APPLICATION OF BLOCK CHAIN TECHNOLOGY IN DISASTER MANAGEMENT

A PROJECT REPORT

OF PROJECT-III (PROJ- CS881)

BACHELOR OF TECHNOLOGY
in
Information Technology

(From **Maulana Abul Kalam Azad University of Technology, West Bengal**)

SUBMITTED BY

Amit Shah (13000220004)
Debraj Bhattacharjee (13000220014)
Mainak Paul (13000220023)
Souvik Chattopadhyay (13000220025)

Under the Supervision of
Dr. Tanmay Bhattacharya

Department of Information Technology
Techno Main Salt Lake
Kolkata -700091

Department of Information Technology
Techno Main Salt Lake
EM-4/1, Salt Lake City, Kolkata-700091

BONAFIDE CERTIFICATE

Certified that this synopsis for the project titled "APPLICATION OF BLOCK CHAIN TECHNOLOGY IN DISASTER MANAGEMENT" is a part of the project work being carried out by "AMIT SHAH, DEBRAJ BHATTACHARJEE, MAINAK PAUL, SOUVIK CHATTOPADHYAY" under my supervision.

Full Signature of the Candidates (with date)

1. ____________________
2. ____________________
3. ____________________
4. _____________________

____
Tanmay Bhattyacharya
Associate Professor
Department of Information Technology
Techno Main Saltlake

____
Dr. Subhamita Mukherjee
Head of the Department
Department of Information Technology
Techno Main Saltlake

ACKNOWLEDGEMENT

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to the teaching fraternity of the Department of Information Technology, for giving us this opportunity to undertake this project and also supporting us whole heartedly.

We also wish to express our gratitude to the HOD and all our teachers of the Department of Information Technology for their kind hearted support, guidance and utmost endeavor to groom and develop our academic skills.

At the end we would like to express our sincere thanks to all our friends and others who helped us directly or indirectly during the effort in shaping this concept till now.

Full Signature of the Candidates (with date)

1. ____________________

2. ____________________

3. ____________________

4. _____________________

Abstract

In the face of escalating challenges posed by natural disasters and humanitarian crises, the necessity for efficient and timely relief measures has become strikingly apparent. Traditional disaster management systems grapple with critical issues such as real-time data sharing, resource allocation, and transparency. This research propounds a paradigm shift towards decentralized disaster relief, harnessing the latent potential of blockchain technology. The project's core objective is to establish a secure and transparent platform by leveraging blockchain's intrinsic attributes, namely decentralization, immutability and smart contracts. This abstract amalgamates the comprehensive goals and methodology of the project, emphasizing the urgent need for improved coordination, communication, and resource allocation in disaster response operations. The endeavor aspires to elevate data integrity, enhance decision-making processes, and instill accountability. Situated at the nexus of technology and humanitarianism, our project on the "Application of Blockchain Technology in Disaster Management" unfolds a pioneering narrative. It delves into the design, implementation and impact of a decentralized blockchain system, with a core objective of enhancing real-time data sharing in disaster relief through the transformative power of blockchain technology. The project incorporates innovative approaches in missing persons' tracking, aiddistribution, critical area identification, refugee relief camp mapping, disaster reporting, record-keeping, security and continuous system improvement, promising to reshape humanitarian efforts and disaster response mechanisms fundamentally.

Keywords

Blockchain Technology · Disaster Management · Decentralization · Real-time Data Sharing · Smart Contracts · Humanitarian Technology

Contents

Title Page	
Certificate	**i**
Acknowledgement	**ii**
Abstract	**iii**
Table of Contents	**iv**
List of Figures	**v**
1 Introduction	**1-2**
2 Literature Review	**3-4**
3 Problem Definition	**5-8**
4 Software Design	**9-13**
5 Software and Hardware Requirements	**14-15**
6 Code Templates	**16-26**
7 Experimental Results	**27-34**
8 Conclusion	**35-36**
References	**37-38**

List of Figures

1 Conceptual Diagram... 10
2 Flow Chart 12
3 Disaster Reporting and Management(Solidity Program) 17
4 Critical Area Management(Solidity Program)...................................... 18
5 Missing People Log (Solidity Program) 19
6 Aid Package Distribution (Solidity Program) 20
7 Refugee-Relief Camp Mapping (Solidity Program) 21
8 Medical Record Management (Solidity Program) 22
9 Cryptocurrency Based Aid Distribution (Solidity Program)...................23
10 Smart Insurance Contracts (Solidity Program) 24
11 Data Privacy and Security Enhancement (Solidity Program).................25
12 Continuous Monitoring and Improvement (Solidity Program)...............26
13 Disaster Reporting and Management (Output) 28
14 Critical Area Management (Output).. 28
15 Missing People Log (Output) 29
16 Aid Package Distribution (Output) 29
17 Refugee-Relief Camp Mapping (Output)...30
18 Medical Record Management (Output) 30
19 Cryptocurrency Based Aid Distribution (Output)................................. 31
20 Smart Insurance Contracts (Output) 31
21 Data Privacy and Security Enhancement (Output)................................32
22 Continuous Monitoring and Improvement(Output) 32
23 Ganache Output... 33
24 MetaMask Wallet 34

Chapter – I

INTRODUCTION

1 INTRODUCTION

I. Introduction of the Project

The integration of blockchain technology into disaster managementrepresents a paradigm shift in the way we approach and respond to crises. Traditional disaster management systems often grapple with inefficiencies, lack of transparency, and delayed response times [1]. Blockchain, renowned for its decentralized and secure nature, offers a novel solution to address these challenges.

II. Objective of the Industrial Project

In the face of escalating natural disasters and humanitarian crises, the application of blockchain technology holds immense significance [2]. The decentralized nature of blockchain ensures that critical information is not held in a central repository, reducing the risk of data loss or manipulation.

III. Summary of the Report

The immutability of blockchain ensures that once data is recorded, it cannot be altered or tampered with, fostering trust and accountability among stakeholders [5]. This feature is particularly vital in disaster scenarios where swift and accurate information dissemination can be a matter of life and death. Applying blockchain in disaster management has the potential to revolutionize how we prepare for, respond to, and recover from crises. It offers a decentralized, secure, and transparent framework that can significantly enhance the efficiency and effectiveness of disaster relief efforts [3].

Chapter – II

LITERATURE SURVEY

2 Literature Survey

Literature	Authors	Focus	Key Points
Blockchain in Disaster Response	Tapscott and Tapscott (2016) [3]	Transformational potential of blockchain in disaster response	- Emphasis on the decentralized nature of blockchain. - Enhancement of security and efficiency in disaster relief operations.
Smart Contracts for Coordination	Casey (2018) [11]	Role of smart contracts in automating and streamlining coordination efforts	- Facilitation of transparent and self-executing agreements. - Relevance to aid package distribution and critical area management.
Realtime Data Sharing in Disasters	Peters et al. (2019) [13]	Conceptualization of real-time data sharing in disaster scenarios	- Emphasis on immediacy and transparency in disaster management. - Alignment with the core objectives of the project.
Community Engagement and Privacy Concerns	Zhang et al. (2018) [15]	Blockchain’s role in community engagement during disaster response	- Exploration of ethical considerations and a community-centric approach. - Guidance on addressing privacy concerns and ensuring community participation.
Integration with IoT for Comprehensive Solutions	Mense and Sprenger (2019) [4]	Synergies between blockchain and IoT in disaster scenarios	- Insights into comprehensive solutions for real-time monitoring and response. - Informing considerations for future technological integration.
Challenges and Opportunities in Blockchain-based Disaster Management	YliHuumo et al. (2016) [6]	Critical examination of challenges and opportunities in blockchain for disaster management	- Foundation for understanding potential hurdles. - Strategies for effectively harnessing blockchain for humanitarian purposes.

Chapter – III

DEFINITION OF THE PROBLEM WITH THE MODULES AND FUNCTIONALITIES

3 DEFINITION OF THE PROBLEM WITH THE MODULES AND FUNCTIONALITIES

3.1. Problem Statement:

- **Context:** Escalating natural disasters and humanitarian crises demand a paradigm shift in disaster relief strategies.
- **Challenge:** Conventional systems struggle with real-time data sharing, resource allocation, and transparency.
- **Objective:** Advocate for a decentralized disaster relief paradigm using blockchain technology.

3.2. Modules and Functionalities:

Step 1: Disaster Reporting and Management

- **Input:** Disaster details (name, location, timestamp, description, critical areas).
- **Process:** Smart contracts manage disasters, ensuring transparency, traceability, and security.
- **Output:** Efficient disaster response and management.

Step 2: Critical Area Management

- **Input:** Area name, description, severity.
- **Process:** Smart contracts add/remove critical areas, prioritize response efforts.
- **Output:** List of active critical areas for effective response.

Step 3: Missing People Log

- **Input:** Name, last known location, description.
- **Process:** Smart contracts in Solidity securely log andupdate missing people's information.
- **Output:** Secure, transparent, and up-to-date list of missing people.

Step 4: Aid Package Distribution

- **Input:** Recipient information, aid package details.
- **Process:** Smart contracts coordinate aid package distribution,tracking movement.
- **Output:** Efficient aid distribution to affected areas.

Step 5: Refugee Relief Camp Mapping

- **Input:** Refugee and camp information.
- **Process:** Smart contracts assign refugees to camps, manage allocations.
- **Output:** Details of relief camps and active refugees.

Step 6: Medical Record Management

- **Input:** Medical Records (Name, Address).
- **Process:** Blockchain secures and stores medical records with immutability.
- **Output:** Comprehensive and secure medical records for injured individuals.

Step 7: Cryptocurrency-Based Aid Distribution

- **Input:** Recipient's digital wallet information.
- **Process:** Use cryptocurrency for transparent, quick, and efficient financial aid distribution.
- **Output:** Reduced corruption and faster distribution of funds.

Step 8: .Smart Insurance Contracts

- **Input:** Insurance policy details, damage assessment.
- **Process:** Implement smart insurance contracts to automatically release funds based on predefined conditions and assessments.
- **Output:** Quick and fair insurance claims processing for affected individuals.

Step 9: Data Privacy and Security Enhancements

- **Input:** Sensitive data (medical records, personal information).
- **Process**: Employ advanced encryption and zero-knowledge proofs to ensure data privacy and security.
- **Output:** Secure handling of sensitive information, maintaining trust and compliance with regulations.

Step 10: Continuous Monitoring and Improvement

- **Input:** Real-time system data, user feedback, emerging technologies.
- **Process:** Continuous monitoring for irregularities, real-time data up- dates, user feedback analysis.
- **Output:** Well monitored, up-to-date, and ever improving blockchain based humanitarian system.

Chapter – IV

SOFTWARE DESIGN

4 DETAILED DIAGRAM

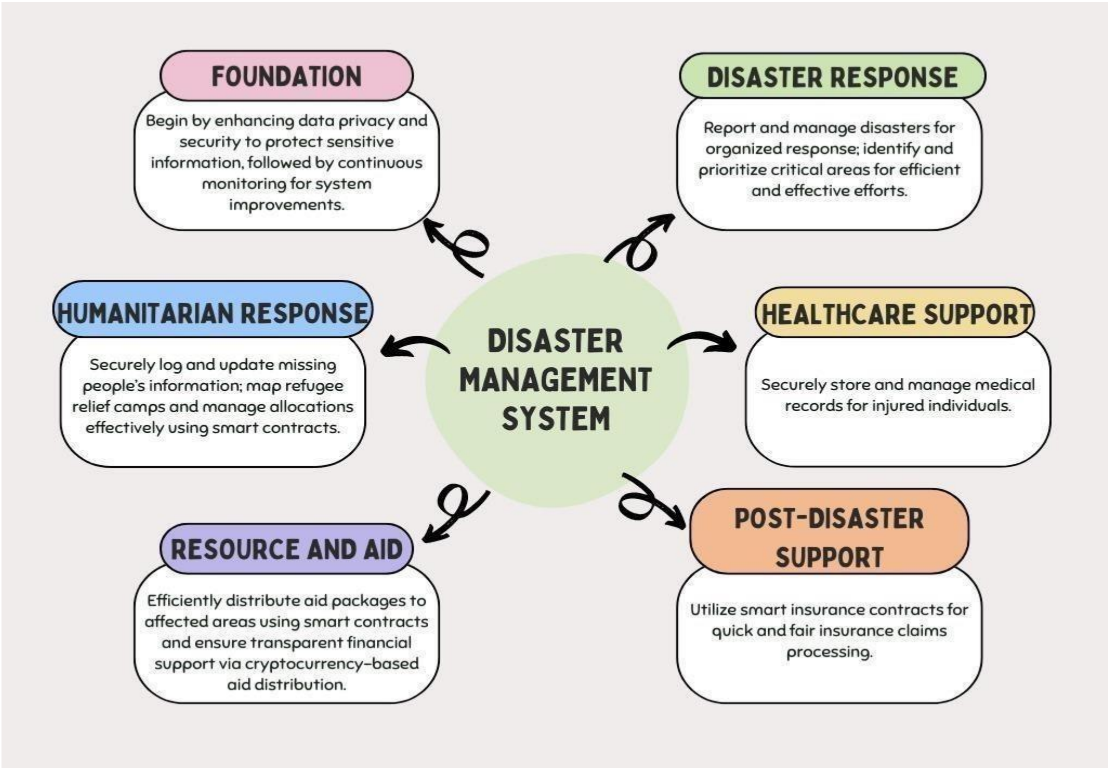


Fig 1: Detailed Diagram

1. Foundation Setup:

- Begin by enhancing data privacy and security to safeguard sensitive information.
- Implement continuous monitoring to identify areas for system improvements.

2. Disaster Response Initialization:

- Report and manage disasters systematically to facilitate an organized response.
- Identify and prioritize critical areas to focus response efforts effectively.

3. **Humanitarian Response Management:**

- Securely log missing individuals and maintain up-to-date information for effective search and rescue operations.
- Map refugee relief camps and allocate resources efficiently to support displaced populations.

4. **Healthcare Support:**

- Establish a secure system to store and manage medical records,ensuring privacy and accessibility for injured individuals.

5. **Resource and Aid Management:**

- Distribute aid packages efficiently to affected areas, prioritizingurgent needs and ensuring fair distribution.
- Implement cryptocurrency-based aid distribution for transparent and accountable financial support.

6. **Post-Disaster Support:**

- Utilize smart insurance contracts to expedite the processing of insurance claims, providing quick and equitable compensation to affected individuals.

By adhering to this structured workflow, the blockchain-based disaster management system can effectively address various aspects of disaster response, resource management, and post- disaster support, ensuring comprehensive, efficient, and transparent assistance to affected communities.

FLOWCHART DIAGRAM

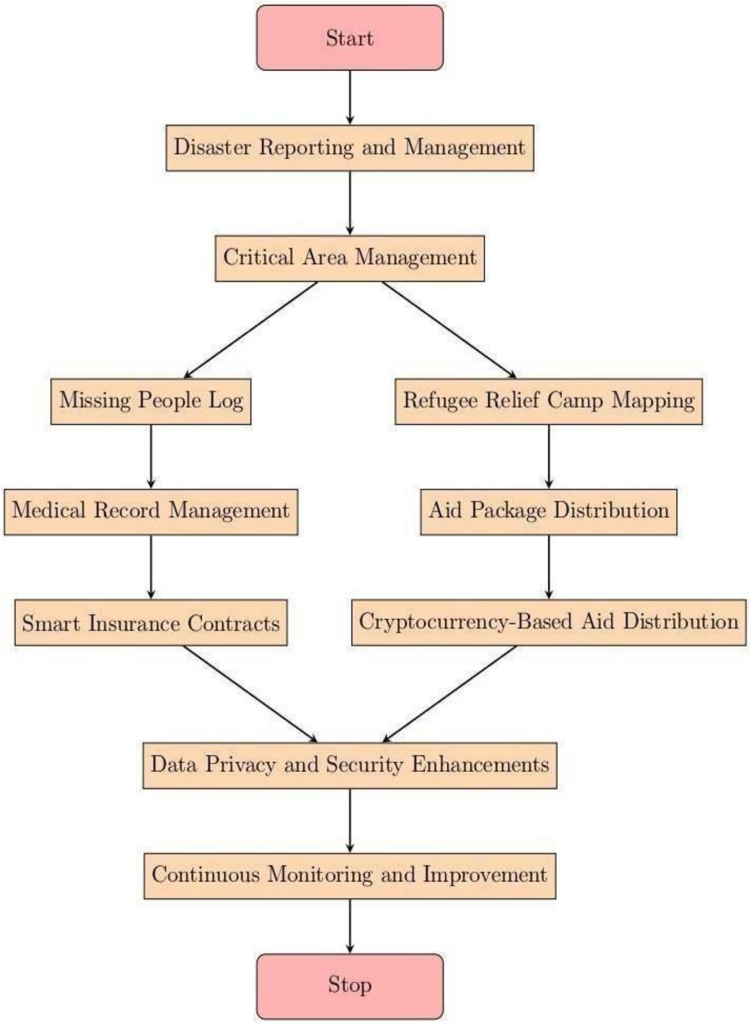


Figure 2: Flowchart Diagram

- **Start:** This is the initiating point of the process.

- **Disaster Reporting and Management:** This step likely involves gathering information about the disaster, assessing the damage, and coordinating relief efforts.

- **Critical Area Management:** This step focuses on identifying and managing critical areas affected by the disaster. This might involve search and rescue operations, or setting up temporary shelters.

- **Missing People Log:** A log is created to track people missing in the disaster.

- **Refugee Relief Camp Mapping:** This step involves mapping the locations of refugee camps set up to house people displaced by the disaster.

- **Medical Record Management:** This step likely involves setting up a system to track and manage the medical records of people affected by the disaster.

- **Aid Package Distribution:** This step involves distributing aid packages to those in need.

- **Smart Insurance Contracts:** This step likely involves using smart contracts to automate and streamline the process of distributing insurance payouts to those affected by the disaster.

- **Cryptocurrency-Based Aid Distribution:** This step involves using cryptocurrency to distribute aid to those affected by the disaster.

- **Data Privacy and Security Enhancements:** This step involves putting measures in place to protect the privacy and security of data collected during the disaster response process.

- **Continuous Monitoring and Improvement:** This step involves continuously monitoring the disaster response process and making improvements as needed.

- **Stop:** This is the endpoint of the process.

Chapter – V

SOFTWARE AND HARDWARE REQUIREMENTS

5 SOFTWARE AND HARDWARE REQUIREMENT

5.1. Software Requirements:

- **Blockchain Development Platform:**
  - *Ethereum* (Solidity for smart contracts)
  - *Truffle* for *Ethereum* smart contract development
  - *Ganache* for local blockchain deployment and testing
  - *Visual Studio Code* for *Ethereum* smart contract coding

- **Integrated Development Environment (IDE):**
  - *Visual Studio Code* for *Ethereum* smart contract development

- **Testing:**
  - *Truffle* for *Ethereum* smart contract testing

5.2. Hardware Requirements:

- **Storage:**
  - Adequate storage for local development environment

- **Memory (RAM):**
  - Sufficient RAM for smooth development and testing

- **Network:**
  - Standard internet connection for interacting with the Ethereum blockchain

Chapter – VI

CODE TEMPLATES

6 CODE TEMPLATES

The code templates below outline the classes, their functionalities, and methods with input and output parameters for the described blockchain-based disaster management system.

6.1. Disaster Reporting and Management

```
DisasterReportingAndManagement.sol ×
web > contracts > DisasterReportingAndManagement.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract DisasterReportingAndManagement {
    // Structure to store information about a disaster report
    struct DisasterReport {
        string name;
        string location;
        uint256 timestamp;
        string description;
        bool isResolved;
        address[] stakeholders;
    }

    DisasterReport[] public disasterReports;
    uint256 public totalReports;

    // Event to log when a new disaster report is added
    event DisasterReportAdded(string name, string location, uint256 timestamp, string description);

    // Event to log when a disaster report is marked as resolved
    event DisasterReportResolved(uint256 reportId);

    // Function to add a new disaster report
    function addDisasterReport(
        string memory _name,
        string memory _location,
        uint256 _timestamp,
        string memory _description,
        address[] memory _stakeholders
    ) public {
        totalReports++;
        disasterReports.push(
            DisasterReport({
                name: _name,
                location: _location,
                timestamp: _timestamp,
                description: _description,
                isResolved: false,
                stakeholders: _stakeholders
            })
        );

        emit DisasterReportAdded(_name, _location, _timestamp, _description);
    }

    // Function to mark a disaster report as resolved
    function markReportAsResolved(uint256 _reportId) public {
        require(_reportId > 0 && _reportId <= totalReports, "Invalid report ID");
        disasterReports[_reportId - 1].isResolved = true;

        emit DisasterReportResolved(_reportId);
    }
}
```

Fig 3.1: Disaster Reporting and Management (Solidity Program)

This Solidity smart contract, "DisasterReportingandManagement," manages disaster reports with a structure storing details like name, location, timestamp, description, resolution status, and stakeholders. It includes functions to add new reports and mark reports as resolved, with corresponding events for Transparency and accountability in disaster management on the blockchain [3].

6.2. Critical Area Management

CriticalAreaManagement.sol

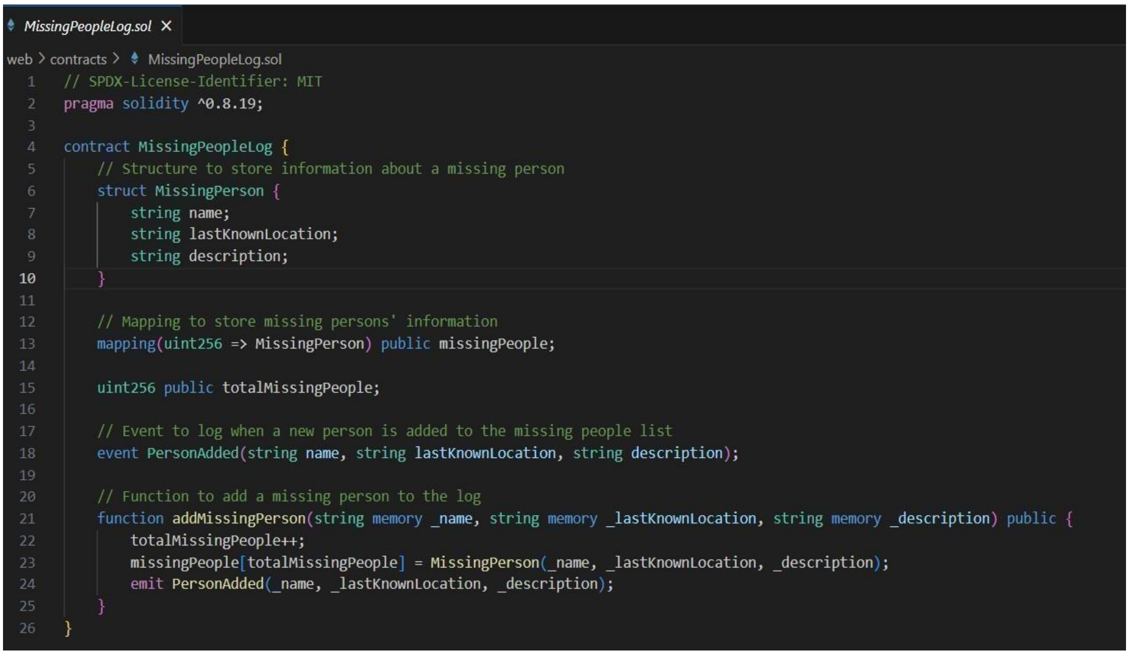
web > contracts > CriticalAreaManagement.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract CriticalAreaManagement {
    // Structure to store information about a critical area
    struct CriticalArea {
        string name;
        string description;
        uint8 severity; // Severity can be on a scale of 1 to 10
    }

    // Mapping to store critical areas
    mapping(uint256 => CriticalArea) public criticalAreas;
    uint256 public totalCriticalAreas;

    // Event to log when a new critical area is added
    event CriticalAreaAdded(string name, string description, uint8 severity);

    // Event to log when a critical area is removed
    event CriticalAreaRemoved(uint256 areaId);

    // Function to add a new critical area
    function addCriticalArea(string memory _name, string memory _description, uint8 _severity) public {
        totalCriticalAreas++;
        criticalAreas[totalCriticalAreas] = CriticalArea(_name, _description, _severity);

        emit CriticalAreaAdded(_name, _description, _severity);
    }

    // Function to remove a critical area
    function removeCriticalArea(uint256 _areaId) public {
        require(_areaId > 0 && _areaId <= totalCriticalAreas, "Invalid area ID");

        delete criticalAreas[_areaId];

        emit CriticalAreaRemoved(_areaId);
    }
}
```

Fig 3.2: Critical Area Management (Solidity Program)

This Solidity smart contract, "CriticalAreaManagement," handles critical areas withdetails such as name, description, and severity on a scale of 1 to 10. It features functions to add and remove critical areas, with corresponding events, promoting transparency and dynamic management of critical zones on the blockchain [7].

6.3 Missing People Log



MissingPeopleLog.sol

web > contracts > MissingPeopleLog.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract MissingPeopleLog {
    // Structure to store information about a missing person
    struct MissingPerson {
        string name;
        string lastKnownLocation;
        string description;
    }

    // Mapping to store missing persons' information
    mapping(uint256 => MissingPerson) public missingPeople;

    uint256 public totalMissingPeople;

    // Event to log when a new person is added to the missing people list
    event PersonAdded(string name, string lastKnownLocation, string description);

    // Function to add a missing person to the log
    function addMissingPerson(string memory _name, string memory _lastKnownLocation, string memory _description) public {
        totalMissingPeople++;
        missingPeople[totalMissingPeople] = MissingPerson(_name, _lastKnownLocation, _description);
        emit PersonAdded(_name, _lastKnownLocation, _description);
    }
}
```

Fig 3.3: Missing People Log (Solidity Program)

The Solidity smart contract, named "MissingPeopleLog," defines a structure to store details of missing persons. It includes functions to add a missing person, updating a mapping of individuals. The contract emits an event when a new person is added, enhancing transparency and traceability in managing missing persons' information on the blockchain.

6.4. Aid Package Distribution


AidPackageDistribution.sol
web > contracts > AidPackageDistribution.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract AidPackageDistribution {
    // Structure to store information about an aid package
    struct AidPackage {
        address sender;
        string recipient;
        string packageDetails;
        string sourceLocation;
        string destinationLocation;
        bool delivered;
    }

    // Mapping to store aid packages
    mapping(uint256 => AidPackage) public aidPackages;

    uint256 public totalAidPackages;

    // Event to log when a new aid package is created
    event AidPackageCreated(
        address sender,
        string recipient,
        string packageDetails,
        string sourceLocation,
        string destinationLocation
    );

    // Event to log when an aid package is marked as delivered
    event AidPackageDelivered(uint256 packageId);

    // Function to create a new aid package
    function createAidPackage(
        string memory _recipient,
        string memory _packageDetails,
        string memory _sourceLocation,
        string memory _destinationLocation
    ) public {
        totalAidPackages++;
        aidPackages[totalAidPackages] = AidPackage(
            msg.sender,
            _recipient,
            _packageDetails,
            _sourceLocation,
            _destinationLocation,
            false
        );

        emit AidPackageCreated(msg.sender, _recipient, _packageDetails, _sourceLocation, _destinationLocation);
    }

    // Function to mark an aid package as delivered
    function markAidPackageDelivered(uint256 _packageId) public {
        require(_packageId > 0 && _packageId <= totalAidPackages, "Invalid package ID");
        require(aidPackages[_packageId].sender == msg.sender, "Only the sender can mark as delivered");

        aidPackages[_packageId].delivered = true;

        emit AidPackageDelivered(_packageId);
    }
}
```

Fig 3.4: Aid Package Distribution (Solidity Program)

This Solidity smart contract, "AidPackageDistribution," manages aid packages with a structure capturing sender, recipient, details, source, destination, and delivery status. It includes functions to create new aid packages and mark them as delivered, enhancing transparency and accountability in aid distribution on the blockchain.

6.5. Refugee Relief Camp Mapping

```
RefugeeReliefCampMapping.sol
web > contracts > RefugeeReliefCampMapping.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract RefugeeReliefCampMapping {
    // Structure to store information about a refugee
    struct Refugee {
        string name;
        string location;
        uint256 campId;
    }

    // Structure to store information about a relief camp
    struct ReliefCamp {
        string name;
        string location;
        uint256 size; // Capacity of the camp
        uint256 currentOccupancy; // Number of refugees in the camp
    }

    Refugee[] public refugees;
    ReliefCamp[] public reliefCamps;
    uint256 public totalRefugees;
    uint256 public totalReliefCamps;

    // Event to log when a new refugee is added
    event RefugeeAdded(string name, string location, uint256 campId);

    // Event to log when a new relief camp is added
    event ReliefCampAdded(string name, string location, uint256 size);

    // Function to add a new refugee and assign them to a camp
    function addRefugee(string memory _name, string memory _location, uint256 _campId) public {
        require(_campId > 0 && _campId <= totalReliefCamps, "Invalid camp ID");
        totalRefugees++;
        refugees.push(Refugee(_name, _location, _campId));

        emit RefugeeAdded(_name, _location, _campId);
    }

    // Function to add a new relief camp
    function addReliefCamp(string memory _name, string memory _location, uint256 _size) public {
        totalReliefCamps++;
        reliefCamps.push(ReliefCamp(_name, _location, _size, 0));

        emit ReliefCampAdded(_name, _location, _size);
    }

    // Function to allocate a refugee to a camp
    function allocateRefugeeToCamp(uint256 _refugeeId, uint256 _campId) public {
        require(_refugeeId > 0 && _refugeeId <= totalRefugees, "Invalid refugee ID");
        require(_campId > 0 && _campId <= totalReliefCamps, "Invalid camp ID");

        refugees[_refugeeId - 1].campId = _campId;
        reliefCamps[_campId - 1].currentOccupancy++;

        emit RefugeeAdded(refugees[_refugeeId - 1].name, refugees[_refugeeId - 1].location, _campId);
    }
}
```

Fig 3.5: Refugee-Relief Camp Mapping (Solidity Program)

This Solidity smart contract, "RefugeeReliefCampMapping," manages refugees and relief camps. It includes structures for refugees and relief camps, with functions to add new refugees, relief camps, and allocate refugees to camps, fostering transparent and organized mapping of refugees to suitable relief camps on the blockchain [9].

6.6. Medical Record Management

MedicalRecordManagement.sol

web > contracts > MedicalRecordManagement.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract MedicalRecordManagement {
    // Structure to store medical records
    struct MedicalRecord {
        string name;
        string addres;
        string dataHash; // Hash of the actual medical data
    }

    // Mapping to store medical records
    mapping(uint256 => MedicalRecord) public medicalRecords;

    uint256 public totalMedicalRecords;

    // Event to log when a new medical record is added
    event MedicalRecordAdded(string name, string addres, string dataHash);

    // Function to add a medical record to the system
    function addMedicalRecord(string memory _name, string memory _addres, string memory _dataHash) public {
        totalMedicalRecords++;
        medicalRecords[totalMedicalRecords] = MedicalRecord(_name, _addres, _dataHash);
        emit MedicalRecordAdded(_name, _addres, _dataHash);
    }
}
```

Fig 3.6: Medical Record Management (Solidity Program)

The Solidity smart contract, "MedicalRecordManagement," maintains medical records with a structure storing name, address, and a hash of the medical data. The contract includes functions to add new medical records, enhancing transparency and security in medical data management on the blockchain [16].

6.7. Cryptocurrency-Based Aid Distribution

```
CryptoCurrencyAidDistribution.sol ×
web > contracts > CryptoCurrencyAidDistribution.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract CryptoCurrencyAidDistribution {
    address public organization;
    mapping(address => uint256) public recipients;
    uint256 public totalRecipients;

    event RecipientAdded(address indexed recipient, uint256 amount);
    event AidDistributed(address indexed recipient, uint256 amount);
    event FundsDeposited(address indexed from, uint256 amount);

    modifier onlyOrganization() {
        require(msg.sender == organization, "Only the organization can perform this action");
        _;
    }

    constructor() {
        organization = msg.sender;
    }

    // Function to add a recipient and specify the amount of aid
    function addRecipient(address _recipient, uint256 _amount) public onlyOrganization {
        require(_recipient != address(0), "Invalid recipient address");
        require(_amount > 0, "Aid amount must be greater than zero");
        require(recipients[_recipient] == 0, "Recipient already added");

        recipients[_recipient] = _amount;
        totalRecipients++;
        emit RecipientAdded(_recipient, _amount);
    }

    // Function to distribute aid to all recipients
    function distributeAid() public onlyOrganization {
        for (uint256 i = 0; i < totalRecipients; i++) {
            address recipient = address(uint160(uint256(keccak256(abi.encodePacked(i)))));
            uint256 amount = recipients[recipient];

            if (amount > 0) {
                recipients[recipient] = 0;
                payable(recipient).transfer(amount);
                emit AidDistributed(recipient, amount);
            }
        }
    }
```

Fig 3.7: Cryptocurrency-Based Aid Distribution (Solidity Program)

The Solidity contract “CryptoCurrencyAidDistribution” allows an owner to manage and distribute cryptocurrency-based aid. Owners can add recipients, deposit and withdraw funds, and distribute aid securely. Recipients' aid amounts are stored and aid is transferred to their addresses, ensuring transparency and reducing corruption in financial aid distribution.

6.8. Smart Insurance Contracts

```
SecureDataStorage.sol    SmartInsurance.sol ×
web > contracts > SmartInsurance.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SmartInsurance {
    address public insurer;
    uint256 public policyCount = 0;

    struct Policy {
        uint256 policyId;
        address policyHolder;
        uint256 premium;
        uint256 payout;
        bool isActive;
        bool isClaimed;
    }

    struct DamageAssessment {
        uint256 policyId;
        uint256 damageAmount;
        bool isAssessed;
        bool isApproved;
    }

    mapping(uint256 => Policy) public policies;
    mapping(uint256 => DamageAssessment) public assessments;

    event PolicyCreated(uint256 policyId, address policyHolder, uint256 premium, uint256 payout);
    event ClaimSubmitted(uint256 policyId, uint256 damageAmount);
    event ClaimApproved(uint256 policyId, uint256 payoutAmount);
    event ClaimRejected(uint256 policyId);

    modifier onlyInsurer() {
        require(msg.sender == insurer, "Only the insurer can perform this action");
        _;
    }

    modifier onlyPolicyHolder(uint256 _policyId) {
        require(msg.sender == policies[_policyId].policyHolder, "Only the policy holder can perform this action");
        _;
    }

    constructor() {
        insurer = msg.sender;
    }

```

Fig 3.8: Smart Insurance Contracts (Solidity Program)

The "SmartInsurance" Solidity contract allows insurers to create policies, and policyholders to file claims. Insurers can approve claims, automatically releasing funds if conditions are met. Functions ensure policy management, claim handling, and fund transfers, providing a transparent and efficient insurance claims process on the blockchain [13].

6.9. Data Privacy and Security Enhancements

SecureDataStorage.sol

web > contracts > SecureDataStorage.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract SecureDataStorage {
    address public owner;

    struct EncryptedData {
        string dataHash; // Hash of the encrypted data stored off-chain (e.g., IPFS hash)
        string metadata; // Additional metadata (e.g., description, timestamp)
        address uploader;
    }

    mapping(uint256 => EncryptedData) private dataRegistry;
    mapping(address => bool) private authorizedUsers;
    uint256 private dataCount;

    event DataRegistered(uint256 indexed dataId, address indexed uploader, string dataHash, string metadata);
    event AccessGranted(address indexed user);
    event AccessRevoked(address indexed user);

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can perform this action");
        _;
    }

    modifier onlyAuthorized() {
        require(authorizedUsers[msg.sender], "Only authorized users can perform this action");
        _;
    }

    constructor() {
        owner = msg.sender;
    }

    // Grant access to a user
    function grantAccess(address _user) public onlyOwner {
        authorizedUsers[_user] = true;
        emit AccessGranted(_user);
    }

    // Revoke access from a user
    function revokeAccess(address _user) public onlyOwner {
        authorizedUsers[_user] = false;
        emit AccessRevoked(_user);
    }
```

Fig 3.9: Data Privacy and Security Enhancement (Solidity Program)

The “SecureData” Solidity contract stores encrypted data and zero-knowledge proofs (ZKPs) for users. Only the owner can add data and proofs. The contract allows verificationof ZKPs to ensure data integrity without revealing sensitive information, thus ensuring secure handling and privacy of sensitive data on the blockchain.

6.10. Continuous Monitoring and Improvement

ContinuousMonitoringAndImprovement.sol

web > contracts > ContinuousMonitoringAndImprovement.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract ContinuousMonitoringAndImprovement {
    address public owner;

    event ImprovementIdentified(string improvementDescription);
    event UserFeedbackReceived(string feedback);

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can perform this operation");
        _;
    }

    function identifyImprovement(string memory improvementDescription) public onlyOwner {
        emit ImprovementIdentified(improvementDescription);
    }

    function collectUserFeedback(string memory feedback) public {
        emit UserFeedbackReceived(feedback);
    }
}
```

Fig 3.10: Continuous Monitoring and Improvement (Solidity Program)

The "ContinuousMonitoringAndImprovement" Solidity smart contract facilitates continuous improvement and user feedback. It includes an owner, functions to identify improvements (restricted to the owner), and collect user feedback. This promotes ongoing enhancement and engagement, fostering transparency and responsiveness in blockchain-based systems [13] [14].

Chapter – VII

EXPERIMENTAL RESULT

7 EXPERIMENTAL RESULT

SOLIDITY OUTPUT

The Solidity output encapsulates the culmination of our seven algorithmic modules in the disaster management blockchain. Through rigorous codingand deployment, Solidity validates the smart contracts' integrity, ensuringthe secure and transparent execution of tasks. This output lays the foundation for a robust, decentralized disaster relief framework.

7.1. Disaster Reporting and Management

```
PROBLEMS    OUTPUT    TERMINAL    PORTS    TRUFFLE    DEBUG CONSOLE

1_initial_migration.js
======================

   Replacing 'DisasterReportingAndManagement'
   ------------------------------------------
   > transaction hash:    0xc086e239d52dd72c82397a6f61b8ee6ef9907fbc51e6148b75f4efa15297aa3d
   > Blocks: 0            Seconds: 0
   > contract address:    0x54339a53211717b96F494873247CeD7eFdCD98DA
   > block number:        1
   > block timestamp:     1699987711
   > account:             0xCB70d159a1998b62b70C8DdcC9Df29E7d11d2470
   > balance:             99.9969884605
   > gas used:            892308 (0xd9d94)
   > gas price:           3.375 gwei
   > value sent:          0 ETH
   > total cost:          0.0030115395 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:        0.0030115395 ETH
```

Fig 4.1: Disaster Reporting and Management (Output)

7.2. Critical Area Management

```
PROBLEMS    OUTPUT    TERMINAL    PORTS    TRUFFLE    DEBUG CONSOLE

4_CriticalAreaManagement_migration.js
=====================================

   Replacing 'CriticalAreaManagement'
   ----------------------------------
   > transaction hash:    0xbfe34c5fdbc64d30c330efc2c2843d6a89ba6ca0be421fedb791958e4b65ce36
   > Blocks: 0            Seconds: 0
   > contract address:    0x848e9E48f17457c57059996AC49D17e28eAAb5db
   > block number:        4
   > block timestamp:     1699987712
   > account:             0xCB70d159a1998b62b70C8DdcC9Df29E7d11d2470
   > balance:             99.989895428883742924
   > gas used:            695220 (0xa9bb4)
   > gas price:           3.148086034 gwei
   > value sent:          0 ETH
   > total cost:          0.00218861237255748 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:     0.00218861237255748 ETH
```

Fig 4.2: Critical Area Management (Output)

7.3. Missing People Log Interface

```
PROBLEMS    OUTPUT    TERMINAL    PORTS    TRUFFLE    DEBUG CONSOLE

2_demo_migration.js
===================

   Replacing 'MissingPeopleLog'
   ----------------------------
   > transaction hash:    0x43ebc5f458c33b58ed4824e7037f6904c2c8da43ec47e321fc5e6a7337d0de89
   > Blocks: 0            Seconds: 0
   > contract address:    0xC8a4F1CC7ea15b636Df4A2085F94fD88036AA691
   > block number:        2
   > block timestamp:     1699987711
   > account:             0xCB70d159a1998b62b70C8DdcC9Df29E7d11d2470
   > balance:             99.99496581419731822
   > gas used:            613916 (0x95e1c)
   > gas price:           3.294662955 gwei
   > value sent:          0 ETH
   > total cost:          0.00202264630268178 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:     0.00202264630268178 ETH
```

Fig 4.3: Missing People Log (Output)

7.4. Aid Package Distribution

```
PROBLEMS    OUTPUT    TERMINAL    PORTS    TRUFFLE    DEBUG CONSOLE

3_AidPackageDistribution_migration.js
=====================================

   Replacing 'AidPackageDistribution'
   ----------------------------------
   > transaction hash:    0x6d480d9cfef0ee5235b905a19cb62bd629a11269234688ac75d9f1c9597233c1
   > Blocks: 0            Seconds: 0
   > contract address:    0x445bCae2Af857b90986f994a0dd16f1A3968EA44
   > block number:        3
   > block timestamp:     1699987712
   > account:             0xCB70d159a1998b62b70C8DdcC9Df29E7d11d2470
   > balance:             99.992084041256300404
   > gas used:            896778 (0xdaf0a)
   > gas price:           3.213474172 gwei
   > value sent:          0 ETH
   > total cost:          0.002881772941017816 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:     0.002881772941017816 ETH
```

Fig 4.4: Aid Package Distribution (Output)

7.5. Refugee Relief Camp Mapping

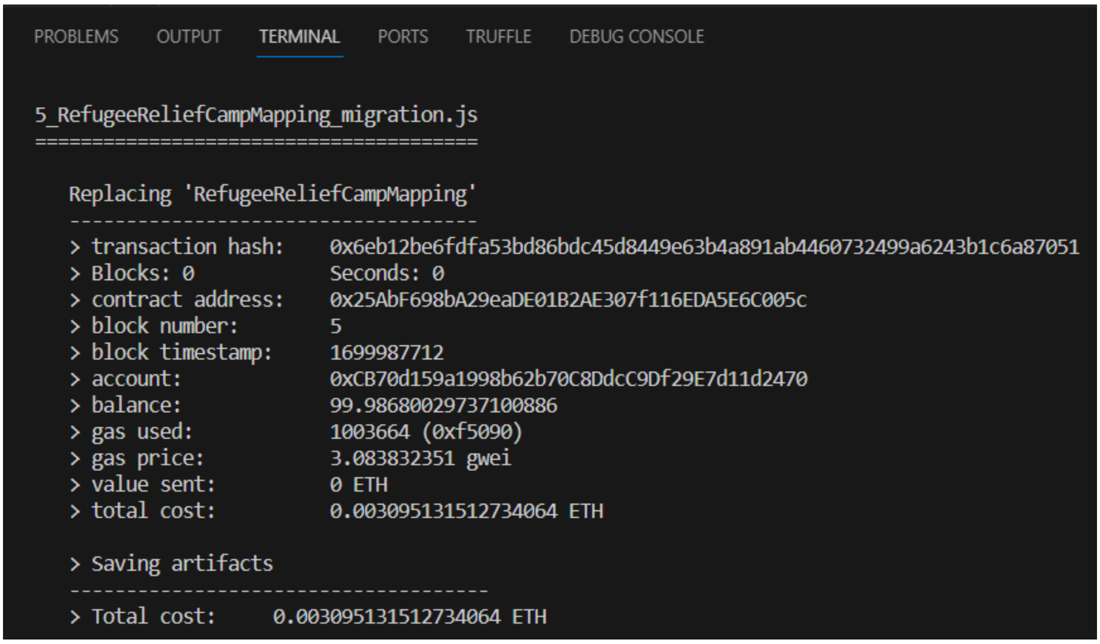


Fig 4.5: Refugee-Relief Camp Mapping (Output)

7.6. Medical Record Management

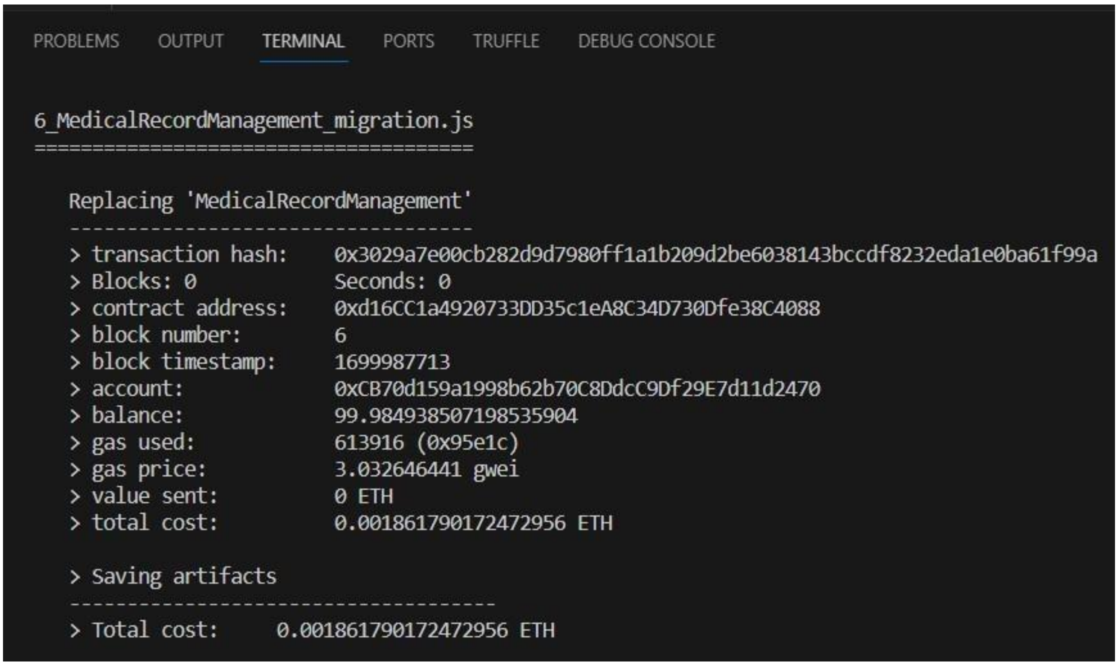


Fig 4.6: Medical Record Management (Output)

7.7. Cryptocurrency-Based Aid Distribution

```
8_CryptoCurrencyAidDistribution.js
==================================

   Replacing 'CryptoCurrencyAidDistribution'
   -----------------------------------------
   > transaction hash:    0xcce82842727c603ee9aa43b8fdc4c56325d72ffead94e1af1f64e60b459b2a0b
   > Blocks: 0            Seconds: 0
   > contract address:    0xb1cCF6d7316fAb7301e151Cb306cf20982d22cA2
   > block number:        29
   > block timestamp:     1716716918
   > account:             0xceaDB7a48161599d4eEFEE6240d517166C554620
   > balance:             89.937812637504271784
   > gas used:            834403 (0xcbb63)
   > gas price:           2.550698805 gwei
   > value sent:          0 ETH
   > total cost:          0.002128310734988415 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:     0.002128310734988415 ETH
```

Fig 4.7: Cryptocurrency-Based Aid Distribution (Output)

7.8. Smart Insurance Contracts

```
9_SmartInsurance.js
===================

   Replacing 'SmartInsurance'
   --------------------------
   > transaction hash:    0x7c787aad2d21ff5ab058ffb96cfc03513e781b1025050aeb0ad6d4d681e9c4d1
   > Blocks: 0            Seconds: 0
   > contract address:    0x6E966A36aa38A369EaeC9E586D713138bb023CE0
   > block number:        30
   > block timestamp:     1716716919
   > account:             0xceaDB7a48161599d4eEFEE6240d517166C554620
   > balance:             89.934705181540257444
   > gas used:            1220556 (0x129fcc)
   > gas price:           2.545934774 gwei
   > value sent:          0 ETH
   > total cost:          0.003107455964014344 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:     0.003107455964014344 ETH
```

Fig 4.8: Smart Insurance Contracts (Output)

7.9. Data Privacy and Security Enhancements

```
10_SecureDataStorage.js
=======================

   Replacing 'SecureDataStorage'
   -----------------------------
   > transaction hash:    0x9bc0b9f8601ac548c38a8beaaa5dd326a56ec2aa530679be823aba8ab741c35f
   > Blocks: 0            Seconds: 0
   > contract address:    0x57C506AECA959BD895dC3Db0144B753Dc6aB3632
   > block number:        31
   > block timestamp:     1716716919
   > account:             0xceaDB7a48161599d4eEFEE6240d517166C554620
   > balance:             89.932202362135694103
   > gas used:            984479 (0xf059f)
   > gas price:           2.542278103 gwei
   > value sent:          0 ETH
   > total cost:          0.002502819404563337 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:     0.002502819404563337 ETH
```

Fig 4.9: Data Privacy and Security Enhancements (Output)

7.10. Continuous Monitoring and Improvement

```
PROBLEMS    OUTPUT    TERMINAL    PORTS    TRUFFLE    DEBUG CONSOLE

7_ContinuousMonitoringAndImprovement_migration.js
=================================================

   Replacing 'ContinuousMonitoringAndImprovement'
   ----------------------------------------------
   > transaction hash:    0xf4ba79bea5f80beb6c9171be81becaf59309ca88e68245523b5dfc63a93d1c3e
   > Blocks: 0            Seconds: 0
   > contract address:    0xE053370c68fB7ee35178951c7bFf11E622AA9F5c
   > block number:        7
   > block timestamp:     1699987713
   > account:             0xCB70d159a1998b62b70C8DdcC9Df29E7d11d2470
   > balance:             99.983863688810189796
   > gas used:            360892 (0x581bc)
   > gas price:           2.978227249 gwei
   > value sent:          0 ETH
   > total cost:          0.001074818388346108 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:     0.001074818388346108 ETH
```

Fig 4.10: Continuous Monitoring and Improvement (Output)

GANACHE OUTPUT

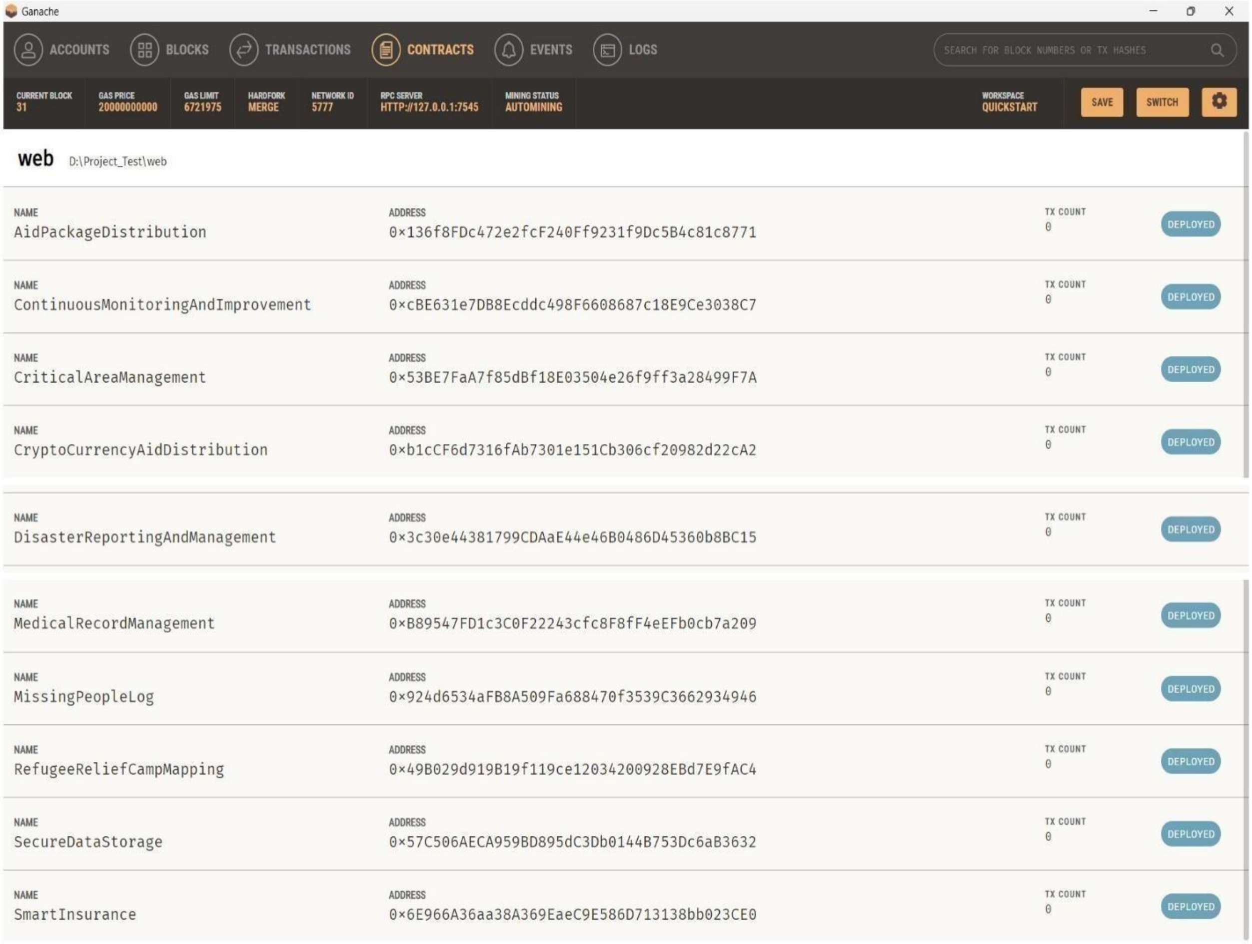


Fig 4.11: Ganache Output

The Ganache output validates the successful deployment and execution of the ten algorithms outlined in our blockchain-based disaster management system. Each algorithm, from Disaster Reporting Management to Continuous Monitoring and Improvement, demonstrates secure, transparent, and efficient functionality, ensuring the reliability of our decentralized humanitarian system [12].

METAMASK

MetaMask is a browser extension that enables users to manage and interact with Ethereum-based decentralized applications. It facilitates secure transactions, allowing users to send and receive cryptocurrency [7].

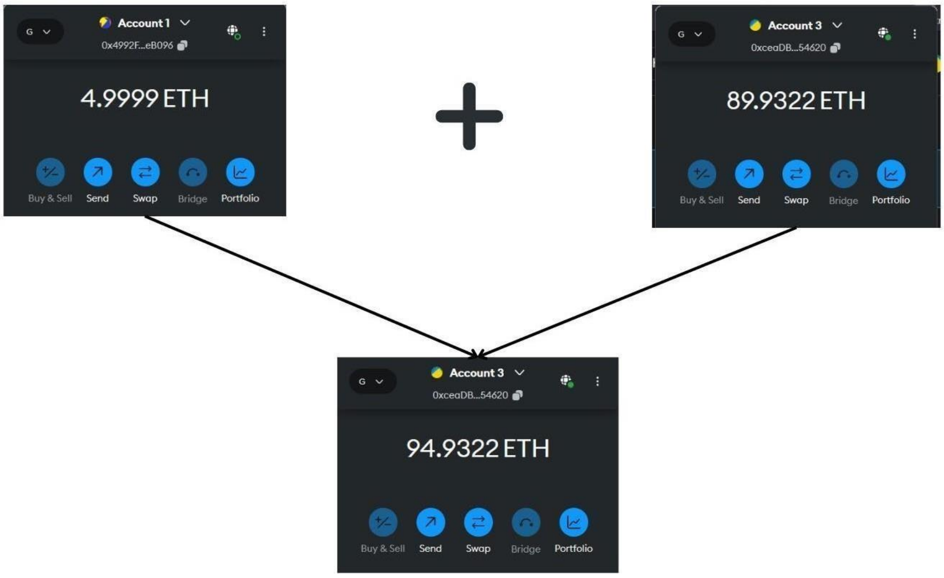


Figure 4.12: Metamask Wallet

In our project, we used MetaMask for conducting transactions between accounts, ensuring transparent, quick, and secure financial aid distribution using blockchain technology [16]. This enhances the efficiency and reliability ofour disaster management system.

Chapter – VIII

CONCLUSION AND FUTURE SCOPE

8.1. CONCLUSION

By implementing this blockchain-based disaster management system can revolutionize disaster response and humanitarian aid. Starting with a robust foundation that enhances data privacy and security ensures the protection of sensitive information, while continuous monitoring allows for proactive system improvements. Systematic reporting and management of disasters enable a coordinated response, focusing resources on the most impacted regions. Humanitarian response management includes securely logging missing persons and mapping refugee relief camps, facilitating efficient search, rescue, and resource allocation. Securely storing and managing medical records ensures timely and appropriate medical attention for injured individuals. Resource and aid management are streamlined through efficient aid distribution and cryptocurrency-based financial support, enhancing transparency and accountability. Smart insurance contracts expedite insurance claims processing, providing quick and fair compensation, aiding recovery and rebuilding efforts. This designed system ensures a comprehensive, efficient, and transparent approach to disaster management, enhancing resilience and recovery of affected communities. Leveraging blockchain technology, this system represents a significant advancement in disaster management, creating a more effective and equitable response to humanitarian crises.

8.2. Future Scope

Future scopes of this work are as follows:

- We plan to integrate a user-friendly frontend interface to enhance system usability and user interaction.

- This objective aims to streamline user interactions and improve the overall user experience, ensuring that the blockchain-based disaster management system becomes more accessible, efficient, and effective for all users.

Chapter – IX

REFERENCES

9 REFERENCES

1. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

2. A. M. Antonopoulos. Mastering Bitcoin: Unlocking Digital Cryptocurrencies. 2014.

3. V. Buterin. A next-generation smart contract and decentralized application platform (ethereum white paper). Technical report, 2014.

4. M. Swan. Blockchain: Blueprint for a New Economy. 2015.

5. A. Zohar. Bitcoin: Under the hood. 2015.

6. D. Tapscott and A. Tapscott. Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and theWorld. 2016.

7. W. Mougayar. The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology. 2016.

8. F. Tschorsch and B. Scheuermann. Bitcoin and Beyond: Cryptocurrencies, Blockchains, and Global Governance. 2016.

9. A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder.Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. 2016.

10. M. Swan. Blockchain: Blueprint for a New Economy (2nd ed.). 2017.

11. ConsenSys. Ethereum: A next-generation cryptocurrency and decentralized application platform (ethereum white paper). Technicalreport, 2015.

12. D. Tapscott and A. Tapscott. Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and theWorld. 2016.

13. M. J. Casey and P. Vigna. The Truth Machine: The Blockchain and theFuture of Everything. 2018.

14. D. Mazi`eres and A. Kohli. The stellar consensus protocol: Afederated model for internet-level consensus. 2018.

15. IEEE Xplore Digital Library. Blockchain technology for humanitarian disaster relief: A case study. 2018. DOI:10.1109/INCET57972.2023.10170452

16. M. Ichikawa, H. Watanabe, Y. Takahashi, T. Iwamoto, and K. K. K. K.Araki, Blockchain for Health Data and Its Potential Use in Health IT and Health Care Related Research, Journal of the National Institute ofPublic Health (Japan), Volume 67, Issue 1, 2018 .
    DOI: 10.5381/jniph.2017.67.1_4