

Proyecto Yoppen y Compra de Cromos

El Proyecto Yoppen busca innovar en el espacio de las criptomonedas, combinando tecnología blockchain avanzada con una visión de inclusión y comunidad. Este repositorio contiene el código fuente de la criptomoneda Yoppen (YPN), la estructura de su Oferta Inicial de Monedas (ICO) y el intercambio descentralizado (DEX) asociado.

Introducción

Yoppen es una criptomoneda diseñada para promover la conectividad y el apoyo dentro de su comunidad de usuarios. Inspirada en valores de cooperación y solidaridad, Yoppen busca ser más que una moneda digital: un movimiento hacia transacciones más justas y equitativas.

Para el nombre de la criptomoneda "Yoppen", inspirado en el vocabulario Selk'nam que significa "amigo" o "compañero", podríamos proponer un acrónimo y símbolo que reflejen tanto la herencia cultural como la naturaleza innovadora de la criptomoneda.

Acrónimo: YPN

"YPN" captura las iniciales de "Yoppen" y lo condensa en una forma fácil de recordar y usar en comunicaciones digitales. Este acrónimo es corto, lo que facilita su uso en el mercado de criptomonedas, donde la simplicidad y la facilidad de recordación son claves para la adopción y el reconocimiento.

Símbolo: Ⓨ

Como símbolo, propongo utilizar "Ⓨ", que es la letra "Y" dentro de un círculo. Esta representación visual hace referencia a la inicial de "Yoppen" y la inclusión en un círculo puede simbolizar la unidad, comunidad y la idea de globalidad, valores importantes para la criptomoneda inspirada en el concepto de amistad y cooperación. Además, el uso de un círculo es una práctica común en logotipos y símbolos para transmitir inclusión y totalidad.

Estas propuestas de acrónimo y símbolo buscan combinar la relevancia cultural del término "Yoppen" con la funcionalidad y el diseño requerido para una identificación clara en el ecosistema de criptomonedas.

Imagen de Yoppen



Características

- **Token ERC-20:** Yoppen utiliza el estándar ERC-20 para garantizar la compatibilidad con la amplia gama de infraestructuras Ethereum.
- **ICO Trustless:** Una ICO descentralizada permite a los inversores participar de manera segura y transparente.

- **DEX Integrado:** La plataforma incluye un DEX para facilitar el intercambio de Yoppen por otras criptomonedas sin intermediarios.

Tecnologías Utilizadas

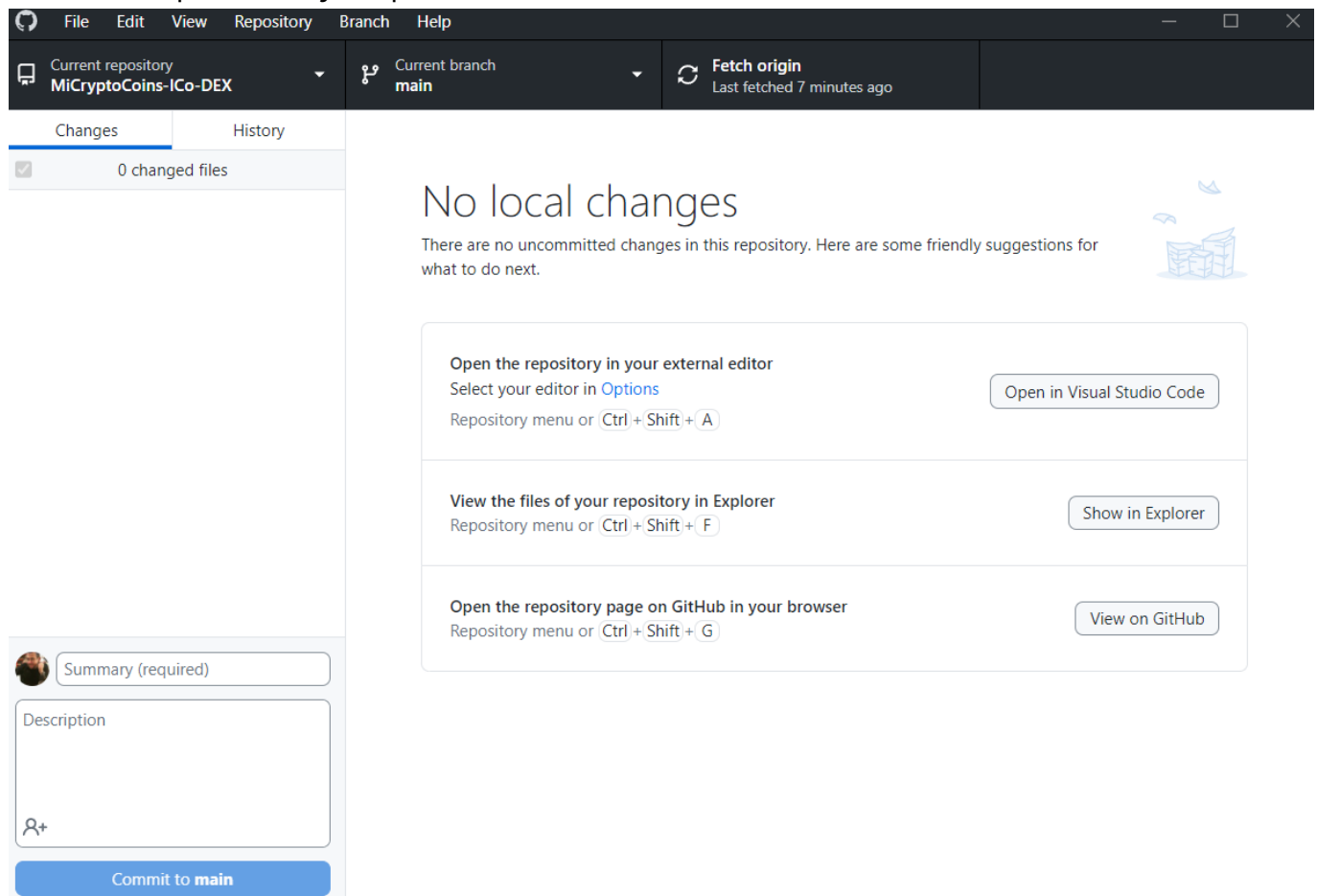
- [Solidity](#)
- [OpenZeppelin](#)
- [Remix IDE](#)
- [Web3.js](#)

Estructura del Repositorio

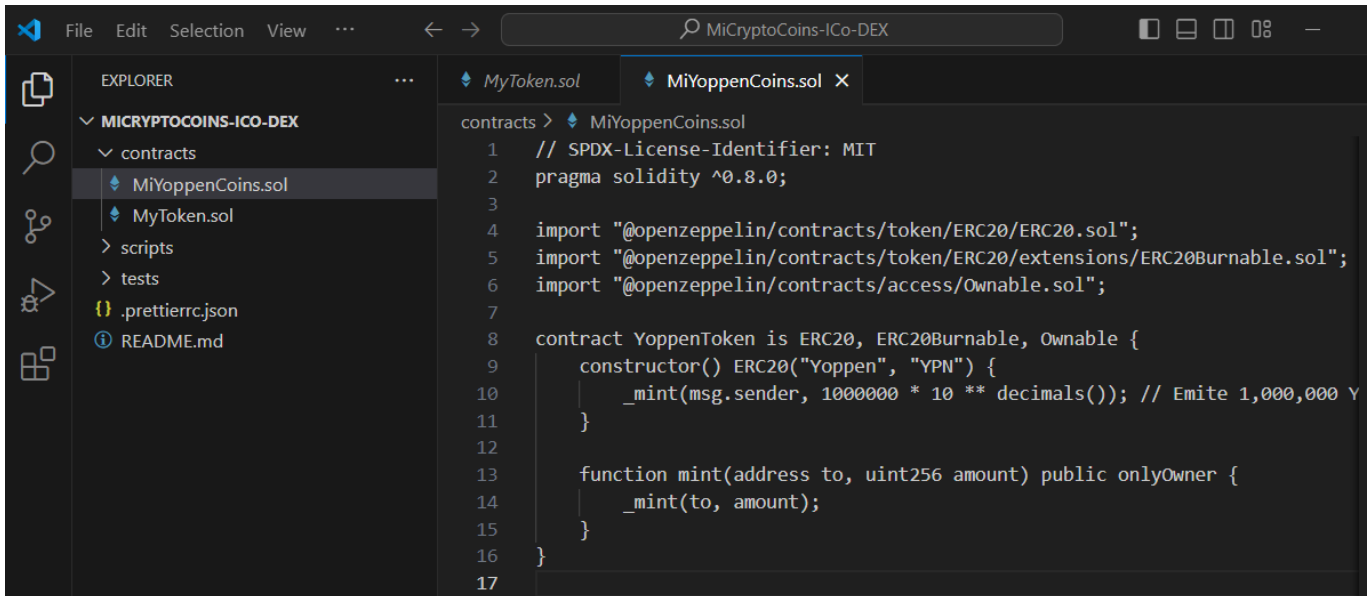
- [contracts/](#) - Contiene los contratos inteligentes de Yoppen, ICO y DEX.
- [migrations/](#) - Scripts de migración para desplegar los contratos en la red Ethereum.
- [tests/](#) - Pruebas automatizadas para validar la lógica de los contratos.

Desarrollo

Para comenzar a trabajar con el proyecto, clona este repositorio y sigue los pasos a continuación para instalar las dependencias y compilar los contratos.



Luego usando Visual Code Studio, creamos los códigos del proyecto



Contribuir

Si estás interesado en contribuir al proyecto Yoppen, por favor lee CONTRIBUTING.md para más información sobre cómo empezar.

Licencia

Este proyecto está bajo la licencia MIT. Ver LICENSE para más detalles.

Contacto

Para más información, por favor contacta a richard.poblete@hotmail.com.

Creación de Token ERC20 YoppenToken

Descripción

YoppenToken es un token ERC-20 desarrollado en Solidity que incorpora varias extensiones de la librería OpenZeppelin para proporcionar funcionalidades avanzadas como quema de tokens, pausabilidad y permisos. Este token ha sido diseñado para ser utilizado en la red de prueba Sepolia de Ethereum, facilitando una amplia gama de operaciones financieras descentralizadas.

Código fuente Solidity de Toekn ERC Yoppen

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Pausable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Permit.sol";
```

```
contract YoppenToken is ERC20, ERC20Burnable, ERC20Pausable, Ownable,
ERC20Permit {
    constructor(address initialOwner)
        ERC20("Yoppen", "YPN")
        Ownable(initialOwner)
        ERC20Permit("Yoppen") {
        _mint(msg.sender, 1000000 * 10 ** decimals()); // Emite 1,000,000
        YPN al desplegar el contrato
    }

    function pause() public onlyOwner {
        _pause();
    }

    function unpause() public onlyOwner {
        _unpause();
    }

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }

    // The following functions are overrides required by Solidity.

    function _update(address from, address to, uint256 value)
        internal
        override(ERC20, ERC20Pausable)
    {
        super._update(from, to, value);
    }
}
```

Características

- **ERC20:** Estándar básico para la creación de tokens intercambiables en la red Ethereum.
- **ERC20Burnable:** Permite a los titulares de tokens destruir (quemar) sus tokens, reduciendo el suministro total.
- **ERC20Pausable:** Introduce la capacidad de pausar y despausar las transferencias de tokens, lo que puede ser útil en situaciones de emergencia o mantenimiento.
- **Ownable:** Restringe ciertas acciones solo al propietario del contrato, como la emisión de nuevos tokens o la pausa del contrato.
- **ERC20Permit:** Permite a los usuarios realizar transacciones sin pagar gas, firmando una autorización.

Funciones Principales

- **constructor(address initialOwner):** Establece el nombre y símbolo del token, el propietario inicial y habilita los permisos según el estándar ERC20Permit. El constructor del contrato crea el token con el nombre "Yoppen" y el símbolo "YPN", y acuña inicialmente 1,000,000 de tokens para el creador del contrato. La función mint permite al propietario del contrato acuñar más tokens en el futuro.

- **pause():** Pausa todas las transferencias de tokens.
- **unpause():** Reanuda todas las transferencias de tokens.
- **mint(address to, uint256 amount):** Permite al propietario del contrato acuñar nuevos tokens a una dirección específica.
- **_update(address from, address to, uint256 value):** Sobrescribe las funciones de actualización de balances de tokens para ser compatibles con la pausabilidad.

Despliegue

El contrato **YoppenToken** fue creado utilizando Visual Studio Code, compilado y desplegado a través de Remix, integrado con MetaMask y utilizando una cuenta de la red de prueba Sepolia.

Licencia

Este proyecto está bajo la licencia MIT.

Resultado del despliegue en Renix Yoppen Token ERC20

- URL del Token Yoppen
<https://sepolia.etherscan.io/address/0x38bc18ae393a7e560f8c26c1490f06d0ee069b73>
- URL Contrato CromoMarket
<https://sepolia.etherscan.io/address/0x265d37eB5f8D9998cBA2E83Ba0C0Da6E9C5431f8>

CromoMarket Smart Contract

Descripción

El contrato inteligente CromoMarket permite la creación, compra y gestión de cromos utilizando el token ERC20 Yoppen. Este contrato gestiona la compra de cromos, verificando que el usuario tenga suficientes tokens Yoppen y transfiriendo la propiedad del cromo al comprador.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract CromoMarket {
    IERC20 public yoppenToken;
    address public owner;

    struct Cromo {
        uint id;
        string name;
        uint price;
        address owner;
    }

    mapping(uint => Cromo) public cromos;
```

```
uint public nextCromoId;

event CromoPurchased(uint cromoSId, address buyer);

constructor(address _yoppenTokenAddress) {
    yoppenToken = IERC20(_yoppenTokenAddress);
    owner = msg.sender;
}

function createCromo(string memory _name, uint _price) external {
    require(msg.sender == owner, "Only owner can create cromos");
    cromos[nextCromoId] = Cromo(nextCromoId, _name, _price,
address(0));
    nextCromoId++;
}

function buyCromo(uint _cromoId) external {
    Cromo storage cromoS = cromos[_cromoId];
    require(cromoS.price > 0, "Cromo does not exist");
    require(cromoS.owner == address(0), "Cromo already sold");
    require(yoppenToken.transferFrom(msg.sender, owner, cromoS.price),
"Token transfer failed");

    cromoS.owner = msg.sender;
    emit CromoPurchased(_cromoId, msg.sender);
}

function getCromo(uint _cromoId) external view returns (Cromo memory) {
    return cromos[_cromoId];
}
}
```

Funciones del Contrato

Constructor

```
constructor(address _yoppenTokenAddress) {
    yoppenToken = IERC20(_yoppenTokenAddress);
    owner = msg.sender;
}
```

Inicializa el contrato configurando la dirección del token Yoppen y estableciendo al desplegador del contrato como propietario.

```
function createCromo(string memory _name, uint _price) external {
    require(msg.sender == owner, "Only owner can create cromos");
    cromos[nextCromoId] = Cromo(nextCromoId, _name, _price, address(0));
    nextCromoId++;
}
```

Permite al propietario del contrato crear un nuevo cromo. Solo el propietario puede ejecutar esta función.

```
function buyCromo(uint _cromoId) external {
    Cromo storage cromo = cromos[_cromoId];
    require(cromo.price > 0, "Cromo does not exist");
    require(cromo.owner == address(0), "Cromo already sold");
    require(yoppenToken.transferFrom(msg.sender, owner, cromo.price),
    "Token transfer failed");

    cromo.owner = msg.sender;
    emit CromoPurchased(_cromoId, msg.sender);
}
```

Permite a un usuario comprar un cromo. Verifica que el cromo exista, que no haya sido vendido y que el comprador tenga suficientes tokens Yoppen.

```
function getCromo(uint _cromoId) external view returns (Cromo memory) {
    return cromos[_cromoId];
}
```

Retorna la información de un cromo específico.

```
event CromoPurchased(uint croMoId, address buyer);
```

Evento que se emite cuando un cromo es comprado exitosamente.

Uso

- Crear Cromo: El propietario del contrato puede crear nuevos cromos especificando el nombre y el precio.
- Comprar Cromo: Los usuarios pueden comprar cromos siempre que tengan suficientes tokens Yoppen.
- Consultar Cromo: Cualquier usuario puede consultar la información de un cromo utilizando su ID.

Este contrato utiliza la interfaz IERC20 de OpenZeppelin para interactuar con el token Yoppen y manejar las transferencias de tokens.

Código react Apps.js

Aquí tienes la explicación del código React proporcionado:

```
import React, { useState, useEffect } from 'react';
import Web3 from 'web3';
import CromoMarket from './contracts/CromoMarket.json';
import IERC20 from './contracts/IERC20.json';

const marketAddress = '0x265d37eB5f8D9998cBA2E83Ba0C0Da6E9C5431f8';
const tokenAddress = '0x38bC18AE393a7e560F8C26c1490f06D0EE069B73';

function App() {
  const [account, setAccount] = useState(null);
  const [web3, setWeb3] = useState(null);
  const [marketContract, setMarketContract] = useState(null);
  const [tokenContract, setTokenContract] = useState(null);
  const [cromos, setCromos] = useState([]);
  const [selectedCromo, setSelectedCromo] = useState(null);

  useEffect(() => {
    const init = async () => {
      if (window.ethereum) {
        const web3 = new Web3(window.ethereum);
        setWeb3(web3);

        const accounts = await web3.eth.requestAccounts();
        setAccount(accounts[0]);

        const marketContract = new web3.eth.Contract(CromoMarket.abi,
marketAddress);
        setMarketContract(marketContract);

        const tokenContract = new web3.eth.Contract(IERC20.abi,
tokenAddress);
        setTokenContract(tokenContract);

        const cromoCount = await
marketContract.methods.nextCromoId().call();
        const cromoList = [];
        for (let i = 0; i < cromoCount; i++) {
          const cromos = await marketContract.methods.getCromo(i).call();
          cromosList.push(cromos);
        }
        setCromos(cromosList);
      } else {
        alert('Please install MetaMask!');
      }
    };

    init();
  }, []);

  const handleBuyCromo = async (cromoId) => {
    const cromos = await marketContract.methods.getCromo(cromoId).call();
    const balance = await tokenContract.methods.balanceOf(account).call();
```



```

    if (parseInt(balance) >= parseInt(cromo.price)) {
      await tokenContract.methods.approve(marketAddress,
cromo.price).send({ from: account });
      const tx = await marketContract.methods.buyCromo(cromoId).send({
from: account });
      alert('Cromo purchased successfully');
    } else {
      alert('Insufficient balance');
    }
  };

  return (
    <div className="App">
      <h1>Cromo Marketplace</h1>
      <div>
        {cromos.map((cromo) => (
          <div key={cromo.id}>
            <h3>{cromo.name}</h3>
            <p>Price: {web3.utils.fromWei(cromo.price, 'ether')} Yoppen</p>
            {cromo.owner === '0x0000000000000000000000000000000000000000' ?
(
              <button onClick={() => handleBuyCromo(cromo.id)}>Buy</button>
            ) : (
              <p>Owned by: {cromo.owner}</p>
            )}
          </div>
        ))}
      </div>
    </div>
  );
}

```

Explicación del Código

Importaciones y Variables de Dirección

```

import React, { useState, useEffect } from 'react';
import Web3 from 'web3';
import CromoMarket from './contracts/CromoMarket.json';
import IERC20 from './contracts/IERC20.json';

const marketAddress = '0x265d37eB5f8D9998cBA2E83Ba0C0Da6E9C5431f8';
const tokenAddress = '0x38bC18AE393a7e560F8C26c1490f06D0EE069B73';

```

- Importaciones: Importa los módulos de React, Web3 y los contratos inteligentes.
- Direcciones: Define las direcciones del contrato de mercado y del token Yoppen.

Estado y Efectos

```
function App() {
  const [account, setAccount] = useState(null);
  const [web3, setWeb3] = useState(null);
  const [marketContract, setMarketContract] = useState(null);
  const [tokenContract, setTokenContract] = useState(null);
  const [cromos, setCromos] = useState([]);
  const [selectedCromo, setSelectedCromo] = useState(null);

  useEffect(() => {
    const init = async () => {
      if (window.ethereum) {
        const web3 = new Web3(window.ethereum);
        setWeb3(web3);

        const accounts = await web3.eth.requestAccounts();
        setAccount(accounts[0]);

        const marketContract = new web3.eth.Contract(CromoMarket.abi,
marketAddress);
        setMarketContract(marketContract);

        const tokenContract = new web3.eth.Contract(IERC20.abi,
tokenAddress);
        setTokenContract(tokenContract);

        const cromosCount = await
marketContract.methods.nextCromoId().call();
        const cromosList = [];
        for (let i = 0; i < cromosCount; i++) {
          const cromos = await marketContract.methods.getCromo(i).call();
          cromosList.push(cromos);
        }
        setCromos(cromosList);
      } else {
        alert('Please install MetaMask!');
      }
    };

    init();
  }, []);
```

Estados: Define estados para almacenar la cuenta del usuario, la instancia de Web3, los contratos y la lista de cromos. useEffect: Inicializa Web3, solicita cuentas de Metamask y configura los contratos de mercado y token. Obtiene la lista de cromos disponibles.

Manejo de Compra de Cromos

```
const handleBuyCromo = async (cromoId) => {
  const cromos = await marketContract.methods.getCromo(cromoId).call();
  const balance = await tokenContract.methods.balanceOf(account).call();
```

```

    if (parseInt(balance) >= parseInt(cromo.price)) {
      await tokenContract.methods.approve(marketAddress,
cromo.price).send({ from: account });
      const tx = await marketContract.methods.buyCromo(cromoId).send({
from: account });
      alert('Cromo purchased successfully');
    } else {
      alert('Insufficient balance');
    }
  };
};

```

handleBuyCromo: Maneja la lógica para comprar un cromó. Verifica el balance de tokens Yoppen del usuario, aprueba la transacción y realiza la compra del cromó.

Renderizado de la Interfaz de Usuario

```

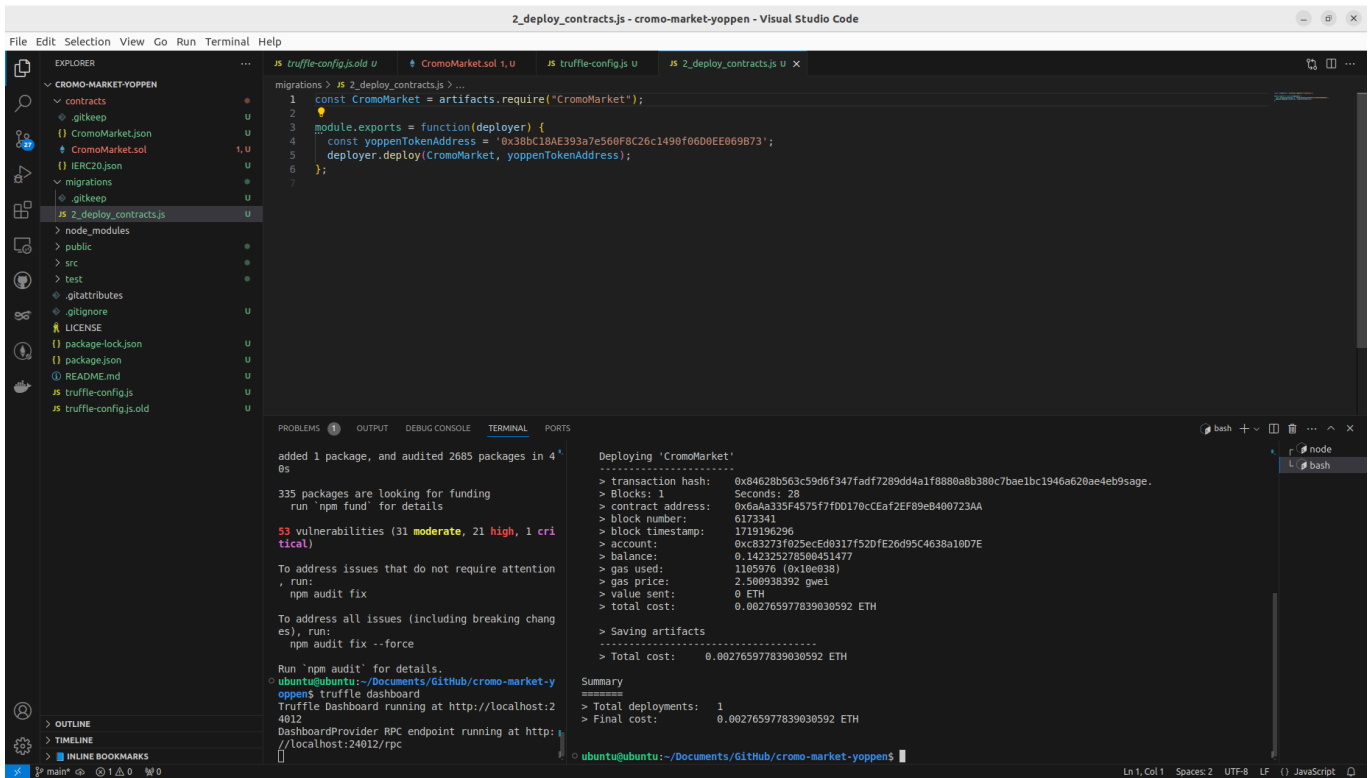
return (
  <div className="App">
    <h1>Cromo Marketplace</h1>
    <div>
      {cromos.map((cromo) => (
        <div key={cromo.id}>
          <h3>{cromo.name}</h3>
          <p>Price: {web3.utils.fromWei(cromo.price, 'ether')} Yoppen</p>
          {cromo.owner === '0x0000000000000000000000000000000000000000' ?
(
          <button onClick={() => handleBuyCromo(cromo.id)}>Buy</button>
        ) : (
          <p>Owned by: {cromo.owner}</p>
        )}
        </div>
      )]}
    </div>
  </div>
);
export default App;

```

- Renderizado: Muestra la lista de cromos disponibles. Permite a los usuarios comprar cromos si no tienen dueño, mostrando el precio en tokens Yoppen.

Este código configura una DApp básica que permite a los usuarios interactuar con contratos inteligentes desplegados en la blockchain Sepolia para comprar cromos utilizando tokens Yoppen.

Uso y Deploy de Apps.js



```
2_deploy_contracts.js - cromomarket-yoppen - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER
  CROMO-MARKET-YOPPEN
    contracts
    gitkeep
    CromoMarket.json
    CromoMarket.sol
    IERC20.json
    migrations
    gitkeep
    2_deploy_contracts.js
    node_modules
    public
    src
    test
    .gitattributes
    .gitignore
    LICENSE
    package-lock.json
    package.json
    README.md
    truffle-config.js
    truffle-config.js.old

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

added 1 package, and audited 2685 packages in 4'
0s

335 packages are looking for funding
run 'npm fund' for details

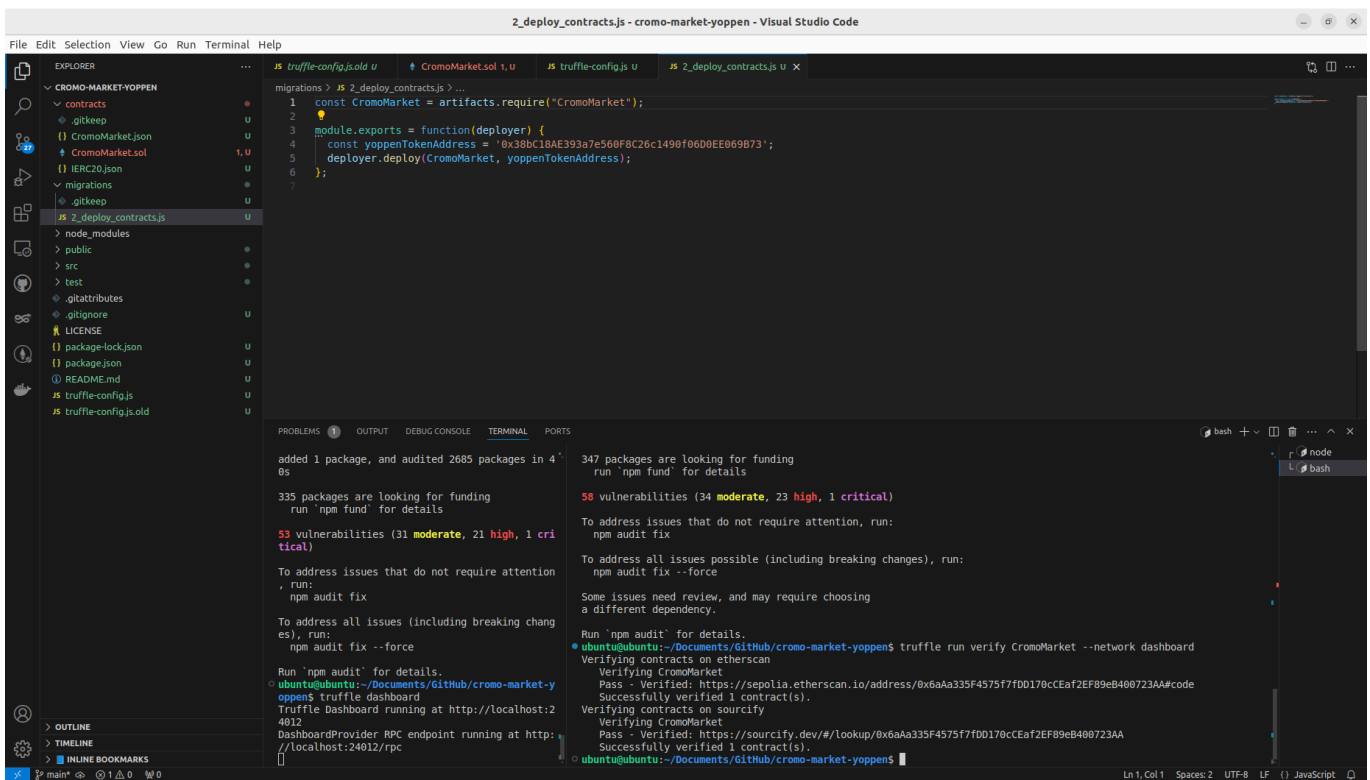
53 vulnerabilities (31 moderate, 21 high, 1 critical)
To address issues that do not require attention, run:
  npm audit fix
To address all issues (including breaking changes), run:
  npm audit fix --force
Run 'npm audit' for details.

Deploying 'CromoMarket'
-----
> transaction hash: 0x84628b563c59d6f347fadf7289dd4a1f8880ab380c7bae1bc1946a628ae4eb9sage.
> Blocks: 1
> contract address: 0x6Aa335F45757f7D0170cEaf2EF89eB400723AA
> block number: 6173341
> block timestamp: 1719196296
> account: 0xc83273f025ecE0317f520fE26d95C4638a1007E
> balance: 0.142325278500451477
> gas used: 1105976 (0x1be038)
> gas price: 2.500938392 gwei
> value sent: 0 ETH
> total cost: 0.002765977839030592 ETH

> Saving artifacts
-----
> Total cost: 0.002765977839030592 ETH

Summary
-----
> Total deployments: 1
> Final cost: 0.002765977839030592 ETH

ubuntu@ubuntu:~/Documents/GitHub/cromo-market-yoppen$
```



```
2_deploy_contracts.js - cromomarket-yoppen - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER
  CROMO-MARKET-YOPPEN
    contracts
    gitkeep
    CromoMarket.json
    CromoMarket.sol
    IERC20.json
    migrations
    gitkeep
    2_deploy_contracts.js
    node_modules
    public
    src
    test
    .gitattributes
    .gitignore
    LICENSE
    package-lock.json
    package.json
    README.md
    truffle-config.js
    truffle-config.js.old

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

added 1 package, and audited 2685 packages in 4'
0s

335 packages are looking for funding
run 'npm fund' for details

53 vulnerabilities (31 moderate, 21 high, 1 critical)
To address issues that do not require attention, run:
  npm audit fix
To address all issues (including breaking changes), run:
  npm audit fix --force
Run 'npm audit' for details.

347 packages are looking for funding
run 'npm fund' for details

58 vulnerabilities (34 moderate, 23 high, 1 critical)
To address issues that do not require attention, run:
  npm audit fix
To address all issues possible (including breaking changes), run:
  npm audit fix --force
Some issues need review, and may require choosing
a different dependency.
Run 'npm audit' for details.

Verify contracts on etherscan
Verifying contracts on etherscan
Pass - Verified: https://sepolia.etherscan.io/address/0x6Aa335F45757f7D0170cEaf2EF89eB400723AA#code
Successfully verified 1 contract(s).
Verifying contracts on sourcify
Verifying CromoMarket
Pass - Verified: https://sourcify.dev/#/lookup/0x6Aa335F45757f7D0170cEaf2EF89eB400723AA
Successfully verified 1 contract(s).

ubuntu@ubuntu:~/Documents/GitHub/cromo-market-yoppen$
```

Sepolia Testnet

Search by Address / Txn Hash / Block / Token

OverviewState

[This is a Sepolia Testnet transaction only]

Transaction Hash:

0xc238d5306671f7597590a6ed2bbca78a778a64d659a89d9f2211f8e66be89778

Status:

Success

Block:

61766773 Block Confirmations

Timestamp:

45 secs ago (Jun-24-2024 01:43:12 PM +UTC)

Transaction Action:

Call Create Cromo Function by 0xc83273f0...638a10D7E on 0x265d37eB...E9C5431f8

From:

0xc83273f025ecEd0317f52DfE26d95C4638a10D7E

To:

0x265d37eB5f8D9998cBA2E83Ba0C0Da6E9C5431f8

Value:

0 ETH (\$0.00)

Transaction Fee:

0.002099474577086398 ETH (\$0.00)

Gas Price:

21.596644246 Gwei (0.000000021596644246 ETH)

Gas Limit & Usage by Txn:

147,273 | 97,213 (66.01%)

Gas Fees:

Base: 20.096644246 Gwei | Max: 28.062544621 Gwei | Max Priority: 1.5 Gwei

Burnt & Txn Savings Fees:

Burnt: 0.001953655077086398 ETH (\$0.00)Txn Savings: 0.000628569573154875 ETH (\$0.00)

Other Attributes:

Txn Type: 2 (EIP-1559)Nonce: 54Position In Block: 67

Input Data:

#	Name	Type	Data
0	_name	string	Messi Copa America 2024 USA
1	_price	uint256	100

React Applocalhost:3000

Cromo Marketplace

Messi Copa America 2024 USA

Price: 0.000000000000000001 Yoppen

Buy

MetaMask1 of 2

Sepolia Account 1Balance50000 YPN

http://localhost:3000

Spending cap request for your

YPN

Verify third-party details

Custom spending cap0.000000000000000001Max

This allows the third party to spend YPN from your current balance.

Learn more

View details

RejectNext

Sepolia Testnet

Search by Address / Txn Hash / Block / Token

Transaction Hash:

0xfd4a11fc9f080a37d3cf2e26022e06844fcbcc58689e6646018a53f315b42996

Status:

Success

Block:

61767192 Block Confirmations

Timestamp:

26 secs ago (Jun-24-2024 01:51:36 PM +UTC)

Transaction Action:

Call Buy Cromo Function by 0xc83273f0...638a10D7E on 0x265d37eB...E9C5431f8

From:

0xc83273f025ecEd0317f52DfE26d95C4638a10D7E

Interacted With (To):

0x265d37eB5f8D9998cBA2E83Ba0C0Da6E9C5431f8

ERC-20 Tokens Transferred:

From 0xc83273f0...638a10D7E To 0xc83273f0...638a10D7E For 0.00000000000000001 Yoppen (YPN)

Value:

0 ETH (\$0.00)

Transaction Fee:

0.001307059835004467 ETH (\$0.00)

Gas Price:

20.146736671 Gwei (0.000000020146736671 ETH)

Gas Limit & Usage by Txn:

108,747 | 64,877 (59.66%)

Gas Fees:

Base: 17.646736671 Gwei | Max: 41.24377191 Gwei | Max Priority: 2.5 Gwei

Burnt & Txn Savings Fees:

Burnt: 0.001144867335004467 ETH (\$0.00)Txn Savings: 0.001368712355200603 ETH (\$0.00)

Other Attributes:

Txn Type: 2 (EIP-1559)Nonce: 56Position In Block: 15

Input Data:

#	Name	Type	Data
0	_cromoId	uint256	0

14 / 15

