Hochschule RheinMain
Fachbereich Design Informatik Medien
Studiengang Informatik - Technische Systeme

Bachelor-Arbeit

zur Erlangung des akademischen Grades

Bachelor of Science - B.Sc.

Study and implementation of a decentralized application that can provide permissionless financial services using an evm based blockchain

Vorgelegt von	Mario Alberto Maita Orozco
am	30.06.2022
Referent	Prof. Dr. Eva-Maria Iwer
Korreferent	Prof. Dr. Marc-Alexander Zschiegner

Erklärung gem. ABPO, Ziff. 4.1.5.4

Ich versichere, dass ich die Bachelor-Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Wiesbaden, 30.06.2022

Mario Alberto Maita Orozco

Erklärung zur Verwendung der Bachelorthesis

Hiermit erkläre ich mein Einverständnis mit den im Folgenden aufgeführten Verbreitungsformen dieser Bachelor-Arbeit:

Verbreitungsform	**Ja**	**Nein**
Einstellung der Arbeit in die Hochschulbibliothek mit Datenträger		×
Einstellung der Arbeit in die Hochschulbibliothek ohne Datenträger	×	
Veröffentlichung des Titels der Arbeit im Internet	×	
Veröffentlichung der Arbeit im Internet	×	

Wiesbaden, 30.06.2022

Mario Alberto Maita Orozco

Abstract

abstract

Contents

Chapter 1

Introduction

1.1 Motivation

In the last years' blockchain technology has gained a lot of traction and many developers are willing to build applications on top of this technology. The Ethereum network with the Ethereum Virtual Machine, or EVM 2.2.2, (whose state is validated and copied by every node of the network) offers a new way of application development. Meaning that developers can now implement some functionality and deploy it to a public EVM blockchain, making the code/functionality tamper-resistant; in addition, the implemented functionality (or smart contract) can be called by any participant or other code/contracts within the network. This composability allows anyone to interact with already deployed contracts and build other applications on top of them. These kinds of applications are called Decentralized applications or Dapps 2.2.5, because they are built on top of a decentralized network such as Ethereum.

1.2 Goals

The goals of this Thesis are, to study and provide a better understanding of:

- How Dapps are built.
- Which technology is used.
- And the development of a Dapp that uses well-known DeFi protocols 3, to let users access to permissionless financial services such as lending and borrowing of crypto assets.

1.3 Thesis Structure

The Thesis is divided into the following chapters:

- **Chapter 2 - Fundamentals**, gives a technical overwiev of the blockchain technology and the Ethereum network.
- **Chapter 3** ...
- **Chapter 4** ...
- **Chapter 5** ...
- **Chapter 6** ...

Chapter 2

Fundamentals

2.1 Blockchain

A blockchain[?][?] is a type of database (append-only data strucuture), which stores data in blocks. These blocks are chronologically ordered by discrete timestamps and linked to each other using cryptographic hash functions[?][?]. Commonly, a blockchain is used as a public distributed ledger of transactions records, shared and synchronized by a per-to-per network, where every party can participate in the validation of new blocks based on a consensus protocol, see 2.1.3. The idea of a cryptographically secured chain of blocks was originally presented by Haber and Stornetta[?] in 1991. However, the implementation and adoption of this technology started with the conception of the bitcoin cryptocurrency whitepaper[?] in 2008.

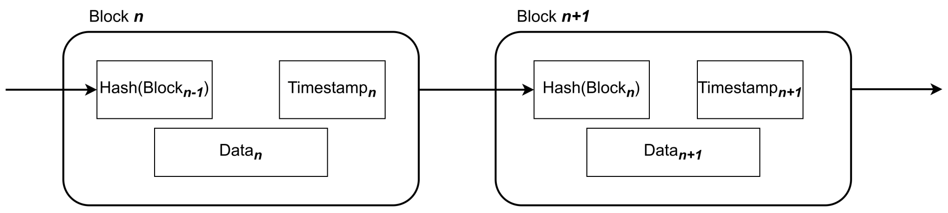

Figure 2.1: View of two blocks in a blockchain

Determined by the blockchain implementation, block contents can be different. A block usually has a timestamp, the data or transactions record, and the hash value of the previous block in the chain, see figure 2.1. The way that a blockchain ensures the security and keeps the integrity of the data is through cryptographic hash functions 2.1.1, and Merkle trees 2.1.2

By adding the cryptographic hash value of the previous block (Block n to Block $n+1$), blockchains ensure the integrity of previous blocks. Because of the second preimage resistance of cryptographic hash functions, see 2.1.1, it is computationally infeasible to find an input other then Block n in order to generate an output = Hash (Block n). Due to that, the further the blockchain grows beyond block n, the more secure is the integrity of block n and previous blocks.

Depending on who can participate in the consensus protocol, blockchain networks can be divided into two groups: permissionless and permissioned blockchains. This Thesis will be focused on permissionless blockchains that use an EVM, such as the Ethereum network.

2.1.1 Cryptographic hash functions

A cryptographic hash function[**?**][**?**] maps a given data of variable size to a fixed length n-bit string called hash value. $H : \{0,1\}^* \rightarrow \{0,1\}^n$ Such hash functions have the following properties:

Collision resistance: Given x_1, x_2 It should be necessary $O(2^{\frac{n}{2}})$ compute power such that $H(x_1) = H(x_2)$.This means, that it will be computationally infeasible that different inputs, i.e., blocks of data, will generate the same hash value.

Preimage resistance: For a hash value h, it should be necessary $O(2^n)$ computation power in order to obtain an x such that $H(x) = h$. In other words, it should be a one-way function. This means that it is computationally infeasible to retrieve the input data from its hash value. For instance, it is nearly impossible to get a private key from its public key.

Second preimage resistance: For an input x_1 and its hash value h_1, it should be necessary $O(2^n)$ computation power in order to obtain a x_2 such that $H(x_2) = h_1$. This means that for a given hash value, it is computationally infeasible to find another input, i.e., a block data input, that generates the same hash value.

Such properties prevent attackers from modifying existing blocks keeping the blockchain intact. For example, the bitcoin network uses the SHA-256 hashing algorithm to validate a block, and append it to the chain. This means once a consensus among all the participants is done, and a new block is added, it will be required $O(2^{256})$ computation power to tamper with the blockchain. In other words, it is nearly impossible to manipulate an added block as a result of the tremendous computer power that would be needed.

2.1.2 Merkle tree

A Merkle tree[**?**][**?**] or hash tree is a binary tree where every leaf node (nodes at the bottom of the tree) is the cryptographic hash value of some data or transaction. A merkle tree is constructed bottom-up; the upwards nodes are the cryptographic hash values of its two child nodes. Therefore, the root node will contain a complete fingerprint of the entire set of transactions.

Merkle trees allow secure and efficient data verification using cryptographic hash functions. For example; if a participant in the network needs to verify the validity of these four transactions, see figure 2.2, only a check of the root hash value is necessary. Due to the Merkle tree structure, if any of the data blocks is modified, so will be the root hash value. Furthermore, Merkle tree is a very efficient data structure; for instance, checking that a given transaction is included in the tree will take a maximum of $2log(N)$ operations.

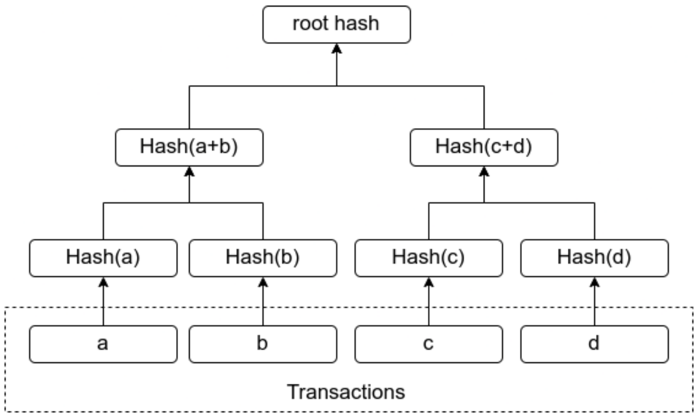

Figure 2.2: Merkle tree of four transactions

2.1.3 Consensus Protocol

The consensus protocol[**?**] of a blockchain network gives a specific rule for verifying whether a transaction is valid or not. As mentioned in 2.1, any participant or node of the network, depending on the blockchain type, can append a new block. For the reason that blockchains typically do not have a centralized authority validating transactions, participants on the blockchain must verify any transaction according to the set of rules or consensus protocol of the blockchain. The most common consensus mechanisms nowadays are:

Proof of work (PoW) In a PoW blockchain, nodes have to solve a cryptographic task in order to validate a block, the first node to find a solution can submit the transaction. Most of the first blockchains in the space use proof-of-work-based protocols; for example, Bitcoin uses PoW based on the cryptographic hash function SHA-256.

Proof of state (PoS) In a PoS blockchain, the entity that can validate a transaction is randomly selected, depending on the "stake" that a node has on the network. Since nodes have a large stake in the blockchain, they will pursue the integrity of the network.

2.1.4 Public-Key Cryptography

In order to interact with a blockchain, participants use public key cryptography or asymmetric key cryptography[?]. The public key acts as the id of the sender or receiver of a given message; and it can be shared with others without any security downfall. On the other hand, the private key should be kept secure and private by the message sender; it is used to sign (digital signature) the messages or transactions, i.e., give permission to change the state of the user records in the blockchain.

Digital signature: The message or transaction can only be signed with a private key, and anyone who knows the public key of the signed message can verify its authenticity. Meaning that any party of the blockchain can participate in the verification of the transactions/messages.

The private key is a random number and the public key is generated from it using one-way cryptographic hash functions, see 2.1.1 or elliptic curve cryptography[?][?]. Elliptic curve cryptography (ECC) is based on the discrete logarithm problem and is the most common public-key algorithm in blockchain networks. In mathematics terms, an elliptic curve is a plane curve over all the points (x, y) that satisfy the following equation: $y^2 = x^3 + ax + b$

2.2 Ethereum

The Bitcoin[?][?] network and its set of technologies enabled for the first time a decentralized, permissionless, trustless, peer-to-peer digital currency system. The distributed computation system that bitcoin introduced: a proof-of-work algorithm to produce a consensus in a distributed system without a central trusted authority, in combination with the blockchain storage system, solved the double-spend problem and the Byzantine generals problem, enabling other applicability beyond digital currencies. In 2014, Vitalik Butering introduced Ethereum[?], a platform that moved beyond cryptocurrency applications.

Similar to Bitcoin, Ethereum[**?**][**?**] is a distributed transaction-based state machine. However, it is not only a cryptocurrency system. Ethereum is a decentralized computing platform or a decentralized, pseudo-turning complete virtual machine, which runs smart contracts and stores its state changes in its blockchain. Like bitcoin, state changes in ethereum are managed by a consensus mechanism, where everyone can participate in the validation of blocks or state changes.

2.2.1 Accounts

Ethereum[**?**][**?**] has an account-based model, where every account represents a state. An account is mapped with a 160 bits address (public key) and there are two types of accounts: *Externally owned account* (EOA), controlled by a private key and (smart) *contract account*, controlled by EVM code. Depending on the account type, an account state is represented by the following four fields: **Nonce** is a counter that represents the number of transactions or contracts created by the account. **Balance** is the amount of Ether owned by the account, expressed in Wei 2.1. **Storage hash** is the root hash of a merkle tree that represents the contents of the account. **Code hash** is the hash that points to the EVM code that the account has.

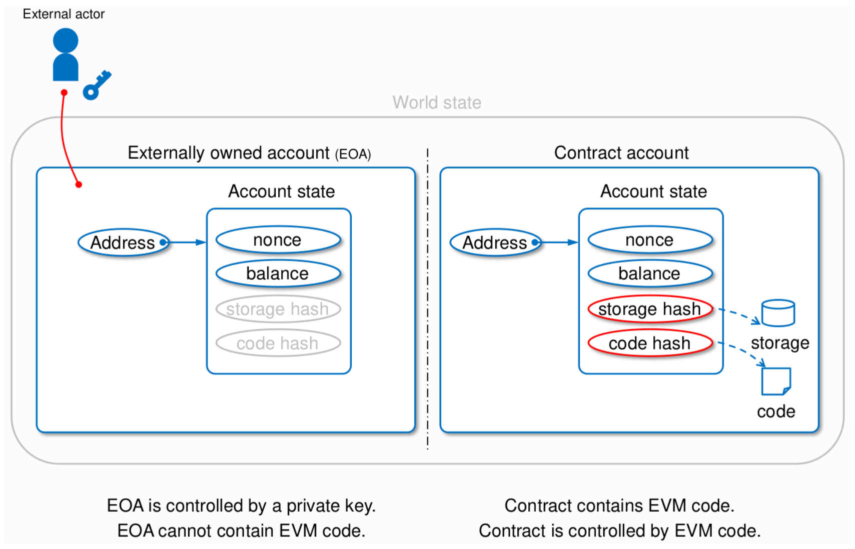

Figure 2.3: Ethereum Accounts[**?**]

Transactions and Messages

A **transaction** is a cryptographically-signed message that can only be sent by an EOA. Transactions can trigger a state change on the blockchain or the execution of EVM code. Transactions can be divided into two groups:

- **Message calls**: The transactions between accounts.
- **Contract creations**: creation of a new account with EVM code in its data/code field.

Messages can be seen as internal transactions between contracts triggered by a message call. A message is like a transaction but produced by a contract. This way messages enable contracts to interact between them.

2.2.2 Ethereum Virtual Machine (EVM)

The EVM[**?**][**?**] is a quasi-Turing-complete machine; the quasi means that its computation is limited by the available **gas** 2.2.2 that the executed contract bytecode has, preventing that accidentally or malicious contracts execute forever. Thus, denial-of-service attacks are not possible on Ethereum. The EVM is a stack machine with 1024 elements. It is big-endian by design with 256-bit word size, which facilitates hashing and elliptic curve operations (Keccak-256 hashes and secp256k1 signatures). In addition to the **Stack**, the machine has two more spaces where operations can access and store data: **Memory**; volatile and initialized to zero. **Account storage**; non-volatile and maintained as part of the Ethereum state, also initialized to zero. Lastly, the EVM code is stored in a separate virtual read-only memory (**ROM**).

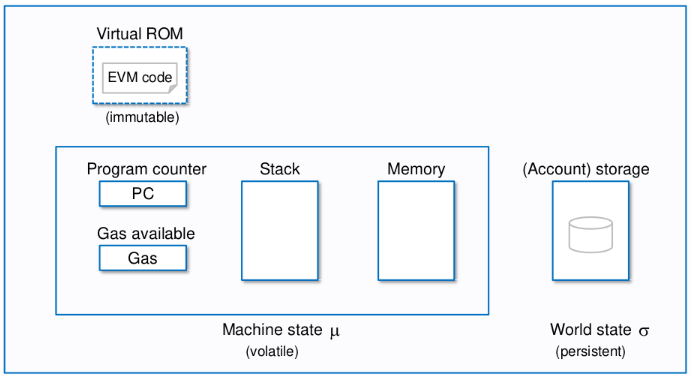

Figure 2.4: Ethereum Virtual Machine[**?**]

Ether and Gas

Ether (ETH) is the native cryptocurrency of Ethereum. Ether is used to pay for transaction fees or EVM computing power in the form of gas units. For example, the cost or gas limit for sending a transaction is 21000 gas units[**?**]. The price for a gas unit is represented in gwei, and is set by the participants of the consensus protocol (miners), based on the supply and demand of the network computational power[**?**]. For instance, with an average gas price of 15 gwei[**?**], sending a transaction at the moment of writing this Thesis would cost: $21000 * 15 = 315000\ gwei$.

Unit Name	**Wei Value**
Ether	1^{18}
Gwei	1^{9}
Wei	1

Table 2.1: Important metrics of *Ether*

2.2.3 Smart Contracts

In the Ethereum network, a smart contract[**?**][**?**] is a computer program that is executed by the EVM. Since smart contacts are also a type of account, see 2.2.1, they have a balance and can send transactions in the form of messages. Furthermore, smart contracts have the following properties:

Immutability
When a smart contract is deployed to the Ethereum platform, the code can not be changed. If a modification (fixing a bug or adding new functionality) needs to be made, a new smart contract needs to be deployed.

Determinism
Given an input and the blockchain state, the result of the execution of the smart contract has to be the same for everyone who calls the contract.

Limitations
By design, smart contracts can not send $HTTP$ requests; this can be solved using decentralized oracles[**?**]. Furthermore, the context of execution is limited to the EVM, meaning that they can only access their own state and the transaction context of the transaction caller. Lastly, smart contracts can not exceed the size of $24KB$.

Lifecycle
Once deployed, smart contracts can no be modified. However, they can be deleted if the smart contract was programed with the $selfdestruct()$ function.

Composability
Since smart contracts can call other smart contracts, functionality of different contracts can be combined. However, such interaction between contracts has to be triggered by a transaction sent by an EOA.

Permissionless
Anyone can deploy a smart contract to the Ethereum network.

Smart contracts can be written in EVM bytecode[**?**]. However, they are usually written in high-level programming languages such as *Solidity*, *Vyper*[**?**] and *Fe*[**?**], among others, being Solidity, the most widely used programming language for smart contracts in EVM-based blockchains. **Solidity**[**?**][**?**] is an object-oriented or contract-oriented high-level programming language; it was created by Dr Gavin Wood and designed to target the EVM. The Solidity compiler *solc* converts Solidity contracts into EVM bytecode and also generates the contract application binary interface (ABI)[**?**], which enables the interaction with contracts in both directions; from outside the blockchain and from contract to contract.

2.2.4 Tokens

Tokens have a widely used description[**?**], in the context of Ethereum; a token[**?**] is a digital representation or abstraction of something that lives on the Blockchain. For instance, tokens can represent currencies, shares in a company, or even a virtual pet. Unlike Ether, which is managed by the Ethereum protocol, tokens are created and handled by smart contracts. Everyone can create a token. Nonetheless, because a smart contract can be programed without any restriction, there are standards that need to be followed for the creation of tokens. The two most used standards are:

The **ERC20** Token Standard[**?**], for the representation of equivalent and interchangeable tokens (fungible tokens). For instance, a token that represents a currency or a share in a company. As defined in the EIP[**?**], the ERC20 contract is composed of the following elements:

Must have methods	**Optional methods**	**Must have events**
$totalSupply()$	$name()$	$Transfer()$
$balanceOf()$	$symbol()$	$Approval()$
$transfer()$	$decimals()$	
$transferFrom()$		
$approve()$		
$allowance()$		

Table 2.2: Components of the ERC20 Standard.

The **ERC721** Token Standard[**?**], for non-fungible tokens or representation of unique goods like an ID or collectibles.

2.2.5 Decentralized Applications (Dapps)

Unlike a web application, a Dapp has its backend running on a decentralized per-to-per network such as Ethereum, inheriting the advantages and drawbacks that the network has.

Chapter 3

Decentralized Finance (DeFi)

3.1 Maker Protocol

3.2 Aave Protocol

Chapter 4

Application Requirements

This Chapter describes the application requirements based on ISO/IEC/IEEE 29148:2018[?] Software Requirements Specification standard.

Project: Thesis.

4.1 Introduction

4.1.1 Purpose

The main purpose of this Chapter is to provide a detailed description of the Decentralized application developed for this Thesis.

4.1.2 Scope

Requirements for a decentralized application that can provide permissionless financial services using an evm based blockchain.

The minimum viable functionality that the application shall offer the users are:

- Sign in with the metamask wallet.
- See which tokens can be deposited.
- Using Aave: Deposit cryptoassets and earn interest.
- Using Aave: Withdrawal of deposited assets.

The application runs online; its frontend may not be decentralized, but its backend runs on top of an EVM-based blockchain such as Polygon Testnet or Mainnet.

4.1.3 Product Perspective

Interface

The application runs in the latest version of Chrome, Firefox, Brave browsers on Windows, Linux and Mac.

4.2 Requirements

4.2.1 Functional

Login

[Thesis-SRS-01] The application shall allow users to login using the Metamask[**?**] wallet.

[Thesis-SRS-02] If the wallet is set with another network, the application should let the user change to the application network with just one button click.

[Thesis-SRS-03] The application shall display the user balance of the supported assets.

[Thesis-SRS-04] The application shall display the user balance of deposit assets and earnings.

[Thesis-SRS-05] The application shall display the interest rates for the supported assets.

Deposit

[Thesis-SRS-06] The application shall allow users to deposit Ether to earn interest.

[Thesis-SRS-07] The application shall allow users to deposit DAI to earn interest.

[Thesis-SRS-08] The application shall convert Ether to Weth, because the protocol only accepts Ether in the form of ERC20, i.e. Weth.

[Thesis-SRS-9] The application shall display the user balance after any deposit transaction.

Withdrawal

[Thesis-SRS-10] The application shall allow users to withdraw deposited cryptoassets.

[Thesis-SRS-11] The application shall display the user balance after any withdrawal transaction.

4.2.2 Nonfunctional

Login

[Thesis-SRS-12] While not logged in, the application should display the available assets to be deposited and its interest rates.

Performance

[Thesis-SRS-13] The transaction cost should be minimized by using the Polygon network.

4.3 Use Case

The following use case diagram describes the minimum functionality that the application shall have.

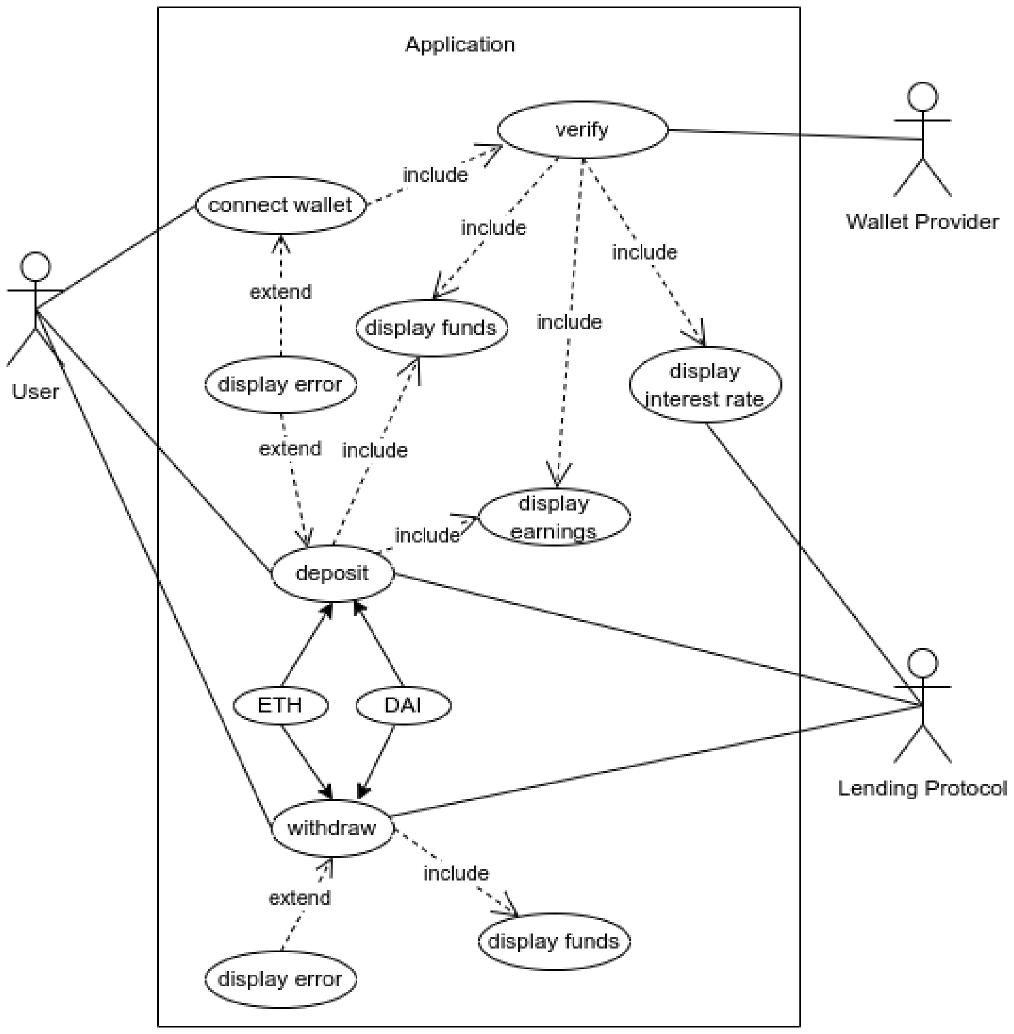

Figure 4.1: Use Case Diagram: Minimum Viable Product.

If the development process goes well and there is enough time, the application should add more functionalities as illustrated in the following use case diagram. The requirement document/chapter[**?**] should also be updated.

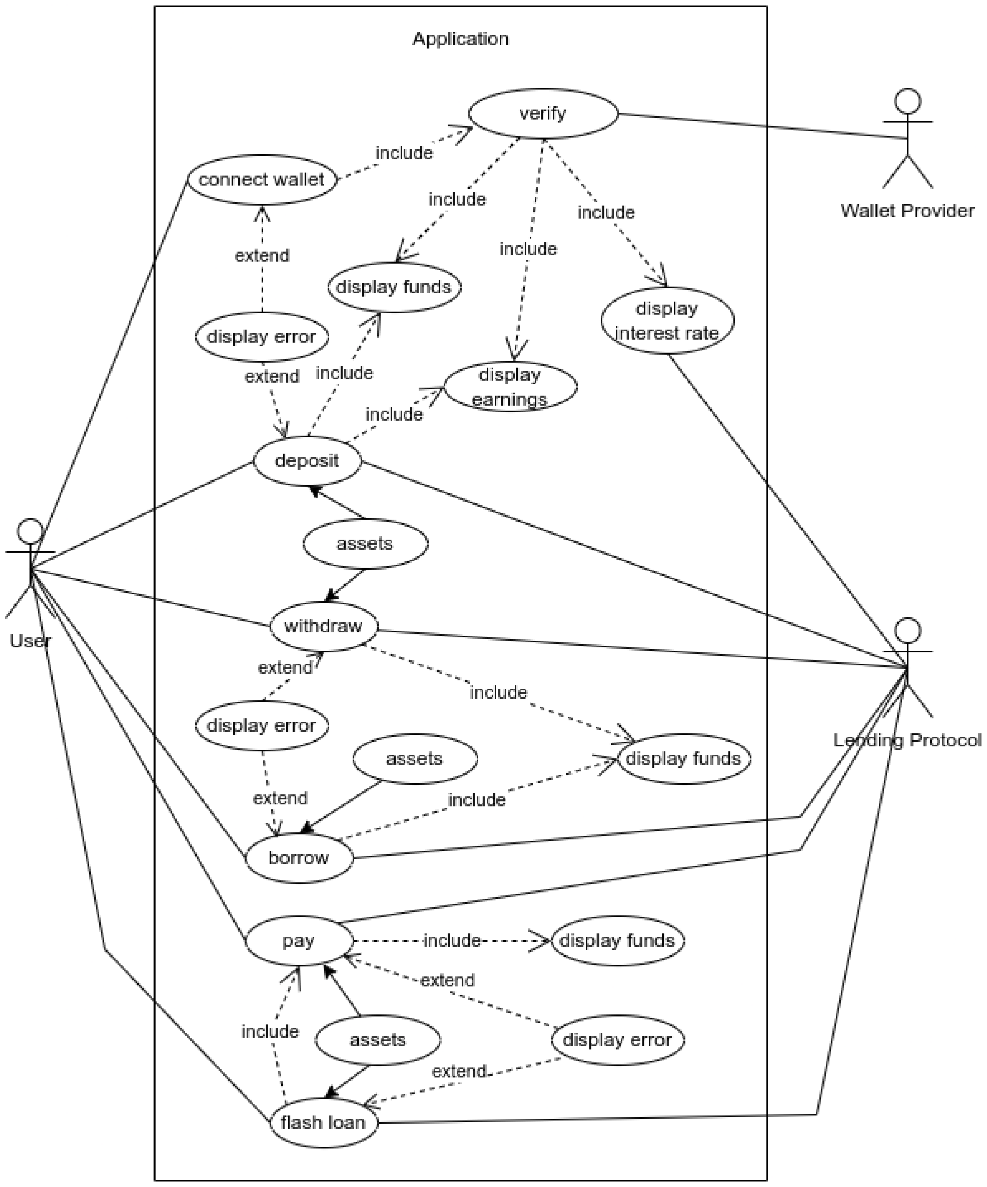

Figure 4.2: Use Case Diagram: Expected Application Functionality.

Chapter 5

Implementation and testing

5.1 Defining the tech stack

5.1.1 Intearction with the smart contracts: Backend

Web3.py : *Brownie*

Testing : *pytest*

5.1.2 Interaction with the users: Frontend

Ethers.js : *wagmi*

Next.js

Testing : *mochajs*

Chapter 6

Conclusion and future work

Bibliography

[1] Andreas M. Antonopoulos. *Mastering Bitcoin*. O'Reilly Media, Inc., first edition, December 2014.

[2] Andreas M. Antonopoulos and Dr. Gavin Wood. *Mastering Ethereum*, pages 1–6. O'Reilly Media, Inc., November 2018.

[3] Andreas M. Antonopoulos and Dr. Gavin Wood. *Mastering Ethereum*, pages 297–314. O'Reilly Media, Inc., November 2018.

[4] Andreas M. Antonopoulos and Dr. Gavin Wood. *Mastering Ethereum*, pages 314–317. O'Reilly Media, Inc., November 2018.

[5] Andreas M. Antonopoulos and Dr. Gavin Wood. *Mastering Ethereum*, pages 127–134. O'Reilly Media, Inc., November 2018.

[6] M. Poongodi Bharat S. Rawal, Gunasekaran Manogaran. *Implementing and Leveraging Blockchain Programming*, pages 12–17. Springer Singapore, January 2022.

[7] M. Poongodi Bharat S. Rawal, Gunasekaran Manogaran. *Implementing and Leveraging Blockchain Programming*, pages 26–27. Springer Singapore, January 2022.

[8] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. 2014. available at `https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf`.

[9] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, pages 99–111, 1991. available at `https://link.springer.com/article/10.1007/BF00196791`.

[10] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*, pages 321–325. CRC Press., 1997.

[11] Ralph C. Merkle. Protocols for public key cryptosystems. pages 122–122, 1980.

[12] Pratyusa Mukherjee and Chittaranjan Pradhan. *Blockchain 1.0 to Blockchain 4.0—The Evolutionary Transformation of Blockchain Technology*, pages 29–49. Springer International Publishing, Cham, 2021.

[13] Rajeev Sobti and Geetha Ganesan. Cryptographic hash functions: A review. *International Journal of Computer Science Issues, ISSN (Online): 1694-0814*, Vol 9:461 – 479, 03 2012.

[14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Berlin Version 934279c.

[15] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Berlin Version 934279c, Apendix G. Fee Schedule.

[16] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. 06 2017.

Online Sources

[17] Contract ABI Specification. https://docs.soliditylang.org/en/latest/abi-spec.html. [Accessed 18 Apr. 2022.].

[18] Ethereum virtual machine opcodes. https://ethervm.io/. [Accessed 18 Apr. 2022.].

[19] Gas tracker. https://etherscan.io/gastracker. [Accessed 17 Apr. 2022.].

[20] Introduction to smart contracs. https://ethereum.org/en/developers/docs/smart-contracts/. [Accessed 18 Apr. 2022.].

[21] ISO. https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:29148:ed-2:v1:en. [Accessed 21 Apr. 2022.].

[22] A next generation, statically typed, future-proof smart contract language for the ethereum virtual machine. https://fe-lang.org/. [Accessed 18 Apr. 2022.].

[23] Oracles. https://ethereum.org/en/developers/docs/oracles/. [Accessed 18 Apr. 2022.].

[24] Solidity. https://docs.soliditylang.org/en/v0.8.13/index.html. [Accessed 18 Apr. 2022.].

[25] Tokens. https://docs.openzeppelin.com/contracts/4.x/tokens. [Accessed 18 Apr. 2022.].

[26] Vyper. https://vyper.readthedocs.io/en/latest/. [Accessed 18 Apr. 2022.].

[27] Vitalik Buterin Fabian Vogelsteller. EIP-20: Token standard. https://eips.ethereum.org/EIPS/eip-20, 2015. [Accessed 18 Apr. 2022.].

[28] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf. [Accessed 5 Apr. 2022.].

[29] Takenobu T. Ethereum evm illustrated. https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf. [accessed 15-April-2022].

[30] Wikipedia. Elliptic-curve cryptography — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Elliptic-curve%20cryptography&oldid=1081874678, 2022. [accessed 13-April-2022].

[31] Wikipedia. MetaMask — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=MetaMask&oldid=1083935548, 2022. [accessed 22-April-2022].

[32] Wikipedia. Token — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Token&oldid=1066372445, 2022. [Accessed 18 Apr. 2022.].

[33] Dieter Shirley William Entriken (@fulldecent). EIP-721: Non-Fungible Token Standard. https://eips.ethereum.org/EIPS/eip-721, 2018. [Accessed 18 Apr. 2022.].