



COOVER

2023

INTELI

DIÁRIO DE BORDO

COOVER

DEPLOY SMART CONTRACT



BEM - VINDO

Autores:

- Eric Tachdjian
- Giovanna Furlan
- Henri Harari
- Maria Luísa Maia
- Matheus Fidelis
- Ueliton Rocha

Orientador:

- Renato Penha

Coordenador:

- Egon Daxbacher

Instituição:

- Instituto de Tecnologia e Liderança

COOVERCHAIN



SUMÁRIO

1. Introdução	4
2. Deploy Truffle	6
3. Deploy Ganache	11
4. Metamask Goerli	14
5. Metamask Deploy	18

COOVERCHAIN

INTRODUÇÃO

Este documento é um diário de bordo, previsto para a documentação de deploy do smart contract, para a empresa Coover. Sendo um registro escrito que documenta os eventos e observações feitas durante o procedimento.

Neste caso, o diário de bordo inclui informações acerca de quais plataformas o deploy foi realizado, qual ETH foi utilizado para a transação, quais foram as maiores dificuldades para o processo, qual foram os maiores acertos, como foi a integração com a wallet e quais são as possíveis melhorias a serem realizadas.

Tais registros são utilizados para acompanhamento do desenvolvimento da aplicação, além de ser útil para identificação de possíveis erros e tendências que possam ocorrer.

TESTNET PÚBLICA

Testnet pública é uma rede de teste da blockchain aberta ao público que é usada para testar e validar, neste caso, contratos inteligentes, sem gastar ether real ou executar transações na rede principal do blockchain. A importância de sua utilização é voltada para criar um ambiente em que consiga-se testar e validar as aplicações antes de implantá-las na rede principal (mainnet). Contribuindo para a redução de risco e falhas, que ocasionam perda de recursos em transações reais na rede principal.



PLATAFORMAS UTILIZADAS

Para realizar o deploy do smart contract, foi-se necessário o uso de quatro plataformas/frameworks, sendo elas:

- **Truffle:** Ferramenta de desenvolvimento de contratos inteligentes para Ethereum, que permite testar e implantar contratos inteligentes com facilidade.
- **Ganache:** Rede de desenvolvimento blockchain local, que simula uma rede Ethereum. Permite teste dos contratos inteligentes sem precisar gastar ether real ou executar transações em uma rede pública.
- **Goerli Faucet:** Plataforma que fornece ether gratuito na rede de testes Goerli do Ethereum para testar contratos inteligentes em um ambiente seguro e sem custo. O faucet do Goerli distribui ether gratuitamente para endereços de carteira que são especificados no site, sem precisar pagar taxas na rede principal do Ethereum.
- **Metamask:** Permite aos usuários acessar aplicativos descentralizados (dApps) baseados em Ethereum. Além do gerenciamento de chaves privadas e permite a interação com contratos inteligentes e outras funcionalidades da blockchain do Ethereum, sem precisar executar um nó completo da rede. O Metamask também fornece um ambiente de teste de rede sem gastar ether real.



DEPLOY TRUFFLE

07 de março de 2023 ás 14h.

CONFIGURAÇÃO AMBIENTE DE DESENVOLVIMENTO:

O ambiente de desenvolvimento utilizado neste deploy foi o truffle, segue a configuração feita.

- Baixar o Node.js no site oficial: <https://nodejs.org/>;
- Abrir uma pasta no VS Code pra o projeto;
- No terminal digitar o seguinte comando para instalar o Truffle:

```
● ● ●  
npm install -g truffle
```

- Executar o comando abaixo para criar um projeto Truffle, estrutura de diretórios básica:

```
● ● ●  
truffle init
```

- Adicionar o arquivo do contrato à pasta */contracts* criada na etapa anterior.



- Escrever no terminal o comando abaixo para conseguir compilar o contrato adicionado e gerar os arquivos de build :

```
● ● ●  
truffle compile
```

- Na pasta `/migrations` criar um arquivo no seguinte formato `"1_deploy.js."` responsável por implantar o contrato na blockchain, escrever o código abaixo para a implantação:

```
● ● ●  
  
const migrations = artifacts.require("cooverContract");  
module.exports = function (deployer) {  
  deployer.deploy(migrations);  
};''
```

OBS: Adicionar os argumentos de inicialização do contrato depois de "migrations"

- Neste caso, como a rede utilizada é a testnet, é preciso selecionar a rede. No arquivo `"truffle-config.js"`, descomentar o seguinte comando das linhas 85 a 91.

```
● ● ●  
  
goerli: {  
  provider: () => new HDWalletProvider( MNEMONIC,  
    `https://goerli.infura.io/v3/${PROJECT_ID}`  
  ),  
  network_id: 5,  
  confirmations: 2,  
  timeoutBlocks: 200,  
  skipDryRun: true  
},
```



- Baixar o “**dotenv**” para poder criar um arquivo para armazenar as variáveis informadas no provider, executar o código abaixo no terminal:

```
npm install dotenv
```

- Criar o arquivo “**.env**” na raiz para armazenar as variáveis que deseja “esconder” nele.
- No arquivo “**truffle-config.js**” e descomentar as linhas 44 a 47, Para atribuir essas variáveis no “**.env**”.

```
require('dotenv').config();
const { MNEMONIC, PROJECT_ID } = process.env;

const HDWalletProvider = require('@truffle/hdwallet-provider');
```

- Para que o comando anterior funcione como previsto é necessário rodar no terminal o código abaixo:

```
npm install @truffle/hdwallet-provider
```

- No site <https://app.infura.io/register> pegar a API KEY e no arquivo .env é necessário inserir os seguintes comandos :

```
MNEMONIC = "sua mnemonic phrase"
INFURA_API_KEY = "api key do infura"
```



- Trocar nos códigos onde existe o seguinte comando "PROJECT_ID" para "INFURA_API_KEY".
- Para finalizar o deploy, inserir o seguinte comando no terminal:

```
● ● ●
truffle deploy --network goerli
```

RESULTADO DO DEPLOY

Após a conclusão da configuração do ambiente de desenvolvimento, o deploy com Truffle foi realizado e nessa sessão apresenta-se os resultados acerca dos erros e acertos de execução.

O contrato foi compilado e seu deploy foi executado com sucesso, na imagem abaixo, apresenta-se as informações do mesmo, juntamente com o endereço do contrato e o tanto de gás que está sendo gasto atualmente.

```
1_coover.js
=====

Deploying 'cooverContract'
-----
> transaction hash: 0x2dd6d59062ccc233707b2c0b96122e7c5a927d764e3be2b0f5bb1c826351289b
> Blocks: 1 Seconds: 25
> contract address: 0x60cd81aeE3fd026231C16F230981352443F18537
> block number: 8630778
> block timestamp: 1678456932
> account: 0x9f80aC649f0244330aDB623D5D56c6b7db71bf3A
> balance: 0.227956548622764424
> gas used: 801134 (0xc396e)
> gas price: 17.279473078 gwei
> value sent: 0 ETH
> total cost: 0.013843173384870452 ETH

Pausing for 2 confirmations...

-----
> confirmation number: 1 (block: 8630779)
> confirmation number: 2 (block: 8630780)
> Saving artifacts
-----
> Total cost: 0.013843173384870452 ETH

Summary
=====
> Total deployments: 1
> Final cost: 0.013843173384870452 ETH
```



COISAS QUE DERÃO CERTO

- No final do processo o deploy foi realizado como esperado;
- Os diretórios com a estrutura de pastas foi criado com sucesso.
- O contrato foi compilado sem problemas;
- A configuração com a rede Goerli foi implementada com sucesso.

COISAS QUE DERÃO ERRADO

- O tutorial utilizado indicava que não era recomendado realizar as etapas com o terminal do powershell, enfrentamos alguns erros utilizando o terminal do CMD e quando passamos para o poshershell, eles se resolveram;
- Tivemos que rodar tudo do inicio umas 5 vezes, por erros de bibliotecas, falta de argumentos no código, além de versões diferentes que precisavam ser atualizadas.

FUTURAS MELHORIAS

- Considerar a inclusão de testes automatizados no diretório /test do Smart Contract;
- Trabalhar na otimização do código, para que o mesmo utilize menor quantidade de gas no deploy.



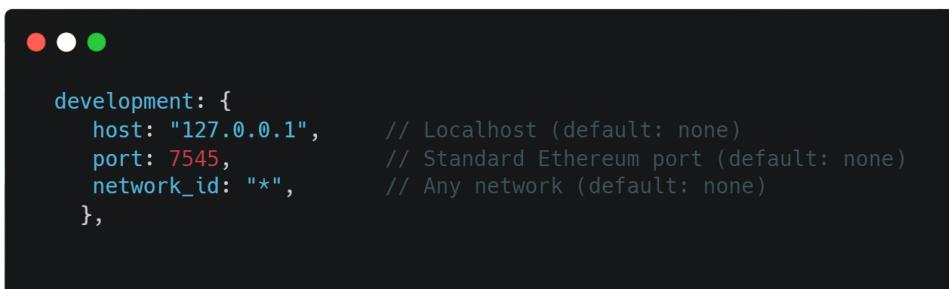
DEPLOY GANACHE

07 de março de 2023 ás 19h.

CONFIGURAÇÃO AMBIENTE DE DESENVOLVIMENTO:

O ambiente de desenvolvimento utilizado neste deploy foi o ganache, segue a configuração feita.

- Seguir os passos do tutorial acima, até o momento que cria-se o arquivo "*1_deploy.js.*" , dentro da pasta */migrations*;
- Após seguir o tutorial até essa etapa, se faz necessário baixar o aplicativo do ganache : <https://trufflesuite.com/ganache/>;
- No arquivo "*truffle-config.js*" tem que descomentar as linhas 67 a 71, para ativar o deploy na rede local.



```
development: {  
    host: "127.0.0.1",      // Localhost (default: none)  
    port: 7545,              // Standard Ethereum port (default: none)  
    network_id: "*",        // Any network (default: none)  
},
```

- Execute o comando abaixo para implantar o smart contract na rede de desenvolvimento Ethereum local.



```
truffle migrate
```



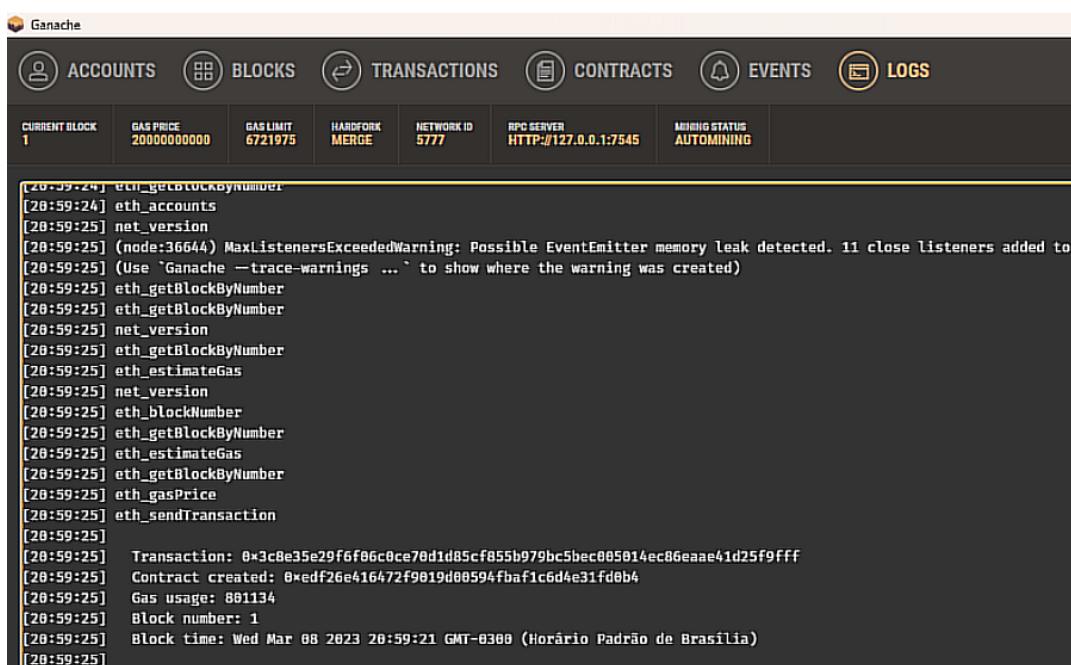
- No terminal rode o comando abaixo, para compilar o arquivo, espera-se ter o seguinte resultado, exibido abaixo:

```
PS C:\GitHub - Projetos\INTELI 2 AND\MODULO 1\Projeto2\src\Deploy Ganache> npx truffle compile
Compiling your contracts...
=====
> transaction hash: 0x3c8e35e29f6f06c0ce70d1d85cf855b979bc5bec005014ec86eaae41d25f9fff
> Blocks: 0
> contract address: 0xEdf26E416472F9019d00594fBaf1C6d4E31fd0b4
> block number: 1
> block timestamp: 1678319961
> account: 0x5b5033f77fA0a93FCEd794f5bEaC6d91187B6841
> balance: 99.99729617275
> gas used: 801134 (0xc396e)
> gas price: 3.375 gwei
> value sent: 0 ETH
> total cost: 0.00270382725 ETH

> Saving artifacts
=====
> Total cost: 0.00270382725 ETH

Summary
=====
> Total deployments: 1
> Final cost: 0.00270382725 ETH
```

- Abra o aplicativo do ganache e clique em "["QUICKSTART Ethereum"](#)" ;
- Após abrir o aplicativo, clique na guia "LOGS" e aparecerá o resultado do deploy, como o endereço do contrato e o tanto de gas utilizado, como na imagem abaixo.



COISAS QUE DERÃO CERTO

- No final do processo o deploy foi realizado como esperado;
- Ao seguir os passos do tutorial antigo, rodou os comandos sem problemas;
- O contrato foi compilado rapidamente;
- A utilização do aplicativo é fácil e intuitiva;
- A hospedagem local com o ganache ocorreu como esperado, sendo implementada com sucesso.

COISAS QUE DERÃO ERRADO

- Não tinha um tutorial específico para o ganache, o que dificultou os passos para implementação;
- Faltou explicação acerca do quanto é útil ter um deploy local, comparado a um hospedado na rede público, como o do truffle realizado acima.

FUTURAS MELHORIAS

- Considerar a inclusão de testes automatizados no diretório /test do Smart Contract (se possível);
- Trabalhar na otimização do código, para que o mesmo utilize menor quantidade de gas no deploy.



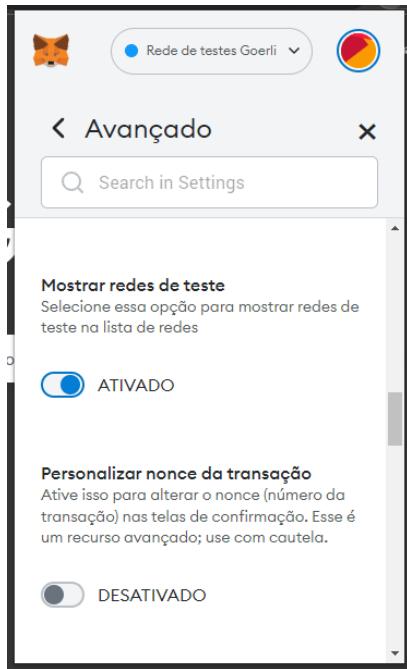
METAMASK GOERLI

07 de março de 2023 ás 20h.

CONEXÃO COM A WALLET E O ETH FAUCET

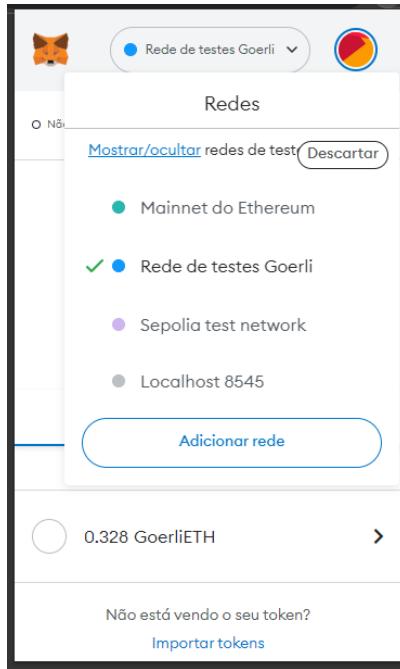
A conexão da wallet Metamask com o ETH Faucet, o Goerli, para realizar o teste do deploy do smart contract, segue a configuração feita.

- Abra a metamask e clique no ícone do campo superior direito, após clique em "**configurações**" e selecione a opção "**"Avançado"**" e ative a opção "**"Mostrar redes de teste"**";

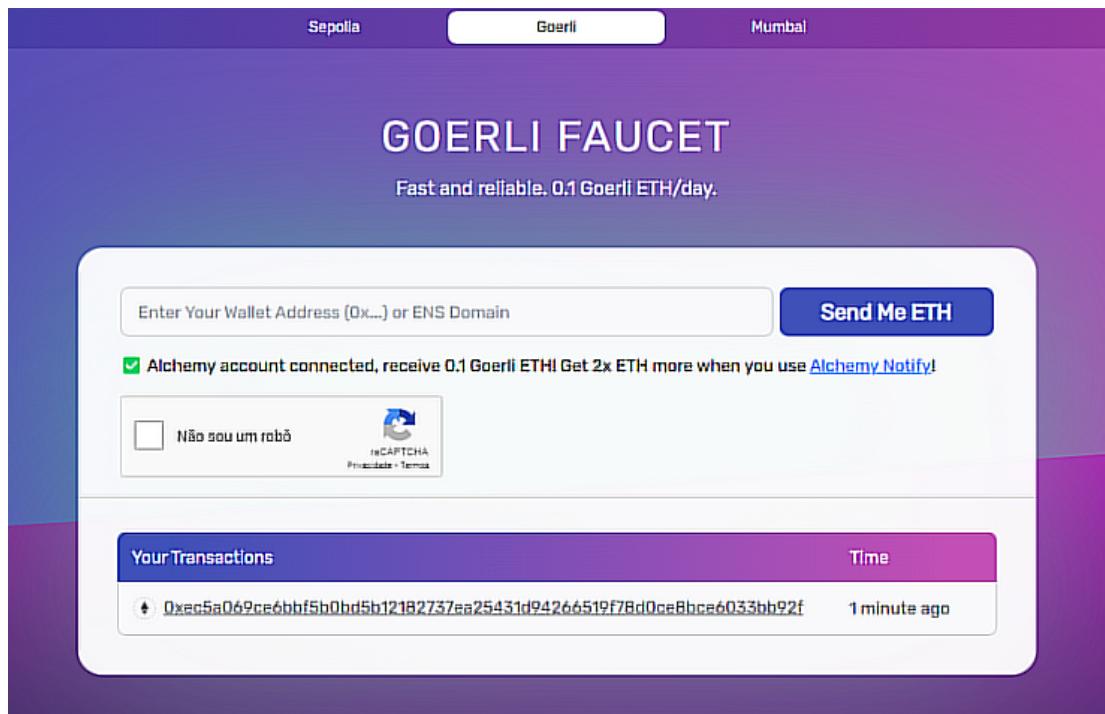


- Selecione a opção "**"Redes"**" no canto superior direito. Em seguida, selecione a rede "**"Goerli"**" que será utilizada, após clique em "**"Adicionar rede"**", como na imagem abaixo:





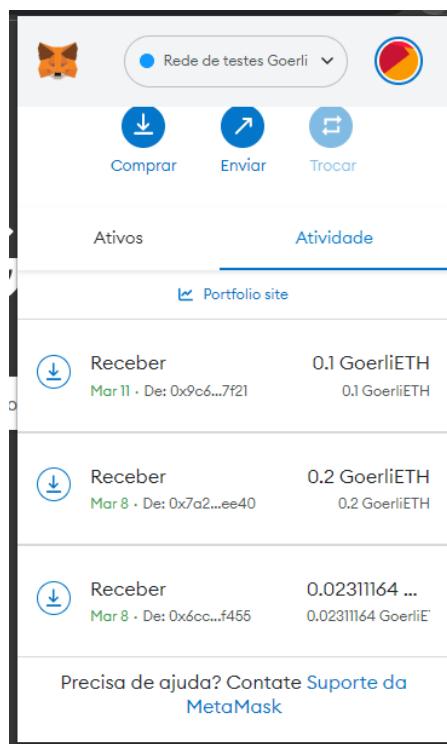
- Para conseguir ETH de teste, é necessário entrar no site <https://goerli-faucet.slock.it/> e criar uma conta, após o login, insira o endereço da carteira no campo "Enter your wallet address"



- Faça a autenticação do "Não sou um robô" e clique no botão "Send me ETH", caso a transação seja realizada com sucesso, aparecerá uma ilustração como esta a seguir.



- Depois de alguns segundos o ETH cai no saldo da carteira, lembrando que a mesma tem que estar na rede de teste do Goerli já cadastrada anteriormente. Verificando o recebimento em "Atividade"



COISAS QUE DERÃO CERTO

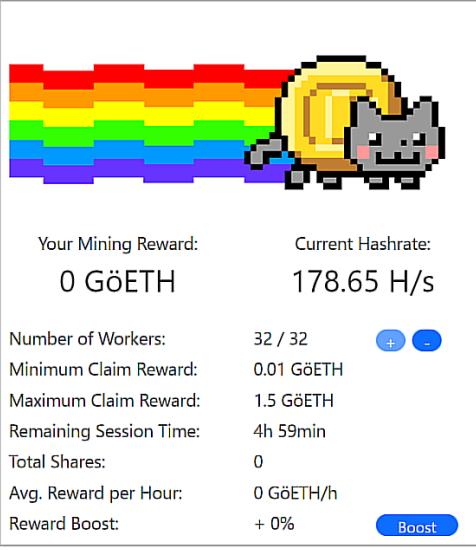
- É muito intuito e rápido o processo;
- A transação é realizada em segundos;
- A plataforma é simples;

COISAS QUE DERÃO ERRADO

- O site do Goerli Faucet apresentou instabilidade, não sendo possível seu acesso por alguns dias, o que dificultou a realização da transação para ter o ETH necessário do deploy.

FUTURAS MELHORIAS

- É importante deixar sites de precaução, caso algo dê errado, como a instabilidade, utilizamos o <https://goerli-faucet.pk910.de/> enquanto o oficial não voltava, mas a mineração era bem demorada. Aqui apresenta uma imagem de como era a mineração no site.



The screenshot shows a mining interface for the Goerli Faucet. At the top, there's a small image of a cat mining on a rainbow-colored block chain. Below it, the text "Your Mining Reward: 0 GöETH" and "Current Hashrate: 178.65 H/s" is displayed. On the right, a "Claim Mining Rewards" button is visible. Further down, there are sections for "Number of Workers: 32 / 32", "Minimum Claim Reward: 0.01 GöETH", "Maximum Claim Reward: 1.5 GöETH", "Remaining Session Time: 4h 59min", "Total Shares: 0", "Avg. Reward per Hour: 0 GöETH/h", and "Reward Boost: + 0%". A "Boost" button is located at the bottom right of this section. To the right of the main interface, a modal window titled "Claim Mining Rewards" provides details about the transaction: "Target Address: 0x9f80aC649f0244330aDB623D5D56c6b7db71bf3A", "Claimable Reward: 0.023 GöETH", and "Claimable until: 2023-03-09 04:44 (11h 44min)". It also includes a message: "Claim Transaction has been confirmed in block #8620487! TX: 0x91eac3c86993c550fee529df9d930f2832da7a152b2505b0bf63384e988e1545". Below this, there are links to "Star" (512), "Tweet", and "Post". A "Close" button is at the bottom right of the modal.



METAMASK DEPLOY

07 de março de 2023

CONEXÃO COM A WALLET E TRUFFLE

A conexão da wallet Metamask com Truffle, segue a configuração feita.

OBS: Para essa etapa espera-se que todos os outros passos apresentados acima já estejam realizados.

- Conecte a carteira com a rede Ethereum que você configurou no arquivo "truffle-config.js" do tutorial acima. Importe a conta que possui Ether para realizar as transações no Smart Contract.
- Verifique se a conta está conectada selecionando a rede configurada no Truffle. Execute o comando "truffle console".

CONEXÃO COM A WALLET E GANACHE

A conexão da wallet Metamask com Ganache, segue a configuração feita.

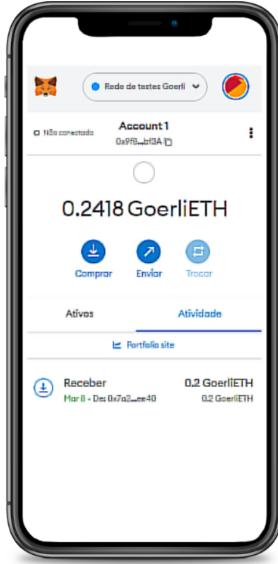
OBS: Para essa etapa espera-se que todos os outros passos apresentados acima já estejam realizados.

- A diferença entre os deploy's é que com o ganache, tem que conectar a Metamask à rede local, selecionando "Custom RPC" e inserindo as informações da rede.



Realize o deploy do contrato e verifique as transações no Metamask. Apresenta-se um exemplo de antes e depois das transações abaixo.

ANTES DO DEPLOY

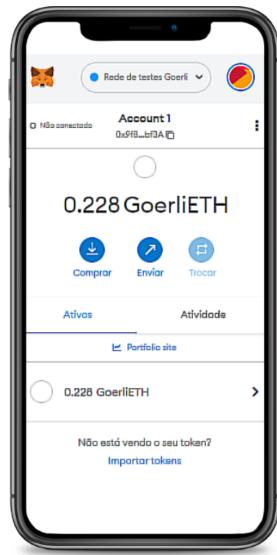


```
1_coover.js
=====
Deploying "cooverContract"
-----
> transaction hash: 0x2dd6d59062ccc233707b2c0b96122e7c5a927d764e3be2b0f5bb1c826351289b
> Blocks: 1
> contract address: 0x60cd81ae3fd026231C16F230981352443F18537
> block number: 8630778
> block timestamp: 1678456932
> account: 0x9f80ac640f0244330aDB623D5D56c6b7db71bf3A
> balance: 0.227956548622764424
> gas used: 801134 (0x396e)
> gas price: 17.279473078 gwei
> value sent: 0 ETH
> total cost: 0.013843173384870452 ETH
```

Valor cobrado para o deploy

-0,0138 GoerliETH

DEPOIS DO DEPLOY



Apresenta-se portanto as versões que foram utilizadas no projeto, na imagem a seguir:

```
C:\GitHub - Projetos\INTELI 2 ANO\MODULO 1\Projeto2\src\Deploy Smart Contract>truffle --version
Truffle v5.7.9 (core: 5.7.9)
Ganache v7.7.5
Solidity v0.5.16 (solc-js)
Node v16.14.2
Web3.js v1.8.2
```



TESTE DO CONTRATO

20 de março de 2023

TESTE DAS FUNÇÕES DO CONTRATO

Essa parte da documentação corresponde aos códigos localizados no:

Projeto2 --> Contratos --> Códigos para testes unitários da funções

1 - Função Constructor

A função apresentada abaixo é responsável por fazer o teste da função Constructor :

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0; // Define a versão do Solidity e a licença do contrato

import "remix_tests.sol"; // Importando o módulo de teste do Remix
import "contracts/cooverContract.sol"; // Importando o contrato cooverContract

contract cooverContractTest { // Definindo um novo contrato chamado cooverContractTest

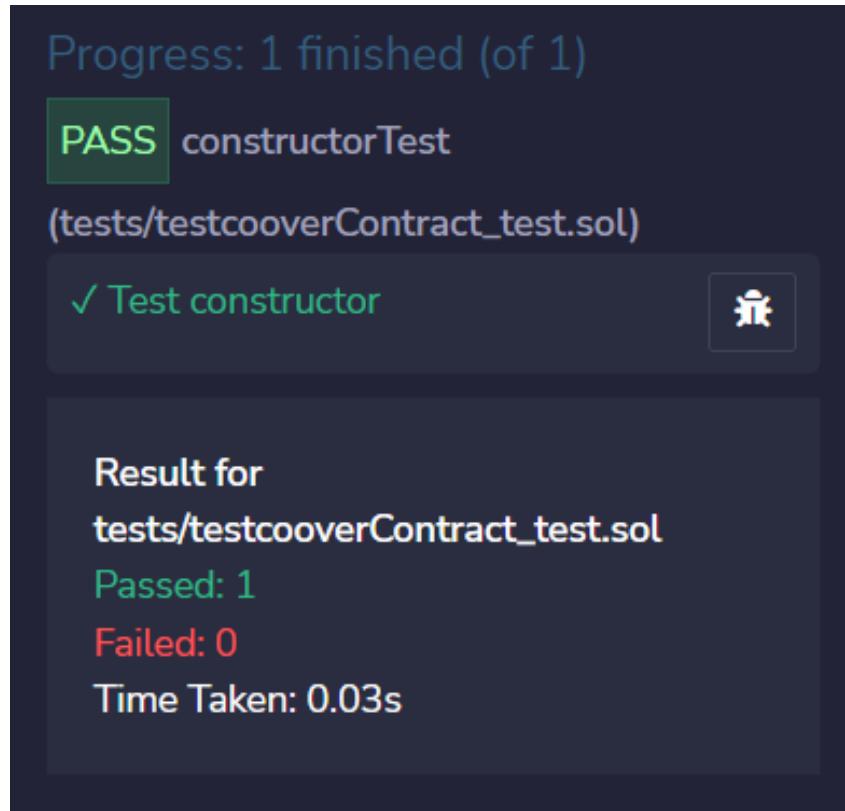
    cooverContract contrato; // Instância do contrato a ser testado

    function beforeAll() public { // Definindo uma função chamada beforeAll, que será executada antes de todos os testes
        address[] memory integrantes = new address[](2); // Declara um array de endereços chamado integrantes com 2 elementos
        uint[] memory imei = new uint[](2); // Declara um array de uint chamado imei com 2 elementos
        integrantes[0] = address(this); // Define o primeiro elemento do array como o endereço do contrato atual
        imei[0] = 123456789; // Define o primeiro elemento do array como 123456789
        integrantes[1] = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4; // Define o segundo elemento do array como o endereço
        imei[1] = 987654321; // Define o segundo elemento do array como 987654321
        contrato = new cooverContract(integrantes, imei); // Cria uma nova instância do contrato
    }

    function saldo_Contrato_Deve_Retornar_Zero() public { // Define uma função chamada saldo_Contrato_Deve_Retornar_Zero para testar a função saldo_Contrato do contrato cooverContract
        uint balance = contrato.saldo_Contrato(); // Chama a função saldo_Contrato() do contrato
        Assert.equal(balance, 0, "O saldo do contrato deve ser zero"); // Compara o valor retornado pela função saldo_Contrato com zero e exibe a mensagem "O saldo do contrato deve ser zero" se a comparação falhar
    }
}
```



- Pré-condição: A pré-condição para que o teste da função constructor seja um sucesso é que dentro do array do tipo "Integrantes" seja adicionado uma hash e dentro do array "Imei" sejam adicionados números que simulam um imei.
- Resultado esperado: Espera-se que a função funcione normalmente e permita a criação do contrato.



```
Progress: 1 finished (of 1)
PASS constructorTest
(tests/testcooverContract_test.sol)
✓ Test constructor
Result for
tests/testcooverContract_test.sol
Passed: 1
Failed: 0
Time Taken: 0.03s
```

- Pós condição: Após o teste a função funcionou como o esperado, possibilitando a criação do contrato.



TESTE DO CONTRATO

20 de março de 2023

2- Função saldo_Contrato()

A função apresentada abaixo é responsável por fazer o teste da função saldo_Contrato e verificar se na criação do contrato o saldo dentro dele é igual a 0

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0; // Define a versão do Solidity e a licença do contrato

import "remix_tests.sol"; // Importando o módulo de teste do Remix
import "contracts/cooverContract.sol"; // Importando o contrato cooverContract

contract cooverContractTest { // Definindo um novo contrato chamado cooverContractTest

    cooverContract contrato; // Instância do contrato a ser testado

    function beforeAll() public { // Definindo uma função chamada beforeAll, que será
        executada antes de todos os testes
        address[] memory integrantes = new address[](2); // Declara um array de endereços
        chamado integrantes com 2 elementos
        uint[] memory imei = new uint[](2); // Declara um array de uint chamado imei com 2
        elementos
        integrantes[0] = address(this); // Define o primeiro elemento do array como o
        endereço do contrato atual
        imei[0] = 123456789; // Define o primeiro elemento do array como 123456789
        integrantes[1] = 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4; // Define o segundo
        elemento do array como o endereço
        imei[1] = 987654321; // Define o segundo elemento do array como 987654321
        contrato = new cooverContract(integrantes, imei); // Cria uma nova instância do
        contrato
    }

    function saldo_Contrato_Deve_Retornar_Zero() public { // Define uma função chamada
        saldo_Contrato_Deve_Retornar_Zero para testar a função saldo_Contrato do contrato
        cooverContract
        uint balance = contrato.saldo_Contrato(); // Chama a função saldo_Contrato() do
        contrato
        Assert.equal(balance, 0, "O saldo do contrato deve ser zero"); // Compara o valor
        retornado pela função saldo_Contrato com zero e exibe a mensagem "O saldo do contrato deve
        ser zero" se a comparação falhar
    }
}
```



- Pré-condição: A pré-condição para que o teste da função saldo_Contrato seja um sucesso é que dentro do saldo do contrato o valor seja igual a 0, tendo certeza de que na criação do contrato, nenhuma conta seja feita de maneira equivocada
- Resultado esperado: Espera-se que a função retorne o valor do contrato igual a 0

```
Progress: 1 finished (of 1)

PASS cooverContractTest
(tests/testcooverContract_test.sol)
✓ Saldo contrato deve retornar zero

Result for
tests/testcooverContract_test.sol
Passed: 1
Failed: 0
Time Taken: 0.02s
```

- Pós condição: Após o teste, verificamos que o valor contido dentro do contrato é realmente igual a 0, confirmando nossas expectativas

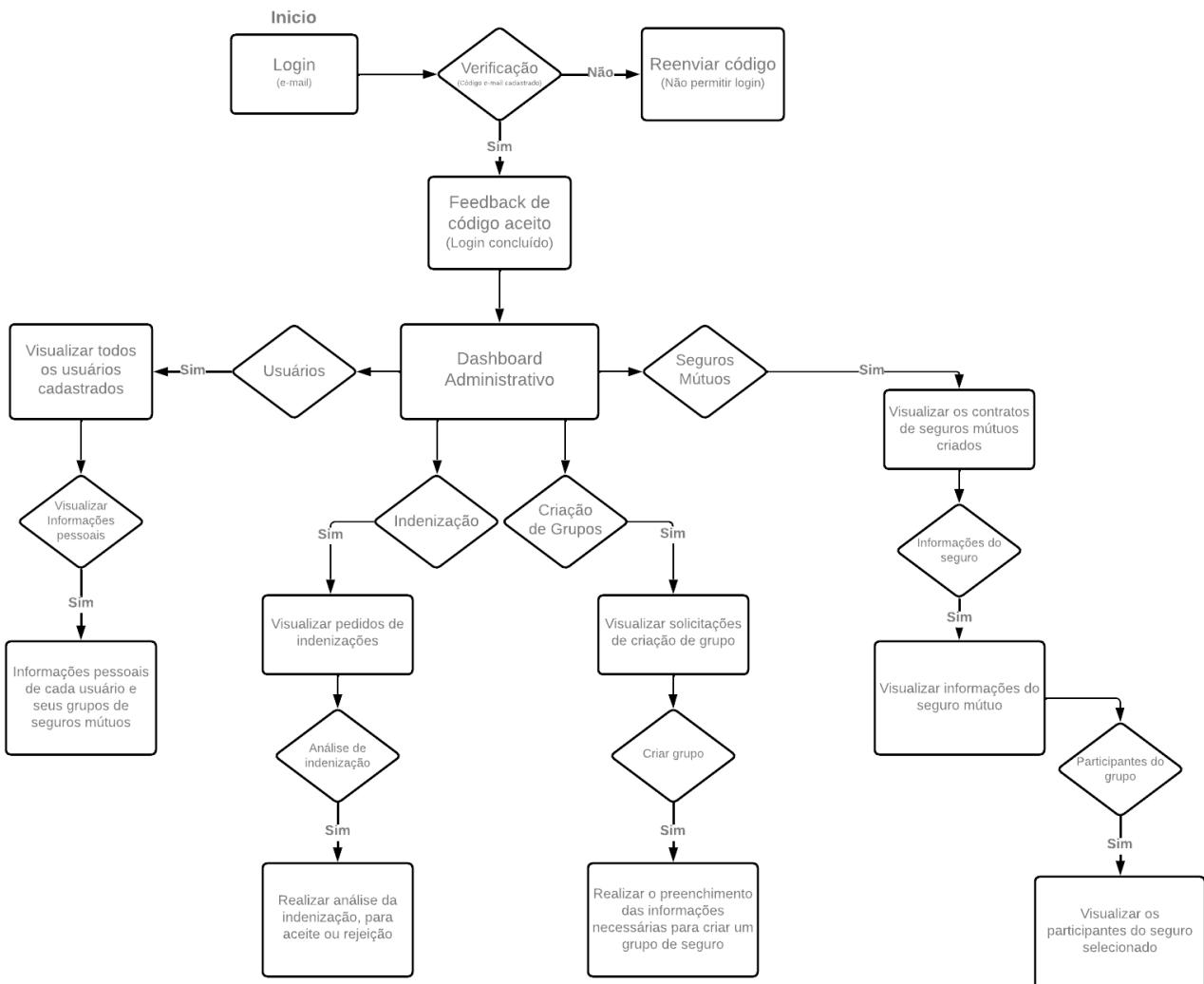


FLUXO DE CONTROLE

24 de março de 2023

Fluxo seguradora

Apresenta-se abaixo o fluxo de controle previsto para a plataforma da seguradora Coover. Exibe-se o caminho previsto para a solução, entre a comunicação das telas, desde a entrada do colaborador até a saída final da plataforma.



Fluxo Usuário

Apresenta-se abaixo o fluxo de controle previsto para a plataforma do usuário Coover. Exibe-se o caminho previsto para a solução, entre a comunicação das telas, desde a entrada do usuário até a saída final da plataforma. Além de constar a integração entre a metamask para conexão e transação.

