Implementando Tokens no Ethereum

Matheus Protásio Martins[1]
[1]Departamento Acadêmico de Computação (DACOM)
Universidade Tecnológica Federal do Paraná (UTFPR)

Abstract

This article aims to simply present what Ethereum is and its tools, what smart contracts are, and how to implement a simple contract using only the Remix IDE.

Resumo

Esse artigo tem como objetivo apresentar de forma simples oque é o Etherium e suas ferramentas, oque são contratos inteligentes e como implementar um contrato simples utilizando apenas o Remix IDE.

1 Introdução

A tecnologia *blockchain* tem revolucionado diversos setores, desde o financeiro até o de logística e saúde, devido à sua capacidade de registrar transações de forma descentralizada, segura e imutável. Em sua essência, uma *blockchain* é um banco de dados distribuído que funciona como um livro-razão público, onde todas as transações são registradas e validadas por uma rede de participantes sem a necessidade de intermediários confiáveis. Essa estrutura garante maior transparência e resiliência contra fraudes e falhas de segurança (Antonopoulos & Wood 2018).

2 Ethereum

Diferente do Bitcoin, que é focado em transferências financeiras, o Ethereum foi projetado como uma plataforma de uso geral para a criação de aplicativos descentralizados (DApps). Lançado em 2015, o Ethereum se destaca pela introdução de Contratos Inteligentes (*Smart Contracts*), que são programas autônomos executados em sua *blockchain*. Esses contratos permitem a automatização de processos com base em regras predefinidas, como transferências financeiras, registro de propriedade e execução de acordos comerciais (Imran 2018).

2.1 Ethereum Virtual Machine

No coração do Ethereum está a *Ethereum Virtual Machine* (EVM), uma máquina virtual que executa o código dos contratos inteligentes e garante que todos os nós da rede Ethereum possam executar as mesmas operações de maneira idêntica. A EVM é responsável por processar e validar todas as transações, garantindo que as condições de um contrato inteligente sejam executadas de forma confiável e segura, independentemente do participante ou local onde ele esteja na rede (Antonopoulos & Wood 2018).

Corresponding author(s):

2.2 Redes

O Ethereum opera em diferentes tipos de redes, que servem para propósitos distintos, principalmente no desenvolvimento e testes de aplicativos e contratos inteligentes.

2.2.1 Rede Principal (Mainnet)

A rede principal do Ethereum é onde ocorrem as transações reais, com valor monetário. Todas as transações e contratos inteligentes na Mainnet são registradas de forma permanente na *blockchain*, e qualquer erro ou falha pode ter consequências financeiras. É a rede mais segura e confiável, onde os usuários podem interagir com contratos inteligentes e DApps de maneira ativa.

2.2.2 Redes de Teste

As redes de teste (testnets) são cópias do Ethereum Mainnet, mas com *tokens* sem valor real, permitindo que desenvolvedores testem contratos inteligentes sem risco financeiro. Existem várias redes de teste, cada uma com suas próprias características:

- Ropsten: Uma rede de teste pública que simula a *Mainnet*, utilizada para testes mais realistas com a mesma arquitetura de consenso da *Mainnet*.
- Rinkeby: Uma rede de teste baseada em um mecanismo de consenso diferente (*Proof of Authority*), mais rápida e estável que Ropsten.
- Goerli: Outra rede de teste que usa *Proof of Authority*, mas com diferentes características de implementação.

Essas redes são cruciais para o desenvolvimento de contratos inteligentes e DApps, pois permitem que os desenvolvedores testem suas aplicações em um ambiente seguro e controlado antes de lançá-las na rede principal (Imran 2018).

3 Contratos Inteligentes

O conceito de Contratos Inteligentes foi introduzido por Nick Szabo em 1997, como uma forma de automatizar a execução de acordos através de software, sem a necessidade de intermediários. Szabo (1997) descreveu contratos inteligentes como "contratos autoexecutáveis, onde os termos do acordo são escritos diretamente em código de computador". Eles são programados para executar automaticamente as condições definidas no contrato quando certos critérios são atendidos, utilizando a tecnologia *blockchain* para garantir a segurança, transparência e imutabilidade.

3.1 Uso e Utilidade

A utilidade dos contratos inteligentes é vasta e se aplica a diversas áreas, como:

- Automação de Processos: Contratos inteligentes podem automatizar processos comerciais e financeiros, como a transferência de fundos, a distribuição de dividendos e a execução de ordens de pagamento, sem a necessidade de intermediários.
- Transparência e Segurança: Como os contratos são executados em uma blockchain, suas condições e resultados são imutáveis e visíveis para todos os participantes, garantindo maior transparência e segurança.
- Eficiência e Redução de Custos: A eliminação de intermediários reduz custos operacionais e aumenta a eficiência. Por exemplo, em plataformas de Finanças Descentralizadas (DeFi), contratos inteligentes são usados para gerenciar empréstimos e transações de maneira automatizada e descentralizada.

Contratos inteligentes têm sido particularmente úteis em sistemas financeiros descentralizados, como o Ethereum, onde contratos são utilizados para criar *tokens*,como ERC-20, e governar transações em plataformas sem depender de uma autoridade central (Szabo 1997).

3.2 Solidity e Implementação

Solidity é a principal linguagem de programação utilizada para escrever contratos inteligentes na *blockchain* Ethereum. Desenvolvida por Gavin Wood e outros desenvolvedores da Ethereum, Solidity foi projetada especificamente para ser utilizada na *Ethereum Virtual Machine* (EVM), permitindo que desenvolvedores criem contratos que interagem diretamente com a *blockchain* (Antonopoulos & Wood 2018).

Solidity é uma linguagem de alto nível, com sintaxe inspirada em JavaScript, Python e C++. Ela permite a criação de contratos que podem armazenar e transferir valores digitais (como *tokens*), executar funções de governança, ou automatizar transações entre as partes. Solidity possui várias funcionalidades avançadas, incluindo:

- Modificadores de função: Permitem adicionar regras de acesso e condições específicas antes que uma função seja executada.
- Eventos: Facilitam a comunicação entre contratos e aplicativos externos.
- Interfaces de contrato: Definem funções externas que outros contratos ou sistemas podem interagir.

4 Tokenização

Tokens são representações digitais de ativos, direitos ou unidades de valor, criados e gerenciados em uma *blockchain*. Diferente de moedas tradicionais, como o dólar ou euro,*tokens* operam em redes descentralizadas, utilizando contratos inteligentes para definir suas características e regras de uso (Antonopoulos & Wood 2018).

Existem dois tipos principais de tokens:

- Tokens Fungíveis: São intercambiáveis e divisíveis, como moedas. Exemplos incluem tokens no padrão ERC-20.
- Tokens Não Fungíveis (NFTs): Representam ativos únicos, como obras de arte digitais ou itens colecionáveis. Seguem padrões como ERC-721 ou ERC-1155.

Esses *tokens* permitem que qualquer ativo, físico ou digital, seja representado em uma *blockchain*, tornando a transferência e rastreabilidade mais eficientes e seguras.

5 Ferramentas de Desenvolvimento

O Remix IDE, Figura 1, é um ambiente de desenvolvimento integrado baseado em navegador, amplamente utilizado para criar, testar e implantar contratos inteligentes na *blockchain* Ethereum. Por ser acessível diretamente através de navegadores modernos, como Chrome e Firefox, o Remix elimina a necessidade de instalações adicionais, o que o torna uma ferramenta prática e acessível para desenvolvedores, especialmente para aqueles que estão começando a trabalhar com contratos inteligentes (Antonopoulos & Wood 2018).

Uma das principais vantagens do Remix IDE é seu suporte completo à Solidity, a linguagem de programação mais utilizada para contratos inteligentes no Ethereum. Ele é compatível com as versões mais recentes de Solidity, permitindo que desenvolvedores escrevam, compilem e depurem contratos de forma eficiente. Além disso, o Remix oferece uma interface intuitiva, dividida em módulos como editor de código, painel de compilação, interface de *deploy* e ferramentas de depuração. Essa organização facilita a navegação e o uso da ferramenta, mesmo para desenvol-

vedores com pouca experiência (Ethereum Foundation 2024).

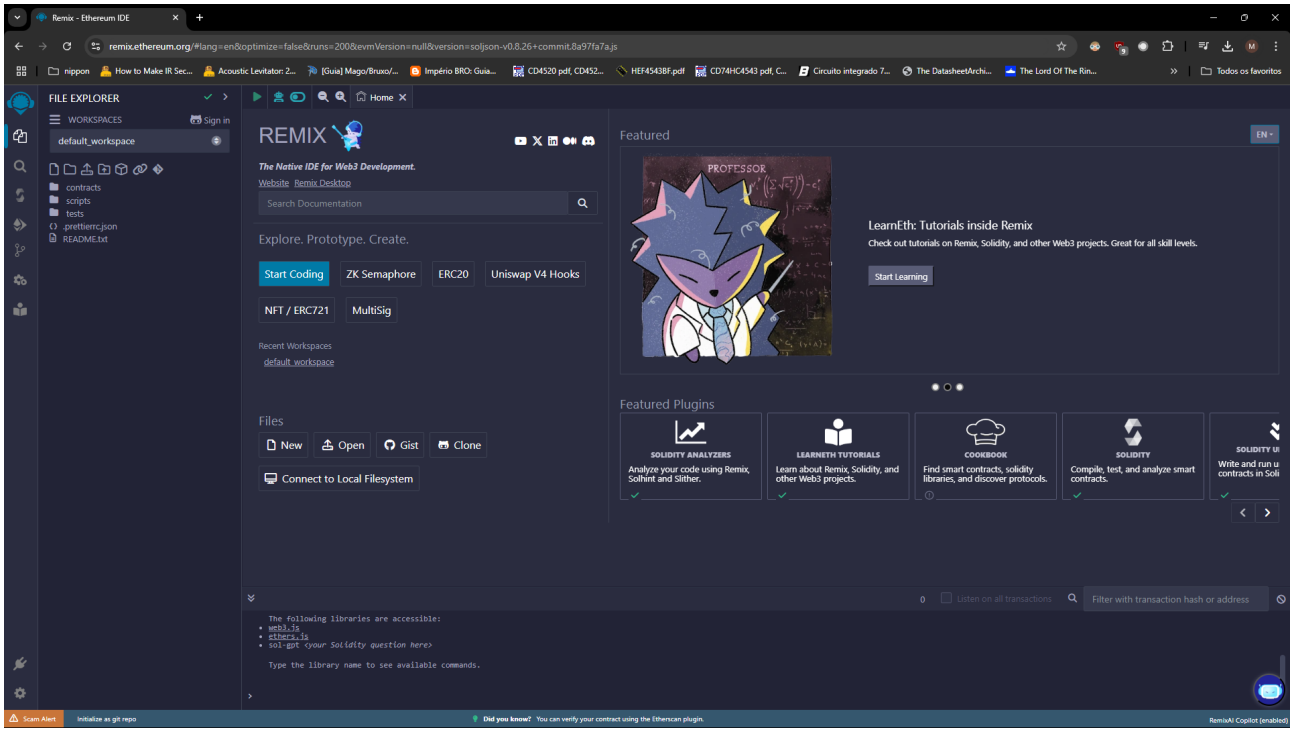

Figura 1: Página inicial do Remix IDE

O Remix também permite a simulação de uma *blockchain* local diretamente no ambiente de desenvolvimento. Isso possibilita o teste de contratos inteligentes em um ambiente seguro e controlado, sem a necessidade de conectá-los a uma rede pública. Essa funcionalidade é crucial para validar o comportamento do contrato e garantir que ele funcione corretamente antes de ser implantado em redes como a Mainnet ou redes de teste, como Ropsten e Goerli. Quando conectado a essas redes Ethereum reais, por meio de carteiras como MetaMask, o Remix também possibilita a interação e implantação de contratos em ambientes descentralizados (Antonopoulos & Wood 2018; Ethereum Foundation 2024).

Além disso, o Remix inclui um depurador avançado que permite inspecionar a execução dos contratos linha por linha. Essa ferramenta é fundamental para identificar e corrigir erros ou problemas no código, garantindo a qualidade e a segurança dos contratos inteligentes. Combinando simplicidade e funcionalidades avançadas, o Remix se tornou uma das ferramentas mais populares para o desenvolvimento de aplicativos descentralizados (DApps) no Ethereum.

6 Especificação e Desenvolvimento do Projeto

Dentro do Remix IDE, foi criado um novo workspace, para poder trabalhar em um ambiente limpo. Foi criada uma nova pasta, chamada TokenProject, e dentro dela, um arquivo MyToken.sol, que conterá o código do do token.

O código utilizado para este contrato pode ser observado no Código 1.

```
1  // SPDX-License-Identifier: MIT
2  // Licença SPDX que indica que o contrato é de código aberto e utiliza a licença
       MIT.
3
4  pragma solidity ^0.8.0;
5  // Define a versão do compilador Solidity necessária para este contrato (>=
       0.8.0).
6
7  contract MyERC20Token {
8      // Mapeamento para armazenar os saldos de tokens de cada endereço.
9      mapping(address => uint256) _balances;
10
11     // Mapeamento para definir permissões de gasto entre endereços.
12     // Permite que um endereço (gastador) mova tokens de outro (proprietário).
13     mapping(address => mapping(address => uint256)) _allowed;
14
15     // Nome do token, que será visível em carteiras como MetaMask.
16     string public name = "My ERC20 Token";
17
18     // Símbolo do token, geralmente usado como abreviação.
19     string public symbol = "MET";
20
21     // Quantidade de casas decimais que o token suporta (0 significa que não há
           frações).
22     uint8 public decimals = 0;
23
24     // Suprimento total de tokens criados no contrato.
25     uint256 private _totalSupply;
26
27     // Eventos que serão emitidos durante transferências e aprovações.
28     event Transfer(address indexed _from, address indexed _to, uint256 _value);
29     event Approval(address indexed _owner, address indexed _spender, uint256
           _value);
30
31     // Construtor que inicializa o contrato e atribui o suprimento total ao
           criador.
32     constructor(uint256 total) {
33         _totalSupply = total; // Define o total de tokens criados.
34         _balances[msg.sender] = _totalSupply; // Atribui todos os tokens ao
               endereço do criador do contrato.
35     }
36
37     // Retorna o suprimento total de tokens em circulação.
38     function totalSupply() public view returns (uint) {
39         // Exclui tokens atribuídos ao endereço zero (que são considerados
               queimados).
40         return _totalSupply - _balances[address(0)];
41     }
42
43     // Consulta o saldo de um endereço específico.
44     function balanceOf(address _owner) public view returns (uint balance) {
45         return _balances[_owner]; // Retorna o saldo do endereço fornecido.
46     }
47
48     // Verifica a quantidade de tokens que um gastador pode gastar em nome de um
           proprietário.
49     function allowance(address _owner, address _spender) public view returns (
           uint remaining) {
50         return _allowed[_owner][_spender]; // Retorna a permissão registrada no
               mapeamento _allowed.
```

```
51      }
52
53      // Transfere tokens do chamador (msg.sender) para outro endereço.
54      function transfer(address _to, uint256 _value) public returns (bool success)
            {
55          // Verifica se o remetente tem saldo suficiente.
56          require(_balances[msg.sender] >= _value, "value exceeds sender's balance
                ");
57
58          // Atualiza os saldos do remetente e do destinatário.
59          _balances[msg.sender] -= _value;
60          _balances[_to] += _value;
61
62          // Emite o evento Transfer para registrar a transação.
63          emit Transfer(msg.sender, _to, _value);
64          return true;
65      }
66
67      // Aprova um endereço para gastar tokens em nome do chamador.
68      function approve(address _spender, uint256 _value) public returns (bool
            success) {
69          // Atualiza a permissão no mapeamento _allowed.
70          _allowed[msg.sender][_spender] = _value;
71
72          // Emite o evento Approval para registrar a permissão.
73          emit Approval(msg.sender, _spender, _value);
74          return true;
75      }
76
77      // Transfere tokens de um endereço para outro, utilizando a permissão
            concedida anteriormente.
78      function transferfrom(address _from, address _to, uint256 _value) public
            returns (bool success) {
79          // Verifica se o remetente tem saldo suficiente.
80          require(_value <= _balances[_from], "Not enough balance");
81
82          // Verifica se o chamador tem permissão para gastar o valor especificado
                .
83          require(_value <= _allowed[_from][msg.sender], "Not enough allowance");
84
85          // Atualiza os saldos do remetente e do destinatário.
86          _balances[_from] -= _value;
87          _balances[_to] += _value;
88
89          // Reduz a permissão disponível no mapeamento _allowed.
90          _allowed[_from][msg.sender] -= _value;
91
92          // Emite o evento Transfer para registrar a transação.
93          emit Transfer(_from, _to, _value);
94          return true;
95      }
96  }
```

Código 1: Código do token

Agora para compilar, na aba Solidity Compiler, é necessário escolher a versão compatível com a escolhido no início do contrato, no caso 0.8.0, com pode ser visto na Figura 2, e também a versão da EVM utilizada, nesse caso 'berlin'. Então o código pode ser compilado. Quando compilado corretamente, é possível notar o surgimento de um *dropbox* contendo o nome do contrato.

Ao acessar a aba "Deploy & run transaction"nos é possível escolher a qual rede será utilizada,

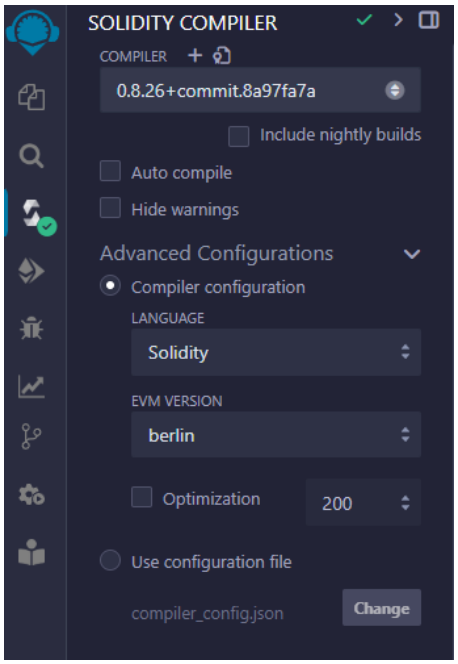

Figura 2: Configuração compilador

Figura 3.

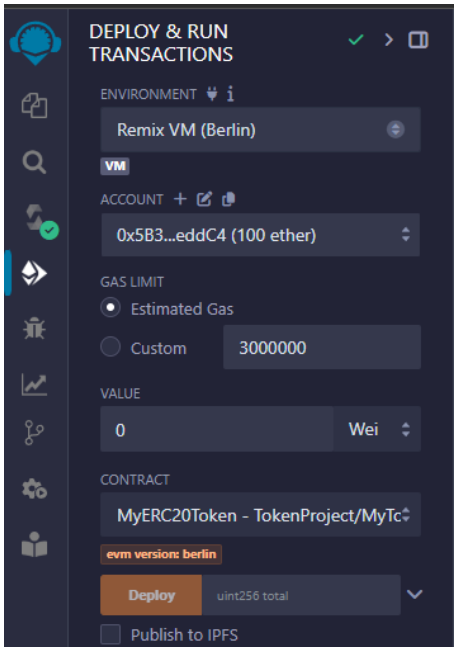

Figura 3: Deploy do contrato

Antes de efetivamente realizar o deploy do contrato, é necessário adicionar um valor para ele, nesse caso foi utilizado 1000,Figura 4.

O sucesso do deploy pode ser observado no console, mais detalhes são mostrados ao clicar no saida no console, Figura 5, como status, nome do contrato, custos da transação, gas, etc.

Ainda na mesma aba, é possível observar uma lista de todos os contratos que foram realizados *deploy*, e ao escolher um abre-se a opção de realizar as operações implementadas no código do contrato, Figura 6.

Ao utilizar a função do contrato *totalSuply*, é possível obter a quantidade total de *tokens* do contrato, como pode ser observador na Figura 7, na saída do terminal em *decoded output*.

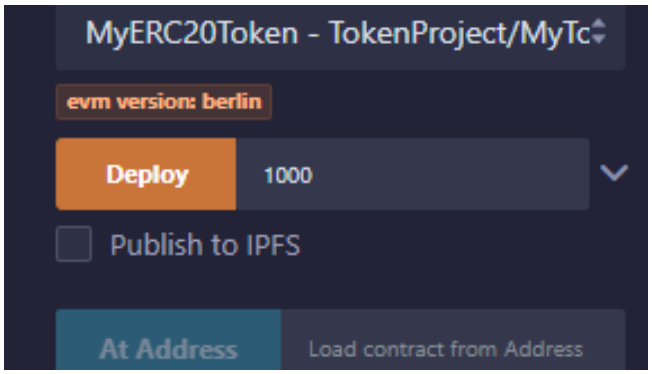

Figura 4: Valor do contrato

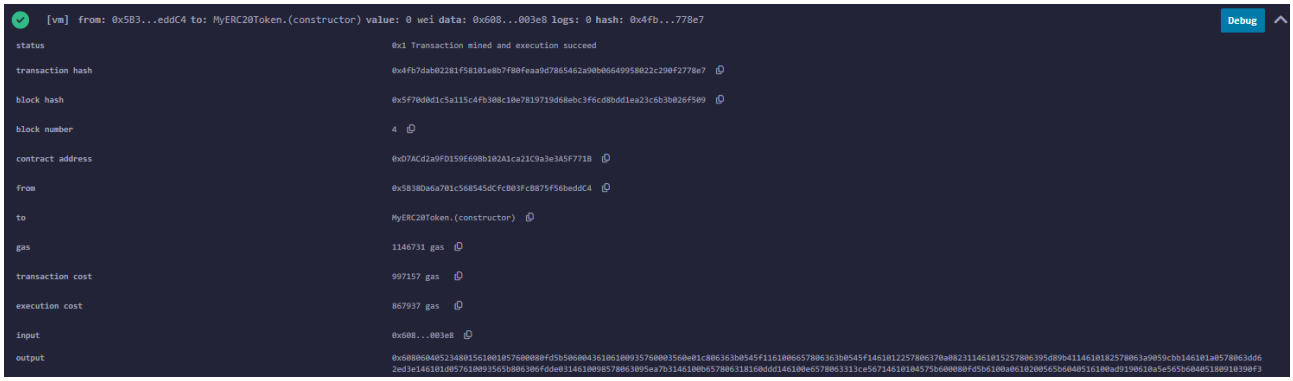

Figura 5: Dados do deploy do contrato

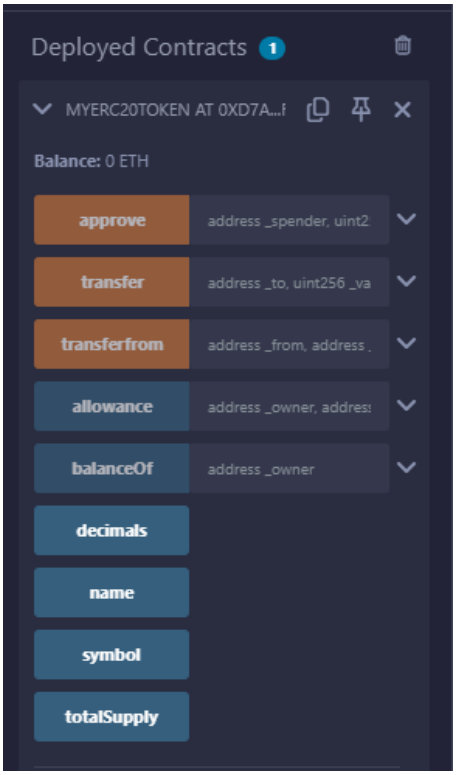

Figura 6: Operações do contrato

7 Considerações Finais

Este projeto demonstrou a criação de um token ERC-20 simples usando o Remix IDE e a linguagem Solidity. O contrato implementa as funções essenciais, como transferência de tokens,

Figura 7: Enter Caption

aprovação de gastos por terceiros e consulta de saldos. Através dessa implementação, foi possível entender os conceitos básicos de como os tokens funcionam na blockchain Ethereum, além de destacar a facilidade de desenvolvimento com ferramentas como o Remix.

Referências

Antonopoulos, A.M. & G. Wood. 2018. *Mastering ethereum: Building smart contracts and dapps*. O'Reilly Media, Incorporated. https://books.google.com.br/books?id=SedSMQAACAAJ.

Ethereum Foundation. 2024. Ethereum documentation. https://ethereum.org. Accessed: 2024-12-02.

Imran, Bashir. 2018. *Mastering blockchain : Distributed ledger technology, decentralization, and smart contracts explained, 2nd edition.* Packt Publishing. https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1789486&lang=pt-br&site=eds-live&scope=site.

Szabo, Nick. 1997. Formalizing and securing relationships on public networks. *First Monday* 2(9). http://dblp.uni-trier.de/db/journals/firstmonday/firstmonday2.html#Szabo97.