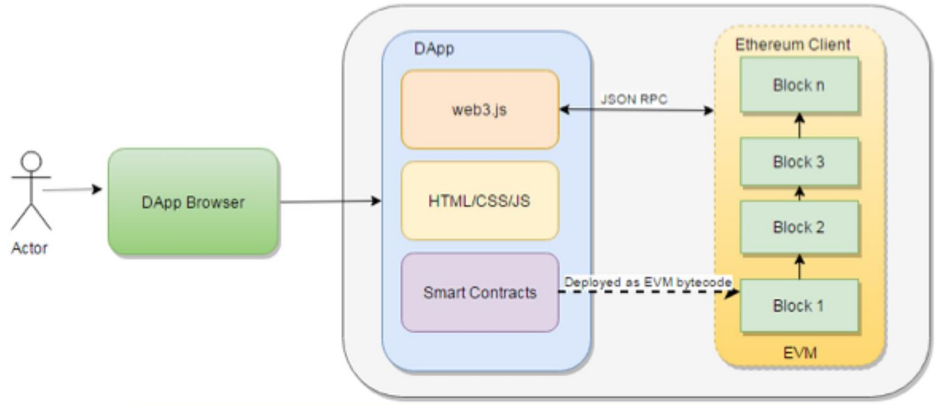
Blockchain

blockchain@alumnos.exa.unicen.edu.ar

Dapps

Development

Franco Daniel Berdun
franco.berdun@isistan.unicen.edu.ar

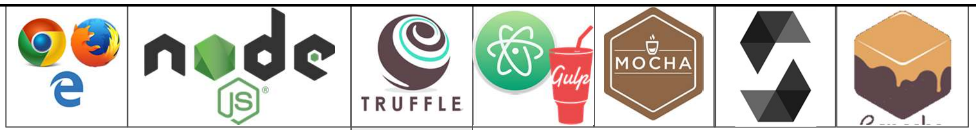


Aplicación — Front-end / Back-end

Lenguaje de Alto Nivel

Sistema Operativo / Compilador — Front-end / Back-end — [Parsing; Type checking; Code generation; Optimization]

Arquitectura del conjunto de Instrucciones



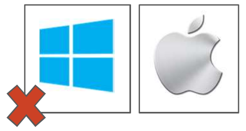
project

code

test

compile

migrate



Ubuntu

```
# Make sure Ubuntu is up to date
sudo apt-get update -y && sudo apt-get upgrade -y
# install nvm  https://github.com/creationix/nvm#install-script
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh | bash
# restart bash to enable nvm (saves you restarting your terminal)
exec bash
# install node and our npm packages
nvm install node
npm install -g truffle ganache-cli
# build essentials gives GCC, etc. that other tools may need. Takes a while to install
sudo apt install build-essential python -y
```

Windows 10

```
#Power Shell (run as Admin)
#install nodejs
#install git
$ npm install -g npm
$ npm install -g -production windows-build-tools
$ npm install -g truffle
npm install -g truffle ganache-cli
```

MacOS

```
#Install HomeBrew
/usr/bin/ruby -e “$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

#Open Mac app store and search for XCode
#Click on Xcode search item from the list and click on Install
#If it’s already installed on your machine then update or skip this step
#To install Xcode command line tools, run following command
xcode-select – install

#To download Ganache framework, open below link in the browser, http://truffleframework.com/ganache/

brew install node
npm install -f truffle

#To install Atom text editor, open below url in the browser https://github.com/atom/atom

apm install language-ethereum
```

Editor environment

Step 1: Install

EtherAtom https://github.com/0mkara/etheratom

Sublime Text https://packagecontrol.io/packages/Ethereum

Visual Studio code https://code.visualstudio.com

both are great tool for editing Solidity smart contracts, and is available on Windows, Mac & Linux. There is a great plugin that enables Syntax highlighting, snippets, and compiling of the current contract (if you aren't using an external tool)

Step 2: Extensions: VS Go into the extensions section, then install these plugins:

- Solidity
- Material Icon Theme

https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity
https://github.com/juanfranblanco/vscode-solidity/

This configuration works really well with Truffle

Workshop Schedule

Smart Contract

Solidity

https://ethereumbuilders.gitbooks.io/guide/content/en/solidity_tutorials.html

Blockchain
blockchain@alumnos.exa.unicen.edu.ar

Pizza Test.

```
npm install gulp-cli -g
npm install gulp -D          //https://gulpjs.com
gulp compile
gulp watch
npm install --global mocha
npm install mocha            //https://mochajs.org
mocha
open ganache                 //change address
ganache-cli
step-by-step
```

Code

```
pragma solidity ^0.4.24;
contract ChuckETHCheese {
   uint public pizzas;
   bool public isPizzaHot;
   address owner;
   mapping(address => uint) public tokenBalances;
   mapping(address => bool) public playingStatus;
   constructor(uint _pizzas) public {
       pizzas = _pizzas;
       owner = msg.sender;
              }
   function setIsPizzaHot(bool _isPizzaHot) public {
       isPizzaHot = _isPizzaHot;
              }
   function purchaseTokens() public payable {
       uint tokens = msg.value;
       tokenBalances[msg.sender] += tokens;
              }
   function playGame() public {
       require(tokenBalances[msg.sender] > 0);
       tokenBalances[msg.sender] -= 1;
       playingStatus[msg.sender] = true;
              }
   function awardWinner(address winner) public onlyOwner {
       winner.transfer(1);
              }
   modifier onlyOwner {
       require(msg.sender == owner);
       _;
              }
}
```

Block, Msg and Tx properties (Global)

- `block.blockhash(uint blockNumber) returns (bytes32)`: hash of a given block - works for last 256, excluding current
- `block.difficulty` (`uint`): returns the difficulty of the current block
- `block.number` (`uint`): returns the current block number
- `block.timestamp` (`uint`): returns the timestamp of the current block in the form of seconds following universal Unix time
- `msg.data` (`bytes`): data sent in the transaction(calldata)
- `msg.gas` (`uint`): returns the remaining gas
- `msg.sender` (`address`): returns sender of the current call
- `msg.sig` (`bytes4`): returns the first four bytes of the data sent in the transaction (i.e. the identifier of the function)
- `msg.value` (`uint`): returns the Wei number sent with the call
- `now` (`uint`): returns the timestamp of the current block (alias of `block.timestamp`)
- `tx.gasprice` (`uint`): returns the gas price of the transaction
- `tx.origin` (`address`): returns the original issuer of the transaction
- **http://solidity.readthedocs.io/en/latest/units-and-global-variables.html#index-4**

Workshop Schedule

Safe Development

Project Lifecycle

Blockchain
blockchain@alumnos.exa.unicen.edu.ar

Mandatory Homework

- Ethereum
  Create own Crypto-Project
- Solidity
  Contract Oriented Programming (COP)
- Truffle
  a. Creating a project
  b. Testing
  c. Compiling
  d. Migrating
- WebApp
  a. Interacting

Creating a Project

Code

Create a new directory for your Truffle project:

```
mkdir MetaCoin
cd MetaCoin
```

Download ("unbox") the MetaCoin box:

```
truffle unbox metacoin
```

Note: To create a bare Truffle project with no smart contracts included, use `truffle init`.

Once this operation is completed, you'll now have a project structure with the following items:

- `contracts/`: Directory for Solidity contracts
- `migrations/`: Directory for scriptable deployment files
- `test/`: Directory for test files for testing your application and contracts
- `truffle.js`: Truffle configuration file

Exploring the Project

Code

1. Open the `contracts/MetaCoin.sol` file in a text editor. This is a smart contract (written in Solidity) that creates a MetaCoin token. Note that this also references another Solidity file `contracts/ConvertLib.sol` in the same directory.
2. Open the `contracts/Migrations.sol` file. This is a separate Solidity file that manages and updates the status of your deployed smart contract. This file comes with every Truffle project, and is usually not edited.
3. Open the `migrations/1_initial_deployment.js` file. This file is the migration (deployment) script for the `Migrations` contract found in the `Migrations.sol` file.
4. Open the `migrations/2_deploy_contracts.js` file. This file is the migration script for the `MetaCoin` contract. (Migration scripts are run in order, so the file beginning with `2` will be run after the file beginning with `1`.)
5. Open the `test/TestMetacoin.sol` file. This is a test file written in Solidity which ensures that your contract is working as expected.
6. Open the `test/metacoin.js` file. This is a test file written in JavaScript which performs a similar function to the Solidity test above.
7. Open the `truffle.js` file. This is the Truffle configuration file, for setting network information and other project-related settings. The file is blank, but this is okay, as we'll be using a Truffle command that has some defaults built-in.

Testing

Code

Truffle comes standard with an automated testing framework to make testing your contracts a breeze: `truffle test`. This framework lets you write simple and manageable tests in two different ways:

- In Javascript, for exercising your contracts from the outside world, just like your application.
- In Solidity, for exercising your contracts in advanced, bare-to-the-metal scenarios.

Both styles of tests have their advantages and drawbacks. See the next two sections for a discussion of each one.

On a terminal, run the Solidity test:

```
  truffle test TestMetacoin.sol
```

You will see the following output

```
   TestMetacoin
      √ testInitialBalanceUsingDeployedContract (71ms)
      √ testInitialBalanceWithNewMetaCoin (59ms)
    2 passing (794ms)
```

These tree tests were run against the contract, with descriptions displayed on what the tests are supposed to do.

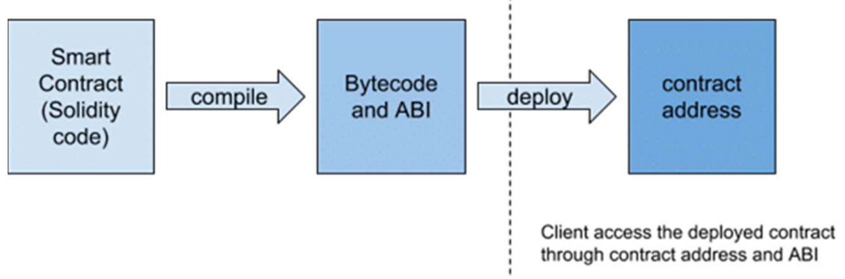
Run the JavaScript test:

```
truffle test metacoin.js
```

You will see the following output

```
 Contract: MetaCoin
    √ should put 10000 MetaCoin in the first account
    √ should call a function that depends on a linked
library (40ms)
    √ should send coin correctly (129ms)
  3 passing (255ms)
```

Compiling



Client access the deployed contract through contract address and ABI

Compile the smart contracts:

```
truffle compile
```

You will see the following output:

```
Compiling .\contracts\ConvertLib.sol...
Compiling .\contracts\MetaCoin.sol...
Compiling .\contracts\Migrations.sol...

Writing artifacts to .\build\contracts
```

Migrating

Code

Ganache is a desktop application to launch your personal blockchain.

it requires editing the Truffle configuration file to point to the Ganache instance.

1. Download and install Ganache.
2. Open `truffle.js` in a text editor. Replace the content with the following:

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*"
    }
  }
};
```

On the terminal, migrate the contract to the blockchain created by Ganache: `truffle migrate`

Interacting with the Contract

To interact with the contract, you can use the Truffle console. The Truffle console is similar to Truffle Develop, except it connects to an existing blockchain (in this case, the one generated by Ganache).

```
truffle console
```

Note: We're using `web3.eth.accounts[]` in these examples, which is an array of all the accounts generated by the mnemonic. So, given the addresses generated by our mnemonic above, specifying `web3.eth.accounts[0]` is equivalent to the address `0x########################################`.

- Check the metacoin balance of the account that deployed the contract:

```
MetaCoin.deployed().then(function(instance){return instance.getBalance(web3.eth.accounts[0]);}).then(function(value){return value.toNumber()});
```

- See how much ether that balance is worth (and note that the contract defines a metacoin to be worth 2 ether):

```
MetaCoin.deployed().then(function(instance){return instance.getBalanceInEth(web3.eth.accounts[0]);}).then(function(value){return value.toNumber()});
```

- Transfer some metacoin from one account to another:

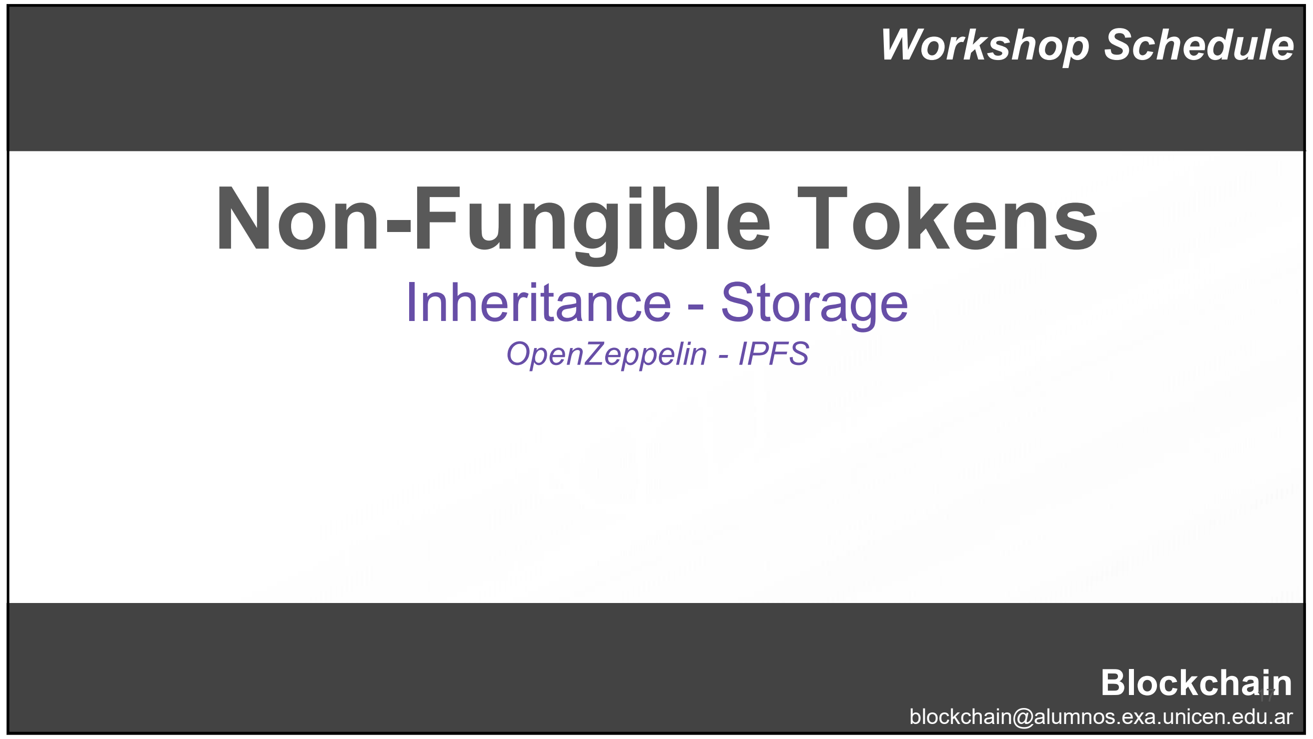
```
MetaCoin.deployed().then(function(instance){return instance.sendCoin(web3.eth.accounts[1], 500);});
```

- Check the balance of the account that *received* the metacoin:

```
MetaCoin.deployed().then(function(instance){return instance.getBalance(web3.eth.accounts[1]);}).then(function(value){return value.toNumber()});
```

- Check the balance of the account that *sent* the metacoin:

```
MetaCoin.deployed().then(function(instance){return instance.getBalance(web3.eth.accounts[0]);}).then(function(value){return value.toNumber()});
```



Workshop Schedule

Non-Fungible Tokens

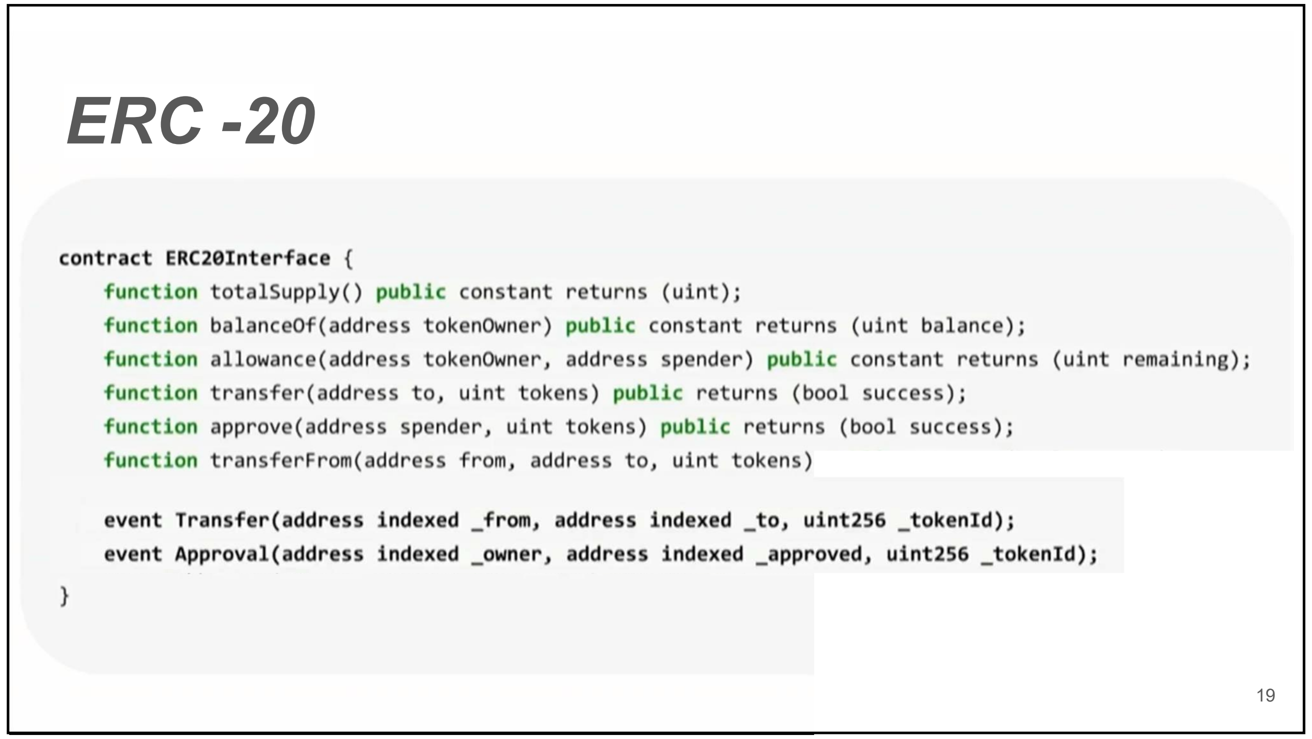
Inheritance - Storage

OpenZeppelin - IPFS

Blockchain

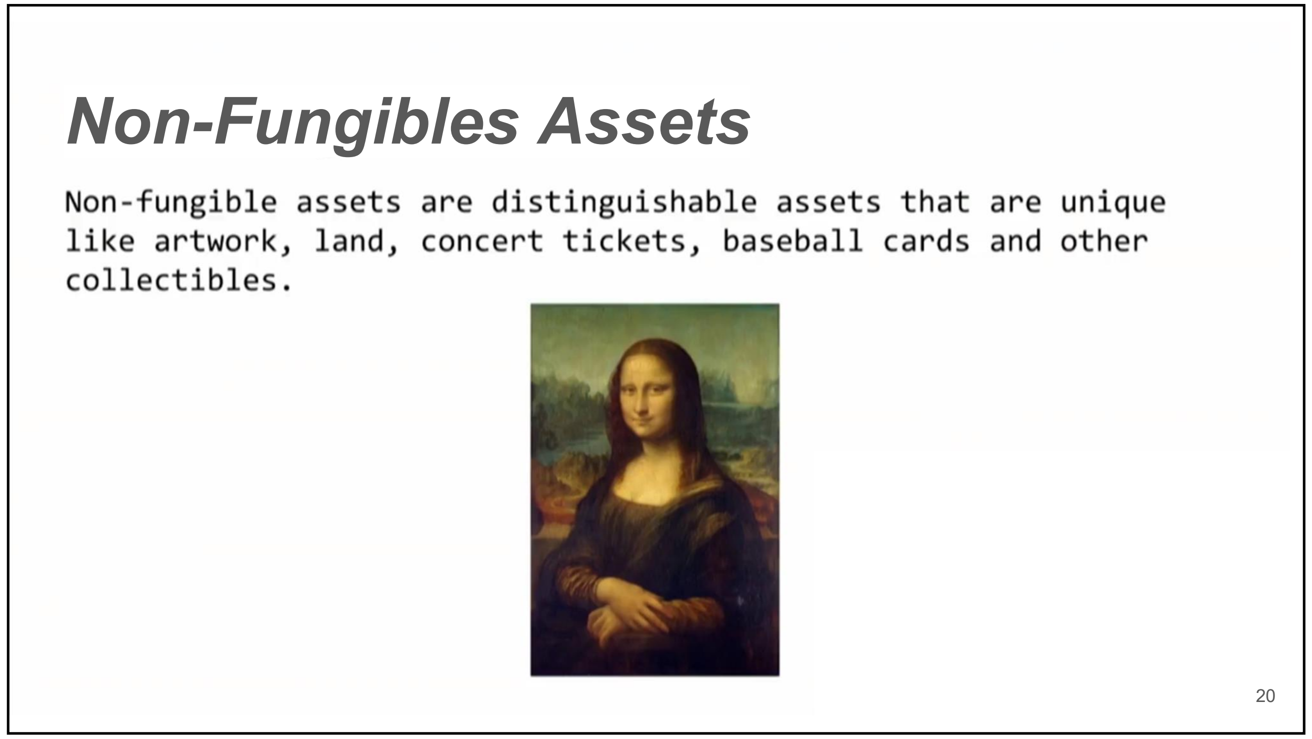
blockchain@alumnos.exa.unicen.edu.ar

Fungibles Assets IRL



ERC -20

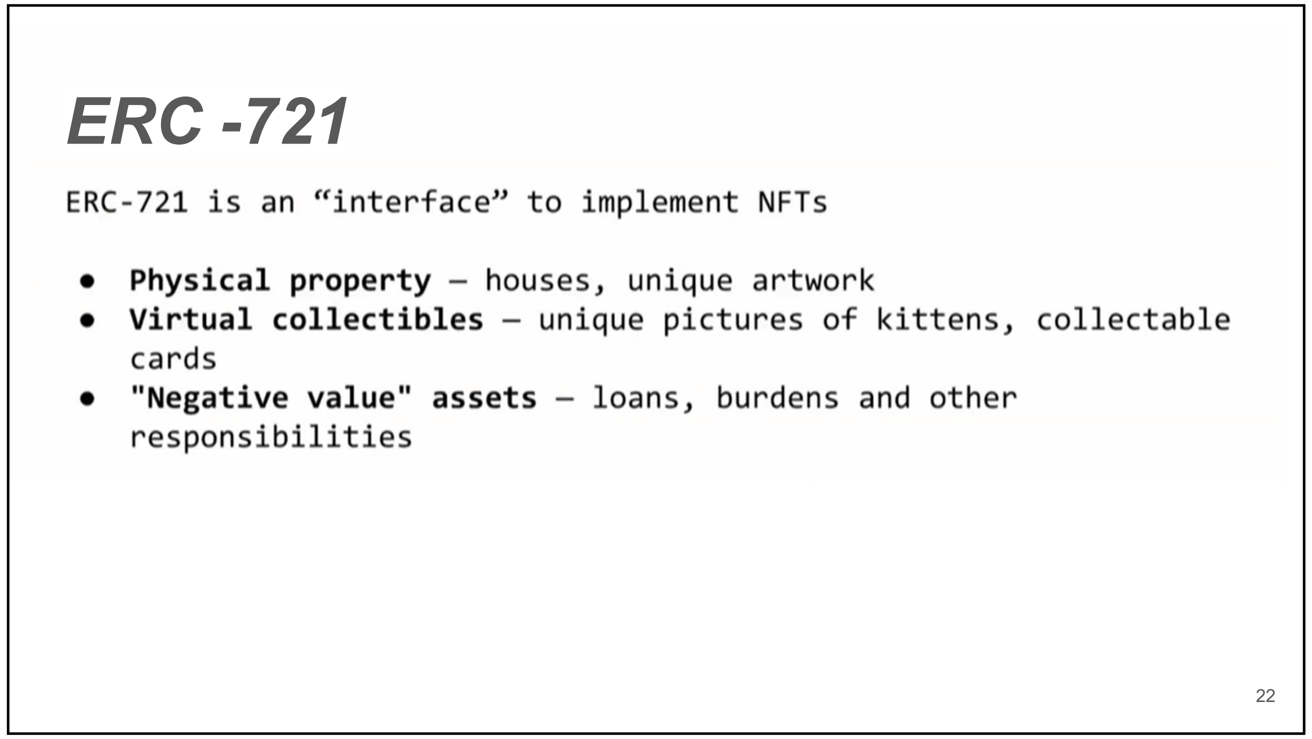
```
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens)

    event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
    event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
}
```



Non-Fungibles Assets

Non-fungible assets are distinguishable assets that are unique like artwork, land, concert tickets, baseball cards and other collectibles.

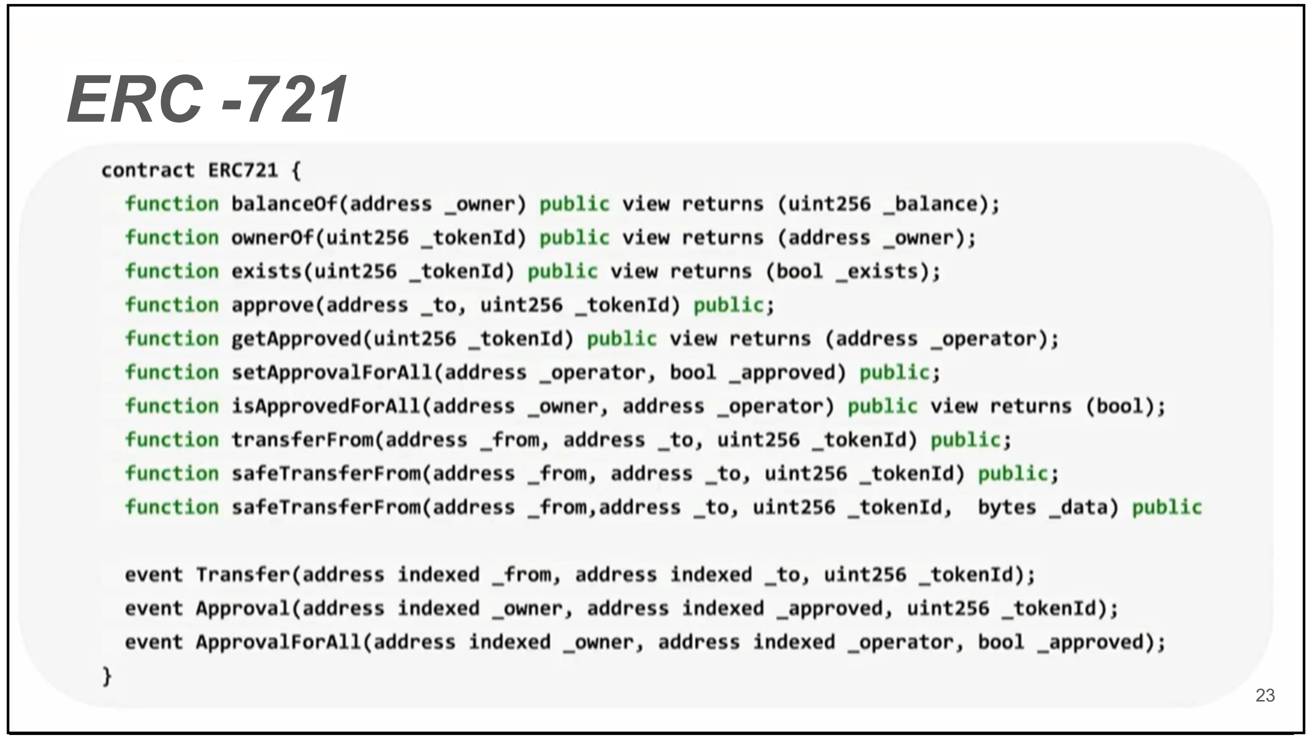
Non-Fungible Collectibles



ERC -721

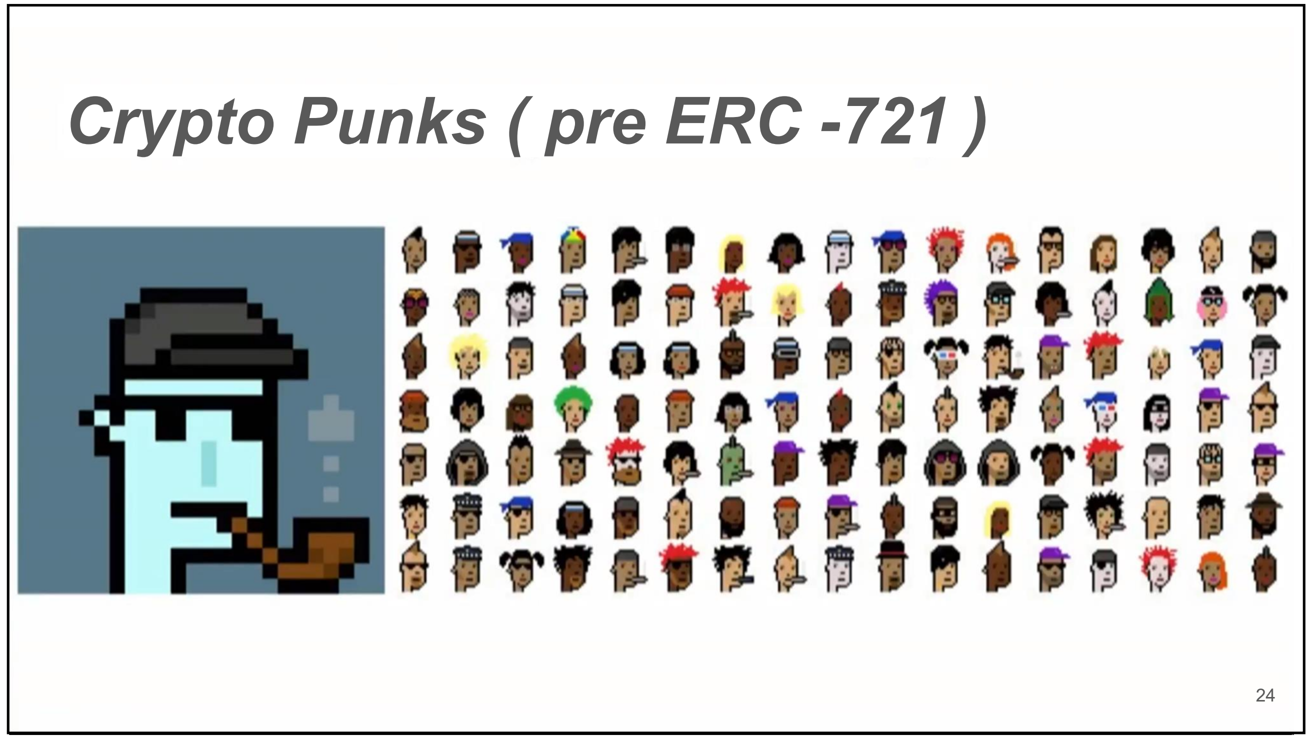
ERC-721 is an “interface” to implement NFTs

- **Physical property** – houses, unique artwork
- **Virtual collectibles** – unique pictures of kittens, collectable cards
- **"Negative value" assets** – loans, burdens and other responsibilities



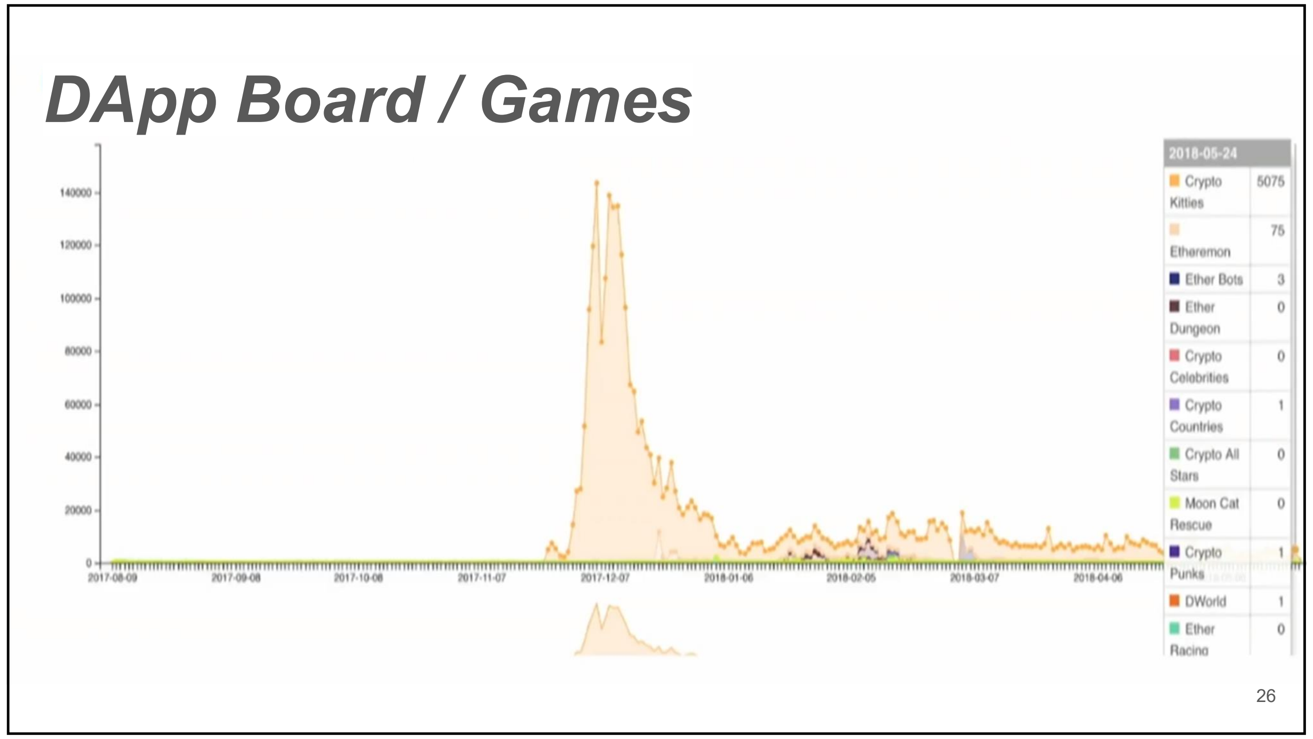
ERC -721

```
contract ERC721 {
  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

  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);
}
```

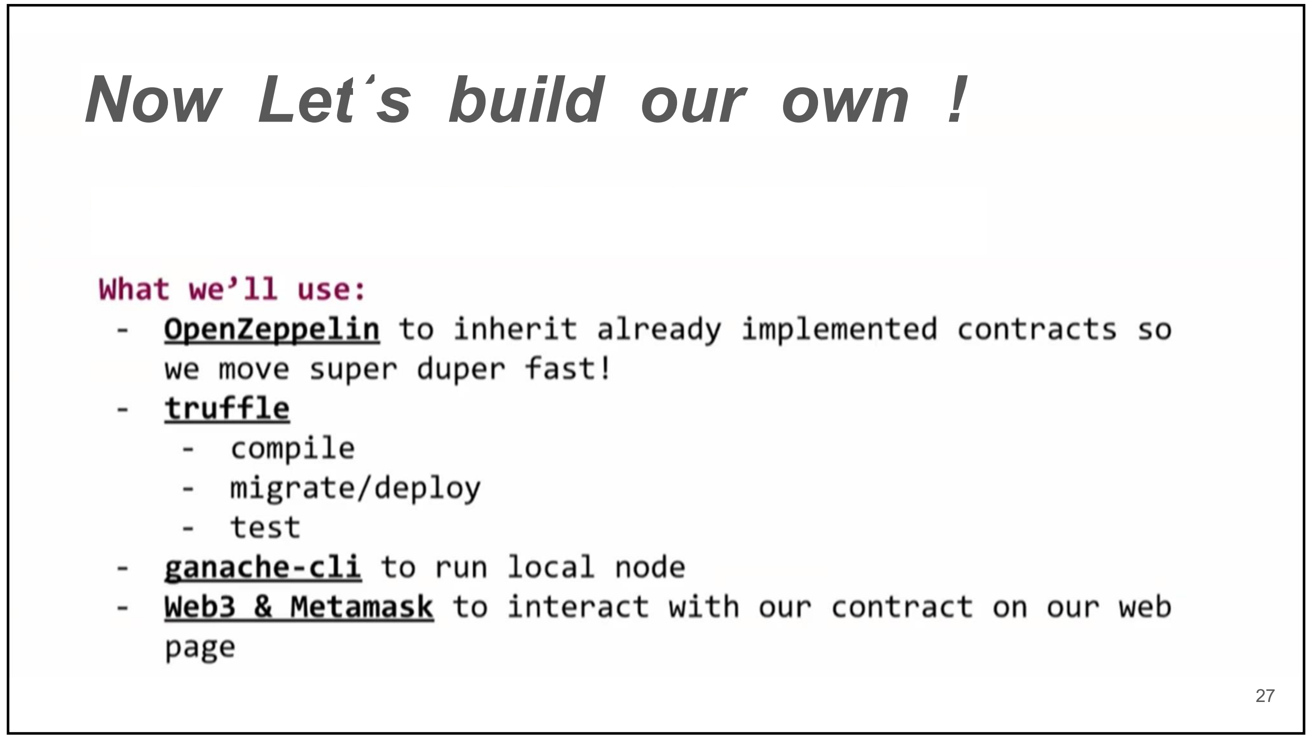


Crypto Punks (pre ERC -721)

Crypto Kitties (ERC-721)



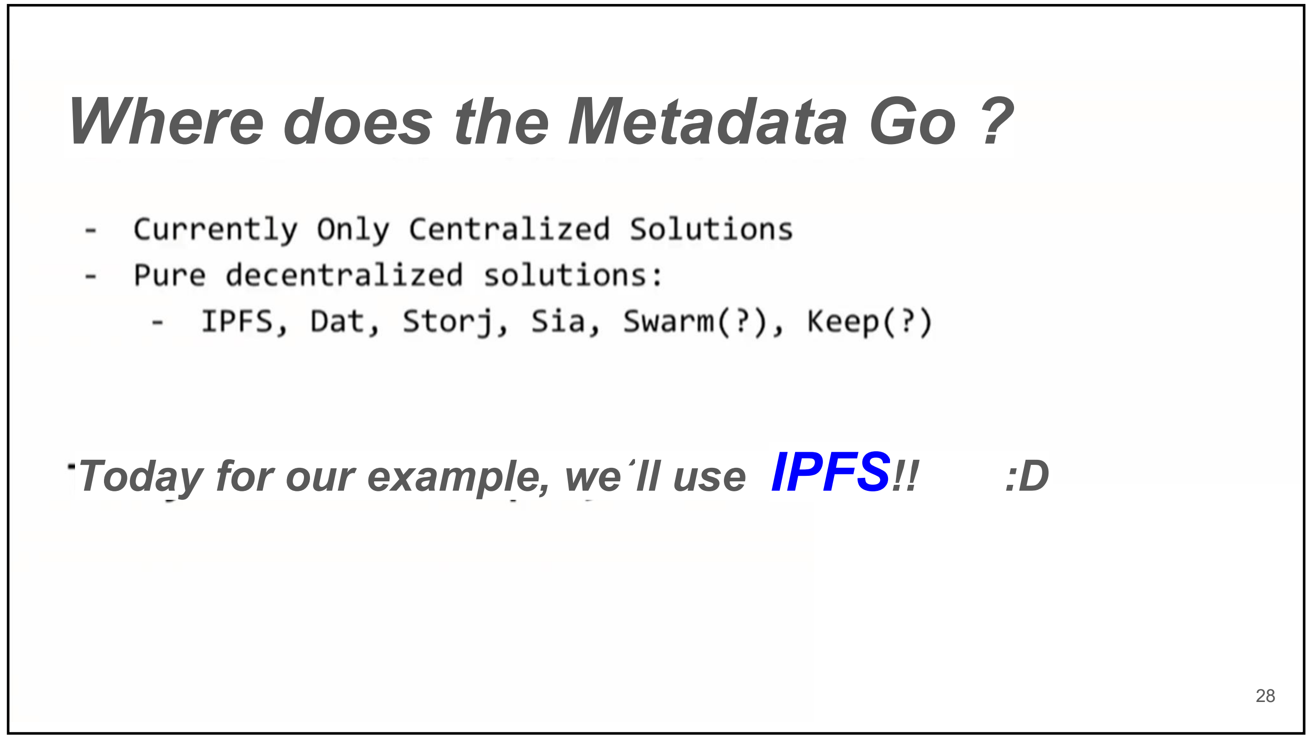
DApp Board / Games



Now Let´s build our own !

What we'll use:

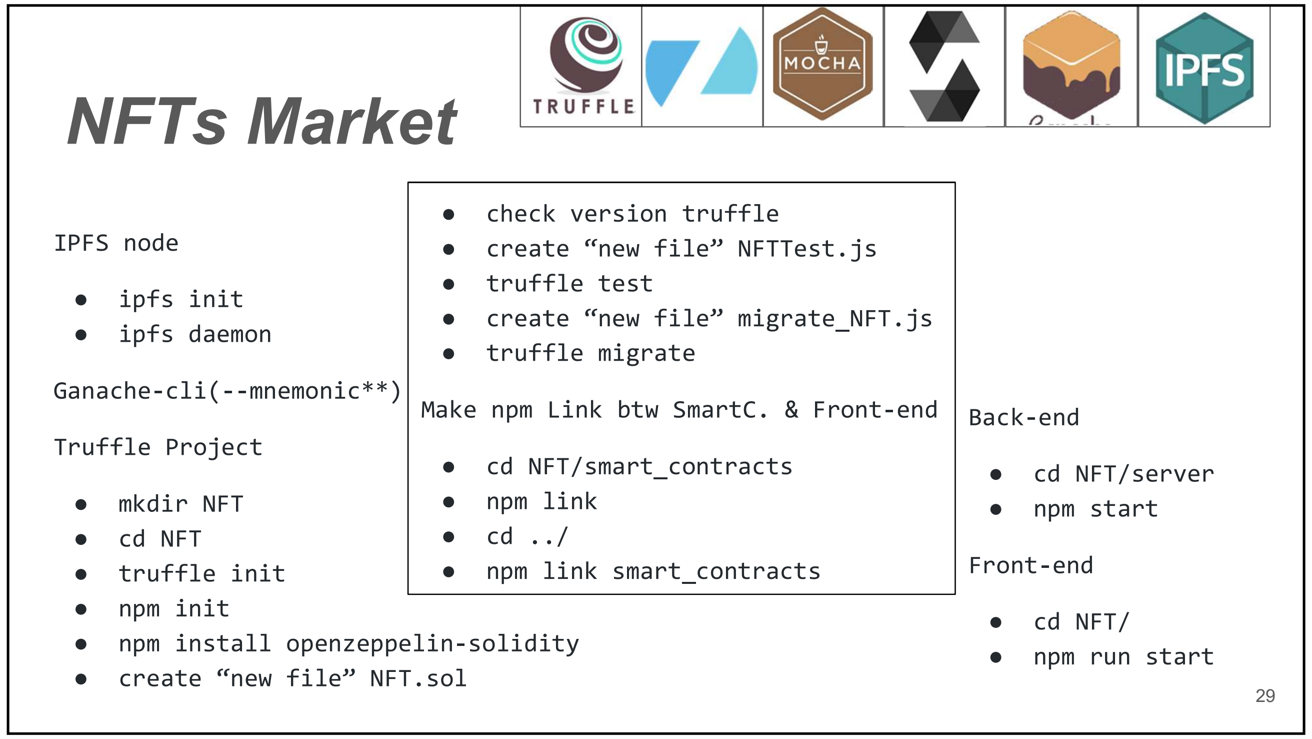
- **OpenZeppelin** to inherit already implemented contracts so we move super duper fast!
- **truffle**
  - compile
  - migrate/deploy
  - test
- **ganache-cli** to run local node
- **Web3 & Metamask** to interact with our contract on our web page



Where does the Metadata Go ?

- Currently Only Centralized Solutions
- Pure decentralized solutions:
  - IPFS, Dat, Storj, Sia, Swarm(?), Keep(?)

*Today for our example, we´ll use **IPFS**!!   :D*



NFTs Market

IPFS node

- ipfs init
- ipfs daemon

Ganache-cli(--mnemonic**)

Truffle Project

- mkdir NFT
- cd NFT
- truffle init
- npm init
- npm install openzeppelin-solidity
- create “new file” NFT.sol

- check version truffle
- create “new file” NFTTest.js
- truffle test
- create “new file” migrate_NFT.js
- truffle migrate

Make npm Link btw SmartC. & Front-end

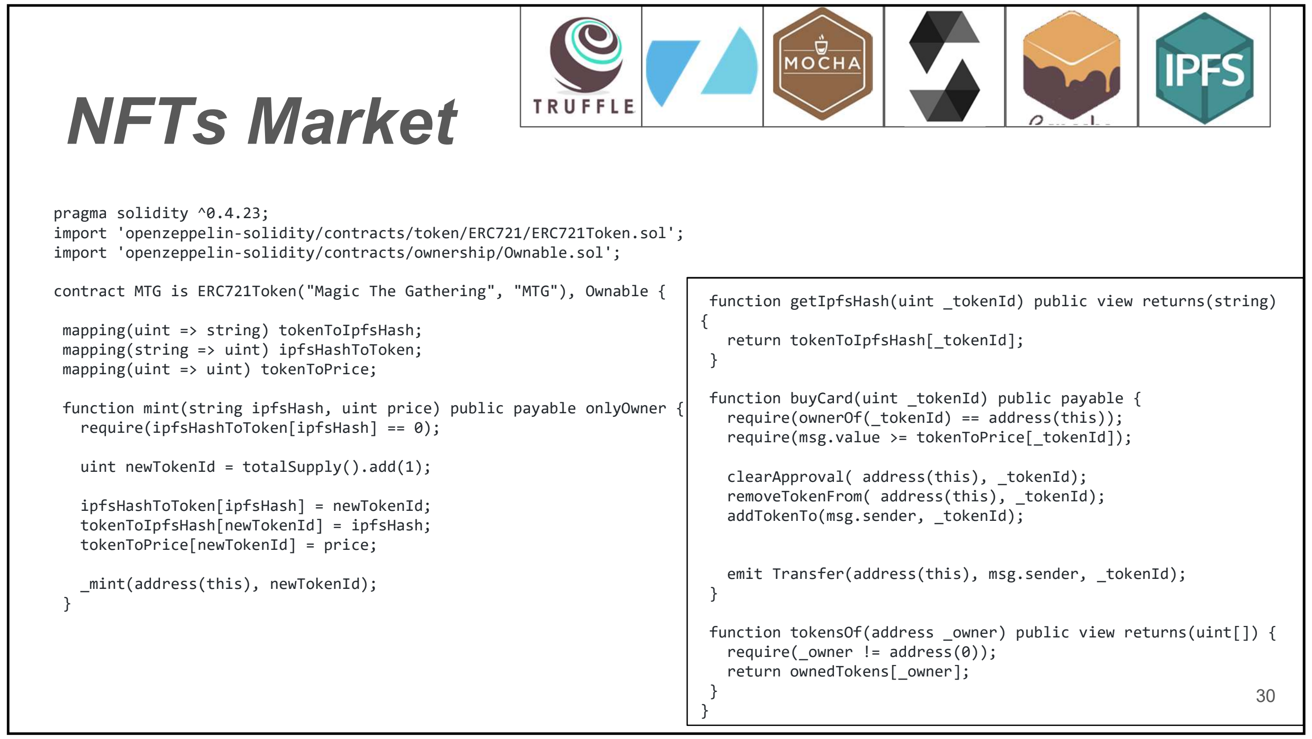
- cd NFT/smart_contracts
- npm link
- cd ../
- npm link smart_contracts

Back-end

- cd NFT/server
- npm start

Front-end

- cd NFT/
- npm run start



NFTs Market

```
pragma solidity ^0.4.23;
import 'openzeppelin-solidity/contracts/token/ERC721/ERC721Token.sol';
import 'openzeppelin-solidity/contracts/ownership/Ownable.sol';

contract MTG is ERC721Token("Magic The Gathering", "MTG"), Ownable {

 mapping(uint => string) tokenToIpfsHash;
 mapping(string => uint) ipfsHashToToken;
 mapping(uint => uint) tokenToPrice;

 function mint(string ipfsHash, uint price) public payable onlyOwner {
   require(ipfsHashToToken[ipfsHash] == 0);

   uint newTokenId = totalSupply().add(1);

   ipfsHashToToken[ipfsHash] = newTokenId;
   tokenToIpfsHash[newTokenId] = ipfsHash;
   tokenToPrice[newTokenId] = price;

   _mint(address(this), newTokenId);
 }
```

```
 function getIpfsHash(uint _tokenId) public view returns(string)
{
   return tokenToIpfsHash[_tokenId];
 }

 function buyCard(uint _tokenId) public payable {
   require(ownerOf(_tokenId) == address(this));
   require(msg.value >= tokenToPrice[_tokenId]);

   clearApproval( address(this), _tokenId);
   removeTokenFrom( address(this), _tokenId);
   addTokenTo(msg.sender, _tokenId);


   emit Transfer(address(this), msg.sender, _tokenId);
 }

 function tokensOf(address _owner) public view returns(uint[]) {
   require(_owner != address(0));
   return ownedTokens[_owner];
 }
}
```