Universitat de Lleida

TREBALL FINAL DE GRAU

ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiants: Aleix Esteve i Kevin Costes

Titulació: Grau en Tècniques d'Interacció Digital i de Computació

Títol de Treball Final de Grau: GameSave, plataforma web d'allotjament de servidors amb tecnologia blockchain

Director/a: Jordi Mateo Fornés

Presentació

Mes: Setembre

Any: 2023

Agraïments

Ens agradaria agrair, en primer lloc, al nostre tutor Jordi Mateo per la seva dedicació i bona orientació que ens ha ofert.

També voldríem agrair a tot el professorat de GTIDIC per tot el coneixement i inspiració que ens ha transmès i per fer-nos créixer tant personalment com acadèmicament.

Resum

Aquest projecte tracta sobre el procés de desenvolupament d'una plataforma web on l'usuari pot comprar servidors per a videojocs que es despleguen automàticament, integrant la nostra pròpia criptomoneda i on l'usuari paga solament pels recursos que necessita.
El procés inclou investigar sobre la tecnología Blockchain, frameworks, disseny i contenerització, així com el desplegament d'una plataforma totalment funcional i preparada per a les necessitats de l'usuari.

Contingut

Introducció

En l'era digital actual, on la demanda de recursos informàtics escalables i flexibles és imperativa, els serveis d'allotjament de servidors per internet han emergit com a pilars fonamentals per a individus i empreses de totes les mides. Aquests serveis, encapçalats per gegants tecnològics com Amazon Web Services (AWS)[1], Microsoft Azure[2] i Google Cloud[3], han revolucionat la forma en la qual les organitzacions aborden l'allotjament i la gestió de les seves aplicacions, llocs web i càrregues de treball en línia. En proporcionar infraestructura virtualitzada sota demanda, aquests serveis d'allotjament de servidors han eliminat les barreres tradicionals relacionades amb la inversió en maquinari costós i l'administració complexa de servidors físics. Aquests negocis han transformat la indústria tecnològica, permetent una major agilitat, escalabilitat i eficiència operativa per a una varietat de necessitats empresarials. En l'àmbit específic de l'allotjament de servidors de videojocs, plataformes com Nitrado[4] o Gportal[5] han tingut un paper fonamental en l'evolució de la indústria de l'entreteniment digital. Aquestes plataformes han creat un ecosistema on els jugadors i els desenvolupadors poden desplegar i gestionar servidors dedicats per a una àmplia gamma de jocs en línia, des de títols populars fins a creacions independents.

En aquest projecte hem creat una plataforma on els usuaris disposen dels beneficis de qualsevol altra plataforma d'allotjament de servidors de videojocs amb unes millores notables, la integració de criptomonedes com a moneda de canvi i l'opció de pagar pels recursos que utilitzem, aquesta última millora ha estat integrada per altres proveïdors però no en l'àmbit dels servidors de videojocs.

Estat de l’art

En aquest apartat explicarem diferents projectes i com aborden els problemes que GameSave intenta solucionar.

Principalment, parlarem dels competidors, Nitrado i Gportal, les dues plataformes per excel·lència a l'hora de parlar d'allotjament de servidors de videojocs, molts d'anys de funcionaments i que han millorat amb el pas del temps.

El problema que trobem en aquestes, és que no sempre implementen funcions que faciliten l'ús a tots els usuaris; algunes d'aquestes plataformes tenen unes característiques mínimes per als servidors, com per exemple un mínim de dies de funcionament o recursos mínims, molts cops, superiors a les necessitats de l'usuari. En el nostre cas volem permetre a l'usuari tenir major control sobre aquests paràmetres, baixant el límit per a poder adaptar-nos al nostre usuari.

Per altra banda, en aquestes plataformes, sols s'accepten pagaments amb targeta de crèdit. Nosaltres integrem els pagaments amb criptomonedes que no sempre necessiten una targeta de crèdit per a poder obtenir-les.

Objectius

Objectius Generals

Els objectius principals que volíem assolir desenvolupant aquest projecte són els següents:

- Dissenyar, desenvolupar i implementar una plataforma web totalment funcional.
- Crear un lloc web on els usuaris poden crear i configurar els seus servidors d'una manera fàcil i intuïtiva.
- Disposar de característiques úniques per a diferenciar-nos de la competència, integrant la tecnologia blockchain i el concepte "pay as you go" [6].

Objectius d'Aleix

Aleix Esteve implementa la creació dels servidors en Docker.

- Integrar la creació automàtica de servidors per a videojocs amb Docker.
- Integrar la interacció de cada tipus de servidor des de la plataforma web.
- Desenvolupar el backend de la web.

Objectius de Kevin

Kevin Costes integra la tecnologia blockchain en el projecte.

- Integrar la tecnologia blockchain a la nostra plataforma per a oferir més flexibilitat i seguretat amb les transaccions.
- Crear una criptomoneda o un token per a ser utilitzada com a mètode de pagament en la nostra plataforma.
- Integrar la tecnologia blockchain a la plataforma per a processar les transaccions i facilitar l'ús.

Degut a la complexitat d'aquest projecte que aborda moltes tecnologies noves i diferents no vistes en el GTIDIC, hem decidit plantejar-ho de forma que poguéssim ajuntar aquests objectius en un mateix projecte.

Metodología

A continuació detallarem les definicions de les tecnologies utilitzades en el projecte:

- **Aplicació web:** Una aplicació web és un programari que s'executa en un navegador web i ofereix funcionalitats i serveis a l'usuari a través d'Internet.
- **Lloc web:** Un lloc web és una col·lecció de pàgines web relacionades que es poden accedir mitjançant un navegador web. Conté informació, contingut multimèdia i enllaços.
- **Interfície d'usuari:** La interfície d'usuari és el punt de contacte entre l'usuari i una aplicació informàtica. Inclou elements visuals i interactius que permeten a l'usuari comunicar-se amb el sistema.
- **Front-end**: El front-end és la part visible d'una aplicació o lloc web que l'usuari final pot veure i interactuar. Inclou elements com el disseny, la presentació i la interacció.
- **Back-end:** El back-end és la part d'una aplicació o lloc web que funciona a nivell del servidor i gestiona les funcionalitats com la base de dades, la lògica empresarial i les operacions del sistema.
- **Contenerització:** La contenerització és una tecnologia de virtualització que reuneix aplicacions i les llibreries i arxius necessaris per al funcionament de l'aplicació en unitats aïllades, anomenades contenidors, perquè funcionin sense problemes en qualsevol plataforma.

A continuació coneixerem les eines necessàries per al projecte:

- **PyCharm:** PyCharm [7] és un entorn de desenvolupament integrat (IDE) específic per a programació en Python. Proporciona eines avançades per a l'escriptura de codi, la depuració i el desenvolupament d'aplicacions Python.
- **Remix Ethereum IDE:** és un entorn de desenvolupament integrat en línia (IDE) dissenyat específicament per a la programació de contractes intel·ligents en la xarxa Ethereum [8].
- **Docker:** Docker és una plataforma de virtualització de contenidors que permet empaquetar i distribuir aplicacions i les seves depèndencies en entorns aïllats anomenats contenidors, que són portables i lleugers [9].
- **Imatge:** En el context de Docker, una imatge és un paquet autocontingut i lleuger que inclou el codi, les dependències i la configuració necessària per a executar una aplicació en un contenidor. Les imatges de Docker es poden utilitzar per crear i executar contenidors d'una manera consistent i reproduïble.
- **Metamask:** MetaMask és una extensió de navegador que permet als usuaris accedir a la xarxa Ethereum i interactuar amb aplicacions descentralitzades (dApps) i carteres de criptomonedes mitjançant el navegador [10].

Tot seguit detallarem els frameworks i llenguatges utilitzats en el nostre projecte:

- **Django:** Django és un framework de desenvolupament web de codi obert escrit en Python. Proporciona eines i components per a la creació ràpida i eficient d'aplicacions web complexes [11].
- **Bootstrap:** Bootstrap és un framework de disseny i desenvolupament front-end que ofereix un conjunt de llibreries i estils predefinits per a crear llocs web responsius i ben dissenyats [12].

- **Python:** Python és un llenguatge de programació d'alt nivell que es caracteritza per la seva llegibilitat i sintaxi clara. És àmpliament utilitzat en desenvolupament web, científic i d'automatització.
- **HTML5:** HTML5 és la cinquena versió del llenguatge d'etiquetes hipertext Markup Language (HTML). S'utilitza per estructurar el contingut d'una pàgina web i afegir elements interactius com àudio, vídeo i formularis.
- **CSS:** CSS (Cascading Style Sheets) és un llenguatge utilitzat per definir l'aparença visual i el format d'un document HTML o XML. S'utilitza per aplicar estils com colors, tipografia i mides.
- **Javascript:** Javascript és un llenguatge de programació utilitzat principalment per afegir interactivitat i funcionalitat a les pàgines web. S'executa al navegador de l'usuari.
- **jQuery:** jQuery és una biblioteca de JavaScript que simplifica la manipulació del document HTML, l'ajuda amb animacions, la interacció amb el servidor i altres tasques comunes en desenvolupament web.

Finalment, algunes definicions sobre la tecnologia blockchain que hem integrat en el nostre projecte:

- **Blockchain:** La blockchain és una estructura de dades distribuïda i immutable que emmagatzema registres en blocs connectats, assegurant la seguretat i la integritat de la informació.
- **Criptomoneda:** Una criptomoneda és una moneda digital segura i descentralitzada que utilitza la criptografia per a garantir transaccions segures i controlar la creació de noves unitats.
- **Token:** En el context de la tecnologia blockchain, un token és una unitat d'actiu digital que pot representar valor, drets o utilitat en una xarxa específica.
- **Testnet:** Una testnet és una xarxa blockchain de proves utilitzada pels desenvolupadors per a experimentar i provar noves funcionalitats sense gastar criptomonedes reals.
- **Ethereum:** Ethereum és una plataforma blockchain que permet la creació de contractes intel·ligents i aplicacions descentralitzades. Utilitza la criptomoneda Ether (ETH) com a mitjà d'intercanvi [13].
- **SmartContract:** Un smart contract (contracte intel·ligent) és un protocol informàtic autoexecutant que facilita, verifica o fa complir la negociació o l'execució d'un acord en una blockchain.
- **Solidity:** Solidity és un llenguatge de programació de contractes intel·ligents dissenyat específicament per a la plataforma Ethereum [14].
- **Gas:** en el context de blockchain fa referència a la unitat de mesura utilitzada per quantificar el cost computacional necessari per executar operacions i transaccions en una cadena de blocs. És una forma de mesurar l'esforç que es requereix per acomplir tasques a la xarxa blockchain, com processar transaccions o executar contractes intel·ligents.

Framework

Per a desenvolupar aquest projecte utilitzarem un framework anomenat Django.

Django és un marc de desenvolupament web de codi obert escrit en Python. Està dissenyat per a simplificar i accelerar el procés de creació d'aplicacions web complexes i dinàmiques. El funcionament de Django es basa en els següents components clau:

- **Models**: Django utilitza models per a definir les estructures de dades de l'aplicació. Aquests models s'assemblen a les taules d'una base de dades, però Django crea les taules i els camps automàticament.
- **Vistes:** Les vistes de Django gestionen les sol·licituds HTTP i determini quina informació s'ha de mostrar a l'usuari. Les vistes poden generar contingut HTML o altres formats de sortida.
- **Plantilles:** Django fa servir plantilles per a generar contingut dinàmic. Aquestes plantilles permeten mesclar codi Python amb HTML per a crear pàgines web dinàmiques amb facilitat.
- **Controladors:** Les vistes actuen com a controladors, gestionant la interacció entre les dades i les plantilles. Determinen quina informació es recull, com s'ha de processar i com s'ha de presentar.
- **URLs:** Les URLconf (configuracions d'URL) de Django mapegen les URL introduïdes pel client a les vistes apropiades. Això permet una navegació coherent i intuïtiva.
- **Gestió de l'administració:** Django proporciona una interfície d'administració generada automàticament per gestionar les dades dels models de manera fàcil i segura.
- **ORM (Object-Relational Mapping):** Django utilitza un ORM per a interactuar amb la base de dades. Això permet als desenvolupadors tractar les dades com a objectes de Python, fent que les interaccions amb la base de dades siguin més naturals i menys propenses a errors.

En resum, Django simplifica el desenvolupament web mitjançant l'ús de patrons i components ben establerts. Els desenvolupadors poden centrar-se en la lògica de l'aplicació i la presentació, mentre Django gestiona molts aspectes tècnics fonamentals. Aquest enfocament ajuda a crear aplicacions web robustes i eficients de manera eficaç.

Al fer servir Python utilitzem un entorn virtual on instal·lem només els paquets necessaris els quals podem trobar en requirements.txt:

- **django:** El framework utilitzat.
- **celery:** Per a programar tasques.
- **docker:** Per a poder interactuar amb la plataforma de docker.
- **web3:** Per a poder interactuar amb el nostre smart contract.
- **mcstatus:** Per a poder realitzar consultes directament als servidors de minecraft.

Containerització i virtualització de MV

Per a fer realitat aquest projecte, vam haber de prendre la decisió d'utilitzar containerització o màquines virtuals (MV), però abans d'entrar en matèria cal parlar sobre la virtualització. La virtualització consisteix en un software o conjunt de software per a simular un ordinador dins d'un ordinador.

	Contenidors	**Màquines virtuals**
Aïllament	Comparteixen el mateix sistema operatiu del host però estan aïllats a nivell de procés. Això significa que són més ràpids i lleugers que les MV.	Emulen un hardware complet i també el sistema operatiu. Això proporciona un major nivell d'aïllament, ja que cada MV té el seu propi sistema operatiu i recursos dedicats. Però això fa que siguin més lentes i pesades que els contenidors.
Eficiència de recursos	Gràcies a la seva arquitectura compartida, els contenidors són més eficients amb l'ús de recursos.	Aquestes requereixen més recursos, ja que cada MV té el seu sistema operatiu complet, la qual cosa comporta un major ús de RAM i emmagatzematge.
Temps d'arrencada	Els contenidors s'inicien i s'aturen ràpidament, cosa que els fa idonis per a la nostra aplicació.	Les MV solen trigar més a arrancar a causa de l'emulació del hardware i haver de posar en marxa un sistema operatiu complet.
Portabilitat	Aquests són altament portàtils perquè al incloure totes les dependències de l'aplicació, es poden executar en qualsevol host que sigui compatible amb, en el nostre cas, Docker.	Són menys portàtils que els contenidors, ja que depenen del software i arquitectura de virtualització específica i poden requerir conversions o configuracions addicionals per a canviar entre diferents plataformes.
Seguretat	Els contenidors comparteixen el mateix kernel del sistema operatiu del host, cosa que pot comportar riscos de seguretat si un contenidor és compromès.	Degut al superior nivell d'aïllament, les màquines virtuals ofereixen un millor grau de seguretat enfront dels contenidors. Cada MV té el seu propi sistema operatiu i kernel, la qual cosa redueix les possibles zones d'atac.

En vista d'aquesta comparativa vam decidir que la solució que més s'adapta al nostre projecte és la containerització.

Planning i Implementació

Per al desenvolupament d'aquesta plataforma, hem hagut de planificar alguns punts i realitzar les diferents implementacions necessàries per a assolir els objectius.

Implementació Global

Branding

El primer pas ha estat fer el branding del nostre projecte, donar-li una imatge d'acord amb el seu objectiu. Per a això hem fet servir els nostres coneixements de disseny per a aconseguir els logotips i noms necessaris.

Tot això ho vam fer seguint una paleta de colors prèviament definida, que farem servir durant tot el projecte.

Vam decidir nomenar la nostra pàgina web GameSave.

Figura 1: Logotip principal

També vam dissenyar un favicon:

Figura 2: Favicon

I finalment, una imatge per a representar el nostre token, anomenat HostCoin:

Figura 3: Logotip HostCoin

Base de dades

Per a un projecte d'aquesta mida necessitem una base de dades molt ben dissenyada, per això vam fer un diagrama de classes UML (Unified Modeling Language).

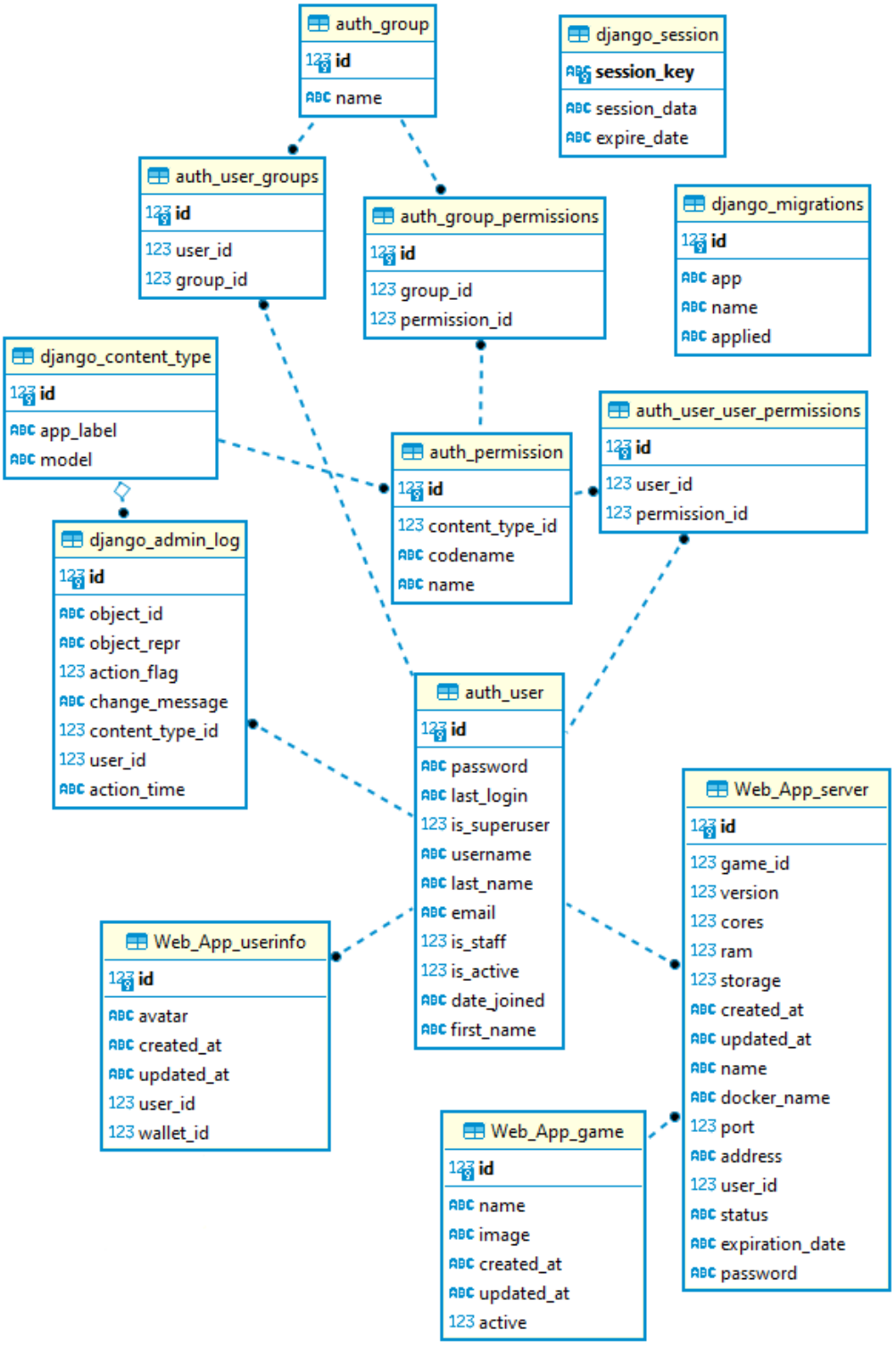


Figura 4: Diagrama de classes UML

Aquesta base de dades, que ha canviat vàries vegades durant el desenvolupament del projecte, disposa de 14 taules.

Moltes d'aquestes taules són necessàries per al framework de Django o per a l'autenticació.

A continuació explicarem les taules necessàries per al projecte:

- **Web_App_userinfo:** Aquesta taula ens serveix per a emmagatzemar informació addicional sobre l'usuari, ja que en **auth_user** no disposem de tots els camps necessaris.
- **Web_App_server:** Aquesta taula fa referència a cada servidor creat, emmagatzema dades com les seves característiques, informació de Docker i a quin usuari pertany.
- **Web_App_game:** Aquesta taula guarda informació sobre els videojocs disponibles per a crear un servidor. També tenim la imatge de Docker que fa servir.

Landing page

En aquest apartat parlarem de la landing page, la pàgina web que un usuari nou veurà el primer cop que vulgui fer servir la nostra plataforma.

Figura 5: Landing Page

Vam optar per un disseny simple i modern, amb uns colors vius que recorden a la majoria dels videojocs. Ja que en aquesta pàgina solament mostrem informació important, vam pensar que la millor era fer un OnePage.

Podem trobar diferents seccions:

- **Top bar/Menú:** Menú superior per a navegar la nostra pàgina web i registrar-se o iniciar sessió i amb el nostre logotip.
- **Benvinguda:** El primer que veu l'usuari, amb una benvinguda interessant.
- **Serveis:** Definició dels serveis que oferim.
- **Jocs:** Llistat dels jocs que actualment suportem.

Figura 6: Llistat dels videojocs suportats

- **Com funciona?:** Explica el funcionament de la nostra pàgina web.
- **Equip:** En aquest apartat els usuaris poden veure imatges i rols de l'equip.

Tauler de control

Dashboard

El tauler de control és la part on els usuaris registrats passen la major part del temps. El primer que veuran és el dashboard, on poden veure ràpidament el seu balanç, els servidors dels quals disposen i el seu estat depenent del seu color (Funcionant o apagat).

També poden anar a la seva cartera, crear un nou servidor o tancar la sessió.

Per a cada servidor podem veure el seu estat amb el color del requadre d'una manera molt intuïtiva. També podem eliminar-lo o entrar a la configuració del servidor.

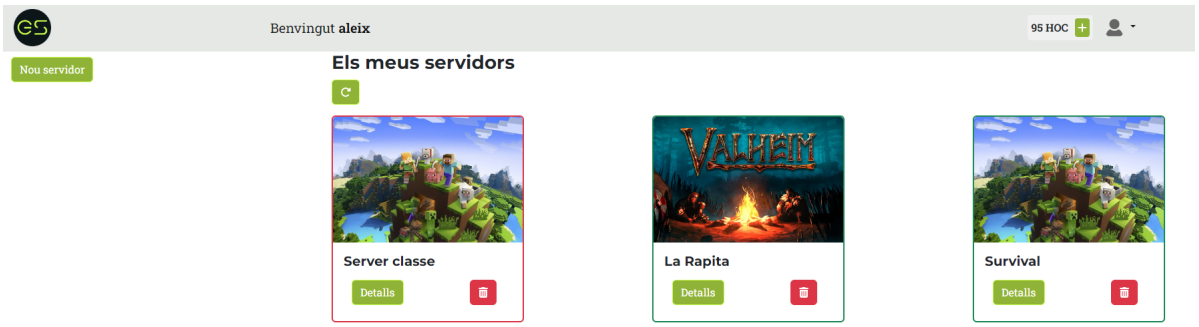


Figura 7: Dashboard

Creació del servidor

La creació del servidor és un procés ben simple. Disposem d'un formulari on podem escollir el nom i els recursos que necessitem per al nostre servidor, també podem escollir el joc.

A la part dreta podem veure el preu diari en HostCoin del nostre servidor i el nostre balanç.

També disposem de missatges d'èxit i d'error controlats per la blockchain.

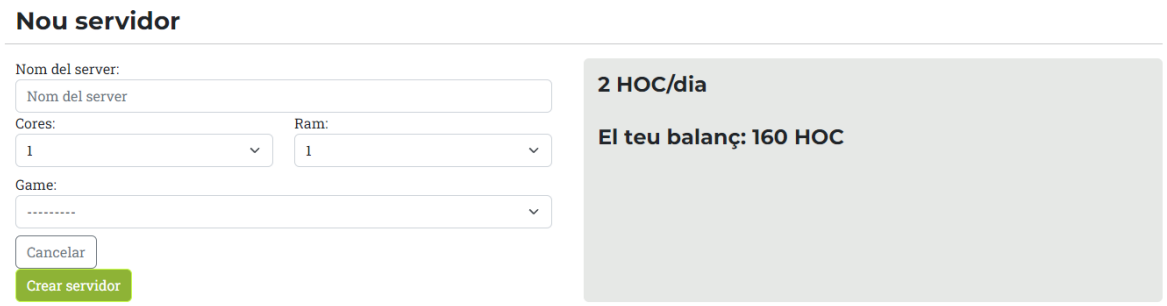


Figura 8: Formulari de creació de servidor

Informació i configuració d'un servidor

En aquesta pàgina podem veure tota la informació i configuració d'un servidor. Disposem de gdos elements estàtics, el panell lateral, que ensenya les dades del servidor com, la seva direcció i l'estat, i els botons superiors per a controlar el servidor, podem iniciar, aturar, reiniciar o esborrar.

A la part esquerra tenim un menú, amb unes opcions diferents depenent del videojoc. En la figura 9 podem observar una pestanya d'informació, la consola, la configuració i l'ftp.

Figura 9: Informació d'un servidor

Cartera

La cartera és una de les pàgines més importants de la nostra plataforma, si l'usuari té una cartera Metamask, necessària per a operar la pàgina, podrà revisar els seus balanços d'Ethereum i de HostCoins.

També podrà comprar revisar la seva direcció de cartera i comprar HostCoins.

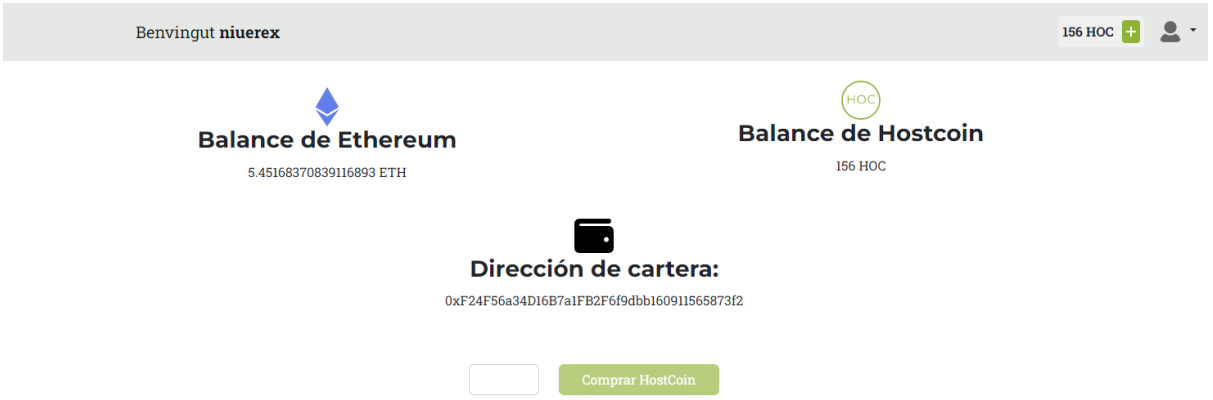


Figura 10: Cartera d'un usuari

Implementació Aleix

La meva finalitat en el treball és integrar la creació automàtica de servidors de videojocs en contenidors de Docker i desenvolupar diferents característiques per a interaccionar amb els servidors a través de la pròpia pàgina web.

La creació d'un servidor ha de ser el més fàcil possible per a l'usuari, per això vam investigar quina és la forma més fàcil per a crear servidors a la part que òptima i robusta. Podíem containeritzar o crear una màquina virtual per a cada servidor, però això malgasta molts recursos de la màquina real, per tant, vam decidir containeritzar.
Per a fer-ho teníem dues plataformes: Kubernetes [15] i Docker. Finalment, vam decidir utilitzar Docker, ja que reuneix les tres característiques que buscàvem, és fàcil d'utilitzar, òptim amb els recursos i robust amb els processos.

Creació dels servidors

Per a poder crear un servidor necessitem una imatge de cada tipus de servidor, és a dir, per a cada joc. Després que l'usuari hagi realitzat la transacció de la nostra criptomoneda des de la pàgina web, el backend farà la resta.

La comunicació entre el backend i el Docker la realitzem a mitjançant un mòdul de Python, d'aquesta manera podem crear el contenidor amb la imatge pertinent directament. També li apliquem les configuracions necessàries per a poder accedir posteriorment a les diferents configuracions des de la pàgina web. Gràcies al mòdul de Docker per a Python, de la mateixa manera que crearíem un contenidor amb comandes, ho fem mitjançant llenguatge Python.

Aquí podem veure els paràmetres que necessitem per a crear un contenidor:

```
client.containers.run('itzg/minecraft-server',
name="Minecraft Server",
mem_limit="4g",
cpuset_cpus="0,1",
ports="25565:25565",
environment={
...
'VERSION': 'latest',
...
},
detach=True,)
```

- La primera línia és on indiquem la imatge que té tot el necessari perquè Docker pugui crear el contenidor i que el servidor finalment funcioni correctament.
- **"name":** És el nom amb el qual podrem identificar el contenidor quan estigui creat.
- **"mem_limit":** És el límit de memòria RAM que tindrà el contenidor.
- **"cpuset_cpus":** Són els nuclis de la CPU dels quals disposarà el contenidor per a poder funcionar.

- **“ports”:** Aquí s’indiquen els ports que necessitarà tenir oberts el contenidor així com a quin port de la màquina real correspondran per a posteriorment poder-nos connectar. Cal dir que cada tipus de servidor de jocs necessita un o més ports específics per a funcionar.
- **“environment”:** En aquest paràmetre s’indiquen diferents variables específiques per a cada tipus de servidor, com ara per exemple la versió del servidor de Minecraft a utilitzar.
- **“detatch”:** Aquí li indiquem que volem que aquest contenidor s’executi de fons, perquè puguem seguir interactuant amb el Docker.

Aquí podem veure el recorregut que farà la petició de creació d’un servidor des de que l’usuari emplena el formulari de creació fins que el contenidor ja està funcionant.

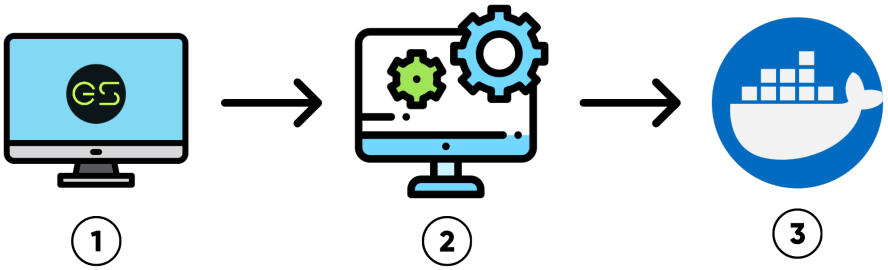


Figura 11: Diagrama del recorregut de la petició de creació d’un contenidor

1. L’usuari selecciona el videojoc del qual es vol crear un servidor, selecciona la CPU i RAM que necessita i efectua el primer pagament per a 1 dia de duració del servei.
2. El backend processa la sol·licitud de l’usuari i configura la petició necessària depenent del servidor a crear i l’envia al Docker.
3. Docker processa la comanda enviada i crea un contenidor amb la imatge necessària, els volums necessaris i el contenidor ja està funcionant i el servidor preparat per accedir-hi.

Quan al docker li arriba la petició de crear un servidor el que passa és el següent:

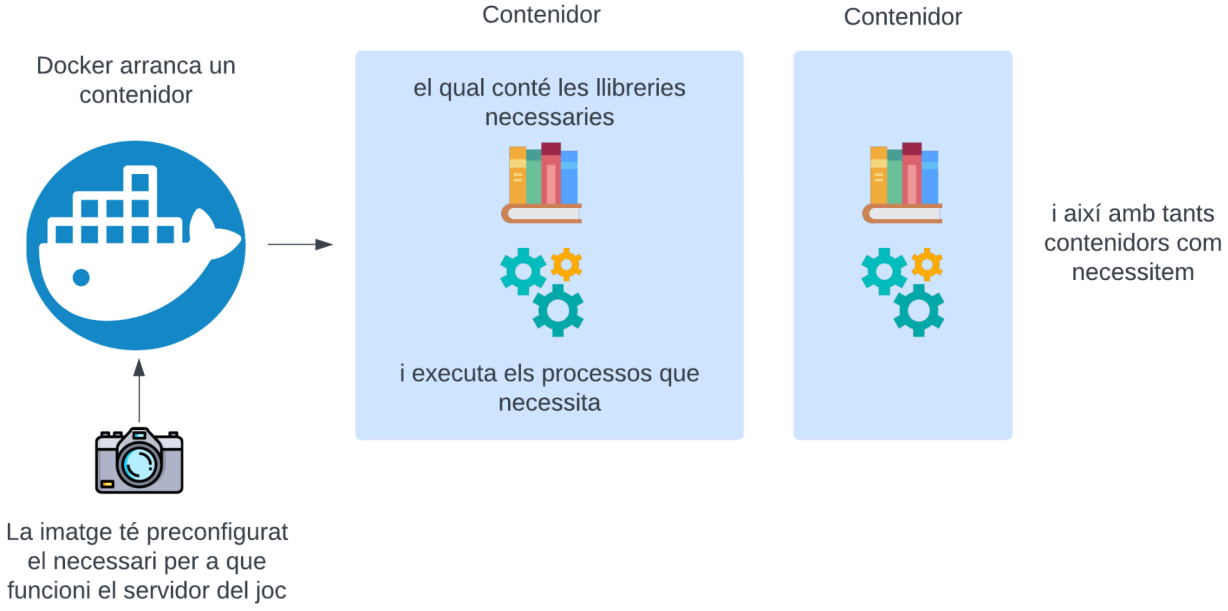


Figura 12: Diagrama d’arranc d’un nou contenidor

Com hem dit, necessitem diferents imatges les quals farem servir unes creades per la comunitat, ja que d'aquesta manera podem interaccionar de forma molt més còmoda amb els servidors des del backend.
En el cas del servidor del joc Minecraft, la imatge que hem utilitzat és "itzg/minecraft-server", la qual té molts de scripts que ens facilitaran la feina.

Aquí podem veure com estan creats els contenidors amb la seva respectiva imatge:

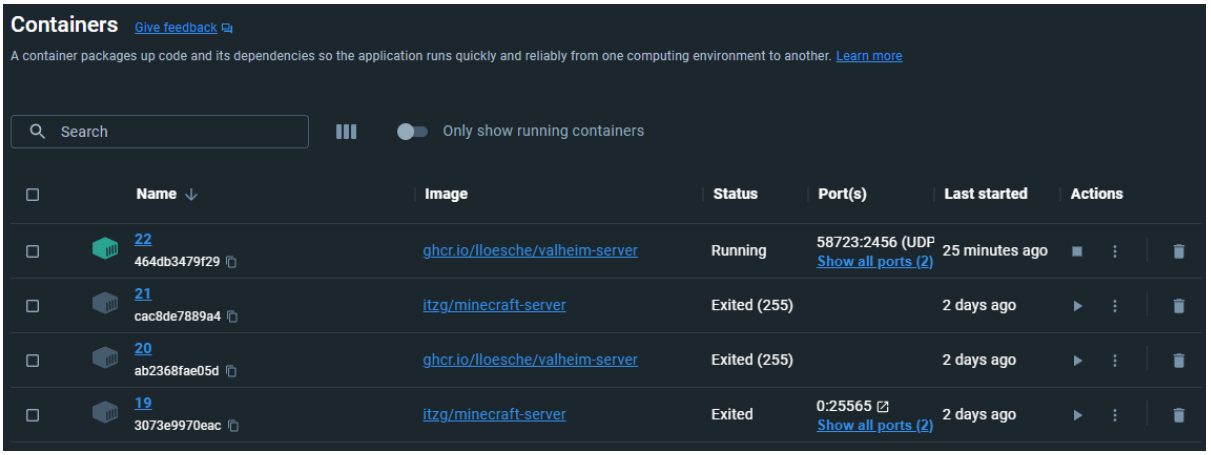


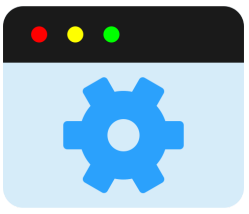
Figura 13: Interfície de Docker Desktop amb la llista de contenidors.

A través de la pàgina web l'usuari ja podrà accedir als seus servidors on hi ha la informació necessària per a connectar-se depenent del tipus de servidor.

Interactuar amb els contenidors

Després de tenir el contenidor amb el servidor funcionant, l'usuari ha de poder configurar el servidor al seu gust, per tant, s'ha de desenvolupar la part necessària per a modificar la configuració.
També ha de poder veure que està passant al servidor en tot moment, per tant, ha de poder veure la consola del servidor, així com enviar comandes al servidor, si aquest és compatible amb comandes, ja que hi ha servidors que no admeten comandes.
Com és d'esperar també ha de poder iniciar, parar o reiniciar els seus servidors, així com esborrar-los en qualsevol moment.

A més a més, l'usuari ha de poder ampliar els dies de servei del servidor en tot moment.



En tots els apartats esmentats anteriorment, qui hi té més pes és tot el que fem amb Django per la part del backend, i, d'altra banda, el Docker el qual és qui gestionarà els contenidors i administrarà els recursos reals de la màquina per a repartir-los entre els diferents

contenidors depenent de la CPU i memòria sol·licitada així com la disponible en cada moment.

Implementació Kevin

La meva finalitat en el treball és integrar la tecnologia blockchain, per a fer això hem hagut d'investigar en aquest àmbit.

L'usuari ha de ser capaç de comprar el nostre token propi oferint els avantatges que aquesta tecnologia comporta. També hem de fer que la nostra plataforma respongui adequadament a les transaccions de tokens, si es fa una transacció ha d'haver-hi una acció. Per últim, facilitar la compra dels nostres tokens als nostres usuaris per a una experiència més satisfactòria.

Primer objectiu: Creació de la criptomoneda

Gràcies a diverses setmanes d'investigació sobre aquesta tecnologia coneixem el procés a seguir i he après molt sobre el funcionament de les blockchains.

Havíem d'escollir amb quina blockchain treballaríem i ens vam decantar per la d'Ethereum, una de les més famoses i que disposa de testnet, una rèplica de la blockchain original per a fer proves sense gastar criptomonedes reals.

El següent pas era crear el Smart Contract, per a això havíem de dissenyar la criptomoneda. Havíem de crear un token el més senzill possible però amb totes les funcionalitats necessàries. També havíem d'aprendre el llenguatge Solidity per a poder programar el contracte.

Les funcions del contracte havien de ser:

- **Comprar token:** L'usuari ha de poder comprar una quantitat n dels nostres tokens amb Ethereum i rebre'ls a la seva cartera.
- **Gastar token:** En aquest mètode es fa la transacció de tokens de l'usuari al contracte.
- **Retirar balanç:** El contracte guarda els Ethereums amb els quals els usuaris han comprat tokens, s'han de poder retirar a un compte personal.

Segon objectiu: Integració en la plataforma

Un cop tenim el token creat, hem de preparar la pàgina web perquè funcioni amb aquests. Després d'investigar vam descobrir que la millor opció era fer servir Metamask, una extensió de navegador web que permet tenir una cartera de criptomonedes i que es pot integrar fàcilment, a banda, una de les més reconegudes en tota la web.

Hem de connectar la nostra cartera Metamask i ser capaç d'intercanviar el nostre token per Ethereum. Per a això crearem una pàgina específica on l'usuari pot visualitzar la seva cartera i els seus balanços, tant d'Ethereum com del nostre token.

Un cop l'usuari és capaç d'aconseguir el nostre token, hem de fer que al comprar un servidor, l'usuari hagi de fer un pagament amb tokens a la plataforma web, si la transacció és exitosa, el servidor de l'usuari es crearà. Per a fer això necessitem trobar una manera de contactar amb la blockchain, ho podem fer amb Python o amb Javascript.

Hem de fer que l'usuari pagui pels recursos que necessita i quan ho necessita, per tant, hem de calcular el preu amb tokens de cada servidor personalitzat al moment de la seva creació. També hem de permetre a l'usuari comprar més temps de funcionament per al seu servidor, per exemple, un usuari pot pagar 1 setmana de temps de funcionament del servidor si ho necessita, però també pot comprar solament 1 dia.

HostCoin

En aquest apartat parlarem de la creació del nostre token HostCoin. Per a fer això, primerament explicarem la diferència entre una criptomoneda i un token.

- **Criptomoneda:** Una criptomoneda és una moneda digital independent que opera en una xarxa blockchain pròpia. Exemples coneguts són Bitcoin i Ethereum. Les criptomonedes són utilitzades principalment com a mitjà d'intercanvi i valor, i tenen la seva pròpia cadena de blocs i protocol de consens.
- **Token:** Un token és una unitat d'actiu digital que es basa en una xarxa blockchain existent, com Ethereum, en el nostre cas. Els tokens no són monedes independents; en canvi, són creats i gestionats utilitzant la infraestructura d'una altra cadena de blocs. Els tokens poden representar diversos propòsits, com ara drets d'accés, utilitat en una plataforma o participació en projectes específics.

Un exemple més clar i proper a molts dels usuaris és: una criptomoneda té un valor equitatiu per a tot el món, per exemple el dòlar o l'euro. En canvi, un token té un valor en algun lloc específic, per exemple punts de fidelitat en alguna pàgina web o botiga física, que podem intercanviar per algun servei o bé material.

Però, per què fer servir una moneda virtual en lloc d'una convencional? Les criptomonedes tenen molts de beneficis que s'adapten a les nostres necessitats:

- **Eficiència:** Els pagaments amb tokens poden ser més eficients en comparació amb les monedes tradicionals, ja que les transaccions poden ser processades de manera més ràpida i a menor cost.
- **Transparència:** Les transaccions amb tokens es registren en la blockchain de manera permanent i transparent, la qual cosa permet a les parts implicades rastrejar i verificar les transaccions amb facilitat.
- **Accessibilitat Global:** Els tokens poden ser utilitzats globalment, sense restriccions geogràfiques. Això és especialment útil per a pagaments internacionals.

- **Baixes Taxes:** Les taxes de transacció per a pagaments amb tokens solen ser més baixes en comparació amb els sistemes de pagament tradicionals.
- **Seguretat:** Les transaccions a la blockchain estan protegides mitjançant criptografia i són immutables, la qual cosa fa que sigui difícil alterar o falsificar les dades.
- **Descentralització:** Les blockchains descentralitzades no depenen d'una única entitat de control. Això pot reduir el risc de manipulació i proporcionar major autonomia als usuaris.
- **Reducció de la Intermediació:** L'ús de blockchain pot reduir la necessitat d'intermediaris en les transaccions, cosa que pot minimitzar els costos i el temps associats.

Sabent això vam decidir que la creació d'un token era molt més viable i s'adaptava més a la nostra causa.

Per a crear una criptomoneda necessitàvem una Blockchain i ens vam decantar per Ethereum, una de les més famoses i amb més documentació que altres. En aquest cas farem servir la seva Testnet, una rèplica de la blockchain real per a poder fer proves i no gastar recursos fent proves. En el nostre cas fem servir la Testnet d'Ethereum Sepolia, una de les més conegudes i utilitzades.

En el nostre cas vam fer servir Alchemy [16], una plataforma que proporciona una API per a interactuar amb la xarxa d'Ethereum i les seves testnets. Ens va facilitar codi i fer comprovacions. També ajuda a connectar una cartera a una testnet.

Un cop tenim la testnet preparada és moment de crear el nostre SmartContract, el token en si. En aquest contracte hem de programar el funcionament del token i les seves funcions. Aquests contractes no poden canviar, un cop estan a la blockchain són immutables, tot i això, podem implementar funcions per a canviar alguns valors importants.

Per a programar aquest contracte fem servir Remix Ethereum IDE una ferramenta en línia que ens permet programar, provar i desplegar contractes amb el llenguatge Solidity.

Un SmartContract, un cop creat, té una adreça a dintre de la blockchain, que farem servir per a accedir a totes les seves funcions.

En el nostre contracte disposem de les següents funcions:

- **constructor(uint _tokenPrice):** Aquesta funció s'executa quan creem el token i té dues funcions molt importants, la primera: s'encarrega d'establir el preu del nostre token, en el nostre cas 1 HostCoin té un valor de 0.0001 Ethereum, que equival aproximadament a 0,16 (Hem de tenir en compte que aquest valor canvia depenent del valor d'Ethereum, per això és interessant tenir una funció per a canviar aquest valor). La segona funció és emmagatzemar el propietari del contracte.
- **function buyTokens(uint _amount):** Aquesta funció serveix per a fer l'intercanvi de Ethereums per HostCoins, multiplica la quantitat que volem comprar pel preu definit prèviament. Si l'usuari compleix amb el pagament sense cap error, li envia els HostCoins a la seva cartera.

- **function spendTokens(uint _amount):** Aquesta funció serveix per a fer pagaments amb HostCoins, si l'usuari vol comprar un servidor, aquesta funció s'encarrega de cobrar-li els HostCoins i avisar si la transacció ha estat exitosa o no.
- **function withdrawFunds():** En el cas del SmartContract de HostCoin, els ethereums amb què els usuaris compren HostCoins, s'emmagatzemen en el maitex contracte fins que es crida aquesta funció, que envia els Ethereum recollits a l'adreça del propietari.

Aquest contracte ha passat per moltes versions diferents, fent proves i corregint errors, fins a arribar a una versió definitiva i funcional. Tot i això, hem de tenir en compte que si hem de fer canvis, necessitarem un contracte nou, i tots els usuaris amb HostCoins en aquest contracte, perdran el seu balanç.

Finalment, amb el mateix Remix, podem compilar el script per a comprovar si hi ha algun error, i desplegar-lo a la testnet.

Integració de pagaments amb criptomonedes

Ja tenim el nostre token creat, ara sols falta que la nostra plataforma web accepti pagaments amb tokens. En aquest apartat explicarem com hem adaptat el nostre projecte per a funcionar amb HostCoin i com hem facilitat als usuaris la compra d'aquests.

El primer pas és connectar amb la cartera de l'usuari, una de les opcions més simple per a l'usuari com per a nosaltres és integrar una cartera que funcioni en el mateix navegador web de l'usuari. En el nostre cas requerim a l'usuari tenir una cartera Metamask per a poder fer servir el nostre lloc web.

Un cop l'usuari entra al nostre tauler de control, Metamask obrirà una finestra emergent per a iniciar sessió. L'usuari igual pot fer servir la pàgina web, però si ha de fer algun pagament serà necessari tenir la cartera.

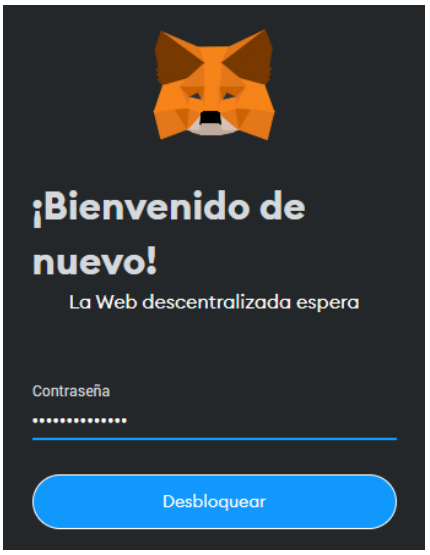


Figura 14: Petició d'accés a Metamask

Si l'usuari disposa d'una cartera Metamask i una quantitat d'Ethereum, es pot dirigir a la seva cartera (Vegeu Figura 10). En la cartera, l'usuari pot veure la seva direcció, el seu balanç d'Ethereum i de HostCoin, i també disposa d'un apartat per a comprar HostCoins. Si l'usuari decideix comprar una quantitat n de HostCoin, una finestra emergent de Metamask s'obrirà perquè l'usuari accepti la transacció.

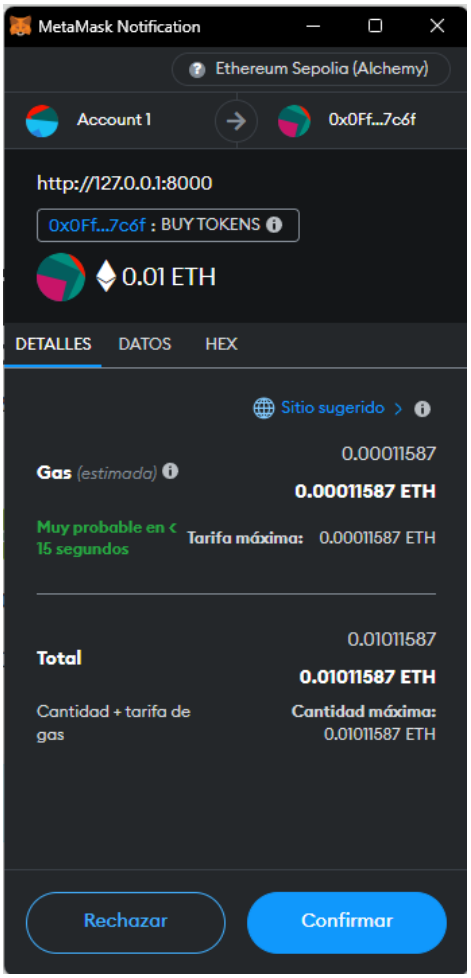


Figura 15: Petició de transacció de Metamask

Si la transacció és correcta, la quantitat n de HostCoins serà afegida a la cartera de l'usuari.

Per a fer això, la pàgina web ha de contactar amb la cartera de l'usuari i amb el contracte per a validar les transaccions i per a aconseguir els balanços de l'usuari. En el nostre cas teníem l'opció de fer-ho amb Python o amb Javascript. En el nostre cas vam decidir fer-ho amb Javascript.

Per a connectar i interactuar amb la cartera i la blockchain, farem servir Web3 [17], una biblioteca JavaScript que permet la comunicació amb xarxes de blockchain, especialment la xarxa Ethereum. Aquesta biblioteca facilita la interacció amb contractes intel·ligents, adreces de carteres i altres elements de la xarxa Ethereum mitjançant codi JavaScript. També disposem d'altres opcions com per exemple la mateixa biblioteca d'Alchemy, mencionada prèviament.

Aquest procés és delicat i els passos a seguir han d'estar molt marcats i complir-se a la perfecció perquè l'usuari tingui una experiència satisfactòria i no perdre informació o tokens:

- **Cartera de l'usuari:** si l'usuari té Metamask sol·licitem accés a la seva cartera (si encara no el tenim), i guardem la seva direcció de cartera. Si l'usuari no te Metamask se li demanarà que l'instal·li per a poder fer servir la nostra plataforma.
- **Recuperar el seu balanç:** Crearem una instància del contracte amb Web3. Aquesta ja disposa d'un mètode per a recuperar els balanços si li passem l'adreça d'una cartera, que hem guardat en el primer pas.
- **Comprar tokens:** Aquest pas és el més complicat, necessitem una instància del contracte, una del compte de la cartera i una de la blockchain. Gràcies a la instància del contracte, podem cridar a la funció buyTokens(), definida al SmartContract que hem mencionat prèviament. Aquesta funció necessita saber la quantitat de tokens que volem comprar, però un cop cridem a la funció, hem de passat uns paràmetres per a enviar la transacció a la Testnet. Els paràmetres són: 'from', la instància del compte de la cartera de l'usuari, i 'value', el cost total de la transacció, això inclou el preu del 'gas'. En aquest punt apareix la finestra que demana a l'usuari que accepti la transacció, on pot veure el preu total i el preu del 'gas' (Vegeu Figura 12).

Si tot ha funcionat correctament, l'usuari no cancel·la la transacció i disposa de suficients Ethereums, i la xarxa, en aquest cas la testnet, té un funcionament correcte, l'usuari rebrà els tokens a la seva cartera i ho podrà comprovar en la mateixa pàgina web.

Ara ja sabem com es fan les transaccions en les quals intercanviem Ethereum per HostCoins, explicarem com fem per a intercanviar HostCoins pel nostre servei. Aquest tipus d'intercanvi es troba en diferents llocs de la nostra pàgina web, però explicarem el cas principal en el qual el trobem, en el moment de comprar un servidor.

El procés és molt paregut al cas anterior, l'únic pas que canvia és l'últim, hem de tenir en compte que els tokens també s'emmagatzemen en la cartera de l'usuari, algunes carteres permeten inclús mostrar el valor d'aquests si són configurades correctament.

- **Cartera de l'usuari:** Idèntic al cas anterior.
- **Recuperar el seu balanç**: Idèntic al cas anterior.
- **Gastar tokens:** Necessitem una instància del contracte, una del compte de la cartera i una de la blockchain. Gràcies a la instància del contracte, podem cridar a la funció spendTokens(), definida al SmartContract que hem mencionat prèviament. Aquesta funció necessita saber la quantitat de tokens que ha de cobrar a l'usuari, però un cop cridem a la funció, hem de passat uns paràmetres per a enviar la transacció a la Testnet. En aquest cas el paràmetre és: 'from', la instància del compte de la cartera de l'usuari. En aquest punt apareix la finestra que demana a l'usuari que accepti la transacció, on pagarà amb tokens i podrà veure el preu del 'gas', ja que les transaccions amb tokens també necessiten d'aquest.

Si tot ha funcionat correctament, l'usuari no cancel·la la transacció i disposa de suficients Ethereums, i la xarxa, en aquest cas la testnet, té un funcionament correcte, la nostra

pàgina web continuarà amb la seva funció, en aquest exemple, crear el servidor automàticament amb els paràmetres que ha introduït l'usuari.

Sistema de facturació

Una de les funcionalitats característica de la nostra plataforma web i que ens diferencia de la competència, és el concepte "pay as you go". Aquesta forma de facturació és la mateixa que fa servir AWS.
Consisteix en pagar pel que has utilitzat, és a dir segons el temps que està funcionant el teu servidor i els recursos que has gastat (de CPU, memòria, tràfic de xarxa, etc.).

Inspirant-nos en Amazon, hem implementat un sistema de facturació similar però més simple amb possibilitat d'ampliació per a aproximar-nos al sistema d'Amazon.
El nostre servei es factura per dies, és a dir que l'usuari paga pels dies que té pensat utilitzar el servidor i en funció dels recursos que ha triat.

Aquí podem veure com en aquest server seleccionant els recursos que necessita l'usuari, el primer cobrament serà de 6 HOCs,

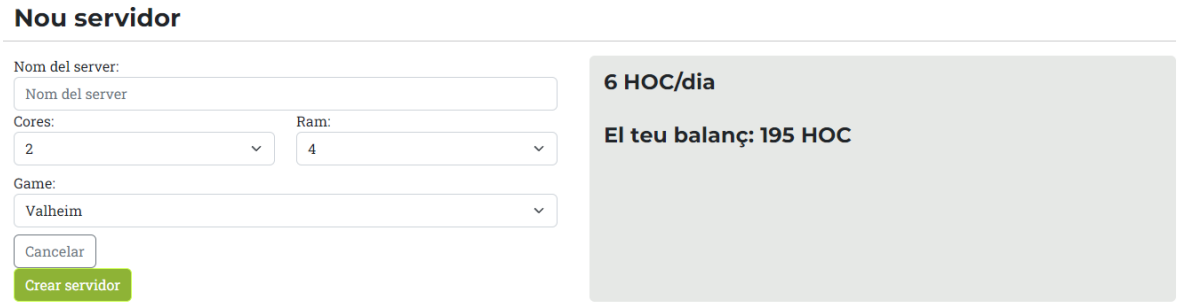


Figura 16: Formulari de creació i pagament d'un servidor

de tal manera que si volem augmentar els dies de servei del servidor, ens costarà en aquest cas 6 HOCs/dia.

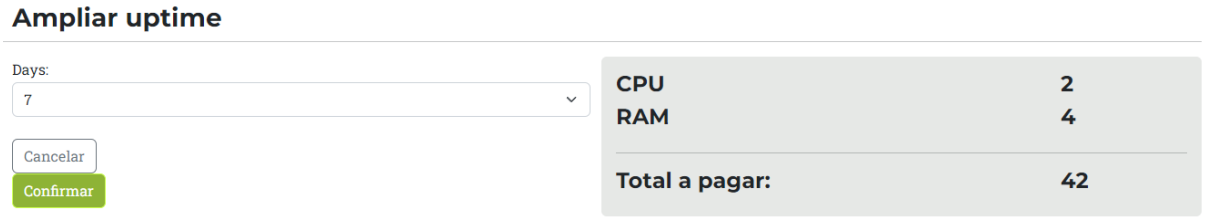


Figura 17: Formulari d'ampliació de servei d'un servidor

Resultats

GameSave com a plataforma, actualment és totalment funcional i està preparada per a ser desplegada per al públic. Tot i que podem implementar moltes millores, ofereix als seus usuaris les funcions necessàries per a crear i administrar un servidor per a gaudir del seu videojoc preferit amb amics.

GameSave aconsegueix solucionar dos problemes molt presents en els usuaris d'altres plataformes similars, i això ens ajuda a diferenciar-nos dels competidors. Implementem els avantatges que proporciona l'ús d'un token i una blockchain, això pot ser una atracció per a molts d'usuaris aprofitant la creixent fama d'aquest món de les criptomonedes. Per una altra banda, implementem el concepte de "Pay as you go", mai implementat en l'àmbit de l'allotjament dels servidors de videojocs.

Per una banda, la creació automàtica dels servidors en contenidors de Docker, ha estat un èxit. Això permet automatitzar la plataforma, no sols en la creació d'aquests, sinó també a l'hora d'accedir a la configuració d'un contenidor concret, o cosa que és el mateix, el servidor d'un usuari.

Per l'altra banda, la creació del nostre propi token, HostCoin, també ha estat un èxit. El token, tot i ser simple, compleix amb les necessitats i ofereix els avantatges promesos a l'usuari.

A continuació adjuntem l'enllaç directe al nostre repositori amb tot el projecte i instruccions per a posar-lo en funcionament. Github: GameSave

Conclusió

Per a concloure el nostre projecte final explicarem la nostra experiència durant el desenvolupament d'aquest projecte.

El projecte ha estat més difícil del que esperàvem a causa de la quantitat de coneixement que mancàvem en alguns àmbits com Django o la tecnologia Blockchain, gran part del nostre temps ha estat invertida en investigar o formar-nos sobre alguns d'aquests temes.

Respecte a la tecnologia Blockchain, ens apassionava molt saber com funciona i crear la nostra pròpia criptomoneda, però ha estat un dels passos més difícils a causa del poc coneixement, com bé hem dit, i a la poca informació que trobem en línia en ser una tecnología nova.

Aprendre sobre Django no va ser una tasca fàcil, ja que havíem fet servir frameworks pareguts, però cap funcionava amb Python com a llenguatge de programació. Afegim un element més a la llista de frameworks que controlem.

Per a acabar, la creació i administració de servidors en Docker des d'una pàgina web, va ser tot un repte. El nostre coneixement sobre Docker era més baix del necessari i tot i anar progressant, hi havia moltes tasques a fer, la majoria d'aquestes les hem assolit, però podríem afegir-ne moltíssimes més.

Al final, el projecte ha estat tot un repte el qual ha estat molt divertit i ha ajudat a la nostra formació i experiència. Hem aplicat moltes tècniques de disseny, planificació i programació que hem anat coneixent mentre cursàvem el nostre grau. Finalment, estem molt satisfets amb el resultat final i la versió actual del projecte.

Treball futur

El nostre projecte en l'estat actual és totalment funcional i compleix amb les característiques que teníem previstes en un principi.

Tot i això, la plataforma pot ser millorada de moltes maneres més i requereix un manteniment, ja que les tecnologies canvien i també apareixen nous jocs.

A continuació detallarem una llista dels que serien els següents passos per a continuar fent créixer la plataforma:

- **Afegir més jocs:** En un món on nous videojocs es tornen rellevant dia a dia, hem d'estar afegint noves opcions de servidors constantment, tant per a videojocs nous com per a altres d'antics que encara no estan implementats en la nostra plataforma.
- **Més opcions de configuració:** En els nostres servidors disposem d'interfícies per fer les configuracions més bàsiques dels nostres servidors, podem afegir altres interfícies o millorar les actuals.
- **Funcionalitats personalitzades per a cada joc:** No tots els videojocs disposen de les mateixes opcions de configuració. Hem d'adaptar el tauler de control a cada videojoc.
- **Millores visuals i de disseny:** El disseny de la pàgina web podria millorar o requerir canvis amb el pas del temps.
- **Millorar la implementació de "Pay as you go":** Aquest concepte pot ser implementat a molts de nivells, en aquest moment està en un nivell molt baix, però podem millorar-ho per a beneficiar als nostres usuaris.

Referències

[1] Amazon Web Services (AWS) . Cloud computing (https://aws.amazon.com/)
[2] Microsoft Azure. Servicios de informática en la nube (https://azure.microsoft.com/)
[3] Google Cloud. Servicios de cloud computing (https://cloud.google.com/)
[4] NITRADO. ¡Alquile un servidor de juegos! (https://server.nitrado.net/)
[5] GPORTAL. Alquilar un servidor de juegos del mejor proveedor (https://www.g-portal.com/)
[6] “Pay as you go”. Pay-as-you-go: ¿el modelo de precios a adoptar en cloud computing? (https://www.appvizer.es/revista/contabilidad-finanzas/procesamiento-de-pagos/pay-as-you-go)
[7] PyCharm. The Python IDE for Professional Developers (https://www.jetbrains.com/pycharm/)
[8] Remix - Ethereum IDE. The Native IDE for Web3 Development. (https://remix.ethereum.org/)
[9] Docker. Use containers to Build, Share and Run your applications (https://www.docker.com/)
[10] Metamask. A crypto wallet & gateway to blockchain apps (https://metamask.io/)
[11] Django. DjangoThe web framework for perfectionists with deadlines (https://www.djangoproject.com/)
[12] Bootstrap. Build fast, responsive sites with Bootstrap (https://getbootstrap.com/)
[13] Ethereum. Ethereum is the community-run technology powering the cryptocurrency ether (ETH) and thousands of decentralized applications. (https://ethereum.org/)
[14] Solidity. A statically-typed curly-braces programming language designed for developing smart contracts that run on Ethereum. (https://soliditylang.org/)
[15] Kubernetes. Orquestación de contenedores para producción (https://kubernetes.io/)
[16] Alchemy. The web3 development platform (https://www.alchemy.com/)
[17] Web3. A JavaScript library for building on Ethereum (https://web3js.org/)