Continuous Big Data Integrity Checking in Decentralized Storage using Blockchain and Smart Contracts

Ayoub Zaroual
Hamza Erraji

January 8, 2024

Contents

1 Introduction

In this project, we aimed to design and implement an efficient system for ensuring continuous data integrity in decentralized storage powered by blockchain and smart contracts. The system performs continuous checks on the stored data using the transparency and immutability features of blockchain.

2 Decentralized Storage

Decentralized storage refers to the distribution of data across multiple nodes, eliminating the need for a centralized server. It enhances security, accessibility, and reliability by removing a single point of failure.

3 Blockchain Technology

Blockchain, a decentralized and distributed ledger, is at the core of this project. It ensures data immutability and transparency by storing data in blocks linked through cryptographic hashes. This technology provides a tamper-proof and secure foundation for our decentralized storage system.

4 Smart Contracts

Smart contracts are self-executing contracts with the terms of the agreement directly written into code. In our project, smart contracts on the Ethereum blockchain handle the continuous checking of data integrity.

5 Architecture

The project architecture consists of:

- **Decentralized Storage Layer:** Utilizing a distributed file system for storing large datasets.
- **Blockchain Layer:** Employing Ethereum blockchain for immutability and transparency.
- **Smart Contracts:** Executing on-chain logic for continuous data integrity checks.

6 Implementation

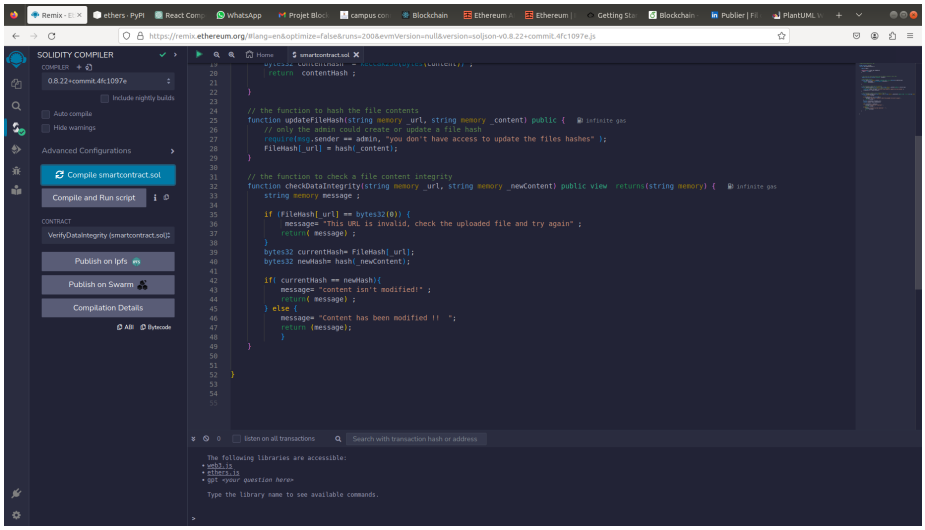
6.1 Smart Contract - VerifyDataIntegrity.sol

```
//SPDX-License-Identifier: MIT

pragma solidity ^0.8.19;

contract VerifyDataIntegrity {

    address public admin;

    // Le constructeur est appelé une seule fois à la déploiement du contrat
    constructor() {
        // Le créateur du contrat (celui qui le déploie) devient l'administrateur
        admin = msg.sender;
    }

    // Fonction interne pour calculer le hash d'une chaîne de caractères
    function hash(string memory content) internal pure returns (bytes32) {
        bytes32 contentHash = keccak256(bytes(content));
        return contentHash;
    }

    // Fonction pour mettre à jour le hash d'un fichier
    function updateFileHash(string memory _url, string memory _content) public {
        // Seul l'administrateur peut créer ou mettre à jour le hash d'un fichier
        require(msg.sender == admin, "you don't have access to update the files hashes");
        FileHash[_url] = hash(_content);
    }

    // Fonction pour vérifier l'intégrité du contenu d'un fichier
    function checkDataIntegrity(string memory _url, string memory _newContent) public view r
        string memory message;

        // Vérifier si le hash associé à l'URL existe (non nul)
        if (FileHash[_url] == bytes32(0)) {
            message = "This URL is invalid, check the uploaded file and try again";
            return message;
        }

        bytes32 currentHash = FileHash[_url];
        bytes32 newHash = hash(_newContent);

        // Comparer les hashs pour déterminer si le contenu a été modifié
        if (currentHash == newHash) {
```

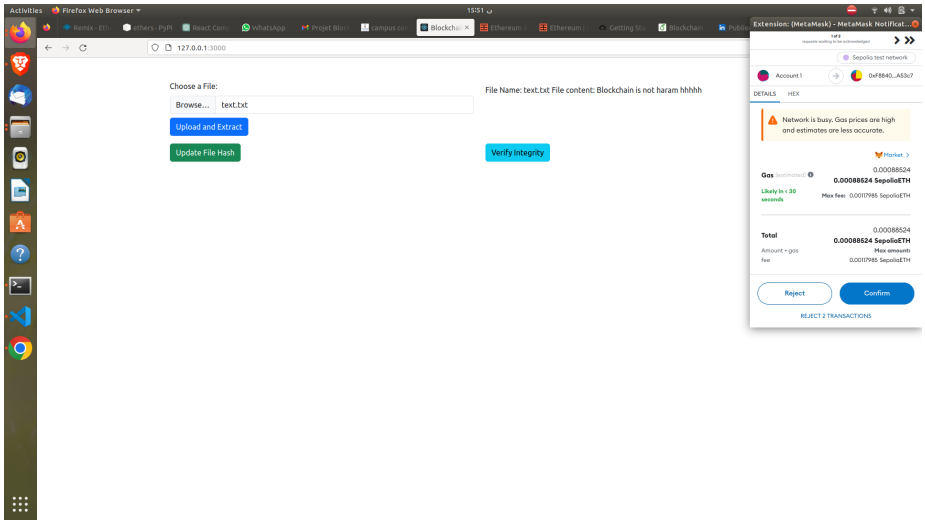
```
            message = "content isn't modified!";
            return message;
        } else {
            message = "Content has been modified !";
            return message;
        }
    }

    // Le mapping pour stocker et extraire les hashs des fichiers en fonction de leurs URLs
    mapping (string => bytes32) internal FileHash;

}
```

6.2 React User Interface - App.js

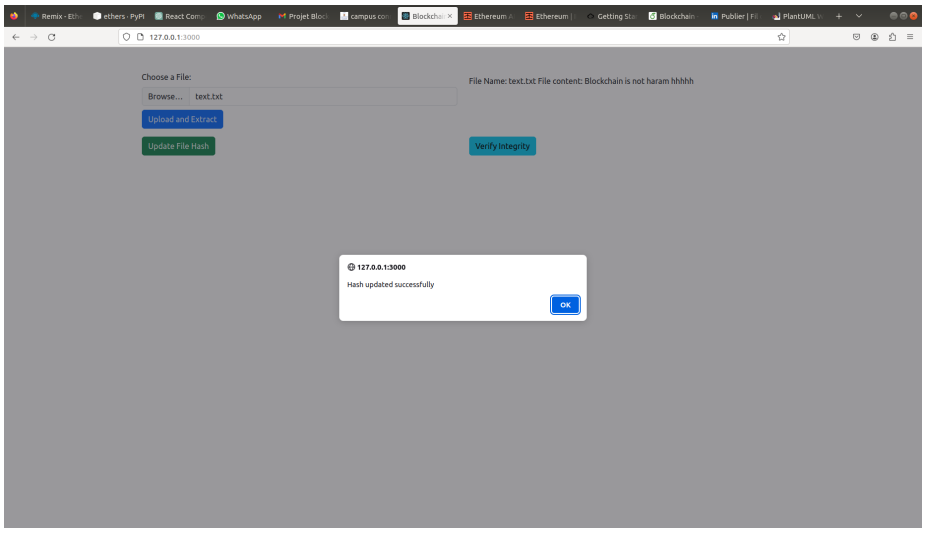
6.2.1 deploy smart contract in remix ide



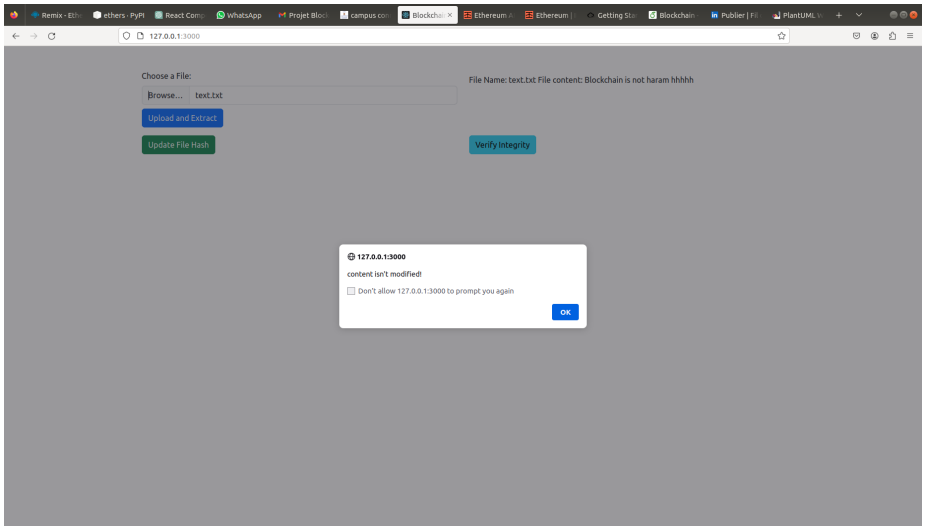
6.2.2 Upload file



6.2.3 Update file hash



6.2.4 verify integrity



7 GitHub Repository

The project source code and documentation are available on our GitHub repository.

`https://github.com/ErHamza/blockchaine`

8 Installation Guide

To set up and run the project locally, follow these steps:

1. Clone the repository: `git clone https://github.com/ErHamza/blockchaine.git`
2. Install dependencies: `npm install`
3. Start the application: `npm start`

9 Conclusion

In conclusion, this project successfully demonstrates the integration of decentralized storage, blockchain, and smart contracts to achieve continuous big data integrity checking. The decentralized and transparent nature of the system enhances security and reliability in data storage. Further improvements and optimizations can be explored for real-world deployment.