

APPLICATION OF BLOCK CHAIN TECHNOLOGY IN DISASTER MANAGEMENT

A PROJECT REPORT

OF PROJECT-II (PROJ-IT781)

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**

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. In response to the pressing need for improved disaster management systems, our project seeks to leverage blockchain technology for decentralized and transparent disaster relief efforts. This progress report outlines the advancements made towards the development and deployment of our blockchain-based solution using Hyperledger Fabric. The 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, aid distribution, 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 · Hyperledger
Fabric

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	5-7
-----------------------------	------------

4 Software Design	8-9
--------------------------	------------

5 Software and Hardware Requirements	10-11
---	--------------

6 Code Templates	12-22
-------------------------	--------------

7 Experimental Results	23-29
-------------------------------	--------------

8 Conclusion	30-31
---------------------	--------------

9 References	32-33
---------------------	--------------

List of Figures

1	Algorithm Flow Diagram	9
2	Hyperledger pre-requisite Installation	13
3	Hyperledger Installation	13
4	Test Network Setup	14
5	Test Channel Setup.....	14
6	assetTransfer	15
7	app.js File.....	15
8	Missing People Log (Solidity Program)	16
9	Disaster Reporting and Management(Solidity Program)	17
10	Aid Package Distribution (Solidity Program)	18
11	Critical Area Management (Solidity Program)	19
12	Refugee-Relief Camp Mapping (Solidity Program).....	20
13	Medical Record Management (Solidity Program).....	21
14	Continuous Monitoring and Improvement (Solidity Program)	22
15	Hyperledger Fabric Sample Output	24
16	Missing People Log (Output)	25
17	Disaster Reporting and Management (Output)	25
18	Aid Package Distribution (Output).....	26
19	Critical Area Management (Output).....	26
20	Refugee-Relief Camp Mapping (Output)... ..	27
21	Medical Record Management (Output)... ..	27
22	Continuous Monitoring and Improvement.....	28
23	Ganache Output.....	29

Chapter – I

INTRODUCTION

1 INTRODUCTION

1. Introduction of the project:

The integration of blockchain technology into disaster management signifies a profound shift in crisis response strategies. Traditional systems often struggle with inefficiencies, lack of transparency, and delayed responses. Blockchain, renowned for its decentralized and secure nature, offers a novel solution to these challenges.

2. Objective of Industrial Project:

In response to escalating natural disasters and humanitarian crises, our project seeks to leverage blockchain technology for decentralized and transparent disaster relief efforts. The immutability of blockchain ensures that critical information is securely recorded, fostering trust and accountability among stakeholders. This feature is particularly crucial in disaster scenarios where swift and accurate information dissemination can be a matter of life and death.

3. Summary of a Report:

As part of our industrial project, we are now running algorithms using Hyperledger Fabric. The decentralized nature of blockchain technology reduces the risk of data loss or manipulation by eliminating the reliance on a central repository. By harnessing Hyperledger Fabric, we aim to revolutionize how we prepare for, respond to, and recover from crises. This progress report outlines our advancements in running a sample test network using Hyperledger Fabric, laying the groundwork for decentralized, secure, and transparent disaster relief efforts.

Chapter – II

LITERATURE SURVEY

2 Literature Survey

In recent years, the transformative potential of blockchain technology in disaster response has garnered attention, as evidenced by the work of Tapscott and Tapscott (2016) [6]. This literature review explores the decentralized nature of blockchain and its potential to enhance security and efficiency in disaster relief operations. The authors emphasize the paradigm shift that blockchain introduces in disaster management, highlighting the system's ability to operate without a central authority. This approach aligns with the overarching goal of improving the resilience and effectiveness of disaster response through innovative technological solutions.

M. Swan (2017) [11] critically examine the challenges and opportunities associated with blockchain in disaster management. Their work lays the foundation for understanding potential hurdles, offering valuable insights for researchers and practitioners. Strategies for effectively harnessing blockchain for humanitarian purposes are outlined, highlighting the importance of a thoughtful and strategic approach to maximize the benefits of this technology in the context of disaster response.

Casey (2018) [12] delves into the realm of smart contracts and their pivotal role in automating and streamlining coordination efforts during disaster response. The focus on transparent and self-executing agreements is crucial for the efficient distribution of aid packages and the management of critical areas. Casey's work underscores the practical application of blockchain through smart contracts, offering a promising avenue for improving the coordination and deployment of resources in disaster-stricken areas.

X Zhang and co. (2018) [15] contribute to the literature by examining blockchain's role in community engagement during disaster response. Their work navigates ethical considerations and adopts a community-centric approach. This perspective aligns with the broader goal of creating inclusive and ethically sound disaster response frameworks.

The concept of real-time data sharing in disaster scenarios takes center stage in the work of Mazi and Kohli (2018) [14]. Their exploration emphasizes the immediacy and transparency required in disaster management. The alignment with the core objectives of disaster response projects underscores the significance of timely information exchange. Blockchain's potential to facilitate real-time data sharing emerges as a key consideration in addressing the dynamic nature of disasters, enabling quicker decision-making and more effective response strategies. The synergies between blockchain and the Internet of Things (IoT) in disaster scenarios are explored by Ichiwaka and co. (2018) [16]. Their work provides insights into comprehensive solutions for real-time monitoring and response, informing considerations for future technological integration. By examining the intersection of blockchain and IoT, the authors contribute to a more holistic understanding of how emerging technologies can collaborate to improve the effectiveness of disaster management strategies.

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. [1]

3.2. Modules and Functionalities:

Step 1: Missing People Log

- **Input:** Name, last known location, description.
- **Process:** Smart contracts in Solidity securely log and update missing people's information.
- **Output:** Secure, transparent, and up-to-date list of missing people [1].

Step 2: 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 [3].

Step 3: 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. [5]

Step 4: 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 [7].

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 [9].

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 [11].

Step 7: Continuous Monitoring and Improvement

- **Input:** Real-time system data, user feedback, emerging technologies.
- **Process:** Continuous monitoring for irregularities, real-time data updates, user feedback analysis.
- **Output:** Well monitored, up-to-date, and ever improving blockchain based humanitarian system [13].

3.3. Future Enhancement:

- **Plan:** Run and deploy smart contracts in Hyperledger Fabric.
- **Objective:** Enhance the system further with emerging technologies [15]

Chapter – IV

SOFTWARE DESIGN

WORK FLOW DIAGRAM

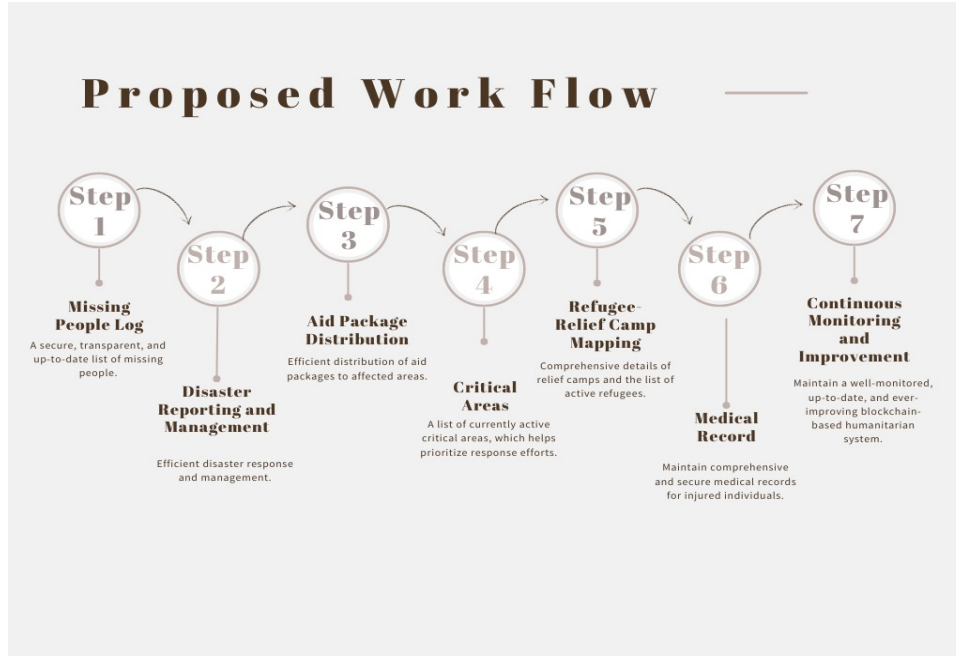


Fig 1: Work Flow Diagram

The blockchain-based disaster management system is intricately designed with seven modules, each implemented as a Solidity smart contract. The "Missing People Log" module ensures transparency by structuring details of missing persons, allowing their addition through emitted events. The "Disaster Reporting and Management" module tracks and resolves disaster reports, capturing crucial details. "Aid Package Distribution" manages aid package distribution, recording sender, recipient, and delivery status. The "Critical Area Management" module handles critical zones based on severity. "Refugee Relief Camp Mapping" efficiently maps refugees to camps, ensuring transparent organization. The "Medical Record Management" module securely stores medical data. The "Continuous Monitoring and Improvement" module, restricted to an owner, identifies improvements and collects user feedback, fostering ongoing system enhancement. Together, these modules form a comprehensive and resilient blockchain solution for disaster management, addressing aspects from missing persons to continuous improvement.

Chapter – V

SOFTWARE AND HARDWARE REQUIREMENTS

5 SOFTWARE AND HARDWARE REQUIREMENT

5.1. Software Requirements:

- **Blockchain Development Platform:**
 - *Ethereum* (Solidity for smart contracts) [1]
 - *Truffle* for *Ethereum* smart contract development [3]
 - *Ganache* for local blockchain deployment and testing [5]
 - *Visual Studio Code* for *Ethereum* smart contract coding [7]
 - *Hyperledger Fabric* for deploying smart contracts in *.js* file[14]
 -
- **Integrated Development Environment (IDE):**
 - *Visual Studio Code* for *Ethereum* smart contract development [9]
 - *Ubuntu (Virtual Machine)* for *Hyperledger* smart contract development[11]
- **Testing:**
 - *Truffle* for *Ethereum* smart contract testing [11]

5.2. Hardware Requirements:

- **Storage:**
 - Adequate storage for local development environment [13]
- **Memory (RAM):**
 - Sufficient RAM for smooth development and testing [15]
- **Network:**
 - Standard internet connection for interacting with the Ethereum blockchain [2]

Chapter – VI

CODE TEMPLATES

6 CODE TEMPLATES

1. Hyperledger Pre-Requisite Setup:

Installed necessary dependencies including Curl, NodeJs, Git, Python, Docker, Docker-Compose, Go, and npm.

```
souvik@souvik-VirtualBox:~$ curl --version
curl 8.1.2 (x86_64-pc-linux-gnu) libcurl/8.1.2 OpenSSL/1.1.1f zlib/1.2.11 brotli/1.0.7 zstd/1.4.4 libidn2/2.2.0 libpsl/0.21.0 (+libidn2/2.2.0) libssh/0.9.3/openssl/zlib nghttp2/1.40.0 librtmp/2.3 libgsasl/1.8.1
Release-Date: 2023-05-30
Protocols: dict file ftp ftps gopher gophers http https imap imaps ldap ldaps mqtt pop3 pop3s rtmp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: alt-svc AsynchDNS brotli gssapi gssapi HSTS HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL threadsafe TLS-SRP UnixSockets zstd
souvik@souvik-VirtualBox:~$ nodejs --version
v12.22.9
souvik@souvik-VirtualBox:~$ git --version
git version 2.34.1
souvik@souvik-VirtualBox:~$ python3 --version
Python 3.10.12
souvik@souvik-VirtualBox:~$ docker --version
Docker version 24.0.5, build ced0996
souvik@souvik-VirtualBox:~$ docker-compose --version
Docker Compose version v2.20.3
souvik@souvik-VirtualBox:~$ go version
go version go1.18.1 linux/amd64
souvik@souvik-VirtualBox:~$ npm --version
8.5.1
souvik@souvik-VirtualBox:~$
```

Fig 2: Hyperledger pre-requisite Installation

2. Hyperledger Installation:

Successfully installed Hyperledger Fabric on the development environment.

```
====> List out hyperledger docker images
hyperledger/fabric-tools    2.5      de9e326e137d    5 weeks ago    544MB
hyperledger/fabric-tools    2.5.6    de9e326e137d    5 weeks ago    544MB
hyperledger/fabric-tools    latest   de9e326e137d    5 weeks ago    544MB
hyperledger/fabric-peer     2.5      af191f403625    5 weeks ago    140MB
hyperledger/fabric-peer     2.5.6    af191f403625    5 weeks ago    140MB
hyperledger/fabric-peer     latest   af191f403625    5 weeks ago    140MB
hyperledger/fabric-orderer  2.5      0ae165f25b4c    5 weeks ago    110MB
hyperledger/fabric-orderer  2.5.6    0ae165f25b4c    5 weeks ago    110MB
hyperledger/fabric-orderer  latest   0ae165f25b4c    5 weeks ago    110MB
hyperledger/fabric-ccenv    2.5      58b926fe2ef5    5 weeks ago    627MB
hyperledger/fabric-ccenv    2.5.6    58b926fe2ef5    5 weeks ago    627MB
hyperledger/fabric-ccenv    latest   58b926fe2ef5    5 weeks ago    627MB
hyperledger/fabric-baseos   2.5      6b2489cc7d4e    5 weeks ago    126MB
hyperledger/fabric-baseos   2.5.6    6b2489cc7d4e    5 weeks ago    126MB
hyperledger/fabric-baseos   latest   6b2489cc7d4e    5 weeks ago    126MB
hyperledger/fabric-ca       1.5      760a0473a384    5 weeks ago    205MB
hyperledger/fabric-ca       1.5.9    760a0473a384    5 weeks ago    205MB
hyperledger/fabric-ca       latest   760a0473a384    5 weeks ago    205MB
souvik@souvik-VirtualBox:~$
```

Fig 3: Hyperledger Installation

3. Test Network Setup:

- Configured a test network using the provided fabric-samples.
- Utilized the network.sh script to set up a peer channel and create a new channel named testchannel.

```
'"1"' '}' '}', ' "mod_policy":' '""', ' "policies":' ' {}', '
icies":' ' {}', ' "values":' ' {}', ' "version":' ' "0"' '}' '}}
+ configtxlator proto_encode --input config_update_in_envel
2024-03-28 07:33:48.304 UTC 0001 INFO [channelCmd] InitCmdF
2024-03-28 07:33:48.349 UTC 0002 INFO [channelCmd] update -
Anchor peer set for org 'Org2MSP' on channel 'testchannel'
Channel 'testchannel' joined
souvik@souvik-VirtualBox:~/fabric-samples/test-network$
```

Fig 4: Test Network Setup

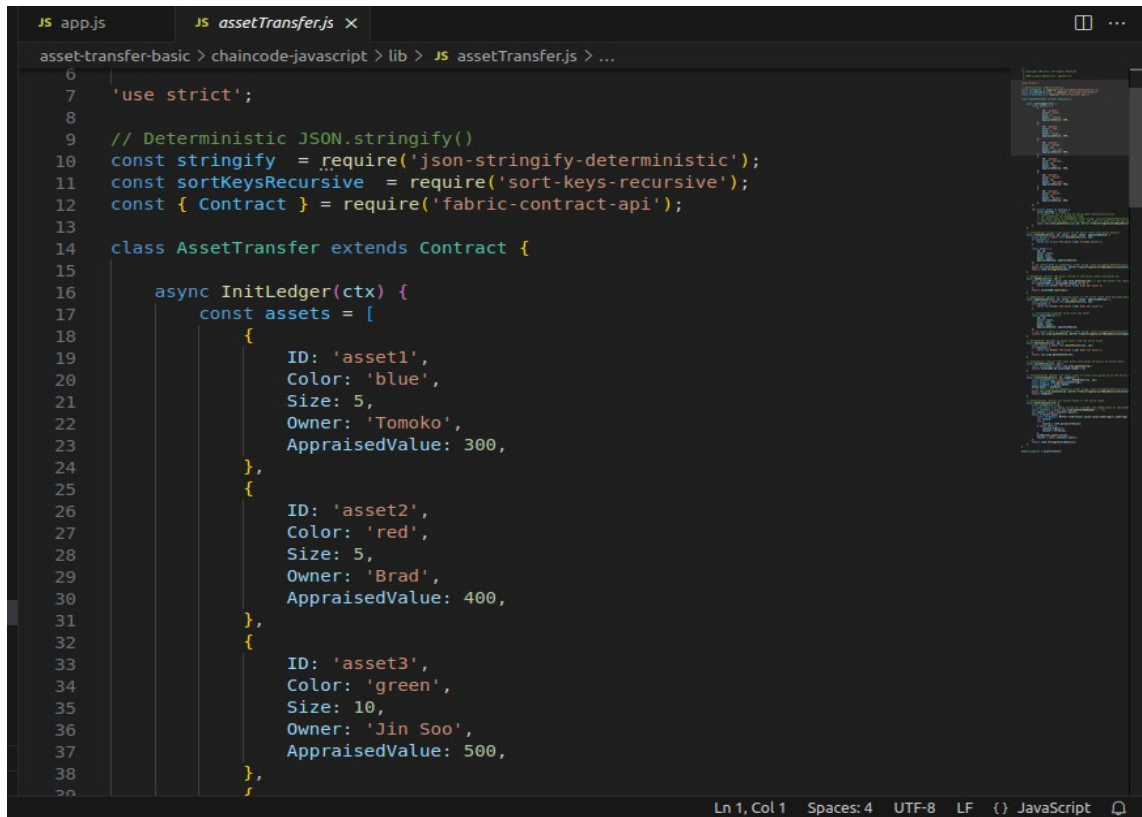
- Verified channel creation and network setup by examining Docker containers and channel listings.
- Ran CouchDB as a state database for the test network and created a new channel named testchannel1.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS NAMES
37492f26d520	hyperledger/fabric-tools:latest	"/bin/bash"	11 seconds ago	Up Less than a second	cli
6989546bfdd8	hyperledger/fabric-peer:latest	"peer node start"	11 seconds ago	Up 3 seconds	0.0.0.0:9051->9051/tcp, ::9051->9051/tcp, 7051/tcp, 0.0.0.0:9445->9445/tcp, ::9445->9445/tcp
ffcf8d1edd3	hyperledger/fabric-peer:latest	"peer node start"	11 seconds ago	Up 2 seconds	0.0.0.0:7051->7051/tcp, ::7051->7051/tcp, 0.0.0.0:9444->9444/tcp, ::9444->9444/tcp
128d269ade2a	hyperledger/fabric-orderer:latest	"orderer"	13 seconds ago	Up 6 seconds	0.0.0.0:7050->7050/tcp, ::7050->7050/tcp, 0.0.0.0:7053->7053/tcp, ::7053->7053/tcp, 0.0.0.0:9443->9443/tcp, ::9443->9443/tcp
7daf3b125d3a	couchdb:3.3.2	"tini -- /docker-ent..."	13 seconds ago	Up 8 seconds	4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp, ::7984->5984/tcp
91935577d26d	couchdb:3.3.2	"tini -- /docker-ent..."	13 seconds ago	Up 7 seconds	4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp, ::5984->5984/tcp

Fig 5: Test Channel Setup

4. Writing Sample Smart Contract:

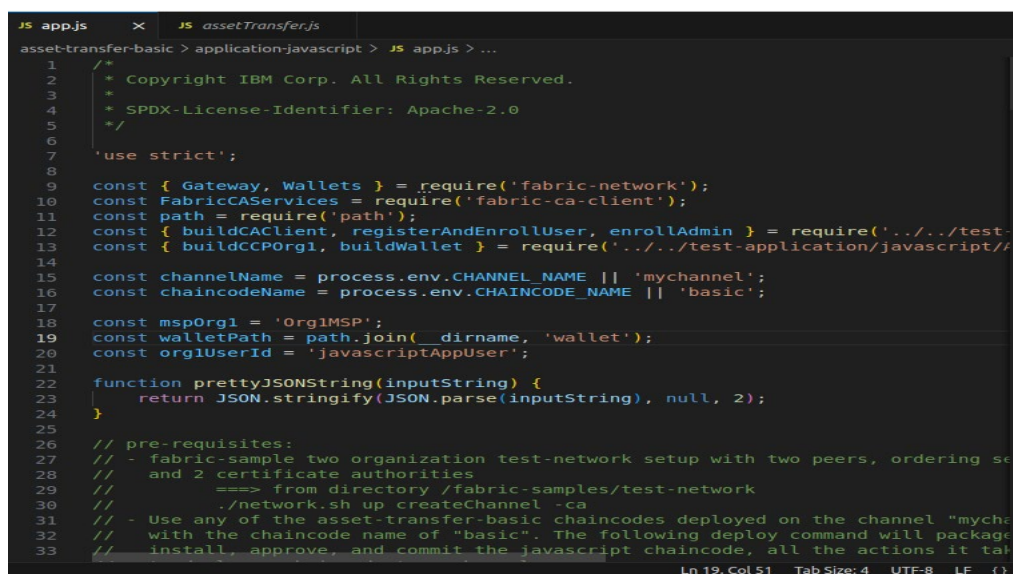
Developed a sample smart contract named "AssetTransfer" using javascript.



```
6
7 'use strict';
8
9 // Deterministic JSON.stringify()
10 const stringify = require('json-stringify-deterministic');
11 const sortKeysRecursive = require('sort-keys-recursive');
12 const { Contract } = require('fabric-contract-api');
13
14 class AssetTransfer extends Contract {
15
16   async InitLedger(ctx) {
17     const assets = [
18       {
19         ID: 'asset1',
20         Color: 'blue',
21         Size: 5,
22         Owner: 'Tomoko',
23         AppraisedValue: 300,
24       },
25       {
26         ID: 'asset2',
27         Color: 'red',
28         Size: 5,
29         Owner: 'Brad',
30         AppraisedValue: 400,
31       },
32       {
33         ID: 'asset3',
34         Color: 'green',
35         Size: 10,
36         Owner: 'Jin Soo',
37         AppraisedValue: 500,
38       },
39     ],
40   },
41 }
```

Fig 6: assetTransfer

Then we are deploying the smart contract using app.js in hyperledger fabric



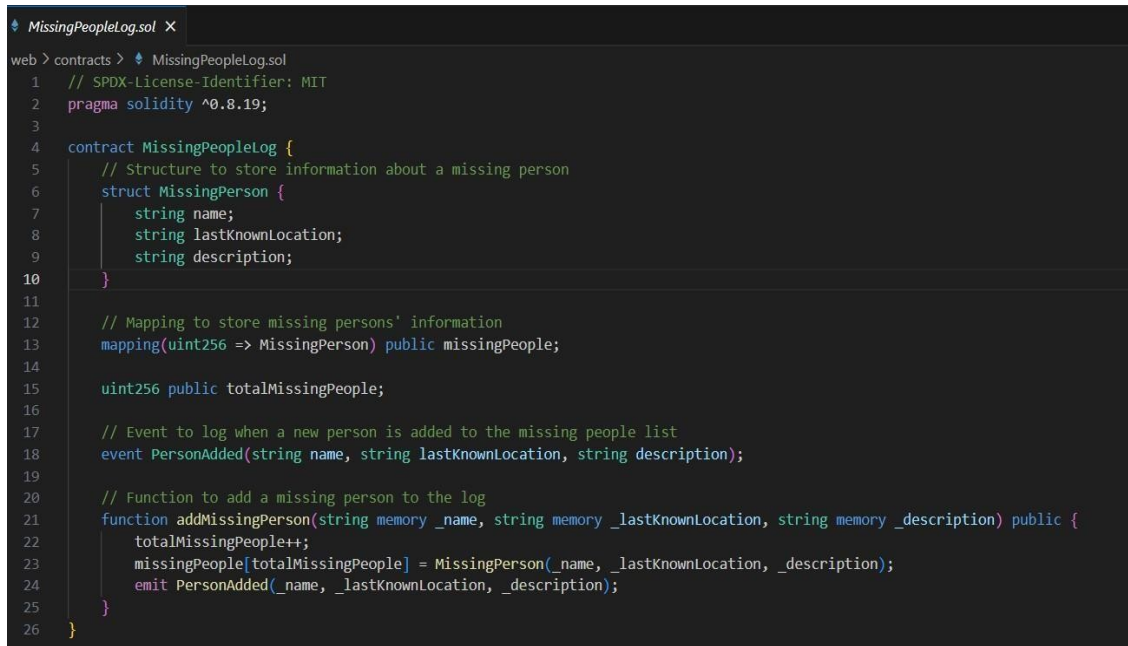
```
1 /*
2  * Copyright IBM Corp. All Rights Reserved.
3  *
4  * SPDX-License-Identifier: Apache-2.0
5  */
6
7 'use strict';
8
9 const { Gateway, Wallets } = require('fabric-network');
10 const FabricCAServices = require('fabric-ca-client');
11 const path = require('path');
12 const { buildCAClient, registerAndEnrollUser, enrollAdmin } = require('../test-');
13 const { buildCCOrg1, buildWallet } = require('../test-application/javascript/');
14
15 const channelName = process.env.CHANNEL_NAME || 'mychannel';
16 const chainCodeName = process.env.CHAINCODE_NAME || 'basic';
17
18 const mspOrg1 = 'Org1MSP';
19 const walletPath = path.join(__dirname, 'wallet');
20 const org1UserId = 'javascriptAppUser';
21
22 function prettyJSONString(inputString) {
23   return JSON.stringify(JSON.parse(inputString), null, 2);
24 }
25
26 // pre-requisites:
27 // - fabric-sample two organization test-network setup with two peers, ordering se
28 //   and 2 certificate authorities
29 //   ==> from directory /fabric-samples/test-network
30 //   ./network.sh up createChannel -ca
31 // - Use any of the asset-transfer-basic chaincodes deployed on the channel "mycha
32 //   with the chaincode name of "basic". The following deploy command will package
33 //   install, approve, and commit the javascript chaincode, all the actions it tak
```

Fig 7: app.js

PREVIOUS WORK

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 Missing People Log



```
MissingPeopleLog.sol X
web > contracts > MissingPeopleLog.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.19;
3
4 contract MissingPeopleLog {
5     // Structure to store information about a missing person
6     struct MissingPerson {
7         string name;
8         string lastKnownLocation;
9         string description;
10    }
11
12    // Mapping to store missing persons' information
13    mapping(uint256 => MissingPerson) public missingPeople;
14
15    uint256 public totalMissingPeople;
16
17    // Event to log when a new person is added to the missing people list
18    event PersonAdded(string name, string lastKnownLocation, string description);
19
20    // Function to add a missing person to the log
21    function addMissingPerson(string memory _name, string memory _lastKnownLocation, string memory _description) public {
22        totalMissingPeople++;
23        missingPeople[totalMissingPeople] = MissingPerson(_name, _lastKnownLocation, _description);
24        emit PersonAdded(_name, _lastKnownLocation, _description);
25    }
26 }
```

Fig 8: 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 [1].

6.2. Disaster Reporting and Management

```
DisasterReportingAndManagement.sol X
web > contracts > DisasterReportingAndManagement.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.19;
3
4 contract DisasterReportingAndManagement {
5     // Structure to store information about a disaster report
6     struct DisasterReport {
7         string name;
8         string location;
9         uint256 timestamp;
10        string description;
11        bool isResolved;
12        address[] stakeholders;
13    }
14
15    DisasterReport[] public disasterReports;
16    uint256 public totalReports;
17
18    // Event to log when a new disaster report is added
19    event DisasterReportAdded(string name, string location, uint256 timestamp, string description);
20
21    // Event to log when a disaster report is marked as resolved
22    event DisasterReportResolved(uint256 reportId);
23
24    // Function to add a new disaster report
25    function addDisasterReport(
26        string memory _name,
27        string memory _location,
28        uint256 _timestamp,
29        string memory _description,
30        address[] memory _stakeholders
31    ) public {
32        totalReports++;
33        disasterReports.push(
34            DisasterReport({
35                name: _name,
36                location: _location,
37                timestamp: _timestamp,
38                description: _description,
39                isResolved: false,
40                stakeholders: _stakeholders
41            })
42        );
43        emit DisasterReportAdded(_name, _location, _timestamp, _description);
44    }
45
46    // Function to mark a disaster report as resolved
47    function markReportAsResolved(uint256 _reportId) public {
48        require(_reportId > 0 && _reportId <= totalReports, "Invalid report ID");
49        disasterReports[_reportId - 1].isResolved = true;
50        emit DisasterReportResolved(_reportId);
51    }
52 }
53
54 }
```

Fig 9: Disaster Reporting and Management (Solidity Program)

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

6.3. Aid Package Distribution

```
AidPackageDistribution.sol
web > contracts > AidPackageDistribution.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.19;
3
4 contract AidPackageDistribution {
5     // Structure to store information about an aid package
6     struct AidPackage {
7         address sender;
8         string recipient;
9         string packageDetails;
10        string sourceLocation;
11        string destinationLocation;
12        bool delivered;
13    }
14
15    // Mapping to store aid packages
16    mapping(uint256 => AidPackage) public aidPackages;
17
18    uint256 public totalAidPackages;
19
20    // Event to log when a new aid package is created
21    event AidPackageCreated(
22        address sender,
23        string recipient,
24        string packageDetails,
25        string sourceLocation,
26        string destinationLocation
27    );
28
29    // Event to log when an aid package is marked as delivered
30    event AidPackageDelivered(uint256 packageId);
31
32    // Function to create a new aid package
33    function createAidPackage(
34        string memory _recipient,
35        string memory _packageDetails,
36        string memory _sourceLocation,
37        string memory _destinationLocation
38    ) public {
39        totalAidPackages++;
40        aidPackages[totalAidPackages] = AidPackage(
41            msg.sender,
42            _recipient,
43            _packageDetails,
44            _sourceLocation,
45            _destinationLocation,
46            false
47        );
48
49        emit AidPackageCreated(msg.sender, _recipient, _packageDetails, _sourceLocation, _destinationLocation);
50    }
51
52    // Function to mark an aid package as delivered
53    function markAidPackageDelivered(uint256 _packageId) public {
54        require(_packageId > 0 && _packageId <= totalAidPackages, "Invalid package ID");
55        require(aidPackages[_packageId].sender == msg.sender, "Only the sender can mark as delivered");
56
57        aidPackages[_packageId].delivered = true;
58
59        emit AidPackageDelivered(_packageId);
60    }
61 }
```

Fig 10: 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 [5].

6.4. Critical Area Management

```
web > contracts > CriticalAreaManagement.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.19;
3
4 contract CriticalAreaManagement {
5     // Structure to store information about a critical area
6     struct CriticalArea {
7         string name;
8         string description;
9         uint8 severity; // Severity can be on a scale of 1 to 10
10    }
11
12    // Mapping to store critical areas
13    mapping(uint256 => CriticalArea) public criticalAreas;
14    uint256 public totalCriticalAreas;
15
16    // Event to log when a new critical area is added
17    event CriticalAreaAdded(string name, string description, uint8 severity);
18
19    // Event to log when a critical area is removed
20    event CriticalAreaRemoved(uint256 areaId);
21
22    // Function to add a new critical area
23    function addCriticalArea(string memory _name, string memory _description, uint8 _severity) public {
24        totalCriticalAreas++;
25        criticalAreas[totalCriticalAreas] = CriticalArea(_name, _description, _severity);
26
27        emit CriticalAreaAdded(_name, _description, _severity);
28    }
29
30    // Function to remove a critical area
31    function removeCriticalArea(uint256 _areaId) public {
32        require(_areaId > 0 && _areaId <= totalCriticalAreas, "Invalid area ID");
33
34        delete criticalAreas[_areaId];
35
36        emit CriticalAreaRemoved(_areaId);
37    }
38 }
```

Fig 11: Critical Area Management (Solidity Program)

This Solidity smart contract, "CriticalAreaManagement," handles critical areas with details 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].

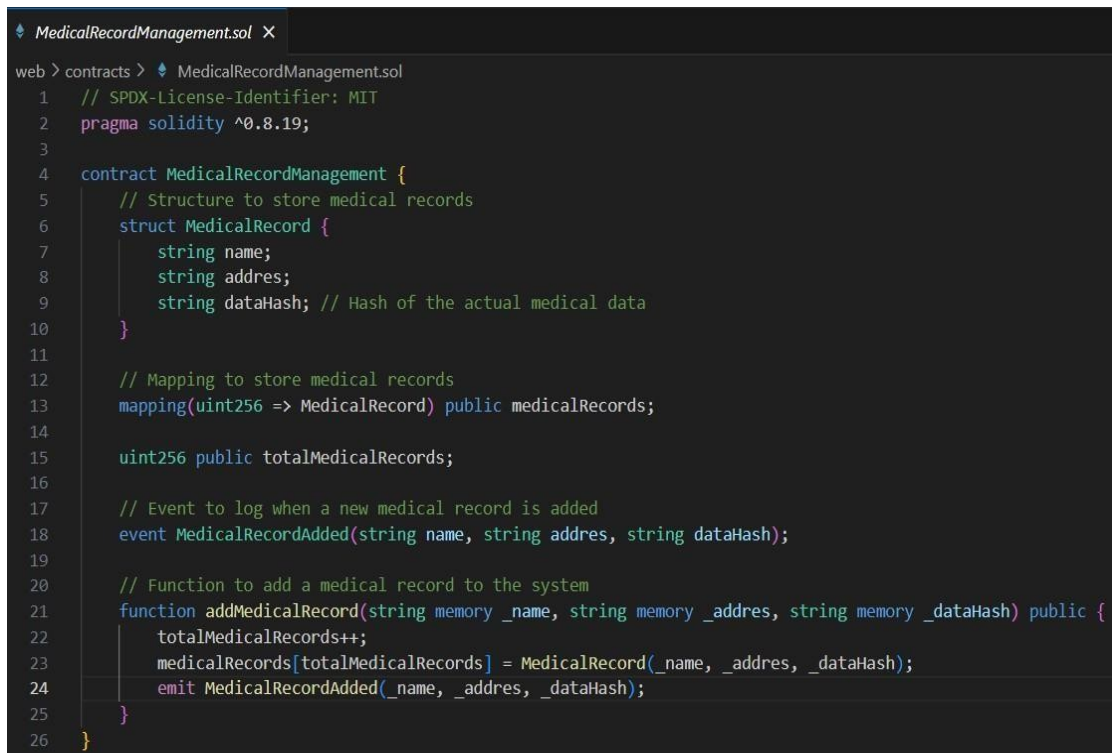
6.5. Refugee Relief Camp Mapping

```
RefugeeReliefCampMapping.sol X
web > contracts > RefugeeReliefCampMapping.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.19;
3
4 contract RefugeeReliefCampMapping {
5     // Structure to store information about a refugee
6     struct Refugee {
7         string name;
8         string location;
9         uint256 campId;
10    }
11
12    // Structure to store information about a relief camp
13    struct ReliefCamp {
14        string name;
15        string location;
16        uint256 size; // Capacity of the camp
17        uint256 currentOccupancy; // Number of refugees in the camp
18    }
19
20    Refugee[] public refugees;
21    ReliefCamp[] public reliefCamps;
22    uint256 public totalRefugees;
23    uint256 public totalReliefCamps;
24
25    // Event to log when a new refugee is added
26    event RefugeeAdded(string name, string location, uint256 campId);
27
28    // Event to log when a new relief camp is added
29    event ReliefCampAdded(string name, string location, uint256 size);
30
31    // Function to add a new refugee and assign them to a camp
32    function addRefugee(string memory _name, string memory _location, uint256 _campId) public {
33        require(_campId > 0 && _campId <= totalReliefCamps, "Invalid camp ID");
34        totalRefugees++;
35        refugees.push(Refugee(_name, _location, _campId));
36
37        emit RefugeeAdded(_name, _location, _campId);
38    }
39
40    // Function to add a new relief camp
41    function addReliefCamp(string memory _name, string memory _location, uint256 _size) public {
42        totalReliefCamps++;
43        reliefCamps.push(ReliefCamp(_name, _location, _size, 0));
44
45        emit ReliefCampAdded(_name, _location, _size);
46    }
47
48    // Function to allocate a refugee to a camp
49    function allocateRefugeeToCamp(uint256 _refugeeId, uint256 _campId) public {
50        require(_refugeeId > 0 && _refugeeId <= totalRefugees, "Invalid refugee ID");
51        require(_campId > 0 && _campId <= totalReliefCamps, "Invalid camp ID");
52
53        refugees[_refugeeId - 1].campId = _campId;
54        reliefCamps[_campId - 1].currentOccupancy++;
55
56        emit RefugeeAdded(refugees[_refugeeId - 1].name, refugees[_refugeeId - 1].location, _campId);
57    }
58 }
```

Fig 12: 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

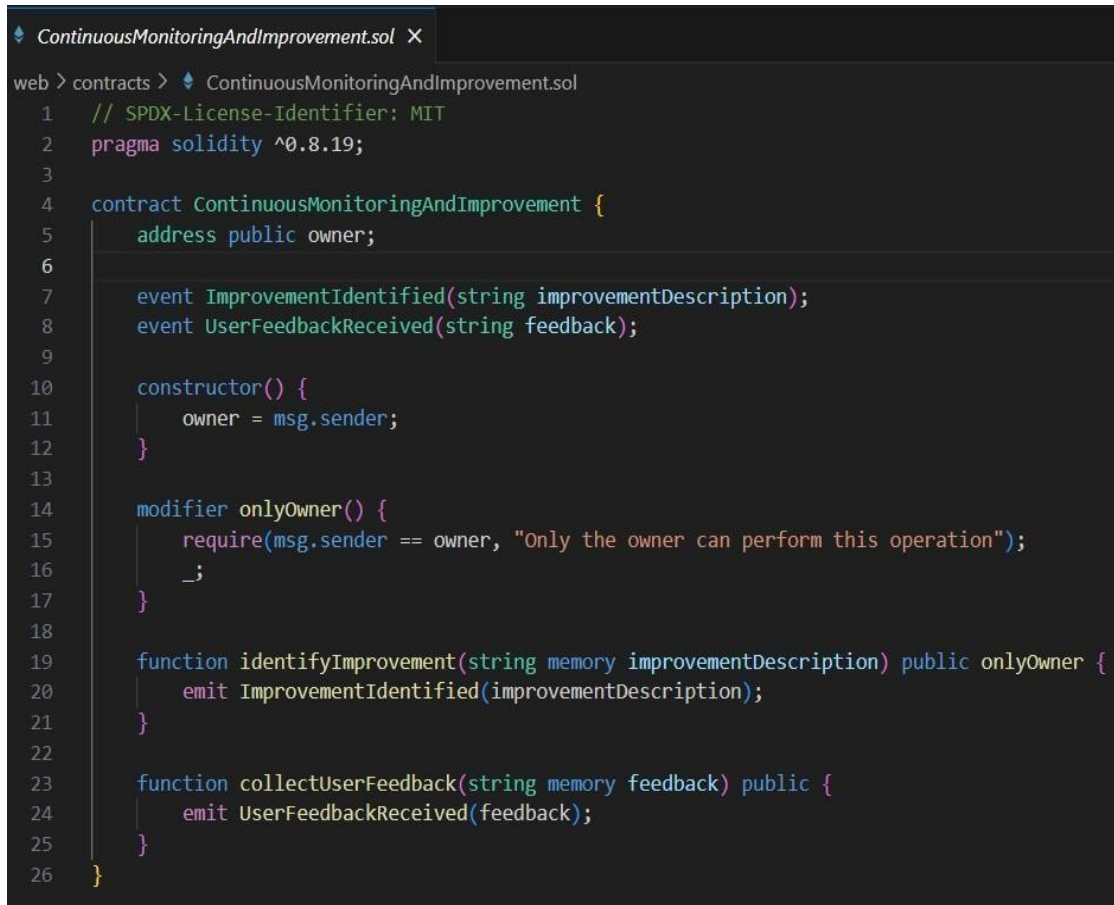


```
MedicalRecordManagement.sol X
web > contracts > MedicalRecordManagement.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.19;
3
4 contract MedicalRecordManagement {
5     // Structure to store medical records
6     struct MedicalRecord {
7         string name;
8         string address;
9         string dataHash; // Hash of the actual medical data
10    }
11
12    // Mapping to store medical records
13    mapping(uint256 => MedicalRecord) public medicalRecords;
14
15    uint256 public totalMedicalRecords;
16
17    // Event to log when a new medical record is added
18    event MedicalRecordAdded(string name, string address, string dataHash);
19
20    // Function to add a medical record to the system
21    function addMedicalRecord(string memory _name, string memory _address, string memory _dataHash) public {
22        totalMedicalRecords++;
23        medicalRecords[totalMedicalRecords] = MedicalRecord(_name, _address, _dataHash);
24        emit MedicalRecordAdded(_name, _address, _dataHash);
25    }
26 }
```

Fig 13: 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. Continuous Monitoring and Improvement



```
ContinuousMonitoringAndImprovement.sol X
web > contracts > ContinuousMonitoringAndImprovement.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.19;
3
4 contract ContinuousMonitoringAndImprovement {
5     address public owner;
6
7     event ImprovementIdentified(string improvementDescription);
8     event UserFeedbackReceived(string feedback);
9
10    constructor() {
11        owner = msg.sender;
12    }
13
14    modifier onlyOwner() {
15        require(msg.sender == owner, "Only the owner can perform this operation");
16        _;
17    }
18
19    function identifyImprovement(string memory improvementDescription) public onlyOwner {
20        emit ImprovementIdentified(improvementDescription);
21    }
22
23    function collectUserFeedback(string memory feedback) public {
24        emit UserFeedbackReceived(feedback);
25    }
26 }
```

Fig 14: 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

Sample Output (Hyperledger Fabric)

```
    "Size": 10,
    "docType": "asset"
  },
  {
    "AppraisedValue": 700,
    "Color": "black",
    "ID": "asset5",
    "Owner": "Adriana",
    "Size": 15,
    "docType": "asset"
  },
  {
    "AppraisedValue": 800,
    "Color": "white",
    "ID": "asset6",
    "Owner": "Michel",
    "Size": 15,
    "docType": "asset"
  }
]
--> Submit Transaction: CreateAsset, creates new asset with ID, color, owner, size, and appraisedValue arguments
*** Result: committed
*** Result: {
  "ID": "asset13",
  "Color": "yellow",
  "Size": "5",
  "Owner": "Tom",
  "AppraisedValue": "1300"
}
```

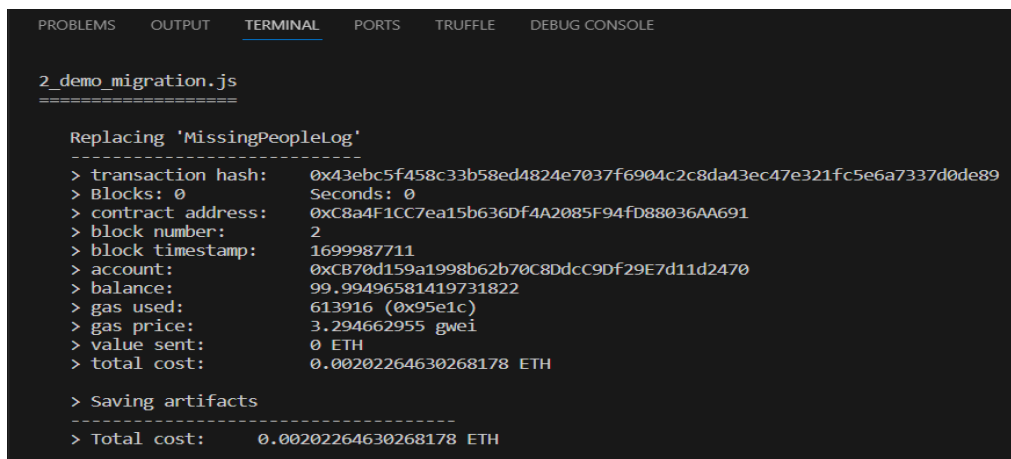
Fig 15: Hyperledger Fabric Sample Output

PREVIOUS WORK

SOLIDITY OUTPUT

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

7.1 Missing People Log Interface



```
PROBLEMS OUTPUT TERMINAL PORTS TRUFFLE DEBUG CONSOLE

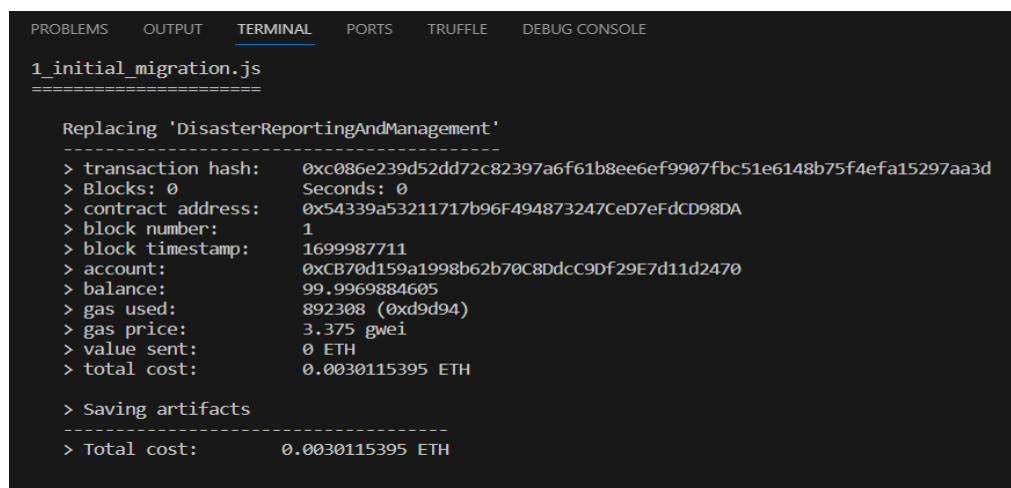
2_demo_migration.js
=====

Replacing 'MissingPeopleLog'
-----
> transaction hash: 0xc43ebc5f458c33b58ed4824e7037f6904c2c8da43ec47e321fc5e6a7337d0de89
> 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 16: Missing People Log (Output)

7.2. 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 17: Disaster Reporting and Management (Output)

7.3. 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 18: Aid Package Distribution (Output)

7.4. Critical Area Management

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  TRUFFLE  DEBUG CONSOLE

4_CriticalAreaManagement_migration.js
=====

Replacing 'CriticalAreaManagement'
-----
> transaction hash:    0xbfe34c5fdbbc64d30c330efc2c2843d6a89ba6ca0be421fedb791958e4b65ce36
> 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 19: Critical Area Management (Output)

7.5. Refugee Relief Camp Mapping

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  TRUFFLE  DEBUG CONSOLE

5_RefugeeReliefCampMapping_migration.js
=====

Replacing 'RefugeeReliefCampMapping'
-----
> transaction hash:    0x6eb12be6fdfa53bd86bdc45d8449e63b4a891ab4460732499a6243b1c6a87051
> Blocks: 0           Seconds: 0
> contract address:   0x25AbF698bA29eaDE01B2AE307f116EDA5E6C005c
> block number:       5
> block timestamp:    1699987712
> account:            0xCB70d159a1998b62b70C8DdcC9Df29E7d11d2470
> balance:            99.98680029737100886
> gas used:           1003664 (0xf5090)
> gas price:          3.083832351 gwei
> value sent:         0 ETH
> total cost:         0.003095131512734064 ETH

> Saving artifacts
-----
> Total cost:         0.003095131512734064 ETH
```

Fig 20: Refugee-Relief Camp Mapping (Output)

7.6. Medical Record Management

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  TRUFFLE  DEBUG CONSOLE

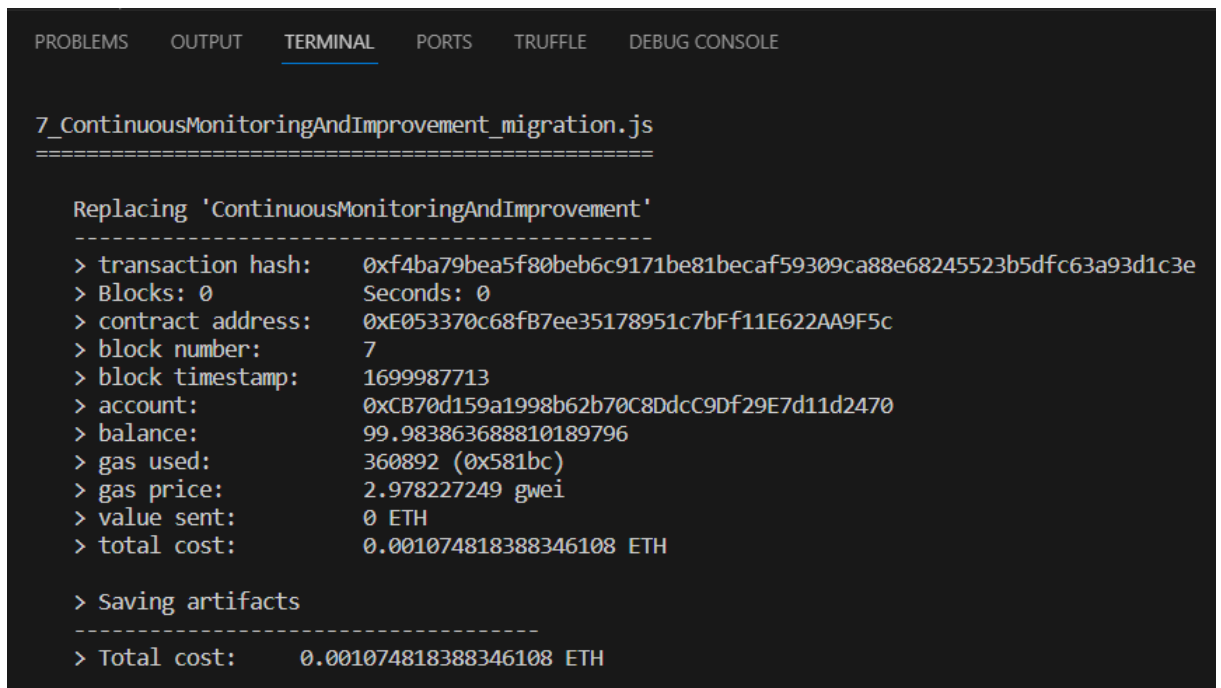
6_MedicalRecordManagement_migration.js
=====

Replacing 'MedicalRecordManagement'
-----
> transaction hash:    0x3029a7e00cb282d9d7980ff1a1b209d2be6038143bccdf8232eda1e0ba61f99a
> Blocks: 0           Seconds: 0
> contract address:   0xd16CC1a4920733DD35c1eA8C34D730Dfe38C4088
> block number:       6
> block timestamp:    1699987713
> account:            0xCB70d159a1998b62b70C8DdcC9Df29E7d11d2470
> balance:            99.984938507198535904
> gas used:           613916 (0x95e1c)
> gas price:          3.032646441 gwei
> value sent:         0 ETH
> total cost:         0.001861790172472956 ETH

> Saving artifacts
-----
> Total cost:         0.001861790172472956 ETH
```

Fig 21: Medical Record Management (Output)

7.7. Continuous Monitoring and Improvement



The screenshot shows a terminal window with tabs for PROBLEMS, OUTPUT, TERMINAL, PORTS, TRUFFLE, and DEBUG CONSOLE. The TERMINAL tab is active, displaying the output of a migration script named 7_ContinuousMonitoringAndImprovement_migration.js. The output indicates that the contract 'ContinuousMonitoringAndImprovement' is being replaced. It lists various transaction details: transaction hash, blocks (0), contract address, block number (7), block timestamp, account, balance, gas used, gas price, value sent, and total cost. It also shows the saving of artifacts and a final total cost.

```
PROBLEMS OUTPUT TERMINAL PORTS TRUFFLE DEBUG CONSOLE

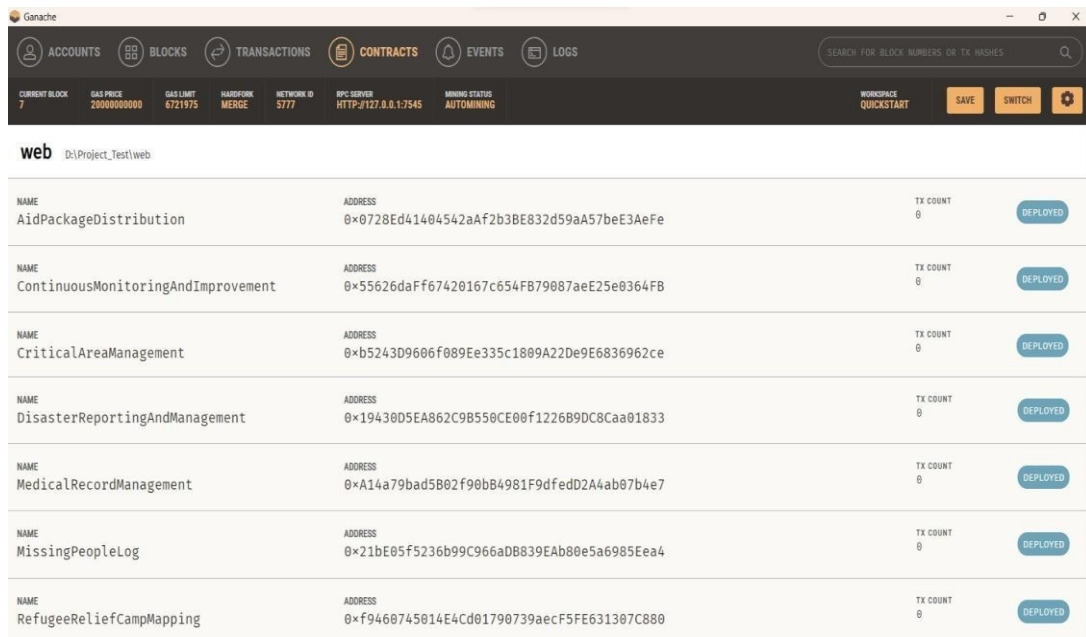
7_ContinuousMonitoringAndImprovement_migration.js
=====

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 22: Continuous Monitoring and Improvement (Output)

GANACHE OUTPUT



The screenshot shows the Ganache application interface. At the top, there's a navigation bar with tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below this, a status bar displays various network metrics like current block, gas price, gas limit, network ID, RPC server URL, and mining status. The main area is titled 'web' and shows a list of deployed contracts. Each contract entry includes its name, address, transaction count, and a 'DEPLOYED' button.

NAME	ADDRESS	TX COUNT	
AidPackageDistribution	0x0728Ed41404542aAf2b3BE832d59aA57beE3AeFe	0	DEPLOYED
ContinuousMonitoringAndImprovement	0x55626daFf67420167c654FB79087aeE25e0364FB	0	DEPLOYED
CriticalAreaManagement	0xb5243D9606f089Ee335c1809A22De9E6836962ce	0	DEPLOYED
DisasterReportingAndManagement	0x19438D5EA862C9B550CE08f1226B9DC8Caa01833	0	DEPLOYED
MedicalRecordManagement	0xA14a79bad5B02f90bB4981F9dfedD2A4ab07b4e7	0	DEPLOYED
MissingPeopleLog	0x21bE05f5236b99C966aDB839EAb80e5a6905Eea4	0	DEPLOYED
RefugeeReliefCampMapping	0xf9460745014E4Cd01790739aecF5FE631307C880	0	DEPLOYED

Fig 23: Ganache Output

The Ganache output validates the successful deployment and execution of the seven algorithms outlined in our blockchain-based disaster management system. Each algorithm, from Missing People Log to Continuous Monitoring and Improvement, demonstrates secure, transparent, and efficient functionality, ensuring the reliability of our decentralized humanitarian system. [14]

Chapter – VIII

CONCLUSION

8 CONCLUSION

The incorporation of blockchain technology into disaster management signifies a paradigm shift in humanitarian endeavors. By deploying decentralized solutions, our project effectively tackles key challenges in missing persons' tracking, aid distribution, critical area management, refugee-relief camp mapping, disaster reporting, and medical record keeping. The successful execution of the project in Solidity, coupled with its deployment on Ganache, serves as noteworthy accomplishments. These milestones lay the foundation for future strides, especially with the planned integration of Hyperledger Fabric. Beyond the immediate improvements in disaster relief efficiency, our project stands as a testament to blockchain's potential to revolutionize crisis response. It introduces a new era of transparency and precision, demonstrating how this technology can be a crucial ally in safeguarding lives during times of calamity.

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. ConsenSys. Ethereum: A next-generation cryptocurrency and decentralized application platform (ethereum white paper). Technical report, 2015.
7. D. Tapscott and A. Tapscott. Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World. 2016.
8. W. Mougayar. The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology. 2016.
9. F. Tschorsch and B. Scheuermann. Bitcoin and Beyond: Cryptocurrencies, Blockchains, and Global Governance. 2016.
10. A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. 2016.
11. M. Swan. Blockchain: Blueprint for a New Economy (2nd ed.). 2017.
12. M. J. Casey and P. Vigna. The Truth Machine: The Blockchain and the Future of Everything. 2018.
13. D. Mazières and A. Kohli. The stellar consensus protocol: A federated model for internet-level consensus. 2018.
14. IEEE Xplore Digital Library. Blockchain technology for humanitarian disaster relief: A case study. 2018. DOI:10.1109/INCET57972.2023.10170452
15. X. Zhang, P. Xu, X. Wang, and X. Li, “Blockchain technology: A panacea or pariah for resources conservation and recycling in the smart city?,” Resources, Conservation & Recycling, vol. 138, pp. 429–431, 2018.
16. M. Ichikawa, H. Watanabe, Y. Takahashi, T. Iwamoto, and 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 of Public Health (Japan), Volume 67, Issue 1, 2018 .
DOI: 10.5381/jniph.2017.67.1_4