ETHEREUM DECENTRALIZED IDENTITY SMARTCONTRACT

PROJECT REPORT

Submitted by

TEAM ID: NM2023TMID00516

GKM

College of Engineering and Technology

Team Members :

K.V.SUJETHA - 410820121035

P.SANDHYA - 410820121032

R.ETTIYAMMAL - 410820121011

A.LAVANYA - 410820121025

TABLE OF CONTENT

CHAPTER – 1

INTRODUCTION

1.1 Project Overview :

Ethereum Decentralized Identity Smart contract

Ethereum decentralized identity smart contracts are a revolutionary application of blockchain technology that aims to empower individuals with control over their personal data and digital identities. These smart contracts utilize the Ethereum blockchain to create a secure, transparent, and self-sovereign system for managing identity information. In this introduction, we will explore the fundamental concepts behind Ethereum decentralized identity smart contracts, their benefits, and the potential they hold in revolutionizing the way we manage and protect our digital identities in an increasingly interconnected world.

1.2 Purpose :

Self-Sovereign Identity: These smart contracts enable individuals to have self-sovereign control over their identity data, reducing reliance on centralized identity providers and giving users the power to manage their personal information.

Privacy and Security: Ethereum decentralized identity smart contracts offer a secure and tamper-resistant way to store and verify identity data, reducing the risk of data breaches and identity theft.

Interoperability: They allow for interoperability across various applications and services, ensuring that a user's identity can be seamlessly utilized across different platforms

CHAPTER-2

LITERATURE SURVEY

2.1 Existing problem :

Ethereum decentralized identity smart contracts face challenges related to scalability, privacy, key management, interoperability, user adoption, regulatory compliance, governance,

security against Sybil attacks and identity fraud, and data portability. These issues hinder the effective implementation and adoption of decentralized identity solutions on the Ethereum blockchain.

2.2 References :

S.NO	NAME OF THE AUTHOR	TITLE OF THE PAPER	REMARKS	YEAR OF PUBLICATIONS
1.	Anil kumar Dixit	Decentralized smart contract voting system.	Decentralize online voting procedure that build the for a long time, interviewing genuine trust needs has already been a challenge. The suggested system uses the ganache tool and technology to create a local primary network that provides additional security and transparency while exploiting.	2022
2.	V.S.Akshaya	Blockchain based Healthcare Data Management.	Exchange of healthcare data between hospitals is limited by privacy and dependency on centralized data management systems. Such a centralised storage can be a concern since it can lead to data leakage, data	2022

			manipulation, mistrust, and single point of failure.	
3.	Zhenhao nong	A Secure decentralized trusttless E-voting system based on smart contract	E-voting plays a significant role in social activities. The trust of the voting results and the privacy of each voter are always the most important concerns in designing secure e-voting system. The contract is deployed on the Ethereum private network. Some feasibility and cost analysis on money and time are also provided.	2019

2.3 Problem Statement Definition :

Scalability Solutions: Ethereum 2.0 (or ETH 2.0) is an upgrade to the Ethereum network designed to improve scalability by implementing Proof of Stake (PoS) and shard chains. Layer 2 scaling solutions, like rollups, can also enhance the efficiency of identity-related transactions.

User Education and Incentives: Initiatives to educate users about the benefits of decentralized identity, along with incentives such as rewards for managing their own data, can encourage adoption.

Identity Recovery Mechanisms: Developing secure and user-friendly identity recovery processes, such as multi-signature accounts or decentralized identity recovery services, can help users regain access to their identities in case of issues like lost keys.

Legal and Regulatory Compliance: Collaborating with legal experts and regulators to ensure compliance with identity and data protection laws is essential. Initiatives like the Self-Sovereign Identity (SSI) Consortium are working on this.

Interoperability Standards: Organizations like the Decentralized Identity Foundation (DIF) are working on developing standards for interoperability between various decentralized identity systems, making it easier to exchange identity information.

CHAPTER – 3

IDEATION & PROPOSED SOLUTION

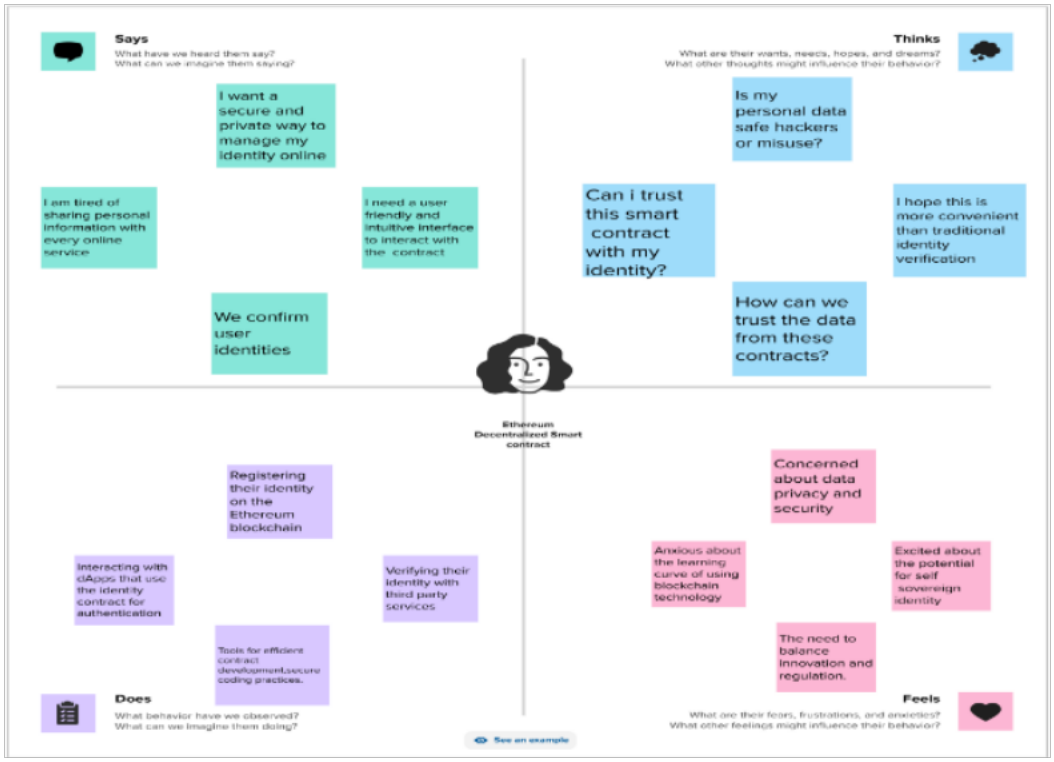
3.1 Empathy Map Canvas :

An empathy map for an Ethereum decentralized identity smart contract helps developers understand the user's perspective and needs.

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.

It is a useful tool to helps teams better understand their users.

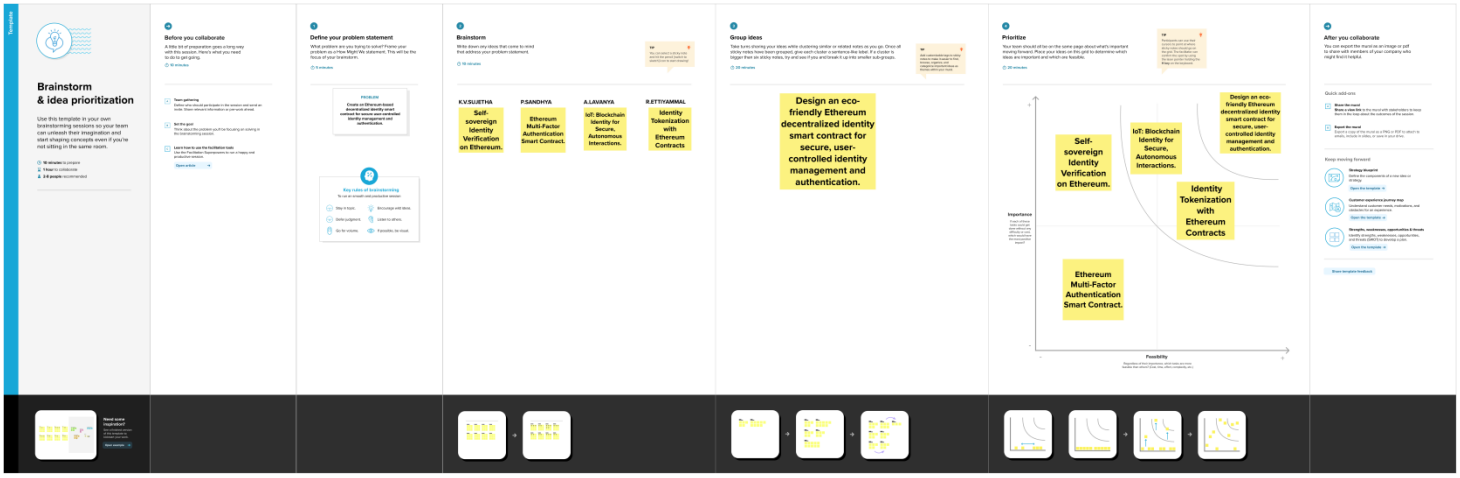
Creating an effective solution reguires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.



3.2 Ideation & Brainstorming :

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions. Implement a smart contract that allows users to create their DIDs, associating them with their Ethereum addresses.Enable DID management, including updates and revocation.Incorporate the ability for users to store verifiable credentials within the smart contract.

Support different types of verifiable credentials, such as academic certificates, professional licenses, and personal identification documents.



CHAPTER-4

REQUIREMENT ANALYSIS

4.1 Functional requirements :

User Registration and Onboarding: Allow users to create their decentralized identity by registering on the blockchain.

Identity Verification: Implement a mechanism for verifying users' identities, possibly through a trusted authority or a decentralized consensus process.

Data Storage: Store user data in a secure and private manner. This data can include personal

information, public keys, or other relevant information.

Access Control: Define who can access and update the identity information. This could involve role-based access control.

Key Management: Manage cryptographic keys securely, including key generation, storage, and recovery mechanisms.

Privacy and Security: Ensure that sensitive user information is stored and transmitted securely and that users have control over their data.

4.2 Non-Functional requirements :

Performance:

Response Time: Define acceptable response times for identity creation, updates, and verifications.

Throughput: Specify the number of identity transactions the smart contract should handle per second.

Scalability:

Elasticity: The smart contract should be able to scale with an increasing number of identities and transactions.

Load Balancing: Implement mechanisms to distribute the load across multiple nodes or shards.

Security:

Data Encryption: Ensure that user data stored on the blockchain is encrypted to protect against unauthorized access.

Auditability: Enable comprehensive auditing and traceability of all identity-related transactions.

DDoS Protection: Implement measures to mitigate Distributed Denial of Service (DDoS) attacks.

High Availability: The smart contract should be available 24/7, with minimal downtime for maintenance.

Fault Tolerance: Design the system to be resilient to failures, including node failures and network disruptions.

Data Consistency: Define how data consistency is maintained in a distributed environment, especially in the case of network partitions or forks.

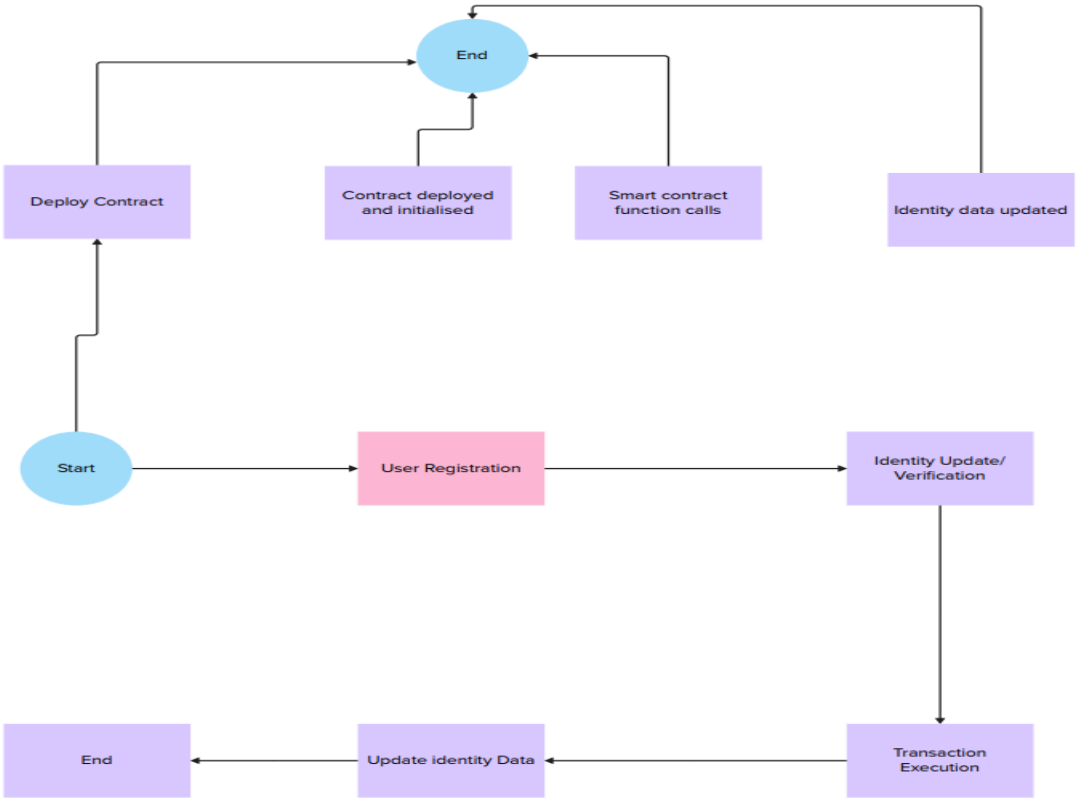
Compliance:

Regulatory Compliance: Ensure that the decentralized identity system complies with relevant legal and regulatory requirements, including data protection and privacy laws.

CHAPTER-5

PROJECT DESIGN

5.1 Data flow diagrams & User stories :



User stories :

Start: The process begins with the user's desire to create or manage their decentralized identity.

User Registration: Users register their identity on the blockchain using the smart contract.

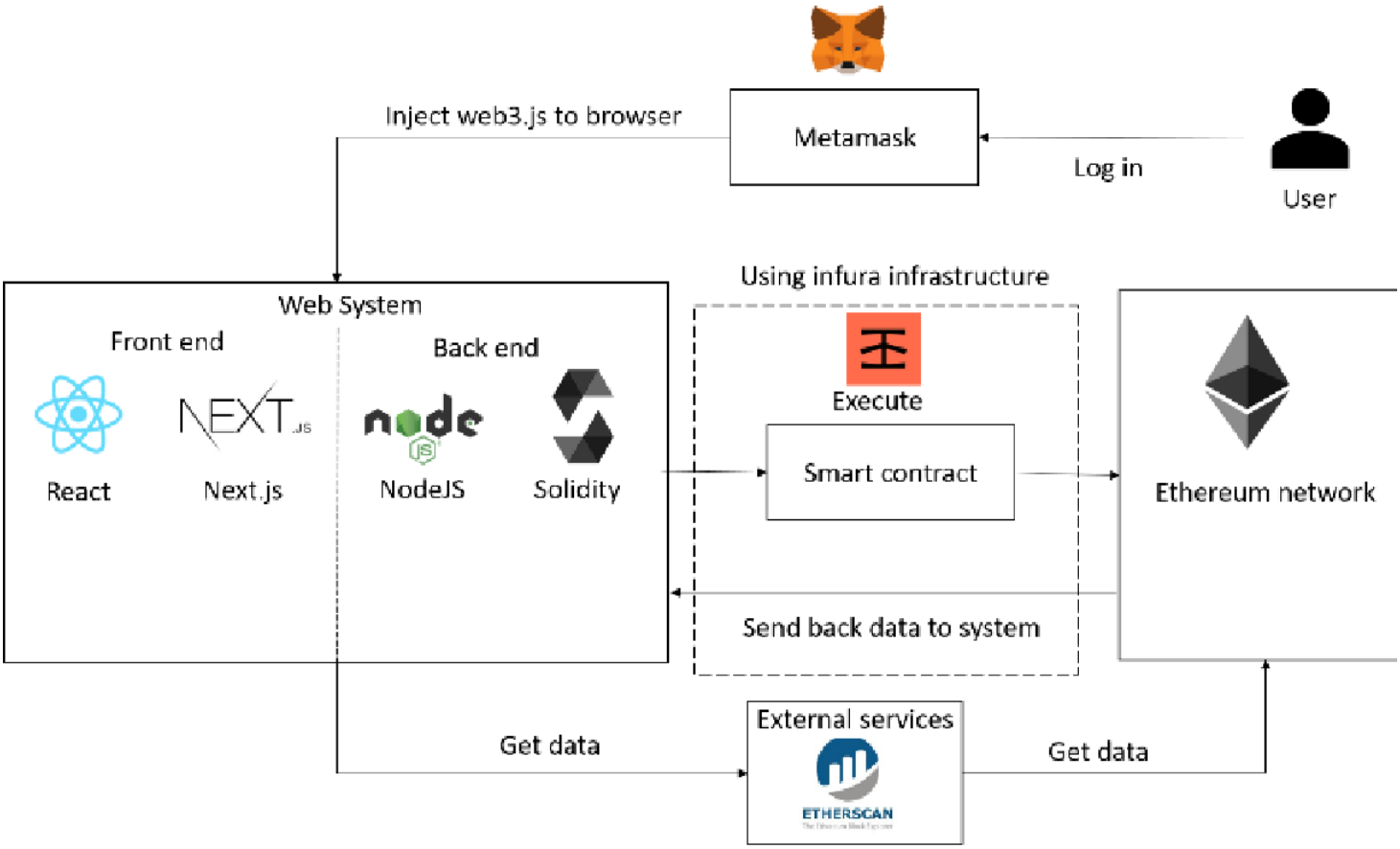
Identity Update or Verification: Users can update their identity data or go through a verification process if required.

Transaction Execution: Users and authorized parties can execute transactions on the smart contract, such as updating identity data or granting/rejecting access to their identity.

Update Identity Data: When transactions are executed, the smart contract updates the user's identity data or access permissions.

End: The process concludes

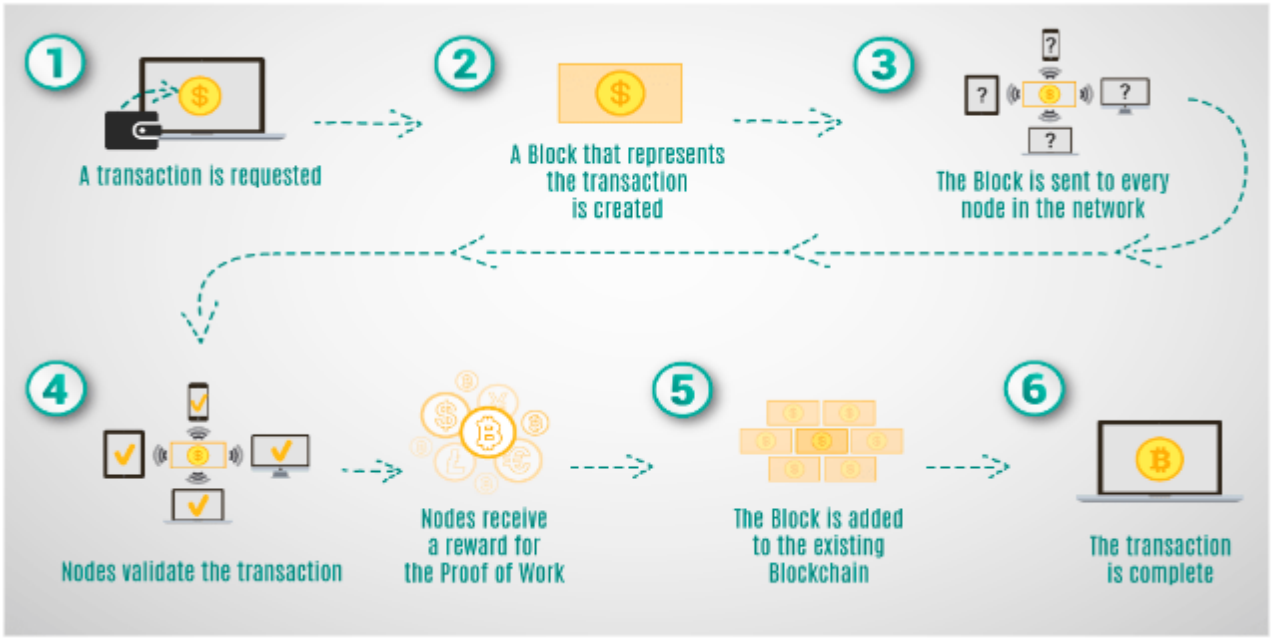
5.2 Solution architecture:

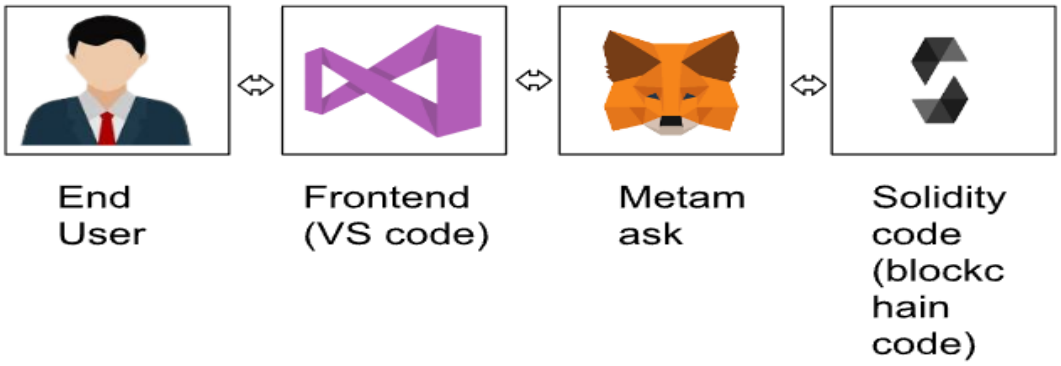


CHAPTER-6

PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture:





6.2 Sprint Planning and Estimation:

Sprint planning and estimation for an Ethereum decentralized identity smart contract project involve breaking down the development work into manageable tasks and assigning time estimates. Here's a simplified example of how you can approach this process:

Product Backlog Creation: Begin by creating a product backlog, listing all the features and tasks required

for your decentralized identity smart contract.

Prioritization: Prioritize the backlog items based on their importance and dependency. Essential features should be at the top of the list.

User Stories: Convert the high-level requirements into user stories or specific tasks. For example, "As a user, I want to register my decentralized identity."

Task Estimation: Estimate the effort required for each task. You can use story points, time units, or any estimation method that works for your team.

Sprint Planning: Decide on the sprint duration (e.g., 2 weeks). Select a set of tasks from the product backlog based on priority and available resources for the sprint.

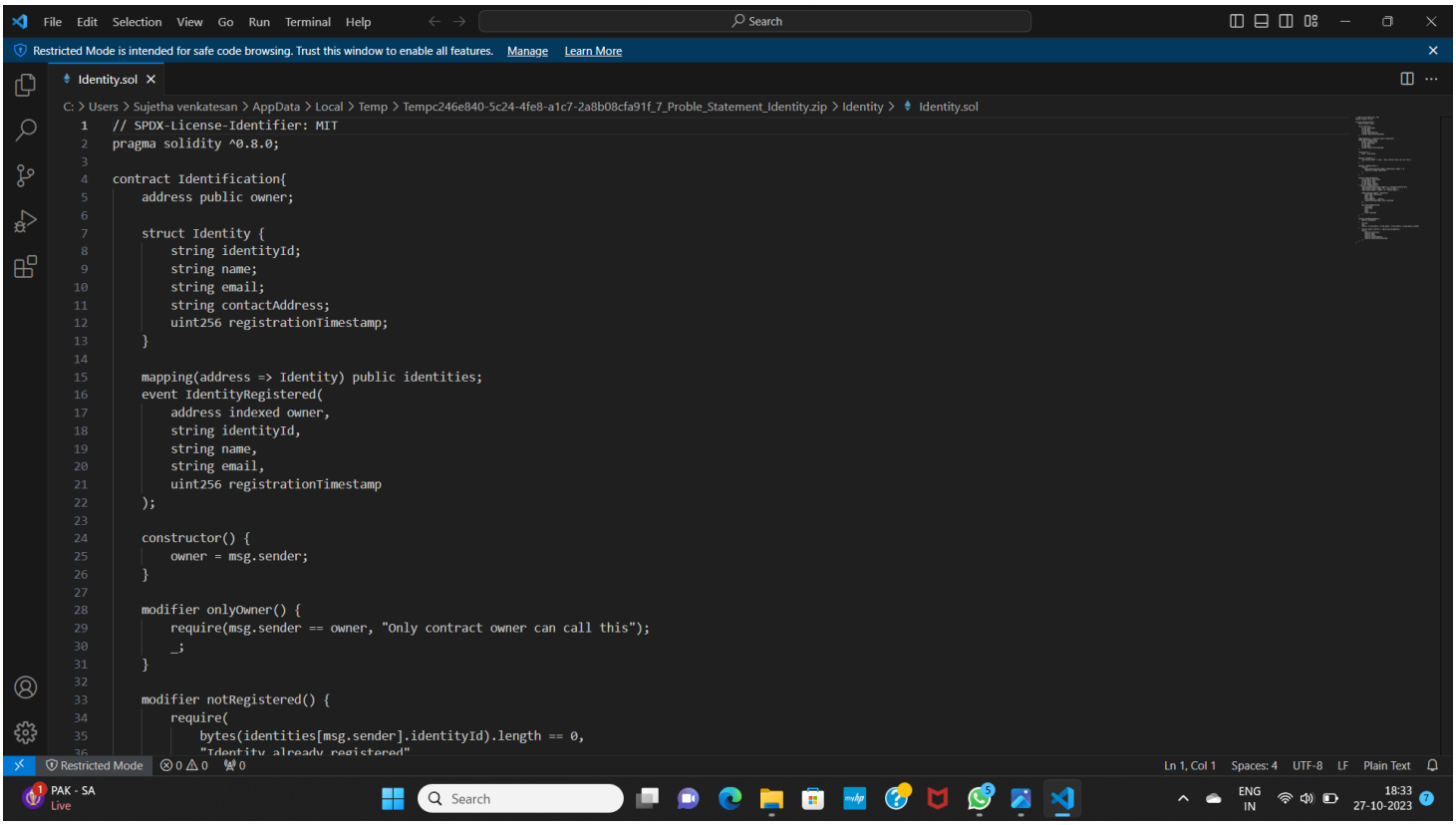
Estimate in Hours or Points: Estimate the time or effort required for each subtask. This could be in hours (e.g., 8 hours) or points (e.g., 5 story points).

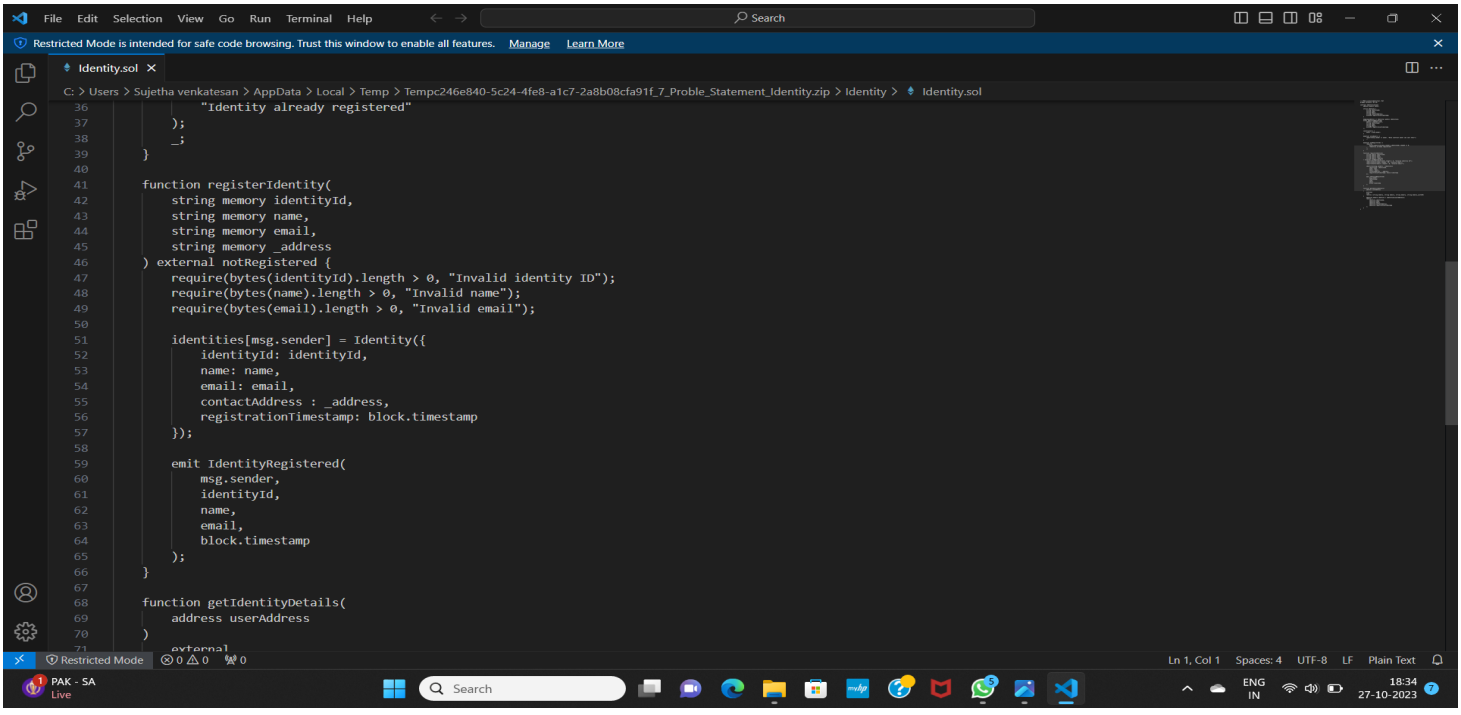
Capacity Planning: Determine the team's capacity for the sprint. How many hours or points can the team complete in one sprint?

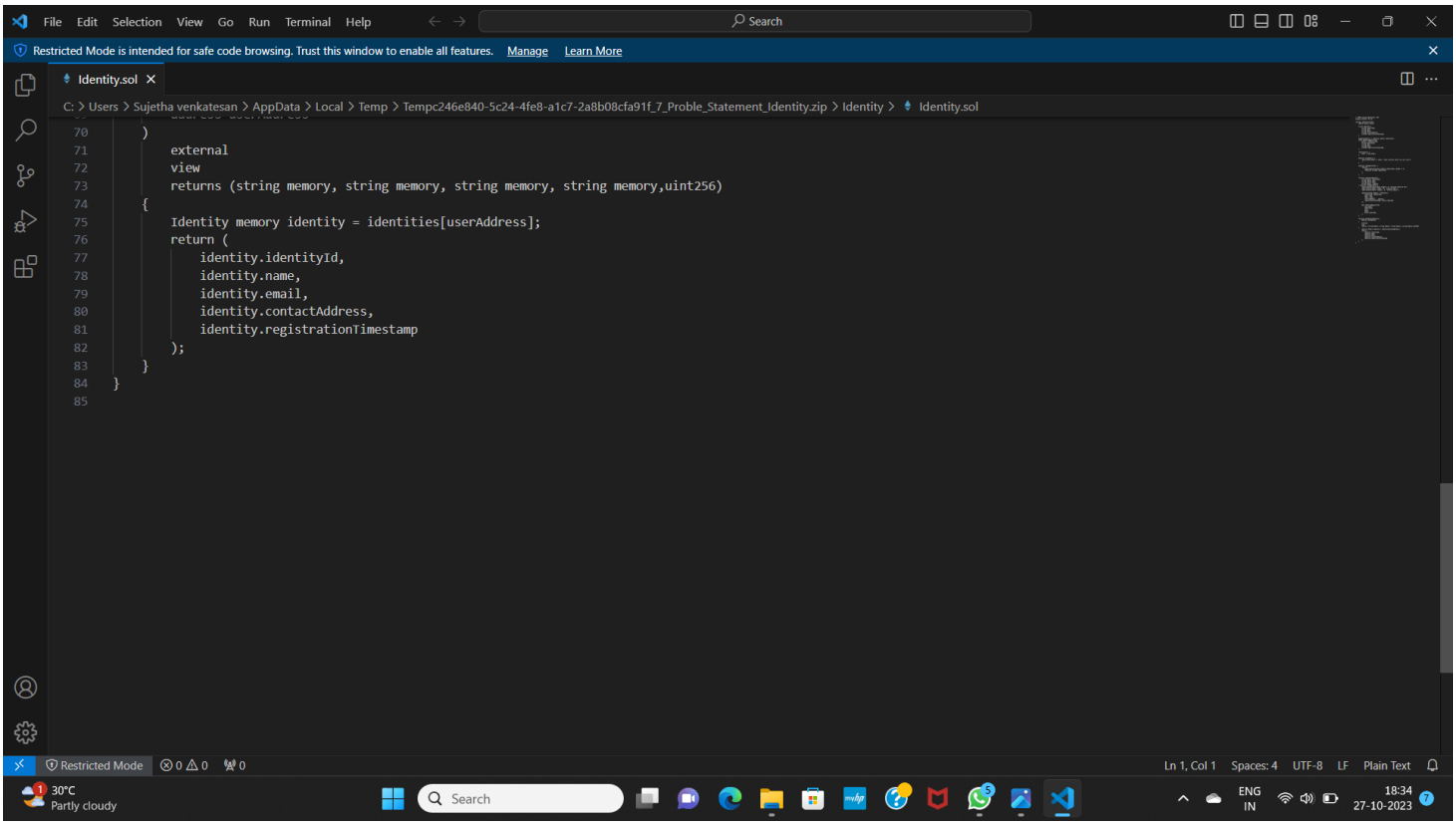
CHAPTER-7

CODING AND SOLUTIONS

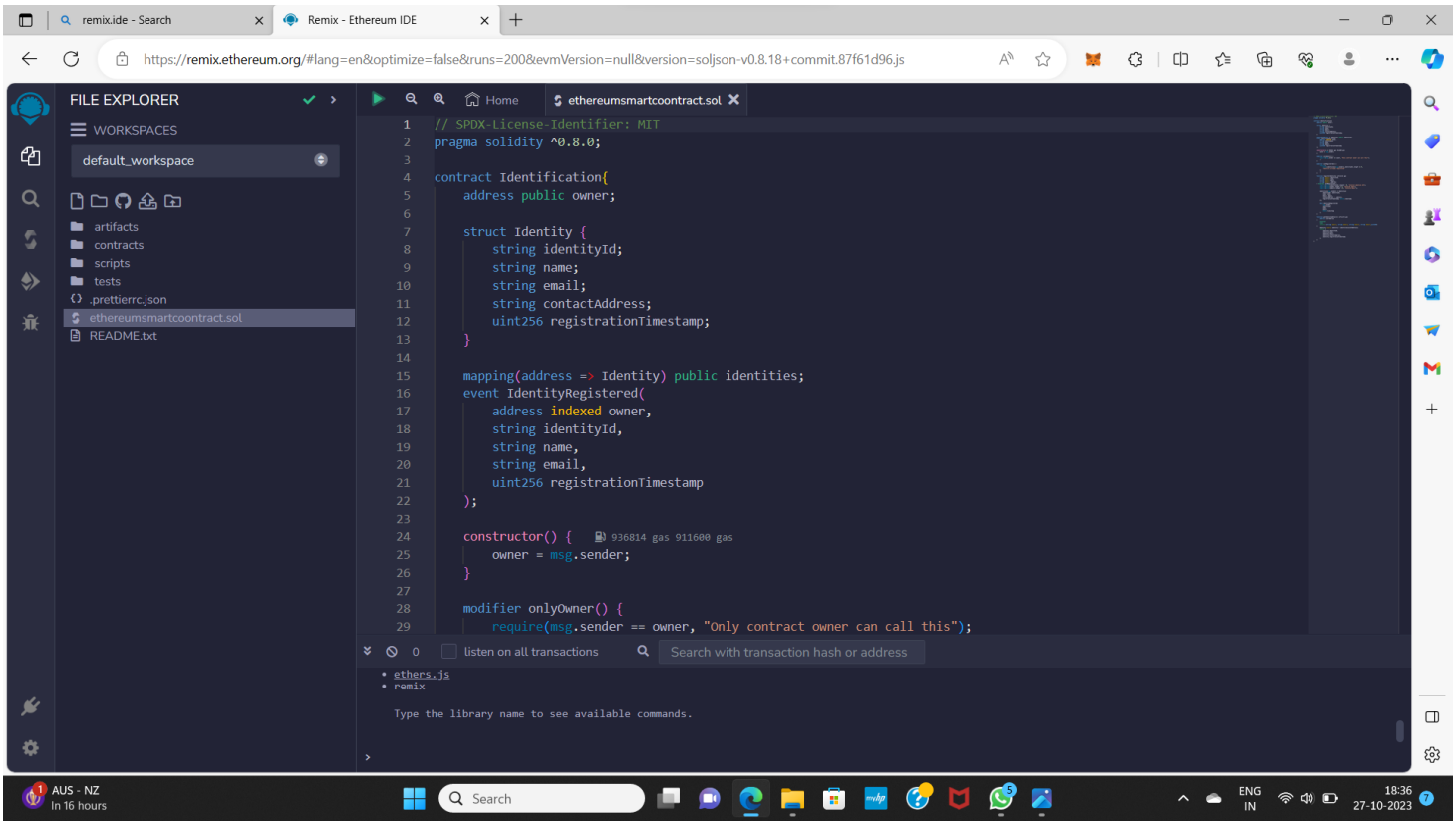
7.1 Visual Studio Coding :

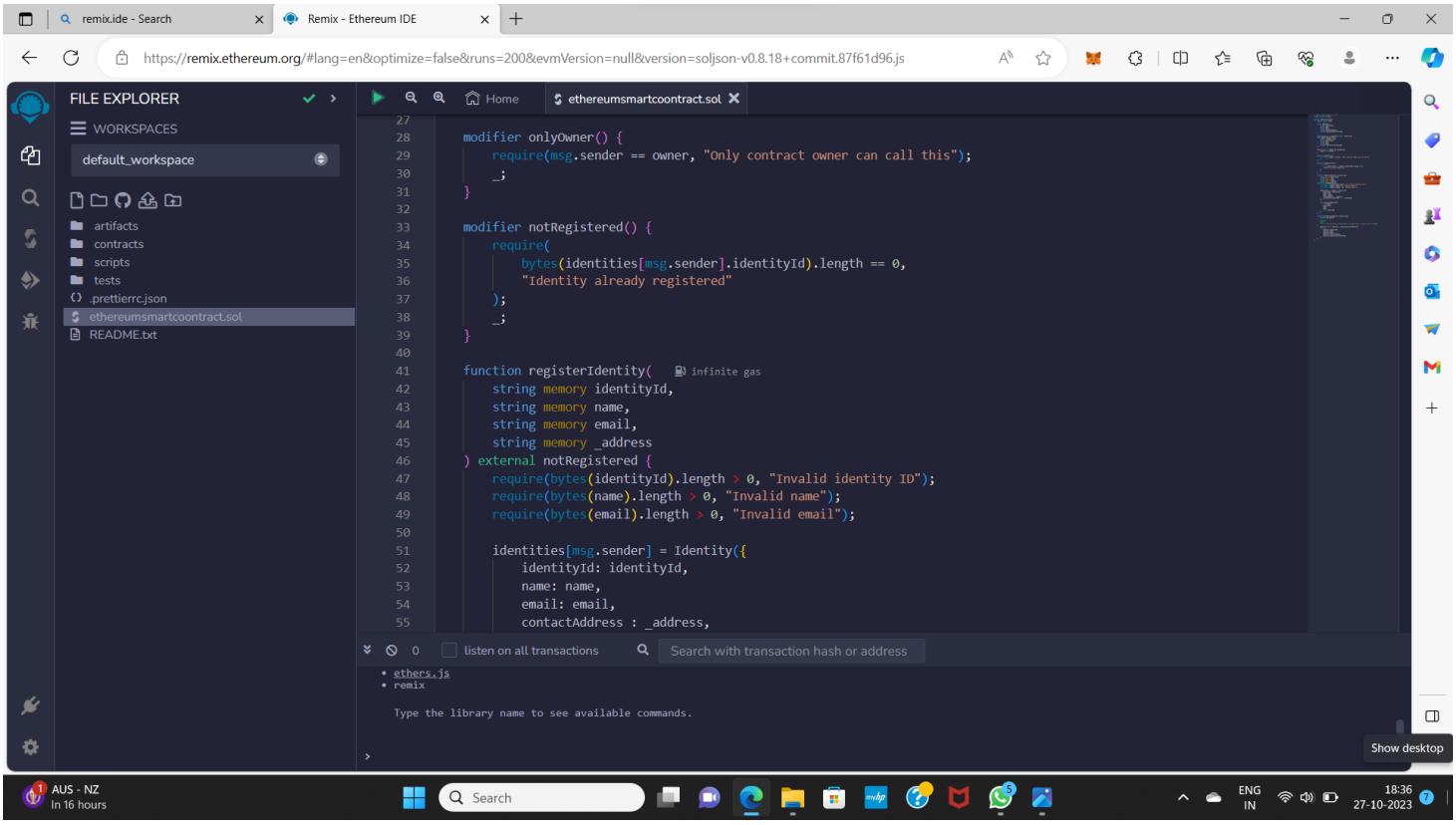

File Edit Selection View Go Run Terminal Help
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
Identity.sol
C: > Users > Sujetha venkatesan > AppData > Local > Temp > Tempc246e840-5c24-4fe8-a1c7-2a8b08cfa91f_7_Proble_Statement_Identity.zip > Identity > Identity.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Identification{
address public owner;
struct Identity {
string identityId;
string name;
string email;
string contactAddress;
uint256 registrationTimestamp;
}
mapping(address => Identity) public identities;
event IdentityRegistered(
address indexed owner,
string identityId,
string name,
string email,
uint256 registrationTimestamp
);
constructor() {
owner = msg.sender;
}
modifier onlyOwner() {
require(msg.sender == owner, "Only contract owner can call this");
_;
}
modifier notRegistered() {
require(
bytes(identities[msg.sender].identityId).length == 0,
Restricted Mode
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text
PAK - SA
Live
Search
ENG
IN
18:33
27-10-2023

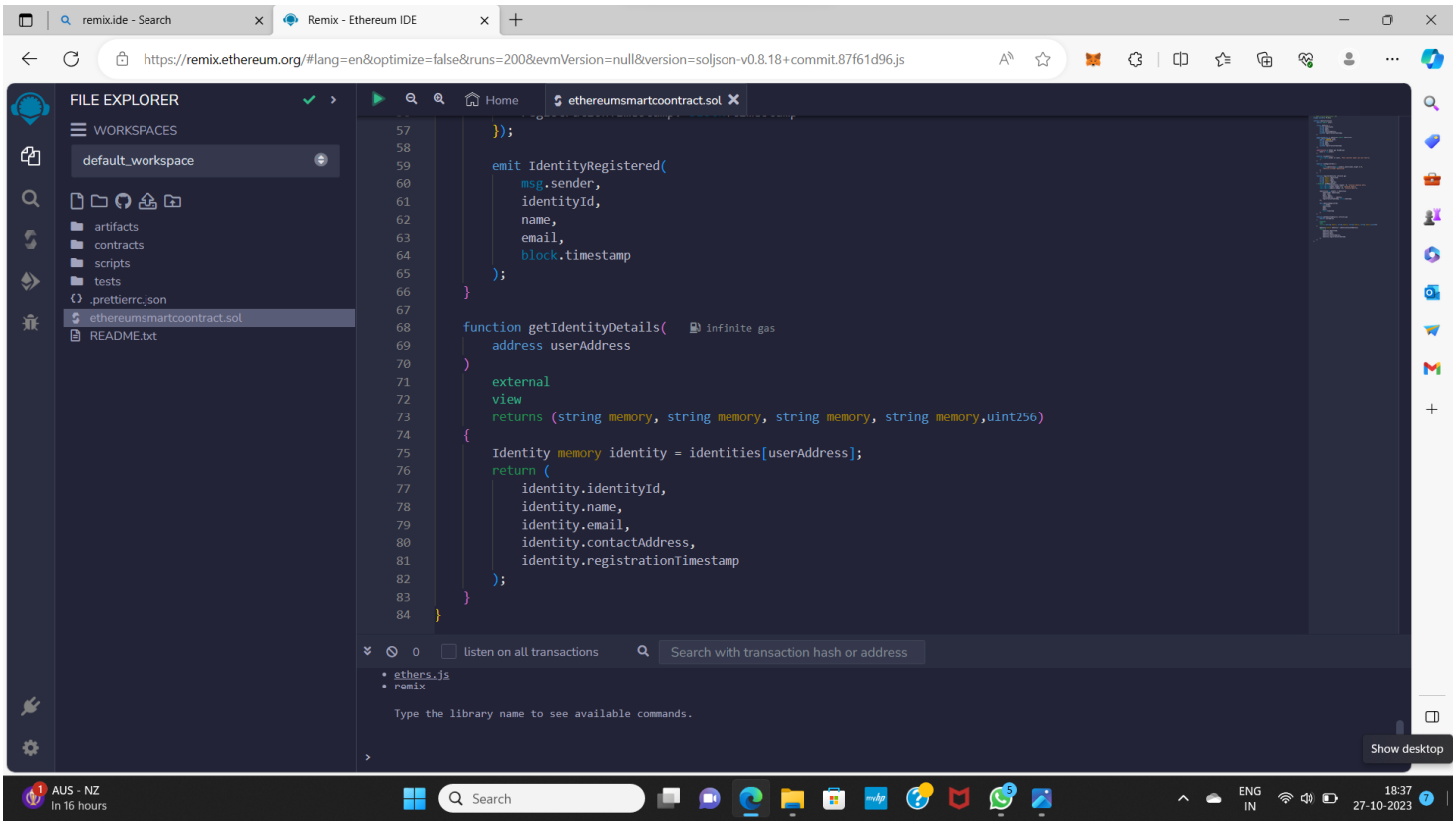

File Edit Selection View Go Run Terminal Help
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
Identity.sol
C: > Users > Sujetha venkatesan > AppData > Local > Temp > Tempc246e840-5c24-4fe8-a1c7-2a8b08cfa91f_7_Proble_Statement_Identity.zip > Identity > Identity.sol
"Identity already registered"
);
_;
}
function registerIdentity(
string memory identityId,
string memory name,
string memory email,
string memory _address
) external notRegistered {
require(bytes(identityId).length > 0, "Invalid identity ID");
require(bytes(name).length > 0, "Invalid name");
require(bytes(email).length > 0, "Invalid email");
identities[msg.sender] = Identity({
identityId: identityId,
name: name,
email: email,
contactAddress : _address,
registrationTimestamp: block.timestamp
});
emit IdentityRegistered(
msg.sender,
identityId,
name,
email,
block.timestamp
);
}
function getIdentityDetails(
address userAddress
)
external
Restricted Mode
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text
PAK - SA
Live
Search
ENG
IN
18:34
27-10-2023

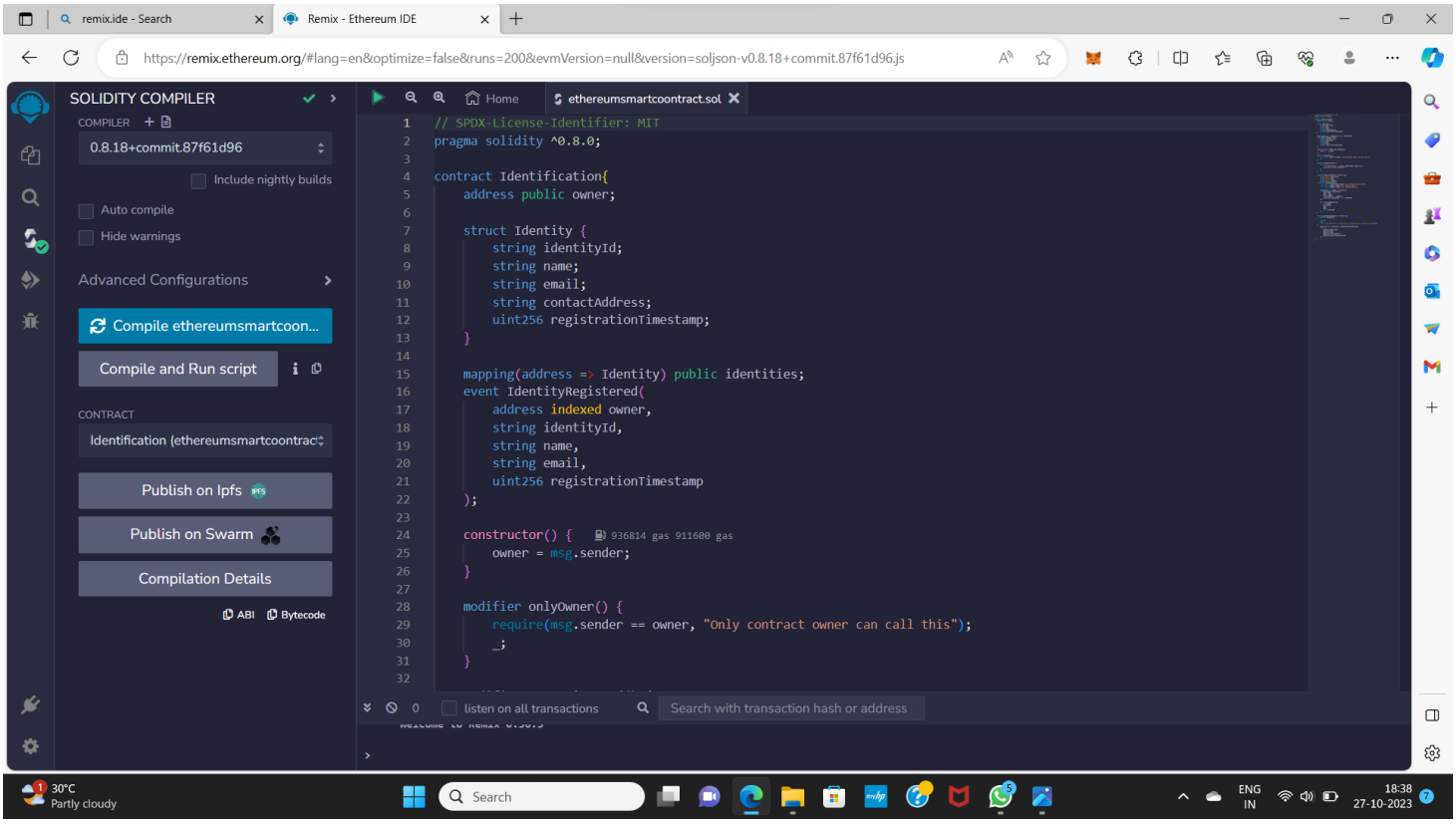


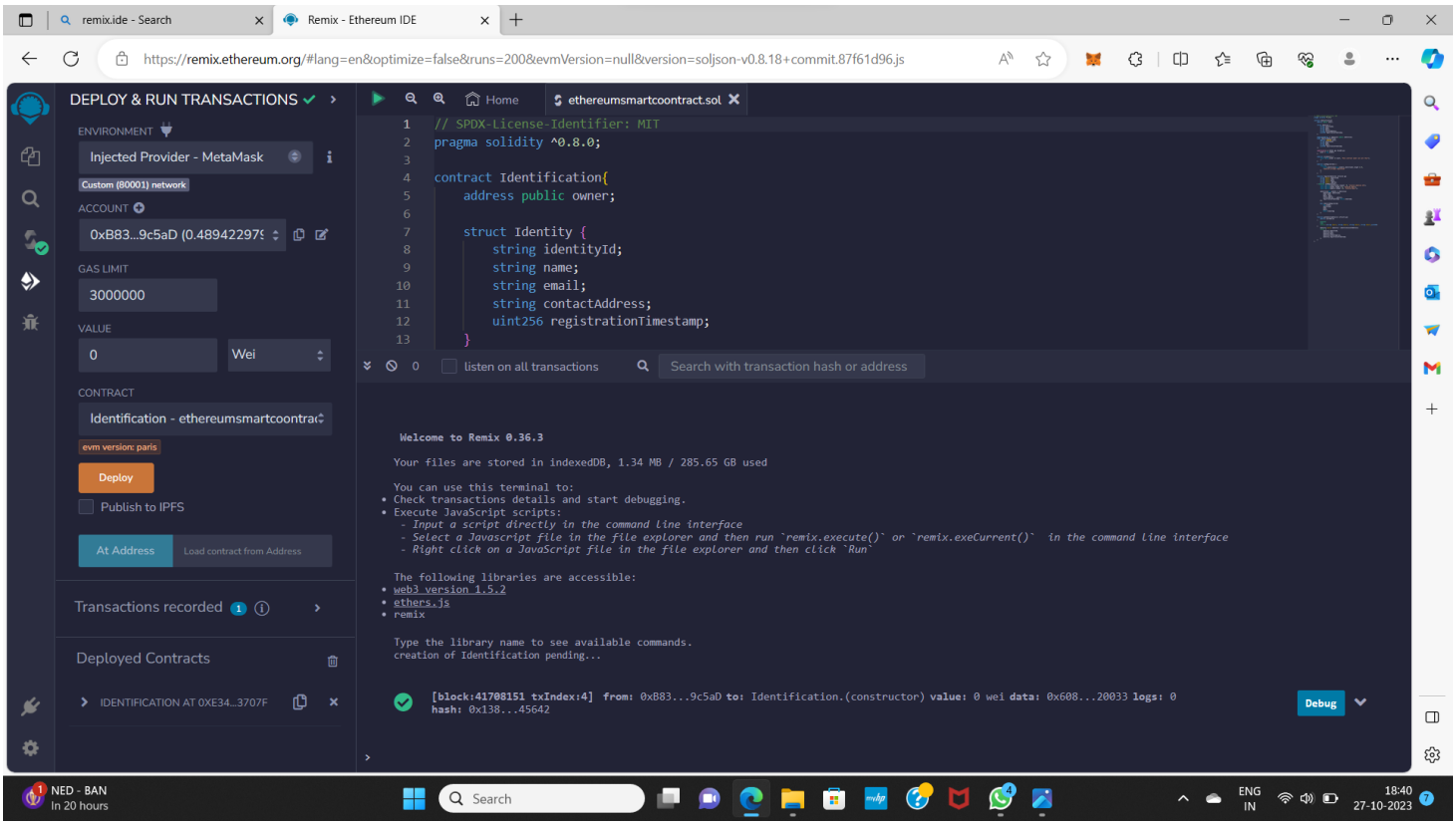
7.2 Remix coding :


remix.ide - Search
Remix - Ethereum IDE
https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
FILE EXPLORER
WORKSPACES
default_workspace
artifacts
contracts
scripts
tests
.prettierrc.json
ethereumsmartcoontract.sol
README.txt
Home
ethereumsmartcoontract.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Identification{
address public owner;
struct Identity {
string identityId;
string name;
string email;
string contactAddress;
uint256 registrationTimestamp;
}
mapping(address => Identity) public identities;
event IdentityRegistered(
address indexed owner,
string identityId,
string name,
string email,
uint256 registrationTimestamp
);
constructor() { 936814 gas 911600 gas
owner = msg.sender;
}
modifier onlyOwner() {
require(msg.sender == owner, "Only contract owner can call this");
listen on all transactions
Search with transaction hash or address
ethers.js
remix
Type the library name to see available commands.
AUS - NZ
In 16 hours
Search
ENG
IN
18:36
27-10-2023


remix.ide - Search
Remix - Ethereum IDE
https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
FILE EXPLORER
WORKSPACES
default_workspace
artifacts
contracts
scripts
tests
.prettierrc.json
ethereumsmartcoontract.sol
README.txt
Home
ethereumsmartcoontract.sol
modifier onlyOwner() {
require(msg.sender == owner, "Only contract owner can call this");
_;
}
modifier notRegistered() {
require(
bytes(identities[msg.sender].identityId).length == 0,
"Identity already registered"
);
_;
}
function registerIdentity(infinite gas
string memory identityId,
string memory name,
string memory email,
string memory _address
) external notRegistered {
require(bytes(identityId).length > 0, "Invalid identity ID");
require(bytes(name).length > 0, "Invalid name");
require(bytes(email).length > 0, "Invalid email");
identities[msg.sender] = Identity({
identityId: identityId,
name: name,
email: email,
contactAddress : _address,
listen on all transactions
Search with transaction hash or address
ethers.js
remix
Type the library name to see available commands.
Show desktop
AUS - NZ
In 16 hours
Search
ENG
IN
18:36
27-10-2023


remix.ide - Search
Remix - Ethereum IDE
https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
FILE EXPLORER
WORKSPACES
default_workspace
artifacts
contracts
scripts
tests
.prettierrc.json
ethereumsmartcoontract.sol
README.txt
Home
ethereumsmartcoontract.sol
57 });
58
59 emit IdentityRegistered(
60 msg.sender,
61 identityId,
62 name,
63 email,
64 block.timestamp
65);
66 }
67
68 function getIdentityDetails(infinite gas
69 address userAddress
70)
71 external
72 view
73 returns (string memory, string memory, string memory, string memory,uint256)
74 {
75 Identity memory identity = identities[userAddress];
76 return (
77 identity.identityId,
78 identity.name,
79 identity.email,
80 identity.contactAddress,
81 identity.registrationTimestamp
82);
83 }
84 }
listen on all transactions
Search with transaction hash or address
ethers.js
remix
Type the library name to see available commands.
Show desktop
AUS - NZ
In 16 hours
Search
ENG
IN
18:37
27-10-2023


remix.ide - Search
Remix - Ethereum IDE
https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
SOLIDITY COMPILER
COMPILER
0.8.18+commit.87f61d96
Include nightly builds
Auto compile
Hide warnings
Advanced Configurations
Compile ethereumsmartcoon...
Compile and Run script
CONTRACT
Identification (ethereumsmartcoontract
Publish on Ipfs
Publish on Swarm
Compilation Details
ABI
Bytecode
Home
ethereumsmartcoontract.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract Identification{
5 address public owner;
6
7 struct Identity {
8 string identityId;
9 string name;
10 string email;
11 string contactAddress;
12 uint256 registrationTimestamp;
13 }
14
15 mapping(address => Identity) public identities;
16 event IdentityRegistered(
17 address indexed owner,
18 string identityId,
19 string name,
20 string email,
21 uint256 registrationTimestamp
22);
23
24 constructor() { 936814 gas 911600 gas
25 owner = msg.sender;
26 }
27
28 modifier onlyOwner() {
29 require(msg.sender == owner, "Only contract owner can call this");
30 _;
31 }
32
listen on all transactions
Search with transaction hash or address
30°C
Partly cloudy
Search
ENG
IN
18:38
27-10-2023


remix.ide - Search
Remix - Ethereum IDE
https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
Injected Provider - MetaMask
Custom (80001) network
ACCOUNT
0xB83...9c5aD (0.48942297
GAS LIMIT
3000000
VALUE
0
Wei
CONTRACT
Identification - ethereumsmartcoontrac
evm version: paris
Deploy
Publish to IPFS
At Address
Load contract from Address
Transactions recorded 1
Deployed Contracts
IDENTIFICATION AT 0XE34...3707F
Home
ethereumsmartcoontract.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Identification{
address public owner;
struct Identity {
string identityId;
string name;
string email;
string contactAddress;
uint256 registrationTimestamp;
}
listen on all transactions
Search with transaction hash or address
Welcome to Remix 0.36.3
Your files are stored in indexedDB, 1.34 MB / 285.65 GB used
You can use this terminal to:
Check transactions details and start debugging.
Execute JavaScript scripts:
- Input a script directly in the command line interface
- Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()` in the command line interface
- Right click on a JavaScript file in the file explorer and then click `Run`
The following libraries are accessible:
web3 version 1.5.2
ethers.js
remix
Type the library name to see available commands.
creation of Identification pending...
[block:41708151 txIndex:4] from: 0xB83...9c5aD to: Identification.(constructor) value: 0 wei data: 0x608...20033 logs: 0
hash: 0x138...45642
Debug
NED - BAN
In 20 hours
Search
ENG
IN
18:40
27-10-2023

CHAPTER-8

PERFORMANCE TESTING

8.1 Performance metrices:

Performance testing a decentralized identity smart contract on the Ethereum blockchain involves assessing how the contract functions under various conditions to ensure it can handle real-world usage. Here's how you can approach it:

Determine what performance metrics are important for your use case. This might include transaction throughput, latency, gas costs, and scalability.

Select Testing Tools: Choose performance testing tools like Truffle, Hardhat, or specialized blockchain testing tools like Ganache, and configure them for your smart contract.

Simulate Realistic Load: Create test scenarios that simulate realistic user loads. This can include varying numbers of concurrent users, types of transactions, and transaction rates.

Measure Transaction Throughput: Evaluate how many transactions per second (TPS) your smart contract can handle. Increase the load until you reach the contract's throughput limits.

Monitor Gas Costs: Assess the gas costs associated with executing transactions. Identify which contract functions are more gas-intensive and optimize if necessary.

Analyze Latency: Measure the time it takes for transactions to be confirmed on the Ethereum blockchain. Understand the latency associated with your smart contract under different loads.

Stress Testing: Push your smart contract to its limits by increasing the load until it starts to show signs of congestion or slower transaction confirmation.

Scalability Testing: Assess how your smart contract scales as more users and transactions are added. Consider Ethereum 2.0 updates if scalability is a concern.

Security Testing: Ensure your smart contract can handle unexpected inputs and that it is secure against potential attacks. This is critical for a decentralized identity system.

Optimize and Refine: Based on the test results, optimize your smart contract code, gas efficiency, and

consider Ethereum layer-2 solutions if necessary to improve performance.

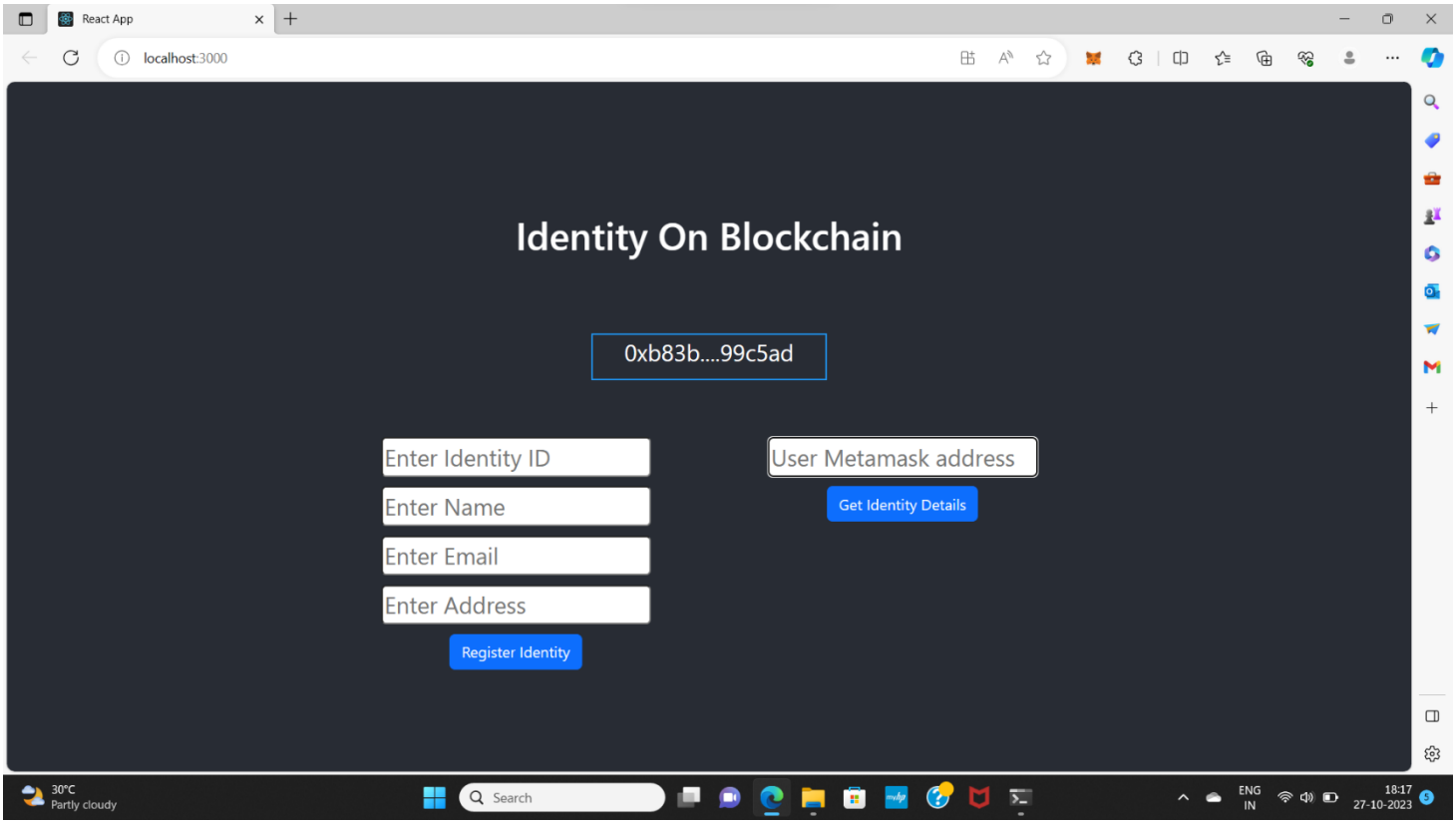
Documentation: Document the testing process, results, and any changes made to the smart contract. This documentation can be critical for audits and future reference.

Repeat Testing: Continue testing at regular intervals, especially when updates to the Ethereum network or your smart contract are made.

CHAPTER-9

RESULTS

9.1 Output :



CHAPTER-10

10.ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

Security: Your identity data is stored securely on the blockchain, reducing the risk of data breaches.

Privacy: Users have greater control over their personal information, choosing what to share and with whom.

Interoperability: Ethereum's decentralized identity standards can work with other blockchain and identity systems, promoting cross-platform compatibility.

User Control: Users have the ability to manage and update their identity without relying on a central authority.

Trust: The transparent and tamper-resistant nature of the blockchain enhances trust in identity verification processes.

Reduced Redundancy: Eliminates the need for redundant identity verification across various services.

Global Accessibility: Accessible to anyone with an internet connection, promoting financial inclusion and global identity access.

Immutable Records: Once recorded on the blockchain, identity information is permanent and resistant to alteration.

DISADVANTAGES:

Scalability: Ethereum's scalability issues can result in slower transaction speeds and higher fees, making it less practical for identity verification at a large scale.

Lack of Recovery: If a user loses access to their private key, there may be no recourse to recover their identity, which can be problematic.

Regulatory Challenges: Decentralized identity may face legal and regulatory challenges, as it can be difficult to enforce identity-related regulations on a decentralized network.

Technical Complexity: Implementing and managing decentralized identity solutions can be technically complex, reguiring a good understanding of blockchain technology.

Limited Adoption: Decentralized identity is still in its early stages, and widespread adoption may take time.

Human Error: Users are responsible for their private keys, and if they are lost or stolen due to human error, it can lead to identity loss.

Data Privacy: While decentralized identity aims to improve privacy, it's crucial to handle data carefully, as any data leakage can be permanent and public.

CHAPTER-11

11.CONCLUSION

In conclusion, Ethereum decentralized smart contracts represent a groundbreaking innovation in the realm of blockchain technology. These self-executing contracts, encoded on the Ethereum blockchain, offer a secure, transparent, and tamper-resistant way to automate, verify, and enforce agreements without the need for intermediaries. By enabling decentralized and trustless interactions, Ethereum smart contracts have the potential to revolutionize a wide range of industries, including finance, supply chain, legal, and more. However, it's important to note that while they offer numerous advantages, challenges such as code vulnerabilities, scalability issues, and legal considerations need to be carefully addressed. Nevertheless, the continued development and adoption of Ethereum smart contracts hold the promise of redefining how we conduct business and interact in the digital age, making processes more efficient, secure, and transparent.

CHAPTER-12

FUTURE SCOPE

The future scope for Ethereum's decentralized smart contracts is exceptionally promising. As one of the pioneers in blockchain technology and smart contract development, Ethereum

continues to evolve and adapt, opening up a world of possibilities. The integration of Ethereum 2.0 with its proof-of-stake mechanism is set to enhance scalability and energy efficiency, making it more accessible for a wider range of applications. This will likely lead to increased adoption in areas such as supply chain management, identity verification, and finance, while also fueling the growth of decentralized finance (DeFi) and non-fungible tokens (NFTs). Moreover, Ethereum's commitment to ongoing upgrades, like Ethereum 2.0 and Layer 2 solutions, is expected to improve transaction speeds and lower fees, further expanding its potential use cases. Overall, Ethereum's decentralized smart contracts are poised to play a pivotal role in reshaping various industries and the way we interact with digital assets, contracts, and applications in the future.

CHAPTER-13

APPENDIX

Source code github link: https://github.com/Sujethavenkat/Ethereum.NM

Video demo link:

https://drive.google.com/file/d/16AUD-mzGQgpMTUWvPGE7tLxPVaH5GxVy/view?usp=drivesdk