PROJETO 2 - Smart Contract de `Escritura de Imóveis`

Curso Blockchain Developer - Turma JUN2018

25 de Junho de 2018

Material produzido por bbchain.

Estrutura do projeto

1. Objetivo

Este projeto tem como objetivo desenvolver um Smart Contract para entender a estrutura básica de um contrato em Solidity. Vamos programar, compilar, fazer deploy e interagir com um Smart Contract capaz de registrar a transferência de propriedade de imóveis dentro de um ambiente virtual simulado da plataforma Ethereum.

---

2. Pré-Requisitos

Conceitos básicos de programação, Smart Contracts e plataforma Ethereum.

---

3. Materiais

- [Google Chrome] (https://www.google.com/chrome/)
- [Remix] (http://remix.ethereum.org)

4. Conceitos Abordados

Antes de entrarmos no desafio deste projeto, vamos explorar, separadamente, alguns conceitos básicos que iremos utilizar:

1. Ambiente de desenvolvimento Remix
2. Estrutura básica de um Smart Contract
3. Variável especial `msg.sender`
4. Expressão de controle `require`

4.1. Ambiente de desenvolvimento Remix

Remix é uma *Integrated Development Environment* (IDE) web para desenvolvimento de Smart Contracts em Solidity para o Ethereum.

Com essa ferramenta, nós podemos:

- Criar e editar arquivos Solidity
- Escrever Smart Contracts em Solidity
- Compilar Smart Contracts
- Fazer deploy de Smart Contracts em um ambiente simulado em Javascript (Javascript VM) ou em redes TestNet ou MainNet (com o auxílio do MetaMask)
- Interagir com Smart Contracts criados
- Salvar arquivos no Gist (GitHub)

Essa ferramenta é muito conveniente para o desenvolvimento e testes de Smart Contracts, pois proporciona um ambiente com `Accounts` com saldo em `ether` e simula uma rede Ethereum, tudo isso sem a necessidade de instalação ou configuração de softwares no computador.

> **Por que não levantar um nó Ethereum localmente para o desenvolvimento?**
> Levantar um nó Ethereum localmente tem algumas desvantagens: 1) a configuração e a interação com o nó é mais complicada; 2) exige que o nó fique minerando as transações

> localmente, consumindo desnecessariamente recursos computacionais. No projeto 6 iremos criar uma rede privada local Ethereum.

4.1.1. Explore o ambiente Remix

Experimente entrar em http://remix.ethereum.org e:

4.1.2.1. Criar um arquivo Solidity

1. [ ] Criar um arquivo novo

4.1.2.2. Escrever Smart Contracts em Solidity

2. [ ] Escreva um Smart Contract simples (não se preocupe em entender o conteúdo, veremos com detalhes mais adiante)

```
pragma solidity ^0.4.18;

contract ContratoQualquer {

        // VARIÁVEL
        string public valor;

        // FUNÇÃO
        function alterarValor(string novoValor) public {
                valor = novoValor;
        }
}
```

4.1.2.3. Compilar Smart Contracts

3. [ ] Compile clicando no botão `Start to Compile` na aba `Compile`
4. [ ] Veja os detalhes da compilação clicando no botão `Details` na aba `Compile`

4.1.2.4. Fazer deploy do Smart Contract

5. [ ] Na **aba** `Run` certifique-se que `Javascript VM` está selecionado no campo `Enviroment`
6. [ ] Verifique as `Accounts` e os saldos disponíveis, na aba `Run`.
7. [ ] Faça o deploy do Smart Contract clicando no o botão `Deploy`.

4.1.2.5. Interagir com o Smart Contract

8. [ ] Verifique o endereço do Smart Contract criado.
9. [ ] Clique no botão da variável `valor`.
10. [ ] Escreva `"Curso Blockchain Developer"` no parâmetro da função `alterarValor`.
11. [ ] Clique no botão da função `alterarValor`.
12. [ ] Clique no botão da variável `valor`.

4.1.2.6. Salvar arquivos no Gist (GitHub)

13. [ ] Clique na aba `Settings`
14. [ ] Clique no link https://github.com/settings/tokens
15. [ ] No GitHub, clique em `Generate new token`
16. [ ] De um nome. Ex.: `acesso-remix`
17. [ ] Marque o checkbox **`gist`** : *`Create gists`*
18. [ ] Clique no botão `Generate token`
19. [ ] Copie o token gerado
20. [ ] No Remix, cole o token no campo `Gist Access Token` e clique no botão `Save`.
21. [ ] Clique no primeiro ícone () do GitHub da esquerda para direita, no menu principal.
22. [ ] Confirme a criação do gist

4.2. Estrutura básica de um Smart Contract

Vamos entender a estrutura mínima de um Smart Contract em Solidity. Basicamente, um Smart Contract precisa ter quatro partes:

1. Versão do compilador
2. Bloco `contract`
3. Variáveis
4. Funções

4.2.1. Versão do compilador

Para evitar que o Smart Contract seja compilado por versões incompatíveis com o contrato, incluímos no cabeçalho uma anotação com a versão do compilador que o Smart Contract deve ser compilado.

Exemplo:

```
pragma solidity ^0.4.18;
```

Veja mais datalhes em: http://solidity.readthedocs.io/en/v0.4.24/layout-of-source-files.html

4.2.2. Bloco `contract`

Contratos em Solidity são similares às classes em linguagens orientadas a objetos.

Exemplo (sintaxe):

```
pragma solidity ^0.4.18;

contract FigurinhaDaCopa {

}
```

Veja mais detalhes em: http://solidity.readthedocs.io/en/v0.4.24/structure-of-a-contract.html

4.2.3. Variáveis

Variáveis representam o estado atual de um Smart Contract. Elas armazenam informações permanentemente na Blockchain e só podem ser alteradas por meio de funções do próprio Smart Contract.

As variáveis no Solidity são fortemente tipadas. Os tipos de variáveis mais utilizados são:

- **`uint`**: valores inteiros numéricos.
- **`string`**: valores em texto.
- **`address`**: endereço na rede Ethereum.
- **`bool`**: valores booleanos, admitindo TRUE ou FALSE como possíveis valores.
- **`array`**: objeto que armazena uma lista de uma determinada variável. Neste exemplo, `uint[] notas;`, a variável `notas` pode armazenar uma lista de inteiros.

- `mapping`: objeto que armazena pares de variáveis no formato chave-valor. Neste exemplo, `mapping(address => uint) saldo;`, o mapa `saldo` pode armazenar uma lista de valores com a chave em formato de endereço e com o valor no formato inteiro (ex.: `saldo['0xdCad3a6d3569DF655070DEd06cb7A1b2Ccd1D3AF'] = 100;`).

Confira os detalhes e as propriedades de cada tipo de variável em: http://solidity.readthedocs.io/en/v0.4.24/types.html#types

As variáveis podem possuir diferentes níveis de visibilidade. Para o propósito deste laboratório, nós usaremos apenas `public`. Para mais detalhes, veja: http://solidity.readthedocs.io/en/v0.4.24/contracts.html#visibility-and-getters

Exemplo (sintaxe):

```
pragma solidity ^0.4.18;

contract FigurinhaDaCopa {

        // VARIÁVEIS
        string public nomeJogador = "Neymar Jr.";
        uint public numeroCamisa = 10;
        string public selecao = "Brasil";
        address public dono;

}
```

Considere que por padrão, as variáveis são públicas. Veja as opções para alterar o modo de visibilidade em: http://solidity.readthedocs.io/en/v0.4.24/contracts.html#visibility-and-getters

4.2.4. Funções

Funções são trechos de código executáveis dentro de um contrato. Assim como na maioria das linguagens de programação, as funções podem receber variáveis por parâmetro e retornar valores.

Exemplo:

```
pragma solidity ^0.4.18;

contract FigurinhaDaCopa {

        // VARIÁVEIS
```

```
        string public nomeJogador = "Neymar Jr.";
        uint public numeroCamisa = 10;
        string public selecao = "Brasil";
        address public dono;

        // FUNÇÕES
        function alterarDono(address novoDono) public returns(address) {
                dono = novoDono;
                return novoDono;
        }
}
```

4.2.4.1. Funções construtora

Há um tipo especial de função que chamamos de **função construtora**. Essa função é executada apenas uma vez, quando o contrato é criado na rede. Ela tem as mesmas propriedades de uma função normal. O que muda é a sintaxe, em vez de `function` nós usamos `constructor` e não definimos um nome para a função.

Exemplo (sintaxe):

```
pragma solidity ^0.4.18;

contract FigurinhaDaCopa {

        // VARIÁVEIS
        string public nomeJogador = "Neymar Jr.";
        uint public numeroCamisa = 10;
        string public selecao = "Brasil";
        address public dono;

        // FUNÇÕES
        function alterarDono(address novoDono) public returns(address) {
                dono = novoDono;
                return novoDono;
        }
        constructor(string _nomeJogador, uint _numeroCamisa, string _selecao) pu
        nomeJogador = _nomeJogador;
        numeroCamisa = _numeroCamisa;
        selecao = _selecao;
        }
}
```

Considere que por padrão, as funções são públicas (qualquer Account dentro da rede pode chamar a função). Veja as opções para alterar o modo de visibilidade de uma função em: http://solidity.readthedocs.io/en/v0.4.24/contracts.html#visibility-and-getters

4.3. Variável especial `msg.sender`

`msg.sender` contém o endereço da Account de quem está chamando a função ou criando o contrato.

Exemplo (sintaxe):

```
pragma solidity ^0.4.18;

contract FigurinhaDaCopa {

        // VARIÁVEIS
        string public nomeJogador = "Neymar Jr.";
        uint public numeroCamisa = 10;
        string public selecao = "Brasil";
        address public dono;

        // FUNÇÕES
        function alterarDono(address novoDono) public returns(address) {
                dono = novoDono;
                return novoDono;
        }
        constructor(string _nomeJogador, uint _numeroCamisa, string _selecao) pu
        nomeJogador = _nomeJogador;
        numeroCamisa = _numeroCamisa;
        selecao = _selecao;
        dono = msg.sender; // ENDREÇO DE QUEM ESTÁ CRIANDO O CONTRATO
        }
}
```

Veja mais detalhes em: http://solidity.readthedocs.io/en/v0.4.24/units-and-global-variables.html

4.4. Expressão de controle `require`

`require` é usado para verificar se uma determinada condição é satisfeita dentro de uma função, caso não seja satisfeita, uma excessão é emitida.

Exemplo (sintaxe):

```
pragma solidity ^0.4.18;

contract FigurinhaDaCopa {

        // VARIÁVEIS
        string public nomeJogador = "Neymar Jr.";
        uint public numeroCamisa = 10;
        string public selecao = "Brasil";
        address public dono;

        // FUNÇÕES
        function alterarDono(address novoDono) public returns(address) {
                // ENDREÇO DE QUEM ESTÁ CHAMANDO A FUNÇÃO TEM QUE SER IGUAL AO E
                require(dono == msg.sender, "Somente o dono pode chamar esta fun
                dono = novoDono;
                return novoDono;
        }
        constructor(string _nomeJogador, uint _numeroCamisa, string _selecao) pu
        nomeJogador = _nomeJogador;
        numeroCamisa = _numeroCamisa;
        selecao = _selecao;
        dono = msg.sender; // ENDEREÇO DE QUEM ESTÁ CRIANDO O CONTRATO
        }
}
```

Veja mais detalhes em: https://solidity.readthedocs.io/en/v0.4.24/control-structures.html?highlight=require

5. Projeto: Smart Contract de `Escritura de Imóveis`

Construir um Smart Contract simplificado para registrar a transferência de propriedade de imóveis.

Cada instância de contrato representa o atual proprietário de um determinado imóvel. É necessário definir no momento da criação do contrato duas variáveis: 1) o **número de matrícula do imóvel**, que representa o imóvel real no cartório de registro de imóveis e; 2) o *endereço* (Account) do **proprietário** atual do imóvel.

Regras: Somente o proprietário atual do imóvel pode transferir a propriedade do imóvel para outro *endereço*, por meio da função **transferir**.

Deploy e interação com o contrato: Após escrever o contrato, crie uma escritura (instância do contrato) e transfira a propriedade do imóvel para outro *endereço*. Tente transferir novamente, simulando um *double spending*.

5.1. Solução: Escritura de Imóveis

ATENÇÃO!!! As instruções abaixo são apenas para referência. É fortemente recomendado resolver o desafio sem esse auxílio.

5.1.1. Crie um arquivo Solidity

1. ☐ Abra o Google Chrome e entre em (http://remix.ethereum.org).
2. ☐ Crie um novo arquivo `EscrituraImovel.sol`.

5.1.2. Programe um Smart Contract

3. ☐ Digite o seguinte código Solidity:

```
pragma solidity ^0.4.18;

contract EscrituraImovel {

    //** VARIÁVEIS **//

    // número da matrícula do imóvel (após definido, não pode mais ser alterado)
    string public numeroMatricula;
    // atual proprietário do imóvel
    address public proprietario;


    //** FUNÇÕES **//

    // Função construtura disparada somente uma vez na criação do contrato
    constructor (string _numeroMatricula, address _proprietario) public {
```

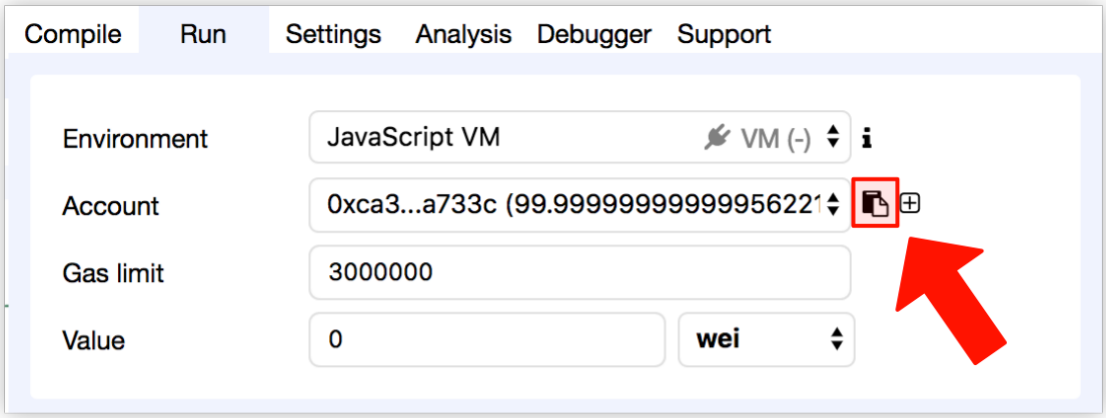
```
        numeroMatricula = _numeroMatricula;
        proprietario = _proprietario;
    }
    // Transferência de um proprietário para outro
    function transferir(address novoProprietario) public {
        require(msg.sender == proprietario);
        proprietario = novoProprietario;
    }
}
```

5.1.3. Compile o Smart Contract

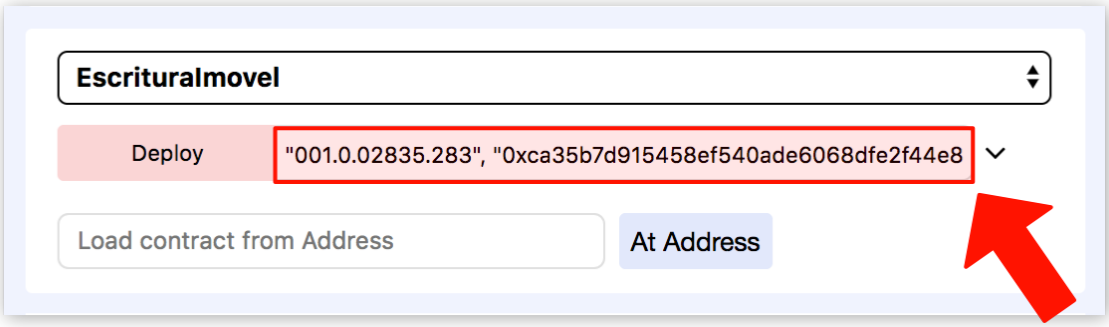
4. ☐ Certifique-se que o `Auto compile` está selecionado na aba `Compile`

5.1.4. Faça o deploy do Smart Contract

5. ☐ Certifique-se que o `Environment` está setado com `Javascript VM` na aba `Run`
6. ☐ Selecione a primeira `Account` disponível na aba `Run`
7. ☐ Clique no botão indicado na figura para copiar o endereço da `Account` selecionada



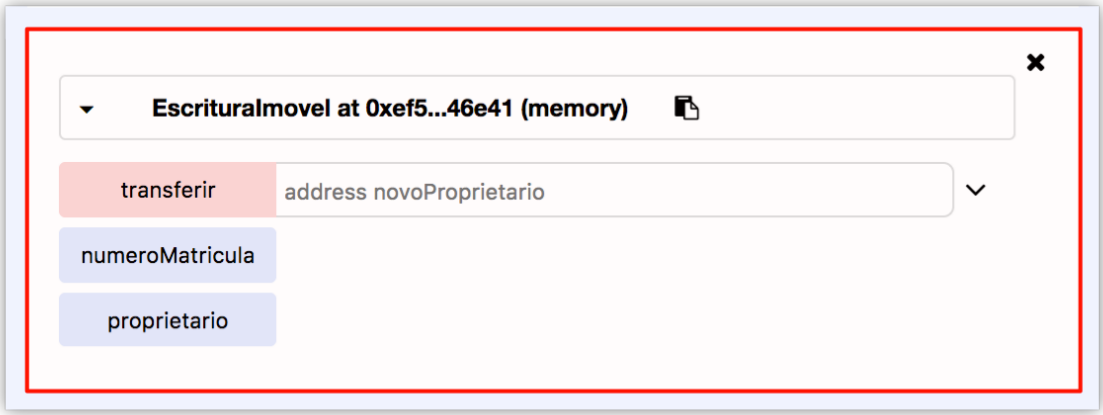
8. ☐ No campo de parâmetros do construtor, indicado na figura, insira o número de matrícula `"001.0.02835.283"` e o endereço da primeira `Account` (use Ctrl+V para colar o endereço).



9. ☐ Clique no botão `Deploy` para criar a instância do contrato na blockchain

5.1.6. Interaja com o Smart Contract

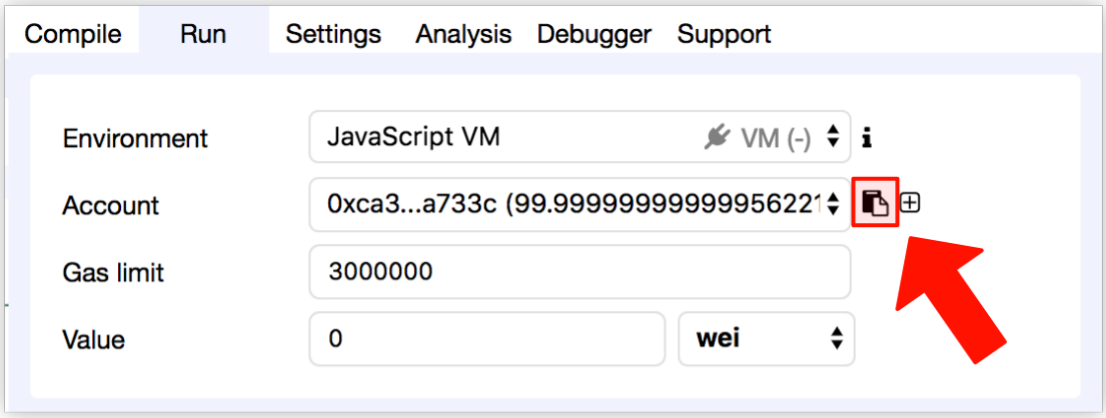
11. [ ] Confira a instância do Smart Contract criado



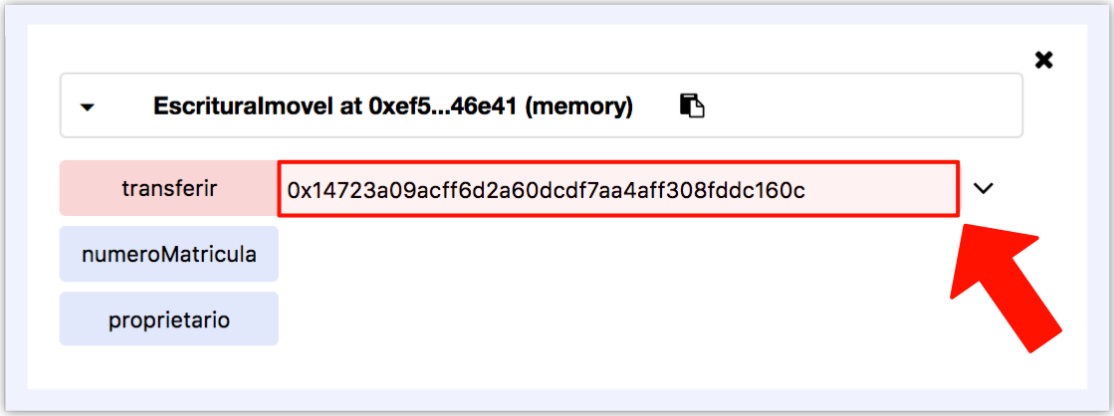
12. [ ] Clique em `numeroMatricula` para consultar o valor da variável
13. [ ] Clique em `proprietario` para consultar o valor da variável (deve ser o mesmo endereço da `Account` selecionada)

5.1.6.1. Transfira a propriedade para outro endereço (Account)

14. [ ] Selecione a segunda `Account`
15. [ ] Clique no botão indicado na figura para copiar o endereço da `Account` selecionada



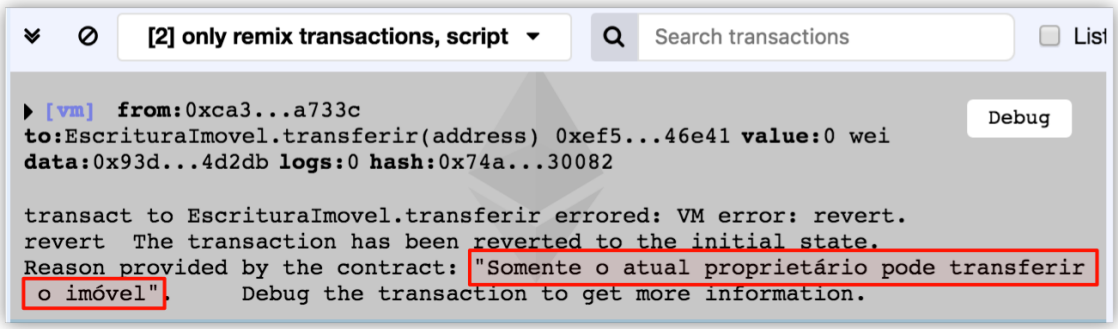
16. [ ] Selecione a primeira `Account` novamente (atual proprietária do imóvel)
17. [ ] Cole (Ctrl+V) o endereço da segunda `Account` no parâmetro `novoProprietario` da função `transferir`



18. [ ] Clique em `transferir` para executar a função na blockchain
19. [ ] Clique em `proprietario` para consultar o novo valor da variável (deve ser o endereço da segunda `Account`)

5.1.6.2. Simule um *double spending* (tente transferir duas vezes o mesmo imóvel)

20. [ ] Ainda com a primeira `Account` selecionada (antiga proprietária), tente transferir novamente para qualquer endereço, simulando um *doble spending*
21. [ ] Confira a exceção emitida no log



5.2. Desafios extra (opcional)

5.2.1. Testemunhas

Inclua variáveis e lógica que permita que a transferência ocorra após a confirmação de duas testemunhas diferentes.

5.2.2. Cartório

Considere que o endereço de quem está criando o contrato é do cartório de registro de imóveis. Crie uma função que permita o cartório alterar o número da matrícula do imóvel.

5.2.3. Data da transferência

Inclua uma variável para registrar a data da última transferência. Consulte: http://solidity.readthedocs.io/en/v0.4.21/units-and-global-variables.html