Entrades NFT per a esdeveniments: Implementació de transaccions segures mitjançant Smart Contracts de Ethereum

Treball Final de Grau 2024/25

Informe de Progrés II

UAB
Universitat Autònoma de Barcelona

Lucas Dalmau Garcés – 1636290

ÍNDEX

INTRODUCCIÓ 3

OBJECTIUS 3

BENEFICIS 4

SISTEMES DE TICKETING: COM FUNCIONEN ACTUALMENT? 5

MODEL DE DADES 7

On-Chain 7
Off-Chain 8
MODEL FUNCIONAL/TECH 8

Smart Contract 8
Distributed Application 8
METODOLOGIA APLICADA 9

Marc de treball pel desenvolupament de la Dapp 9
VUE.JS *10*
REACT.JS *10*
Selecció final *11*
Aprenentatge: *11*
ARQUITECTURA TECH I EINES 11

Llibreries utilitzades 11
Wagmi *11*
react-router-dom *12*
react-icons *12*
qrcode.react *12*
Wagmi Hooks Utilitzats 12
AVENÇOS II: PROCÉS DE DESENVOLUPAMENT 13

Prototip 13
Esdeveniments *13*
Pagina de compra d'entrada (tiquets en estoc) *14*
Revenda *14*
Revenda (esdeveniment específic) *15*
Els meus tiquets *15*
Tiquet Adquirit *16*
Colorimetria, nom i icona per la Dapp: 16
Components persistents 17
Account header *17*
Disconnect *17*
ChainSwitcher *17*
Menu *18*
Components de pàgina 18
Events *18*
Resale *22*
MyTickets *25*
Desplegament del Smart Contract a una Testnet 32
OBJECTIUS COMPLETATS 34

CANVIS EN ELS OBJECTIUS 35

EventURI i optimització de bytecode 35
ERC721AQueryable i optimitzacions de bytecode 36
EnumerableSet i gestió de tiquets en venda 38
Implementació d'un esquema commit-reveal per bescanviar entrades 38
Actualització del preu d'un tiquet en revenda 40
Implementació d'un IPFS pels NFT 40
PLANIFICACIÓ DEL TREBALL: SEGUIMENT II 41

Anàlisi de la Segona fase 41
Tercera Fase: Preparació 41
RESULTATS OBTINGUTS 42

CONCLUSIONS PROVISIONALS 43

BIBLIOGRAFIA 45

Introducció

La indústria dels tiquets inicialment va ser creada com una forma d'organització més per poder controlar l'aforament i regular l'accés del públic. Inicialment, s'utilitzaven com a entrades bitllets de papers que podien ser bescanviats pel mateix accés. Per exemple esdeveniments com ara els teatres, utilitzaven aquests tipus de sistemes simples on ja tractaven de fer dissenys únics i especials que reflectien el caràcter de l'esdeveniment.

Una vegada es van implementar els sistemes informàtics, la indústria va evolucionar amb aquests sistemes portant els primers portals de venda en línia, els quals oferien la possibilitat de fer compres anticipades, simplificant l'experiència tant per als usuaris com per als organitzadors, que podien "despreocupar-se" una mica més d'aquesta gestió. Malgrat això, aquests nous avenços també van comportar nous desafiaments importants, com les revendes d'entrades i la falsificació, els quals encara persisteixen avui dia.

Actualment per frenar aquests problemes, s'han presentat solucions on habitualment es tracta de vincular a la persona física amb l'entrada, per exemple, utilitzant el seu identificador, domicili o número de telèfon. Però aquestes solucions donen lloc a llargues polítiques i consentiments en termes de privacitat i tractament de dades sumat al fet que sovint els organitzadors no s'encarreguen de revisar tot això per una entrada, fet que provoca una pèrdua d'efectivitat en aquest tipus de solucions. És per tot això que la descentralització de la blockchain en combinació dels NFTs han aparegut com una solució potencial per tractar aquests problemes, ja que els NFT permeten la creació d'actius digitals únics, inalterables i fàcilment verificables a la vegada que ofereixen una seguretat i transparència úniques. Gràcies a les bondats d'aquestes tecnologies es podrien crear nous sistemes d'entrades els quals poden reduir el frau, la dependència d'intermediaris i l'especulació que comporta a la revenda d'entrades amb preus desorbitats.

Aquest projecte busca explotar aquestes noves tecnologies proposant una solució moderna a problemes actuals mitjançant el desenvolupament de smart contracts[1] per la gestió combinat amb una interfície d'usuari que sigui segura, accessible i estigui alineada amb els objectius i necessitats del sector del ticketing i els esdeveniments.

Objectius

L'objectiu d'aquest treball és demostrar de manera pràctica com les tecnologies blockchain i NFT poden oferir una alternativa real als problemes del ticketing tradicional.

Dintre del primer informe, es va desenvolupar el codi que anirà a la cadena de blocs i conté tota la lògica necessària per poder fer la compra, la revenda i els bescanvis. Però, actualment l'única forma per interactuar amb aquest 'programa' seria que l'usuari per si mateix cridés a les funcions, cos la qual és poc accessible pels usuaris, ja que haurien de tenir coneixements tècnics sobre com funcionen totes aquestes tecnologies. És per això que en aquest segon informe es proposa la creació d'una

[1] Smart contract: Programa que es llença a la cadena de blocs i qualsevol node el pot executar de maner determinista.

interfície d'usuari que permeti gestionar els tiquets NFT sense necessitat d'entendre com funciona la cadena de blocs. Els objectius que cal complir per poder fer una bona interfície d'usuari son:

En primer lloc, s'ha de facilitar la compra de tiquets, de forma que qualsevol persona pugui comprar una entrada per un esdeveniment en un parell de clics, sense fer cues i únicament des del seu navegador i una 'cartera' digital.

En segon lloc, és important que quan un usuari vulgui bescanviar l'entrada per l'accés tot aquest procés sigui segur. És per això que es pretén generar un codi d'un únic ús de forma que només l'usuari que l'ha generat pugui accedir a l'esdeveniment amb el seu tiquet.

Com a tercer punt, la interfície ha de permetre als usuaris revendre els seus tiquets de froma segura, i vigilant que el preu no superi un 30% per sobre del preu de venda en estoc. Encara que el smart contract ja rebutjaria un preu més gran que això, és important que la interfície no permeti llençar una acció com aquesta, ja que si no pot frustrar a l'usuari.

Per últim, la compra de tiquets ha de mantenir la privacitat de l'usuari en tot moment, és a dir, la interfície mai demanarà dades personals a l'usuari ni adreces de correu, n'hi haurà prou amb la seva "cartera" digital per gestionar tot.

Les meves motivacions per aquest projecte és el fet de poder crear un sistema alternatiu als sistemes de compravenda d'entrades tradicionals, crec que és molt interessant una eina que permeti la revenda segura, que pugui mantenir la privacitat dels usuaris i que a l'hora sigui segura i transparent per tothom, A més crec que el fet que les entrades siguin NFT pot ser una bona iniciativa perquè els usuaris tinguin un record de l'esdeveniment en forma d'un art digital inspirat en aquell concert o en aquell partit que van veure. És per tot això que vull combinar la innovació que ofereixen aquestes tecnologies per obtenir una solució moderna i de confiança.

Beneficis

El sistema proposat converteix la compra i gestió d'entrades en un procés transparent i segur per a tots els usuaris. De forma que, qualsevol usuari podria fer un seguiment sobre cada tiquet i verificar a qui pertany el tiquet. A més permet la compra, la reveneda i el bescanvi dels tiquets sense haver de donar dades personals, només amb l'adreça de la wallet [*OBS: una adreça és un identificador públic al qual té una quantitat de monedes associades*]

Desglossant els beneficis particulars del sistema implementat:

- **Bescanvi**: el sistema contempla un esquema commit-reveal amb el qual els usuaris estan segurs de que ningú que no siguin ells poden bescanviar l'entrada per l'accés al esdeveniment.
- **Revenda:** Els tiquets només es poden vendre per un 30% superior al preu de les entrades d'estoc de forma que es frena l'especulació i es protegeix a la comunitat de fans.

Per la banda dels benèfics pels organitzadors, la plataforma elimina els intermediaris i els costos abusius de “gestió” que sovint s’apliquen als usuaris en comprar una entrada pel manteniment del sistema de ticketing tradicional, ja que en aquest sistema tot està basat en una solució distribuïda que a més possibilita el rastreig gràcies al registre immutable dintre de la cadena de blocs. A més, i gràcies aquest registre, també es podran implementar en un futur sistemes que revisin aquests registres de la cadena de blocs per a poder fer anàlisis de vendes, revendes i bescanvis per a poder analitzar les tendències dels usuaris amb la finalitat d’elaborar noves estratègies de màrqueting, saber quines hores són les més habituals de compra/revenda, quins esdeveniments tenen major percentatge de tiquets en revenda i moltes altres mètriques més.

D’altra banda, mitjançant la integració dels tiquets com a NFTs que incorporen un art digital que pertany al usuari, possibilita el col·leccionisme en mercats de revenda d’art, de forma que s’amplia la visibilitat de l'esdeveniment i es fomenten altres models de negoci que retroalimenten l’ús del sistema.

Per últim, el projecte també aporta valor a la comunitat acadèmica i tecnològica gràcies al fet que obre la porta a investigadors a explotar aquesta combinació de la cadena de blocs amb aplicacions distribuïdes, impulsant així l’adopció de solucions descentralitzades en altres sectors que requereixen característiques com les abans mencionades.

Sistemes de ticketing: Com funcionen actualment?

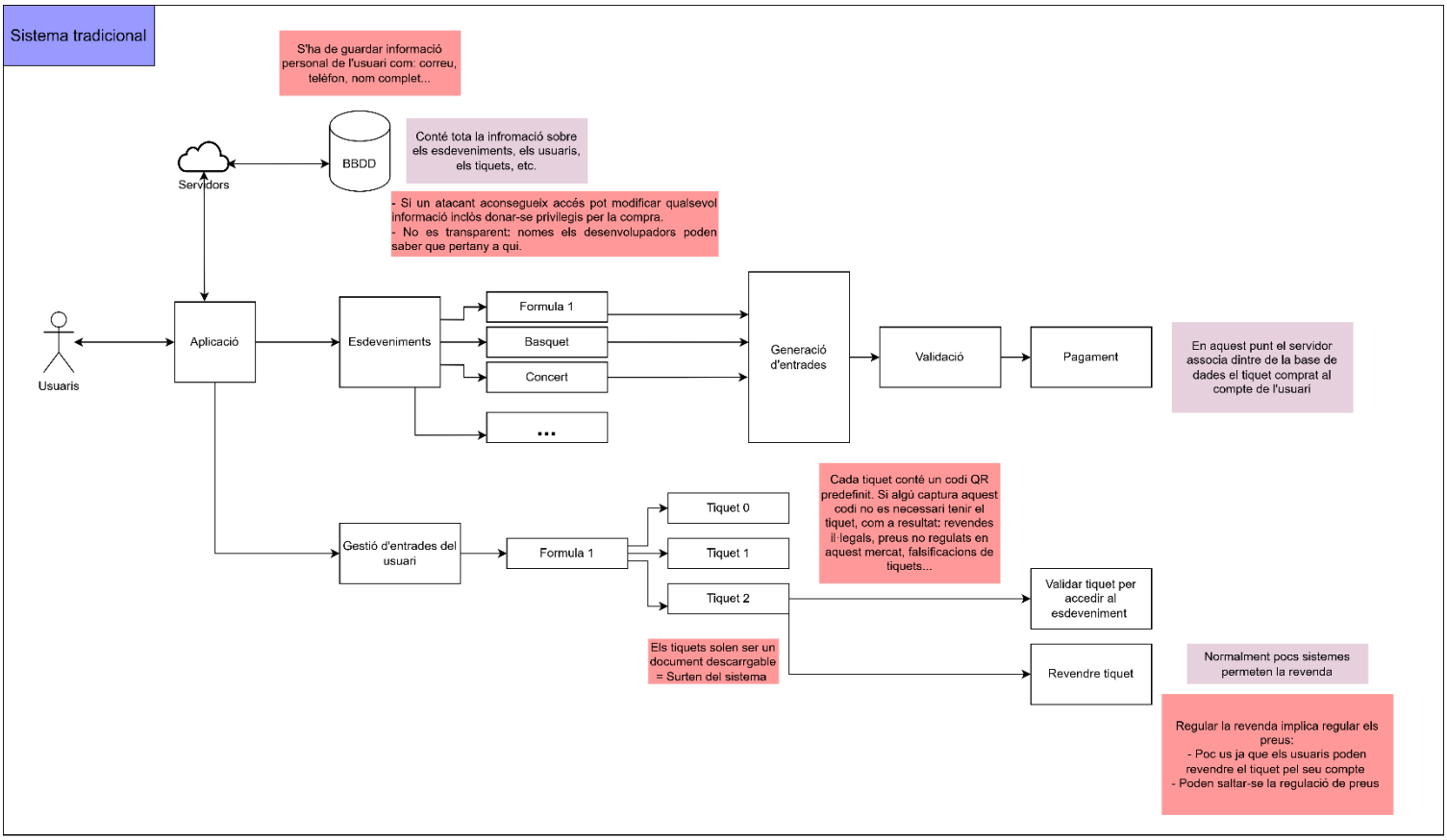
Abans de continuar i aprofundir més en el sistema desenvolupat, analitzem de manera més general el funcionament dels sistemes de ticketing per esdeveniments que s’utilitzen en l’actualitat.

Avui dia, tenim sistemes que es basen en un funcionament bàsic, seguint un esquema tradicional de client i servidor amb una base de dades. On l’usuari ha de registrar-se al sistema i introduir les seves dades personals per a després poder fer la compra de tiquets. Dintre d’aquest model clàssic trobem dos variants:

La primera variant, seria un sistema on una vegada l’usuari fa la compra, el tiquet s’externalitza, sovint en format d’un document PDF que conté el tiquet amb un QR per poder bescanviar-lo per l’accés. Però, això comporta que la revenda queda totalment desprotegida, de forma que, qualsevol persona pot duplicar el tiquet, fer revendes amb preus abusius o inclòs es podrien fer falsificacions. És per això que podem concloure que en aquesta variant, els tiquets estan desprotegits i no hi ha cap forma de tenir un control.

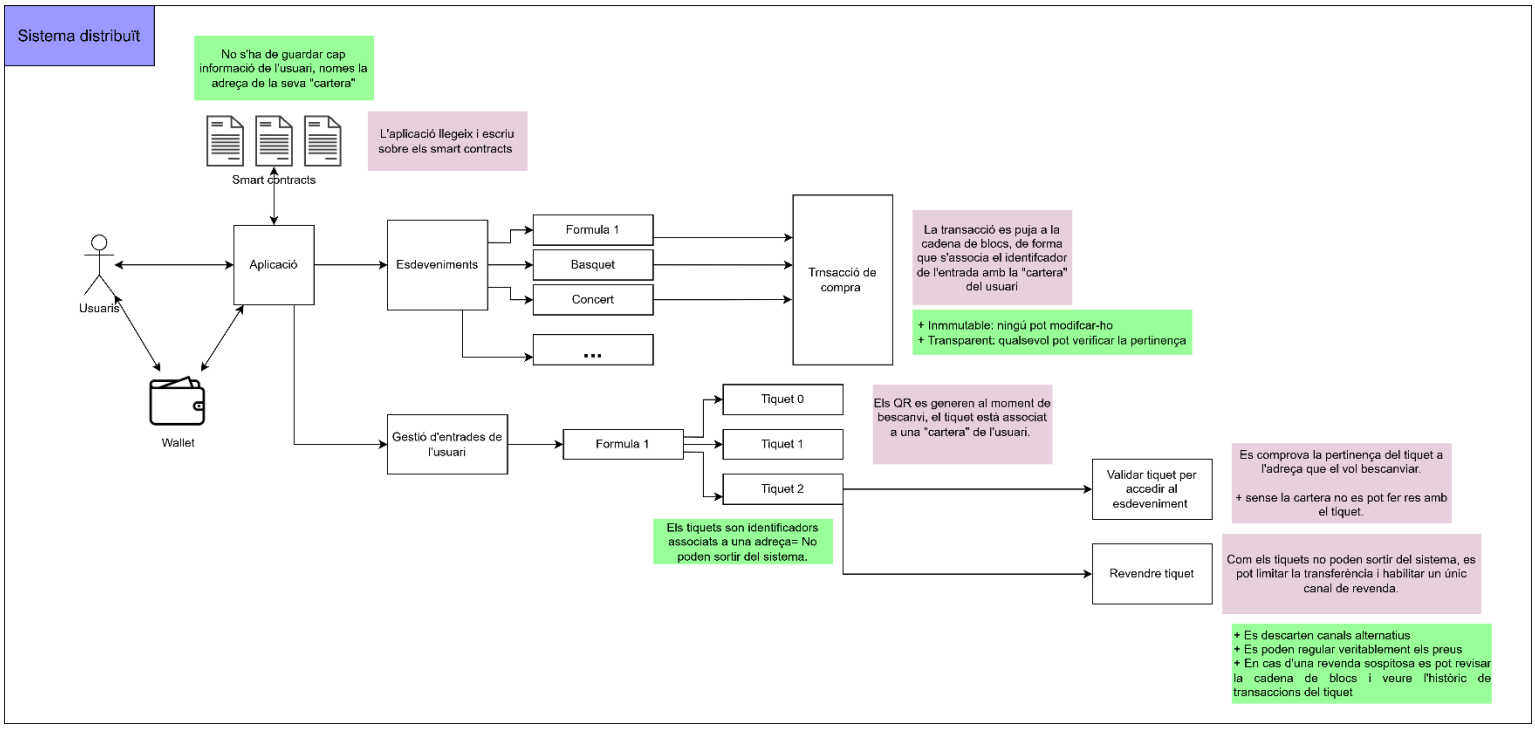
La segona variant, té un enfocament una mica més restrictiu, on els tiquets s’internalitzen, és a dir, no descarregues cap PDF sinó que el tiquet es desa als servidors del sistema. Aquest sistema, pot diluir els problemes d'autenticitat i duplicació de tiquets, però, que passa amb la revenda? Els sistemes estan obligats a permetre als usuaris a “rectificar” les seves, és per això que moltes estratègies de revenda malicioses creen comptes per comprar les entrades i vendre aquest compte perquè després el comprador pugui canviar les dades per les seves. De forma que no s’atura la revenda ni l'especulació de les entrades.

A continuació es presenta un diagrama que modela el funcionament dels sistemes tradicionals:



En contra part a aquest model, i com a resolució d'aquests problemes, trobem la solució distribuïda que es desenvolupa en aquest treball. Com ja hem esmentat abans, els beneficis són múltiples i permeten una gestió de la revenda d'entrades molt més organitzada i controlada. Primerament, perquè cap entrada pot sortir del sistema, i en segon lloc, perquè en aquest sistema es fa inviable el model de revendre una adreça que tingui les entrades, ja que per a transferir una adreça, has de donar obligatòriament la clau privada, la qual és única per cada direcció de cartera que es genera *[Observació: les claus no es poden canviar pel fet que aquestes estan relacionades matemàticament, més informació: https://www.ibm.com/docs/es/integration-bus/10.0?topic=overview-public-key-cryptography]*. Llavors la desconfiança de compartir una clau que mai es pot canviar, vol dir que el venedor podria comportar-se de manera malintencionada venent l'adreça a múltiples persones i per exemple, com ell també coneix la clau privada, aprofitar per bescanviar-la per l'accés abans que la resta. Com a resultat, es pot veure que és totalment inviable la revenda d'adreces amb entrades ja comprades, ja que has de confiar encara mes en el venedor que dintre dels sistemes tradicionals.

Per representar de manera més global el sistema que es desenvoluparà, a continuació es mostra un diagrama que exemplifica de forma simplificada el sistema a construir.



[Nota: Tot i que aquets model es millor en molts sentits, cal mencionar que si un usuari perd les claus, aquestes son irrecuperables degut a que son criptogràgicament segures i només es podrien trobar amb mètodes de força bruta i milions d'anys de assaig i error]

Model de dades

Continuant amb el sistema a desenvolupar, el diagrama anterior simplifica bastant el model de dades i cal esmentar que aquest projecte adopta un model híbrid combinant informació dintre de la cadena de blocs (on-chain) i fora de la cadena de blocs (of-chain). De forma que la informació crítica es manté dintre del smart contract i la resta es manté fora per a no ocupar espai dintre del smart contract, Aquesta informació fora de la cadena es guarda de forma distribuïda dintre d'un IPFS que es comentarà mes endavant el seu funcionament i ús dintre del projecte.

On-Chain

Dintre de cada esdeveniment podem distingir entre dos tipus d'informació:

Primerament, tenim la informació necessària per complir amb l'estàndard dels NFT, (anomenat ERC-721) amb informació com: el tokenURI (enllaç que porta a la informació que es troba fora de la cadena), el nombre màxim de NFT que es poden generar i el propietari de cada NFT que es genera.

D'altra banda, per a la mateixa gestió dels tiquets es necessita molta més informació com el límit de tiquets que un usuari pot comprar, l'hora de finalització de l'esdeveniment, la data de quan comença la compra d'entrades, el preu dels tiquets, quins tiquets té a la venda cada usuari i en cas d'estar a la venda un tiquet, qui és el venedor i a quin preu es ven.

Off-Chain

Per a guardar la informació fora de la cadena de blocs s'ha optat per guardar-ho dintre d'un IPFS, allà es guarden dos tipus de JSON *[Observació: un JSON es una estructura de dades que conté una llista de tupes clau-valor, per exemple: {nom: Lucas, edat: 22 }]*:

Informació sobre el esdeveniment: Conté informació com, el nom de l'esdeveniment la imatge oficial de l'esdeveniment i una descripció sobre aquest. D'aquesta forma l'aplicació distribuïda pot reconstruir l'esdeveniment per poder mostrar-ho a l'usuari.

L'enllaç a aquesta informació es guarda a una variable dintre del smart contract de forma que el sistema pot preguntar per aquesta variable per després extreure la informació.

Informació dels tiquets: Conté informació com el nom del NFT, una descripció sobre aquest, i un enllaç a la imatge NFT que compra l'usuari amb la pròpia entrada.

NOTA: Es pot trobar més informació sobre aquesta implementació al subtítol "Implementació d'un IPFS pels NFT".

Model funcional/tech

El sistema que es presentarà en les següents pàgines i com ja s'ha pogut veure al diagrama incial, està construït en dues capes ben diferenciades: el smart contract i la Dapp (la interfície). A continuació es presenten les funcionalitats de cada una d'aquestes parts.

Smart Contract

Aquesta capa desenvolupada purament en Solidity (llenguatge per a la creació de smart contracts per Ethereum) representa tota la lògica del sistema i determina quines accions es poden fer i quines no. Principalment, el contracte es centra en generar NFTs, i per sobre s'apliquen una sèrie de restriccions a aquests NFT relacionades amb la compra, venda i distribució pròpies del tiqueting tradicional.

Les funcionalitats més importants i que després es poden veure reflectides a la següent capa serien la possibilitat d'encunyament[2] de tiquets NFT, la revenda dels tiquets, i la gestió dels tiquets de l'usuari. En aquest cas, una vegada un usuari ha comprat un tiquet pot o bé bescanviar-lo pel mateix accés a l'esdeveniment o revendre el tiquet. Per aquesta tasca es disposen d'estats que indiquen la situació actual del tiquet, sigui Active, Pending o Redeemed. D'altra banda, també es contempla la possibilitat de transferir el tiquet pel seu compte, però amb la condició que aquest tiquet no pot estar a la venda dintre del contracte, ja que podria provocar errors en el comportament de l'aplicació.

Distributed Application

Dintre d'aquesta capa trobem el frontend, és a dir, l'aplicació que fa únicament d'interfície perquè l'usuari pugui visualitzar els esdeveniments i les entrades i pugui interactuar amb aquests. Aquesta interfície s'ha desenvolupat utilitzant tecnologies com React i Wagmi que es descriuran més endavant. Però aquesta interfície no només ha de permetre gestionar les entrades, sinó que ha de tenir una certa lògica perquè els usuaris no puguin realitzar accions que no siguin invàlides pel sistema, ja que enviar

[2] Encunyament d'un NFT: Procés de crear un "item" únic dintre de la cadena de blocs. Es especialment útil aplicat a les entrades ja que cada entra es única i irrepetible.

una acció al sistema té un cost i no podem permetre malgastar diners en accions que es revertiran. És per això que abans de permetre a l'usuari fer res, es llegeixen les variables d'estat dels tiquets i esdeveniments per a prevenir accions no autoritzades que desemboquin en transaccions errònies.

Algunes de les lectures que es fan sovint són, per exemple:

Les dates dels esdeveniments per saber si els usuaris poden comprar entrades, el nombre de tiquets que té l'usuari o fins i tot es comprova que l'usuari es troba en la xarxa en què s'ha llençat el smart contract a fi de poder interactuar amb aquest. *[Observació: aquesta última part és especialment important, ja que existeixen moltes cadenes de blocs diferents, per exemple existeix la Mainnet que és la cadena de blocs coneguda popularment com a Ethereum, però existeixen moltes altres. És per això que l'usuari ha d'estar connectat a la cadena correcta, ja que si no la interfície no trobarà el smart contract i no podrà interactuar amb ell]*

Metodologia aplicada

La metodologia aplicada per aquest treball ha sigut un model Waterfall, on a través del diagrama de Gantt exposat en la secció Planificació del treball: Seguiment II, cada fase es culmina abans d'avançar a la següent amb una definició prèvia. Les fases han sigut:

- **Planificació i anàlisi de requisits**: Correspon a la tasca d'investigació sobre la Dapp, avaluar eines, quines funcionalitats / requeriments aporten les diferents tecnologies, quines encaixen millor amb el projecte, etc.
- **Disseny de l'arquitectura i selecció de tecnologies**: Correspondria a la tasca d'aprendre un llenguatge per la Dapp que inclou la definició de quins criteris son clau per la selecció.
- I**mplementació**: Aquesta fase marcaria l'inici del desenvolupament on es treballa en el codi de la Dapp i es fan modificacions al smart contract segons sigui necessari.
- **Testage:** Si bé aquesta fase es realitza de manera més lleugera durant el desenvolupament, una vegada aquesta finalitza, tots els esforços es dediquen a aquesta part per poder detectar i corregir errors abans del desplegament.
- **Desplegament:** Finalment, es desplega el smart contract dintre d'una testnet de forma que qualsevol persona pot interactuar amb aquest a través de la Dapp publicada.

Marc de treball pel desenvolupament de la Dapp

Per desenvolupar una aplicació descentralitzada (la interfície) és indispensable seleccionar un framework [3] adequat, ja que faciliten la creació de la interfície gràcies a components i funcions predefinides que estalvien temps al desenvolupador.

En el cas d'aquest projecte, es necessita una sèrie de requisits indispensables per agilitzar el desenvolupament de la interfície i que aquesta pugui interactuar amb els smart contracts, el quals son els que contenen tota la lògica que hem definit pels esdeveniments i els tiquets.

[3] Framework: Conjunt d'eines predefinides per a poder desenvolupar un projecte. Per exemple, JavaScript és un llenguatge de programació, però existeixen múltiples eines que utilitzen aquest llenguatge per crear funcions i estructures predefinides que faciliten el desenvolupament.

Per a fer la selecció d'un framework, es tindran en compte les següents característiques:

- **Integració amb Metamask:** Aquest és un dels objectius primaris que es van definir dintre del marc de desenvolupament de la Dapp. Per tant es indispensable que el framework pugui connectar-se-
- **Facilitat d'ús i corba d'aprenentatge:** És indispensable que la selecció del framework no doni una corba d'aprenentatge massa gran o que pugui alentir el desenvolupament en gran manera.
- **Suport de la comunitat:** Important tenir-ho en consideració, ja que per aprendre de forma autònoma resulta beneficiós comptar amb projectes similars, documentació feta per la comunitat, pàgines web per aprendre de forma eficient, etc.
- **Recursos i documentació existent:** De la mateixa manera que va ser molt útil disposar d'una documentació consistent pel desenvolupament del Smart Contract, per aquesta segona fase és igual d'important, ja que una documentació clara, ben estructurada i amb suport pot marcar una gran diferència tractant-se d'un aprenentatge autodidàctic.

Fent una cerca exhaustiva sobre les diferents opcions, els candidats proposats són:

VUE.JS

A l'inici del plantejament d'aquest projecte, vaig esmentar a vue.js per una raó de pes, ja que, durant l'any passat una de les assignatures que es van cursar (Sistemes i Tecnologies Web) ens van ensenyar aquest framework de JavaScript reconegut àmpliament. És per això que es presenta com un candidat gràcies a la base de coneixements adquirits sobre aquest. A més, permet l'ús de llibreries indispensables com ethers.js i altres més específiques per a fer aplicacions distribuïdes. També té una comunitat considerable encara que aquesta no està especialment enfocada en la creació d'aplicacions distribuïdes, fent així que el nombre de projectes d'aquest tipus no siguin el punt més fort d'aquest framework. D'altra banda, la documentació oficial es veu clara i estructurada per a una millor qualitat durant el desenvolupament, però en relació amb eines de desenvolupament per a projectes distribuïts com el que es planteja en aquest informe, és més limitat al no comptar amb guies enfocades en el desenvolupament d'aplicacions descentralitzades, fet que podria portar un esforç addicional la resolució de desafiaments tècnics propis del desenvolupament d'aplicacions basades en la cadena de blocs.

REACT.JS

Aquest framework és considerat com una de les opcions més rellevants dintre de la creació d'aplicacions distribuïdes. Disposa d'algunes llibreries especialitzades que poden ajudar a accelerar el desenvolupament en tasques com la connexió al wallet de Metamask o la gestió de les transaccions que pot donar com a resultat una integració més robusta i optimitzada. Per la banda de l'aprenentatge, pot presentar un desafiament vers l'anterior opció ja que s'hauria d'aprendre a utilitzar la part més general de React i la part més específica per fer la integració amb la blockchain. Però pot presentar un problema més petit del que pot semblar, ja que compta amb un gran nombre de repositoris, projectes de referència, tutorials i fòrums específics sobre la creació d'aplicacions distribuïdes. En últim lloc, i, revisant la documentació, React disposa d'una base molt àmplia combinada amb guies pràctiques i exemples especialitzats que aporten un gran valor per poder aprendre de forma autodidàctica.

Selecció final

Finalment, després d'haver analitzat aquests dos candidats, s'ha pogut comprovar que ambdós són eines molt vàlides, Vue.js ofereix un desenvolupament més àgil gràcies a l'experiència obtinguda durant el darrer curs, però la principal diferència amb React és que hi ha més recursos com llibreries, documentació, projectes, guies i una base de coneixements especialitzats amb els quals guanya i el posicionen com una opció més adequada donades les característiques d'aquest projecte.

Aprenentatge:

L'aprenentatge per aquesta segona part ha sigut m´ws lleugera que la primera, ja que durant la carrera hem treballat amb JavaCcript i frameworks derivats com Node.js i ja estic acostumat a utilitzar aquest tipus d'eines. No obstant per no deixar-me res i recordar conceptes, he realitzat el tutorial de React que hi ha a la pàgina de w3schools i he estat testejant pel meu compte els react hooks[4] que mostren. Aquest aprenentatge juntament amb els exemples de projectes trobats a ethereum.org, m'ha servit com una base suficient per poder començar a desenvolupar la meva pròpia interfície.

Arquitectura tech i eines

Ara que s'ha seleccionat un framework adequat, a continuació es defineixen quines llibreries s'han utilitzat per a aquest projecte i quins altres components derivats han sigut utilitzats. D'aquesta forma, es podrà comprendre millor les eines que s'han fet ús durant el desenvolupament de la interfície i quina funció tenen dintre del projecte. Aquesta part es una mica mes tècnica i serveix com a base per definir les eines que s'ha estat utilitzant durant el desenvolupament.

Llibreries utilitzades

Wagmi

Dintre de la comunitat existeixen diverses llibreries que ajuden el desenvolupador a fer que la seva web pugui connectar-se al wallet del usuari, fer operacions amb els smart contracts, etc. La llibreria més utilitzada i coneguda és ethers.js la qual conté allò més bàsic i essencial perquè el desenvolupador pugui construir per ell mateix altres funcions i blocs de codi que permetin la connexió. Però moltes vegades és difícil tractar amb cadascuna de les APIs[5] de cada eina. És per això que he volgut fer servir wagmi per aquest projecte. Ja que inclou Hooks específics ja testejats i funcionals per poder connectar-se a wallets com Metamask, Coinbase, etc. A més, també té altres Hooks útils per poder fer diverses lectures i escriptures als smart contracts en una sola crida de funció, fet que és de gran ajuda, perquè ajuda a fer que el desenvolupador es pugui centrar en la lògica i el disseny en comptes de les interconnexions amb elements externs. El següent vídeo m'ha sigut de gran utilitat per poder escollir quina llibreria fer servir, pel fet que encara que existeixen llibreries amb components més específics amb dissenys preestablerts, i lògiques molt més avançades pels smart contracts, el

[4] React Hooks: Es una funció que gestiona l'estat d'una variable o que encapsula accions asíncrones com la lectura a un servidor o un temporitzador.
[5] API: Conjunt de regles i funcions que permeten que dos aplicacions puguin parlar entre si. En aquest cas, la API serveix per a que la interfície d'usuari pugui parlar amb el wallet de l'usuari per poder obtenir informació rellevant com el numero de compte, quants diners té, etc.

meu objectiu és fer una Dapp, no muntar-la a partir de llibreries. És per tot això que finalment he decidit una llibreria de nivell intermedi com Wagmi.

react-router-dom

És una llibreria de React que permet simular “enrutaments” per a diverses pàgines, de forma que a partir de rutes permet la renderització d'un component o un altre. Per exemplificar-ho, si la meva pagina es www.laMevaInterficie.net , puc ‘simular’ que hi ha diverses pàgines fent es www.laMevaInterficie.net/PaginaInici, o www.laMevaInterficie.net/ElsMeusTiquets. En el meu cas és útil perquè la web té tres seccions i, en utilitzar aquesta llibreria, en comptes de generar noves pàgines que comportaria fer una càrrega de la nova pàgina. El que es fa és que cadascuna de les seccions sigui una funció, la qual només s’activa quan estem en la ruta indicada. Per exemple si estem a /ElsMeusTiquets s’activa aquesta funció, i la funció PaginaInici no es crida i per tant, no s’executa. D’aquesta forma dintre de la mateixa pàgina es mostra al usuari el contingut que ell vol sense demanar al servidor carregar una nova pàgina web.

react-icons

Llibreria de React la qual conté milers d’icones que es poden utilitzar com si fos un Hook més. És molt útil per fer més intuïtiva l’aplicació.

qrcode.react

Llibreria que permet generar codis QR. En aquest projecte s’ha utilitzat per poder bescanviar els tiquets per l’accés a l'esdeveniment de manera ràpida i segura.

Wagmi Hooks Utilitzats

Els hooks de wagmi ha sigut una de les principals raó per utilitzar aquesta llibreria. Per posar una mica de context, els hooks són funcions que encapsulen el comportament i poden guardar informació, escoltar canvis en variables o executar accions asíncrones a l'interior. En aquesta secció es fa un llistat dels hooks de la llibreria Wagmi que he fet servir, explicant la seva funció dins del projecte. Amb aquesta part es busca poder definir aquells elements tècnics que s’han fet servir.

useAccount: Identifica si hi ha alguna adreça connectada a la interfície, i en cas de haver-hi, obté informació sobre l’adreça com la direcció connectada i la cadena de blocs a la que està connectada.

useConnect: S’encarrega del procés de connexió de la wallet de l’usuari a la Dapp. Es pot especificar quin tipus de wallet s’espera per a crear botons personalitzats els quals integren l'API del wallet corresponent.

useReconnect: Si un usuari surt de la Dapp i torna, si no usem aquest hook, la Dapp demanarà una altra vegada connectar el wallet de l'usuari. Però gràcies a aquest hook la Dapp recorda l’última connexió i torna a fer-la de forma que connecta automàticament la wallet del usuari quan aquest torna a la interfície.

useDisconnect: Proporciona una opció per desconnectar el wallet del usuari a la Dapp. És especialment útil per la versió mòbil, ja que permet a l'usuari no haver de sortir al wallet per a desconnectar-se sinó que pot fer-ho directament des de la Dapp.

useSwitchChain: Genera una petició de canvi de cadena de blocs per a la wallet connectada de l’usuari. És molt útil perquè si l’usuari no està en la xarxa correcta per

interactuar amb el Smart contract, aquest hook genera una petició pel wallet de l'usuari, i si l'accepta, el canvia directament a la xarxa correcta que hem definit.

useBalance: Hook que captura la quantitat de monedes que posseeix l'usuari sobre la xarxa que indiquem. S'ha fet servir per mostrar el saldo que té l'usuari i així no hagi de revisar la seva cartera per veure si pot o no pot comprar una entrada.

useReadContract / useReadContracts: Son dos tipus de hooks que en essència llegeixen informació sobre el smart contract. La diferència és que el primer només fa una lectura mentre que el segon pot fer múltiples lectures d'un sol cop, ja que se li pot donar una estructura amb múltiples lectures perquè només hagi de cridar al node d'Ethereum una única vegada i aquest li pugui respondre totes les consultes a l'hora (d'aquesta forma només enviem un paquet amb tot el que necessitem).

useWriteContract: És la contrapart del hook anterior i es fa servir per escriure dintre del smart contract i poder actualitzar l'estat d'alguna variable d'aquest, per exemple, quan volem comprar un tiquet, canviar-ho d'estat, etc.

useWaitForTransactionReceipt: És especialment útil per escoltar quan una acció que ha fet l'usuari s'ha integrat dintre de la cadena de blocs, per exemple per després demanar a la interfície que quan vegi aquella acció a la cadena de blocs que demani el nou estat i el representi sobre allò que ha interactuat l'usuari.

Avenços II: Procés de desenvolupament

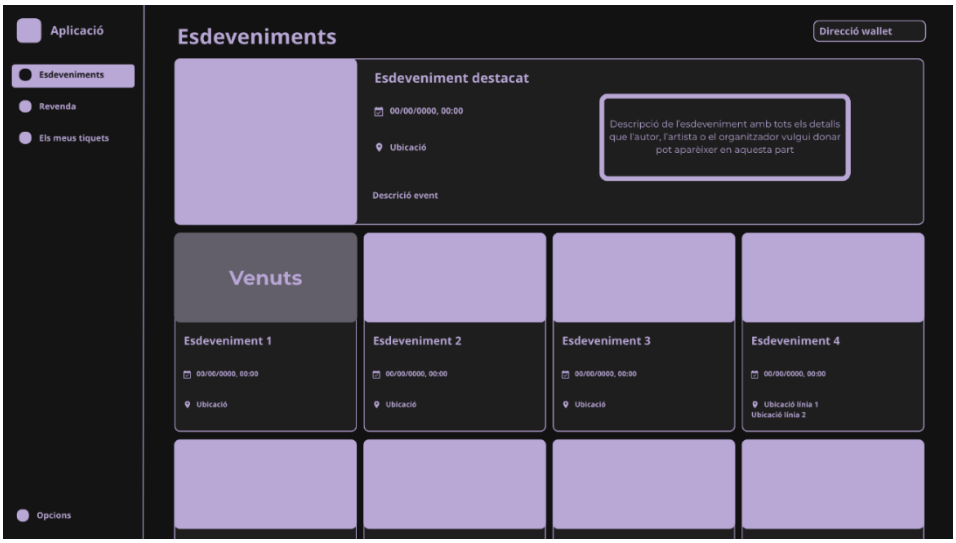
En aquesta part s'analitzarà les diferents implementacions que he fet per el projecte mostrant els esbossos inicials de cada element i la versió final dissenyada explicant detalls d'alguns components.

Prototip

Abans de començar a desenvolupar el disseny i la seva lògica, s'ha realitzat un disseny preliminar per poder visualitzar a grans trets com seria l'experiència d'usuari. Com a finestres principals, he decidit fer 3: Esdeveniments, Revenda i Els meus tiquets.

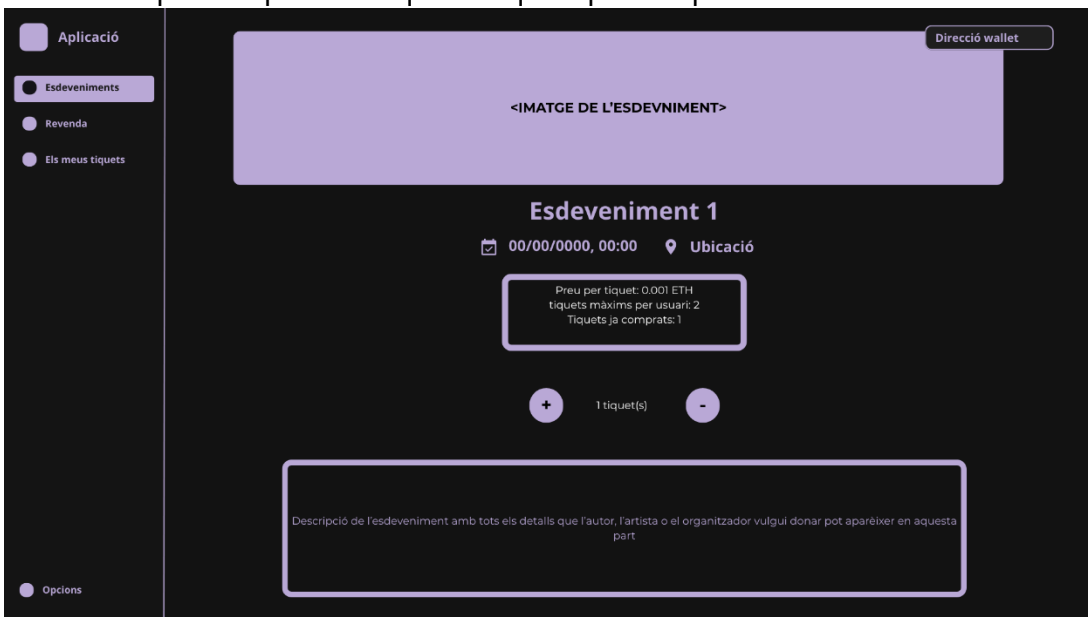
Esdeveniments

És la pàgina principal a l'entrar i mostra tots els esdeveniments amb la data del esdeveniment, en quin lloc es realitza, etc. Funciona com la porta d'accés a comprar entrades d'estoc.



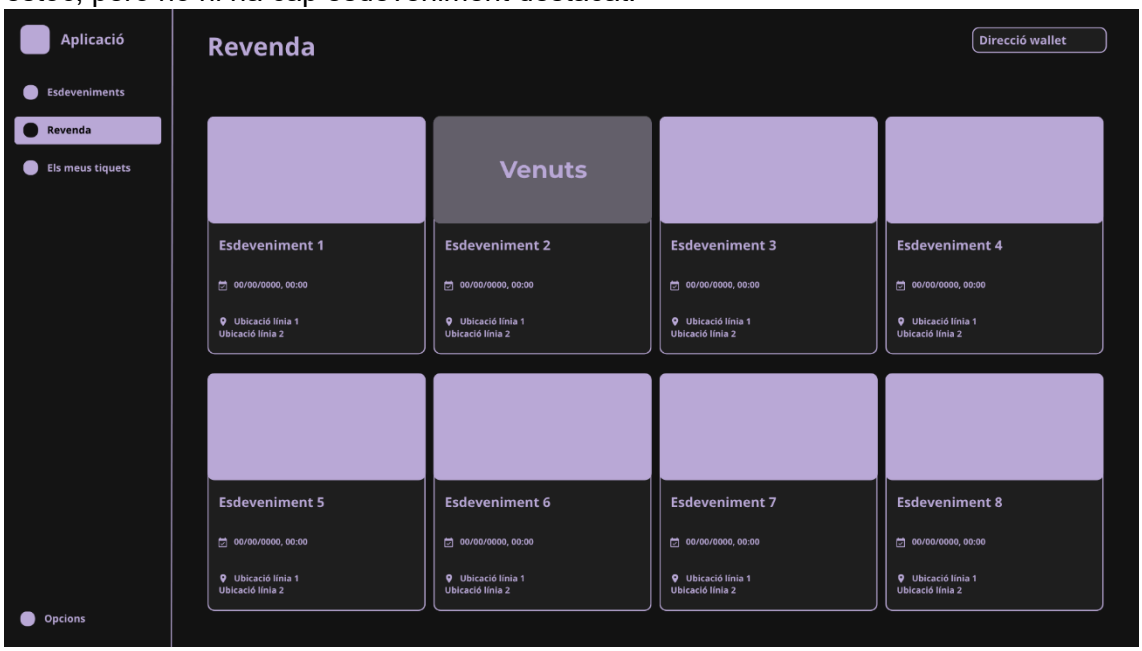
Pagina de compra d’entrada (tiquets en estoc)

Quan l’usuari clica una de les targetes anteriors, es mostrarà informació rellevant sobre els tiquets disponibles i quants tiquets pot comprar l’usuari.



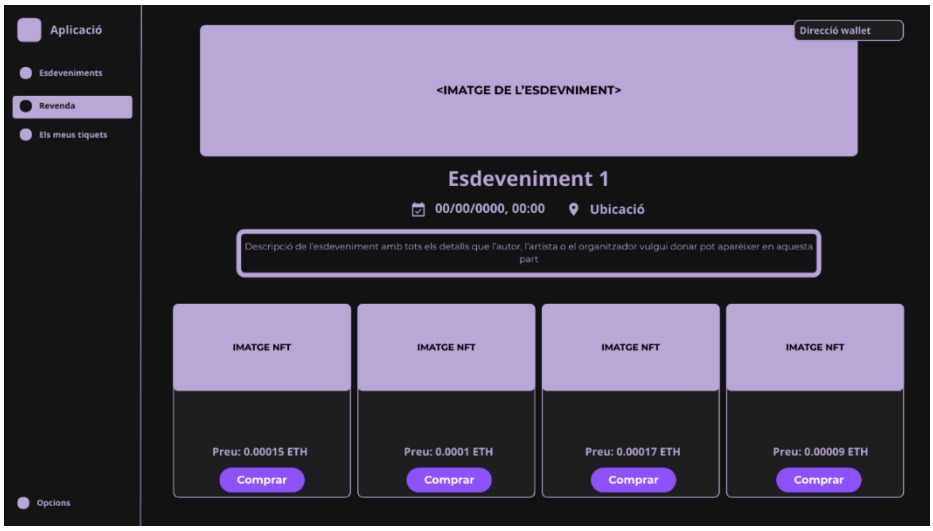
Revenda

Es mostren les entrades venudes per altres usuaris. És similar a la pàgina de venda en estoc, però no hi ha cap esdeveniment destacat.



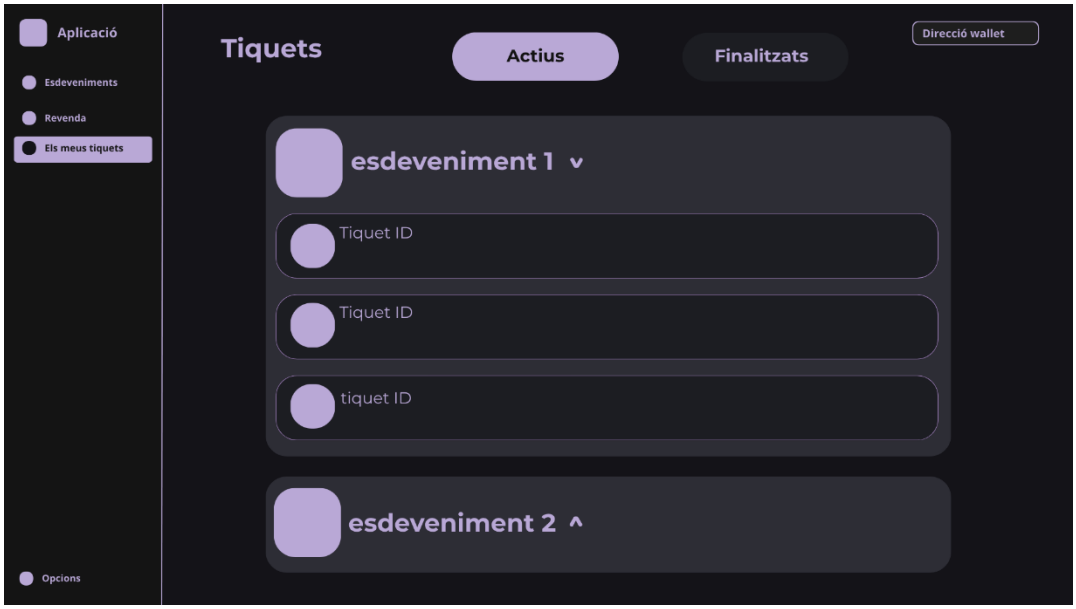
Revenda (esdeveniment específic)

Quan un usuari fa clic en una de les targetes anteriors, es mostren els tiquets que altres usuaris han posat a vendre (sempre tenint en compte que el preu no pot superar en més d'un 30% el preu original). Els tiquets es mostren amb la imatge del NFT que obtenen els usuaris en comprar el tiquet, d'aquesta manera tenen una visualització més pròxima a quan un usuari ha comprat el tiquet, permetent diferenciar-ho amb la venda en estoc.



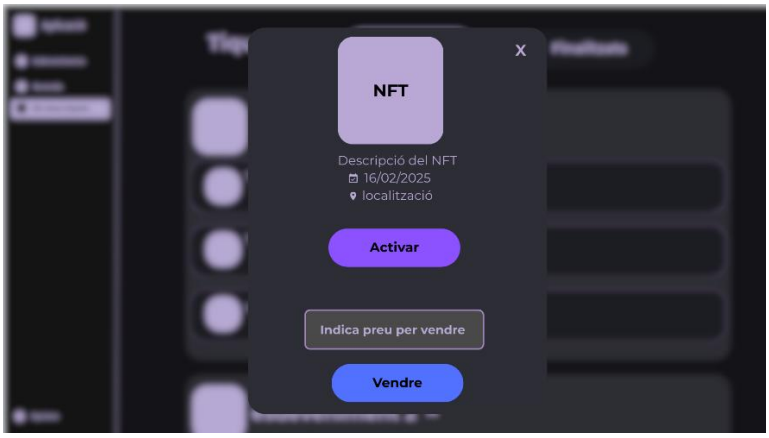
Els meus tiquets

En aquesta finestra es mostren els tiquets que l'usuari té en el seu poder. Es podrà distingir entre els diferents estats per a poder trobar-los de manera mes còmoda.



Tiquet Adquirit

Si l'usuari vol interactuar amb el seu tiquet nomes a de seleccionar-ho en la finestra d'abans. Una vegada aparegui la targeta pot bescanviar-lo per mostrar el QR (i així escanejar el QR i obtenir l'accés al esdeveniment) o vendre l'entrada a altres usuaris



Colorimetria, nom i icona per la Dapp:

Com ja s'ha pogut veure en el prototip mostrat, l' esquema de colors, de l'aplicació serà una paleta de colors lila-violeta en diferents tonalitats més clares i més fosques. Aquesta decisió del disseny es bàsicament pel fet que el lila es el color que sempre m'arriba al cap quan penso en Ethereum, i aquest combina molt bé amb els colors foscos.

Una vegada resolt els colors, una altra cosa que volia fer en aquest projecte és donar-li personalitat, i per això necessito dos elements essencials: la icona que representarà a la aplicació web i un nom que sigui apropiat.

Per no estendre aquesta part, la meva idea per la icona ha estat un color violeta que representés un tiquet i a més la cadena de blocs, ja que el logotip d'Ethereum és massa reconegut. És així que després de diverses iteracions la icona final ha sigut el següent:

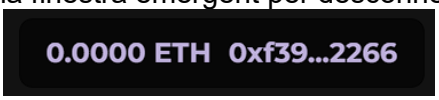
Per la part del nom, com l'aplicació utilitza smart contracts i tiquets, conversant amb diferents amics el nom finalment ha sigut Mintatix, que junta les paraules: "Mint" que és l'acció d'encunyar amb "Tix" que és una forma abreviada d'anomenar "Tiquet" en anglès. El resultat seria la idea d'encunyar un tiquet, que és exactament el que passa amb les entrades-NFT del meu projecte quan una persona en compra un.

Components persistents

En aquesta secció es mostraran components gràfics de la interfície que he desenvolupat i que estan presents sempre en qualsevol lloc de la interfície on es trobi l'usuari. Aquests components son molt simples en quant a lògica ja que s'enfoquen en realitzar una única tasca per l'usuari.

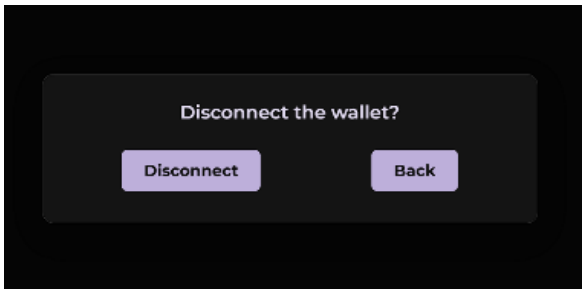
Account header

Es un Hook que permet la visualització de la quantitat de monedes que té la cartera que ha connectat l'usuari. A més, mostra l'adreça connectada de forma escurçada, mostrant els 3 primers caràcters i els 3 últims per no ocupar molt espai. D'aquesta forma si l'usuari si té diverses carteres, pot identificar quina cartera ha connectat a la interfície. A més, al ser polsat a sobre, porta a una finestra emergent per desconnectar la cartera del usuari.



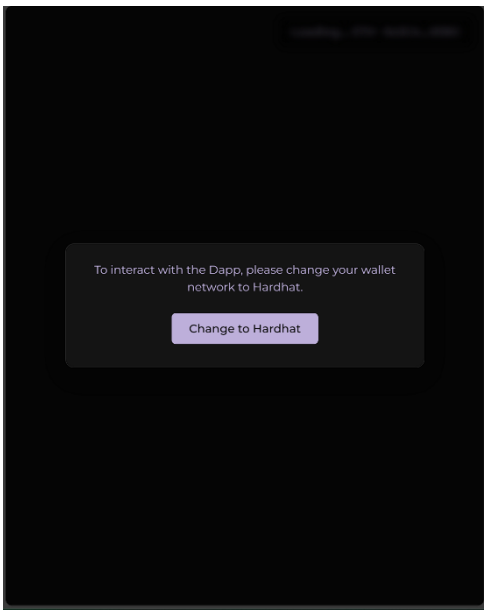
Disconnect

Com s'ha dit abans, al polsar sobre l'anterior component es mostra una finestra emergent que permet a l'usuari desconnectar la seva cartera de la interfície. Aquest component activa un dels hooks personalitzats de wagmi per poder fer la desconnexió. El botó "back" desactiva el component per continuar mostrant la pàgina on es trobava l'usuari.



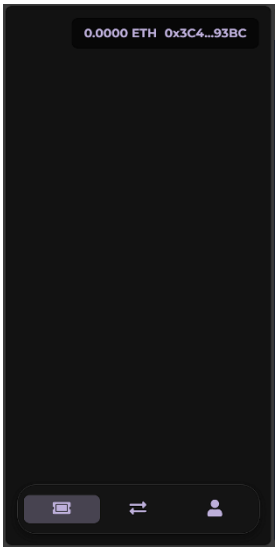
ChainSwitcher

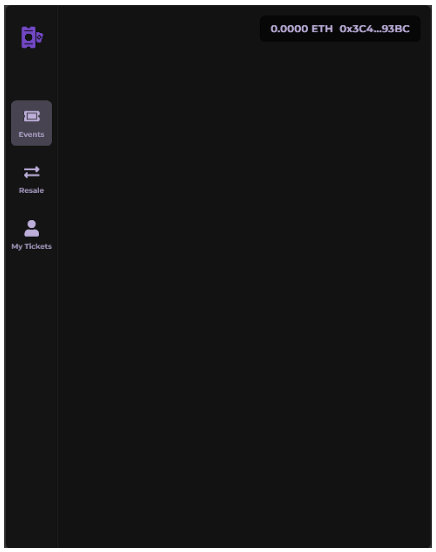
Aquest component es essencial per la interacció de l'usuari amb el smart contrcat que es troba dintre de la cadena de bloc. Ja que per a que l'usuari pugui interactuar amb el smart contract l'usuari ha d'estar connectat a la cadena de blocs on es troba el smart contract. És per això que s'ha dissenyat un component el qual revisa quina és la cadena de blocs en la qual es troba l'usuari, de forma que, si no és la correcta, aquest component s'activa automàticament i ensenya una finestra demanant a l'usuari que canviï a la cadena de blocs correcta.



Menu

La interfície que s'està desenvolupant, s'ha de poder utilitzar en qualsevol tipus de dispositiu, és per això que s'ha establert tres visualitzacions del menú de l'aplicació: la versió mòbil, una versió intermèdia per pantalles com les tauletes o mòbils grans i finalment la versió d'escriptori, aquest menú utilitza el react router dom per poder mostrar els diversos apartats i alguns icones perquè sigui més intuïtiu el seu ús.





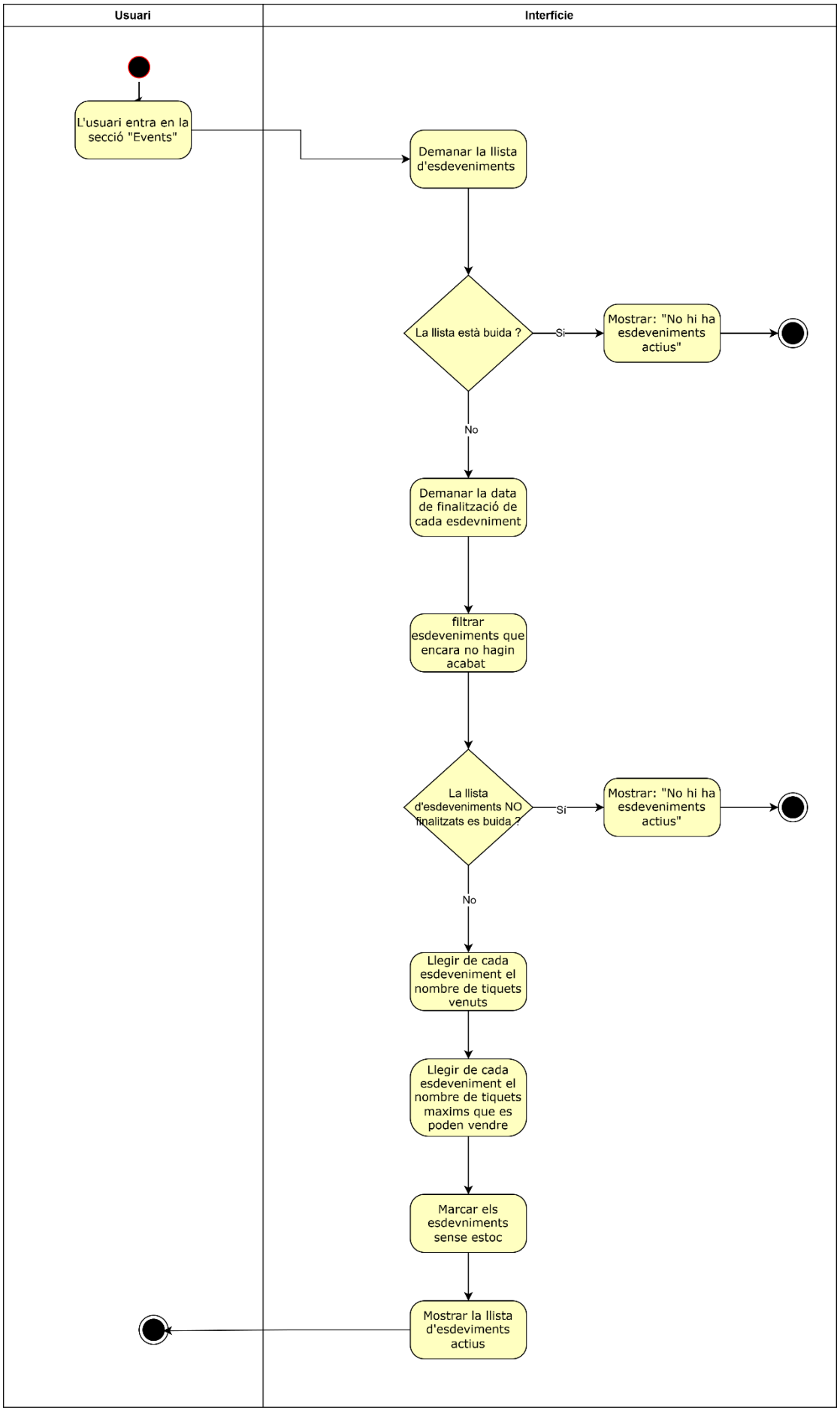
Components de pàgina

A continuació es presenten els components que no són estàtics dintre de la interfície, és a dir, aquells components que es mostren en funció d'en quina part de la interfície es trobi l'usuari. Podem distingir entre tres parts ben diferenciades, les seccions:

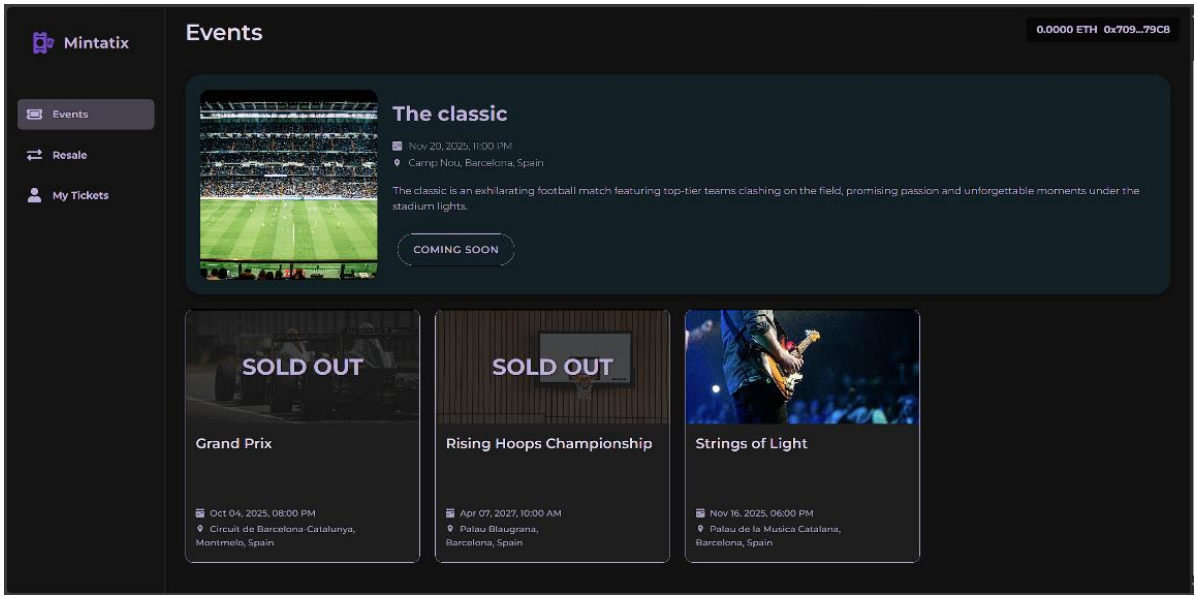
- Events: On es mostren els esdeveniments actuals i es poden comprar entrades en estoc.
- Resale: On es mostren els esdeveniments actius i les entrades que venen altres usuaris.
- My Tickets: On es mostren els tiquets que té l'usuari que està connectat a fi que pugui interactuar amb els seus tiquets.

Events

La secció Events és la porta d'accés a l'usuari per veure quins són els esdeveniments actius que pot comprar entrades en estoc, inclou una primera presentació de cada esdeveniment en format de targetes les quals en fer clic mostra l'esdeveniment de forma més detallada. Funcionalment, aquesta finestra demana al smart contract quins esdeveniments hi ha i la data de finalització de cadascun per saber si ha de mostrar-ho o no. A continuació es presenta el diagrama de flux que representa de manera global el seu funcionament.

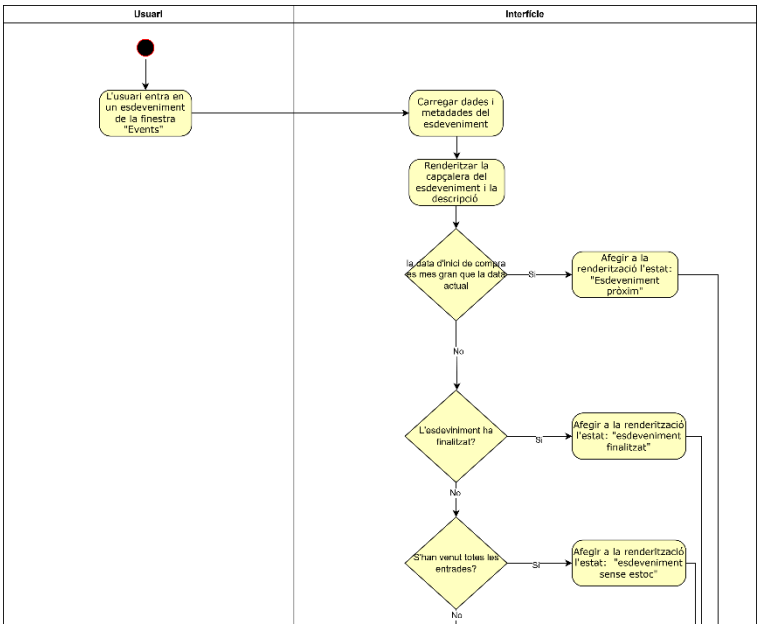

Usuari
Interfície
L'usuari entra en la secció "Events"
Demanar la llista d'esdeveniments
La llista està buida ?
Si
Mostrar: "No hi ha esdeveniments actius"
No
Demanar la data de finalització de cada esdevniment
filtrar esdeveniments que encara no hagin acabat
La llista d'esdeveniments NO finalitzats es buida ?
Si
Mostrar: "No hi ha esdeveniments actius"
No
Llegir de cada esdeveniment el nombre de tiquets venuts
Llegir de cada esdeveniment el nombre de tiquets maxims que es poden vendre
Marcar els esdevniments sense estoc
Mostrar la llista d'esdeviments actius

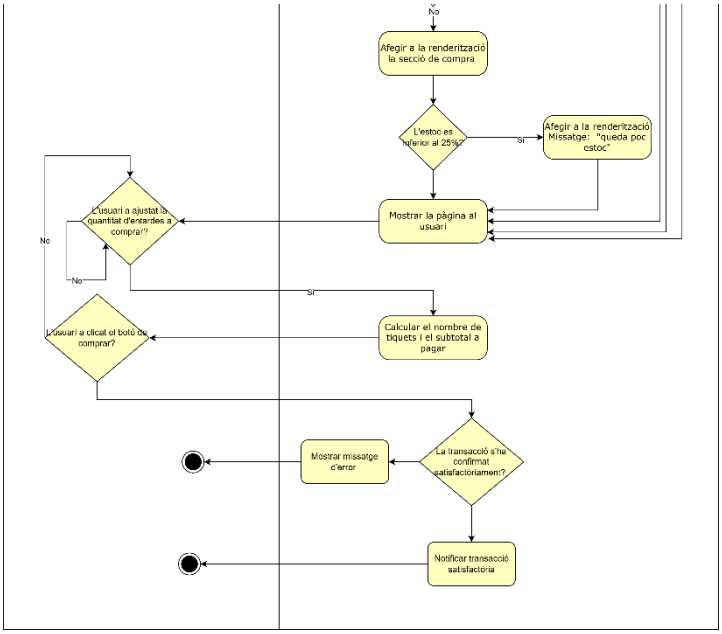
Visualment, el resultat de la finestra “Events” ha sigut el següent:



EventDetailPage

Quan una de les targetes anteriors és clicat per l’usuari, el porta a aquest nou component amb l’objectiu que l’usuari pugui visualitzar millor la informació i pugui comprar entrades en estoc per l’esdeveniment.





De maner mes tècnica, i per explicar el diagrama anterior, en aquest hook es fan lectures a eventURI[6], startPurchaseTime, ticketPrice, maxTicketsPerAddress, ticketsPurchased, maxSupply (el nombre màxim de entrades que es poden generar), totalSupply (el nombre d'entrades generades) i eventEndTime.

Una vegada obtinguda tota la informació s'utilitza un useEffect amb un comptador perquè a cada segon es torni a renderitzar la pàgina. [*Observació: que l'aplicació renderitzi no vol dir tornar a cirdar al smart contract, ja que els hooks de wagmi guarden l'estat durant un temps per defecte en comptes de fer la cerca immediata*] D'aquesta forma es comprova a cada segon si l'esdeveniment ha acabat per a poder mostrar l'apartat de compra o notificar al usuari que l'esdeveniment a finalitzat. Per als estats dels esdeveniments, s'han definit 3 estats:

- Coming soon: La venda d'entrades no ha començat encara
- Event Finished: El temps actual és superior al eventEndTime agafat del contracte
- Sold Out: Les entrades en estoc per l'esdeveniment estan esgotades

També s'han creat 2 variables d'estat addicionals per notificar a l'usuari:

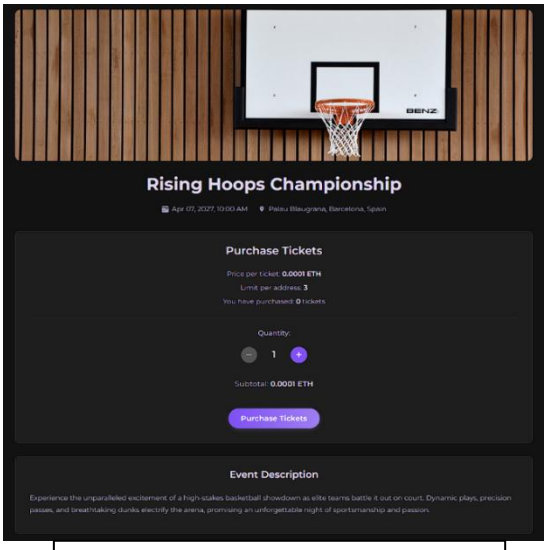
- Remaining Supply: Serveix per després calcular si l'usuari pot comprar menys tiquets que el límit per adreça estipulat a causa del estoc que queda.
- Low Supply: Aquest estat s'activa quan calcula que queden un 25% o inferior del total d'entrades, de forma que mostra un missatge a l'usuari amb la quantitat d'entrades restants.

A més, quan l'usuari fa la compra, s'utilitza un altre hook de wagmi que escolta dintre de la blockhcain quan es completa la transacció, de forma que obliga a llegir de nou el total de tiquets que ha comprat el usuari i la quantitat d'estoc que queda encara, de forma que podem saber quin és el nou límit d'entrades que pot comprar l'usuari i quin és el total d'entrades venudes per si és necessari mostrar l'estat de Sold Out.

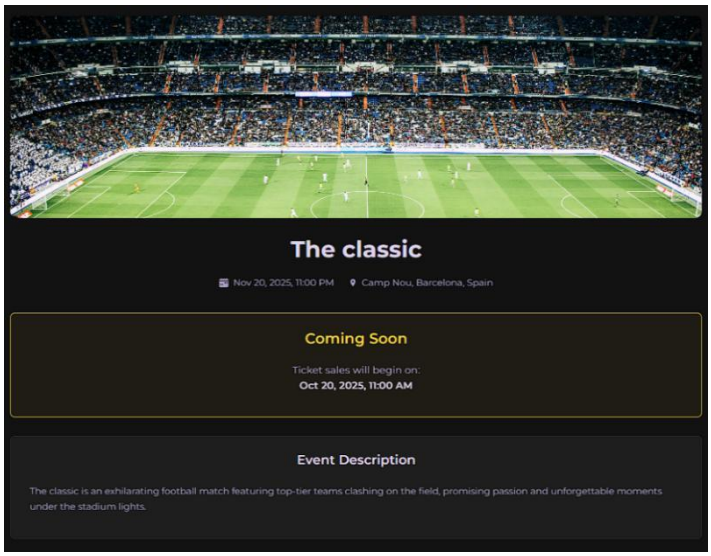
També es mostra al usuari quin es el límit d'entrades que pot comprar i quantes entrades porta comprades ja que s'ha de tenir en compte que el recompte d'entrades que ha comprat afecta tant si encara les té com si ja no las té per que les ha venut ja que sino es fa això podrien incitar als usuaris a comprar i vendre en bucle, creant un negoci il·lícit de compra venda d'entrades aprofitant-se de que el sistema permet vendre les entrades un 30% superior al preu de la entrada en estoc.

[6] EventURI: variable que conté una adreça on es troba informació externa, en aquest cas, sobre l'esdeveniment, per exemple, una descripció d'aquest, la imatge principal, el nom de l'esdeveniment, etc.

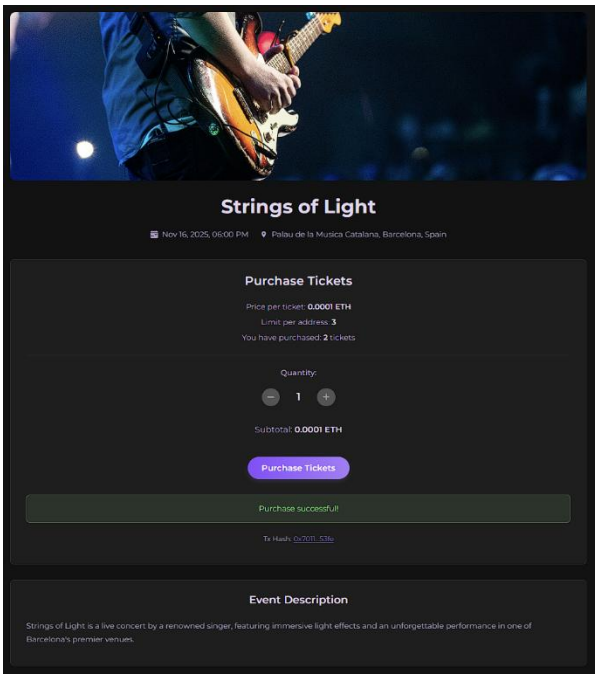
A continuació es mostra el resultat final d'aquest component amb diferents escenaris:



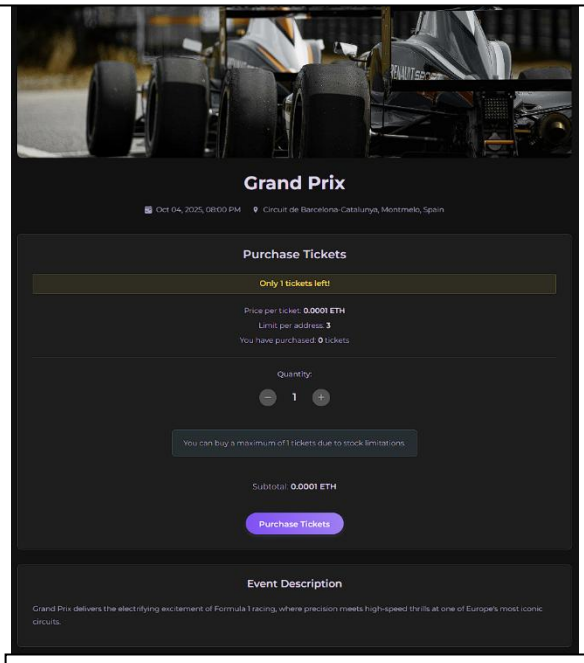
1.Hi ha més d'un 25% d'estoc



2. El temps de compra encara no ha arribat



3. L'usuari ha completat la transacció



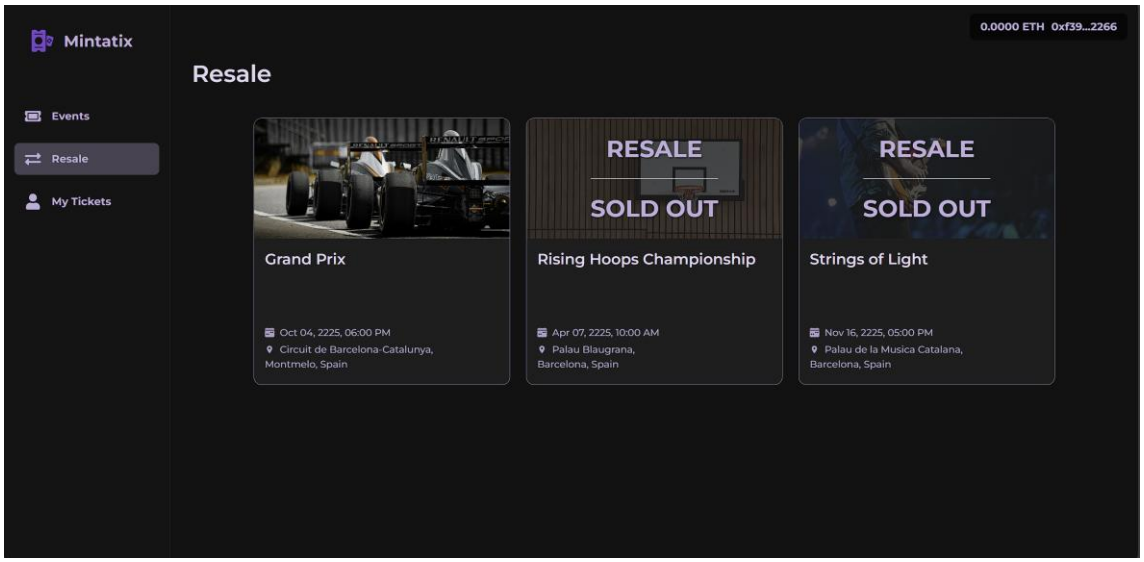
4. L'estoc està per sota del 25% i queden menys entrades que el màxim permès per adreça

Observació: per a fins de mostrar els quatre escenaris, s'han generat esdeveniments amb un estoc de 4 entrades i un límit de 3 entrades per adreça, de forma que el 25% seria quan només en queda una (cas 4).

Resale

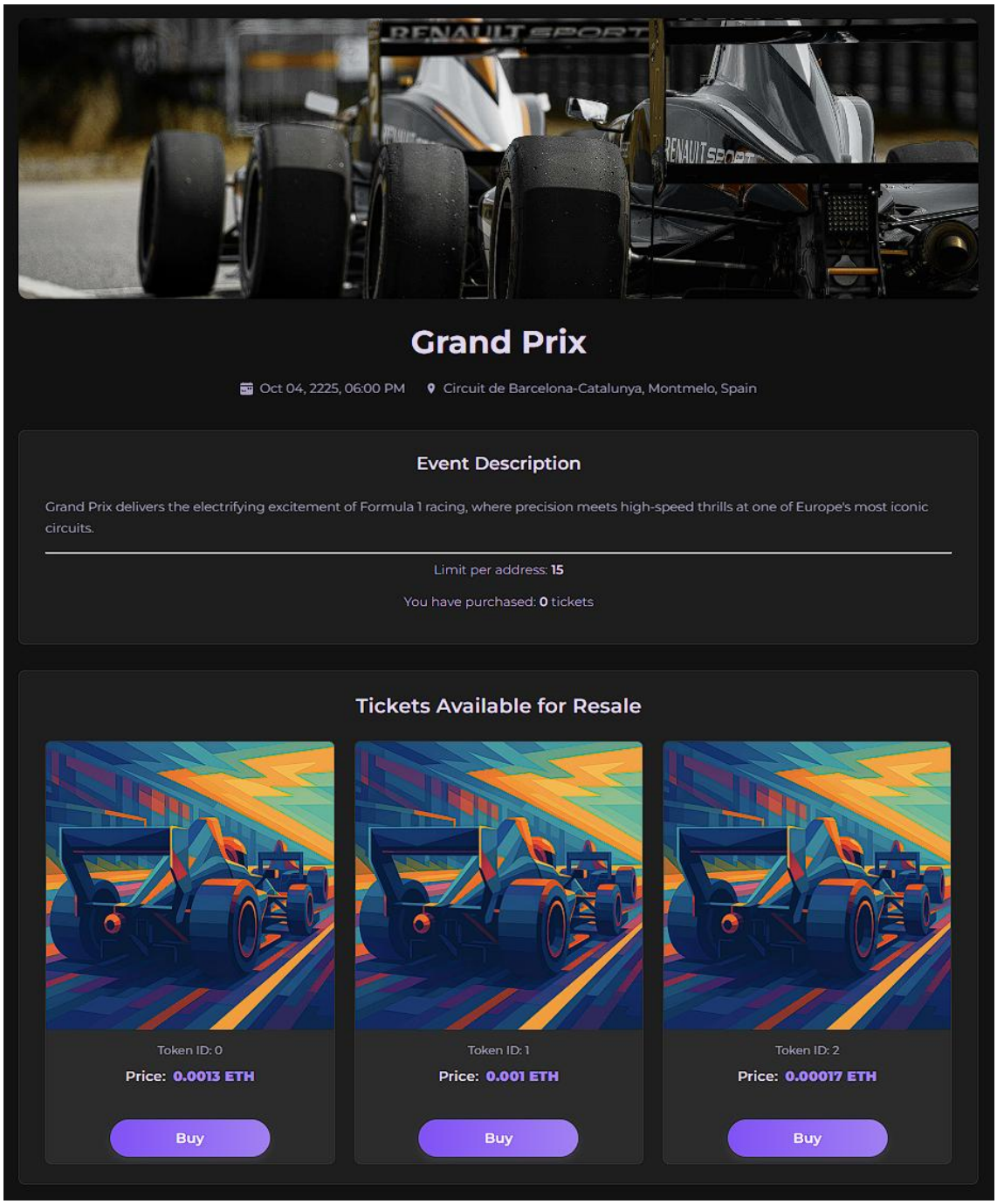
Aquesta secció permet a l'usuari veure sobre els esdeveniments que es troben actius, si hi ha entrades d'altres usuaris a la venda. En aquest cas, s'han desenvolupat quatre components, dels quals dos d'ells (EventGridResales i EventCardResale) són una reutilització de les versions anteriors que s'han fet per la finestra "Events". Com a únics canvis a destacar és que, per a determinar si representar un esdeveniment, han de ser purament actius, per tant, s'ha de verificar que la data de compra de tiquets per

l’esdeveniment ja l’hem passat, ja que no tindria sentit posar esdeveniments els quals els usuaris encara no poden comprar entrades d’estoc.



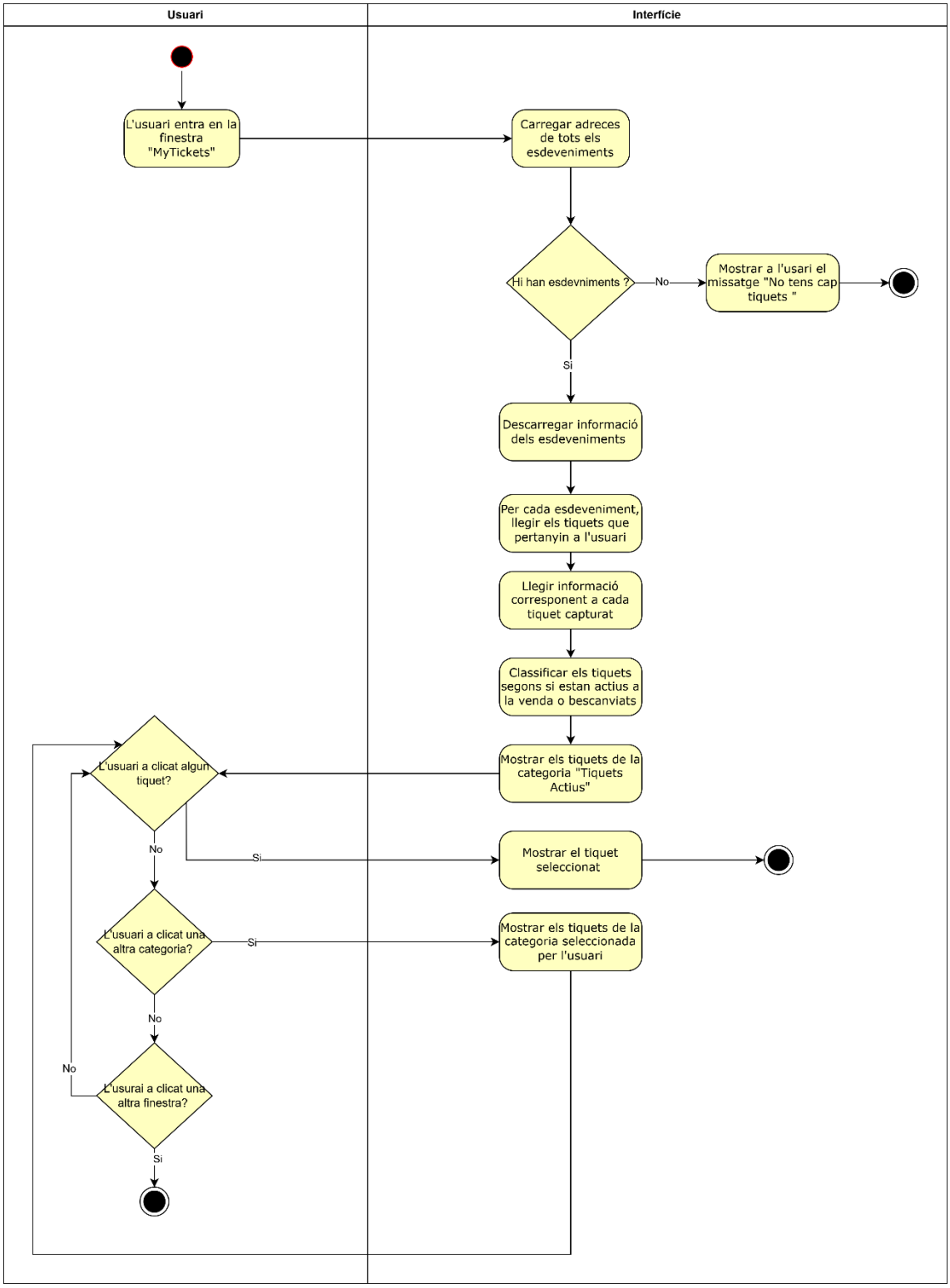
EventDetailPageResale

Si bé aquest component també és molt similar al que trobem en la pàgina de “Events”, s’ha hagut de fer algunes modificacions extres com l’eliminació de l'estat “coming soon” (ja que només es mostren els esdeveniments purament actius), i s’han canviat les lectures d’estoc de la finestra Events, per la lectura dels tiquets ID que pertanyen al smart contract (excloent els identificadors que són del mateix usuari). Una vegada s’obtenen els identificadors, per cada tiquet es crea una estructura que conté tota la infromació per a que es mostri una targeta per cada entrada a la venda.



MyTickets

Aquesta pàgina mostra les entrades NFT que té l'usuari, així com el seu estat: si estan a la venda, si estan actives per poder-se bescanviar o si estan finalitzades (quan l'esdeveniment ha finalitzat o l'entrada ha estat bescanviada).



De maner mes tècnica i sobre el codi desenvolupat, aquesta finestra s'encarrega d'obtenir tota la informació necessària per a després fer la representació gràfica. En concret, necessita:

- Els tiquets que té l'usuari (siguin seus o els que pertanyen al contracte perquè estan a la venda)
- El temps de finalització de l'esdeveniment
- La infromació sobre el tiquet i l'esdeveniment que es troba fora de la cadena de blocs
- La informació de cada tiquet [l'estat, si està pendent de bescanviar-se, el temps restat per poder bescanviar el tiquet, i altres paràmetres com el preu de venda, el venedor o el comitHash generat per l'usuari quan vol bescanviar l'entrada per l'accés].

Una vegada s'obté tota la informació, per a cada esdeveniment els tiquets es filtren en 3 categories:

Active: Tiquets que pertanyen de forma directa a l'usuari i que el seu estat no sigui el de "redeemed".

OnSale: Tiquets que el propietari és el smart contract i el esdeveniment encara no ha finalitzat.

Finished: Són els tiquets que es troben amb l'estat redeemed o si estaven a la venda però l'esdeveniment ha finalitzat.

OBS: l'estat de Pending no es té en compte, ja que és un estat intermedi derivat de l'active.

A continuació es presenten altres components desenvolupats per la finestra MyTickets:

EventAccordion

Aquest component rep un objecte que conté tota la informació sobre un dels esdeveniments, i rep també un callback[7]. Aquest callback és essencial perquè dintre del component pare es pugui invocar a TicketDetailModal (visualització d'un tiquet en específic quan l'usuari clica un).

El component també s'encarrega d'extreure la imatge i el nom de l'esdeveniment per a muntar el desplegable. Una vegada es fa clic sobre el desplegable es posa a true el booleà i comença a tants components TicketItems com tiquets tingui l'usuari per aquell esdeveniment.

TicketItem

Rep com a paràmetres l'adreça de l'esdeveniment, un objecte tiquet que conté tota la informació sobre l'estat del tiquet, el temps de bescanvi i un callback de eventAccordion.

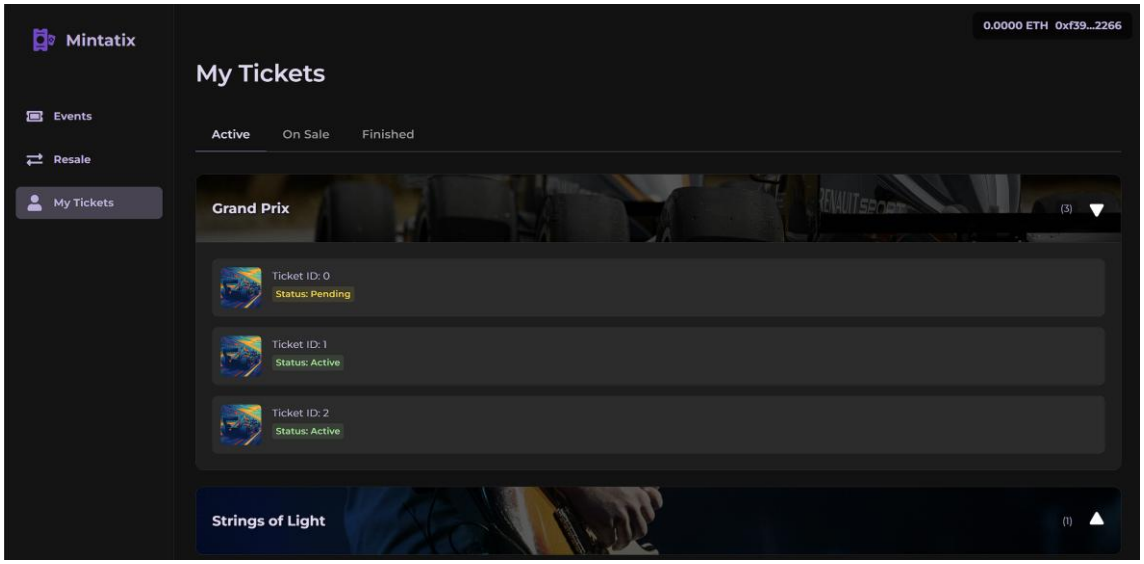
Inicialment, llegeix el tokenURI del tiquet per extreure la imatge del NFT i així representar-la com el tiquet que ha comprat l'usuari. Una vegada té la informació,

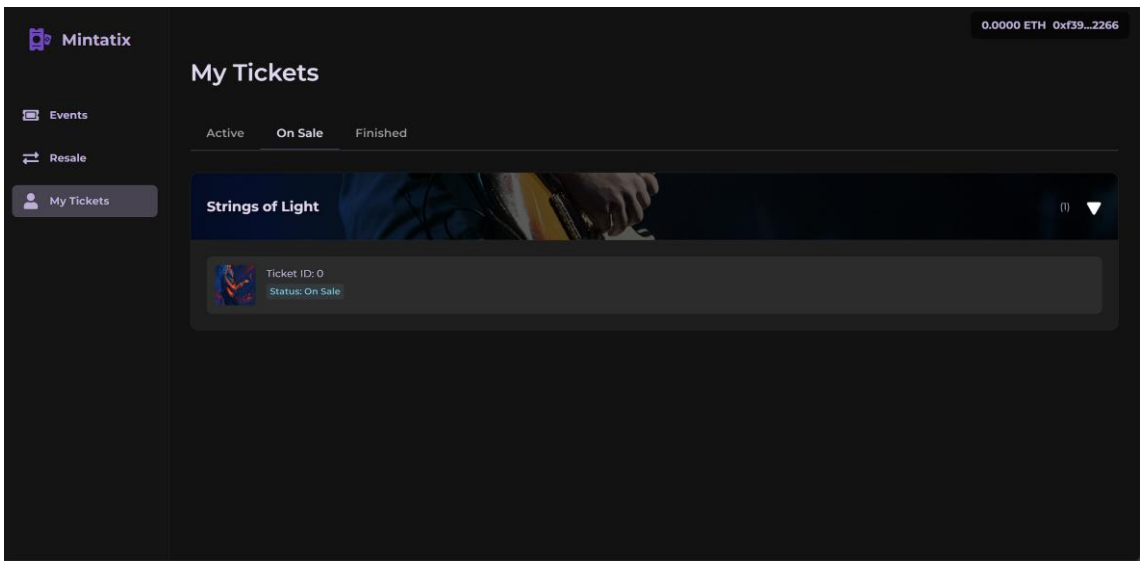
[7] Callback: Funció que es passa com argument a una altra funció per a que l'executi quan sigui necessari. Per exemple, en aquest cas el callback executa una funció que activa la finestra que mostra el tiquet del usuari.

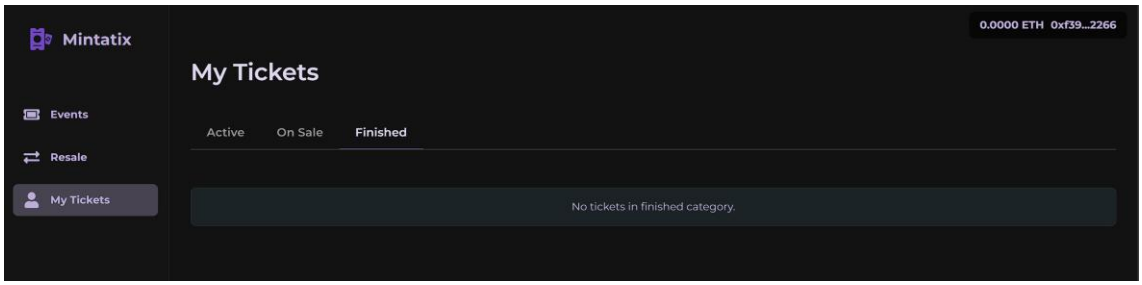
renderitza la imatge NFT , l'identificador del tiquet i el estat del tiquet, que pot ser ‘Active’, ‘Pending’, ‘OnSale’, ‘Redeemed’.

Quan l’usuari fa clic a un TikcetItem aquest avisa al component pare “EventAccordion”, i seguidament aquest envia la infromació sobre el tiquet clicat a la finestra MyTickets de forma que aquest genera un objecte amb la informació mínima necessària per a poder representar el TicketDetailModal on és mostra a l'usuari tota la informació del tiquet seleccionat.

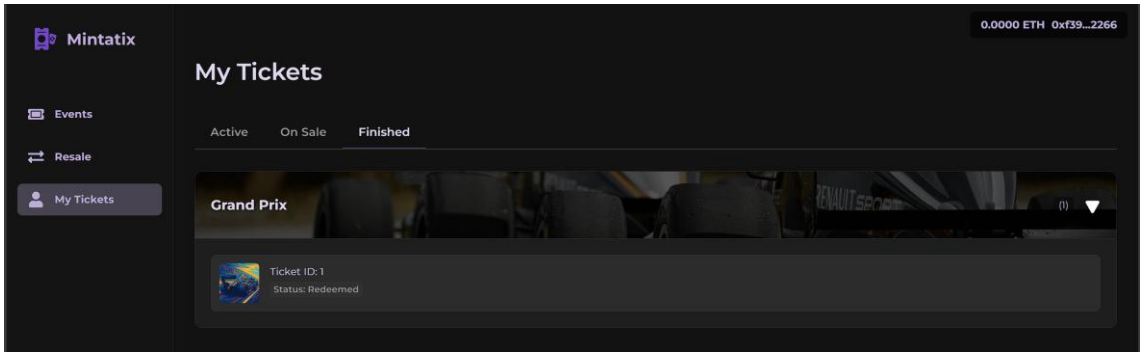
A continuació es mostra gràficament un cas d’us de la finestra MyTickets on un usuari ha comprat 3 tiquets per l’esdeveniment “Grand Prix” (un d’ells pendent de bescanvi) i 2 tiquets per l’esdeveniment “Strings of Lights” on el eventAccordion no està obert i un dels tiquets està a la venda.







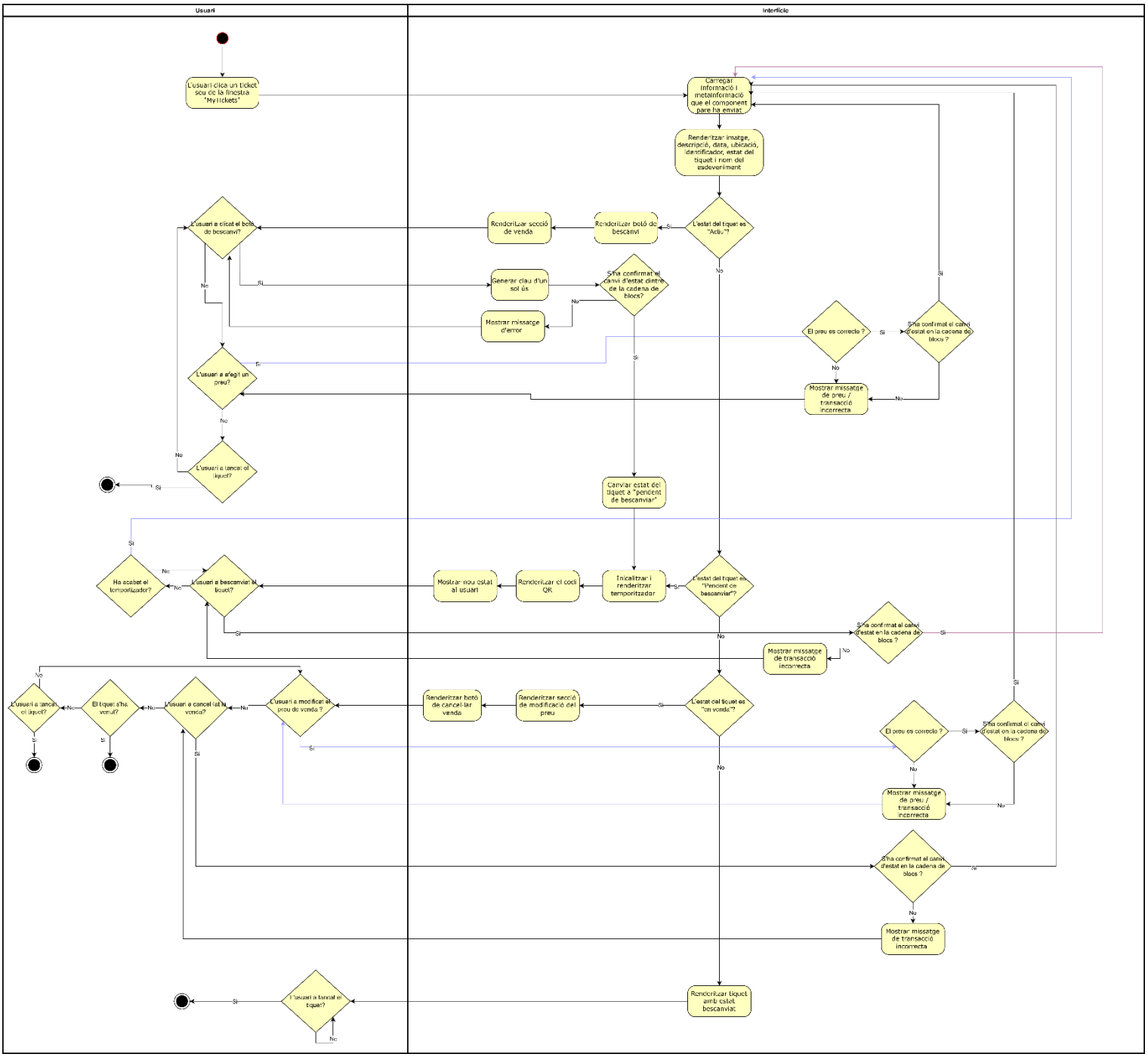
Cas en que l'usuari a bescanviat el tiquet que estava en "pendig" per l'accés al esdeveniment.



TicketDetailModal

Aquest component te l'objectiu de mostrar la informació rellevant sobre el tiquet que ha seleccionat l'usuari, de forma que el component MyTickets quan se selecciona un tiquet, prepara una estructura de dades que envia a quest component per a poder renderitzar l'estat actual del tiquet sigui "Active", "Pending", "OnSale" o "Redeemed".

A continuació es presenta el diagrama de flux que representa el funcionament d'aquest últim component.



Per resumir el diagrama de flux i explicar de manera mes tècnica, el component rep del component pare (la finestra My Tickets) una estructura de dades i un callback per poder indicar al pare que aquest component no s'ha de visualitzar. El component també rep una funció la qual en executar-se crida al pare per a que faci les lectures dels estats del tiquets a la cadena de blocs, de forma que aquest component pugui actualitzar l'estat quan l'usuari fa alguna acció.

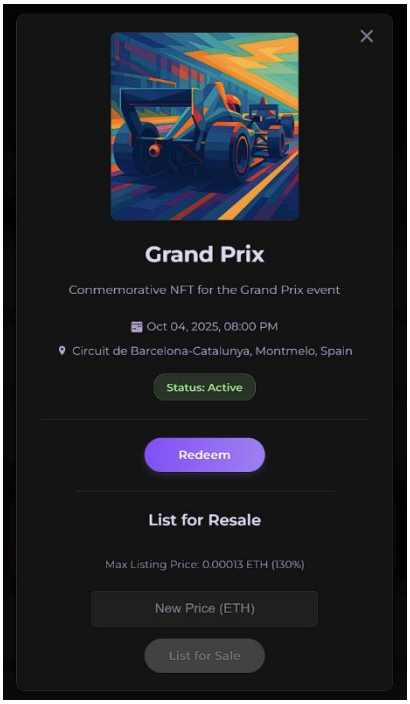
Respecte a funcionalitats, aquest component permet a l'usuari bescanviar el tiquet per l'accés generant un codi QR[8] d'un sol ús que haurà de ser llegit per un validador. Una vegada es crea aquest QR, i es posa l'estat de "pending" es genera un comptador per a poder mostrar a l'usuari el temps restant que té per bescanviar l'entrada abans d'haver de tornar a posar en Active. Per aquesta part, testejant la versió mòbil, em vaig adonar que com per a signar la transacció de canvi d'estat l'usuari ha d'anar a l'aplicació o té la seva cartera, en tornar a la web, aquesta es refrescava, de forma que es perdia el valor secret i, per tant, no podia tornar a generar el QR.

És per això que una vegada es genera el QR amb el valor secret, aquest es guarda a la memòria del navegador, de forma que, quan retorni, el component verifica si havia un valor secret guardat i, si hi havia un i ara l'estat del tiquet és "pending" llavors torna a renderitzar el temporitzador i el QR, possibilitant així que l'usuari pugui sortir i entrar del component sense perdre el QR ni el temporitzador. [*Observació: també s'ha tingut en compte que potser el tiquet s'ha posat en pending dintre d'altre dispositiu, llavors si està en pending, però no tenim res a la memòria de navegador, vol dir que l'usuari ha utilitzat una altra sessió amb la mateixa wallet per fer el canvi d'estat, llavors es mostrarà al dispositiu un missatge informant que el tiquet s'està bescanviant en una altra sessió.*].

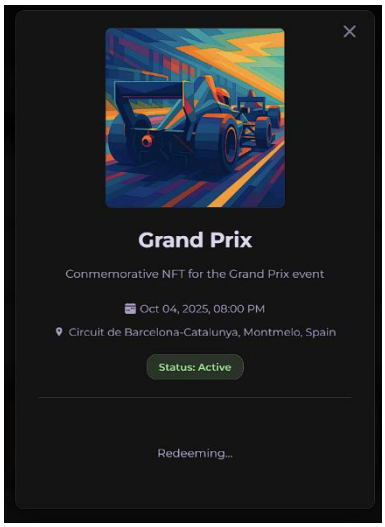
A més, dintre de l'estat en venda, mostra a l'usuari l'opció de cancel·lar la venda o canviar el preu . [*Observació: si l'usuari posa un preu superior al 130% del preu original, mostra un avís i no envia la transacció al smart contract*]

A contentació es presenten tres escenaris per mostrar el component:

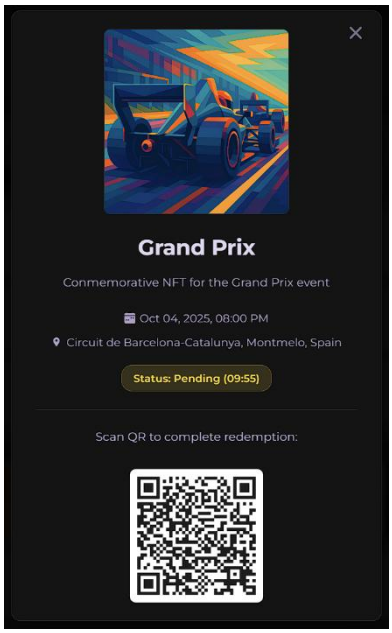
Escenari 1: L'usuari bescanvia el tiquet:



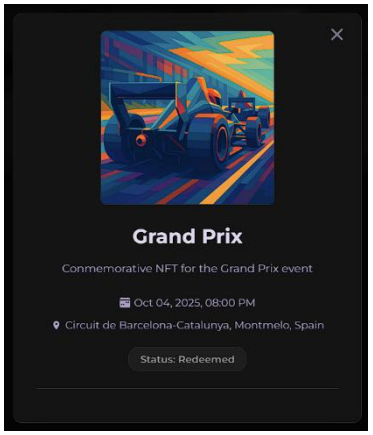
L'usuari selecciona un tiquet



L'usuari clica el botó "Redeem"



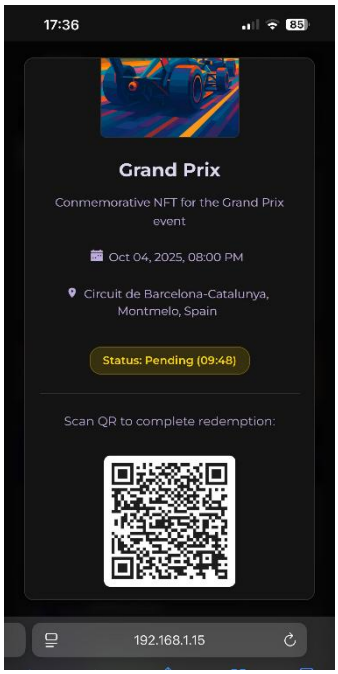
Es genera el QR i el temporitzador



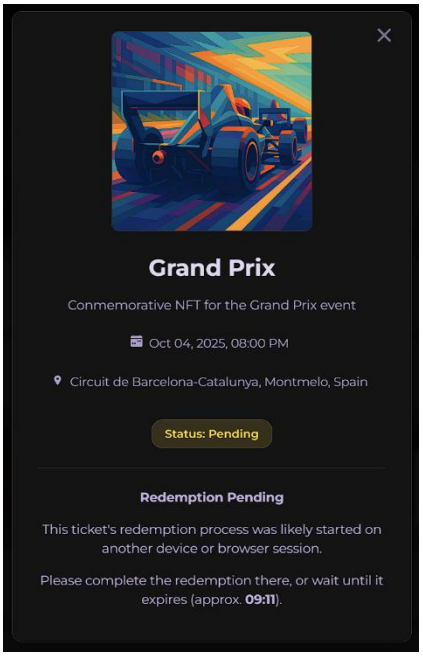
L'usuari ha escanjat el QR i s'ha bescanviat

[8] Es pot trobar mes infromació sobre aquesta implantació al apartat "*Implementació d'un esquema commit-reveal per bescanviar entrades*"

Escenari 2: L'usuari bescanvia el tiquet al dispositiu mòbil i vol veure el tiquet al ordinador

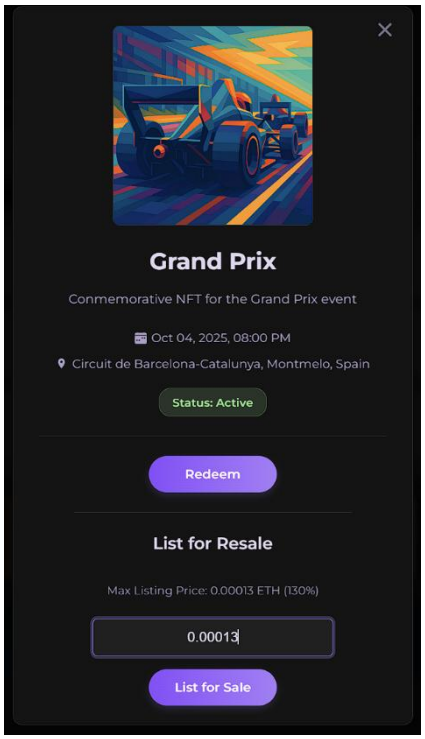


L'usuari bescanvia el tiquet al mòbil

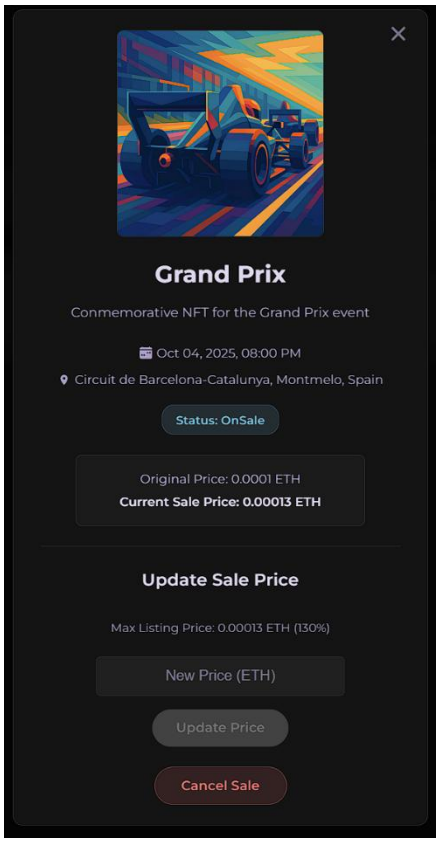


L'usuari revisa el tiquet en l'ordinador

Escenari 3: L'usuari posa el tiquet a la venda:



L'usuari introdueix el preu per revendre



El tiquet mostra l'estat en revenda

Desplegament del Smart Contract a una Testnet

Una vegada conclòs el desenvolupament del smart contract i s'ha revisat el correcte funcionament de les característiques desitjades per la Dapp, s'ha decidit llençar el smart contract a una xarxa de prova real. Dintre de Metamask (la cartera que estic fent servir per testejar el projecte) ja ve integrada una xarxa anomenada "Sepolia" que és unes de les xarxes de test més conegudes per Ethereum per la seva estabilitat. És per això que em vaig decantar per aquesta xarxa.

Ara, amb la xarxa seleccionada, per a poder pujar el smart contract primer de tot s'ha d'aconseguir monedes, perquè quan es puja un smart contrcat s'han de consumir recursos per processar la transacció i pujar el smart contract, ja que aquest ocupa espai dintre d'un bloc.

Per obtenir les monedes dintre d'una adreça meva, he utilitzat dos faucets[9] : Google Cloud: sepolia Faucet (regala 0.05 ETH diaris) i Sepolia PoW Faucet la qual a partir d'un treball computacional, et dona a canvi ETH en funció del treball realitzat.

Una vegada obtinguts els fons necessaris, únicament s'ha hagut de canviar uns paràmetres al fitxer de configuració de Hardhat (el framework que va ser utilitzat durant el desenvolupament del smart contract) per habilitar despagaments a Sepolia.

A més, s'ha d'indicar quin node farà el desplegament, és per això que he hagut de llegir la documentació de metamask i he hagut de fer-me un compte de desenvolupador. Amb aquest compte he obtingut accés a la meva API, amb la qual he pogut fer el desplegament. Una vegada tot configurat, com durant la realització del primer informe vaig crear un fitxer que feia els desplegaments de forma automàtica per a la meva xarxa local de desenvolupament, només ha sigut necessari indicar que ara el fitxer ha de fer el desplegament en la xarxa de Sepolia.

A continuació adjunto una captura de la comanda final per fer el desplegament i les adreces corresponents als quatre esdeveniments i el contracte fabrica (contracte que permet crear nous esdeveniments).

```
npx hardhat ignition deploy ./Mintatix.js --network sepolia
√ Confirm deploy to network sepolia (11155111)? ... yes
Hardhat Ignition 🚀

Deploying [ Mintatix ]
```

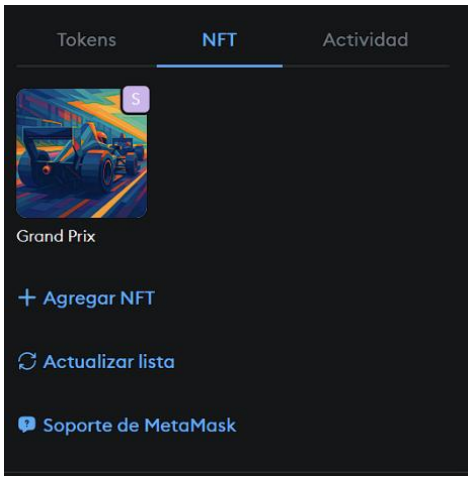
```
Deployed Addresses

Mintatix#EventTicketFactory - 0x473Ab55c19EC8Fdb4878Ba6FB8B81D40d8B0b788
Mintatix#logicAddr - 0x8e0A7b41bFe9A53Ff5f42F560719bB58c51053C4
Mintatix#proxy_EF - 0xD702c5EC254ccd7E6a32E606C50859616c11250e
Mintatix#proxy_GP - 0x10B088F6Ef254fd5358C44FB001695EBDcF93198
Mintatix#proxy_RC - 0x1AF43eB3489A9124ddE7a9280cA095B7E08B2CFA
Mintatix#proxy_SOL - 0xbBC19933FD534Bf33B3864da034f855c271fe516
```

Una vegada fet el desplegament, per poder interactuar amb ells esdeveniments desplegats amb la interfície desenvolupada, únicament ha sigut necessari canviar l'adreça a la qual la interfície demanava la llista de les adreces dels esdeveniments, i l'adreça del contracte amb la lògica.

[9] Faucet: Dintre del mon de les criptomònades, es una eina que permet obtenir monedes de manera gratuïta (o complit alguns requisits) a l'adreça que donis. Normalment s'utilitza per promocionar xarxes noves o per a que els desenvolupadors puguin fer-la servir.

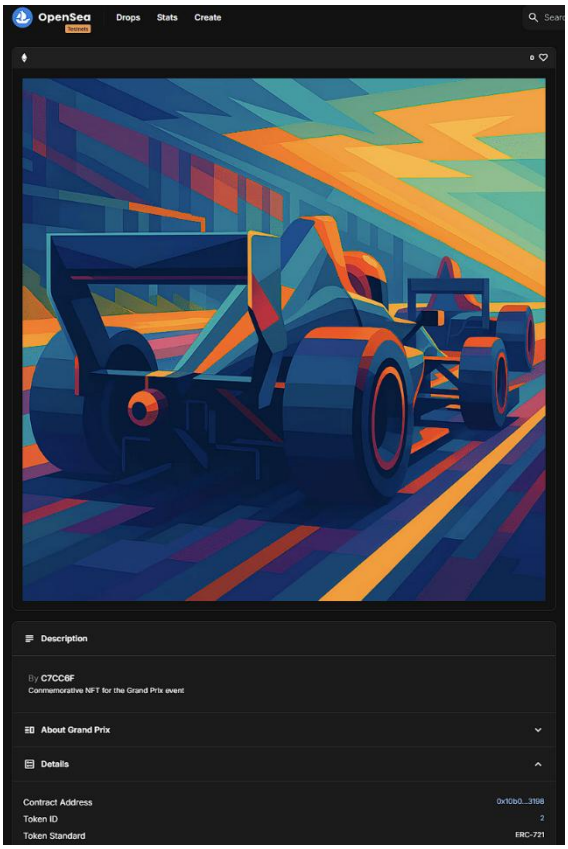
A més s'ha publicat un servidor provisional amb la Dapp, de forma que qualsevol persona pot testejar i fer-ne us de la interfície per interactuar amb els smart contracts desplegats ala xarxa de Sepolia [https://mintatix.ddns.net/].

A continuació es presenta una captura del wallet de metamask on es pot observar la compra satisfactòria d'una entrada NFT.



Com ara el smart contract forma part d'una xarxa de test real, es poden utilitzar buscadors que revisen la xarxa de Sepolia per poder veure les transaccions que hi hagin sobre el els esdeveniments desplegats. Adjunto una captura de pantalla dintre d'un d'aquests buscadors, amb el qual, he pogut observar la compra d'una entrada que he fet amb la meva carteta digital.

[Enllaç a la pàgina on es pot observar la transacció de compra realitzada i la pertinença de l'entrada NFT a la meva adreça virtual: https://testnets.opensea.io/assets/sepolia/0x10b088f6ef254fd5358c44fb001695ebdcf93198/2]



Objectius completats

Finalment, durant el desenvolupament d'aquesta fase només restaven per fer com a objectiu primari el desenvolupament de l'aplicació web / interfície que permeti interactuar amb el smart contract.

Aquest s'ha completat de manera satisfactòria durant aquesta fase tal com s'ha anat descrivint en aquest informe.

No obstant això, també s'han complert els objectius secundaris que encara no s'havien realitzat en el primer informe. Aquests objectius eren:

- Creació de codis QR per a una gestió d'accés als esdeveniments més fluida.
- Creació del token URI i implementació a un IPFS per afegir més informació a l'entrada (nom de l'esdeveniment, data i hora, preu de compra, informació postvenda...).
- Llençar el Smart Contract a una Testnet d'Ethereum sortint així del entorn local.

Com a resultat d'aquesta fase, s'han complert tots els objectius que es van esmentar a l'informe inicial d'aquest projecte. Els quals han sigut un total de 14 objectius. (10 primaris i 5 secundaris):

- Permetre la compra d'entrades NFT en estoc.
- Permetre la venda d'entrades NFT.
- Permetre la consulta d'entrades NFT a la venda al mercat amb el seu preu.
- Permetre la compra d'entrades NFT que estiguin venent altres usuaris.
- Establir mesures preventives contra monopoli d'entrades, revenda massiva, inflació de preus, venda d'entrades ja bescanviades i compravenda d'entrades quan l'esdeveniment ha terminat.
- Complir amb els estàndards de seguretat per a evitar possibles vectors d'atac.
- Complir amb els estàndards dels NFT com el ERC721[10] i el ERC165[11]
- Poder consultar la legitimitat d'una entrada.
- Desenvolupar una aplicació web que pugui interactuar amb el Smart Contract de forma que faci de pont/interfície entre el Wallet del usuari i el Smart Contract.
- Implementació avançada del codi perquè les transaccions siguin eficients pel que fa a la despesa de Gas.
- Establiment de rols com verificadors (un verificador seria l'operari encarregat de verificar la pertinença de l'entrada i bescanviar-la per l'accés a l'esdeveniment).
- Creació de codis QR per a una gestió d'accés als esdeveniments més fluida.
- Creació del token URI i implementació a un IPFS per afegir més informació a l'entrada (nom de l'esdeveniment, data i hora, preu de compra, informació postvenda...).
- Llençar el Smart Contract a una Testnet d'Ethereum sortint així del entorn local.

[10] **ERC-721**: Conjunt de regles establertes per a que es puguin manipular els NFT entre aplicacions i Smart Contracts .

[11] **ERC-165**: Regla que defineix un mètode per a que els Smart Contrats detectin i publiquin quins altres estàndards compleixen.

Canvis en els objectius

En aquesta secció es defineixen diferents iteracions on s'ha necessitat revisar i modificar el smart contract per incloure noves funcionalitats, canvis o optimitzacions per alinear el Smart Contrcat i la Dapp.

EventURI i optimització de bytecode[12]

S'ha hagut de modificar el Smart contrcat per a poder afegir un EventURI a cada Smart Contract, de forma que cada esdeveniment té un enllaç IPFS a un fitcher el qual indica el nom de l'esdeveniment, una descripció, el CID[13] a la imatge de la portada que utilitza el Smart Contract i la ubicació de l'esdeveniment. També s'ha afegit una nova variable que indica a partir de quina data es poden comprar les entrades, de forma que els usuaris dintre de la Dapp podran veure els nous esdeveniments i informar-se sobre quan estarà disponible la compra d'entrades.

En fer aquests canvis, s'ha trobat un problema amb el nombre de paràmetres que pot tenir una funció, el problema resideix en què en incloure la nova variable eventURI, hi ha massa paràmetres dintre de l'inicialitzador d'esdeveniments. Per solucionar-ho s'ha hagut de crear una estructura que porta totes les variables a dins, de forma que Solidity el compta com una única variable i no sobrepassa el límit permès.

Solucionar aquest error introduint una estructura de dades, ha produït una altra vegada l'aparició de la limitació de la mida del bytecode. Aquesta vegada, l'excedent ha sigut d'uns 1000 bytes i s'ha optat per aplicar els següents tres canvis:

1. Una de les comprovacions més recurrents dintre del smart contract és la data de finalització de l'esdeveniment. Aquesta comprovació s'utilitza en 5 funcions. És per això que s'ha decidit crear una funció que inclou aquesta comprovació i una nova comprovació per veure si els usuaris ja poden començar a comprar entrades o no. D'aquesta forma s'elimina redundància, i s'ha aconseguit disminuir el bytecode uns 300 bytes.

2. En segon lloc, s'han reduït la mida dels missatges que surten quan una comprovació falla, d'aquesta forma bytecode no ha de guardar llargues cadenes de text. Aquest canvi ha comportat una disminució del bytecode d'uns 670 bytes. Però en fer aquest canvi s'ha hagut de modificar la majoria dels tests, ja que per poder veure si un test funcionava bé, la forma de verificar que una comprovació s'activa és llegint el missatge que porta.

3. Finalment, per poder reduir els últims 30 bytes restants que es troben per sobre del límit s'ha eliminat el setter que permetia al propietari modificar el temps que

[12] Bytecode: Conjunt d'instruccions en les quals s'ha de compilar un smart contract per poder desplegar-lo en una xarxa. Aquest ha de respectar una mida màxima per les restriccions imposades en les xarxes d'Ethereum.

[13] CID (Content Identifier): Identificador únic que apunta al contingut emmagatzemat dintre del IPFS [NOTA: El sistema IPFS s'analitza en detall en la secció (Implementació d'un IPFS pels NFT)]

una entrada podia estar en pending, ja que actualment estava codificat per 10 minuts i no té massa sentit modificar-ho durant l'esdeveniment.

Observació: Com ara tenim dos temps, el d'inici de compra i el de finalització de l'esdeveniment s'ha optat per canviar la forma d'introduir els límits de temps Ara al moment de crear el smart contract s'ha d'afegir els temps en format Unix, de forma que es té un control més precís de quan comença la venda d'entrades i quan acaba l'esdeveniment.

ERC721AQueryable i optimitzacions de bytecode

Dintre de la Dapp, hi ha un apartat anomenat "My tiquets" que gestiona els tiquets que el usuari ha comprat. Per a poder saber quins tiquets pertanyen a l'usuari tenim dos opcions:

La primera opció, seria obrir un servidor encarregat d'escoltar tots els esdeveniments que llença el smart Contract, de forma que es pot generar una "base de dades" fora de la cadena a partir d'aquesta informació.

La segona opció, seria tractar de crear mètodes que siguin eficients i capaços de poder llegir quins tiquets corresponen a cada usuari a partir de la seva adreça. Aquesta opció encara que pot fer més costos el desplegament, encaixa millor amb la metodologia del treball que s'ha estat seguint fins ara, i és per això que s'ha decidit començar explorant aquesta opció.

En una primera cerca, com el smart contract ja utilitza la llibreria ERC721A[14], he indagat en la documentació per poder veure si hi havia alguna informació per trobar quins NFT pertanyen a cada usuari. Durant la cerca, s'ha trobat que els mateixos desenvolupadors de la llibreria ERC721A tenen una extensió anomenada "ERC721AQueryable" la qual permet exactament fer les cerques a partir de l'adreça de la cartera de l'usuari. Per implementar-ho només ha sigut necessari referenciar aquesta extensió i afegir el seu inicialitzador per poder començar a utilitzar les funcions que integra l'expansió. No obstant això, en afegir-ho, com la mida del bytecode de cada llibreria utilitzada se suma a la mida del bytecode del contracte, s'ha tornat a trobar amb el límit de mida del bytecode. Aquesta vegada, sobrepassant el límit per 2408 bytes tal com es pot apreciar a la següent imatge:

```
Warning: Contract code size is 26984 bytes and exceeds 24576 bytes (a limit introduced in Spurious Dragon).
 This contract may not be deployable on Mainnet. Consider enabling the optimizer (with a low "runs" value!)
, turning off revert strings, or using libraries.
  --> contracts/EventTicketLogic.sol:13:1:
```

Davant d'aquest problema tan recurrent, he considerat si hauria d'utilitzar les optimitzacions del compilador, però, agafant els raonaments que ja es van donar a l'informe anterior, s'ha dut a terme una anàlisi de quines possibilitats tinc per a poder reduir aquesta quantitat i s'han aplicat les següents mesures:

Primerament, en ser una quantitat de bytes gran, és necessari fer alguna modificació que pugui reduir l'espai de manera significativa, és per això que com a primera solució

[14] ERC721A: Llibreria utilitzada en el desenvolupament del smart contract, permet crear diferents entrades NFT a la vegada reduint la computació necessària, i per tant redueix les taxes atribuïdes a l'execució d'una transacció.

s'ha optat per treure tots els missatges d'error dels "require[15]" de tot el projecte. Encara que aquest ja estaven reduïts com a solució de l'anterior problema, els missatges d'error es guarden directament en el bytecode per després mostrar-los al usuari. Com cada caràcter del missatge ocupa un byte, el número de bytes totals pot fer una suma considerable. En aplicar aquest canvi, l'alliberament d'espai ha sigut suficient per a solucionar el problema de la mida del bytecode. Però, a efectes futurs per a debuggar el codi, treure aquests missatges no és gens pràctic, i és per això que s'ha tornat a fer una altra recerca de com es podria abordar el problema.

En aquesta segona cerca, s'ha acudit a la web "Downsizing contracts to fight the contract size limit" la qual vaig utilitzar la primera vegada per poder veure quines possibilitats de reducció del bytecode tenia. Dintre d'aquesta web, en l'apartat "Medium impact" es menciona l'ús de "custom errors" en comptes de fer servir els require amb missatges d'error (també anomenats revert strings).

Aquesta solució es basa en com solidity emmagtazema els "custom errors", i és que, els custom errors es codifiquen com funcions i es compacten en 4 bytes de forma que aquests errors es poden posar qualsevol nom i sempre ocupen la mateixa mida per al bytecode. És per això que en un intent de recuperar almenys el tipus d'error, s'ha decidit aplicar els canvis, creant així un total de 16 custom errors. A efectes pràctics, els custom errors funcionen de manera semblant als revert, ja que també s'han d'encapsular en un condicional que, a l'avaluar-se com a True, aquests s'executen revertint la transacció de l'usuari perquè no pugui realitzar una acció no permesa.

Una vegada implementats, es pot observar com, si bé encara es segueix sobrepassant la mida límit, el bytecode del contracte es troba a 404 bytes d'arribar a la mida màxima suportada.

És a dir, canviar els revert i require que contenien strings pels nous custom errors ha fet que la mida del codi es redueixi uns 2000 bytes.

```
Warning: Contract code size is 24980 bytes and exceeds 24576 bytes (a limit introduced in Spurious Dragon).
 This contract may not be deployable on Mainnet. Consider enabling the optimizer (with a low "runs" value!)
, turning off revert strings, or using libraries.
  --> contracts/EventTicketLogic.sol:13:1:
```

Però encara és necessari reduir més la mida. Així que s'ha tornat a fer una altra cerca per investigar noves possibles solucions. A causa de l'esgotament d'idees d'optimització, s'ha considerat la possibilitat d'eliminar alguna de les dependències que tenia dins del smart contract. Fent una anàlisi exhaustiva de l'ús de cadascun, considero que totes les dependències utilitzades estan justificades i són rellevants per al projecte, però hi ha una dependència en concret que podria fer la meva pròpia implementació (mes lleugera) i aconseguiria arribar a la mida màxima permesa. Aquesta dependència ha sigut "AccessControlUpgradeable". Aquesta dependència l'utilitzava per establir rols dintre del smart contract, però pel projecte només és necessari el rol de validador a més del rol de owner. A més, el rol de validador només serveix per invocar a una única funció: "setRedeemedToTKT" que fa la transició del tiquet que està en pendent de bescanvi a bescanviat (Redeemed).

[15] Require: Són condicions que pot definir el desenvolupador dintre del smart contract, si la condició és falsa, la transacció de l'usuari es cancel·la. El desenvolupador pot afegir un missatge per a que l'usuari entengui perquè la seva transacció s'ha cancel·lat.

Si bé utilitzar la dependència és una forma més robusta, aquesta està pensada per a projectes que requereixen diversos tipus de rols, ja que conté optimitzacions per poder tenir molts rols ocupant poc espai. Però, el seu ús per un únic rol, no és l'òptim possible, pel fet que generar un mapping[16] que contingui l'adreça del validador (clau ùncia)i un booleà (el valor), és més lleuger quan només es necessita un únic rol. És per això que, com a mesura per reduir el bytecode, s'ha optat per aquesta nova implementació del mapping, eliminant la dependència AccessControlUpgradeable. Finalment, i després d'haver aplicat aquest últim canvi, la mida del bytecode ha baixat per sota del límit, de forma que el compilador ja no dona cap advertència.

EnumerableSet i gestió de tiquets en venda

L'usuari necessita saber quins tiquets té a la venda. Per a poder conèixer aquesta informació, podria agafar l'extensió "ERC721AQueryable" que va ser afegida anteriorment, i preguntar per tots els tiquets que tenen com a owner l'adreça del contracte. Si bé aquesta crida em donaria accés a tots els identificadors dels tiquets que es troben a la venda, una vegada obtinguts, seria necessari llegir quina es l'adreça que està venent cada tiquet i agafar les que coincideixin amb l'adreça de l'susari connectat.

Raonant, s'ha arribat a la conclusió que aquesta metodologia no és òptima, ja que comporta que cada usuari ha de fer tantes crides al smart contract, com el nombre d'entrades hi hagi en venda.

És per això que com a solució s'ha afegit un nou array que pugui guardar per una determinada adreça quins són els TokenID que estan a la venda. De forma que amb una única crida ja obtenim els tokenID per poder representar els tiquets de l'usuari.

Però, per a fer-ho seria òptim que aquest array fos dinàmic, ja que l'usuari pot afegir o treure la seva entrada, però, en Solidity no hi ha cap forma de variar la mida d'un array de forma nativa. És per això que fent una cerca sobre aquest problema, s'ha trobat com la llibreria de Openzeppelin, té una implementació que fa justament això, anomenada "EnumerableSet", de forma que és possible eliminar i afegir identificadors perquè després amb una única lectura al smart contrcat, aquest ens pugui retornar directament quines són les entrades a la venda. Afortunadament, la inclusió d'aquesta nova llibreria no ha provocat l'excés de la mida del bytecode.

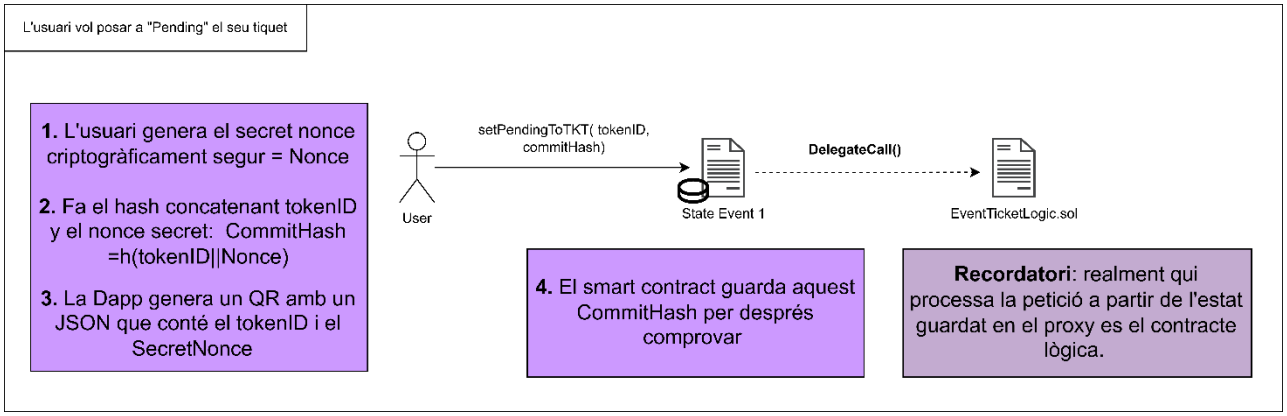
Implementació d'un esquema commit-reveal per bescanviar entrades

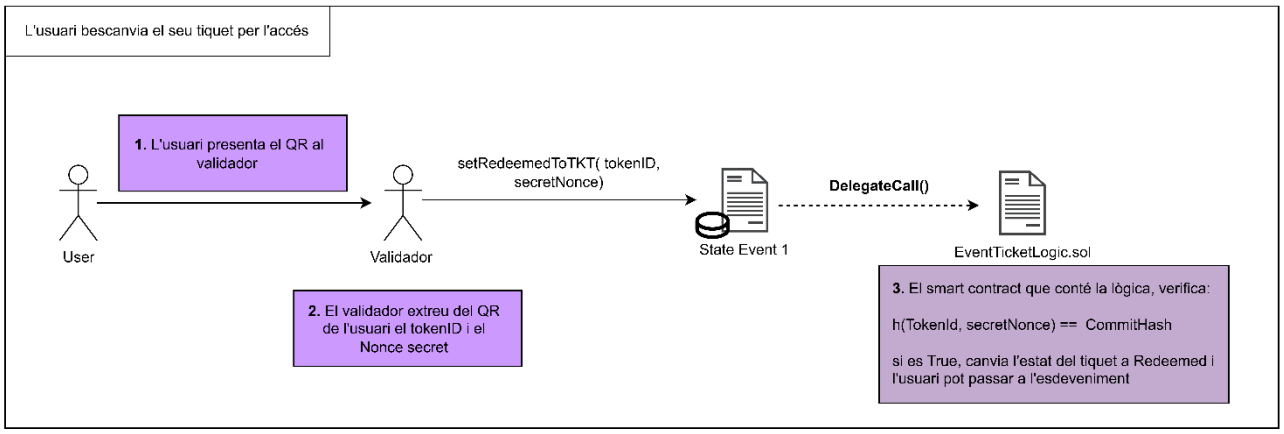
Durant la definició de la Dapp, un dels punts febles que no havia tingut en compte era l'operació de passar el tiquet de l'estat pending a l'estat "Redeemed" (bescanviat). Això és perquè fins ara, la informació a presentar a un validador per poder bescanviar l'entrada era simplement l'identificador d'entrada, fet que podria aprofitar un atacant per observar el canvi d'estat de l'entrada i presentar-se abans que l'usuari legítim per bescanviar ell l'entrada. Si bé aquest atac té una finestra d'acció bastant limitada pel fet que l'estat de "pending" només dura 10 minuts. Considero que podria arribar a ser prou temps perquè un atacant pugui efectuar l'operació. És per això, que s'ha optat per

[16] Mapping: Estructura semblant a un diccionari que permet associar una clau única amb un valor, de forma que es facilita la cerca d'informació.

crear un esquema de commit-reveal senzill perquè cap atacant pugui aprofitar aquesta vulnerabilitat.

Per implementar aquest esquema, ara la funció setPendingToTKT necessita com a paràmetres un hash keecka256[17] que serà el resltat de passar-li la concatenació del TokenID i un valor secret generat per l'usuari. D'aquesta forma, quan l'usuari vulgui bescanviar la seva entrada, el QR que presenta al validador haurà d'aportar l'identificador del tiquet i el valor secret utilitzat que va generar quan va posar a "pending" el tiquet, de forma que, l'única manera de bescanviar l'entrada per l'accés és conèixer aquest valor secret. Ja que dintre del smart contract es verifica que el hash que es va enviar per posar el tiquet en "pending" sigui el mateix que el generat quan fa hash(tokenID, valor_secret). D'aquesta manera s'afegeix una capa de seguretat extra per l'usuari. A continuació es mostres dos diagrames que exemplifiquen aquest esquema aplicat al projecte.





Observació 1: El QR només es pot generar en la màquina que ha fet el setPendingToTKT, ja que encara que tinguis la mateixa wallet en dos llocs diferents, no poden recalcular un valor aleatori, pel fet que es fa ús d'una funció criptogràfica de ethers.js i el càlcul no és "determinista". Si bé es podria fer de manera determinista que en comptes d'un valor aleatori fos el tokeId signat amb la clau privada de l'adreça, crec que això afecta negativament l'experiència d'usuari, sobretot en dispositius mòbils, ja que per signar has de canviar d'aplicació entre el wallet i la Dapp. És per això que la decisió final ha sigut deixar-ho d'aquesta forma i que només el dispositiu on s'ha posat el tiquet a "pending" sigui l'únic amb la capacitat de poder generar el QR. Aquesta nova implantació ha comportat el canvi dels tests automàtics

[17] Keccak256: Funció criptogràfica utilitzada en Etehreum per poder generar hash de 256 bits únics i irrepetibles.

de la funció setPendingToTKT de dintre del smart contract, i s'han generat 2 nous tests per comprovar el correcte funcionament dels custom Errors per validar el hash i el valor secret.

Observació 2: Normalment, en aquests tipus d'esquema es necessari signar amb la clau privada per a corroborar que ha sigut realment l'usuari qui s'ha compromés amb aquell valor secret, però com per a cridar a la funció de pending has de ser el propietari, al moment de bescanviar l'accés es pressuposa que únicament el propietari coneix el valor secret que ell mateix va generar.

Actualització del preu d'un tiquet en revenda

Abans, una vegada s'havia posat un preu al tiquet, si l'usuari volia canviar el preu, s'havia de: primer cancel·lar la venda i després tornar-ho a posar a la venda amb el nou preu, fent que l'usuari hagi de fer dues transaccions a la blockchain per únicament canviar el preu del seu tiquet en venda. És per això que s'ha modificat la funció que posava tiquets a la venda perquè si el tiquet ja està en venda i es torna a trucar a la funció amb un preu diferent, es modifiqui el preu del tiquet. Aquesta nova implantació ha comportat el canvi dels tests automàtics de la funció addTicketsForSale, i s'han generat 2 nous tests per comprovar el correcte funcionament dels custom Errors i per validar el correcte funcionament del canvi de preu.

Implementació d'un IPFS pels NFT

Si bé aquesta part estava prevista pel següent informe, per poder configurar la disposició dels elements i no haver de fer modificacions a posteriori en la lògica de la Dapp, he optat per avançar aquesta part.

Primerament, el IPFS són les sigles per definir l'Interplanetary File System. El qual es tracta d'un protocol dissenyat per poder emmagatzemar fitxers de manera descentralitzada. Aquest protocol utilitza una xarxa P2P[18] que permet que qualsevol node pugui rebre i emetre els seus propis fitxers. Per a compartir-los, cada fitxer pujat genera un CID que és un identificador únic que apunta al contingut emmagatzemat dintre del IPFS. I el node que té el fitxer passa a ser un "proveïdor" del fitxer. A més, si un fitxer és molt demanat és possible que altres nodes repliquin el fitxer per a distribuir-ho més ràpid.

Per aquest treball és útil utilitzar aquesta eina, ja que permet pujar fitxers com ara les imatges dels NFT i els JSON tant de l'esdeveniment com del NFT, actuant així com una "base de dades "però que només guarda fitxers immutables sense relacions entre aquests. Per a utilitzar-ho per al meu propi compte amb un node personal, la xarxa triga massa temps a aconseguir la informació a falta de nodes que repliquin la informació, ja que ara mateix no hi ha ningú que necessiti l'informació. És per això que he decidit fer servir una plataforma anomenada Pinata. Aquesta ofereix gratuïtament fins a 1 GB d'espai per posar els fitxers en diversos nodes de forma que la informació estigui replicada i sigui molt més accessible dintre de la xarxa P2P. Si bé aquesta opció rellisca amb la filosofia del projecte perquè s'ha de confiar en una empresa externa, mentre l'aplicació estigui en fases de prova i no sigui utilitzada per un conjunt de persones, els fitxers no s'estenen per la xarxa empitjorant l'experiència d'usuari a l'hora de fer servir la Dapp. Però una vegada estigués en "producció" si hi ha un

[18] Xarxa P2P: Sistema de comunicació on cada dispositiu pot fer de client i de servidor a la vegada, de forma que poden compartir i redistribuir fitxers directament entre nodes sense un servidor central.

nombre d'usuaris considerables seria factible usar el meu propi node combinat amb la replicació de la xarxa gràcies a la demanda dels usuaris pels fitxers.

Planificació del treball: Seguiment II

Anàlisi de la Segona fase

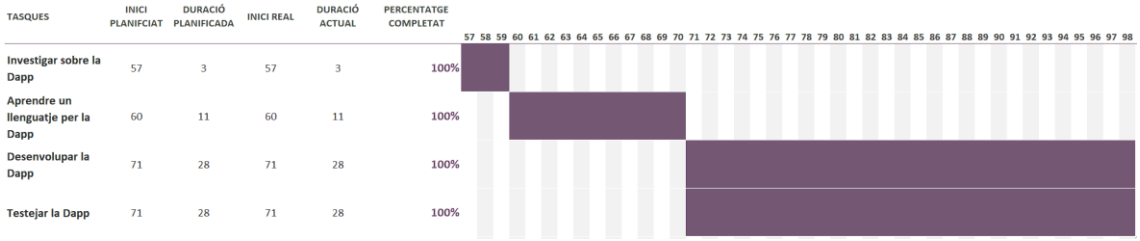
Continuant amb la planificació de l'Informe de seguiment I, les tasques que es van planificar per aquest informe son:

- Investigar sobre la Dapp (14 d'abril – 16 d'abril)
- Aprendre un llenguatje per la Dapp (17 d'abril -27 d'abril)
- Desenvolupar la Dapp (28 d'abril – 25 de maig)
- Testejar la Dapp (28 d'abril – 25 de maig)

Totes les tasques que es van mencionar s'han complert de manera satisfactòria i dintre dels terminis indicats. Únicament, s'han trobat més inconvenients durant el desenvolupament de la Dapp, ja que per complir amb els terminis ha sigut necessari dedicar un gran nombre d'hores diàriament, el qual ha sigut molt superior a la mitja d'hores diàries de les altres tasques a realitzar.

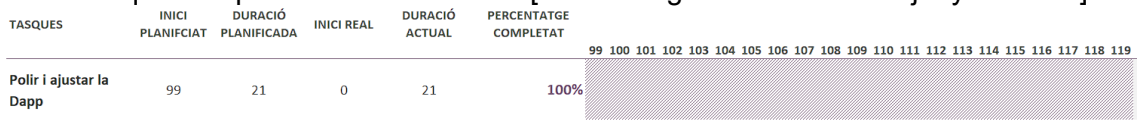
A més el fet de fer el desenvolupament de la Dapp de forma satisfactòria ha inclòs la realització de diversos objectius secundaris que ja hem mencionat com: la creació de codis QR per la gestió d'accés, la creació del token URI i implementació a un IPFS i el llançament del Smart Contract a una Testnet d'Ethereum.

Gràficament i, segons el diagrama de Gantt establert: 14 d'abril de 2025 fins al 25 de maig de 2025. [Dies 57 a 98]



Tercera Fase: Preparació

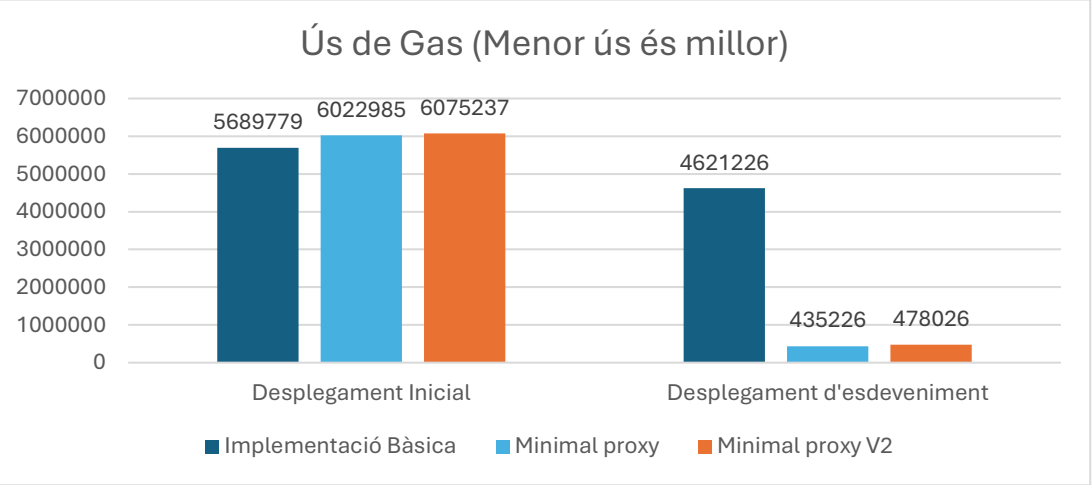
Si bé per la següent fase una de les tasques era acabar d'unificar Smart Contract i Dapp, com actualment el sistema ja està unificat i plenament funcional aquesta última tasca canvia a un rol més de depuració i ajustaments per acabar de polir el sistema desenvolupat. De forma que només quedaria aquesta última tasca dins del diagrama de Gantt que compren els dies 99 a 119 [26 de maig de 2025 – 15 de juny de 2025].



Resultats obtinguts

Després d'haver fet alguns canvis dintre del smart contract tal i com s'ha esmentat al apartat "Canvis en els Objectius", a continuació es presenta l'impacte que han tingut aquests canvis dintre del smart contract.

Per mesurar els canvis realitzats, s'utilitza com a mesura del Gas. El Gas és com en Ethereum es diu al treball computacional que s'ha de fer per processar una operació. Segons el quee s vulgui fer es consumeixen més recursos o menys, per exemple, no és el mateix fer una operació de transferir uns diners d'una cartera a una altra que fer un desplegament d'un smart contract. O no és el mateix fer una operació de suma que una operació d'emmagatzematge. En definitiva, cada transacció / instrucció que s'executa consumeix recursos i aquesta consumició té un preu que es diu Gas. En la gràfica que es va presentar a l'informe anterior es podia observar els consums de Gas d'enviar el smart contract a la xarxa de Ethereum (Desplegament inicial) i el cost de desplegar un esdeveniment, a continuació es compara les anteriors versions que es van presentar a l'informe anterior amb el smart contract actual després d'haver aplicat una sèrie de modificacions.



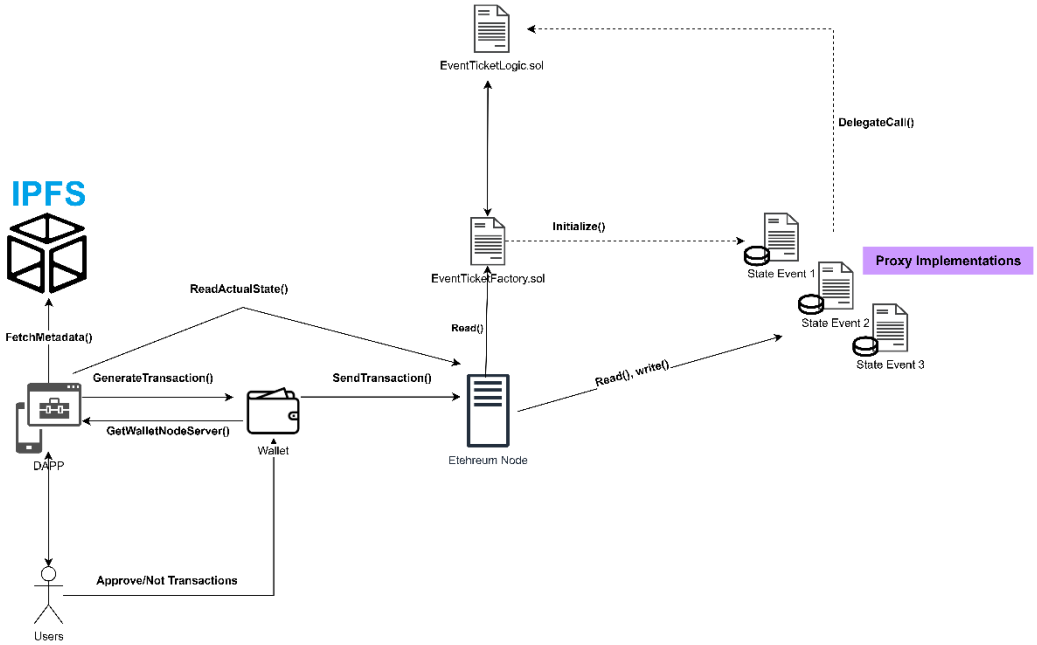
Gràcies al gràfic, podem veure com després de totes les modificacions, millores i optimitzacions aplicades descrites dintre de l'apartat "Canvis en els objectius", observem com el cost computacional (Gas) del desplegament del smart contrcat ha augmentat en 52252 unitats de Gas mentre que ara el cost computacional de generar un nou esdeveniment ha augmentat 42800 unitats de Gas.

Percentualment, seria un increment d'un 0,87% i un 9,83% respectivament vers la versió inicial del Minimal Proxy (l'última versió que es va esmentar dintre de l'Informe I). És a dir, degut a les noves funcionalitats i modificacions que s'han afegit, hem augmentat una mica el cost computacional de generar esdeveniments i desplegar el smart contract però és un bé necessari, ja que tots els canvis realitzats tenien com a objectiu minimitzar el nombre de lectures i escriptures dintre del smart contract fent així un sistema més eficient.

Per la banda de la Dapp, com ja s'ha estat esmentant al llarg d'aquest informe, s'han produït canvis al smart contract per a minimitzar al màxim possible les lectures i escriptures al smart contract, a la vegada que s'ha procurat no augmentar excessivament el consum de Gas. D'altra banda, a nivell més gràfic crec que la interfície desenvolupada durant aquest informe és prou entenedora i intuïtiva, ja que

s'ha buscat de reduir el nombre de clics dintre de la interfície, garantint un bon rendiment durant l'ús de la Dapp.

A continuació es presenta el diagrama final que mostra l'arquitectura del sistema desenvolupat de la manera mes acurada possible:



Com es pot observar en aquest diagrama, l'usuari interactua amb la Dapp (la interfície) utilitzant la seva cartera. Les "hot wallets" com Metamask, incorporen un enllaç a un node de la xarxa d'Ethereum al qual fan les peticions de llegir l'estat de la cadena de blocs i envien transaccions per poder comprar tiquets, gestionar els seus, etc. La Dapp aprofita aquest node enllaçat amb la cartera de l'usuari per a poder llegir l'estat actual de la cadena de blocs, i, si l'usuari decideix fer una acció (sigui: comprar, vendre, bescanviar...), la Dapp genera una transacció que envia a la cartera de l'usuari de forma que aquest pugui acceptar-la.

[Nota: L'acció d'"enviar la transacció al wallet" és estrictament necessari, ja que és l'usuari qui ha d'aprovar la transacció. Una vegada aquesta s'aprova, el wallet signa la transacció en nom de l'usuari perquè tothom sàpiga que aquella transacció és de l'adreça del usuari.]

Cal remarcar que, dintre de la informació que llegeix la Dapp, es troben els respectius enllaços al IPFS que contenen informació addicional de l'esdeveniment i/o tiquet seleccionat. Aquest enllaç s'utilitza per buscar al IPFS el fitxer immutable que conté aquesta informació addicional per a poder representar els esdeveniments i els tiquets a la interfície.

Conclusions Provisionals

Al llarg de tot el treball, s'ha pogut demostrar com la combinació de tecnologies blockchain i NFT pot donar una resposta satisfactòria i efectiva als reptes tradicionals del ticketing.

Primerament, durant el desenvolupament del projecte, s'han assolit de forma satisfactòria els objectius bàsics que complien amb el tema d'aquest treball per gestionar transaccions segures d'entrades mitjançant la xarxa d'Ethereum. A més, s'han realitzat múltiples iteracions dintre del smart contract a fi de reduir la mida del bytecode, aplicar optimitzacions per evitar grans despeses de gas (cost

computacional) i desenvolupar noves característiques que enriqueixen aquest projecte com per exemple, el sistema de bescanvi d'entrades per poder garantir l'accés als esdeveniments únicament als propietaris legítims de les entrades NFT o el sistema de revenda segur.

En segon lloc, el desenvolupament de la Dapp ha permès crear una interfície d'usuari que podria descriure com intuïtiva, àgil i capaç de connectar-se de manera segura als diferents wallets disponibles en el mercat, encara que l'enfocament més directe ha sigut amb Metamask. En aquesta part, hi ha hagut moltes decisions de disseny i grans esforços en tractar d'adaptar la Dapp per a qualsevol classe de dispositiu, sigui un ordinador o la pantalla d'un mòbil, per a poder garantir sempre una bona experiència d'usuari des de la compra de les entrades, fins al bescanvi per accedir als esdeveniments. Combinant-ho tot amb el sistema IPFS perquè tota la infraestructura sigui distribuïda i transparent amb els usuaris.

En tercer lloc, voldria destacar els esforços realitzats en fer d'aquest projecte, un sistema segur i resistent a diferents vectors d'atacs com ja es va comentar en el primer informe, i com s'ha continuat millorant i cobrint nous escenaris com la implementació del commit-reveal pel bescanvi d'entrades a fi de protegir a l'usuari en tot moment.

Com a resolució final, aquest treball no només ha demostrat la viabilitat tècnica que hi ha per implementar la tecnologia blockchain i els NFT com una transformació del sector del ticketing i la reveneda d'entrades, sinó que ha anat un pas més, implementat un sistema complet que cobreix des de l'adquisició fins als casos de revenda i bescanvi.

No obstant això, soc conscient que hi ha marge de millora per aprofitar més les característiques dels NFT i deixo com a línia futura la possibilitat de desenvolupar un sistema de revelació diferida dels atributs del NFT. De forma que, una vegada l'usuari hagi bescanviat la seva entrada per l'accés o l'esdeveniment s'hagi acabat, aleatòriament es calculin una sèrie d'atributs per l'entrada NFT com per exemple, la raresa de l'entrada (comú, rara, èpica, i llegendària) inferint un art diferent depenent de la categoria, i donant-li un estat al NFT, el qual podria donar-li un aspecte de desgast. Per exemple, una carta podria tenir l'estat de "desgastada" mostrant per sobre de l'art que hagi tocat un aspecte desgastat, amb algunes taques, colors menys vius, etc. O en contrapart un atribut com "nou" que mostri l'art de l'esdeveniment molt més viu, sense taques, etc. De forma que es podria crear un mercat de revenda d'entrades i generar un cert col·leccionisme al voltant de les entrades ja bescanviades o d'esdeveniments finalitzats.

Per últim i com a altre línia futura, en cas d'un desplegament real, el mes ideal seria poder desplegar el sistema sobre una cadena de blocs que tingui paritat amb una moneda física. D'aquesta forma els preus no fluctuarien tant i seria mes fàcil pels usuaris entendre el preu real de l'entrada. Fent una cerca ràpida he trobat la cadena de blocs "Gnosis", que compleix exactament amb aquestes característiques, ja que la seva moneda té paritat amb el dolar. A més, el temps de minat de cada bloc es de 5 segons, que, comparat als 13 segons per bloc de la mainnet i altres cadenes, pot ser beneficiós, ja que els usuaris han d'esperar menys de la meitat de temps per incloure les seves transaccions a la cadena de blocs, resultant en menys temps d'espera i una millor experiència d'usuari.

Bibliografia

Run a development network | MetaMask developer documentation [en línea], (sin fecha). Home | MetaMask developer documentation. Disponible en: https://docs.metamask.io/wallet/how-to/run-devnet/

Convenience libraries | MetaMask developer documentation [en línea], (sin fecha). Home | MetaMask developer documentation. Disponible en: https://docs.metamask.io/wallet/concepts/convenience-libraries/#:~:text=The%20MetaMask%20Ethereum%20provider%20API,tools'%20documentation%20to%20use%20them

Inicio rápido – React [en línea], (sin fecha). React. Disponible en: https://es.react.dev/learn

JavaScript + Wagmi (recommended) | MetaMask developer documentation [en línea], (sin fecha). Home | MetaMask developer documentation. Disponible en: https://docs.metamask.io/sdk/quickstart/javascript-wagmi/#:~:text=JavaScript%20+%20Wagmi%20

React Native | MetaMask developer documentation [en línea], (sin fecha). Home | MetaMask developer documentation. Disponible en: https://docs.metamask.io/sdk/quickstart/react-native/

Reddy, S., (2023). Build a full stack dApp using React.js [en línea]. Medium. Disponible en: https://medium.com/simform-engineering/build-a-full-stack-dapp-using-react-js-4cd1ff39e544

Viem · TypeScript Interface for Ethereum [en línea], (sin fecha). Viem · TypeScript Interface for Ethereum. Disponible en: https://viem.sh/

Why Wagmi | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/why

Introducción — Vue.js [en línea], (sin fecha). Vue.js. Disponible en: https://es.vuejs.org/v2/guide/

Vue.js [en línea], (sin fecha). Vue.js - The Progressive JavaScript Framework | Vue.js. Disponible en: https://vuejs.org/guide/quick-start.html

Polimi, T., (2022). Blockchain : Day 2—EtherJS + VueJS [en línea]. Medium. Disponible en: https://medium.com/coinmonks/blockchain-day-2-etherjs-vuejs-4a2896a50ef2

web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation [en línea], (sin fecha). web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation. Disponible en: https://web3js.readthedocs.io/en/v1.10.0/

Documentation [en línea], (sin fecha). Disponible en: https://docs.ethers.org/v6/getting-started/

APRENDRE A UTILITZAR REACT AMB WAGMI

Building a user interface for your contract [en línea], (sin fecha). ethereum.org. Disponible en: https://ethereum.org/en/developers/tutorials/creating-a-wagmi-ui-for-your-contract

Getting Started | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/getting-started

Jarrod Watts, (2023). Don't Use Ethers.js [en línea]. YouTube. Disponible en: https://www.youtube.com/watch?v=gUsgFDAY_3w

Ethereum Development Tutorials [en línea], (sin fecha). ethereum.org. Disponible en: https://ethereum.org/en/developers/tutorials/

INFORMACIÓ ADICIONAL

Fees | WalletConnect [en línea], (sin fecha). Home | WalletConnect. Disponible en: https://docs.walletconnect.network/token-dynamics/fees

Known issues and supported browsers | MetaMask Help Center [en línea], (sin fecha). MetaMask Help Center | MetaMask Help Center. Disponible en: https://support.metamask.io/manage-crypto/transactions/metamask-activity/known-issues-and-supported-browsers/

Mustarik, A., (2024). Why You Can’t Use async Functions Directly in useEffect and How to Handle Asynchronous Side Effects… [en línea]. Medium. Disponible en: https://medium.com/@almustarik/why-you-cant-use-async-functions-directly-in-useeffect-and-how-to-handle-asynchronous-side-effects-fbf529242e49

WAGMI HOOKS

useAccount | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useAccount

useConnect | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useConnect

useReconnect | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useReconnect

useDisconnect | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useDisconnect

useSwitchChain | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useSwitchChain

useBalance | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useBalance

useReadContract | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useReadContract

useReadContracts | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useReadContracts

useWriteContract | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useWriteContract

useWaitForTransactionReceipt | Wagmi [en línea], (sin fecha). Wagmi | Reactivity for Ethereum apps. Disponible en: https://wagmi.sh/react/api/hooks/useWaitForTransactionReceipt

DESPLEGAMENT DEL SMART CONTRACT A UNA TESTNET

7. Deploying to a live network | Ethereum development environment for professionals by Nomic Foundation [en línea], (sin fecha). Hardhat | Ethereum development environment for professionals by Nomic Foundation. Disponible en: https://hardhat.org/tutorial/deploying-to-a-live-network

Configuration variables | Ethereum development environment for professionals by Nomic Foundation [en línea], (sin fecha). Hardhat | Ethereum development environment for professionals by Nomic Foundation. [Consultado el 19 de mayo de 2025]. Disponible en: https://hardhat.org/hardhat-runner/docs/guides/configuration-variables

Infura | Metamask [en línea], (sin fecha). Disponible en: https://developer.metamask.io/

Ethereum Sepolia Faucet [en línea], (sin fecha). Cloud Computing Services | Google Cloud. Disponible en: https://cloud.google.com/application/web3/faucet/ethereum/sepolia

Sepolia PoW Faucet [en línea], (sin fecha). Sepolia PoW Faucet. Disponible en: https://sepolia-faucet.pk910.de/

TOKENURI & EVENTURI

ERC721A Documentation [en línea], (sin fecha). Site not found · GitHub Pages. Disponible en: https://chiru-labs.github.io/ERC721A/#/erc721a?id=_baseuri

ERC721A Documentation [en línea], (sin fecha-b). Site not found · GitHub Pages. Disponible en: https://chiru-labs.github.io/ERC721A/#/erc721a?id=tokenuri

Ethereum Improvement Proposals [en línea], (sin fecha). Ethereum Improvement Proposals. Disponible en: https://eips.ethereum.org/EIPS/eip-721

Stack Exchange [en línea], (sin fecha). ERC721 Metadata - TokenURI - Return the full JSON String Instead of URL Pointer?- Stack Exchange. Disponible en: https://ethereum.stackexchange.com/questions/109421/erc721-metadata-tokenuri-return-the-full-json-string-instead-of-url-pointer

Metadata Standards [en línea], (sin fecha). OpenSea Developer Documentation. Disponible en: https://docs.opensea.io/docs/metadata-standards

Yu, B., (2022). How to create NFT Metadata [en línea]. Medium. Disponible en: https://medium.com/@intenex/how-to-create-nft-metadata-a88d4d12dca3

CREACIÓ DE FITXERS PEL IPFS

Pixaby [en línea], (sin fecha). Disponible en: https://pixabay.com/es/

500+ Imágenes sin derechos de autor [HD] | Descargar imágenes gratis en Unsplash [en línea], (sin fecha). Beautiful Free Images & Pictures | Unsplash. Disponible en: https://unsplash.com/es/images/stock/non-copyrighted

AI Image Editor - Pixelcut [en línea], (sin fecha). Pixelcut | Free AI Photo Editor. Disponible en: https://www.pixelcut.ai/ai-image-editor?tool=upscale

Easily compress images at optimal quality in seconds. [en línea], (sin fecha). iLoveIMG | The fastest free web app for easy image modification. Disponible en: https://www.iloveimg.com/compress-image

IMPLEMENTACIÓ DEL IPFS

Get Started | IPFS Docs [en línea], (sin fecha). IPFS Documentation | IPFS Docs. Disponible en: https://docs.ipfs.tech/install/

Best Practices for Storing NFT Data using IPFS | IPFS Docs [en línea], (sin fecha). IPFS Documentation | IPFS Docs. Disponible en: https://docs.ipfs.tech/how-to/best-practices-for-nft-data/#persistence-and-availability

IPFS Desktop Tutorial | IPFS Docs [en línea], (sin fecha). IPFS Documentation | IPFS Docs. Disponible en: https://docs.ipfs.tech/how-to/desktop-app/#install-ipfs-desktop

Public IPFS Utilities | IPFS Docs [en línea], (sin fecha). IPFS Documentation | IPFS Docs. Disponible en: https://docs.ipfs.tech/concepts/public-utilities/#public-ipfs-gateways

Pinata [en línea], (sin fecha). Pinata | Effortless IPFS File Management. Disponible en: https://app.pinata.cloud/ipfs/files

Quickstart - Pinata Docs [en línea], (sin fecha). Quickstart - Pinata Docs. Disponible en: https://docs.pinata.cloud/quickstart

Reddit.com [en línea], (sin fecha). Disponible en:. https://www.reddit.com/r/ipfs/comments/1divwn4/whats_point_of_using_pinata_cloud/https://www.infura.io/blog/post/the-developers-guide-to-file-storage-and-ipfs

OPTIMITZACIONS DEL SMART CONTRACT

Downsizing contracts to fight the contract size limit [en línea], (sin fecha). ethereum.org. Disponible en: https://ethereum.org/en/developers/tutorials/downsizing-contracts-to-fight-the-contract-size-limit/

Custom Errors in Solidity | Solidity Programming Language [en línea], (sin fecha). Solidity Programming Language. Disponible en: https://soliditylang.org/blog/2021/04/21/custom-errors/

A Collection of Gas Optimisation Tricks [en línea], (sin fecha). OpenZeppelin Forum. Disponible en: https://forum.openzeppelin.com/t/a-collection-of-gas-optimisation-tricks/19966/6

ERC721A Documentation [en línea], (sin fecha). Site not found · GitHub Pages. Disponible en: https://chiru-labs.github.io/ERC721A/#/erc721a-queryable

Utilities - OpenZeppelin Docs [en línea], (sin fecha). Documentation - OpenZeppelin Docs. Disponible en: https://docs.openzeppelin.com/contracts/5.x/api/utils#EnumerableSet

Gnosis Chain | Gnosis Chain [en línea], (sin fecha-b). Gnosis Chain | Gnosis Chain. Disponible en: https://docs.gnosischain.com/developers/overview