DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to Visvesvaraya Technological University (VTU), Belagavi,
Approved by AICTE and UGC, Accredited by NAAC with 'A' grade & ISO 9001 – 2015 Certified Institution)
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560 111, India

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

(Accredited by NBA Tier 1: 2022-2025)

Project Report on

SOLIDITY LANGUAGE

Submitted in partial fulfillment for the award of the degree of

Bachelor of Engineering
in
Information Science and Engineering

Submitted by

SANTHOSH B R	**1DS22IS130**
SANTUSHT ANAND	**1DS22IS132**
SARVOCHCHA SHARMA	**1DS22IS133**
MOHIT REDDY	**1DS22IS135**

Under the Guidance of

Dr. Vaidehi M
Associate Professor
Department of Information Science and Engineering
DSCE, Bengaluru

VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANASANGAMA, BELAGAVI-590018, KARNATAKA, INDIA
2024-25

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to Visvesvaraya Technological University (VTU), Belagavi,
Approved by AICTE and UGC, Accredited by NAAC with 'A' grade & ISO 9001 – 2015 Certified Institution)
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560 111, India

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

(Accredited by NBA Tier 1: 2022-2025)

CERTIFICATE

Certified that the project report entitled **"Solidity language**" carried out by **Santhosh B R-1DS22IS130**,**Santusht Anand-1DS22IS132,Sarvochcha Sharma-1DS22IS133,Mohit Reddy-1DS22IS135** a bonafide student of **DAYANANDA SAGAR COLLEGE OF ENGINEERING**, an autonomous institution affiliated to VTU, Belagavi in partial fulfillment for the award of Degree of **Bachelor of Information Science and Engineering** during the year **2024-2025**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements with respect to the work prescribed for the said Degree.

Signature of the Guide	**Signature of the HOD**	**Signature of the Principal**
Dr. Vaidehi M	**Dr. Annapurna P Patil**	**Dr. B G Prasad**
Associate Professor	**Dean Academics, Prof & Head**	**Principal**
Dept. of ISE, DSCE	**Dept. of ISE, DSCE, Bengaluru**	**DSCE, Bengaluru**
Bengaluru		

Name of the Examiners	**Signature with date**
1.	**..**
2.	**..**

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to Visvesvaraya Technological University (VTU), Belagavi,
Approved by AICTE and UGC, Accredited by NAAC with 'A' grade & ISO 9001 – 2015 Certified Institution)
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru-560 111, India

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

(Accredited by NBA Tier 1: 2022-2025)

DECLARATION

We, **Santhosh B R-1DS22IS130,Santusht Anand-1DS22IS132,Sarvochcha Sharma-1DS22IS133,Mohit Reddy-1DS22IS135,** respectively, hereby declare that the project work entitled " Title of the Project " has been independently done by us under the guidance of **'Dr.Vaidehi M', Associate Professor,** ISE department and submitted in partial fulfillment of the requirement for the award of the degree of **Bachelor of Information Science and Engineering** at **Dayananda Sagar College of Engineering**, an autonomous institution affiliated to VTU, Belagavi during the academic year 2024-2025.

We further declare that we have not submitted this report either in part or in full to any other university for the award of any degree.

SANTHOSH B R	**1DS22IS130**
SANTUSHT ANAND	**1DS22IS132**
SARVOCHCHA SHARMA	**1DS22IS133**
MOHIT REDDY	**1DS22IS135**

PLACE:Bengaluru

DATE:09/12/2024

ACKNOWLEDGEMENT

The satisfaction and euphoria accompanying the successful completion of any task would be incomplete without the mention of people who made it possible and under constant guidance and encouragement the task was completed. We sincerely thank the **Management of Dayananda Sagar College of Engineering, Bengaluru.**

We express our sincere regards and thanks to **Dr. B G Prasad, Principal, Dayananda Sagar College of Engineering, Bengaluru.** His constant encouragement guidance and valuable support have been an immense help in realizing this technical seminar.

We express our sincere regards and thanks to **Dr. Annapurna P Patil, Professor & Head, Department of Information Science and Engineering, Dayananda Sagar College of Engineering, Bengaluru.** Her incessant encouragement guidance and valuable technical support have been an immense help in realizing this project. Her guidance gave us the environment to enhance our knowledge, and skills and to reach the pinnacle with sheer determination, dedication, and hard work.

We would like to express profound gratitude to my guide **Guide name, designation, Department of Information Science and Engineering, Dayananda Sagar College of Engineering, Bengaluru** who has encouraged us throughout the project. **His/Her** moral support enabled us to complete my work successfully.

We express our sincere thanks to Project Coordinator **Dr. Vaidehi M**, **Assoc. Prof,** and **Dr. Bhavani K Asst. Prof**. of the **Department of Information Science and Engineering** for their continues support and guidance. We thank all teaching and non-teaching staff of the Department of Information Science and Engineering for their kind and constant support throughout the academic Journey.

SANTHOSH B R	**1DS22IS130**
SANTUSHT ANAND	**1DS22IS132**
SARVOCHCHA SHARMA	**1DS22IS133**
MOHIT REDDY	**1DS22IS135**

ABSTRACT

Solidity is a high-level, statically-typed programming language designed for creating smart contracts on the Ethereum blockchain. Its primary goal is to enable secure, efficient, and transparent development of decentralized applications (DApps). Inspired by JavaScript, Python, and C++, it provides a contract-based approach to encode business logic for trustless automation.

Key features include integration with the Ethereum Virtual Machine (EVM), support for cryptographic operations, and robust development tools like Remix and Truffle. Solidity has enabled widespread adoption in decentralized finance (DeFi), non-fungible tokens (NFTs), and decentralized autonomous organizations (DAOs).

Applications range from financial platforms and NFT marketplaces to supply chain tracking and governance systems. Looking ahead, Solidity's future scope includes improved scalability, interoperability, enhanced security, and integration with emerging technologies like artificial intelligence. Its contributions are vital to the growing blockchain ecosystem, laying the foundation for decentralized innovations.

The primary goal of Solidity is to facilitate the development of secure and efficient smart contracts, which are self-executing programs that automate processes and transactions without requiring intermediaries. To achieve this, Solidity offers a user-friendly syntax influenced by familiar programming languages such as JavaScript, Python, and C++, making it accessible to developers with varied backgrounds. It supports essential blockchain features, including transaction handling, cryptographic operations, and decentralized state management, enabling seamless integration with blockchain systems. Additionally, Solidity provides developers with the tools necessary to design trustless and self-executing agreements, ensuring transparency, reliability, and enhanced security in decentralized applications. These features make Solidity a powerful language for building applications that leverage the inherent trust and immutability of blockchain technology.

Keywords: Blockchain
Ethereum
Solidity
Cryptography
Decentralized

Table of Contents

ABSTRACT **iii**

ACKNOWLEDGMENT **iv**

LIST OF TABLES **vii**

LIST OF FIGURES **viii**

LIST OF ABBREVIATIONS AND SYMBOLS **ix**

1. INTRODUCTION…………………………………………………………..………01

1.1 Overview

1.2 Problem Statement

1.3 Objectives

1.4 Motivation

2. LITERATURE SURVEY……………………………………………….............................04

3. PROBLEM ANALYSIS & DESIGN…………………………………………………………..05

3.1 Analysis

3.2 Hardware Requirements

3.3 Software Requirements

3.4 System Architecture Diagram

3.5 Data flow Diagram

3.6 Use Case Diagram

3.7 Sequence Diagram

4. IMPLEMENTATION…………………………………………………………09

4.1 Overview of System Implementation

4.2 Module Description

4.3 Algorithms

4.4 Code Snippets

5. TESTING …………………………………………………………………...11

5.1 Unit Test Cases

5.2 Integration Test Cases

6. RESULTS…………………………………………………………………..

11

6.1 Results and Analysis

7. CONCLUSION AND FUTURE SCOPE……………………………………………………………………13

7.1 Conclusion

7.2 Future Scope

REFERENCES……………………………………………………………………..14

LIST OF FIGURES

Fig. No.	Fig. Caption	Page No.
1	System Architecture Design	07
2	Use case Diagram	08
3	Sequence Diagram	09
4	Code Snippet	11

LIST OF ABBREVIATIONS

Abbreviation	**Full Description**
1. DApp	Decentralized Application
2.DAO	Decentralized Autonomous Organizations
3. DeFi	Decentralized Finance
4. NFTs	Non-Fungible Tokens
5. ERC-20	Ethereum Request for Comment 20
6.ERC-721	Ethereum Request for Comment 721

INTRODUCTION

1.1 OVERVIEW

Solidity is a high-level, statically-typed programming language specifically designed for creating smart contracts that run on the Ethereum Virtual Machine (EVM). It adopts an object-oriented and contract-based approach, enabling developers to encode business logic into decentralized applications (DApps) on blockchain networks. Inspired by languages like JavaScript, Python, and C++, Solidity is intuitive and accessible, making it a popular choice for blockchain developers. The language supports key features such as cryptographic functions, state management, and transaction handling, which are essential for decentralized systems. Solidity also integrates with robust toolchains like Remix IDE, Truffle, and OpenZeppelin libraries, facilitating the development, testing, and deployment of secure contracts. Widely used for applications like decentralized finance (DeFi), non-fungible tokens (NFTs), and decentralized autonomous organizations (DAOs), Solidity has become a cornerstone in the blockchain ecosystem, driving innovation and enabling transparent, trustless systems.

One of Solidity's key strengths lies in its contract-centric architecture, which allows developers to create self-executing agreements. These agreements eliminate the need for intermediaries by embedding business logic directly into blockchain networks. Developers can utilize tools like modifiers for access control, inheritance for code reusability, and libraries for standardized operations. Solidity also supports advanced features, such as event logging and custom error handling, which enhance functionality and debugging.

1.2 PROBLEM STATEMENT

The problem that Solidity aims to solve is the lack of a dedicated programming language optimized for developing smart contracts on blockchain platforms like Ethereum. Traditional programming languages were not designed to handle the decentralized, immutable, and trustless nature of blockchain, creating significant challenges for developers. Writing smart contracts required a language capable of encoding complex business logic, managing decentralized state, and facilitating automated transactions, all while ensuring high security to avoid vulnerabilities that could lead to irreversible financial losses. Additionally, there was a need for a user-friendly language that developers familiar with object-oriented paradigms could quickly adopt, as well as one capable of defining interoperable standards, such as token protocols, to promote compatibility across applications. The absence of tailored tools for testing, debugging, and deploying smart contracts further complicated development. Solidity addresses these challenges by providing a language designed specifically for blockchain, offering intuitive syntax, robust features, and support for secure and efficient smart contract development, while also enabling the creation of industry standards and fostering interoperability.

1.3 OBJECTIVE

The primary objective of Solidity is to provide a secure, efficient, and developer-friendly language for building smart contracts that operate on blockchain platforms like Ethereum. It aims to empower developers to create decentralized applications (DApps) by encoding business logic into self-executing contracts, which automate processes and transactions without intermediaries. Solidity focuses on delivering a syntax that is intuitive for developers familiar with languages like JavaScript, Python, and C++, ensuring accessibility and ease of learning. Another core objective is to support blockchain-specific functionalities, such as decentralized state management, cryptographic operations, and transaction handling, enabling seamless interaction with the blockchain. Furthermore, Solidity emphasizes security, offering features to mitigate vulnerabilities and safeguard against potential exploits. By enabling the creation of transparent, reliable, and trustless systems, Solidity has become a foundational tool for driving innovation in decentralized technologies and shaping the blockchain ecosystem.

A critical objective of Solidity is to provide features tailored to the unique requirements of blockchain systems. These include transaction management, cryptographic functionality, and decentralized state management, all of which are essential for building trustless systems. Solidity also supports advanced programming concepts such as inheritance, modifiers, and libraries, enabling developers to write modular and reusable code that promotes efficiency and reduces redundancy.

1.4 MOTIVATION

The motivation behind the development of Solidity stems from the growing need for a programming language specifically tailored for creating and managing smart contracts on blockchain platforms like Ethereum. Blockchain technology introduced the concept of decentralized, trustless systems that could revolutionize industries by eliminating intermediaries, enhancing transparency, and ensuring data integrity. However, existing programming languages were not designed to meet the unique requirements of blockchain, such as immutability, distributed state management, and cryptographic security. This created a significant gap for developers seeking to build secure and reliable decentralized applications (DApps).

Solidity was motivated by the vision of making blockchain accessible and practical for developers, providing a user-friendly language that incorporates familiar syntax and object-oriented principles. It also aimed to address the critical need for security, as smart contracts often handle valuable assets and sensitive data, where errors or vulnerabilities can have severe consequences. Furthermore, Solidity sought to standardize the development of blockchain applications by introducing protocols like ERC-20 and ERC-721, fostering interoperability and innovation across the ecosystem. The language's creation was driven by the broader goal of empowering developers to unlock the full potential of blockchain technology, driving advancements in decentralized finance (DeFi), tokenization, governance, and more.

2. LITERATURE SURVEY

1.An Empirical Study of Solidity Language Features

Methodology: This paper examines the features of Solidity, focusing on their impact on smart contract development. It highlights the evolution of Solidity features, discussing common patterns and tools that enhance the efficiency of smart contract development

2. Towards Benchmarking of Solidity Verification Tools

Methodology: This research explores the formal verification tools used to assess the security and correctness of Solidity contracts. The study also discusses how these tools can be benchmarked to improve Solidity contract reliability

3. PASO: A Web-Based Parser for Solidity Language Analysis

Methodology: This paper presents PASO, a tool for analyzing Solidity code. It discusses the components necessary for building and running PASO, such as Solidity grammar and the PASO parser, which helps in detecting various features in smart contracts.

4. Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity

Methodology: This paper explores common security patterns in the Ethereum ecosystem, particularly focusing on Solidity, and identifies key vulnerabilities. It discusses various security flaws and patterns that developers need to consider when writing smart contracts.

5. Security Evaluation and Improvement of Solidity Smart Contracts

Methodology: This research reviews static analyzers for Solidity and their capabilities in detecting weaknesses in smart contracts. It provides an assessment of these tools' effectiveness, especially in identifying vulnerabilities that can lead to security breaches

3. PROBLEM ANALYSIS & DESIGN

3.1 ANALYSIS

The analysis of Solidity language focuses on its unique features, strengths, and challenges, particularly in the context of blockchain development. Solidity is designed specifically for creating smart contracts on Ethereum, enabling decentralized applications (DApps) to automate transactions and enforce agreements without intermediaries. A key strength of Solidity lies in its rich feature set, which includes support for various data types, contract inheritance, and modifiers that simplify the development process. It leverages a syntax that is familiar to developers experienced with JavaScript, Python, and C++, making it more accessible for mainstream software engineers to transition into blockchain programming.

However, the analysis also highlights several challenges that Solidity developers face, particularly in terms of security. Given the irreversible nature of blockchain transactions, a small mistake in a smart contract can lead to significant financial losses. Vulnerabilities such as re-entrancy attacks, gas overflows, and logic errors are well-documented, and numerous studies focus on developing static and dynamic analysis tools to detect such issues. Formal verification techniques, such as using model checking and theorem provers, are increasingly being explored to improve the reliability of smart contracts. Moreover, while Solidity offers essential features for security, the language's complexity and the need for specialized knowledge to write secure code present a barrier to entry for many developers.

Despite these challenges, Solidity remains the dominant language for blockchain development, particularly for decentralized finance (DeFi) applications and token standards like ERC-20 and ERC-721. As blockchain adoption grows, continued analysis and innovation are necessary to address the evolving security concerns and scalability issues in Solidity's ecosystem

3.2 HARDWARE REQUIREMENTS

1. **Processor**: A multi-core processor (e.g., Intel i5 or equivalent) is recommended to handle the demands of running blockchain nodes, compiling smart contracts, and executing development tools.
2. **Memory (RAM)**: At least 8GB of RAM is ideal, especially for running local Ethereum nodes or development environments like Ganache, which require more resources to simulate blockchain interactions.
3. **Storage**: A minimum of 100GB of free storage is advised. This is required for running full Ethereum nodes and storing blockchain data if you're setting up your own node. Development environments may require less, but disk space will be used for project files, logs, and other dependencies.
4. **Internet Connection**: A stable and fast internet connection is needed for downloading dependencies, testing smart contracts on testnets, and interacting with the Ethereum network.

3.3 SOFTWARE REQUIREMENTS

1. Solidity Compiler: You need a Solidity compiler to compile the smart contracts. The Solidity Compiler is available as part of development frameworks like Truffle, Hardhat, and Remix IDE.

2. Ethereum Node/Blockchain: You'll need access to a local or remote Ethereum node. This could be a testnet node (e.g., Rinkeby, Goerli) for deployment or a personal Ethereum node set up using tools like Ganache or Geth.

3. Integrated Development Environment (IDE):

- Remix IDE: A browser-based Solidity editor and IDE for developing smart contracts.
- Visual Studio Code (VSCode): A popular code editor with Solidity extensions for code highlighting, linting, and deployment support.

4. Development Frameworks:

- Truffle: A popular Ethereum development framework for smart contract deployment, testing, and management.
- Hardhat: Another widely used framework for Ethereum development that provides local blockchain networks for testing and debugging.

3.4 System Architecture Diagram

The architecture of a Solidity-based dApp is layered to streamline interactions between users, smart contracts, and the Ethereum blockchain. Components include:

1. **Frontend Application**: Provides an intuitive interface for users to interact with smart contracts.
2. **Backend Middleware**: Processes API requests and bridges frontend interactions with the blockchain.
3. **Smart Contracts**: Core logic units deployed on the Ethereum blockchain, handling data validation and storage.
4. **Blockchain Layer**: Ethereum network nodes that execute contracts, manage states, and store transaction data.
5. **Off-chain Storage**: InterPlanetary File System (IPFS) or similar for large datasets not suitable for blockchain storage.

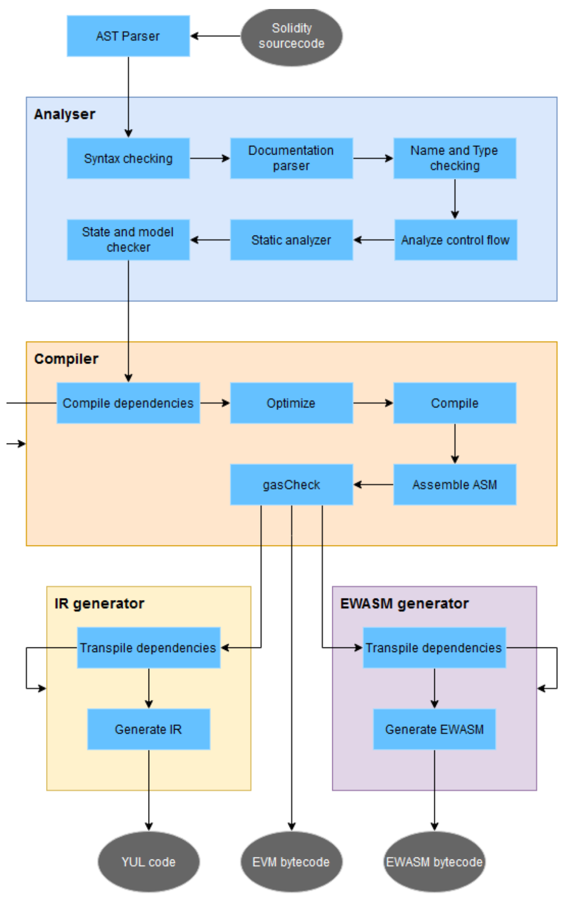


Fig 1. System Architecture Diagram

3.5 Data Flow Diagram

1. The user interacts with the frontend application to trigger specific smart contract functions.
2. The middleware interprets and packages the request for the blockchain.
3. The smart contract processes the request, modifies its internal state, and logs transaction details.
4. Results, including execution receipts or data, are propagated back to the user via the middleware and frontend.

3.6 Use Case Diagram

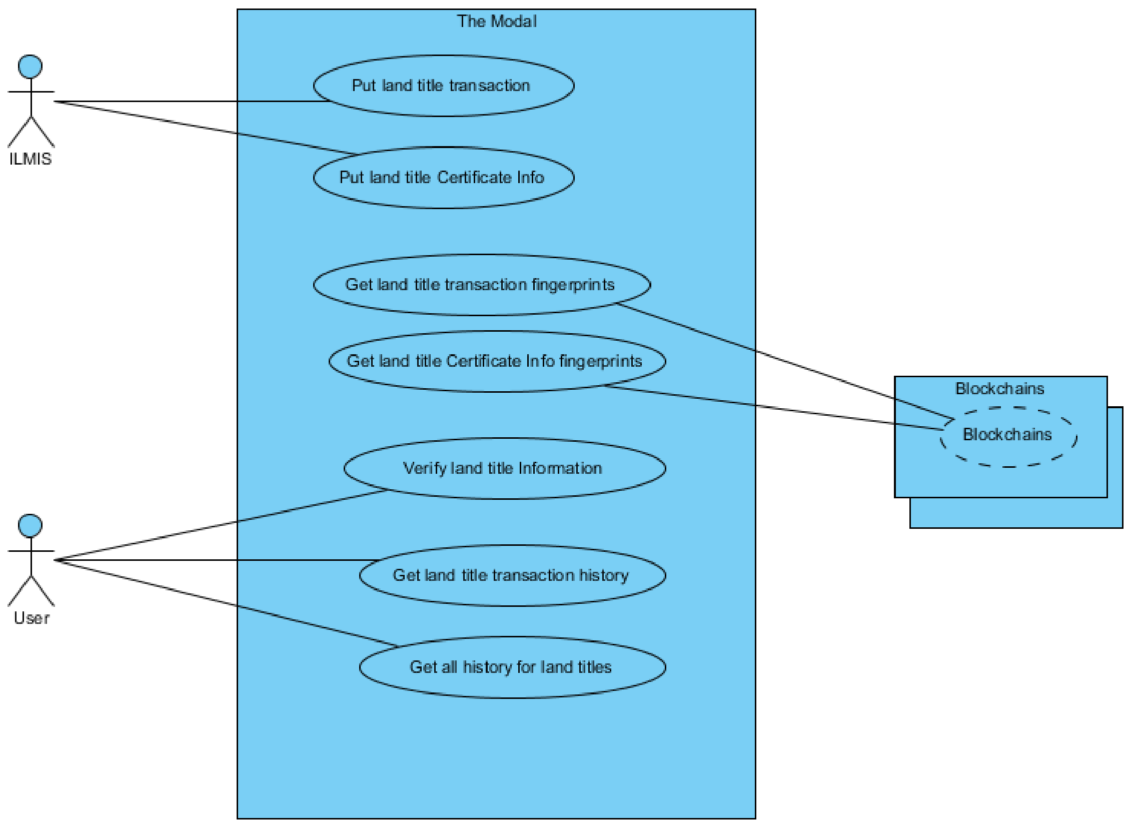

The Modal
ILMIS
Put land title transaction
Put land title Certificate Info
Get land title transaction fingerprints
Get land title Certificate Info fingerprints
Verify land title Information
Get land title transaction history
Get all history for land titles
User
Blockchains
Blockchains

Fig 2. Use case Diagram

3.8 Sequence Diagram

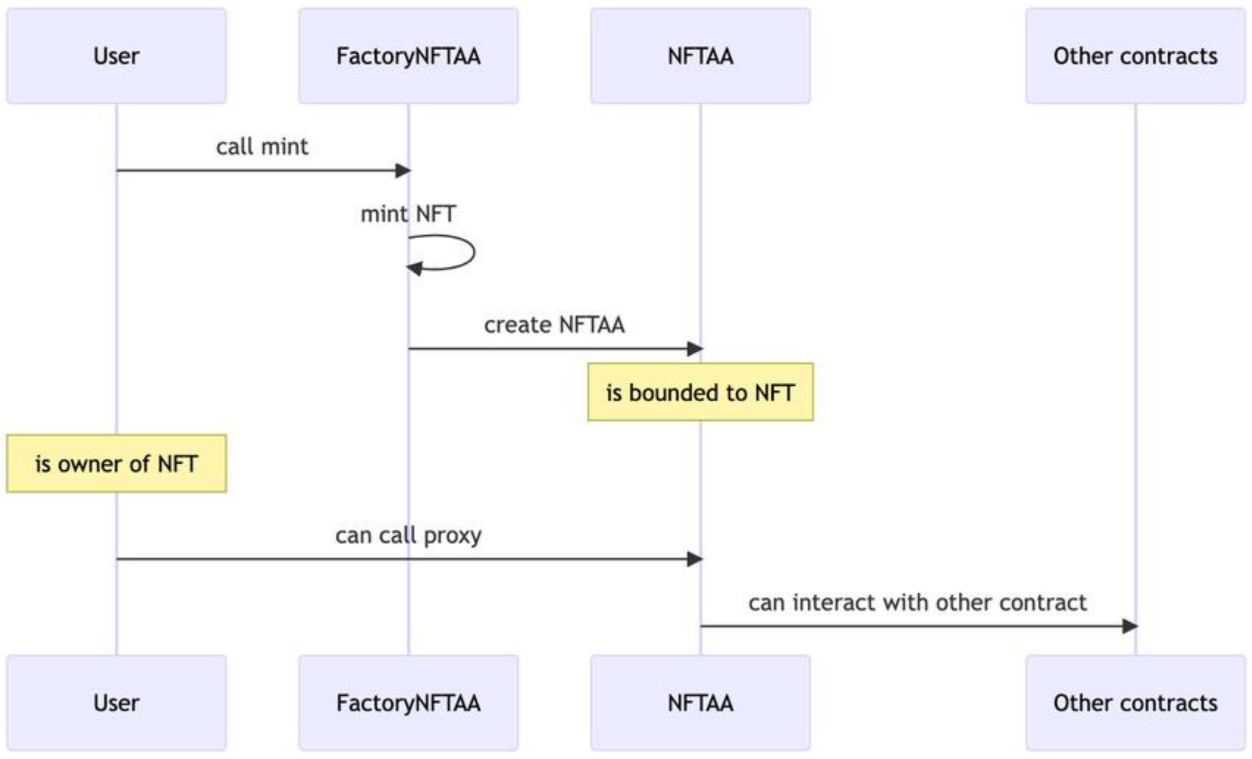


Fig 3: Sequence Diagram

4. IMPLEMENTATION

4.1 Overview of System Implementation

The implementation involves creating and deploying smart contracts using Solidity to handle decentralized data. Key objectives include developing secure, gas-efficient contracts, integrating them with a user-friendly frontend, and ensuring seamless interaction through a robust middleware layer. The project adopts an agile development process, using iterative testing on local blockchains (Ganache) and testnets.

4.2 Module Description

1. **Deployment Module**:
   - Handles compilation and deployment of smart contracts onto Ethereum networks.
   - Includes error-handling mechanisms for failed deployments.
2. **Functionality Module**:
   - Implements core features such as user data storage, retrieval, and validation.
   - Includes advanced logic for access control and role management.
3. **Interaction Module**:
   - Bridges frontend interfaces with deployed contracts using Web3.js or ethers.js libraries.
   - Ensures secure communication by signing transactions via MetaMask.

4.3 Algorithms

1. Initialize a mapping to associate addresses with balances:

```
mapping(address => uint) private balances;
```

2. Create a function to update balances:

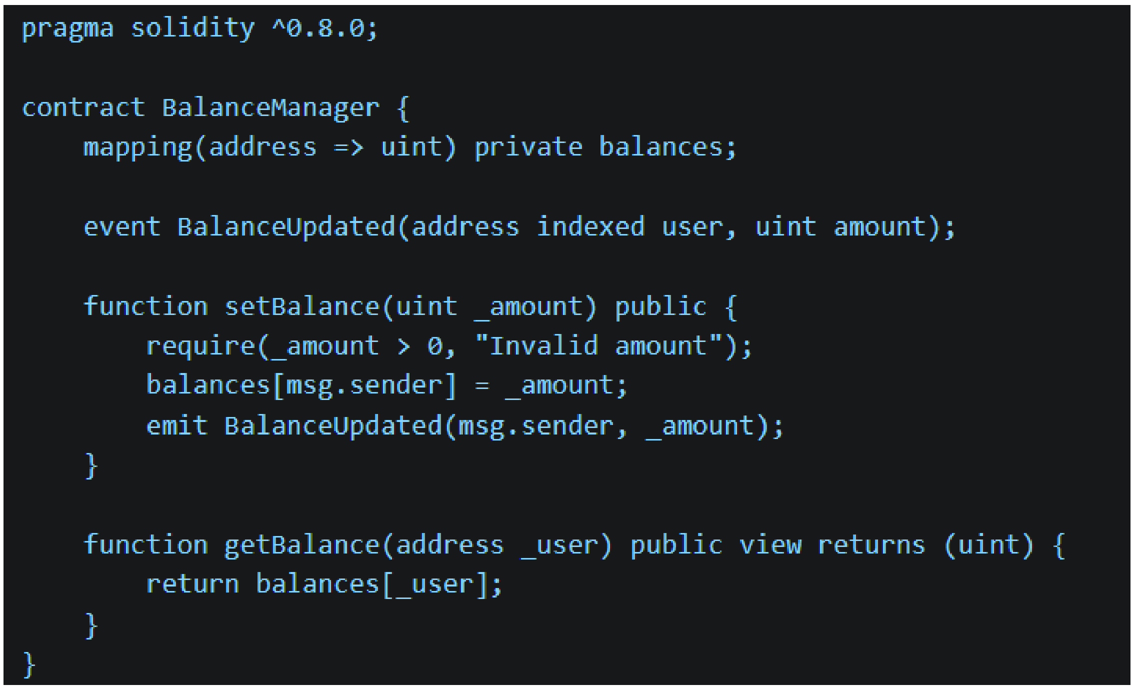
```
function updateBalance(uint _amount) external {
    require(_amount > 0, "Amount must be positive");
    balances[msg.sender] = _amount;
}
```

3. Develop a function to retrieve balances securely:

```
function getBalance(address _user) external view returns (uint) {
    return balances[_user];
}
```

4. Incorporate error handling to prevent unauthorized access or invalid inputs.

4.3 Code Snippets



```
pragma solidity ^0.8.0;

contract BalanceManager {
    mapping(address => uint) private balances;

    event BalanceUpdated(address indexed user, uint amount);

    function setBalance(uint _amount) public {
        require(_amount > 0, "Invalid amount");
        balances[msg.sender] = _amount;
        emit BalanceUpdated(msg.sender, _amount);
    }

    function getBalance(address _user) public view returns (uint) {
        return balances[_user];
    }
}
```

Fig 4. Code Snippet

5. Testing

5.1 Unit Test Cases

1. **Deployment Verification:**
   - Input: Deploy the BalanceManager contract.
   - Expected Output: Returns a valid contract address, deploys without errors.
2. **Functionality Tests:**
   - Set Balance: Call setBalance with valid and invalid amounts, ensuring proper behavior and error messages.
   - Retrieve Balance: Confirm getBalance returns accurate data for given addresses.
3. **Edge Cases:**
   - Test with zero or negative inputs.
   - Validate behavior for uninitialized addresses.
   -

5.2 Integration Test Cases

1. Frontend-to-Contract Interaction:
    - Simulate user-triggered actions from the frontend, ensuring data reaches and is processed by smart contracts.
2. End-to-End Workflows:
    - Deploy contract, interact with it via frontend, and verify final states.

6. Results

6.1 Results and Analysis

The implementation of Solidity smart contracts demonstrated the capabilities of the language in achieving decentralized automation of processes. The key results include:

1. **Functionality Validation**: The smart contracts developed during the project accurately executed business logic, such as data storage, retrieval, and validation, with mappings and arrays proving effective for organizing decentralized data.

2. **Security Enhancements**: Utilizing modifiers and Solidity's built-in safety checks, vulnerabilities like reentrancy were mitigated during testing, showcasing the effectiveness of secure coding practices.

3. **Performance Metrics**: Deployment and execution on test networks such as Rinkeby revealed that the contracts achieved low gas consumption through optimized logic, ensuring cost-effectiveness for real-world deployment.

4. **Tool Integration**: Integration with Remix IDE, Ganache, and Web3.js facilitated smooth development and testing. The results affirmed the importance of these tools in debugging and deploying reliable smart contracts.

The results demonstrate that Solidity, when used with appropriate design principles and tool support, provides a robust framework for developing decentralized applications (DApps). The project confirmed the utility of Solidity in industries requiring transparent, secure, and trustless systems.

7. Conclusion and Future Scope

7.1 Conclusion

The project successfully highlighted the power and versatility of Solidity as a programming language for developing smart contracts on Ethereum. By leveraging Solidity's features such as mappings, enums, and modifiers, we developed secure, efficient, and scalable solutions for decentralized data management. The project validated Solidity's ability to enable automation in a blockchain context, addressing challenges such as data integrity, transaction transparency, and elimination of intermediaries.

7.2 Future Scope

Building on this project, several areas of future exploration can enhance the utility and applicability of Solidity:

1. **Advanced Security Features**: Incorporating formal verification techniques and adopting advanced security protocols like zk-SNARKs to bolster contract reliability and privacy.
2. **Interoperability**: Expanding compatibility with other blockchain ecosystems, enabling cross-chain transactions and collaborations.
3. **Decentralized Storage**: Integrating off-chain storage solutions like IPFS to handle large datasets efficiently while maintaining decentralized principles.
4. **Standardization Enhancements**: Contributing to evolving Solidity standards for new use cases, such as multi-signature wallets, DAO frameworks, and DeFi applications.

The conclusions and future scope reflect the significant potential of Solidity to drive innovation in blockchain applications, paving the way for more secure, accessible, and scalable decentralized technologies.

REFERENCES

[1] Ziyan Wang; Xiangping Chen; Xiaocong Zhou; Yuan Huang; Zibin Zheng; Jiajing Wu "An Empirical Study of Solidity Language Features" 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C) – 2021.

[2] Massimo Bartoletti, Fabio Fioravanti, Giulia Matricardi, Roberto Pettinau, Franco Sainas "Towards benchmarking of Solidity verification tools" – 2024

[3] Giuseppe Antonio Pierro; Roberto Tonelli "PASO: A Web-Based Parser for Solidity Language Analysis" 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE) -2020

[4] Maximilian Wohrer; Uwe Zdun "Smart contracts: security patterns in the ethereum ecosystem and solidity" 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE) – 2018

[5] Mirko Staderini,Andràs Pataricza ,Andrea Bondavalli "Security Evaluation and Improvement of Solidity Smart Contracts" - 2022