

**Abdullah Bashir Yasmin**

Student ID: 2305871

**Project Title: Decentralized marketplace**

Course Name: Msc Software Engineering and Applications

MSc Project Final Report

School of Computer Science and Technology

Supervisor: Mohammad Ahmad

Academic Year: 2023/2024

# Abstract

This project creates a decentralised marketplace where users can buy and sell personal items (excluding NFT art) using *ETHEREUM*. This project is built with frontend technologies such as React.js. It does not have a backend because blockchain provides one. The *dApp* uses smart contracts that are written in Solidity and are deployed on *ETHEREUM* network to manage item listings, purchasing, and paying. An *escrow* smart contract system that holds the seller’s funds until the buyer confirms the receipt of the item. Thus, guaranteeing the trust and confidence between both parties.

The marketplace uses blockchain technology to provide a collaborative commerce platform that offers greater security and transparency, getting rid of the need for intermediaries. The Blockchain technology, using blocks, controls item listings (creation, deletion, and edition), as well as all the transactions related to item purchases. This project gives new enhancement to online shopping by combining traditional marketplaces with decentralised technology, that results in a secure and efficient solution for online transactions.

# Keywords

*dApp*, blockchain, ethereum, smart contract, cryptocurrency, marketplace

# Acknowledgements

I would like to thank my professor and supervisor, Mohammad Ahmad, for his valuable input and ongoing support throughout this project. His knowledge of blockchain technology, combined with the ability to review my logbooks and provide insightful feedback, was critical to the successful completion of this project. His encouragement and advice were invaluable in dealing with the difficulties of this project, and I am very thankful for his help.

# Dedication

I dedicate this project to my friend Eugeni. He was the first to bring up this idea with me, and without his inspiration, this project would not have happened.

Contenido

[Table of Figures 5](#_Toc176398549)

[CHAPTER 1: Introduction 6](#_Toc176398550)

[Aims 6](#_Toc176398551)

[Objectives 7](#_Toc176398552)

[Artefact Description 8](#_Toc176398553)

[Thesis structure 8](#_Toc176398554)

[CHAPTER 2: Technology, Application and Literature Overview 10](#_Toc176398555)

[CHAPTER 3: Research methodology 12](#_Toc176398556)

[CHAPTER 4: Development 13](#_Toc176398557)

[FRONT-END AND SMART CONTRACTS INTERACTION 13](#_Toc176398558)

[FRONTEND 13](#_Toc176398559)

[MOCK UPS 13](#_Toc176398560)

[DESIGNING 15](#_Toc176398561)

[SMART CONTRACTS 16](#_Toc176398562)

[SOLIDITY 16](#_Toc176398563)

[FOLDER STRUCTURE FOR SMART CONTRACT DEVELOPMENT 19](#_Toc176398564)

[DEVELOPMENT OF THE CONTRACT(S) 20](#_Toc176398565)

[METAMASK AND INTERACTING WITH SMART CONTRACT 21](#_Toc176398566)

[CHAPTER 5: Interaction between all components 23](#_Toc176398567)

[CHAPTER 5: Testing 24](#_Toc176398568)

[CHAPTER 6: Discussion and Conclusions 26](#_Toc176398569)

[References 28](#_Toc176398570)

[Appendices 29](#_Toc176398571)

[Appendix A 29](#_Toc176398572)

[Appendix B 29](#_Toc176398573)

# Table of Figures

[Fig. 1 – Overview of communication between frontend and blockchain 8](#_Toc176439230)

[Fig. 2 – Workflow of the escrow system 11](#_Toc176439231)

[Fig. 3 – Browsing items that can be purchased by the user. 13](#_Toc176439232)

[Fig. 4 – User can check their items as well as edit or delete them and listing a new one 14](#_Toc176439233)

[Fig. 5 – Form to add a new item 14](#_Toc176439234)

[Fig. 6 – User can see the purchases he has made 15](#_Toc176439235)

[Fig. 7 – Ant design’s button UI component 15](#_Toc176439236)

[Fig. 8 – Creation of escrow contract flow 16](#_Toc176439237)

[Fig. 9 – Item smart contract’s function to add a new item 18](#_Toc176439238)

[Fig. 10 – Folder structure provided by hardhat 19](#_Toc176439239)

[Fig. 11 – Activation of Celo Alfajores in Infura 20](#_Toc176439240)

[Fig. 12 – Infura’s Celos Alfajores testing network’s address 20](#_Toc176439241)

[Fig. 13 – Adding Celo Alfajores network to the MetaMask wallet 21](#_Toc176439242)

[Fig. 14 – Call to smart contract’s function that returns all user's items 21](#_Toc176439243)

[Fig. 15 – The execution of the test run and successfully passed 25](#_Toc176439244)

[Fig. 16 – Graphic to determine the optimization of the code 25](#_Toc176439245)

# CHAPTER 1: Introduction

This thesis projects develops a *dApp*. A *DApp* is a decentralized application that works on a Peer-to-Peer (P2P) blockchain network. As it runs over the blockchain layer, it inherits the security properties (Do *et al.,* ). A *dApp* is developed by creating smart contracts, that are piece of code and is stored and run on the blockchain (Park, Jeong and Yeom, 2023).

Some of the biggest advantages of *dApp*s include:

* Zero downtime. If a *dApp* (contract) has been deployed over the network, it is available immediately.
* Privacy. We do not require you to provide real data to deploy a *dApp* over the network.
* No censorship. No one on the network can deny users to make interaction with our *dApp* (smart contract).
* Security. We do not need any third party to put our trust in. Blockchain itself is self-sufficient.

The implementation of a decentralized marketplace application is carried out in this project. A *dApp* that allows users to purchase and sell items and all transactions are run on *ETHEREUM* network. The frontend of the application is developed using *ReactJs*. Whereas smart contracts are written in Solidity programming language to handle the transactions and communication to the blockchain.

As mentioned before, the advantages that blockchain provides are numerous, to state some, security, transparency and decentralization. Regular marketplace applications do lack security as well as transparency. By using blockchain technology we try to remove the intermediaries and enhance security.

The new feature in this *dApp* is that unlike regular marketplaces, we use a centralized payment system. We ensure that all the transactions are secure. And we use an *escrow* contract, a contract that holds the funds and does not release them until the buyer and seller are both satisfied; thus, this builds trust and reduces the risk of fraud.

## Aims

The main aim of this project is to create a *dApp* that allows users to buy and sell items while, at the same time, providing transparency and security and using *ETHEREUM* as a medium for the payment. The aims are:

* Removing any type of intermediaries using *escrow* contract and reducing the transaction costs, providing security and providing transparency in each transaction.
* Developing smart contracts to manage secure transactions and creating trust between buyer and seller by using *escrow* contract. The aim is to reduce fraud and increase trust.
* Design a user-friendly front end that allows app’s users to interact with the blockchain more easily, list/purchase items and manage them.
* Integration of the frontend with *MetaMask* so user can connect the wallet that holds the *ETHEREUMs*.

## Objectives

The main objectives of this *dApp* are:

* User wallet integration
  + Integration between frontend and user’s wallet (*MetaMask*)
* Items management
  + Develop a user-friendly interface that allows users to create (providing name, description, photo and price), edit and delete items
  + Make item purchases
  + List items for sell
  + List user’s purchases
  + List user’s sales
* Transaction management
  + Develop Item smart contract to store all the items created by different users
  + Develop *escrow* smart contract in solidity to handle purchases
  + Implement *escrow* contract that releases the funds only when the buyer confirms the reception of the item
  + Test the smart contracts so they are secure and efficient
* User interface development
  + Develop a responsive, user-friendly, secure and fast interface to allow user to interact with the marketplace
  + Make sure user can list, edit or delete items
  + Make sure user can make a purchase
  + Integrate the *MetaMask* wallet to the frontend
* Testing
  + Perform multiple tests to ensure a smooth frontend and a secure and reliable smart contract

Diagrama, Escala de tiempo

Descripción generada automáticamente

Fig. 1 – Overview of communication between frontend and blockchain

## Artefact Description

The artifact developed in this project is a decentralized application (*dApp*) built on the *ETHEREUM* blockchain that allows users to list and sell items. It uses smart contracts to create the logic for items and purchases as well as the logic to develop the *escrow* contract. A contract that holds user’s funds until the buyer confirms the reception of the item, providing security and transparency. The application has only frontend (developed in *ReactJs* and using UI components from *AntDesign*) and no backend as the backend is provided by the blockchain. It interacts with the blockchain using a smart wallet (*MetaMask*).

## Thesis structure

Chapter 1: Introduction – This chapter introduces the project main aims and objectives. It also provides an information about decentralized applications and their features such as security and transparency.

Chapter 2: Technology Overview and Application – This chapter explains about the usage and what blockchain technology is. It also discusses the most common descentralized marketplace applications currently available in the market and discusses what’s missing from them so our application can take advantage of.

Chapter 3: Research methodology – This chapter how the research was carried out. Explaining the research methodology and the project management.

Chapter 4: Development and Implementation – This chapter provides a detailed explanation on how the project was developed. Discussing the frontend’s mockups, designing of the frontend and the development of the smart contract’s and everything related to them (deployment).

Chapter 5: Communication between all components – This chapter talks about how the communication is done between the frontend and the smart contracts.

Chapter 6: Testing – This chapter talks about the testing, how the data was acquired to perform it and how it was taken to make the application secure and “cheap”.

Chapter 7: Conclusions – This chapters discusses all the solutions and problems faced along the way of the development of this project.

# CHAPTER 2: Technology, Application and Literature Overview

To start by, we are going to explain what really a blockchain is. Blockchains operate via a peer-to-peer network. Machines (hosts) that connect to the network are known as nodes. Nodes are responsible for confirming data changes on the blockchain. These modifications to the are called transactions. Once a transaction is verified, it is converted into blocks and permanently linked to the chain's end point. On the blockchain, every transaction's complete history is kept. Since previous blocks are rarely updated, the blockchain itself serves as an immutable record of all data changed on a blockchain network (Biscontini, 2023). From this definition, we can conclude that the blockchain technology is a good replacement for storing data that is unchangeable and can be accessed and seen by anyone publicly.

Once the blockchain was revised and studied, lets get into the applications of blockchain.

There have been several usages of the blockchain in the e-commerce and supply chain industry; one example is giving solution to the communication issues between the grower, supplier, and customer. Using blockchain technology, which minimizes the communication gaps and enhances supply chain accountable and transparency. The status about plants or crops can include seed quality, growth, yield, water requirements, price, market trends, product necessity, demand, and supplier details (D. Sivakumar, S. Nagarajan and R. Maruthi, 2023).

Also, blockchain is playing a crucial role in the e-commerce such as safety, transparency and fraud. As the identity verification from different reliable sources is required for blockchain's decentralized database, it is difficult for the scammers to pretend to be real, which helps trusting the system more. Furthermore, blockchain technology is continuously enhancing in the supply chain management by using secure records (blocks) and smart contracts to improve product traceability and reduce the chance of fraud (Mohammed et al., 2021).

There is a use case of blockchain in marketplaces. The blockchain technology will allow marketplaces to operate without the need of intermediaries. Traditional platforms like Amazon or Ebay are controlled by a central authority that manages it. But using blockchain this can be cut off allowing peer-to-peer transactions and lowering the costs for users. In traditional marketplaces, users put their trust on the marketplace’s authority, but user still does not know it so it’s a blind trust, but with blockchain technology user’s still need to trust the developer who wrote the smart contract. While blockchain has the potential to decentralized marketplaces, it still faces challenges and technical limitations (Mik and Eliza, 2018).

For developing countries that tend to go more for Sustainable Development Goals (SDGs) that are set by the UN to address global challenges and create a sustainable and equitable world, blockchain technology comes very handy. As it provides the transparency, immutability and security of the information it becomes very useful in small producers as it improves their reputation by showing where the items come from and where they were made. They have proposed a blockchain based marketplace that uses smart contracts to trace the products authenticity and information about them. On top of that, they use cloud computing to accelerate the system and make it cheaper to use (M. P. Lamela *et al.,* 2022).

A decentralized marketplace where people can trade commodities for real money, having no single entity that controls the marketplace as it runs in the blockchain, there is no need of a central authority. The development done in Python and producing a lightweight, terminal-based application and make payments via Paypal (The and Reinbergen, 2015).

All these research studies about the application of blockchain technology in the e-commerce, supply chain and marketplace made me realize how successful and secure it was and I was doing my research on the right track. I acquired knowledge from these studies to determine what are the needed components, what data to store and learnt about *escrow* contract.

So, let’s start looking at some current decentralized marketplace applications. Traditional decentralized marketplace applications are:

* Origin Protocol: Origin Protocol is a decentralized marketplace that is built on the *ETHEREUM* network and links buyers and sellers. On the platform, we have the option to buy and sell goods using *their specific digital currency*, that is OGN.
* OpenBazaar: OpenBazaar is a peer-to-peer marketplace where users can buy and sell products using cryptocurrency without any central authority. Payments are secure without requiring any form of authorization. A *download* of the client (desktop version) is required to operate. This project closely resembles the one we are working on.
* Particl: Particl is a descentralized marketplace that allows users to purchase and sell personal goods and belongings. It is a secure, transparent offering *escrow* contract to secure the payment. It uses a *specific digital currency* to make the transactions.
* District0x: This platform allows users to create decentralized marketplaces called "districts." It uses a *specific digital currency* to make the transactions.
* OpenSea: This platform only provides the selling and purchasing of *NFT*. Digital art.

Our *dApp* has an advantage over all other blockchain projects mentioned. It offers the sale of physical goods, make secure payments, no need to download a desktop app and without the need to use a new cryptocurrency. As we can see in the current *dApps* mentioned above, some of them focus on selling digital assets (NFTs), and others use their own custom cryptocurrency to allow exchanges on their platforms, and some need to be installed. It also makes sure that safe transactions are made, are publicly available, do not require any sort of application to be downloaded and does not use a custom cryptocurrency.

In the traditional systems, there always trust lacking between both parties in a transaction. For example, if the buyer is paying first the money to the seller, the seller may deny the agreement and keep the money or if the seller pays first the money, the buyer may cheat on the seller and not to pay. In both cases we need third party system to solve this issue so both of the parties trust the third system, called *escrow system*. Escrow system is when a third party is present between the buyer and the seller, and its job is to hold the funds when there has been a transaction between the buyer and seller. Escrow is a smart contract and deployed on the blockchain and as per naturality of blockchain, it is secure. Funds that are being held by the escrow contract are only released when the conditions are met. In case of any dispute, it can be solved by an *arbitrator* and refund the adequate party (A. Endurthi and A. Khare, 2021). Escrow system has many uses in fields like e-commerce and supply chain.

How does our *dApp* prevent fraud? Our *dApp* prevents fraud and the need for middlemen by utilizing smart contracts and an *escrow* mechanism to ensure that the funds are only released when mutual confirmation of the transaction is occured. This way we fix many of the issues that traditional marketplaces and offers a reliable, transparent, and safe environment for the purchase and sale of goods.

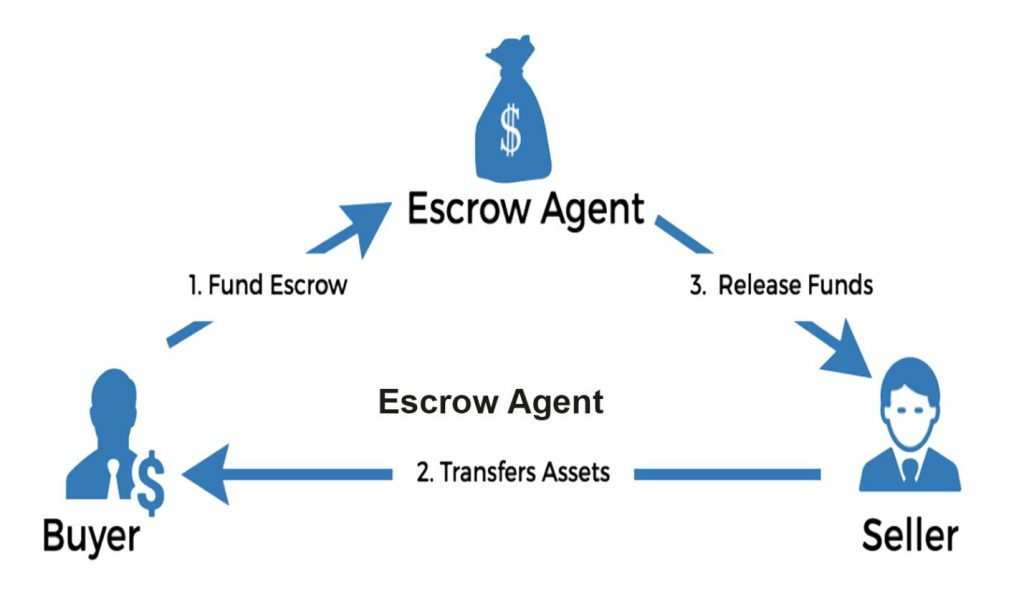


Fig. 2 – Workflow of the escrow system. Source: <https://phemex.com/academy/what-is-bitcoin-escrow>

# CHAPTER 3: Research methodology

Starting by researching some research papers, internet articles, whitepapers and official websites related to blockchain and decentralized applications to identify a gap that others *dApp*s related to marketplace selling goods did not fulfill. After that I make a comparison table stating the advantages and disadvantages that the applications current applications. This will help me identify the missing enhancements to accomplish in my project. Thus, the data is collected reading at some research papers on blockchain and *dApps* and reading on the websites of already existing applications what they offer and how they really work.

I follow an iterative development pattern. I would make a *sketch* of the applications focusing more on the essential actions such as adding, editing, deleting and purchasing item. Based on the feedback such as performance, myself being a user’s input and testing I kept on refining the application. An intensive testing is run on the smart contracts to ensure the security of the application. Especially handling the transaction thru the *escrow* contract. This step is focused on debugging and adjusting the user interface.

I have made my research in this way so firstly I can see detect something that other *dApps* don’t offer and mine can offer something unique. Then, the data collection is necessary so I can have a real-world experience, knowledge and more practical application. Next, in the iterative development I can quickly identify the issues and fix them in the next *sprint*. So, this guarantees a robust final product. This also helped me for the continuous enhancement of the project. Moreover, the testing is important as this is going to help me identify the risks and security vulnerabilities.

Regarding the project management, I have used the *Kanban* methodology. I have made a *kanban* board and create three columns (To Do, Progress and Done). This would allow me continuous monitoring of the application and make the adjustments as soon as possible and prioritizing tasks. Regarding the unplanned obstructions (such as removing the backend and creating a new smart contract) I analyzed them, then gave them the adequate priority and put them in the “To Do” column.

# CHAPTER 4: Development

## FRONT-END AND SMART CONTRACTS INTERACTION

In the same folder structure, I have installed *ReactJs* using Vite and, I have developed the smart contract. Because thru the frontend we are going to interact directly with the blockchain, so technically there is no need to have a backend or a database.

## FRONTEND

### MOCK UPS

In the very first screen user can see all items that are available to purchase, except the ones that are already sold. User can see the image of the item, name, description and price (in *ETHs*).

Diagrama

Descripción generada automáticamente

Fig. 3 – Browsing items that can be purchased by the user.

The next screen is where user can see only items listed by them. They see an image, title, description and price. Several options are provided such as editing the item or deleting it. Also, a button is available to make a new listing of the item.

Diagrama

Descripción generada automáticamente

Fig. 4 – User can check their items as well as edit or delete them and listing a new one

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Fig. 5 – Form to add a new item

The last screen is where user can see a list of items they have purchased. User can confirm the reception of the item, so funds are released and sent to the seller’s account as well as checking the status of the item.

Diagrama

Descripción generada automáticamente

Fig. 6 – User can see the purchases he has made

### DESIGNING

As far as the designing is concerned, I have used some pre-made UI components. I have used Antd (AntDesign), that is a popular UI design framework for building web applications. It provides a very extended range of components and tools that help the developers create high-quality user interfaces.

It has the basic UI components such as buttons, forms, alerts, modals… It follows a visual consistency to maintain a coherent look. The design of every component is responsive, so it fits every screen smoothly. Every UI component is customizable. And it has a very active community.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Fig. 7 – Ant design’s button UI component

## SMART CONTRACTS

Smart contracts are just a piece of code (functions) we interact with the blockchain thru. Each call to a function of smart contract requires gas fee which is small amount of *ETHEREUM*s in this case. To write and deploy smart contracts we can use Solidity or Vyper. Both are developer-friendly programming languages.

We are using Solidity to write our smart contract as it is the most used to write smart contracts for *ETHEREUM* network. We will be writing the *escrow contract* and the *item*. *Escrow* contract will hold the buyer’s payment to the seller until the buyer confirms the reception of the purchased item. In a regular application, this would be the application itself. But we do it over the blockchain for the advantages already stated. And in the item contract we will be storing all related to user’s published items and purchased ones.

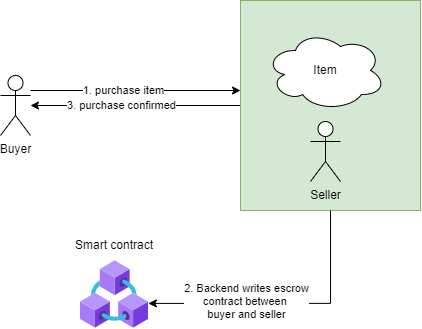


Fig. 8 – Creation of escrow contract flow

### SOLIDITY

Solidity is a high-level programming language that is designed to write smart contracts and run on the *ETHEREUM* blockchain. It is very similar to *Javascript* which makes it easy for web developers. Each function defined in the solidity can be executed by paying some *gas fee* and is run on the *ETHEREUM* Virtual Machine and can also interact with other smart contracts (as in my case Item contract interacts with Escrow contract). These contract handle transactions, tokens and store the data on the blockchain in form of block. Once the block is written, it is immutable. As the data is immutable, security becomes crucial in the development of the smart contracts, so developers must carefully handle thinks and do an intensive testing.

The *Item* smart contract smart contract holds the following fields and their corresponding data type:

|  |  |  |
| --- | --- | --- |
| Field | Data type | Use |
| itemId | Integer | To store item’s ID |
| title | String | Item’s main title |
| Description | String | Item’s extended description |
| Price | Integer | Item’s price in Integer as in the blockchain is represented as Hex |
| Image | String | Item’s image unique UUID |
| createdAt | Integer | The date item was created. It is stored in seconds. |
| Buyer | Hexadecimal | Buyer’s wallet address |
| hasBuyer | Boolean | Flag wh*ETH*er an item has a buyer |
| Escrow | Escrow | Item’s purchase |
| Seller | Hexadecimal | Seller’s wallet address |
| items | Array of items | All user items |
| Sellers | Array of hexadecimals | All seller’s wallet addresses |

In the Appendix A we can see the corresponding code and declaration for the fields and their respective data type.

The Escrow smart contract holds the following fields and their corresponding data type:

|  |  |  |
| --- | --- | --- |
| Field | Data type | Use |
| boughtAt | Integer | Item’s purchase date time |
| currState | AWAITING\_DELIVERY,SHIPPED\_OUT\_BY\_SELLER, DELIVERY\_CONFIRMED\_BY\_BUYER,DISPUTE\_OPENED, DISPUTE\_CLOSED | Current purchase status |
| Buyer | Hexadecimal | Buyer’s wallet address |
| Seller | Hexadecimal | Seller’s wallet address |
| arbitrator | Hexadecimal | Arbitrator’s wallet address |
| disputeHistory | Array of strings | Dispute and purchase history |
| sellerProof | String | Seller’s image proof |
| buyerProof | String | Buyer’s image proof |
| trackingNumber | String | Item’s tracking number |

In the Appendix B we can see the corresponding code and declaration for the fields and their respective data type.

After setting the fields and their respective data types, I have started creating the necessary functions to handle the smart contract’s logic.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Fig. 9 – Item smart contract’s function to add a new item

The reason behind creating two different smart contracts is to separate the logic between item and purchase. Item smart contract hold and manages everything related to Items (adding, editing, eliminating, purchasing) and escrow manages everything related to the purchase (shipping, cancelling, opening a dispute).

By this way, each smart contract is easy to develop, test and maintain. Furthermore, each smart contract can be audited and tested for vulnerabilities. Also, when making a new *bug fix*, there is no need to alter the other contract. New features and enhancements can be added without affecting the other making the system more adaptable to future needs.

### FOLDER STRUCTURE FOR SMART CONTRACT DEVELOPMENT

To deploy smart contracts, I have used Nodejs, but we also do need some libraries that would make the smart contract development easy. For that purpose, the following library has been used:

* Hardhat: This library provides us the basic environment to develop smart contracts. It allows us to test smart contracts before deploying them.

Hardhat provides us the following folder structure:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Fig. 10 – Folder structure provided by hardhat

In this folder structure:

* artifacts: Holds the ABI of the contract. ABI is a template that describes the m*ETH*ods and variables that a smart contract has written in JSON format.
* contracts: Holds the smart contracts written in Solidity language
* deploy: Holds the deployment we write for the contracts.
* ignition: Holds the contract’s binaries.
* test: Holds the tests to be performed
* hardhat.config.js: Hold all the configuration related to the network settings we are connecting and deploying our contract to.

### DEVELOPMENT OF THE CONTRACT(S)

Once we have written the *escrow* smart contract, we just compile it and deploy it. But to deploy the contract on the *ETHEREUM* network we need to pay. For the sake of this project, we are going to sign up to a platform that provides the access to *ETHEREUM* network without the need to run it in our own node. This platform is Infura. It allows us to deploy and interact with smart contracts *(Infura, n.d.)*.

Once we have written the smart contract, using hardhat we just compile it and deploy it. But where do we deploy it? As early mentioned, to deploy a smart contract on the *ETHEREUM* blockchain we need gas (*ETHs*). That means real money, but we are going to use an alternative. We can use some tools that are already available for us by other developers. We would be using Celo Alfajores. This is a test network which means every transaction we do is not written to *ETHEREUM* network, but to the testing network Celo provides. Celo is designed to be compatible with *ETHEREUM* blockchain, so it can interact with *ETHEREUM*’s ecosystem. Apart from that, Celo provides us a platform (Alfajores Token Faucet) that sends to our wallet some coins so we can deploy the smart contract and interact with it.

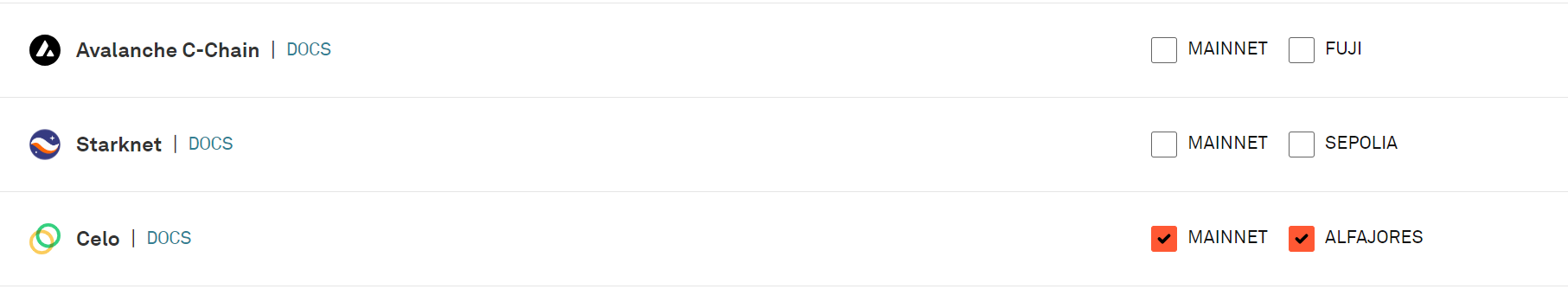


Fig. 11 – Activation of Celo Alfajores in Infura

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Fig. 12 – Infura’s Celos Alfajores testing network’s address

Once we have deployed our smart contract to Infura’s Celo network, we need to add this test network to the *MetaMask* wallet. In *MetaMask* we activate the “Testing Networks” and add the Celo Alfajores address. To simplify this process, we can also access *ChainList* and search for *Celo Alfajores* and add it in the wallet *(chainlist.org, n.d.)*.

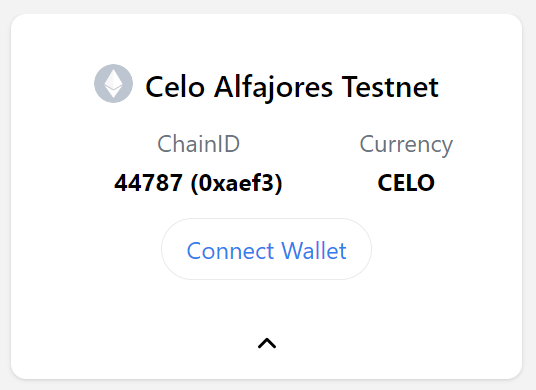


Fig. 13 – Adding Celo Alfajores network to the MetaMask wallet

### METAMASK AND INTERACTING WITH SMART CONTRACT

By interaction with smart contract, we simply mean to call functions written in the smart contracts. But, to connect with the blockchain and interact with Web3 we need to have some sort of logic that allows us to do so. For this purpose, we use *MetaMask*, a wallet that stores user’s assets (crypto currencies). For every transaction we make with the blockchain (calling a smart contract’s function) we do spend some gas fee (a very small amount of *ETHEREUM*). Gas fee is needed to execute transactions on the contracts that run over the *ETHEREUM* network.

When we define a smart contract, we need to interact with it using functions that are defined in the smart contract. For this purpose, we use ABI (Application binary interface) that allows us to interact with the *ETHEREUM* ecosystem from outside of the blockchain and between contracts. The ABI is a human-readable list of all the functions, the parameters that the function takes as well as the data type.

Following is the part of the code that reads user’s wallet and makes the call to the function of the smart contract that returns all user’s published items.

Texto

Descripción generada automáticamente

Fig. 14 – Call to smart contract’s function that returns all user's items

Regarding the libraries to interact the frontend with *MetaMask* we use *ETHers*. The *ETH*ers.js library seeks to be a comprehensive and compact tool for connecting to the *ETHEREUM* Blockchain and its surrounding ecosystem *(docs.ETHers.org, n.d.)*.

# CHAPTER 5: Interaction between all components

The interaction and workflow of the application is as following:

* Thru the frontend user asks for information (adding item, browsing items…)
* Request with the necessary parameters is sent to the smart contract thru the contract’s ABI to determine the conditions met
* Request is sent to the smart contract and is executed on the EVM
* Response is returned to the user

Once the smart contract is compiled, it generates bytecode that is almost impossible to understand by human but by the machine. Let’s us understand what ABI is. Application Binary Interface is file that holds the data according to its type and its specifications. It has defined all the functions and data types of the variables available in out smart contract. It is an interpreter that makes the communication easy with the EVM. So, ABI and bytecode work together to define the smart contract’s functionality.

# CHAPTER 5: Testing

The testing was very extensive. I have done testing only for the smart contracts. I’ve used a library called *chai*, *faker* and *Hardhat*.

Once the development of the contract was done, I conducted a series of tests in *JavaScript* and using frameworks like *hardhat* I simulated scenarios like adding, editing, deleting and purchasing item. All these tests used the *Faker* library to generate realistic data. If the contract’s testing didn’t work as expected, debugging was done to identify the cause. If the cause was identified I rapidly fixed it before proceeding with a new feature. Otherwise, I just postponed it and give it appropriate priority and continued with the work.

I repeatedly ran the tests with different sets of data. The same library *faker* every time generated a new set of data totally random. The evaluation strategy I used was logging the time. Before and after every test I timed it and made a table with the name of each function, the three tests I did and how long it did take. Also, it was focused more on the gas usage and execution time. As we have said, every call to the smart contract’s function uses a gas fee. I have also tracked by every transaction how much gas fee was being used. This is not 100% accurate as depending on the function’s logic, such as the operations it performs and the complexity of the function and the current gas fee price that is determined by the supply and demand network. So, depending on the gas fee used I have changed the variables data types and made the function less complex.

The tests were done in the local environment of *Alfajores* network. As this was the closest to the reality. So, this gave me a real-world idea of the performance and security of the smart contracts.

An example of the *Item* smart contract’s *addItem* function. The title was the field that was modified each time to see the different behavior of gas fee and execution time:

|  |  |  |
| --- | --- | --- |
| Title | Gas fee | Execution time |
| Item 1 | 277328 | 51.146ms |
| Item es simplemente el texto | 277592 | 23.837ms |
| Item es simplemente el texto de relleno de las imprentas y archivos de texto. Lorem Ipsum | 345171 | 28.567ms |

In the figure 15 we can see an example of the test run; in the Appendix C we can see the source code of testing and in the figure 16 we can see the graphic I followed to determine the gas fee used to determine the optimization of the code. So, by this, I tried using less variables and optimizing the cost to reduce the gas fee. User can tolerate the execution time as it runs on the blockchain, and transactions can be slow and most of the times user is aware of this. But the gas fee is what user could not compromise with.

Texto

Descripción generada automáticamente

Fig. 15 – The execution of the test run and successfully passed

Gráfico, Gráfico de líneas

Descripción generada automáticamente

Fig. 16 – Graphic to determine the optimization of the code

# CHAPTER 6: Discussion and Conclusions

I will start off this chapter by stating what I have achieved. User in this project is the listing, editing, deletion and purchasing an Item. In the listing user can add a title, description, amount and an image. Also, user can edit all four fields if he wants to unless there has already been a purchase. For a purchased item user has up to 15 minutes to cancel the purchase. After 15 minutes it is not possible to cancel it. When a purchase is made, the fuds are held in the *escrow* contract. The seller can deliver the item after 15 minutes and provide the tracking code. After that If buyer has received the item, buyer can confirm it and fuds are released from the *escrow* contract and given to the seller. If an issue occurs, buyer can open a dispute and both, buyer and seller provide a proof, so the final decision is upon the arbitrator. This is another wallet that based on the proofs decides who to refund (buyer or seller). In terms of achieving what I proposed initially, I have accomplished it. As a software engineer, I have developed a novelty on an already existing application. If we see applications like OpenSea we come to realize we can only upload the digital art and not physical belongings, or we need a certain coin to trade in with or we need to download a app. So, our application gives the solution to it.

Along the development of this project, I have come across many problems and new knowledge for me. I have never worked with decentralized applications before. I started understanding how blockchain does work. Before I started the project, I have known that everything in the blockchain is a block. But I didn’t know that even a modification in a “block” creates another block. So here the theory of the blocks being immutable comes true.

I will discuss the issues I had and how I solved them.

The communication and interaction to *MetaMask* and decision of what library to use to interact with *MetaMask* was hard. I was looking at many examples in the Internet and all of them used many different libraries. So at the end I made a comparison table to decide what library I would use.

I am not a designer but a developer offering solutions to existing *logical* problems. In this case, my objective is to develop the *dApp*. So, the *designing* of the interface was heavy on me. Thankfully, I could find already designed *components* to use in my React application.

The testing of the smart contracts Item and *Escrow* was a problem too. I am using the *ETHers* library to connect *MetaMask* to the blockchain. To tun the tests for the smart contract we also need *ETHers* library but on a lower version. So this made the code to run in a circular dependency and it took me a very long time to solve the issue.

At the middle of the project development, I came to realize that the PostgreSQL database it not necessary. This would just make my application bigger and hard to maintain. So, I decided to get rid of the database and make another smart contract called Item. This would store everything related to the Item. I got rid of the backend and the database.

Adding picture for the item was a trouble. First place I tried to find every solution to copy the image to my local storage and server it directly from the react application. Unfortunately, from the React there is no way to interact with the operating system and copy or store image to a local route. The solution was to develop a API in Nodejs that would server and store the image locally.

Debugging smart contract is a bit complex. As we cannot set *prints* in the code (we can but it does not show as it runs in the blockchain backend). To overcome this issue, I had to set events in the solidity code and debug it.

Another issue was that once I redeploy a smart contract as I made some changes to it, all the data (blocks) is lost. So, I would need to add data (items) again.

The price is stored as hexadecimal value. If we type in 1 it is represented as 0x1. And there is another concept called *Gwei*. *Gwei* is the smallest fraction of *ETHs*. This is 1 1,000,000,000 *Gwei* as 1 *ETH*er. The first time I was looking up on the *ETHEREUM* documentation about transferring funds and storing the price, I got confused and stored *Gwei*. So, when I wanted to store an item that costs 3 *ETH*, this was represented as 3,000,000,000. So, every transaction I made related to purchasing and refunding, there was only a small fraction moved to the wallet. In this case if user was refunded, they only received 0.0000000003 *ETHs*. But thanks to an extended reading of the documentation and looking for the solutions on the internet I came to realize that the way I was parsing the number input before sending it to the Solidity was wrong. I just fixed it and seemed to work fine.

I wanted to make this application as a final product to be sold or to be run on the internet and earn some passive income. There are few thinks I would have loved they were done during the development of this project if I had the economic and human resources as well as more time:

* Deploying the smart contract on a *ETHEREUM* network. My main goal at the end is to make this application accessible globally. But, to deploy it requires a significant amount of money.
* Do some marketing. As it is necessary to do some marketing to make people aware of this project. I would have done some social media marketing and direct marketing.
* Hire another blockchain developer. Every programming language has its own complexity and *tricks*. So, a blockchain developer would’ve really helped me make my code less complex and optimized.
* Direct communication with the delivery company and follow up on the delivery updated. Currently, seller just types in the tracking code but buyer manually needs to track it.
* Keeping history of the purchases for a Item and leaving a review. Now a item has only one purchase. If user cancels a purchase for a Item, this current *purchase* is lost and we cannot track it anymore. Keeping a log would be a good idea for further improvement. Also leaving a review for the seller by the buyer would be a good feature to determine the reliability of the seller.

# References

Do, H., Son, Nguyen, D., Hao, Thi, T., Quynh, T. and Le Quang, M. *W2E (Workout to Earn): A Low Cost DApp based on ERC-20 and ERC-721 standards*

Park, J., Jeong, S. and Yeom, K. (2023) *Smart Contract Broker: Improving Smart Contract Reusability in a Blockchain Environment* MDPI AG

Ünal, H.T., Mete, S., Vurgun, ÖU., Mendi, A.F., Özkan, Ö and Nacar, M.A. (2023) *Postgresql Database Management System: ODAK* IEEE

*Express - Node.js web application framework (no date).* [*https://expressjs.com/*](https://expressjs.com/)*.*

*Node.js — Run JavaScript everywhere (no date).* [*https://nodejs.org/en*](https://nodejs.org/en)*.*

Bhatt, C., Tiwari, D., Dua, D., Gupta, S. and Singh, T. (2024) *Elevating Online Retail: An In-Depth Look at the Implementation of React JS in Advanced E-commerce* IEEE

*chainlist.org. (n.d.). Chainlist. [online] Available at:* [*https://chainlist.org/*](https://chainlist.org/)*.*

*Infura. (n.d.). ETHEREUM API | IPFS API & Gateway | ETH Nodes as a Service. [online] Available at:* [*https://www.infura.io/*](https://www.infura.io/)*.*

*docs.soliditylang.org. (n.d.). Contract ABI Specification — Solidity 0.8.21 documentation. [online] Available at:* [*https://docs.soliditylang.org/en/latest/abi-spec.html*](https://docs.soliditylang.org/en/latest/abi-spec.html)*.*

*docs.ETHers.org. (n.d.). Documentation. [online] Available at:* [*https://docs.ETHers.org/v5/*](https://docs.ETHers.org/v5/)*.*

Biscontini, T. (2023) *Blockchain (technology)* Salem Press

D. Sivakumar, S. Nagarajan and R. Maruthi (2023) *Blockchain Technology for Agricultural Supply Chain Management to Enhance Farming.* pp. 622

*Sabah Mohammed, Jinan Fiaidhi, Carlos Ramos, Tai-Hoon Kim, Wai Chi Fang, and Tarek Abdelzaher. "Blockchain in eCommerce: A Special Issue of the ACM Transactions on Internet of Things." ACM Transactions on Internet Technology 21, no. 1 (2021): Article 4.* [*https://doi.org/10.1145/3445788*](https://doi.org/10.1145/3445788)*.*

*Amure, T.O. (2024) What is OpenSea?* [*https://www.investopedia.com/what-is-opensea-6362477*](https://www.investopedia.com/what-is-opensea-6362477)*.*

*Academy, B. (2023) 'OpenBazaar, un marketplace descentralizado ¿Qué es y cómo funciona?,' Bit2Me Academy, 10 May.* [*https://academy.bit2me.com/en/que-es-openbazaar/*](https://academy.bit2me.com/en/que-es-openbazaar/)*.*

*Amure, T.O. (2024) What is OpenSea?* [*https://www.investopedia.com/what-is-opensea-6362477*](https://www.investopedia.com/what-is-opensea-6362477)*.*

*Apps that respect your rights (no date).* [*https://particl.io/marketplace*](https://particl.io/marketplace)*.*

*district0x (no date).* [*https://district0x.io/*](https://district0x.io/)*.*

*Origin Protocol (no date).* [*https://www.originprotocol.com/*](https://www.originprotocol.com/)*.*

*A. Endurthi and A. Khare (2021) Cheat Proof Escrow System for Blockchain. pp. 294*

*M. P. Lamela, J. Rodríguez-Molina, M. Martínez-Núñez and J. Garbajosa (2022) 'A Blockchain-Based Decentralized Marketplace for Trustworthy Trade in Developing Countries', IEEE Access, 10, pp. 79100–79123 Available at: 10.1109/ACCESS.2022.3194511*

*The, M. and Reinbergen, H. (2015). Tsukiji, a decentral marketplace.*

*Mik, Eliza. "Blockchains as Decentralized Marketplaces?" Impact of Technology on International Contract Law: Smart Contracts and Blockchain Technologies. Available at SSRN:* [*https://ssrn.com/abstract=3264565*](https://ssrn.com/abstract=3264565)

# Appendices

## Appendix A

struct ItemStruct {

        uint itemId;

        string title;

        string description;

        uint price;

        string image;

        uint createdAt;

        address buyer;

        bool hasBuyer;

        Escrow escrow;

        address seller;

    }

    mapping(address => ItemStruct[]) private items;

## Appendix B

enum State {

        AWAITING\_DELIVERY,

        SHIPPED\_OUT\_BY\_SELLER,

        DELIVERY\_CONFIRMED\_BY\_BUYER,

        DISPUTE\_OPENED,

        DISPUTE\_CLOSED

    }

    uint public boughtAt;

    State public currState;

    address payable public buyer;

    address payable public seller;

    address private arbitrator;

    string[] public disputeHistory;

    bool private sellerSubmittedProof;

    string public sellerProof;

    string public buyerProof;

    bool public isDisputeOpen;

    string private trackingNumber;

## Appendix C

function printGasUsage(tx, description) {

    return tx.wait().then(receipt => {

      console.log(`${description} Gas used: ${receipt.gasUsed.toString()}`);

    });

  }

  function printExecutionTime(fn, description) {

    console.time(description);

    return fn().finally(() => console.timeEnd(description));

  }

  it("should add an item", async function () {

    await printExecutionTime(async () => {

      const Item = await ethers.getContractFactory("Item");

      itemContract = await Item.deploy();

      [owner, seller, buyer] = await ethers.getSigners();

      const addItemTx = await itemContract.connect(seller).addItem(

        "Item 1",

        "This is item 1",

        ethers.parseUnits("1", "ether"),

        "image.jpg"

      );

      await printGasUsage(addItemTx, "Add Item");

      const sellerItems = await itemContract.connect(seller).retrieveUserItems();

      expect(1).to.equal(1);

    }, "Add Item Execution Time");

  });

  it("should add an item2", async function () {

    await printExecutionTime(async () => {

      const Item = await ethers.getContractFactory("Item");

      itemContract = await Item.deploy();

      [owner, seller, buyer] = await ethers.getSigners();

      const addItemTx = await itemContract.connect(seller).addItem(

        "Item es simplemente el texto",

        "This is item 1",

        ethers.parseUnits("1", "ether"),

        "image.jpg"

      );

      await printGasUsage(addItemTx, "Add Item");

      const sellerItems = await itemContract.connect(seller).retrieveUserItems();

      expect(1).to.equal(1);

    }, "Add Item Execution Time");

  });

  it("should add an item2", async function () {

    await printExecutionTime(async () => {

      const Item = await ethers.getContractFactory("Item");

      itemContract = await Item.deploy();

      [owner, seller, buyer] = await ethers.getSigners();

      const addItemTx = await itemContract.connect(seller).addItem(

        "Item es simplemente el texto de relleno de las imprentas y archivos de texto. Lorem Ipsum",

        "This is item 1",

        ethers.parseUnits("1", "ether"),

        "image.jpg"

      );

      await printGasUsage(addItemTx, "Add Item");

      const sellerItems = await itemContract.connect(seller).retrieveUserItems();

      expect(1).to.equal(1);

    }, "Add Item Execution Time");

  });