1. 实验环境

① Windows11 VsCode ：代码编写，项目创建

② node.JS version:16 ：包管理，安装依赖环境

③ hardhat DEV ： 提供账户，导入之后可以与Remix IDE交互部署智能合约

④ Metamask：用私钥导入hardhat里的账户

⑤端口：8080 用于连接IPFS | 8545 用于用remixd连接remix IDE | 3000 用于后端运行 | 3001 用于前端运行

⑥ Remix IDE ：用于简单的部署智能合约

1. 产品设计思路
2. 合约设计
   1. ERC20.sol合约
   2. ERC721.sol合约
   3. nft-market.sol合约
3. 功能设计
   1. safeMint()方法，mint NFT
   2. 上架NFT
   3. 下架NFT
   4. NFT所有者可以更改NFT价格
   5. 查看所有NFT
   6. 购买NFT
   7. NFT信息上传到IPFS
4. 前端接口交互设计
   1. 购买NFT(buy)：第一步，查询allowance，如果allowance=0，则需要调用approve(address \_to,uint256 \_value)，原因是购买NFT时是在market进行购买；第二步，需要用拥有NFT的地址对market进行approve（这是ERC20的approve方法），让market合约能花拥有NFT地址的usdt。第三步，调用market.buy(tokenId)方法。
   2. 挖 NFT(safeMint)：第一步，图片上传到IPFS，返回CID；第二步，生成Metadata,返回CID；第三步，Mint NFT；
   3. 修改价格（changePrice）：market.changePrice(address \_seller, uint256 price)
   4. 下架(cancelOrder)：market.cancelOrder()
   5. 获取所有NFT(getAllNFTs)：第一步，调用nft.token(string URI)；第二步，market.getAllNFTs
   6. 获取个人NFT(getMyNFTs)：第一步，查看address账户里是否拥有余额，调用erc721.balanceOf(address)；第二步，通过调用erc721.tokenOfOwnerbyIndex(address \_owner,uint256 index)查看Owner账户的tokenId；第三步，通过erc721.tokenURI(uint256 tokenId)；第四步，调用market.getMyNFTs()；
5. 后端设计(基于express.js)
   1. 图片上传到IPFS(ipfs-uploader.js)：提供两个方法，一个是把json上传到IPFS(addJSONtoIPFS)，一个是把file文件上传到IPFS(addFiletoIPFS)。通过json可以获取到metadata的Cid.
   2. 挖 NFT(nft-minter)：提供一个mint(address,uri)方法，通过JSONProvider来与以太坊节点交互.
   3. 在app.js主要用的框架是express.js,主页面的地址是”localhost:3000/”,上传页面的地址是”localhost:3000/upload”.upload的设计思路是,
6. 页面设计
   1. 导航栏:如果监听到metamask连接了账户则显示账户地址,如果地址更换,显示地址也更换;如果没有连接metamask则可以点击按钮连接metamask
   2. market页面:由多个NFTCard组成
   3. NFT详情页:显示request body和metadata中的数据比如tokenId,title,price,description,seller的地址等等
   4. 上传图片页面:跟后端的展示差不多,换一个形式
   5. 上传图片成功页面:返回metadata,
7. 前端设计(基于react.js 这里重点说与合约交互的方式)
   1. 在前端将合约引入:以.json形式导入,这个.json文件在Remix IDE的abi那里可以复制.
   2. 通过工具类将各个合约的方法引入并放在utils包里
8. 产品合约实现 (8545端口)
9. 一个符合ERC20标准的token,名字为cUSDT，精度decimal=18，totalSupply=100 000 000。
   1. cUSDT中有2个事件。Transfer事件的作用是从账户\_from转到账户\_to，转账的金额是value；Approval事件的作用是通过账户\_owner来同意账户\_spender花费\_value这么多金额。事件的代码如下：

event Transfer(address indexed \_from, address indexed \_to, uint256 \_value)

event Approval(address indexed \_owner, address indexed \_spender, uint256 \_value)

* 1. cUSDT中有9个方法 。name、symbol、decimals、totalSupply四个方法分别是用来获取这个cUSDT的名字name，标志symbol，精度decimals，和发行的usdt的总量totalSupply；balanceOf方法是用来获取输入地址的余额balance；transfer方法使用来从当前合约转账给to地址，value这个值这么多的USDT,然后触发Transfer事件；transferFrom是用来同意合约账户代表你转账你的账户余额,这意味着\_owner账户对合约账户有approval事件的触发，而且必须触发Transfer事件； approve方法是用来同意\_spender账户可以花费\_owner账户的额度；allowance方法是用来返回\_spender目前还可以花费的\_owner账户的额度。方法的代码如下,：

mapping(address account => uint256) private \_balances;

mapping(address account => mapping(address spender => uint256)) private \_allowances;

uint256 private \_totalSupply;

string private \_name;

string private \_symbol;

function name() public view returns(string memory){

return \_name;

}

function totalSupply() public view returns(uint256){

return \_totalSupply;

}

function symbol() public view returns(string memory){

return \_symbol;

}

function decimals() public view returns(uint8){

return 18;

}

function balanceOf(address \_owner) public view returns(uint 256){

Return \_balances[account];

}

Function transfer(address \_to,uint256 \_value) public returns(bool){

Require(msg.sender != address(0),”sender can not be 0 address”);

Require(\_to != address(0),”to can not be 0 address”);

Require(\_balances[msg.sender] >= \_value,”balance is not enough”);

Emit Transfer(msg.sender,\_to,\_value);

Return true;

}

Function transferFrom(address \_from,address \_to,uint256 \_value) public returns(bool){

\_approve(\_from,\_to,\_value);

Allowance[\_from][\_to] = \_value;

Emit Transfer(\_from,\_to,\_value);

Return true;

}

Function \_approve(address \_from,address \_to,uint256 \_value) public returns(bool){

Require(\_from != address(0),”sender can not be 0 address”);

Require(\_to != address(0),”to can not be 0 address”);

Require(\_balances[msg.sender] >= \_value,”balance is not enough”);

Emit Approval(\_from,\_to,\_value);

Return true;

}

Function approve(address spender,uint256 value) public returns(bool){

require(msg.sender != address(0),”can not be 0 adderss”);

Require(spender != address(0),”can not be 0”);

Allowance[owner][spender]=value;

Emit approval(owner,spender,value);

Return true;

}

Function allowance(address \_owner,address \_spender) public returns()uint256{

Return allowance[\_owner][\_spender];

}

因为要开发测试代码，所以直接引入官方的库。主要代码如下：

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

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

contract cUSDT is ERC20 {

constructor() ERC20("fake USDT", "cUSDT") {

\_mint(msg.sender, 1 \* 10 \*\* 8 \* 10 \*\* decimals());

}

}

1. 一个符合ERC721标准的token，名字为NFTM，它和ERC20的主要区别是加了**tokenURI**和**safeTransferFrom**两个方法

function tokenURI(uint256 tokenId) public view returns (string memory) {

\_requireOwned(tokenId);

string memory baseURI = “”;

return bytes(baseURI).length > 0 ? string.concat(baseURI, tokenId.toString()) : "";

}

function \_checkOnERC721Received

(address from, address to, uint256 tokenId, bytes memory data)

private {

if (to.code.length > 0) {

try IERC721Receiver(to).onERC721Received(\_msgSender(), from, tokenId, data) returns (bytes4 retval) {

if (retval != IERC721Receiver.onERC721Received.selector) {

revert ERC721InvalidReceiver(to);

}

} catch (bytes memory reason) {

if (reason.length == 0) {

revert ERC721InvalidReceiver(to);

} else {

/// @solidity memory-safe-assembly

assembly {

revert(add(32, reason), mload(reason))

}

}

}

}

}

function safeTransferFrom(address from, address to, uint256 tokenId, bytes memory data) public {

transferFrom(from, to, tokenId);

\_checkOnERC721Received(from, to, tokenId, data);

}

在ERC721.sol中，主要方法的代码如下：

function safeMint(address to, string memory uri) public onlyOwner {

uint256 tokenId = \_nextTokenId++;

\_safeMint(to, tokenId);

\_setTokenURI(tokenId, uri);

}

function tokenURI(

uint256 tokenId

) public view override(ERC721, ERC721URIStorage) returns (string memory) {

return super.tokenURI(tokenId);

}

function supportsInterface(

bytes4 interfaceId

)

public

view

override(ERC721, ERC721Enumerable, ERC721URIStorage)

returns (bool)

{

return super.supportsInterface(interfaceId);

}

}

1. nft-market的功能
   1. 用户在上架NFT的时候可以标明价格
   2. NFT拥有者可以改变NFT的价格
   3. 下架NFT
   4. 购买NFT
   5. NFT信息上传到IPFS

实现功能的思路:

1. 购买功能buy:第一步,判断tokenId是否在market列表里;第二步,通过orderOfId(存取的是tokenId对应的Order)这个映射获取seller和buyer账户的地址和nft的价格;第三步,market在得到buyer账户的approve后,用transerFrom转price个usdt给seller;第四步,转账usdt成功后,用safeTransferFrom方法将NFT转给buyer;第五步,删除tokenId在market的列表并释放事件Deal.
2. 上架功能placeOrder:第一步,判断价格price是否大于0,如果大于0那么获取通过idToOrderIndex(记录tokenId和Order的index的映射)seller,price,tokenId;第二步,将seller账户要上架的tokenId push到market的列表;第三步,在idToOrderIndex这个mapping里删除上架的tokenId;第四步,释放NewOrder事件.
3. 下架功能cancelOrder:第一步,判断想要下架的tokenId是否在market的列表里,如果在,则通过orderOfId获取seller;第二步,market通过safeTransferFrom,转回给seller账户;第三步,通过removeList方法将tokenId从market列表删除;第四步,释放cancelOrder事件.
4. 改变价格changePrice:第一步,判断此tokenId是否在market列表里;第二步,判断现在信息的账户地址是否为seller的地址;第三步,修改Order里存储的price为输入的\_price;第四步,释放changePrice事件
5. 获取当前账户的NFT getMyNFTs:通过for循环获取当前账户与seller账户相等的地址赋值到一个新的Order数组并返回.
6. 获取market里所有的NFT getAllNFTs:通过order数组返回.

event Deal(address buyer, address seller, uint256 tokenId, uint256 price);

event NewOrder(address seller, uint256 tokenId, uint256 price);

event CancelOrder(address seller, uint256 tokenId);

event ChangePrice(

address seller,

uint256 tokenId,

uint256 previousPrice,

uint256 price

);

主要方法的代码如下：

struct Order {

address seller;

uint256 tokenId;

uint256 price;

}

mapping(uint256 => Order) public orderOfId; // token id to order

Order[] public orders;

mapping(uint256 => uint256) public idToOrderIndex;

// 购买

function buy(uint256 \_tokenId) external {

require(isListed(\_tokenId), "Market: Token ID is not listed");

address seller = orderOfId[\_tokenId].seller;

address buyer = msg.sender;

uint256 price = orderOfId[\_tokenId].price;

require(

erc20.transferFrom(buyer, seller, price),

"Market: ERC20 transfer not successful"

);

erc721.safeTransferFrom(address(this), buyer, \_tokenId);

removeListing(\_tokenId);

emit Deal(buyer, seller, \_tokenId, price);

}

// 上架

function placeOrder(

address \_seller,

uint256 \_tokenId,

uint256 \_price

) internal {

require(\_price > 0, "Market: Price must be greater than zero");

orderOfId[\_tokenId].seller = \_seller;

orderOfId[\_tokenId].price = \_price;

orderOfId[\_tokenId].tokenId = \_tokenId;

orders.push(orderOfId[\_tokenId]);

idToOrderIndex[\_tokenId] = orders.length - 1;

emit NewOrder(\_seller, \_tokenId, \_price);

}

// 下架

function cancelOrder(uint256 \_tokenId) external {

require(isListed(\_tokenId), "Market: Token ID is not listed");

address seller = orderOfId[\_tokenId].seller;

require(seller == msg.sender, "Market: Sender is not seller");

erc721.safeTransferFrom(address(this), seller, \_tokenId);

removeListing(\_tokenId);

emit CancelOrder(seller, \_tokenId);

}

// 改变价格

function changePrice(uint256 \_tokenId, uint256 \_price) external {

require(isListed(\_tokenId), "Market: Token ID is not listed");

address seller = orderOfId[\_tokenId].seller;

require(seller == msg.sender, "Market: Sender is not seller");

uint256 previousPrice = orderOfId[\_tokenId].price;

orderOfId[\_tokenId].price = \_price;

Order storage order = orders[idToOrderIndex[\_tokenId]];

order.price = \_price;

emit ChangePrice(seller, \_tokenId, previousPrice, \_price);

}

// 获取所有NFT

function getAllNFTs() public view returns (Order[] memory) {

return orders;

}

// 获取当前账户的NFT

function getMyNFTs() public view returns (Order[] memory) {

Order[] memory myOrders = new Order[](orders.length);

uint256 myOrdersCount = 0;

for (uint256 i = 0; i < orders.length; i++) {

if (orders[i].seller == msg.sender) {

myOrders[myOrdersCount] = orders[i];

myOrdersCount++;

}

}

Order[] memory myOrdersTrimmed = new Order[](myOrdersCount);

for (uint256 i = 0; i < myOrdersCount; i++) {

myOrdersTrimmed[i] = myOrders[i];

}

return myOrdersTrimmed;

}

// 购买之后将 order 移除列表

function removeListing(uint256 \_tokenId) internal {

delete orderOfId[\_tokenId];

uint256 orderToRemoveIndex = idToOrderIndex[\_tokenId];

uint256 lastOrderIndex = orders.length - 1;

if (lastOrderIndex != orderToRemoveIndex) {

Order memory lastOrder = orders[lastOrderIndex];

orders[orderToRemoveIndex] = lastOrder;

idToOrderIndex[lastOrder.tokenId] = orderToRemoveIndex;

}

orders.pop();

}

}

1. 产品后端实现 (3000端口)
2. 文件上传到IPFS的功能(ipfs-uploader)

提供将JSON上传到IPFS(addJSONtoIPFS)和将文件上传到IPFS(addFiletoIPFS)两个方法.实现的思路是:第一步,创建ipfs的url;第二步,通过JSON.stringify(json) 将 JSON 对象转换为字符串表示形式然后将这个字符串添加到IPFS.通过fs库里面的读写文件路径对应的文件,然后向IPFS添加.主要代码如下:

const ipfs = create(new URL(process.env.IPFS\_URL))

export async function addJSONToIPFS(json) {

try {

const result = await ipfs.add(JSON.stringify(json));

return result;

} catch (error) {

console.error('Failed to add JSON to IPFS:', error);

}

}

export async function addFileToIPFS(filePath) {

try {

const file = fs.readFileSync(filePath);

const result = await ipfs.add({ path: filePath, content: file });

return result;

} catch (error) {

console.error('Failed to add file to IPFS:', error);

}

}

1. 挖 NFT的功能(nft-minter):提供向配置文件.dev中的地址挖NFT,主要思路是:第一步,通过JSONRpcProvider获取provider,通过provider.getSigner获取signer;第二步,将provider和signer作为获取contract的参数,通过ethers.Contract获取MyNFT.sol的合约方法safeMint.主要代码如下:

export async function mint(address, uri) {

const provider = new JsonRpcProvider(process.env.RPC);

const signer = await provider.getSigner()

const MyNFTAbi = JSON.parse(fs.readFileSync('./abis/MyNFT.json'));

const MyNFTContract = new ethers.Contract(process.env.NFT, MyNFTAbi, signer);

const result = await MyNFTContract.safeMint(address, uri)

}

1. 网页测试的功能(app.js):提供了 “localhost:3000/” 和 “localhost:3000/upload”;实现思路是:第一步,通过get设置根路径/跳转到home.ejs页面;第二步,/upload是要传metadata信息的,所以传送信息的方式应该是post的方式,在/upload里先判断files目录下是否有文件,如果没有,则没有下文;然后将metadata通过addJSONtoIPFS添加到IPFS上.上传文件也是相同的思路,同时实现将上传的图片作为NFT的图片,mint 到部署合约的账户主要代码如下:

app.get('/', (req,res) => {

res.render('home')

})

app.post('/upload', (req, res) => {

if (!req.files || Object.keys(req.files).length === 0) {

return res.status(400).json({ message: 'No files were uploaded.' });

}

const file = req.files.file;

const fileName = file.name

const filePath = 'files/' + fileName

const title = req.body.title

const description = req.body.description

const address = req.body.address

file.mv(filePath, async (err) => {

if (err) {

console.log('error: failed to download the file.')

return res.status(500).send(err)

}

const fileResult = await addFileToIPFS(filePath)

console.log('File added to IPFS:', fileResult.cid.toString());

const metadata = {

title,

description,

image: 'http://127.0.0.1:8080/ipfs/' + fileResult.cid.toString() + '/' + fileName

}

const jsonResult = await addJSONToIPFS(metadata)

console.log('Metadata added to IPFS:', jsonResult.cid.toString());

const userAddress = address || process.env.ADDRESS;

await mint(userAddress, 'http://127.0.0.1:8080/ipfs/' + jsonResult.cid.toString())

res.json({

message: 'File uploaded successfully.',

metadata

});

});

});

1. 产品前端实现 (3001端口)
2. 页面实现:这个项目主要包括四个页面,主页面(market),图片上传成功页面(uploadSuccess),NFT详情页(NFTDetail),图片上传页面(uploadImage);而market页面可以分为Navbar导航栏和若干个NFTCard组成;图片上传页面由Navbar导航栏,title,description,选择文件和上传图片的按钮;图片上传成功页面包括title,descripstion,imageurl;NFT详情页,包含Navbar导航栏和title,tokenId,price,seller还有IPFS的图片;主要的代码如下:

const Navbar = ({ onConnectWallet, address }) => {

return (

<nav className="navbar">

<div className="navbar-brand">NFT Marketplace</div>

<Link to="/">Home</Link>

<Link to="/create-nft">Create NFT</Link>

<div className="navbar-menu">

<button className="connect-wallet-button" onClick={onConnectWallet}>

{address || "Connect Wallet"}

</button>

</div>

</nav>

);

};

const NFTGrid = () => {

const [nfts, setNfts] = useState([]);

const navigate = useNavigate();

const handleCardClick = (tokenId) => {

navigate(`/nft-detail/${tokenId}`);

};

const NFTCard = ({tokenId, onClick}) => {

// console.log(tokenId)

const [metadata, setMetadata] = useState('');

const [order, setOrder] = useState('');

useEffect(() => {

const getInfo = async () => {

const metadata = await getMetadata(tokenId);

const order = await getOrder(tokenId);

setMetadata(metadata);

setOrder(order);

}

getInfo();

}, []);

const NFTDetail = () => {

const { tokenId } = useParams();

const [metadata, setMetadata] = useState('');

const [order, setOrder] = useState('');

const [allowance, setAllowance] = useState(0);

const getWalletAddress = async () => {

if (window.ethereum) {

try {

const accounts = await window.ethereum.request({ method: 'eth\_requestAccounts' });

return accounts[0];

} catch (error) {

console.error('Error connecting to wallet:', error);

}

}

};

const handleBuyClick = async () => {

if (allowance === 0) {

await approve("market address");

} else {

await buy(tokenId);

}

};

const UploadSuccess = () => {

return (

<div>

<h1>Upload Successfully</h1>

<p>Your image has been uploaded to IPFS successfully!</p>

</div>

);

};

function UploadImage({ address }) {

const [title, setTitle] = useState('');

const [description, setDescription] = useState('');

const fileInputRef = useRef(null);

const navigate = useNavigate();

const handleCancel = () => {

// Reset the form or specific form fields

setTitle('');

setDescription('');

if (fileInputRef.current) {

fileInputRef.current.value = "";

}

};

const handleUpload = async (event) => {

event.preventDefault(); // Prevent the default form submission

if (fileInputRef.current.files.length === 0) {

alert('Please select a file to upload.');

return;

}

const formData = new FormData();

formData.append('title', title);

formData.append('description', description);

formData.append('file', fileInputRef.current.files[0]);

formData.append('address', address);

try {

const response = await axios.post('http://127.0.0.1:3000/upload', formData, {

headers: {

'Content-Type': 'multipart/form-data'

}

});

navigate('/success');

} catch (error) {

// Handle the error here

console.error('Error uploading file:', error);

}

};

return (

<div className="upload-container">

<h1>Upload Image to IPFS and Mint NFT</h1>

<form className="upload-form" onSubmit={handleUpload}>

<label htmlFor="title">Title \*</label>

<input

type="text"

id="title"

placeholder="Enter image title"

value={title}

onChange={(e) => setTitle(e.target.value)}

required

/>

<label htmlFor="description">Description</label>

<textarea

id="description"

placeholder="Describe your image"

value={description}

onChange={(e) => setDescription(e.target.value)}

/>

<label htmlFor="file">Image \*</label>

<input

type="file"

id="file"

ref={fileInputRef}

required />

<div className="buttons">

<button type="button" className="cancel-button" onClick={handleCancel}>Cancel</button>

<button type="submit" className="upload-button">Upload</button>

</div>

</form>

</div>

);

}

1. 与合约交互(与后端不同,前端通过ethers.BrowserRpcProvider)分别由三个工具类里面提供market,nft,usdt的功能,从而通过market,nft,usdt的abi.json文件和合约交互
   1. usdt.js(与usdt的合约交互):提供approve和allowance两个方法,用来在buy方法里让buyer对market同意;主要的代码如下:

export async function approve(spender, amount) {

const contract = new ethers.Contract(contractAddress, ABI, await provider.getSigner());

const result = await contract.approve(spender, amount);

console.log(result.hash);

}

export async function getAllowance(owner, spender) {

const contract = new ethers.Contract(contractAddress, ABI, await provider.getSigner());

const result = await contract.allowance(owner, spender);

return Number(result);

}

* 1. nft.js(与MyNFT的合约交互):提供balanceOf,tokenOfOwnerbyIndex,tokenURI,getMetadata四个方法,用于在NFT详情页和market主页面.代码如下:

export async function balanceOf(address) {

const result = await contract.balanceOf(address);

return Number(result);

}

export async function tokenOfOwnerByIndex(owner, index) {

const result = await contract.tokenOfOwnerByIndex(owner, index);

return Number(result);

}

export async function tokenURI(tokenId) {

const result = await contract.tokenURI(tokenId);

console.log(result);

}

export async function getMetadata(tokenId) {

const result = await contract.tokenURI(tokenId);

const response = await axios.get(result);

return {

title: response.data.title,

description: response.data.description,

imageURL: response.data.image,

}

}

* 1. market.js(与market进行交互):提供购买,改变价格,获取所有NFT,获取个人的NFT;用于market主页面,和NFT详情页面主要代码如下:

export async function buy(tokenId) {

const result = await contract.buy(tokenId);

console.log('buy', result.hash);

}

export async function changePrice(tokenId, price) {

const result = await contract.changePrice(tokenId, price);

console.log('change price', result.hash);

}

export async function cancelOrder(tokenId) {

const result = await contract.cancelOrder(tokenId);

console.log('cancel order', result.hash);

}

export async function getAllNFTs() {

const result = await contract.getAllNFTs();

console.log(result);

}

export async function getMyNFTs() {

const result = await contract.getMyNFTs();

console.log(result);

}

export async function getOrder(tokenId) {

const result = await contract.orderOfId(tokenId);

return {

seller: result[0],

tokenId: Number(result[1]),

price: Number(result[2]) / 1e18,

}

}

6. 产品开发心得体会

1. 学习到的知识
   * + 1. Package.json的作用
     1. “main”中指定运行类
     2. “script”中指定运行或者启动命令的简化命令
     3. 在安装其他依赖项之后可以在package.json里查看版本号
        1. Express.js
     4. 添加路由

app.get | post(“/address/to”,(req,res)=>{})

* + 1. 中间件:用于接收到用户的请求,但是还没有发回response时发挥作用;有三个参数(req,res,next),可用于其他组件

const express = require('express')

const app = express()

app.use((req, res, next) => {

console.log('Time:', Date.now())

next()

})

* + 1. 模板引擎(用类似html的语言,但是没有中括号和闭合的)

app.set(“view engine”, “pug”);

app.set(“views”, “./views”)

* + - 1. React.js
    1. 组件(在主类里可以使用,一个一个组件凑成一个页面像积木一样)
    2. JSX:在一个文件里,既可以写css样式又可以写JavaScript逻辑,很方便
    3. 状态管理(useState)
    4. 属性传递Props(useEffect)
       1. IPFS(InterPlanetary File System):IPFS是一套用于组织和传输数据的模块化协议，采用内容寻址和对等网络的原则从头设计。因为IPFS是开源的，所以IPFS有多种实现。虽然IPFS有多个用例，但它的主要用例是以一种分散的方式发布数据(文件、目录、网站等);它的好处是:可以永久的存储一个文件、目录、网站等一系列url.因为传统的上传图片的网站可能过期,查询不到;

1. 遇到过的问题
   * + 1. safeTransferFrom(address \_from,address \_to,uint256 \_value,string data),传data的时候要传32位的十六进制,否则交易会失败,并且solidity中不存在小数,而是通过更小的单位来避免小数.
       2. Seller在market上架的时候,market要收到seller的erc721.approve;buyer在market购买的时候,market要收到buyer的erc20.approve,否则,Transfer事件和TransferFrom事件会因为allowance不够而交易失败.
       3. 前端功能正确的前提是后端代码能运行,如果在写前端(localhost:3001)的时候发现功能有问题,则可以先检查后端(localhost:3000)功能是否能正常运行