Generació i gestió d'NFTs sobre Ethereum

Marc González Amores

Resum–
Una de les aplicacions més interessants de la tecnologia Blockchain són els NFT, o tokens no fungibles. Aquests, han guanyat popularitat en els darrers anys com una forma de representar la propietat i autenticitat d'elements digitals, com ara obres d'art, música, vídeos, jocs i altres actius virtuals. Representen un certificat digital d'autenticació que es crea en la tecnologia Blockchain, similar a altres monedes virtuals o criptoactius. Aquest TFG es centra en crear una col·lecció de NFT (Tokens No Fungibles) i desenvolupar una plataforma de gestió i compra/venda d'NFTs totalment descentralitzada sobre la blockchain d'Ethereum, per facilitar les transaccions d'aquests actius digitals únics.

Paraules clau– Cadena de blocs, tokens no fungibles, Ethereum, Smart Contracts, aplicacions descentralitzades

Abstract– This work focuses on the creation of a collection of NFTs (Non-Fungible Tokens) and the development of a fully decentralized NFT management and buying/selling platform on the Ethereum blockchain to facilitate transactions of these unique digital assets. One of the most interesting applications of Blockchain technology are NFTs, or non-fungible tokens. These, have gained popularity in international markets as a way to represent ownership and authenticity of digital items, such as artwork, music, videos, videos and other virtual assets. They represent a digital certificate of authentication that is created on Blockchain technology, similar to other virtual wallets or cryptoassets.

Keywords– Blockchain, non-fungible tokens, Ethereum, Smart Contracts, decentralized applications

1 Introducció - Context del treball

Una cadena de blocs [1], és una base de dades distribuïda que registra les transaccions o altres tipus de dades compartides pels nodes d'una xarxa d'ordinadors. Cada bloc conté les seves dades i un hash criptogràfic del bloc anterior, de manera que formen una cadena en la qual cada bloc nou s'enllaça amb els blocs anteriors. Gràcies a aquesta propietat les transaccions de la cadena de blocs són irreversibles és a dir, un cop són registrades, les dades d'un bloc determinat no es poden canviar sense que s'alterin tots els blocs posteriors. Normalment, solen ser gestionades per una xarxa P2P pel seu ús com una base de dades distribuïda, en el que tots els nodes s'ajusten a un protocol d'algoritme de consens per afegir i validar nous blocs de transaccions. La innovació d'aquesta tecnologia és que assegura la fidelitat i la seguretat d'un registre de dades i genera confiança sense necessitat d'un tercer de confiança. És més coneguda com la tecnologia que es troba darrere de criptomonedes com el Bitcoin [2] o Ethereum [3] , però té moltes altres aplicacions potencials. Una de les aplicacions més interessants de la tecnologia Blockchain són els NFT [4], o tokens no fungibles. Els tokens no fungibles (NFT) representen un certificat digital d'autenticació que es crea en la tecnologia Blockchain, similar a altres monedes virtuals o criptoactius. Un NFT és un tipus especial d'actiu digital o token que es pot demostrar que és únic i no intercanviable. Per això s'anomena "token no fungible". Per entendre millor el concepte és útil comparar un NFT amb un token o actiu fungible. Els tokens fungibles, que són la majoria en l'àmbit de la Blockchain, són tokens que tenen les mateixes característiques que qualsevol altre i, per tant, es poden substituir fàcilment per qualsevol token idèntic. La il·lustració més familiar d'un actiu "fungible"són els diners en efectiu, ja que un bitllet de 10 euros es pot intercanviar per qualsevol altre bitllet de 10 euros. Per contra, exemples reals d'actius no fungibles són les entrades a esdeveniments, els registres legals de propietat d'actius, com ara els títols d'una propietat, així com les obres d'art Els NFT es creen d'acord amb determinats marcs o normes i es despleguen a la cadena. Actualment, la Blockchain més popular per als NFT és Ethereum, i l'estàndard més utilitzat és l'ERC-721 [5] d'Ethereum, que determina certes característiques per als NFT. Aquestes característiques els ha fet populars entre

• E-mail de contacte: marcgonzaam@gmail.com
• Menció realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Jordi Herrera Joancomarti (DEIC)
• Curs 2022/2023

col·leccionistes, creadors i inversors que veuen el potencial dels NFT per pertorbar les indústries tradicionals i crear noves oportunitats en el món digital. Les més importants són les següents:

- Immutables: Un cop creats no es poden alterar ni destruir, ja que s'emmagatzemen en la cadena de blocs. Això fa que es pugui demostrar l'autenticitat i la propietat dels actius digitals.
- Valor: Tenen valor perquè són únics i limitats, per tant, poden comprar-se i vendre's per un preu igual que si fossin actius físics. El seu preu pot estar determinat per la seva raresa, popularitat del creador o la demanda.
- Unicitat: Cada NFT és únic i té la seva identitat en la cadena de blocs. No poden ser replicats.
- Traçables: Com que s'emmagatzemen a la cadena de blocs, totes les transaccions es guarden i això facilita el seguiment de l'historial de propietat d'un NFT.
- Interoperables: Poden negociar-se i intercanviar-se en diferents plataformes i mercats.

2 OBJECTIUS DEL TREBALL

Per aquest projecte, l'objectiu principal és la creació d'una plataforma de gestió i compra/venda d'NFTs totalment descentralitzada sobre la blockchain d'Ethereum utilitzant emmagatzematge distribuït IPFS [6]. Per aconseguir assolir aquest objectiu general ens caldrà treballar en els següents objectius específics:

1. Crear els NFT
2. Desenvolupar els smart contracts [7] amb Solidity [8]
3. Crear un entorn local de la testnet d'Ethereum
4. Desplegar un smart contract en un entorn de testnet
5. Crear les dApps fent servir tecnologies de Web3 i llibreries de JavaScript
6. Aprofundir sobre com emmagatzemar informació en IPFS
7. Aplicar bones pràctiques en la codificació dels smart contracts, per tal que aquests siguin segurs

3 ESTAT DE L'ART

La tecnologia blockchain ha emergit com una solució innovadora amb un impacte significatiu en diversos sectors. En particular Ethereum, ha destacat, en el desenvolupament d'aplicacions descentralitzades (dApps), gràcies a la seva capacitat per executar smart contract i la seva àmplia comunitat de desenvolupadors.

En aquest context, els tokens no fungibles (NFT) han guanyat una gran popularitat i rellevància en els darrers anys i la seva utilitat, continua explorant-se i no es deixen de desenvolupar nous casos d'ús. A mesura que la tecnologia evoluciona, podem esperar veure aplicacions encara més innovadores i creatives dels NFT els pròxims anys. La capacitat dels NFTs per autenticar, verificar la propietat i establir l'escassetat digital ha obert noves oportunitats per a artistes, creadors de contingut i propietaris d'actius digitals, permetent monetitzar i comercialitzar les seves creacions d'una manera segura i transparent.

D'altra banda, el desenvolupament de dApps a Ethereum ha donat lloc a una àmplia gamma d'aplicacions descentralitzades que van més enllà de les transaccions financeres. Aquestes dApps permeten la creació de sistemes de votació, plataformes de préstecs, jocs i molt més, sense dependre d'intermediaris centralitzats. Això proporciona més transparència, seguretat i governança comunitària als usuaris, empoderant-los amb més control sobre les seves dades i actius digitals. A més, la interoperabilitat entre blockchains diferents i la creació d'estàndards comuns són temes clau en l'evolució d'aquesta tecnologia. Projectes com Chainlink estan treballant en solucions que facilitin la transferència d'actius i la comunicació entre diferents xarxes blockchain. Així mateix, els estàndards com ERC-20 (per a tokens fungibles) i ERC-721 (per a NFTs) a Ethereum han permès una major interoperabilitat i han impulsat el desenvolupament d'ecosistemes rics i diversificats.

Tot i els avenços aconseguits, l'adopció massiva de blockchain, NFTs i dApps encara enfronta reptes importants. L'escalabilitat de les xarxes blockchain, els costos de transacció i la sostenibilitat energètica són aspectes que cal abordar per permetre un creixement i una adopció més amplis. A més, l'educació i la consciència pública sobre aquestes tecnologies són fonamentals perquè el públic en general comprengui el seu potencial i se senti còmode en utilitzar-les en la vida diària.

4 METODOLOGIA I PLANIFICACIÓ

4.1 Metodologia

Per a la realització del projecte s'ha utilitzat la metodologia cascada, ja que les fases i activitats a realitzar depenen de les anteriors, és a dir, per saltar d'una fase a un altre necessitem haver completat la fase anterior. Per aquesta raó el més adequat ha estat utilitzar aquesta metodologia. Per a tenir un control del treball, s'ha fet ús de la metodologia de Kanban, utilitzant l'eina de Trello [9], una eina visual que ens permet gestionar el nostre projecte i flux de treball, així com supervisar les activitats que estem duent a terme. Per documentar el meu treball s'ha GitHub.

4.2 Planificació

Principalment, el projecte està dividit en dues fases ben diferenciades, que són, en primer lloc, la creació d'una col·lecció de 1000 NFT i la seva respectiva dApp on els usuaris poden connectar-se amb la seva wallet i poder comprar-los, i la segona part en la creació del marketplace on poder fer la compra i venda dels NFT que un usuari té de la col·lecció. Ambdues parts estan dividides en un Back End i un Front End. El Back End consisteix en la creació de smart contracts a la testnet d'Ethereum anomenada Goerlï on cada smart contract tindrà unes funcions determinades. Pel que fa al Front End la seva funció és la de connectar el smart contract a la blockchain i poder fer a l'usuari més senzill interactuar amb el contracte creat.

La primera fase del projecte, la creació dels NFT, ha estat constituïda per diverses activitats, en primer lloc, la creació gràfica dels NFT on utilitzant Photoshop s'ha creat una estructura de dibuixos per capes, on cada capa serà una part del personatge. Un cop creada l'estructura de capes, s'ha utilitzat l'eina hashlips art engine [10] que ens permet superposar aquestes capes de manera que s'han creat els personatges de manera automàtica i única, és a dir, que no n'hi ha cap amb les mateixes característiques. En segon lloc, un cop creada la col·lecció, s'han pujat tant els arxius de metadada dels NFT com les imatges, a un IPFS per poder guardar de manera descentralitzada les nostres imatges i assegurar-nos que sempre siguin accessibles. Per això s'ha utilitzat NFTStorage. La tercera activitat duta a terme ha estat la creació del smart contract fent servir l'ERC-721, l'estàndard dels NFT. Això ens permetrà registrar la nostra col·lecció a la cadena de blocs, que en el meu cas serà en la cadena de testnet d'Ethereum Goerlï. I per finalitzar, s'ha desplegat el smart contract dels NFT creats. Un cop desplegat el smart contract a la cadena de blocs, la segona part d'aquesta fase és la creació del Front End. Per tal que els usuaris puguin interactuar amb el nostre smart contract de la manera més senzilla possible, s'ha creat un Front End per tal que puguin fer mint dels NFT creats només fent un clic. Per dur-lo a terme s'ha fet servir React, que és una biblioteca de JavaScript molt utilitzada per la construcció d'aplicacions descentralitzades (DApps).

Pel que fa a la segona fase del projecte, la creació del marketplace on poder fer la compra i venda dels NFT, està format principalment per dues parts que són el Back End on està tota la lògica i les dades, i, per altra part, el Front End que és el que l'usuari veu per interactuar amb el Back End. D'una banda, el Back End està format pel smart contract que està escrit en el llenguatge de Solidity, que permet que l'usuari pugui fer la lògica del Marketplace. A més a més del smart contract, també s'ha utilitzat The Graph per llegir els events que emet el contracte. Per l'altra banda el Front End utilitza el framework de React i JavaScript per poder crear una pàgina on l'usuari es pot connectar utilitzant el seu wallet, i així poder interactuar de manera còmoda amb el contracte creat al Back End.

5 Desenvolupament de l'aplicació

5.1 Entorn de desenvolupament

En aquesta secció s'explicaràn les diverses tencologies utilitzades per realitzar el desenvoulpament del projecte.

5.1.1 Hardhat

Hardhat és un conjunt d'eines de desenvolupament de software per Ethereum que s'utilitza per compilar, provar, implementar i desenvolupar smart contracts. Ens permet interactuar amb una xarxa de prova local i simular transaccions d'Ethereum. També ens permet desplegar els smart contracts a xarxes reals com la Mainnet, tot i que en el nostre cas el treballarem amb la xarxa de prova d'Ethereum anomenada Goerli

5.1.2 Solidity

Solidity és un llenguatge de programació utilitzat per escriure contractes intel·ligents a Ethereum. Permet la creació de contractes amb regles, i pot fer càlculs complexos. Facilita la interacció amb altres contractes i la gestió de criptomonedes. També ofereix característiques de seguretat i es compila a bytecode per executar-se a la Màquina Virtual d'Ethereum. Solidity és àmpliament adoptat en l'ecosistema de blockchain i és essencial per al desenvolupament d'aplicacions descentralitzades a Ethereum. En el projecte s'ha utilitzat pel desenvolupament dels smart contracts.

5.1.3 OpenZeppelin

OpenZeppelin és una biblioteca de smart contracts de codi obert i una plataforma de desenvolupament fiable per a Ethereum i altres blockchains. Proporciona contractes segurs i reutilitzables, així com eines per facilitar el desenvolupament d'aplicacions descentralitzades. És àmpliament utilitzat i es considera un estàndard de seguretat a la indústria blockchain. Més concretament en el projecte s'han utilitzat els contractes:

- ERC721URIStorage: és una extensió del contracte d'ERC721 d'OpenZeppelin. En el nostre cas l'utilitzem per establir URIs de tokens individuals mentre fent mint dels NFT aleatoris
- Ownable: Ens dona un mecanisme de control que permetrà fer ús d'una funció de retirar els diners només al propietari del smart contract, és a dir aquell qui el desplega.
- Strings: Proporciona funcions útils per poder treballar amb cadenes de caràcters.

5.1.4 IPFS

IPFs és un protocol i sistema d'arxius descentralitzat dissenyat per crear una xarxa global peer to peer d'emmagatzemament i distribució d'arxius. En comptes de guardar els NFT en un servidor centralitzat com podria ser el nostre ordinador, IPFS fa servir una xarxa de nodes connectats entre si per guardar i distribuir arxius en tot el món. Per fer ús IPFS he utilitzat NFTStorage [11]. En el projecte es fa servir perquè els nostres NFT siguin accessibles per tothom i no depenguin de si els tenim guardats de manera local.

5.1.5 Ethers.js

Ethers.js simplifica la comunicació amb la xarxa Ethereum en proporcionar una API robusta i fàcil de fer servir. Permet als desenvolupadors enviar i signar transaccions, interactuar amb contractes intel·ligents, consultar saldos de comptes, fer crides a contractes, escoltar esdeveniments de contractes i molt més. Al projecte s'ha utilitzat per connectar el Front End al smart contract.

5.1.6 Metamask

Metamask és una extensió de navegador que actua com una cartera digital que permet als usuaris interactuar amb aplicacions descentralitzades (dApps) basades en la tecnologia

blockchain. Metamask proporciona als usuaris una cartera digital en què poden emmagatzemar, enviar i rebre criptomonedes, com Ether (ETH), així com tokens ERC-20 i altres actius digitals compatibles amb Ethereum. En el projecte aquesta tecnologia és essencial per connectar-te com a usuari a la blockchain d'Ethereum i poder fer crides als smart contracts.

5.1.7 React.js

React és una biblioteca de JavaScript de codi obert desenvolupada per Facebook que es fa servir per construir interfícies d'usuari (UI) interactives i reactives. Es basa en el concepte de components reutilitzables, cosa que significa que les interfícies es divideixen en components independents que es poden reutilitzar i combinar per construir aplicacions complexes. En el desenvolupament de dApps, s'utilitza per construir interfícies d'usuari interactives i reactives. Els components reutilitzables, l'estat i props, el renderitzat condicional i la integració amb contractes intel·ligents i blockchain són aspectes clau en utilitzar React en el desenvolupament de dApps.

5.1.8 Chainlink

Chainlink és una xarxa descentralitzada i una plataforma d'Oracle a l'ecosistema de blockchain. El seu objectiu principal és proporcionar dades i serveis del món real a contractes intel·ligents d'una blockchain, permetent així la interconnexió entre els contractes intel·ligents i fonts externes d'informació. Com que les blockchains són sistemes deterministes, no poden tenir aleatorietat. En el projecte farem servir els VRFs de Chainlink que són una manera d'aconseguir els nombres aleatoris. Aquests nombres aleatoris ens interessen, ja que quan una persona faci mint dels NFT, volem que li surti un dels 1000 de manera aleatòria, i no que vagi fent mint en ordre, pel fet que d'aquesta manera potser els últims NFT mai sortirien.

5.1.9 The Graph

The Graph és una infraestructura i una eina utilitzada en el desenvolupament de dApps (aplicacions descentralitzades) per facilitar l'obtenció i el processament eficient de dades de blockchain. The Graph permet indexar i consultar dades específiques de la blockchain de manera ràpida i senzilla, cosa que millora el rendiment i l'escalabilitat de les dApps. S'ha utilitzat The Graph, per accedir a dades específiques de la blockchain de manera més eficient i optimitzada. Ens permet, a la nostra dApp, mostrar informació en temps real, com saldos de comptes, registres de transaccions, esdeveniments de contractes intel·ligents.

5.2 Creació de la col·lecció de NFTs

5.2.1 Disseny gràfic

Aquest ha estat el primer pas del meu treball, la creació dels NFT. Per tal de crear-los, he utilitzat PhotoShop, on he utilitzat una estructura de capes, és a dir, per cada part del NFT com per exemple la boca, la samarreta o els ulls, he creat més d'una versió, de manera que per cada capa hi ha diferents versions. El meu NFT es basa en 7 capes que estan sobre posades entre elles, començant per les de sota serien: el fons, el cos, els ulls, el barret (o orelles), la boca, el nas i la samarreta. I de cada capa hi ha diferents versions. 5 fons diferents, 1 cos, 5 ulls, 3 barrets, 6 boques, 1 nas i 6 samarretes. Això permetria tenir un total de 2700 NFT, però per aquesta col·lecció limitarem el número a 1000 NFT. Un cop creades les diferents capes d'imatges, he fet servir un programa en JavaScript [12] que barreja les diverses capes, és a dir, per cada capa agafa una imatge i les ajunta per tal de crear un NFT. Per exemple podem veure que el resultat d'un NFT seria aquest:

Fig. 1: Raccoons 337

En aquest NFT podem veure que té les 7 capes diferenciades que serien les explicades anteriorment. A l'hora d'ajuntar les diverses capes cada imatge d'una capa té una probabilitat associada, per exemple el barret que té aquest NFT és l'element que menys probabilitat té a què surti en un personatge. Llavors aquest programa s'executa i realitza un total de 1000 NFT que tenen característiques diferents i cap és igual a un altre. Com s'ha comentat anteriorment, en comptes de guardar els NFT en un servidor centralitzat com podria ser el nostre ordinador, s'ha utilitzat IPFS.

5.2.2 Smart Contract

En aquesta part del treball s'ha desenvolupat el smart contract relacionat als NFT per poder interactuar amb la nostra col·lecció creada. Totes les funcions a les que es fa referència es poden trobar al Apendix 1.2.

L'objectiu d'aquest contracte és poder permetre a qualsevol usuari de la blockchain, de Goerli en aquest cas, poder interactuar amb aquest contracte per tal de poder aconseguir un NFT de la col·lecció de Raccoons. Per tal d'aconseguirne un n'haurà de pagar un mínim de 0.01 ETH. Per tal de poder dur a terme el desenvolupament d'aquest smart contract hem hagut d'importar smart contracts d'OpenZeppelin que s'han comentat abans.

A més, com s'ha explicat anteriorment, he utilitzat Chainlink per poder afegir aleatorietat a la creació de NFTs. Per tal d'utilitzar els VRFs de Chainlink [13] hem utilitzat el mètode de subscription [14] que consisteix a crear un compte de subscripció i afegir-li tokens LINK. De manera que podem autoritzar el smart contract a la nostra subscripció fent servir un subscriptionId perquè l'smart contract pugui sol·licitar aquests nombres aleatoris [15]. El detall del funcionament és el següent:

Per tant el procés que es seguiria seria:

1. Un usuari crida la funció `requestNFT()` per fer mint del NFT
2. La funció requestNFT cridarà a requestRandomWords en el coordinador VRF

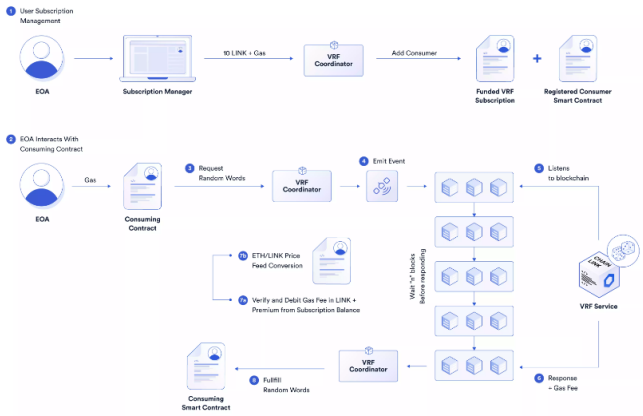

Fig. 2: Chainlink VRF

3. El node VRF ens contesta amb el número random
4. El coordinador VRF verifica i fa un callback amb la funció `fulfillRandomwords()`
5. S'executa aquesta funció generant un NFT random

Principalment, el smart contract està format per tres funcions.

En primer lloc, la funció `requestNFT` que és la que crida un usuari en fer la generació d'un NFT. Es comprova que l'usuari estigui pagant la comissió per la generació que en el nostre cas hem fixat a 0,01 ETH i en cas que no, es rebutjarà la transacció. Després es farà una crida a `requestRandomWords()` que ens retornarà el nombre aleatori després de realitzar tota la lògica explicada anteriorment. Un cop amb el nombre aleatori, s'assignarà aquest tokenID a qui ha cridat la funció.

La segona funció és la de `fullfillRandomWords`, que és la funció que cridarà com a callback el coordinador VRF i és realment on es farà la generació del NFT. Es sobreescriu la funció, ja que és una funció creada en un altre contracte heretat i la volem modificar a la nostra manera. El que fem inicialment és assignar el propietari del NFT que estem generant. Seguit el que farem serà agafar el randomWords[0], que és un array amb el nombre de nombres aleatoris que hem demanat al VRF, que en el nostre cas és un, i fem un mòdul de 1000, ja que tenim 1000 NFT. Amb aquest número tenim l'ID que es generarà i el passarem a una funció que comprova que aquest ID no hagi estat ja generat per un altre usuari. Un cop comprovat, afegeix aquest ID a la llista dels ID generats i crida la funció `safeMint` passant-li el propietari del NFT i l'ID. Finalment assigna l'URI d'aquest NFT.

Finalment, tenim la funció withdraw que ens permet com a propietari del smart contract retirar totes les comissions que ens han pagat per fer la generació dels NFT.

5.2.3 FrontEnd

Com ja s'ha comentat anteriorment, per facilitar la interacció dels usuaris amb el smart contract s'ha desenvolupat un Front End amb React. Per tal de poder utilitzar-lo hem de tenir en compte diversos factors:

- Aquest front-end ha de tenir connexió amb la blockchain de manera que es pugui comunicar de manera sencilla amb el nostre smart contract. Per això cal utilitzar llibreries com la d'ethers.js en aquest cas.
- La nostra aplicació està dividida en diversos components en el que cadascun té una funció diferent, com per exemple el component del Header que té un botó per connectar la teva wallet al Front End o el component de MintNFT que és l'encarregat d'interactuar amb el smart contract.
- Utilització de state Hooks que ens permetin anar actualitzant de manera automàtica variables que facin referència al nostre smart contract.

El nostre front end es troba al fitxer index.js. Inicialment, importem els components que estarem utilitzant, en aquest cas, el Header i el MintNFT. A més a més, importarem la llibreria useMoralis de react-moralis. Pel que fa als components:

En primer lloc el Header, utilitza un component de la llibreria web3uikit [16] anomenat ConnectButton que ens permet agregar un botó perquè l'usuari pugui connectar la seva wallet al nostre front-end.

En segon lloc, MintNFT és l'encarregat a realitzar totes les crides al nostre smart contract. En primer lloc, importem l'ABI i l'adreça del smart contract que desplegem. Aquests paràmetres es passen del script que hi ha al BackEnd 02-update-front-end, que envia aquesta informació al path d'aquest projecte. També es fa servir useEffect i useState de la llibreria react per tal d'utilitzar State hooks. A més a més, es fan servir components de web3uikit com l'useNotification o Button. I finalment la llibreria ethers per poder comunicar-nos amb la blockchain. Bàsicament, el que es fa en aquest component és crear el botó de Mint NFT fent servir la llibreria web3uikit. Dintre d'aquest botó, un cop es clica, fa una crida a la funció `requestNFT` del nostre smart contract. A la funció de `requestNFT` se li passa l'ABI i l'adreça del nostre smart contract, el nom de la funció, el valor d'ETH que passem, (que seria la comissió per la generació) i els paràmetres que en aquest cas no hi ha. La comissió per la generació l'obtenim fent una crida al getter del smart contract i guardant-la en un State hook. Amb tots els getters farem el mateix, de manera que cada cop que es cridin, podrem actualitzar-los. Si l'execució de la funció `requestNFT` és satisfactòria, crida a la funció `handleSuccess` que actualitza els valors dels State hooks i emet una alerta indicant que la transacció s'ha completat.

Finalment, a sota del botó de generació dels NFT, hi ha una mica d'informació com per exemple el preu de la generació 0.01 ETH, i la quantitat d'NFT que queden. Per agafar aquesta informació només fa falta cridar els state hooks que hi ha declarats al codi. El resultat és un front end com el que es mostra a la Figura 3.

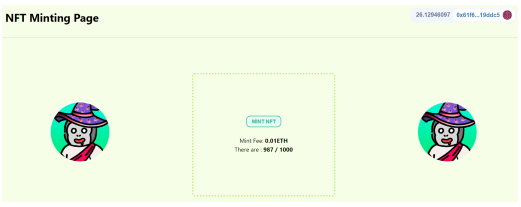

Fig. 3: Front-end Raccoons

Per tal que el nostre front-end sigui accessible per tothom està hostejat a Vercel, una aplicació per poder hostejar de manera gratuïta aplicacions en react. Està enllaçat al repositori de github de manera que si es fa algun canvi, s'actualitza.

5.3 Creació del marketplace

Aquesta fase del projecte ha estat la més llarga, la que més temps he trigat a fer desenvolupar i la que més m'ha costat realitzar. Un cop finalitzada la pàgina on qualsevol persona pot fer la generació d'un NFT de la meva col·lecció, el treball a fer en aquesta fase era desenvolupar un Marketplace on aquelles persones que són propietaris de qualsevol NFT de la col·lecció pugui vendre els seus NFT o que una persona que no en té cap, pugui comprar-ne un. Per dur a terme aquest Marketplace, de la mateixa manera que per la pàgina de la col·lecció dels NFT, necessitarem un Back End és a dir un smart contract i a diferència de la primera part del treball, utilitzarem el protocol de The Graph [17]. D'altra banda, hi haurà un Front End que serà el que qualsevol usuari que entri a la pàgina web veurà.

El market place esta format principalment per dues parts:

- Back end: està format pel smart contract que està escrit en el llenguatge de Solidity, que permet que l'usuari pugui fer la lògica del Marketplace. A més a més del smart contract, també utilitzo The Graph per llegir els events que emet el contracte.

- Front end: fa ús del framework de React i JavaScript per poder crear una pàgina on l'usuari es podrà connectar utilitzant el seu wallet, i així poder interactuar de manera còmoda amb el contracte creat al back end.

5.3.1 BackEnd

Aquesta part consisteix en el desenvolupament del smart contract. El seu objectiu és que qualsevol wallet pugui interactuar amb el Marketplace, sigui per llistar, vendre o comprar un NFT. Aquest contracte també permetrà a aquells usuaris que venguin els seus NFT al Marketplace poder retirar els seus ethers al seu wallet. En aquest smart contract he utilitzat els modifiers de Solidity. Un modifier és codi que pot executar-se abans i/o després d'una crida a una funció. Es pot utilitzar per restringir l'accés, validar els inputs o per protegir el contracte d'un reentrancy attack [18], que més endavant s'explicarà que és.

Per poder entendre millor l'explicació del smart contract, en primer lloc, s'explicaran les variables d'estat [19]. Hi ha les següents:

- Struct: Tenim un struct de Listings que té com a valors un preu i l'adreça del venedor del NFT.

- Mappings [20]: Tenim dos tipus de mappings, en primer lloc, tenim un mapping que guardarà els diners que té un venedor, per tant, serà un mapping entre la seva adreça i un valor (uint256). Per altra banda, tenim un mapping anidat, és a dir que farem un mapping de l'adreça del NFT amb el seu ID, i el struct que he explicat anteriorment, de manera que podrem guardar tota la informació sobre el NFT, el seu venedor i el preu.

Aquest smart contract està format principalment per cinc funcions principals. En primer lloc, la funció d'`itemList`, pren com a paràmetres l'adreça del NFT que volem llistar, el tokenID del NFT i el preu al qual volem vendre el NFT. El seu objectiu és llistar els NFT al Marketplace. És una funció externa, és a dir, només altres contractes o comptes poden cridar-la. Primer de tot, comprova si el token està llistat amb el modifier notListed, en cas que estigui llistat l'NFT, farà una reversió [21] de la transacció. Després comprova si l'adreça que crida la funció és el propiertari del NFT que vol llistar, en cas negatiu, farà una reversió de la transacció. Un cop comprovat que l'adreça que vol llistar un NFT sigui el propiertari d'aquest NFT i que no estigui llistat al Marketplace, comprovarà que el preu al qual el vol vendre sigui major que 0. Seguit comprova si l'adreça que està cridant el contracte té permís per gestionar l'NFT. En cas positiu, actualitza el mapping anidat s-listing amb un struct que té el preu i l'adreça del propietari i emet un event de què l'NFT ha estat llistat. Això farà que faci dues transaccions, una en la que el Marketplace demana permís a l'owner per poder fer transaccions amb aquest NFT i l'altra per fer la funció de llistat al Marketplace.

En segon lloc, tenim la funció d'`itemBuy`. Aquesta funció pren com a paràmetres l'adreça del NFT i el tokenID. El seu objectiu és que qualsevol usuari pugui comprar un NFT que estigui llistat al marketplace. Inicialment comprova que l'NFT que s'està comprant estigui llistat. En cas que estigui llistat, comprova si el que estem pagant és major o igual que el preu al qual està llistat, en cas que sigui menor farà una reversió de la transacció. El que farà un cop comprovat això serà afegir els ethers de l'NFT venut al mapping de s-balance del venedor del NFT i esborrar l'NFT del mapping dels elements llistats. El que farem com a pas final serà utilitzar la funció safeTransferFrom [22] de l'estàndard ERC721, per tal de protegir de Reentrancy attacks. L'ordre en què executem aquestes funcions és molt important per protegir-nos d'aquest atac. És un dels atacs més coneguts i perillosos d'Ethereum actualment per això més endavant explicaré amb més detall que és aquest atac i realment com he protegit el contracte d'això. Finalment, emet un event de què l'NFT ha estat comprat.

En tercer lloc, està la funció de `cancelListing` que els seus inputs són l'adreça del NFT o el seu id. Com el seu nom ja indica, l'objectiu d'aquesta funció és que un usuari que ha llistat el seu NFT pugui cancel·lar el llistat. Inicialment, comprova que l'usuari és el propietari del NFT. Després comprova que l'NFT estigui llistat al marketplace. Finalment, elimina l'NFT del mapping dels items llistats (s-listing) i emet un event de què l'NFT ha estat esborrat del marketplace.

En cuart lloc hi ha la funció de `updateListing`. Aquesta funció pren com a paràmetres d'entrada l'adreça del NFT, el seu id i el nou preu del NFT llistat. El seu objectiu és que un usuari que hagi llistat el seu NFT al marketplace pugui canviar el preu. Comprova si l'NFT està llistat. Després comprova que l'adreça que vol llistar el NFT sigui el propietari i en cas que si, canvia el preu del nft per al nou preu i emet un event de què el NFT ha canviat de preu.

L'última funció és la de `withdraw`, que permet que qualsevol usuari que hagi fet una venda d'un NFT al Marketplace pugui retirar els seus ethers del Marketplace, ja que quan un usuari ven, es queden els ethers al Market-

place. No té inputs. El primer pas és guardar el balanç de l'adreça que crida la funció i comprovar que el seu balanç sigui major que 0. En cas que sigui major li canvia el balanç a 0 i fa una crida per enviar-li els ethers que tenia al market place. Si la crida falla, fa un revert de la funció.

5.3.2 FrontEnd

Un cop realitzat l'smart contract i ja està la primera part del Back End preparada, s'ha desenvolupat el Front End amb el que qualsevol persona podrà interactuar si vol utilitzar el Marketplace. Aquest Front End segueix la mateixa estructura que el de la pàgina de generació dels NFT. S'ha utilitzat React, que és una biblioteca de JavaScript molt utilitzada per la construcció d'aplicacions descentralitzades (DApps). Per tal de connectar el Front End a l'Smart contract de manera senzilla i que pugui comunicar-se amb la cadena de blocs, s'ha utilitzat la llibreria d'ethers.js. El Front End està format per diversos components on cadascú té una funció diferent, que s'explicarà més en detall després. A més a més, ja que s'està fent servir React, s'ha fet ús de State hooks per actualitzar de manera automàtica les variables que s'agafen del smart contract.

Al Marketplace hi ha tres pàgines. En primer lloc, tenim el index.js que seria la pàgina de Home del nostre market place (Figura 4). En segon lloc, la pàgina del perfil d'usuari (Figura 5), on un usuari connectat amb el seu wallet al Marketplace podrà veure els NFT que té a la seva wallet (de moment només podrà veure aquells NFT que siguin de la col·lecció de Raccoons). Finalment, la pàgina de venda dels NFT (Figura 6), on qualsevol usuari propietari d'un NFT de la meva col·lecció podrà llistar el seu NFT al Marketplace.

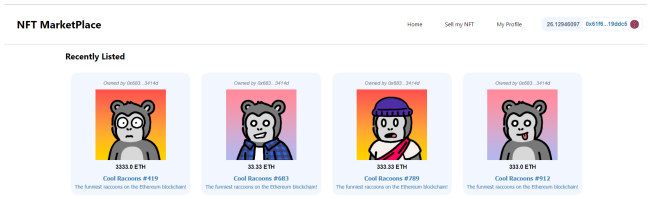

Fig. 4: Home page

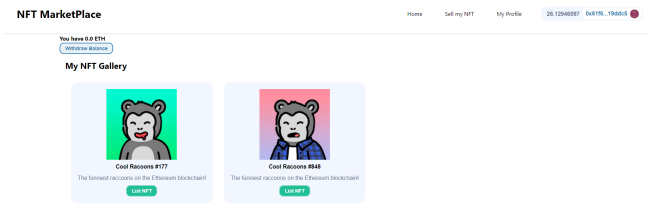

Fig. 5: Profile page

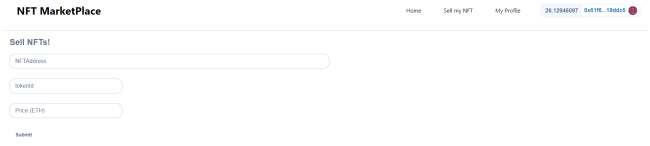

Fig. 6: Sell page

Com que per fer el Front End s'ha fet servir el framework de React, s'han utilitzat components per poder mostrar-los depenent de la interacció de l'usuari. En primer lloc, tenim el component del Header, que mostranel menú de navegació de la part superior del Marketplace així com el botó de connect. És el mateix component que el Front End de la pàgina dels NFT.

Fig. 7: Header component

El segon component desenvolupat ha estat el d'NFTBox, la funció visual d'aquest component és que l'usuari pugui veure els NFT i alguna informació sobre aquests, però realment és el component més important. La seva funcionalitat és, en primer lloc, poder veure el NFT amb el seu propietari, el nom de l'NFT i una descripció. Si ets el propietari de l'NFT pots canviar el seu preu. També podrà cancel·lar un llistat. Si cliques en un NFT que no és teu el podràs comprar. Totes les funcions que fa aquest component, per darrere fan crides a les funcions del smart contract utilitzant ethers.js com ja havia fet anteriorment en el cas del Front End del NFT. El component es veu tal com mostra la Figura 8:

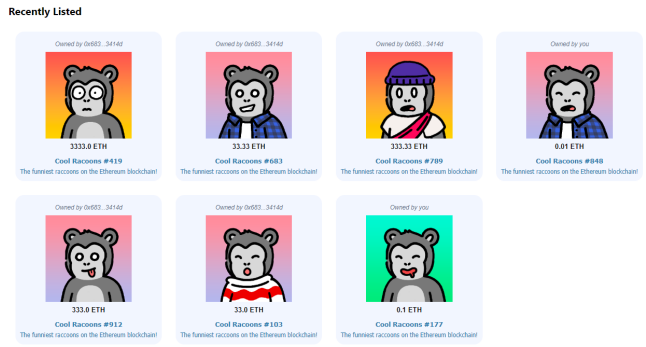

Fig. 8: NFTBox Component

Finalment, tenim el component de ListingUpdate, que la seva funcionalitat és que un cop un usuari clica en un NFT del qual és owner se li obre un component que li permet canviar el preu del seu NFT.

Fig. 9: ListingUpdate Component

5.3.3 Seguretat del smart contract

Abans de començar a explicar l'atac explicaré com funcionen les funcions fallback [23] en un contracte. En Solidity un contracte pot tenir una funció sense nom, que no tingui arguments ni retornar res. Les funcions fallback s'executen si es crida a un contracte i cap altra funció coincideix amb l'identificador de funció especificat o si no se subministren dades. Aquestes funcions també s'executen sempre que un contracte rebi ethers, sense cap dada, en aquest cas la funció

ha de ser payable. Com s'ha comentat abans, s'ha protegit l'smart contract d'un atac conegut com a atac de reentrada (Reentrancy Attack [18]). És un tipus d'atac que pot passar en contractes intel·ligents que permeten l'execució de codi extern no fiable dins del contracte. Això pot passar quan un contracte intel·ligent crida a un contracte extern, i el contracte extern torna a cridar al contracte original, causant potencialment un bucle infinit. Veiem un exemple del que no hauríem de fer (Apendix 2.a) i el codi de l'atacant (Apendix 2.b). Ens fixarem en la següent línia de codi (Figura 10):

```
function attack() external payable {
    require(msg.value >= 1 ether);
    etherStore.deposit{value: 1 ether}();
    etherStore.withdraw();
}
```

Fig. 10: Línea vulnerable

Es crida a la funció del smart contract per dipositar 1 ether i seguidament es crida la funció withadraw. Ara veurem l'atac. En el codi de l'atacant té una funció de fallback maliciosa (Figura 11):

```
// Fallback is called when EtherStore sends Ether to this contract.
fallback() external payable {
    if (address(etherStore).balance >= 1 ether) {
        etherstore.withdraw();
    }
}
```

Fig. 11: Funció fallback

Aquesta funció crida de manera recursiva la funció withdraw, que és on es troba la vulnerabilitat. Si ara ens fixem en la funció de withdraw del smart contract, (Figura 12) és una crida externa a la funció fallback del atacant abans que la funció de withdraw es completi.

```
function withdraw() public {
    uint bal = balances[msg.sender];
    require(bal > 0);

    (bool sent, ) = msg.sender.call{value: bal}("");
    require(sent, "Failed to send Ether");

    balances[msg.sender] = 0;
}
```

Fig. 12: Funció withdraw

Per tant, aquesta funció es cridarà de manera recursiva fins que dreni tots els ethers del smart contract. Per protegir el contracte d'aquest atac n'hi ha diverses tècniques, la que jo he utilitzat ha estat el CEI Pattern [24]. Si posem a zero el balanç de l'usuari abans de fer la crida per passar els ethers a l'altre contracte, la funció de fallback no podrà treure els diners, ja que no hi haurà de més.

6 Conclusions

En el següent apartat s'explicarà les conclusions del projecte així com els coneixements assolits durant la realització i les possibles millores que podria tenir el meu treball. Durant el desenvolupament del projecte, penso que he assolit diversos coneixements necessaris per poder desplegar una dApp en la xarxa d'Ethereum.

D'una banda, he pogut reforçar els meus coneixements sobre el funcionament de la blockchain i he aprofundit en el funcionament de la blockchain d'Ethereum. Una part a destacar aquí ha estat la manera de connectar la blockchain amb el Front End, on he pogut entendre com una aplicació descentralitzada pot connectar-se a la blockchain a través d'ethers.js i Metamask. Pel que fa als smart contracts, he pogut aprendre a desenvolupar contractes de major nivell on es té en compte la seguretat, permetent així que el nostre contracte no sigui vulnerable davant un atacant. També el fet d'aprendre a utilitzar tecnologies com IPFS i The Graph, ens ha permès aconseguir el meu objectiu d'utilitzar tecnologies de Web3 per fer el projecte. Pel que fa als problemes trobats s'han de destacar dos: En primer lloc, vaig tenir molts problemes per poder indexar de manera correcta els events de l'smart contract a The Graph, però finalment ho vaig solucionar. Vaig haver de crear un altre repositori des del qual faig les consultes al meu subgraph. En segon lloc, un problema amb què em vaig trobar va ser al moment de fer el component de my profile. Per tal que els NFT que té una wallet es puguin veure he fet ús del SDK d'Alchemy [25], fent servir la funció `getNftsForOwner` [26] que ens permet veure els NFT que té una adreça. El problema és que això retorna tots els NFT inclosos aquells que no són la meva col·lecció, llavors aquells que no tinc l'ABI afegit al projecte no es poden visualitzar. Per solucionar això el que he fet ha sigut que el Marketplace només es puguin veure els de la meva col·lecció simplificant així el Marketplace.

Per l'altra banda, pel que fa al desenvolupament web, he pogut aprendre a desenvolupar un Front End amb React.js que mai l'havia fet servir i he hagut d'aprendre a utilitzar-lo. Per tant, he assolit un dels meus objectius, utilitzar llibreries de JavaScript per desenvolupar el Front End. Pel que fa a aquesta part, es podria haver aprofundit una mica en el desenvolupament de la web i afegir noves funcionalitats.

6.1 Línies Futures

Aquest apartat té com a objectiu explicar possibles funcionalitats addicionals que es podrien crear al meu projecte.

En primer lloc, seria el fet d'utilitzar crides directament a la blockchain i no utilitzar The Graph, per fer el projecte més descentralitzat. Al projecte jo he utilitzat The Graph però hi ha altres maneres de llegir aquests events a la cadena de blocs. La forma més descentralitzada que potser hagués estat la millor opció, seria utilitzar la llibreria de ethers per exemple per poder llegir els events de la cadena de blocs. Un avantatge d'això seria que la nostra aplicació seria més descentralitzada, ja que estem llegint de la cadena de blocs, però les dades serien més difícils de manejar. Si utilitzem The Graph, es poden manejar millor les dades fent consultes amb GraphQL i ens serà més fàcil que utilitzar les dades de la cadena de bloc directament, però, en canvi, la nostra aplicació estarà utilitzant tercers que poden ser centralitzats. En segon lloc, en comptes de que quan un usuari clica a un NFT el compri de manera directa, que aquest usuari pugui ser capaç de veure informació d'aquest NFT. És a dir, que pugui veure les característiques que té, el percentatge de NFT amb aquestes característiques, les transaccions que ha tingut, etc. En tercer lloc, seria poder permetre que qualsevol col·lecció d'NFT pugui ser llistada a Marketplace.

AGRAÏMENTS

M'agradaria agrair a totes les persones que han contribuït de manera significativa en el desenvolupament i finalització d'aquest treball de fi de grau. En primer lloc, agrair al meu tutor Jordi Herrera pels seus consell durant tot el transcurs del treball. I en segon lloc, donar les gràcies als meus familiars i amics pel seu continu suport.

REFERÈNCIES

[1] ≪¿Qué es la tecnología blockchain? - IBM Blockchain,≫ [En línea]. Available: https://www.ibm.com/es-es/topics/what-is-blockchain.

[2] G. W. Andreas M. Antonopoulos, ≪GitHub - Mastering Bitcoin 2nd Edition - Programming the Open Blockchain,≫ [En línea]. Available: https://github.com/bitcoinbook/bitcoinbook.

[3] G. W. Andreas M. Antonopoulos, ≪Github-Mastering Ethereum,≫ [En línea]. Available: https://github.com/ethereumbook/ethereumbook.

[4] N.-f. t. (NFT), ≪ethereum.org,≫ [En línea]. Available: https://ethereum.org/en/nft/.

[5] "ERC721," OpenZeppelin Docs, https://docs.openzeppelin.com/contracts/3.x/erc721

[6] ≪IPFS Powers the Distributed Web,≫ [En línea]. Available: https://ipfs.tech/.

[7] ≪What Are Smart Contracts? Ïntroduction — Chainlink,≫ [En línea]. Available: https://chain.link/education/smart-contracts.

[8] ≪Solidity,≫ [En línea]. Available: https://docs.soliditylang.org/en/v0.8.19/.

[9] ≪Trello brings all your tasks, teammates, and tools together,≫ [En línea]. Available: https://trello.com/.

[10] ≪HashLips/hashlips_art_engine: HashLips Art Engine is a tool used to create multiple different instances of artworks based on provided layers,≫ [En línea]. Available: https://github.com/HashLips/hashlips_art_engine.

[11] ≪NFT.Storage - Free decentralized storage and bandwidth for NFTs on IPFS Filecoin.,≫ [En línea]. Available: https://nft.storage/.

[12] MozDevNet, "JavaScript," MDN, https://developer.mozilla.org/es/docs/Web/JavaScript

[13] ≪blockchain oracles,≫ [En línea]. Available: https://chain.link/education/blockchain-oracles

[14] ≪Subscription Method — Chainlink Documentation,≫ [En línea]. Available: https://docs.chain.link/vrf/v2/subscription/ .

[15] ≪Get a Random Number — Chainlink Documentation,≫ [En línea]. Available: https://docs.chain.link/vrf/v2/subscription/examples/get-a-random-number/.

[16] [En línea]. Available: https://github.com/web3ui/web3uikit.

[17] ≪The Graph Docs,≫ [En línea]. Available: https://thegraph.com/docs/en/.

[18] ≪Reentrancy After Istanbul - OpenZeppelin blog,≫ [En línea]. Available: https://blog.openzeppelin.com/reentrancy-after-istanbul.

[19] ≪Reading and Writing to a State Variable — Solidity by Example — 0.8.17,≫ [En línea]. Available: https://solidity-by-example.org/state-variables/.

[20] ≪Reading and Writing to a State Variable — Solidity by Example — 0.8.17,≫ [En línea]. Available: https://solidity-by-example.org/state-variables/.

[21] ≪Expressions and Control Structures - Solidity 0.8.13 documentation,≫ [En línea]. Available: https://docs.soliditylang.org/en/v0.8.13/control-structures.html#error-handling-assert-require-revert-and-exceptions.

[22] ≪ERC 721 - OpenZeppelin Docs,≫ [En línea]. Available: https://docs.openzeppelin.com/contracts/2.x/api/token/erc721/IERC721-safeTransferFrom-address-address-uint256-bytes-.

[23] ≪Fallback — Solidity by Example — 0.8.17,≫ [En línea]. Available: https://solidity-by-example.org/fallback/.

[24] ≪Checks Effects Interactions — solidity-patterns,≫ [En línea]. Available: https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html.

[25] ≪Alchemy SDK - The Best Way to Develop Web3 Dapps,≫ [En línea]. Available: https://www.alchemy.com/sdk.

[26] ≪How to Check the Owner of an NFT,≫ [En línea]. Available: https://docs.alchemy.com/docs/how-to-check-the-owner-of-an-nft.

APÈNDIX

1.2 NFT Smart Contract

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.7.0 <0.9.0;

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@chainlink/contracts/src/v0.8/interfaces/VRFCoordinatorV2Interface.sol";
import "@chainlink/contracts/src/v0.8/VRFConsumerBaseV2.sol";
import "@openzeppelin/contracts/utils/Strings.sol";

contract Raccoons is VRFConsumerBaseV2, ERC721URIStorage, Ownable {
    // Eventss
    event NFTRequested(uint256 indexed requestId, address requester);
    // event NFTMinted(NinjaType ninjaType, address minter);
    address private _owner;

    // Variables NFT
    uint256 internal immutable i_mintFee;
    mapping(uint256 => address) public s_requestIdToSender;
    uint256 private s_tokenCounter;
    string private baseURI = "ipfs://bafybeih56fprl5vsfx75baxwsyp4t6qd2hop7ujmabxd
    uint256[] public tokensID;
    uint256 private constant NUM_NFTS = 1000;

    // Variables Chainlink VRF
    VRFCoordinatorV2Interface private immutable i_vrfCoordinator;
    uint64 private immutable i_subscriptionId; // get subscription ID from vrf.chain.link
    bytes32 private immutable i_keyHash;
    uint16 private constant REQUEST_CONFIRMATIONS = 3;
    uint32 private immutable i_callbackGasLimit;
    uint32 private constant NUM_WORDS = 1;

    constructor(
        uint256 mintFee,
        address vrfCoordinatorV2Address,
        uint64 subscriptionID,
        //string memory _initBaseURI,
        bytes32 keyHash,
        uint32 callbackGasLimit
    ) VRFConsumerBaseV2(vrfCoordinatorV2Address) ERC721("Raccoons", "COONS") {
        i_mintFee = mintFee;
        s_tokenCounter = 0;
        _owner = msg.sender;
        // Variables Chainlink VRF
        i_vrfCoordinator = VRFCoordinatorV2Interface(vrfCoordinatorV2Address);
        i_subscriptionId = subscriptionID;
        i_keyHash = keyHash;
        i_callbackGasLimit = callbackGasLimit;
    }

    function requestNFT() public payable returns (uint256 requestId) {
        // mintfee is paid?
        require(msg.value >= i_mintFee, "Not enough mint fee!");

        // keyHash: specifies which gas lane to use, which is how much maximum gas
        //price you're willing to pay (in Wei) for a randomness request
        // subscriptionID: he subscription Id that you get after creating a subscription
        //from the Subscription Manager at vrf.chain.link
        // REQUEST_CONFIRMATIONS: how many confirmations the VRF node should wait for
        //before responding
        // callbackGasLimit: gas limit for the callback request to our fullRandomWords()
        // NUM_WORDS: how many random numbers do we want to request? In our case, we only want 1.
        requestId = i_vrfCoordinator.requestRandomWords(
            i_keyHash,
            i_subscriptionId,
            REQUEST_CONFIRMATIONS,
            i_callbackGasLimit,
            NUM_WORDS
        );

        // Mapping of caller to their requestsIDs
        s_requestIdToSender[requestId] = msg.sender;

        // Event to log requestID and caller address
        emit NFTRequested(requestId, msg.sender);
    }

    function fulfillRandomWords(
        uint256 requestId,
        uint256[] memory randomWords
    ) internal virtual override {
        // First step - Assign the NFTOwner
        address nftOwner = s_requestIdToSender[requestId];

        // Second step - NFT minting
        //uint256 tokenCounter = s_tokenCounter;
        uint256 randomNumber = (randomWords[0] % 1000) + 1;
        uint256 tokenID = checknumber(randomNumber);

        tokensID.push(tokenID);

        _safeMint(nftOwner, tokenID);

        _setTokenURI(
            tokenID,
            string(abi.encodePacked(baseURI, "/", Strings.toString(tokenID), ".json"))
        );
    }

    function checknumber(uint256 randomNumber) internal view returns (uint256) {
        for (uint256 i = 0; i < tokensID.length; i++) {
            if (randomNumber > 1000) randomNumber = 0;
            if (tokensID[i] == randomNumber) return checknumber(randomNumber + 1);
        }
        return randomNumber;
    }

    function withdraw() public onlyOwner {
        require(msg.sender == _owner, "You are not the owner!");
        uint256 amount = address(this).balance;
        (bool success, ) = payable(msg.sender).call{value: amount}("");

        if (!success) {
            revert("Withdrawal failed!");
        }
    }

    function getMintFee() public view returns (uint256) {
        return i_mintFee;
    }

    function getTotalNFT() public pure returns (uint256) {
        return NUM_NFTS;
    }

    function getMintedNFT() public view returns (uint256) {
        return tokensID.length;
    }
}
```

1.3 MarketPlace Smart Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

error NFTMarketPlace__PriceMustBeAboveZero();
error NFTMarketPlace__NotApprovedForMarketplace();
error NFTMarketPlace__AlreadyListed(address nftAddress, uint256 tokenID);
error NFTMarketPlace__NotOwner();
error NFTMarketPlace__NotListed(address nftAddress, uint256 tokenID);
error NFTMarketPlace__PriceError(
    address nftAddress,
    uint256 tokenID,
    uint256 price
);
error NFTMarketPlace__noBalance();
error NFTMarketPlace__notSuccess();

contract NFTMarketPlace is ReentrancyGuard {
    constructor() {}

    struct Listing {
        uint256 price;
        address seller;
    }

    event ItemListed(
        address indexed seller,
        address indexed nftAddress,
        uint256 indexed tokenID,
        uint256 price
    );

    event ItemBought(
        address indexed buyer,
        address indexed nftAddress,
        uint256 indexed tokenID,
        uint256 price
    );

    event ItemCanceled(
        address indexed seller,
        address indexed nftAddress,
        uint256 indexed tokenID
    );

    // NFT CA -> NFT TokenID -> Listing
    mapping(address => mapping(uint256 => Listing)) private s_listings;
    // Seller Address -> Amount earned
    mapping(address => uint256) private s_balance;

    // Modifiers //
    modifier notListed(address nftAddress, uint256 tokenID) {
        Listing memory listing = s_listings[nftAddress][tokenID];
        if (listing.price > 0) {
            revert NFTMarketPlace__AlreadyListed(nftAddress, tokenID);
        }
        _;
    }

    modifier isOwner(
        address nftAddress,
        uint256 tokenID,
        address spender
    ) {
        IERC721 nft = IERC721(nftAddress);
        address owner = nft.ownerOf(tokenID);
        if (spender != owner) {
            revert NFTMarketPlace__NotOwner();
        }
        _;
    }

    modifier isListed(address nftAddress, uint256 tokenID) {
        Listing memory listing = s_listings[nftAddress][tokenID];
        if (listing.price <= 0) {
            revert NFTMarketPlace__NotListed(nftAddress, tokenID);
        }
        _;
    }

    // MAIN Functions //

    /// @notice Method for listing a NFT in the MarketPlace
    /// @param nftAddress: is the address of the NFT contract
    /// @param tokenID: the token ID of the NFT
    /// @param price: is the price of the listed NFT sale

    function itemList(
        address nftAddress,
        uint256 tokenID,
        uint256 price
    )
        external
        notListed(nftAddress, tokenID)
        isOwner(nftAddress, tokenID, msg.sender)
    {
        if (price <= 0) {
            revert NFTMarketPlace__PriceMustBeAboveZero();
        }
        IERC721 nft = IERC721(nftAddress);
        if (nft.getApproved(tokenID) != address(this)) {
            revert NFTMarketPlace__NotApprovedForMarketplace();
```

```
        }
        s_listings[nftAddress][tokenID] = Listing(price, msg.sender);
        emit ItemListed(msg.sender, nftAddress, tokenID, price);
    }

    function itemBuy(
        address nftAddress,
        uint256 tokenID
    ) external payable isListed(nftAddress, tokenID) {
        Listing memory item = s_listings[nftAddress][tokenID];
        if (msg.value < item.price) {
            revert NFTMarketPlace__PriceError(nftAddress, tokenID, item.price);
        }
        s_balance[item.seller] = s_balance[item.seller] + msg.value;
        delete (s_listings[nftAddress][tokenID]);
        // We put this line at the bottom for protecting agains Reentrancy-attack
        IERC721(nftAddress).safeTransferFrom(item.seller, msg.sender, tokenID);
        emit ItemBought(msg.sender, nftAddress, tokenID, item.price);
    }

    function cancelListing(
        address nftAddress,
        uint256 tokenID
    )
        external
        isOwner(nftAddress, tokenID, msg.sender)
        isListed(nftAddress, tokenID)
    {
        delete (s_listings[nftAddress][tokenID]);
        emit ItemCanceled(msg.sender, nftAddress, tokenID);
    }

    function updateListing(
        address nftAddress,
        uint256 tokenID,
        uint256 priceUpdated
    )
        external
        isListed(nftAddress, tokenID)
        isOwner(nftAddress, tokenID, msg.sender)
    {
        s_listings[nftAddress][tokenID].price = priceUpdated;
        emit ItemListed(msg.sender, nftAddress, tokenID, priceUpdated);
    }

    function withdraw() external {
        uint256 balance = s_balance[msg.sender];
        if (balance <= 0) {
            revert NFTMarketPlace__noBalance();
        }
        s_balance[msg.sender] = 0;
        (bool success, ) = payable(msg.sender).call{value: balance}("");
        if (!success) {
            revert NFTMarketPlace__notSuccess();
        }
    }

    // Getter Functions //

    function getListing(
        address nftAddress,
        uint256 tokenID
    ) external view returns (Listing memory) {
        return s_listings[nftAddress][tokenID];
    }

    function getBalance(address seller) external view returns (uint256) {
        return s_balance[seller];
    }
}
```

2 ATAC DE REENTRADA

```
contract EtherStore {
    mapping(address => uint) public balances;

    function deposit() public payable {
        balances[msg.sender] += msg.value;
    }

    function withdraw() public {
        uint bal = balances[msg.sender];
        require(bal > 0);

        (bool sent, ) = msg.sender.call{value: bal}("");
        require(sent, "Failed to send Ether");

        balances[msg.sender] = 0;
    }

    // Helper function to check the balance of this contract
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```

(a) Codi amb la vulnerabilitat

```
contract Attack {
    EtherStore public etherStore;

    constructor(address _etherStoreAddress) {
        etherStore = EtherStore(_etherStoreAddress);
    }

    // Fallback is called when EtherStore sends Ether to this contract.
    fallback() external payable {
        if (address(etherStore).balance >= 1 ether) {
            etherStore.withdraw();
        }
    }

    function attack() external payable {
        require(msg.value >= 1 ether);
        etherStore.deposit{value: 1 ether}();
        etherStore.withdraw();
    }

    // Helper function to check the balance of this contract
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
}
```

(b) Codi del atacant

Fig. 13: Reentrancy attack