CÓMO DESARROLLAR APLICACIONES DESCENTRALIZADAS EN LA BLOCKCHAIN DE ETHEREUM

ADOLFO SANZ DE DIEGO

CODEMOTION 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

2.2 DISCLAIMER

- Mi intención con esta charla es dar unas pincelas para los que queréis empezar a desarrollar aplicaciones **ahorraros muchas horas de trabajo** de investigación.
- Espero haberlo conseguido.

3 TEORÍA

3.1 ¿QUÉ ES BITCOIN?

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

3.2 ¿QUÉ ES BLOCKCHAIN?

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

3.3 ¿QUÉ ES EL MINADO?

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

3.4 ¿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.

3.5 ¿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.

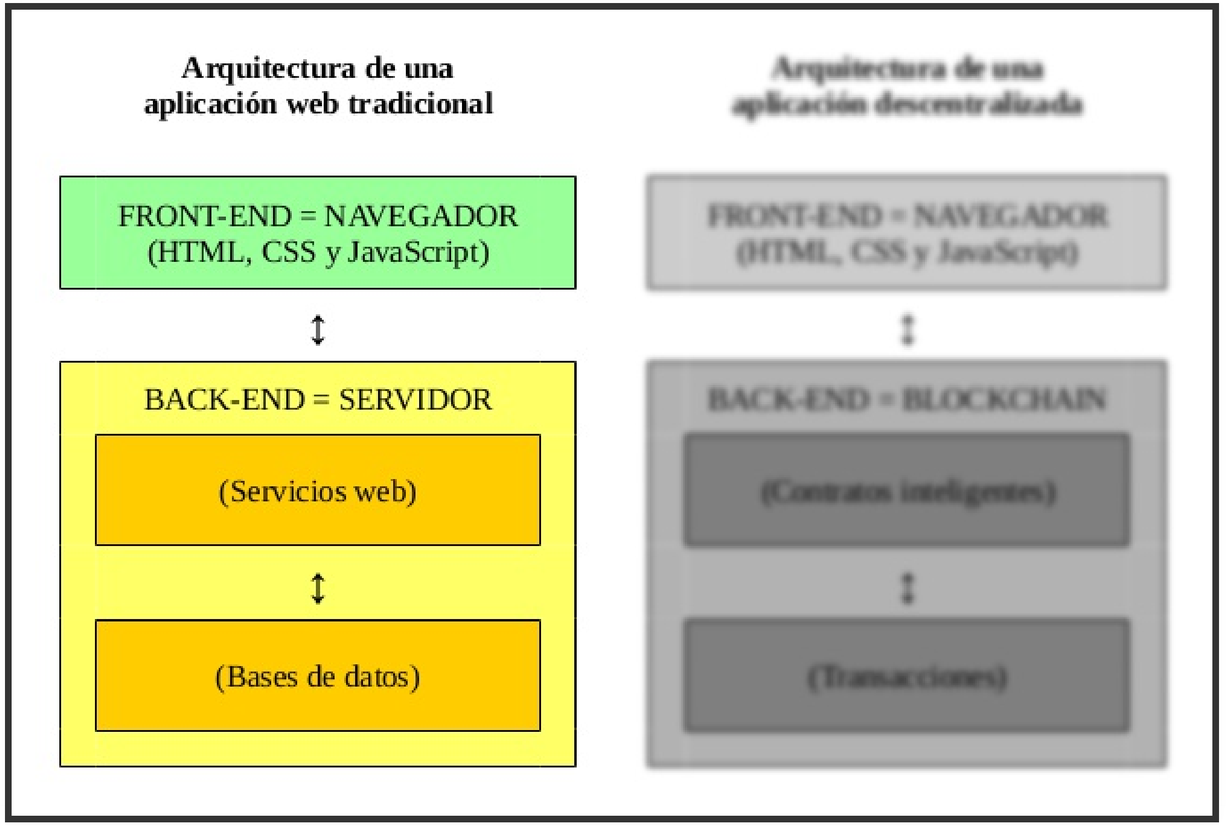
3.6 ¿QUÉ ES ETHEREUM?

- 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.

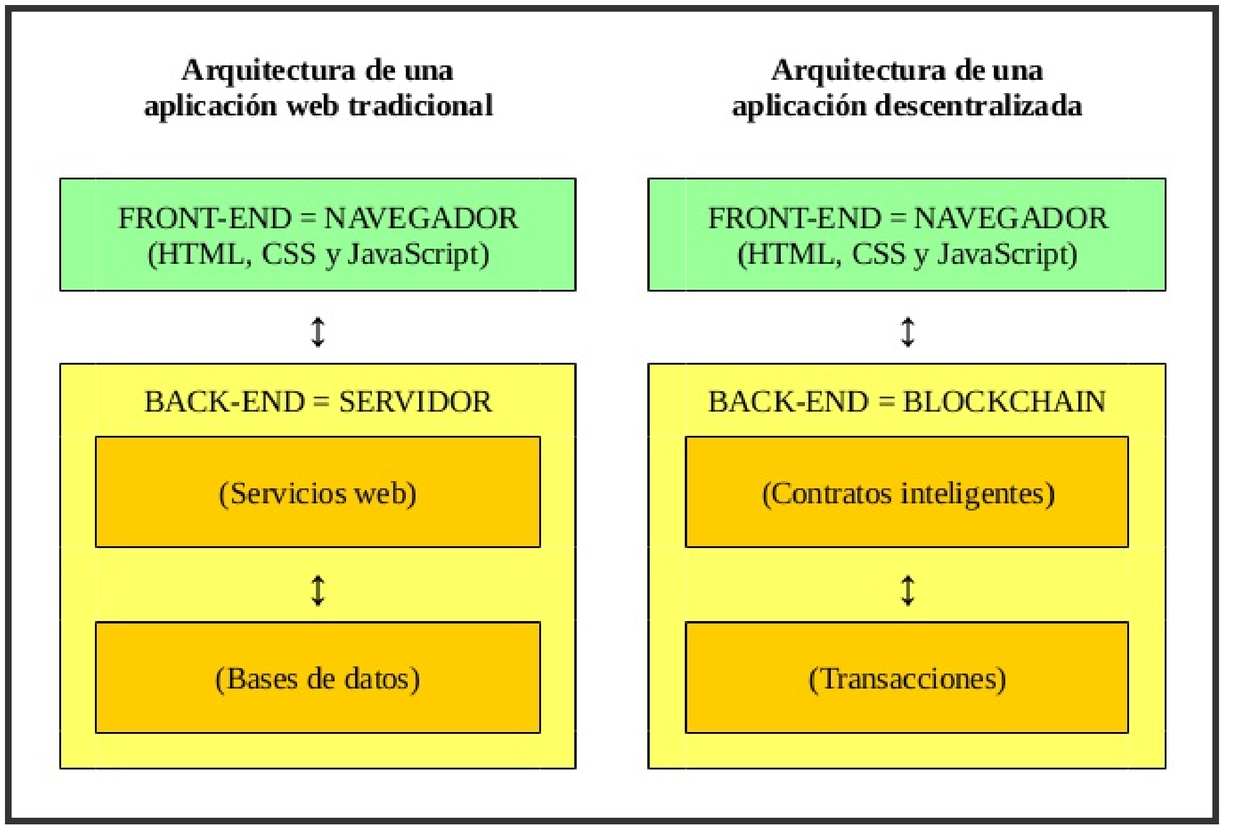
3.7 ¿QUÉ ES UN SMART CONTRACT?

- 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.

3.8 ¿APPS TRADICIONALES?

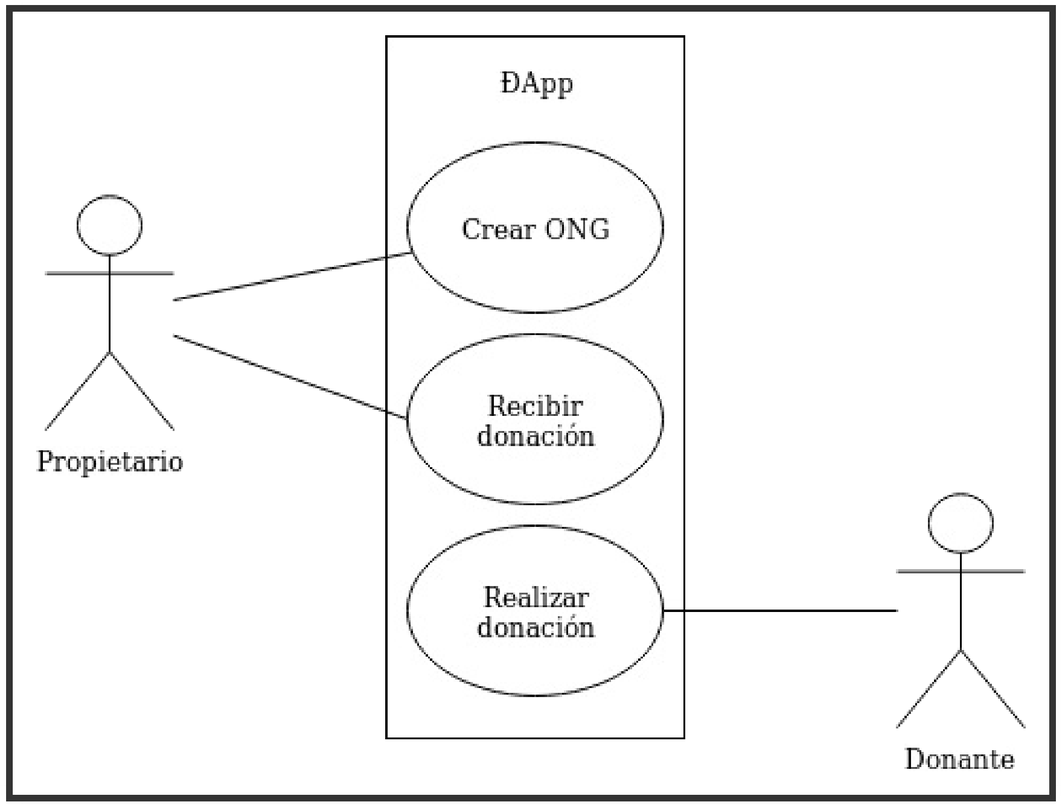


3.9 ¿APPS DESCENTRALIZADAS?



4 CREACIÓN SMART CONTRACTS

4.1 ĐAPP DE DONACIONES A ONG



4.2 ¿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

4.3 ¿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**.

4.4 EJEMPLO

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

// Array
Organization[] public organizations;

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

4.5 ¿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.

4.6 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");
    _;
}
```

4.7 ¿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.

4.8 ¿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.

4.9 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

4.10 ¿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.

5 TESTING SMART CONTRACTS

5.1 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)"
```

5.2 TRUFFLE

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

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

5.3 COMPILAR

- Una vez creado los smarts contracts hay que compilarlos:

```
# compilar
truffle compile
```

5.4 MIGRAR

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

```
# migrar
truffle migrate
```

5.5 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

5.6 TESTEAR

- Una vez creados los tests los lanzamos:

```
# testear
truffle test
```

6 CREACIÓN INTERFAZ USUARIO

6.1 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 :-)

6.2 ANGULAR

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

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

6.3 METAMASK

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

6.4 WEB3JS

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

6.5 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);
    });
}
```

6.6 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

6.7 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;
}
```

6.8 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

6.9 NAVEGADOR

- Una vez desarrollada la UI podemos lanzar Angular y ver la aplicación en http://localhost:4200/ ejecutando:

```
# ejecutar
ng serve
```

7 SUBIDA A ENTORNOS DE PRUEBA

7.1 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.
- Con Angular ejecutar:

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

7.2 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**.

7.3 RINKEBY

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

7.4 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

7.5 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
```

7.6 DEMO (RINKEBY)

https://nongovernmentalorganizations.github.io

8 RESUMEN Y CONCLUSIONES

8.1 LINKS INTERESANTES

Mastering Bitcoing

Los videotutoriales de Nicolas Palacios sobre Solidity

El tutorial interactivo de CryptoZombies

El tutorial “Ethereum Overview”

El tutorial "Ethereum Pet Shop"

El tutorial "Truffle testing Smart Contracts"

Pedir ayuda en el canal de Truffle del chat de Gitter

8.2 RESUMEN TEORÍA

- **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.

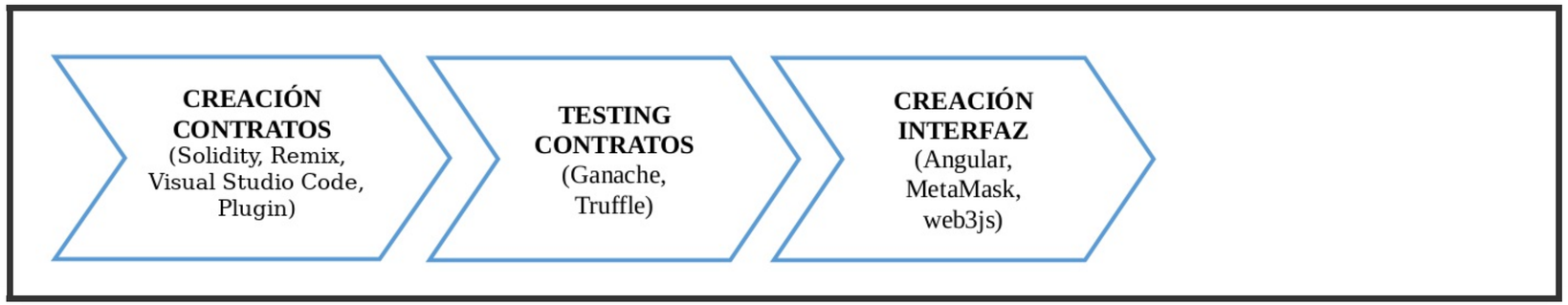
8.3 RESUMEN PRÁCTICA

Creación de Smart Contracts

8.4 RESUMEN PRÁCTICA

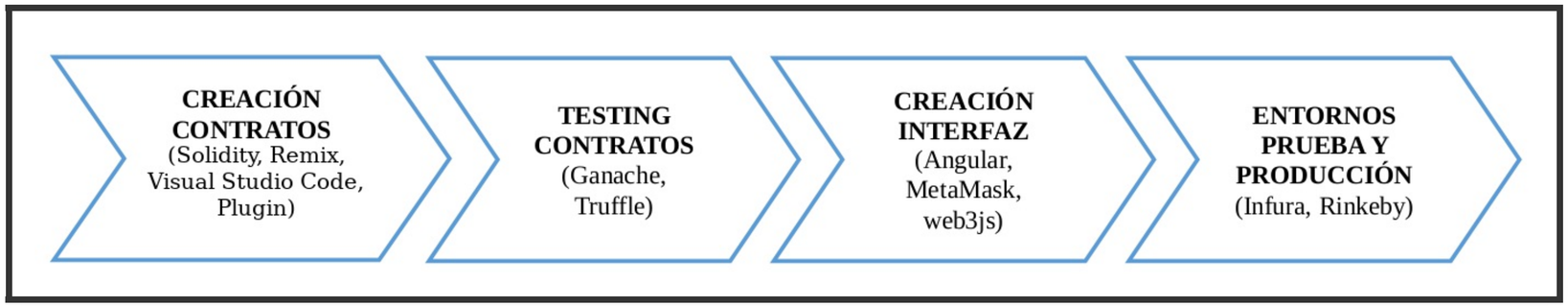
Testing de Smart Contracts

8.5 RESUMEN PRÁCTICA



Creación de Interfaz de Usuario

8.6 RESUMEN PRÁCTICA



Subida a entornos de prueba y producción

8.7 ¿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.

9 ACERCA DE

9.1 LICENCIA

Creative Commons Reconocimiento-CompartirIgual 3.0

9.2 FUENTES

github.com/asanzdiego/codemotion-charla-blockchain

9.3 SLIDES

Las slides están hechas con **MarkdownSlides**.

10 PREGUNTAS

11 GRACIAS