Projeto de desenvolvimento de Contratos Inteligentes na Ethereum

Alunos:
Victor Daniel Manfrini Pires
Emica Oliveira da Costa

Matéria:
Sistemas Distribuídos

Gerenciador de tarefas

Um gerenciador de tarefas feito com contratos inteligentes com Ethereum usando o Solidity. O projeto consiste em criar um contrato inteligente taskManager (Gerenciador de tarefas), uma tarefa no nosso contrato contém uma estrutura como mostra logo abaixo:

```
struct TaskStruct {
    address owner;
    string name;
    TaskPhase phase;
    TypeTask ttype;
    uint priority;
}
```

Cada uma de nossas tarefas podem ser classificadas de acordo com a fase, essa fase pode assumir cinco modos diferentes. Que podem ser ToDo, InProgress, Done, Blocked, Review, Postponed, Canceled.

Quando criamos um contrato, criamos também suas próprias regras. As que criamos estão listadas logo abaixo.

1. Cada pessoa, representada por sua conta Ethereum, pode ter várias tarefas.
2. Quando uma tarefa for adicionada, um evento será emitido, ou seja, qualquer sistema que esteja monitorando o Blockchain pode acompanhar a criação de tarefas.
3. Dado um índice da tarefa, queremos saber seus detalhes: dono, nome, fase e prioridade.
4. Queremos ter uma lista com os índices de todas as nossas tarefas.
5. Cada pessoa só pode ver a sua lista de tarefas.
6. Qualquer pessoa pode adicionar uma tarefa para si mesma, mas não pode adicionar para outras pessoas.
7. Depois que uma tarefa for criada, a única coisa que pode ser alterada é a sua fase.
8. Lista por prioridade
9. Atualização do nome de uma tarefa

Setup

Para desenvolver essa aplicação usamos dentro do vs code a extensão do Solidity de Juan Blanco. O solidity é a linguagem de programação para desenvolver contratos inteligentes dentro do Ethereum Virtual Machine.

Como não publicamos de verdade em um Blockchain, usamos o Truffle que é um blockchain pessoal para seu desenvolvimento com Ethereum que você pode usar para implantar contratos, desenvolver, compilar e executar testes.

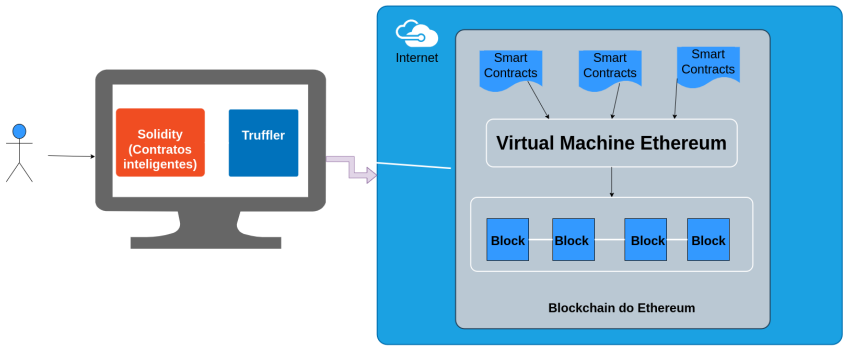
Primeiros passos:

- Dentro do vs code, vá em Extensions digite no campo Solidity e baixe a extensão do Juan Blanco
- Para instalar o Truffle no linux, vá no terminal e digite *npm install -g truffle*

Inicializando o projeto com o Truffle dentro da sua pasta de trabalho:

- *truffle init*
- *npm init -y*
- *truffle develop*
- *.exit*

Arquitetura



Interface

getTask() : → Dado um índice da tarefa, retorna saber seus detalhes: dono, nome, fase e prioridade
listMyTask() →Retorna uma lista com os índices de todas as nossas tarefas
addTask(): → Adicinar uma nova tarefa para si mesma
updatePhase(): →Atualiza fase da tarefa
uupdateName(): →Atualiza o nome da tarefa
listMyTaskByPriority() →Retorna uma lista por prioridade com os índices

implementação: Tutorial simples, foto do código com explicação

Contrato inteligente: Para iniciar a criação de um contrato, primeiro declaramos *contract* e definimos um nome para ele o nome. Todo o conteúdo do contrato está definido dentro dele.

```
pragma solidity 0.5.4;
pragma experimental ABIEncoderV2;

contract TaskManager {

}
```

Para nosso contrato criamos uma variável de estado, que guarda a informações que estão armazenadas no nosso contrato, a nossa variável é nTasks que guarda o número de tarefas. Em seguida criamos dois enum, eles são um tipo de criado pela própria linguagem Solidity que cria um tipo que o desenvolvedor deseja. Os nossos enum são *TaskPhase*; onde são definidas as fases do nosso projeto, e *TypeTask*; onde são definidos os tipos das nossas tarefas.

```
uint public nTasks;

enum TaskPhase {ToDo, InProgress, Done, Blocked, Review, Postponed,
Canceled}

enum TypeTask {Personal, Family, Home, Work, Review, Student}
```

Continuando a construção, criamos uma struct com as variáveis owner, name, phase e ttype para formar o nosso tipo de dado. Depois criamos dois *arrays* de *struct*, um para guardar as tarefas e outro as tarefas por prioridade.

Struct: TaskStruct
Arrays: task e taskByPriority

```
   /*
       Estrutura da tarefa:
           criador/dono,
           nome da tarefa,
           fase da tarefa,
           tipo de dprioridade
```

```
    */
    struct TaskStruct {
        address owner;
        string name;
        TaskPhase phase;
        TypeTask ttype;
        /*
            A prioridade varia de 1-5. Sendo 1 prioridade muito alta e 5
menos importante.
        */
        uint priority;
    }

    TaskStruct[] private tasks;
    TaskStruct[] private taskByPriority;
```

Agora criamos dois tipos diferentes de variáveis; *myTasks* que é do tipo *mapping* e o TaskAdded do tipo *event*.
O *mapping* é uma estrutura do tipo chave valor, que funciona da seguinte forma dado uma chave ela retorna um valor. Já o *event* é um envio de log para o mundo externo, isto é, toda vez em que criamos um evento de uma nova tarefa igual no nosso caso geramos um evento enviando para todos os blockchain que estão monitorando essa tarefa.

```
    mapping (address => uint[]) private myTasks;

    event TaskAdded(address owner, string name, TaskPhase phase, uint
priority);
```

Temos um método para modificar o comportamento das nossas funções o *modifier*. Ele é usado no nosso código pelo o owner para verificar uma tarefa a partir de um índice.

```
    modifier onlyOwner (uint _taskIndex) {
         if  (tasks[_taskIndex].owner == msg.sender) {
           _;
        }
    }
```

Outro método usado é o *constructor*, assim como os construtores usados em outras linguagens de programação como C/C++, o constructor do Solidity é executado somente uma vez no momento em que criamos os contratos inteligentes.

```
    constructor() public {
```

```
        nTasks = 0;
        addTask ("Create Task Manager", TaskPhase.Done, TypeTask.Work,
1);
        addTask ("Create Your first task", TaskPhase.ToDo,
TypeTask.Personal, 2);
        addTask ("Clean your house", TaskPhase.ToDo, TypeTask.Home, 5);
        addTask ("Watch new spider man movie", TaskPhase.InProgress,
TypeTask.Family, 3);
        addTask ("watch new episode of onpiece", TaskPhase.ToDo,
TypeTask.Personal, 5);
    }
```

Por fim criamos as nossas regras de contratos por meio das funções

```
    /*
        Função que lista todas as tarefas de um usuario(dono)
    */
    function listMyTasks() public view returns (uint[] memory) {
        return myTasks[msg.sender];
    }

    /*
        Função responsavel por atualizar o array de tarefas de acordo
com uma prioridade
    */
    function updateArrayTaskByPriority(uint _priority) private {
        /*
            Limpa o array de tarefas por prioridade
        */
        delete taskByPriority;

        uint[] memory myTasksAll = myTasks[msg.sender];

        for (uint index = 0; index < myTasksAll.length; index++) {
            if (tasks[myTasksAll[index]].priority == _priority) {
                TaskStruct memory taskAux = TaskStruct ({
                    owner: msg.sender,
                    name: tasks[myTasksAll[index]].name,
                    phase: tasks[myTasksAll[index]].phase,
                    ttype: tasks[myTasksAll[index]].ttype,
                    priority: tasks[myTasksAll[index]].priority
                });
```

```
                taskByPriority.push(taskAux);
            }
        }
    }

    /*
        Função que lista todas as tarefas de um usuario(dono)
        de uma determinada prioridade.
    */
    function listMyTasksByPriority(uint _priority) public returns
(TaskStruct[] memory) {
        /*
            Lança um exeção se a prioridade passada for
            menor que 0 ou maior que 5
        */
        require ((_priority >= 1 && _priority <=5), "Priority must be
between 1 and 5");
        /*
            Chama a função que limpa o array de tarefas por prioridade,
            e depois insere apartir de uma nova prioridade
        */
        updateArrayTaskByPriority(_priority);
        /*
            Lança um exeção se a o array de prioridades for
            menor que ou igual a 0
        */
        require ((taskByPriority.length > 0), "No tasks with this
priority found");

        return taskByPriority;
    }

    /*
        Função que cria uma nova tarefa
    */
    function addTask(string memory _name, TaskPhase _phase, TypeTask
_type, uint _priority) public returns (uint index) {
        require ((_priority >= 1 && _priority <=5), "priority must be
between 1 and 5");
        TaskStruct memory taskAux = TaskStruct ({
            owner: msg.sender,
            name: _name,
            phase: _phase,
```

```
            ttype: _type,
            priority: _priority
        });
        index = tasks.push (taskAux) - 1;
        nTasks ++;
        myTasks[msg.sender].push(index);
        emit TaskAdded (msg.sender, _name, _phase, _priority);
    }

    /*
        Função que atualiza a fase do projeto
    */
    function updatePhase(uint _taskIndex, TaskPhase _phase) public
onlyOwner(_taskIndex) {
        tasks[_taskIndex].phase = _phase;
    }

    /*
        Função que atualiza a o nome do projeto
    */
    function updateName(uint _taskIndex, string _name) public
onlyOwner(_taskIndex) {
        tasks[_taskIndex].name = _name;
    }
```