A BLOCKCHAIN APPLICATION FOR PAYMENT AND TRAFFIC MANAGEMENT IN SMART VEHICLES

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
Izmir Institute of Technology
in Partial Fulfillment of the Requirements for the
Degree of

MASTER OF SCIENCE

in Computer Engineering

by
Boğaçhan YİĞİTBAŞI

July 2022
İZMİR

ACKNOWLEDGMENTS

I would like to thank to my advisor Assoc. Prof. Dr. Tolga AYAV for all his guidance and effort through this journey and appreciate his suggestions to turn this idea into a project and therefore a company.

I would like to thank to my school which has been a home to me for more than 12 years. I will never be able to repay its generosity, indulgence and vision. Hopefully, one day I may have a chance to pay all back to my country with my achievements.

Finally, I would like to thank to my wife who has never stopped her endless support for every adventure in my career.

ABSTRACT

A BLOCKCHAIN APPLICATION FOR PAYMENT AND TRAFFIC MANAGEMENT IN SMART VEHICLES

The proposed solution offers an alternative way to our current retail shopping of fuel fees. It can be applied to any retail shopping process but this phase of the project is considered an initiation of upcoming. The next phases of the project, include full integration with smart cars in order to handle all procedures automatically. In the traditional way, when you buy some gas from a station with your credit card, the station owner pays some fee to his bank and it has to wait for some time to be able to get that money. You as an individual have to expose your identity so they can track your shopping habits and follow your expenses. Sometimes they may offer some loyalty discounts or gifts but with really ridiculous rates. Our system offers a digital payment system based on the Ethereum blockchain. It has its own token called TRANT (Transport Token) and by this token, any client with a digital wallet (Metamask) is able to pay their gas fees without exposing their real identity -only their wallet address-, and get some rewards in terms of TRANT for their loyalties and using our DEX (Decentralized Exchange) exchange them into the ether which can be converted into real fiat money easily. On the other hand, the proposed solution also has some advantages for the other party in this equation such as gas station owners, they immediately get their money at that very first moment without any remittance.

ÖZET

AKILLI ARAÇLARDA ÖDEME VE TRAFİK YÖNETİM BLOKZİNCİR UYGULAMASI

Önerilen çözüm, geleneksel olarak kullanılan perakende benzin satışına bir alternatif olarak geliştirilmiştir. Geleneksel yöntemde kredi kartınızla benzin aldığınızda benzin istasyonu bankaya bir miktar ödeme yapar ve paranın hesabına geçmesi için bir süre bekler. Alışveriş alışkanlıklarınızı ve harcamalarınızı takip edebilsinler diye gerçek kimliğinizi başkalarıyla paylaşmak zorunda kalırsınız. Bazen bunu kampanya üyelik kartları ile küçük hediyeler ve indirimler vererek yaparlar. Önerilen dijital ödeme sistemi Ethereum blokzinciri tabanlı bir sistemdir. TRANT adı verilen kendi token'ını kullanır ve bu token aracılığı ile herhangi bir kullanıcı bir sanal cüzdan yardımı ile, kimliğini açığa çıkarmasına gerek kalmadan yakıt ödemelerini yapar. Bireysel kullanıcılarımız servisi kullandıkları için, yine TRANT cinsinden bir ödül kazanırlar ve bu TRANT token web sayfamızdaki merkeziyetsiz borsa aracılığı ile ether'e dönüştürülebilir. Ether tüm borsalarda işlem gören ve kolayca gerçek paralara dönüştürülebilen bir kripto para birimidir. Benzin istasyonu sahiplerine sağlanan önemli avantajlarsa; para hesaplarına hemen geçer ve herhangi bir kesinti ücreti ödemezler.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With the advent of Bitcoin, humankind is able to imagine more than today's world gifts in terms of financial technologies such as traditional finance, centralized databases, centralized exchanges and payment systems. For almost five centuries, we have been relying on central banks for even smaller amount of transactions as intermediaries. These intermediaries still today keep our money for a while, charge us as with a remittance and transactions make up to days. Last couple of years have proven to us we are able to say no to this exploitation and it articulates definitely we are on the edge of a new era with this amazing technology. Decentralized Applications (dApps) are the applications that uses blockchains as their backends, through an API able to communicate with it. Why are they so popular? Because they are decentralized in nature which provides lots of advantages including following: they are hard to hack, they can not be turned off by someone maliciously or unintentionally, they are immutable, and the integrity of the data is preserved by a very easy yet effective way.

1.1. Aim of the Project

The purpose of the application is to demonstrate that blockchain technology can easily adopt to our lives including retail shopping. There is an increasing demand to smart payment methods with mobile devices, web applications through a wallet. To participate a blockchain network as a user you only need a wallet. Using this wallet, you are able to send transactions to anyone in the world in seconds.

For this purpose, we implement an application that is using Ethereum Kovan chain as it's backend. This web application is an interface that simply act as a fuel or electric car charging station. The station only accepts its governance token called TRANT which is a digital asset created on Ethereum chain with the ERC-20 standard. Through the user

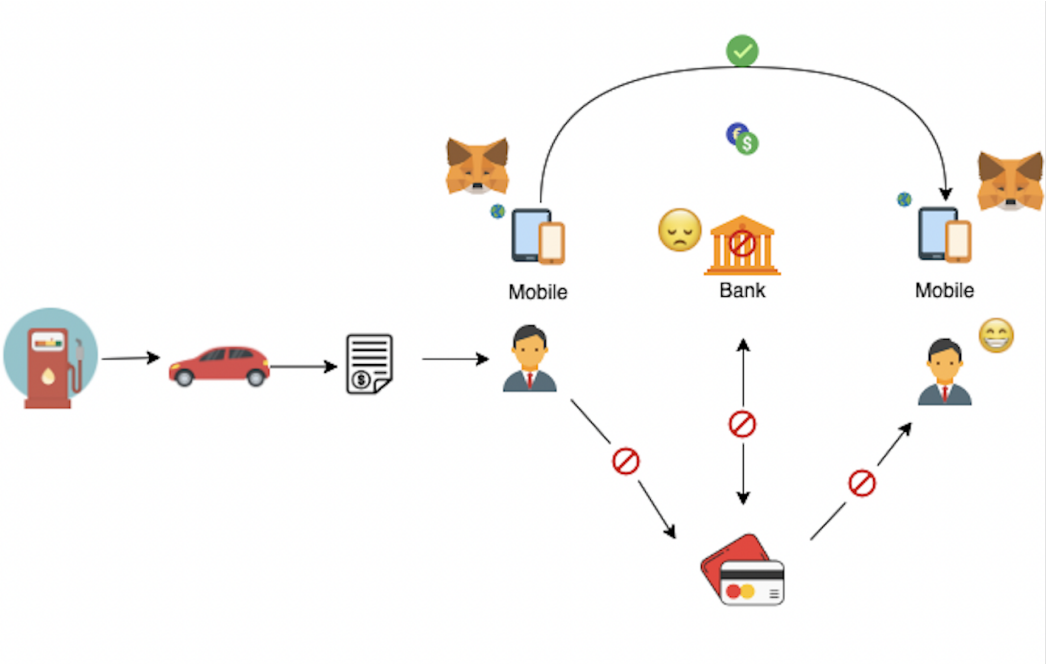


Figure 1.1. Proposed Solution

interface, any user is able to exchange their ether (currency in Ethereum) with TRANT. When a user wants to buy a gasoline, he/she should enter the amount of it in terms of liter and application will calculate the cost. Through this process, the application will connect an oracle to get the price feed of GAS/USD and ETH/USD. It requires a Metamask wallet from the user and with a connect button user is able to connect to it. Overall proposed solution is described in Figure 1.1

This procedure in traditional world takes a couple of steps that includes banks as the intermediaries and has some disadvantages for example from the user point of view revealing identity, sharing personal data without anything in return and from the station owner point of view remittance fee, commission, transaction time. We depicted traditional payment in Figure 1.2.

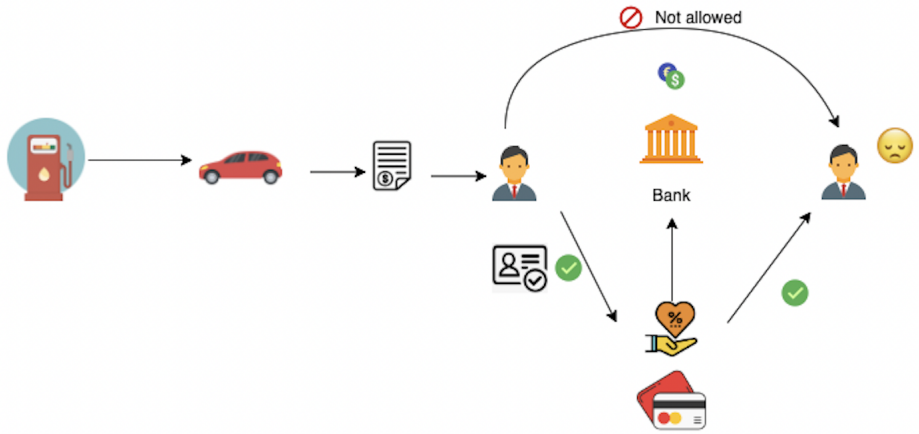


Figure 1.2. Traditional Payment in Gas Station

1.2. Organization of Thesis

In Chapter 2, we described a general overview about the blockchain technology, it's backbone and detailed information about Ethereum. This chapter entails technologies that are used in the application. Chapter 3 is created to describe how the application is implemented both on the backend and frontend. Lastly, our conclusions are given in Chapter 4.

CHAPTER 2

TECHNOLOGICAL STACK

2.1. What is blockchain?

In 2008, there was an anonymous white paper leaked to a mail group. It was describing a peer-to-peer digital cash payment system. This revolutionary invention was targeting central modern-day banking and it proved that there is no need to depend on it anymore. In fact, we need banks for a couple of reasons, and obviously, the first one is trust. This emerging new generation of money called Bitcoin Nakamoto (2008) which was described in the paper was solving this trust issue in a very logical and easy way with the technology it's behind. Unlike the modern-day central banking system, it does not require any money to transfer from one account to another. It does not require time to send out EFTs anywhere, the money does not kept by anyone in between and you can trace the track of the transaction and you still remain anonymous. This game-changer invention depends on a very well-established technology called the blockchain. Blockchain is a form of a linked list as a data structure that has nodes linked to one another. Basically, it is an append-only, timestamped structure. These nodes entail a pointer to the previous one, a hash value of the data which is an output of the hashing algorithm that is implemented by the chain, Merkle root value which is again a hash value - it's a root of a constructed hash tree called Merkle tree that can be used for finding whether a data is in a dataset in logarithmic time -, data itself, and a value of nonce. Nonce value has an importance in this architecture because of the well-known procedure with a worth of billions called mining. Mining procedure can be defined as an ongoing activity to find a solution to an algorithmic problem or in Bitcoin's case encrypted string puzzle output. When a transaction or any kind of data comes into a transaction pool, a miner steps up and starts to calculate a hash value. Every time it calculates, increases the value of nonce by one appends it to the

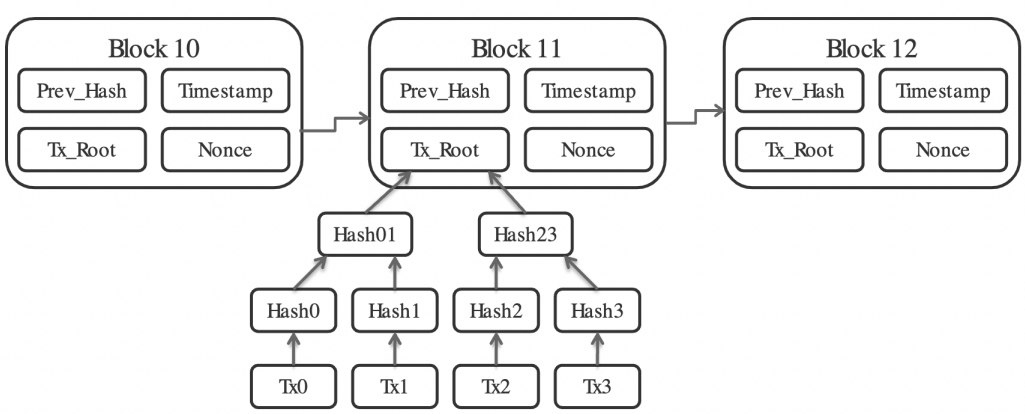


Figure 2.1. Blockchain Architecture

hash value and re-hash it. This procedure continue till the expected value is calculated, in Bitcoin's case for example this output must have at least 10 leading zeroes in its calculated hash value to be accepted as valid. Difficulty level changes time to time according to the Bitcoin network's traffic and the number of miners based on blockchain's algorithm. Mining process is a competition among all of nodes who are trying to validate data and its result is a reward for the first one who has managed to solve the cryptic puzzle. Every participant of the network has a copy of the transactions and anyone can see the transaction history and details using a block explorer and because of it's decentralization, and hash constructed baseline we can easily say that it is tamper proof. If you want to change any value prior in the set, you have to verify all blocks' current hashes and previous hashes one by one, you have to change all of them. And even if you could for example, if it is a local computer, then you still need to make these same changes all around the world because copy of data is everywhere. Based on these very simple conditions we can say that it is secure, immutable and transparent.

2.2. Consensus Algorithms

Modern day consensus algorithms depend on the permissioned networks. In a permissioned network, you have a set of nodes and every party knows the others because

there is a central authority which governs the network. A node who wishes to join the network must get a permission from it. However, apparently this has a huge drawback when you want to provide anonymity, or it strongly violates one of the most prominent features of blockchain which is decentralization since requires custody of these fedarators. An authority is equal to the definition of centralization. Satoshi Nakamoto solved this problem by implementing a Proof of Work (PoW) based permissionless, public consensus algorithm called Nakamoto Consensus Nakamoto (2008). According to his algorithm, any node who wants to become a validator can join this network freely without any kind of permission. And the rest of the network don't have to trust this new node because maliciously or benign either way it has to solve the cryptic puzzle. There are more eco-friendly alternatives to PoW such as PoS (Proof of Stake), Proof of Authority (PoA), and PoA (Proof of Activity). Because no matter how many validators worked to find the correct hash value, only one of them gets the reward after the majority agreed with it and the rest's the effort, time, and electricity kind of resources are wasted. As a result, PoS based consensus blockchains have dominated the crypto ecosystem thanks to their environmental awareness, transaction per second (TPS), finality and latency advantages. The right to validation of a block is given by the staked amount of that validator. If a node wants to validate a block, it needs to increase the staked amount so that its chances will be increased, and based on this, the possibility of being randomly selected increases. The staked amount is most of the time locked in as collateral in an escrow contract, to punish the validator in case it acts maliciously.

2.3. Ethereum

According to its founder WOOD (2014), Ethereum initially used PoW and was giving 5 ether (currency of Ethereum blockchain) as a reward to the validators and in addition to that, there is a total gas fee to add on. Gas is an important element of Ethereum chain and in fact, it's crucial for almost all permissionless chains. It is a security mechanism that protects the chain from intentional or unintentional loops. Basically, if either a smart contract or EOA (Externally Owned Account) wants to store a value on-chain

then it has to pay a gas fee that is calculated based on the size of data. According to the Ethereum Community (2022) document the reason of the minimum amount of Gas units equals 21000 is in the following. “21000 gas is charged for any transaction as a ”base fee”. This covers the cost of an elliptic curve operation to recover the sender address from the signature as well as the disk and bandwidth space of storing the transaction.“ The main currency in the Ethereum chain is ether. Ether can be exchanged in smaller units like Gwei and Wei and their conversion rate is highlighted as depicted in Figure 2.2.

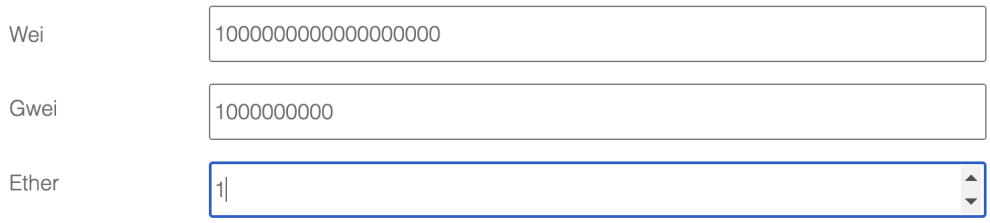


Figure 2.2. Ether, Gwei and Wei Conversion

Changing data on-chain is costly because it changes a state. If there is a transaction that changes a state then it needs to be taken care of in all copies of data amongst all the computers in the network. EVM (Ethereum Virtual Machine) is a state-keeping machine and acts like literally a virtual machine with only one difference it operates all around the world as an interrelated computer system. If a computer wants to join the Ethereum network to validate blocks and earn rewards, it has to install and run a client. These client applications (OETH, Geth) run on that particular machine connect it to the mainnet and from that moment it can start the validation process. EVM itself has no capability to run Solidity codes directly since it is a high-level programming language. Instead, EVM is only able to execute bytecodes therefore Ethereum keeps bytecodes on-chain. When a developer deploys a contract there are at least 3 outputs such as an address of that contract, an ABI (Application Binary Interface) and that bytecode. ABI itself is a JSON formatted object array that simply refers to each function, variable, and returns the value in the contract in other words their bytecode versions. Using this ABI, applications and EVM

can communicate. When a user clicks a button that triggers a function, for example, a transaction, initially that application loads the ABI selects that function to trigger it sends its corresponding index value to the EVM to invoke the relevant bytecode in Figure 2.6.

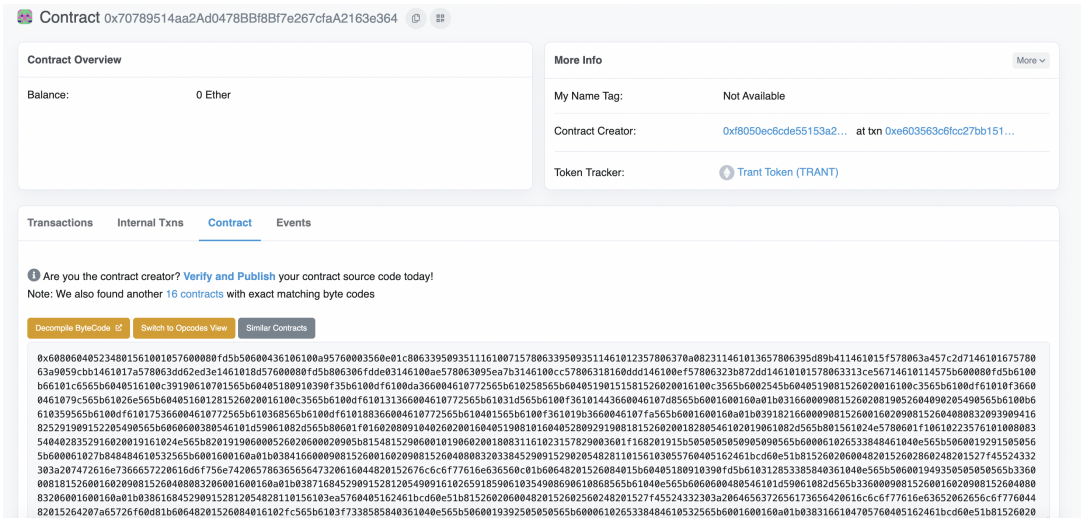


Figure 2.3. Trant Token bytecode on Etherscan

2.3.1. Ethereum Token Standards

ERC-20 is a token standard that every fungible token has to implement. Ethereum provides a baseline platform to create fungible tokens and these tokens are interchangeable with their kind in exchanges. Like fiat money, for example, 10 fungible tokens are equal to the 2 fungible tokens of 5 or 1 fungible token of 10. Bitcoin and ether itself are fungible tokens. These Layer-2's which means living on another Layer-1 create a solution specific economy. The token is a digital asset that is created, minted, and available for transactions on a Layer-1 blockchain. Layer-1 chains have their own standard to mint its own tokens such as BEP-20, CIS-2, and TZIP-16. There are wrappers that wrap up a token from another Layer-1 and hide it in another's form so that people use it. But every token, while it's in the minting process, has to share the same standardized basic functions. It is an interface that makes sure that a token that is tradeable needs to have some default functions built in such as approve(), transfer(), transferFrom(), balance(), owner(),

etc.

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns
(uint256 balance)
function transfer(address _to, uint256 _value) public
returns (bool success)
function transferFrom(address _from, address _to,
uint256 _value)
public returns (bool success)
function approve(address _spender, uint256 _value)
public returns (bool success)
function allowance(address _owner, address _spender)
public view returns (uint256 remaining)
```

ERC-721 is a token standard that every non-fungible token has to implement. An NFT is a tokenized or digitized version of any kind of any physical and digital asset in the universe. The main use case of the NFTs, is creating scarcity and uniqueness for a particular asset. During the Covid-19 crisis, there are a couple of industries suffered from it a lot including artists, singers, etc. Thanks to brilliant artists all around the world, they saw potential and created the first logical NFTs in space. That is why people relate mostly NFTs with digital art. But as humans, like we always do, we have found a way to monetize it to make a quick buck including minting tweets, creating pixel-wise cats, and millions of dollars campaigns of apes. However, tokenizing an asset means more than that. NFTs have a huge amount of potential in terms of creating value for a lot of domains including but not limited to real estate, entertainment, finance, art, music, sports, gaming, metaverse, ESG (Environmental, Social and Governance), etc. We can even turn our identities into NFTs or SBTs (Soul Bound Token) as clearly stated in the following Weyl et al. (2022). SBTs have not gotten traction yet, but they are offering a new kind

of standard for non-transferable tokens to create an identity in the space. For example, a birth certificate, university diploma, or even a passport can be tokenized as SBTs and bound to a wallet for good. Users can use this to validate or verify their identity or as proof of their attendance (Proof of Attendance Protocol) in that sense as pointed out by Sizon and Nayak (2018).

NFT(Non-Fungible Token) standard makes sure that every NFT token have same default functions built in.

```
function balanceOf(address _owner) external view
returns (uint256);
function ownerOf(uint256 _tokenId) external view
returns (address);
function safeTransferFrom(address _from, address _to,
uint256 _tokenId,
bytes data) external payable;
function safeTransferFrom(address _from, address _to,
uint256 _tokenId)
external payable;
function transferFrom(address _from, address _to,
uint256 _tokenId)
external payable;
function approve(address _approved, uint256 _tokenId)
external payable;
function setApprovalForAll(address _operator,
                           bool _approved) external;
function getApproved(uint256 _tokenId) external view
returns (address);
function isApprovedForAll(address _owner,
                                address _operator)
external view returns (bool);
```

2.3.2. Ethereum Testnets and Faucets

In addition to Ethereum's production blockchain called Mainnet, there are alternatives to test smart contracts before they are deployed to production. Because smart contract bug fix is not an easy thing to do in its nature, a deployed contract can be updated with some methods but there are certain conditions that may make this impossible. There are lots of problematic cases and dramatic references about it but any bug or a back door on a smart contract may end up with stealing from the contract or unintentionally locking the assets to it for good as such 300 Million Dollars Lost (2022) and 30 Million Dollars Locked Up (2022). To prevent any kind of this situation Ethereum documentation incentivize the developers to deploy and test their contracts first in these testnets such as Kovan, Rinkeby, Goerli, and Ropsten. These are almost identical to the Ethereum's Mainnet in terms of latency, throughput, block time, etc. There is only a difference which is the consensus algorithm. Testnets (2022b) use Proof of Authority, there are chosen nodes to validate the blocks and create them. There are Testnets (2022a) which are sharing reward cryptocurrencies after completion of some tasks or playing a game. Alternatively, there are faucets that share ethers mined on these test networks. Using these faucets developers get enough funds to test their smart contracts thoroughly.

2.4. Wallets and Metamask

A wallet is clearly stated in Wikipedia contributors (2022) "Cryptocurrency wallet is a device, physical medium, program or a service which stores the public and/or private keys for cryptocurrency transactions." A wallet is a guardian of private or public keys. When a transaction happens of some amount of money, that implies that money is encrypted with the receiver's public key and there is only one person who has the private key and therefore is able to decrypt and use it thanks to PKI (Public Key Infrastructure) in this case the wallet of EOA. They can be either software and hardware or custodial or non-custodial. CEXes (Centralized Exchanges) offers custodial wallets and their private key's security. In these types of exchanges, they create a wallet for the account owner and

keep it in their own custody. Apparently, from a security perspective, this introduces a possibility of vulnerability. Metamask (2022) is a wallet that is used for managing assets on the Ethereum chain, it offers both web extension and mobile application. While setting up your account Metamask demonstrates lots of words (mnemonic phrases) to create your recovery string to memorize or store it. Based on your mnemonic message, it uses that as an input to create your private key and after that your public key. This algorithm was first introduced with Bitcoin and today most blockchains still use the same mnemonic key generation algorithm which is known as BIP-0039.

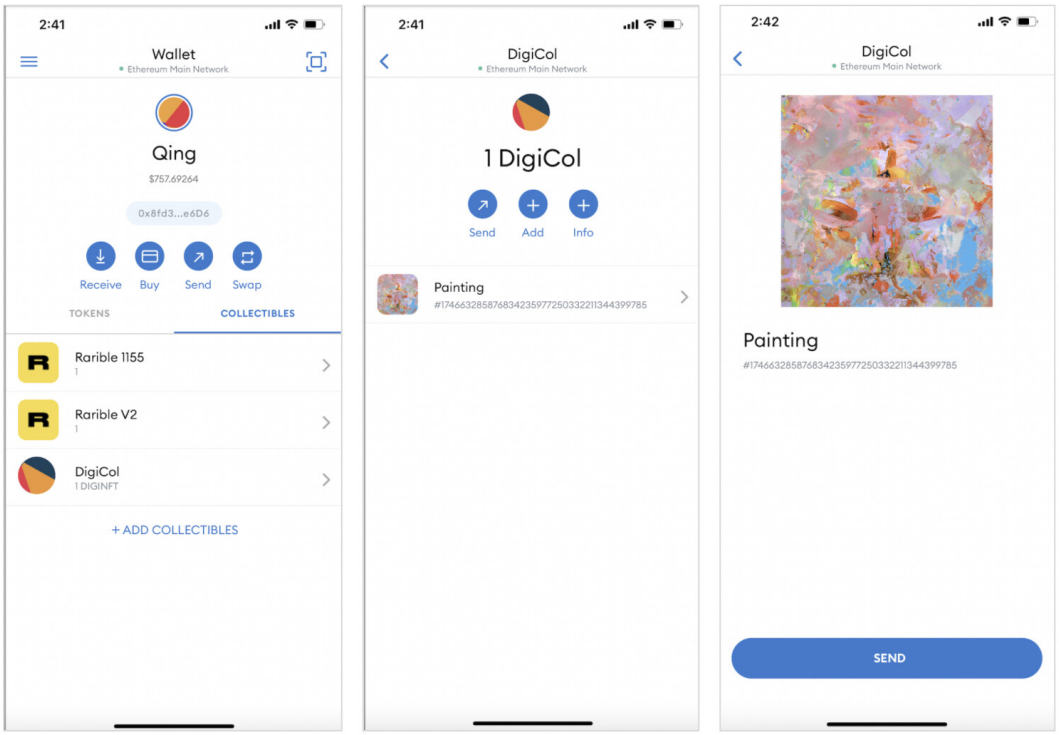


Figure 2.4. Metamask Web Extension Wallet

2.5. Smart Contracts and Solidity

A guy named Nick Szabo who is a lawyer, a computer scientist and the inventor of smart contracts clearly stated that Nick Szabo (Nick Szabo) “The basic idea of smart contracts is that many kinds of contractual clauses (such as liens, bonding, delineation of

Figure 2.5. Exchange Contract Solidity

property rights, etc.) can be embedded in the hardware and software we deal with, in such a way as to make breach of contract expensive (if desired, sometimes prohibitively so) for the breacher."

Smart contracts are the programs coded living on-chain which is implemented to execute the agreements in terms of transaction orders, conditions, or changing ownership of assets by the developers. A smart contract is a set of instructions that requires certain conditions to be executed. According to the Wang et al. (2018), "Smart contracts are computer protocols intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract. Smart contracts have a broad range of applications, such as financial services, prediction markets, and Internet of Things (IoT), etc." In Ethereum, it can be coded in Solidity or Vyper and there are other Layer-1 blockchains that also allow programming smart contracts in C++ (eos), Rust (Solana, Polkadot, Concordium), and WASM (Oasis Protocol, Concordium). While deploying these Turing complete high-level programming languages compile into bytecode and stored on-chain. Any application that wants to invoke a function uses an interface like the ABI and triggers it.

One of the most important things about smart contracts is their immutability because the operating environment is naturally decentralized. Once they are deployed on a blockchain, they live forever and be executed as coded. While deploying them, they are issued to the creator account and produce an address to the contract. In Etherscan which is a block explorer, anyone who wants to see about the holdings of a contract or an account can access the outputs of the block explorers, and thanks to technology's nature,

it is transparent. The code embedded in that contract is also publicly open to any access. This eliminates the trust issue and with the power of transparency, this smart contract technology will change our lives forever including retail shopping, finance, politics, art, real estate, etc.

Solidity is a smart contract programming language developed by the Ethereum community. It is an object-oriented, high-level language for implementing smart contracts and is affected by C++, Javascript, and Python. Solidity Community (2022) A clear definition of Solidity is in the following: ”Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features. Using Solidity, it is possible to create contracts for voting, crowdfunding, blind auctions, multi-signature wallets, and more.”Hegedűs (2019) Its syntax is easy to learn especially for whom someone who comes from the programming field. When we compare it with Rust, certainly it has some performance drawbacks but on the other hand, it is definitely the most well-known and commonly used in the space by the developer community.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.16 <0.9.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

Above contract shows a very simple smart contract written in Solidity. It has an uint storedData and using set() function that stores input on chain. get() function is

responsible for getting the value of that data please note that ”view” function does not cost any gas fee as it changes no state of EVM.

2.6. Oracles

Chainlink (2022), one of the most important oracles in crypto industry explains that “Blockchain oracles are entities that connect blockchains to external systems, thereby enabling smart contracts to execute based upon inputs and outputs from the real world.“

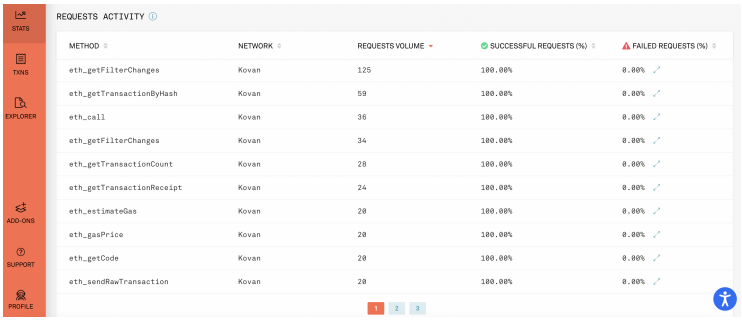
Blockchains and smart contracts are unable to access to the any kind of data which reside off chain. Therefore, according to the implementation they need to be fed by some third parties in order to update their status, exchange rate, result of an election or a game, any kind of sensor data or whether a condition sufficed or not. Based on this information a smart contract can execute it’s promised set of instructions and it may rely on this sensitive data and any wrong information can cause millions of USD. While developing a decentralized app, that is certainly a thing needs to be considered and, in most cases, multiple oracles feed gathered and used after that. Chainlink is one of the most widely used oracles in the world, they have direct integration with Ethereum and many other protocl and they provide aggregators to feed smart contracts with the feeds such as BTC/USD, ETH/USD, USD/EUR. and some functionalities like VRFCoordinator (Verifiable Random Generator)

2.7. OpenZeppelin, Infura and Brownie

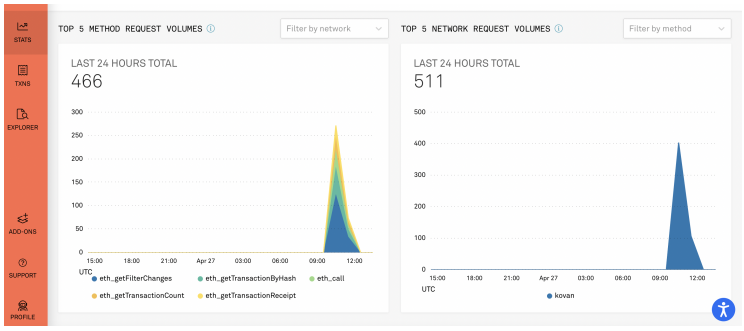
OpenZeppelin (2022) is a library that serves to the developers to implement secure, audited, safe and standardized smart contracts. It provides set of tools and smart contract libraries to implement decentralized applications including minting an ERC20 token with proper set of functions to become valid, ownership contracts to make sure that developers implement more bug free software.

Infura (2022) is a Web3 service provider that is used for deploying smart contracts easily using an API and HTTPS address. It supports multi blockchains including

Ethereum Mainnet, Rinkeby, Kovan, Ropsten, Goerli, IPFS, Filecoin and Ethereum 2.0 and recently scale up to Layer-2 solutions as well like Polygon. For the developers, it also provides a toolset to establish instant connection by using an RPC connection directly.



(a)



(b)

Brownie (2022) is a Python-based development and testing framework for smart contracts targeting the Ethereum Virtual Machine. This development framework recently has started to gainging popularity among developers as it has more developer-friendly usage, Pythonic community. This space has already taken by the Javascript based alternatives like Truffle and Hardhat. Before blockchain hype started, people was lead by the artificial intelligence and machine learning solutions to the python programming language in order to code very complicated software with the ease of it. Because of this, software developers has a huge number of community that includes lots of Python programmers which eventually emerges a smart contract development framework supported by this brilliant programming language.

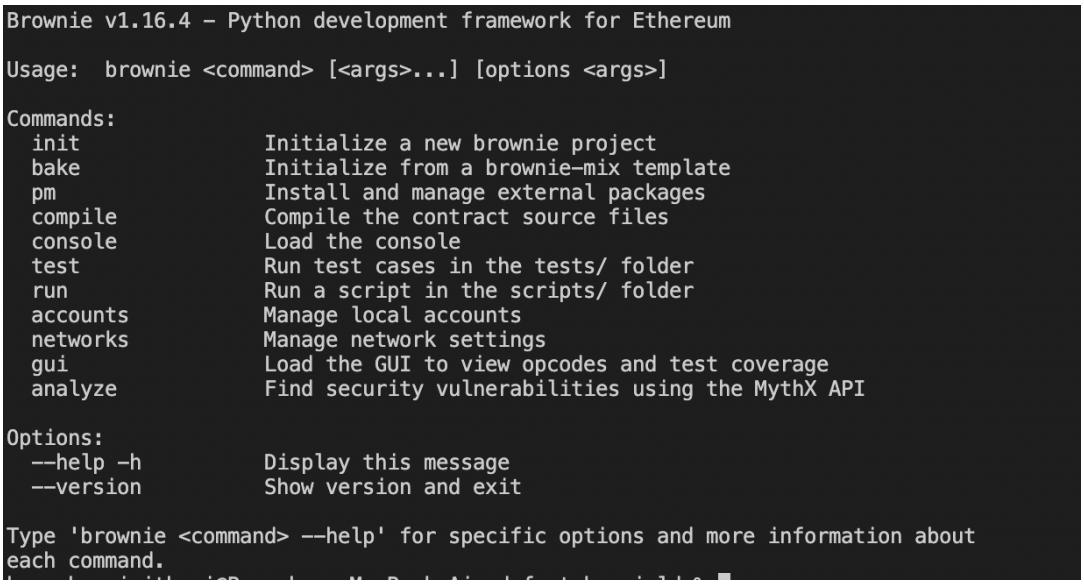


Figure 2.6. Brownie help commands

It supports Solidity and Vyper, enabled pytest to test the contracts and uses Web3.py as a backbone. With the power of developer friendly documentation and easy to use nature it will become much more popular soon.

2.7.1. Ethereum vs Bitcoin? What is the next?

Ethereum and Bitcoin are the two largest blockchains in the space. Bitcoin is the first blockchain and still today has the biggest portion of the crypto-economy. However, when we look at its technology we have very limited transactions per second and long finality. To be more accurate, it takes 10 minutes to process a block and to be completely sure needs 6 blocks to process which makes up to 60 minutes. On the other hand, as mentioned in Das et al. (2019) the smart contracts in the Bitcoin ecosystem do not allow to implementation of complex contracts and it's too expensive to execute. Because of Bitcoin's one specific goal-oriented design, Vitalik Buterin and Gavin Wood designed Ethereum (WOOD (2014)) and these two geniuses did a great job in terms of transaction per second, execution fee, and fully programmable smart contracts with their programming language. The world's computer, Ethereum Virtual Machine (EVM) is able to pro-

cess a block in 14 seconds and transaction per second is 20 on average as described in Etherscan - Blocktime (2022) and Ethereum TPS (2022). The finality will take 6 blocks and therefore you have to wait 84 seconds approximately to be sure.

However, if blockchain technologies are going to create a new world and revolutionize the current economic model then certainly they need to beat VISA. When we compare these with the Visa which is capable to process 2000 transactions so there is a lot to do. Because of the PoW these two brilliant technologies are not competent with VISA at all this makes them not useful for retail payments due to time issues, electricity consumption, and too much transaction cost. Luckily, there are other Layer-1 blockchains like Solana, Avalanche, Algorand, Concordium, etc. These PoS consensus algorithm based blockchains decrease the finality in seconds and increase the amount of TPS to way above than VISA. In addition to that, there are plenty of Layer-2 scaling solutions depending on Zero Knowledge Proofs and new innovations. The future of the blockchain industry is in safe hands.

CHAPTER 3

IMPLEMENTATION OF THE PROJECT

In section 3.1 the source codes of the contracts articulately demonstrated and each functionality, their goal to implement and requirements is defined. Secondly, the section **??** demonstrates how to test a smart contract before deployment, then section 3.3 describes how to deploy contracts using Brownie Framework including configuration file.

3.1. Smart Contracts

All of the contracts that are used in the application complied in solc which is a compiler of solidity with version above 0.8.8. There are alternative compilers both online and offline. Remix IDE (2022) provides an online platform to compile smart contracts using Solidity. While implementing this project Visual Studio Code (VSCode) is used because of it's more developer friendly user interface.

3.1.1. ERC-20 Token Contract

ERC-20 is an Ethereum standard to mint Layer 2 tokens reside on the Ethereum chain. This standard enforces the developers to implement common functions to make sure that minimize the risk of unacceptable and risky creations of tokens. In addition to that, there are lots of implementations of those basic functions but this may introduce bugs. In smart contracts, having a bug sometimes means losing a lot of money. There are institutions like OpenZeppelin who has created libraries that implemented all basic functionalities for standards to help developers. These libraries are more secure in terms of attacks, tested by community and their usage strongly suggested by Ethereum community as well.

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

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract TrantToken is ERC20 {
    constructor() public ERC20("Trant Token", "TRANT") {
        _mint(msg.sender, 1000e18);
    }
}
```

Source code of OpenZeppelin ERC-20 (2022) can be clearly viewed on their Github page. It takes two arguments to constructor which are token name and symbol. mint function requires an address value which is a sender's address and the number of token which is initial supply.

3.1.2. Decentralized Exchange Contract

Decentralized Exchange (DEX) Contract includes basic functionalities of exchanges such as buy and sell. Instead of centralized exchanges (CEX), DEXes are gaining popularity with the increasing demand of the Decentralized Finance (DeFi). DEXes are slightly different in the sense that their architectural design in terms of the centralization, operation. In traditional finance, we have banks to govern all financial actions, it is not possible to participate if you are not following their rules. Centralized Finance (CeFi) requires a bank account therefore Know Your Customer (KYC) that means you have to share all your identity with these institutions. If you want to lend your money, they give you very limited and few amount of interest rates. If you want to use a loan or borrow a some money, they require not only collateral but also your proven financial record and obviously all these serves comes with a price to their pocket.

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

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "./TokenBase.sol";

contract Exchange is Ownable {
uint256 public creationTime = block.timestamp;
uint256 public rate = 1e14; // tokens for 1 eth
uint256 public rateMultiply = 1e4;
event Bought(uint256 amount);
event Sold(uint256 amount);

IERC20 public token;

mapping(address => uint256) tokenPriceFeed;

constructor(address _trantTokenAddress) {
    token = IERC20(_trantTokenAddress);
}

function checkBalance() public view returns (uint256) {
    return address(this).balance;
}

function checkTokenBalance() public view returns
                                            (uint256) {
    return token.balanceOf(address(this));
}
```

```
        //10000 unit buy
    // should equal to 1 ether
function buyToken(uint256 amountInEth) public payable {
    uint256 dexBalance = token.balanceOf(address(this));
    require(amountInEth > 0, "You need to send some ether");

    uint256 tokenReturn = amountInEth / rate;
    assert(token.transfer(msg.sender, tokenReturn));
    emit Bought(amountInEth);
}

function buyTokenOnPayload() public payable {
    uint256 amountInEth = msg.value;
    // uint256 dexBalance = token.balanceOf(address(this));
    require(amountInEth > 0, "You need to send some ether");

    uint256 tokenReturn = (amountInEth * 1 ether) / rate;
    assert(token.transfer(msg.sender, tokenReturn));
    emit Bought(amountInEth);
}


function sellToken(uint256 amount) public {
    uint256 exchangeEth = amount / rateMultiply;
    require(amount > 0, "You need to sell at
                                    least some tokens");
    uint256 allowance = token.allowance(msg.sender,
                                        address(this));
    require(allowance >= amount, "Check the token
                                            allowance");
    token.transferFrom(msg.sender, address(this), amount);
```

```
    payable(msg.sender).transfer(exchangeEth);
    emit Sold(amount);
}

function sendReward(uint256 amount, address user) external
{
    uint256 dexBalance = token.balanceOf(address(this));
    require(amount <= dexBalance, "Not enough tokens
                                        in the reserve");
    token.approve(address(this), amount / 20);
    assert(token.transfer(user, amount / 20));
    emit Bought(1);
}
}
```

Source code of OpenZeppelin ERC-20 (2022) and OpenZeppelin Ownable (2022) can be clearly viewed on their Github page. Exchange constructor takes one argument which is the address of the token that will govern in the exchange. In *buyToken* method, gets the amount of ether sent to buy TRANT token in exchange of with it by the user by *msg.value*. When a transaction interacts with a smart contract to send some ether in Ethereum it has to have at couple of parameters such as sender address (*msg.sender*), receiver address and the amount of ether(*msg.value*). This function simply calculates gets the ether value in payload, calculates the amount of TRANT with respect to the ratio, checks whether the contract has enough about that amount of TRANT. Since this function is belonging to a DEX contract it has no right - at least not implemented- to ming TRANT token. This exchange like the others only a exchange therefore it has to have TRANT to be able to send others. Because of that we need to check it's TRANT balance and if its sufficient, the contract sends them to the sender and finally emitting the Bought event to notify. You can observe the transaction output on both Etherscan and the web application.

sellToken takes an input that is sent by the user, through web application. The amount of the TRANT has a value in terms of ether with respect the ratio. Next thing this function handled is checking the allowence of the TRANT. If you want to transfer a token

on Ethereum chain, as a user you should not only send but also approve the transaction to be able to send by allowance function meaning user account is approving the amount of TRANT can be transferred from his account to any other EOA. Since this is a DEX it has have some ether to be able to buy TRANT from the users. Based on the ratio, this function calculates the amount of the ether to be able to cover the amount of TRANT and if it's sufficient than makes the transaction. After this step, tokens would be transferred into smart contract account and the ether would be sent to the user account.

sendReward function takes amount of TRANT and the address of the user. This function invoked right after the fuel payment in station contract and sends the reward back to the client. Please note that, before send the reward you should approve it first for the sake of transaction.

3.1.3. Station Contract - Token Base Contract

TokenBase contract is the station contract which implements price feed aggregators, keeps the token balances of the stations, station and user lists, checks the payment token type, and handles the rewards after the payment is done. It also has some helper functions to use them while testing the contract. Chainlink oracles are used to get the OIL/USD and ETH/USD values to be able to calculate the gasoline fee. The contract of the aggregator can be found on the following Github page ChainLink Aggregator (2022) . In addition to OpenZeppelin and ChainLink contracts, TokenBase contract also imports the exchange contract to be able to access send, buy, and reward functions. It's constructor expects four input parameters such as priceFeedAddress which is the oracle's contract address to be able to get the value of ETH/USD, priceFeedAddressO which is another contract address of ChainLink oracles to be able to get OIL/USD, _exchange which is the the address of the deployed exchange contract, and _token which is the address of the ERC-20 token's (TRANT) address.

`brownie-config-file.yaml`

includes the address values of the aggregators, while deploying this contract they are embedded to it by reading from the config file. Some functions have onlyOwner modifier

tags for example sendToken that means that that particular functions requires to be invoked by the contract creator, no one else can make the call.

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

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./Exchange.sol";
import "./Manage.sol";
import "@chainlink/contracts/src/v0.8/interfaces
                        /AggregatorV3Interface.sol";


contract TokenBase is Ownable {
address[] public allowedTokenList;
mapping(address => mapping(addresås => uint256))
    public tokenUserBalancePayment;

address[] public users;

address[] public stationList;

AggregatorV3Interface internal ethUSDPriceFeed;
AggregatorV3Interface internal sUSDPriceFeed;

IERC20 public trantToken;

address public exchange;

// There are 2 transfer function in ethereum chain
```

```
// one is the transfer I can use it only I am the owner
// of the contract transferFrom is the publicly used
one. here used only one contract this. But needed to
//expand peer to peer using this contract.

constructor(
    address _priceFeedAddress,
    address _priceFeedAddressO,
    address _exchange,
    address _token
) {
ethUSDPriceFeed = AggregatorV3Interface(_priceFeedAddress);
sUSDPriceFeed = AggregatorV3Interface(_priceFeedAddressO);
trantToken = IERC20(_token);
exchange = _exchange;
}

// tested
function getPriceFeedEth() public view returns (uint256) {
    (,int256 price, , ,)=ethUSDPriceFeed.latestRoundData();
    uint256 adjustedPrice = uint256(price) * 10**10;
    //because returned data is already has 8 decimals
    // return uint256(price);
    return adjustedPrice;
}

//tested
function getPriceFeedS() public view returns (uint256) {
    (,int256 price, , ,)=sUSDPriceFeed.latestRoundData();
    uint256 adjustedPrice = uint256(price) * 10**10;
    //because returned data is already has 8 decimals
```

```
    //return uint256(price);
    return adjustedPrice;
}

//tested
function sendToken(uint256 _amount, address _token) public{
    // what token can they use?
    // how much can they send?
    require(isTKAllowed(_token), "You should send TRANT");
    require(_amount > 0, "Amount must be more than 0!");
    require(trantToken.balanceOf(address(msg.sender))
                                                > _amount);
    trantToken.transferFrom(msg.sender,
                                        address(this), _amount);
    Exchange ex = Exchange(exchange);
    ex.sendReward(_amount, msg.sender);
}

//tested
function enrollStation(address _state) public onlyOwner {
    stationList.push(_state);
}

//tested
function getStation(uint256 index) public view onlyOwner
returns (address) {
    if (stationList.length > index)
        return stationList[index];
}

//tested
```

```
function addAllowedTokens(address _token) public onlyOwner {
    allowedTokenList.push(_token);
}

//tested
function getAllowedTokens(uint256 index)
    public
    view
    onlyOwner
    returns (address)
{
    if (allowedTokenList.length > 0)
        return allowedTokenList[index];
}

// indirectly tested
function isTKAllowed(address _token) public returns (bool){
    for (
        uint256 allowedTokenCounter = 0;
        allowedTokenCounter < allowedTokenList.length;
        allowedTokenCounter++
    )
    {
    if (allowedTokenList[allowedTokenCounter] == _token)
        return true;
    }
    return false;
}
}
```

Source codes of the imported libraries can be viewed on their source which are referenced on previous sections. Function *sendToken* is the payment function, it receives the

TRANT, first check its amount then it's type in order to make sure the transferring token is TRANT. Transfers the token if all required conditions provided to the station account which is given in the constructor. It adds the sender account to the users, to use them in the future for example for an internal lottery. There are some helper and test functions like add addAllowedToken and isTKAllowed, getAllowedTokens to add TRANT as expected token, check if it's TRANT or not and while testing to control incoming with it.

3.2. Unit Testing

In this section, unit testings of each function in smart contracts are performed. As a unit testing library pytest is chosen. "Exchange Test" functions check the balance after some buy and sell operations and "Token Base Test" controls the reward, token balance and compares the two different accounts' balances after buy and sell operations.

3.2.1. Exchange Test

```
from brownie import accounts, network, Exchange, web3
from scripts.deploy import deploy_exchange

from scripts.helper import network, get_EOA
import pytest
from web3 import Web3

# unit test function of buy tokens
# assert in 2 cases such as
# 1 if total supply = inital total + allocated for test
# + sold amount
# 2 if buyer has that amount of token
def test_buy():
    account = get_EOA()
```

```
    non_owner = get_EOA(index=1)
    exchange, trant = deploy_exchange()
    exchange.buy({"from": non_owner, "value": 100})
    assert exchange.checkTokenBalance() ==
                            trant.totalSupply()- 100
    # print(non_owner.balance)
    assert trant.balanceOf(non_owner.address) == 100

def test_sell():
    account = get_account()
    non_owner = get_account(index=1)
    exchange, trant = deploy_exchange()
    exchange.buyToken({"from": non_owner, "value": 100})
    trant.approve(exchange.address, 3e18,
                                {"from": non_owner})
    exchange.sellToken(100, {"from": non_owner})
    assert trant.balanceOf(non_owner.address) == 0
```

3.2.2. Token Base Test

```
# amount, token address, station address
def test_send_token():
    account = get_account()
    non_owner = get_account(index=1)
    token_base, trant, exchange = deploy_token_base()

    trant.approve(token_base.address, 3e18,
                                {"from": account})

    # exchange.approve(account.address, 10
```

```
                            , {"from": account})
    # trant.approve(exchange.address, 3e18
                            , {"from": account})
    token_base.sendToken(100, trant.address,
                    non_owner.address,{"from": account})
    # exchange.sell(100, {"from": non_owner})

    assert trant.balanceOf(non_owner.address) == 100

    # print(trant.balanceOf(account.address))
    assert trant.balanceOf(account.address) == 100
```

3.3. Deployment

Brownie is Python based framework to deploy smart contracts to the Ethereum chain and implement applications working on top of it. In the example below, the exchange contract's deploy function is highlighted. We first call get_account function, which is located in helper's but it's main job is returning an account object by using the inputs from the configuration file

`brownie-config-file.yaml`

. Since the exchange contract requires the address of the TRANT, that means we need to deploy it first to be able to access it's address value. When the TRANT is deployed and it's transaction is placed into the blockchain, we are abale to access it's address to give it as an input of the exchange's constructor. Exchange contract's constructor requires only the TRANT token's address and owner account in our case the "account". The key here is the account because it has to be the same account which is owner of TRANT contract to be able to send tokens to the exchange contract. When exchange is deployed, as owner we call the transfer function to send supply to the contract in order to provide liquidity of TRANT in it otherwise no one can buy from any token anywhere. As you can see we sent all supply we have by calling the

```
trant_token.totalSupply()
```

function which returns the amount of token supplied and set

```
    publish_source=True
```

to allow publishig the contract to make it available to view on the Etherscan block explorer.

```
def deploy_exchange():
    account = get_account()
    trant_token = TrantToken.deploy({"from": account}
                                , publish_source=True)

    print(f"TRANT supply {trant_token.totalSupply()}")

    exchange_ = Exchange.deploy(
        trant_token.address,
        {"from": account},
        publish_source=True
    )
    # sending most of the supply to the exchange
    trx = trant_token.transfer(
        exchange_.address, trant_token.totalSupply()
                , {"from": account}, publish_source=True
    )
    trx.wait(1)

    print(f"Balance {exchange_.checkTokenBalance()}")

    return exchange_, trant_token
```

CHAPTER 4

CONCLUSION AND FUTURE WORK

With this project, we wanted to point out an alternative way to the current payment systems we have. Besides the fact that the blockchains' popularity and hype, their advantages are huge in terms of storing data decentralized and immutable, removing intermediaries and remittances, and for this particular project the need for identity. TRANT requires only a Metamask account from its user, and she doesn't have anything like a credit card with a name on it to pay the gasoline fee. If she would use a credit card, for example, the cashier would know her identity. Users can prefer to use this method as an alternative to prevent any identity theft. On the other hand, the station can sell or use this data to analyze her shopping habits and make more profit without sharing anything with her and/or even not asking permission. For the product aspect, we can say that our proposed solution has characteristics as a proof of concept, we believe that our PoC simply demonstrates that is possible and ready to use for all kinds of retail shopping however, from this point we are aiming to improve the project as a ready to use the product. Our roadmap about the project includes three phases, this thesis work covers the first part as PoC. Secondly, we are going to implement a mobile application and start to collect GPS data from our users. Collection of that data, willingly, will return as a reward to our customers. By using that data, we are planning to implement a machine learning model to forecast traffic. This off-chain model and data provider will be synchronized with the on-chain data and we will be able to access the real data in real-time. After that point, we are going to get into the final phase, turning all these into a navigation application. We are planning to implement a Web3.0 navigation application where users can exchange their sensitive data like location with money. The main goal will be implementing Zero-Knowledge Proof roll-ups to as not to reveal their identity while operating. From the protocol point of view, Ethereum is the second largest blockchain and the first one that can actually be programmable, however, it has lots of drawbacks such as low throughput, high latency and finality, and too much electricity consumption because of PoW. There

are a couple of alternatives to solving these problems for example Avalanche, Solana, Algorand, and Polkadot. All these Layer-1 blockchains are using PoS and thanks to their eco-friendly implementation gaining popularity nowadays. But still, Ethereum is the most popular protocol among users and developers. That is why this project is implemented in Solidity and Ethereum protocol but for further implementations, any one of the previously highlighted alternative Layer-1's can be used. They are all EVM compatible meaning they have a bridge and all Ethereum standards are applicable and integrateable easily. For the sake of our project, if the Ethereum 2.0 release will be postponed again, we are going to look for alternatives at that moment.

REFERENCES

30 Million Dollars Locked Up (2022). 30 million dollars locked up to nft contract. https://www.theblockcrypto.com/linked/143198/smart-contract-mistake-locks-up-34-million-of-eth-for-nft-project. [Online; accessed 28-April-2022].

300 Million Dollars Lost (2022). 300 million dollars. https://www.theguardian.com/technology/2017/nov/08/cryptocurrency-300m-dollars-stolen-bug-ether. [Online; accessed 28-April-2022].

Brownie (2022). Brownie. https://eth-brownie.readthedocs.io/en/stable/. [Online; accessed 28-April-2022].

Chainlink (2022). Oracle. https://chain.link/education/blockchain-oracles/. [Online; accessed 28-April-2022].

ChainLink Aggregator (2022). Chainlink aggregator. https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol. [Online; accessed 14-May-2022].

Das, P., L. Eckey, T. Frassetto, D. Gens, K. Hostáková, P. Jauernig, S. Faust, and A.-R. Sadeghi (2019). Fastkitten: Practical smart contracts on bitcoin. In *IACR Cryptol. ePrint Arch.*

Ethereum Community (2022). Ethereum rational document. https://github.com/ethereum/ethereum-org-website. [Online; accessed 8-June-2022].

Ethereum TPS (2022). Ethereum tps. https://ethtps.info/. [Online; accessed 23-July-2022].

Etherscan - Blocktime (2022). Etherscan - blocktime. https://etherscan.io/chart/blocktime. [Online; accessed 23-July-2022].

Hegedűs, P. (2019). Towards analyzing the complexity landscape of solidity based ethereum smart contracts. *Technologies 7*(1).

Infura (2022). Oracle. https://infura.io/. [Online; accessed 28-April-2022].

Metamask (2022). Cryptocurrency wallet. https://metamask.io/. [Online; accessed 28-April-2022].

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *https://bitcoin.org/en/*.

Nick Szabo. Smart contracts. https://www.semanticscholar.org/paper/Smart-Contracts-%3A-Building-Blocks-for-Digital-Szabo/9b6cd3fe0bf5455dd44ea31422d015b003b5568f. [Online; accessed 28-April-2022].

OpenZeppelin (2022). https://docs.openzeppelin.com/contracts/4.x/. [Online; accessed 28-April-2022].

OpenZeppelin ERC-20 (2022). Openzeppelin erc-20. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol. [Online; accessed 7-May-2022].

OpenZeppelin Ownable (2022). Openzeppelin ownable. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol. [Online; accessed 7-May-2022].

Remix IDE (2022). Remix ide. https://remix.ethereum.org/#optimize=false&runs=200&evmVersion=null. [Online;

accessed 7-May-2022].

Sizon, S. and J. Nayak (2018). Decentralized proof of attendance protocol.

Solidity Community (2022). Solidity. https://docs.soliditylang.org/en/v0.8.13/. [Online; accessed 28-April-2022].

Testnets (2022a). Chainlink faucets. https://faucets.chain.link/. [Online; accessed 28-April-2022].

Testnets (2022b). Ethereum test networks. https://ethereum.org/en/developers/docs/networks/. [Online; accessed 28-April-2022].

Wang, S., Y. Yuan, X. Wang, J. Li, R. Qin, and F.-Y. Wang (2018). An overview of smart contract: Architecture, applications, and future trends. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 108–113.

Weyl, E. G., P. Ohlhaver, and V. Buterin (2022). Decentralized society: Finding web3's soul. [Online; accessed 28-Mayıs-2022].

Wikipedia contributors (2022). Cryptocurrency wallet — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Cryptocurrency_wallet&oldid=1083764632. [Online; accessed 28-April-2022].

WOOD, D. G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *https://ethereum.github.io/yellowpaper/paper.pdf*.