Decentralized Application (DApp) Challenge Task: Designing, Securing, and Analyzing a Blockchain-Based Lottery System

Junyong Cao[a], Yunxiang Guo[a] and Yunlong Li[a]

[a]Department of Informatics
University of Zurich

Abstract. For this challenge task, we developed a lottery system that allows users to place orders and set their own bet amounts. All transactions are carried out on the blockchain, including the award distribution system. To build this system, we used REACT for the front-end design, a test blockchain network from the University of Zurich for the back-end blockchain, and Remix to write the Solidity smart contracts. Finally, we utilized web communication for the front-end and implemented a security strategy to protect the website's blockchain, including communication and random number generation. We also conducted security analysis of the lottery system from the other group.

1 Introduction

This report presents the design, implementation, and security assessment of a Decentralized Application (DApp) developed as part of the Challenge Task (CT) for this semester. The CT focused on creating a lottery application and securing it against potential threat vectors. Our group diligently worked on the three stages of the CT, which included application design and planning, implementation with added security mechanisms, and the exchange of source code for security analysis. The objective of this CT was to showcase our ability to develop a self-contained DApp that operates entirely within Smart Contracts (SC) on the blockchain. We aimed to create a lottery application that not only fulfilled the core functionality requirements but also validated the proposed use case. To enhance user interaction, we incorporated a Graphical User Interface (GUI), allowing users to engage with the DApp conveniently. Throughout the implementation phase, we focused on striking a balance between security and portability. Portability was a key characteristic that needed to be ensured at all stages of the CT, allowing for straightforward deployment and operation of the lottery application on different machines. We made careful design choices to avoid additional security services that could hinder its portability, such as user-based random number generator that relied on addresses and keys of users and admins.

The second stage of the CT was dedicated to securing our lottery application against potential threat vectors. Extensive research and investigation were conducted to identify and evaluate the various threats that could potentially compromise the integrity and security of our DApp. Considering the limited time frame and wide range of threat vectors, we selected suitable security mechanisms and design choices to mitigate the identified risks. These security measures were implemented to safeguard the DApp while still ensuring its portability. In the final stage of the CT, we exchanged source code with another group and performed a comprehensive security analysis. This analysis covered both the smart contract and the use case of the provided DApp. Our goal was to identify any vulnerabilities or weaknesses in the code and propose necessary improvements to enhance the overall security and robustness of the DApp.

This report serves as a documentation of our journey throughout the CT. It provides detailed insights into the design, implementation, and security analysis of our lottery application. Additionally, the report contains comprehensive technical information that will aid in understanding our solution during the presentation and demo. We adhered to the guidelines provided, allowing us to utilize existing libraries and code, provided they were published under suitable open software licenses. This approach enabled us to leverage existing solutions effectively and focus on integrating them seamlessly into our DApp. To conclude, this CT provided us with valuable hands-on experience in developing decentralized applications, securing them against potential threats, and conducting thorough security analyses. Through this report, we aim to demonstrate our proficiency in blockchain technology, smart contract development, and our ability to ensure the robustness and security of our DApp.

This project focuses on the design and implementation of a secure decentralized lottery system. In Section 1, the challenge task is introduced, emphasizing the importance of creating and securing such systems. Section 2 discusses the architectural design of the decentralized application (DApp), including both front-end and back-end components and the technologies used. It also explains the design choices made to ensure security and user-friendliness. Section 3 focuses on the security design, outlining the measures taken to protect the blockchain-based lottery system, such as secure communication and random number generation. Section 4 delves into the implementation details, explaining the development and integration of the front-end and back-end components, as well as the findings from the security assessment phase. Section 5 presents a code analysis, highlighting vulnerabilities and weaknesses found and providing recommendations for improvement. Finally, in the Conclusion, the overall outcomes of the project are summarized, emphasizing the successful development of the decentralized lottery system and the importance of continuous testing and improvement for blockchain-based applications.

* Github: https://github.com/xiaoxiaoshikui/Lottery-System-Blockchain-Challenge-Task.git

2 Architecture & Design

The design overview of Decentralized Application (DApp) is shown in Figure 1. In the front end, users can use this application to purchase and open lottery tickets. The rule is to generate a random number for lottery award. The back-end uses its own designed SC to communicate with the WEB. We refer the UI design of a lottery system[1] for our UI.

In this lottery, participants will use ERC20 tokens as their bet.

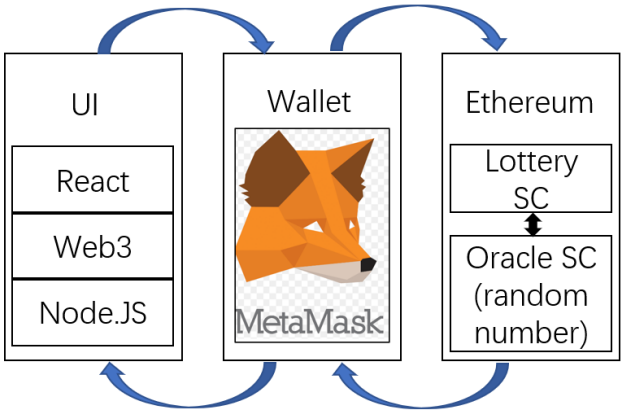

Figure 1. Architecture Overview

Before joining, participants must purchase their bet and pay the entry fee. Each lottery session lasts for around 1 minute, during which a random number will be generated by the smart contract to select a single winner. The winner will receive all the entry fees collected for that session.

2.1 Front End

The front end was implemented by REACT [2, 4], and the communication between the front end and smart contract was implemented by Web3 and Node JS [5]. It mainly contains several functions. First, we could log in the system via the address and private key from each participant, please see Figure 2.

To register or log in, please use the provided address and the key from participants as shown in Figure 4. Once logged in, you will need to increase your account balance by selecting the "Add Money" option. The administrator will be responsible for adding funds to your account. Once you have received the funds from the administrator, you can press the "Add Money" button. The added money will be displayed in your account balance, enabling you to participate in the game.

To join the game, navigate to the game section. At this point, the administrator should execute the "play-manage-first.js" file using Node.js. This step is crucial for enabling participants to engage in the game. Once the file is executed, players can place their bets. The "SPIN" function will generate random numbers to determine the winners, and the results will be verified to identify the prize recipient.

[1] https://github.com/peterdurham/roulette/tree/master

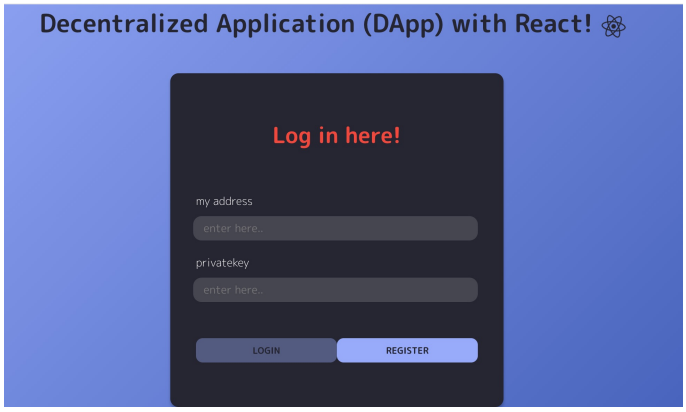

Figure 2. Login

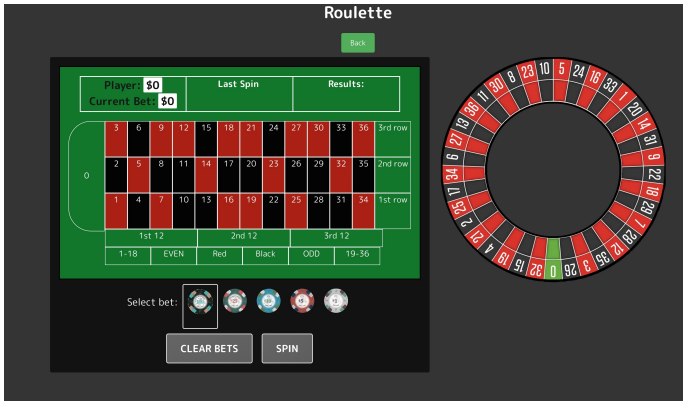

Figure 3. Game Layout

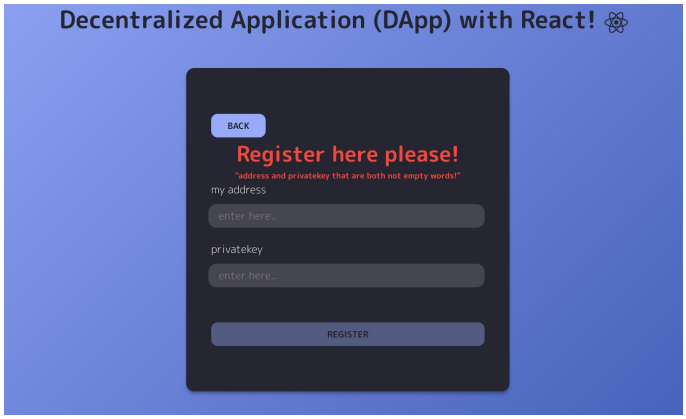

Figure 4. Register

2.2 Communication between Front-End and Back-End

We use a JavaScript implementation for interacting with an Ethereum blockchain using the Web3 library and ethers.js. It showcases various functions and transactions related to a smart contract.

We begin by initializing a Web3 provider and creating instances of the smart contracts using their respective ABIs (contract interface) and addresses. The functions in the code represent different interactions with the contracts, such as approving a transaction, registering users, refreshing balances, placing bets, and picking winners. These functions utilize the provided addresses, private keys, and function arguments to generate transaction data, sign the transactions, and send them to the Ethereum network.

We also include an Express server that exposes several endpoints for handling HTTP requests. These endpoints allow users to perform actions like logging in, registering, adding money to their wallets, and placing bets. The server communicates with the Ethereum network by calling the corresponding functions and returning the results as JSON responses.

2.3 BackEnd

The implementation of the back end consists of three smart contracts [1, 3, 6]. The first contract, ERC20.sol, is an interface contract called IERC20 that represents the ERC20 token standard. It defines a set of functions and events commonly associated with ERC20 tokens. The second contract, DAI.sol, implements the ERC20 standard and introduces a custom token called 'Stakes' with the symbol 'L' [2]. It enables token transfers, approvals, minting, and burning functionalities. Finally, the Lottery.sol contract facilitates a lottery system where participants can register, place bets using an ERC20 token, and have a chance to win based on a randomly chosen number.

2.3.1 Smart Contract: ERC20.sol

The provided Solidity smart contract is an interface contract called IERC20 that represents the ERC20 token standard. This interface defines a set of functions and events commonly associated with ERC20 tokens.

The functions include "totalSupply()" to retrieve the total number of tokens in circulation, "balanceOf()" to check the token balance of a specific address, "transfer()" to transfer tokens from the caller's address to a recipient, "allowance()" to view the approved amount of tokens by an owner for a spender, "approve()" to approve a spender to spend a certain amount of tokens, and "transferFrom()" to transfer tokens on behalf of an owner.

Additionally, the interface contract emits two events: Transfer and Approval. The Transfer event is triggered when tokens are transferred from one address to another, while the Approval event is emitted when an owner approves a spender to spend a specified amount of tokens.

In summary, the IERC20 interface contract provides a standardized set of functions and events for interacting with ERC20 tokens. It serves as a blueprint for implementing ERC20 token contracts, ensuring compatibility and facilitating token transfers, approvals, and balance inquiries within the Ethereum ecosystem.

2.3.2 Smart Contract: DAI.sol

The contract implements the ERC20 standard and defines a custom token called "Stakes" with the symbol "L". It has a total supply, and the balances of token holders are stored in the balanceOf mapping.

The contract allows token transfers between addresses through the "transfer()" function. It deducts the transferred amount from the sender's balance and adds it to the recipient's balance. The contract emits a Transfer event to notify external listeners about the transfer. Token holders can approve other addresses to spend a certain amount of their tokens on their behalf using the "approve()" function. The approved amount is stored in the allowance mapping. The "transferFrom()" function allows approved addresses to transfer tokens on behalf of the token owner.

It deducts the approved amount from the allowance and the sender's balance, and adds the transferred amount to the recipient's balance. The contract includes a "mint()" function that can only be called by the contract creator. It allows the creator to mint new tokens and assign them to a specified address. The total supply and the recipient's balance are increased accordingly, and a Transfer event is emitted. Token holders can also burn their own tokens using the "burn()" function. It reduces the token holder's balance and the total supply by the specified amount, and emits a Transfer event with the recipient set to the zero address. The contract includes a onlyCreator modifier that restricts access to certain functions, allowing only the contract creator to execute them.

Overall, this contract represents a basic implementation of an ERC20 token with functionality for token transfers, approvals, minting, and burning.

2.3.3 Smart Contract:Lottery.sol

The contract allows participants to register for the lottery by calling the "register()" function. Once registered, participants can place bets on a set of numbers using the "placeBet()" function. Each bet consists of a list of numbers and corresponding stakes.

The contract uses an ERC20 token for the betting process, which is specified during contract deployment. Participants must have sufficient balance of the token to place their bets. The contract maintains a list of participants and their relevant information such as balances, bet history, and registration status. The "balanceRefresh()" function can be called to update the participants' token balances. Once all the available slots for participants are filled, indicated by the totalParticipants variable, the contract stops accepting new bets. The "pickWinner()" function is then triggered to determine the winner.

The winner is chosen based on a random number generated using a pseudo-random number generation algorithm. The "getRandomNumber()" function generates the random number by combining block information, difficulty, and a seed value derived from the chosen numbers. The winner's index is obtained by taking the modulo of the generated random number with 36. After determining the winner, the contract transfers the corresponding winnings to the winner's address. The winnings are calculated by multiplying the bet amount by 36. The contract then resets all variables, clears the participant list, and allows new bets to be placed for the next round. The contract also provides several getter functions for retrieving information such as the winner's index, participant balances, and total stake amount.

Overall, this contract facilitates a lottery system where participants can register, place bets using an ERC20 token, and have a chance to win based on a randomly chosen number.

3 Security Design

The security design is of great importance for the smart contract since it can prevent different attacks and safe the cryptocurrencies.

Our main focus is the random number generating process, which is essential for the lottery system.

The given algorithm 1 describes a function called "getRandomNumber" that generates a random number within a specified range.

The function takes no input parameters and returns a random number as its output. It utilizes the keccak256 hash function, which is commonly used in cryptographic applications, to generate a pseudo-random number. The inputs to the hash function include the current timestamp and the difficulty level of the block.

To ensure that the generated random number falls within a specific range, a modulo operation is performed. In the provided example, the modulo operation limits the random number to a range between 0 and 35 (inclusive), as it is computed as "randomNumber mod 36".

Algorithm 1 Function getRandomNumber

function GETRANDOMNUMBER
    Input: None
    Output: Random number

    uint256 randomNumber ← uint256(keccak256(
        abi.encodePacked(block.timestamp,
        block.difficulty, seeds))))

    Modulo operation to limit the range of the random number
    $randomNumber \leftarrow randomNumber \mod 36$ **[If you want numbers between 0 and 35]**
    return $randomNumber$
end function

Algorithm 2 Contract Lottery

Require: $address\_token, uint256tp$

$address[]participants$
$IERC20token$
$uint256winnerIndex$
$uint256totalParticipants$
$uint256pickwinnerStatus = 1$
$addresscreator$
$uint256seeds = 1$
$mapping(address \rightarrow participantInfo)allParticipants$

function LOTTERY($address\_token, uint256tp$)
    $creator \leftarrow$ msg.sender
    $token \leftarrow$ IERC20($\_token$)
    $totalParticipants \leftarrow tp$
end function

The "Contract Lottery" algorithm 2, as described in the provided pseudocode , presents the initial structure of a smart contract designed for a blockchain-based lottery system. The algorithm includes variables like participants, token, and totalParticipants, and features a constructor function called Lottery to initialize the contract. This algorithm serves as a foundational framework for developing a functional lottery system on a blockchain platform. Citation: OpenAI, "Pseudocode for the 'Contract Lottery' smart contract: An initial structure for a blockchain-based lottery system."

The "register" algorithm 3 outlines a function designed to allow users to register for a lottery system. The algorithm checks if the user calling the function is already registered by verifying their registration status in the allParticipants mapping. If the user is not already registered and the number of participants is below or equal to the total allowed participants, the function proceeds to register the user by assigning default values to their balance and bet flag, setting their registration status to 1, and adding their address to the participants array. However, if the maximum number of participants has been reached, an exception is thrown indicating that the round is full. Similarly, if the user is already registered, an exception is thrown indicating that they have already been registered.

Algorithm 3 Function Register

function REGISTER
    $addressusr \leftarrow$ msg.sender
    if $allParticipants[usr].register\_status \neq 1$ **then**
        if $participants.length \leq totalParticipants$ **then**
            $allParticipants[usr].balance \leftarrow 0$
            $allParticipants[usr].betFlag \leftarrow 0$
            $allParticipants[usr].register\_status \leftarrow 1$
            $participants.push(usr)$
        else
            throw "This round is full of players"
        end if
    else
        throw "You have been registered"
    end if
end function

Algorithm 4 Function PickWinner

function PICKWINNER
    if $participants.length > 0$ **then**
        $winnerIndex \leftarrow$ getRandomNumber()
        for $i \leftarrow 0$ **to** $participants.length$ **do**
            $addressusr\_id \leftarrow participants[i]$
            $token.transfer(usr\_id,$
$allParticipants[usr\_id].BetInfo[winnerIndex] \times 36)$
        end for
        for $i \leftarrow 0$ **to** $participants.length$ **do**
            $participants[i]$
            delete allParticipants[participants[i]]
        end for
        $seeds \leftarrow 1$
        delete participants
        $pickwinnerStatus \leftarrow 1$
    end if
end function

The "pickWinner" algorithm 4 represents a function that selects a winner in a lottery system and distributes the corresponding prize. The algorithm first checks if there are any participants in the lottery by verifying the length of the participants array. If participants are present, the algorithm proceeds to pick a random winner by calling the getRandomNumber() function. Then, in a loop iterating over each participant, the algorithm transfers tokens to the winner based on their bet information stored in the BetInfo mapping of the allParticipants mapping. After distributing the prizes, the algorithm removes the participant entries from the allParticipants mapping and resets the necessary variables for a new round.

4 Implementation

4.1 FrontEnd

To set up the local development environment, Node.js is required, and all other dependencies, including React, can be installed using the npm install command. The application can be started with npm run dev and accessed at http://localhost:3000. The template mentions that the page will automatically reload upon code changes, and lint errors will be displayed in the console.

Testing is considered optional but can be performed using the npm run test command, which launches the test runner in an interactive watch mode. The template also provides guidance for macOS users encountering an error related to 'fsevents'.

To build the application for production, the npm run build command is used. This generates a build folder with optimized and minified files, including bundled React code.

4.2 BackEnd

The back end code is written in Solidity and implements the smart contract for the lottery system. The contract is named "Lottery" and starts with the SPDX-License-Identifier and pragma statements to specify the license and version of Solidity used. The code also imports the IERC20 interface from the OpenZeppelin library to interact with ERC20 tokens.

The contract includes various state variables, such as the participants array to store the addresses of registered participants, the token variable of type IERC20 to represent the ERC20 token used for betting, the winnerIndex to store the randomly selected winner, the totalParticipants to define the maximum number of participants, and the pickwinnerStatus to indicate the status of the winner selection process.

The contract also includes several getter functions for retrieving information such as the winnerIndex, participant balances, and stake sums.Overall, the back end implementation provides the necessary functionality for participants to register, place bets, and select a winner in the lottery system. The code includes appropriate checks and restrictions to ensure the integrity and security of the lottery process.

5 Code analysis

In this code analysis section, we aim to analyze the lottery systems implemented in blockchain technology of *Group Apollo*. Specifically, the focus will be on assessing the security vulnerabilities of the smart contracts used in these systems. By conducting a thorough examination of the code, we intend to identify potential weaknesses and propose recommendations for enhancing the security of the lottery systems.

5.1 Findings

Based on the code analysis, the following vulnerabilities and weaknesses were identified in the lottery systems' smart contracts:

1. **Randomness Generation**: Although the smart contract (SC) utilizes a pseudo-random number generator, the seeds for the random number generator are generated by the player's decision of a lucky number. This approach closely resembles an off-chain solution, providing users with the ability to verify and ensure the randomness.
2. **PickWinner function's weakness** if there are no winner, the SC might get stuck in this function, Thats is because no matter how many times you call this function, the random value seeds won't change, therefore, it might be a better idea to call setRandom for one times within this function so that the seeds can be changed based on the block.timestamp.
3. **Front-Running Attacks**: During the reveal phase, an attacker may monitor pending transactions and quickly execute their own reveal transaction with a lucky number that gives them an advantage in winning the lottery.
4. **Transaction Cost**: The gas fees associated with the pickWinner phase can be prohibitively expensive. This is because the function requires looping through all participants to determine a winner, which can potentially incur high gas costs. To mitigate this issue, it would be more efficient to implement a binary search algorithm, which can significantly reduce the gas consumption by optimizing the process of finding the winner.

5.2 Recommendations

To address the vulnerabilities and weaknesses identified in the code analysis, the following recommendations are suggested:

1. Gas Efficiency: The code could be optimized for gas efficiency. For example, the picking of a winner involves looping through all entries, which could be expensive if there are many participants. Consider using more efficient algorithms, like binary search, to improve gas consumption.
2. Security Considerations: The code could benefit from additional security measures. For instance, it's advisable to use the blockhash of a previous block (blockhash(block.number - 1)) instead of relying solely on the current block's timestamp for random number generation.
3. Code Organization: The code could be further organized to enhance readability and maintainability. Group related functions together, use comments to explain complex logic, and follow consistent naming conventions.
4. Input Validation: The code should incorporate additional input validation checks to ensure that all required parameters are provided and valid. For example, the "reveal" function should validate the revealed lucky number against the stored commit and ensure it is within the valid range.
5. Considerations for Large Lotteries: If the number of participants (lottery size) becomes very large, the looping through all entries could become inefficient. Consider using a different approach, such as using a priority queue or maintaining a sorted list of participants based on their range ends, to speed up winner selection. Remember that code security and efficiency are critical in smart contracts, as they handle valuable assets. It is highly recommended to conduct thorough testing and auditing of the contract before deployment to ensure its reliability and security.

6 Discussion

The main contribution of this project is the design, implementation, and analysis of a decentralized lottery system using blockchain technology. The project successfully developed a secure and transparent lottery application where users can participate by placing bets using ERC20 tokens. The architecture and design of the application were carefully planned, considering both the front-end and back-end components. Security measures, such as secure communication and

random number generation, were implemented to protect the system against potential threats. The code analysis phase identified vulnerabilities and weaknesses, and recommendations were provided to address these issues and enhance the security of the lottery system. Overall, this project demonstrates the practical implementation of blockchain technology in a real-world application and emphasizes the importance of security and user-friendliness in decentralized systems.

7 Conclusion

In conclusion, this challenge task provided us with an opportunity to design, implement, and analyze a blockchain-based lottery system. We successfully developed a decentralized application (DApp) that allows users to participate in a lottery by placing bets using ERC20 tokens. The entire process, from registration to winner selection, is executed on the blockchain, ensuring transparency and security.

During the design phase, we carefully planned the architecture of our DApp, considering both the front-end and back-end components. We utilized React for the front-end design, implemented smart contracts using Solidity, and established communication between the front-end and back-end using Web3 and Node.js. This allowed for seamless interaction between users and the blockchain-based lottery system.

Security was a significant focus throughout the development process. We implemented several security measures, such as protecting the blockchain communication and ensuring secure random number generation. By using the keccak256 hash function and incorporating block information and difficulty, we generated pseudo-random numbers within a specified range. While this approach provides a level of randomness, further enhancements could be made by incorporating additional sources of randomness, such as blockhash.

During the security analysis phase, we exchanged source code with another group and conducted a comprehensive assessment of their lottery system. This analysis helped identify potential vulnerabilities and weaknesses in the code. We provided recommendations to address these issues, such as optimizing gas efficiency, improving input validation, and considering alternative algorithms for winner selection in large lotteries.

In conclusion, this challenge task allowed us to showcase our skills in designing, implementing, and securing a blockchain-based lottery system. We gained valuable experience in developing decentralized applications, integrating smart contracts, and conducting security analyses. By addressing the identified vulnerabilities and incorporating the recommended improvements, we can enhance the security and reliability of the lottery system. Overall, this project has deepened our understanding of blockchain technology and its applications in real-world scenarios.

Acknowledgements

We would like to express our gratitude to Prof. Dr. Burkhard Stiller for his outstanding teaching and expertise, which greatly contributed to our understanding and knowledge in the field. We are also immensely thankful to Dr. Bruno Bastos Rodrigues for his valuable comments and assistance throughout the entire project. His guidance and insights significantly improved the quality and depth of our work. Additionally, we extend our appreciation to Mr. Jan von der Assen for his assistance in the exercise section, which provided us with a solid foundation in the concepts essential for the project's success.

References

[1] Vitalik Buterin, 'Ethereum: A next-generation smart contract and decentralized application platform', *white paper*, (2013).

[2] Barry Daly, Thomas Mellor, Peter Chapman, and Dimitrios Panaritis, 'Reactjs: A javascript library for building user interfaces', *IEEE International Conference on Software Architecture (ICSA)*, 505–508, (2018).

[3] Loi Luu, Duc-Hiep Chu, Han Olickel, and Prateek Saxena, 'Making smart contracts smarter', *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 254–269, (2016).

[4] Andrew Martin, 'React: A javascript library for building user interfaces', *Journal of Object Technology*, **16**(2), 1–2, (2017).

[5] Stefan Tilkov and Steve Vinoski. Node.js: Using javascript to build high-performance network programs. Whitepaper, 2010.

[6] Gavin Wood, 'Ethereum: A secure decentralised generalised transaction ledger', *Ethereum Project Yellow Paper*, **151**, (2014).