Blockchain for Central Bank Smart Contract

Documentation Report

Submitted by

Team ID	**NM2023TMID06094**
Team Lead	**Anandu Ashokan**
Team Member 01	**Eby Thomas**
Team Member 02	**Gokul B T**
Team Member 03	**Nikhil P A**

INDEX

1. INTRODUCTION

In the realm of healthcare innovation, where the demands for security, interoperability, and accessibility converge, emerges a groundbreaking solution – "**Blockchain for Central Bank Smart contract**." At its core, this project pioneers the fusion of two transformative technologies: blockchain and central bank smart contract. Understanding the nuances of these technologies is pivotal to comprehending the revolutionary impact our project aims to achieve. Blockchain, often regarded as the cornerstone of the digital revolution, is a decentralized and distributed ledger technology. In essence, it's a chain of blocks, each containing a record of transactions. What makes blockchain revolutionary is its decentralized nature. These nodes work collaboratively to validate and record transactions, ensuring transparency, security, and immutability. Through cryptographic techniques, transactions on a blockchain are secured, making it virtually impossible for unauthorized entities to alter the data. This immutability ensures the integrity of information, a crucial attribute when dealing with sensitive data .

1.1 PROJECT OVERVIEW

Automation of Monetary Policy: Develop smart contracts that can automate the execution of monetary policies, such as setting interest rates, controlling money supply, and managing inflation.

Digital Currency Management: Create smart contracts to issue, distribute, and manage digital currencies, like Central Bank Digital Currencies (CBDCs), in a secure and efficient manner.

Financial Market Oversight: Develop smart contracts that enable the central bank to monitor and regulate financial markets, including setting trading rules and managing interbank transactions.

Data Security and Privacy: Implement robust security measures and privacy-enhancing features to safeguard sensitive financial data and transactions.

Interoperability: Ensure compatibility and interoperability with existing financial systems, blockchain networks, and regulatory frameworks.

Smart Contract Development: Design and develop smart contracts using a suitable blockchain platform (e.g., Ethereum, Hyperledger Fabric, or a custom blockchain solution).

User Interface: Create a user-friendly interface for central bank personnel to interact with and control the smart contracts.

Blockchain Integration: Establish connections with other blockchain networks, financial institutions, and regulatory bodies to ensure seamless data exchange and compliance.

Security: Implement robust security protocols, encryption, and authentication mechanisms to protect the smart contract and the associated data.

Testing and Quality Assurance: Conduct thorough testing, including code audits, vulnerability assessments, and performance testing to ensure the reliability and efficiency of the smart contracts.

Regulatory Compliance: Ensure that the smart contracts adhere to all relevant regulatory and legal requirements.

Education and Training: Provide training and educational resources for central bank employees to understand and operate the smart contracts effectively.

1.2 PURPOSE

The purpose of central bank smart contracts is to enhance the efficiency, transparency, and security of central banking operations. These smart contracts are designed to automate various functions and processes within central banks, contributing to improved monetary policy implementation, financial market oversight, and the management of digital currencies. Here are the primary purposes of central bank smart contracts:

Monetary Policy Implementation: Central banks use smart contracts to automate the execution of monetary policies. They can set and adjust interest rates, manage money supply, and control inflation more effectively through programmable rules embedded in smart contracts. This automation reduces the need for manual intervention and enhances the precision and timeliness of policy implementation.

Digital Currency Management: Central banks may issue and manage digital currencies, such as Central Bank Digital Currencies (CBDCs), using smart contracts. These contracts handle tasks like issuance, distribution, and tracking of digital currency transactions, making it easier to regulate and monitor the circulation of digital money.

Financial Market Oversight: Smart contracts allow central banks to monitor and regulate financial markets more efficiently. They can establish rules and execute market interventions automatically to maintain stability and prevent financial crises. For example, they can manage interbank transactions and enforce trading regulations using smart contracts

.

2. LITERATURE SURVEY

A literature survey on central bank smart contracts can provide you with valuable insights into the existing research and developments in this field. Central banks around the world have been exploring the use of smart contracts, blockchain technology, and digital currencies to enhance monetary policy, payment systems, and financial stability. Here's a list of key academic papers and rep that can help you gain a comprehensive understanding of central bank smart contracts:.

2.1 EXISTING PROBLEM

Privacy and Security Concerns : Ensuring the privacy and security of transactions is a significant challenge. Smart contracts typically operate on public blockchains, which may not be suitable for handling sensitive financial data. Central banks need to find ways to implement privacy-enhancing technologies without compromising the transparency and security of transactions.

Scalability : As blockchain networks grow, scalability becomes a pressing issue. Smart contracts on certain blockchain platforms may suffer from network congestion, slow transaction processing times, and high fees. This scalability problem needs to be addressed to handle a large volume of transactions efficiently.

2.2 REFERENCE

Title	Year	Authors
Central Bank Digital Currency and Fintech: A Literature Review" □ This paper provides a comprehensive review of the literature on central bank	**2019**	Jonathan Chiu ,Mohammed Davoodalhosseini

• Discusses the potential role of central banks in digital innovation, including the use of smart contracts and blockchain technology.	**2017**	Morten Bech, Rodney Garatt
"The Role of Central Bank Digital Currencies in Monetary Policy" • Examines how central bank digital currencies, including those utilizing smart contracts	**2019**	Barry Eichengreen

2.3 PROBLEM STATEMENT DEFINITION

"The current manual and paper-based processes within central banks for managing monetary policy, currency issuance, and financial transactions are inefficient, prone to errors, and lack transparency. The need for a secure, automated, and transparent solution to streamline these critical functions and ensure compliance with regulatory requirements is evident. Therefore, there is a pressing need to develop a central bank smart contract system that leverages blockchain technology to digitize and automate various central banking operations while maintaining security, auditability, and regulatory compliance."

This problem statement highlights the key issues in traditional central banking processes and identifies the need for a smart contract-based solution to address these challenges

3. IDEATION & PROPOSED SOLUTION

Ideation for a central bank smart contract system involves brainstorming potential features, use cases, and benefits of implementing blockchain-based smart contracts in central banking. Here are some ideas for a central bank smart contract system:

Monetary Policy Automation: Develop smart contracts that automatically adjust interest rates, money supply, and other monetary policy parameters based on predefined economic indicators. This could help in responding more quickly to economic fluctuations and automating decision-making processes.

.

Proposed Solution:

.

Central Bank Digital Currency (CBDC) Issuance:

Develop smart contracts for the issuance, management, and redemption of CBDCs. These contracts should integrate robust identity verification and anti-counterfeiting features.

Enable secure and instant transactions with CBDCs while ensuring regulatory compliance.

.

Regulatory Compliance:

Implement smart contracts that enforce financial regulations, such as Anti-Money Laundering (AML) and Know Your Customer (KYC) requirements, in real-time.

Automate the reporting and auditing processes for regulatory authorities.

.

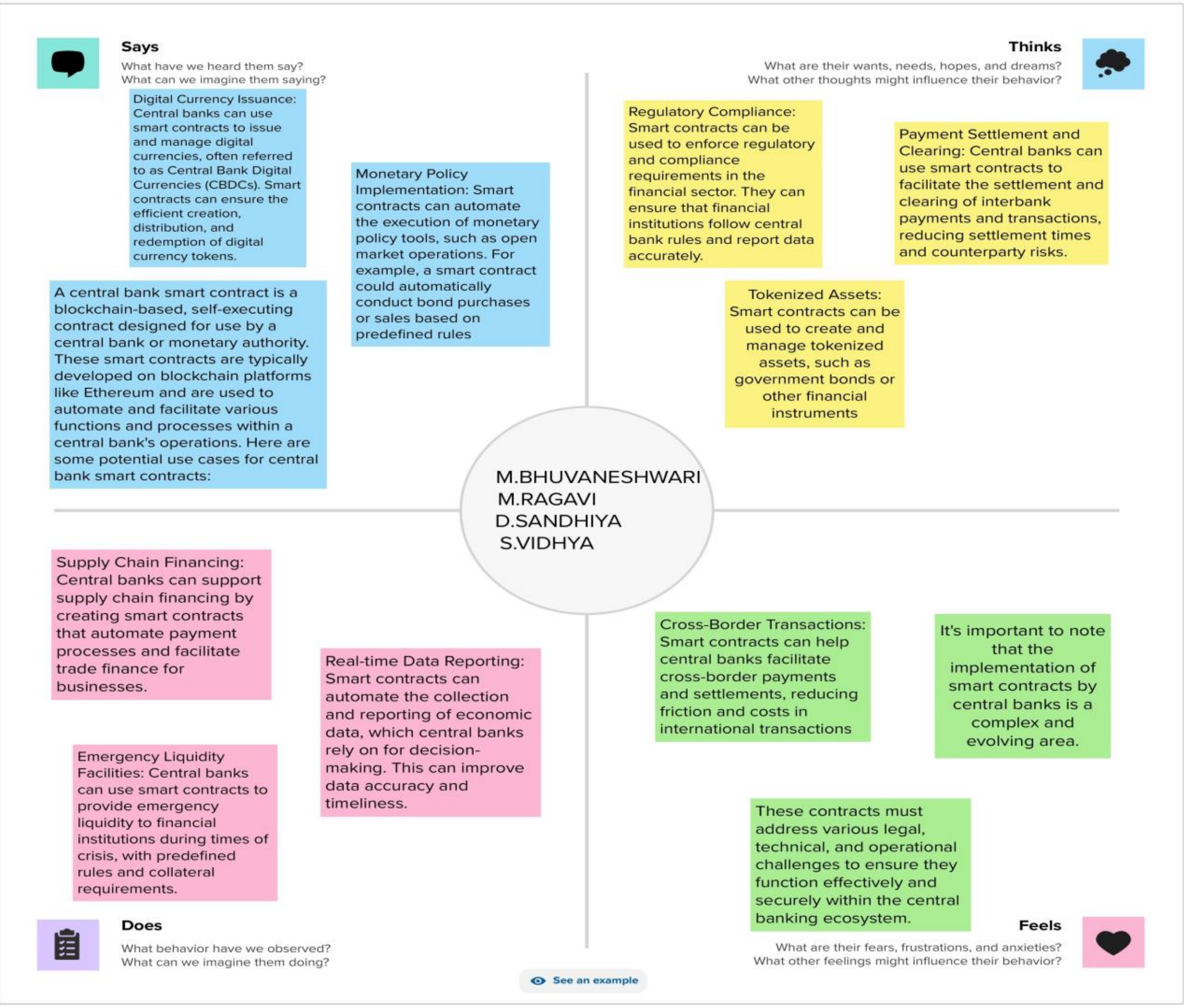
Cross-Border Payments

Create smart contracts for cross-border payments that facilitate instant and low-cost

international transactions, utilizing CBDCs or tokenized assets.Ensure seamless integration with other central banks' smart contract systems for cross-border transactions.

3.1 EMPATHY MAP CANVAS

An empathy map is a visual tool used to understand and empathize with users' experiences, thoughts, feelings, and needs. It helps project teams gain deeper insights into their users' perspectives. The map typically includes sections for what users see, hear, think and feel, say and do, and their pains and gains. By considering these aspects, teams can develop a more profound understanding of user behavior and create products or solutions tailored to users' genuine needs and emotions.



Says
What have we heard them say?
What can we imagine them saying?

Digital Currency Issuance: Central banks can use smart contracts to issue and manage digital currencies, often referred to as Central Bank Digital Currencies (CBDCs). Smart contracts can ensure the efficient creation, distribution, and redemption of digital currency tokens.

Monetary Policy Implementation: Smart contracts can automate the execution of monetary policy tools, such as open market operations. For example, a smart contract could automatically conduct bond purchases or sales based on predefined rules

A central bank smart contract is a blockchain-based, self-executing contract designed for use by a central bank or monetary authority. These smart contracts are typically developed on blockchain platforms like Ethereum and are used to automate and facilitate various functions and processes within a central bank's operations. Here are some potential use cases for central bank smart contracts:

Thinks
What are their wants, needs, hopes, and dreams?
What other thoughts might influence their behavior?

Regulatory Compliance: Smart contracts can be used to enforce regulatory and compliance requirements in the financial sector. They can ensure that financial institutions follow central bank rules and report data accurately.

Payment Settlement and Clearing: Central banks can use smart contracts to facilitate the settlement and clearing of interbank payments and transactions, reducing settlement times and counterparty risks.

Tokenized Assets: Smart contracts can be used to create and manage tokenized assets, such as government bonds or other financial instruments

M.BHUVANESHWARI
M.RAGAVI
D.SANDHIYA
S.VIDHYA

Supply Chain Financing: Central banks can support supply chain financing by creating smart contracts that automate payment processes and facilitate trade finance for businesses.

Real-time Data Reporting: Smart contracts can automate the collection and reporting of economic data, which central banks rely on for decision-making. This can improve data accuracy and timeliness.

Emergency Liquidity Facilities: Central banks can use smart contracts to provide emergency liquidity to financial institutions during times of crisis, with predefined rules and collateral requirements.

Does
What behavior have we observed?
What can we imagine them doing?

Cross-Border Transactions: Smart contracts can help central banks facilitate cross-border payments and settlements, reducing friction and costs in international transactions

It's important to note that the implementation of smart contracts by central banks is a complex and evolving area.

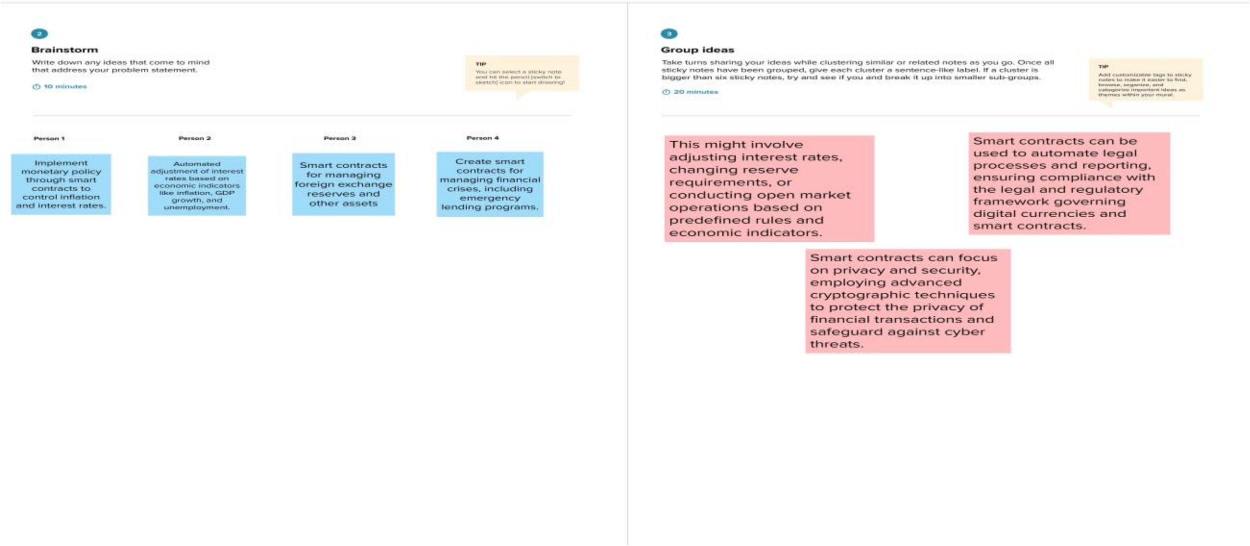
These contracts must address various legal, technical, and operational challenges to ensure they function effectively and securely within the central banking ecosystem.

Feels
What are their fears, frustrations, and anxieties?
What other feelings might influence their behavior?
See an example

3.2 **IDEATION & BRAINSTORMING**

Ideation and brainstorming are creative techniques used to generate a diverse range of ideas for solving a problem or exploring new opportunities. During ideation, participants engage in a free-flowing, non-judgmental exchange of ideas. By encouraging open thinking and collaborative input, diverse concepts emerge. Brainstorming sessions often involve structured activities or discussions, sparking creativity and innovation within a team. These processes are essential for generating innovative solutions and fostering a collaborative, creative environment.

Idea listing



2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Person 1

Implement monetary policy through smart contracts to control inflation and interest rates.

Person 2

Automated adjustment of interest rates based on economic indicators like inflation, GDP growth, and unemployment.

Person 3

Smart contracts for managing foreign exchange reserves and other assets

Person 4

Create smart contracts for managing financial crises, including emergency lending programs.

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

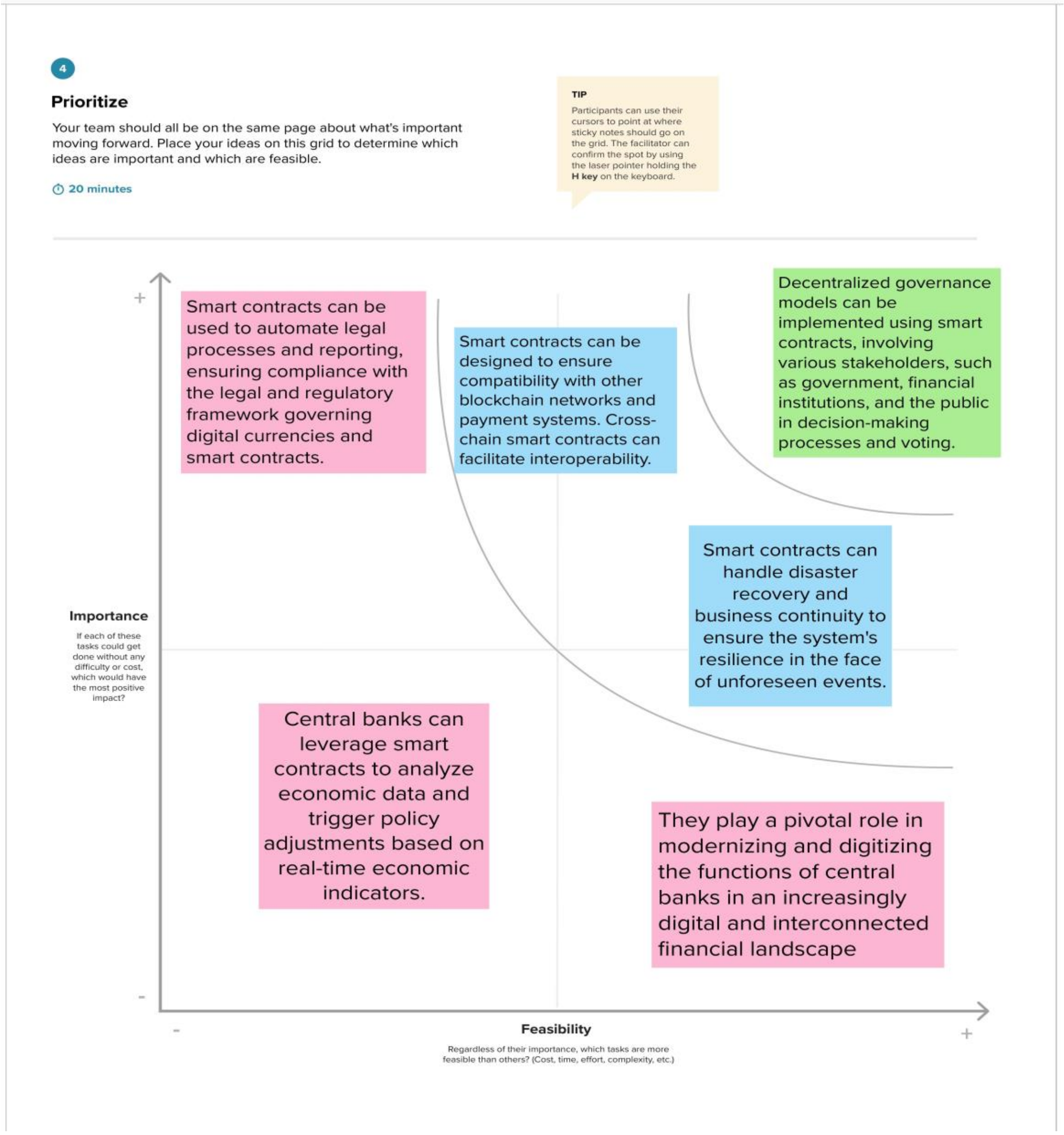
TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

This might involve adjusting interest rates, changing reserve requirements, or conducting open market operations based on predefined rules and economic indicators.

Smart contracts can be used to automate legal processes and reporting, ensuring compliance with the legal and regulatory framework governing digital currencies and smart contracts.

Smart contracts can focus on privacy and security, employing advanced cryptographic techniques to protect the privacy of financial transactions and safeguard against cyber threats.



4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.

Smart contracts can be used to automate legal processes and reporting, ensuring compliance with the legal and regulatory framework governing digital currencies and smart contracts.

Smart contracts can be designed to ensure compatibility with other blockchain networks and payment systems. Cross-chain smart contracts can facilitate interoperability.

Decentralized governance models can be implemented using smart contracts, involving various stakeholders, such as government, financial institutions, and the public in decision-making processes and voting.

Smart contracts can handle disaster recovery and business continuity to ensure the system's resilience in the face of unforeseen events.

Central banks can leverage smart contracts to analyze economic data and trigger policy adjustments based on real-time economic indicators.

They play a pivotal role in modernizing and digitizing the functions of central banks in an increasingly digital and interconnected financial landscape

+
Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
-
-
Feasibility
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)
+

4. REQUIREMENT ANALYSIS

Requirement analysis is a critical phase in the software development lifecycle where project teams gather, document, and analyze the needs and expectations of stakeholders. This process forms the foundation for designing and developing a system that fulfills these requirements effectively. It involves understanding the project's scope, objectives, and constraints, as well as the functional and non-functional requirements.

4.1 FUNCTIONAL REQUIREMENTS

Crisis Management:

Implement smart contracts for crisis management that automatically execute predefined actions in response to financial downturns or emergencies.

Integrate communication and coordination with government agencies and financial institutions.

Identity Verification:

Utilize blockchain-based identity verification solutions to secure and streamline user identity verification for financial transactions. Ensure compliance with regulatory identity verification requirements.

.

Smart Bonds:
Create smart contracts for issuing and managing smart bonds that automatically make interest and principal payments based on predefined criteria. Streamline bond management, reduce administrative overhead, and minimize errors.

.

Environmental and Social Impact Monitoring:

Develop smart contracts and data tracking mechanisms to monitor the environmental and social impact of central bank operations.

Provide transparent reporting and analysis of these impacts, aligning with sustainable finance goals.

4.2 NON-FUNCTIONAL REQUIREMENTS

.

Monitoring and Reporting:

Real-time Monitoring: Provide real-time monitoring tools to track the health, performance, and security of the system.

Alerting: Set up alerting systems to notify administrators of critical issues or anomalies.

.

Data Privacy:

User Privacy: Protect user data and transaction information, adhering to data privacy regulations.

GDPR Compliance: If applicable, ensure compliance with the General Data Protection Regulation (GDPR) or relevant data protection laws.

.

Environmental Impact:

Energy Efficiency: Consider energy-efficient blockchain solutions to minimize the environmental impact of blockchain operations.

Sustainability: Align the system with sustainability and environmental goals as required by the central bank's mission.

.

User Experience:

Usability: Create a user-friendly interface for central bank staff, financialinstitutions, and other authorized entities to interact with the smart contract system.

Performance Responsiveness: Ensure that transaction confirmation times are reasonable to provide a responsive user experience.

.

Disaster Recovery:

Backup and Recovery: Establish robust data backup and recovery mechanisms to safeguard against data loss or system failure.

Redundancy: Implement redundancy at critical system components to ensure continued operation in the event of hardware or software failures.

.

Monitoring and Reporting:

Real-time Monitoring: Provide real-time monitoring tools to track the health, performance, and security of the system.

Alerting: Set up alerting systems to notify administrators of critical issues or anomalies.

.

Data Privacy:

User Privacy: Protect user data and transaction information, adhering to data privacy regulations.

GDPR Compliance: If applicable, ensure compliance with the General Data Protection Regulation (GDPR) or relevant data protection laws.

.

Environmental Impact:

Energy Efficiency: Consider energy-efficient blockchain solutions to minimize the environmental impact of blockchain operations.

Sustainability: Align the system with sustainability and environmental goals as required by the central bank's mission.

These non-functional requirements are essential for ensuring the overall effectiveness, security, and compliance of a central bank smart contract system, as well as its ability to meet the central bank's operational and regulatory needs.

5. PROJECT DESIGN

The design of "Blockchain for Central Bank Smart Contract" focuses on creating a user-friendly, secure, and efficient platform. Here's a high-level overview of the design components:

User Interface (UI):

Dashboard: A centralized dashboard for healthcare providers and patients displaying relevant information and quick access to features.

Authentication: Secure login and authentication mechanisms, incorporating both traditional methods and blockchain-based authentication for enhanced security.

Permission Management: Intuitive interfaces for users to manage permissions, controlling who can access specific parts of their records.

Backend:

Blockchain Integration: Integration with a blockchain network (like Ethereum) for storing encrypted patient records securely.

Smart Contracts: Smart contracts governing data access, ensuring only authorized users can read or modify specific data.

Data Encryption: Implementation of advanced encryption algorithms for end-to-end data security.

Interoperability: Standardized data formats and APIs for seamless data exchange with other healthcare systems.

Metamask Integration: Integration of Metamask to enable secure and user-friendly blockchain interactions, simplifying the user experience**.**

5.1 DATA FLOW DIAGRAMS

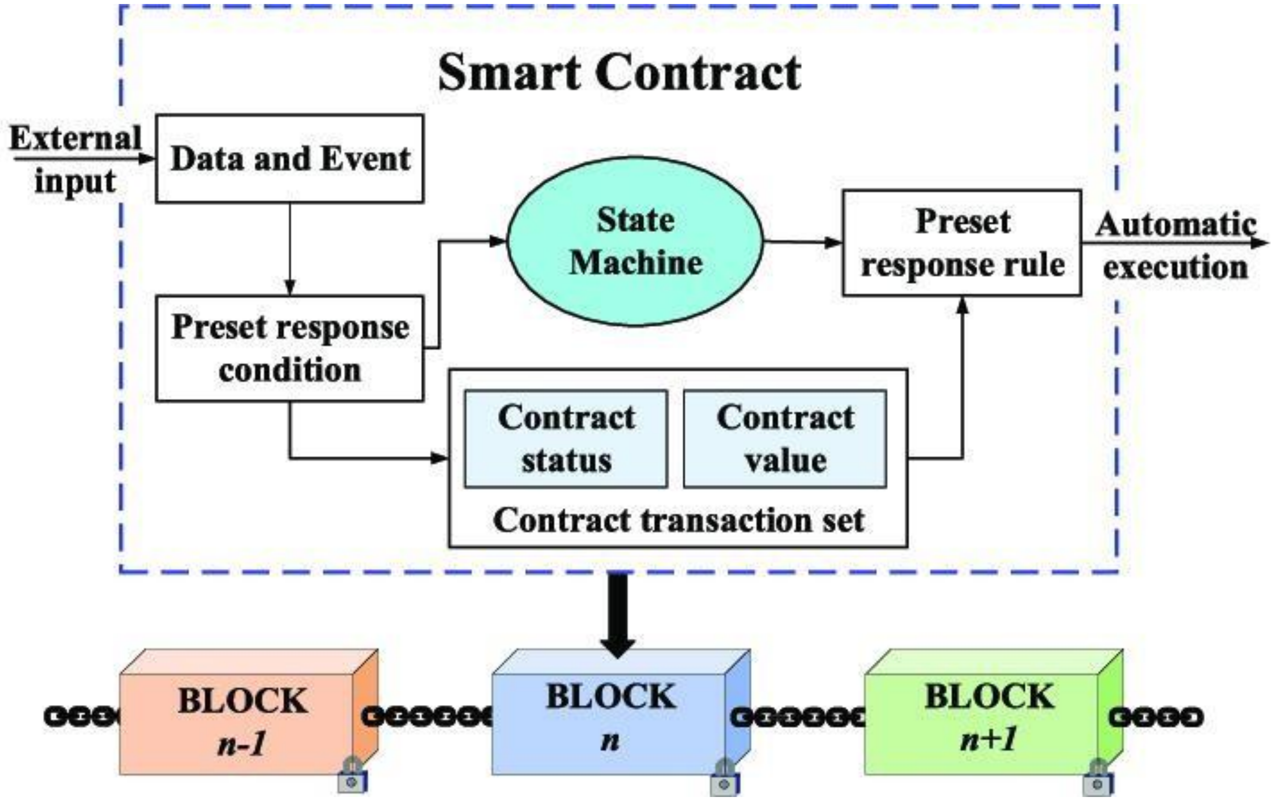
A Data Flow Diagram (DFD) for the system would illustrate the flow of information within the platform.

For example:

Processes: User authentication, data encryption, smart contract execution, permission management, etc

Data Flows: User data requests, blockchain transactions, encrypted data transmission, etc.

External Entities:bank,digital currency and the blockchain network.



5.2 SOLUTION ARCHITECTURE

The solution architecture for "Blockchain for Electronic Health Record" is designed to provide a secure, interoperable, and user-friendly platform for managing electronic health records (EHR). Here's an overview of the solution's architecture:

1. **Presentation Layer:**

   User Interface (UI): Developed using React and HTML/CSS, the UI provides intuitive dashboards .

Authentication Module: Integrates Metamask for blockchain-based authentication and ensures secure login for users.

2. **Application Layer:**

   Backend Services: Built using Node.js, the backend services handle user requests, business logic, and communication with the blockchain network.

   Smart Contracts: Implemented in Solidity (Ethereum's smart contract language), these contracts govern access control, data storage, and encryption. They enforce rules for data interactions on the Ethereum blockchain.

3. **Data Layer:**

   Blockchain Network: Utilizes Ethereum or a similar blockchain network for decentralized and immutable storage of patient records. Each patient record is encrypted and stored as a transaction on the blockchain, ensuring data integrity and security.

   Decentralized File Storage : Integrates with IPFS (InterPlanetary File System) for storing larger files, such as medical images or reports, off-chain while maintaining their integrity and availability.

4. Security Layer:

<u>Data Encryption:</u> Employs advanced encryption algorithms (AES, RSA) to encrypt patient records before storing them on the blockchain, ensuring confidentiality and privacy.

<u>Access Control:</u> Smart contracts enforce access control policies, specifying who can read, update, or delete specific patient records. Permission management is decentralized and transparent.

<u>DDoS Protection:</u> Implements DDoS protection mechanisms to safeguard the system from distributed denial-of-service attacks.

5. Integration Layer:

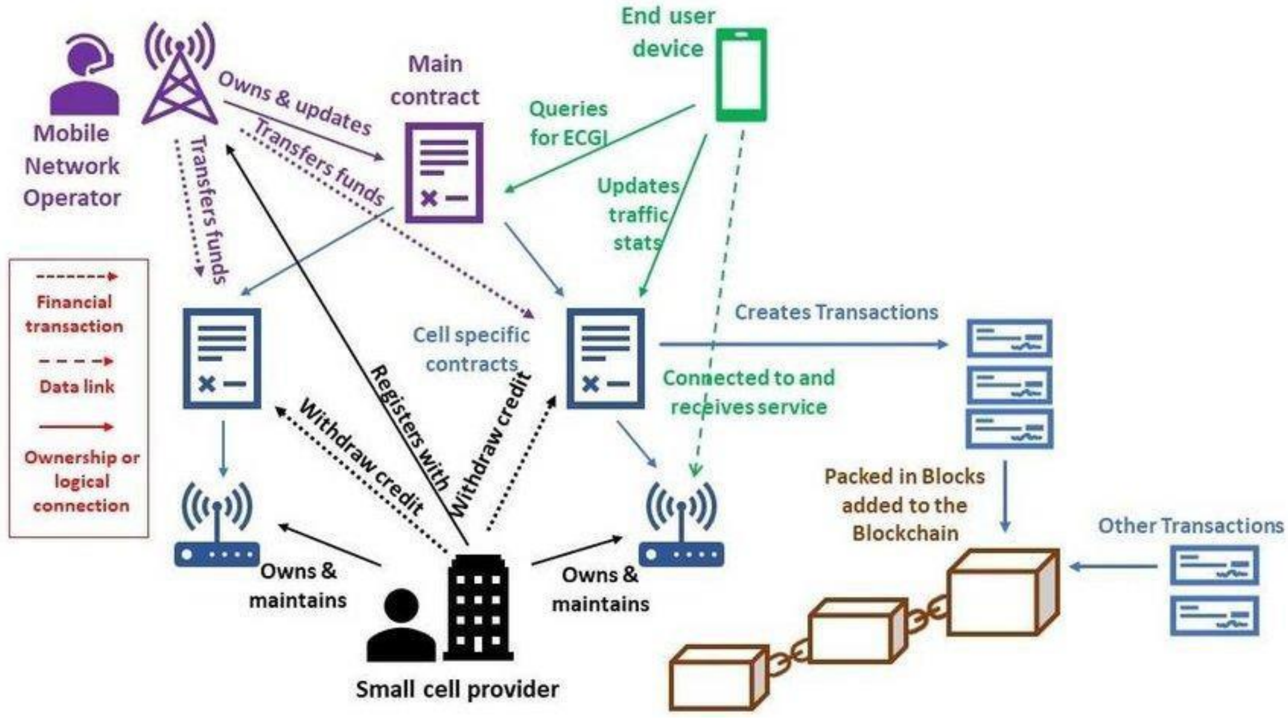
<u>APIs:</u> Provides RESTful APIs for integration with external systems, allowing interoperability with healthcare providers, labs, and pharmacies.

<u>Blockchain API:</u> Interfaces with the blockchain network via Web3.js, enabling seamless interaction between the application and the Ethereum blockchain.

5. Monitoring and Management:

<u>Logging and Monitoring:</u> Implements logging mechanisms to track user activities and system events. Utilizes monitoring tools for real-time performance analysis and issue detection.

System Administration: Provides a dashboard for system administrators to manage user accounts, monitor system health, and deploy updates.



6. PROJECT PLANNING & SCHEDULING

Clearly outlining project goals, stakeholders, and boundaries to establish project focus.

- **Requirement Analysis and Documentation:**

  Gathering and documenting detailed project requirements, including user stories and system features.

- **Task Breakdown and Work Breakdown Structure (WBS):**

  Breaking down project tasks into smaller components and creating a hierarchical structure for tasks and dependencies.

- **Resource Allocation:**

  Assembling a skilled team and assigning specific roles and responsibilities based on expertise.

- **Project Timeline and Milestones:**

  Defining key project milestones and creating a detailed timeline for task completion.

- **Risk Management:**

Identifying potential risks and developing strategies to mitigate them.

- **Development and Iterative Testing:**

  Working in iterative sprints for continuous development and testing of features.

- **Quality Assurance and User Acceptance Testing:**

  Conducting rigorous testing, including functional, performance, and user acceptance testing.

- **Deployment and Implementation:**

Creating a deployment plan and deploying the system in stages, ensuring a smooth transition.

- **Monitoring and Maintenance:**

  Implementing monitoring tools to track system performance and scheduling regular maintenance for updates and bug fixes.

- **Documentation and Knowledge Transfer:**

  Creating technical documentation and conducting knowledge transfer sessions for team members' understanding.

6.1 TECHNICAL ARCHITECTURE

The technical architecture of "Blockchain for Electronic Health Record" is designed to ensure a robust, secure, and scalable system. Here's an overview of the technical components and their interactions:

1. Frontend:

   User Interface (UI): Developed using React.js, providing an intuitive

Authentication: Implements secure login mechanisms, including traditional username/password and blockchain-based authentication methods.

2. Backend Services:

   Node.js Server: Acts as the backend server, handling user requests, processing business logic, and communicating with the blockchain network.

   APIs: Provides RESTful APIs for seamless integration with external systems and services, ensuring interoperability.

Smart Contract Integration: Utilizes Web3.js to interact with smart contracts on the Ethereum blockchain, enabling secure data transactions and access control.

3. <u>Blockchain Integration:</u>

   Ethereum Blockchain: Utilizes the Ethereum blockchain for decentralized storage of encrypted bank records. Smart contracts define access control rules and data interactions.

4. <u>Security Measures:</u>

   Data Encryption: Employs advanced encryption algorithms (AES, RSA) to encrypt patient records before storing them on the blockchain, ensuring confidentiality and privacy.

   Access Control: Smart contracts manage access permissions, ensuring that only authorized users can view or modify specific patient data.

   Secure Communication: Utilizes HTTPS and other secure communication protocols to protect data transmission between the frontend, backend, and blockchain nodes.

5. <u>External Integrations:</u>

   Metamask Integration: Allows secure blockchain interactions and ensures user authentication while interacting with the Ethereum network.

External APIs: Integrates with external bank details' APIs for data exchange, enabling seamless collaboration and information sharing.

6. Scalability and Performance:

   Load Balancing: Implements load balancing techniques to distribute incoming traffic across multiple servers, ensuring system stability and performance under varying loads.

7. Monitoring and Analytics:

   Logging and Monitoring: Implements logging mechanisms to capture user activities and system events. Utilizes monitoring tools for real-time performance analysis and issue detection.

   Analytics: Integrates analytics tools to gain insights into user behavior, system usage, and performance metrics for continuous improve

6.2 SPRINT PLANNING & ESTIMATION

Sprint planning is a meeting held at the beginning of each sprint in Agile development. During this session, the team discusses and prioritizes the tasks to be completed in the upcoming sprint. It involves selecting user stories from the product backlog, breaking them down into smaller tasks, estimating the effort required, and defining the sprint goals. Sprint planning ensures a clear direction for the team, aligning everyone on what needs to be accomplished within the sprint.

Estimation:

Estimation in Agile involves predicting how much effort a task or user story will require. It's typically done using story points or time-based estimates like hours or days. Team members collectively estimate the complexity and effort of tasks during sprint planning. Estimation helps teams understand the workload, plan capacity, and make informed decisions on what can be achieved within a sprint. It provides a basis for prioritization and ensures a realistic approach to task completion within the given timeframe.

6.3 SPRINT DELIVERY SCHEDULE

- **Regular Intervals:** Sprints occur at regular intervals, with a consistent duration agreed upon by the team.

- **Sprint Goals:** Each sprint begins with sprint planning, where specific goals and tasks are defined based on the prioritized backlog items.

- **Development Phase:** During the sprint, the team works on the planned tasks, ensuring they align with the sprint goals.

- **Daily Standups:** Daily standup meetings are conducted to track progress, discuss challenges, and make necessary adjustments.

- **Sprint Review:** At the end of the sprint, a sprint review meeting is held to showcase the completed work to stakeholders and gather feedback.

- **Sprint Retrospective:** A retrospective meeting allows the team to reflect on the sprint, identify areas for improvement, and plan for the next sprint.

- **Delivery:** The completed and tested user stories, features, or bug fixes are delivered to stakeholders and may be deployed to production, depending on the project's release schedule.

- **Next Sprint Planning:** Following the sprint review and retrospective, the team conducts sprint planning for the next sprint, defining new goals and tasks based on updated priorities.

7.CODING & SOLUTIONING

Coding refers to the process of translating a software design into a functional program using programming languages like JavaScript, Python, or Java. It involves writing code, debugging, and optimizing algorithms to create a solution for a specific problem or requirement.

Solutioning involves designing a comprehensive solution strategy before coding. It includes problem analysis, architectural design, selecting appropriate technologies, and planning for scalability and security. Solutioning ensures that the software addresses the problem effectively and aligns with the project's goals.

7.1FEATURE 1

Feature 1: User Authentication

```
function authenticateUser(username, password) {

// Code to validate username and password
```

```
                    if (isValidCredentials(username, password)) {
                    return "Authentication successful";
                     }
                   else {
                            return "Authentication failed";
                             }
                   }
function isValidCredentials(username, password) {
   // Code to check credentials against database or backend
system
   // Return true if valid, false otherwise
}
```

Explanation:

The code snippet checks the provided username and password against a database or backend system. If the credentials are valid, the function returns "Authentication successful"; otherwise, it returns "Authentication failed." This feature ensures secure user access to the system.

7.2 FEATURE 2

Feature 2: Data Encryption

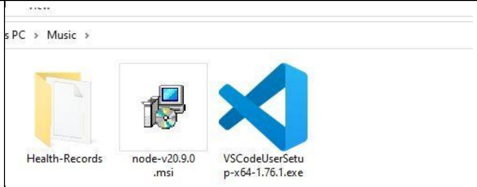
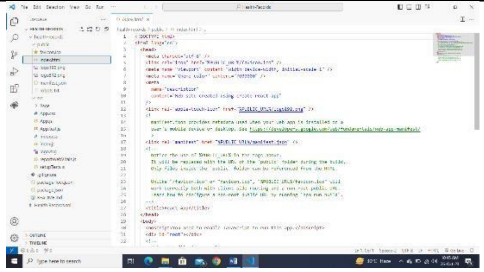
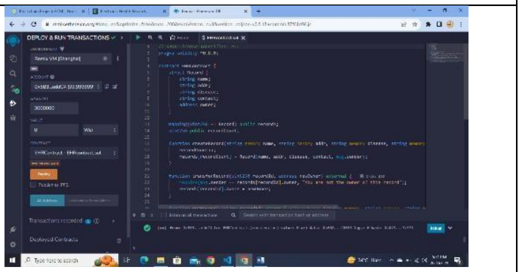
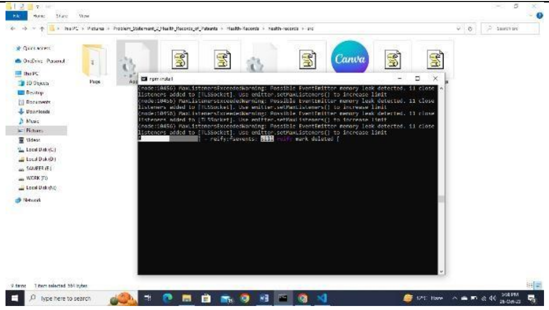
```
const crypto = require('crypto');
function encryptData(data, key) {
    const cipher = crypto.createCipher('aes-256-cbc', key);
    let encryptedData = cipher.update(data, 'utf-8', 'hex');
    encryptedData += cipher.final('hex');
    return encryptedData;
}
function decryptData(encryptedData, key) {
    const decipher =                    Decipher('aes-256-cbc', key);
    let decryptedDa                    update(encryptedData, 'hex',
'utf-8');
    decryptedData += decipher.final('utf-8');
    return decryptedData;
}
```

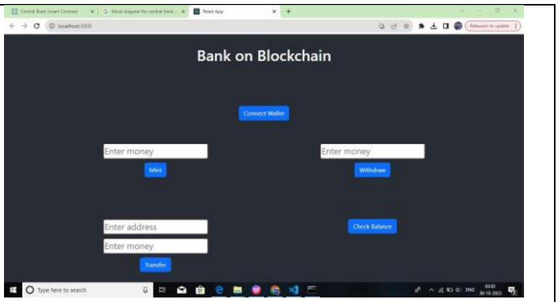
Explanation:

This code snippet demonstrates data encryption and decryption using the AES encryption algorithm. encryptData function takes data and a key, encrypts it, and returns the encrypted data in hexadecimal format. decryptData function reverses the process, decrypting the data using the same key.

8.PERFORMANCE TESTING

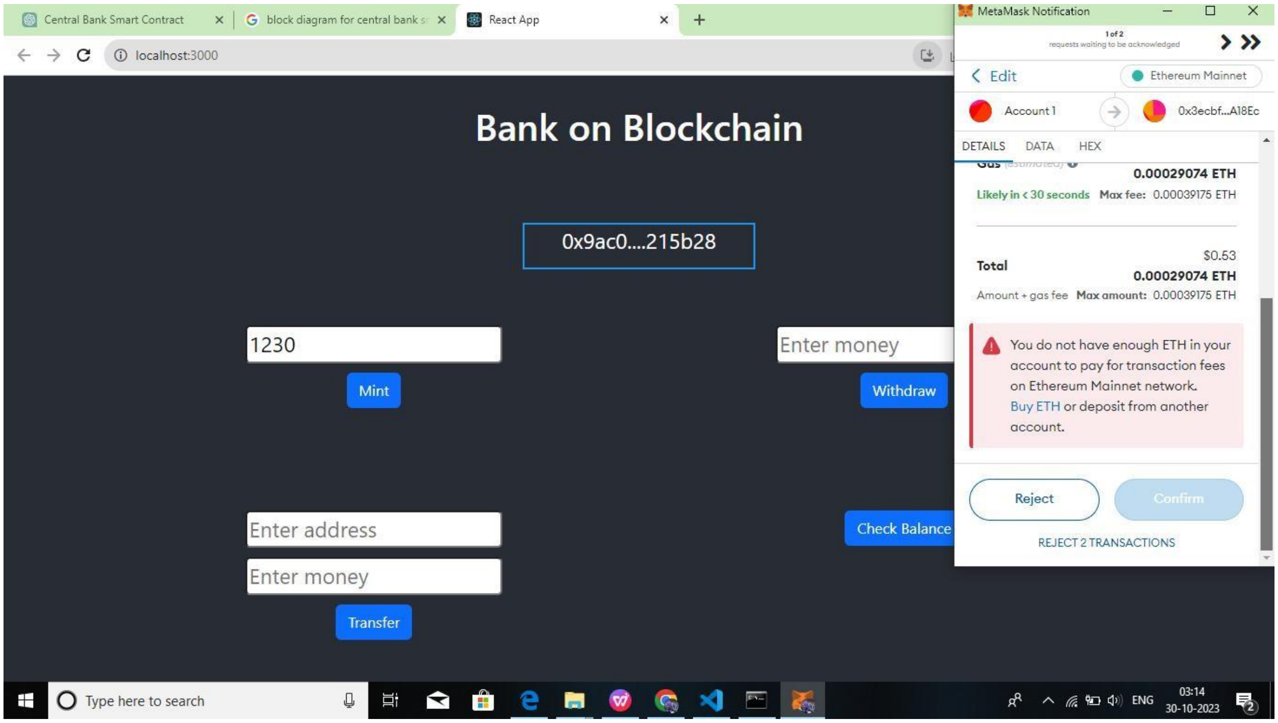
8.1 PERFORMANCE METRICS

S. No	Parameters	Values	Screenshots
1.	Information Gathering	Setup all the prerequisite	 s PC › Music › Health-Records node-v20.9.0.msi VSCodeUserSetup-x64-1.76.1.exe
2.	Extract the zip file	Open to vs code	
3.	Remix IDE exploring	Deploy the smart contract code Deploy and run the transaction. By selecting the environment - inject the MetaMask.	
4.	Open filemanager	Open the extracted file and click on the folder. Open src, and search for utiles. Open cmd enter commands 1. npm install 2. npm bootstrap 3. npm start	 Problem_Statement_2_Health_Records_of_Patients › Health-Records › health-records › src Canva npm install (node:10456) MaxListenersExceededWarning: Possible EventEmitter memory leak detected. 11 close listeners added to [TLSSocket]. Use emitter.setMaxListeners() to increase limit (node:10456) MaxListenersExceededWarning: Possible EventEmitter memory leak detected. 11 close listeners added to [TLSSocket]. Use emitter.setMaxListeners() to increase limit (node:10456) MaxListenersExceededWarning: Possible EventEmitter memory leak detected. 11 close listeners added to [TLSSocket]. Use emitter.setMaxListeners() to increase limit (node:10456) MaxListenersExceededWarning: Possible EventEmitter memory leak detected. 11 close listeners added to [TLSSocket]. Use emitter.setMaxListeners() to increase limit] - reify:fsevents: sill reify mark deleted [

5.	**LOCALHOST IP ADDRESS**	**Copy the address and open it to chrome so you can see the frontend of your project.**	 React App localhost:3000 Bank on Blockchain Connect Wallet Enter money Mint Enter money Withdraw Enter address Enter money Transfer Check Balance Type here to search

9. RESULTS

9.1 OUTPUT SCREENSHOTS



1230

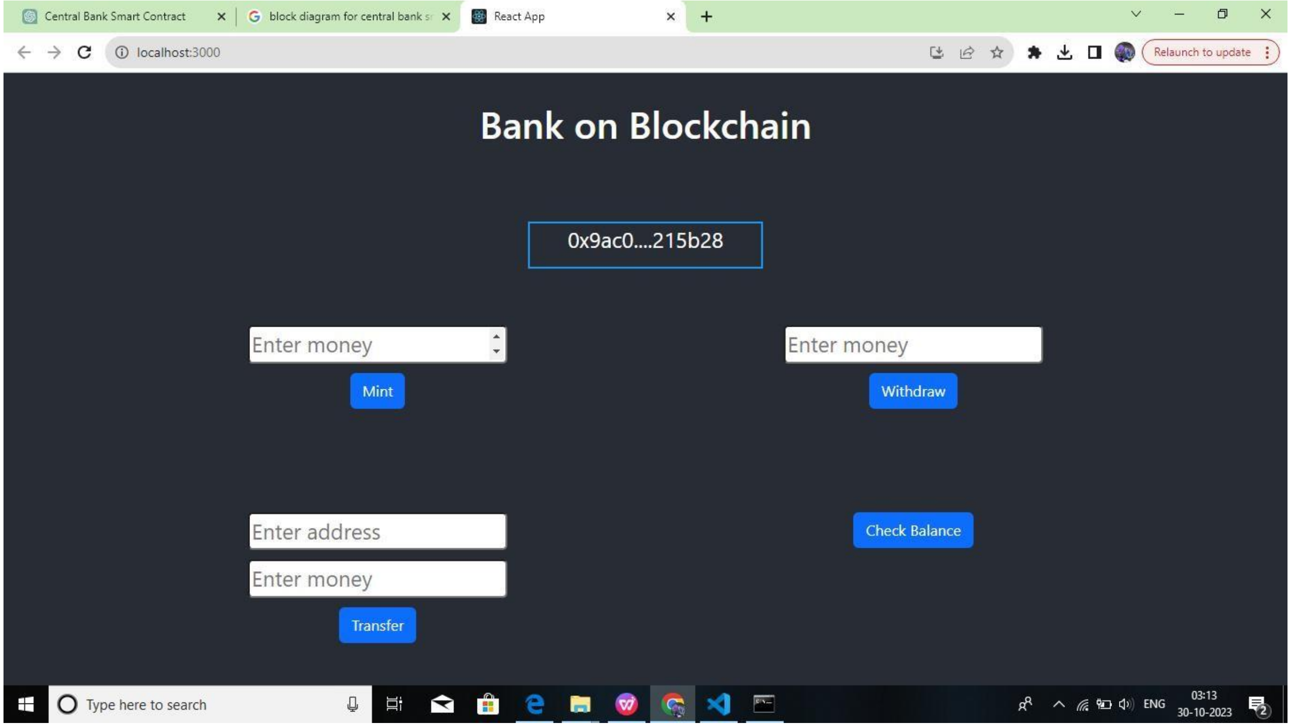
Gas (estimated) 0.00029074 ETH

Likely in < 30 seconds Max fee: 0.00039175 ETH

Total $0.53 0.00029074 ETH

Amount + gas fee Max amount: 0.00039175 ETH

You do not have enough ETH in your account to pay for transaction fees on Ethereum Mainnet network. Buy ETH or deposit from another account.



10.ADVANTAGES & DISADVANTAGES

Advantages :

Enhanced Data Security: Utilizing blockchain technology ensures enhanced security and immutability of central bank smart contract, protecting sensitive patient information from unauthorized access and tampering.

Efficient Data Management: Smart contracts automate data-sharing agreements, reducing administrative overhead and enabling more efficient and standardized data management processes.

Transparency and Trust: Blockchain's transparent nature builds trust among stakeholders. Every action within the system is recorded, enhancing accountability and transparency.

Innovation and Future-Readiness: Embracing cutting-edge technologies like blockchain and smart contracts positions the project as an innovative solution, ready to adapt to evolving healthcare needs and technological advancements.

Disadvantages of the Whole Project:

Complex Implementation: Implementing blockchain solutions can be complex and require specialized knowledge, potentially leading to development challenges and delays.

Integration Challenges: Integrating the blockchain-based system with existing healthcare infrastructure might pose challenges, especially if the legacy systems are outdated or incompatible.

Data Privacy Concerns: Although blockchain offers enhanced security, ensuring complete data privacy, especially in regions with stringent regulations like GDPR, requires meticulous design and compliance efforts.

Scalability: Blockchain networks, particularly public ones, might face scalability issues when dealing with a large volume of transactions. Ensuring the system can scale to meet increasing demands is crucial.

Regulatory Compliance: Adhering to data regulations, which vary across jurisdictions, poses a challenge. Ensuring the project complies with regional laws and regulations is critical for legal acceptance and trust.

11.CONCLUSION

In conclusion, the "Blockchain for Central Smart Bank Contract" project represents a significant step toward revolutionizing healthcare data management. By harnessing the power of blockchain technology, the project offers a secure, interoperable, and solution. The

advantages of enhanced data security, improved interoperability, empowerment, transparency, and innovation showcase the project's potential to transform healthcare systems.

However, it is crucial to acknowledge the challenges, including complexity in implementation, integration issues, data privacy concerns, and the need for regulatory compliance. Addressing these challenges through meticulous planning, continuous monitoring, and adaptation strategies is essential for the project's success.

12 .FUTURE SCOPE

The "Blockchain for Central Bank Smart Contract" project opens doors to several future opportunities and advancements in the healthcare industry. Here are some potential future scopes:

1. Blockchain-based Health Apps: Expand the project into mobile applications, allowing patients to securely access and manage on their smartphones. Integration with IoT devices could enable real-time

2. monitoring.

3. Research and Data Analysis: Utilize blockchain to create a secure, decentralized platform for medical research data. Researchers can access anonymized data securely, fostering collaborative research efforts and accelerating medical discoveries.

4. Supply Chain Management: Apply blockchain for tracking supplies throughout the supply chain.

5. Data Monetization and Incentives: Allow patients to share their health data securely with researchers, , or advertisers in exchange for tokens or incentives, promoting data-driven innovations.

6. IoT Security: Enhance the security of IoT devices in by integrating blockchain. Ensure the integrity of data collected by medical devices, preventing tampering and unauthorized access.

7. AI and Predictive Analysis: Combine blockchain with artificial intelligence for predictive analytics. Securely analyze large datasets to predict outbreaks, optimize resources, and improve care.

8. Compliance and Regulatory Solutions: Develop blockchain-based tools to providers adhere to complex regulatory requirements. Smart contracts can automate compliance checks, ensuring adherence to regional

13.APPENDIX

1. Technical Specifications:

Detailed technical specifications including programming languages, frameworks, and libraries used.Database schema diagrams.

Blockchain integration details, such as the chosen blockchain platform (Ethereum, Hyperledger, etc.) and smart contract code snippets.

2. User Guides:

Comprehensive user guides for providers, , and administrators, explaining system functionalities, authentication processes, and data management procedures. Metamask setup instructions for users unfamiliar with blockchain interactions.

3. Code Samples:

Code snippets demonstrating key features like user authentication, data encryption, smart contract interactions, and API integrations. Examples of error handling and edge cases in the codebase.

4. Data Security Measures:

Detailed information about encryption algorithms used for data security. Explanation of access control mechanisms implemented in smart contracts. Protocols and policies ensuring data confidentiality and integrity during transmission and storage.

5. Performance Testing Reports:

Performance testing methodologies, including tools used and test scenarios.Performance test results, including response times, throughput, and system scalability under various loads.

6. Regulatory Compliance Documentation:

Documentation showcasing how the project complies with healthcare data regulations (HIPAA, GDPR, etc.). Details about user consent management and data deletion policies.

7. User Feedback and Improvement Reports:

Summaries of user feedback collected during usability testing. Reports on system improvements and enhancements made based on user suggestions.

8. Future Enhancements:

Detailed plans for future enhancements, including new features, integrations, and technology upgrades. Roadmap outlining the project's evolution over the next few years.

9. References:

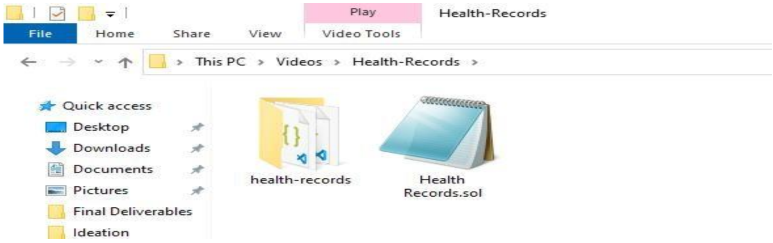
Citations and references for research papers, articles, and resources used during the project development. Links to relevant documentation, libraries, and frameworks.

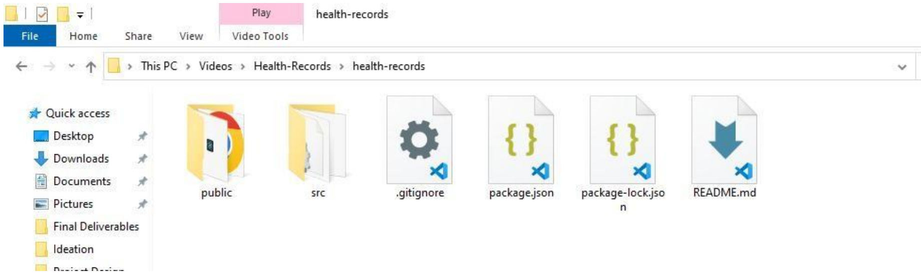
10. Glossary:

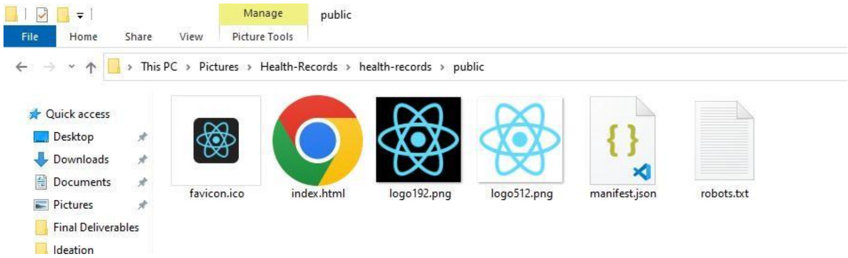
Definitions and explanations of technical terms, acronyms, and industry-specific jargon used throughout the project documentation.

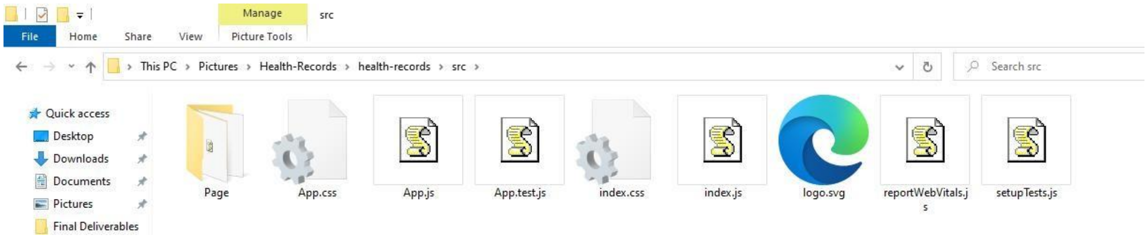
SOURCE CODE

Folder Structure :









INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />

    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder
      during the build. Only files inside the `public` folder can
      be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico",
      "%PUBLIC_URL%/favicon.ico" will work correctly
      both with client-side routing and a non-root public
      URL. Learn how to configure a non-root public URL by
      running `npm run build`.
    -->
    <title>React App</title>
</head>
<body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.


      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.


      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
</body>
</html>
```

MANIFEST.JSON

```
{
  "short_name": "React App",

  "name": "Create React App Sample",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64 32x32 24x24 16x16",
      "type": "image/x-icon"
    },
    {
      "src": "logo192.png",
      "type": "image/png",
      "sizes": "192x192"
    },
    {
      "src": "logo512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ],
  "start_url": ".", "display":
  "standalone",
  "theme_color":
  "#000000",
  "background_color": "#ffffff"
}
```

CONNECTOR.JS

```
const { ethers } = require("ethers"); const

abi =[


    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "internalType": "uint256",
                "name": "recordId",
                "type": "uint256"
            },
            {
                "indexed": true,
                "internalType": "address",
                "name": "patientAddress",
                "type": "address"
            }
        ],
        "name":
        "RecordCreated",
        "type": "event"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "internalType": "uint256",
                "name": "recordId",
                "type": "uint256"
            },
            {
                "indexed": true,
                "internalType": "address",
                "name": "from",
                "type": "address"
            },
            {
                "indexed": true,
                "internalType": "address",
                "name": "to",
                "type": "address"
            }
        ],
        "name": "RecordTransferred",
        "type": "event"
    },
    {

        "inputs": [
            {
                "internalType": "uint256",
                "name": "recordId",
                "type": "uint256"
            },
            {
                "internalType": "string",
                "name": "name",
                "type": "string"
            },
            {
                "internalType":  "address",
                "name":
                "_patientAddress", "type":
                "address"
            },
            {
                "internalType": "string",
                "name": "_diseases",
                "type": "string"
            },
        {
            "internalType": "string",
            "name": "_contactInfo",
            "type": "string"
        }
    ],
    "name": "createRecord",
    "outputs": [], "stateMutability":
    "nonpayable", "type": "function"
},
{
    "inputs": [
        {
            "internalType": "uint256",
            "name": "recordId",
            "type": "uint256"
        }
    ],
    "name":
    "getRecordData",
    "outputs": [
        {
            "internalType": "string",
            "name": "",

            "type": "string"
        },
        {
            "internalType": "address",
            "name": "",
            "type": "address"
        },
        {
            "internalType": "string",
            "name": "",
            "type": "string"
        },
        {
```

```
                            "internalType": "string",
                            "name": "",
                            "type": "string"
                        }
                    ],
                    "stateMutability": "view",
                    "type": "function"
                },
                {
                    "inputs": [
                        {
                            "internalType": "uint256",
                            "name": "recordId",
                            "type": "uint256"
                        }
                    ],
                    "name": "getRecordOwner",
                    "outputs": [
                        {
                            "internalType": "address",
                            "name": "",
                            "type": "address"
                        }
                    ],
"stateMutability": "view",

                    "type": "function"
                },
                {
                    "inputs": [
                        {
                            "internalType": "uint256",
                            "name": "",
                "type": "uint256"
            }
        ],
        "name": "records",
        "outputs": [
            {
                "internalType": "string",
                "name": "Name",
                "type": "string"
            },
            {
                "internalType": "address",
                "name": "patientAddress",
                "type": "address"
            },
            {
                "internalType": "string",
                "name": "dieses",
                "type": "string"
            },
            {
                "internalType": "string",
                "name": "contactInfo",
                "type": "string"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [
            {
                "internalType": "uint256",
                "name": "recordId",
                "type": "uint256"
            },
            {
                "internalType": "address",
                "name": "newOwner",
                "type": "address"
            }
        ],
        "name": "transferRecord",
        "outputs": [], "stateMutability":
        "nonpayable",

        "type": "function"
    }
]

if (!window.ethereum) {
  alert('Meta Mask Not Found')
  window.open("https://metamask.io/download/")
}

export const provider = new
ethers.providers.Web3Provider(window.ethereum); export const signer =
provider.getSigner();
export const address =

"0xbCE87F01326965253e338bD5738587C02B481017" export const

contract = new ethers.Contract(address, abi, signer)
```

HOME.JS

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';

import '../../node_modules/bootstrap/dist/css/bootstrap.min.css';
import { contract } from "./connector";

function Home() {
 const [Id, setId] = useState(""); const
 [name, setName] = useState("");
 const [pAddr, setpAddr] = useState("");
 const [disease, setdisease] = useState("");
 const [contact, setContact] = useState("");
 const [recordId, setrecordId] = useState(""); const
 [newOwner, setNewOwner] = useState("");
 const [recordIdData, setrecordIdData] = useState("");
 const [Data, setData] = useState("");
  const [Wallet, setWallet] = useState("");


 const handleId = (e) => {
  setId(e.target.value)
 }

 const handleName = (e) => {
  setName(e.target.value)
 }

 const handlePatientAddress = (e) => {
  setpAddr(e.target.value)
 }

 const handleDisease = (e) => {
  setdisease(e.target.value)
 }
```

```
const handleContact = (e) => {
  setContact(e.target.value)
}

const handleCreateRecord = async() => {
  try {
    let tx = await contract.createRecord(Id, name, pAddr, disease, contact) let
    wait = await tx.wait()
    alert(wait)
    console.log(wait.transactionHash);
  } catch (error) {
    alert(error)
  }

}

const handleRecordId = (e) => {
  setrecordId(e.target.value)
}

const handleNewOwner = (e)
  => {
  setNewOwner(e.target.value)
}

const handleTransferRecord = async () => {
  try {
    let tx = await contract.transferRecord(recordId.toString(),newOwner)
    let wait = await tx.wait()
    alert(wait.transactionHash)
    console.log(wait);
  } catch (error) {
    alert(error)
  }
}

const handleRecordDataId = (e) => {
 setrecordIdData(e.target.value)
 }

 const handleRecordData =async () =>
  { try {
  let tx = await contract.getRecordData(recordIdData) let
  arr = []
  tx.map(e => arr.push(e))
    setData(arr)
  // alert(tx) console.log(tx);

  } catch (error) {
   alert(error)
  }
 }

  const handleWallet = async () => {
    if (!window.ethereum) {
      return alert('please install metamask');
    }

    const addr = await window.ethereum.request({

      method: 'eth_requestAccounts',
    });

    setWallet(addr[0])

  }

 return (
  <div>
   <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Health Records
Using Blockchain</h1>
     {!Wallet ?

       <Button onClick={handleWallet} style={{ marginTop: "30px",
marginBottom: "50px" }}>Connect Wallet </Button>
```

```
              :
              <p style={{ width: "250px", height: "50px", margin: "auto",
marginBottom: "50px", border: '2px solid #2096f3' }}>{Wallet.slice(0, 6)}
 ............................................................................................................. {Wallet.slice(-
6)}</p>
          }
          <Container style={{ margin:"Auto" }}>
              <Row >
          <Col>
            <div>



              <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleId} type="number" placeholder="Enter Record Id" value={Id} />
<br
/>

              <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleName} type="string" placeholder="Enter name" value={name}
/> <br
/>

              <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handlePatientAddress} type="string" placeholder="Enter patient
Address" value={pAddr} /><br />

              <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleDisease} type="string" placeholder="Enter disease"
value={disease}
/><br />
              <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleContact} type="string" placeholder="Enter contact Info"
value={contact} /><br />
```

```
          <Button onClick={handleCreateRecord} style={{ marginTop: "10px" }}
variant="primary">Create Record</Button>
         </div>
        </Col>
        <Col>
          <div>
          <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleRecordId} type="number" placeholder="Enter new record Id"
value={recordId} /><br />
          <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleNewOwner} type="string" placeholder="Enter new owner
metamask address" value={newOwner} /><br />
          <Button onClick={handleTransferRecord} style={{ marginTop: "10px" }}
variant="primary">Transfer Record</Button>

          </div>
        </Col>
   </Row>
      <Col>
         <Row style={{marginTop:"100px"}}>
         <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleRecordDataId} type="string" placeholder="Enter Id"
value={recordIdData} /><br />
         <Button onClick={handleRecordData} style={{ marginTop: "10px" }}
variant="primary">Get Record Data</Button>
            {Data? Data?.map(e =>
            { return <p>
               {e.toString()}
            </p>
            }
            ) : <p></p>}
         </Row>
        </Col>
    </Container>
   </div>
 )
} export default Home;
```

APP.CSS

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color:
  #282c34; min-height:
  100vh; display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin
  { from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

APP.JS

```
import './App.css';
import Home from './Page/Home'

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <Home />
      </header>
    </div>
  );
}

export default App;
```

APP.TEST.JS

```
import { render, screen } from '@testing-library/react'; import
App from './App';

test('renders learn react link', () => { render(<App
  />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
```

INDEX.CSS

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

INDEX.JS

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';

import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

REPORTWEBVITALS.JS

```
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) =>
      { getCLS(onPerfEntry);
```

```
        getFID(onPerfEntry);
        getFCP(onPerfEntry)
        ;
        getLCP(onPerfEntry)
        ;
        getTTFB(onPerfEntr
        y);}}
export default reportWebVitals;
```

SETUPTESTS.JS

```
// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)

// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';
```

PACKAGE.JSON

```
{
  "name": "health-records",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "bootstrap": "^5.3.1",
    "ethers": "^5.6.6",
    "react": "^18.2.0",
    "react-bootstrap": "^2.8.0",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start", "build":
    "react-scripts build", "test": "react-
    scripts test", "eject": "react-scripts
    eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",

      "last 1 safari version"
    ]
  }
}
```

GITHUB & DEMO VIDEO

GitHub Link :https://github.com/Anandu01/NM2023TMID06094

Demo Video : https://drive.google.com/file/d/137T8NwuTBHBl7G0soJJCPN-M6JbzRQMN/view?usp=drivesdk