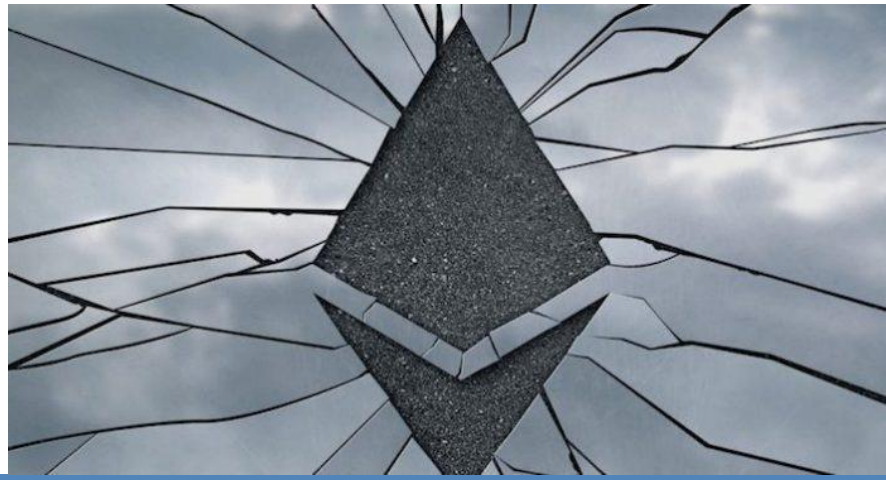
Curso desarrollo Blockchain Ethereum con Solidity

Clase 6

Introducción al standard

Introducción al standard

- Los **EIPS** (Ethereum Improvement Proposal), son básicamente propuestas de mejora a incluir en futuras versiones*
- Dentro de los EIPS se encuentran todos los **ERC** (Ethereum Request for Comments)**
- El EIP número 20, que se llamó **ERC20** era el que proponía un estándar de funcionamiento para los Tokens
- La aceptación supuso una mejora en el uso, ya que desde entonces los tokens son compatibles con este estándar y eso permite que los wallets puedan soportar los nuevos tokens por defecto.



Diferencias entre ER20 y ERC721

Diferencias entre ER20 y ERC721

- Es importante entender que si bien existen otros estándares, los **er20** y **erc721** son los más utilizados
- Cada ERC tiene su especificacion definida
- Un token ERC20 NO puede ser un token ERC721
- Ambos son tradeables dentro de la Ethereum Blockchain (y fuera*)

ERC20  ERC721

Diferencias entre ER20 y ERC721

- Cuando alguien dice que tiene un token "**ERC-20**" solo significa que ese contrato de token responde a una serie común (predefinida) de métodos.
- Significa que el token puede ser **transferido**, **consultado**, **aprobado**, etc.
- **No** significa que el token tenga un valor ($$$)
- **No** significa que el token haga algo*
- Es posible ver un **totalSupply****
- **Está totalmente pensado para ser Dinero**

Diferencias entre ER20 y ERC721

- La interfaz a implementar para un token de tipo **ERC20**

```
pragma solidity ^0.4.18;
/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 * @jds 2018
 */
contract ERC20 {
  function totalSupply() public view returns (uint256);

  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);

  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns (bool);
  function approve(address spender, uint256 value) public returns (bool);

  event Transfer(address indexed from, address indexed to, uint256 value);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

Diferencias entre ER20 y ERC721

- Para entender la utilidad de un token **ERC-721** basta con pensar en objetos coleccionables.
- Figuritas de un álbum, cartas especiales, cualquier objeto distinto
- La definición del ERC721 lo categoriza como "**non-fungible**"
- Cada token del estándar es **único** e **irrepetible**
- Un token de tipo ERC721 **NO puede ser dividido** ya que funciona como una unidad*
- El caso de éxito más conocido sin dudas fue CryptoKitties
- **Está totalmente pensado para cosas, y cualquier tipo compatible con "cosas"**

ERC721

Diferencias entre ER20 y ERC721

- La interfaz a implementar para un token de tipo **ERC721**

```
pragma solidity ^0.4.18;
/**
 * @title ERC721 Non-Fungible Token Standard basic interface
 * @dev see https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md
 * JdS 2018
 */
contract ERC721Basic {
  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
  event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);

  function balanceOf(address _owner) public view returns (uint256 _balance);
  function ownerOf(uint256 _tokenId) public view returns (address _owner);
  function exists(uint256 _tokenId) public view returns (bool _exists);

  function approve(address _to, uint256 _tokenId) public;
  function getApproved(uint256 _tokenId) public view returns (address _operator);

  function setApprovalForAll(address _operator, bool _approved) public;
  function isApprovedForAll(address _owner, address _operator) public view returns (bool);

  function transferFrom(address _from, address _to, uint256 _tokenId) public;
  function safeTransferFrom(address _from, address _to, uint256 _tokenId) public;
  function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes _data) public;
}
```

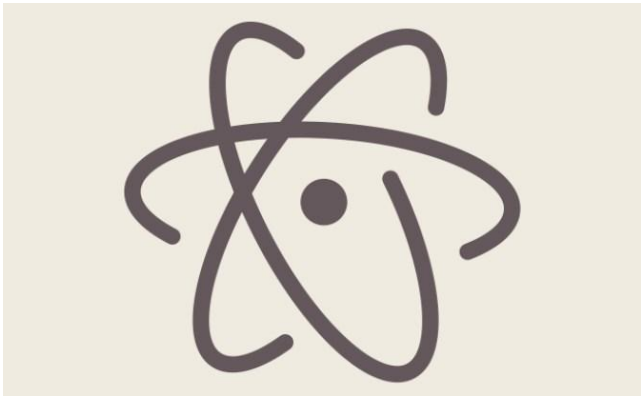
Consideraciones

Consideraciones

Antes de empezar, será necesario tener instalado las siguiente cosas

- Truffle
- Ganache-cli (testRPC)
- Atom

También se podrá desarrollar puramente en Remix*

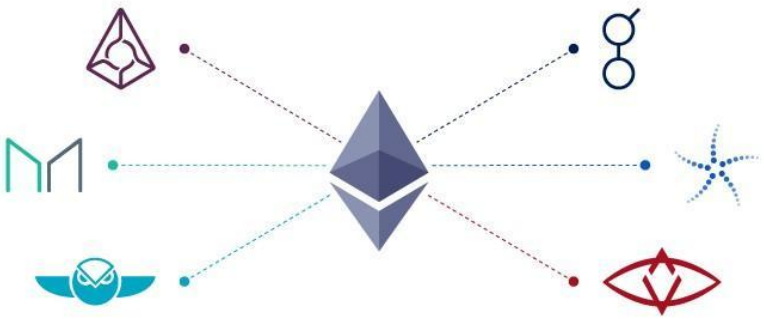


Definición del Smart Contract

Definición del Smart Contract

Desarrollaremos un token compatible con el estándar **ERC20**. Para ello deberemos implementar los métodos

- function **totalSupply**() public view returns (uint256);
- function **balanceOf**(address who) public view returns (uint256);
- function **transfer**(address to, uint256 value) public returns (bool);
- function **allowance**(address owner, address spender) public view returns (uint256);
- function **transferFrom**(address from, address to, uint256 value) public returns (bool);
- function **approve**(address spender, uint256 value) public returns (bool);



Definición del Smart Contract

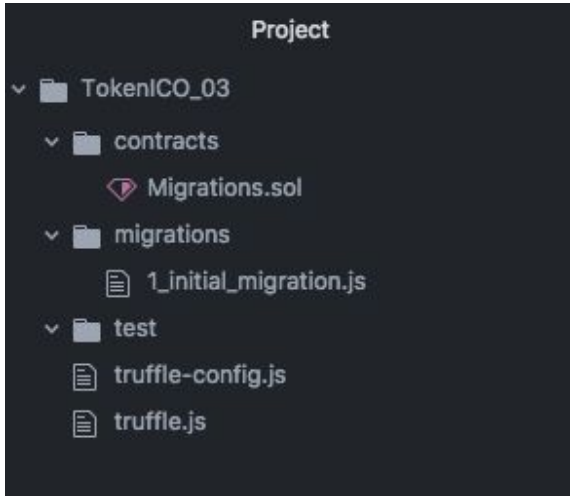
Iniciaremos el proceso de creación de nuestro token propio con su correspondiente ICO ejecutando el comando **truffle init**

```
MacBook-Pro-de-mac:TokenICO_03 mac$ truffle init
Downloading...
Unpacking...
Setting up...
Unbox successful. Sweet!

Commands:

  Compile:        truffle compile
  Migrate:        truffle migrate
  Test contracts: truffle test
MacBook-Pro-de-mac:TokenICO_03 mac$
```

Verificaremos que la estructura de nuestro proyecto sea como la siguiente

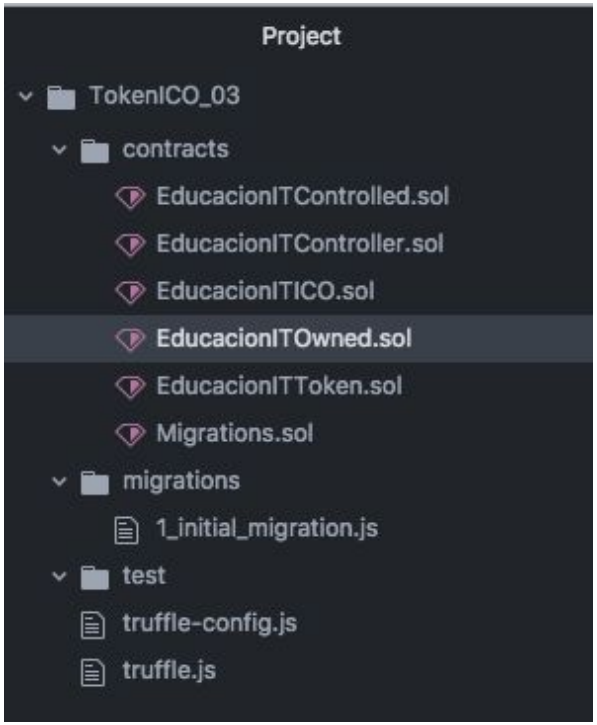


Definición del Smart Contract

Agregaremos los contratos necesarios (a nivel archivo) para que el proyecto tome la dimensión adecuada. Para el caso del token propio con ICO necesitaremos generar los siguientes archivos:

- **EducacionITToken**.sol
- **EducacionITICO**.sol
- **EducacionITController**.sol
- **EducacionITControlled**.sol
- **EducacionITOwned**.sol

De manera que nuestro directorio ha de quedar como el siguiente



Definición del Smart Contract

Antes de continuar, deberemos configurar el archivo **truffle.js** para poder efectuar correctamente el deploy de nuestro contrato inteligente. El mismo quedará de la siguiente manera

```
/*
@JDS: Recordar que el GAS Sobrante volverá y que cuanto mas paguemos por cada
unidad de gas mayor será la prioridad dentro de la red
 */

module.exports = {
  networks : {
    development: {
      host: "localhost",
      port: 8545,
      network_id: "*",
      gas: 4000000,//4 millones
      gasPrice: 30e9//30 x 10 elevado a la 9
    }
  }
};
```

Definición del Smart Contract

Configuraremos el archivo EducacionITController.sol, de manera que quede de la siguiente manera

```
pragma solidity ^0.4.24;

contract EducacionITController {
  func proxyPayment(address _dir) payable returns (bool);
}
```

Lo que estamos definiendo dentro de este “contrato” es en realidad una simple interfaz que luego implementaremos en pasos subsiguientes

Definición del Smart Contract

Configuraremos nuestro archivo **EducacionITControlled** de la siguiente manera

```
pragma solidity ^0.4.24;

contract EducacionITControlled {
  address controller;

  modifier onlyController() {
    require(msg.sender == controller);
    _;
  }
}
```

Aquí hemos definido una dirección **controller** de tipo address que luego utilizamos para crear el modificador de función "**onlyController**", el cual más adelante será utilizado en la validación de determinadas funciones para que éstas **solo puedan ser ejecutadas** por esa dirección predefinida.

Definición del Smart Contract

Ahora, generaremos dos funciones extra en EducacionITControlled que luego serán de gran utilidad. Dejando el contrato de la siguiente manera

```
pragma solidity ^0.4.24;

contract EducacionITControlled {
  address controller;

  constructor() public {
    controller = msg.sender;
  }

  function changeController(address _newController) onlyController public {
    controller = _newController;
  }

  modifier onlyController() {
    require(msg.sender == controller);
    _;
  }
}
```

Definición del Smart Contract

Configuraremos nuestro archivo EducacionITToken de la siguiente manera

```
pragma solidity ^0.4.24;
import "./EducacionITControlled.sol"

contract EducacionITToken is EducacionITControlled {

  constructor() public {
    controller = msg.sender;
  }

}
```

Lo que hemos hecho fue

- Importar el contrato **EducacionITControlled**.sol
- Indicar que nuestro contrato **EducacionITToken** es, en efecto, un contrato controlado
- Generar el constructor del **Token**, asignando como **controller** al sender de la transacción

Definición del Smart Contract

De manera que nuestro contrato inicial quedará de la siguiente manera*

```
pragma solidity ^0.4.24;
import "./EducacionITControlled.sol"

contract EducacionITToken is EducacionITControlled {

  mapping(address => uint256) public balances;//balances de cuentas - tokens
  uint256 public totalSupply;//Maxima cantidad posible de tokens

  constructor() public {
    controller = msg.sender;
  }

  function balanceOf(address _owner) constant returns (uint256 balance) {
    returns balances[_owner];
  }

  function transfer(address _to, uint256 _value) returns (bool success) {
    return true;//Temporal, luego se cambiará en función de lo que suceda
  }

  function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
    return true;//Temporal, luego se cambiará en función de lo que suceda
  }
}
```

Definición del Smart Contract

Configuraremos nuestro archivo "**EducacionITOwned**.sol" de la siguiente manera

```
pragma solidity ^0.4.24;
/// @dev jds `EducacionITOwned` es un contrato base que asigna un dueño (owner)
///  que puede cambiar luego
contract EducacionITOwned {
  address public owner;
  address public newOwner;

  constructor() {
    owner = msg.sender;
  }

  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }

  function changeOwner(address _newOwner) onlyOwner {
    newOwner = _newOwner;
  }

  //NOTE: JDS -  Recordar que uno de los patrones más utilizados es el
  // de aceptación por parte del usuario receptor (que invocaría a esta
  // función para aceptar sus tokens)
  function acceptOwnership() public {
    if (msg.sender == newOwner) {
      owner = newOwner;
    }
  }

}
```

Definición del Smart Contract

Configuraremos nuestro archivo "**EducacionITICO**.sol" de la siguiente manera

```
pragma solidity ^0.4.24;
import "./EducacionITControlled.sol";
import "./EducacionITOwned.sol";
import "./EducacionITToken.sol";

contract EducacionITICO is EducacionITOwned, EducacionITController {

  //Definimos la cantidad máxima de dinero (en ether) que aceptaremos
  uint256 constant public limit = 200 ether;
  //Definimos una equivalencia de la cantidad de "EducacionITToken" que daremos
  //por cada Ether recibido. En nuestro caso será de 230
  uint256 constant public equivalence = 230;
  //Cantidad total
  uint256 public totalCollected;

  EducacionITToken tokens;

  //Cuenta en la que se depositarán los ether recibidos
  address happyOwner = 0x3a7D7125ba608175bBb48E014274127D81f4c25a;

}
```

Definición del Smart Contract

Creamos el constructor para igualar el owner a quien deploye nuestro contrato e inicializamos en cero la cantidad total recibida de ether.

```
constructor() {
  owner = msg.sender;
  totalCollected = 0;
}

function initializeToken(address _token, address _destiny) {
  //Solo se podrá inicializar la ICO una vez
  require(address(tokens) == 0x0);

  tokens = EducacionITToken(_token);
  require(tokens.totalSupply == 0);
  require(tokens.controller == address(this));

  happyOwner = _destiny;
}
```

Definición del Smart Contract

Implementamos la función **proxyPayment** (que viene de **EducacionITController**.sol) y generamos una llamada **realBuy** donde pondremos la lógica de **compra**

```
//Implementación de proxyPayment (viene de EducacionITController)
function proxyPayment(address _dir) payable returns (bool) {
  return realBuy(_dir, msg.value);
}

function realBuy(address _sender, uint256 _amount) public returns (bool) {
  //calculamos la cantidad de tokens generados
  uint256 tokensGenerated = _amount * equivalence;
  //revisamos no pasarnos de lo estipulado en el contrato
  require(totalCollected + _amount <= limit);
  //generamos los tokens para la cuenta
  assert(tokens.generateTokens(_sender, tokensGenerated));
  //Enviamos el ether a nuestra cuenta segura
  happyOwner.transfer(_amount);
  //Actualizamos la cantidad actual recibida de ether
  totalCollected = totalCollected + _amount;

  return true;
}
```

Token

Token

La particularidad de nuestro **TOKEN** es que debe implementar el estándar **ERC20**

```
pragma solidity ^0.4.18;
/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 * @jds 2018
 */
contract ERC20 {
  function totalSupply() public view returns (uint256);

  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);

  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns (bool);
  function approve(address spender, uint256 value) public returns (bool);

  event Transfer(address indexed from, address indexed to, uint256 value);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

Token

El **TOKEN** es, en efecto, otro **SmartContract**

```
pragma solidity ^0.4.24;
import "./EducacionITControlled.sol"

contract EducacionITToken is EducacionITControlled {

  mapping(address => uint256) public balances;//balances de cuentas - tokens
  uint256 public totalSupply;//Maxima cantidad posible de tokens

  constructor() public {
    controller = msg.sender;
  }

  function balanceOf(address _owner) constant returns (uint256 balance) {
    //TODO: JDS - Mencionar caso especial cero
    returns balances[_owner];
  }

  function transfer(address _to, uint256 _value) returns (bool success) {
    //TODO: JDS - Implementar!
    return true;//Temporal, luego se cambiará en función de lo que suceda
  }

  function transferFrom(address _from, address _to, uint256 _value) onlyController returns (bool success) {
    //TODO: JDS - Implementar!
    return true;//Temporal, luego se cambiará en función de lo que suceda
  }
}
```

Token

Finalmente, implementaremos un método **realTransfer** para transferir nuestros **tokens**

```
function realTransfer(address _from, address _to, uint256 _value) internal returns (bool) {
  if (_value == 0) return true;//si el monto a transferir es cero lo daremos por OK

  //NOTE: JDS - Evitar que el destinatario sea la dirección cero evitará la "quema" de tokens
  require((_to!=0) && (_to!= address(this)));

  //recordar que "balanceOf" es un método de ERC20
  uint256 previousBalanceFrom = balanceOf(_from);
  //si quien envía los tokens no tiene la cantidad suficiente terminamos aquí
  if (previousBalanceFrom < _value) {
    return false;
  }
  //Restamos los tokens del emisor
  balances[_from] = balances[_from] - _value;

  uint256 previousBalanceTo = balanceOf(_to);
  require(previousBalanceTo + _value > previousBalanceTo);
  //incrementamos los tokens del receptor
  balances[_to] = balances[_to] + _value;
  //Llamamos al evento
  Transfer(_from, _to, _value);
}
```

Generación

Generación

Reservaremos la siguiente función para la **generación** de nuestros **tokens**. No obstante, cabe mencionar que la implementación podría diferir acorde a las necesidades propias del token y/o empresa para la que se desarrolle el mismo.

```
function generateTokens(address _owner, uint256 _amount) onlyController returns (bool) {
  uint256 currentTotalSupply = totalSupply;
  require(currentTotalSupply + _amount >= totalSupply);//evitamos overflow

  uint256 previousBalanceTo = balanceOf(_owner);
  require(previousBalanceTo + _amount > previousBalanceTo);//evitamos overflow

  //incrementamos el totalSupply actual
  totalSupply = currentTotalSupply + _amount;
  //Incrementamos la cantidad de tokens del usuario
  balances[_owner] = previousBalanceTo + _amount;
  //Invocamos al evento desde una dirección cualquiera
  Transfer(0, _owner, _amount);

  return true;
}
```

Envío

Envío

El envío de **tokens** se apega, en nuestro caso, al estándar **ERC20**, por lo cual se realizaría de la siguiente en los métodos **transfer** y **transferFrom** propios del ERC20*

```
function transfer(address _to, uint256 _value) returns (bool success) {
  return true;//Temporal, luego se cambiará en función de lo que suceda
}

function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
  return true;//Temporal, luego se cambiará en función de lo que suceda
}
```

Como en nuestro caso hemos llevado toda la lógica al método interno **realTransfer**, solo falta **invocarlo** con el pool de parámetros indicado **dentro de cada función**

```
function transfer(address _to, uint256 _value) returns (bool success) {
  return realTransfer(msg.sender, _to, _value);
}

function transferFrom(address _from, address _to, uint256 _value) onlyController returns (bool success) {
  return realTransfer(_from, _to, _value);
}
```

Recepción

Recepción

De la misma manera en que el envío de tokens se realiza mediante la implementación del **ERC20**, la recepción ocurre de igual manera.
En nuestro contrato, podemos apreciar que el **envío** (asignación) ocurre dentro del **realTransfer**

```
uint256 previousBalanceTo = balanceOf(_to);
require(previousBalanceTo + _value > previousBalanceTo);
//incrementamos los tokens del receptor
balances[_to] = balances[_to] + _value;
//Llamamos al evento
```

Luego, la función invoca al evento Transfer (definido al inicio del contrato) para notificarle a quien quiera escucharlo

```
event Transfer(address indexed _from, address indexed _to, uint256 _value);

//Llamamos al evento
Transfer(_from, _to, _value);
}
```

Compatibilidad

Compatibilidad

Como regla general, elegir un **estándar** de token (o tipo de **token**) siempre será la clave

- No todo token se apega al **ERC20**
- Si la finalidad es un bien único el **ERC721** es el indicado
- La utilización de estándares fomentará su **aceptación** por parte de la comunidad
- La utilización de dichos estándares permitirá la rápida integración con productos existentes
- Podríamos perfectamente, integrar nuestro token con Metamask (en el próximo curso)