllBids : Admin and bidders can view all bids made for a particular tender identified by tenderId using the viewAllBids function, a public view method in a Solidity smart contract. Using the tenderExists modifier, it first determines whether the tender is active, and then it uses the onlyAfterDeadline modifier to confirm that the bidding deadline has passed. The function returns the array of bids linked to the specified tender ID if both requirements are satisfied. This promotes transparency

in the bidding process by making it simple for users to access and review every bid made after the tender deadline.

ViewWinnerAndBidAmount : The viewWinnerAndBidAmount function is a Solidity smart contract's public view function that allows bidders and admin view the winning bid along with information for a particular tender that is identified by the tenderId. The tenderExists modifier is used to initially determine whether the tender is active, and the onlyAfterDeadline modifier is used to verify that the bidding deadline has passed. Next, the function verifies that a minimum of one offer has been made for the tender. All bids are iterated through until the one with the lowest bid amount is determined to be the winning bid, and the winning bid is initially set as the first bid that is submitted. Lastly, it gives back the winning bid amount, the winning bidder's address, and the name of the business that placed the winning bid. Users may easily find out who won and how much the bidder offered .

Chacha20 Encryption and SHA-256 Hashing :

ChaCha20 is a common encryption method used to protect data in systems such as e-tendering platforms based on blockchain. This type of encryption combines data with a stream of randomly generated bytes depending on a key and a nonce to encrypt it. Sensitive bidder documents can be encrypted using this approach in an e-tendering system , protecting the data's confidentiality.

The primary secret used by ChaCha20 to encrypt and decrypt the data is a 256-bit key, which is 32 bytes in size. To guarantee that only authorized users can access the encrypted data, this key must be sent or stored securely. ChaCha20 requires a 96-bit nonce (12 bytes) in addition to the key. Each encryption session has its own nonce, which guarantees that even if the same key is used again, the result will be distinct. Because of the unpredictability the nonce offers, attackers are less likely to find patterns in encrypted data.

The ciphertext, which is unintelligible without the key and nonce, is created by combining these bytes with the original data (using XOR operations). The

ciphertext is given the same key and nonce for decoding. The ciphertext is changed back into the original document by reversing the XOR process. The blockchain-based e-tendering system, ChaCha20, makes sure that bidder documents are safely encrypted, rendering them unintelligible without the right key and nonce. This guarantees that private bid data is safeguarded until it is decrypted by authorized parties.

SHA-256 Hashing :

A cryptographic hash algorithm called SHA-256 (Secure Hash Algorithm 256-bit) generates a fixed-size 256-bit (32-byte) hash from input data. Because of its security features, blockchain systems make significant use of it. To ensure data integrity in a blockchain-based e-tendering system, SHA-256 can be used to bidder documents and bid amounts to create a distinct digital fingerprint. SHA-256 performs mathematical operations on data in discrete blocks to generate a fixed-length output, called the hash when hashing a file or bid amount. This hash is specific to the original input; a single-byte change in the data will produce an entirely different hash.

3.6 Evaluate the solution :

Several significant issues with conventional tendering procedures, including efficiency, security, and transparency, are successfully addressed by the deployment of the blockchain-based e-tendering system. The system promises a safe, decentralized, and automated procedure for creating tenders, submitting bids, and choosing winners by leveraging blockchain technology, smart contracts, and cryptographic techniques including SHA-256 hashing and ChaCha20 encryption. An analysis of the system's various components may be found below:

1. Transparency :

The use of blockchain naturally increases the transparency of the e-tendering system. The blockchain permanently records every transaction, including the creation of tenders, the submission of bids, and the selection of winners. This ensures that the integrity of the entire process can be checked by bidders and admin.

The system adds another level of transparency by putting the smart contracts on the Sepolia testnet and making the contract data viewable on Etherscan. The transaction hash, block confirmation, and gas fees are among the contract details that the general public can check. This makes it easier to ensure that the deployment is safe and accessible to all parties involved.Transparency in the bid selection process is promoted by the viewActiveTenders, viewAllBids, and viewWinnerAndBidAmount features, which enable participants to monitor the progress of tenders and bids in real time.

2. Security : To guarantee security, the system makes use of both blockchain technology and cryptographic methods:

Solidity smart contracts automate crucial aspects of the tendering process, such as deadline setting, bid verification, and winner selection. Because the contract functions independently and cannot be changed once it is deployed, this reduces the possibility of human error or manipulation. By using MetaMask, the bidding process is secured because only authorized users are able to sign and submit transactions.

To enhance system security, ChaCha20 encryption is used to safeguard important bidder papers. This keeps bidder information safe from unwanted access, guaranteeing the privacy of every bid. The solution efficiently protects data from possible breaches during transport or storage due to a 256-bit encryption key. To generate distinct digital fingerprints for the data, the system hashes bid documents and amounts using SHA-256. Because it detects any changes made to the original data, it guarantees data integrity. The hash is updated to reflect any changes made to the documents or bid values, warning the system of possible manipulation.

3. Efficiency : The blockchain architecture and smart contracts automate time-consuming and repetitive activities, which streamlines the tendering process:

The createTender feature reduces the administrative load by making the process of creating new tenders easier. Important information can be entered by admin, and the smart contract makes sure that deadlines and other requirements are fulfilled. Bidders can submit bids quickly and securely using the submitBid function which has safeguards in place to ensure that the tender rules are followed. The system will select the best or lowest bid automatically after the tender deadline due to the viewWinnerAndBidAmount function. By doing away with the necessity of manual decision-making, this saves time and lowers the possibility of human error or partiality.

4. User Experience : Both the bidders and the tendering organization (admin) will find the system design to be user-friendly:

Users' interactions with the Ethereum blockchain are made easier by the system's integration of MetaMask. As a bridge, MetaMask makes it simple for users to sign transactions and control their cryptocurrency wallets. A simple and easy-to-use interface is offered by the MetaMask pop-up for transaction confirmation while completing tasks like deploying contracts or placing bids. The solution uses Remix IDE, a well-known tool for creating and evaluating smart contracts. It provides a smooth environment in which to write and test Solidity code, ensuring that the contracts work as intended before deployment.

Utilizing the inherent transparency and security of blockchain technology, the blockchain-based e-tendering system is a well-executed solution that maximizes the tendering process. The system provides an extremely safe and transparent environment for admins and bidders as well, with smart contracts automating crucial tasks and cryptographic approaches ensuring data confidentiality.

Chapter 4

Result analysis

4.1 Investigate the final result

The blockchain-based e-tendering system has created a more efficient, transparent, and safe process. Blockchain greatly reduces the potential of manipulation and fosters confidence by ensuring tamper-proof bid submissions and real-time tracking of all activities. Smart contracts ensure a fair and accurate process by automating bid reviews and winner selection. Tendering is more reliable and efficient due to this safe, decentralized technology. We examine the outcomes in this section, emphasizing how the system's improved security, transparency, and efficiency have transformed conventional tendering procedures.

Smart Contracts :

We first implemented smart contracts for the blockchain-based e-tendering system. A notification from MetaMask asking for confirmation to complete the transaction arrived after the contract was deployed.

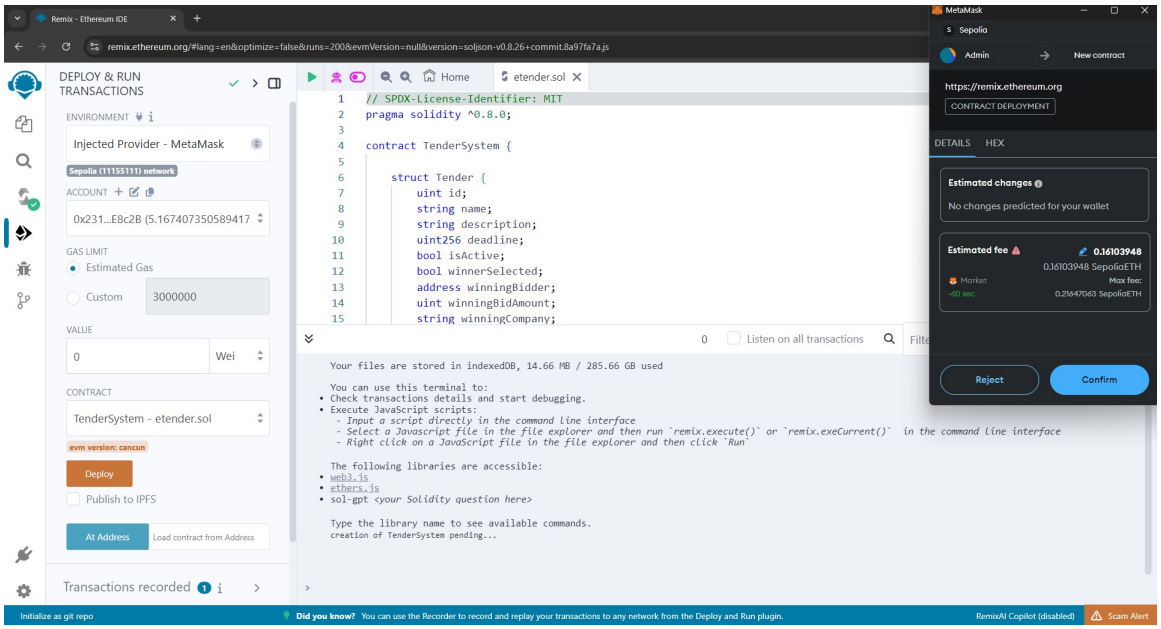


Figure 4.1: The deployment of Smart Contract

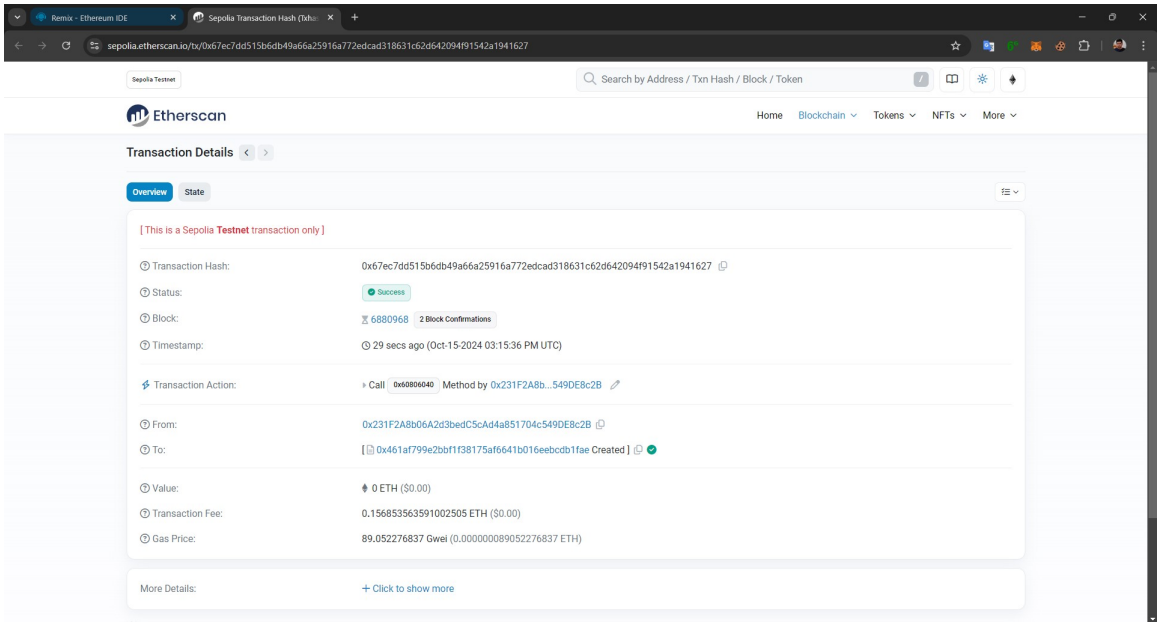


Figure 4.2: Smart Contract deployment details on Etherscan

After the deployment was confirmed by MetaMask, Etherscan shows the transaction data. To verify that the smart contract was properly deployed using the

Sepolia testnet, Etherscan displays essential details like as the transaction hash, block number, and gas fee.

Tender Creation(Admin) :

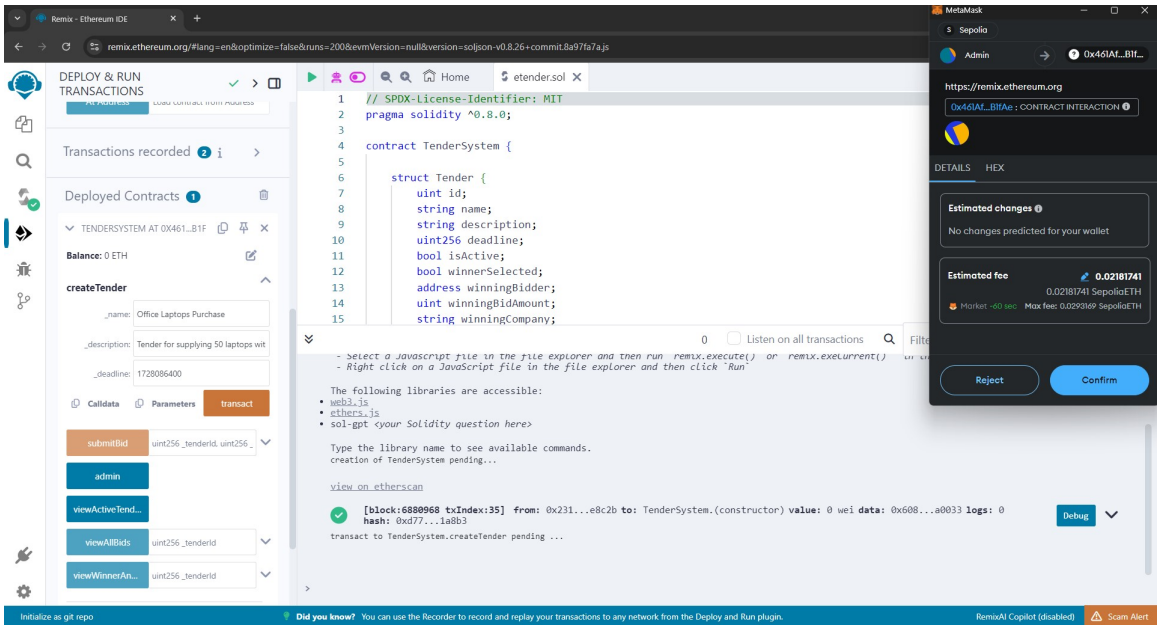


Figure 4.3: Creating Tenders on Smart Contracts

A tender is created by entering the name, description, and deadline after deployment.

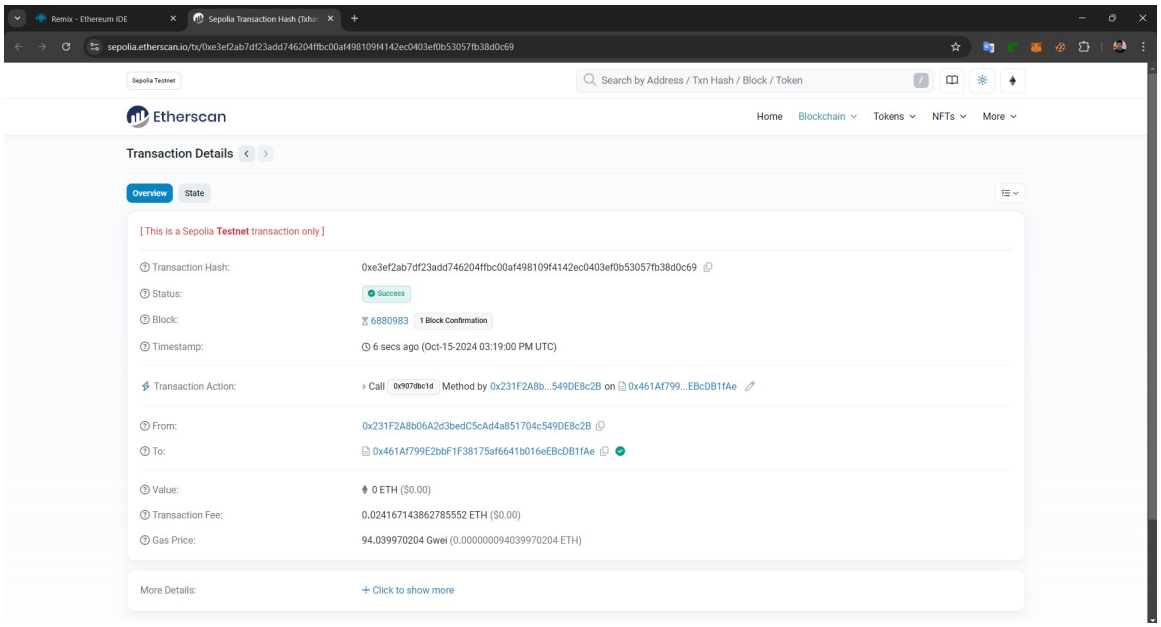


Figure 4.4: Etherscan shows Details of Tender Creation

Bid Submission (Bidder 1 :)

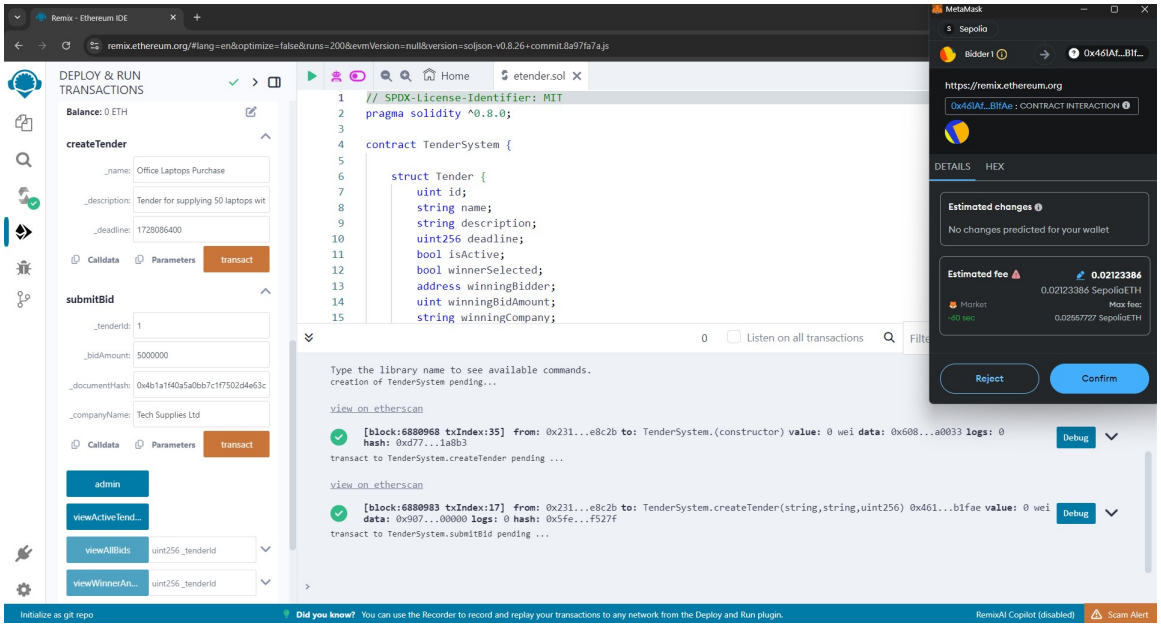


Figure 4.5: Bid Submission by Bidder 1

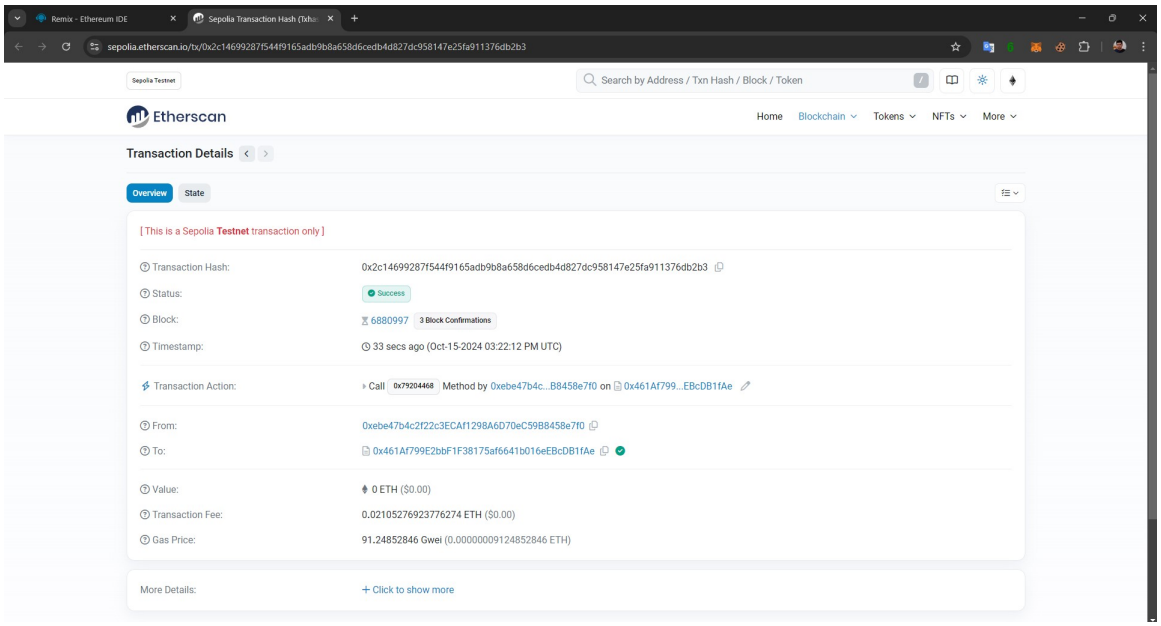


Figure 4.6: Etherscan shows Bid Submission Details

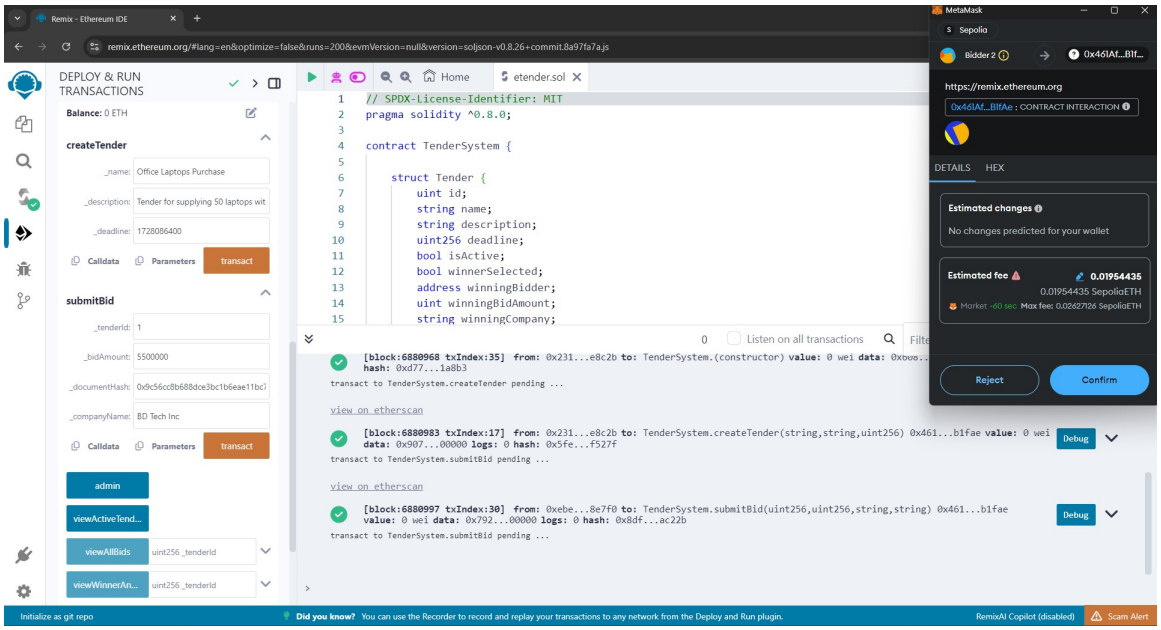


Figure 4.7: Bid Submission by Bidder 2

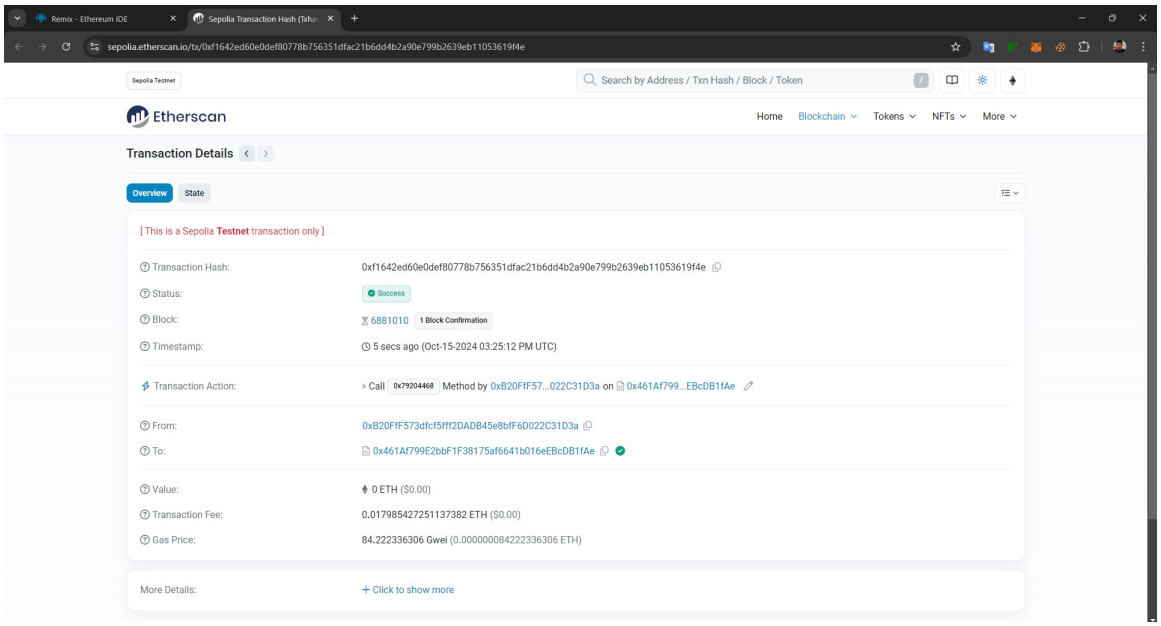


Figure 4.8: Etherscan shows Bid Submission Details

After the tender creation, the bidder can bid for the tender by giving tenderid, company name,bidamount,documenthash.

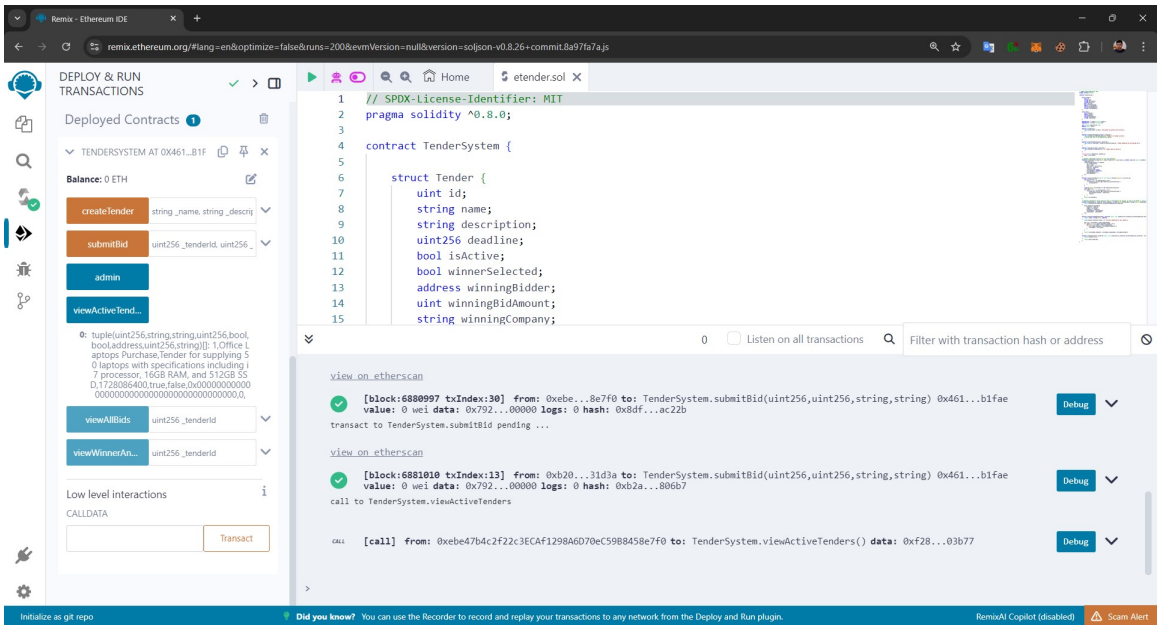


Figure 4.9: View Active Tenders by Admin and Bidder

Both the admin and the bidder could view the active tender after it was created.

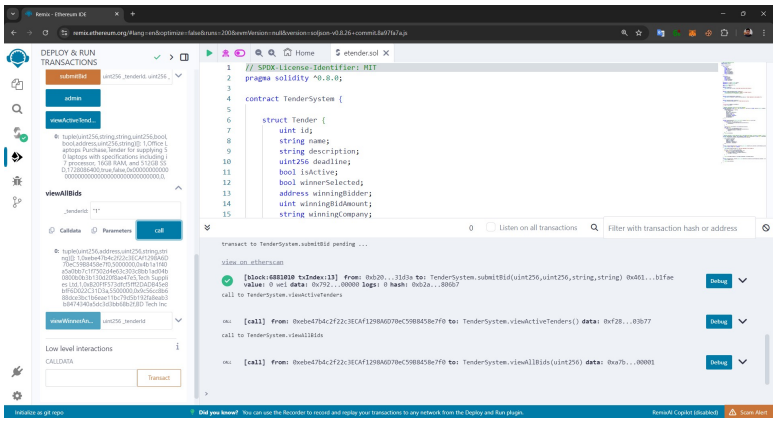


Figure 4.10: View All Bids by Admin and Bidder

The bidder and the admin can view all bids for a certain tender by giving its tender ID after the deadline.

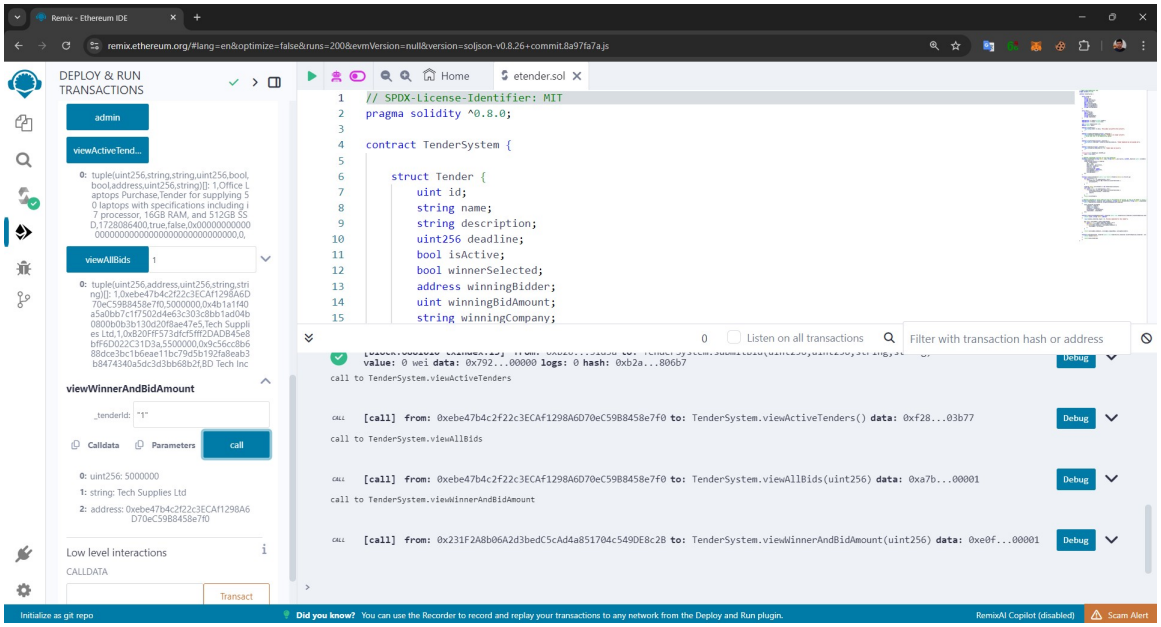


Figure 4.11: View Winner And Bid Amount by Admin and Bidder

By providing its tender ID, the admin and bidder can view the tender's winner and bid amount after the deadline.

Web Application

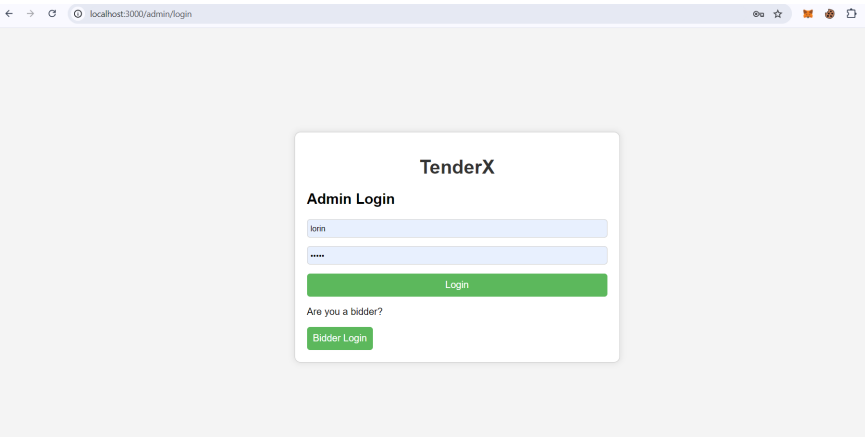


Figure 4.12: Administrator Login

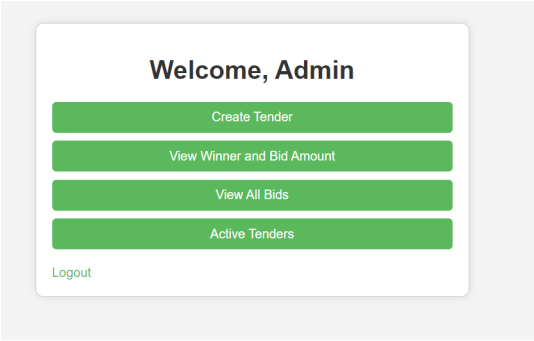


Figure 4.13: Admin Panel

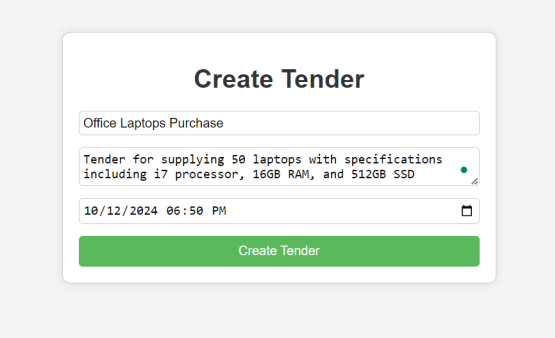


Figure 4.14: Admin creates the tender.

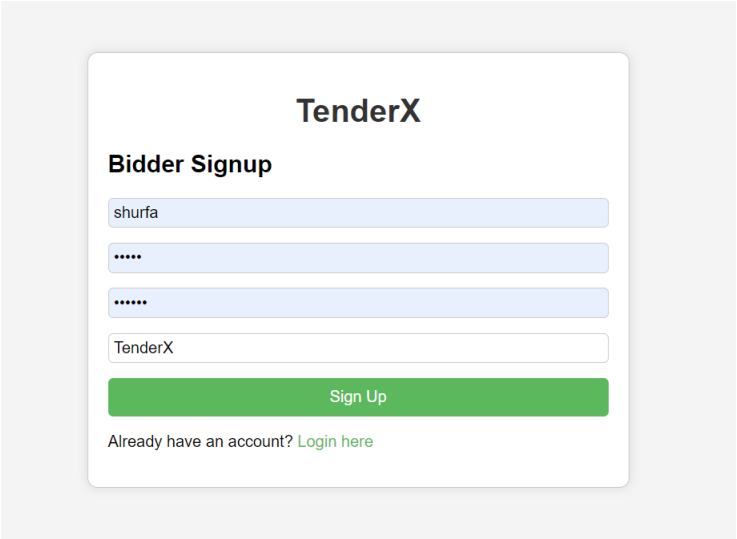


Figure 4.15: Signing Up as a Bidder

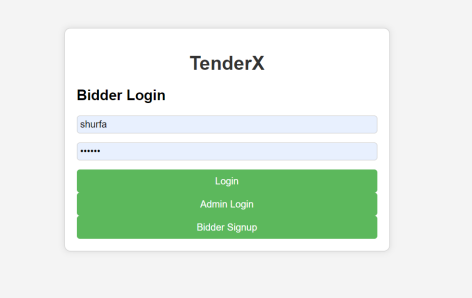


Figure 4.16: Login as Bidder

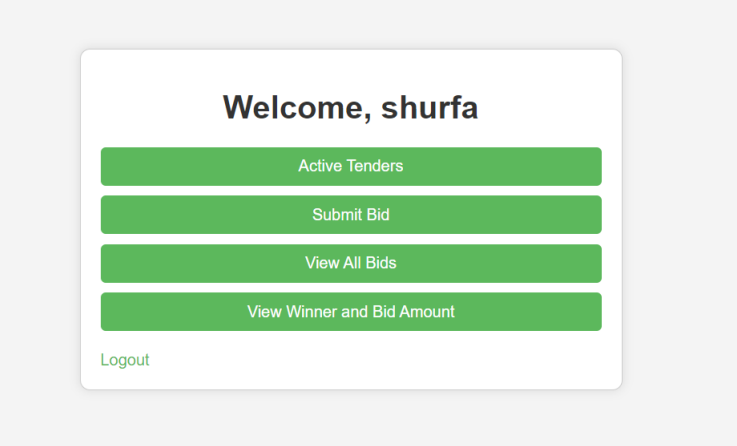


Figure 4.17: Bidder Panel

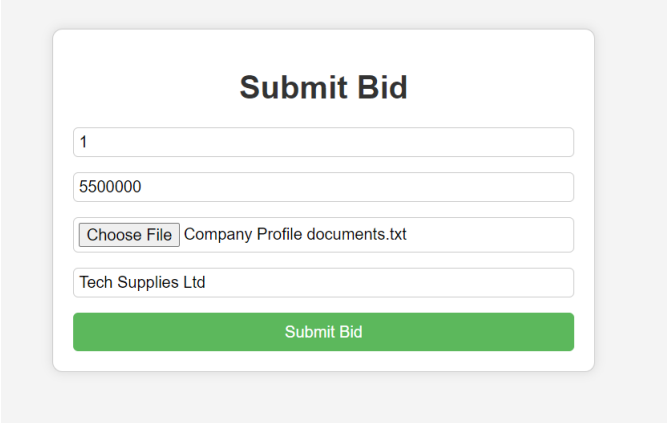


Figure 4.18: Bidder submit Bid

Encryption and Hashing of Bidder Documents (Testing Server)) :

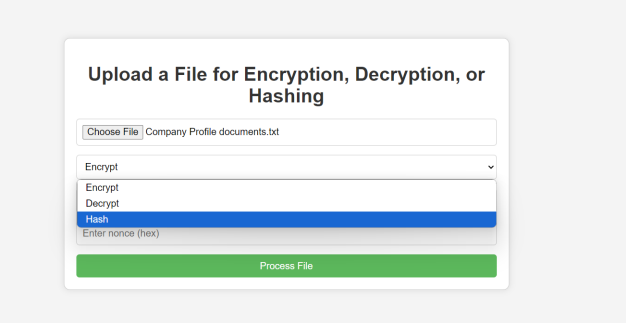


Figure 4.19: Selection of Document Hash

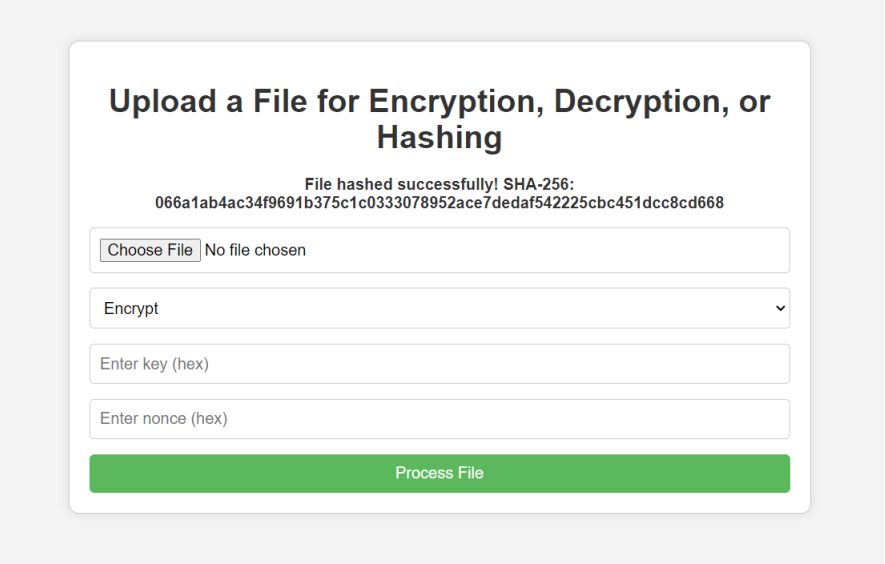


Figure 4.20: Hash Generated

Figure 4.21: Selection of File Encryption

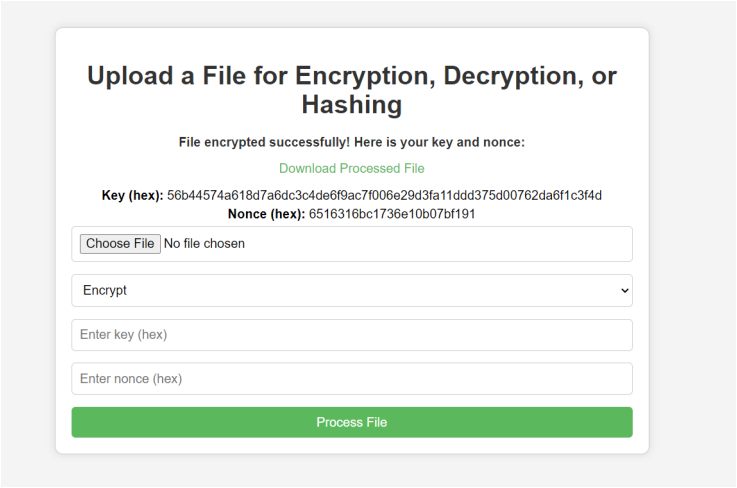


Figure 4.22: Encryption of Documents

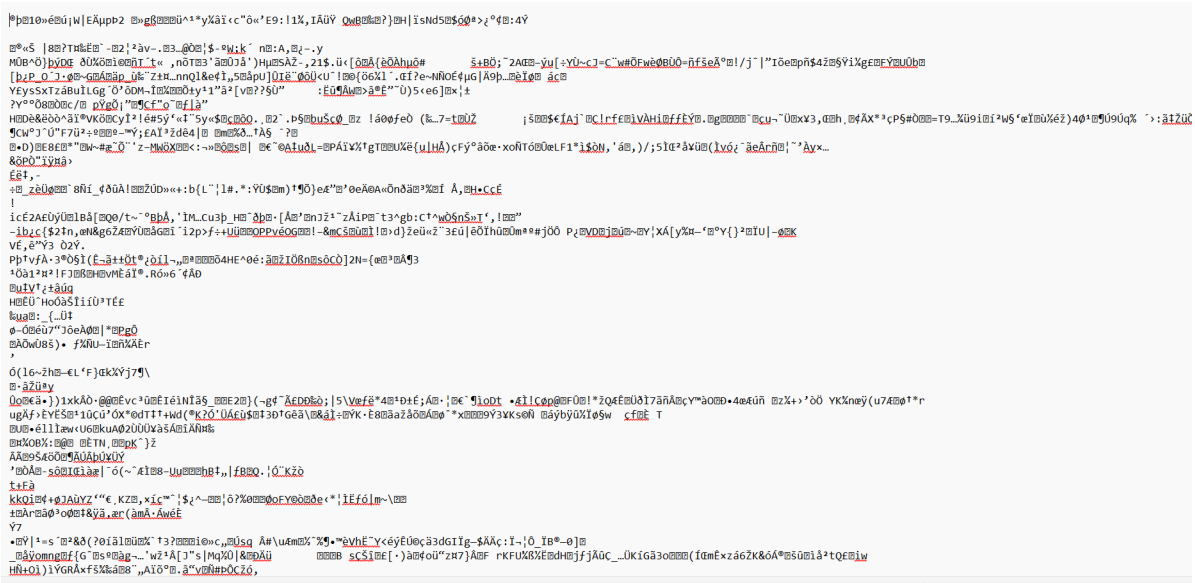


Figure 4.23: Documents Encrypted

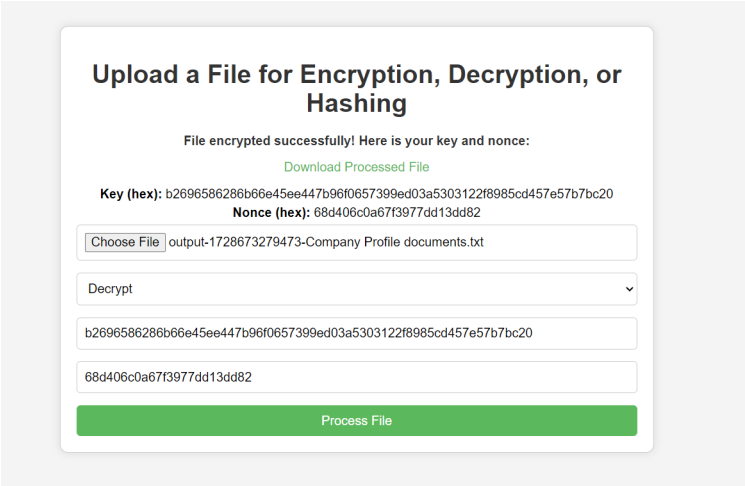


Figure 4.24: Decryption of Documents

Figure 4.25: File Decrypted

Chapter 5

Conclusions

5.1 Societal Impacts and Mitigation Strategies for E-Tendering System

To ensure effective implementation, several societal concerns raised by the blockchain-based e-tendering system need to be addressed. Although the system encourages equity and openness, accessibility is still an issue, especially for smaller businesses or areas with less-developed technology. The digital divide may worsen if smaller bidders who are not familiar with blockchain technology find it difficult to participate. Moreover, using smart contracts to automate tender procedures may lessen human oversight, which could result in inaccurate bid evaluations or disputes—particularly if all parties involved lack a thorough understanding of the technology. The advantages of adopting blockchain could be undermined by this ignorance, which could lead to misunderstandings and poor handling of the tendering process.

Course of Action to Mitigate Concerns:

To address these societal issues, several proactive measures need to be implemented. The first step is to provide thorough training and support programs to close the digital divide and guarantee fair access. By training all participants—especially smaller bidders and organizations with fewer technology resources—these programs should help close the knowledge gap and promote wider participation. For non-technical users, the system should also have easy-to-use interfaces and technical support to reduce entrance barriers. Lastly, to ensure that the system runs

smoothly while maintaining some degree of human judgment, hybrid oversight models that combine automated smart contracts with human review may be able to assist prevent errors and disagreements. These steps will contribute to the inclusiveness, accessibility, and equity of blockchain-based e-tendering platforms for all parties involved.

5.2 Impact on Environment and Sustainability

The blockchain-based e-tendering procedure affects sustainability and the environment in both positive and potentially negative ways. The system minimizes the carbon footprint and paper waste involved with printing, distributing, and physically storing tender documents by drastically reducing the requirement for paper-based procedures. By using fewer resources, this move to digital processes encourages sustainability.

However, it's also important to take into account how blockchain technology itself affects the environment. To validate transactions, public blockchains like Ethereum need a lot of processing power, which raises energy consumption and the corresponding carbon emissions. So the system uses blockchain platforms that use less energy, like Proof of Stake consensus processes, which have a lower environmental impact than energy-intensive Proof of Work systems. By using blockchain technology and automating conventional bidding procedures, the project can make a positive impact on environmental sustainability.

5.3 Ethical and professional principles

Professional and ethical standards were respected throughout the blockchain-based e-tendering system's development by making sure the design was safe, open, and in line with the law. As engineers, it is our job to create a system that would ensure the integrity and privacy of any bid information while encouraging openness in the bidding process. By including failure analysis and hazards, Design for Safety techniques were incorporated into the project with an emphasis on preventing

unauthorized access, tampering, and data breaches. By employing SHA-256 hashing and ChaCha20 encryption, the system minimizes the danger of cyberattacks and maintains data integrity while protecting critical bid data.

Furthermore, the smart contract logic guarantees that the tender process is carried out automatically, lowering the possibility of fraud or manipulation and fostering equity. Potential points of failure, such as blockchain network assaults or smart contract vulnerabilities, were found using hazard analysis and addressed with strong security procedures and frequent audits. The system complies with national and international regulatory standards, such as those about data protection and procurement, guaranteeing that the technology is used in the public and private sectors ethically. These actions show a dedication to professional integrity, legal compliance, and public safety throughout the development process.

5.4 Brief Summary

The blockchain-based e-tendering system is anticipated to revolutionize the procurement environment by improving the efficiency, security, and transparency of the tendering process. One of the main results is the establishment of a transparent environment that minimizes the likelihood of fraud and corruption by providing real-time information about bids and tender documents to all parties involved. More companies are expected to participate in the tendering process as a result of the increased participant trust brought about by this transparency.

Through the combination of blockchain technology and advanced encryption techniques, the project seeks to enhance sensitive data security in addition to transparency. This will safeguard the integrity of the procurement process by preventing illegal access and data manipulation. It is anticipated that the implementation of smart contracts would automate several steps in the tendering process, resulting in a major decrease in the time and expense involved in procurement. Organizations will be able to more efficiently allocate resources due to this efficiency, which will ultimately speed up project durations.

The system's goal is to encourage fair competition by giving all bidders equal access to opportunities and information. This will improve bid quality overall and

result in greater value for projects in the public and private sectors. Additionally, the project will guarantee adherence to national and international legal standards for data privacy and procurement, upholding the validity of the bidding procedure and safeguarding the rights of stakeholders. Another important result of the system is environmental sustainability, as it is anticipated to decrease paper use and encourage environmentally friendly procurement methods. The initiative will support environmental conservation efforts by making it possible to track and verify sustainable supply chain activities.

The blockchain-based e-tendering system's overall goal is to involve a range of stakeholders in its creation and application, including corporations, government agencies, and the general public. This interaction will foster confidence and guarantee that the system satisfies the various needs of every user. In the end, this initiative is anticipated to produce a more reliable, effective, and sustainable procurement environment, which will benefit all parties financially in the long run.

5.5 Future works

To expand the blockchain-based e-tendering system's functionality and encourage broader use, several areas for further research have been highlighted. Research will be conducted to optimize the system's performance to handle a higher volume of transactions, guaranteeing efficiency even in high-demand situations. Scalability enhancements will be a major focus. Integration with current procurement databases and systems is another crucial area. It would be essential to provide smooth integration techniques so that the platform can integrate smoothly with traditional systems and be widely used in various industries.

Furthermore, by identifying irregularities, anticipating hazards, and enhancing the system's intelligence, the integration of AI and advanced analytics could significantly improve the bid review procedure. To make the system more user-friendly and accessible to users with various levels of technical expertise, extra work is also required to enhance the user interface and experience.

Improvements in regulatory and compliance are also essential since the system's legal integrity and broad acceptance will depend on continued legal framework research and conformity to international standards. Lastly, there's a chance to investigate sustainable blockchain solutions. Research into alternate consensus methods or energy-efficient blockchain technologies may be able to lessen the system's negative environmental effects and bring it into line with international sustainability objectives. These future developments are intended to improve the system's usability, scalability, and sustainability over the long run.

References

Alves Batista, D. (2024). Enhancing transparency and accountability in public procurement: exploring blockchain technology to mitigate records fraud. *Records Management Journal*.

Averin, A., & Averina, O. (2019). Review of blockchain technology vulnerabilities and blockchain-system attacks. In *2019 international multi-conference on industrial engineering and modern technologies (fareastcon)* (pp. 1–6).

Begum, A., Tareq, A., Sultana, M., Sohel, M., Rahman, T., & Sarwar, A. (2020). Blockchain attacks analysis and a model to solve double spending attack. *International Journal of Machine Learning and Computing*, *10*(2), 352–357.

Bikkad, R. (2023). Web based tender management system using blockchain. *International Research Journal of Modernization in Engineering Technology and Science*. doi: 10.56726/IRJMETS33398

Chen, Y.-H., Chen, S.-H., & Lin, I.-C. (2018). Blockchain based smart contract for bidding system. In *2018 ieee international conference on applied system invention (icasi)* (pp. 208–211).

Dasgupta, D., Shrein, J. M., & Gupta, K. D. (2019). A survey of blockchain from security perspective. *Journal of Banking and Financial Technology*, *3*, 1–17.

De Santis, F., Schauer, A., & Sigl, G. (2017). Chacha20-poly1305 authenticated encryption for high-speed embedded iot applications. In *Design, automation & test in europe conference & exhibition (date), 2017* (pp. 692–697).

Dubey, S., Singh, P., Verma, R. K., & Kamboj, D. (2023). Government tender allocation using blockchain technology. In *2023 international conference on iot, communication and automation technology (icicat)* (pp. 1–6).

Gaikwad, A., Thube, A., Londhe, V., & Pachhade, P. R. (2023, May). Enhancing security and transparency in digital tendering using blockchain technology. *International Journal of Research Publication and Reviews*, *4*(5), 6538-6540.

Gong, T., Tao, X., Das, M., Liua, Y., & Chenga, J. (2022). Blockchain-based e-tendering evaluation framework. *International Journal of Automation & Digital Transformation*, *1*(1), 75–93.

Hiremath, C. S. A. R., Abhinandan B M, & V, G. (2023). Blockchain based e-tendering system. *JETIR*, *10*(4). (Publication Date: April 2023)

König, L., Unger, S., Kieseberg, P., Tjoa, S., & Blockchains, J. R. C. (2020). The risks of the blockchain a review on current vulnerabilities and attacks. *J. Internet Serv. Inf. Secur.*, *10*(3), 110–127.

Kuntamukala, S., & Raju, A. A. (2024). Smart tender contract management system using block-chain.

Li, H., & Xue, W. (2021). A blockchain-based sealed-bid e-auction scheme with smart contract and zero-knowledge proof. *Security and Communication Networks*, *2021*(1), 5523394.

Li, X. (2020). Bces: a blockchain based credible e-bidding system. In *2020 ieee 6th international conference on computer and communications (iccc)* (pp. 2415–2421).

Mali, D., Mogaveera, D., Kitawat, P., & Jawwad, M. (2020). Blockchain-based e-tendering system. In *2020 4th international conference on intelligent computing and control systems (iciccs)* (pp. 357–362).

Marati, V., Pinagani, S., Sathwika, N., Lakshmi, V., & Yasaswini, A. (2023). Revolutionizing tender management through blockchain technology. In *2023 international conference on inventive computation technologies (icict)* (pp. 1172–1177).

Nir, Y., & Langley, A. (2015). *Chacha20 and poly1305 for ietf protocols.* (RFC 7539)

Pawar, R., Pohare, R., Wasnik, S., Jengthe, S., Paunikar, S., & Gondane, S. (2023). Contract/tendering system using blockchain.

Reddy, E. S. T. K., Sathvik, M., Rajaram, V., & Rao, C. P. (2023). An intelligent tender management system using block chain and ipfs. In *2023 international conference on sustainable computing and smart systems (icscss)* (pp. 1497–1502).

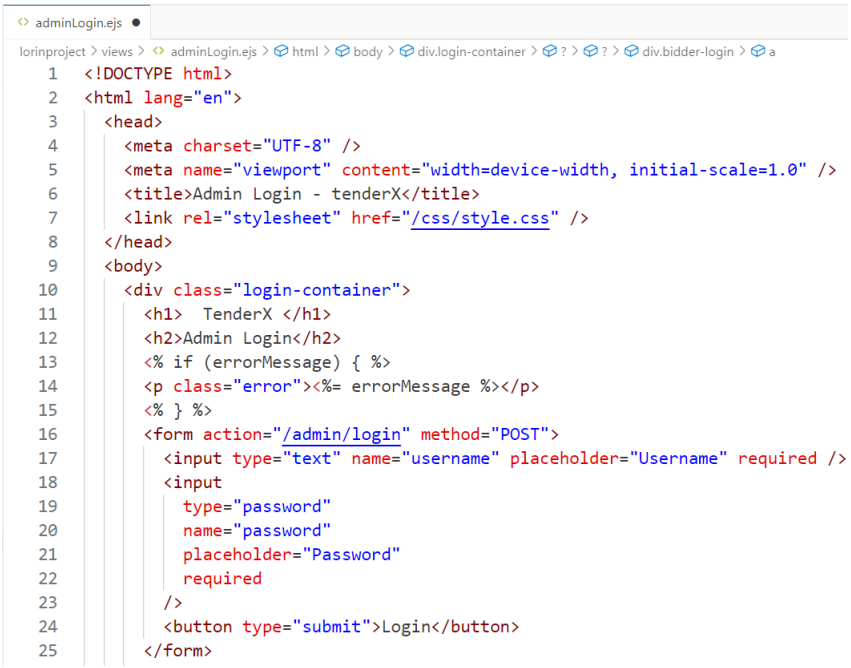
Shelar, S., Sharma, & Chandratre. (2022). Auctionaire - a blockchain-based online auction system. *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*, *10*(III), 2065-2071. doi: 10.22214/ijraset.2022.41039

Vokerla, R. R., Shanmugam, B., Azam, S., Karim, A., De Boer, F., Jonkman, M., & Faisal, F. (2019). An overview of blockchain applications and attacks. In *2019 international conference on vision towards emerging trends in communication and networking (vitecon)* (pp. 1–6).

Appendices (Optional)

Website Code

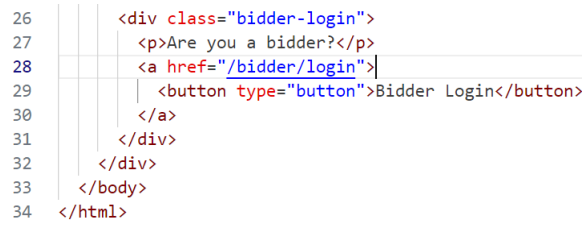
Frontend Code (Admin) :



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Admin Login - tenderX</title>
    <link rel="stylesheet" href="/css/style.css" />
  </head>
  <body>
    <div class="login-container">
      <h1>  TenderX </h1>
      <h2>Admin Login</h2>
      <% if (errorMessage) { %>
      <p class="error"><%= errorMessage %></p>
      <% } %>
      <form action="/admin/login" method="POST">
        <input type="text" name="username" placeholder="Username" required />
        <input
          type="password"
          name="password"
          placeholder="Password"
          required
        />
        <button type="submit">Login</button>
      </form>
```

adminlogin.ejs 1

Frontend Code (Admin) :



```
      <div class="bidder-login">
        <p>Are you a bidder?</p>
        <a href="/bidder/login">
          <button type="button">Bidder Login</button>
        </a>
      </div>
    </div>
  </body>
</html>
```

adminlogin.ejs 2

createTender.ejs

lorinproject > views > admin > createTender.ejs > html > body

```ejs
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Create Tender</title>
    <link rel="stylesheet" href="/css/style.css">
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js@1.3.0/dist/web3.min.js"></script>
    <script src="/js/web3.js"></script>
</head>
<body
    <div class="login-container">
        <h1>Create Tender</h1>
        <form id="create-tender-form">
            <input type="text" name="name" placeholder="Tender Name" required>
            <textarea name="description" placeholder="Tender Description" required></textarea>
            <input type="datetime-local" name="deadline" required>
            <button type="submit">Create Tender</button>
        </form>
    </div>
    <script nonce="<%= nonce %>">
        document.getElementById('create-tender-form').addEventListener('submit', async function(event) {
            event.preventDefault();

            const name = document.querySelector('input[name="name"]').value;
            const description = document.querySelector('textarea[name="description"]').value;
            const deadline = new Date(document.querySelector('input[name="deadline"]').value).getTime() /
```

Createtender.ejs 1

createTender.ejs

lorinproject > views > admin > createTender.ejs > html > body

```ejs
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Create Tender</title>
    <link rel="stylesheet" href="/css/style.css">
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js@1.3.0/dist/web3.min.js"></script>
    <script src="/js/web3.js"></script>
</head>
<body
    <div class="login-container">
        <h1>Create Tender</h1>
        <form id="create-tender-form">
            <input type="text" name="name" placeholder="Tender Name" required>
            <textarea name="description" placeholder="Tender Description" required></textarea>
            <input type="datetime-local" name="deadline" required>
            <button type="submit">Create Tender</button>
        </form>
    </div>
    <script nonce="<%= nonce %>">
        document.getElementById('create-tender-form').addEventListener('submit', async function(event) {
            event.preventDefault();

            const name = document.querySelector('input[name="name"]').value;
            const description = document.querySelector('textarea[name="description"]').value;
            const deadline = new Date(document.querySelector('input[name="deadline"]').value).getTime() /
```

Createtender.ejs 2

dashboard.ejs

lorinproject > views > admin > dashboard.ejs > ...

```ejs
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Dashboard</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <div class="login-container">
        <h1>Welcome, Admin</h1>
        <nav>
            <ul>
                <li><a href="/admin/createTender">Create Tender</a></li>
                <li><a href="/admin/viewWinnerAndBidAmount">View Winner and Bid Amount</a></li>
                <li><a href="/admin/viewAllBids">View All Bids</a></li>
                <li><a href="/admin/activeTenders">Active Tenders</a></li>
            </ul>
        </nav>
        <a href="/logout">Logout</a>
    </div>
</body>
</html>
```

Dashboard.ejs

viewActiveTenders.ejs

lorinproject > views > admin > viewActiveTenders.ejs > html

```ejs
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Active Tenders</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <div class="login-container">
        <h1>Active Tenders</h1>
        <ul>
            <% if (activeTenders.length > 0) { %>
                <% activeTenders.forEach(tender => { %>
                    <li>
                        <strong><%= tender.name %></strong><br>
                        Deadline: <%= new Date(tender.deadline * 1000).toLocaleString() %><br>
                        Description: <%= tender.description %>
                    </li>
                <% }) %>
            <% } else { %>
                <li>No active tenders available at the moment.</li>
            <% } %>
        </ul>
    </div>
</body>
</html>
```

ViewActiveTender.ejs

viewAllBids.ejs

lorinproject > views > admin > viewAllBids.ejs > html > body > script > addEventListener('submit') callback

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - View All Bids</title>
    <link rel="stylesheet" href="/css/style.css">
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js@1.3.0/dist/web3.min.js"></script>
    <script src="/js/web3.js"></script>
</head>
<body>
    <div class="login-container">
        <h1>View All Bids</h1>
        <form id="view-bids-form">
            <label for="tenderId">Select Tender:</label>
            <select name="tenderId" id="tenderId" required>
                <% activeTenders.forEach(tender => { %>
                    <option value="<%= tender.id %>"><%= tender.name %></option>
                <% }) %>
            </select>
            <button type="submit">View Bids</button>
        </form>
        <div id="bids-details" style="display: none;">
            <h2>Bids for Tender</h2>
```

ViewAllBids.ejs 1

viewAllBids.ejs

lorinproject > views > admin > viewAllBids.ejs > html > body > script > addEventListener('submit'

```
<html lang="en">
<body>
    <div class="login-container">
        </form>
        <div id="bids-details" style="display: none;">
            <h2>Bids for Tender</h2>
            <table>
                <thead>
                    <tr>
                        <th>Bidder Address</th>
                        <th>Bid Amount (ETH)</th>
                        <th>Company Name</th>
                        <th>Document Hash</th>
                    </tr>
                </thead>
                <tbody id="bids-table-body">
                </tbody>
            </table>
        </div>
    </div>
```

ViewAllBids.ejs 2

viewAllBids.ejs ●

lorinproject > views > admin > viewAllBids.ejs > html > body > div.login-container

```
<html lang="en">
<body>
    <div class="login-container">
    </div>
    <script>
        document.getElementById('view-bids-form').addEventListener('submit', async function(event) {
            event.preventDefault();

            const tenderId = document.getElementById('tenderId').value;

            try {
                if (typeof contract !== 'undefined') {
                    const bids = await contract.methods.viewAllBids(tenderId).call();

                    const tableBody = document.getElementById('bids-table-body');
                    tableBody.innerHTML = '';

                    bids.forEach(bid => {
                        const row = document.createElement('tr');

                        const bidderCell = document.createElement('td');
                        bidderCell.textContent = bid.bidder;
                        row.appendChild(bidderCell);

                        const bidAmountCell = document.createElement('td');
                        bidAmountCell.textContent = web3.utils.fromWei(bid.bidAmount, 'ether');
                        row.appendChild(bidAmountCell);
```

ViewAllBids.ejs 3

viewAllBids.ejs ●

lorinproject > views > admin > viewAllBids.ejs > html > body > div.login-container

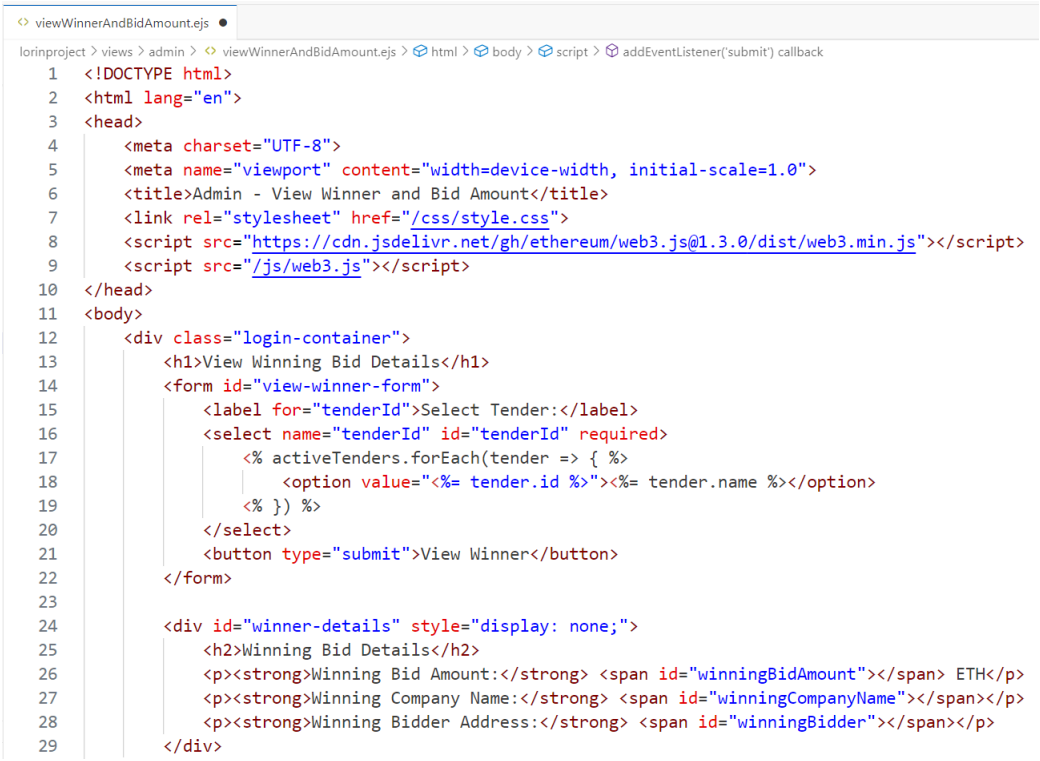
```
<html lang="en">
<body>
    <script>
        document.getElementById('view-bids-form').addEventListener('submit', async function(event) {
                    bids.forEach(bid => {
                        const companyNameCell = document.createElement('td');
                        companyNameCell.textContent = bid.companyName;
                        row.appendChild(companyNameCell);

                        const documentHashCell = document.createElement('td');
                        documentHashCell.textContent = bid.documentHash;
                        row.appendChild(documentHashCell);

                        tableBody.appendChild(row);
                    });

                    document.getElementById('bids-details').style.display = 'block';
                } else {
                    alert("Contract or web3 is not initialized.");
                }
            } catch (error) {
                console.error("Error fetching bids:", error);
                alert("Failed to fetch bids.");
            }
        });
    </script>
</body>
</html>
```

ViewAllBids.ejs 4



```
viewWinnerAndBidAmount.ejs
lorinproject > views > admin > viewWinnerAndBidAmount.ejs > html > body > script > addEventListener('submit') callback
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin - View Winner and Bid Amount</title>
    <link rel="stylesheet" href="/css/style.css">
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js@1.3.0/dist/web3.min.js"></script>
    <script src="/js/web3.js"></script>
</head>
<body>
    <div class="login-container">
        <h1>View Winning Bid Details</h1>
        <form id="view-winner-form">
            <label for="tenderId">Select Tender:</label>
            <select name="tenderId" id="tenderId" required>
                <% activeTenders.forEach(tender => { %>
                    <option value="<%= tender.id %>"><%= tender.name %></option>
                <% }) %>
            </select>
            <button type="submit">View Winner</button>
        </form>

        <div id="winner-details" style="display: none;">
            <h2>Winning Bid Details</h2>
            <p><strong>Winning Bid Amount:</strong> <span id="winningBidAmount"></span> ETH</p>
            <p><strong>Winning Company Name:</strong> <span id="winningCompanyName"></span></p>
            <p><strong>Winning Bidder Address:</strong> <span id="winningBidder"></span></p>
        </div>
```

ViewWinnerAndBidAmount.ejs 1

```
viewWinnerAndBidAmount.ejs
lorinproject > views > admin > viewWinnerAndBidAmount.ejs > html > body > script > addEventListener('submit') callback
<html lang="en">
<body>
    <script>
        document.getElementById('view-winner-form').addEventListener('submit', async function(event) {
            event.preventDefault();

            const tenderId = document.getElementById('tenderId').value;
            try {
                if (typeof contract !== 'undefined') {
                    const result = await contract.methods.viewWinnerAndBidAmount(tenderId).call();

                    const winningBidAmount = web3.utils.fromWei(result[0], 'ether');
                    const winningCompanyName = result[1];
                    const winningBidder = result[2];
                    document.getElementById('winningBidAmount').textContent = winningBidAmount;
                    document.getElementById('winningCompanyName').textContent = winningCompanyName;
                    document.getElementById('winningBidder').textContent = winningBidder;
                    document.getElementById('winner-details').style.display = 'block';
                } else {
                    alert("Contract or web3 is not initialized.");
                }
            } catch (error) {
                console.error("Error fetching winner details:", error);
                alert("Failed to fetch winner details.");
            }
        });
    </script>
</body>
</html>
```

ViewWinnerAndBidAmount.ejs 2

Frontend Code (Bidder) :

signup.ejs

lorinproject > views > bidder > signup.ejs > ...

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Bidder Signup - SecureTender Management System</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <div class="login-container">
        <h1>TenderX</h1>

        <h2>Bidder Signup</h2>
        <form action="/bidder/signup" method="POST">
            <input type="text" name="username" placeholder="Username" required>
            <input type="password" name="password" placeholder="Password" required>
            <input type="password" name="confirmPassword" placeholder="Confirm Password" required>
            <input type="text" name="companyName" placeholder="Company Name" required>
            <button type="submit">Sign Up</button>
        </form>
        <p>Already have an account? <a href="/bidder/login">Login here</a></p>
    </div>
</body>
</html>
```

Signup.ejs

bidderLogin.ejs

lorinproject > views > bidderLogin.ejs > html > body > div.login-container > form

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Bidder Login - TenderX</title>
    <link rel="stylesheet" href="/css/style.css" />
  </head>
  <body>
    <div class="login-container">
      <h1> TenderX </h1>
      <h2>Bidder Login</h2>
      <form action="/bidder/login" method="POST">
        <input type="text" name="username" placeholder="Username" required />
        <input
          type="password"
          name="password"
          placeholder="Password"
          required />
        <button type="submit">Login</button>
      </form>
      <form action="/admin/login" method="GET" style="margin-top: 10px;">
        <button type="submit">Admin Login</button>
      </form>
      <form action="/bidder/signup" method="GET" style="margin-top: 10px;">
        <button type="submit">Bidder Signup</button>
      </form>
    </div>
  </body>
```

login.ejs

dashboard.ejs

lorinproject > views > bidder > dashboard.ejs > html > body > div.login-container > nav > ul > li

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Bidder Dashboard</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <div class="login-container">
        <h1>Welcome, <%= bidder.username %></h1>
        <nav>
            <ul>
                <li><a href="/bidder/activeTenders">Active Tenders</a></li>
                <li><a href="/bidder/submitBid">Submit Bid</a></li>
                <li><a href="/bidder/viewAllBids">View All Bids</a></li>
                <li><a href="/bidder/viewWinnerAndBidAmount">View Winner and Bid Amount</a></li>
            </ul>
        </nav>
        <a href="/logout">Logout</a>
    </div>
</body>
</html>
```

Dashboard.ejs

viewActiveTenders.ejs

lorinproject > views > bidder > viewActiveTenders.ejs > html > body > div.login-container > ul

```ejs
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Active Tenders</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <div class="login-container">
        <h1>Active Tenders</h1>
        <ul>
            <% if (activeTenders.length > 0) { %>
                <% activeTenders.forEach(tender => { %>
                    <li>
                        <strong><%= tender.name %></strong><br>
                        Deadline: <%= new Date(tender.deadline * 1000).toLocaleString() %><br>
                        <form action="/bidder/submitBid" method="GET" style="display:inline;">
                            <input type="hidden" name="tenderId" value="<%= tender.id %>">
                            <button type="submit">Bid Tender</button>
                        </form>
                    </li>
                <% }) %>
            <% } else { %>
                <li>No active tenders available at the moment.</li>
            <% } %>
        </ul>
    </div>
</body>
</html>
```

ViewActiveTender.ejs

submitBid.ejs

lorinproject > views > bidder > submitBid.ejs > html > body > script > addEventListener('submit') callback

```ejs
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Submit Bid</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <div class="login-container">
        <h1>Submit Bid</h1>
        <form id="submit-bid-form" method="POST" enctype="multipart/form-data">
            <select name="tenderId" required>
                <% activeTenders.forEach(tender => { %>
                    <option value="<%= tender.id %>"><%= tender.name %></option>
                <% }) %>
            </select>
            <input type="text" name="bidAmount" placeholder="Bid Amount" required>
            <input type="file" name="bidFile" id="bidFile" required>
            <button type="submit">Submit Bid</button>
        </form>
    </div>
    <script>
        document.getElementById('submit-bid-form').addEventListener('submit', async function(event) {
            event.preventDefault();
            const formData = new FormData();
            const tenderId = document.querySelector('select[name="tenderId"]').value;
            const bidAmount = document.querySelector('input[name="bidAmount"]').value;
            const bidFile = document.getElementById('bidFile').files[0];
```

SubmitBid.ejs 1

submitBid.ejs

lorinproject > views > bidder > submitBid.ejs > html > body > script > addEventListener('submit') callback

```
<html lang="en">
<body>
    <script>
        document.getElementById('submit-bid-form').addEventListener('submit', async function(event) {
            const bidFile = document.getElementById('bidFile').files[0];
            formData.append('tenderId', tenderId);
            formData.append('bidAmount', bidAmount);
            formData.append('bidFile', bidFile);
            try {
                const response = await fetch('/bidder/submitBid', {
                    method: 'POST',
                    body: formData
                });

                const result = await response.json();

                if (response.ok) {
                    alert("Bid submitted successfully!");
                } else {
                    alert("Failed to submit bid: " + result.message);
                }
            } catch (error) {
                console.error("Error submitting bid:", error);
                alert("Failed to submit bid.");
            }
        });
    </script>
</body>
</html>
```

SubmitBid.ejs 2

viewAllBids.ejs

lorinproject > views > bidder > viewAllBids.ejs > html > body > script > addEventListener('submit') callback

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>View All Bids</title>
    <link rel="stylesheet" href="/css/style.css">
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js@1.3.0/dist/web3.min.js"></script>
    <script src="/js/web3.js"></script>
</head>
<body>
    <div class="login-container">
        <h1>View All Bids</h1>
        <form id="view-bids-form">
            <label for="tenderId">Select Tender:</label>
            <select name="tenderId" id="tenderId" required>
                <% activeTenders.forEach(tender => { %>
                    <option value="<%= tender.id %>"><%= tender.name %></option>
                <% }) %>
            </select>
            <button type="submit">View Bids</button>
        </form>
        <div id="bids-details" style="display: none;">
            <h2>Bids for Tender</h2>
            <table>
                <thead>
                    <tr>
                        <th>Bidder Address</th>
                        <th>Bid Amount (ETH)</th>
```

ViewAllBids.ejs 1

viewAllBids.ejs

lorinproject > views > bidder > viewAllBids.ejs > html > body > script > addEventListener('submit') callback

```
<html lang="en">
<body>
    <div class="login-container">
        <div id="bids-details" style="display: none;">
            <table>
                                <th>Company Name</th>
                                <th>Document Hash</th>
                            </tr>
                        </thead>
                        <tbody id="bids-table-body">
                        </tbody>
                    </table>
                </div>
            </div>
            <script>
                document.getElementById('view-bids-form').addEventListener('submit', async function(event) {
                    event.preventDefault();
                    const tenderId = document.getElementById('tenderId').value;
                    try {
                        if (typeof contract !== 'undefined') {
                            const bids = await contract.methods.viewAllBids(tenderId).call();
                            const tableBody = document.getElementById('bids-table-body');
                            tableBody.innerHTML = '';
                            bids.forEach(bid => {
                                const row = document.createElement('tr');
                                const bidderCell = document.createElement('td');
                                bidderCell.textContent = bid.bidder;
                                row.appendChild(bidderCell);
                                const bidAmountCell = document.createElement('td');
```

ViewAllBids.ejs 2

viewAllBids.ejs

lorinproject > views > bidder > viewAllBids.ejs > html > body > script > addEventListener('submit') callback

```
<html lang="en">
<body>
    <script>
        document.getElementById('view-bids-form').addEventListener('submit', async function(event)
                    bids.forEach(bid => {
                        const bidAmountCell = document.createElement('td');
                        bidAmountCell.textContent = web3.utils.fromWei(bid.bidAmount, 'ether');
                        row.appendChild(bidAmountCell);
                        const companyNameCell = document.createElement('td');
                        companyNameCell.textContent = bid.companyName;
                        row.appendChild(companyNameCell);

                        const documentHashCell = document.createElement('td');
                        documentHashCell.textContent = bid.documentHash;
                        row.appendChild(documentHashCell);
                        tableBody.appendChild(row);
                    });
                    document.getElementById('bids-details').style.display = 'block';
                } else {
                    alert("Contract or web3 is not initialized.");
                }
            } catch (error) {
                console.error("Error fetching bids:", error);
                alert("Failed to fetch bids.");
            }
        });
    </script>
</body>
</html>
```

ViewAllBids.ejs 3

viewWinnerAndBidAmount.ejs

lorinproject > views > bidder > viewWinnerAndBidAmount.ejs > html > body > div.login-container

```ejs
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>View Winning Bid</title>
    <link rel="stylesheet" href="/css/style.css">
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js@1.3.0/dist/web3.min.js"></script>
    <script src="/js/web3.js"></script>
</head>
<body>
    <div class="login-container">
        <h1>View Winning Bid</h1>
        <form id="view-winner-form">
            <label for="tenderId">Select Tender:</label>
            <select name="tenderId" id="tenderId" required>
                <% activeTenders.forEach(tender => { %>
                    <option value="<%= tender.id %>"><%= tender.name %></option>
                <% }) %>
            </select>
            <button type="submit">View Winner</button>
        </form>
        <div id="winner-details" style="display: none;">
            <h2>Winning Bid Details</h2>
            <p><strong>Winning Bid Amount:</strong> <span id="winningBidAmount"></span> ETH</p>
            <p><strong>Winning Company Name:</strong> <span id="winningCompanyName"></span></p>
            <p><strong>Winning Bidder Address:</strong> <span id="winningBidder"></span></p>
        </div>
    </div>
```

ViewWinnerAndBidAmount.ejs 1

viewWinnerAndBidAmount.ejs

lorinproject > views > bidder > viewWinnerAndBidAmount.ejs > html > body > script > addEventListener('submit') callback

```ejs
<html lang="en">
<body>
    <script nonce="<%= nonce %>">
        document.getElementById('view-winner-form').addEventListener('submit', async function(event) {
            event.preventDefault();
            const tenderId = document.getElementById('tenderId').value;
            try {
                if (typeof contract !== 'undefined') {
                    const result = await contract.methods.viewWinnerAndBidAmount(tenderId).call();
                    const winningBidAmount = web3.utils.fromWei(result[0], 'ether');
                    const winningCompanyName = result[1];
                    const winningBidder = result[2];
                    document.getElementById('winningBidAmount').textContent = winningBidAmount;
                    document.getElementById('winningCompanyName').textContent = winningCompanyName;
                    document.getElementById('winningBidder').textContent = winningBidder;
                    document.getElementById('winner-details').style.display = 'block';
                } else {
                    alert("Contract or web3 is not initialized.");
                }
            } catch (error) {
                console.error("Error fetching winner details:", error);
                alert("Failed to fetch winner details.");
            }
        });
    </script>
</body>
</html>
```

ViewWinnerAndBidAmount.ejs 2

404.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>404 - Page Not Found</title>
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <div class="login-container">
        <h1>404 - Page Not Found</h1>
        <p>Sorry, the page you are looking for does not exist.</p>
        <a href="/">Go back to Home</a>
    </div>
</body>
</html>
```

404.ejs

Routes Code :

adminRoutes.js

```
const express = require("express");
const router = express.Router();
const adminController = require("../controllers/adminController");
const auth = require("../middlewares/auth");

router.get("/login", (req, res) => res.render("adminLogin", { errorMessage: null }));
router.post("/login", adminController.login);
router.get("/dashboard", auth.adminAuth, adminController.dashboard);
router.get("/createTender", auth.adminAuth, (req, res) => res.render("admin/createTender"));
router.post("/createTender", auth.adminAuth, adminController.createTender);
router.get("/activeTenders", auth.adminAuth, adminController.activeTenders);
router.post("/viewAllBids", auth.adminAuth, adminController.viewAllBids);
router.post("/viewWinnerAndBidAmount", auth.adminAuth, adminController.viewWinnerAndBidAmount);
module.exports = router;
```

AdminRoutes.js

bidderRoutes.js

routes > bidderRoutes.js > ...

```
1  const express = require("express");
2  const router = express.Router();
3  const bidderController = require("../controllers/bidderController");
4  const auth = require("../middlewares/auth");
5  const upload = require("../middlewares/upload");
6
7  router.get("/signup", (req, res) => res.render("bidder/signup"));
8  router.post("/signup", bidderController.signup);
9  router.get("/login", (req, res) => res.render("bidderLogin"));
10 router.post("/login", bidderController.login);
11 router.get("/dashboard", auth.bidderAuth, bidderController.dashboard);
12
13 router.get("/activeTenders", auth.bidderAuth, async (req, res) => {
14     try {
15         const activeTenders = await bidderController.getActiveTendersForBidding(req, res);
16         res.render("bidder/viewActiveTenders", { activeTenders });
17     } catch (error) {
18         res.status(500).send("Error fetching active tenders: " + error.message);
19     }
20 });
21
22 router.get("/submitBid", auth.bidderAuth, (req, res) => {
23     const tenderId = req.query.tenderId;
24     res.render("bidder/submitBid", { tenderId });
25 });
26
27 router.post("/submitBid", auth.bidderAuth, upload, (req, res) => {
28     if (!req.files || !req.files.bidFile) {
29         return res.status(400).send('No file was uploaded.');
30     }
31     const bidFile = req.files.bidFile;
32     const uploadPath = '/path/to/save/file/' + bidFile.name;
33     bidFile.mv(uploadPath, (err) => {
34         if (err) {
```

BidderRoutes.js 1

bidderRoutes.js

routes > bidderRoutes.js > ...

```
27 router.post("/submitBid", auth.bidderAuth, upload, (req, res) => {
33     bidFile.mv(uploadPath, (err) => {
35             return res.status(500).send(err);
36         }
37         req.filePath = uploadPath;
38         bidderController.submitBid(req, res);
39     });
40 });
41
42 router.post("/viewAllBids", auth.bidderAuth, bidderController.viewAllBids);
43 router.post("/viewWinnerAndBidAmount", auth.bidderAuth, bidderController.viewWinner);
44 module.exports = router;
```

BidderRoutes.js 2

Controllers Code :

adminController.js

controllers > adminController.js > ...

```
const Tender = require('../models/Tender');
const Bid = require('../models/Bid');
const Admin = require('../models/Admin');

exports.login = (req, res) => {
    const { username, password } = req.body;

    Admin.findByUsername(username).then(admin => {
        if (admin && admin.password === password) {
            req.session.admin = admin;
            res.redirect("/admin/dashboard");
        } else {
            res.render("adminLogin", { errorMessage: "Invalid username or password" });
        }
    }).catch(err => {
        console.error("Login error:", err);
        res.status(500).send("Server error");
    });
};

exports.dashboard = (req, res) => {
    res.render("admin/dashboard", { admin: req.session.admin });
};

exports.createTender = (req, res) => {
    const { name, description, deadline } = req.body;
    const tenderDeadline = new Date(deadline).getTime() / 1000;

    Tender.create(name, description, tenderDeadline).then(() => {
        res.redirect("/admin/activeTenders");
    }).catch(err => {
        console.error("Error creating tender:", err);
        res.status(500).send("Server error");
    });
```

AdminController.js 1

adminController.js

controllers > adminController.js > ...

```
};

exports.activeTenders = (req, res) => {
    Tender.findAllActive().then(tenders => {
        res.render("admin/activeTenders", { activeTenders: tenders });
    }).catch(err => {
        console.error("Error fetching active tenders:", err);
        res.status(500).send("Server error");
    });
};

exports.viewAllBids = (req, res) => {
    const { tenderId } = req.body;

    Bid.findByTenderId(tenderId).then(bids => {
        res.render("admin/viewAllBids", { bids: bids, tenderId });
    }).catch(err => {
        console.error("Error fetching bids:", err);
        res.status(500).send("Server error");
    });
};

exports.viewWinnerAndBidAmount = (req, res) => {
    const { tenderId } = req.body;

    Bid.findWinningBid(tenderId).then(winner => {
        res.render("admin/viewWinnerAndBidAmount", { winner: winner });
    }).catch(err => {
        console.error("Error fetching winner:", err);
        res.status(500).send("Server error");
    });
};
```

AdminController.js 2

```
bidderController.js
controllers > bidderController.js > ...
1  const Tender = require('../models/Tender');
2  const Bid = require('../models/Bid');
3  const Bidder = require('../models/Bidder');
4  const bcrypt = require('bcrypt');
5  const { web3, contract } = require('../config/web3');
6
7  exports.signup = async (req, res) => {
8      const { username, password, confirmPassword, companyName } = req.body;
9      if (!companyName) {
10         return res.status(400).send("Company name is required.");
11     }
12     if (password !== confirmPassword) {
13         return res.status(400).send("Passwords do not match.");
14     }
15     try {
16         const hashedPassword = await bcrypt.hash(password, 10);
17         await Bidder.create(username, hashedPassword, companyName);
18         res.redirect('/bidder/login');
19     } catch (err) {
20         console.error("Error during signup:", err);
21         res.status(500).send("Server error");
22     }
23 };
24
25 exports.login = async (req, res) => {
26     const { username, password } = req.body;
27     try {
28         const bidder = await Bidder.findByUsername(username);
29         if (bidder && await bcrypt.compare(password, bidder.password)) {
30             req.session.bidder = bidder;
31             res.redirect('/bidder/dashboard');
32         } else {
33             res.render('bidderLogin', { errorMessage: "Invalid username or password" });
34         }
```

BidderController.js 1

```
bidderController.js
controllers > bidderController.js > ...
25 exports.login = async (req, res) => {
35     } catch (err) {
36         console.error("Login error:", err);
37         res.status(500).send("Server error");
38     }
39 };
40
41 exports.dashboard = (req, res) => {
42     res.render('bidder/dashboard', { bidder: req.session.bidder });
43 };
44
45 exports.getActiveTendersForBidding = async (req, res) => {
46     try {
47         const tenders = await Tender.findAllActive();
48         res.render('bidder/viewActiveTenders', { activeTenders: tenders });
49     } catch (err) {
50         console.error("Error fetching active tenders:", err);
51         res.status(500).send("Server error");
52     }
53 };
54
55 exports.submitBid = async (req, res) => {
56     const { tenderId, bidAmount } = req.body;
57     const fileHash = req.file.filename;
58     try {
59         const accounts = await web3.eth.getAccounts();
60         await contract.methods
61             .submitBid(tenderId, bidAmount, fileHash, req.session.bidder.companyName)
62             .send({ from: accounts[0] });
63         res.redirect('/bidder/dashboard');
64     } catch (err) {
65         console.error("Error submitting bid:", err);
66         res.status(500).send("Server error");
67     }
```

BidderController.js 2

bidderController.js

controllers > bidderController.js > ...

```
55  exports.submitBid = async (req, res) => {
68  };
69
70  exports.viewAllBids = async (req, res) => {
71      const { tenderId } = req.body;
72      try {
73          const bids = await Bid.findByTenderId(tenderId);
74          res.render('bidder/viewAllBids', { bids });
75      } catch (err) {
76          console.error("Error fetching bids:", err);
77          res.status(500).send("Server error");
78      }
79  };
80
81  exports.viewWinner = async (req, res) => {
82      const { tenderId } = req.body;
83      try {
84          const winner = await Bid.findWinningBid(tenderId);
85          res.render('bidder/viewWinnerAndBidAmount', { winner });
86      } catch (err) {
87          console.error("Error fetching winner details:", err);
88          res.status(500).send("Server error");
89      }
90  };
```

BidderController.js 3

Middlewares Code :

auth.js

middlewares > auth.js > ...

```
1   exports.adminAuth = (req, res, next) => {
2       if (req.session.admin) {
3           next();
4       } else {
5           res.redirect('/admin/login');
6       }
7   };
8
9   exports.bidderAuth = (req, res, next) => {
10      if (req.session.bidder) {
11          next();
12      } else {
13          res.redirect('/bidder/login');
14      }
15  };
```

Auth.js

upload.js

middlewares > upload.js > ...

```
const fs = require('fs');
const path = require('path');
const fileUpload = require('express-fileupload');

const uploadDir = path.join(__dirname, 'uploads');
if (!fs.existsSync(uploadDir)) {
    fs.mkdirSync(uploadDir);
}

module.exports = fileUpload({
    limits: { fileSize: 50 * 1024 * 1024 }, // 50 MB max file size
    safeFileNames: true,
    preserveExtension: true,
    createParentPath: true,
    abortOnLimit: true,
    responseOnLimit: 'File size limit has been reached',
    limitHandler: (req, res) => {
        return res.status(413).send('File size exceeds the maximum limit');
    }
});
```

Upload.js

Web3.js :

web3.js

config > web3.js > ...

```
const Web3 = require("web3");
const TenderContract = require("../contracts/TenderContract.json");
let web3;
let contract;

if (typeof window !== "undefined" && typeof window.ethereum !== "undefined") {
  web3 = new Web3(window.ethereum);
  window.ethereum
    .request({ method: "eth_requestAccounts" })
    .then((accounts) => {
      web3.eth.defaultAccount = accounts[0];
      const contractAddress = "0x625aEFB6eCA664D698916C20733887CE0D9BEC3E";
      contract = new web3.eth.Contract(TenderContract.abi, contractAddress);
    })
    .catch((error) => {
      console.error("Error accessing MetaMask accounts:", error);
    });
} else {
  console.log("MetaMask is not detected. Please install MetaMask.");
}

module.exports = {
  web3,
  contract,
};
```

Web3.js

Database.js :

database.js

config > database.js > ...

```
1  const mysql = require("mysql2");
2  require("dotenv").config();
3
4  const db = mysql.createConnection({
5    host: "localhost",
6    user: "root",
7    password: "",
8    database: "etender",
9  });
10
11 db.connect((err) => {
12   if (err) {
13     console.error("Error connecting to the database:", err.stack);
14     return;
15   }
16   console.log("Connected to the database as ID", db.threadId);
17 });
18 module.exports = db;
```

Database.js

Models Code :

Admin.js

models > Admin.js > ...

```
1  const db = require('../config/database');
2
3  const Admin = {
4      create: (username, hashedPassword) => {
5          const query = 'INSERT INTO admins (username, password) VALUES (?, ?)';
6          return new Promise((resolve, reject) => {
7              db.query(query, [username, hashedPassword], (err, results) => {
8                  if (err) return reject(err);
9                  resolve(results.insertId);
10             });
11         });
12     },
13
14     findByUsername: (username) => {
15         const query = 'SELECT * FROM admins WHERE username = ?';
16         return new Promise((resolve, reject) => {
17             db.query(query, [username], (err, results) => {
18                 if (err) return reject(err);
19                 resolve(results[0]);
20             });
21         });
22     }
23 };
24
25 module.exports = Admin;
```

Admin.js

```
Bidder.js
models > Bidder.js > ...
1  const db = require('../config/database');
2
3  const Bidder = {
4      create: (username, hashedPassword, companyName) => {
5          const query = 'INSERT INTO bidders (username, password, company_name) VALUES (?, ?, ?)';
6          return new Promise((resolve, reject) => {
7              db.query(query, [username, hashedPassword, companyName], (err, results) => {
8                  if (err) return reject(err);
9                  resolve(results.insertId);
10             });
11         });
12     },
13
14     findByUsername: (username) => {
15         const query = 'SELECT * FROM bidders WHERE username = ?';
16         return new Promise((resolve, reject) => {
17             db.query(query, [username], (err, results) => {
18                 if (err) return reject(err);
19                 resolve(results[0]);
20             });
21         });
22     },
23
24     findById: (id) => {
25         const query = 'SELECT * FROM bidders WHERE id = ?';
26         return new Promise((resolve, reject) => {
27             db.query(query, [id], (err, results) => {
28                 if (err) return reject(err);
29                 resolve(results[0]);
30             });
31         });
32     }
33 };
34 module.exports = Bidder;
```

Bidder.js

```
Tender.js
models > Tender.js > ...
1  const db = require('../config/database');
2
3  const Tender = {
4      create: (name, description, deadline) => {
5          const query = 'INSERT INTO tenders (name, description, deadline) VALUES (?, ?, ?)';
6          return new Promise((resolve, reject) => {
7              db.query(query, [name, description, deadline], (err, results) => {
8                  if (err) return reject(err);
9                  resolve(results.insertId);
10             });
11         });
12     },
13
14     findAllActive: () => {
15         const query = 'SELECT * FROM tenders WHERE is_active = TRUE AND winner_selected = FALSE';
16         return new Promise((resolve, reject) => {
17             db.query(query, (err, results) => {
18                 if (err) return reject(err);
19                 resolve(results);
20             });
21         });
22     },
23
24     findById: (id) => {
25         const query = 'SELECT * FROM tenders WHERE id = ?';
26         return new Promise((resolve, reject) => {
27             db.query(query, [id], (err, results) => {
28                 if (err) return reject(err);
29                 resolve(results[0]);
30             });
31         });
32     },
33
```

Tender.js 1

Tender.js 2

Bid.js

Smart Contract Code (TenderContract.sol) :

TenderContract.sol ●

contracts > TenderContract.sol

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.0;
3
4  contract TenderSystem {
5
6      struct Tender {
7          uint id;
8          string name;
9          string description;
10         uint256 deadline;
11         bool isActive;
12         bool winnerSelected;
13         address winningBidder;
14         uint winningBidAmount;
15         string winningCompany;
16     }
17
18     struct Bid {
19         uint tenderId;
20         address bidder;
21         uint bidAmount;
22         string documentHash;
23         string companyName;
24     }
25
26     mapping(uint => Tender) private tenders;
27     mapping(uint => Bid[]) private bids;
28
29     uint private tenderCounter = 0;
30     address public admin;
31
32     modifier onlyAdmin() {
33         require(msg.sender == admin, "Only admin can perform this action");
34         _;
```

TenderContract 1

TenderContract.sol ●

contracts > TenderContract.sol

```
4  contract TenderSystem {
32     modifier onlyAdmin() {
35     }
36
37     modifier onlyBeforeDeadline(uint _tenderId) {
38         require(block.timestamp <= tenders[_tenderId].deadline, "Tender deadline has passed");
39         _;
40     }
41
42     modifier onlyAfterDeadline(uint _tenderId) {
43         require(block.timestamp > tenders[_tenderId].deadline, "Tender deadline has not passed yet");
44         _;
45     }
46
47     modifier tenderExists(uint _tenderId) {
48         require(tenders[_tenderId].id != 0, "Tender does not exist");
49         _;
50     }
51
52     constructor() {
53         admin = msg.sender;
54     }
55
56     function createTender(string memory _name, string memory _description, uint256 _deadline) public onlyAdmin {
57         require(_deadline >= block.timestamp, "Tender deadline must be in the future");
58         tenderCounter++;
59         tenders[tenderCounter] = Tender({
60             id: tenderCounter,
61             name: _name,
62             description: _description,
63             deadline: _deadline,
64             isActive: true,
65             winnerSelected: false,
```

TenderContract 2

TenderContract 3

TenderContract 4

Encryption and Hashing Code :

encrypt.js

lorinproject > utils > encrypt.js > decryptFile

```
1  const crypto = require('crypto');
2  const fs = require('fs');
3  const path = require('path');
4  const Web3 = require('web3');
5  const web3 = new Web3(Web3.givenProvider);
6  const algorithm = 'chacha20-poly1305';
7  const key = crypto.randomBytes(32); // 256-bit key
8  const nonce = crypto.randomBytes(12); // 96-bit nonce
9  // Encrypt a file
10 function encryptFile(filePath, outputFilePath, callback) {
11     const cipher = crypto.createCipheriv(algorithm, key, nonce);
12     const input = fs.createReadStream(filePath);
13     const output = fs.createWriteStream(outputFilePath);
14
15     input.pipe(cipher).pipe(output);
16     output.on('finish', () => {
17         console.log('File encrypted successfully.');
18         callback(null, outputFilePath);
19     });
20     output.on('error', (err) => {
21         console.error('Encryption error:', err);
22         callback(err);
23     });
24 }
```

Encrypt.js 1

encrypt.js

lorinproject > utils > encrypt.js > decryptFile

```
}
// Decrypt a file
function decryptFile(filePath, outputFilePath, callback) {
    const decipher = crypto.createDecipheriv(algorithm, key, nonce);
    const input = fs.createReadStream(filePath);
    const output = fs.createWriteStream(outputFilePath);
    input.pipe(decipher).pipe(output);
    output.on('finish', () => {
        console.log('File decrypted successfully.');
        callback(null);
    });
    output.on('error', (err) => {
        console.error('Decryption error:', err);
        callback(err);
    });
}
// Hash a file using SHA-256
function hashFile(filePath) {
    const fileBuffer = fs.readFileSync(filePath);
    const hash = crypto.createHash('sha256');
    hash.update(fileBuffer);
    return hash.digest('hex');
}
// Hash the bid amount using SHA-256
function hashBidAmount(bidAmount) {
    const hash = crypto.createHash('sha256');
    hash.update(bidAmount.toString());
    return hash.digest('hex');
}
```

Encrypt.js 2

encrypt.js

lorinproject > utils > encrypt.js > decryptFile

```
async function storeBidOnBlockchain(documentHash, bidHash) {
    const contractABI = [ /contracts/TenderContract.json];
    const contractAddress = 0x625aEFB6eCA664D698916C20733887CE0D9BE;
    const accounts = await web3.eth.getAccounts();
    const contract = new web3.eth.Contract(contractABI, contractAddress)
    try {
        const tx = await contract.methods.storeBid(documentHash, bidHash).send({
            from: accounts[0], //
            gas: 3000000 //
        });
        console.log('Transaction confirmed:', tx.transactionHash);
    } catch (error) {
        console.error('Error storing bid on blockchain:', error);
        throw error;
    }
}

module.exports = {
    encryptFile,
    decryptFile,
    hashFile,
    hashBidAmount,
    storeBidOnBlockchain
};
```

Encrypt.js 3

Server Code :

server.js

```
const express = require("express");
const bodyParser = require("body-parser");
const session = require("express-session");
const path = require("path");
const fileUpload = require("express-fileupload");
const helmet = require("helmet");
const db = require("./config/database");
const contract = require("./config/web3");
const crypto = require('crypto');

const adminRoutes = require("./routes/adminRoutes");
const bidderRoutes = require("./routes/bidderRoutes");

const app = express();

app.set("view engine", "ejs");
app.set("views", path.join(__dirname, "views"));

app.use((req, res, next) => {
  const nonce = crypto.randomBytes(16).toString("base64");
  res.locals.nonce = nonce;
  res.setHeader(
    "Content-Security-Policy",
    `
    default-src 'self';
    script-src 'self' 'nonce-${nonce}' https://cdn.jsdelivr.net https://unpkg.com 'unsafe-eval';
    style-src 'self' 'nonce-${nonce}' https://cdn.jsdelivr.net https://fonts.googleapis.com https://cdnjs.cloudflare.com;
    img-src 'self' data: https://cdn.jsdelivr.net;
    connect-src 'self' http://localhost:5001;
    font-src 'self' https://fonts.gstatic.com https://fonts.googleapis.com https://cdnjs.cloudflare.com;
    object-src 'none';
    media-src 'self';
    frame-src 'none';
    `.replace(/\s{2,}/g, " ").trim()
```

Server 1

server.js

server.js > ...

```
19  app.use((req, res, next) => {
35    );
36    next();
37  });
38
39  app.use(bodyParser.urlencoded({ extended: true }));
40  app.use(express.static(path.join(__dirname, "public")));
41  app.use(
42    session({
43      secret: "your-secret-key",
44      resave: false,
45      saveUninitialized: true,
46      cookie: { secure: false },
47    })
48  );
49  app.use(fileUpload());
50
51  app.use(helmet({
52    contentSecurityPolicy: false,
53  }));
54
55  app.use((req, res, next) => {
56    req.db = db;
57    req.contract = contract;
58    next();
59  });
60
61  app.get("/test-db", async (req, res) => {
62    try {
63      const [rows] = await req.db.query("SELECT 1 + 1 AS result");
64      res.send(`Database connected successfully. Result: ${rows[0].result}`);
65    } catch (err) {
66      res.status(500).send(`Database connection failed: ${err.message}`);
```

Server 2

server.js

server.js > ...

```
61  app.get("/test-db", async (req, res) => {
67    }
68  });
69
70  app.use("/admin", adminRoutes);
71  app.use("/bidder", bidderRoutes);
72
73  app.get("/", (req, res) => {
74    res.redirect("/bidder/login");
75  });
76
77  app.get("/logout", (req, res) => {
78    req.session.destroy(() => {
79      res.redirect("/");
80    });
81  });
82
83  app.use((req, res, next) => {
84    res.status(404).render("404", { url: req.originalUrl });
85  });
86
87  app.use((err, req, res, next) => {
88    console.error(err.stack);
89    res.status(500).send("Something went wrong!");
90  });
91
92  const PORT = process.env.PORT || 3000;
93  app.listen(PORT, () => {
94    console.log(`Server is running on port ${PORT}`);
95  });
```

Server 3

Testing Server :

encryption.ejs ●

lorinproject > view > encryption.ejs > html > body > div.container > ? > ? > ? > ? > ? > ? > ? > ? > form

```
2   <html lang="en">
9   <body>
10      <div class="container">
12
13          <% if (message) { %>
14              <p class="message"><%= message %></p>
15          <% } %>
16
17          <% if (downloadLink) { %>
18              <a class="download-link" href="<%= downloadLink %>">Download Processed File</a>
19          <% } %>
20
21          <% if (key && nonce) { %>
22              <div class="key-nonce">
23                  <p><strong>Key (hex):</strong> <%= key %></p>
24                  <p><strong>Nonce (hex):</strong> <%= nonce %></p>
25              </div>
26          <% } %>
27          <form action="/upload" method="POST" enctype="multipart/form-data">
28              <input type="file" name="file" required>
29              <select name="action" required>
30                  <option value="encrypt">Encrypt</option>
31                  <option value="decrypt">Decrypt</option>
32                  <option value="hash">Hash</option>
```

Encryption.ejs 1

```
27          <form action="/upload" method="POST" enctype="multipart/form-data">
28              <input type="file" name="file" required>
29              <select name="action" required>
30                  <option value="encrypt">Encrypt</option>
31                  <option value="decrypt">Decrypt</option>
32                  <option value="hash">Hash</option>
33              </select>
34              <input type="text" name="key" placeholder="Enter key (hex)">
35              <input type="text" name="nonce" placeholder="Enter nonce (hex)">
36              <button type="submit">Process File</button>
37          </form>
38      </div>
39  </body>
40  </html>
```

Encryption.ejs 2

Servers.js

```
servers.js
servers.js > ...
1  const express = require('express');
2  const multer = require('multer');
3  const path = require('path');
4  const crypto = require('crypto');
5  const fs = require('fs');
6
7  const app = express();
8  const upload = multer({ dest: 'uploads/' });
9
10 app.set('view engine', 'ejs');
11 app.set('views', path.join(__dirname, 'view'));
12 app.use(express.static('public'));
13
14 const algorithm = 'chacha20-poly1305';
15
16 function isValidHex(value, expectedLength) {
17     return /^[0-9a-fA-F]+$/.test(value) && value.length === expectedLength;
18 }
19
20 function encryptFile(filePath, outputFilePath) {
21     const key = crypto.randomBytes(32); // Generate a 32-byte key
22     const nonce = crypto.randomBytes(12); // Generate a 12-byte nonce
23     const cipher = crypto.createCipheriv(algorithm, key, nonce, { authTagLength: 16 });
24
25     const input = fs.createReadStream(filePath);
26     const output = fs.createWriteStream(outputFilePath);
27
28     let startTime = Date.now();
29
30     input.pipe(cipher).pipe(output);
31
32     return new Promise((resolve, reject) => {
33         output.on('finish', () => {
34             const authTag = cipher.getAuthTag();
```

Servers.js 1

servers.js
servers.js > ...
20 function encryptFile(filePath, outputFilePath) {
32 return new Promise((resolve, reject) => {
33 output.on('finish', () => {
35 fs.appendFile(outputFilePath, authTag, (err) => {
36 if (err) return reject(err);
37 console.log('File encrypted successfully.');
38 let encryptionTime = Date.now() - startTime;
39 console.log(`Encryption took ${encryptionTime} milliseconds.`);
41 resolve({ key: key.toString('hex'), nonce: nonce.toString('hex') });
42 });
43 });
45 output.on('error', (err) => reject(err));
46 });
47 }
49 function decryptFile(filePath, outputFilePath, key, nonce) {
50 return new Promise((resolve, reject) => {
51 try {
52 key = key.trim();
53 nonce = nonce.trim();
54 if (!isValidHex(key, 64) || !isValidHex(nonce, 24)) {
55 return reject
56 (new Error('Invalid key or nonce format. Key must be 64 hex characters and nonce 24 hex characters.'));
57 }
58 let startTime = Date.now();
59 const fileBuffer = fs.readFileSync(filePath);
60 const authTag = fileBuffer.slice(-16);
61 const encryptedData = fileBuffer.slice(0, -16);
62 const decipher = crypto.createDecipheriv(
63 algorithm,
64 Buffer.from(key, 'hex'),

Servers.js 2

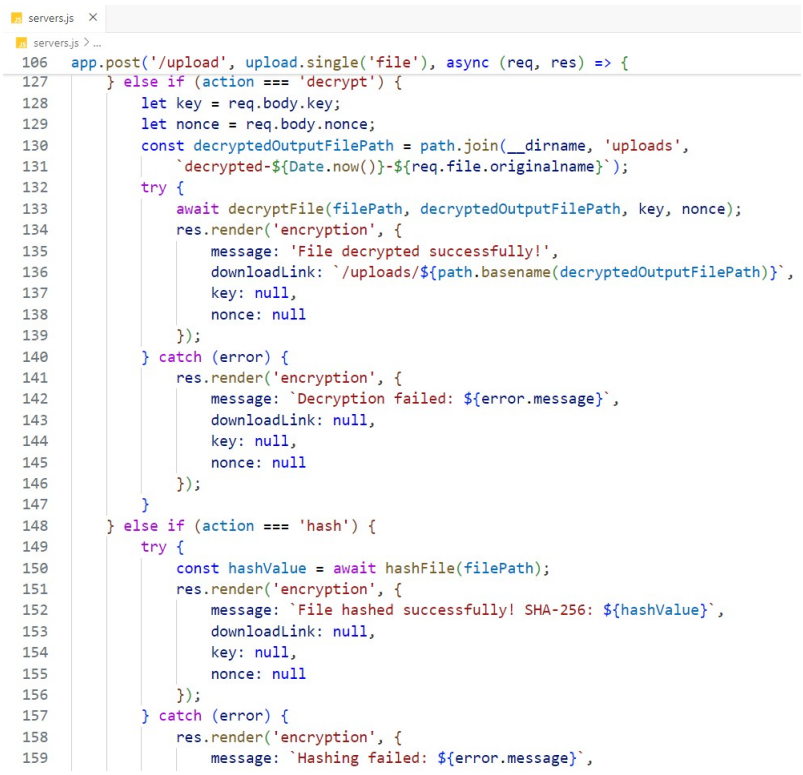
servers.js

```
49  function decryptFile(filePath, outputFilePath, key, nonce) {
50      return new Promise((resolve, reject) => {
62              const decipher = crypto.createDecipheriv(
65                  Buffer.from(nonce, 'hex'),
66                  { authTagLength: 16 }
67              );
68              decipher.setAuthTag(authTag);
69              const output = fs.createWriteStream(outputFilePath);
70              const encryptedStream = fs.createReadStream(filePath, { end: fileBuffer.length - 17 });
71              encryptedStream.pipe(decipher).pipe(output);
72              output.on('finish', () => {
73                  console.log('File decrypted successfully.');
74                  let decryptionTime = Date.now() - startTime;
75                  console.log(`Decryption took ${decryptionTime} milliseconds.`);
76
77                  resolve();
78              });
79
80              output.on('error', (err) => reject(err));
81          } catch (error) {
82              reject(error);
83          }
84      });
85  }
86
87  function hashFile(filePath) {
88      return new Promise((resolve, reject) => {
89          try {
90              const fileBuffer = fs.readFileSync(filePath);
91              const hash = crypto.createHash('sha256');
92              hash.update(fileBuffer);
93              const hashValue = hash.digest('hex');
94              console.log('File hashed successfully:', hashValue);
95              resolve(hashValue);
```

Servers.js 3

servers.js
servers.js > ...
G:\project\servers.js
87 function encryptFile(filePath) {
88 return new Promise((resolve, reject) => {
96 } catch (error) {
97 reject(error);
98 }
99 });
100 }
101
102 app.get('/', (req, res) => {
103 res.render('encryption', { message: null, downloadLink: null, key: null, nonce: null });
104 });
105
106 app.post('/upload', upload.single('file'), async (req, res) => {
107 const filePath = req.file.path;
108 const action = req.body.action;
109 const outputFilePath = path.join(__dirname, 'uploads', `output-${Date.now()}-${req.file.originalname}`);
110 if (action === 'encrypt') {
111 try {
112 const { key, nonce } = await encryptFile(filePath, outputFilePath);
113 res.render('encryption', {
114 message: 'File encrypted successfully! Here is your key and nonce:',
115 downloadLink: `/uploads/${path.basename(outputFilePath)}`,
116 key: key,
117 nonce: nonce
118 });
119 } catch (error) {
120 res.render('encryption', {
121 message: `Encryption failed: ${error.message}`,
122 downloadLink: null,
123 key: null,
124 nonce: null
125 });
126 }

Servers.js 4



```
servers.js
servers.js > ...
106  app.post('/upload', upload.single('file'), async (req, res) => {
127      } else if (action === 'decrypt') {
128          let key = req.body.key;
129          let nonce = req.body.nonce;
130          const decryptedOutputFilePath = path.join(__dirname, 'uploads',
131              `decrypted-${Date.now()}-${req.file.originalname}`);
132          try {
133              await decryptFile(filePath, decryptedOutputFilePath, key, nonce);
134              res.render('encryption', {
135                  message: 'File decrypted successfully!',
136                  downloadLink: `/uploads/${path.basename(decryptedOutputFilePath)}`,
137                  key: null,
138                  nonce: null
139              });
140          } catch (error) {
141              res.render('encryption', {
142                  message: `Decryption failed: ${error.message}`,
143                  downloadLink: null,
144                  key: null,
145                  nonce: null
146              });
147          }
148      } else if (action === 'hash') {
149          try {
150              const hashValue = await hashFile(filePath);
151              res.render('encryption', {
152                  message: `File hashed successfully! SHA-256: ${hashValue}`,
153                  downloadLink: null,
154                  key: null,
155                  nonce: null
156              });
157          } catch (error) {
158              res.render('encryption', {
159                  message: `Hashing failed: ${error.message}`,
```

Servers.js 5

```
servers.js
servers.js > ...
106  app.post('/upload', upload.single('file'), async (req, res) => {
160                      downloadLink: null,
161                      key: null,
162                      nonce: null
163                  });
164              }
165          }
166  });
167
168  app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
169  app.listen(3001, () => {
170      console.log('Server is running on http://localhost:3001');
171  });
```

Servers.js 6