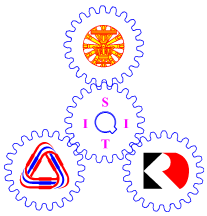

School of Information, Computer and Communication Technology
Sirindhorn International Institute of Technology
Thammasat University

DES484 Blockchain Development Capstone Project

Submitted to

Dr. Watthanasak Jeamwatthanachai

By

Ms. Manlika Lau	ID. 6322770643
Mr. Pannawish Vangtal	ID. 6322772276
Mr. Bhuth Supasaowapak	ID. 6322773787

A REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE ENGINEERING IN DIGITAL ENGINEERING (INTERNATIONAL PROGRAM)
SIRINDHORN INTERNATIONAL INSTITUTE OF TECHNOLOGY
THAMMASAT UNIVERSITY
ACADEMIC YEAR 2023

List of Figure

Table of Contents

Abstract

This report delineates the development and implementation of a decentralized voting application leveraging smart contracts on the Ethereum blockchain. The primary objective was to design a secure and transparent platform enabling users to create, participate in, and manage polls autonomously. The project encompassed two fundamental components: the backend smart contract architecture crafted in Solidity and the frontend interface constructed using React.js.

Key aspects covered in the report include an exploration of the problem domain, identification of requirements, architectural design of the smart contract-based voting system, and the development of a user-friendly frontend interface using React.js. The report outlines the functionalities of the smart contract, such as poll creation, voting mechanisms, closure of polls, and tallying of votes.

Through the utilization of Ethereum's smart contracts, the system ensures immutable and transparent poll management. The React.js frontend offers an intuitive user experience, coupled with Firebase authentication for secure user access.

The report further discusses challenges encountered during development, including gas fees associated with Ethereum transactions and potential future enhancements such as wallet integration for increased security. Additionally, it highlights the utilization of Truffle for smart contract development and deployment.

In conclusion, this project exemplifies the potential of blockchain technology in establishing trustworthy and decentralized voting systems. Despite existing challenges, the Secure Voting System marks a significant stride towards transparent and secure poll-based decision-making processes.

1. Introduction

1.1. Background of Knowledge

A smart contract is a self-executing, programmable agreement that operates on a blockchain. It is a set of code instructions that automatically execute and enforce the terms of a contract when predefined conditions coded within the contract are met. Smart contracts run on decentralized networks, facilitating secure, transparent, and trustless transactions without the need for intermediaries. They enable the automation of various processes and agreements, providing reliability, efficiency, and immutability through their execution on the blockchain. Many industries, including finance, supply chain, healthcare, and governance, stand to benefit significantly from the implementation of smart contracts. The inherent qualities of immutability, decentralization, and transparency offer a robust foundation for enhancing processes and resolving critical issues prevalent in traditional systems.

1.2. Purpose

The purpose of this voting system is to allow users to create a poll, their options and the poll should be open for voting once the conditions of the poll are satisfied by users. Then the users can cast their vote within the time frame of the system. The voting application developed for this project was predicated on the principles of a smart contract, ensuring that each vote cast was cryptographically secure, immutable, and transparent. Utilizing the decentralized nature of blockchain, the application aimed to provide voters with a seamless and trustworthy platform to participate in elections or decision-making processes.
The goal of this project is to delve into the intricate aspects of smart contract development, focusing on the creation of a robust and secure voting application that aligns with the principles of transparency while harnessing the transformative potential of blockchain technology.

2. System Requirement

2.1. Technical Specification

2.1.1. Backend (Smart Contract Development)

- Blockchain Platform: Ethereum
- Language: Solidity for smart contract development
- Development Framework: Truffle was utilized for development, testing, and deployment of smart contracts. However, during the collaborative phase with the frontend development, the usage of Truffle was discontinued in favor of Python scripts for contract deployment.
- Deployment: Deploying smart contracts on Ethereum for testing and on Ethereum mainnet for production
- Security Measures: Implementing secure coding practices, access control, and testing methodologies to ensure smart contract security

2.1.2. Frontend (Mobile Interface)

- Target Platforms: Android and iOS
- Framework: React Native for cross-platform mobile application development

- Navigation: Using React Navigation or similar libraries for defining and managing navigation between screens
- User Authentication: Implementing user authentication functionalities using Firebase Authentication or similar services
- Data Interaction: Integrating web3.js to interact with Ethereum smart contracts from the mobile interface
- User Interface (UI/UX): Creating a responsive user interface for an efficient voting experience

2.2. Key functionalities

2.2.1. Account Management

- Login: Users can log into their accounts..
- Account Creation: Users can create accounts to participate in voting.

2.2.2. Voting Operations

- Vote Casting: Users can cast their votes in open polls.
- View Polls: Users can view available polls for voting.

2.2.3. Poll Management

- Poll Creation: Users have the ability to create their polls with specified questions/topics and choices/options.
- Vote on Other Polls: Users can vote on polls created by others.
- Poll Closing: Poll creators can close their polls to prevent further voting.
- View Closed Polls: Users can view closed polls along with their respective voting results.

3. System Design

3.1. System Overview

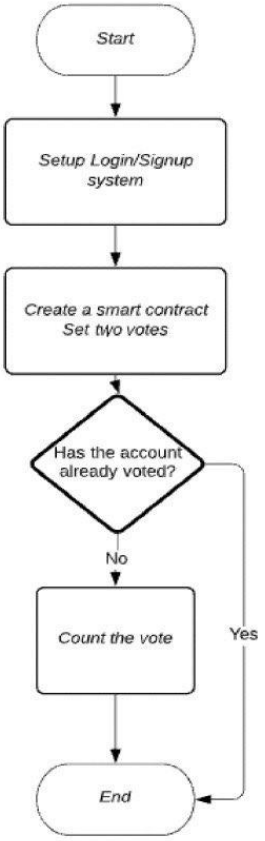

Figure 1: Workflow diagram

Key actions:

- User Authentication: Login/Signup.
- Smart Contract Creation: Setting up the blockchain-based voting system.
- Vote Submission: Managing user votes and ensuring each account votes only once.
- Vote Counting: Tallying votes for each option selected by users.
- Handling Previously Voted Accounts: Providing appropriate feedback for accounts that have already voted.

-

4. System Development

4.1. Backend development

4.1.1. Voting Smart Contract

The smart contract manages the creation, voting, and closure of polls, ensuring transparency and security within the decentralized voting system. It provides an organized structure to handle various functionalities and data related to polls and voting activities on the Ethereum blockchain.

1. **Struct 'Poll'**

```
// Structure to represent a poll
struct Poll {
    address creator;
    string question;                // The question for the poll
    string[] options;               // Array of options for the poll
    mapping(address => uint256) votes;  // Map voter address to their vote (optionIndex + 1)
    bool isClosed;                  // Flag to indicate whether the poll is closed
    uint256 creationTime;           // Timestamp indicating when the poll was created
    mapping(uint256 => uint256) voteCounts;  // Map optionIndex to vote count
    string imageUrl;  // New field to store the image URL or IPFS hash
}
```

Figure 2: Structure defining a poll

2. **Events**
   - PollCreated Event: Indicates the creation of a new poll and triggered when a new poll is created.
   - VoteCast Event: Signals the casting of a vote in a poll. Includes the information about the poll ID, voter's address, and the selected option index. This event is triggered when a vote is cast.
   - PollClosed Event: Notifies the closure of a poll by its creator and emits the poll ID of the closed poll and triggers when a poll is closed.

```
// Event emitted when a new poll is created
event PollCreated(uint256 pollId, string question, string[] options, string imageUrl);

// Event emitted when a vote is cast
event VoteCast(uint256 pollId, address indexed voter, uint256 optionIndex);

// Event emitted when a poll is closed
event PollClosed(uint256 pollId);
```

Figure 3: Events defined in the contract

These events serve as notifications that can be listened to by external applications or contracts. By emitting these events upon specific contract actions, they enable external systems to track and react to the state changes or important actions within the Secure Voting System contract, providing transparency and auditability to the voting process.

3. **Modifiers**
    - onlyPollCreator: Ensures that only the poll creator can perform certain operations (e.g., closing the poll).
    - pollIsOpen: Restricts actions to be performed only on open polls.

```
// Modifier to ensure that the caller is the creator of the poll
modifier onlyPollCreator(string memory pollId) {
    require(msg.sender == polls[pollId].creator, "You are not the creator of this poll");
    _;
}
// Modifier to ensure the poll is open
modifier pollIsOpen(string memory pollId) {
    require(!polls[pollId].isClosed, "Poll is closed or does not exist");
    _;
}
```

Figure 4: Modifiers for the creation of the poll and poll open confirmation

4. **Functions**
    - createPoll: Allows the creation of a new poll with specified question, options, and image URL and creator information. Assigns metadata and emits PollCreated events.
    - closePoll: Permits the poll creator to close a specific poll, updating its status to closed and emitting a PollClosed event.
    - castVote: Enables users to cast votes in open polls, storing vote counts and emitting a VoteCast event.
    - getVoteCount: Retrieves the vote count for a specific option in a poll.
    - viewPoll: Returns details of a specified poll by taking in the 'pollID' parameter.
    - getPollList: Retrieves a list of all poll IDs.

These functions together enable the creation, closure, voting, and retrieval of poll details within the smart contract.

Frameworks and Libraries Used:

1. **Web3**
   'Web3' library facilitates communication with the Ethereum blockchain network and Allows interaction with smart contracts deployed on Ethereum (functions invocation, transactions).
2. **JSON**
   'json' library handles serialization and deserialization of JSON data. It is used to loads and parses JSON-formatted data from files or strings into Python objects.
3. **Hashlib (SHA256)**
   hashlib module provides hashing algorithms (SHA256) for secure hash value generation.It is used for creating unique identifiers (like pollId) by hashing specific data inputs.

4.2. Frontend development

To implement the voting system powered by the smart contract, it is necessary to create a frontend interface. React and React Native are the primary frameworks used for building the user interface and managing components in the application.

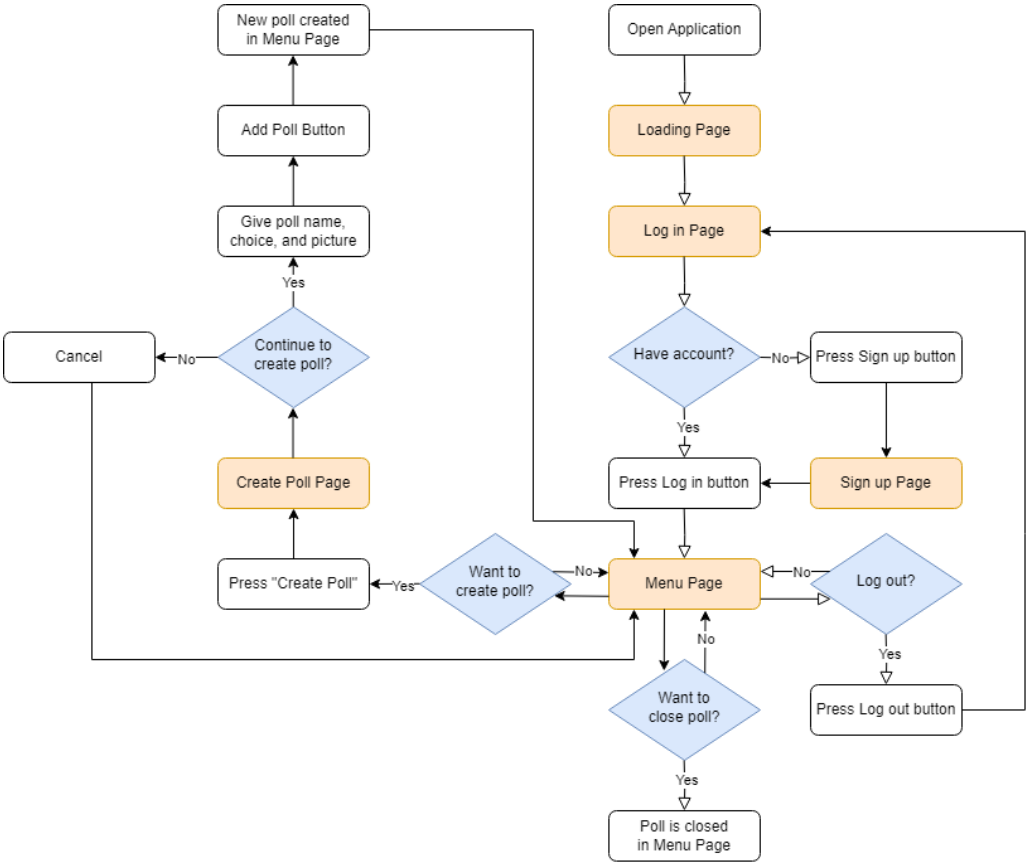

Figure 5: System workflow diagram

Frameworks and Libraries Used:

1. **React and React Native**
   'React' and 'React Native' are the primary frameworks used for building the user interface and managing components in the application.
2. **React Navigation**
   '@react-navigation/native' and '@react-navigation/stack' are used for navigation purposes within the app, managing between four screens (LoadingScreen, LoginScreen, SignUpScreen, MenuPage) and their transitions.
3. **Firebase (Firebase Authentication)**
   Firebase SDK for user authentication (firebase/auth) is used to handle user authentication, For instance, getAuth to retrieve the authentication service instance and signInWithEmailAndPassword takes user-provided credentials as parameters and attempts to authenticate the user. Upon successful authentication, it grants access to the user's account.

4. **Expo SDK**
   'expo' is utilized for various functionalities like handling image picking (expo-image-picker), app root component registration (registerRootComponent), etc.
5. **Web3.js (Ethereum Interaction)**
   Web3.js was employed to interact with the Ethereum blockchain, facilitating communication with smart contracts, reading data, sending transactions, and executing decentralized operations within the application.
6. **Babel (babel-preset-expo)**
   Babel is utilized for transpiling JavaScript code in the project, ensuring compatibility across different environments.

4.3. User Interface

Figure 6: Loading page and the Login page

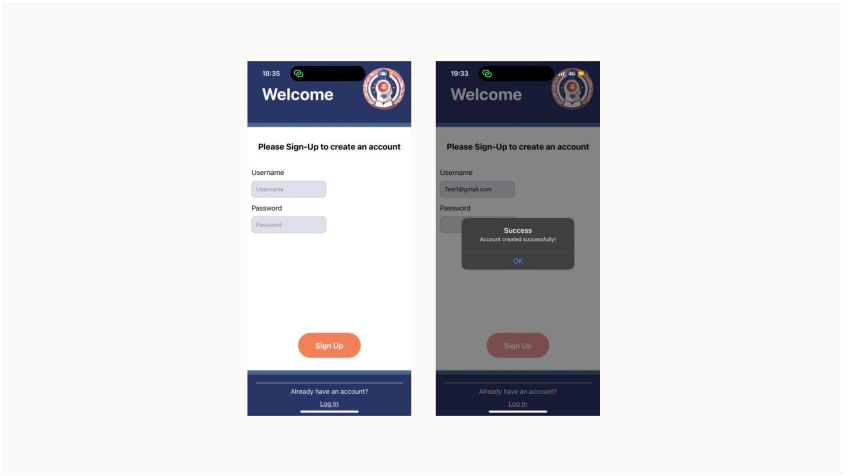

Figure 7: Successful login

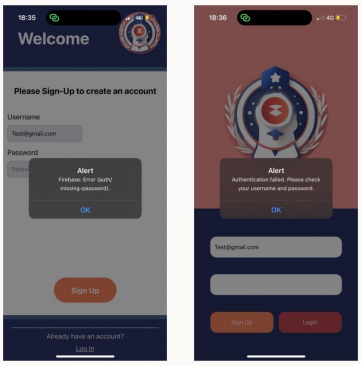

Figure 8: Failed login

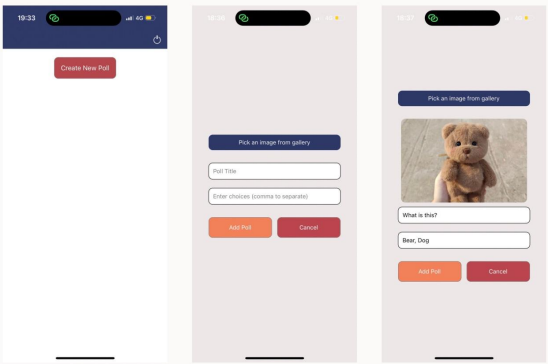

Figure 9: Poll creation process

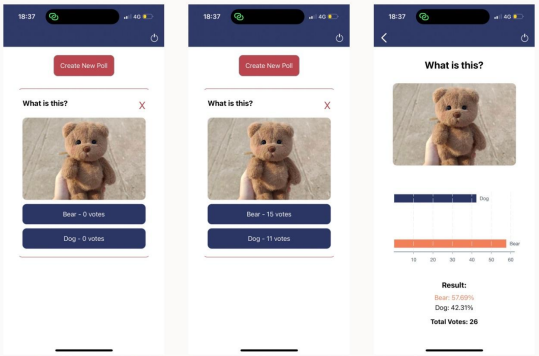

Figure 10: Voting process

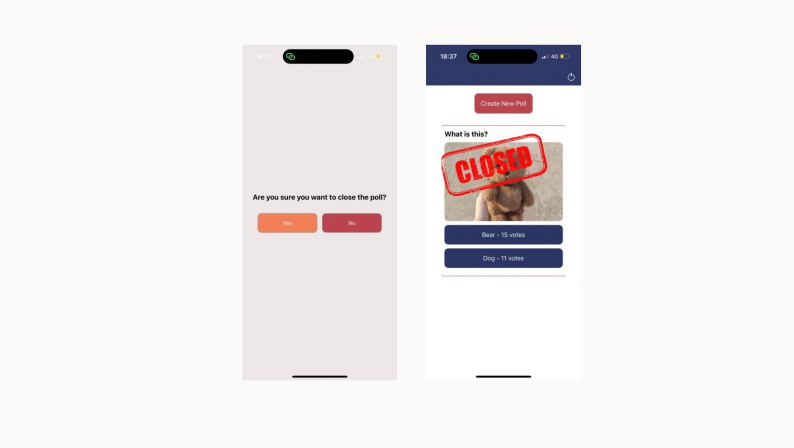

Figure 11: Closing a poll

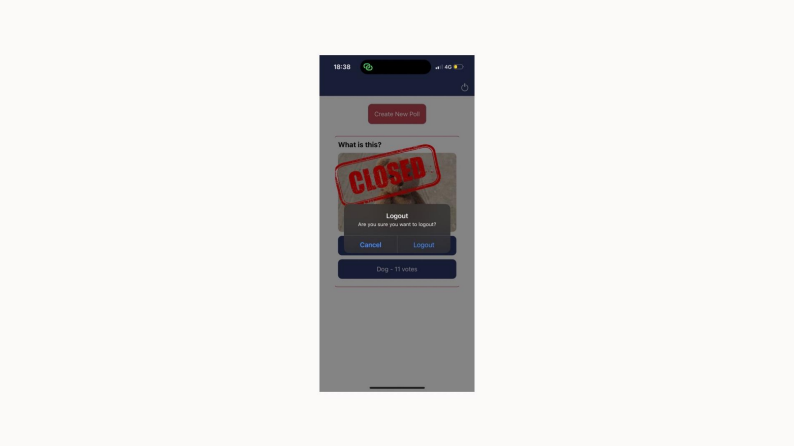

Figure 12: Log out

4.4. Testing

A set of JavaScript tests was written using the Mocha and Chai testing frameworks to test the functionalities of the SecureVotingSystem contract on the Ethereum blockchain.
Test Cases Overview:

1. **createPoll Tests:**
   - Case 1: Checks whether a poll can be created without an image. Verifies that the generated poll ID is not 0.
   - Case 2: Verifies the creation of a poll with an associated image.
2. **closePoll Tests:**
   - Case 1: Tests the closure of a poll by its creator. Verifies the emission of the PollClosed event and checks if the poll is closed.
   - Case 2: Ensures that a non-creator is unable to close a poll and verifies that the poll remains open.
3. **castVote Tests:**
   - Case 1: Validates that a user can cast a vote in a poll. Checks if the correct voter address and voted option index are recorded.

- Case 2: Tests that a user cannot vote twice in the same poll.
- Case 3: Ensures that voting is not allowed in a closed poll.

4. **getVoteCount Test:**
   - Case 1: Checks the functionality to retrieve the vote count for a specific option in a poll after multiple votes have been cast.

```
Contract: SecureVotingSystem
  ✔ should create a poll without an image (152ms)
  ✔ should create a poll with an image (103ms)
  ✔ should close a poll by the creator (93ms)
  ✔ should not allow closing a poll by a non-creator (289ms)
  ✔ should allow a user to cast a vote in a poll (150ms)
  ✔ should not allow a user to vote twice in the same poll (165ms)
  ✔ should not allow voting in a closed poll (247ms)
  ✔ should get the vote count for an option in a poll (232ms)


8 passing (2s)
```

Figure 13: Test cases results

All test cases passed successfully, validating the intended functionality of the contract. The tests examined the contract's capability to manage polls, handle votes, and ensure the security and correctness of the voting process. The contract behaves as expected in various scenarios, showcasing its readiness for deployment in decentralized voting systems.

5. Discussion

5.1. Challenge

The challenge related to gas in Ethereum smart contracts involves managing and optimizing the cost associated with executing functions or transactions on the blockchain. Gas is the unit used to measure the computational effort required to execute operations on the Ethereum network. Every operation (function call, data storage, etc.) consumes a certain amount of gas, and users must pay gas fees to miners to execute transactions or interact with smart contracts.

5.2. Solution

5.2.1. Code Optimization

Implementing efficient coding practices and algorithms can significantly reduce gas costs. Techniques like loop unrolling, minimizing storage usage, and utilizing cheaper operations help decrease gas consumption.

5.2.2. Gas Estimation

Ganache's gas estimation capabilities help assess the efficiency and cost-effectiveness of their smart contracts and transactions during the development phase, ensuring smoother deployments on the Ethereum mainnet.

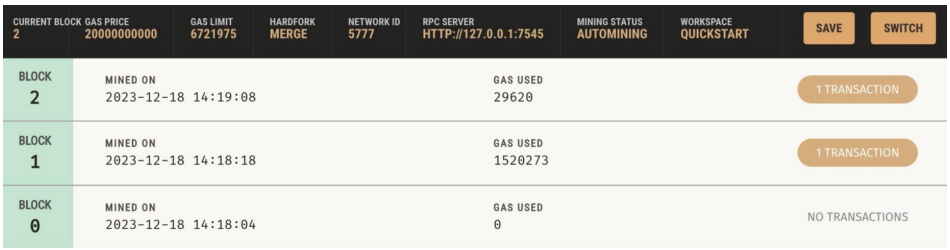

Figure 14: Gas estimation

5.3. Future development

1. **Enhanced Security Measures**
   Implementation of additional security layers, like multi-factor authentication (MFA) or cryptographic enhancements, to fortify user authentication and prevent unauthorized access.
2. **Encryption Techniques**
   Utilization of encryption methods to secure sensitive data, such as voting details or user information, ensuring confidentiality and privacy.
3. **Improved User Experience (UX)**
   Enhancement of the frontend interface with better UX/UI design, intuitive navigation, and user-friendly features, promoting ease of use for voters.

6. Conclusion

The development and implementation of the Secure Voting System created a decentralized and secure platform for conducting transparent and immutable polls on the Ethereum blockchain. By leveraging the power of smart contracts, this system allows users to create, participate in, and manage polls while ensuring the integrity and confidentiality of the voting process.

The system's backend, built with Solidity smart contracts, facilitates the creation and closure of polls, tracks votes, and provides transparent results. Events emitted by the contract enable external systems to monitor key actions, ensuring transparency and accountability throughout the voting process.

On the frontend, the React.js-based interface offers a user-friendly experience, enabling seamless interaction with the Ethereum blockchain. Firebase authentication ensures secure user login and management, while Expo's image handling capabilities enhance the visual aspects of the application.

Despite the successful implementation, there remain areas for potential future enhancements, including improved security measures, wallet integration, and scalability solutions. Additionally, the exploration of emerging technologies and compliance with regulatory standards will contribute to the system's evolution and broader adoption.

References

[1] *DappVotes*, 2022. https://dapp-vote-three.vercel.app/

[2] "Ethereum for Rust developers," *ethereum.org*. https://ethereum.org/en/developers/docs/programming-languages/rust/#getting-started-with-smart-contracts-and-solidity

[3] "React Developer Tools – React," *react.dev*.https://react.dev/learn/react-developer-tools#mobile-react-native

[4] Firebase Authentication: "Firebase Authentication - Usage" https://rnfirebase.io/auth/usage