Relatório de Projeto

Francisco Barbeitos Soares Pereira

Engenharia Informática

jul | 2023

GUARDA
POLI
TÉCNICO

POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

TECNOLOGIAS BLOCKCHAIN

PROJETO EM CONTEXTO DE ESTÁGIO
PARA OBTENÇÃO DO GRAU DE LICENCIADO EM ENGENHARIA INFORMÁTICA

Francisco Barbeitos Soares Pereira
Julho / 2023

Escola Superior de Tecnologia e Gestão

TECNOLOGIAS BLOCKCHAIN

PROJETO EM CONTEXTO DE ESTÁGIO
PARA OBTENÇÃO DO GRAU DE LICENCIADO(A) EM ENGENHARIA INFORMÁTICA

Professor(a) Orientador(a): Paulo Alexandre de Andrade Vieira

Professor(a) Coorientador(a): Eduardo dos Santos Fonseca

Francisco Barbeitos Soares Pereira

Julho / 2023

Agradecimentos

Em primeiro lugar gostaria de agradecer ao Instituto Politécnico da Guarda pela oportunidade da realização do estágio, pois tinha uma grande curiosidade em trabalhar e perceber melhor o que é realmente esta nova tecnologia blockchain. Assim como agradeço todas as oportunidades que me foram dadas pelo instituto ao longo do meu percurso académico.

Agradeço também a todos os professores que eu conheci desde que entrei no meu percurso académico, desde os mais atenciosos aos mais rigorosos.

Agradeço aos familiares e amigos que me apoiam, e que sempre me tentaram ajudar em tudo, que estiveram sempre do meu lado, apoiando as minhas decisões.

Quero agradecer também ao meu orientador, professor Paulo Vieira, pois foi com ele que falei quando quis estagiar nesta área, e ele para além de ter feito a ponte entre mim e o IPG para que este estágio se realizasse, sempre se disponibilizou para ajudar.

Gostaria também de agradecer ao meu supervisor, professor Carlos Fonseca, que mesmo não estando tão presente quando o professor orientador, sempre tentou ajudar e discutir ideias para se chegar à melhor solução.

Ficha de Identificação

Aluno

Nome: Francisco Barbeitos Soares Pereira

Número: 1704082

Licenciatura: Engenharia Informática

Estabelecimento de Ensino

Instituto Politécnico da Guarda (IPG)

Escola Superior de Tecnologia e Gestão (ESTG)

Entidade Acolhedora do Estágio

Nome: Instituto Politécnico da Guarda (IPG)

Morada: Av. Dr. Francisco Sá Carneiro 50, 6300-559 Guarda

Contacto Telefónico: 271 220 100

Duração do Estágio: 240 Horas

Supervisor de Estágio

Nome: Carlos Eduardo dos Santos Fonseca

Função: Especialista de Informática

Docente Orientador de Estágio

Nome: Paulo Alexandre de Andrade Vieira

Grau Académico: Doutor em Informática e Automação, 2015, Universidade de Salamanca, Espanha

Resumo

Ao contrário do que muitas pessoas associam, Blockchain não são criptomoedas. As Blockchain são um conjunto de tecnologias e metodologias que permitem acrescer à segurança, transparência, e autenticidade da informação, é mesmo por isso que as criptomoedas usam a tecnologia blockchain. A sua origem e aparecimento surge com a resolução do problema de um conjunto de problemas associados à digitalização que vão desde como realizar assinaturas digitais e terminam na descoberta da resolução do problema do double spending (duplicação da mesma informação). As Blockchains são redes peer-to-peer em que os vértices da rede, os nodes, contêm dois tipos de dados os on-Chain e os dados off-Chain. Os dados on-Chain de uma Blockchain (BC) devem ser comuns em todos os nodes, os dados off-Chain podem divergir entre os diferentes nodes e servem para apoiar o seu funcionamento. Uma Blockchain pode ter vários tipos de nodes, dependo da sua funcionalidade. Por exemplo na Ethereum há os: full-nodes, light-nodes, mobile-nodes.

O uso da tecnologia blockchain está inevitavelmente e cada vez mais presente nas nossas vidas. O uso desta tecnologia traz grandes benefícios em segurança e transparência, mas tem custos elevados em consumo de energia elétrica devido a um elevado processamento e grande necessidades de espaço para armazenamento.

A interação com a blockchain é feita por meio de wallets (carteiras) e de DApps (aplicações descentralizadas). As DApps são Apps que consistem em interfaces de Smart Contracts, o seu backend é um node móbile de um Blockchain.

Neste trabalho foram programados e criados Smart Contracts usando o Remix IDE.com os Smart contracts criados, foi feita a também sincronizados tecnologias tais como: o editor de código remix com o simulador de Blockchains, o Ganache; e a uma carteira de contas de Blockchain o MetaMask. No trabalho foi criada e testada uma rede peer to peer e a sua conexão à Ethereum. Foi ainda criada uma DApp e foi desenhado um curso em Blockchain com 5 módulos.

Abstract

Contrary to what many people associate, Blockchain is not cryptocurrencies. Blockchain is a set of technologies and methodologies that add to the security, transparency, and authenticity of information, which is why cryptocurrencies use blockchain technology. Its origin and appearance arises from solving the problem of a set of problems associated with digitisation ranging from how to perform digital signatures and ending in the discovery of how to solve the problem of double spending (duplication of the same information).. Blockchains are peer-to-peer networks in which the vertices of the network, the nodes, contain two types of data on-Chain and off-Chain. The on-Chain data of a Blockchain (BC) must be common in all the nodes, the off-Chain data can differ between the different nodes and serve to support its operation. A Blockchain can have several types of nodes, depending on its functionality. For example in Ethereum there are: full-nodes, light-nodes, mobile-nodes.

The use of blockchain technology is inevitable and increasingly present in our lives. The use of this technology brings great benefits in terms of security and transparency, but it has high costs in terms of electrical energy consumption due to high processing and large storage space needs.

Interaction with the blockchain is done through wallets (wallets) and DApps (decentralized applications). DApps are Apps that consist of Smart Contracts interfaces, their backend is a mobile node of a Blockchain.

In this work, Smart Contracts were programmed and created using the Remix IDE.com the Smart contracts created, technologies were also synchronized such as: the remix code editor with the Blockchains simulator, Ganache; and a wallet of Blockchain or MetaMask accounts. At work, a peer to peer network and its connection to Ethereum were created and tested. A DApp was also created and a Blockchain course with 5 modules was designed.

Palavras Chaves

Smart Contact, solidity, peer2peer, blockchain, Ethereum, DApp, nodes, localhost, rede de teste.

Índice

Índice de figuras

Índice de tabelas

Lista de Siglas e Acrónimos

Acrónimos:

Android Studio Ambiente de desenvolvimento integrado (IDE) para o desenvolvimento de aplicações Android.

Blockchain Tecnologia de registo distribuído que visa a descentralização, segurança e transparência.

Ethereum Plataforma de computação distribuída baseada em blockchain e criptomoeda.

Flutter Framework de desenvolvimento de aplicações móveis multiplataforma, desenvolvida pelo Google.

front-end A camada de interface do utilizador de uma aplicação ou website.

Ganache Uma cadeia de blocos pessoal para o desenvolvimento de aplicações Ethereum.

Javascript Linguagem de programação popular para desenvolvimento web e interações dinâmicas em páginas da WEB.

Kotlin Linguagem de programação moderna, concisa e segura para desenvolvimento de aplicações Android.

localhost Termo utilizado para se referir ao endereço IP local (127.0.0.1) de um dispositivo, representando o próprio dispositivo.

mainNet A principal rede blockchain da Ethereum em operação.

MetaMask Extensão de navegador que permite a execução de aplicações descentralizadas baseadas em blockchain na web.

Python Linguagem de programação de alto nível e fácil leitura.

React Biblioteca JavaScript de código aberto para construção de interfaces de utilizador.

React Native Framework de desenvolvimento de aplicações móveis que permite criar aplicações nativas usando JavaScript e React.

RemixIDE Ambiente de desenvolvimento integrado para Smart Contracts na plataforma Ethereum.

Solidity Linguagem de programação de Smart Contracts usada para desenvolver aplicações na plataforma Ethereum.

Vue.js Framework JavaScript progressivo para a construção de interfaces de utilizador.

Node.js Ambiente de tempo de execução JavaScript que permite executar JavaScript no lado do servidor.

Siglas:

API	Interface de Programação de Aplicações (Application Programming Interface).
DApps	Aplicações Descentralizadas (Decentralized Applications).
ESTG	Escola Superior de Tecnologia e Gestão.
IDE	Ambiente Integrado de Desenvolvimento, uma ferramenta que oferece recursos para desenvolver e testar programas.
IPG	Instituto Politécnico da Guarda.
JSON	Notação de Objetos JavaScript (JavaScript Object Notation).
Npm	Node Package Manager.

1. Introdução

O presente relatório descreve de forma detalhada um projeto desenvolvido em contexto de estágio realizado no IPG (Instituto Politécnico da Guarda). Estágio realizado pelo aluno Francisco Barbeitos Soares Pereira no contexto da disciplina de projeto de informática, pertencente ao 3º ano do curso de licenciatura em Engenharia Informática da ESTG do IPG. Neste relatório, serão apresentados de forma sistemática e rigorosa os elementos essenciais do projeto, incluindo as metodologias adotadas e os resultados obtidos.

1.1. Enquadramento e Motivação

Na disciplina de Sistemas Distribuído tive a oportunidade de conhecer um pouco melhor a tecnologia blockchain, e com a procura de um estágio na área, surgiu a oportunidade de realizar um estágio com o objetivo de aprofundar o conhecimento sobre blockchain, explorando suas aplicações e a programação de Smart Contracts.

Neste estágio, o principal propósito é o aprofundamento de conhecimento na tecnologia blockchain, e assim ser possível obter o conhecimento necessário para fazer aplicações desta tecnologia num contexto prático e real.

A motivação para a realização deste estágio baseou-se na vontade de adquirir habilidades práticas e conhecimento aprofundado em blockchain. A perceção de que a tecnologia blockchain está cada vez mais presente nas nossas vidas, mesmo que de forma invisível, despertou em mim o interesse em compreender o seu funcionamento e explorar as suas possibilidades.

O estímulo para a escolha de um estágio nesta área foi impulsionada pela vontade de adquirir habilidades práticas em blockchain, compreender sua aplicação além das criptomoedas e estar preparado para enfrentar os desafios e aproveitar as oportunidades num campo tecnológico em constante evolução.

1.2. O Datacenter do Politécnico da Guarda

Durante o estágio realizado no âmbito da disciplina de Projeto de Informática, tive a oportunidade de desenvolver o projeto no Instituto Politécnico da Guarda (IPG).

O estágio foi realizado em colaboração com o datacenter da instituição(IPG) do polo ESTG, que desempenhou um papel fundamental como entidade acolhedora, e onde foram feitas reuniões e discussões sobre o estágio e progresso do mesmo.

O estágio foi feito no datacenter e foram totalizadas 420 horas de estágio, cumprindo até mais do que o tempo exigido pela disciplina de Projeto de Informática. Durante esse período, dediquei-me integralmente ao projeto, explorando os conceitos, desenvolvendo soluções, realizando testes e aprimorando habilidades na área de blockchain.

O estágio foi focado nas tecnologias blockchain, proporcionando uma imersão no campo dessa tecnologia em constante evolução. O objetivo principal do estágio foi aprofundar meu conhecimento sobre blockchain.

1.3. Objetivos do Projeto

O objetivo principal deste projeto está dividido em vários pontos, todos eles estão diretamente e indiretamente ligados à tecnologia blockchain:

- Criação de Smart Contracts.
- Fazer a ligação do remix IDE com o Ganache e o MetaMask.
- Criação de uma rede peer to peer (p2p, P2P) em python.
- Interação com a Blockchain em Javascript e no localhost.
- Interação com a Blockchain em python no localhost e entre computadores diferentes.
- Conexão da rede p2p em python à Ethereum.
- Criação de uma rede de teste na Ethereum.
- Criação de DApps com o Ganache.
- Criação de um curso sobre Blockchain.

1.4. Estrutura do Documento

Este relatório encontra-se estruturado em cinco capítulos:

No primeiro capítulo, procede-se à realização da introdução do projeto, onde é fornecido o enquadramento em que este se insere, juntamente com os objetivos e as metas a serem alcançadas.

O segundo capítulo descreve detalhadamente as plataformas que foram usadas neste projeto, aprofunda-se o tema de tecnologias blockchain, e ainda faz-se referência a algumas topologias de redes.

No terceiro capítulo, é um pouco mais extenso pois descreve-se como se implementa uma rede peer to peer, python, e se se implementa um sistema de visualização da blockchain, em javascript, ainda neste capítulo, é descrita também como um node é conectado à Ethereum e como isso permite que uma rede peer to peer se torne uma rede teste da Ethereum.

No quinto capítulo, é descrita a realização de testes aos Smart Contracts, à conexão do Remix IDE com o MetaMask e o Ganache, à interação com a blockchain e testes relativos à criação de uma rede peer to peer e da sua conexão à Ethereum.

No sexto capítulo, procede-se a realização de uma conclusão e observações sobre a tecnologia blockchain.

2. Estado da Arte

Este ponto descreve as plataformas e ferramentas utilizadas para a execução dos objetivos propostos.

2.1. Plataformas

Para a execução deste estágio foi necessário o uso de diversas plataformas, tanto de programação como de tecnologia blockchain.

2.1.1. Remix

O Remix *IDE* é um ambiente de desenvolvimento online para desenvolvimento de soluções em Solidity e outras linguagens relacionadas à Ethereum. Permite que os desenvolvedores criem Smart Contracts diretamente nos browsers (Google Chrome), sem ser necessário instalar software adicional.

O Remix permite criar, importar, escrever, testar, implantar Smart Contracts numa rede Ethereum, e interagir com eles usando uma interface simples. Como IDE também possui ferramentas integradas de análise de código, depuração e simulação.

O Remix é uma das ferramentas mais populares para desenvolvimento de Smart Contracts em Solidity, e é frequentemente usado tanto por desenvolvedores iniciantes como experientes (Figura 1) [1].

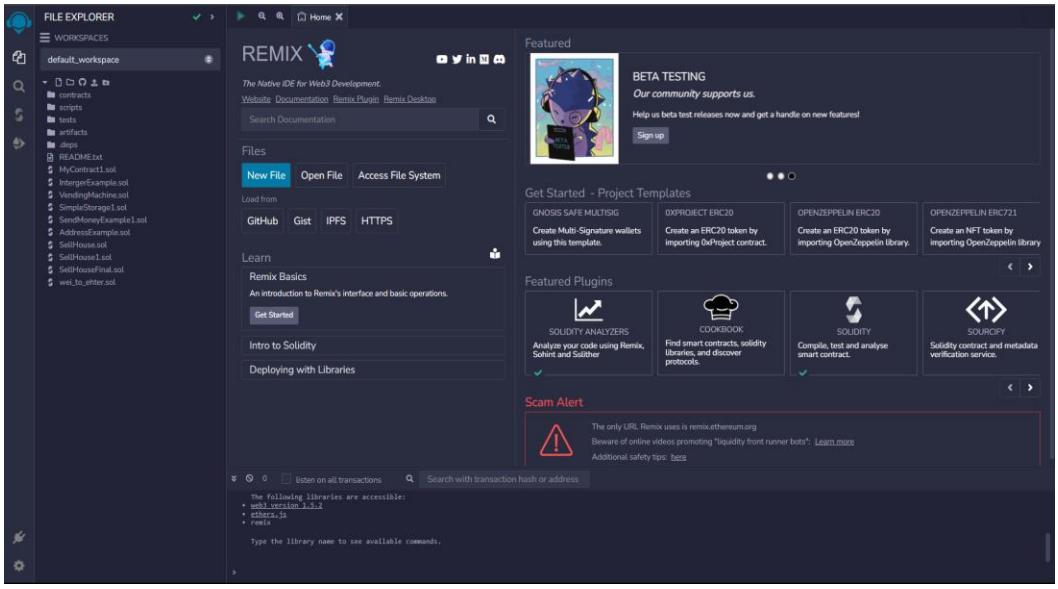


Figura 1 - Interface Remix IDE

Source: https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.19+commit.7dd6d404.js

2.1.2. Ganache

O Ganache é uma ferramenta de desenvolvimento para blockchain usada principalmente para testar Smart Contracts. Fornece um ambiente de simulação no qual é possível executar testes e depurar Smart Contracts sem precisar de haver conexão a uma blockchain real (Figura 2).

O Ganache é executado na sua própria rede blockchain privada localmente, e pode ser integrado a outras ferramentas de desenvolvimento, como o RemixIDE, também permite criar carteiras virtuais com fundos fictícios para simular transações reais (Figura 3).

É uma ferramenta muito útil para desenvolvedores de blockchain que desejam criar, testar e depurar Smart Contracts de forma eficiente [2].

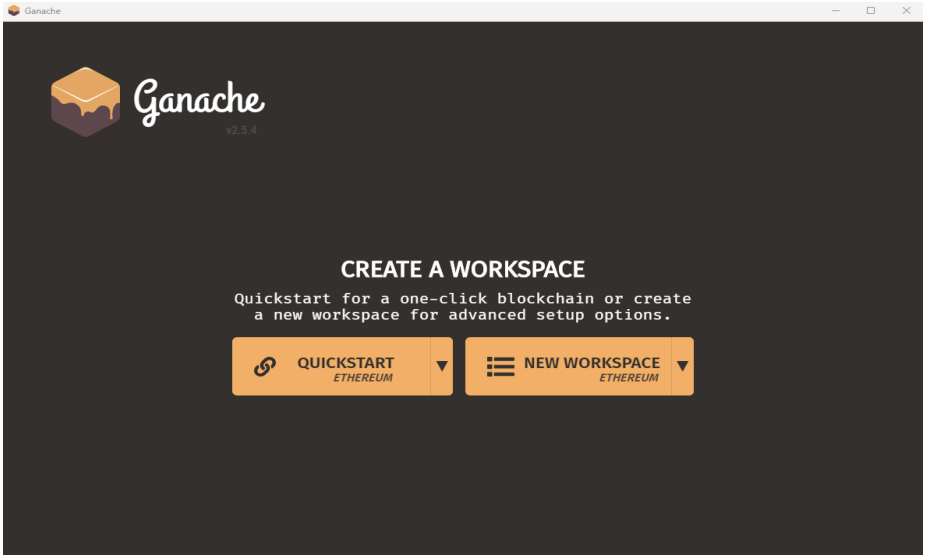


Figura 2 - Interface Inicial do Ganache

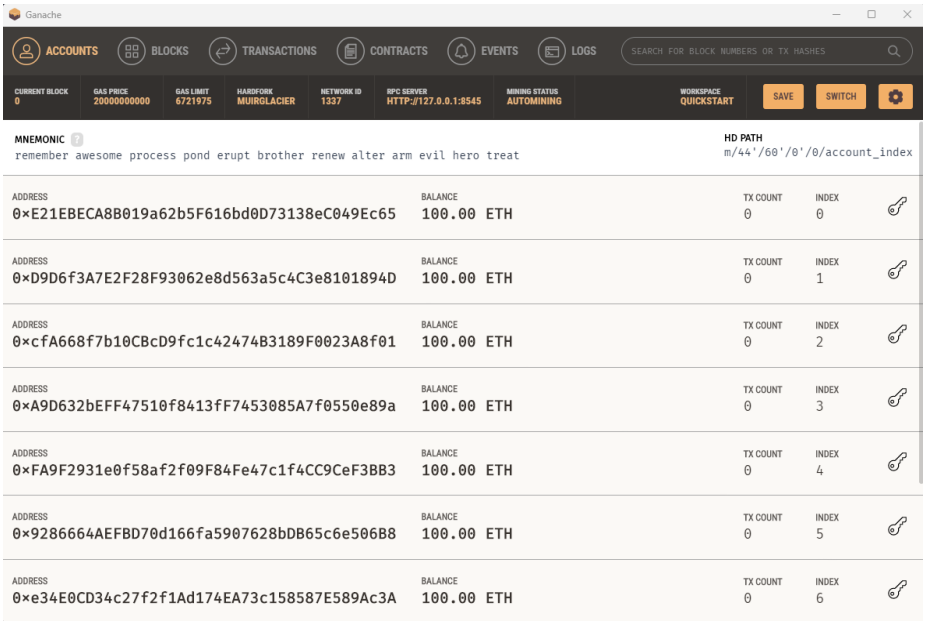


Figura 3 - Carteiras Virtuais Fictícias

Source:https://trufflesuite.com/ganache/

2.1.3. MetaMask

MetaMask é uma carteira digital que permite aos utilizadores gerir as contas, endereços de uma blockchain, nomeadamente da Ethereum. Além disso, o MetaMask funciona como uma extensão do browsers como o Google Chrome, Firefox, Brave e outros, permitindo, assim, que os utilizadores interajam com as DApps baseados na blockchain Ethereum ou outras diretamente dos browsers.

Com o MetaMask, os utilizadores podem facilmente criar e gerir várias contas Ethereum, bem como assinar transações digitalmente, sem precisar de sair do seu browser. A carteira também inclui recursos de segurança, como backup de carteira com frase de recuperação, bloqueio de senha e suporte para hardware wallets.

Além disso, o MetaMask ainda permite que os utilizadores interajam com a web3, a nova versão da internet baseada em blockchain, que é descentralizada e sem intermediários. Isso significa que os utilizadores podem ter acesso às DApp's, jogos, mercados e outras plataformas baseadas em blockchain diretamente nos seus browsers, sem precisar confiar em intermediários centralizados (Figura 4) [3].

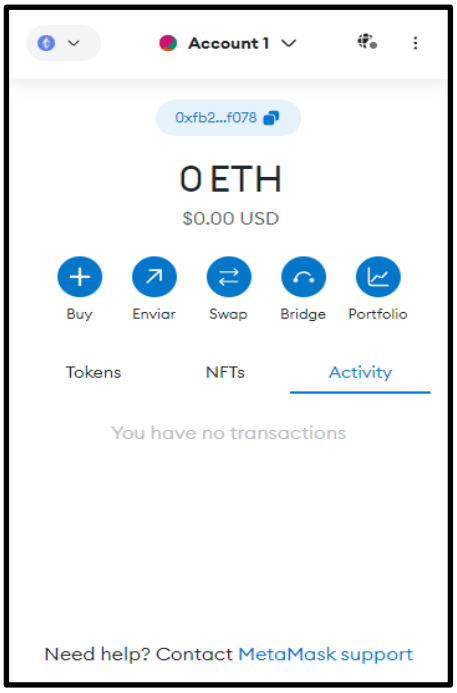


Figura 4 - Interface da extensão MetaMask

Source:https://metamask.io/

2.1.4. Visual Studio Code

Visual Studio Code é um editor de código-fonte gratuito e de código aberto, desenvolvido pela Microsoft, que suporta diversas linguagens de programação e é compatível com vários sistemas operacionais. É uma ferramenta muito utilizada por desenvolvedores para escrever, depurar e testar código de forma eficiente e produtiva (Figura 5).

O Visual Studio Code é altamente personalizável, com suporte a extensões que adicionam funcionalidades e recursos adicionais, como depuração remota, controle de versão integrado, gerenciamento de pacotes e muito mais. Também inclui recursos como realce de sintaxe, sugestões de código, formatação automática, linting e refatoração, para ajudar a melhorar a qualidade do código e a produtividade do desenvolvedor.

O Visual Studio Code também é uma excelente ferramenta para o desenvolvimento de aplicativos Web, com suporte a frameworks populares, como React, Angular e Vue.js, e integração com ferramentas de desenvolvimento de front-end, como o Node.js e o npm. É também uma boa opção para o desenvolvimento de aplicações móveis, com suporte a ferramentas como o React Native e o Flutter [4].

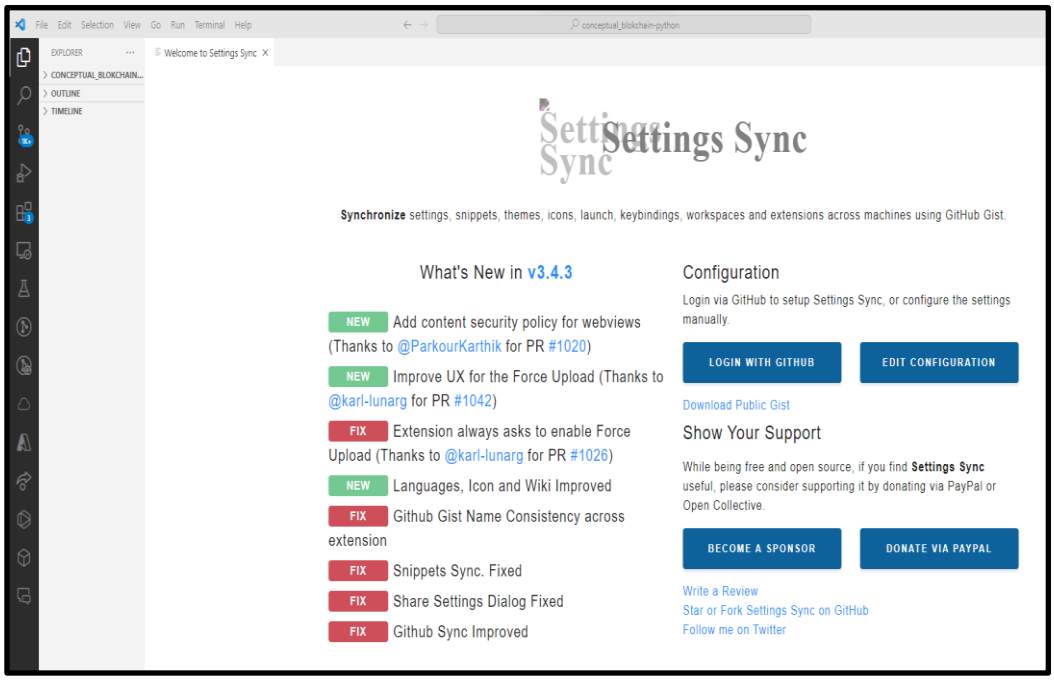


Figura 5 - Interface do VSCode

Source:https://code.visualstudio.com/

2.1.5. Android Studio

O Android Studio é uma ferramenta essencial para desenvolvedores que desejam criar aplicações móveis (Figura 6).

Uma das principais vantagens do Android Studio, é que oferece um ambiente de desenvolvimento completo, com todas as ferramentas necessárias para criar, depurar e otimizar uma aplicação Android.

Outro recurso importante do Android Studio é o emulador de equipamentos Android integrados, que permite testar e depurar as aplicações em diferentes equipamentos virtuais com diferentes versões do Android. Isso é extremamente útil para garantir que uma aplicação funcione corretamente numa ampla variedade de equipamentos antes de lançá-la para o público [5].

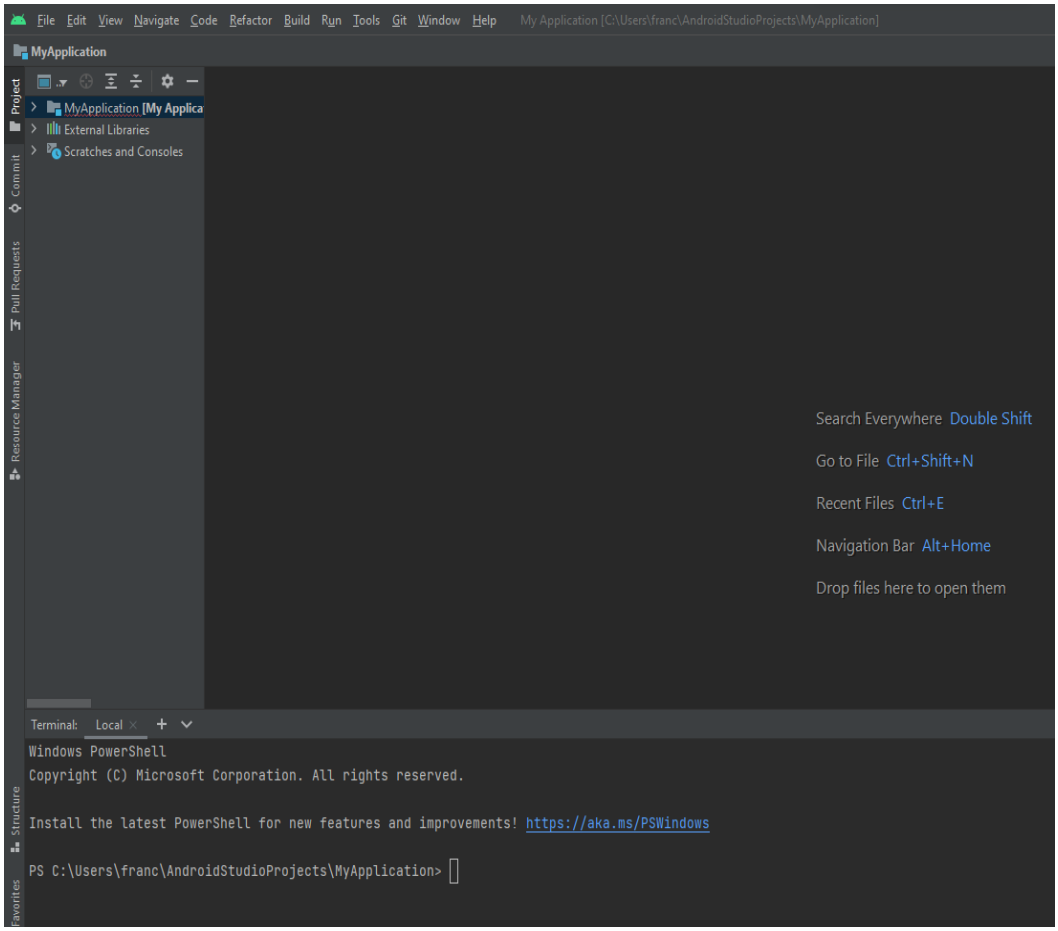


Figura 6 - Interface do Android Studio

Source:https://developer.android.com/studio

2.2.Criptografia

A criptografia é uma técnica matemática que utiliza algoritmos para codificar informações de forma que somente as partes autorizadas possam decodificá-las. No contexto da blockchain, a criptografia é empregada para garantir a integridade, autenticidade e privacidade dos dados.

Existem diferentes tipos de criptografia utilizados na blockchain, sendo os principais:

Criptografia de chave pública (assimétrica): É uma forma de criptografia que utiliza um par de chaves: uma chave pública e uma chave privada. A chave pública é partilhada livremente e usada para criptografar informações, enquanto a chave privada é mantida em segredo pelo proprietário e é usada para descriptografar essas informações. É muito utilizada para assinar transações e garantir a autenticidade de dados na blockchain.

Criptografia hash: É uma função matemática que transforma dados de tamanho variável em um valor fixo, geralmente uma sequência de números e letras chamada de "hash". É usado na blockchain para criar identificadores únicos para blocos, transações e outros dados. Isso permite a verificação rápida da integridade dos dados, pois qualquer alteração nos dados resultará em um hash completamente diferente.

Assinaturas digitais: São um componente importante da criptografia de chave pública. Permitem que um utilizador prove a autoria de uma mensagem ou transação sem revelar sua chave privada. Ao assinar digitalmente uma transação, outros participantes da rede podem verificar sua autenticidade ou usar a chave pública correspondente.

2.3.Blockchain

A blockchain é uma tecnologia que permite a criação de um registo digital compartilhado e descentralizado de transações financeiras, Smart Contracts e outras informações. O primeiro uso da tecnologia Blockchain foi o suportar moeda digital, a Bitcoin, mas tem sido cada vez mais utilizada em outras aplicações que requerem um registo seguro e transparente de informações.

A principal característica da blockchain é a sua capacidade de manter um registo imutável e distribuído de todas as transações que ocorrem na rede. Significa que sempre que

uma transação seja registada na blockchain, ela não pode ser apagada ou alterada, e todas as cópias da rede possuem uma cópia idêntica e atualizada do registo.

Esta tecnologia é baseada num modelo de consenso distribuído, onde todos os nodes da rede devem chegar a um acordo sobre a validade das transações registadas no ledger da blockchain. Isso torna a blockchain altamente resistente a ataques e fraudes, tornando-a uma tecnologia segura e confiável para o registo de informações sensíveis.

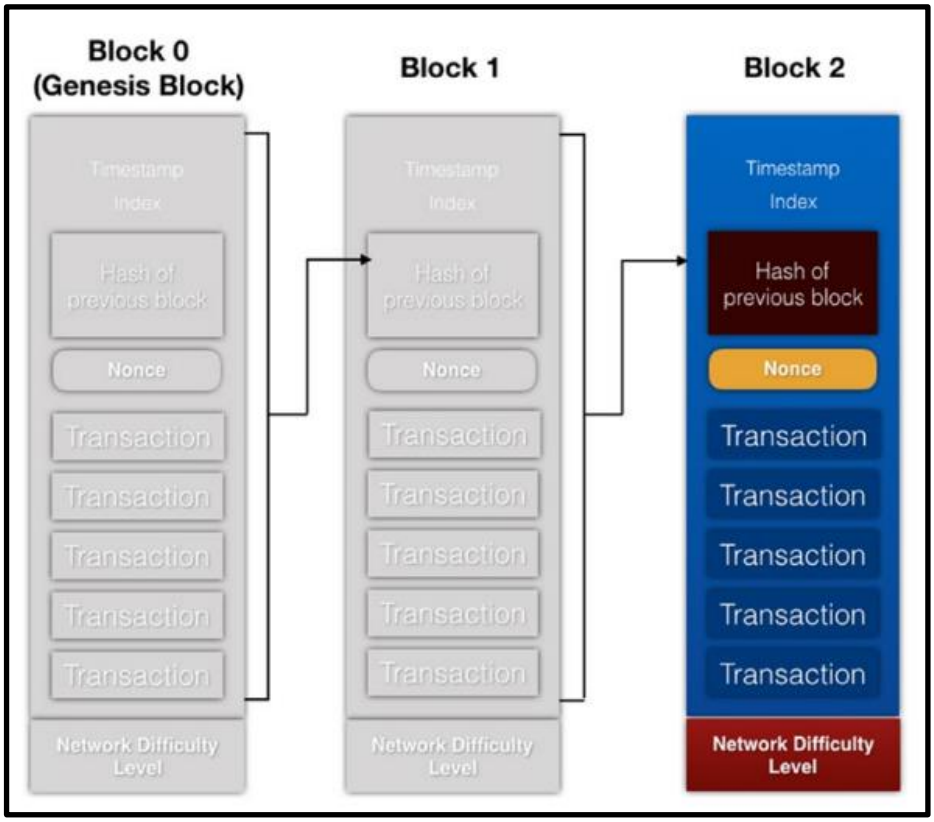


Figura 7 - criação de blocos[9]

2.3.1. Security

A segurança da blockchain é baseada em diversos mecanismos que trabalham em conjunto para garantir a integridade dos dados e a confiança entre os participantes da rede. Entre os principais mecanismos de segurança da blockchain, podemos destacar:

- **Criptografia**: é fundamental para proteger a privacidade e a segurança dos dados armazenados na blockchain.
- **Consenso distribuído**: a blockchain utiliza um sistema de consenso distribuído para validar as transações e garantir que todos os participantes da rede concordem sobre o estado atual da blockchain, este sistema de

consenso é resistente a ataques maliciosos e garante a integridade da blockchain. Proof of Work e a Proof of Stake são das mais usadas.

  - **Proof of Work**: é um mecanismo utilizado em algumas blockchains para registar um bloco e garantir a segurança da rede, exige que os participantes da rede resolvam um problema matemático complexo para registar um bloco.
  - **Proof of Stake**: é um mecanismo utilizado em algumas blockchains para registar um bloco e garantir a segurança da rede; obriga a que os participantes da rede comprovem que possuem uma certa quantidade de tokens ou criptomoedas para registar um bloco.

- **Smart Contract**: são código (de uma linguagem de programação) executável num endereço blockchain.
- **Imutabilidade**: é um dos principais pilares da blockchain, garantindo que as transações não possam ser alteradas ou apagadas após terem sido registadas e válidas num bloco e incluídas na blockchain o que garante a integridade dos dados armazenados na blockchain e evita fraudes.

Esses mecanismos de segurança tornam a blockchain uma tecnologia altamente segura e confiável, a informação é armazenada e transferida de forma descentralizada.

2.3.2. Dinheiro Digital

O dinheiro digital é implementado em Blockchain, o que permite a transferência de valor de uma entidade para outra, sem a necessidade de um intermediário, como um banco ou uma instituição financeira. É baseado em criptografia e tecnologia de blockchain, garantindo assim a segurança, transparência e resistência à falsificação. O dinheiro digital é emitido pela plataforma Blockchain e pode ser armazenado em dispositivos como smartphones ou computadores, permitindo que os utilizadores realizem transações instantâneas e globais sem as limitações das fronteiras físicas ou restrições geográficas. Além disso, o dinheiro digital pode ser usado para pagamentos online e offline, tornando-o uma opção flexível e conveniente para transações financeiras.

2.3.3. Mineração

A mineração é um processo fundamental na tecnologia blockchain. A mineração é o processo pelo qual os novos blocos de transações são adicionados à cadeia de blocos existente, o Ledger. Numa blockchain, como o Bitcoin, os mineradores competem para resolver problemas matemáticos complexos usando poder computacional.

Os mineradores usam o seu poder de processamento para realizar cálculos intensivos, tentando encontrar a solução correta para o problema matemático proposto. Esse processo é conhecido como "prova de trabalho". O primeiro minerador a encontrar a solução correta ganha o direito de adicionar um novo bloco à cadeia e é recompensado com uma quantidade específica de dinheiro digital como incentivo.

É importante ressaltar que nem todas as blockchains utilizam o processo de mineração da mesma forma. Alguns blockchains, como o Ethereum, estão migrando para um modelo de consenso chamado "PoS-Proof of Stake", em que os participantes da blockchain são selecionados para criar novos blocos, com base na quantidade de dinheiro digital que possuem, e estão dispostos a "apostar", como garantia de participação honesta no sistema.

2.3.4. Proof of Work

Proof of Work (PoW) é um algoritmo de consenso utilizado em muitas blockchains, incluindo o Bitcoin. A ideia básica por trás do PoW é que, para validar um bloco de transações e adicioná-la à blockchain, um participante da rede, um node, deve provar que realizou um trabalho computacional significativo.

Na prática, os participantes da rede, conhecidos como mineradores, competem para resolver um problema criptográfico complexo que usa um grande poder de processamento. O primeiro minerador que encontrar a solução para o problema é recompensado com novas moedas e pode adicionar um novo bloco à blockchain, contendo um conjunto de transações validadas.

O PoW é considerado um algoritmo de consenso seguro porque os mineradores gastam muita energia elétrica e recursos computacionais significativos para encontrar a solução para o problema criptográfico, tornando difícil que seja do interesse de alguém, atacante ou não, usar tantos recursos para uso de fraude sendo esta de óbvia descoberta no momento em que for tentado o seu uso.

2.3.5. Smart Contracts

Smart Contracts, código (de uma linguagem de programação) executável num endereço blockchain, quando certas condições pré-definidas são cumpridas. Eles são projetados para facilitar, verificar ou executar um contrato de forma automatizada e segura.

Os Smart Contracts são escritos em linguagens de programação específicas e são implantados numa Blockchain, o que significa que são executados numa rede distribuída de computadores. São capazes de realizar uma ampla variedade de funções, desde transações financeiras automatizadas até a implementação de soluções complexas de votação eletrônica. A execução de um Smart Contract não requer a intervenção humana, o que elimina a possibilidade de erro humano ou fraude.

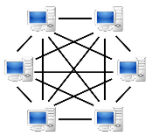
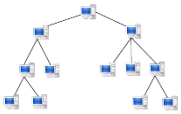
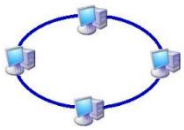
Os Smart Contracts são um dos principais recursos impulsionadores da inovação na Blockchain, permitindo a criação de DApps e sistemas autônomos que operam com transparência e segurança, sem a necessidade de intermediários ou confiança em terceiros. Também têm o potencial de tornar a execução de contratos mais eficiente e acessível, além de reduzir o tempo e o custo associados aos processos de negociação e execução de contratos tradicionais.

2.4.Topologia e Arquitetura de rede

Topologia de rede é uma estrutura física ou lógica que define como o tráfego de dados é encaminhado entre os dispositivos. Existem várias topologias de rede, cada uma com suas próprias vantagens e desvantagens em relação à segurança, escalabilidade, confiabilidade e custo. Cada topologia de rede tem suas próprias características, e a escolha da topologia adequada depende das necessidades e objetivos da rede (Tabela 1).

Tabela 1 - Topologias de rede

tipo de rede	imagem	descrição

peer to peer		A topologia peer-to-peer (P2P), é um tipo de rede, em que cada node ou dispositivo na rede funciona tanto como cliente quanto como servidor. Isto significa que cada dispositivo na rede tem a capacidade de comunicar diretamente com outros dispositivos, sem a necessidade de um servidor centralizado ou intermediário.
estrela		A topologia em estrela é um tipo de configuração de rede de computadores em que todos os dispositivos (como computadores, impressoras e servidores) são conectados a um único dispositivo central chamado de "hub" ou "switch".
árvore		A topologia em árvore é uma forma de organização de uma, em que os dispositivos são conectados num padrão hierarquizado, semelhante a uma árvore. Os dispositivos são conectados a um dispositivo central, geralmente um switch ou um router, que se conecta a outros dispositivos centrais em níveis superiores da hierarquia.
anel		A topologia em anel é um tipo de configuração de, em que cada dispositivo é conectado a outro dispositivo, formando um circuito fechado em forma de anel. Os dados são transmitidos numa única direção ao longo do anel, passando por cada dispositivo até chegar ao destino final.
malha		A topologia de malha, como mostra a Figura 5 é um tipo de rede em que cada dispositivo é conectado a todos os outros dispositivos na rede, criando uma rede completamente interconectada.

Numa plataforma Blockchain podem ser encontradas várias topologias de rede entre os diferentes nodes. A topologia peer to peer é das mais importantes, algumas Blockchains mais pequenas são mesmo a única topologia implementada.

As plataformas Blockchain são plataformas distribuídas ou descentralizadas. Há neste momento uma grande tendência para a descentralização da plataforma, tanto por uma questão de pragmatismo de implementação como por essa arquitetura ser a mais harmônica com as metodologias da Blockchain.

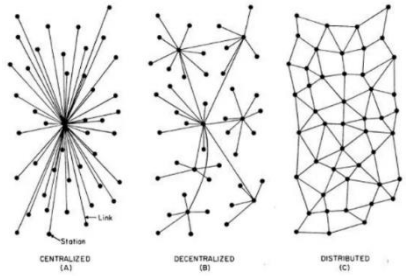


Figura 8 - Arquiteturas de redes: centralizada, descentralizada, distribuída

source:https://medium.com/delta-exchange/centralized-vs-decentralized-vs-distributed-41d92d463868 [julho 2023].

2.5.Nodes da Blockchain

Os computadores conectados à rede blockchain são conhecidos como nodes. Nos mining nodes, a principal responsabilidade é reunir transações em blocos e, em seguida, tentar adicionar o bloco ao blockchain encontrando o nonce que satisfaça a dificuldade da rede. Os mining nodes também são conhecidos como full nodes.

Atenção pois os full nodes não são necessariamente mining nodes. No entanto, os mining nodes precisam ser um full node.

O objetivo de um full node é garantir a integridade da blockchain. Por outro lado, os mining nodes de são recompensados quando adicionam um bloco à blockchain.

Cada full node possui uma cópia de toda a cadeia de blocos, também validam todos os blocos e transações apresentados a ele. Além dos full nodes, também existem os light nodes. Os light nodes ajudam a verificar transações usando o método de verificação simplificada de pagamento (SPV). O SPV permite que um node verifique se uma transação foi incluída num bloco, sem a necessidade de descarregar toda a blockchain. Com o SPV, os light nodes conectam-se aos full nodes e passam transações aos full nodes para verificações. Os light nodes precisam apenas armazenar os cabeçalhos de bloco de todos os blocos na blockchain.

Resumindo os tipos de nodes temos:

Full Node:

- Mantém uma cópia completa da blockchain
- Capaz de verificar todas as transações desde o início
- Verifica um bloco recém-criado e o adiciona à blockchain

Mining nodes (deve ser um full node):

- Trabalha num problema (encontrando o nonce)

Light node:

- Mantém os cabeçalhos da blockchain
- Usa SPV para verificar se uma transação está presente e válida num bloco

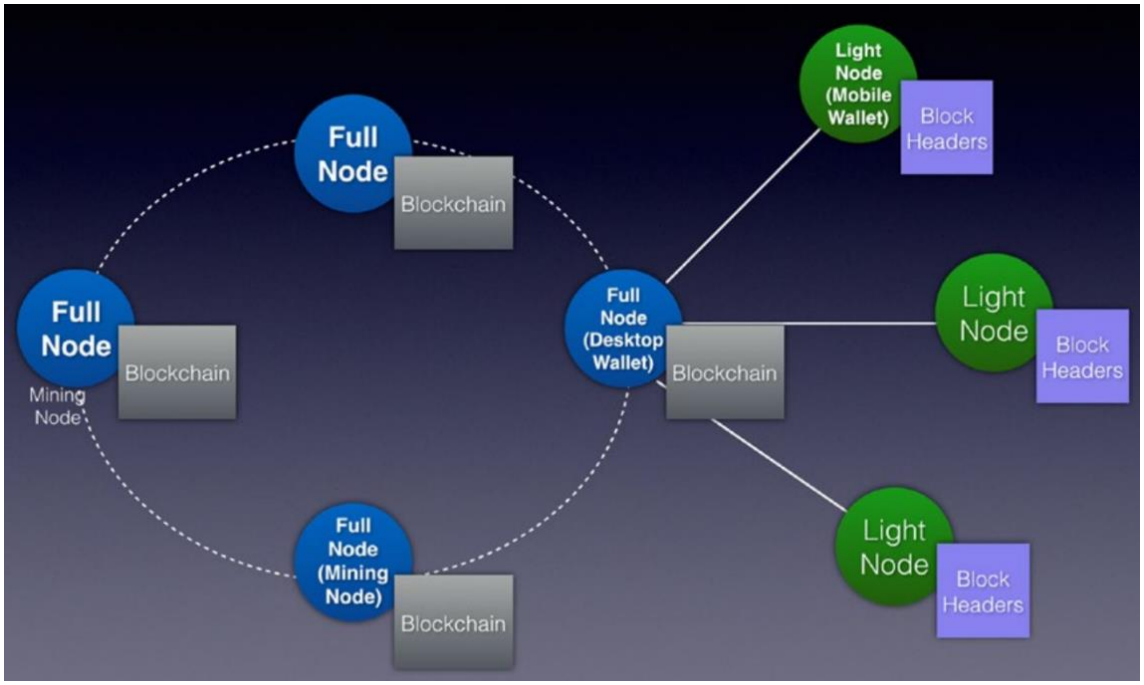


Figura 9 - Tipos de nodes

3. Descrição do Trabalho

O projeto abrange diversas facetas da tecnologia blockchain, focando-se em criar Smart Contracts e estabelecer a conexão entre o ambiente de desenvolvimento (remix IDE), ganache e MetaMask. Uma parte crucial do projeto é a implementação de uma rede peer to peer em python, permitindo a interação com a blockchain como no localhost, abrangendo a comunicação entre diferentes computadores e localmente. A integração da rede p2p em python com a plataforma Ethereum é um dos principais objetivos, bem como a criação de uma rede de teste na Ethereum. Além disso, o projeto abarca a construção de DApps com o Ganache e a concepção de um curso dedicado ao tema da Blockchain.

3.1.Criação de Smart Contracts

Neste estágio foram desenvolvidos vários Smarts Contracts com diferentes tipos de dificuldade e complexidade, para assim conseguir obter conhecimento sobre Smart Contracts e a linguagem de programação Solidity [6] [7].

3.1.1. HelloWorld

O contrato HelloWorld é um contrato simples que declara uma variável pública de string chamada myString com o valor inicial 'hello world'.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract MyContract1 {
    //Declarar uma  variavel
    string public myString = 'hello world';
}
```

3.1.2. Ver_e_definir_variavel

O contrato ver_e_definir_vareavel declara uma variável pública chamada myUint e possui uma função setMyUint que permite atualizar o valor dessa variável, dando assim para chamar a função setMyUint e modificar o valor de myUint.

```
// SPDX-License-Identifier: GPL-3.0
    pragma solidity ^0.8.1;
    contract IntegerExample {
        //declarar variavel
        uint public myUint;
        function setMyUint(uint _myUint) public {
            //introduzir valor à variavel
            myUint = _myUint;
```

```
        }
    }
```

3.1.3. Maquina_vendas

O contrato Maquina_vendas representa uma máquina de venda de cupcakes. O contrato mantém um registo da quantidade de cupcakes, permite que o dono do contrato reabasteça o stock e permite que qualquer pessoa compre cupcakes, desde que pague a quantia definida e haja cupcakes suficientes disponíveis.

```
pragma solidity 0.8.7;
contract VendingMachine {
    // Declare state variables of the contract
     address public owner;
     mapping (address => uint) public cupcakeBalances;
    // When 'VendingMachine' contract is deployed:
    // 1. set the deploying address as the owner of the
contract
    // 2. set the deployed smart contract's cupcake
balance to 100
    constructor() {
        owner = msg.sender;
        cupcakeBalances[address(this)] = 50;
    }
 // Allow the owner to increase the smart
// contract's cupcake balance
    function refill(uint amount) public {
        require(msg.sender == owner, "Only the owner can
refill.");
        cupcakeBalances[address(this)] += amount;
    }
    // Allow anyone to purchase cupcakes
    function purchase(uint amount) public payable{
        require(msg.value >= amount * 1 ether, "You must
pay at least 1 ETH per cupcake");
        require(cupcakeBalances[address(this)] >= amount,
"Not enough cupcakes in stock to complete this
purchase");
        cupcakeBalances[address(this)] -= amount;
        cupcakeBalances[msg.sender] += amount;
    }
}
```

3.1.4. Enviar_dinheiro

O contrato Enviar_dinheiro permite que qualquer pessoa envie ethers para o contrato por meio da função receiveMoney. O saldo total recebido é armazenado na variável balanceReceived. A função getBalance pode ser usada para visualizar o saldo atual do contrato.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract SendMoneyExample1 {

     //declarar variavel
      uint public balanceReceived;

      //atribuição para atribuir um valor
      function receiveMoney() public payable {

            balanceReceived += msg.value;
      }

      //soma o valor ao saldo atual
      function getBalance() public view returns(uint) {

          //visualizar o valor da transaçao
            return address(this).balance;
      }

}
```

3.1.5. Endereço

O contrato Endereço permite armazenar um endereço na variável myAddress, e visualizar o saldo em ether desse endereço, usando a função getBalanceOfAccount. A função setAddress é usada para definir o endereço desejado.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.1;
contract AddressExample {
              //declarar uma variavel como address
      address public myAddress;

      //para procurar a conta pretendida
      function setAddress(address _address) public {
            myAddress = _address;
      }
      //retorna o saldo da conta escolhida
      function getBalanceOfAccount() public view
returns(uint) {
            return myAddress.balance;
      }
}
```

3.1.6. Wie_para_Ether

O contrato Wei_para_Ether utiliza a biblioteca SafeMath para realizar operações matemáticas seguras. A função weiToEther converte uma quantidade em Wei para sua equivalente em ether, usando a divisão segura fornecida pela biblioteca SafeMath.

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/utils/math/SafeMath.sol";

contract MyContract {
    using SafeMath for uint256;

    function weiToEther(uint256 _amountInWei) public pure
returns (uint256) {

        uint256 etherAmount = _amountInWei.div(1 ether);
```

```
            return etherAmount;
        }

    }
```

3.1.7. Sell_House

O contrato Sell_house representa um contrato para compra e venda de uma casa, permite que o vendedor define e reedite informações sobre a casa, e o processo de compra e venda e o comprador efetue a compra.

Neste trecho de código, tem-se a declaração das variáveis e do struct utilizados no contrato.

```
pragma solidity ^0.8.11;
import "@openzeppelin/contracts/utils/math/SafeMath.sol";

contract PurchaseHouse {
    // Variáveis
    address payable public seller;
    uint public balanceReceived;
    using SafeMath for uint256;

    // Descrição do imóvel
    struct HouseDescription {
        string country;
        string district;
        string parish;
        string avenue_street_square;
        string building_type;
        string Description_of_building;
        string total_area_of_land;
        string year_of_enrollment_in_the_matrix;
        uint256 housevalue;
    }
    // Declaração do struct
    HouseDescription hd;
```

Neste trecho de código, temos o constructor do contrato, responsável por permitir criar um contrato. O constructor recebe os dados da descrição do imóvel como parâmetro e valida-os usando a cláusula require para garantir que todas as informações obrigatórias sejam fornecidas corretamente.

```
constructor(string memory country,string memory district,string
memory parish, string memory avenue_street_square, string memory
building_type,
    string memory Description_of_building, string memory
total_area_of_land, string memory
year_of_enrollment_in_the_matrix , uint  housevalue) payable {
        seller = payable(msg.sender);
        hd.country = country;
        require(bytes(hd.country).length > 0 &&
bytes(hd.country).length <= 50, "Country must be between 1 and
50 characters");
        hd.district = district;
        require(bytes(hd.district).length > 0, "District must be
specified");
        hd.parish = parish;
        require(bytes(hd.parish).length > 0, "Parish must be
specified");
        hd.avenue_street_square = avenue_street_square;
        require(bytes(hd.avenue_street_square).length > 0,
"Building type must be specified");
        hd.building_type = building_type;
        require(bytes(hd.building_type).length > 0, "Building
type must be specified");
        hd.Description_of_building = Description_of_building;
        require(bytes(hd.Description_of_building).length > 0,
"Building type must be specified");
        hd.total_area_of_land = total_area_of_land;
        require(bytes(hd.total_area_of_land).length > 0,
"Building type must be specified");
        hd.year_of_enrollment_in_the_matrix =
year_of_enrollment_in_the_matrix;

require(bytes(hd.year_of_enrollment_in_the_matrix).length > 0,
"Building type must be specified");
```

```
        hd.housevalue = housevalue * 1000000000000000000;
    }
```

Neste trecho de código, tem-se funções que alteram dados no contrato (calls ao contrato):

```
function editHouseDescription(string memory country,string
memory district,string memory parish, string memory
avenue_street_square, string memory building_type,
        string memory Description_of_building, string memory
total_area_of_land, string memory
year_of_enrollment_in_the_matrix , uint  housevalue) public
onlySeller inState(State.Created) {
        hd.country = country;
        hd.district = district;
        hd.parish = parish;
        hd.avenue_street_square = avenue_street_square;
        hd.building_type = building_type;
        hd.Description_of_building =  Description_of_building;
        hd.total_area_of_land = total_area_of_land;
        hd.year_of_enrollment_in_the_matrix =
year_of_enrollment_in_the_matrix;
        hd.housevalue = housevalue * 1000000000000000000;
        state = State.Edit; //estado editado 1

    }

    //Compra/venda
    function purchase() public inState(State.Created) payable{
        require(msg.value >=  hd.housevalue, "not enough money
") ;
        balanceReceived = msg.value;
        state = State.Confirmed_Purchase;
    }
    function abort() external onlySeller
        inState(State.Created){
            state = State.Inactive;
    }
```

Neste trecho de código, tem-se três funções que permitem obter informações do contrato, sem modificar o seu estado (get's ao contrato):

```
function gethd() public view returns (string memory,
string memory, string memory, string memory, string
memory, string memory, string memory, string memory, uint)
{

         return (
            hd.country,
            hd.district,
            hd.parish,
            hd.avenue_street_square,
            hd.building_type,
            hd.Description_of_building,
            hd.total_area_of_land,
            hd.year_of_enrollment_in_the_matrix,
            hd.housevalue
        );
    }

    function getBalance() public view returns(uint) {
        return hd.housevalue ;
    }

    function withdrawMoney() onlySeller public
inState(State.Confirmed_Purchase) {
        seller.transfer( balanceReceived);
        state = State.Money_Sent_To_Seller;

    }
```

Neste trecho de código, tem-se estruturas importantes para garantir que o contrato funcione corretamente e que as ações sejam realizadas somente pelos participantes autorizados em estados apropriados.

```
enum State {
        Created,
        Edit,
        Confirmed_Purchase,
        Money_Sent_To_Seller,
        Inactive
    }
    State public state;


    error InvalidState();
    error OnlySeller();

    modifier inState(State state_){
        if(state != state_ ){
            revert InvalidState();
        }
        _;
    }

    modifier onlySeller() {
        if (msg.sender != seller) {
            revert OnlySeller();
        }
        _;
    }
```

3.2. Blockchain Peer2Peer Network

Foram programada 2 redes peer to peer (seguindo os livros: The Blockchain Developer [8], Beginning Ethereum Smart Contracts Programming [9]), a primeira programada em NodeJs, e a segunda programada em Python [10] .

3.2.1. Rede P2P em NodeJs

Implementação de um sistema peer-to-peer que usa a biblioteca discovery-swarm em Node.js. Permite a criação de uma rede descentralizada, na qual os peers se podem conectar, trocar mensagens e partilhar dados entre si.

Começa a importar os módulos necessários, como crypto, Swarm, defaults e getPort. Em seguida, são definidas variáveis e objetos, incluindo um objeto para armazenar os peers conectados e um identificador único gerado aleatoriamente para o peer atual.

O código configura a rede swarm utilizando as configurações padrão e o identificador do peer atual, inicia a escuta em uma porta disponível e junta-se aos peers pré-definidos, há funções para enviar mensagens para todos os peers conectados (writeMessageToPeers) e para enviar mensagens para um peer específico (writeMessageToPeerToId). Essas funções utilizam a função sendMessage para enviar a mensagem desejada para o peer correto.

Num resumo, o código demonstra como estabelecer uma rede peer-to-peer, conectar-se a outros peers, trocar mensagens e manipular eventos de conexão [11].

O código encontrasse completo encontra-se no Github(https://github.com/franciscop101/Estagio_Blockchain_1704082/tree/bauxiliar/blockchain_platform/conceptual_blokchain-nodejs).

3.2.2. Explicação do código por partes

Nesta parte do código, estão a ser importados os módulos necessários (crypto, Swarm, defaults, getPort). Em seguida, são declaradas as variáveis peers, connSeq e channel. myPeerId é gerado usando o módulo crypto para criar um identificador único para este peer.

```
const crypto = require('crypto'),
    Swarm = require('discovery-swarm'),
    defaults = require('dat-swarm-defaults'),
    getPort = require('get-port');

  const peers = {};
  let connSeq = 0;
  let channel = 'myBlockchain';

  const myPeerId = crypto.randomBytes(32);
  console.log('myPeerId: ' +      myPeerId.toString('hex'));
```

No código abaixo, é definido o objeto config com as configurações para a rede P2P, incluindo o id que recebe o valor do myPeerId. Em seguida, é criada uma instância do Swarm com as configurações fornecidas.

```
      const config = defaults({
            id: myPeerId,
            announce: [],
      // desative o DHT para evitar problemas com o NAT
            dht: false,
       });

     const swarm = Swarm(config);
```

Este trecho envolve uma função assíncrona imediatamente invocada. Primeiro, obtém-se uma porta disponível usando getPort(). Em seguida, o servidor é iniciado com swarm.listen(), e os peers existentes são adicionados à rede usando swarm.join(). Os

endereços dos peers estão definidos em peerAddresses. É exibida uma mensagem para cada peer que se conecta à rede.

```
      (async () => {
        const port = await getPort();

        swarm.listen(() => {
            console.log(`Listening on port ${port}`);
        });

  // adicione os outros peers para o array "peers"
        const peerAddresses = [
            '/ip4/192.168.206.91/tcp/4000',
            '/ip4/192.168.18.9/tcp/4000',
            '/ip4/192.168.18.13/tcp/4000'
            ];

        peerAddresses.forEach(peer => {
            swarm.join(peer);
            console.log(`Joined peer ${peer}`);
         });
```

Neste fragmento de código, são tratados os eventos de conexão com outros peers usando swarm.on('connection'). É definida uma sequência seq para identificar a conexão e o identificador peerId para o peer conectado. Em seguida, o evento conn.on('data') é tratado, onde os dados recebidos são analisados como uma mensagem JSON e exibidos na consola.

```
swarm.on('connection', (conn, info) => {
        const seq = connSeq;
        const peerId = info.id.toString('hex');
        console.log(`Connected #${seq} to peer: ${peerId}`);
        if (info.initiator) {
            try {
                conn.setKeepAlive(true, 600);
            } catch (exception) {
                console.log('exception', exception);
            }
        }
        conn.on('data', data => {
            let message = JSON.parse(data);
            console.log('----------- Received Message start ----
---------');
            console.log(
                'from: ' + peerId.toString('hex'),
                'to: ' + peerId.toString(message.to),
                'my: ' + myPeerId.toString('hex'),
                'type: ' + JSON.stringify(message.type)
            );
            console.log('----------- Received Message end ------
-------');

         });
```

As funções, writeMessageToPeers() e writeMessageToPeerToId(), enviam mensagens para os peers. writeMessageToPeers() itera sobre todos os peers e chama a

função sendMessage() para cada um. Já writeMessageToPeerToId() verifica se o ID fornecido corresponde a um peer específico antes de enviar a mensagem.

```
writeMessageToPeers = (type, data) => {
    for (let id in peers) {
        console.log('-------- writeMessageToPeers start --------
');
        console.log('type: ' + type + ', to: ' + id);
        console.log('-------- writeMessageToPeers end ----------
- ');
        sendMessage(id, type, data);
    }
};

writeMessageToPeerToId = (toId, type, data) => {
    for (let id in peers) {
        if (id === toId) {
            console.log('-------- writeMessageToPeerToId start -
------- ');
            console.log('type: ' + type + ', to: ' + toId);
            console.log('-------- writeMessageToPeerToId end ---
-------- ');
            sendMessage(id, type, data);
        }
    }
};
```

A função, sendMessage(), envia uma mensagem para um peer específico. A mensagem é convertida para uma string JSON e escrita na conexão correspondente ao peer.

```
sendMessage = (id, type, data) => {
    peers[id].conn.write(JSON.stringify(
        {
            to: id,
            from: myPeerId,
            type: type,
            data: data
        }
    ));
};
```

Finalmente, após um atraso de 10 segundos, a função writeMessageToPeers() é chamada para enviar uma mensagem inicial "hello" para todos os peers conectados.

```
setTimeout(function(){
    writeMessageToPeers('hello', null);
}, 10000);
```

3.2.3. Rede P2P em python

O código fornecido implementa uma blockchain simples usando a linguagem de programação Python e o framework Flask. O framework Flask permite criar uma API REST. A API criada permite obter a cadeia de blocos completa, minerar novos blocos, adicionar transações e sincronizar com outros nodes da rede.

A rede funciona com a execução num servidor Flask que responde a solicitações HTTP e mantém a integridade e consistência da blockchain, garantindo que todas as transações sejam validadas e adicionadas corretamente aos blocos.

Num resumo, o código implementa uma blockchain com um servidor Flask que fornece uma interface para interagir com a blockchain. Ele segue os princípios básicos da blockchain, como validação de transações, prova de trabalho e encadeamento de blocos, e permite que os utilizadores adicionem transações, minerem novos blocos e façam a sincronização com outros nodes da rede [12].

O código encontrasse completo encontra-se no Github(https://github.com/franciscop101/Estagio_Blockchain_1704082/tree/bauxiliar/blockchain_platform/conceptual_blokchain-python).

3.2.4. Explicação do código por partes

Importar os módulos necessários, como Flask que implementa um ambiente web, requests que é um módulo que implementa uma API REST para solicitações HTTP e json para codificar e descodificar objetos JSON.

```
import sys
import hashlib
import json
from time import time
from uuid import uuid4
from flask import Flask, jsonify, request
import requests
from urllib.parse import urlparse
```

Definir a classe Blockchain e suas funções:

```
class Blockchain(object):
```

Definir a dificuldade alvo, dificuldade alvo que o hash do bloco deve atender para ser considerado válido. Aqui, o objetivo é ter o hash a começar com quatro zeros.

```
difficulty_target = "0000"
```

Inicializa uma nova instância da blockchain. Ele cria o conjunto de nodes, a lista de blocos e a lista de transações atuais . Também cria o bloco genesis com um hash específico e um nonce encontrado usando PoW.

```
def __init__(self):
        self.nodes = set()
    # stores all the blocks in the entire blockchain
        self.chain = []
    # temporarily stores the transactions for the
        # current block
        self.current_transactions = []
    # create the genesis block with a specific fixed hash
    # of previous block genesis block starts with index 0
        genesis_hash = self.hash_block("genesis_block")
self.append_block(hash_of_previous_block=genesis_hash,nonce=sel
f.proof_of_work(0, genesis_hash, []))
```

Adicionar um novo node.

```
        def add_node(self, address):
        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)
        print(parsed_url.netloc)
```

Esta função é usada para verificar se uma blockchain é válida ou não. Ela percorre todos os blocos na cadeia, verifica se o hash do bloco anterior, corresponde ao hash do bloco atual, e se o nonce do bloco atual é válido.

```
 # determine if a given blockchain is valid
def valid_chain(self, chain):
        last_block = chain[0]  # the genesis block
        current_index = 1  # starts with the second block
        while current_index < len(chain):
            block = chain[current_index]
            if block['hash_of_previous_block']
!=self.hash_block(last_block):
                return False
            # check for valid nonce
            if not
self.valid_proof(current_index,block['hash_of_previous_block'],
block['transactions'],block['nonce']):
                return False
            # move on to the next block on the chain
            last_block = block
            current_index += 1
        # the chain is valid
        return True
```

Método que atualiza a blockchain desta instância.

```
def update_blockchain(self):
        # get the nodes around us that has been registered
        neighbours = self.nodes
        new_chain = None
        # for simplicity, look for chains longer than ours
        max_length = len(self.chain)
     # grab and verify the chains from all the nodes in our
        # network
        for node in neighbours:
            # get the blockchain from the other nodes
            response                                              =
requests.get(f'http://{node}/blockchain')
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']
            # check if the length is longer and the chain
            # is valid

            if length > max_length and self.valid_chain(chain):
                max_length = length
                new_chain = chain
        # replace our chain if we discovered a new, valid
        # chain longer than ours
        if new_chain:
            self.chain = new_chain
            return True
        return False
```

Método que usa o PoW para encontrar o nonce para o bloco atual. Ele gera um nonce com valor 0 e tenta calcular o hash, da combinação, do hash do bloco anterior com o nonce atual, relacionados com todas as transações. Repete essa operação aumentando o nonce até que seja encontrado um hash com o prefixo correto para satisfazer o nível de dificuldade atual.

```
# use PoW to find the nonce for the current block
      def proof_of_work(self, index,
hash_of_previous_block,transactions):
        # try with nonce = 0
        nonce = 0
        # try hashing the nonce together with the hash of the
        # previous block until it is valid
        while self.valid_proof(index,
hash_of_previous_block,transactions, nonce) is False:
            nonce += 1
        return nonce
```

Método que verifica se o nonce atual satisfaz o nível de dificuldade.

```
def valid_proof(self, index,
hash_of_previous_block,transactions, nonce):
        # create a string containing the hash of the previous
        # block and the block content, including the nonce
        content
=f'{index}{hash_of_previous_block}{transactions}{nonce}'.encode
()
        # hash using sha256
        content_hash = hashlib.sha256(content).hexdigest()
        # check if the hash meets the difficulty target
        return       content_hash[:len(self.difficulty_target)]
==self.difficulty_target
        # creates a new block and adds it to the blockchain
```

Método que adiciona um bloco à cadeia de blocos. Ele recebe o nonce e o hash do bloco anterior e cria um novo bloco com as informações do bloco atual e o adiciona à cadeia de blocos.

```
def append_block(self, nonce, hash_of_previous_block):
        block = {
            'index': len(self.chain),
```

```
            'timestamp': time(),
            'transactions': self.current_transactions,
            'nonce': nonce,
            'hash_of_previous_block': hash_of_previous_block
        }
        # reset the current list of transactions
        self.current_transactions = []
        # add the new block to the blockchain
        self.chain.append(block)
        return block
```

Esse método adiciona uma nova transação à lista de transações pendentes da blockchain. Ele recebe três parâmetros: sender, recipient e amount.

```
         def add_transaction(self, sender, recipient,
amount):
         self.current_transactions.append({
             'amount': amount,
             'recipient': recipient,
             'sender': sender,})
              return self.last_block['index'] + 1
```

Cria uma instância da biblioteca Flask com o nome da aplicação. Gera um identificador único para o node atual, usando a biblioteca UUID. Por fim, é instanciada a classe Blockchain criada anteriormente.

```
app = Flask(__name__)
# generate a globally unique address for this node
node_identifier = str(uuid4()).replace('-', "")
# instantiate the Blockchain
blockchain = Blockchain()
```

Cria uma rota HTTP com Flask para retornar a cadeia de blocos inteira. Quando se faz uma solicitação GET para /blockchain, a função full_chain() é executada. Dentro dessa função, cria um dicionário JSON com as informações da cadeia de blocos e seu comprimento. Em seguida, esse dicionário é serializado em JSON e retornado como uma resposta.

```
    # return the entire blockchain
    @app.route('/blockchain', methods=['GET'])
    def full_chain():
        response = {
            'chain': blockchain.chain,
            'length': len(blockchain.chain),
        }
        return jsonify(response), 200
```

Este trecho de código implementa a rota '/mine' do aplicativo Flask que permite a mineração de um novo bloco.

```
@app.route('/mine', methods=['GET'])
def mine_block():

blockchain.add_transaction(sender="0",recipient=node_identifier
,amount=1,)
    # obtain the hash of last block in the blockchain
    last_block_hash
=blockchain.hash_block(blockchain.last_block)
    # using PoW, get the nonce for the new block to be added
    # to the blockchain
    index = len(blockchain.chain)
    nonce = blockchain.proof_of_work(index,
last_block_hash,blockchain.current_transactions)
    # add the new block to the blockchain using the last block
    # hash and the current nonce
```

```
    block = blockchain.append_block(nonce, last_block_hash)
    response = {
        'message': "New Block Mined",
        'index': block['index'],
        'hash_of_previous_block':
            block['hash_of_previous_block'],
        'nonce': block['nonce'],
        'transactions': block['transactions'],
    }

    return jsonify(response), 200
```

Define uma rota para criar uma nova transação na blockchain. Espera receber um objeto JSON com três campos obrigatórios: "sender“, "recipient"e "amount".

```
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    # get the value passed in from the client
    values = request.get_json()
    # check that the required fields are in the POST'ed data
    required_fields = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required_fields):
        return ('Missing fields', 400)
    # create a new transaction
    index                                                    =
blockchain.add_transaction(values['sender'],values['recipient']
,values['amount'])
    response = {'message':f'Transaction will be added to Block
{index}'}
    return (jsonify(response), 201)
```

A rota itera por cada um dos endereços e adiciona-os à lista de nodes da blockchain usando o método add_node da instância da classe Blockchain. A rota retorna uma resposta JSON com uma mensagem informando que novos nodes foram adicionados e uma lista atualizada de todos os nodes na rede blockchain.

```
@app.route('/nodes/add_nodes', methods=['POST'])
def add_nodes():
    # get the nodes passed in from the client
    values = request.get_json()
    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Missing node(s) info", 400
    for node in nodes:
        blockchain.add_node(node)
    response = {
        'message': 'New nodes added',
        'nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201
```

Define a rota /nodes/sync que sincroniza a blockchain atual com as outras nodes na rede e verifica se a blockchain está atualizada. Em seguida, ele inicia a aplicação Flask para ser executada no host 0.0.0.0 na porta especificada como argumento na linha de comando.

```
@app.route('/nodes/sync', methods=['GET'])
def sync():
    updated = blockchain.update_blockchain()
    if updated:
        response = {
            'message':'The blockchain has been updated to the
latest',
            'blockchain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our blockchain is the latest',
            'blockchain': blockchain.chain
        }
    return jsonify(response), 200
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=int(sys.argv[1]))
```

3.3. BlockChain Ethereum connection

Com a rede peer to peer a funcionar corretamente, o próximo passo é conectar a rede à rede da Ethereum. Para isso vamos precisar de instalar o client da Ethereum. Neste caso, o escolhido foi o Geth cliente.

3.3.1. Clients Ethereum

Para conectar à Ethereum blockchain,é necessário um client Ethereum, que é uma aplicação que roda como um node na Ethereum blockchain.

O client executa tarefas como:

- Minerar Ethers.
- Transferir Ethers de uma conta para outra.
- Exibir informações do bloco.
- Criar e implantar contratos inteligentes.
- Usa e interage com Smart Contracts.

Alguns Etheruem Clients:

- Eth – A C++ Ethereum client (https://github.com/ethereum/aleth).
- Geth – The official Ethereum client implemented using the Go programming language(https://geth.ethereum.org/downloads).
- Parity – An Ethereum client written using the Rust programming language (https://www.parity.io/technologies/ethereum/).

3.3.2. Downloading and Installing Geth

Para fazer a conexão utiliza-se o Client Geth, para isso houve a necessidade de fazer o download do Geth e fazer também a sua instalação [13].

3.3.3. Sync Modes

Geth suporta três modos de sincronização:

Full node mode (com sincronização rápida) -> client Geth inicia sem especificar a opção --syncmode, o modo padrão usado pelo Geth é nodes completos com sincronização rápida. O Geth executa o download da blockchain completa para o computador. Quando a sincronização atinge o último bloco da rede Ethereum, muda para o modo de sincronização completa.

Full node mode -> sincroniza um node completo começando no bloco de gênese e verificando todos os blocos e executando todas as transações. Este modo é mais lento do que o modo de sincronização rápida, mas vem com maior segurança.

Light node mode -> Faz o download apenas da cadeia de cabeçalho e solicita todo o restante sob demanda da rede. Podem verificar a validade dos dados em relação às raízes do estado nos cabeçalhos dos blocos.

Ao executar uma sincronização com o Geth, pode-se saber o estado da sincronização com a propriedade 'web3.eth.syncing'.

3.4.Ethereum Rede Teste

O Primeiro passo é criar uma pasta no computador com o nome por exemplo MyTestNet.

Dentro da pasta MyTestNet houve a necessidade da criação de mais 1 pasta com o nome data e da criação do nosso bloco genesis.

```
{
  "config": {
     "chainId": 10,
     "homesteadBlock": 0,
     "eip155Block": 0,
     "eip158Block": 0
  },
  "alloc": {},
  "coinbase": "0x0000000000000000000000000000000000000000",
  "difficulty": "0x20000",
  "extraData": "",
  "gasLimit": "0x2fefd8",
  "nonce": "0x0000000000000042",
```

```
  "mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
  "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
  "timestamp": "0x00"
}
```

Feito isto, dentro da pasta data foi criada mais 2 pastas para os nodes (node1, node2).

3.4.1. Genesis Block

O Gênesis Block é um ficheiro JSON e o início da blockchain – o primeiro bloco (bloco 0) e o único bloco que não aponta para um bloco predecessor.

O protocolo Ethereum garante que mais nenhum node concorde com a versão da blockchain, a menos que eles tenham o mesmo bloco de gênese, para assim poder criar quantas blockchains testnet privadas forem necessárias.

3.4.2. Criação de uma conta Ethereum

Neste paço houve a criação de uma conta ethereum, para fazer a criação da conta é necessário abrir o cmd e fazer a execução do comando: .\geth --datadir C:\MyTestNet\data\node1 account new, onde logo a seguir vai pedir para criar uma password para a conta e a sua confirmação, depois das passwords verificadas já temos a nossa conta criada para neste caso o node 1.

3.4.3. Criação do 1º node na pasta node1

Para criar um node na rede de teste, é necessário criá-lo usando o bloco genesis criado anteriormente. Para isso abre-se o cmd e encaminhamos para o pasta-diretório onde o Geth. Chegados ao pasta-diretório, digita-se no cmd o comando: .\geth --datadir C:\MyTestNet\data\node1 init C:\MyTestNet\genesisblock.json como mostra a Figura 10.

Resposta:

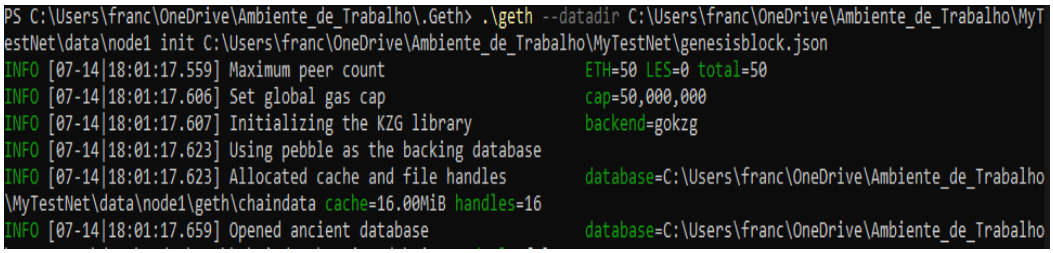


Figura 10 - Criação de um node, no node1

3.4.4. Criação do 1º node na pasta node2

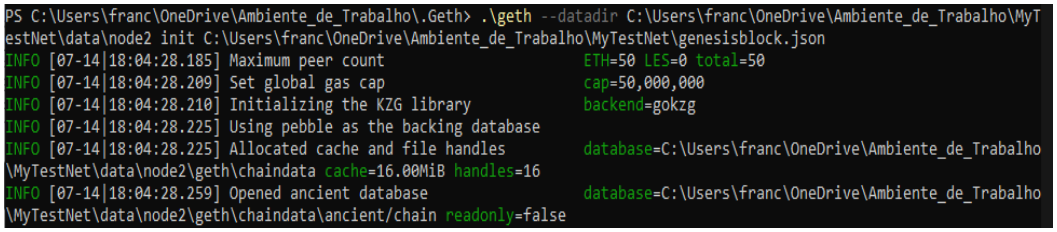


Figura 11 - Criação de um node, no node2

No node2, para criar um node ,executa-se no cmd o comando: .\geth --datadir C:\MyTestNet\data\node2 init C:\MyTestNet\genesisblock.json como mostra a Figura 11.

Em consequência dos últimos passos, 3.3.3 e 3.3.4, nas pastas node1 e node2 foram criadas automaticamente mais duas pastas e um ficheiro (geth, keystore, history respetivamente), a pasta geth contém duas pastas que armazena as blockchains – chaindata e lightchaindata, enquanto a pasta keystore contém informações de contas.

3.4.5. Inicialização node1

Para inicializar o node 1 executa-se o seguinte comando: .\geth --datadir C:\MyTestNet\data\node1 console 2>console1.log como mostra a Figura 12 .

Resposta:

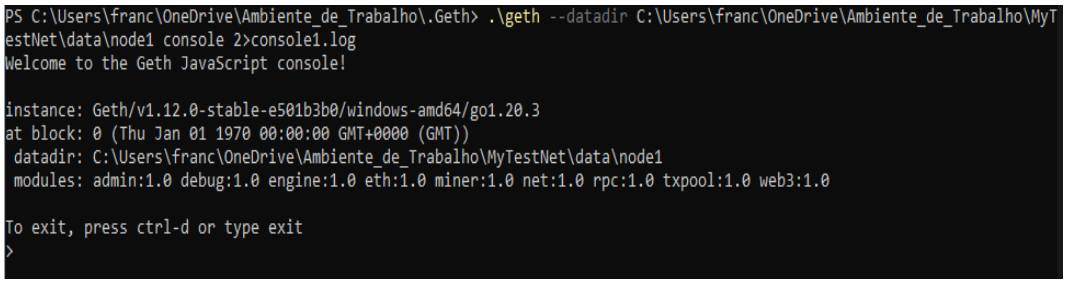


Figura 12 - Consola Javascript

Assim vai abrir uma consola javascrip que funciona com comandos de javascript. Então com o comando eth.accounts conseguimos ver as contas que estão no node1, que serão mostradas como Array e os detalhes estão armazenados num arquivo (com o nome UTC--...) n C:\MyTestNet\data\node1\keystore.

3.4.6. Verificar o saldo de uma conta

Para consultar o saldo de uma conta, é necessário usar a função eth.getBalance(): e executando o comando: eth.getBalance(eth.accounts[0]) consegue-se ver o saldo da primeira conta do node, como a lista de conta está num Arrey o 0 significa que é a primeira conta, 1 a segunda e por aí adiante...

A conta foi apenas criada, a resposta da consola vai ser 0, pois ainda não temos saldo na conta, no entanto, a unidade exibida para o saldo é Wei, onde 1 Ether é 10000000000000000000 Wei (1 seguido de 18 zeros), mas a função web3.fromWei(), converte o saldo de Wei para Ether: com o comando, web3.fromWei(eth.getBalance(eth.accounts[0]), "ether").

3.4.7. Inicialização node2

Com o node1 encerrado, inicializa-se o node2.

Comando: .\geth --datadir C:\MyTestNet\data\node2 --port 30304 --nodiscover --networkid 2345 console 2>console2.log

A inicialização do node2 é ligeiramente diferente do node 1 pois agora usa-se a opção --port que define a porta como 30304. Pois assim evitará conflitos com outro node que está a usar a porta padrão, logo em seguida usa-se a opção --nodiscover que serve para que os pares precisam ser adicionados manualmente, e por fim a opção --networkid especifica o id de rede para que outros nodes possam se conectar à rede com o mesmo ID.

3.4.8. Inicialização node1 na networkid

Voltando ao node1 houve a necessidade de inicializar outra vez, mas desta vez com uma pequena alteração no comando, .\geth --datadir C:\MyTestNet\data\node1 --networkid 2345 console 2>console1.log

O node1 é inicializado com a opção --networkid 2345, isso é necessário para que possa ser adicionado como um par ao node2 posteriormente.

Mais uma vez irá abrir uma consola Javascript.

3.4.9. Obter informações sobre o node

Na consola usa-se o comando, admin.nodeInfo para se obter os dados do node como mostra a Figura 13.

Resposta:

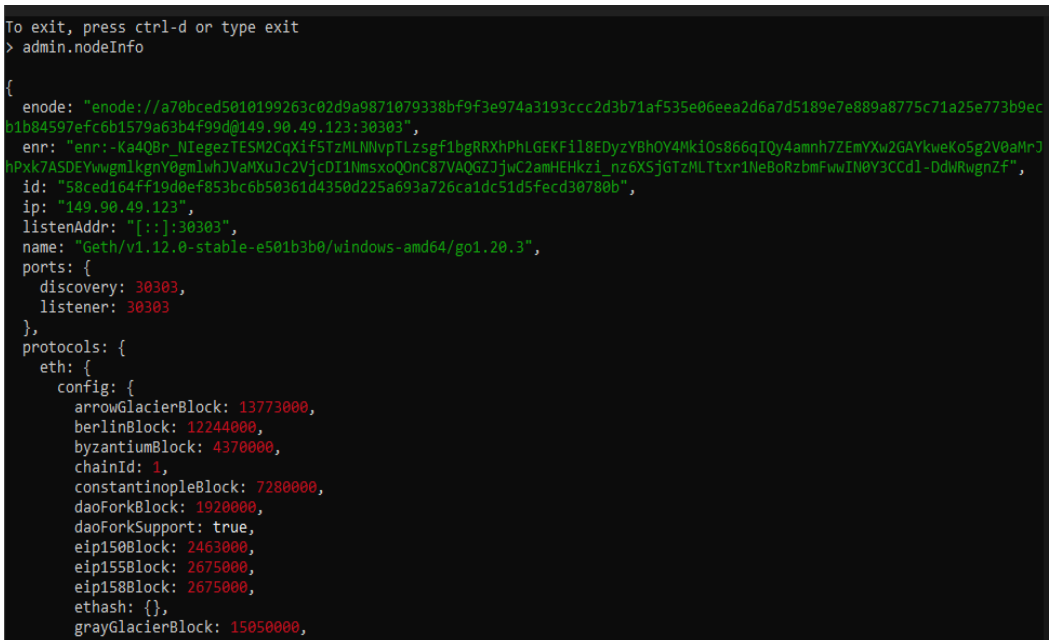


Figura 13 - Dados do node

Assim consegue-se ver o enode para avançar para o passo seguinte.

3.4.10. Conectando os nodes

Para conectar os nodes tem de se usar a função admin.addPeer(). A função adiciona um ponto ao node atual usando o valor enode do ponto, para isso usa-se o

comando:

admin.addPeer("enode://a70bced5010199263c02d9a9871079338bf9f3e974a3193ccc2d3b71af535e06eea2d6a7d5189e7e889a8775c71a25e773b9ecb1b84597efc6b1579a63b4f99d@149.90.49.123:30303"), e a resposta obtida será: true.

Por fim como mostra a Figura 14 para verificar se o par foi adicionado com sucesso, usa-se o comando, admin.peers, que se tudo estiver certo vai ser esta a resposta:

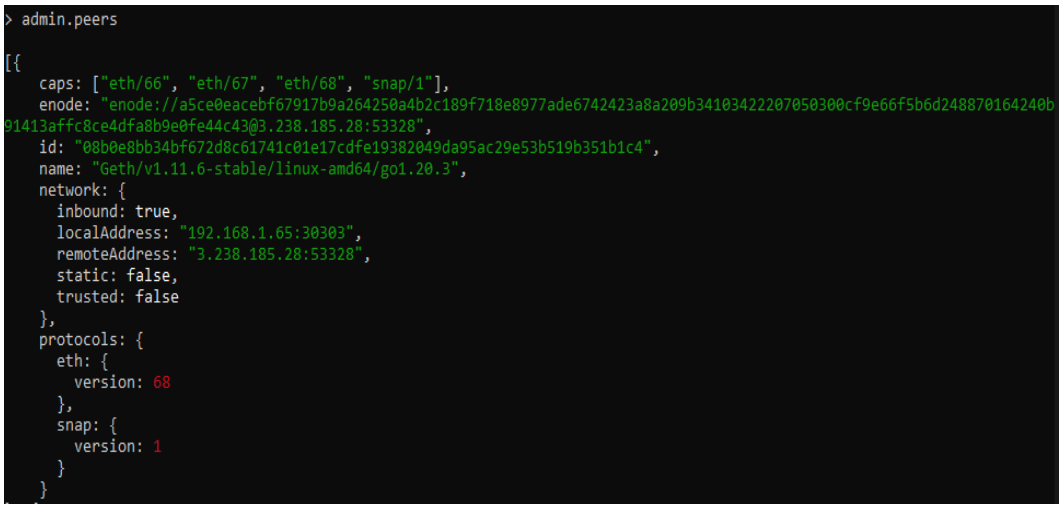


Figura 14 - Verificação do par adicionado

3.5.Criação da DApp Sell House

Para criar uma DApp precisamos primeiro, de criar uma interface para a aplicação. Foi usado o Android Studio para desenvolver essa interface. A DApp foi programada na linguagem de programação Kotlin. Para a DApp desenvolvida o smart contract usado foi o sell house, a interface desenvolvida foi baseada na interface do Remix IDE.

3.5.1. Interface da DApp

A interface Home é a página inicial da DApp, exibindo o logo da aplicação e um botão "start" que redireciona para a segunda interface como é demonstrado na figura 15.

Nessa interface, os utilizadores têm o primeiro contato com a plataforma, que tem um aspeto simples e intuitivo. [14].

Interface home:

Figura 15 - Página Home

Interface Contract Data:

Na interface Contract Data, como mostra a Figura 16 pede as informações necessárias sobre a casa escolhida para ser colocada à venda. Submetendo estes dados é criado um contrato na blockchain.

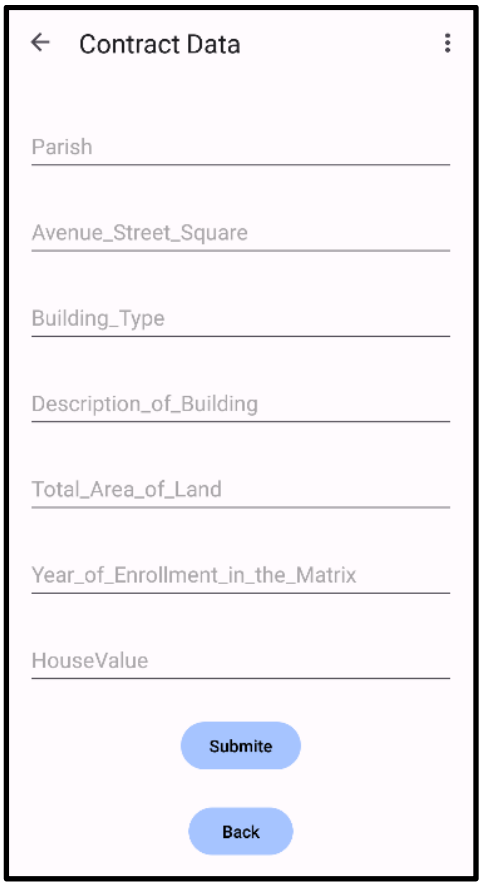


Figura 16 - Página Contract Data

Interface Operations:

Na interface Operations, como demonstra a Figura 17, mostra um conjunto de operações que podem ser feitas pelo utilizador. Podem ser realizadas três tipos de operações com Smart Contracts: criação de um Smart Contrat (criação de um endereço na blockchain em que o smart contract- código reside), alteração de dados num Smart Contract já residente na blockchain (botões laranjas e vermelho), obter dados da blockchain (botões azuis)

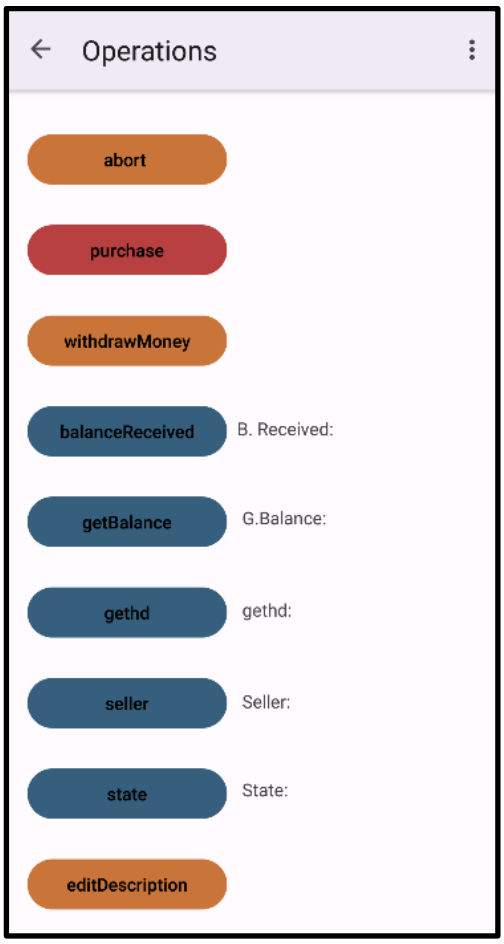


Figura 17 - Página Operations

Interface Edit House Description:

A interface Edit House Description, Figura 18 é exclusiva ao dono do imóvel, e é onde ele pode alterar os dados do imóvel, mesmo depois de ser colocado à venda.

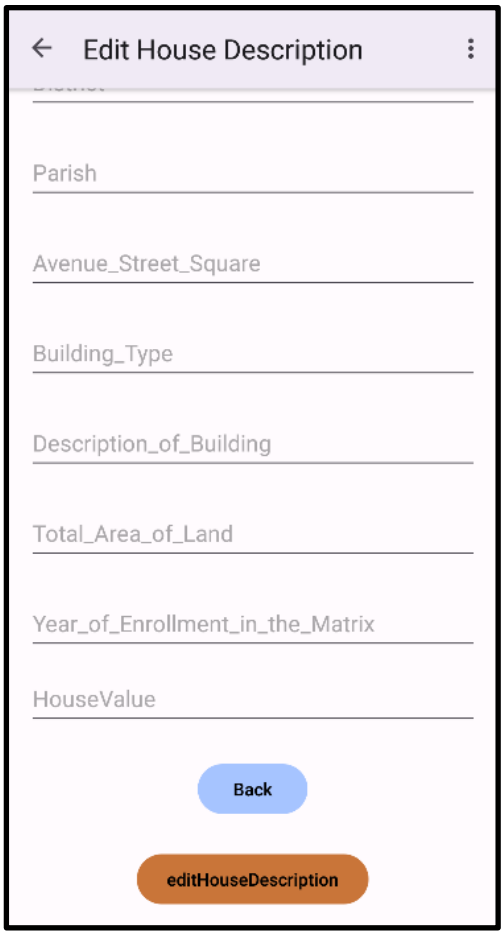


Figura 18 - Página Edit House Description

4. Criação de um curso sobre Blockchain

Durante o percorrer deste estágio juntamente com o professor Paulo Vieira, foi criado um curso sobre as tecnologias Blockchain, esse curso está dividido em 5 módulos:

4.1. Blockchain Introdução à tecnologia Blockchain

O primeiro módulo faz uma introdução sobre a tecnologia blockchain, e nos familiarizamos com os conceitos principais da tecnologia.

Figura 19 - Módulo 1

Source: Imagem própria

4.2. Implementando a tua própria blockchain

No segundo módulo descreve-se como se implementa uma rede peer to peer, python, e se implementa um sistema de visualização da blockchain, em javascript.

Figura 20 - Módulo 2

Source: Imagem própria

4.3.Ligação à Ethereum

No terceiro módulo, explica e exemplifica como se ligar a Ethereum.

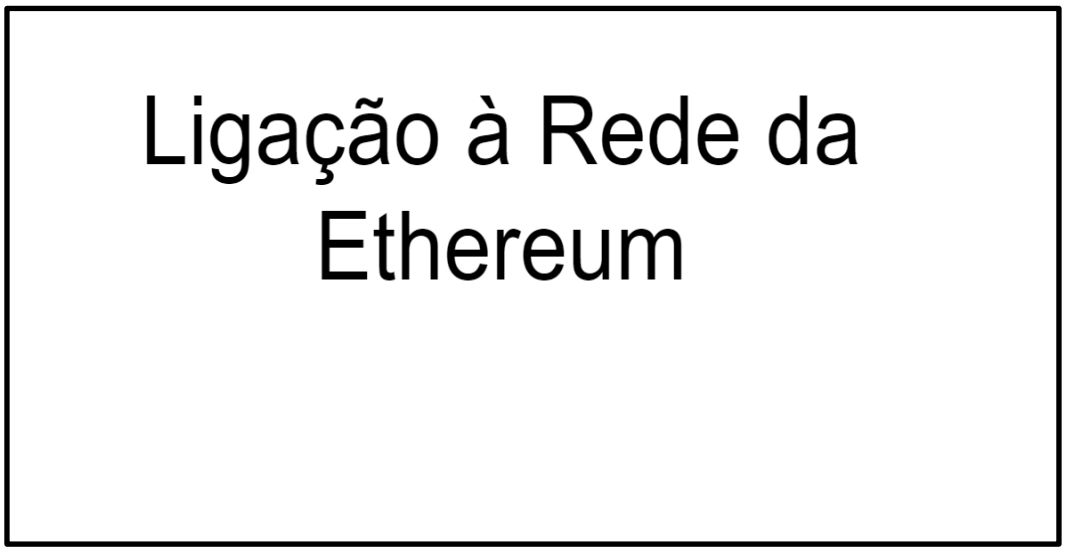


Figura 21 - Módulo 3

Source: Imagem própria

4.4.Criação rede de teste na Ethereum

No quarto módulo, explica e exemplifica como se cria uma rede de testes na Ethereum.

Figura 22 - Módulo 4

Source: Imagem própria

4.5.Ganache

O quinto módulo, mostra a conexão do ganache com o remix, algumas transações e Smart Contracts.

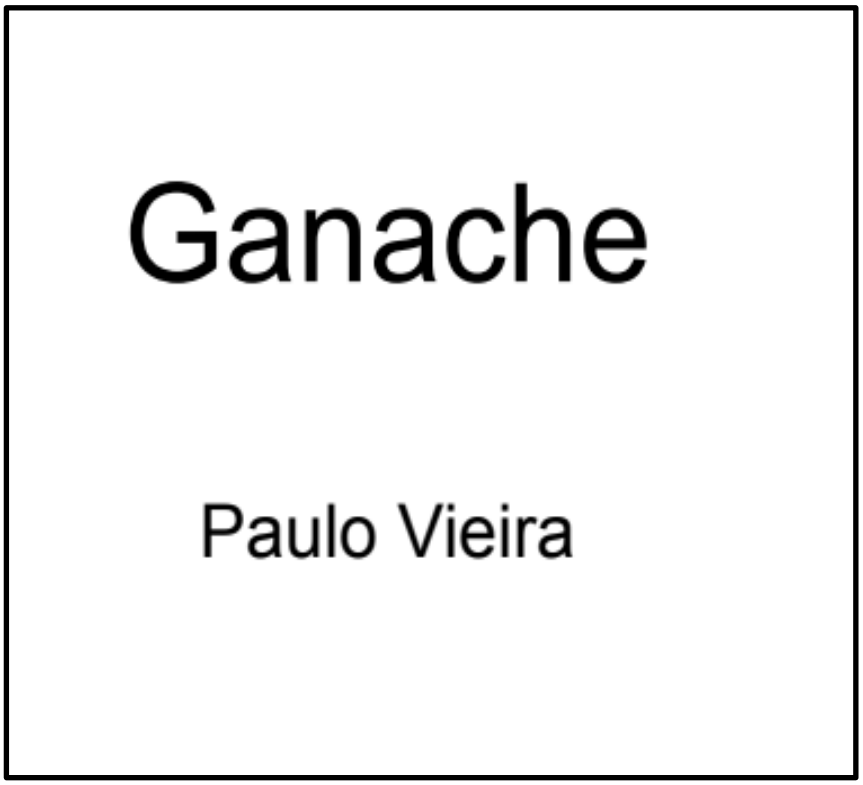


Figura 23 - Módulo 5

Source: Imagem própria

5. Testes

Durante o período do estágio, foram conduzidos diversos testes com o objetivo de avaliar a qualidade e o desempenho dos sistemas e projetos em desenvolvimento. Os principais testes realizados, foi a criação de Smart Contracts (HelloWorld, Ver_e_definir_variavel, Maquina_vendas, Enviar_dinheiro, Endereço, Wie_para_Ether, Sell_house), foi testado também ligação do Remix IDE com o Ganache e MetaMask, a interação com a Blockchain em Javascript e no localhost, a interação com a Blockchain em Python tanto no Localhost como em computadores diferentes, e por fim c conexão à rede da Ethereum.

5.1.Criação de Smart Contracts

Com os Smart Contracts criados, foi necessário testar a ver se correspondia ao objetivo pretendido.

5.1.1. HelloWorld

A Figura 24 mostra a execução feita através do Remix IDE do Smart Contract HelloWorld, onde verificamos que a variável obtida é hello world.

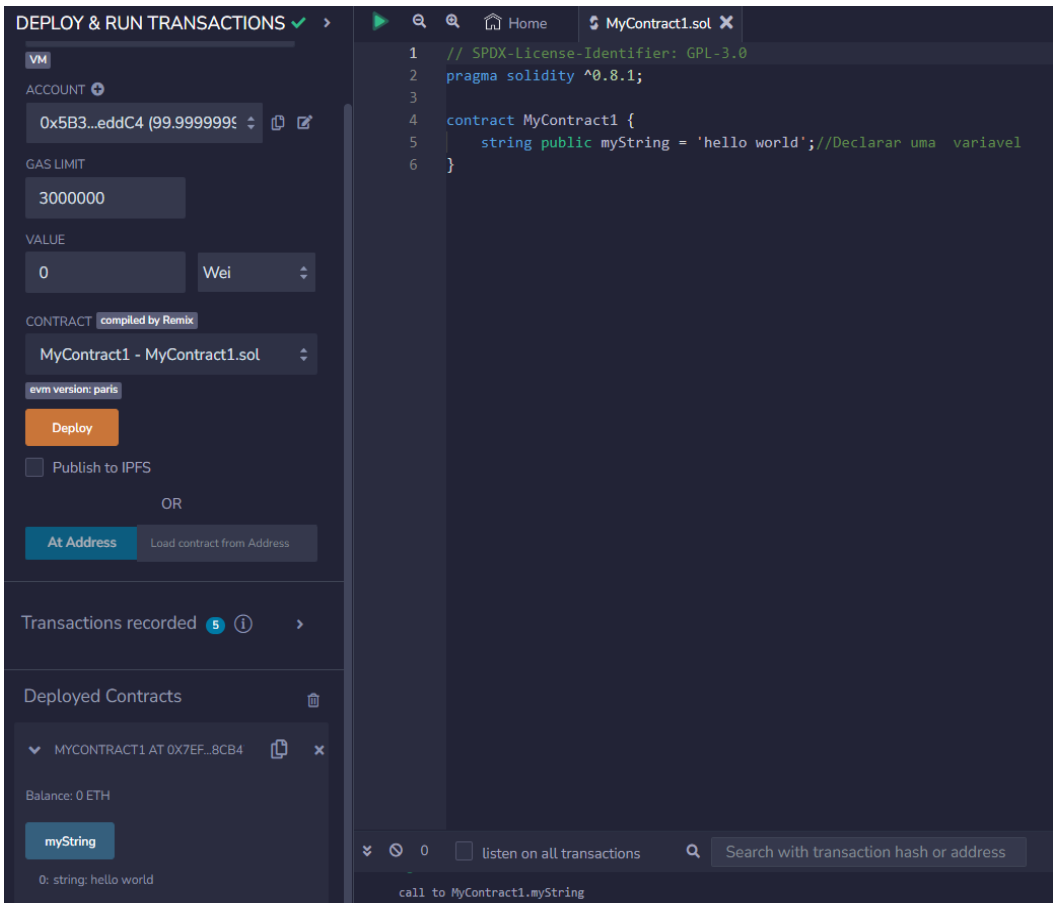


Figura 24 -Teste do Smart Contract Hello World

5.1.2. Ver_e_definir_variavel

A Figura 25 mostra a execução feita através do Remix IDE do Smart Contract Ver e definir variável.

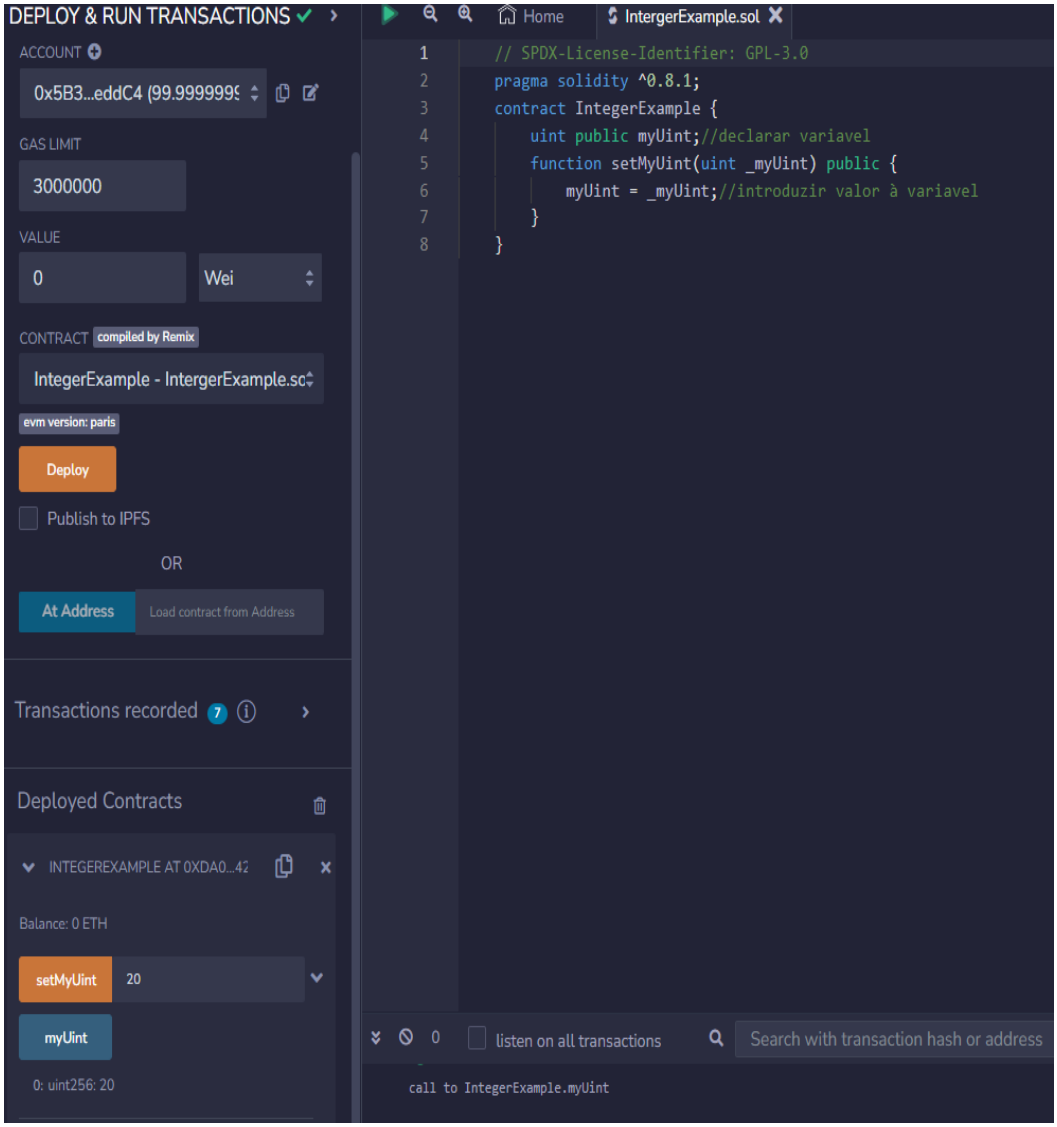


Figura 25 -Teste do Smart Contract Ver e definir variável

Figura 24 -Teste do Smart Contract Ver e definir variável

Como se pode verificar ao carregar no botão myUint, como no Smart Contract chama a função que por sua vez irá devolver o valor da variável, neste caso seria 0, mas quando usamos o botão setMyUint consegue-se alterar o valor da variável, e ao voltar a usar o botão myUint vais devolver o valor da variável já alterada.

5.1.3. Maquina_vendas

A Figura 26 mostra a execução feita através do Remix IDE do Smart Contract Máquina vendas, onde podemos comprar cupcakes.

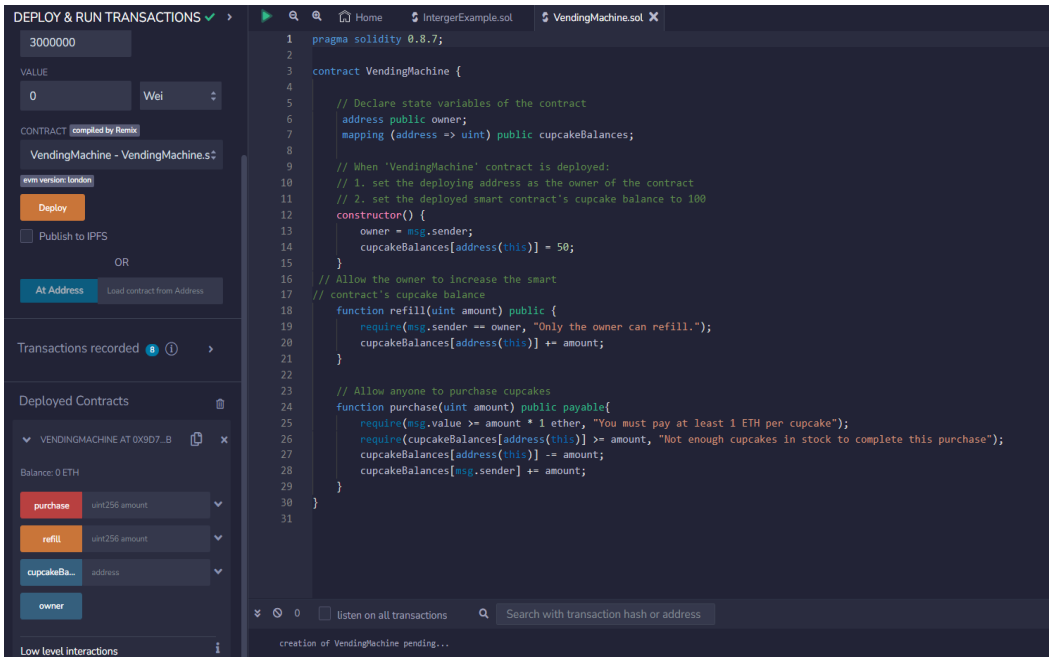


Figura 26 - Teste do Smart Contract Maquina de vendas

5.1.4. Enviar_dinheiro

A Figura 27 mostra a execução através do Remix IDE do Smart Contract Enviar dinheiro para o contrato e ver o dinheiro enviado e acumulado nele com as variáveis receiveMoney, balanceRecieve e getBalence.

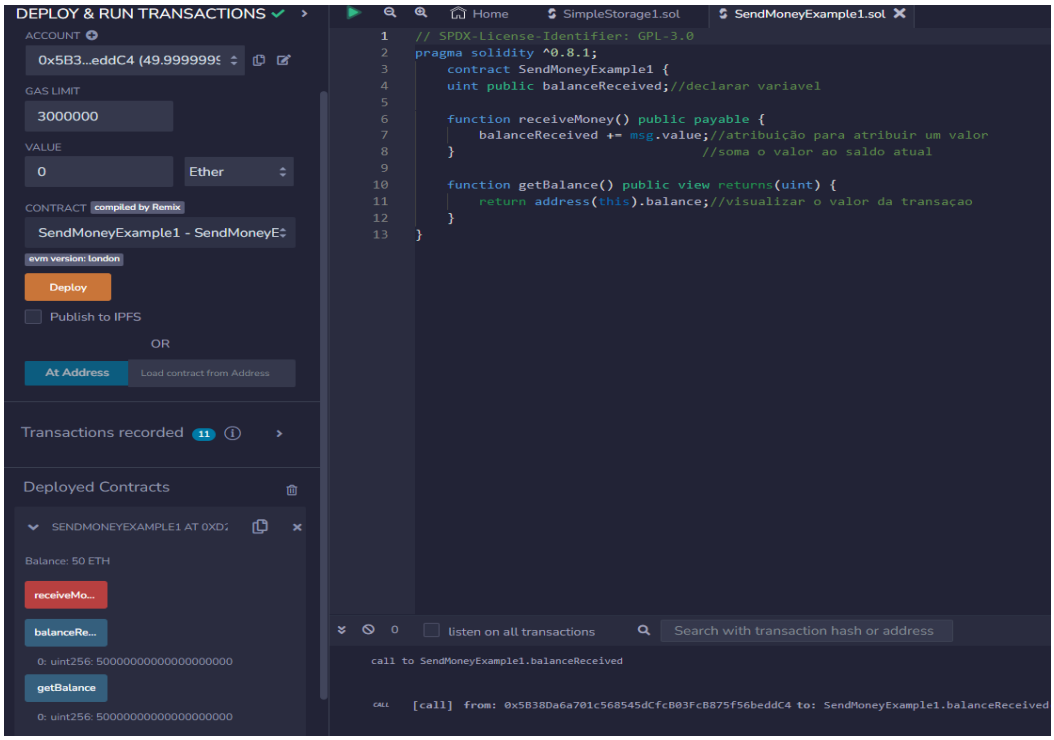


Figura 27 - Teste do Smart Contract Enviar dinheiro

5.1.5. Endereço

A Figura 28 mostra a execução feita através do Remix IDE do Smart Contract Endereço.

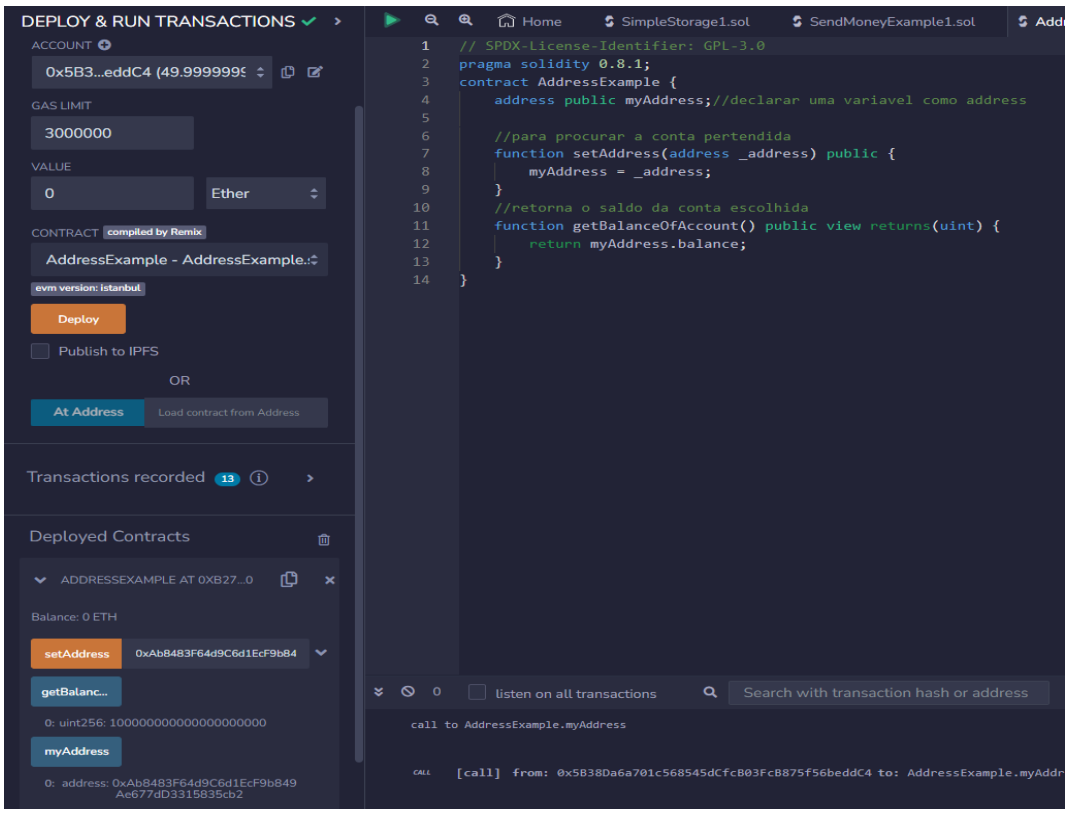


Figura 28 - Teste do Smart Contract Endereço

Com Smart Contract só necessitamos de introduzir um endereço de uma wallet escolhida, a carregar no botão setAddress como mostra o exemplo para dizer ao contrato qual é a wallet, após esse passo executado, ao utilizar o botão myAddress conseguimos ver se realmente é o endereço de wallet escolhido ou se houve algum erro, e por fim utilizando o botão getBalenceOfAccount conseguimos ver o saldo contido na wallet escolhida.

5.1.6. Wie_para_Ether

A Figura 29 mostra a execução feita através do Remix IDE do Smart Contract Wei para Ether, designado o número de weis que queremos converter para ethers.

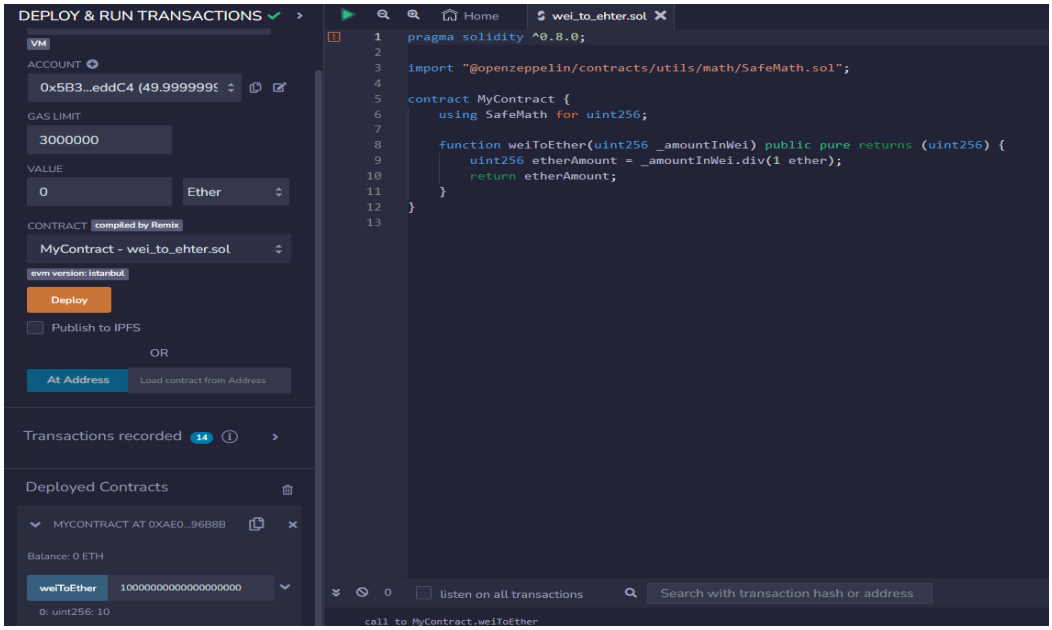


Figura 29 - Teste do Smart Contract Wei para Ether

5.1.7. Sell_House

A Figura 30 mostra a execução feita através do Remix IDE do Smart Contract Sell House, onde o dono pode fazer sets ao contrato para abortar o contrato, receber o dinheiro que está no contrato e editar os dados do mesmo, os outros utilizadores podem fazer gets ao contracto para poderem ver os dados do contrato e por fim comprar caso pretendam.

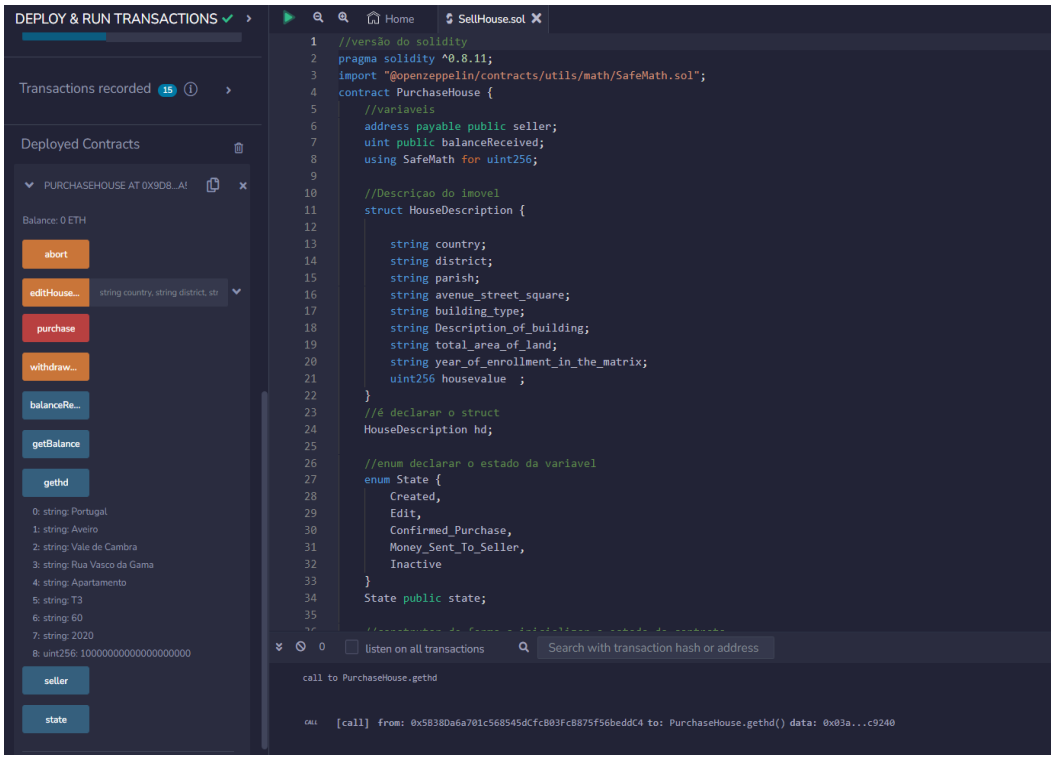


Figura 30 - Teste do Smart Contract Sell House

5.2.Fazer a ligação do Remix IDE com o Ganache e MetaMask

Aqui consegue-se ver a conexão estabelecida entre Remix IDE e o Ganache, também foi estabelecida uma conexão entre o Remix IDE com a wallet MetaMask.

5.2.1. Ligação do Remix IDE com o Ganache

Aqui executa-se a conexão do Remix IDE com o Ganache, consegue-se ver que foi estabelecida a conexão pois no primeiro campo de Remix IDE estamos a utilizar a opção do Ganache e conseguimos verificar que o endereço e dados da wallet

selecionado no Remix IDE coincide com o primeiro endereço e dados da wallet do Ganache.

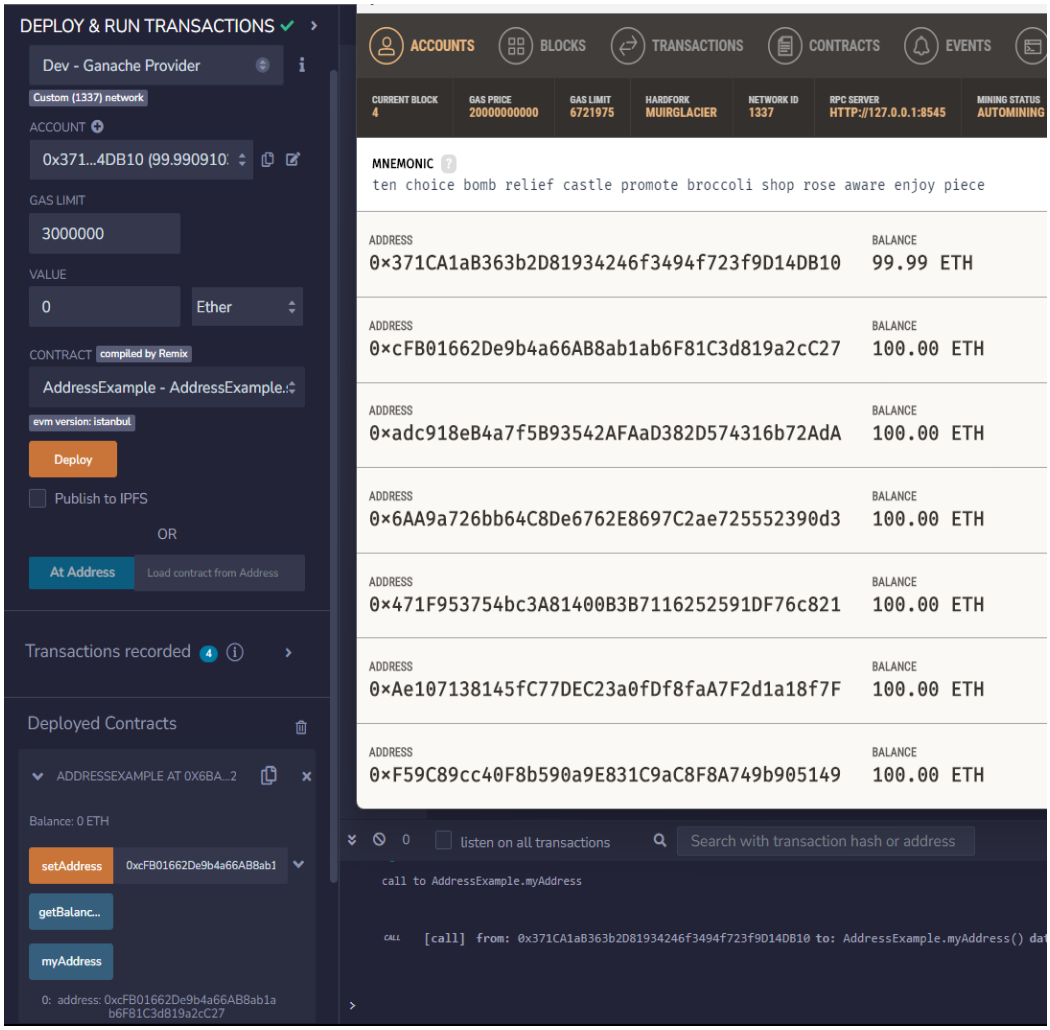


Figura 31 - Remix e Ganache Conectados

5.2.2. Ligação do Remix IDE com o MetaMask

Aqui executa-se a conexão do Remix IDE com o Ganache

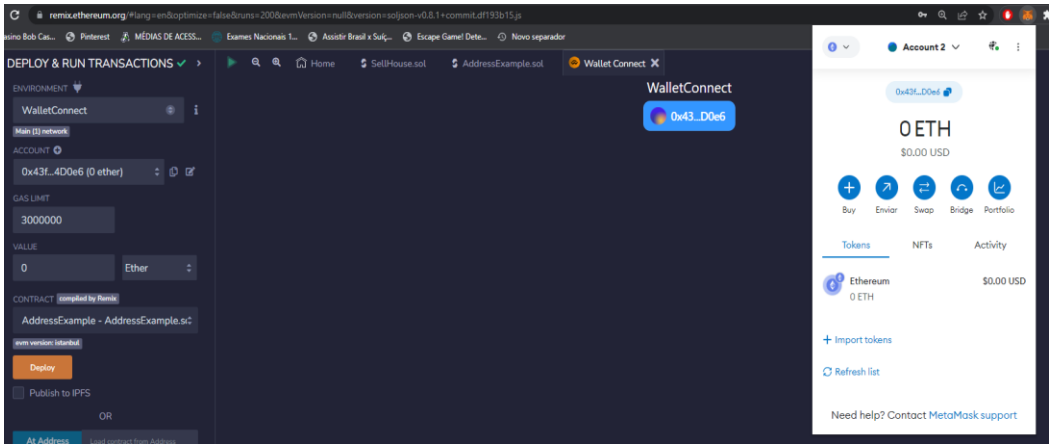


Figura 32 - Remix e MetaMask Conectados

5.3.Interação com a Blockchain em Javascript e no localhost.

```
C:\Users\franc>cd C:\Users\franc\OneDrive\Ambiente de Trabalho\Blockchain\blockchain_platform\conceptual_blokchain-no
dejs

C:\Users\franc\OneDrive\Ambiente de Trabalho\Blockchain\blockchain_platform\conceptual_blokchain-nodejs>node p2p.js
myPeerId: bbfa942d7d5d847dfc5cdacf5899988c4e07a8584f73f53465b851cba994b8ac
Listening port: 56993
Connected #${seq} to peer: ${peerId}
Connected #${seq} to peer: ${peerId}
Connection ${seq} closed, peerId: ${peerId}
-------- writeMessageToPeers start --------
type: hello, to: c30caba71546c57f024ef4ffa4dff7f74d5ce08bb6bdfaafde78216357539f73
-------- writeMessageToPeers end -----------
```

Figura 33 - Mensagem enviada

```
C:\Users\franc>cd C:\Users\franc\OneDrive\Ambiente de Trabalho\Blockchain\blockchain_platform\conceptual_blokchain-no
dejs

C:\Users\franc\OneDrive\Ambiente de Trabalho\Blockchain\blockchain_platform\conceptual_blokchain-nodejs>node p2p.js
myPeerId: c30caba71546c57f024ef4ffa4dff7f74d5ce08bb6bdfaafde78216357539f73
Listening port: 56986
Connected #${seq} to peer: ${peerId}
Connected #${seq} to peer: ${peerId}
Connection ${seq} closed, peerId: ${peerId}
----------- Received Message start -------------
from: bbfa942d7d5d847dfc5cdacf5899988c4e07a8584f73f53465b851cba994b8ac to: bbfa942d7d5d847dfc5cdacf5899988c4e07a8584f
73f53465b851cba994b8ac my: c30caba71546c57f024ef4ffa4dff7f74d5ce08bb6bdfaafde78216357539f73 type: "hello"
----------- Received Message end -------------
```

Figura 34 - Mensagem recebida

5.4.Interação com a Blockchain em Python.

Este capítulo aborda como é feita a interação com uma blockchain.

5.4.1. No LocalHost

Para conseguir instanciar as redes, é necessário abrir 3 CMD's e abrir a pasta onde a programação da rede se encontra, dentro da pasta, para fazer a instanciação da rede criada usa-se o comando: python nome_de_ficheiro.py e a porta (ex: python blockchain.py 5000), isto para 2 dos CMD's, nota, a escolha da porta para os 2 CMD's tem de ser diferente.

Feito isto num no último CMD, é utilizado o comando curl para fazer uma mineração, cal as cadeias de blocos e a sua sincronização.

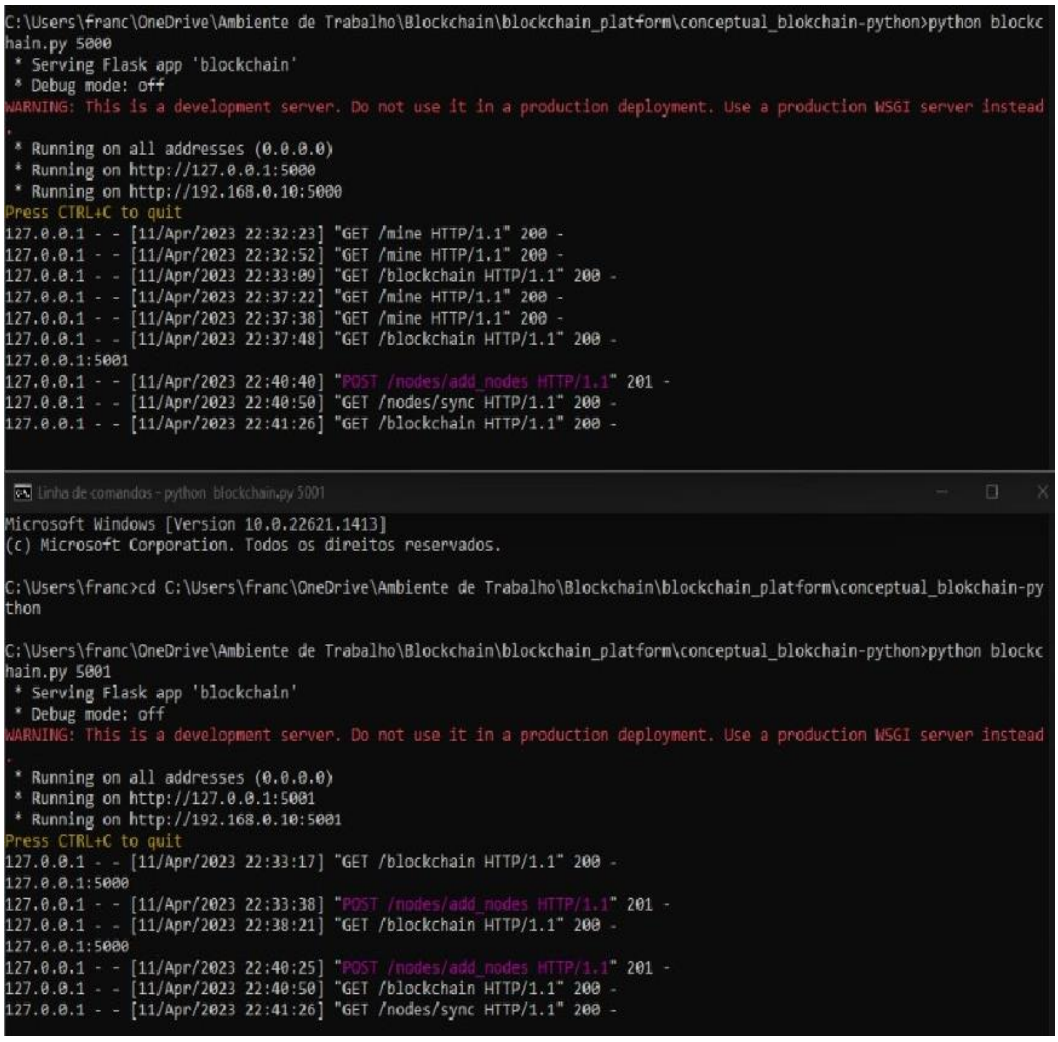


Figura 35 - Instanciação da rede 2p2 em portas diferentes

Figura 36 - Sincronização das redes 2p2

5.4.2. Em Computadores Diferentes

Para conseguir instanciar as redes, é necessário abrir o CMD e abrir a pasta onde a programação da rede se encontra, dentro da pasta, para fazer a instanciação da rede criada usa-se o comando: python nome_de_ficheiro.py e a porta (ex: python blockchain.py 5000), isto para cada um dos computadores, nota, a escolha da porta para os 2 computadores tem de ser diferente como mostra a Figura 37 e 38.

Feito isto num outro computador é necessário abrir o CMD e abrir a pasta onde a programação da rede se encontra, e é utilizado o comando curl para fazer uma mineração, e se obtém a cadeia de blocos em cada computador como mostra a Figura 39.

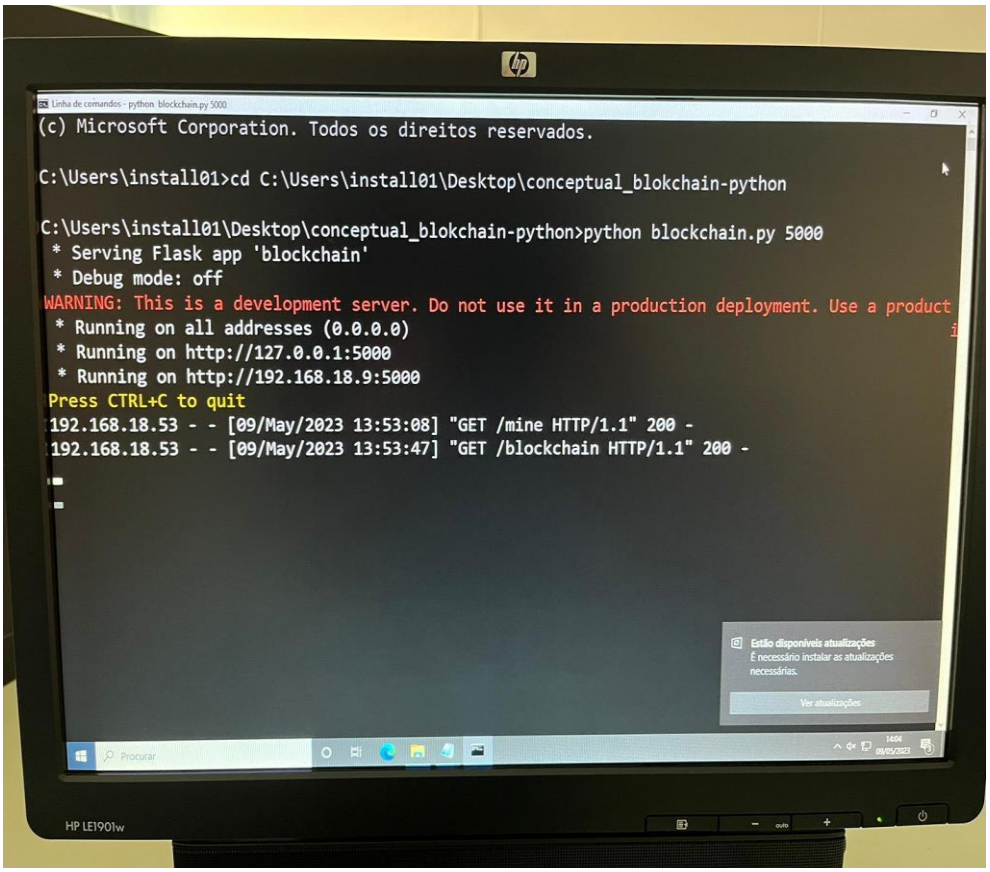


Figura 37 - Instância a rede (pc1 porta:5000 ip:192.168.18.9)

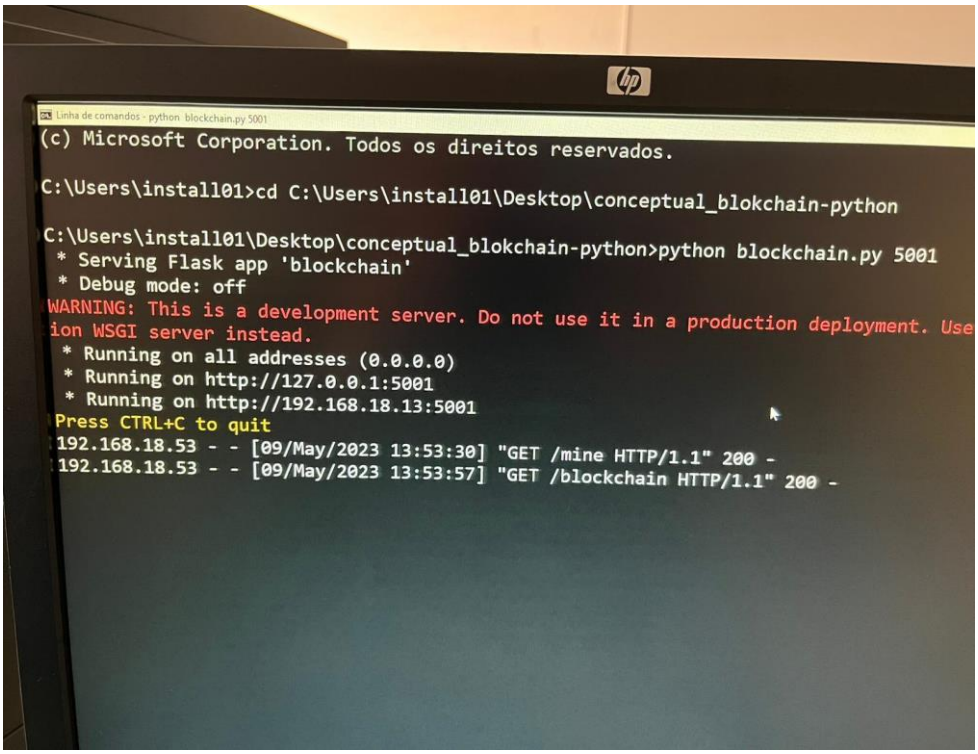


Figura 38 - Instância a rede (pc2 porta:5000 ip:192.168.18.13)

Figura 39 - Ações na blockchain

5.5.Conexão à rede da Ethereum

Para fazer a conexão à rede da Ethereum, o Primeiro passo é fazer a criação de uma pasta para armazenar a blockchain, keystore e outros dados do client local (ex: c:\.ethereum-testnet).

Com os os requisitos API's instaladas, abrimos o cmd vamos fazer a instanciação da nossa rede peer to peer na porta 30303, pois é a porque que o Client Geth utiliza por defeito, feito isto, há a necessidade de abrir outro cmd, e através desse cmd abrimos a nossa pasta onde foi feita a instalação do Client Geth, a partir daqui já só falta o último passo que escolher qual rede da Ethereum queremos, para isso no segundo cmd escrevemos este comando: **.\geth --datadir c:\.ethereum-testnet**.

A opção (--datadir) especifica o caminho local a ser usado para armazenar a blockchain, keystore e outros dados do client local. (c:\.ethereum-testnet).

Toda a blockchain da rede estará guardada no computador, neste caso, na pasta c:\.ethereum-testnet.

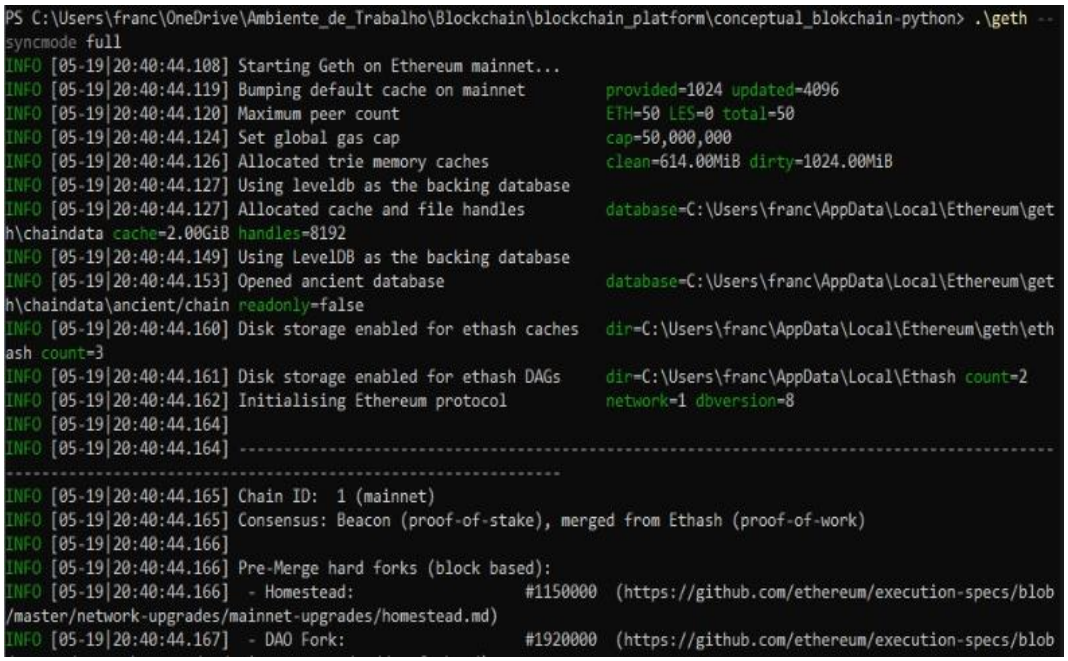


Figura 40 - Ligação à mainnet da Ethereum

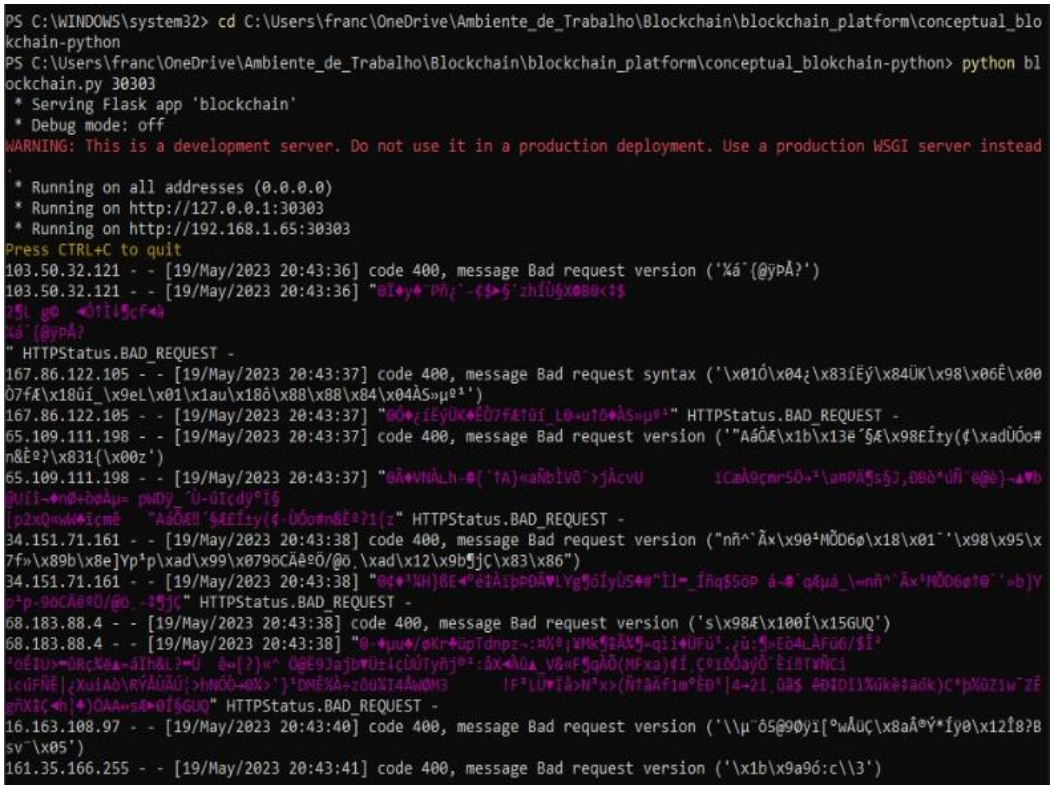


Figura 41 - Rede p2p conectada à Ethereum

6. Conclusão

Durante o período de estágio, foi possível explorar e compreender melhor a tecnologia blockchain e o seu potencial uso e impacto em diversas áreas. Foi observado que, apesar de muitas pessoas associarem blockchain às criptomoedas, essa tecnologia vai além disso, oferecendo segurança e transparência para dados e informação.

O maior desafio que eu me apercebi ao longo deste estágio, em relação à tecnologia blockchain, é sem haver diminuição da segurança na rede aumentar a rentabilidade baixando o consumo energético e computacional, para isso tenta-se utilizar as provas de consenso, que não fazem apenas minerações.

No geral, o estágio proporcionou uma imersão no mundo da blockchain, permitindo compreender sua aplicabilidade, desafios e benefícios. A familiarização com a programação de Smart Contracts, a criação de redes peer to peer e a implementação de DApps contribuíram para a formação de habilidades essenciais, nesse campo em constante evolução. O conhecimento obtido durante o estágio certamente será aplicado em futuros projetos e contribuirá para acompanhar as inovações e avanços no campo da blockchain.

Bibliografia

[1] "Remix IDE," [Online]. Available: https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.19+commit.7dd6d404.js[Acedido em 20 julho 2023].

[2] "Ganache," [Online]. Available: https://trufflesuite.com/ganache/[Acedido em 20 julho 2023].

[3] "MetaMask," [Online]. Available: https://metamask.io/[Acedido em 20 julho 2023].

[4] "Visual Studio Code," [Online]. Available: https://code.visualstudio.com/[Acedido em 20 julho 2023].

[5] "Android Studio," [Online]. Available: https://developer.android.com/studio[Acedido em 20 julho 2023].

[6] "Solidity Tutorial," [Online]. Available: https://www.tutorialspoint.com/solidity/index.htm[Acedido em 20 julho 2023].

[7] "SolidityLang," [Online]. Available: https://docs.soliditylang.org/en/v0.8.20/[Acedido em 20 julho 2023].

[8] E. Elrom, The Blockchain Developer: A Practical Guide for Designing, Implementing, Publishing, Testing, and Securing Distributed Blockchain-based Projects, Nova York, EUA: Apress, 2020.

[9] W.-M. Lee, Beginning Ethereum Smart Contracts Programming: With Examples in Python, Solidity, and JavaScript, Nova York, EUA: Apress, 2019.

[10] "GitHub," [Online]. Available: https://github.com/franciscop101/Estagio_Blockchain_1704082[Acedido em 20 julho 2023].

[11] "Javascript Tutorial," [Online]. Available: https://www.w3schools.com/js/[Acedido em 20 julho 2023].

[12] "Python Tutorial," [Online]. Available: https://www.w3schools.com/python/[Acedido em 20 julho 2023].

[13] "Geth Client," [Online]. Available: https://geth.ethereum.org/downloads[Acedido em 20 julho 2023].

[14] "GitHub," [Online]. Available: https://github.com/franciscop101/Estagio_Blockchain_1704082_DApp[Acedido em 20 julho 2023].

Anexos

Converter Weis para Ethers:

Table 4-1. *Units in Ethereum*

Unit	Wei Value	Wei
wei	1 wei	1
Kwei (babbage)	10^3 wei	1,000
Mwei (lovelace)	10^6 wei	1,000,000
Gwei (shannon)	10^9 wei	1,000,000,000
microether (szabo)	10^{12} wei	1,000,000,000,000
milliether (finney)	10^{15} wei	1,000,000,000,000,000
ether	10^{18} wei	1,000,000,000,000,000,000

Figura 42 - Conversor de Wei para Ether