DESARROLLO DE APLICACIONES DESCENTRALIZADAS EN LA BLOCKCHAIN DE ETHEREUM

ADOLFO SANZ DE DIEGO

COMMIT-CONF 2018

2 AUTOR

2.1 ADOLFO SANZ DE DIEGO

Asesor. Desarrollador. Profesor. Formador.

- Blog: asanzdiego.com
- Correo: asanzdiego@gmail.com
- GitHub: github.com/asanzdiego
- Twitter: twitter.com/asanzdiego
- LinkedIn: in/asanzdiego
- SlideShare: slideshare.net/asanzdiego

3 INDICE

3.1 ¿QUÉ VAMOS A VER?

1. Parte teorica
2. Descripción ĐApp
3. Creación Smart Contracts
4. Testing Smart Contracts
5. Creación UI
6. Subida a entornos de prueba

4 TEORÍA

4.1 ¿QUÉ ES BITCOIN?

- **Protocolo y red P2P** = dinero digital.
- **Claves privadas** = transferir fondos.
- **Descentralizado** = no necesitan un tercero.

4.2 ¿QUÉ ES BLOCKCHAIN?

- Datos se guardan en **cadena de bloques** con información del anterior.
- **Histórico transacciones** difícilmente falsificable.
- Transacciones verificadas **de forma descentralizada** ¿cómo?

4.3 ¿QUÉ ES EL MINADO?

- Generación nuevos bloques.
- Problema **elegir el bloque correcto** ¿cómo?

4.4 ¿MECANISMOS DE CONSENSO?

- **Prueba de trabajo o PoW** = recompensa al primer minero en resolver problema gasto computacional elevado, pero verificación inmediata.
  - Elevado gasto energético.
- **Prueba de participación o PoS** = probabilidad de obtener recompensa proporcional a monedas acumuladas.
  - Problemas en caso de una bifurcación de la cadena.

4.5 ¿TIPOS DE BLOCKCHAINS?

- **públicas**: cualquiera puede minar y las transacciones son públicas.
- **privadas**: solo se puede minar por invitación y las transacciones solo las pueden ver los mineros.
- **mixtas**: solo se puede minar por invitación pero las transacciones son públicas.

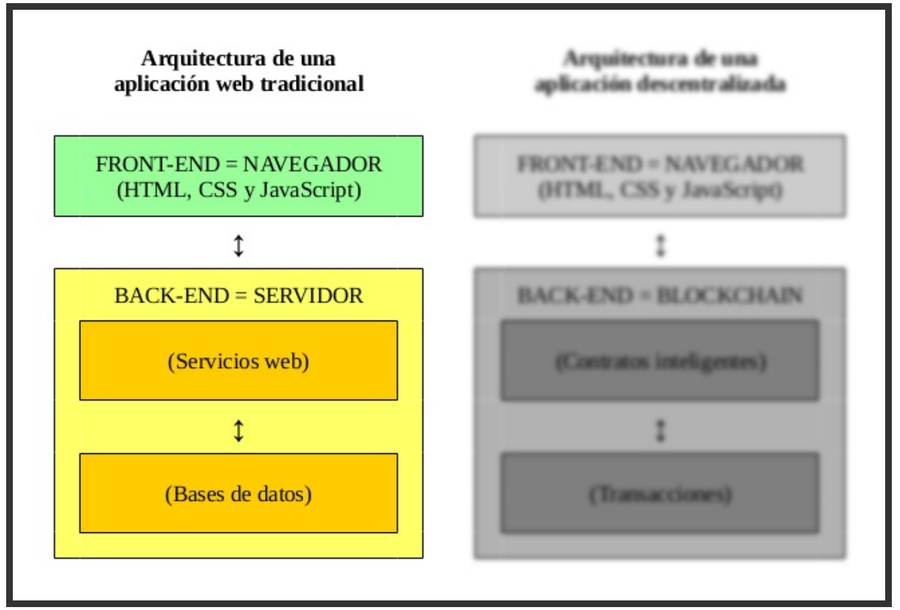
4.6 ¿QUÉ ES ETHEREUM?

- Blockchain pública que **ejecuta código y guarda datos** de forma descentralizada.
- **Solidity** = compila a bytecode interpreta Ethereum Virtual Machine (EVM).
- **Gas** = coste ejecución en Ethers.

4.7 ¿QUÉ ES UN SMART CONTRACT?

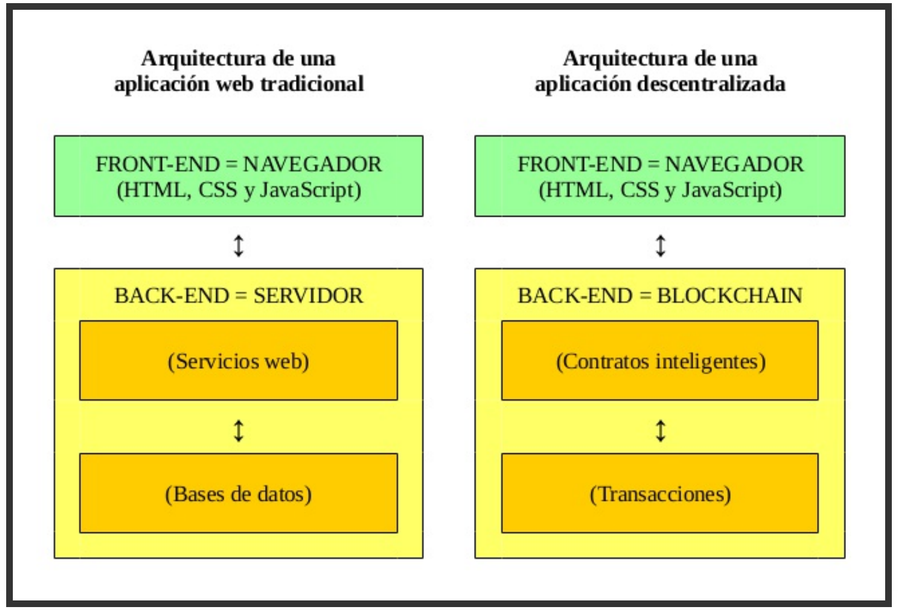
- Programa informático en blockchain.
  - **Autónomo**: no necesita ningún servidor.
  - **Confiable**: trazabilidad de los datos.
  - **Imparable**: ejecutarse siempre.
  - **Inmutable**: no se puede modificar.
  - **Descentralizado**: no necesita tercero para ser verificado.

4.8 ¿APPS TRADICIONALES?



Apps

4.9 ¿APPS DESCENTRALIZADAS?



Apps VS ĐApps

4.10 RESUMEN

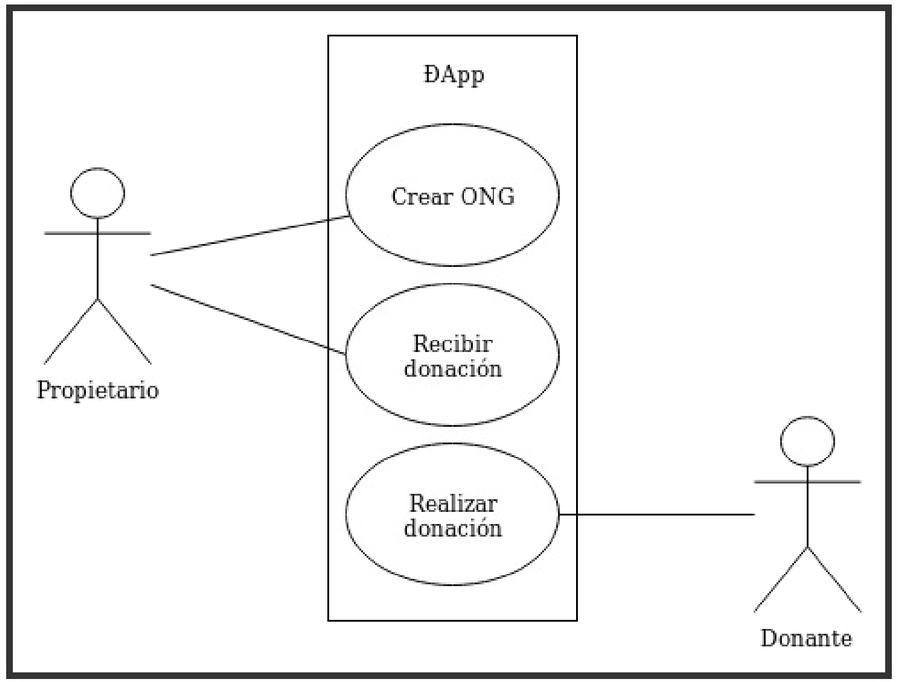
- **Bitcoin** = dinero descentralizado.
- **Blockchain** = cadena de bloques dificilmente falsificable.
- **Minado** = geneación bloques con mecanismos de consenso.
- **Ethereum** = blockchain pública permite ejecutar código y guardar datos.
- **Smart Contract** = programa que se ejecuta en una blockchain.
- **ĐApp** = aplicación que utiliza smart contracts.

5 DESCRIPCIÓN ĐAPP

5.1 ĐAPP DE DONACIONES A ONG

- Vamos a desarrollar una ĐApp para gestionar donaciones a ONG con 2 tipos de usuarios:
  - **Propietarios**: crea la ONG y recibe las donaciones.
  - **Otros usuarios**: realizan donaciones a las ONG.

5.2 CASOS DE USO



5.3 DEMO (RINKEBY)

- El back desplegado en la **red de pruebas** de Rinkeby.
- El front desplegado en **GitHub Pages** (siendo puristas habría que usar IPFS).
  - https://nongovernmentalorganizations.github.io

6 CREACIÓN SMART CONTRACTS

6.1 ¿ENTORNO DE DESARROLLO?

- **Solidity**: lenguaje de programación.
- **Remix**: IDE en un navegador.
- **Visual Studio Code**: IDE Software Libre
- **Plugin de Solidity de Juan Blanco**: Plugin de Solidity para Visual Studio Code

6.2 ¿CÓMO GUARDAR DATOS?

- Los datos se guardan en “structs” o **estructuras**.
- Las estructuras de datos se guardan en **arrays**.
- Para las relaciones se utilizan los **mappings**.

6.3 EJEMPLO

```
// Estructura
struct Organization {
    uint id;
    address owner;
    string name;
}

// Array
Organization[] public organizations;

// Mapping
mapping(address => uint) public ownerToOrganizationId;
```

6.4 ¿OTROS ELEMENTOS?

- **constructores** : solo se ejecutan cuando el contrato inteligente es desplegado en la blockchain.
- **eventos**: permiten trazar lo que sucede en los contratos inteligentes.
- **modificadores personalizados**: permiten hacer chequeos antes de ejecutar la lógica de una función.

6.5 EJEMPLO

```
// Constructor
constructor() public {
    // to fix problem with nulls
    organizations.push(Organization(0, 0, "0"));
}

// Eventos
event OrganizationCreated(
    uint indexed id, address indexed owner, string name);

// Modificadores personalizados
modifier ownerNotExists() {
    require(ownerToOrganizationId[msg.sender] == 0,
    "suplied owner already have an organization");
    _;
}
```

6.6 ¿MODIFICADORES DE VISIBILIDAD?

- Para variables de estado y para funciones:
  - **public**: desde otros contratos y desde el propio contrato.
  - **external**: desde otros contratos pero no desde el propio contrato.
  - **internal**: desde el propio contrato o de contratos que hereden de él.
  - **private**: sólo desde el propio contrato.

6.7 ¿OTROS MODIFICADORES?

- Para variables de estado:
  - **constant**: pueden ser modificadas.
- Para funciones:
  - **view**: no pueden modificar ninguna variable de estado (no consumen Gas).
  - **pure**: no pueden ni ver ni modificar ninguna variable de estado (no consumen Gas).
  - **payable**: admiten envío de dinero.

6.8 EJEMPLO

```
function addOrganization(string _name) external
    ownerNotExists() {
    ...
    emit OrganizationCreated(organizationId, msg.sender, _name);
}

function donation(uint _organizationId) external payable
    ownerExists(_organizationId) {
    ...
    emit DonationSubmitted(_organizationId, owner, msg.sender, msg.value);
}

function getOrganizationsLength() external view returns(uint) {

    return organizations.length;
}
```

contracts/NonGovernmentalOrganizations.sol

6.9 ¿COMENTARIOS GENERALES?

- Muchas limitaciones, por eso reducir la lógica al mínimo.
- Los bucles están muy desaconsejados (gas), por eso se usan los mappings.
- La ejecución es lenta, por eso implementar políticas de cacheo.
- Aspectos en proceso de mejora como tratamiento excepciones.

6.10 RESUMEN

- **Entorno**: Solidity, Remix, Plugin, Solium.
- **Datos**: estructuras, arrays y mapings.
- **Otros**: constructores y eventos.
- **Modificadores**: visibilidad, otros y personalizado.
- **Comentarios**: limitaciones, ejecución lenta, mejorando.

7 TESTING SMART CONTRACTS

7.1 GANACHE

- Ganache es un nodo privado para desarrollar y testear sin coste.

```
# instalar
npm install -g ganache-cli
```

```
# ejecutar
ganache-cli --gasLimit 7000001 --mnemonic "$(cat wallet.mnemonic)"
```

7.2 TRUFFLE

- Truffle es un framework de desarrollo de smarts contracts de Ethreum.

```
# instalar
npm install -g truffle
```

7.3 COMPILAR

- Una vez creado los smarts contracts hay que compilarlos:

```
# compilar
truffle compile
```

7.4 MIGRAR

- Una vez levantado el nodo privado (Ganache) en una terminal independiente, tenemos que migrar los smarts contracts compilados:

```
# migrar
truffle migrate
```

7.5 TESTS

```
var NonGovernmentalOrganizations = artifacts
    .require("NonGovernmentalOrganizations");
contract("NonGovernmentalOrganizations", async (accounts) => {
    it("addOrganization - ok", async () => {
        let instance = await NonGovernmentalOrganizations.deployed();
        let tx = await instance.addOrganization(expectedName,
            { from: expectedOwner });
        assert.equal(tx.logs[0].event, "OrganizationCreated");
        let result = await instance.organizations.call(expectedId);
        assert.equal(result[2], expectedName);
        let ownerId = await instance.ownerToOrganizationId(expectedOwner);
        assert.equal(ownerId, expectedId);
        let length = await instance.getOrganizationsLength();
        assert.equal(length, expectedLength);
    });
});
```

test/TestNonGovernmentalOrganizations.js

7.6 TESTEAR

- Una vez creados los tests los lanzamos:

```
# testear
truffle test
```

7.7 RESUMEN

- **Instalar Ganache**: npm install -g ganache-cli
- **Ejecutar Ganache**: ganache-cli --gasLimit --mnemonic
- **Instalar truffle**: npm install -g truffle
- **Compilar**: truffle compile
- **Migrar**: truffle migrate
- **Testear**: truffle test

8 CREACIÓN UI

8.1 TRUFFLE BOXES

- Truffle boxes proporciona boilerplates para no tener que empezar a desarrollar ĐApps desde cero.
- Hay para React, aunque yo soy más de Angular :-)

8.2 ANGULAR

- Aunque la UI se puede hacer con otros frameworks yo he usado Angular y Angular Material.

```
# instalar
npm install -g @angular/cli
```

8.3 NAVEGADOR

- Una vez instalado podemos verlo en http://localhost:4200/ ejecutando:

```
# ejecutar
ng serve
```

8.4 METAMASK

- **MetaMask**: es nodo ligero y wallet de Ethereum que permite ejecutar ĐApps en un navegador.

8.5 WEB3JS

- **web3js**: es una librería de comunicación entre la interfaz de usuario y un nodo de Ethereum.

8.6 ADDRESS USUARIO

```
getUserAddress() {
    if (!this.web3) {
        this.messagesService.sendErrorMessage('Try MetaMask.');
    }
    this.web3.eth.getAccounts().then(accounts => {
        if (!accounts || accounts.length === 0) {
            this.messagesService.sendErrorMessage('No user accounts.');
        }
        if (StorageUtil.getUserAddress() !== accounts[0]) {
            StorageUtil.setUserAddress(accounts[0]);
            this.messagesService.sendNewUserAddressMessage(accounts[0]);
        }
    }).catch(error => {
        this.messagesService.sendErrorMessage(error);
    });
}
```

8.7 CONTRACT INSTANCE

```
async getContractInstance(): Promise<any> {
    if (!this.web3) {
        throw new Error('web3 server not found. Try MetaMask.');
    }
    const ngoContract = contract(artifacts);
    ngoContract.setProvider(this.web3.currentProvider);
    try {
        const ngoInstance = await ngoContract.deployed();
        return ngoInstance;
    } catch (error) {
        console.log(error);
        throw new Error('Contract has not been deployed to network.');
    }
}
```

src/app/services/web3.service.ts

8.8 ADD ORGANISATION

```
async add(organization: Organization): Promise<Organization> {
    const contractInstance = await this.web3Service.getContractInstance();
    const oldOrganization = await this.getCurrentUserOrganizationAsOwner();
    if (oldOrganization) {
        throw new Error('The user is already owner of an organization.');
    }
    const transaction = await contractInstance.addOrganization(
        organization.name, { from: this.senderAddress });
    const newOrganization = this._getOrganizationFromTransaction(transaction
    this.organizations.push(newOrganization);
    console.log('OrganizationService->add', newOrganization);
    return newOrganization;
}
```

8.9 DONATION

```
async donation(id: number, ethValue: number): Promise<Donation> {
    const organization = await this.getOne(id);
    const contractInstance = await this.web3Service.getContractInstance();
    const weiValue = this.web3Service.etherToWei(ethValue.toString());
    const transaction = await contractInstance.donation(organization.id,
        { value: weiValue, from: this.senderAddress });
    const donation = await this._getDonationFromTransaction(transaction);
    console.log('OrganizationService->donation', donation);
    return donation;
}
```

src/app/services/organizations.service.ts

8.10 RESUMEN

- **Truffle boxes**: boilerplates.
- **Angular**: npm serve
- **MetaMask**: nodo ligero y wallet de Ethereum.
- **web3js**: librería de comunicación.

9 SUBIDA A ENTORNOS DE PRUEBA

9.1 FRONT

- Primero hacemos el build y luego desplegamos en un servidor web. Puede ser en **GitHub Pages**, u otro, pero siendo puristas habría que usar IPFS.

```
# empaquetar
ng build --prod
```

9.2 INFURA

- **Infura**: simplifica el despliegue de ĐApps en redes de prueba y en la red principal.
- Te creas un usuario y te facilita un **API Key**.

9.3 RINKEBY

- **Rinkeby**: red de pruebas para probar ĐApps.
- Mediante **faucets** consigues ETH de la red.

9.4 TRUFFLE.JS

```
var HDWalletProvider = require("truffle-hdwallet-provider");
module.exports = {
  networks: {
    rinkeby: {
      provider: function () {
        let provider = new HDWalletProvider(
          walletMmnemonic,
          "https://rinkeby.infura.io/v3/" + apiKey);
        return provider;
      },
      gas: 7000001,
      network_id: 4
    }
  }
};
```

truffle.js

9.5 MIGRAR A RINKEBY

- Para migrar a Rinkeby tenemos que usar la **apiKey** de Infura y una **walletMmnemonic** de una wallet con saldo suficiente en Rinkeby y ejecutar:

```
truffle migrate --network rinkeby
```

9.6 RESUMEN

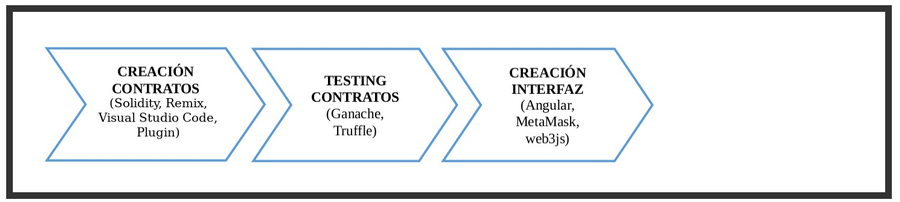
- **Front**: desplegar como siempre o mejor en IPFS.
- **Infura**: simplifica el despliegue de ĐApps.
- **Rinkeby**: red de pruebas para probar ĐApps.
- **truffle.js**: apiKey y walletMmnemonic.
- **Migrar**: truffle migrate --network rinkeby

10 RESUMEN Y CONCLUSIONES

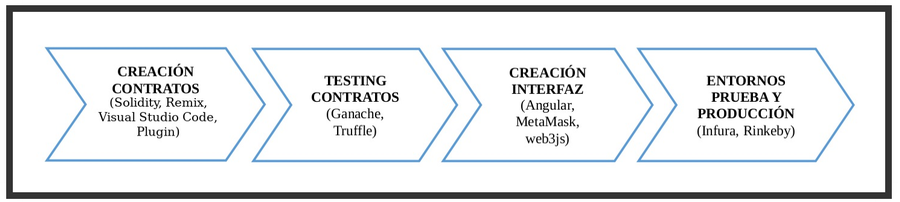
10.1 ¿CREACIÓN CONTRATOS INTELIGENTES?

10.2 ¿TESTING CONTRATOS INTELIGENTES?

10.3 ¿CREACIÓN INTERFAZ DE USUARIO?



10.4 ¿SUBIDA A ENTORNOS DE PRUEBA?



10.5 ¿CONCLUSIONES?

- Tecnologías nuevas, pero ya se pueden empezar a implantar los primeros proyectos en producción.
- Tecnologías muy disruptivas por la descentralización y por la trazabilidad de los datos.
- Problemas de escalabilidad y de volatilidad de precios.

11 ACERCA DE

11.1 LICENCIA

Creative Commons Reconocimiento-CompartirIgual 3.0

11.2 FUENTES

github.com/asanzdiego/commit-conf-taller-blockchain

11.3 SLIDES

Las slides están hechas con **MarkdownSlides**.

12 PREGUNTAS

13 GRACIAS