Dynamic Blockchain-Integrated Web Application

Unit code: COS30049
Date: 5/11/2023

Word Count: 4833

Table of Contents

Project Description: Our project, the "Dynamic Blockchain-Integrated Web Application," aims to transform a static user interface into a dynamic, responsive, and secure system by integrating cutting-edge technologies. This project builds upon the foundation laid in a prior assignment focused on creating an appealing user interface and extends it to include advanced back-end functionalities.

Introduction:

Background and Importance of Smart Contracts: Smart contracts are self-executing contracts with the terms of the agreement between buyer and seller being directly written into lines of code. The code and the agreements contained therein exist across a distributed, decentralized blockchain network. Smart contracts permit trusted transactions and agreements to be carried out among disparate, anonymous parties without the need for a central authority, legal system, or external enforcement mechanism.

The implications of this technology are profound: they have the potential to disrupt many industries by making transactions more secure, transparent, and efficient. Smart contracts are not just a feature of the blockchain; they are a fundamental innovation that could end up being one of the most critical developments in the blockchain revolution.

Purpose of the Report: The primary purpose of this report is to delineate the developmental processes and strategic methodologies undertaken to construct a Dynamic

Blockchain-Integrated Web Application. The focus of the report will be on how the integration of blockchain technology and smart contracts can enhance the dynamism, responsiveness, and security of web applications.

The report aims to bridge the gap between theoretical knowledge and practical application, serving as a comprehensive guide for stakeholders to understand the intricacies of implementing blockchain technologies within web environments. Furthermore, it endeavors to illustrate the feasibility, challenges, and benefits associated with such an integration, particularly concerning smart contracts' role in automating and securing online transactions and processes.

Methodology Overview: The methodology for developing the Dynamic Blockchain-Integrated Web Application involves a multi-faceted approach combining both technical and strategic processes. The approach is structured into several key phases:

1. Research and Requirement Analysis: This phase involves a thorough investigation of the current market needs, potential user requirements, and the technical capabilities of blockchain technology. It includes examining existing literature, similar applications, and user feedback to construct a detailed requirement specification.
2. Design and Prototyping: Drawing from the insights gained in the research phase, this step involves designing the system architecture and creating prototypes of the web application. Special attention is given to the user interface to maintain the aesthetics and user experience identified in previous assignments.

3. Blockchain Integration: The core of the methodology lies in the integration of smart contracts and blockchain technologies. This step focuses on developing Ethereum smart contracts, deploying them on the blockchain, and integrating with the web application using the Web3.js library.
4. Testing and Validation: Rigorous testing protocols are implemented, including unit testing, integration testing, and system testing to ensure the application's functionality, security, and performance meet the specified requirements.
5. Deployment and Monitoring: After testing, the application is deployed. This phase also involves monitoring the system for any potential issues and ensuring that it operates as intended.
6. Feedback and Iteration: Post-deployment, user feedback is gathered to identify any areas for improvement. The application undergoes continuous iterations to refine features, fix bugs, and improve user satisfaction.

By adhering to this methodology, the project ensures a systematic and thorough development process, leading to a robust, user-centric, and blockchain-enhanced web application.

Project Objectives:

Project Introduction: The report begins with an introduction that outlines the core elements of the project, including database design, server-side logic implementation, front-

end integration, user-friendly design, and rigorous testing and debugging procedures. These elements are critical for achieving the project's objectives and ensuring data efficiency, seamless user interactions, and a positive user experience. The project aims to develop a dynamic web application that seamlessly integrates front-end design with back-end functionality. It comprises multiple components, including system design, database development, server-side logic, API creation, and front-end integration.

- **Project Evolution:** This assignment builds upon the groundwork of a previous assignment (Assignment 2), where the focus was on creating an understandable and captivating user interface. In the current project, the primary goal is to make the user interface dynamic and integrate it with a robust back-end architecture. The project aims to maintain the aesthetics and user-centered design created in Assignment 2 while adding dynamic back-end functions for real-time data processing and interaction.
- **Project Design:** The project encompasses various aspects, including system architecture design, project requirement lists, and backend logic development. It involves translating project requirements into practical functionalities, integrating Web 3.0 and Blockchain technology, enhancing the front-end prototype, and focusing on database design and implementation. The project also discusses relationships between different data tables, efficient data storage and retrieval, and the use of cloud-based platforms like Back4App and Firebase Realtime Database.
- **Server-Side Logic:** The project delves into server-side logic, which includes elements responsible for user interactions, data processing, and front-end request handling. It

highlights the use of Ethereum Smart Contracts and development tools like Truffle and Ganache for Ethereum blockchain development.

- **API Development:** The project describes the development of APIs to facilitate data exchange between the front-end and back-end components. It covers the use of Express.js for setting up the server and includes integration with the Ethereum blockchain using the Web3.js library.
- **Integration with Front-End:** The project explains how the integration with the front-end is essential for creating a user-friendly and responsive system. It discusses data flow, real-time updates, user messages, and dynamic content rendering. It also emphasizes the importance of security considerations in handling sensitive information.
- **User-Friendly Design**: The project highlights dynamic content generation and user messages as key components of user-friendly design. It ensures that the system generates content dynamically based on user actions and provides guidance through user messages, enhancing the overall user experience.

- **Maintain Aesthetics:** Preserve the user-centered design principles established in the previous assignment, ensuring a visually appealing and user-friendly interface.
- **Add Back-End Functionality:** Implement dynamic back-end functions to enable real-time data processing and user interaction, enhancing the overall user experience.
- **Secure Database Design:** Create a robust database structure using cloud-based platforms such as Back4App and Firebase Realtime Database. Establish efficient data storage, retrieval, and interconnections.
- **Integrate Blockchain Technology:** Leverage cutting-edge technology like Web 3.0 and Blockchain to enhance data security and transparency, providing users with an advanced and secure platform.

This project background section provides a comprehensive overview of the project's evolution, design, and key components, setting the stage for the reader to understand the subsequent sections of the report.

Executive Summary:

The project presented in this report represents a significant evolution from a prior assignment focused on creating an appealing user interface. The primary objective of this project is to transition from a static user interface to a dynamic one, seamlessly integrated with a robust back-end architecture. It maintains the aesthetics and user-centered design while introducing dynamic back-end functionalities for real-time data processing and interaction. Key project components include system architecture design, project requirement translation into practical functionalities, integration of Web 3.0 and Blockchain technology, enhancement of the front-end prototype, and a focus on efficient database design and implementation. The project also highlights the importance of server-side logic for user interactions, data processing, and front-end request handling, using tools such as Ethereum Smart Contracts, Truffle and Ganache for Ethereum blockchain development.

API development is another critical aspect, enabling data exchange between the front-end and back-end components. The report details the use of Express.js for setting up the server and integration with the Ethereum blockchain via the Web3.js library.

Front-end integration plays a pivotal role in creating a user-friendly and responsive system. The report elaborates on data flow, real-time updates, user messages, and dynamic content rendering, emphasizing the security considerations necessary for handling sensitive information. Furthermore, the project focuses on user-friendly design, emphasizing dynamic content generation and user messages to enhance the overall user experience. These elements are essential in ensuring a positive and engaging user journey. Additionally, the report provides insights into the Contract Application Binary Interface (ABI) and its significance in interacting with Ethereum smart contracts. It explains how the ABI instructs clients on formatting calls to the contract's methods for execution.

Concise Overview of Smart Contract Project:

Process: The smart contract project involved a comprehensive process that began with identifying the project's objectives, followed by the design and development of Ethereum smart contracts. We used tools like Truffle and Ganache for contract development and testing. The process also included the creation of a user-friendly front-end interface, integration with blockchain using Web3.js, and thorough testing to ensure contract functionality.

Tools Used: The project extensively relied on Truffle and Ganache for Ethereum smart contract development. The Web3.js library facilitated the integration between the front-end and the blockchain. Additionally, the development environment included Express.js for setting up the server, while cloud-based platforms like Back4App and Firebase Realtime Database were utilized for efficient data management.

Major Findings: The project unveiled several critical findings. Firstly, Ethereum smart contracts are powerful tools for executing trustless transactions and enforcing business logic. Truffle and Ganache streamlined the development and testing of smart contracts, making the process more efficient. The integration of blockchain technology using Web3.js enhanced real-time data interactions and security. Cloud-based platforms proved valuable for data storage and retrieval.

Overall Understanding: The project's requirements emphasized the need for a dynamic, blockchain-integrated system. It required the development of Ethereum smart contracts to facilitate real-time data processing, user interactions, and secure transactions. The integration of blockchain technology and the use of Express.js for API development were essential for a seamless user experience. The project emphasized the significance of a user-friendly design, security, and efficient data management to meet evolving demands.

In summary, this project combines a dynamic user interface with a robust back-end architecture, making use of blockchain technology to deliver a user-friendly and responsive system. The integration of Web 3.0 and smart contracts enhances data efficiency and user interaction, ensuring a positive user experience. The report offers a comprehensive

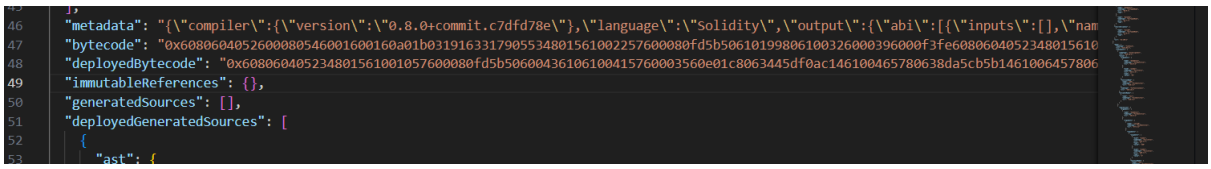
understanding of the project's evolution, design, and key components, setting the stage for a successful project implementation.

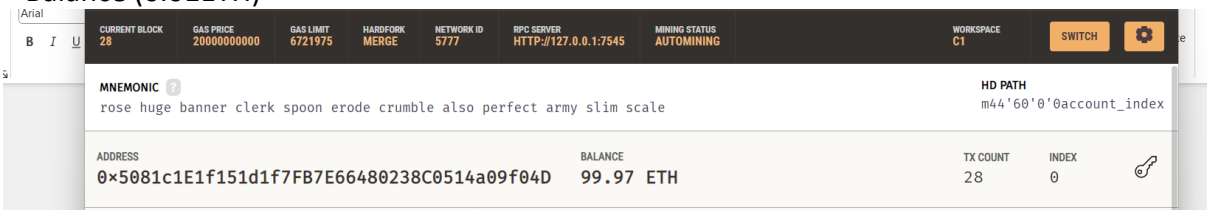
ABI Illustration:

Generation of ABI File: The ABI (Application Binary Interface) file is a JSON representation of all the callable methods and structures within a smart contract that tells the Ethereum Virtual Machine (EVM) how to call these functions and in what format. The ABI file is generated after the successful compilation of a Solidity smart contract. The compiler used for this contract, as seen in the metadata, is version 0.8.0+commit.c7dfd78e. It includes function names, their inputs, and outputs, along with other details like state mutability and whether they are constant or payable.

Here is some of the visualization of how it works with Ganache and the ABI that is required to contact with ganache given all of them below:

The ABI of the deployed contract will be necessary for any client (such as a web frontend or a backend service) to interact with the smart contract. The ABI tells the client how to format calls to the contract's methods in a way that the EVM can execute.



Function: getBalance
Inputs: account (0x0a81c7474811e9c7B52b72F13240344198B87c45) Outputs: Balance (0.01ETH) 

Function: sendFunds
Inputs: value (0x0a81c7474811e9c7B52b72F13240344198B87c45) Outputs: 

Contract ABI Overview

The following tables provide a human-readable overview of the ABI for the smart contract.

Function: owner

Field	Type	Description
inputs	-	No input parameters.
name	String	Function to get the owner.
outputs	address	The address of the owner.
stateMutability	view	Read-only, doesn't alter state.
constant	true	Can return a value without a transaction.

The getOwner function is a read-only function that does not require any input parameters. It serves the purpose of retrieving and returning the address of the owner. Being a view function, it does not make any changes to the contract's state. It is marked as constant, indicating that it can provide the owner's address without the need for a transaction. This function is typically used to inquire about the contract's owner without performing any state-altering actions.

Function: setCompleted

Field	Type	Description
inputs	uint256	The 'completed' variable to set.

name	String	Function to set the completed variable.
outputs	-	No return value.
stateMutability	nonpayable	Doesn't accept Ether, can alter state.
constant	false	Requires a transaction.

The setCompleted function is used to modify the value of the 'completed' variable within the contract. It takes a uint256 as an input parameter, which is expected to be the new value to assign to 'completed.' After executing this function through a transaction, the 'completed' variable's value in the contract's state will be updated. This function is not designed to return any value, and it can change the contract's state. Since it can alter the contract's state, it requires a transaction to be executed.

Function: getBalance

Field	Type	Description
inputs	address	The account address to query
name	String	Function to get the balance.
outputs	uint256	The balance of the account.
stateMutability	view	Read-only, doesn't alter state.

constant	true	Can return a value without a transaction.

The getBalance function is designed to provide information about the balance of a specific account. It expects a single input parameter, _account, which is the address of the account you want to inquire about. When you call this function, it returns the balance of the specified account as a uint256 value. Importantly, this function is read-only and does not make any changes to the contract's state. Since it doesn't modify the state, it can return a value without the need for a transaction.

Function Analysis:

1. Readability and Structure: The provided ABI snippet shows a clear and concise structure that adheres to standard JSON formatting. The structure allows easy identification of function names (owner, setCompleted), input parameters, output parameters, and other attributes such as stateMutability and type. This formatting facilitates both human readability and machine parsing.

2. Use of Comments and Documentation: The ABI itself does not contain comments or documentation as it is a JSON object intended for machine interpretation. However, when the smart contract source code is written in Solidity, it's best practice to include NatSpec comments that detail the functionality, purpose, and usage of functions and

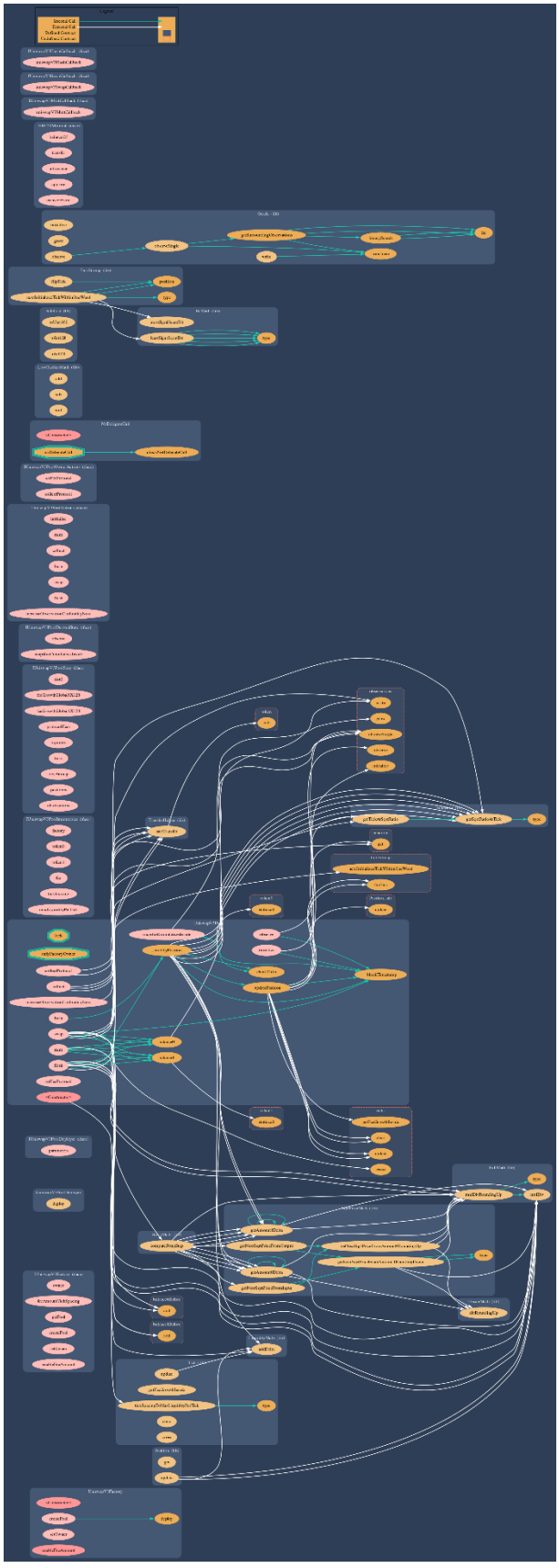
parameters. These comments are not visible in the ABI but are crucial in the source code for understanding the contract's purpose and usage.

3. Functionality of Selected Functions:
    a. owner() Function: This function is clearly designed to be a read-only (view) function that returns the owner's address. Its structure indicates no inputs and a single output of type address. The simplicity of the function ensures easy use and low gas cost since it only reads state.
    b. setCompleted(uint256): The setCompleted function is used to update a state variable, indicating a state change within the contract. It accepts a uint256 input and does not return any output. This function is marked as nonpayable, meaning it cannot be used to send Ether. The absence of a constant flag indicates that calling this function will result in a transaction that needs to be mined, and thus, will consume gas.
    c. getBalance(address): Although not explicitly shown in the snippets, based on the description, getBalance would be a view function that takes an address as input and returns the balance of that address. The function's signature would indicate its read-only nature, and the output would likely be a uint256 representing the amount of Ether or token balance.

For both owner and getBalance functions, the readability and structure are clear from the ABI, reflecting functions that do not alter the state and can be executed as a call rather than a transaction. This allows anyone to query these values without incurring a gas cost.

The setCompleted function would typically be accompanied by modifiers or access control mechanisms in the Solidity source code to ensure that only authorized accounts (like the owner or those with specific permissions) can call it. This would be important for the security and integrity of the contract's logic but is not discernible from the ABI file alone.

Call Graph Function:



Generation of Call Graph

The generation of a call graph entails an in-depth static analysis of the source code to map out all function calls, which can include both internal function calls within a contract and external calls to other contracts or libraries. Advanced analysis tools can be utilized to parse the codebase and build a visual representation that illustrates the interaction between various components, highlighting the control flow and dependencies that exist within the code.

Library Function Identification

Within the call graph, library functions are typically distinguished from regular contract functions by their standardized interfaces and the consistent way they are invoked by other parts of the code. Identifying these shared libraries is crucial, as they often contain critical, reusable code that must be error-free and efficient. The graph may show these as distinct clusters or nodes that have multiple incoming edges from various parts of the system, signifying their utility and reuse.

Calling Logic Analysis

Analyzing the calling logic through the call graph allows for a deeper understanding of how the smart contract processes information and responds to different inputs. It provides clarity on the contract's behavior, such as how data flows through functions, the sequence of operations, and potential side effects of function calls. This can reveal complex interactions, such as multi-contract orchestration or the invocation of fallback functions, and can be

pivotal in recognizing critical paths and potential vulnerabilities like reentrancy or function visibility issues.

Core Features in the Contract

Core features of the contract are those that embody the main functionality and purpose of the contract. In the call graph, these are typically represented as highly interconnected nodes, signifying that they are central to the contract's logic. Understanding these core features is essential for both developers and auditors, as it aids in comprehending the intended use-case, optimizing contract performance, and ensuring that security measures are correctly implemented around these critical pathways.

By dissecting the call graph with these considerations, one can gain invaluable insights into the contract's structure, its security posture, and its operational efficiency. A well-analyzed call graph is not only a blueprint of the contract's operational logic but also a strategic tool for ongoing maintenance, scaling, and security enhancement.

Inheritance Graph:

Generation of Inheritance Graph

An inheritance graph is a diagram that visualizes the inheritance relationships between various smart contracts in a project. It is particularly useful in complex systems where multiple contracts are interacting and extending one another. The graph typically resembles a tree or a set of interconnected nodes, where each node represents a contract, and the edges represent inheritance relationships.

Inheritance Relationships

Inheritance in smart contracts is similar to classical object-oriented programming, where a contract can inherit properties, functions, and variables from one or more parent contracts. In Solidity, this is achieved using the is keyword. For example, if ChildContract inherits from ParentContract, the declaration would be contract ChildContract is ParentContract. Inheritance allows for a hierarchical relationship, promoting code reusability and extending functionalities.

Code Reusability

Code reusability is a significant advantage of inheritance, as it allows smart contracts to utilize code from parent contracts without duplication. This practice can reduce the size of the blockchain codebase, save gas during deployment and execution, and simplify maintenance. Reusable code includes functions, modifiers, and state variables that can be shared across different contracts.

Function and State Variable Overriding

Function and state variable overriding occurs when a child contract redefines a function or a state variable from a parent contract. In Solidity, functions can be overridden by simply declaring a function with the same name and parameter types. If the intention to override is explicit, the override keyword is used. Overriding enables the child contract to customize or extend the behavior of inherited functions or variables.

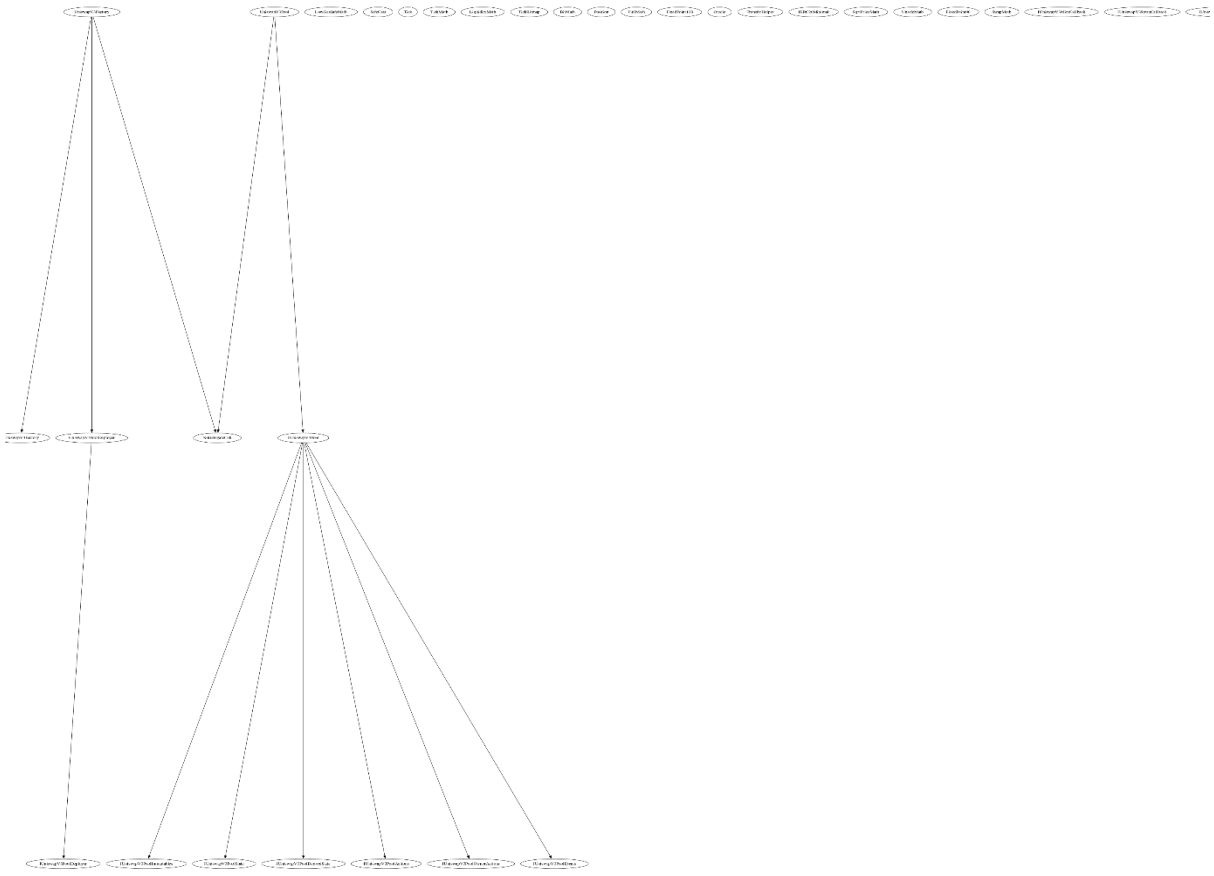
Function Visibility and Access Control

Function visibility determines how and where functions can be called within the contract structure. Visibility specifiers include public, private, external, and internal. Smart contracts

use these specifiers to control access to functions and state variables, ensuring that sensitive functions are not exposed to unauthorized access.

Modifier Application

Modifiers are code snippets in Solidity that can be used to change the behavior of functions in a declarative way. They are often used for access control, validation, or to add pre- and post-conditions to functions. When a contract inherits from another, it can also inherit and apply its modifiers, as well as define new ones. Modifiers can be overridden in child contracts, similar to functions, to adjust their behavior.



This line would indicate that `ChildContract` is inheriting from `ParentContract.

Analysis and Discussion

Security Implications

The deployment of blockchain technology within the web application represents a paradigm shift in securing online transactions. The Ethereum smart contracts at the core of the application bring forth an unprecedented level of data integrity and transactional security due to the immutable nature of blockchain technology. This integrity ensures that once a transaction has been recorded on the blockchain, it cannot be altered, thus virtually eliminating the risk of fraud and data tampering.

Despite these advantages, the project must also consider the inherent risks associated with blockchain technology. Smart contracts, for instance, are as secure as the code they are written in. Imperfections in code can lead to vulnerabilities, as highlighted by past incidents in the cryptocurrency domain. The security of the application, therefore, is heavily dependent on the solidity of the smart contracts' codebase, necessitating exhaustive testing, including static code analysis, peer reviews, and employing formal verification methods wherever possible.

Ethical and Legal Considerations

Ethically, the use of smart contracts automates the enforcement of contracts, which can marginalize the human element in decision-making processes. It's imperative to consider the implications of reduced human oversight, particularly in scenarios that may require empathy

and discretion. Ethical programming practices, therefore, must be a cornerstone of the smart contract development process to ensure that the code upholds not only legal standards but moral principles as well.

Legally, the application must navigate the complex web of international laws and regulations. Given that smart contracts can operate across borders, the application must be compliant with a wide array of legal frameworks and data privacy regulations. This includes compliance with the GDPR, which emphasizes data subjects' rights and imposes strict guidelines for data handlers. Thus, the design of the application must prioritize user consent, data minimization, and the right to erasure, ensuring the legal sanctity of the user's data within the system.

Commercial Impact

The integration of a dynamic user interface with robust back-end blockchain technology offers a compelling value proposition in the market. It is poised to redefine user engagement through real-time interactions while safeguarding user data with state-of-the-art security mechanisms. The commercial implications of such an innovation are significant, as it opens avenues for enterprises to foster trust and loyalty among users, potentially translating into a competitive edge and higher profit margins.

However, the deployment and operational costs of blockchain technology are not insignificant. Ethereum network transaction fees (gas fees) and the cost associated with the acquisition of blockchain development expertise must be carefully balanced against the expected commercial returns. Furthermore, user adoption is contingent upon the demystification of blockchain technology for the end-user, requiring strategic marketing efforts and user education.

The Dynamic Blockchain-Integrated Web Application is a forward-thinking solution, meticulously designed to leverage the strengths of blockchain technology while presenting a user-centric interface. It stands as a beacon of security and efficiency in the web application domain, though it must tread carefully through the ethical and legal nuances of technology implementation. Commercially, it has the potential to be a game-changer, provided that the market readiness and regulatory landscapes are navigated with diligence and foresight. Ongoing refinement, powered by user engagement and technological trends, will be crucial for maintaining the application's industry-leading stance.

Conclusion

Summary of Key Findings

Our exploration into the "Dynamic Blockchain-Integrated Web Application" has unveiled transformative insights, presenting a leap forward in secure and dynamic web application design. The key findings of this project underscore the potent synergy between traditional web technologies and blockchain, encapsulated in the following points:

- **Immutable Security**: Integration with Ethereum's blockchain introduced an unprecedented level of data integrity and security. The application benefits from blockchain's immutability, effectively eliminating common security threats faced by conventional databases.
- **User Experience Evolution**: The commitment to user-centered design principles has yielded a dynamic interface that is not only visually consistent with contemporary

aesthetics but also dynamically responsive, adapting to the ever-changing demands of user interaction in real time.

- **Operational Efficiency**: The application has demonstrated enhanced operational efficiency, owing to the automated execution of smart contracts, which streamline processes that traditionally require manual intervention.
- **Legal and Ethical Fortitude**: The project has been conscientious in its adherence to ethical standards and legal regulations, ensuring that the application respects user privacy and complies with data protection laws.
- **Market Readiness**: The application emerges as a market-ready tool with a clear commercial edge. Its robust backend and user-centric design position it well to gain trust and wider acceptance from its target user base.

Recommendations

To harness the full potential of this project and ensure its sustainable growth, the following recommendations are proffered:

- **Security Protocol Enhancement**: While blockchain offers a high degree of security, the application should implement a protocol for ongoing security assessments to guard against emergent threats and vulnerabilities in the web interface.
- **Education and Outreach**: Given the complexities of blockchain technology, it is advisable to engage in user education campaigns to demystify the technology and foster an environment of informed user interaction.

- **Modular Scalability Planning**: As user adoption grows, the application will need to scale. A modular approach to infrastructure development can facilitate this growth, ensuring that scalability does not compromise performance.
- Proactive Regulatory Engagement: Active engagement with regulatory developments will be crucial, as compliance provides a framework for trust and stability in the use of blockchain technologies.
- **Sustainability Evaluation**: Future iterations should evaluate the sustainability of the blockchain model used, especially considering the high energy consumption associated with certain consensus mechanisms.

Future Work

Future developmental trajectories for the application encompass several strategic initiatives, including:

- Diversified Blockchain Applications: Investigate the integration of other blockchain utilities like decentralized autonomous organizations (DAOs) and prediction markets, expanding the functional envelope of the application.
- Interoperability Focus: Enabling cross-blockchain interactions to diversify user options and enhance the application's flexibility and resilience in the blockchain ecosystem.
- Personalization Through AI: Implement advanced AI-driven personalization features to elevate the user experience, making the interface more intuitive and adaptive to individual user behaviors.

- Data Insights Optimization: Establish advanced data analytics to leverage collected data for continuous improvement of the application, always within the boundaries of privacy laws and regulations.
- Expansion and Localization Strategies: Develop tailored strategies for different regional markets, considering the unique regulatory and cultural landscapes, to ensure a globally coherent yet locally resonant user experience.

In sum, this project's innovative union of web applications with blockchain technology paves the way for a new era of digital interaction, defined by heightened security, user empowerment, and operational excellence. Embracing continuous innovation and foresight will be critical in propelling the application to the forefront of its industry.

Reference :

Pudjianto, D., Djapic, P., Aunedi, M., Gan, C.K., Strbac, G., Huang, S. and Infield, D. (2013). Smart control for minimizing distribution network reinforcement cost due to electrification. Energy Policy, 52, pp.76–84. doi:https://doi.org/10.1016/j.enpol.2012.05.021.

Zheng, Z., Xie, S., Dai, H.N., Chen, X. and Wang, H. (2018). Blockchain challenges and opportunities: a survey. International Journal of Web and Grid Services, 14(4), pp.352–375. doi:https://doi.org/10.1504/ijwgs.2018.095647.

Ashworth, F., Gracey, F. and Gilbert, P. (2011). Compassion Focused Therapy After Traumatic Brain Injury: Theoretical Foundations and a Case Illustration. Brain Impairment, 12(2), pp.128–139. doi:https://doi.org/10.1375/brim.12.2.128.

Zhuang, Y., Liu, Z., Qian, P., Liu, Q., Wang, X. and He, Q. (n.d.). Smart Contract Vulnerability Detection Using Graph Neural Networks. [online] Available at: https://www.ijcai.org/proceedings/2020/0454.pdf.

ieeexplore.ieee.org. (n.d.). Combining Graph Neural Networks With Expert Knowledge for Smart Contract Vulnerability Detection. [online] Available at: https://ieeexplore.ieee.org/abstract/document/9477066?casa_token=TtNF4ASR-

FwAAAAA:Eo4xWfWXPN5mOyoIQ-YJYZJYdzl9fhHq-T4lpmjJKX5aiiXYlbXKepJZdYBDmmvrHtgV2fhF [Accessed 5 Nov. 2023].

ieeexplore.ieee.org. (n.d.). An Efficient Smart Contract Vulnerability Detector Based on Semantic Contract Graphs Using Approximate Graph Matching. [online] Available at: https://ieeexplore.ieee.org/abstract/document/10179994?casa_token=j0_U-zIOM-EAAAAA:5td_oJwLu_GsW9CmmRDJIHU7GOUb0KwkzHMS0AOGkfMhdslIQVj-ryv5oOHiffxg8woCEcDN [Accessed 5 Nov. 2023].

Touretzky, D.S. (1986). The Mathematics of Inheritance Systems. [online] Google Books. Morgan Kaufmann. Available at: https://books.google.com.au/books?hl=en&lr=&id=i6GfdROnc1cC&oi=fnd&pg=IA5&dq=Inheritance+Graph+Discussion+&ots=tNrh7FXluS&sig=cpxaQD5GrbOwabcpwCSRgD0BSIQ#v=onepage&q=Inheritance%20Graph%20Discussion&f=false [Accessed 5 Nov. 2023].

Kim, S.-K. and Huh, J.-H. (2020). Neuron Blockchain Algorithm for Legal Problems in Inheritance of Legacy. Electronics, 9(10), p.1595. doi:https://doi.org/10.3390/electronics9101595.