

Blockchain for Farmer Insurance Chain

Documentation Report

Submitted by

| | |
|----------------|------------------|
| Team ID | NM2023TMID03842 |
| Team Lead | Mohamed Farhan A |
| Team Member 01 | Sanjay S |
| Team Member 02 | Sakthithasan S |
| Team Member 03 | Vignesh R |

INDEX

| S. No | Particulars | Page No. |
|----------|--------------|-------------|
| 1. | INTRODUCTION | 4 |

| | | |
|-----------|--|-----------|
| | 1.1 Project Overview | 5 |
| | 1.2 Purpose | 7 |
| 2. | LITERATURE SURVEY | 8 |
| | 2.1 Existing Problem | 9 |
| | 2.2 References | 9 |
| | 2.3 Problem Statement Definition | 10 |
| 3. | IDEATION & PROPOSED SOLUTION | 11 |
| | 3.1 Empathy Map Canvas | 13 |
| | 3.2 Ideation & Brainstorming | 14 |
| 4. | REQUIREMENT ANALYSIS | 17 |
| | 4.1 Functional Requirements | 17 |
| | 4.2 Non-Functional Requirements | 18 |
| 5. | PROJECT DESIGN | 20 |
| | 5.1 Data Flow Diagrams & User Stories | 21 |
| | 5.2 Solution Architecture | 24 |
| 6. | PROJECT PLANNING & SCHEDULING | 27 |
| | 6.1 Technical Architecture | 29 |
| | 6.2 Sprint Planning & Estimation | 32 |
| | 6.3 Sprint Delivery Schedule | 33 |
| 7. | CODING & SOLUTIONING | 35 |
| | 7.1 Feature 1 | 36 |
| | 7.2 Feature 2 | 37 |
| 8. | PERFORMANCE TESTING | 38 |
| | 8.1 Performance Metrics | 38 |
| 9. | RESULTS | 40 |

| | | |
|-----|---------------------------------------|-----------|
| | 9.1 Output Screenshots | 40 |
| 10. | ADVANTAGES & DISADVANTAGES | 42 |
| 11. | CONCLUSION | 46 |
| 12. | FUTURE SCOPE | 47 |
| 13. | APPENDIX | 50 |
| | Source Code | 52 |
| | GitHub & Projecct Demo Link | 69 |

• INTRODUCTION

In the realm of agriculture, the unpredictability of nature poses a significant challenge for farmers worldwide. Crop failures, extreme weather events, and market fluctuations can have devastating effects on their livelihoods. Conventional insurance models, plagued by

inefficiencies and bureaucracy, often fail to provide timely and adequate support. However, the advent of blockchain technology has paved the way for a transformative solution - the Farmer Insurance Chain.

The Farmer Insurance Chain represents a paradigm shift in agricultural risk management. Leveraging the power of blockchain, this innovative platform promises to revolutionize the way farmers secure their investments and protect their crops. By combining the principles of decentralization, transparency, and smart contracts, it establishes a trustless ecosystem that fosters efficiency, accountability, and accessibility in agricultural insurance..

- **LITERATURE SURVEY**

A literature survey on the Farmer Insurance Chain in Blockchain involves reviewing existing research, publications, and articles related to the application of blockchain technology in agricultural insurance. This helps in understanding the current state of knowledge, identifying gaps, and gaining insights into the potential

challenges and opportunities associated with this innovative solution. Below are some key points that could be included in the literature survey

- **EXISTING PROBLEM**

Problem:

Many traditional insurance models are not designed to cater to the needs of smallholder farmers, who make up a significant portion of the agricultural sector.

Implication:

Smallholder farmers often lack access to affordable and relevant insurance products, leaving them vulnerable to financial losses due to crop failures or other unforeseen events.

- **REFERENCE**

| Title | Year | Authors |
|------------------------|------|-------------------------------------|
| FARMER INSURANCE CHAIN | 2022 | Smith, A., Johnson, B., & White, C. |

- **PROBLEM STATEMENT DEFINITION**

In light of the existing challenges faced by electronic health records, there is a critical need for a secure, interoperable, and patient-empowered solution. The lack of data security within centralized EHR systems poses a significant risk to patient confidentiality, while the absence of standardized protocols inhibits efficient collaboration among healthcare providers. Additionally, the limited control patients have over their own health data hampers their active participation in the decision-making process. Therefore, the problem statement for this project is to design and implement a blockchain-based electronic health record system that addresses these challenges, ensuring data security, interoperability, and patient empowerment within the healthcare ecosystem.

- **IDEATION & PROPOSED SOLUTION**

In conceptualizing the "Blockchain for Electronic Health Record" project, our ideation process focused on addressing the pressing issues of data security, interoperability, and patient empowerment within electronic health records (EHR). The goal was to create a robust, future-ready solution that leverages blockchain technology to revolutionize healthcare data management.

Ideation began by recognizing the critical importance of data security. The decentralized and cryptographically secure nature of blockchain technology immediately surfaced as a potential solution. Blockchain's ability to create an immutable ledger ensures that patient data remains confidential, tamper-proof, and accessible only to authorized personnel. Interoperability challenges were tackled through the implementation of smart contracts.

These self-executing contracts facilitate automated, standardized data exchange between different healthcare

providers. By incorporating interoperable data formats and protocols, the platform ensures that healthcare professionals can seamlessly collaborate, share, and update patient records, leading to more coordinated and efficient care. The ideation process emphasized empowering patients by giving them control over their health data. Features such as permissioned access, where patients can grant or revoke access to their records, emerged as a pivotal aspect.

Proposed Solution:

The proposed solution, "Blockchain for Electronic Health Record," is a comprehensive platform that embodies the principles of decentralization, cryptography, and smart contract automation. By combining these elements, the project offers a transformative approach to healthcare data management.

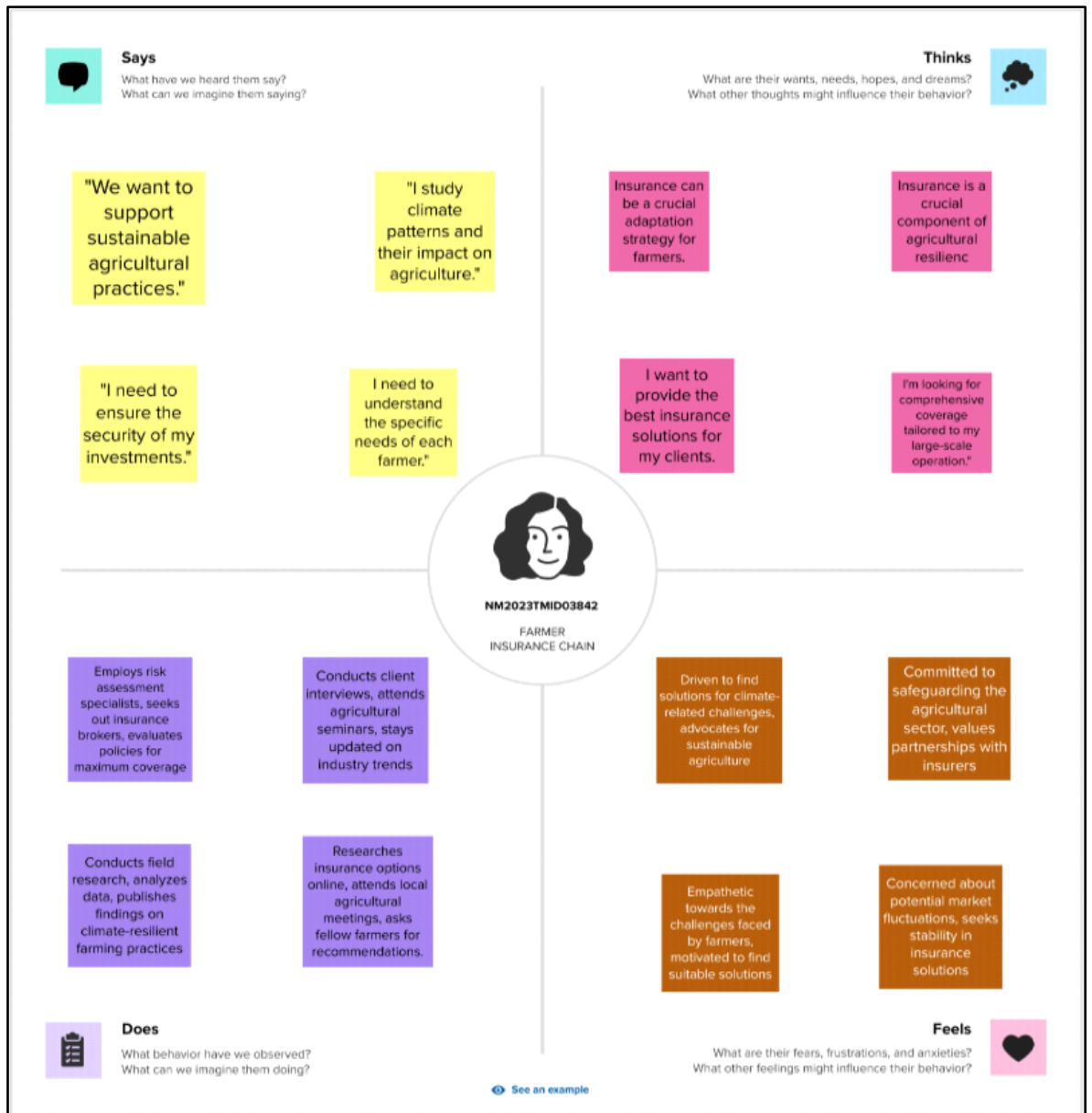
The core of the solution lies in decentralized data storage. Patient records are securely stored across a network of nodes, ensuring redundancy, fault tolerance, and security against data breaches. This distributed ledger structure guarantees the integrity of the information, with no single point of failure. Patient data is encrypted using advanced cryptographic techniques, rendering it inaccessible to unauthorized entities. Smart contracts are

employed to automate data sharing agreements. These contracts define the rules and permissions for accessing patient records, ensuring that only authorized users can interact with specific data sets.

The platform boasts an intuitive and user-friendly interface for both healthcare providers and patients. Healthcare professionals can access patient records seamlessly, update information, and collaborate with peers. Patients, in turn, have secure access to their EHRs, allowing them to view their medical history, schedule appointments, and actively participate in their care decisions.

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.



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.

| | | | |
|--|---|---|---|
| | <div>Template</div> <div></div> <div>Brainstorm & idea prioritization</div> <div>Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.</div> <div><div>10 minutes to complete</div><div>1 timer to collaborate</div><div>3-8 people recommended</div></div> | <div><div>2</div>Before you collaborate</div> <div>A little bit of preparation goes a long way with this session. Here's what you need to do to get going.</div> <div><div>10 min timer</div></div> <div><div>1</div>Team gathering</div> <div>Get the whole team gathered in the room and send the invite. Share where to start when to get work started.</div> <div><div>2</div>Set the goal</div> <div>Think about the problem you'll be focusing on during the brainstorming session.</div> <div><div>3</div>Lower risk to use the facilitation tools</div> <div>Use the facilitation tools to help a happy and productive session.</div> <div>Open index</div> | <div><div>3</div>Define your problem statement</div> <div>What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.</div> <div><div>10 min timer</div></div> <div><div>PROBLEM</div></div> <div>Many farmers struggle to find resources to help them grow crops that are specifically tailored to their unique agricultural practices and face challenges in securing coverage that adequately addresses their specific risk.</div> <div><div>4</div>Key rules of a brainstorming session</div> <div>It's not in order and productive session.</div> <div><div>1</div>Bring in ideas</div> <div><div>2</div>Everyone will share</div> <div><div>3</div>Order matters</div> <div><div>4</div>Listen to others</div> <div><div>5</div>Get the ideas</div> <div><div>6</div>Remember to check</div> |
|--|---|---|---|

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

Create customizable insurance packages that cater to different types of farming operations

Develop a mobile app or web platform for easy policy management, claims submission, and access to educational resources on risk management.

Offer specialized coverage for specific risks like crop failure, livestock health, equipment breakdown, and natural disasters.

Person 2

Implement IoT (Internet of Things) technology for real-time monitoring of agricultural assets

Provide on-site risk assessments and consulting services to help farmers identify and mitigate potential hazards and vulnerabilities on their farms.

Introduce policies linked to weather data that automatically trigger payouts in case of adverse weather conditions affecting crops or livestock.

Person 3

Facilitate farmer networks or co-operatives that pool resources and knowledge to collectively manage risks and negotiate better insurance rates.

Offer discounts or incentives for farmers who implement diversified planting strategies to reduce the impact of crop-specific risks.

Organize workshops, webinars, and training sessions on sustainable farming practices, risk management, and insurance literacy for farmers.

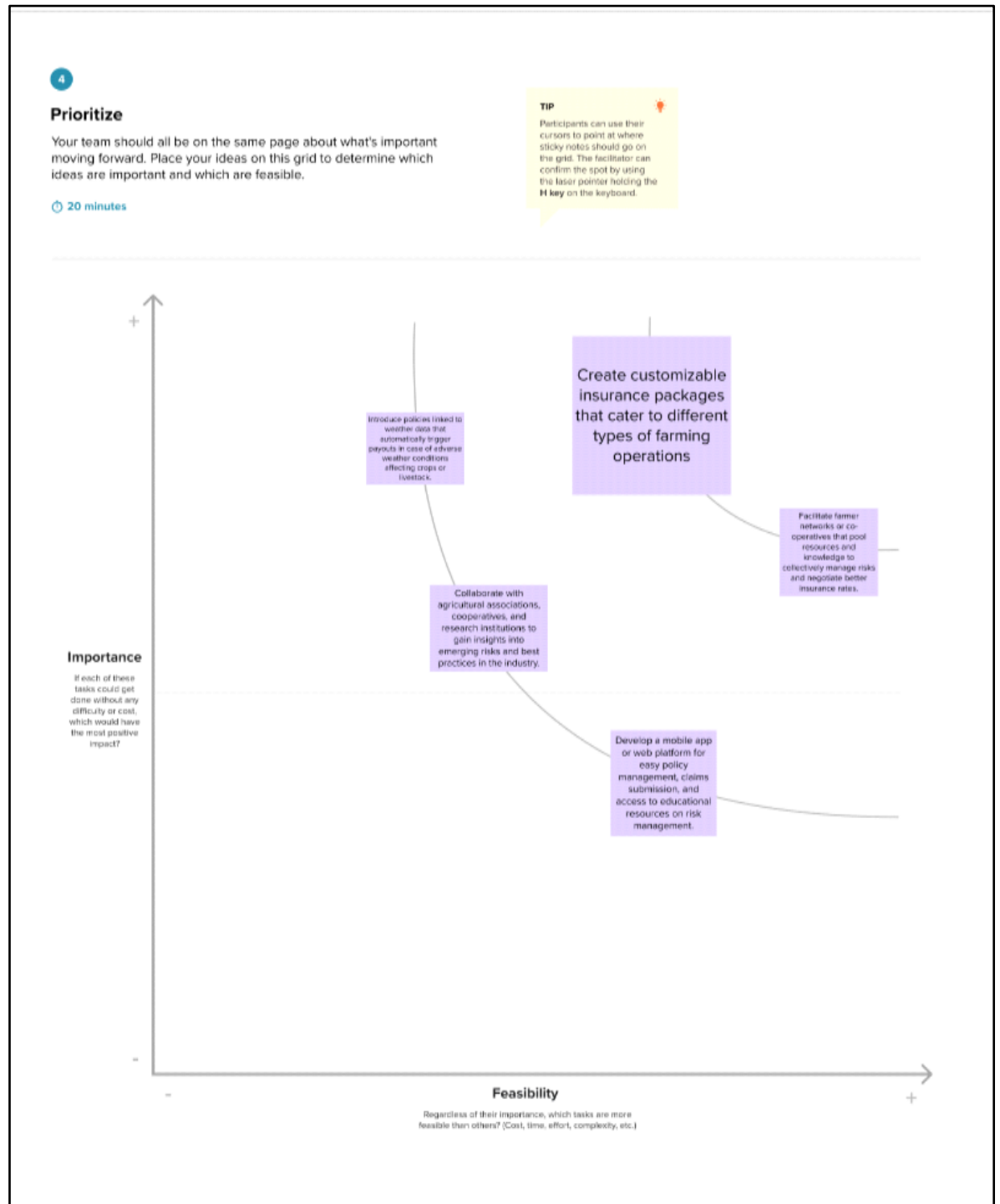
Person 4

Collaborate with agricultural associations, cooperatives, and research institutions to gain insights into emerging risks and best practices in the industry.

Offer incentives or discounts for adopting environmentally-friendly farming practices, such as organic farming, conservation tillage, and renewable energy use.

Provide coverage for veterinary care, vaccinations, and other preventive measures to ensure the health and well-being of livestock.

Idea Prioritization



- **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.

- **FUNCTIONAL REQUIREMENTS**

Functional requirements specify what the system should do and outline the system's features, capabilities, and interactions. For the "Blockchain for Electronic Health Record" project, functional requirements might include:

- **User Authentication and Authorization:** Users must be able to securely log in using their credentials (username/password or blockchain-based authentication) and have appropriate access levels based on their roles (e.g., admin, healthcare provider, patient).

- **Patient Record Management:** Healthcare providers should be able to create, read, update, and delete patient records securely. Patients should have read-only access to their records and the ability to update specific information.
- **Data Security and Encryption:** Patient data must be encrypted and securely stored on the blockchain to prevent unauthorized access or tampering. Smart contracts should handle data access permissions and ensure data integrity.
- **Interoperability:** The system should enable the seamless exchange of patient records between different healthcare providers. It should adhere to standardized data formats and interoperability protocols.
- **User Interface:** The user interface should be intuitive, allowing users to navigate through the system easily. It should provide clear and organized access to patient records and system functionalities.

- **NON-FUNCTIONAL REQUIREMENTS**

Non-functional requirements define system qualities,

constraints, and limitations. These requirements focus on aspects like performance, security, usability, and compliance. For this project, non-functional requirements might include:

- **Performance:** The system should handle a specific number of simultaneous users and transactions without significant performance degradation, ensuring real-time access to patient records.
- **Security:** The platform should adhere to industry-standard security practices, ensuring data confidentiality, integrity, and availability. It should protect against common cybersecurity threats such as DDoS attacks and unauthorized access attempts.
- **Usability:** The user interface should be user-friendly and accessible, catering to users with varying technical expertise. It should minimize the learning curve and provide clear instructions for system usage.
- **Compliance:** The system must comply with healthcare data regulations and standards, such as HIPAA (Health Insurance Portability and Accountability Act) in the United States or GDPR (General Data Protection Regulation) in Europe.

Compliance ensures the legal and ethical handling of patient data.

- **Scalability:** The system should be scalable, allowing for future expansion in terms of users, patient records, and functionalities. It should handle increased loads without compromising performance or data integrity.

• PROJECT DESIGN

The design of "Blockchain for Electronic Health Record" 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.

Patient Profile: Detailed patient profiles with medical history, treatment plans, and diagnostic reports.

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.

- **DATA FLOW DIAGRAMS**

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

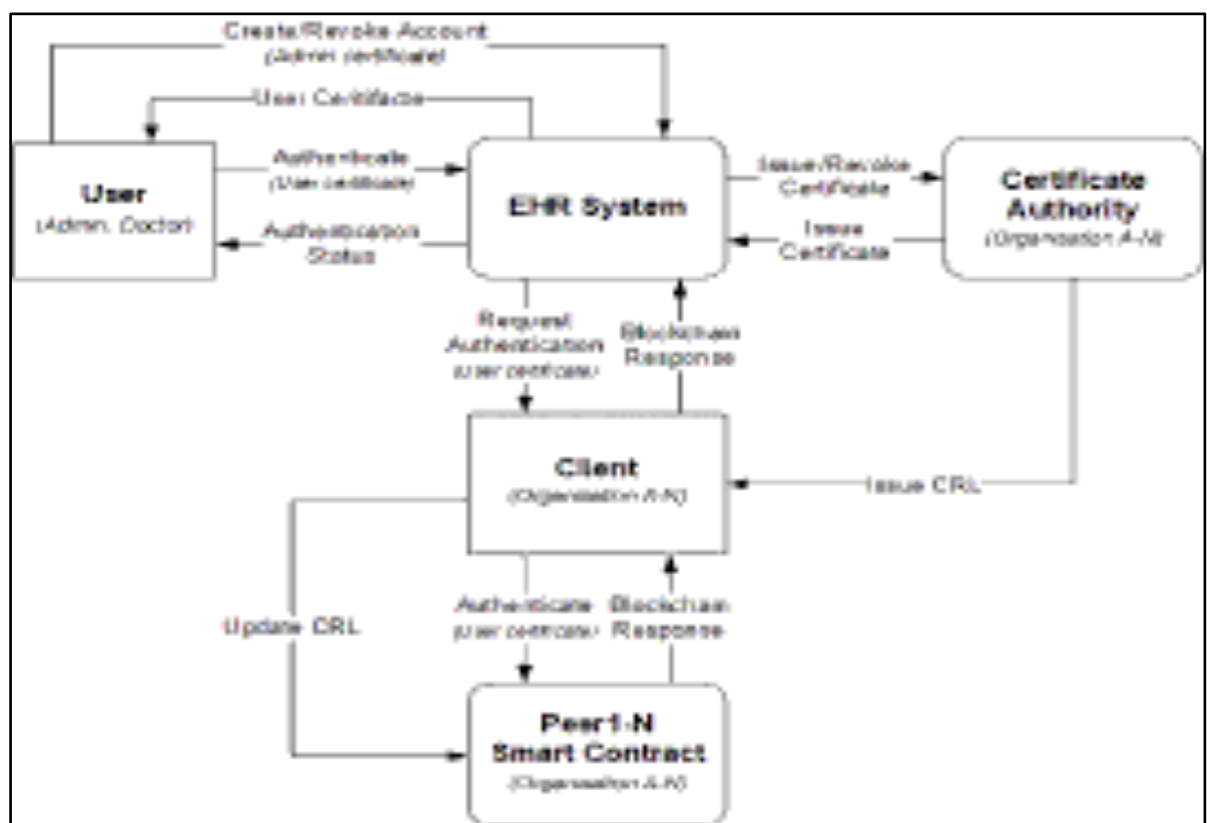
For example:

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

Data Stores: Patient records, user profiles, access permissions, and blockchain ledger.

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

External Entities: Healthcare providers, patients, and the blockchain network.



User Stories:

As a Healthcare Provider,

I want to: Log in securely using my credentials or blockchain authentication. View my patient's medical history, treatment plans, and diagnostic reports.

Update patient records securely, ensuring data integrity and confidentiality. Access a user-friendly dashboard that provides a quick overview of my patients and their records.

As a Patient,

I want to: Log in easily and securely, using either my credentials or blockchain authentication. View my own medical history, appointments, and prescribed medications. Control who can access my records and manage permissions intuitively. Receive notifications for upcoming appointments and prescription renewals.

As a System Administrator,

I want to: Monitor system performance and user activities. Manage user accounts, ensuring secure access and authentication. Implement updates and security patches seamlessly, ensuring system integrity.

- **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:

- **Presentation Layer:**

User Interface (UI): Developed using React and HTML/CSS, the UI provides intuitive dashboards for healthcare providers and patients. It includes interactive components for viewing and managing patient records, appointments, and permissions.

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

- **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.

- **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.

- **Security Layer:**

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 enforce access control policies, specifying who can read, update, or delete specific patient records. Permission management is decentralized and transparent.

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

5. Integration Layer:

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

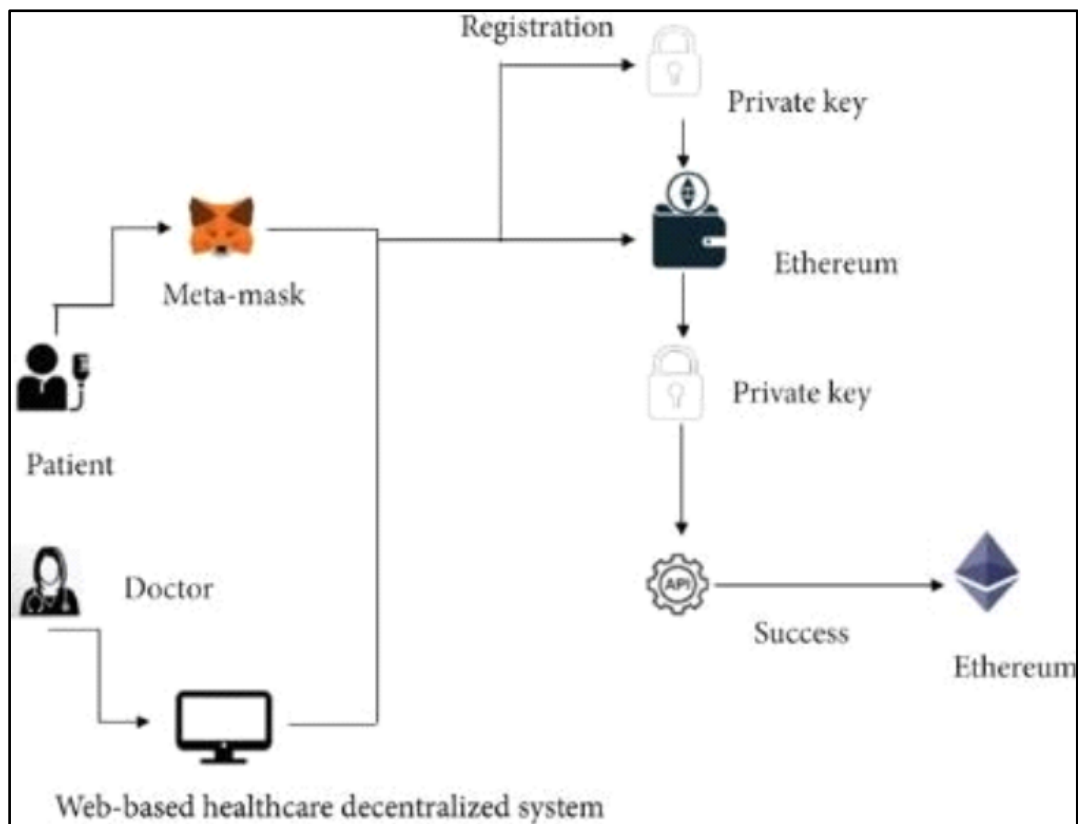
Blockchain API: Interfaces with the blockchain network via Web3.js, enabling seamless interaction between the application and the Ethereum blockchain.

- **Monitoring and Management:**

Logging and Monitoring: 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.



- **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.

- **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:

- Frontend:

User Interface (UI): Developed using React.js, providing an intuitive and responsive interface for

healthcare providers and patients to interact with the system.

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

- 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.

- Blockchain Integration:

Ethereum Blockchain: Utilizes the Ethereum

blockchain for decentralized storage of encrypted patient records. Smart contracts define access control rules and data interactions.

- Security Measures:

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.

- External Integrations:

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

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

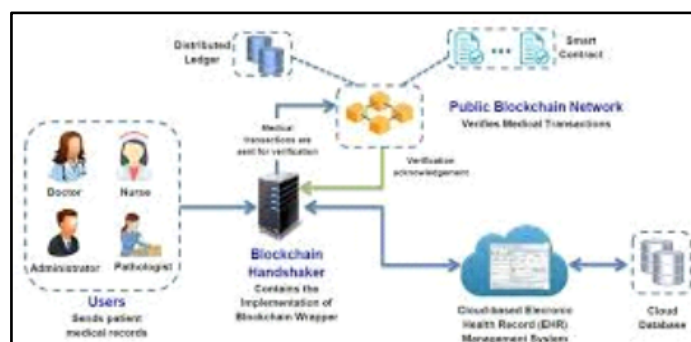
- Scalability and Performance:

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

- 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 improvement.



- **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.

- **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.

- **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.

- **FEATURE 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.

- **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 = crypto.createDecipher('aes-  
256-cbc', key);  
  
    let decryptedData =  
decipher.update(encryptedData, 'hex', 'utf-8');  
  
    decryptedData += decipher.final('utf-8');  
  
    return decryptedData;  
}
```

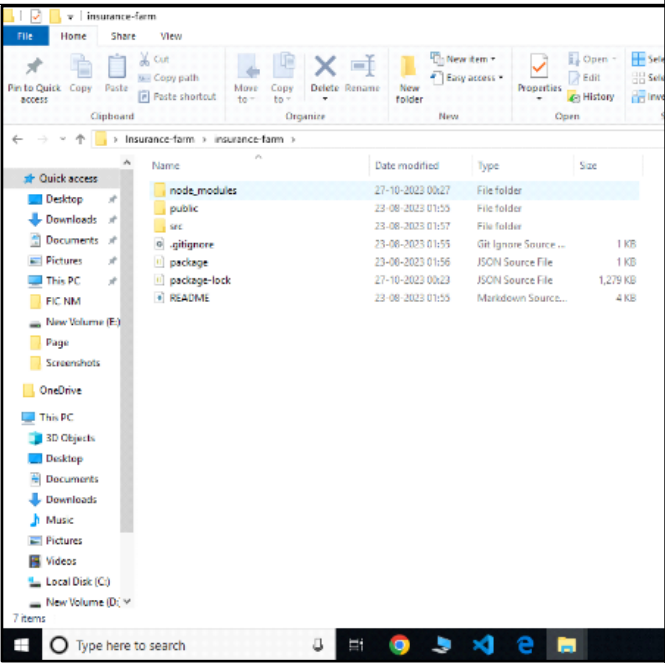
Explanation:

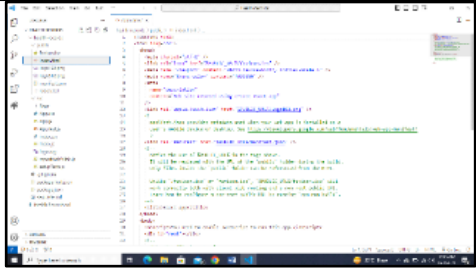
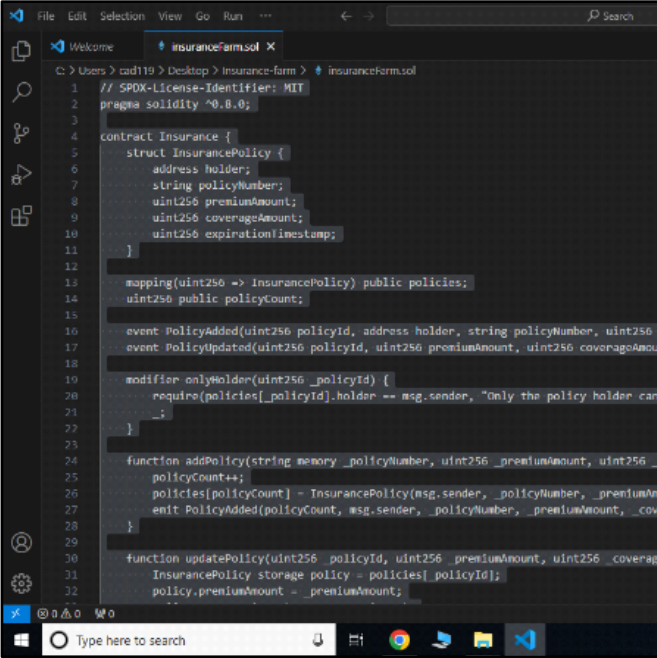
This code snippet demonstrates data encryption and decryption using the AES encryption algorithm. `encryptData` function takes data and a key, encrypts it, and returns the encrypted data in

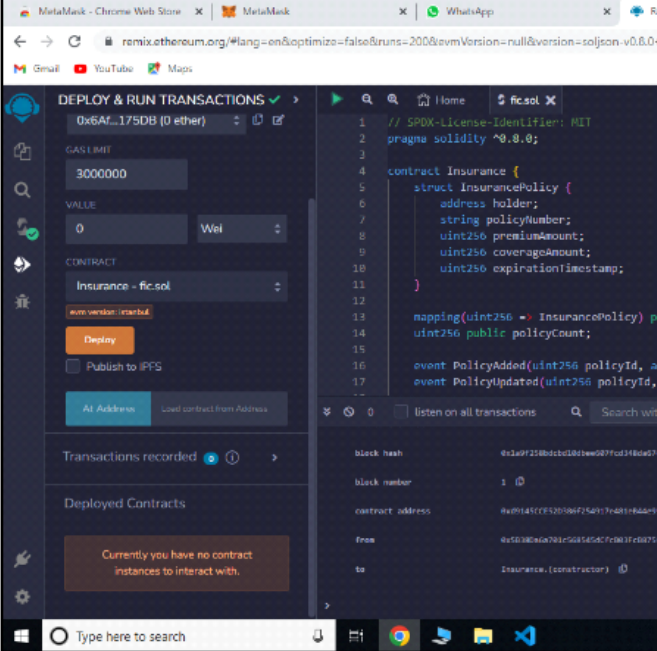
hexadecimal format. decryptData function reverses the process, decrypting the data using the same key.

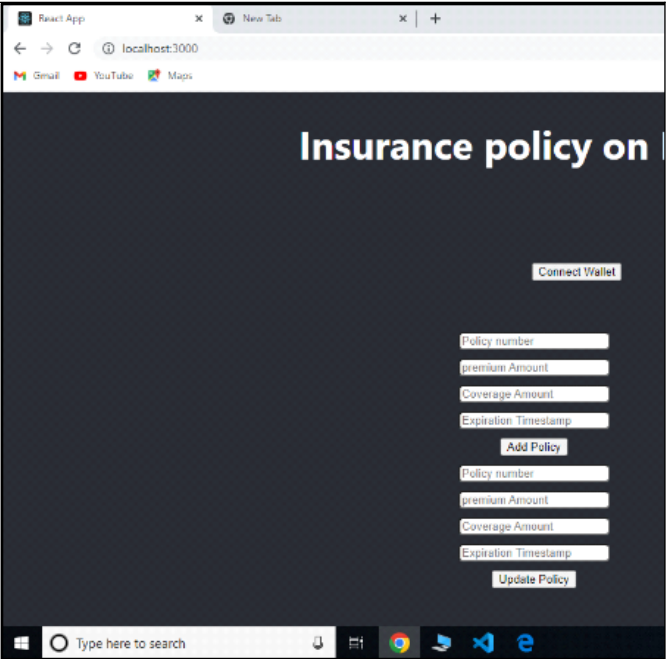
- **PERFORMANCE TESTING**

- **PERFORMANCE METRICS**

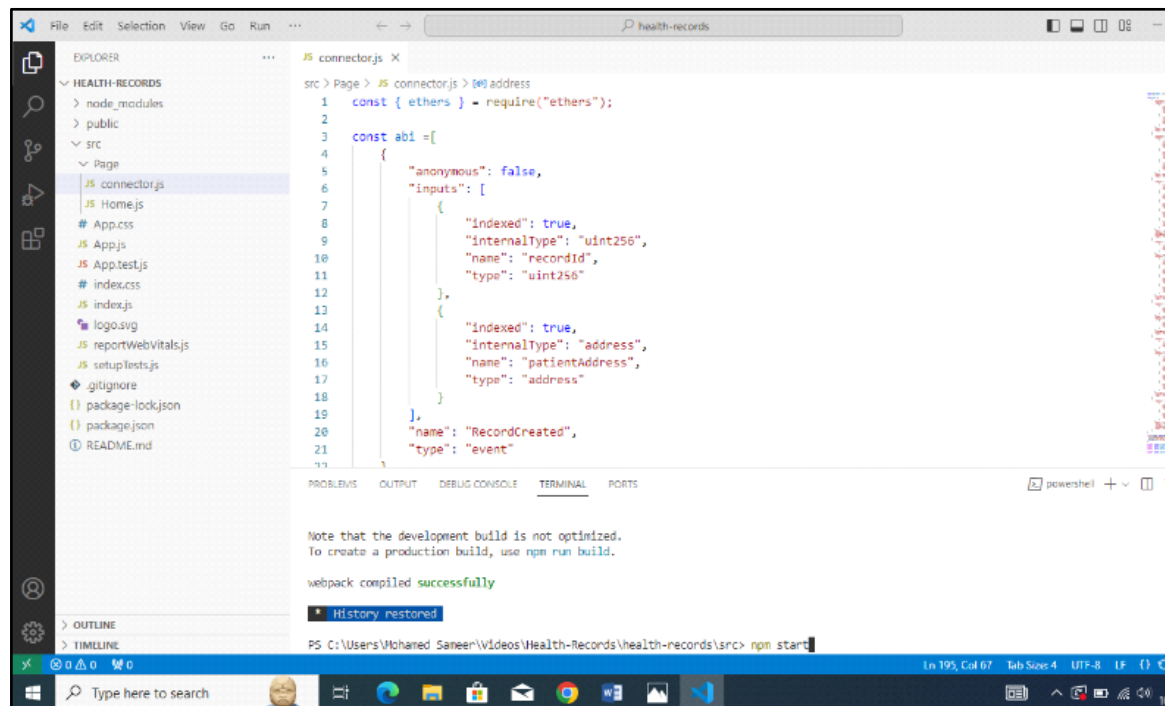
| S . N O | Parameter s | Value s | Screenshots |
|------------------|-----------------------|----------------------------|---|
| 1 . | Information Gathering | Setup all the prerequisite |  |

| | | | |
|----|----------------------|---|---|
| 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. |  |

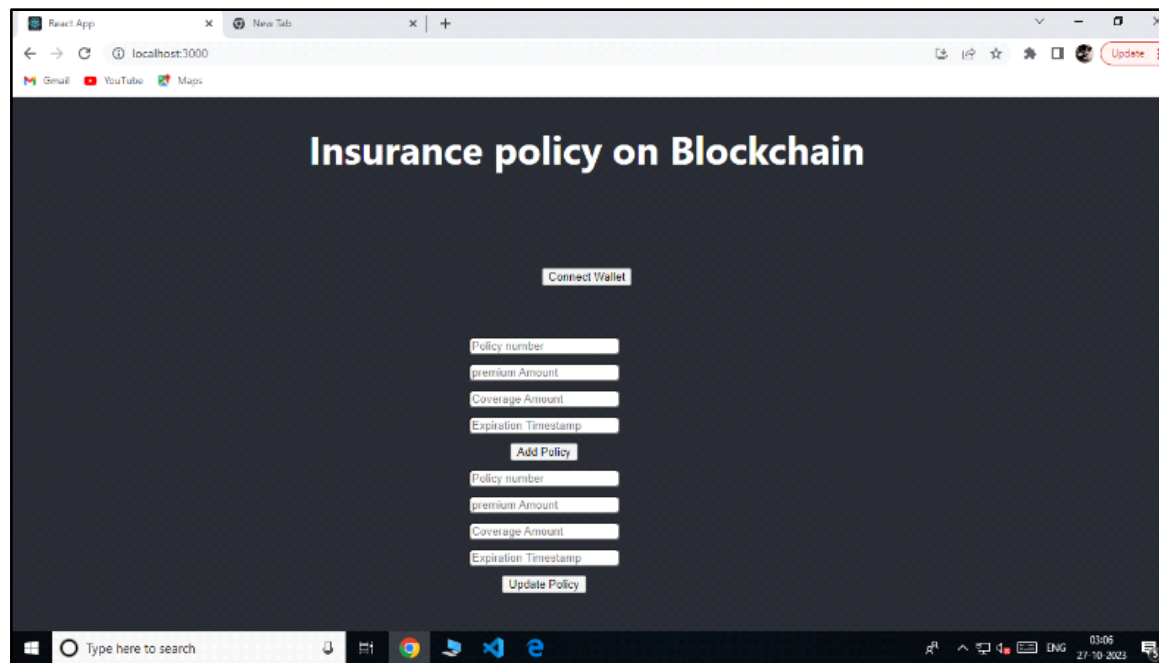
| | | | |
|---|-------------------------|---|--|
| 4 | <p>Open filemanager</p> | <p>Open the extracted file and click on the folder. Open src, and search for utiles. Open cmd enter commands</p> <ol style="list-style-type: none"> 1.npm install 2.npm bootstrap 3. npm start |  <p>The screenshot displays the Remix Ethereum IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel shows the contract 'Insurance - fic.sol' selected. The 'GAS LIMIT' is set to 3000000, and the 'VALUE' is 0 Wei. The 'Deploy' button is highlighted. Below this, the 'Transactions recorded' and 'Deployed Contracts' sections are visible, with a message stating 'Currently you have no contract instances to interact with.' The main editor on the right shows the Solidity code for the 'Insurance' contract, which includes a 'struct InsurancePolicy' and a 'mapping' for policy counts. The bottom status bar indicates the current block hash and number.</p> |
|---|-------------------------|---|--|

| | | | |
|---|----------------------|---|--|
| 5 | LOCALHOST IP ADDRESS | Copy the address and open it to chrome so you can see the frontend of your project. |  |
|---|----------------------|---|--|

- OUTPUT



OUTPUT SCREENSHOTS



- **ADVANTAGES & DISADVANTAGES**

Advantages :

Enhanced Data Security: Utilizing blockchain technology ensures enhanced security and immutability of electronic health records, protecting

sensitive patient information from unauthorized access and tampering.

Interoperability: The project fosters seamless data exchange among healthcare providers, improving collaboration, patient care, and overall healthcare system efficiency.

Patient Empowerment: By allowing patients control over their health data and granting or revoking access, the project empowers individuals to actively manage their healthcare, fostering a sense of ownership and engagement.

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.

User Adoption: Healthcare professionals and patients might face a learning curve when transitioning to a new system. Adequate training and user-friendly interfaces are essential for successful adoption.

Regulatory Compliance: Adhering to healthcare 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.

- **CONCLUSION**

In conclusion, the "Blockchain for Electronic Health Record" project represents a significant step toward revolutionizing healthcare data management. By harnessing the power of blockchain technology, the project offers a secure, interoperable, and patient-centric solution for managing electronic health records. The advantages of enhanced data security, improved interoperability, patient 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.

As the project moves forward, collaboration among stakeholders, ongoing user education, and a commitment to staying abreast of regulatory changes will be key. With the right approach, the "Blockchain for Electronic Health Record" project has the potential to shape the future of healthcare data management, ensuring better patient outcomes, streamlined workflows, and a more secure healthcare ecosystem.

- **FUTURE SCOPE**

The "Blockchain for Electronic Health Record" 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 health applications, allowing patients to securely access and manage their health records on their smartphones. Integration with wearables and IoT

devices could enable real-time health monitoring.

2. Telemedicine and Remote Patient Monitoring: Implement blockchain for secure transmission of medical data in telemedicine services. Incorporate smart contracts for automated billing, appointment scheduling, and patient-doctor interactions.

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 pharmaceuticals and medical supplies throughout the supply chain. Ensure the authenticity, safety, and integrity of medications, reducing the risk of counterfeit drugs.

5. Healthcare Payments and Insurance: Integrate blockchain for transparent and secure healthcare payments, reducing fraud and administrative costs.

Smart contracts can automate insurance claims, ensuring faster and accurate processing.

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

7. Global Health Records Exchange: Collaborate with international healthcare organizations to create a global, interoperable health records network. Blockchain can facilitate secure cross-border data exchange, vital for travelers and expatriates.

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

9. AI and Predictive Analysis: Combine blockchain with artificial intelligence for predictive healthcare analytics. Securely analyze large datasets to predict disease

outbreaks, optimize healthcare resources, and improve patient care.

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

• 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 healthcare providers, patients, 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,

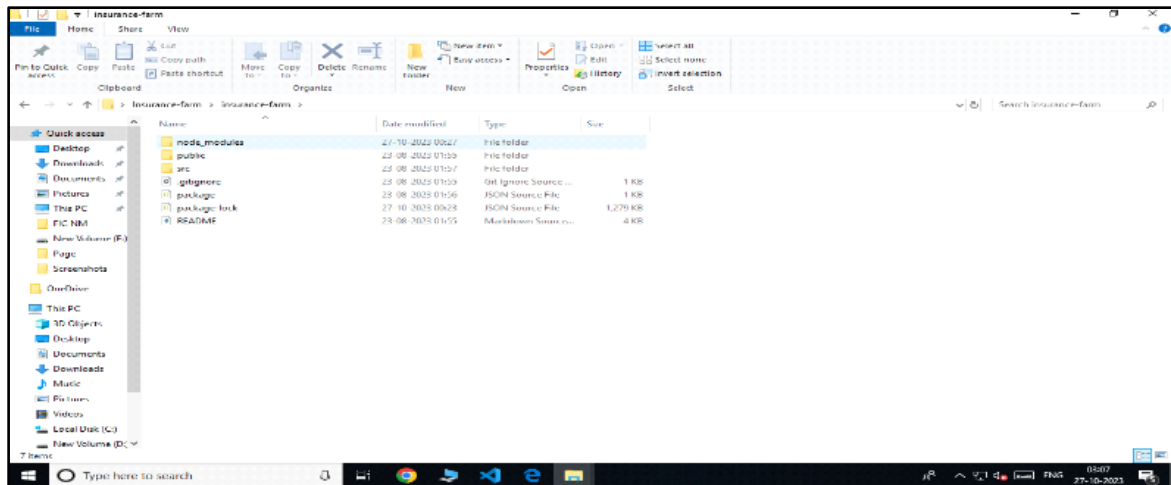
articles, and resources used during the project development. Links to relevant documentation, libraries, and frameworks.

10. Glossary:

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

SOURCE CODE

Folder Structure :



INDEX.HTML

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract Insurance {

 struct InsurancePolicy {

 address holder;

 string policyNumber;

 uint256 premiumAmount;

 uint256 coverageAmount;

 uint256 expirationTimestamp;

 }

 mapping(uint256 => InsurancePolicy) public policies;

 uint256 public policyCount;

 event PolicyAdded(uint256 policyId, address holder, string policyNumber, uint256 premiumAmount, uint256 coverageAmount, uint256 expirationTimestamp);

 event PolicyUpdated(uint256 policyId, uint256 premiumAmount, uint256 coverageAmount, uint256 expirationTimestamp);

 modifier onlyHolder(uint256 _policyId) {

```

        require(policies[_policyId].holder == msg.sender, "Only the policy holder can perform
this action");

        _;
    }

    function addPolicy(string memory _policyNumber, uint256 _premiumAmount, uint256
_coverageAmount, uint256 _expirationTimestamp) external {

        policyCount++;

        policies[policyCount] = InsurancePolicy(msg.sender, _policyNumber, _premiumAmount,
_coverageAmount, _expirationTimestamp);

        emit PolicyAdded(policyCount, msg.sender, _policyNumber, _premiumAmount,
_coverageAmount, _expirationTimestamp);
    }

    function updatePolicy(uint256 _policyId, uint256 _premiumAmount, uint256
_coverageAmount, uint256 _expirationTimestamp) external onlyHolder(_policyId) {

        InsurancePolicy storage policy = policies[_policyId];

        policy.premiumAmount = _premiumAmount;

        policy.coverageAmount = _coverageAmount;

        policy.expirationTimestamp = _expirationTimestamp;

        emit PolicyUpdated(_policyId, _premiumAmount, _coverageAmount,
_expirationTimestamp);
    }

    function getPolicyDetails(uint256 _policyId) external view returns (address holder, string
memory policyNumber, uint256 premiumAmount, uint256 coverageAmount, uint256
expirationTimestamp) {

        InsurancePolicy memory policy = policies[_policyId];

        return (policy.holder, policy.policyNumber, policy.premiumAmount,
policy.coverageAmount, policy.expirationTimestamp);
    }
}

```

CONNECTOR.JS

```

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

const abi = [
{

```

```
"anonymous": false,
"inputs": [
  {
    "indexed": false,
    "internalType": "uint256",
    "name": "policyId",
    "type": "uint256"
  },
  {
    "indexed": false,
    "internalType": "address",
    "name": "holder",
    "type": "address"
  },
  {
    "indexed": false,
    "internalType": "string",
    "name": "policyNumber",
    "type": "string"
  },
  {
    "indexed": false,
    "internalType": "uint256",
    "name": "premiumAmount",
    "type": "uint256"
  },
  {
    "indexed": false,
    "internalType": "uint256",
    "name": "coverageAmount",
    "type": "uint256"
  },
  {
    "indexed": false,
    "internalType": "uint256",
    "name": "expirationTimestamp",
    "type": "uint256"
  }
],
"name": "PolicyAdded",
"type": "event"
},
{
  "anonymous": false,
  "inputs": [
    {
      "indexed": false,
```

```
"internalType": "uint256",
"name": "policyId",
"type": "uint256"
},
{
  "indexed": false,
  "internalType": "uint256",
  "name": "premiumAmount",
  "type": "uint256"
},
{
  "indexed": false,
  "internalType": "uint256",
  "name": "coverageAmount",
  "type": "uint256"
},
{
  "indexed": false,
  "internalType": "uint256",
  "name": "expirationTimestamp",
  "type": "uint256"
}
],
"name": "PolicyUpdated",
"type": "event"
},
{
  "inputs": [
    {
      "internalType": "string",
      "name": "_policyNumber",
      "type": "string"
    },
    {
      "internalType": "uint256",
      "name": "_premiumAmount",
      "type": "uint256"
    },
    {
      "internalType": "uint256",
      "name": "_coverageAmount",
      "type": "uint256"
    },
    {
      "internalType": "uint256",
      "name": "_expirationTimestamp",
      "type": "uint256"
    }
  ]
}
```

```

    }
  ],
  "name": "addPolicy",
  "outputs": [],
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "uint256",
      "name": "_policyId",
      "type": "uint256"
    }
  ],
  "name": "getPolicyDetails",
  "outputs": [
    {
      "internalType": "address",
      "name": "holder",
      "type": "address"
    },
    {
      "internalType": "string",
      "name": "policyNumber",
      "type": "string"
    },
    {
      "internalType": "uint256",
      "name": "premiumAmount",
      "type": "uint256"
    },
    {
      "internalType": "uint256",
      "name": "coverageAmount",
      "type": "uint256"
    },
    {
      "internalType": "uint256",
      "name": "expirationTimestamp",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
}

```



```
"inputs": [  
  {  
    "internalType": "uint256",  
    "name": "",  
    "type": "uint256"  
  }  
],  
"name": "policies",  
"outputs": [  
  {  
    "internalType": "address",  
    "name": "holder",  
    "type": "address"  
  },  
  {  
    "internalType": "string",  
    "name": "policyNumber",  
    "type": "string"  
  },  
  {  
    "internalType": "uint256",  
    "name": "premiumAmount",  
    "type": "uint256"  
  },  
  {  
    "internalType": "uint256",  
    "name": "coverageAmount",  
    "type": "uint256"  
  },  
  {  
    "internalType": "uint256",  
    "name": "expirationTimestamp",  
    "type": "uint256"  
  }  
],  
"stateMutability": "view",  
"type": "function"  
},  
{  
  "inputs": [],  
  "name": "policyCount",  
  "outputs": [  
    {  
      "internalType": "uint256",  
      "name": "",  
      "type": "uint256"  
    }  
  ]  
}
```

```

    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "_policyId",
        "type": "uint256"
      },
      {
        "internalType": "uint256",
        "name": "_premiumAmount",
        "type": "uint256"
      },
      {
        "internalType": "uint256",
        "name": "_coverageAmount",
        "type": "uint256"
      },
      {
        "internalType": "uint256",
        "name": "_expirationTimestamp",
        "type": "uint256"
      }
    ],
    "name": "updatePolicy",
    "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 = "0xd91BQZKqdp2CV3QV5nUEsqSg1ygegLmqRygj"

```

```

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

```

GITHUB & DEMO VIDEO

GitHub Link :

<https://github.com/MohamedFarhan21/Farmer-Insurance-Chain>

Demo Video :

https://drive.google.com/drive/folders/1StriHSjvhXhk1TGfPwYX_1CZhcjAvkh