2018

Rifes en Blockchain

TREBALL DE FINAL DE CARRERA

LAILA ABJIL

CENTRE EASY | Universitat de Girona

1 INDEX

2 INTRODUCCIÓ 5

2.1 INTRODUCCIÓ 5

2.2 MOTIVACIONS 6

2.3 PROPÒSIT 7

2.4 OBJECTIUS 8

3 ESTUDI DE VIABILITAT 9

4 METODOLOGIA 10

5 PLANIFICACIÓ 12

6 MARC DE TREBALL I CONCEPTES PREVIS 14

6.1 Blockchain 14

6.2 Smart Contracts 17

6.3 Wallet o Cartera d'ethereum 18

6.4 Solidity 20

6.5 NodeJS 22

7 REQUISITS DEL SISTEMA 24

7.1.1 Requisits funcionals, no funcionals i d'interfície 24

7.1.2 Requisits d'interfície 25

7.1.3 Requisits funcionals 32

7.1.4 Requisits no funcionals 48

7.2 Requisits de l'entorn 50

7.2.1 Programari necessari 50

7.2.2 Entorn de desenvolupament 51

7.2.3 El navegador 51

7.2.4 Proves 51

8 ANÀLISI I DISSENY DEL SISTEMA 52

8.1 Diagrames de casos d'ús 52
8.1.1 Inici 52
8.1.2 Login/Registre 53
8.1.3 Veure perfil 53
8.1.4 Veure contractes 54
8.1.5 Editar contracte 54
8.1.6 Acabar contracte 55
8.1.7 Crear contracte 55
8.1.8 Afegir detalls del contracte 56
8.1.9 Afegir percentatges del contracte 56
8.1.10 Participar en una rifa 57
8.1.11 Comprar tiquets 57
8.2 Fitxes de casos d'ús 58
8.3 Diagrama d'activitats 63
9 Arquitectura del sistema 65
10 IMPLEMENTACIÓ I PROVES 67
10.1 BASE DE DADES 67
10.1.1 Lottery 67
10.1.2 Ticket 68
10.1.3 User 69
10.1.4 UserBlock 70
10.2 SERVIDOR 71
10.3 FUNCIONALITATS 72
10.3.1 Crear contracte 72
10.3.2 REGISTRE 83
10.3.3 LOGIN 86

10.3.4 Veure perfil 89
10.3.5 Veure contractes 91
10.3.6 Afegir detalls a contracte 92
10.3.7 Afegir percentatges a contracte 95
10.3.8 Editar contracte 97
10.3.9 Acabar contracte 100
10.3.10 Veure llista de rifes 104
10.3.11 Comprar tiquet 106
10.4 CLIENT 110
10.5 PROBLEMES 113
11 IMPLEMENTACIÓ I RESULTATS 114
11.1 Implementació 114
11.2 RESULTATS 117
11.2.1 ENGEGAR BLOCKCHAIN 117
11.2.2 INICI 118
11.2.3 LOGIN 118
11.2.4 REGISTRE 119
11.2.5 VEURE PERFIL 119
11.2.6 VEURE CONTRACTES 120
11.2.7 CREAR CONTRACTE 120
11.2.8 AFEGIR DETALLS A CONTRACTE 121
11.2.9 AFEGIR PERCENTATGES AL CONTRACTE 122
11.2.10 VEURE CONTRACTES 123
11.2.11 PARTICIPAR EN UNA RIFA 124
11.2.12 COMPRAR TIQUETS 124
11.2.13 EDITAR CONTRACTE 127

11.2.14 ACABAR CONTRACTE 128
11.3 DEMOSTRACIONS 131
11.3.1 Participants 131
11.3.2 Cas d'ús 132
11.3.3 Opinions 133
12 CONCLUSIONS 134
13 TREBALL FUTUR 136
14 ANNEXOS 137
15 BIBLIOGRAFIA 138
16 MANUAL D'USUARI I/O INSTAL·LACIÓ 142

2 INTRODUCCIÓ

2.1 INTRODUCCIÓ

Les donacions sempre han tingut un problema de manca de transparència, les donacions no arriben al destinatari i hi ha desconfiança alhora que els administradors no en facin un bon ús.

Aquest projecte està destinat a resoldre aquest problema mitjançant les tecnologies Blockchain i SmartContract.

En aquest projecte, crearem una aplicació basada en un Smart contract que controli el sistema de rifes per tal que no sigui l'administrador el que gestioni els pagaments sinó el Smart contract.

2.2 MOTIVACIONS

La meva motivació per realitzar aquest projecte ha estat poder conèixer el món del blockchain, aprendre sobre aquest món que ha sorgit fa poc i volia tenir una experiència creant una aplicació amb tecnologies que no havia utilitzat anteriorment, que són útils en altres sectors enllaçant-la amb un Smart contract propi d'aplicacions Blockchain.

El projecte de final de carrera és una oportunitat per aprendre, investigar i posar en pràctica les aptituds que hem adquirit durant la carrera.

Avui en dia, les tecnologies basades en Blockchain s'han posat molt de moda, i crec que tenir l'oportunitat de conèixer aquest món i crear una aplicació centrada amb aquesta tecnologia era la millor manera de posar en pràctica el meu projecte final.

2.3 PROPÒSIT

El propòsit d'aquest projecte és mostrar la realització del sistema de rifes amb Ethereum SmartContracts, mostrar uns casos d'ús i una interfície bàsica per entendre com s'utilitzaria en un camp real.

El sistema consta en què el mateix creador de la rifa, sempre pagarà tots els tiquets que li comprin. És a dir, un client quan compri un tiquet d'una rifa pagarà la tarifa corresponent i, el creador els pagarà en el Smart contract.

Les rifes les gestionarà el contracte que tindrà un adreça usuari per cada client. L'usuari pagarà cada tiquet que hagi comprat un client amb aquestes adreces de manera que tot el gestionat de manera real quedi reflectit en el contracte. Així, un cop acabat el contracte, el contracte retornarà quina adreça ha estat la guanyadora i, el creador, haurà de pagar al guanyador per obtenir el seu premi.

Amb aquest sistema, com que l'usuari creador ha de pagar tots els tiquets i un percentatge extra en el sistema, controlarem que faci el pagament real al guanyador perquè si no ell sortirà perdent diners.

2.4 OBJECTIUS

L'objectiu del projecte és desenvolupar amb Ethereum un mecanisme de donacions amb rifes que sigui transparent, traçable i s'executi automàticament un cop s'han declarat les regles de participació.

Els objectius es poden dividir en:

- Crear una aplicació adaptativa amb una interfície senzilla que mostri el sistema.
- Crear el Smart contract amb Solidity.
- Pujar el Smart contract en el Blockchain i lligar-la amb l'aplicació.

3 ESTUDI DE VIABILITAT

L'aplicació que es vol desenvolupar en el projecte està en un entorn de Blockchain pel que s'haurà de configurar tot l'entorn de si mateix juntament amb la tecnologia necessària per desenvolupar l'aplicació web.

És del tot necessari desenvolupar una planificació per poder ajustar el pressupost i els recursos humans necessaris en forma d'hores.

Els requisits necessaris per desenvolupar el projecte són:

- Disposar d'un ordinador amb connexió a internet.
- Disposar dels programes necessaris per a desenvolupar el projecte:
    - Navegador web
    - Programari necessari per executar el blockchain:
        - *Ethereum Wallet*, *Geth* i ***NodeJS***: *Testrpc*, *Truffle*
    - Programari necessari per l'aplicació web:
        - Base de dades *MongoDB*
        - ***NodeJS***: *Mongoose*, *Express*, *Passport*, *Connect-flash*, *Morgan*, *Cookie-parser*, *Body-parser*, *Express-session*, *fs* i *https*.

Un cop realitzada l'avaluació s'ha arribat a la conclusió que el projecte és viable, i que el treball d'una sola persona és necessari per al desenvolupament del projecte amb una dedicació aproximada de hores. Per això, un cop conclòs que s'ha comprovat la total viabilitat tecnològica i humana es pot començar a desenvolupar el Projecte de Final de Carrera.

4 METODOLOGIA

Entenem per metodologia una seqüència d'etapes i procediments recomanats per a ser utilitzats durant el procés de desenvolupament d'un sistema informàtic.

La metodologia utilitzada per fer aquest projecte és la metodologia àgil, ja que ens trobem davant d'un projecte difícil de dissenyar amb antelació a la implementació.

Les metodologies àgils són aquelles metodologies que es basen en un desenvolupament iteratiu i incremental. Les metodologies àgils intenten minimitzar el risc desenvolupant el projecte en iteracions.

Cada iteració és com un projecte en miniatura del projecte final, i inclou totes les tasques necessàries ordenades per prioritat per després implementar les funcionalitats noves. Les iteracions estan formades per diferents etapes, dividies en: Planificació, anàlisi de requisits, disseny, codificació, testatge i documentació.

Aquestes metodologies tenen les següents característiques:

- **Desenvolupament iteratiu i incremental:** Desenvolupar de manera incremental afegint poques funcionalitats en cada iteració per assegurar-se del correcte funcionament. Així, serem capaços d'evitar errors en estats més avançats del projecte i haver de fer canvis posteriors en aspectes del projecte que donàvem per correctes.

- **Testeig continu i unitari:** s'ha de testejar cada funcionalitat de manera precisa i unitària. D'aquesta manera, podrem detectar de manera més precisa on es produeix l'error.

- **Interacció desenvolupador-client:** és recomanable que hi hagi bona comunicació entre el client i el desenvolupador.

- **Correcció d'errors:** Abans d'afegir qualsevol funcionalitat nova cal assegurar-se que tots els errors s'hagin corregit.

- **Refactorització de codi:** consisteix a reescriure el codi per millorar la seva llegibilitat i manteniment però sense canviar el seu comportament.

- **Codi compartit:** Es busca que qualsevol persona que formi part de l'equip de desenvolupament sigui capaç d'entendre, corregir i expandir qualsevol part del codi. Per aconseguir-ho cal realitzar un codi simple i entenedor.

5 PLANIFICACIÓ

Per la realització del treball es va decidir dividir-ho en diferents blocs:

- **Aprenentatge del funcionament del Blockchain,Smart Contract i les tecnologies necessàries per al desenvolupament de l'aplicació web.**

  Donat que no es disposa de cap coneixement sobre Blockchain i SmartContracts i que tampoc es disposa de coneixements de NodeJS per poder crear l'aplicació web, s'hauran d'invertir un conjunt d'hores per aprendre sobre el seu funcionament bàsic.

- **Requisits.** Definirem els requisits del projecte i planificarem les iteracions necessàries abans de començar a treballar.
  - **Definir els requisits d'interfície, els funcionals i els no funcionals.** D'aquesta manera sabrem els requisits necessaris a implementar.
  - **Generar la matriu de dependències.** Per saber quins requisits depèn dels altres, s'ha de generar una matriu de dependències i així se sap amb antelació si un requisit depèn d'un altre i, per tant, s'ha de fer abans.
  - **Descomposició en paquets de treball.** Es mirarà l'abast del projecte i es descompondrà en paquets de treball individuals. Els paquets es subdividiran fins que puguem estimar i gestionar el temps i el cost i puguem determinar un lliurable. Cada paquet es podrà descompondre en diferents subpaquets.
  - **Generar la fitxa de paquet.** Per cada paquet, s'haurà de fer una fitxa que contindrà:
    - Nom del paquet.
    - Tasques a realitzar.
    - Personal implicat.
    - Estimació de temps.
    - Recursos necessaris.
    - Lliurables.

- **Matriu de traçabilitat.** Matriu on es defineix cada paquet de treball a quin requisit correspon. Cada requisit tindrà com a mínim una tasca relacionada en un paquet de treball. A cada paquet com a mínim hi haurà un requisit associat. Es representa en forma matricial.

- **Implementació.** S'implementarà i es testejarà cada paquet de treball respectant la metodologia àgil descrita anteriorment.

- **Anàlisis i disseny.** S'analitzarà mitjançant diagrames de casos d'ús, fitxes de casos d'ús i diagrames d'activitats els diferents requisits que ha de satisfer l'aplicació.

6 MARC DE TREBALL I CONCEPTES PREVIS

En aquest projecte es vol proporcionar una aplicació web que basada en el Blockchain i Smart contract gestioni un sistema de rifes.

Per poder entendre aquest treball, es necessiten conèixer una sèrie de conceptes i tecnologies que a continuació explicaré.

6.1 BLOCKCHAIN

Blockchain és una base de dades distribuïda que contenia els registres de totes les transaccions que es realitzaven a la xarxa.

És una tecnologia que permet la realització confiable i segura de qualsevol tipus de transacció entre dos o més persones sense la necessitat d'intermediaris, a través d'Internet.

Va néixer a partir de la criptomònada Bitcoin, la 1ra plataforma blockchain. Originalment, Bitcoin es va crear com a un sistema electrònic de pagament entre parells (Peer-toPeer Electronic Cash System), per això es coneix com a "diners virtuals".

Blockchain és una articulació de tecnologies estructurades en un sistema naturalment encriptat, la qual cosa proporciona als usuaris involucrats protecció de les seves identitats i de les dades de les seves transaccions.

- És un llibre de comptabilitat digital on s'anoten totes les transaccions que succeeixen a la xarxa, agrupades en blocs que contínuament són enllaçats linealment entre si: el primer bloc amb el segon, el segon amb el tercer, i així successivament (d'aquí el nom de 'cadena de blocs').

- Les carteres digitals són interfícies gràfiques per interactuar amb la xarxa blockchain que permeten als usuaris realitzar transaccions i manejar les seves identitats digitals.

- Els miners, ordinadors que s'encarreguen d'autoritzar que s'afegeixin els blocs de transaccions que realitzen les carteres digitals a la cadena de blocs, en resoldre *hashos*, rebent a canvi recompenses en la moneda digital pròpia de la xarxa blockchain.

- Els nodes són computadors que s'encarreguen d'emmagatzemar una còpia exacta del Blockchain i de fer complir les regles de la xarxa. Alguns s'encarreguen a més d'escoltar noves transaccions i agrupar-les en blocs per proposar-los com a treballs als miners, que després de ser confirmats són propagats a la xarxa i afegits a la cadena.

Una transacció comuna en blockchain comença amb l'enviament d'un actiu digital d'una cartera digital a una altra. Aquesta transacció és escoltada per diversos nodes i agrupada amb altres transaccions que aquests nodes hagin escoltat. Aquest grup de transaccions és després enviat als miners com un treball a resoldre. Els miners prenen el treball i

competeixen entre si per aconseguir un valor (nonce) que resol un hash que autoritza al miner que ho troba a proposar el seu bloc, amb les transaccions en ell, a ser agregat a la cadena. El bloc proposat, a més de contenir les transaccions, també conté la identificació del bloc anterior i un valor d'emissió de monedes per part de la xarxa (la recompensa), les quals s'adrecen a la cartera del miner guanyador.

Tot aquest procés, al que se li coneix com a mineria, succeeix en una transacció en blockchain. Per als usuaris, l'única cosa visible són la quantitat de confirmacions que les seves transaccions reben i el temps que aquestes prenen a realitzar-se.

La tecnologia blockchain garanteix que els registres de les transaccions realitzades siguin vàlids i inalterables. Podem veure cada bloc com cadascuna de les pàgines d'un mateix llibre comptable pràcticament infinit, només que aquí el que ha estat escrit no pot esborrar-se ni repetir-se: cada transacció o dada es protegeix amb una empremta digital única. Això es coneix com a inmutabilitat.

Les primeres i més conegudes blockchains, fins als moments, són mantingudes per molts nodes i miners al voltant de tothom i cap té realment el poder per controlar-la i aprovar o desaprovar transaccions segons el seu propi criteri, al que se li crida descentralització: el poder no està centrat en una sola part, sinó distribuït entre moltes parts que han d'arribar a un acord.

6.2 SMART CONTRACTS

El món de les criptomònades va començar amb la creació del Bitcoin l'any 2009 i amb ell va arribar la tecnologia Blockchain.

Uns anys més tard, gràcies a Etherum, es va crear un projecte molt més potent i amb moltes més utilitats, els Smart Contracts.

Els contractes intel·ligents són programes informàtics que executen acords establerts entre dos o més parts quan es dóna una condició programada amb anterioritat. És a dir, són contractes que s'executen i es fan complir a si mateixos de manera automàtica i autònoma.

Perquè els smart contracts s'executin necessiten instruccions escrites en codi.

Igual que al món real existeixen multitud de llenguatges al món informàtic també existeixen multitud de llenguatges de programació, amb els Smart contracts s'utiltza el Solidity.

6.3 WALLET O CARTERA D'ETHEREUM

Així com per guardar els nostres euros o dòlars podem acudir a una gran diversitat de bancs o fins i tot posar-los a Paypal, existeixen també diverses formes de guardar nostres criptomonedes.

Un moneder o wallet de Ethereum és qualsevol via que ens permeti guardar, rebre i transferir nostres Ether.

Cada wallet de té una adreça pròpia d'aquest tipus:

```
0x5705c6210a8bDA00dfE297a6i=F60fb05f6Fbf
```

Que se sol mostrar també amb un codi QR associat per poder realitzar pagaments o enviaments de Ether amb el mòbil de manera més senzilla:

Figura 134: Codi QR

És a dir, si jo tinc un moneder de Ethereum amb 1.86Ether dins i vull enviar la meitat d'aquesta quantitat a la meva mare com a regal, solament hauria de saber l'adreça del seu moneder per poder fer-ho a l'instant.

Així mateix, cada moneder té una clau de seguretat pròpia (anomenada private key) per poder accedir a ell.

No és el mateix un moneder de Bitcoin que un de Ethereum que un altre de qualsevol altre criptomoneda. Hi ha serveis que et permeten tenir diverses monedes en una mateixa web o lloc (com Coinbase) però això significa que tens també diversos moneders separats en allà.

És a dir, generalment en un mateix moneder no poden haver-hi diferents monedes. Cada adreça, com la qual hem vist a dalt, correspon a una sola criptomoneda.

En el meu cas, he fet servir el Ethereum Wallet amb el qual tenia la meva cartera amb les creades a myEtherWallet.com

6.4 SOLIDITY

Solidity és un llenguatge d'alt nivell orientat a contractes. La seva sintaxi és similar a la de Javascript i està enfocat específicament a la Màquina Virtual de Etehreum (EVM).

Solidity està desenvolupat de manera estàtica i accepta, entre altres coses, herències, llibreries i tipus complexos definits per l'usuari.

Quan hem dit que Solidity és un llenguatge de programació d'alt nivell es referia al fet que era un llenguatge 'Turing Complet'.

Aquest concepte, ideat per Alan Turing, es refereix en informàtica a aquell llenguatge que té un poder computacional equivalent al que es denomina una "Màquina de Turing Universal".

La Ethereum Virtual Machine (EVM) és considerada una Màquina de Turing Universal, terme que es refereix al programari que és prou hàbil com per executar qualsevol codi definit pel desenvolupador.

Amb el Solidity és possible crear contractes per votar, per a subhastes a cegues, per a moneders muti signatures, i molt més.

Dit d'una altra manera, el terme Turing Complet aplicat a la tecnologia Blockchain i especialment als contractes intel·ligents, es refereix a la capacitat que té un llenguatge

informàtic per resoldre qualsevol problema computacional i per afegir regles complexes com, per exemple, els bucles.

La millor manera de provar Solidity ara mateix és usant Remix.

6.5 NODEJS

NodeJS és un entorn d'execució multiplataforma de codi obert per desarrotllar aplicacions web. Aquesta llibreria s'executa sobre JavaScript i ha sigut creada per Google.

Node (o més correctament: Node.js) és un entorn que treballa en temps d'execució, de codi obert, multi-plataforma, que permet als desenvolupadors crear tota classe d'eines de costat servidor i aplicacions en Javascript.

L'execució en temps real està pensada per usar-se fóra del context d'un explorador web (és a dir, executar-se directament en una computadora o sistema operatiu de servidor). Com a tal, l'entorn omet les APIs de Javascript específiques de l'explorador web i afegeix suport per APIs de sistema operatiu més tradicionals que inclouen HTTP i biblioteques de sistemes de fitxers.

Des d'una perspectiva de desenvolupament de servidor web, Node té un gran nombre d'avantatges:

Alt rendiment: Node ha estat dissenyat per optimitzar el rendiment i l'escalabilitat en aplicacions web i és un molt bon complement per a molts problemes comuns de desenvolupament web (ej, aplicacions web en temps real).

El codi està escrit en "simple Javascript", la qual cosa significa que es perd menys temps ocupant-se de les "commutacions de context" entre llenguatges quan estàs escrivint tant el codi de l'explorador web com del servidor.

Javascript és un llenguatge de programació relativament nou i es beneficia dels avanços en disseny de llenguatges quan es compara amb altres llenguatges de servidor web tradicionals (ej, Python, PHP, etc.) Molts altres llenguatges nous i populars es compilen/converteixen a Javascript de manera que pots també usar CoffeeScript, ClosureScript, Scala, LiveScript, etc.

El gestor de paquets de Node (NPM de l'anglès: Node Packet Manager) proporciona accés a centenars o milers de paquets reutilitzables. Té a més la millor a la seva classe resolució de dependències i pot usar-se per automatitzar la major part de la cadena d'eines de compilació.

És portable, amb versions que funcionen en Microsoft Windows, US X, Linux, Solaris, FreeBSD, OpenBSD, WebOS, i NonStop US. A més, està ben suportat per molts dels proveïdors d'hostalatge web, que proporcionen infraestructura específica i documentació per a hostalatge de llocs Node.

Té un ecosistema i comunitat de desenvolupadors de tercers molt activa, amb quantitat de gent desitjosa d'ajudar.

7 REQUISITS DEL SISTEMA

7.1.1 Requisits funcionals, no funcionals i d'interfície

L'objectiu de l'especificació dels requeriments és expressar d'una forma clara què ha de fer el sistema que s'ha de desenvolupar i quins problemes s'han de resoldre. En aquest cas, s'especificaran els requeriments del sistema: els d'interfície, funcionals i no funcionals.

- **Requisits d'interfície:** descriuen com s'ha de mostrar per pantalla.

- **Requisits funcionals:** descriuen quins serveis ha d'oferir l'aplicació independentment de l'implantació. Requereixen una descripció més dinàmica i detallada per a ser entesa. Per això, els diagrames de casos d'ús, les fitxes de casos d'ús i els diagrames d'activitats són una bona eina per capturar i expressar detalladament el comportament de l'aplicació.
    - **Diagrames de casos d'ús:** aquest tipus de diagrames ens mostren visualment la comunicació entre els usuaris i els casos d'ús. Aquests casos descriuen les funcionalitats sense tenir en compte la implementació.
    - **Fitxes de casos d'ús:** la informació sobre qui ha de realitzar la funcionalitat i quins passos s'ha de seguir per tal d'arribar a finalitzar l'acció amb èxit.
    - **Diagrames d'activitat:** aquest tipus de diagrama ens mostra el flux d'accions que durà a terme l'activitat durant tot el procés a realitzar.

- **Requeriments no funcionals:** descriuen aspectes generals de l'aplicació que calen complir però que no tenen a veure amb les funcionalitats del sistema.

7.1.2 Requisits d'interfície

A continuació es veurà com es mostra per pantalla el projecte. Al ser una aplicació web, la interfície es compon de diferents pantalles pel que es mostrarà cada una de la interfície de cada una de les pantalles.

7.1.2.1 Pantalla inicial

Es començarà amb una pantalla on es pot veure un text explicant la funcionalitat dels 2 botons i, 2 botons. Un botó per crear una rifa i un botó per participar-ne en una. El botó crear_rifa et redirigeix a la pantalla de login i el botó participar_rifa et redirigeix a la pantalla participar.

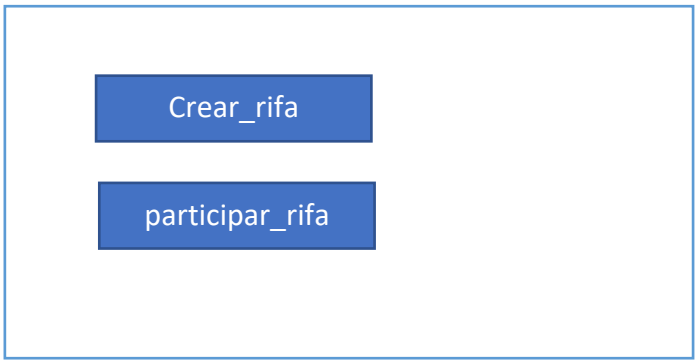


Figura 1: pantalla inicial

7.1.2.2 Pantalla de login

En aquesta pantalla es mostrarà dos inputs per entrar el correu i la contrasenya i un botó, per poder-te loguejar. Després també es mostrarà un text on conté un enllaç per registrar-te. El enllaç condueix a la pantalla de registre. Al loguejar-te, et redirigeix a la pantalla de perfil.

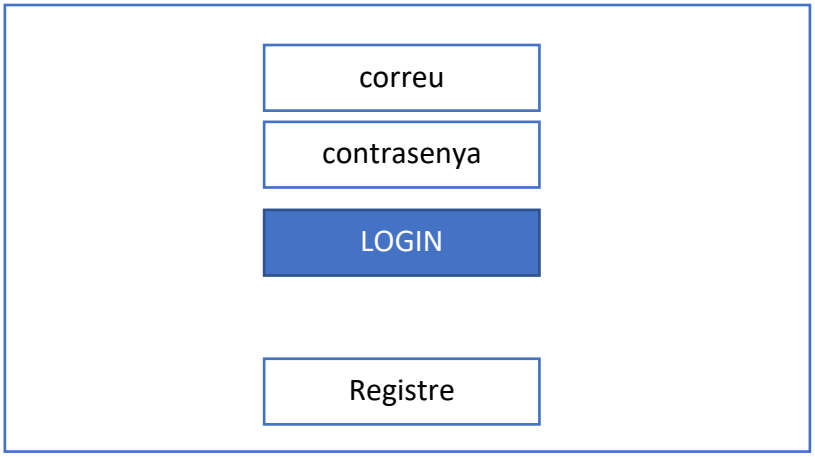


Figura 2: pantalla login

7.1.2.3 Pantalla de registre

En aquesta hi ha una sèrie de inputs demanant el correu, la contrasenya, el nom, la data naixement, el gènere, el número de telèfon, l'adreça, la població i el codi postal; i un botó que redirigeix a la pantalla de perfil.

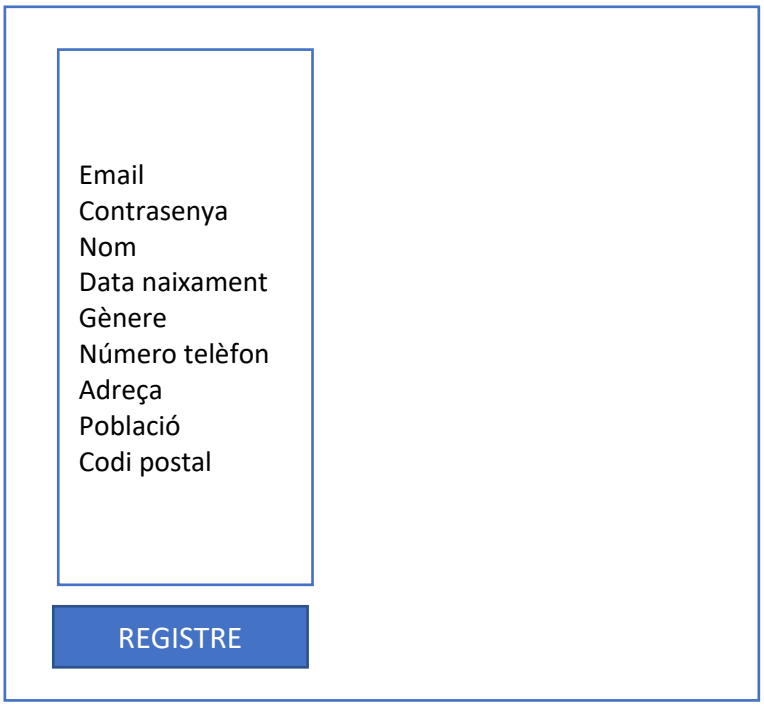


Figura 3: pantalla registre

7.1.2.4 Pantalla de perfil

En aquesta pantalla es mostra el perfil amb l'email, el nom, el dia de naixement, el telèfon i la direcció, un botó a la dreta pel *Logout* i un botó *Contractes* que redirigeix a la pantalla de contractes.

Figura 4: pantalla perfil

7.1.2.5 Pantalla de contractes

En aquesta pantalla es mostra un botó LogOut, una llista de les rifes del propietari amb els elements: index, nom, descripció, participants i accions, on a la casella d'accions hi ha 2 botons: *editar* i *acabar*. EL botó *editar* et redirigeix a la pantalla editar i el botó acabar et redirigeix a la pantalla acabar_contracte. Finalment, després de la llista es mostra un botó *crear_contracte* que redirigeix a la pantalla crear_contracte.

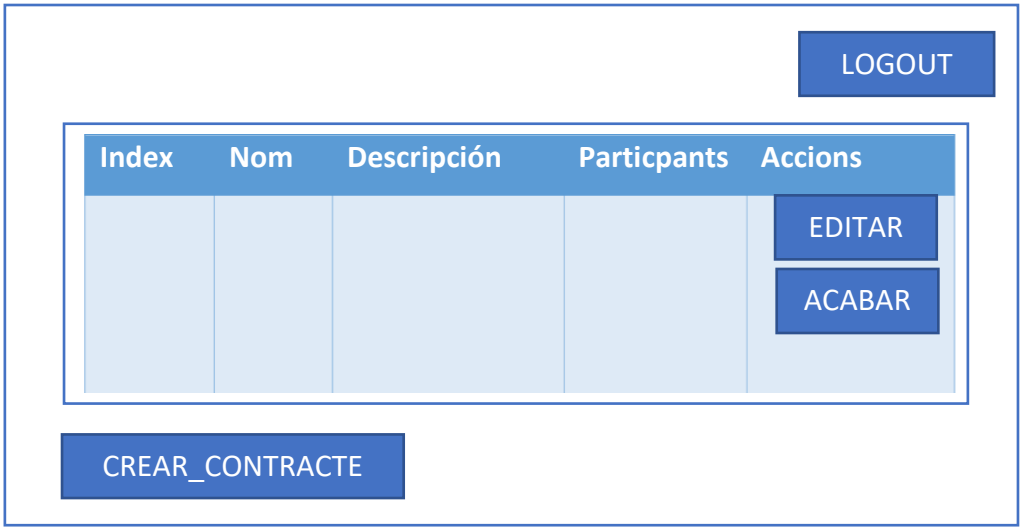


Figura 5: pantalla contractes

7.1.2.6 Pantalla de crear_contracte

Pantalla on es mostren 3 inputs: nom, descripció i adreça de la rifa; i un botó *Següent* que redirigeix a detalls_contracte.

Figura 6: pantalla crear_contracte

7.1.2.7 Pantalla de detalls_contracte

Pantalla on es mostra 2 inputs demanant el màxim de guanyadors i el màxim de tiquets, després un botó *modificar*; 1 input més demanant el preu per tiquet i un botó per *afegir_preu*; i un boto següent al final que redirigeix a la pantalla percentatges.

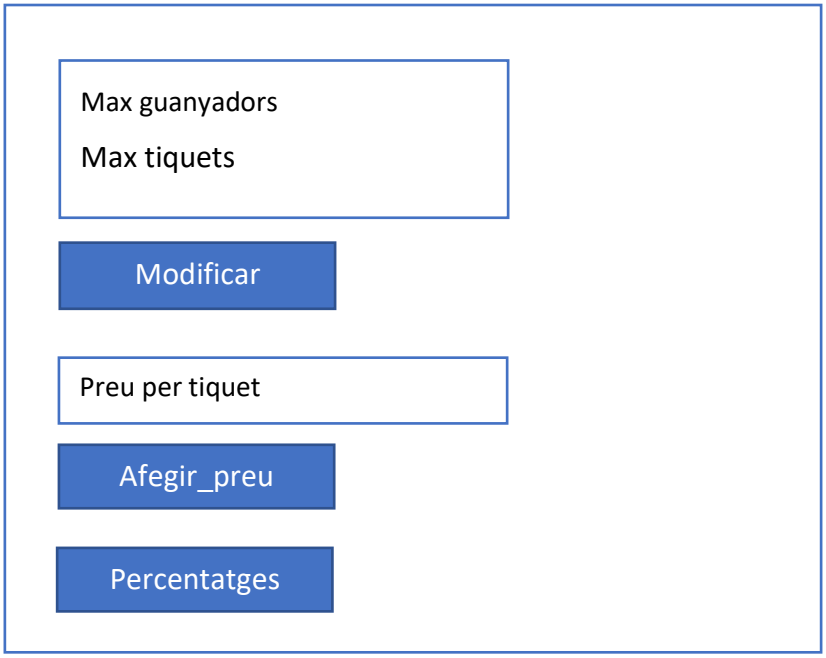


Figura 7: pantalla detalls contracte

7.1.2.8 Pantalla de percentatges

Pantalla on es mostren 2 inputs: % del premi del propietari, % del premi del sistema i un botó *Canviar* que redirigeix al perfil.

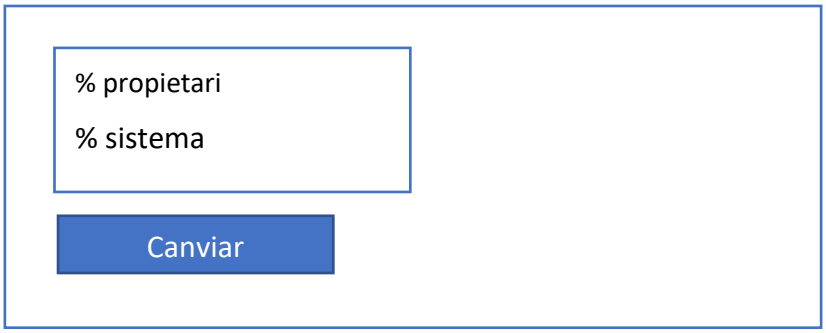


Figura 8: pantalla percentatges

7.1.2.9 Pantalla de editar

Pantalla on es mostra 3 inputs: el màxim de guanyadors, el màxim de tiquets i el preu per tiquet; i un botó *Editar* que redirigeix a la pantalla perfil.

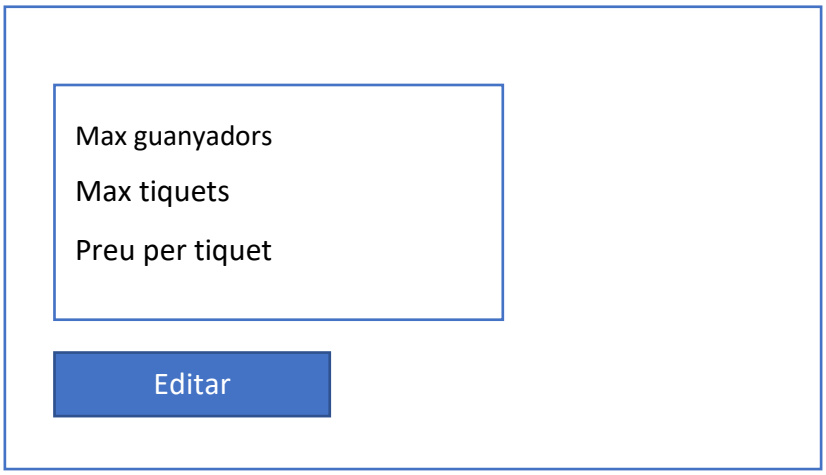


Figura 9: pantalla editar

7.1.2.10 Pantalla de acabar_contracte

Pantalla on es mostra un text informant del guanyador i del que ha de fer per reclamar el premi i un botó *Rebre_premi* que redirigeix a la pantalla perfil.

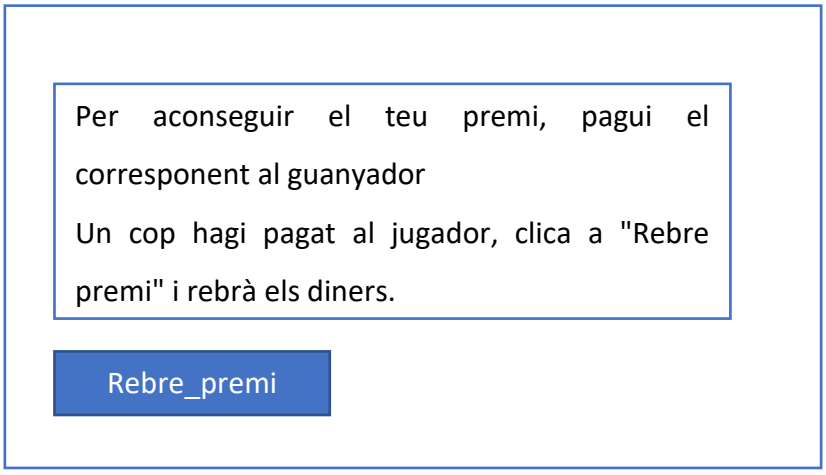


Figura 10: pantalla acabar contracte

7.1.2.11 Pantalla de participar

En aquesta pantalla es mostra un botó Logout, una llista de les rifes amb els elements: index, nom, descripció, participants i accions, on a la casella d'accions hi

ha 1 botó: *comprar_tiquets*. EL botó *comprar_tiquets* et redirigeix a la pantalla comprar_tiquet

Figura 11: pantalla participar

7.1.2.12 Pantalla de comprar_tiquet

Pantalla on es mostra 2 inputs: el telèfon i el número de tiquets a comprar; i un botó *Comprar* que redirigeix a la pantalla participar on es mostra un popup mostrant un missatge notificant que es rebrà un missatge en el telèfon entrat amb les dades del tiquet.

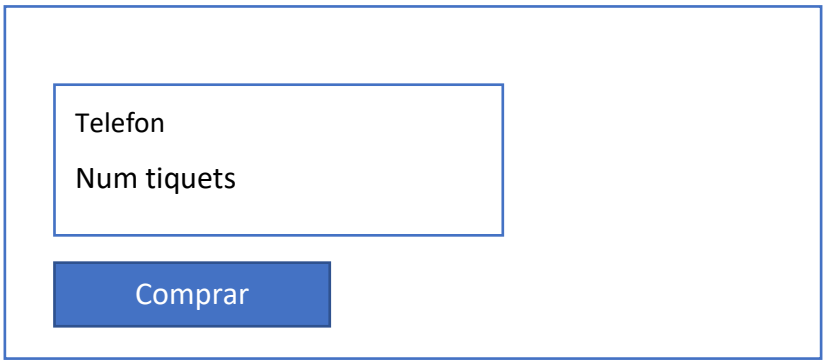


Figura 12: pantalla comprar tiquet

7.1.3 Requisits funcionals

7.1.3.1 REQUISITS FUNCIONALS

Número del requisit	RF1
Nom del requisit	Crear el smart contract
Prioritat del requisit	Alta/Essencial

Número del requisit	RF2
Nom del requisit	Configurar el Blockchain el en ordinador
Prioritat del requisit	Alta/ Essencial

Número delrequisit	RF3
Nom del requisit	Sistema de logueig i registrar
Prioritat del requisit	Alta/ Essencial

Número del requisit	RF4
Nom del requisit	Crear Perfil
Prioritat del requisit	Alta/ Essencial

Número del requisit	RF5
Nom del requisit	Veure llista de contractes de un usuari
Prioritat del requisit	Mitja/Desitjat

Número del requisit	RF6
Nom del requisit	Crear contracte

Prioritat del requisit	Alta/ Essencial

Número del requisit	**RF6.1**
Nom del requisit	Afegir detalls del contracte
Prioritat del requisit	Alta/ Essencial

Número del requisit	**RF6.2**
Nom del requisit	Afegir percentatges del contracte
Prioritat del requisit	Alta/ Essencial

Número del requisit	**RF7**
Nom del requisit	Participar al contracte
Prioritat del requisit	Alta/ Essencial

Número del requisit	**RF8**
Nom del requisit	Veure llista de tots els contractes
Prioritat del requisit	Alta/ Essencial

Número del requisit	**RF9**
Nom del requisit	Comprar tiquet
Prioritat del requisit	Alta/ Essencial

Número del requisit	**RF9.1**
Nom del requisit	Enviar missatge al mòbil al comprar tiquet
Prioritat del requisit	Opcional

Número del requisit	RF10
Nom del requisit	Editar contracte
Prioritat del requisit	Mitja/Desitjat

Número del requisit	RF11
Nom del requisit	Acabar contracte
Prioritat del requisit	Alta/ Essencial

Número del requisit	RF11.1
Nom del requisit	Enviar missatge al acabar al contracte al guanyador per reclamar el premi
Prioritat del requisit	Opcional

Número del requisit	RF11.2
Nom del requisit	Reclamar el premi del propietari
Prioritat del requisit	Alta/ Essencial

Número del requisit	RF12
Nom del requisit	Logout
Prioritat del requisit	Mitja/Desitjat

Número del requisit	RF13
Nom del requisit	Crear la base de dades on guardarem les dades
Prioritat del requisit	Alta/ Essencial

Número del requisit	RF14
Nom del requisit	Dissenyar interfícies
Prioritat del requisit	Alta/ Essencial

Número del requisit	RF15
Nom del requisit	Recollir els requisits del client
Prioritat del requisit	Alta/ Essencial

Número del requisit	RF16
Nom del requisit	Recollir els requisits funcionals i no funcionals
Prioritat del requisit	Alta/ Essencial

7.1.3.2 *MATRIU DE DEPENDÈNCIES*

	1	2	3	4	5	6	6.1	6.2	7	8	9	9.1	10	11	11.1	11.2	12	13	14	15	16
1																		X	X		
2																		X	X		
3																		X	X		
4																		X	X		
5																		X	X		
6																		X	X		
6.1						X												X	X		
6.2						X												X	X		
7																		X	X		
8																		X	X		
9																		X	X		
9.1											X							X	X		
10																		X	X		
11																		X	X		
11.1														X				X	X		
11.2														X				X	X		
12			X															X	X		
13																					X
14																					X
15																					
16																				X	

7.1.3.3 DESCOMPOSICIÓ EN PAQUETS DE TREBALL

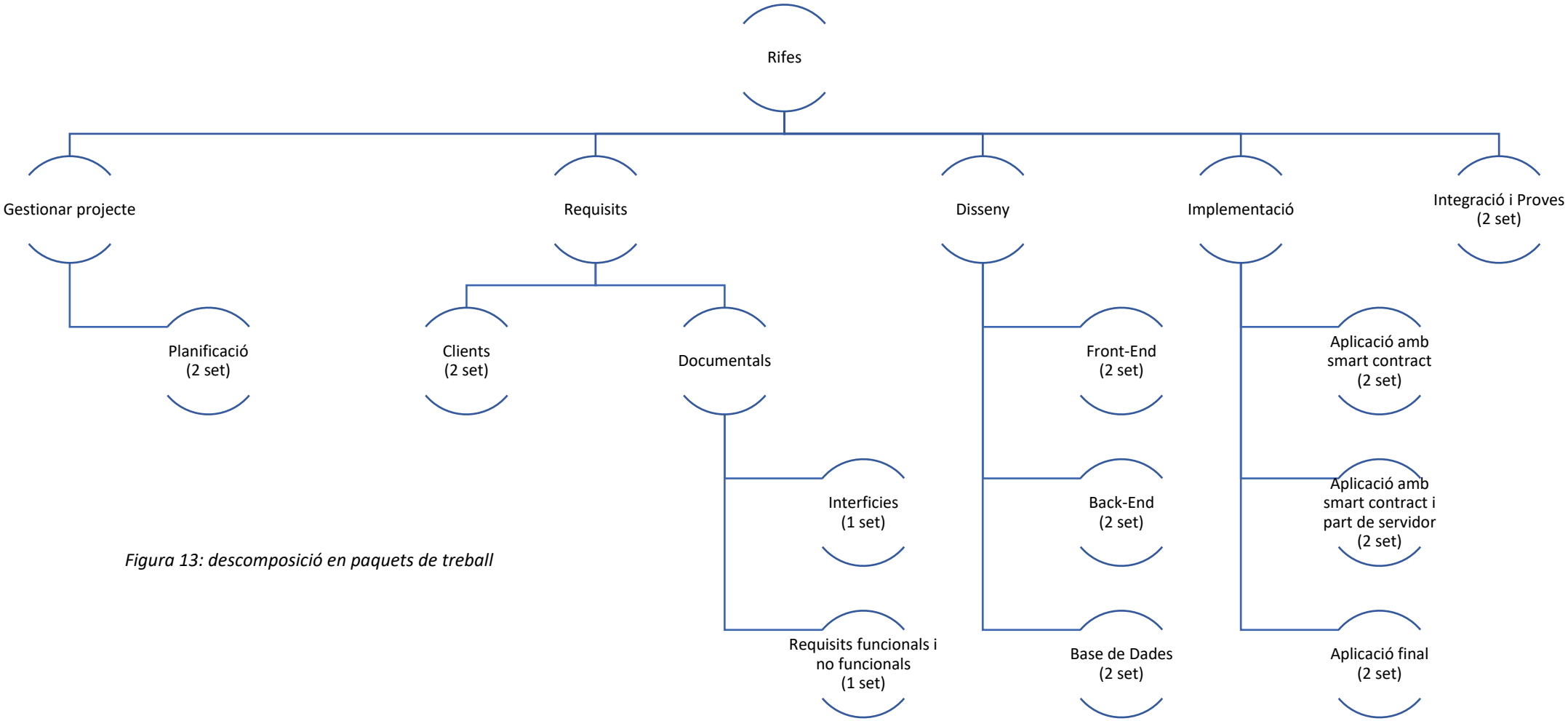


Figura 13: descomposició en paquets de treball

7.1.3.4 MATRIU DE TRACABILITAT

Nom del projecte:	**Rifes Blockchain**					
Nom Tasca	**Codi Tasca**	**Tasques a realitzar**	**Personal implicat**	**Estimació temps**	**Recursos**	**Lliurables**
Planificació	T-1	Crear una base de dades amb la informació necessària	Jo	1 set	Ordinador	Document amb el codi per crear la BD.
		Dissenyar les interfícies	Jo		Ordinador	Document amb el disseny de les interfícies.
Requisits dels clients	T-2.1	Recollir els requisits del client	Jo i el professor	2 set	Ordinador	Document amb els requisits dels clients.
Interfícies	T-2.2.1	Dissenyar les interfícies	Jo	1 set	Ordinador	Document amb el disseny de les interfícies.
Requisits funcionals i no funcionals	T-2.2.1	Recollir els requisits funcionals i no funcionals	Jo i el professor	1 set	Ordinador	Document amb els requisits funcionals i no funcionals.
Front-End	T-3.1	Veure el perfil	Jo	2 set	Ordinador	Document amb el codi per poder veure el perfil.
		Veure la llista de contractes de un usuari	Jo		Ordinador	Document amb el codi per poder veure la llista de contractes d'un usuari.
		Veure la llista de contractes de tots els usuaris	Jo		Ordinador	Document amb el codi per poder veure la llista de contractes de tots els usuaris.

		Editar contracte	Jo		Ordinador	Document amb el codi per poder veure l'edició del contracte.
		Dissenyar les interficies	Jo		Ordinador	Document amb el codi de les interfícies.
Back-End	T-3.2	Crear el smart contract amb totes les funcions	Jo	2 set	Ordinador	Document amb el codi per crear el smart contract.
		Configurar el Blockchain en l'ordinador	Jo		Ordinador	Carpeta amb els codis necessaris per executar el Blockchain.
		Crear sistema de logueig i registre	Jo		Ordinador	Document amb el codi per loguejarte.
		Crear contracte	Jo		Ordinador	Document amb el codi per crear el contracte.
		Afegir detalls del contracte	Jo		Ordinador	Document amb el codi per afegir els detalls del contracte.
		Afegir percentatges del contracte	Jo		Ordinador	Document amb el codi per afegir els percentatges del contracte.
		Comprar tiquets	Jo		Ordinador	Document amb el codi per comprar els tiquets.
		Editar contracte	Jo		Ordinador	Document amb el codi per editar el contracte.
		Acabar contracte	Jo		Ordinador	Document amb el codi per acabar el contracte.
		Logout	Jo		Ordinador	Document amb el codi per des-loguejar-te.

		Enviar missatge al guanyador per reclamar el premi	Jo		Ordinador	Document amb el codi per enviar el missatge.
		Reclamar premi propietari	Jo		Ordinador	Document amb el codi per reclamar el premi del propietari.
		Enviar missatge amb dades de la compra del tiquet	Jo		Ordinador	Document amb el codi per enviar el missatge.
Base de Dades	T-3.3	Crear una base de dades amb la informació necessària	Jo	2 set	Ordinador	Document amb el codi per crear la BD.
Aplicació amb smart contract	T-4.1	Crear el smart contract	Jo	2 set	Ordinador	Document amb el codi per crear el smart contract.
		Configurar el Blockchain en l'ordinador	Jo		Ordinador	Carpeta amb els codis necessaris per executar el Blockchain.
Aplicació amb smart contract i Servidor	T-4.2	Crear contracte	Jo	2 set	Ordinador	Document amb el codi per crear el contracte.
		Afegir detalls del contracte	Jo		Ordinador	Document amb el codi per afegir els detalls del contracte.
		Afegir percentatges del contracte	Jo		Ordinador	Document amb el codi per afegir els percentatges del contracte.
		Comprar tiquet	Jo		Ordinador	Document amb el codi per comprar els tiquets.
		Editar contracte	Jo		Ordinador	Document amb el codi per editar el contracte.

		Acabar contracte	Jo		Ordinador	Document amb el codi per acabar el contracte.
		Sistema de logueig i registrar	Jo		Ordinador	Document amb el codi per loguejarte.
		Crear perfil	Jo		Ordinador	Document amb el codi per poder veure el perfil.
Aplicació final	T-4.3	Veure llista de contractes de un usuari	Jo	2 set	Ordinador	Document amb el codi per poder veure la llista de contractes d'un usuari.
		Participar al contracte	Jo		Ordinador	Document amb el codi per participar al contracte.
		Veure llista de tots els contractes	Jo		Ordinador	Document amb el codi per poder veure la llista de contractes de tots els usuaris.
		Crear logout	Jo		Ordinador	
		Enviar missatge al guanyador per reclamar el premi	Jo		Ordinador	Document amb el codi per enviar el missatge.
		Enviar missatge amb dades al comprar un tiquet	Jo		Ordinador	Document amb el codi per enviar el missatge.
		Reclamar premi propietari	Jo		Ordinador	Document amb el codi per reclamar el premi del propietari.

REQUISITS																				
TASQUES		1	2	3	4	5	6	6.1	6.2	7	8	9	9.1	10	11	11.1	11.2	12	13	14	15	16
	1																		X	X		
	2.1																				X	
	2.2.1																			X		
	2.2.2																					X
	3.1				X	X					X			X						X		
	3.2	X	X	X			X	X	X			X	X	X	X	X	X	X				
	3.3																		X			
	4.1	X	X																			
	4.2			X	X		X	X	X			X		X	X							
	4.3					X				X	X		X			X	X	X				

7.1.3.5 Cronograma – Diagrama de Gantt

Un cop tenim les fitxes de tots els paquets , tenim l'estimació de temps de cada paquet.

Crearem el diagrama de Gantt, agafant els paquets que no depenen un dels altres executant-los al mateix temps i, els paquets que depenen d'ells posant-los conseqüentment. D'aquesta manera tindrem el temps total del projecte.

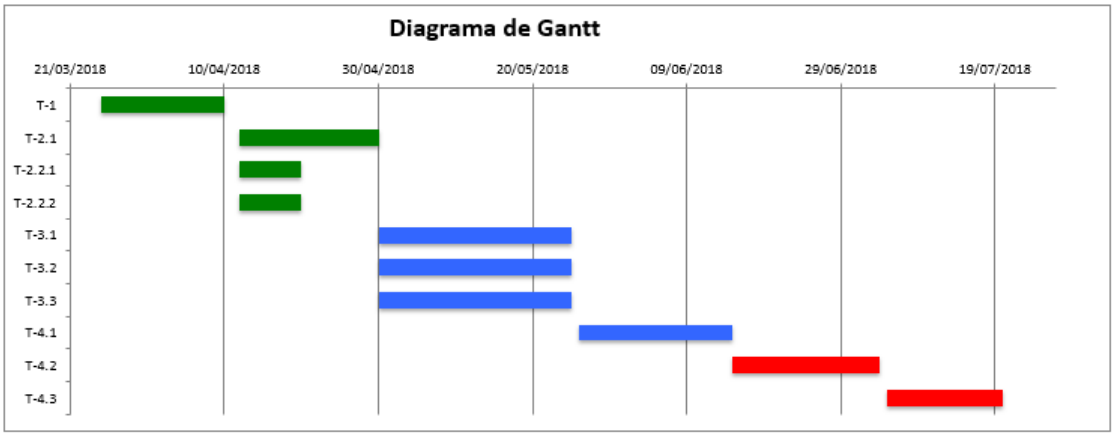


Figura 14: Diagrama de Gantt

7.1.3.6 Pressupost amb punts de funció

Per calcular el pressupost utilitzarem el model dels punts de funció.

La mètrica dels punts de funció permet estimar la mida d'una aplicació independentment del llenguatge de programació i de les tecnologies usades. La mida del sistema es mesura en termes de la quantitat de funcionalitat del Sistema.

El recompte dels punts de funció, deixa alguns aspectes a la interpretació del qui en fa la mesura, podent-se adaptar a la realitat (bases de dades relacionals, nous llenguatges i entorns de programació, noves eines de programació visual, orientació a objectes, etc.).

Es basa en: La identificació dels requisits funcionals del sistema i la seva classificació en cinc tipus:

- Entrades
- Sortides
- Interaccions amb l'usuari
- Interfícies externes
- Arxius usats pel sistema

7.1.3.6.1 PAS 1 – REQUISITS FUNCIONALS

Partint del document de requisits analitzem el nombre de

- LIF. Taules de la base de dades
- EIF: fitxers o taules que l'aplicació usa però no crea ni manté
- EI: entrades a l'aplicació
- EO: sortides que genera l'aplicació
- EQ: consultes interiors, consultes a fitxers o taules pròpies

Cada un es classifica segons el nivell de complexitat

- Baix
- Mig
- Alt

Punts de Funció sense Ajustar			
Tipus	**Descripció**	**Complexitat**	**Total**
EI	Entrades externes	Mitjà: 6 x 4	20
EO	Sortides externes	Mitjà: 5 x 5	25
LIF	Fitxers lògics interns	Baix: 3 x 7	21
EIF	Interfícies lògiques externs	Baix: 11 x 5	55
EQ	Consultes externes	Alt: 4 x 6	20
Total de punts de funció sense ajustar(TUFP)			**141**

7.1.3.6.2 PAS 2 – CARACTERÍSTIQUES GENERALS DEL SISTEMA

Factor ambiental	**Rànquing**
Es requereix comunicació de dades?	4
Existeixen funcions o procediments distribuïts?	3
És crític el rendiment?	3
S'executarà el sistema en un entorn operatiu existent i forçosament utilitzat? Hi ha restriccions de plataforma?	3

El sistema tindrà una carga transaccional alta o baixa?	4
Nivell de disponibilitat	4
Eficiència del usuari final requerit (usabilitat)	5
Actualització en línia	4
Complexitat del processament	4
El sistema ha d'estar dissenyat i implementat per ser reutilitzable?	4
El sistema ha de ser dissenyat per ser fàcil d'instal·lar i de portar?	3
Facilitat d'ús	4
El sistema ha de suportar múltiples instal·lacions en diferents organitzacions?	3
El sistema ha d'estar dissenyat i implementat per facilitar canvis?	4
Total	52

7.1.3.6.3 PAS 3 – CÀLCUL DEL FACTOR D'AJUSTAMENT DE LA COMPLEXITAT

N = 51

CAF = 0,65 + 0,01 * N = 1,17

7.1.3.6.4 PAS 4 – CÀLCUL DELS PUNTS DE FUNCIÓ AJUSTATS

FP = 141, CAF = 1,17

AFP = FP * CAF = 141 * 1,17 = 164,97

7.1.3.6.5 PAS 5 – EQUIVALÈNCIA PF A HORES

Considerant jornades de 8 hores de treball on 5 són productives:

- Hores/Home = AFP * 8 = 164.97 * 8 =1319.76 h

Considerem que de les 8 hores de treball, 5 són productives per 20 dies al mes.

- Dies de treball = Total hores / 5 = 1319.76 / 5 = 263.952 dies aprox. 264 dies.
- Mesos de treball = Dies de treball / 20 = 13,2 mesos

En aquest pressupost cal afegir-li les despeses obtingudes en contractar el servei encarregat d'enviar els serveis. En aquest cas, per fer la demostració, hem contractat una tarifa de 10€ que suposen uns 100 missatges.

7.1.4 Requisits no funcionals

Els requisits no funcionals descriuen els aspectes generals de l'aplicació que no tenen a veure amb les funcionalitats que té el sistema sinó que tenen a veure amb com ha de ser el sistema perquè l'aplicació funcioni correctament.

7.1.4.1 Usabilitat

S'entén per usabilitat a la qualitat d'un programa informàtic que és senzill d'utilitzar perquè facilita la lectura dels texts, descarregant ràpidament la informació i presentant funcions i menús senzills pel qual l'usuari troba còmode el seu ús.

Veiem les característiques que hauran de satisfer les interfícies implementades amb aquest mòdul per tal que el grau d'usabilitat d'aquestes sigui el més alt possible:

- **Intuïtives:** les interfícies hauran de ser el més intuïtives possible, per tal que l'usuari pugui ser capaç d'utilitzar-les sense necessitat d'un apartat d'ajuda específic.
- **Flexibilitat i eficiència:** les interfícies hauran de ser fàcils d'utilitzar per els usuaris.
- **Velocitat:** es permetrà a l'usuari una visualització àgil de les nostres pàgines i els processos de correcció s'implementaran de tal manera que ens retornin el resultat de la correcció en el menor temps possible.
- **Accessibilitat i comprensibilitat:** Les imatges es podran veure de forma nítida al igual que tot el text que aparegui en l'aplicació.
- **Coherència en l'estil:** al dissenyar les interfícies es mantindrà una coherència amb l'estil de l'aplicació. D'aquesta manera l'usuari veurà un estil comú entre totes les pàgines per on passa.

7.1.4.2 Seguretat

La nostra aplicació ens facilita les següents mesures de seguretat:

- Per poder accedir a crear rifes cal estar identificat.
- Es treballa sobre una connexió segura a través del protocol HTTPS. D'aquesta manera la informació que circula a través d'Internet entre el servidor i el client sempre estarà encriptada.

7.1.4.3 Rendiment

La tecnologia utilitzada és NodeJS. EL rendiment amb aquesta tecnologia és molt alt ja que tot el codi es carrega a l'inici de la càrrega de servidor. Les peticions que es generen a l'exercici són *events* i la velocitat és molt alta, per això és recomanable utilitzar aquesta tecnologia.

L'altre tecnologia utilitzada és el Smart contract en Blockchain. En aquest cas, al estar en un Blockchain de tester, el rendiment d'aquesta tecnologia es pot situar en mitjà-alt però si fos en Blockchain real per cada transacció tardaria uns segons més pel qual el rendiment seria una mica més baix situant-se en mitjà.

7.2 REQUISITS DE L'ENTORN

7.2.1 Programari necessari

El sistema ha de complir una sèrie de requisits.
Per la part del Blockchain, per poder utilitzar-lo amb l'aplicació s'ha de tenir instal·lats el *Truffle*, el *TestRPC* i el *NodeJS*.

També cal instal·lar el MIST que instal·la juntament el GETH per poder tenir el Blockchain descargat i sincronitzat i s'haurà de sincronitzar el ROPSTEN. Caldrà tenir unes adreces creades a ROPSTEN per poder-les utilitzar en el TestRPC. En aquest cas, en vaig crear 12.

Per la part de l'aplicació caldrà tenir el MongoDB (amb el servidor inclòs) i el NodeJS amb les llibreries següents:

- **Express:** es el framework web de Node. Proporciona el sistema de rutes que s'utilitza en les peticions HTTP.
- **Path:** serveix per poder treballar amb els paths dels directoris i fitxers.
- **Mongoose:** per connectar els models de la BD amb l'aplicació.
- **Passport, Connect-flash, cookie-parser, Morgan, Express-session:** s'utilitzen per manejar les sessions d'usuaris.
- **Fs, Https:** per implementar el protocol de seguretat HTTPS.

En el NodeJS s'instal·len les dependències següents:

- **Connect-flash, cookie-parser, express-session, Morgan, passaport, fs, express i mongoose:** dependències necessàries per utilitzar les llibreries descrites anteriorment.
- **Bcrypt-nodejs, body-paser, passaport-local:** També necessàries per manejar les sessions d'usuaris.
- **Ejs:** és un senzill llenguatge de plantilla que permet generar un marc de JavaScript senzill en un fitxer HTML.

- **Ganache-cli, truffle, web3:** dependències necessàries per poder utilitzar i crear el smart contract dins l'aplicació i crear el blockchain tester.
- **jQuery:** per poder utilitzar llenguatge jQuery dins el node.
- **Nodemon:** aquesta dependència és util per alhora de desenvolupar, si hi ha qualsevol canvi en el servidor reinici automàticament.
- **text-magic-rest-client:** s'utilitza el Text-Magic per poder enviar missatges des de l'aplicació a un número de telèfon.

7.2.2 Entorn de desenvolupament

Com a eina de desenvolupament es pot utilitzar qualsevol, en aquest cas utilitzaré l'entorn de desenvolupament integrat anomenat ***ATOM***. Aquesta aplicació ens donarà suport per editar el codi, compilar-lo, depurar-lo i executar-lo. *L'ATOM* té disponibles diversos paquets per instal·lar que ajuden a que sigui més còmode el desenvolupament.

7.2.3 El navegador

També serà necessari l'ús d'un navegador web per visualitzar el codi que anem realitzant. En aquest cas, s'utilitzarà el Mozilla Firefox o el Google Chrome.

7.2.4 Proves

Alhora de provar el funcionament del SmartContract he utilitzat el REMIX, una eina òptica i la més utilitzada per compilar i crear Smart Contracts.

8 ANÀLISI I DISSENY DEL SISTEMA

Un cop analitzats els requisits del sistema, cal desenvolupar els diferents diagrames de casos d'ús, les fitxes corresponents i els diagrames d'activitat.

8.1 DIAGRAMES DE CASOS D'ÚS

Els diagrames de casos d'ús ens mostren visualment la comunicació entre els actors i els casos d'ús. Aquests casos descriuen les funcionalitats sense tenir en compte com cal realitzar la implementació de les accions.

8.1.1 Inici

Quan el jugador entra a l'aplicació, pot o bé participar en una rifa o crear-ne una. Per crear una rifa cal identificar-te.

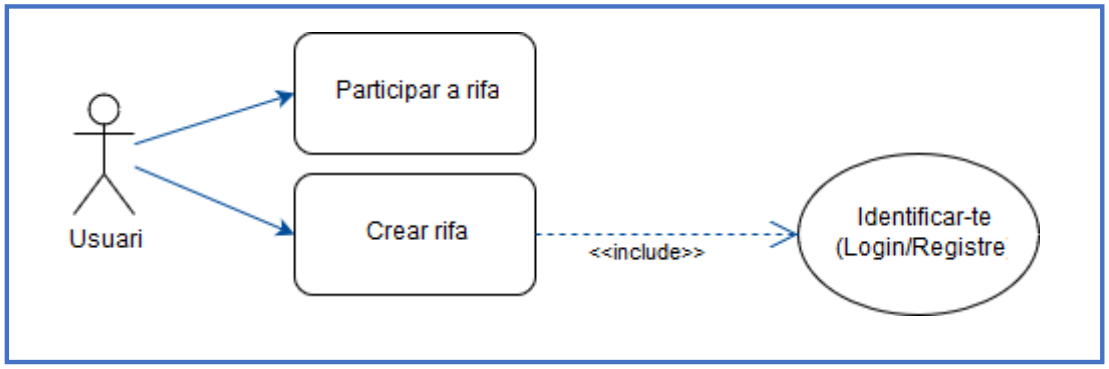


Figura 15: diagrama de casos d'ús - Inici

8.1.2 Login/Registre

Quan el jugador entra en aquest cas d'ús, es carrega la vista de Login. Si no està registrat, cal accedir a l'opció de registre. Un cop loguejat/registrat correctament et redirecciona a "Veure Perfil".

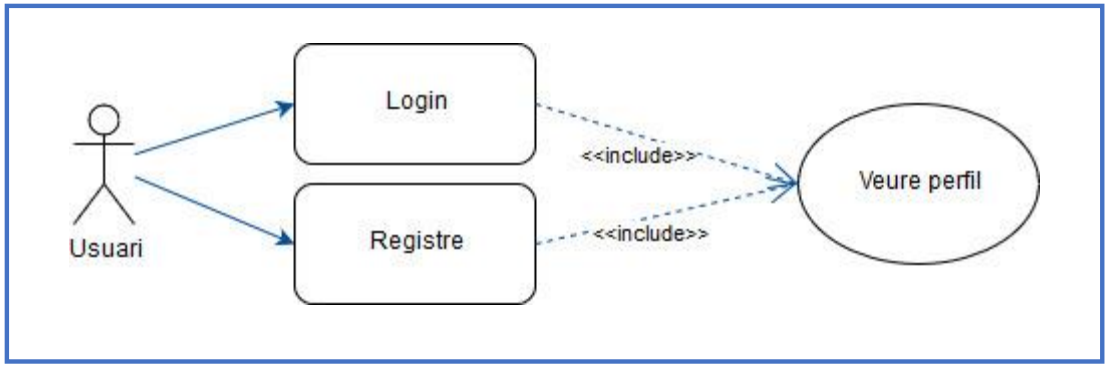


Figura 16: diagrama de casos d'ús – login/registre

8.1.3 Veure perfil

Aquest cas d'ús mostra la pantalla principal de l'usuari registrat. Es mostra les dades del perfil i un botó que mostrarà els contractes de l'usuari.

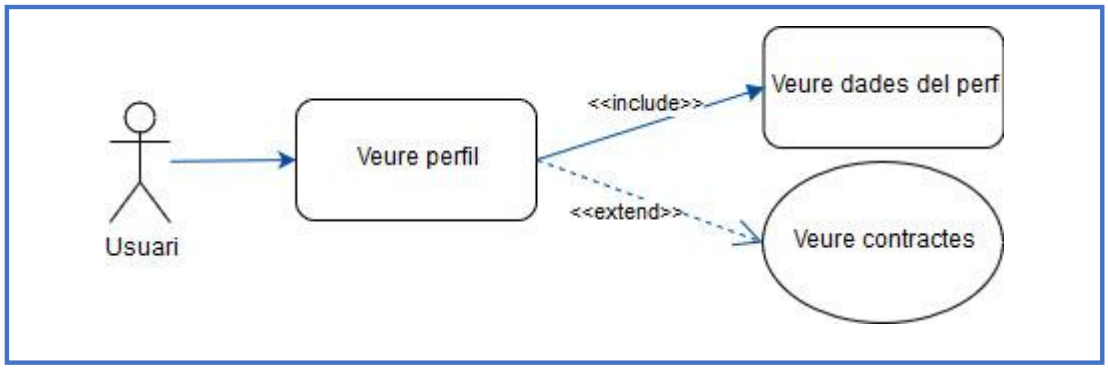


Figura 17: diagrama de casos d'ús – veure perfil

8.1.4 Veure contractes

Aquest cas d'ús mostra un llistat de tots els contractes (rifes) de l'usuari registrat, amb el nom, descripció, participants i accions que es poden fer. Les accions que es poden fer són editar i acabar el contracte. Sota el llistat de contractes, es mostra un botó que crea contractes.

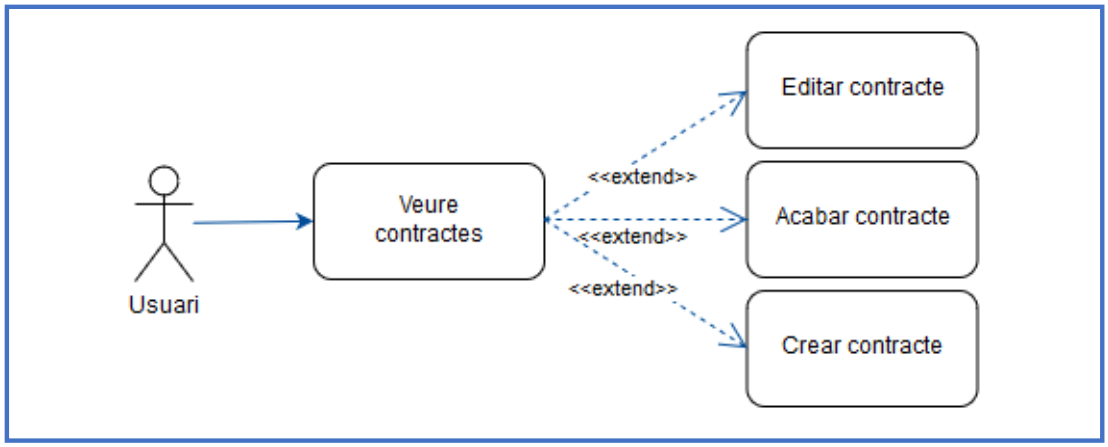


Figura 18: diagrama de casos d'ús – veure contractes

8.1.5 Editar contracte

Aquest cas d'ús mostra una plantilla la qual demana les dades per editar el contracte. Les dades són: màxim de guanyadors, màxim de tiquets i preu per tiquet.

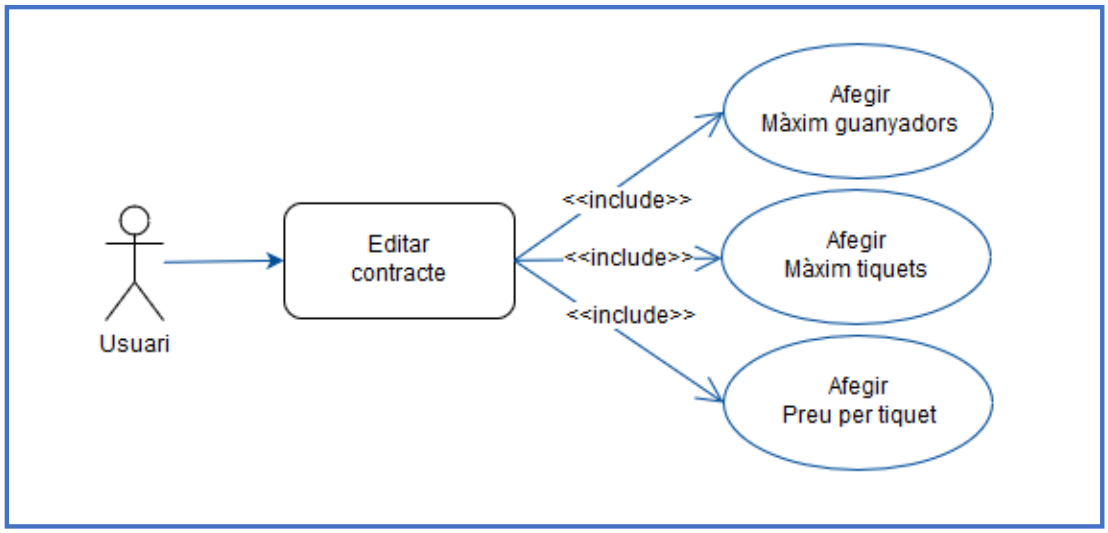


Figura 19: diagrama de casos d'ús – editar contracte

8.1.6 Acabar contracte

Aquest cas d'ús mostra el guanyador del premi. S'envia un missatge al guanyador notificant-lo que vagi a reclamar el premi. Si el propietari de la rifa ha pagat el premi, pot rebre el seu premi.

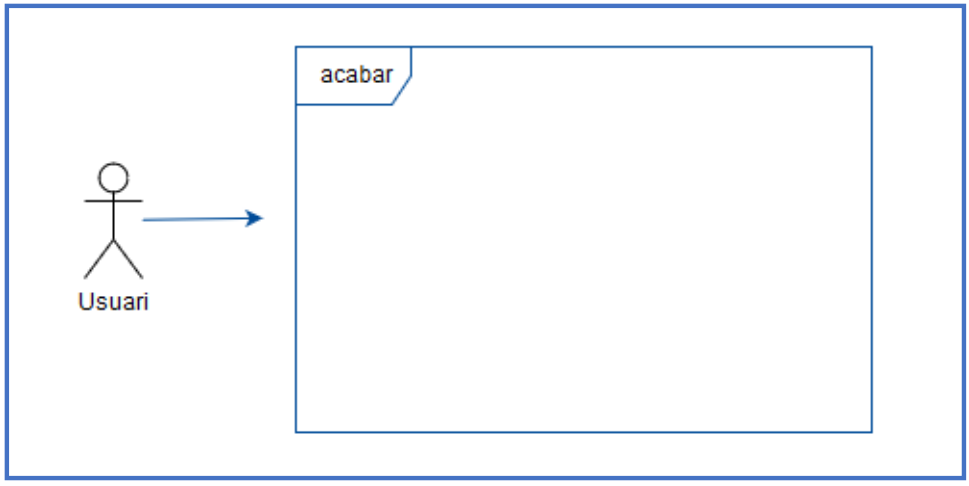


Figura 20: diagrama de casos d'ús – acabar contracte

8.1.7 Crear contracte

Aquest cas d'ús mostra una plantilla la qual demana les dades per crear el contracte. Les dades són: el nom de la rifa, la descripció i l'adreça per les gestions. Un cop afegides, vas a afegir detalls de contracte.

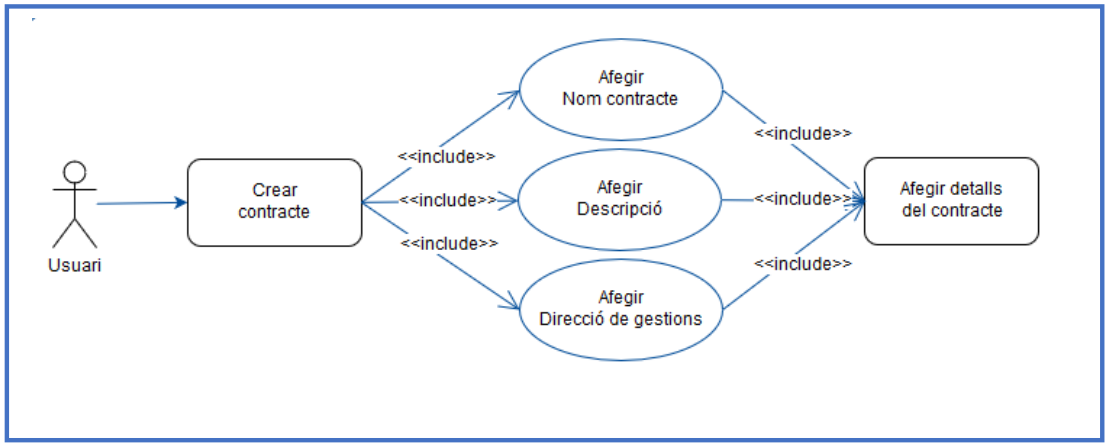


Figura 21: diagrama de casos d'ús – crear contracte

8.1.8 Afegir detalls del contracte

Aquest cas d'ús mostra una plantilla la qual demana les dades a modificar en el contracte.

Les dades són: màxim de guanyadors, màxim de tiquets i preu per tiquet. Un cop afegides, vas a afegir percentatges de contracte.

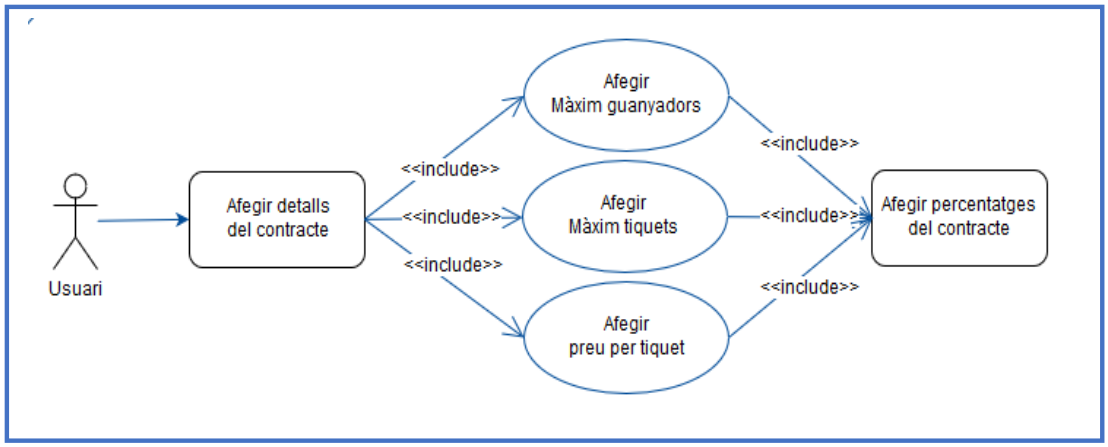


Figura 22: diagrama de casos d'ús – afegir detalls contracte

8.1.9 Afegir percentatges del contracte

Aquest cas d'ús mostra una plantilla la qual demana els percentatges del premi que tindrà cada part en el contracte. Les dades són: percentatge per el propietari i percentatge per el sistema. La resta serà per el guanyador. Un cop afegides, vas al perfil.

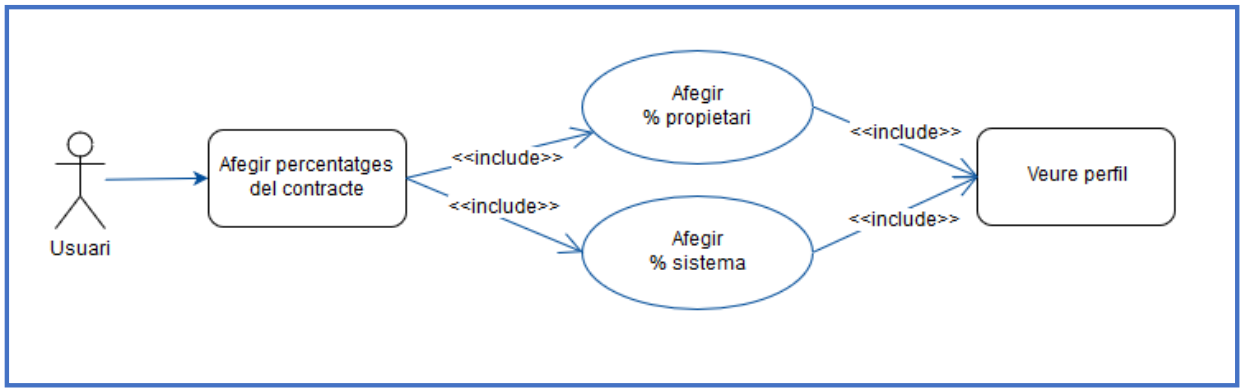


Figura 23: diagrama de casos d'ús – afegir percentatges del contracte

8.1.10 Participar en una rifa

Aquest cas d'ús mostra un llistat de tots els contractes (rifes) que existeixen a l'aplicació, amb el nom, descripció, participants i l'acció que es pot fer. L'única acció que es pot fer és comprar tiquets.

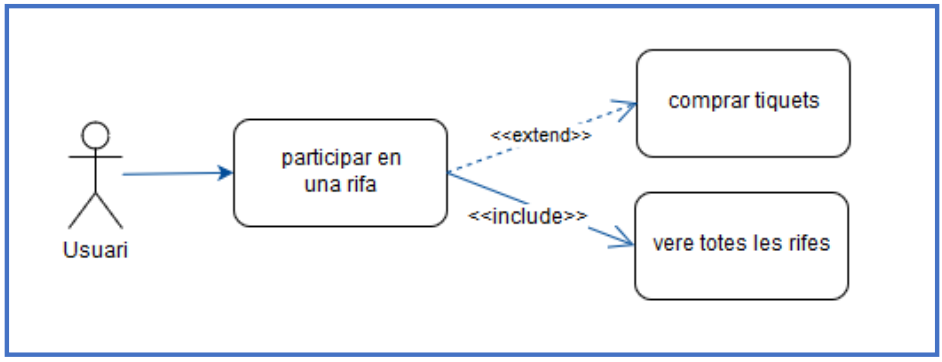


Figura 24: diagrama de casos d'ús – participar en una rifa

8.1.11 Comprar tiquets

Aquest cas d'ús mostra una plantilla la qual demana el número de telèfon de l'usuari i el número de tiquets a comprar. Un cop afegides, rep un missatge amb el token.

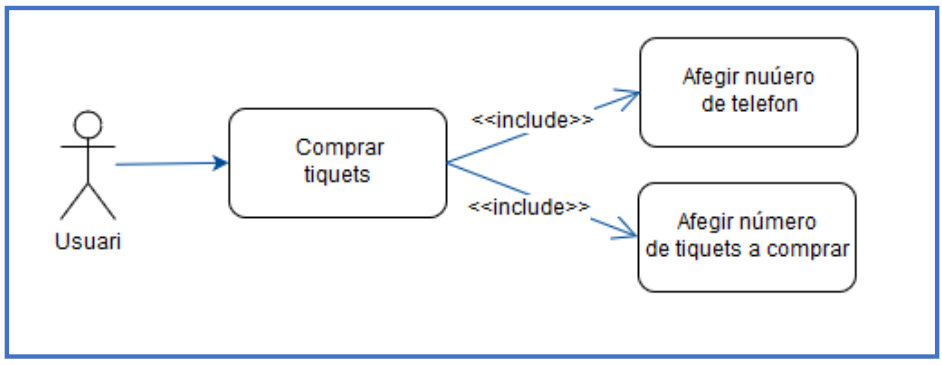


Figura 25: diagrama de casos d'ús – comprar tiquets

8.2 FITXES DE CASOS D'ÚS

Cas d'ús	Inici
Actor	Usuari
Descripció	Redirecciona al login/registre o a participar rifes segons l'opció escollida.
Pre condició	Cert
Fitxa	0. Si es vol participar en una rifa 1. Veure llista de rifes. 2. Si es vol crear una rifa 3. Login/Registre.
Post condició	L'usuari és redireccionat al cas d'ús corresponent.

Cas d'ús	Login/Registre
Actor	Usuari
Descripció	Mostra la pantalla de login/registre i passa al cas d'ús corresponent.
Pre condició	Cert
Fitxa	0. Si està registrat 1. Afegir correu 2. Afegir contrasenya 3. Clicar "Login" 4. Altrament 5. Registre. 6. Si registre o login han acabat correctament 7. Veure perfil. 8. Altrament 9. Tornar al punt 0.
Post condició	L'usuari ha entrat al sistema.

Cas d'ús	Registre
Actor	Usuari
Descripció	Mostra la pantalla de login/registre i passa al cas d'ús corresponent.
Pre condició	Cert
Fitxa	0. Afegir el email. 1. Afegir la contrasenya. 2. Afegir el nom complert. 3. Afegir la data de naixement 4. Afegir el gènere. 5. Afegir l'adreça. 6. Afegir la població 7. Afegir el codi postal. 8. Clicar a "Registre". 9. Si registre ha acabat correctament 10. Veure perfil. 11. Altrament 12. Tornar al punt 0.
Post condició	L'usuari s'ha registrat.

Cas d'ús	Veure contractes
Actor	Usuari
Descripció	Veure contractes del usuari.
Pre condició	Usuari registrat.
Fitxa	13. Anar al perfil. 14. Clicar a "contractes".
Post condició	Mostra els contractes de l'usuari.

Cas d'ús	Editar contracte
Actor	Usuari
Descripció	Editar el contracte canviant els detalls del contracte.
Pre condició	Usuari registrat i contracte sense cap participant.
Fitxa	0. Clicar a "editar" en la rifa que es vol editar. 1. Afegir el màxim de guanyadors. 2. Afegir el màxim de tiquets. 3. Afegir el preu per tiquets. 4. Clicar "Editar". 5. Si no hi ha cap participant inscrit al contracte 6. Es modifica el contracte. 7. Altrament 8. Es mostra un error.
Post condició	Contracte modificat.

Cas d'ús	Acabar contracte
Actor	Usuari
Descripció	Finalitzar el contracte i pagar els premis.
Pre condició	Usuari registrat.
Fitxa	0. Clicar a "acabar" en la rifa que es vol finalitzar. 1. Es mostra el guanyador al que se li ha de pagar el premi. 2. S'envia un missatge al guanyador per reclamar el premi. 3. Si s'ha pagat al guanyador. 4. Clicar " rebre premi".
Post condició	El guanyador, el propietari i el sistema han rebut el premi.

Cas d'ús	Crear contracte
Actor	Usuari
Descripció	Crear contracte (rifa).
Pre condició	Usuari registrat.
Fitxa	0. Afegir el nom del contracte.

	1. Afegir la descripció del contracte. 2. Afegir la direcció de gestions del contracte. 3. Clicar a “Següent”.
Post condició	Contracte creat.

Cas d’ús	**Afegir els detalls del contracte**
Actor	Usuari
Descripció	Afegir els detalls del contracte (max guanyadors, max tiquets, preu per tiquet).
Pre condició	Usuari registrat i contracte creat.
Fitxa	0. Afegir el màxim de guanyadors. 1. Afegir el màxim de tiquets. 2. Afegir el preu per tiquet. 3. Clicar a “Següent”.
Post condició	Detalls del contracte afegits.

Cas d’ús	**Afegir percentatges al contracte**
Actor	Usuari
Descripció	Afegir els percentatges del propietari i del sistema.
Pre condició	Usuari registrat i contracte creat amb els detalls afegits.
Fitxa	0. Afegir el percentatge del premi pel guanyador. 1. Afegir el percentatge del premi pel sistema. 2. Clicar a “Canviar”.
Post condició	Percentatges afegits.

Cas d'ús	Veure llista de rifes
Actor	Usuari
Descripció	Veure la llista de rifes que existeixen
Pre condició	Cert
Fitxa	0. Clicar a "participar rifes".
Post condició	Es mostren les llistes de rifes.

Cas d'ús	Comprar tiquets
Actor	Usuari
Descripció	Comprar tiquets
Pre condició	Cert
Fitxa	0. Clicar a "participar a rifes". 1. Clicar a "comprar tiquet" en la rifa a la que es vol participar. 2. Afegir el número de telèfon de l'usuari. 3. Afegir el número de tiquets que es vol comprar. 4. Clicar a "comprar". 5. S'envia un missatge al mòbil de l'usuari amb el token de la compra. 6. A partir del token, es va a pagar el tiquet a la direcció que et diu el missatge.
Post condició	Tiquets comprats.

8.3 DIAGRAMA D'ACTIVITATS

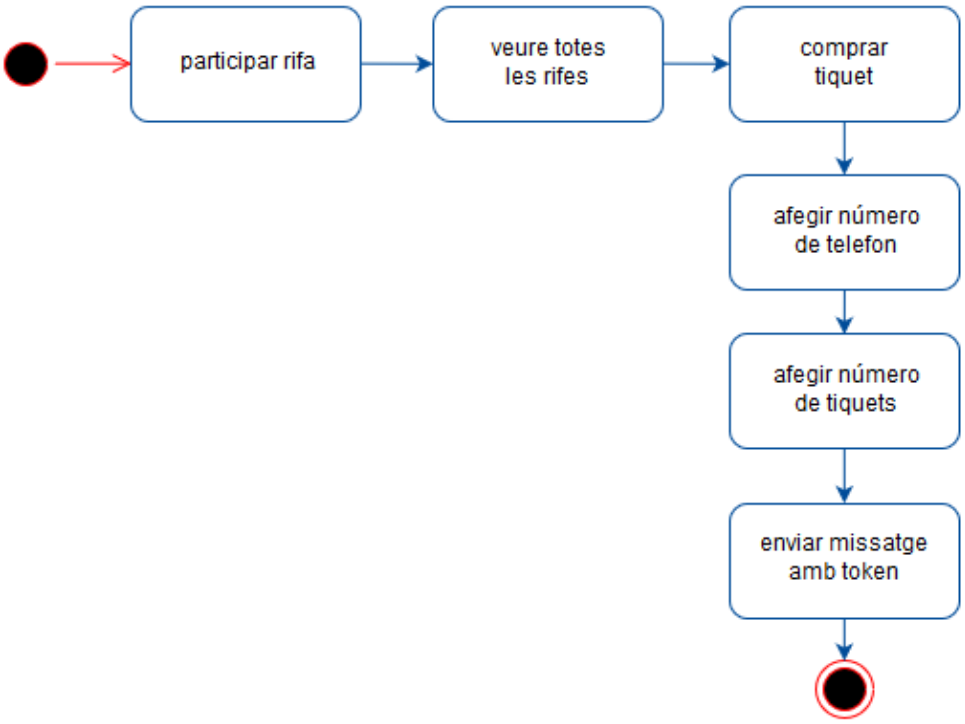


Figura 26: diagrama d'activitats – participar rifa

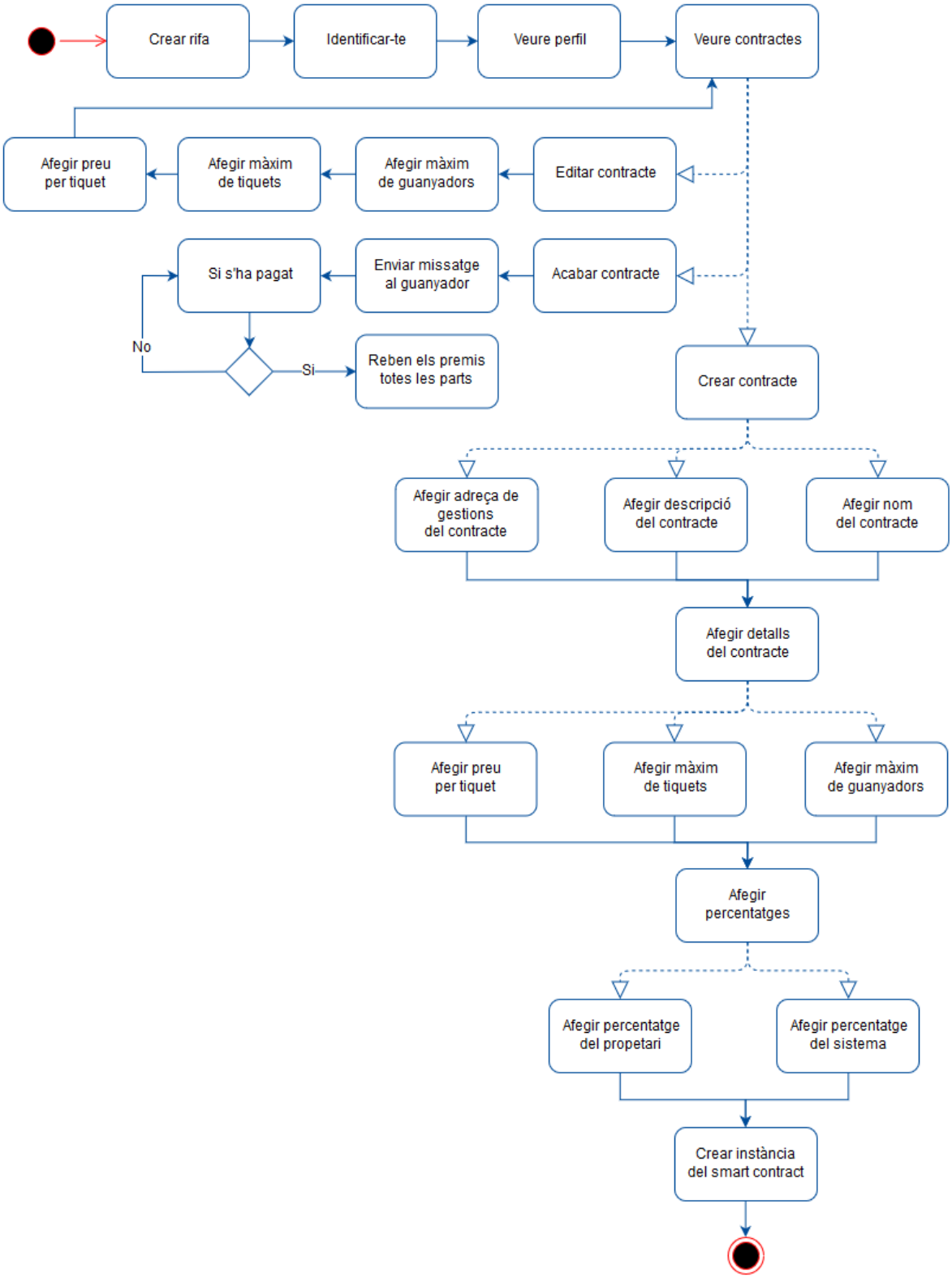


Figura 27: diagrama d'activitats – crear rifa

9 ARQUITECTURA DEL SISTEMA

Com es pot veure en la figura 135, l'arquitectura del sistema, consta de 2 parts:

- **El client:** part que es compon per la interfície de l'aplicació, que es mostra en el navegador escollit.

- **El servidor:** compost per tres parts:

    - **Servidor NodeJS:** servidor de l'aplicació que s'encarrega de llançar les peticions al smart contract instanciat al blockchain.

    - **Base de dades MongoDB:** base de dades que emmagatzema les dades necessàries.

    - **Blockchain:** Instància composta per el web3j que s'encarrega de fer les peticions al smart contract des de l'aplicació i el Blockchain. En el blockchain està el smart contrac

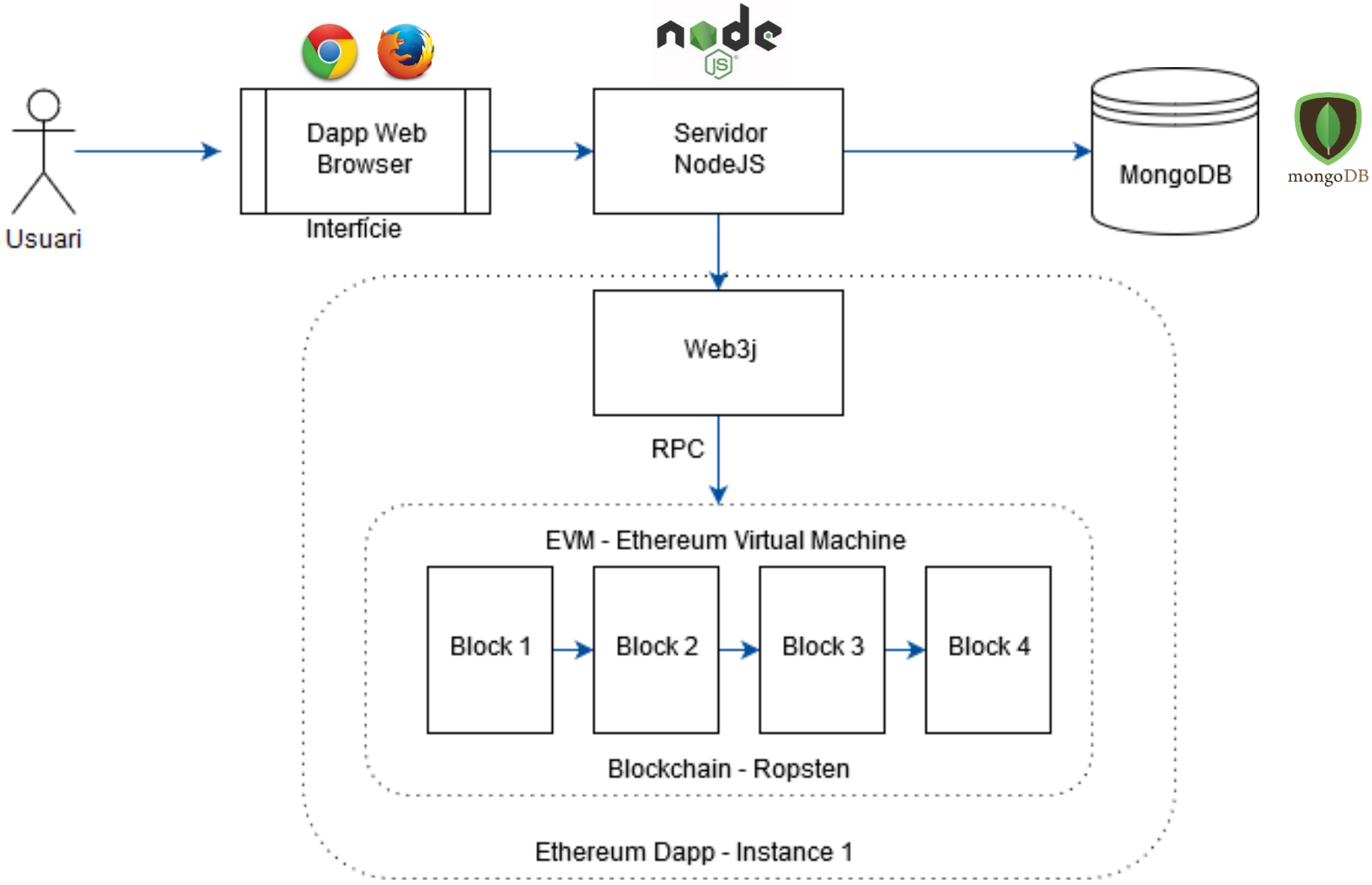


Figura 135: Arquitectura del sistema

10 IMPLEMENTACIÓ I PROVES

En aquesta secció s'explica com s'ha implementat els elements més representatius del projecte.

10.1 BASE DE DADES

La base de dades utilitzada per implementar l'aplicació ha sigut MongoDB. Els models implementats han sigut 4:

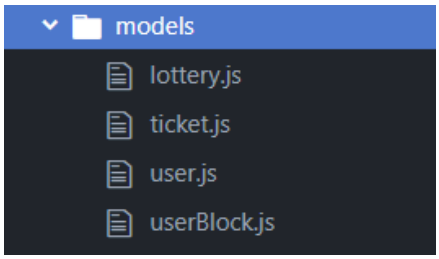


Figura 28: models base de dades

10.1.1 Lottery

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const LotterySchema = Schema({
    index: Number,
    name: String,
    description: String,
    adrecaContact: String,
    userOwner: String,  //email de l'usuari que crea el contracte
    addressContract: String, //adreça del contract
    priceForTicket: Number,
    pagat: {
      type: Boolean,
      default: false
    },
    userParticipants: [
      {
        userReal: Number, //usuari que crea el contracte - el seu numero de telefon
        userBlock: Number, //enter que identifica la posició de l'array d'usuaris en el blockchain
        userAddressBlock: String,
        addressContract: String,
        numTickets: [ { type: mongoose.Schema.Types.ObjectId, ref: "Ticket"} ]
      } ] //usuaris participants de
});

module.exports = mongoose.model("Lottery", LotterySchema);
```

Figura 29: models BD - lottery

És el model encarregat de guardar les rifes a la base de dades. Té els següents atributs:

- **Index:** index que identifica a la rifa.
- **name:** nom de la rifa
- **description:** descripció de la rifa
- **adrecaContract:** adreça de gestions de la rifa
- **userOwner:** usuari creador de la rifa
- **addressContract:** adreça del smart contract que identifica a la rifa
- **priceForTicket:** preu per tiquet de la rifa
- **pagat:** identifica si s'ha pagat o no al guanyador
- **userParticipants:** llista de usuaris participants a la rifa.

10.1.2 Ticket

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const TicketSchema = Schema({
    numTickets: Number, //nombre de tickets que te l'usuari
    addressContract: String, //adreça del contracte on es tenen els tickets
    userReal: Number
});

module.exports = mongoose.model("Ticket",TicketSchema);
```

Figura 30: models BD - ticket

És el model encarregat de guardar els tiquets que es compren a la base de dades. Té els següents atributs:

- **numTickets:** número de tiquets que s'han comprat
- **addressContract:** adreça del smart contract que identifica a la rifa
- **userReal:** telèfon de l'usuari que ha comprat el tiquet.

10.1.3 User

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt-nodejs');
const Schema = mongoose.Schema;

const UserSchema = Schema({
  local: {
    //name: String, //nom del usuari
    email: String,
    password: String,
    name: String,
    dateBirth: String,
    telephone: String,
    gender: String,
    carrer: String,
    codipostal: String,
    poblacio: String,

    tickets: [ { type: mongoose.Schema.Types.ObjectId, ref: "Ticket"} ] //Lista de Tickets
  }
});

//encripta la contransenya
UserSchema.methods.generateHash = function (password) {
  return bcrypt.hashSync(password, bcrypt.genSaltSync(8),null);
}

//compara la contrasenya amb la que et loguejes amb la guardada a la BD
UserSchema.methods.validatePassword = function (password) {
  return bcrypt.compareSync(password,this.local.password);
}

module.exports = mongoose.model("User", UserSchema);
```

Figura 31: models BD - user

És el model encarregat de guardar els usuaris a la base de dades. Té els següents atributs:

- **email:** correu de l'usuari.
- **password:** contrasenya de l'usuari.
- **name:** nom complert de l'usuari.
- **dateBirth:** data de naixement de l'usuari amb format (DD/MM/YYYY).
- **telephone:** número de telèfon de l'usuari
- **gender:** gènere de l'usuari
- **carrer:** adreça de l'usuari
- **codipostal:** codi postal de l'usuari

- **població:** població de l'usuari

10.1.4 UserBlock

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const UserBlockSchema = Schema({
    userReal: Number, //usuari que crea el contracte - el seu numero de telefon
    userBlock: Number, //enter que identifica la posició de l'array d'usuaris en el blockchain
    userAddressBlock: String,
    addressContract: String
});

module.exports = mongoose.model("UserBlock", UserBlockSchema);
```

Figura 32: models BD – userblock

És el model encarregat de guardar els usuaris participants de les rifes a la base de dades. Té els següents atributs:

- userReal: el telèfon de l'usuari participant a la rifa.
- userBlock: index que marca la posició de l'usuari en el vector d'addresses del blockchain.
- userAddressBlock: adreça que identifica l'usuari en el smart contract.
- addressContract: adreça del smart contract que identifica a la rifa.

10.2 SERVIDOR

ESTRUCTURA DE DIRECTORIS

Observant la figura 33 es pot veure l'estructura de directoris del projecte del servidor.

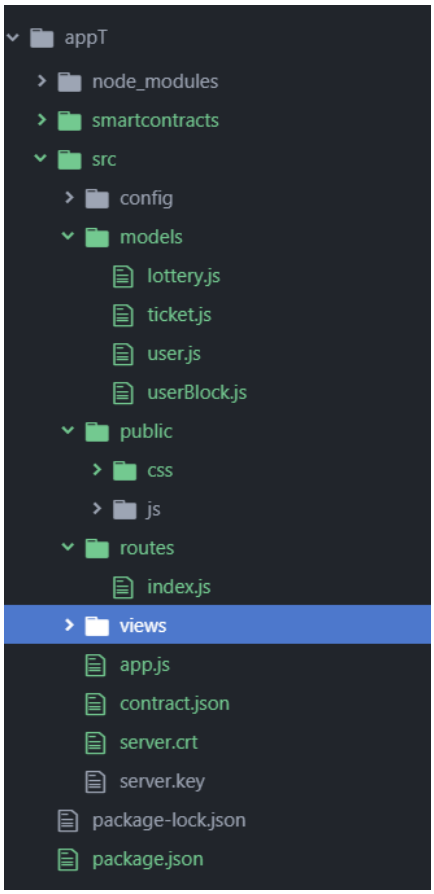


Figura 33: estructura de directoris del servidor

- **Node_modules:** carpeta on s'instal·len tots els mòduls necessaris.
- **Smartcontract:** carpeta que conté el smart contract creat.
- **Config:** carpeta que conté les configuracions de les sessions i de la base de dades.
- **Models:** dins aquesta carpeta hi ha els models de la BD que utilitza el moongose dins l'aplicació.
- **Public:** carpeta on està guardada l'estructura de directoris del client.
- **Routes:** carpeta on hi ha els mòduls que apliquen un seguit de funcions relacionades amb algun objecte de la base de dades.

- **App.js** és l'executable de node i fa el set de les primeres variables del servidor.
- **Package.json:** inclou el json amb les llibreries necessàries per al projecte i que el npm les pugui instal·lar automàticament.

10.3 FUNCIONALITATS

En aquest apartat explicaré els objectes participants en cada funcionalitat i com s'han implantat.

10.3.1 Crear contracte

Per crear un contracte, haurem de tenir primer creat el smart contract.

10.3.1.1 Smart Contract

El smart contract el vaig crear en el REMIX, editor utilitzat per crear smart contract que et proporciona un compilador i els detalls del smart contract necessaris per desenvolupar el smart contract a l'aplicació.

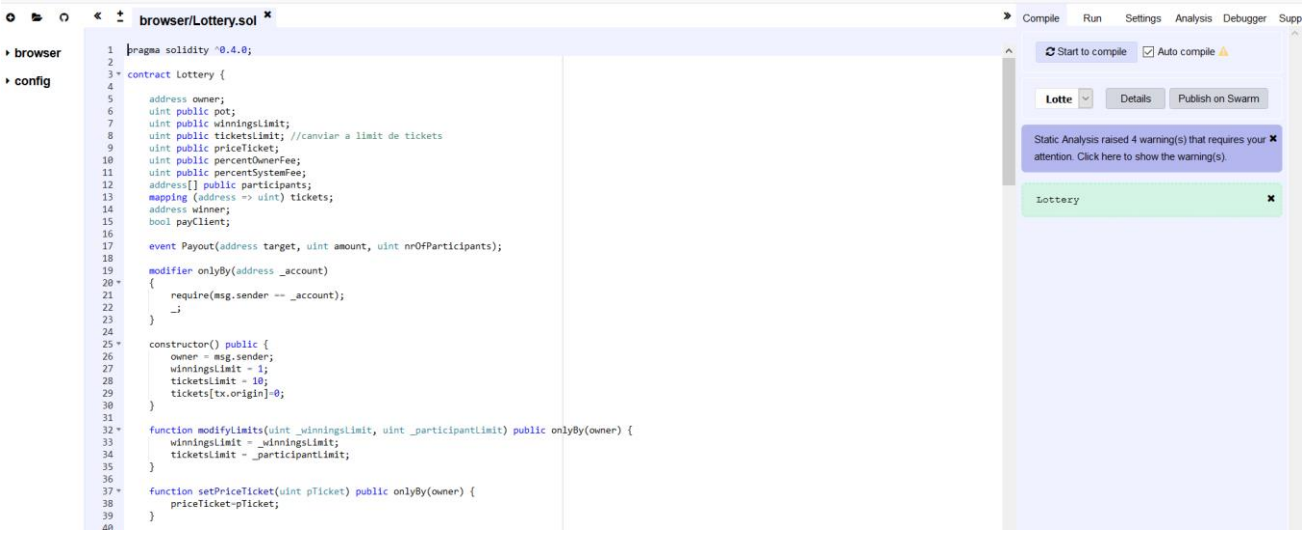


Figura 34: Remix amb el smart contract creat

10.3.1.1.1 Atributs

El smart contract té els següents atributs:

- **address owner:** propietari del contracte.
- **uint pot:** saldo en el contracte.
- **uint winningsLimit:** màxim de guanyadors del contracte.
- **uint ticketsLimit:** màxim de tiquets del contracte.
- **uint priceTicket:** preu per tiquet del contracte.
- **uint percentOwnerFee:** percentatge que s'emporta el propietari del premi.
- **uint percentSystemFee:** percentatge que s'emporta el sistema del premi.
- **address[] participants:** llista d'adreces dels participants.
- **mapping (address => uint) tickets:** mapeig de les adreces disponibles amb enters que corresponen al número de tiquets dels que disposa.
- **address winner:** adreça del guanyador del contracte.
- **bool payClient:** identifica si el guanyador ha rebut la seva part del premi.

10.3.1.1.2 Funcions

10.3.1.1.2.1 event Payout(address target, uint amount, uint nrOfParticipants)

El Solidity no té cap funció que mostri per pantalla cap sortida, de manera que es necessàri un event per mostrar els elements necessaris alhora de testejar el contracte.

10.3.1.1.2.2 Modifier onlyBy(address _account)

```
modifier onlyBy(address _account)
{
    require(msg.sender == _account);
    _;
}
```

Figura 35: Funció onlyBy en el smart contract

S'utilitza per exigir a les funcions que només l'usuari account pugui executar la funció.

10.3.1.1.2.3 Constructor()

```
constructor() public {
    owner = msg.sender;
    winningsLimit = 1;
    ticketsLimit = 10;
    tickets[tx.origin]=0;
}
```

Figura 36: Funció constructor en el smart contract

Funció que una instància del smart contract. L'adreça que crida la funció s'afegeix com l'usuari owner. Es declaren per defecte els màxims dels guanyadors i de tiquets amb 1 i 10 respectivament i, el map de tiquets amb tots els elements per 0.

10.3.1.1.2.4 function modifyLimits(uint _winningsLimit, uint _participantLimit) public onlyBy(owner)

```
function modifyLimits(uint _winningsLimit, uint _participantLimit) public onlyBy(owner) {
    winningsLimit = _winningsLimit;
    ticketsLimit = _participantLimit;
}
```

Figura 37: Funció modifyLimits en el smart contract

Funció que modifica els límits de guanyadors i tiquets per els que et passen a la funció. Requereix que qui cridi la funció sigui el owner del contracte.

10.3.1.1.2.5 function setPriceTicket(uint pTicket) public onlyBy(owner)

```
function modifyLimits(uint _winningsLimit, uint _participantLimit) public onlyBy(owner) {
    winningsLimit = _winningsLimit;
    ticketsLimit = _participantLimit;
}
```

Figura 38: Funció setPriceTicket en el smart contract

Funció que modifica el preu de tiquet per nTicket. Requereix que qui cridi la funció sigui el owner del contracte.

10.3.1.1.2.6 function setFeeOwner(uint fee) public onlyBy(owner)

```
function setPriceTicket(uint pTicket) public onlyBy(owner) {
    priceTicket=pTicket;
}
```

Figura 39: Funció setFeeOwner en el smart contract

Funció que modifica el percentatge que s'emportarà el propietari. Requereix que qui cridi la funció sigui el owner del contracte.

10.3.1.1.2.7 function setFeeSystem(uint fee) public onlyBy(owner)

```
function setFeeOwner(uint fee) public onlyBy(owner) {
    if (fee > 60) revert();
    else percentOwnerFee=fee;
}
```

Figura 40: Funció setFeSystem en el smart contract

Funció que modifica el percentatge que s'emportarà el sistema. Requereix que qui cridi la funció sigui el owner del contracte.

10.3.1.1.2.8 function payLottery() payable públic

```
function payLottery() payable public {
    if(msg.value!=priceTicket) revert();
    else {
        participants.push(msg.sender);
        tickets[msg.sender]++;
        pot += msg.value;
        if (participants.length == ticketsLimit) {
            winner = terminate();
        }
    }
}
```

Figura 41: Funció payLottery en el smart contract

Funció amb el que l'usuari compra tiquets .

10.3.1.1.2.9 function getPot() public view returns (uint)

```
function getPot() public view returns (uint)  {
    return pot;
}
```

Figura 42: Funció getPot en el smart contract

Funció que retorna el saldo del contracte.

10.3.1.1.2.10 Function getWinLimits() public view returns (uint)

```
function getWinLimits() public view returns (uint) {
    return winningsLimit;
}
```

Figura 43: Funció getWinLimits en el smart contract

Funció que retorna el màxim de guanyadors del contracte.

10.3.1.1.2.11 function getTickLimits() public view returns (uint)

```
function getTickLimits() public view returns (uint) {
    return ticketsLimit;
}
```

Figura 44: Funció getTickLimits en el smart contract

Funció que retorna el màxim de tiquets del contracte.

10.3.1.1.2.12 function getTickets(address participantAddress) public view returns (uint)

```
function getTickets(address participantAddress) public view returns (uint) {
    return tickets[participantAddress];
}
```

Figura 45: Funció getTickets en el smart contract

Funció que retorna el tiquets que té un participant amb adreça *participantAddress* en el contracte.

10.3.1.1.2.13 function getOwner() public view returns (address)

```
function getOwner() public view returns (address) {
    return owner;
}
```

Figura 46: Funció getOwner en el smart contract

Funció que retorna el propietari del contracte.

10.3.1.1.2.14 function getWinner() public view returns (address)

```
function getWinner() public view returns (address) {
    return winner;
}
```

Figura 47: Funció getWinner en el smart contract

Funció que retorna el guanyador del contracte.

10.3.1.1.2.15 function getPriceTicket() public view returns (uint)

```
function getPriceTicket() public view returns (uint) {
    return priceTicket;
}
```

Figura 48: Funció getPriceTIcket en el smart contract

Funció que retorna el preu del tiquet del contracte.

10.3.1.1.2.16 function getOwnerFee() public view returns (uint)

```
function getOwnerFee() public view returns (uint) {
    return percentOwnerFee;
}
```

Figura 49: Funció getOwnerFee en el smart contract

Funció que retorna el percentatge del premi que obtindrà el propietari del contracte.

10.3.1.1.2.17 function getSystemFee() public view returns (uint)

```
function getSystemFee() public view returns (uint) {
    return percentSystemFee;
}
```

Figura 50: Funció getSystemFee en el smart contract

Funció que retorna el percentatge del premi que obtindrà el propietari del contracte.

10.3.1.1.2.18 function payPot() public

```
function payPot() public {

    address system = 0xfF1FbFe19950b12ACe5101dE1459d765e062Cb59;
    uint totalPayout = pot;
    uint ownerFee = 0;
    uint systemFee = 0;
    uint payoutToWinner = 0;

    if(pot>0) {
        //ownerFee = totalPayout / 20; //5%
        ownerFee = totalPayout * (percentOwnerFee/ 100);
        systemFee = totalPayout * (percentSystemFee / 100);
        payoutToWinner = totalPayout - ownerFee - systemFee;
    }
    if(participants.length > 0 && pot>0 && payClient==true) {
        winner.transfer(payoutToWinner);
        owner.transfer(ownerFee);
        system.transfer(systemFee);
    }
}
```

Figura 51: Funció payPot en el smart contract

Funció que al finalitzar el contracte, reparteix el premi.

10.3.1.1.2.19 function terminate() private returns (address)

```
function terminate() private returns (address){
    address winnerPart = owner;
    payClient=false;
    if(participants.length > 0 && pot>0) {
        uint winnerIndex = 0;
        winnerIndex = uint(blockhash(block.number-1)) % (participants.length);
        if(winnerIndex==participants.length) winnerIndex--;
        winnerPart = participants[winnerIndex];

    }
    emit Payout(winnerPart, pot, winnerIndex);
    return winnerPart;
}
```

Figura 52: Funció terminate en el smart contract

Funció que al finalitzar, fa un random i identifica el guanyador.

10.3.1.1.2.20 function payoutClient() públic

```
function payoutClient() public {
    payClient=true;
}
```

Figura 53: Funció payoutClient en el smart contract

Funció que posa payClient a true per identificar que el guanyador ha rebut els diners.

10.3.1.1.2.21 function murder() public onlyBy(owner)

```
function murder() public onlyBy(owner) {
    winner = terminate();
}
```

Figura 54: Funció murder en el smart contract

Funció que acaba el contracte.

10.3.1.1.2.22 function deleteContract() public onlyBy(owner)

```
function deleteContract() public onlyBy(owner) {
    selfdestruct(owner);
}
```

Figura 55: Funció deleteContract en el smart contract

Funció que destrueix el contracte.

10.3.1.1.2.23 function getContractAddress() constant public returns (address)

```
function getContractAddress() constant public returns (address){
    return this;
}
```

Figura 56: Funció getCOntractAddress en el smart contract

Funció que retorna l'adreça del contracte.

10.3.1.2 Index.js

10.3.1.2.1 GET /crear_contracte/:id

```
/*
  des de la llista de contractes es vol crear un contracte.
  Envia a html que demana el nom, la descripcio i "el usuari" **********!!!!!!!!!!!!!!
*/
app.get('/crear_contracte/:id', async (req,res) => {

  const { id } = req.params;
  const user = await User.findById(id);
  res.render('crear_contracte', {
    user: user
  });
});
```

Figura 57: Ruta GET crear_contracte en el fitxer index.js

Aquesta ruta ens redirigeix a la pàgina html crear_contracte enviant-li l'usuari propietari.

10.3.1.2.2 POST /crear_contracte/:id

```
async function crearContracte(id) {
  var web3 = new Web3(new Web3.providers.HttpProvider(
    'http://localhost:8545'
  ));

  //es declara el contracte
  var lotteryContract =
  web3.eth.contract([{"constant":true,"inputs":[{"name":"participantAddress","type":"address"}],"name":"getTickets","outp
  ew","type":"function"},{"constant":true,"inputs":[],"name":"getContractAddress","outputs":[{"name":"","type":"address"
  ","type":"function"},{"constant":false,"inputs":[{"name":"pTicket","type":"uint256"}],"name":"setPriceTicket","outputs
  ayable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"getPriceTicket","output
  nt":true,"inputs":[],"name":"priceTicket","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"
  outputs":[{"name":"","type":"address"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"
  ity":"view","type":"function"},{"constant":false,"inputs":[{"name":"fee","type":"uint256"}],"name":"setFeeSystem","outp
  nction"},{"constant":true,"inputs":[],"name":"percentOwnerFee","outputs":[{"name":"","type":"uint256"}],"payable":fals
  256"},{"indexed":false,"name":"nrOfParticipants","type":"uint256"}],"name":"Payout","type":"event"}]);

  //es crea un nou contracte
  var lottery = lotteryContract.new(
   {
    from: web3.eth.accounts[0],
    data:
     '0x60806040523480156100105760008054600160a060020a0319163317815560016002556006003553281526008602052604081
     5c5780635a58cd4c1461027157806364c16534146102865780637b5406714610295b57806378b91d4a146102b0578063893d20e8146102c557806
     56101ab57600080fd5b506101b46103a2565b6040805160160a060020a0390216825251908190036020019f35b3480156101dc57600080fd5
     4e565b3480156102a757600080fd5b5061016361045565b3480156102bc57600080fd5b5061016361045a565b3480156102d157600080fd5b5
     0409020549056 5b60025490565b60065490565b3090565b60078054829081106103b457fe5b60009182526020909120015460160a060020a03
     8c57600080fd5b6007805460018181010183557fa66cc928b5edb82af9bd49922954155ab7b0942694bea4ce44661d9a8736c688909101805473f
     6338114610588576000 80fd5b603c8211156105965760008 0fd5b506005555565b60055481565b60005460160a060020a031633811461 05ba57
     7957600019015b6007805482908110610685757fe5b600091825260208220015460405160160a060020a0390911698508891845156108fc029181
     a1a405ec65f6f6c08f55e1f379c2c45cd10f23cb29e0e31918190036060019 0a150949695505050505050565 0a165627a7a7230582017da0fbc
     gas: '4700000000'
   }, async function (e, contract){
    console.log(e, contract);
    if ( typeof contract.address !== 'undefined') {
        console.log('Contract mined! address: ' + contract.address + ' transactionHash: ' + contract.transactionHash);
        //const lotte =  Lottery.findById(id);
        console.log(id);
        var a = contract.address;
        await Lottery.findOneAndUpdate(
          {'_id': id},
          {'addressContract': a},
          {'upsert': true}
        );

    }
  });
```

Figura 136: Funció crearContracte en el fitxer index.js

```
/*
  Arriba de la pàgina crear_crontacte, amb els elements  {nom, la descripcio} en el req.body
  i el id del usuari passat per parametre en el req.params.
  Busca l'usuari i crea la loteria.
  Passa a la pàgina contract l'usuari i la lotteria.
  !!!!!!!!!!!! Cal que afegeixi directament l'usuari enlloc de demanar-lo !!!!!!!!!!!!!!!!
*/
app.post('/crear_contracte/:id', async (req, res) => {
  const { id } = req.params;
  const user = await User.findById(id);
  var i = await Lottery.count({}, function(err,count){
    if(err) console.log(err);
  });
  var lot = new Lottery({
    'name': req.body.name,
    'description': req.body.description,
    'adrecaContact': req.body.adreca_gestio,
    'userOwner': user.local.email,
    'index': i
  });
  await lot.save();

  var tm = await Lottery.find({index: i}).select('_id').exec(function(err,docs){
    docs = docs.map(function(doc) {
      return doc._id;
    });
    console.log("lotte id:" + docs[0]);
    crearContracte(docs[0]);
  });

  res.render('contract', {
    user : user,
    lote: lot
  });

});
```

Figura 58: Ruta POST crear_contracte en el fitxer index.js

Aquesta ruta crea una rifa a la base de dades amb les dades aportades des del fitxer ejs i executa la funció *crearContracte*() que crea una instància del smart contract i afegeix l'adreça del smart contract a la rifa creada.

10.3.1.3 Crear_contracte.ejs

Consta de 3 inputs dins un form demanant el *nom*, la *descripció* i *l'adreça de gestions* del contracte (rifa). El *form* crida l'acció POST */crear_contracte/* passant-li per paràmetre *l'id* de l'usuari.

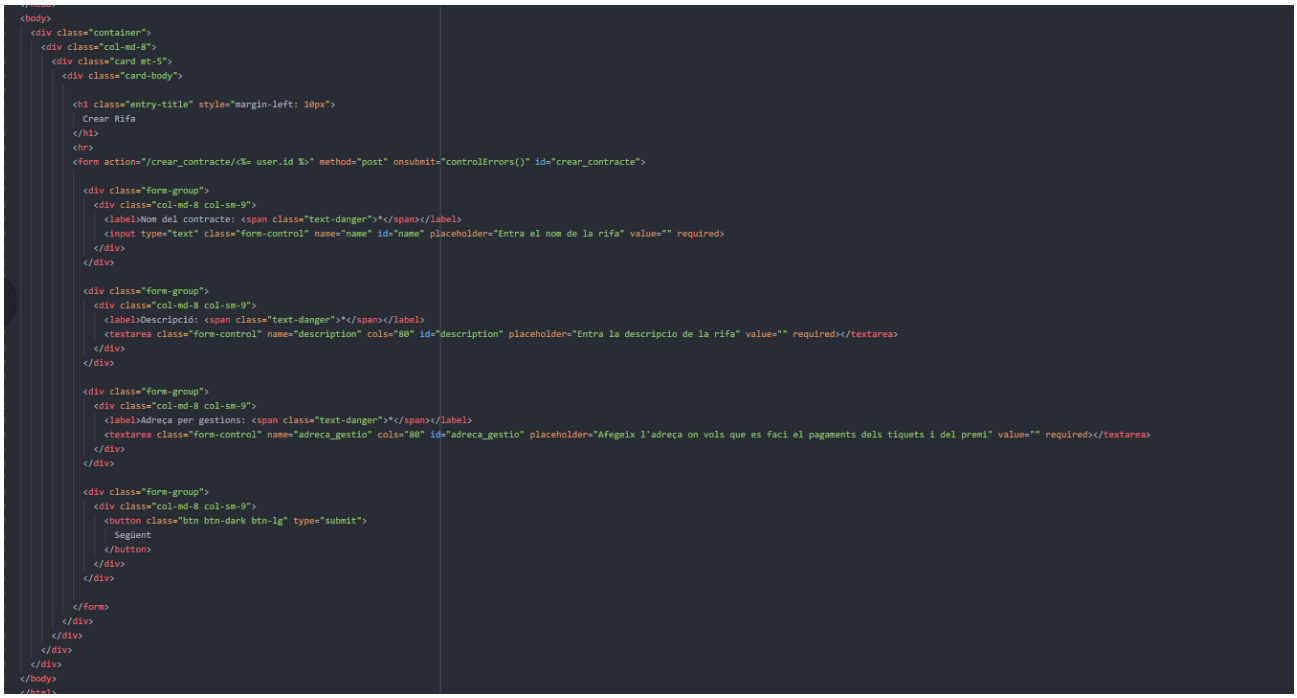


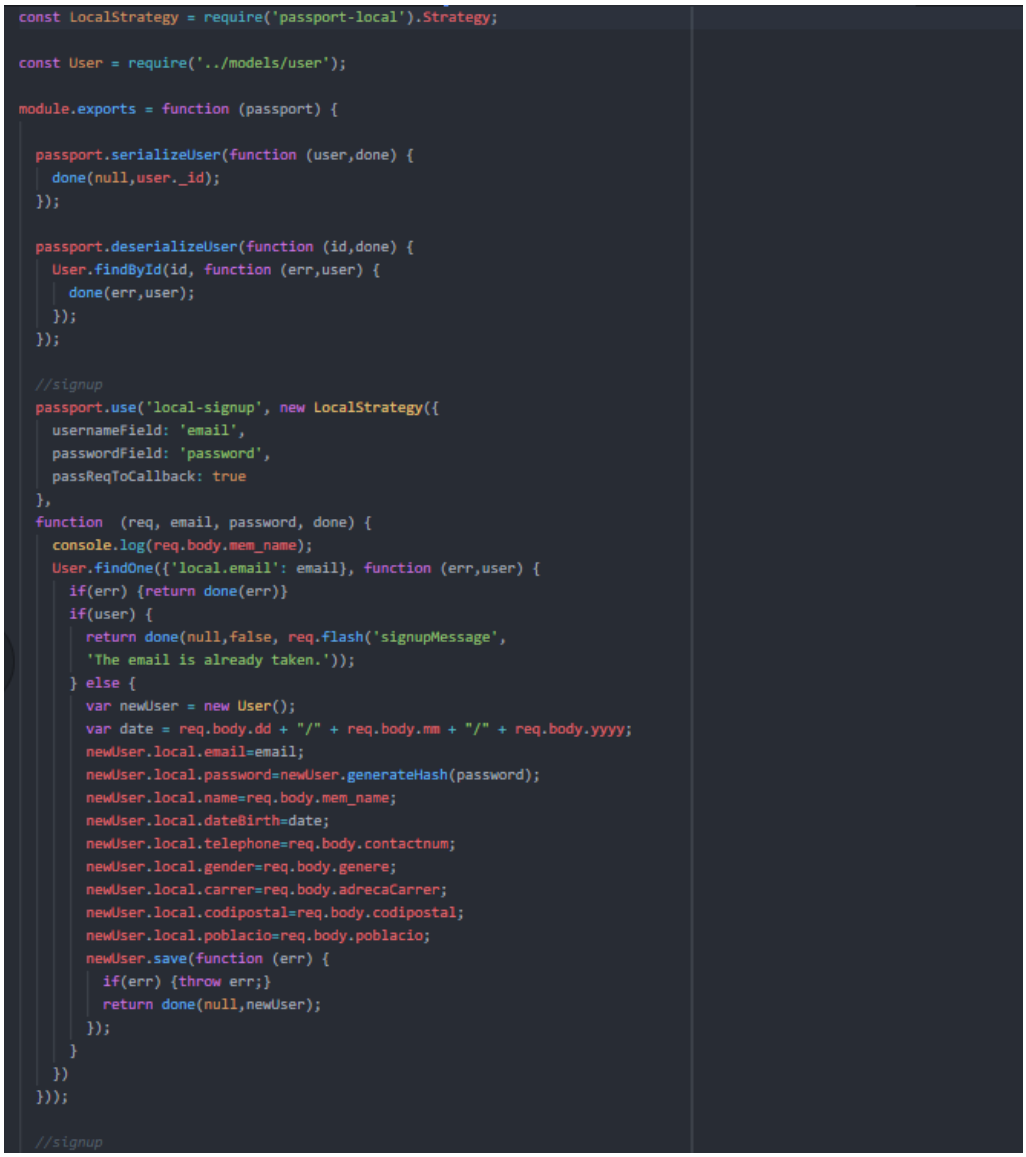
Figura 59: Body de la Plantilla EJS crear_contracte

10.3.2 REGISTRE

Per crear el registre hem utilitzat el passport que ens ofereix l'express per fer el control de usuaris.

10.3.2.1 Passport.js

En aquest fitxer es fa el control d'usuaris registrats.



```
const LocalStrategy = require('passport-local').Strategy;

const User = require('../models/user');

module.exports = function (passport) {

  passport.serializeUser(function (user,done) {
    done(null,user._id);
  });

  passport.deserializeUser(function (id,done) {
    User.findById(id, function (err,user) {
      done(err,user);
    });
  });

  //signup
  passport.use('local-signup', new LocalStrategy({
    usernameField: 'email',
    passwordField: 'password',
    passReqToCallback: true
  },
  function  (req, email, password, done) {
    console.log(req.body.mem_name);
    User.findOne({'local.email': email}, function (err,user) {
      if(err) {return done(err)}
      if(user) {
        return done(null,false, req.flash('signupMessage',
        'The email is already taken.'));
      } else {
        var newUser = new User();
        var date = req.body.dd + "/" + req.body.mm + "/" + req.body.yyyy;
        newUser.local.email=email;
        newUser.local.password=newUser.generateHash(password);
        newUser.local.name=req.body.mem_name;
        newUser.local.dateBirth=date;
        newUser.local.telephone=req.body.contactnum;
        newUser.local.gender=req.body.genere;
        newUser.local.carrer=req.body.adrecaCarrer;
        newUser.local.codipostal=req.body.codipostal;
        newUser.local.poblacio=req.body.poblacio;
        newUser.save(function (err) {
          if(err) {throw err;}
          return done(null,newUser);
        });
      }
    })
  }));

  //signup
```

Figura 60: Fitxer passaport.js

10.3.2.2 Index.js

10.3.2.2.1 GET /signup

En aquesta ruta, es redirigeix a la pàgina HTML *signup.ejs* enviant-li el missatge signupMessage.

```
/*
  Pagina registre.
  Li passa message a la pagina signup on conté signupMessage que el
  proporciona el fitxer passport al fer la autentificació.
*/
app.get('/signup', (req, res) => {
  res.render('signup', {
    message: req.flash('signupMessage')
  });
});
```

Figura 61: Ruta GET signup en el fitxer index.js

10.3.2.2.2 POST /signup

Aquesta ruta redirigeix al perfil si el registre ha funcionat correctament, altrament redirigeix a la ruta GET */signup*.

```
/*
  Post signup a BD.
  si la autentificació ha estat correcta, afegeix el usuari a la BD i
  redirigeix al perfil. ALtrament, torna a la pàgina actual.
*/
app.post('/signup', passport.authenticate('local-signup', {
  successRedirect: '/profile',
  failureRedirect: '/signup',
  failureFlash: true
}));
```

Figura 62: Ruta POST signup en el fitxer index.js

10.3.2.3 Signup.ejs

Consta de inputs dins un form demanant el *email, contrasenya, nom complert, data de naixement, gènere, número de telèfon, adreça, població i codi postal* del usuari. El *form* crida l'acció POST */signup*. També conté un enllaç que et redirigeix a la ruta GET /*login*.

Figura 63: Body de la Plantilla EJS signup

10.3.3 LOGIN

Per crear el login hem utilitzat el passport que ens ofereix l'express per fer el control de usuaris.

10.3.3.1 Passport.js

En aquest fitxer es fa el control d'usuaris registrats.

```
const LocalStrategy = require('passport-local').Strategy;

const User = require('../models/user');

module.exports = function (passport) {

  passport.serializeUser(function (user,done) {
    done(null,user._id);
  });

  passport.deserializeUser(function (id,done) {
    User.findById(id, function (err,user) {
      done(err,user);
    });
  });

  //signup
  passport.use('local-signup', new LocalStrategy({=
  function  (req, email, password, done) {=

  //signup
  passport.use('local-login', new LocalStrategy({
    usernameField: 'email',
    passwordField: 'password',
    passReqToCallback: true
  },
  function  (req, email, password, done) {

    User.findOne({'local.email': email}, function (err,user) {
      if(err) {return done(err)}
      if(!user) {
        return done(null,false, req.flash('loginMessage',
        'No user found.'));
      }
      if(!user.validatePassword(password)) {
        return done(null,false, req.flash(
          'loginMessage','Wrong password'));
      }
      return done(null,user);
    })
  }));


}
```

Figura 64: Fitxer passaport.js

10.3.3.2 Index.js

10.3.3.2.1 GET /login

En aquesta ruta, es redirigeix a la pàgina HTML *login.ejs* enviant-li el missatge loginMessage.

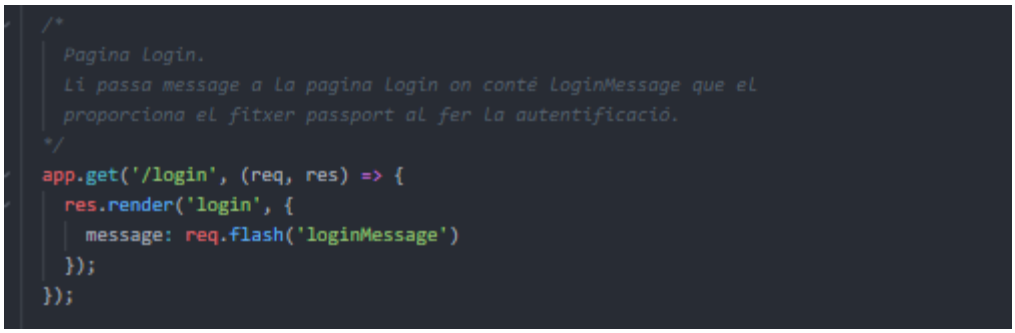


Figura 65: Ruta GET login en el fitxer index.js

10.3.3.2.2 POST /login

Aquesta ruta redirigeix al perfil si el registre ha funcionat correctament, altrament redirigeix a la ruta GET */login*.

Figura 66: Ruta POST login en el fitxer index.js

10.3.3.3 Login.ejs

Consta de inputs dins un form demanant el *email, contrasenya* del usuari. El *form* crida l'acció POST */login*. També conté un enllaç que et redirigeix a la ruta GET /*signup*.

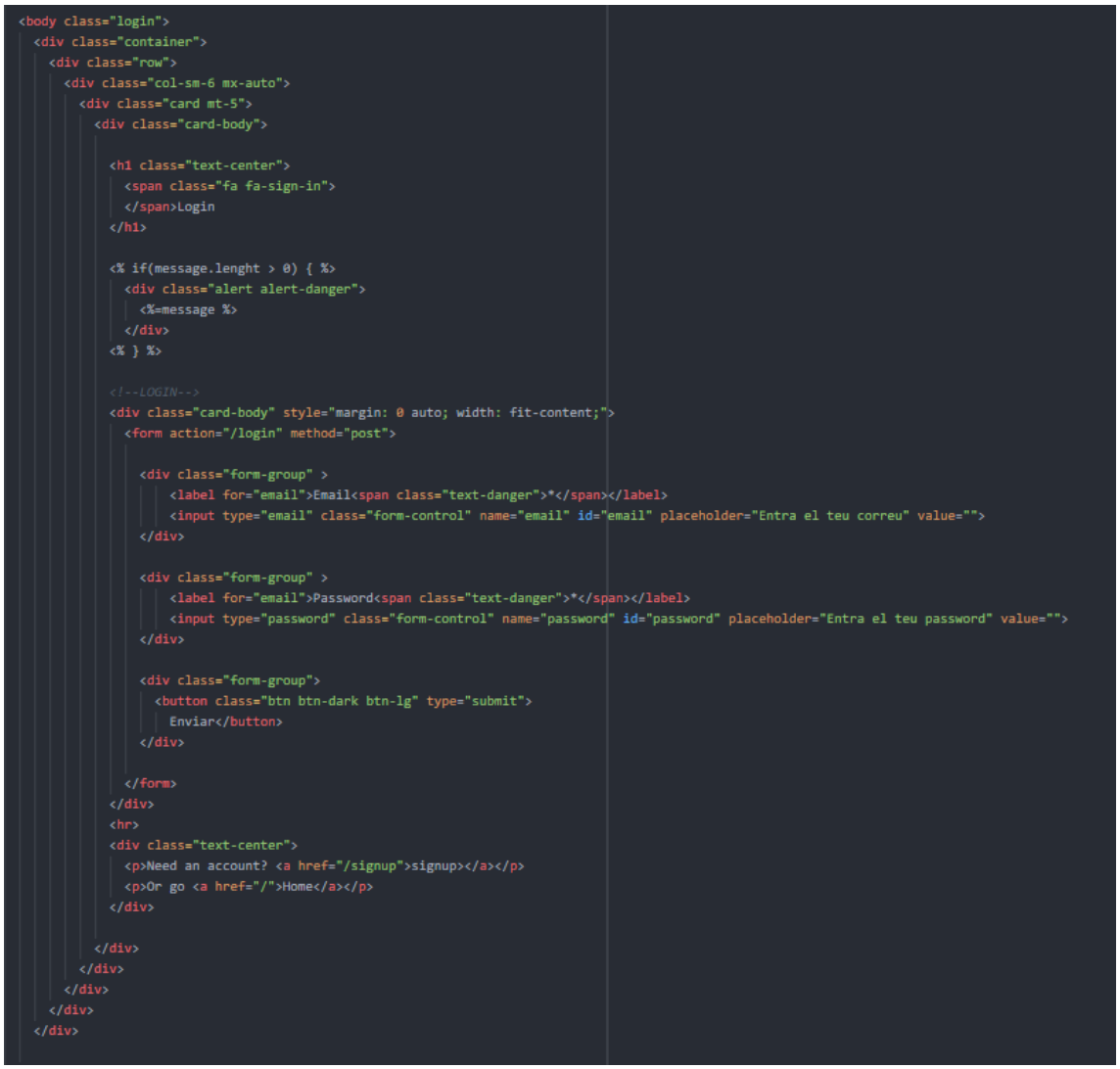


Figura 67: Body de la Plantilla EJS login

10.3.4 Veure perfil

10.3.4.1 Index.js

El perfil només conté una ruta GET per poder anar a la vista corresponent *profile.ejs.*

```
/*
  pagina perfil.
  Es pot veure l'id, email i password de l'usuari i un botó
  que et redirigeix a list_contracts.
  Li passa el usuari que ha rebut en el req amb nom user.
*/
app.get('/profile', (req,res) => {
  res.render('profile', {
    user: req.user
  });
});
```

Figura 68: Ruta GET profile en el fitxer index.js

10.3.4.2 Profile.ejs

Plantilla on es mostren les dades de l'usuari i un botó que redirigeix a la ruta */list_contract* passant-li per paràmetre *l'id* de l'usuari.

```
<body class="profile">
  <a href="/logout" class="btn btn-light" style="float:right; margin-right: 15px; margin-top: 20px">
    Logout</a>
  <div class="container">
    <div class="container" id="perfil">
      <div class = "row">

        <!-- CAIXA AMB EL PERFIL DE L'USUARI-->
        <div class="col-md-8" style="margin-top:25px">
          <div class="title-perfil">
            <h1 class="card-title">
              <strong>Perfil</strong>
            </h1>
          </div>
          <div class="card mt-5">
            <div>
              <div class="card-body">
                <h3 class="card-title">
                  <span class="fa fa-user"></span> Local</h3>

                  <strong>Email:</strong> <%= user.local.email %><br>
                  <strong>Nom complert:</strong> <%= user.local.name %><br>
                  <strong>Dia de naixament:</strong> <%= user.local.dateBirth %><br>
                  <strong>Telefon:</strong> <%= user.local.telephone %><br>
                  <strong>Carrer:</strong> <%= user.local.carrer %>, <%= user.local.poblacio %> (<%= user.local.codipostal %>)

              </div>
            </div>
            <!--CONTRACTES QUE TENS-->
            <div class="form-group">
              <div class="col-md-5">
                <div class="col-xs-offset-3 col-xs-10">
                  <form action="/list_contracts/<%= user.id%>" method="get">
                    <button class="btn btn-primary btn-dark" type="submit">
                      Contracts
                    </button>
                  </form>
                </div>
              </div>
            </div>

          </div>
        </div>


      </div>

    </div>
  </div>
</body>
```

Figura 69: Body de la Plantilla EJS profile

10.3.5 Veure contractes

10.3.5.1 Index.js

El perfil només conté una ruta GET per poder anar a la vista corresponent *list_contracts.ejs.*

```
/*
  Prové de la pàgina del perfil, si es volen veure els contractes.
  Passen per paràmetre l'id de l'usuari.
  Li passen a la pàgina list_contracts l'usuari i la llista de loteries del usuari.
*/
app.get('/list_contracts/:id', async (req, res) => {
  const { id } = req.params;
  const user = await User.findById(id);
  const lotteries = await Lottery.find({'userOwner': user.local.email});

  res.render('list_contracts', {
    user: user,
    no_correcte: "fals",
    list_lotteries: lotteries
  });
});
```

Figura 70: Ruta GET list_contracts en el fitxer index.js

10.3.5.2 list_contracts.ejs

Mostra un llistat de tots els contractes que pertanyen al usuari registrat. Cada contracte mostra el index, el nom, la descripció, els participants i les accions disponibles. Dins de les accions disponibles, la rifa disposa de 2 botons: editar i acabar.

Editar crida la ruta GET */edit* pasant-li per paràmetres l'*adreça del contracte* i la *id* del usuari i Acabar crida la ruta GET */delete* pasant-li per paràmetres l'*adreça del contracte* i la *id* del usuari.

Al final del llistat, també hi ha un botó "Crear contracte" que crida la ruta GET/crear_contracte passant-li per paràmetre l'id de l'usuari.

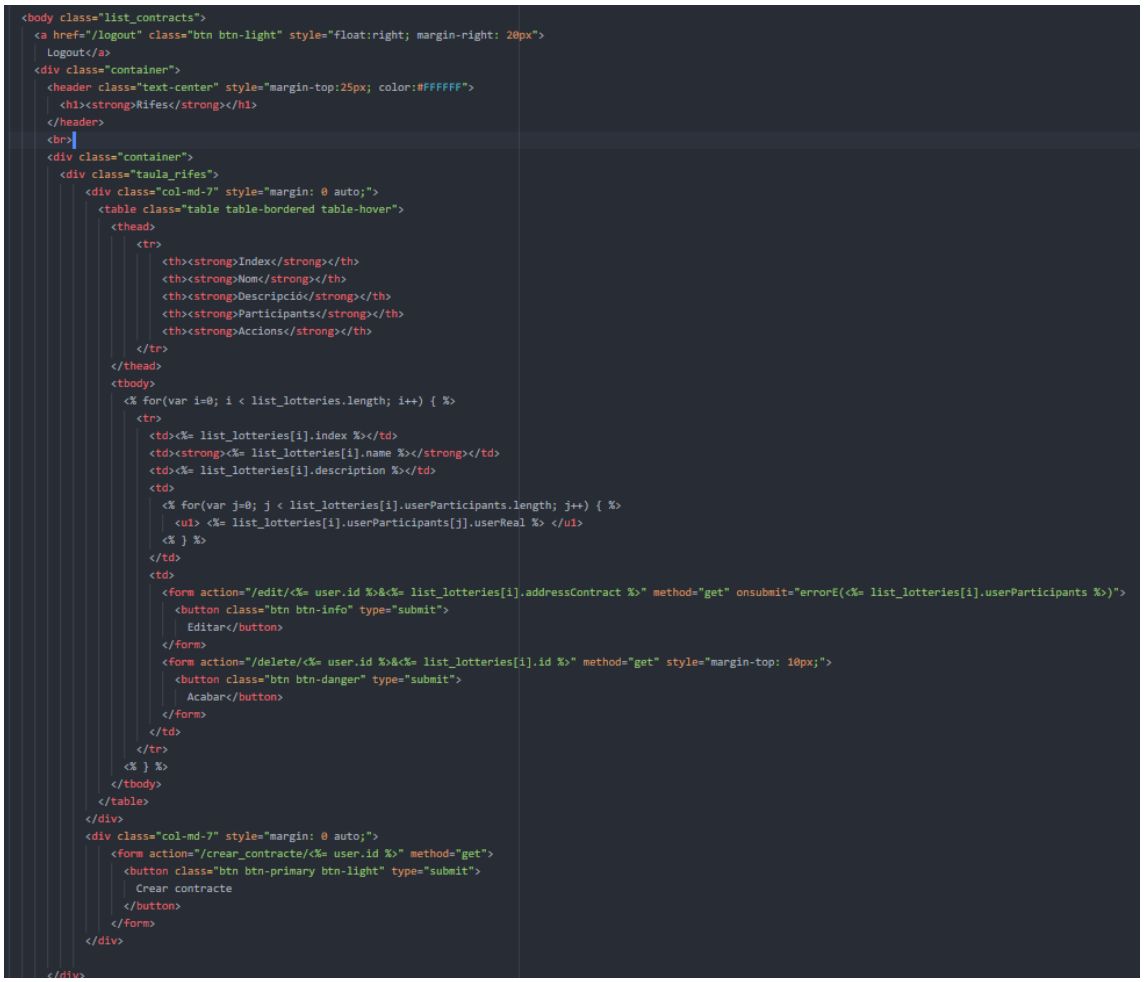


Figura 71: Body de la Plantilla EJS list_contracts

10.3.6 Afegir detalls a contracte

10.3.6.1 Index.js

Afegir detalls al contracte només conté una ruta POST per modificar la rifa a la BD. Afegeix el preu del tiquet a la rifa i crida a la funció *modificar*().

La funció modificar() crida la funció *modifyLimits()* del smart contract que modifica el màx de guanyadors i tiquets; i crida la funció *setPriceTicket()* del smart contract que modifica el preu del tiquet. Tot seguit, redirigeix a la vista *percentatges.ejs.*

```
/*
  Prové de la pàgina contract on es modifiquen els settings del contracte.
  S'han modificat els limits de guanyadors, tiquets i el preu per ticket.
  Passen per paràmetre l'id de l'usuari i la lotteria. En el req.body hi ha l'adressa del contrac
  Afegeix a la lotteria l'adreça del contracte i el seu index.
  Envia a contract_view l'usuari i la lotteria.
*/
app.post('/contract/:id_usr&:id_lot', async (req, res) => {
  var id_contract = req.body.contractAddressLottery;
  const { id_usr } = req.params;
  const user = await User.findById(id_usr);
  const { id_lot } = req.params;
  const lotte = await Lottery.findById(id_lot);

  modificar(lotte.addressContract, req.body.winMax, req.body.partMax, req.body.priceTicket);
  await Lottery.findOneAndUpdate(
    {'_id': id_lot},
    {'priceForTicket': req.body.priceTicket},
    {'upsert': true}
  );

  res.render('percentatges', {
    user: user,
    lotte: lotte
  });
});
```

Figura 72: Ruta POST contract en el fitxer index.js

```
function modificar(address,winMax,partMax,priceTicket) {
  var Web3 = require('web3');
  var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
  var abi =
  [{"constant":true,"inputs":[{"name":"participantAddress","type":"address"}],"name":"getTickets","outputs":[{"na
  on"},{"constant":true,"inputs":[],"name":"getContractAddress","outputs":[{"name":"","type":"address"}],"payable
  "},{"constant":false,"inputs":[{"name":"pTicket","type":"uint256"}],"name":"setPriceTicket","outputs":[],"payab
  teMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"getPriceTicket","outputs":[{"name"
  :[],"name":"payPot","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":
  payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"getOwner","outp
  t":false,"inputs":[],"name":"payLottery","outputs":[],"payable":true,"stateMutability":"payable","type":"functi
  me":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs
  nonpayable","type":"function"},{"inputs":[],"payable":false,"stateMutability":"nonpayable","type":"constructor"

  var LotteryContract = web3.eth.contract(abi);
  var contractInstance = LotteryContract.at(address);
  contractInstance.modifyLimits(winMax,partMax,{from: web3.eth.accounts[0], gas: 100000});
  contractInstance.setPriceTicket(priceTicket,{from: web3.eth.accounts[0], gas: 100000});
}
```

Figura 72: Funció modificar en el fitxer index.js

10.3.6.2 Contract.ejs

```
<body>
  <div class="container">
    <div class="col-md-8" style="margin: 0 auto;">
      <div class="card text-center mt-5">
        <div class="card-body">
          <h1 class="entry-title" style="margin-left: 10px">
            Crear Rifa
          </h1>
          <hr>
          <div class="card-body">
            <form action="/contract/<%= user.id %>&<%= lote.id %>" method="post" onsubmit="pasarParametreFunction()">
              <div class="form-group">
                  <label for="winMax"> Max de guanyadors: </label>
                  <div class="input-group">
                    <input class="form-control" type="text" name="winMax" id="winMax" placeholder="Insereix el max de guanyadors que vols que tingui la rifa."required>
                  </div>
              </div>

              <div class="form-group">
                  <label for="partMax"> Max de tiquets: </label>
                  <input class="form-control" type="text" name="partMax" id="partMax" placeholder="Insereix el max de tiquets que vols que es venguin."required>
              </div>


              <div class="form-group">
                  <label for="priceTicket"> Preu per tiquet : </label>
                  <input class="form-control" type="text" name="priceTicket" id="priceTicket" placeholder="Insereix el preu per el qual vols que es vengui el tiquet" required>
              </div>

              <hr>
            <button class="btn btn-dark btn-lg" type="submit">
              Següent
            </button>

            </form>
            <label>
              <small><strong>Nota:</strong>
                Pagaràs amb ethers cada tiquet que et comprin. Equivalencies: 1€ equival a 0,00400 ethereums.
              </small>
            </label>
          </div>
        </div>
      </div>
    </div>
```

Figura 74: Body de la Plantilla EJS contract

Consta de 2 inputs que demanen el màxim de guanyadors i el màxim de tiquets. Tot seguit, hi ha un input més demanant el preu del tiquet. Al final, mostra un botó "Següent" que redirigeix a la ruta POST /*contract* passant-li per paràmetre l'id de l'usuari i l'id de la rifa.

```
function pasarParametreFunction() {
  var x1 = $('#pr').html();
  var p = $('#priceTicket').val();
  $("#contractAddressLottery").val(x1);
  $("#priceTicket2").val(p);
  //alert($("#contractAddressLottery").val());
}
```

Figura 75: Funcions de la Plantilla EJS contract

10.3.7 Afegir percentatges a contracte

10.3.7.1 Index.js

Afegir percentatges només conté una ruta POST que crida la funció *afegirPercentatges()* passant-li per paràmetre l'adreça del contracte, el % del propietari i el % del sistema.

```
/*
  Prové de la pàgina contract on es modifiquen els settings del contracte.
  S'han modificat els límits de guanyadors, tiquets i el preu per ticket.
  Passen per paràmetre l'id de l'usuari i la lotteria. En el req.body hi ha l'adressa del contracte i el preu del t
  Afegeix a la lotteria l'adreça del contracte i el seu index.
  Envia a contract_view l'usuari i la lotteria.
*/
app.post('/percent/:id_usr&:id_lot', async (req, res) => {

  const { id_usr } = req.params;
  const user = await User.findById(id_usr);
  const { id_lot } = req.params;
  const lotte = await Lottery.findById(id_lot);
  afegirPercentatges(lotte.addressContract, req.body.feeOwner, req.body.feeSystem);
  res.render('profile', {
    user: user
  });
});
```

Figura 76: Ruta POST percent en el fitxer index.js

La funció afegirPercentatges crida la funció setFeeOwner() del smart contract passant-li per paràmetre el % del propietari i la funció setFeeSystem() del smart contract passant-li per paràmetre el % del sistema.

```
/*obte el contracte amb l'addreça address i
modifica els percentatges de l'owner per feeOwner i del sistema per feeSystem
*/
function afegirPercentatges(address,feeOwner,feeSystem) {
  var Web3 = require('web3');
  var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
  var abi =
  [{"constant":true,"inputs":[{"name":"participantAddress","type":"address"}],"name":"getTickets","outputs":[{"name":"","ty
  on"},{"constant":true,"inputs":[],"name":"getContractAddress","outputs":[{"name":"","type":"address"}],"payable":false,"s
  "},{"constant":false,"inputs":[{"name":"pTicket","type":"uint256"}],"name":"setPriceTicket","outputs":[],"payable":false,
  teMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"getPriceTicket","outputs":[{"name":"","type"
  :[],"name":"payPot","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inpu
  payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"getOwner","outputs":[{"na
  t":false,"inputs":[],"name":"payLottery","outputs":[],"payable":true,"stateMutability":"payable","type":"function"},{"con
  me":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"name"
  nonpayable","type":"function"},{"inputs":[],"payable":false,"stateMutability":"nonpayable","type":"constructor"},{"anonym

  var LotteryContract = web3.eth.contract(abi);
  var contractInstance = LotteryContract.at(address);
  console.log(feeOwner + " " + feeSystem);
  contractInstance.setFeeOwner(feeOwner,{from: web3.eth.accounts[0]});
  contractInstance.setFeeSystem(feeSystem,{from: web3.eth.accounts[0]});

  var contractInstance2 = LotteryContract.at(address);
  console.log("feeOwner:" + contractInstance2.getOwnerFee({from: web3.eth.accounts[0]}));
  console.log("feeSystem:" + contractInstance2.getSystemFee({from: web3.eth.accounts[0]}));
}
```

Figura 77: Funció afegirPercentatges en el fitxer index.js

10.3.7.2 Percentatges.ejs

Consta de 2 inputs que demanen el percentatge del premi per el propietari i el percentatge del premi per el sistema. El form envia una crida a la ruta POST */percent* passant-li per paràmetre l'id del usuari i de la rifa.

```
<body>
  <div class="container">  
    <div class="col-md-8" style="margin: 0 auto;">
    <div class="card text-center mt-5">
      <div class="card-body">

        <h1 class="entry-title" style="margin-left: 10px">
          Percentatges Rifa
        </h1>
        <hr>

        <form action="/percent/<%= user.id %>&<%= lotte.id %>" method="post" onsubmit="pasarParametreFunction()">
          <div class="col-xs-12">
            <div class="form-group">
              <label for="feeOwner"> Percentatge per el propietari: (<60%) </label>
              <input class="form-control" type="text" name="feeOwner" id="feeOwner"
              placeholder="Insereix el percentatge sobre el premi que vols tenir" required>
            </div>
          </div>

          <div class="col-xs-12">
            <div class="form-group">
              <label for="feeSystem"> Percentatge per el sistema: (>1%) </label>
              <input class="form-control" type="text" name="feeSystem" id="feeSystem"
              placeholder="Insereix el percentatge sobre el premi que vols que guanyi el sistema" required>
            </div>
          </div>

          <p> El percentatge restant és pel guanyador </p>

          <button class="btn btn-dark btn-lg" type="submit">
            Canviar
          </button>
        </form>

      </div>
    </div>
  </div>
  </div>
```

Figura 78: Body de la Plantilla EJS percentatges

10.3.8 Editar contracte

10.3.8.1 Index.js

10.3.8.1.1 GET /edit

En aquesta ruta, es redirigeix a la pàgina HTML *edit.ejs* enviant-li l'usuari, un string "fals" i l'adreça de contracte.

```
app.get('/edit/:id_usr&:address', async (req, res) => {
  const { id_usr } = req.params;
  const user = await User.findById(id_usr);
  const { address } = req.params;

  res.render('edit_contract', {
    user: user,
    no_correcte: "fals",
    lote_address: address
  });
});
```

Figura 79: Ruta GET edit en el fitxer index.js

10.3.8.1.2 POST /edit

Si hi ha algun participant en el contracte, redirigeix al fitxer HTML *edit_contract.ejs* amb els paràmetres user, "cert" i l'adreça del contracte. Altrament, s'executa la funció *editContracte()* passant-li per paràmetre l'adreça del contracte, el màx de guanyadors, el màx de tiquets i el preu per tiquet; redirigeix al fitxer HTML *profile*.ejs

```
app.post('/edit/:id_usr&:address', async (req, res) => {
  const { id_usr } = req.params;
  const user = await User.findById(id_usr);
  const { address } = req.params;

  var tm = Ticket.find({"addressContract": address}).select('userReal').exec(function(err,docs){
    docs = docs.map(function(doc) {
      return doc.userReal;
    });
    console.log(docs.length);
    if(docs.length==0) {
      editContracte(address, req.body.winMax, req.body.partMax, req.body.priceTicket);
      res.render('profile', {
        user: user
      });
    }
    else {
      console.log("contracte");
      res.render('edit_contract', {
        user: user,
        no_correcte: "cert",
        lote_address: address
      });
    }
  });
```

Figura 80: Ruta POST edit en el fitxer index.js

La funció *editContracte* crida la funció *modifyLimits*() del smart contract passant-li per paràmetre el màxim de guanyadors i el màxim de tiquets i la funció *setPriceTickets*() del smart contract passant-li per paràmetre el preu per tiquet.

Figura 81: Funció editContracte en el fitxer index.js

10.3.8.2 Edit_contract.ejs

Consta de 3 inputs demanat el màx de guanyadors, el màx de tiquets i el preu per tiquet. El form crida la ruta POST /edit passant per paràmetre el id del usuari i de la rifa.

Hi ha un altre botó “Contract” que crida la ruta GET /list_contracts passant per paràmetre el id del usuari.

Figura 82: Body de la Plantilla EJS edit

10.3.9 Acabar contracte

10.3.9.1 Index.js

En aquesta ruta, es crida a la funció *murderContract*() passant-li per paràmetre l'adreça del contracte, el màx de guanyadors, el màx de tiquets i el preu per tiquet, també la funció *getWinner*() passant l'adreça del contracte.

Després busca el telèfon que correspon amb l'adreça del guanyador i li envia un missatge notificant-lo que és el guanyador i a on ha d'anar a reclamar el premi.

Després redirigeix a la pàgina HTML *pagar_client.ejs* enviant-li l'usuari, un string "fals", el telèfon del guanyador i la rifa.

```
app.get('/delete/:id_usr&:id_lot', async (req, res) => {
  const { id_usr } = req.params;
  const user = await User.findById(id_usr);
  const { id_lot } = req.params;
  const lotte = await Lottery.findById(id_lot);
  var address = lotte.addressContract;
  murderContracte(address, req.body.winMax, req.body.partMax, req.body.priceTicket);
  var winner = getWinner(address);
  var telephone=0;
  var mida=0;
  var trobat=false;
  const lotteries = await UserBlock.find({'addressContract': address, 'userAddressBlock': winner}).select('userReal').exec(function(err,docs){
      docs = docs.map(function(doc) {
        return doc.userReal;
      });
      console.log(docs[0]);
      telephone=docs[0];
      console.log("phone:" + telephone);
      var message = "Ets el guanyador!!! Ves a reclamar el teu premi i quan acabis clica en el següent enllaç: "
      + "https://192.168.1.34:3000/payClient/" + id_lot + "&" + telephone;

      //enviar missatge
      var TMClient = require('textmagic-rest-client');
      telephone = "+34" + telephone;
      var c = new TMClient('lailaabjil', '502Lx3yCwIrXW3Ybxz6WTDNnvb2hC6');
      c.Messages.send({text: message, phones: telephone}, function(err, res){
          console.log('Messages.send()', err, res);
      });

      res.render('pagar_client', {
        user: user,
        no_correct: "false",
        winner: telephone,
        lotte: lotte
      });
    });

});
```

Figura 83: Ruta POST delete en el fitxer index.js

```
function getWinner(address) {
  var Web3 = require('web3');
  var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
  var abi =
  [{"constant":true,"inputs":[{"name":"participantAddress","type":"address"}],"name":"getTickets","outpu
  on"},{"constant":true,"inputs":[],"name":"getContractAddress","outputs":[{"name":"","type":"address"}]
  "},{"constant":false,"inputs":[{"name":"pTicket","type":"uint256"}],"name":"setPriceTicket","outputs":
  teMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"getPriceTicket","outputs"
  :[],"name":"payPot","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"c
  payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"getOwn
  t":false,"inputs":[],"name":"payLottery","outputs":[],"payable":true,"stateMutability":"payable","type
  me":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":fals
  nonpayable","type":"function"},{"inputs":[],"payable":false,"stateMutability":"nonpayable","type":"con

  var LotteryContract = web3.eth.contract(abi);
  var contractInstance = LotteryContract.at(address);
  return contractInstance.getWinner({from: web3.eth.accounts[0], gas: 100000});
}
```

Figura 84: Funció getWinner en el fitxer index.js

Figura 85: Funció murderContract en el fitxer index.js

Un cop clicat a “rebre_premi”, si l’atribut de la rifa “pagat” és cert (passa a cert quan el guanyador clica a l’enllaç posant constància de que ha cobrat el premi) crida la funció pay() passant-li per paràmetre l’adreça del contracte i redirecciona al fitxer HTML *list_contracts*.ejs, altrament redirecciona a *pagar_client.*ejs.

Figura 86: Ruta POST rebre_premi en el fitxer index.js

La funció pay() crida a la funció del smart contract payPot() que paga els premis.

Figura 87: Funció pay en el fitxer index.js

En aquesta ruta, es modifica l'atribut de la rifa "pagat" passant-li a true i redirecciona al fitxer HTML *options.ejs*.

```
app.get('/payClient/:id_lot&:telephone', async (req,res) => {
  const { id_lot } = req.params;
  const lotte = await Lottery.findById(id_lot);
  var address = lotte.addressContract;
  await Lottery.findOneAndUpdate(
    {'_id': id_lot},
    {'pagat': true},
    {'upsert': true}
  );

  res.render('options', {
    pagat: "cert"
  });
});
```

Figura 88: Ruta POST payClient en el fitxer index.js

10.3.9.2 Pagar_client.ejs

Mostra el guanyador del premi i un botó que crida la ruta /rebre_premi passant-li per paràmetre l'id de la rifa i del usuari.

```
<body>
  <div class="container">  
    <form action="/rebre_premi/<%= lotte.id %>&<%= user.id %>" method="post">
      <div class="row">
        <div class="col-md-8" style="margin: 0 auto;">
          <div class="card text-center mt-5">
            <div class="card-body">
              <p> Per aconseguir el teu premi, pagui el corresponent al guanyador: <%= winner %>
                <br>Un cop hagi pagat al jugador, clica a "Rebre premi" i rebrà els diners <p>

                <input class="form-control" type="hidden" name="winnerClient" id="winnerClient" value="<%= winner %>">
                <button class="btn btn-dark btn-lg" type="submit">
                  Rebre premi
                </button>
            </div>
          </div>
        </div>
      </div>
    </form>
  </div>
</body>
</html>
```

Figura 89: Body de la Plantilla EJS pagar_client

10.3.10 Veure llista de rifes

10.3.10.1 Index.js

Aquesta ruta redirigeix al fitxer HTML *participate*.ejs passant-li per paràmetre la llista de les rifes.

```
/*
  Prové de /, mostra totes les lotteries que existeixen.
  Envia a participate la llista de loteries amb nom list_lotteries
*/
app.get('/participate_lotteries', async (req,res)=> {
  const lotteries = await Lottery.find();
  res.render('participate', {
    list_lotteries: lotteries
  });
});
```

Figura 90: Ruta GET participate_lotteries en el fitxer index.js

10.3.10.2 Participate.ejs

Mostra un llistat amb totes les rifes que existeixen a l'aplicació. Cada rifa mostra l'index, el propietari, el nom, la descripció i l'acció que es pot fer. A la casella d'acció es mostra un botó "comprar tiquets" que crida la ruta GET */buy_tickets* passant-li per paràmetres l'id de la loteria.

```
<body class="particpate_lot">
  <a href="/" class="btn btn-light" style="float:right; margin-right: 20px">Home</a>
  <div class="container">
    <header class="text-center" style="margin-top:25px; color:#FFFFFF">
      <h1><strong>Rifes</strong></h1>
    </header>
    <br>

    <div class="container">
      <div class = "participate_rifes">
        <!--CONTRACTES QUE TENS-->
        <div class="col-md-7" style="margin: 0 auto;">
          <table class="table table-bordered table-hover">
            <thead>
              <tr>
                <th>Index</th>
                <th>Propietari</th>
                <th>Nom</th>
                <th>Descripció</th>
                <th>Accions</th>
              </tr>
            </thead>
            <tbody>
              <% for(var i=0; i < list_lotteries.length; i++) { %>
                <tr>
                  <td><%= list_lotteries[i].index %></td>
                  <td><%= list_lotteries[i].userOwner %></td>
                  <td><strong><%= list_lotteries[i].name %></strong></td>
                  <td><%= list_lotteries[i].description %></td>
                  <td>
                    <a href="/buy_tickets/<%= list_lotteries[i].id %>" class="btn btn-md btn-outline-info">Comprar tiquets</a>
                  </td>
                </tr>
              <% } %>
            </tbody>
          </table>
        </div>
      </div>
    </div>
```

Figura 91: Body de la Plantilla EJS participate

10.3.11 Comprar tiquet

10.3.11.1 Index.js

10.3.11.1.1 GET buy_tickets

Aquesta ruta redirecciona al fitxer HTML *buy_ticket.ejs.*

```
//pagina per obtenir numero de tel i num de tickets a comprar
app.get('/buy_tickets/:id', async (req,res) => {
  const { id } = req.params;
  const lotte = await Lottery.findById(id);
  res.render('buy_ticket', {
    lotte: lotte
  })
});
```

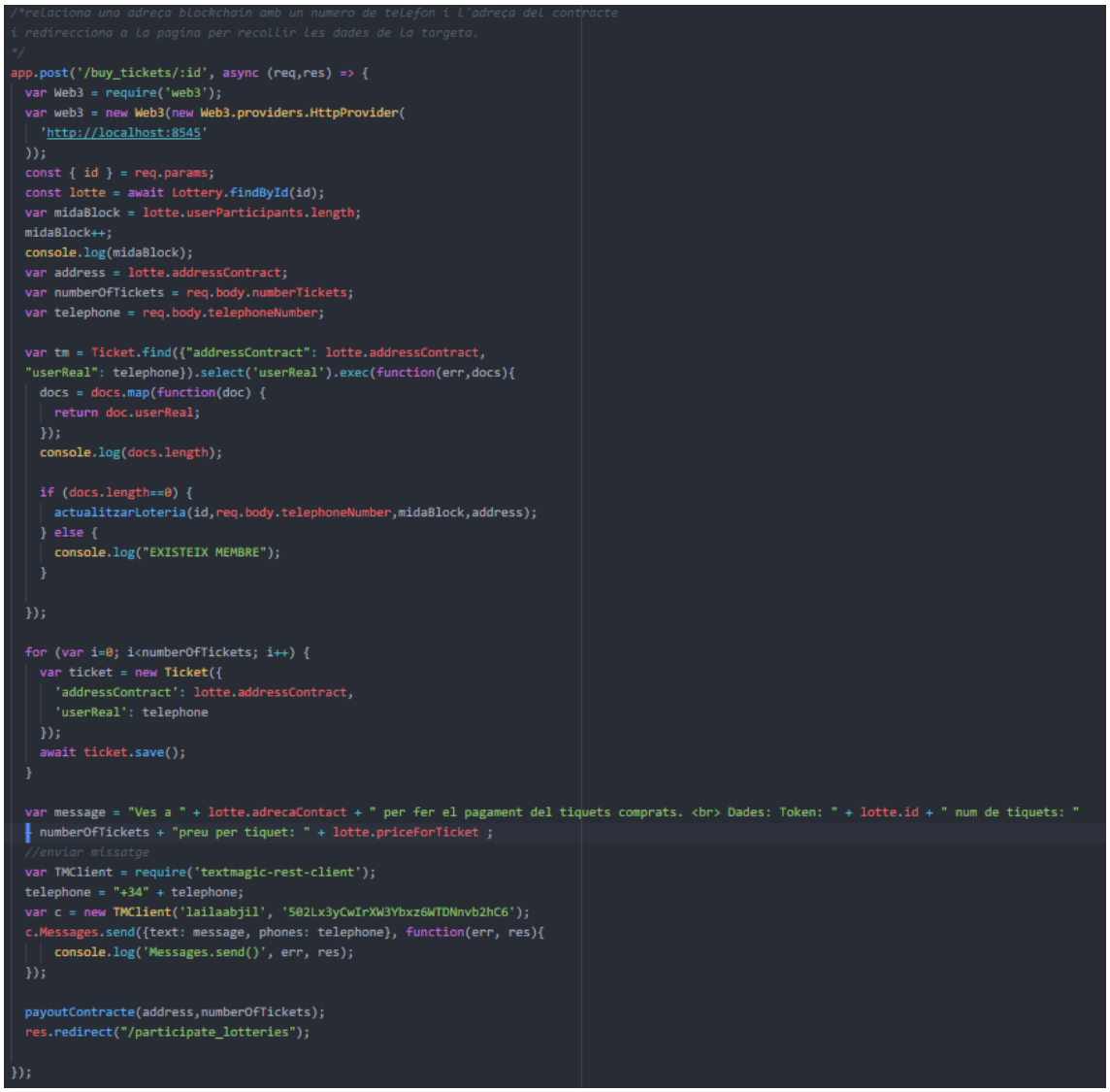
Figura 92: Ruta GET buy_tickets en el fitxer index.js

10.3.11.1.2 POST buy_tickets

Aquesta ruta crida la funció *actualitzarLoteria*() passant-li per paràmetre l'id de la rifa, el telèfon del usuari, el número de l'array de usuaris del blockchain que li pertoca al usuari real i l'adreça de la rifa.

També crea un tiquet amb l'adreça del contracte i el telèfon de l'usuari que l'ha comprat.

Envia un missatge amb les dades necessàries al usuari i crida la funció *payoutContracte*() passant-li per paràmetre l'adreça de la rifa i el número de tiquets comprats.



```
/*relaciona una adreça blockchain amb un numero de telefon i l'adreça del contracte
i redirecciona a la pagina per recollir les dades de la targeta.
*/
app.post('/buy_tickets/:id', async (req,res) => {
  var Web3 = require('web3');
  var web3 = new Web3(new Web3.providers.HttpProvider(
    'http://localhost:8545'
  ));
  const { id } = req.params;
  const lotte = await Lottery.findById(id);
  var midaBlock = lotte.userParticipants.length;
  midaBlock++;
  console.log(midaBlock);
  var address = lotte.addressContract;
  var numberOfTickets = req.body.numberTickets;
  var telephone = req.body.telephoneNumber;

  var tm = Ticket.find({"addressContract": lotte.addressContract,
  "userReal": telephone}).select('userReal').exec(function(err,docs){
    docs = docs.map(function(doc) {
      return doc.userReal;
    });
    console.log(docs.length);

    if (docs.length==0) {
      actualitzarLoteria(id,req.body.telephoneNumber,midaBlock,address);
    } else {
      console.log("EXISTEIX MEMBRE");
    }

  });

  for (var i=0; i<numberOfTickets; i++) {
    var ticket = new Ticket({
      'addressContract': lotte.addressContract,
      'userReal': telephone
    });
    await ticket.save();
  }

  var message = "Vas a " + lotte.adrecaContact + " per fer el pagament del tiquets comprats. <br> Dades: Token: " + lotte.id + " num de tiquets: "
  + numberOfTickets + "preu per tiquet: " + lotte.priceForTicket ;
  //enviar missatge
  var TMClient = require('textmagic-rest-client');
  telephone = "+34" + telephone;
  var c = new TMClient('lailaabjil', '502Lx3yCwIrXW3Ybxz6WTDNnvb2hC6');
  c.Messages.send({text: message, phones: telephone}, function(err, res){
      console.log('Messages.send()', err, res);
  });

  payoutContracte(address,numberOfTickets);
  res.redirect("/participate_lotteries");

});
```

Figura 93: Ruta POST buy_ticket en el fitxer index.js

```
async function actualitzarLoteria(id,telephoneNumber,midaBlock,address) {
  var Web3 = require('web3');
  var web3 = new Web3(new Web3.providers.HttpProvider(
    'http://localhost:8545'
  ));

  var userBlock = new UserBlock({
    'userReal': telephoneNumber,
    'userBlock': midaBlock,
    'userAddressBlock': web3.eth.accounts[midaBlock],
    'addressContract': address
  });
  await userBlock.save();

  await Lottery.findOneAndUpdate(
    {'_id': id},
    {$push:
      {'userParticipants':
        {
          'userReal': telephoneNumber,
          'userBlock': midaBlock,
          'userAddressBlock': web3.eth.accounts[midaBlock],
          'addressContract': address
        }
      }},
    {'upsert': true}
  );
}
```

Figura 94: Funció actualitzarLoteria en el fitxer index.js

```
/*obte el contracte amb l'addreça address i
executa el murder.
*/
function payoutContracte(address,numberTickets) {

    var Web3 = require('web3');
    var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
    var abi =
    [{"constant":true,"inputs":[{"name":"participantAddress","type":"address"}],"name":"getTickets","outputs":[{"name":""
    nction"},{"constant":true,"inputs":[],"name":"getContractAddress","outputs":[{"name":"","type":"address"}],"payable":
    function"},{"constant":false,"inputs":[{"name":"pTicket","type":"uint256"}],"name":"setPriceTicket","outputs":[],"pay
    ":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"getPriceTicket","outputs":[{
    ":false,"inputs":[],"name":"payPot","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"
    "type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"g
    e":"function"},{"constant":false,"inputs":[],"name":"payLottery","outputs":[],"payable":true,"stateMutability":"payab
    getOwnerFee","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"c
    yable":false,"stateMutability":"nonpayable","type":"function"},{"inputs":[],"payable":false,"stateMutability":"nonpay

    var LotteryContract = web3.eth.contract(abi);
    var contractInstance = LotteryContract.at(address);
    var ticket = contractInstance.getPriceTicket({from: web3.eth.accounts[0], gas: 100000});
    console.log(ticket);
    for(var i=0; i<numberTickets; i++) {
        contractInstance.payLottery({from: web3.eth.accounts[1],value:ticket, gas: 1000000});
    }
    var contractInstance2 = LotteryContract.at(address);
    console.log("pot:" + contractInstance2.getPot({from: web3.eth.accounts[0]}));

}
```

Figura 95: Funció payoutContracte en el fitxer index.js

10.3.11.2 Buy_ticket.ejs

Consta de 2 inputs demanant el número de telèfon i el número de tiquets que l'usuari vol comprar. El *form* executa la crida la ruta POST */buy_tickets* passant-li per paràmetre l'id de la rifa.

```
<body class="buyTickets">
  <div class="container">  
    <div class="col-md-8">
      <div class="card mt-5">
        <div class="card-body">
          <h1 class="entry-title" style="margin-left: 10px">
            <strong>Comprar Tiquets</strong>
          </h1>
          <hr>

          <form action="/buy_tickets/<%= lotte.id %>" method="post" onsubmit="enviarMissatge()">

            <div class="form-group">
              <div class="col-md-8 col-sm-9">
                <label for="telephoneNumber"> Número de telefon: </label>
                <input class="form-control" type="text" id="telephoneNumber" name="telephoneNumber" required>
              </div>
            </div>

            <div class="form-group">
              <div class="col-md-8 col-sm-9">
                <label for="numberTickets"> Número de tiquets que es volen comprar: </label>
                <input class="form-control" type="text" id="numberTickets" name="numberTickets" required>
              </div>
            </div>

            <div class="form-group">
              <div class="col-md-8 col-sm-9">
                <button class="btn btn-dark btn-lg" type="submit">
                  Next
                </button>
              </div>
            </div>

          </form>
        </div>
      </div>
    </div>
  </div>
</body>
```

Figura 96: Body de la Plantilla EJS buy_tickets

10.4 CLIENT

ESTRUCTURA DE DIRECTORIS

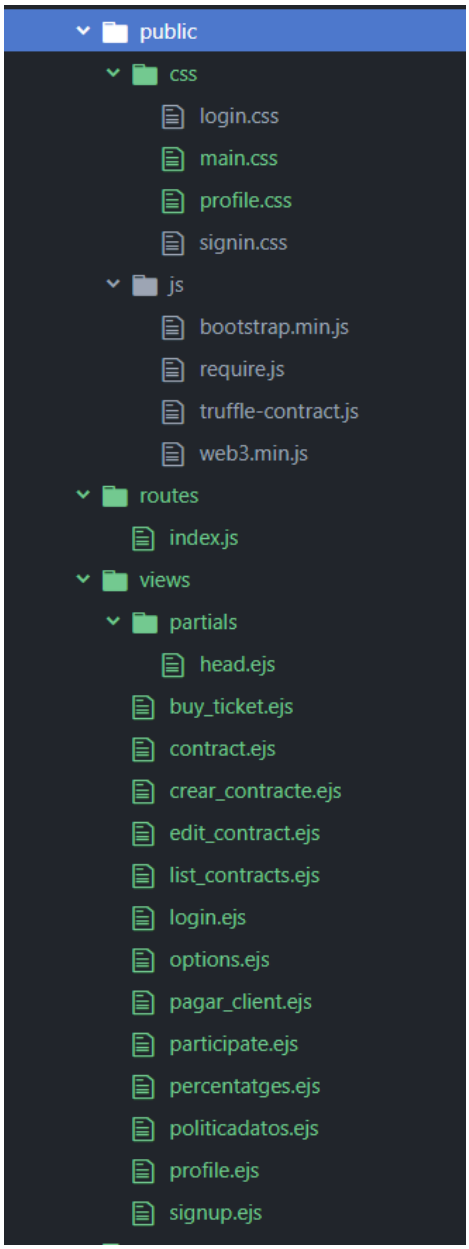


Figura 97: estructura de directoris del client

- **public/css:** carpeta que conté diversos fitxers d'estils del projecte.

- **Login.css:** fulla d'estil de la plantilla que pertany al login.

```
body, html{
    background-color:#34515E;
    font-family: 'Roboto', sans-serif;
    color:#333;
    line-height: 22px;
}
```

Figura 98:login.css

- **Profile.css:** fulla d'estil de la plantilla que pertany al perfil.

```
body, html{
    background-color:#34515E;
    font-family: 'Roboto', sans-serif;
    color:#333;
    line-height: 22px;
}

th {
  background-color:#81DAF5;
  color:#000000;
}
```

Figura 99: profile.css

- **Signin.css:** fulla d'estil de la plantilla que pertany al registre.

```
body {
    background: #fff;
  font-family: 'Roboto', sans-serif;
  color:#333;
  line-height: 22px;
}
h1, h2, h3, h4, h5, h6 {
  font-family: 'Roboto Condensed', sans-serif;
  font-weight: 400;
  color:#111;
  margin-top:5px;
  margin-bottom:5px;
}
h1, h2, h3 {
  text-transform:uppercase;
}

input.upload {
    position: absolute;
    top: 0;
    right: 0;
    margin: 0;
    padding: 0;
    font-size: 12px;
    cursor: pointer;
    opacity: 1;
    filter: alpha(opacity=1);
}
```

Figura 100: signup.css

- **Main.css:** fulla d'estil de la plantilla que defineix l'estil general de l'aplicació en genera.

```css
.footer {
    position: fixed;
    left: 0;
    bottom: 0;
    width: 100%;
    background-color: grey;
    color:#000000;
    text-align: right;
}

.politica {
  background-color:#81DAF5;
  color:#000000;
}

.signup {
  background-color:#04B4AE;
  font-family: 'Roboto', sans-serif;
  color:#333;
  line-height: 22px;
}

.options {
  background-color:#04B4AE;
  font-family: 'Roboto', sans-serif;
  color:#333;
  line-height: 22px;
}

.particpate_lot {
  background-color:#34515E;
  font-family: 'Roboto', sans-serif;
  color:#333;
  line-height: 22px;
}

.buyTickets {
  background-color:#34515E;
  font-family: 'Roboto', sans-serif;
  color:#333;
  line-height: 22px;
}

.btn-outline-info {
    color: #17a2b8;
    background-color: #f8f9fa;
    background-image: none;
    border-color: #17a2b8;
}

th {
  background-color:#FFFFFF;
  color:#000000;
}

a {
  color:#FFFFFF;
}

td {
  background-color:#34515E;
  color:#FFFFFF;
}
```

Figura 101: main.css

- **public/js:** carpeta que conté els diversos fitxers javascript del projecte.

10.5 PROBLEMES

Al principi el projecte s'havia plantejat com una aplicació mòbil desenvolupant-lo amb llenguatge Android.

Avui en dia, encara no s'ha acabat de resoldre bé la relació entre el blockchain i el Android de manera que alhora d'implementar les funcions de pagament del smart contract em van sorgir molts problemes i vaig arribar a la conclusió que no es podia implementar.

Degut a això, vaig haver de canviar l'implementació degut a aquest problema i el vam solucionar el problema canviant l'aplicació Android per una aplicació web, la qual té unes prestacions similars davant dels requeriments inicials.

També vaig tenir problemes alhora de implementar la part de l'aplicació connectada amb el smart contract ja que hi ha molt poca informació disponible.

11 IMPLEMENTACIÓ I RESULTATS

11.1 IMPLEMENTACIÓ

Al implementar l'aplicació el primer que vaig va ser crear el smart contract amb el REMIX per poder compilar-lo i testejar-lo.

Un cop creat vaig procedir a fer la configuració del blockchain. Vaig instal·lar-me el "Ethereum Wallet", i sincronitzar-me el testnet ROPSTEN. Després vaig crear 12 adreces i vaig procedir a instal·lar-me el TestRPC i el NodeJS.

El TestRPC em servirà per executar el entorn i tenir el blockchain amb les adreces que vaig crear. En aquest cas executo la següent comanda:

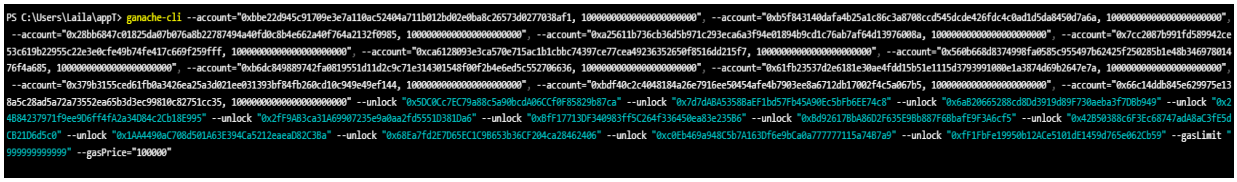


Figura 102: TestRPC executant el blockchain

El NodeJS és necessàri per poder-te descargar el TestRPC.

Ara ja puc procedir a crear l'aplicació.

Es va començar per crear la funció del servidor que crea el contracte cridant la funció corresponent del smart contract. Per fer-ho, vaig crear el fitxer *HTML* amb els inputs necessaris: el nom, la descripció i l'adreça de gestions del contracte; i el *JavaScript* que crea el smart contract. També creo a la BD una rifa amb el nom, la descripció i l'adreça. Explicat en el *apartat 9.3.1.2* i el *9.3.1.3* del document.

Després de crear el contracte, vaig procedir a afegir els detalls del contracte: el màxim de guanyadors, el màxim de tiquets que es poden comprar i el preu per tiquet.
Per fer-ho, vaig crear el fitxer *HTML* amb els inputs necessaris i amb el *JavaScript* que afegeix els detalls i el preu per tiquet; i s'afegeix un index i el preu del tiquet en la rifa a la BD. Explicat en el *apartat 9.3.6* del document.

Al afegir els detalls del contracte, puc afegir l'apartat dels percentatges. Vaig crear el fitxer *HTML* amb 2 inputs: % del propietari i % del sistema. Després en el fitxer *index.js* es criden les funcions corresponents del smart contract per modificar-lo. Explicat en el *apartat 9.3.7* del document.

Un cop fet aquest cicle procedim a fer la gestió d'usuaris. Creo tant el login i el registre. Es crea un fitxer *passport.js* amb el control de usuaris. Els fitxers *HTML* del login i el registre amb les dades necessàries i en el fitxer *index.js* afegim les rutes que criden les funcions del *passport*. Explicat en el *apartat 9.3.2* i *9.3.3* del document.

Ara que tinc el control d'usuaris fet, puc crear el perfil del usuari. En aquest cas, cal solament el fitxer *HTML* amb les vistes necessàries i la ruta en el fitxer *index.js* la qual redirigeixi al fitxer *HTML* creat. Explicat en el *apartat 9.3.4* del document.

En el perfil es pot anar a veure els contractes. Per poder veure el llistat de contractes cal crear el fitxer *HTML* amb una llista de les rifes del usuari i la ruta en el fitxer *index.js* que busca totes les rifes del usuari registrat i redirigeix al fitxer *HTML* creat. Explicat en el *apartat 9.3.5* del document.

Un cop fet, procedeixo a crear l'apartat de participar rifes. Aquí cal veure el llistat de contractes de tots els usuaris. Per poder veure el llistat de contractes cal crear el fitxer *HTML* amb una llista de totes les rifes i la ruta en el fitxer *index.js* que busca totes les rifes de la BD i redirigeix al fitxer *HTML* creat. Explicat en el *apartat 9.3.10* del document.

A cada rifa es pot cridar l'opció de comprar tiquets. Al fer-ho, cal crear un fitxer *HTML* demanant com a inputs el núm de telèfon del usuari i el núm de tiquets que es volen

comprar; i la ruta en el fitxer *index.js* s'afegeix el participant dins la rifa, es creen els tiquets dins la BD, s'envia un missatge al telèfon indicat notificant les dades del tiquet i es crida a la funció del smart contract per comprar tiquets. Explicat en el *apartat 9.3.11* del document.

Ara caldrà crear l'opció de finalitzar el contracte. Per fer-ho cal crear el fitxer *HTML* per notificar el guanyador del premi; i la ruta en el fitxer *index.js* la qual crida la funció del smart contract que acaba el contracte i envia un missatge notificant al guanyador del seu premi i de l'enllaç que ha de clicar quan rebi al premi. Un cop clicat l'enllaç i pagat el premi, el propietari rep el premi. Explicat en el *apartat 9.3.9* del document.

Per últim creo l'opció d'editar el smart contract. Es crea un fitxer HTML demanant els 3 inputs necessaris: màx guanyadors, màx tiquets i preu per tiquet; i una ruta en el fitxer *index.js* que si no hi ha cap participant al contracte, crida la funcions que modifiquen aquests elements del smart contract i, si n'hi ha algun, et mostra un error. Explicat en el *apartat 9.3.8* del document.

11.2 RESULTATS

En aquest apartat demostrarem que s'han realitzat tots els requeriments de l'aplicació descrits a l'apartat 7 i 8.

11.2.1 ENGEGAR BLOCKCHAIN

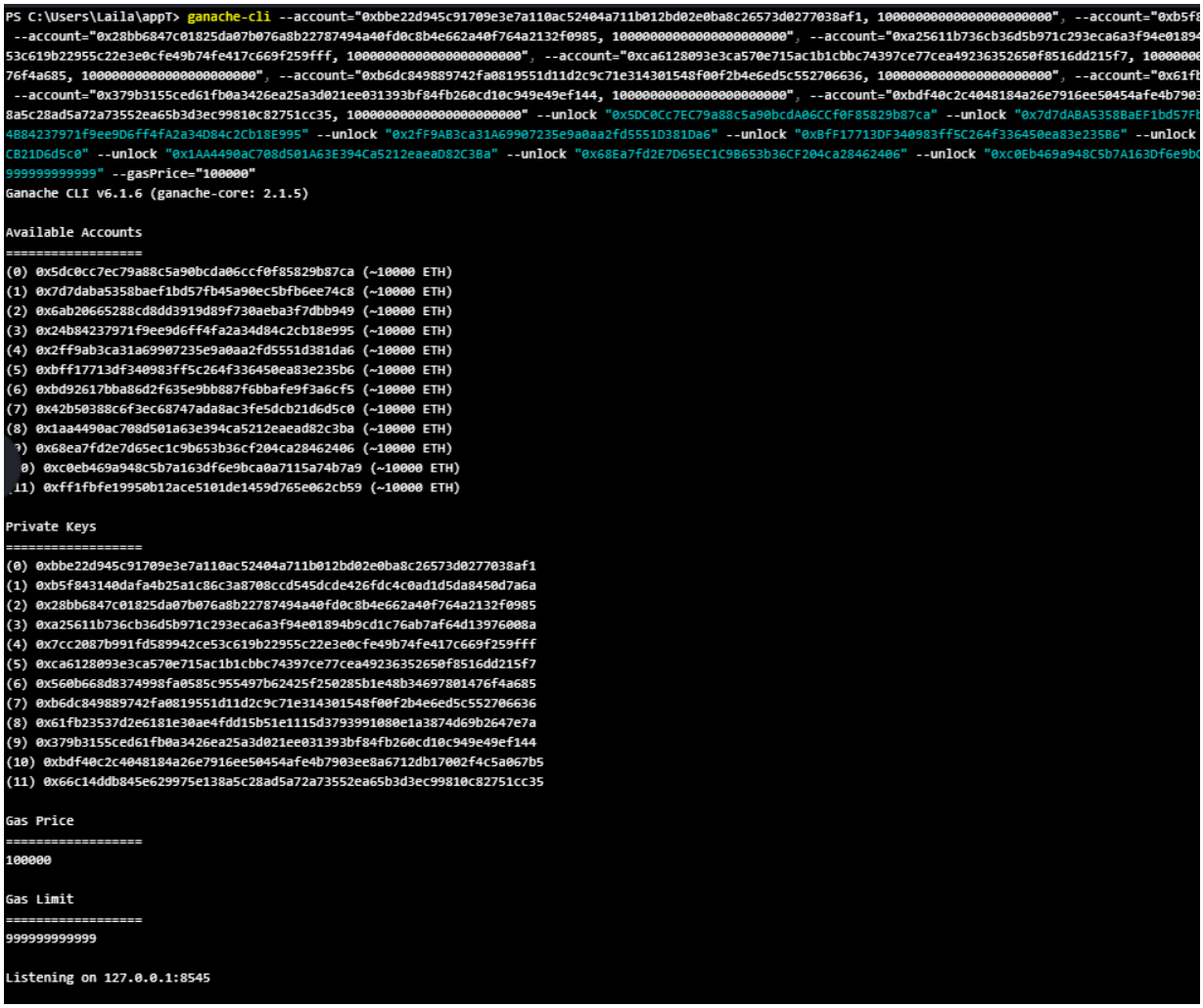


Figura 103: TestRPC executant el blockchain

11.2.2 INICI

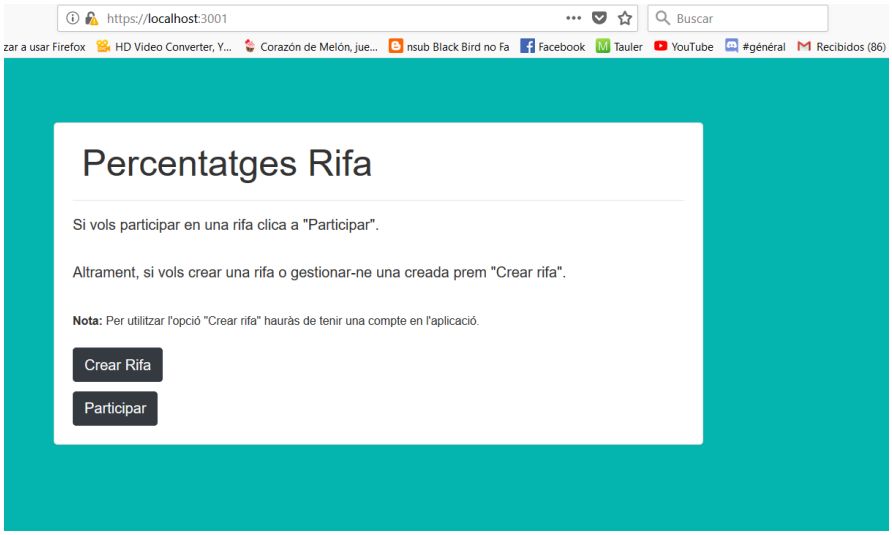


Figura 104: inici

11.2.3 LOGIN

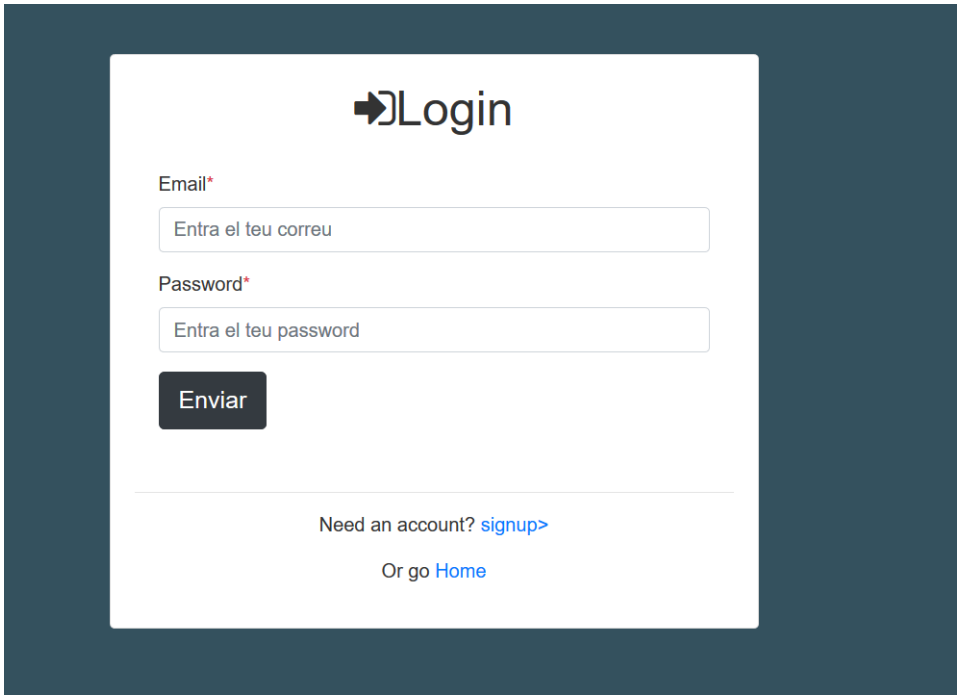


Figura 105: login

11.2.4 REGISTRE

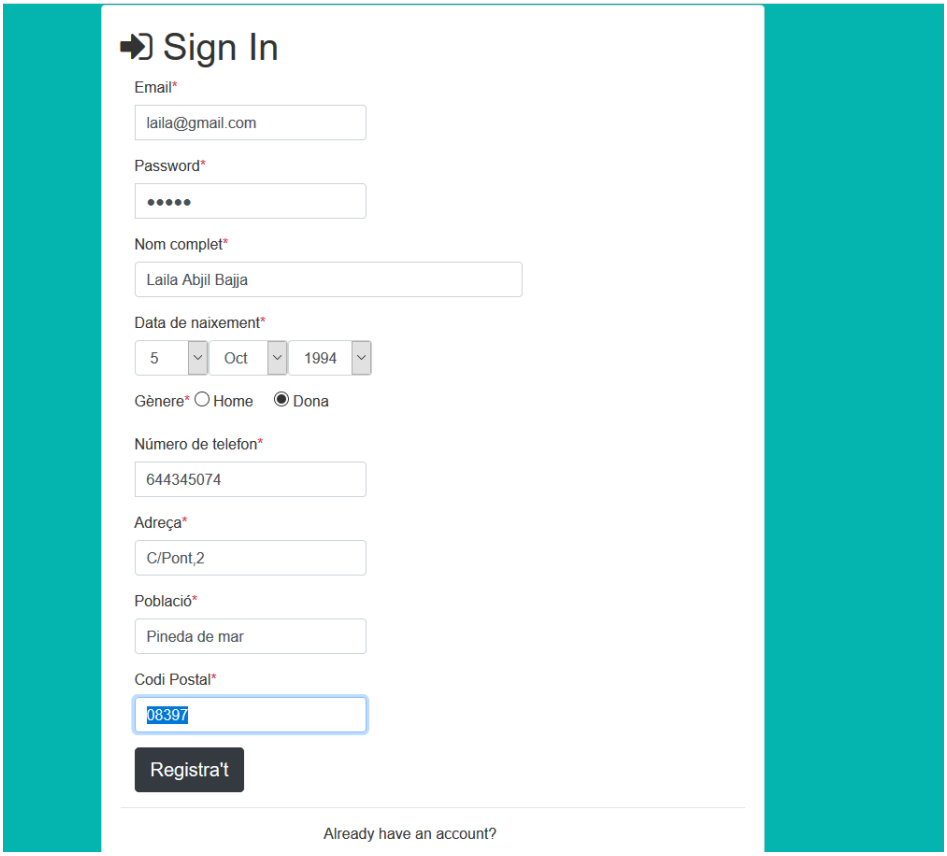


Figura 106: registre

11.2.5 VEURE PERFIL

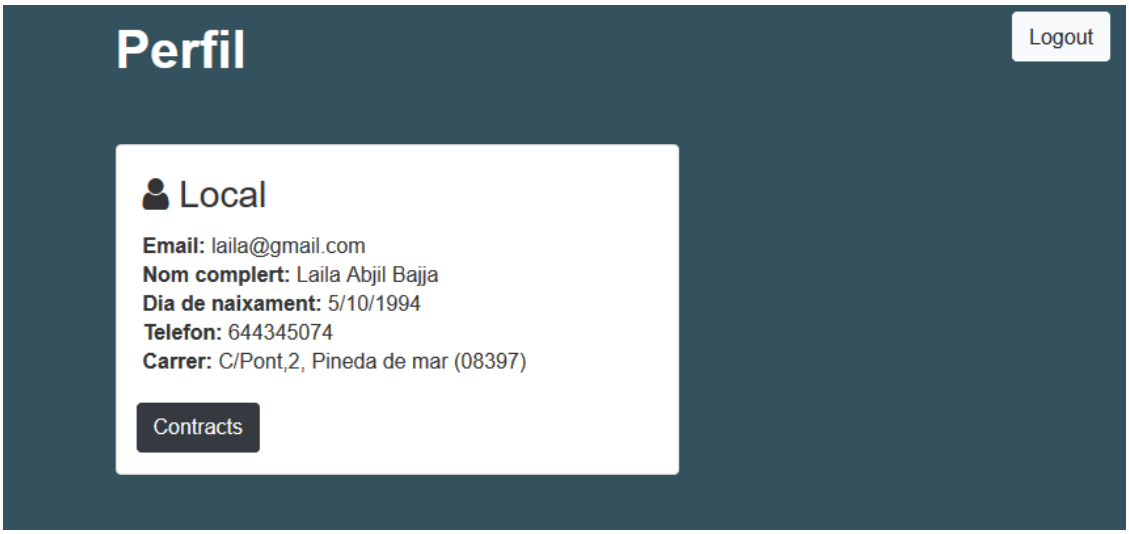


Figura 107: perfil

11.2.6 VEURE CONTRACTES

Figura 108: contractes del usuari

11.2.7 CREAR CONTRACTE

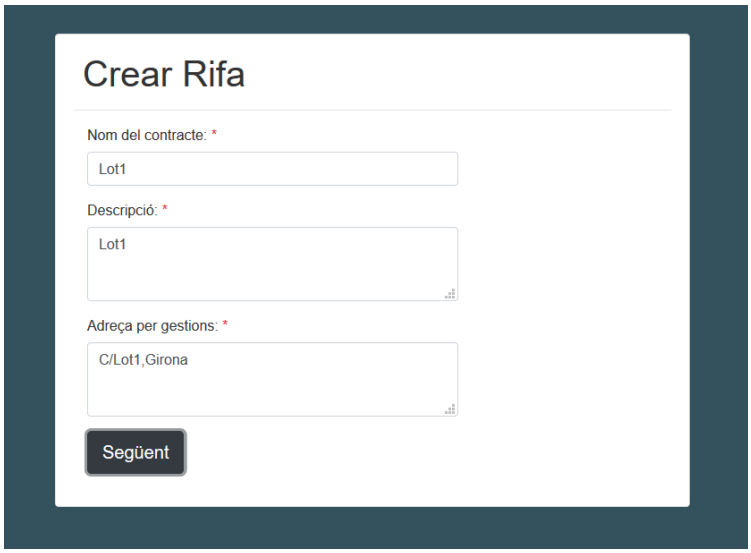


Figura 109: crear un contracte

Un cop cliques a següent, veus que en el blockchain es crea el contracte.

```
Listening on 127.0.0.1:8545
eth_accounts
eth_sendTransaction

  Transaction: 0xa01d55bba9f3503f63a64c80a2c65f2bd97604edba5c8453ba7fe4485344a555
  Contract created: 0x607388403975927d752ffc97044ddba744bee34e
  Gas usage: 710213
  Block Number: 1
  Block Time: Thu Aug 30 2018 12:36:56 GMT+0200 (Hora de verano romance)

eth_newBlockFilter
eth_getFilterChanges
eth_getTransactionReceipt
eth_getCode
```

Figura 110: Transacció Blockchain – Crear contracte

I a la BD es crea un document a la col·lecció *lotteries*.

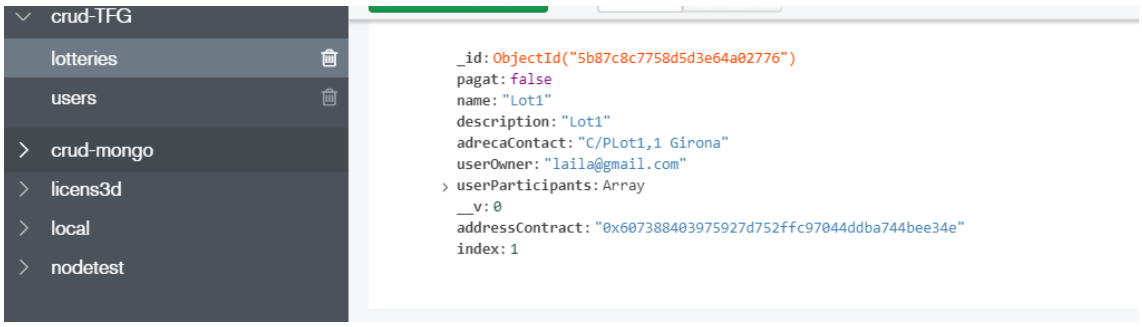


Figura 111: MongoDB – Lotteries

11.2.8 AFEGIR DETALLS A CONTRACTE

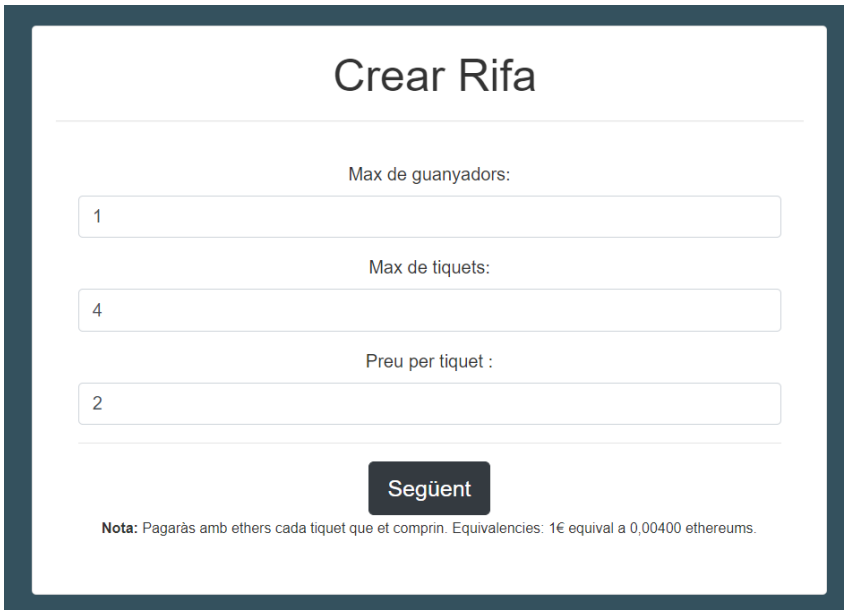


Figura 112: Afegir detalls al contracte

En el blockchain es veuen 2 transaccions. La primera és la transacció que modifica el màxim de guanyadors i el màxim de tiquets; i la segona afegeix el preu del tiquet.

```
eth_sendTransaction

  Transaction: 0xbc7cf2184b8656e3fcc40a00df548821c08d3310c914db506300ed714dcc1bde
  Gas usage: 32269
  Block Number: 2
  Block Time: Thu Aug 30 2018 12:41:34 GMT+0200 (Hora de verano romance)

eth_accounts
eth_sendTransaction

  Transaction: 0x599b57e7ea704f4be339427b92d4cea038c1fcab67e05c1e2ae65becaf00395d
  Gas usage: 42037
  Block Number: 3
  Block Time: Thu Aug 30 2018 12:41:36 GMT+0200 (Hora de verano romance)

eth_call
```

Figura 113: Transacció Blockchain – Afegir detalls al contracte

A la BD s'afegeix el preu del tiquet.

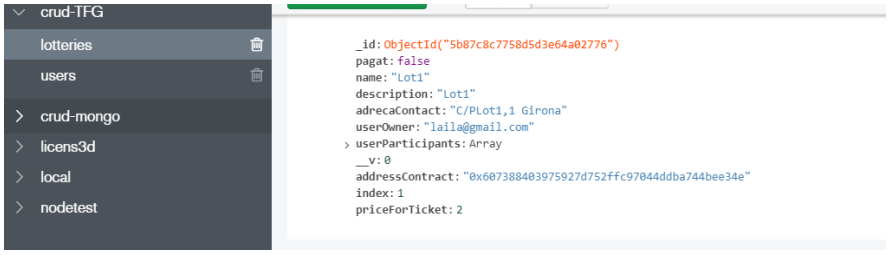


Figura 114: MongoDB – loteries modificat

11.2.9 AFEGIR PERCENTATGES AL CONTRACTE

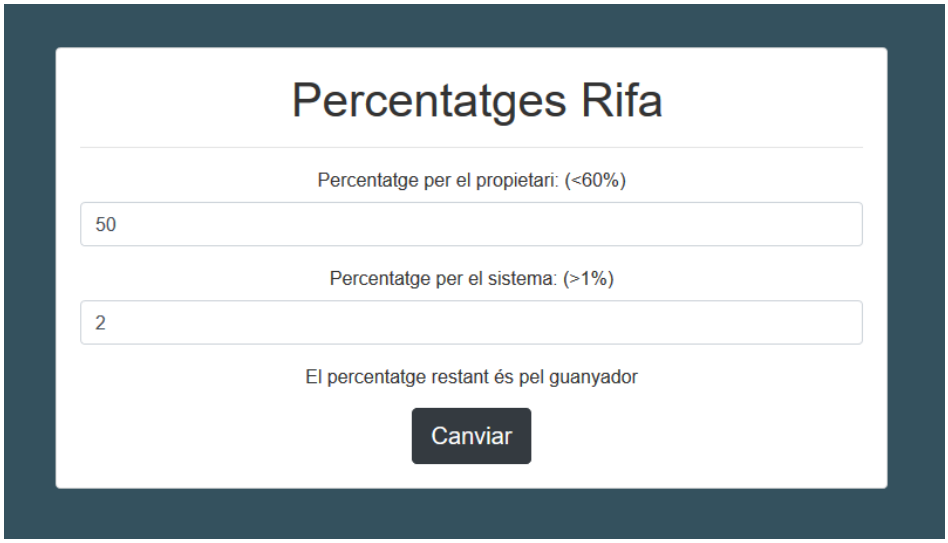


Figura 115: Afegir percentatges

Al clicar “Canviar”, s’executa en el blockchain una transacció que modifica els percentatges del propietari i del sistema.

Figura 116: Transacció Blockchain – Afegir percentatges

11.2.10 VEURE CONTRACTES

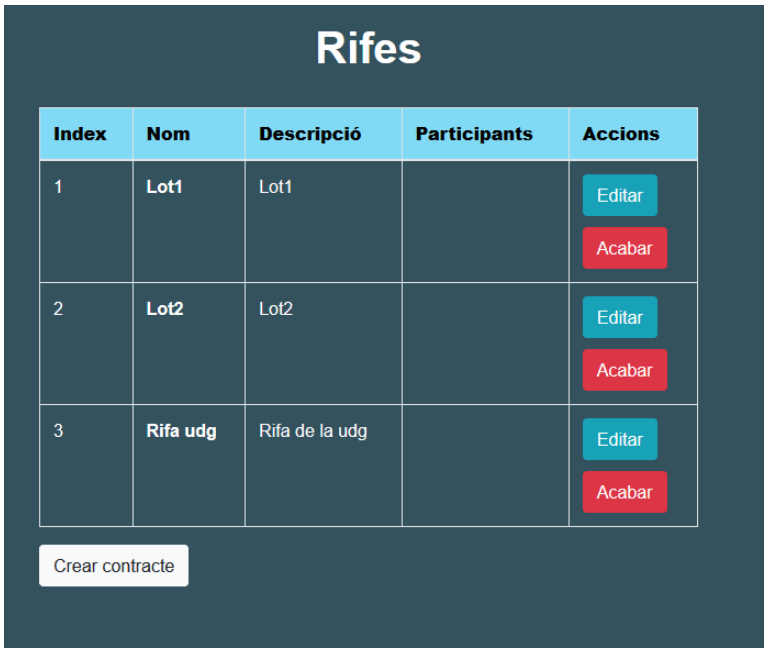


Figura 117: veure contractes

11.2.11 PARTICIPAR EN UNA RIFA

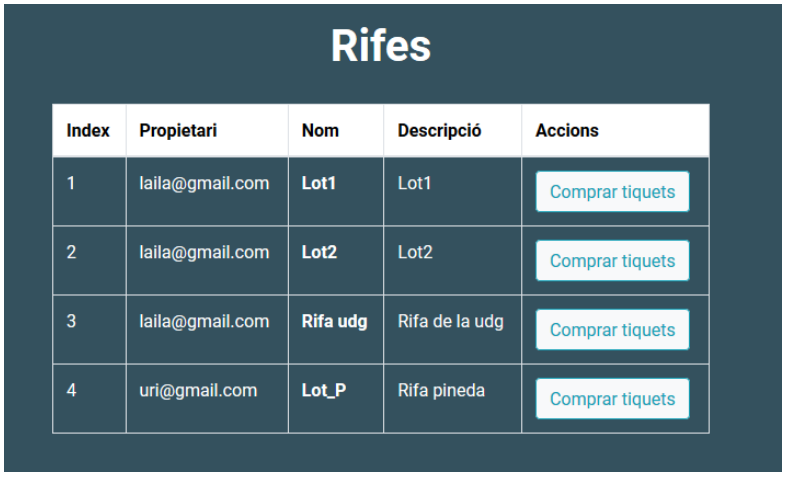


Figura 118: veure tots els contractes

11.2.12 COMPRAR TIQUETS

S'ha clicat a "comprar tiquets" de la rifa "Lot1".

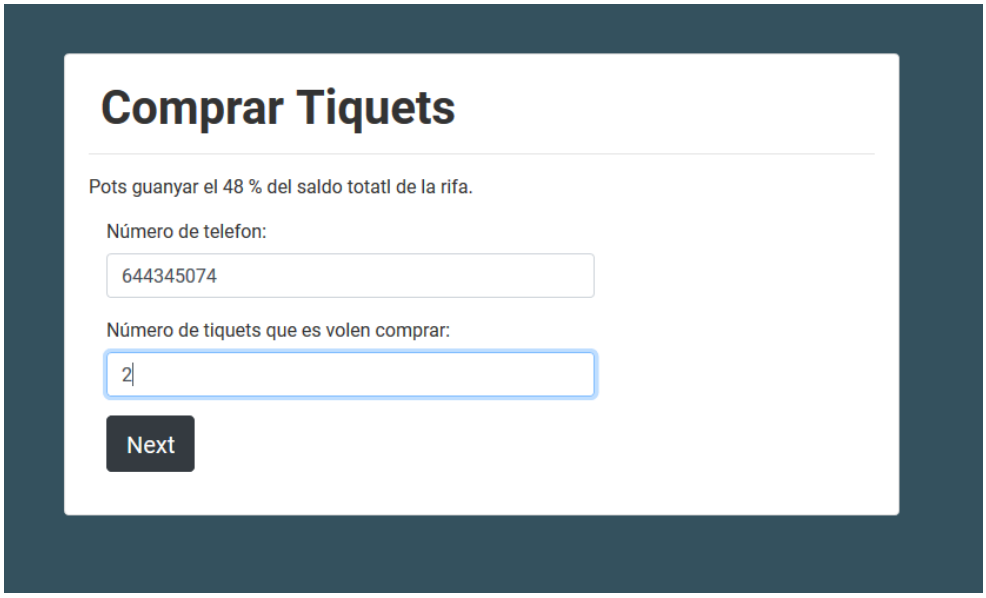


Figura 119: Comprar tiquets

Es mostra la notificació dient que rebràs el missatge al mòbil.

Figura 120: Comprar tiquets - notificació

En el blockchain es veu una transacció per cada tiquet comprat. En aquest cas, hem comprat 2 tiquets i, per tant, veiem dues transaccions.

```
eth_accounts
eth_sendTransaction

  Transaction: 0x1a8488010f8994702b5d726a7082a9ac94e04233604ccda9b60e80c8de669fcd
  Gas usage: 103398
  Block Number: 21
  Block Time: Thu Aug 30 2018 13:02:45 GMT+0200 (Hora de verano romance)

eth_accounts
eth_sendTransaction

  Transaction: 0xd7e6ba476e94fe8b11aecbc1461c553d191452a1e09884278a26bdf2209bfc94
  Gas usage: 58398
  Block Number: 22
  Block Time: Thu Aug 30 2018 13:02:46 GMT+0200 (Hora de verano romance)

eth_accounts
eth_call
```

Figura 121: Transacció Blockchain – Comprar tiquet

Aquí es veu com ha arribat el missatge al mòbil.

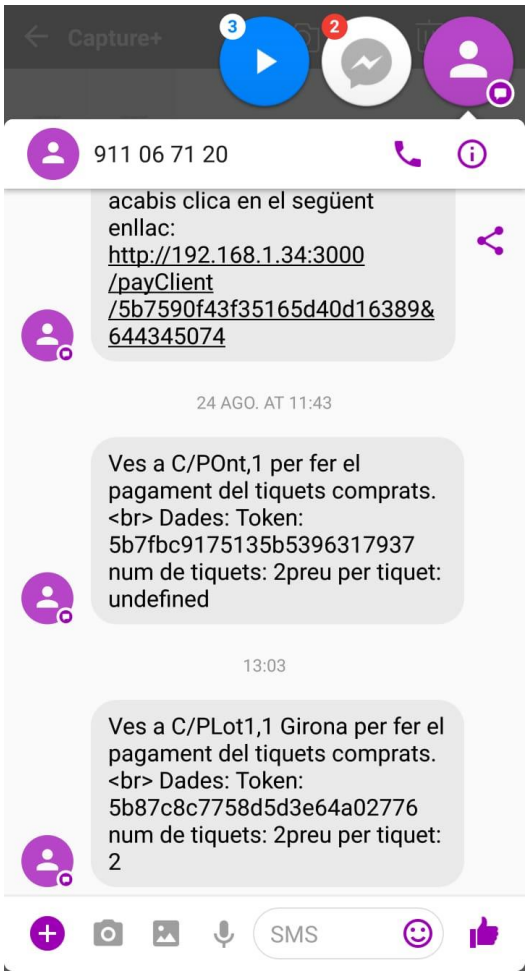


Figura 122: missatge al mòbil del usuari participant amb les dades del tiquet

En les figures 123, 124 i 125 es poden veure els canvis de la BD.

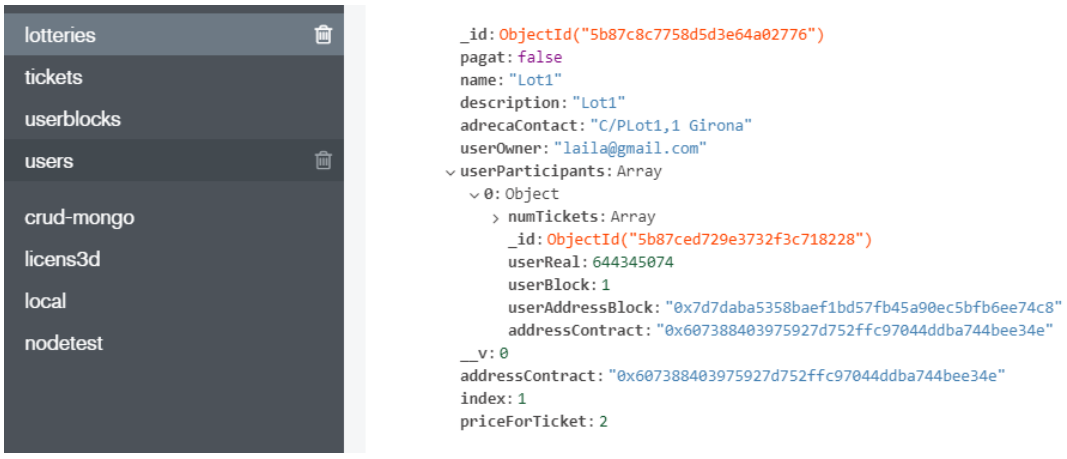


Figura 123: MongoDB - lotteries

Figura 124: MongoDB – tiquets creats

Figura 125: MongoDB – userblocks, usuari participant

11.2.13 EDITAR CONTRACTE

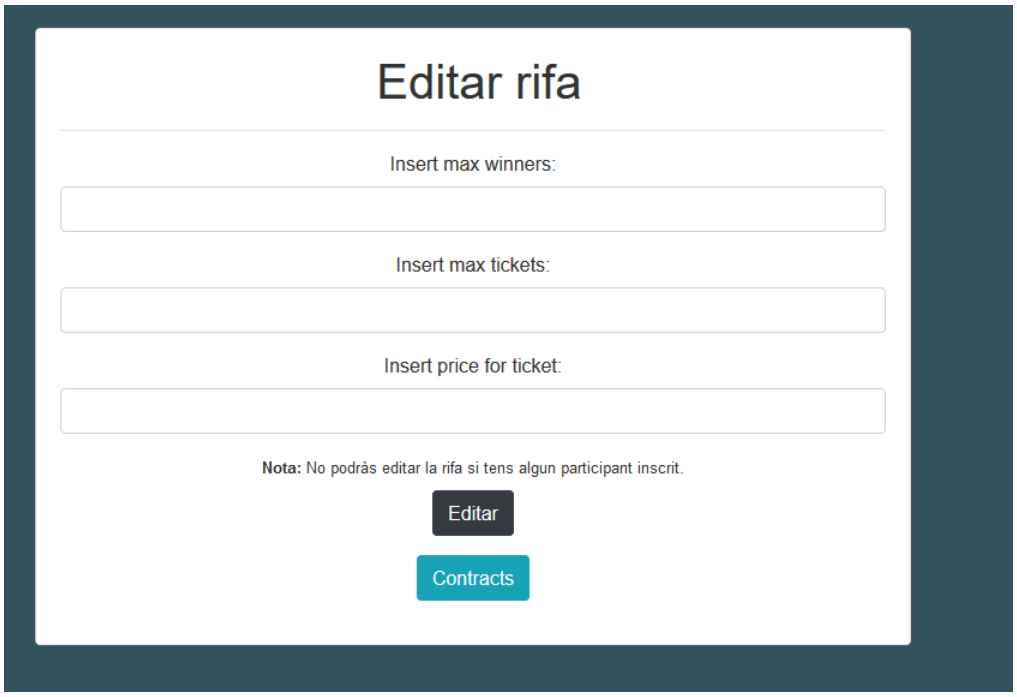


Figura 126: Editar rifa

Al tenir ja un participant, mostra el següent error.

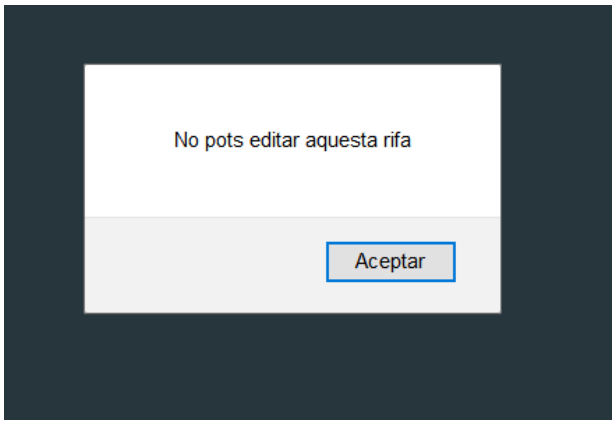


Figura 127: Editar rifa - error

S'edita la rifa "Lot2" que si es pot editar ja que no té cap participant.

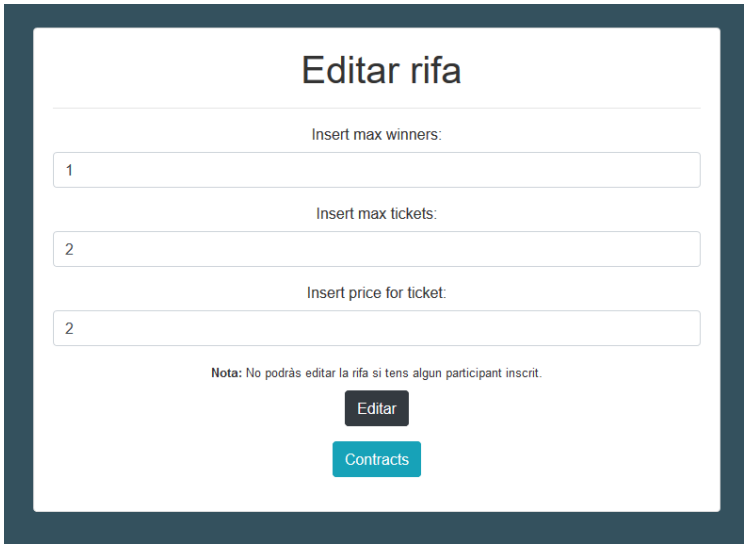


Figura 128: Editar rifa

11.2.14 ACABAR CONTRACTE

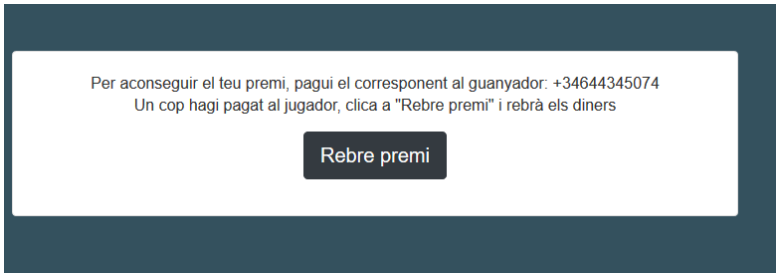


Figura 129: Acabar el contracte

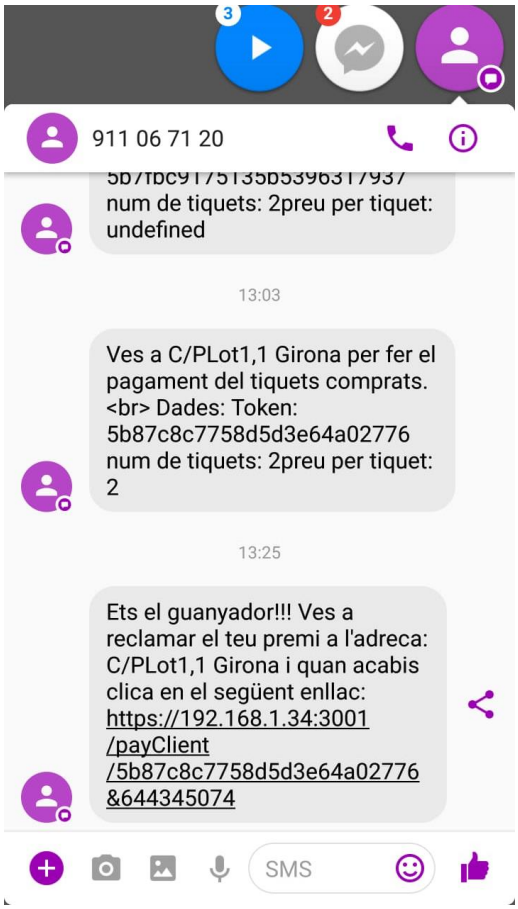


Figura 130: Missatge que rep el guanyador.

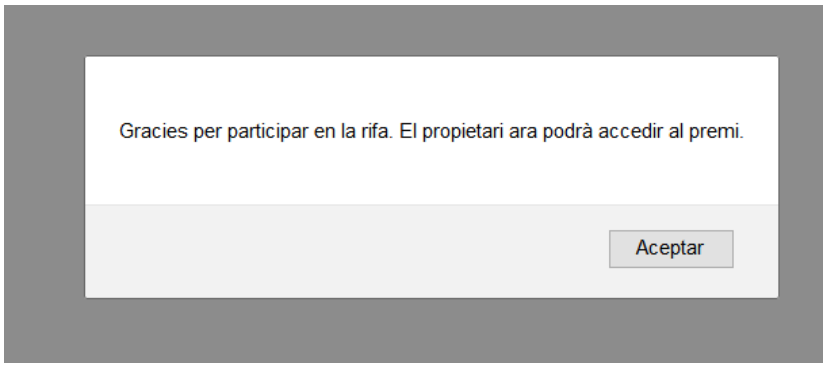


Figura 131: Missatge que es mostra al clicar al enllaç /payClient/5b87c8c7758d5d3e64a02776&644345074

```
eth_accounts
eth_sendTransaction

  Transaction: 0x2ad3c4ab501325b8980cd3a3e05d5f6cbeccb6445fd8eff0f128f7822cb19610
  Gas usage: 51544
  Block Number: 25
  Block Time: Thu Aug 30 2018 13:25:42 GMT+0200 (Hora de verano romance)

eth_accounts
eth_call
```

Figura 132: Transacció Blockchain – Acabar contracte

Al clicar al enllaç rebut, es pot clicar al botó “rebre premi”.

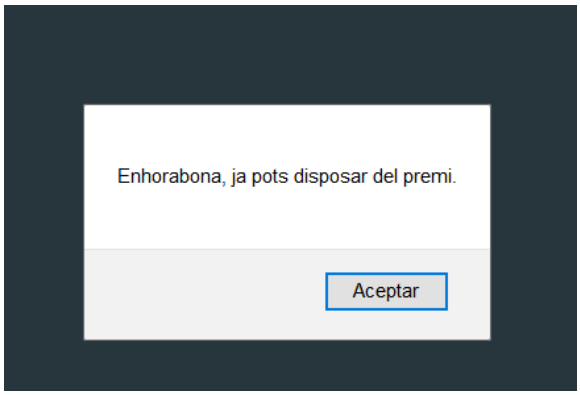


Figura 133: Missatge al rebre el premi.

11.3 DEMOSTRACIONS

Per provar l'aplicació i veure quina resposta tenia davant de la gent, s'ha fet una demo que es pot trobar al enllaç que està compartit en el apartat dels Annexes.

11.3.1 Participants

Els participants de la demostració han sigut 3:

Meriem Abjil

23 anys

Enginyera Informàtica

Judit Garcia

24 anys

Projectista d'espais

Oriol Cardona

28 anys

Tècnic d'Automatització i Robòtica Industrial

11.3.2 Cas d'ús

En aquest cas, la participant *Meriem* es crearà un usuari i crearà una rifa amb les dades següents:

Nom: Rifa Prova

Descripció: rifa de prova

Adreça de gestions: C/Pont,4 Girona

Després afegirà els detalls corresponents.

Màxim de guanyadors: 1

Màxim de tiquets: 5

Preu per tiquet: 2

A continuació, afegirà els percentatges del premi del propietari i del sistema.

Percentatge del propietari: 55

Percentatge pel sistema: 2

Un cop creat el contracte, procedirem a que cada un compri tiquets.

La participant *Judit* comprarà 2 tiquets del contracte creat amb el número *616366904*, la participant *Meriem* comprarà 1 tiquet amb el número *689590737* i el participant *Oriol* comprarà 3 tiquets amb el número *680493036*.

Un cop fet, la participant Meriem anirà al seu contracte i li donarà a "Acabar". Finalment, el guanyador premerà el enllaç enviat i el propietari juntament amb el sistema rebran el premi.

11.3.3 Opinions

“Sí crec que sigui més segur degut a que el sistema està basat en Blockchain”.
Opinió de Meriem.

“La aplicació hauria de ser una mica més intuïtiva” .
Opinió de Judit.

“Sí que compraria amb diners reals, em sento més segura que dipositant els diners a una entitat externa”.
Opinió de Meriem.

“Seria més segur si es pagués electrònicament” .
Opinió de Oriol.

“El Blockchain no es allò que té a veure amb el Bitcoin?”
Opinió de Judit. Al explicar-li en què consistia el Blockchain, es va mostrar més receptiva a confiar amb l’aplicació.

En general han sigut bones opinions, excepte la participant Meriem que ja tenia coneixements sobre Blockchain, els altres dos participants al explicar en què consistia s’han mostrat més receptius per utilitzar l’aplicació.

12 CONCLUSIONS

Els objectius que havíem marcat al principi del projecte eren:

- Crear una aplicació adaptativa amb una interfície senzilla que mostri el sistema.
- Crear el Smart contract amb Solidity.
- Pujar el Smart contract en el Blockchain i lligar-la amb l'aplicació.

Com hem pogut veure en el apartat 10.2, s'han assolit totes les interfícies de totes les funcionalitats.

El Smart Contract ha sigut creat al principi i utilitzat per fer les funcionalitats següents: *crear contracte*, *afegir detalls a contracte*, *afegir percentatges*, *comprar tiquets*, *editar contracte* i *acabar contracte*. Es poden veure més detalladament en el apartat 10.2.

Per últim, hem pogut veure en el cas implementat per demostrar en el apartat 10 que s'han pogut crear els contractes en el Blockchain i al implementar les funcions descrites en el paràgraf anterior s'han creat les funcions que lligaven l'aplicació amb les funcions del Smart contract.

Per tant, podem concloure que s'han pogut assolir tots els objectius inicials de l'aplicació.

Com he comentat al apartat 9.5, la principal desviació en aquest projecte ha sigut alhora de plantejar al principi l'aplicació com una aplicació android i s'ha hagut de canviar per una aplicació web degut als problemes de sincronització entre l'Android i el Blockchain alhora de comprar tiquets.

Cal dir que es podria millor la interfície per què sigui més intuïtiva per la gent que utilitzi l'aplicació i, també podria gestionar-se els pagaments de forma electrònica enlloc d'haver-ho de fer en forma física. També cal saber que per poder fer aquests pagaments, caldria modificar tota l'estructura blockchain ja que s'hauria de fer sobre Blockchain real.

13 TREBALL FUTUR

Algunes de les possibles millores i treballs futurs que crec que es podrien realitzar són:

- Millorar la interfície gràfica perquè sigui més dinàmica i visualment més agradable.

- Fer els pagaments de manera electrònica mitjançant el Stripe, de tal manera que l'usuari no s'hagi de desplaçar per obtenir el premi o pagar els tiquets comprats.

- Passar el Blockchain tester a Blockchain real ja que ara mateix el propietari de la rifa no paga diners reals, sinó que son ficticis. D'aquesta manera, s'hauria d'implementar també un conversor de moneda, que passi els euros a Ethereums i després que pogués pagar amb aquesta moneda. Caldria també canviar el sistema en que està sincronitzat el Blockchain amb l'aplicació.

- Pujar tot el projecte en un servidor extern de producció.

14 ANNEXOS

El projecte està penjat a una carpeta en el Google Drive compartit amb el següent enllaç: https://drive.google.com/drive/folders/1761vJl7QAxkF-4Jx2a1fIktaEH5xVJPC?usp=sharing

En aquest enllaç també hi trobareu la demo feta en el apartat 10.3 i un altre demo creada per demostrar el funcionament de l'aplicació, juntament amb el codi de l'aplicació.

15 BIBLIOGRAFIA

OpenWebinars. (Març 2018). *Qué es la Metodología Agile*. Recuperat de *https://openwebinars.net/blog/que-es-la-metodologia-agile/*

Remix. (Març 2018). *Solidity IDE*. Recuperat de *https://remix.ethereum.org/#optimize=true&version=soljsonv0.4.24+commit.e67f0147.js*

MyEthereumWallet. (Març 2018). *My EtherWallet*. Recuperat de *https://www.myetherwallet.com/*

ETHEREUM. (Març 2018). *Install the Command Line Tools*. Recuperat de *https://www.ethereum.org/cli*

Bit2Me. (Març 2018). *Smart contracts, ¿Qué son, cómo funcionan y qué aportan?.* Recuperat de *https://academy.bit2me.com/que-son-los-smart-contracts/*

Bit2Me. (Març 2018). *Smart Contracts: Una guia para principiantes.* Recuperat de *https://blog.bit2me.com/es/smart-contracts/*

Medium. (Març 2018). *Build Your First Smart Contract*. Recuperat de *https://medium.com/crypto-currently/build-your-first-smart-contract-fc36a8ff50ca*

CriptoNotices. (Març 2018). *¿Qué es la tecnologia de contabilidad distribuïda o blockchain?.* Recuperat de *https://www.criptonoticias.com/informacion/que-es-tecnologia-contabilidad-distribuida-blockchain/*

UDEMY. (Abril 2018). *Desarrollo dApps en la Blockchain de Ethereum, 1*. Recuperat de *https://www.udemy.com/desarrollo-dapps-en-ethereum-1/*

MiEthereum. (Abril 2018). *Guía Completa sobre Ethereum Wallets / Monederos.* Recuperat de *https://miethereum.com/ether/wallet-monedero/*

ETHEREUM. (Abril 2018). *Create a cryptocurrency contract in Ethereum*. Recuperat de *https://www.ethereum.org/token*

web3j (Abril 2018). *Getting Started – web3j 3.4.0 documentation*. Recuperat de *https://docs.web3j.io/getting_started.html*

MEDIUM. (Abril 2018). *A 101 Noob Intro to Programming Smart Contracts on Ethereum*. Recuperat de *https://medium.com/@ConsenSys/a-101-noob-intro-to-programming-smart-contracts-on-ethereum-695d15c1dab4*

Truffle. (Abril 2018). *Sweet Tools for Smart Contract*. Recuperat de *https://truffleframework.com/*

CodeBurst.io. (Maig 2018). *Build Your First Ethereum Smart Contract with Solidity – Tutorial*. Recuperat de *https://codeburst.io/build-your-first-ethereum-smart-contract-with-solidity-tutorial-94171d6b1c4b*

Medium. (Maig 2018). *Full Stack Hello World Voting Ethereum Dapp Tutorial*. Recuperat de *https://medium.com/@mvmurthy/full-stack-hello-world-voting-ethereum-dapp-tutorial-part-1-40d2d0d807c2*

MetaMask. (Maig 2018). *Metamask*. Recuperat de *https://metamask.io/*

Mozilla Firefox. (Juny 2018). *Introducción a Express/Node*. Recuperat de *https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction*

Fazt Code. (Juny 2018). *Nodejs y Mongodb CRUD | Aplicación web desde cero con Express, Nodejs, Mongodb, mongoose, bootstrap4* [Vídeo]. Recuperat de *https://www.youtube.com/watch?v=3J925fRl_UE*

Medium. (Juny 2018). *Starting with Authentication (A tutorial with Node.js and* BongoDB. Recuperat de *https://medium.com/createdd-notes/starting-with-authentication-a-tutorial-with-node-js-and-mongodb-25d524ca0359*

Font Awesome. (Juny 2018). *Icons | Font Awesome*. Recuperat de *https://fontawesome.com/icons?from=io*

Bootstrap. (Juny 2018). *The most popular HTML, CSS, and JS librari in the world*. Recuperat de *http://getbootstrap.com/.*

NodeJs. (Juny 2018). *NodeJs*. Recuperat de *https://nodejs.org/es/*

W3Schools.com. (Juny 2018) *.Node.js Get Started*. Recuperat de *https://www.w3schools.com/nodejs/nodejs_get_started.asp*

Kiessling, M. (Juny 2018). *El Libro para Principiantes en Node.js >> Un tutorial completo de node.js*. Recuperat de *https://www.nodebeginner.org/index-es.html*

Tutorialspoint. (Juny 2018). *Node.js Tutorial.* Recuperat de *https://www.tutorialspoint.com/nodejs/*

Mozilla Firefox. (Juny 2018). *Express Web Framework (Node.js/JavaScript)*. Recuperat de *http://expressjs.com/es/*

CloseBrace. (Juny 2018). *Creating a Simple RESTful Web App with Node.js, Express, and MongoDB*. Recuperat de *https://closebrace.com/tutorials/2017-03-02/creating-a-simple-restful-web-app-with-nodejs-express-and-mongodb*

Google. (Agost 2018). *Draw.IO.* Recuperat de *https://www.draw.io/#G1WE_Py6nl1sjpJNEJNIKdvx_XMaoekfOn*

16 MANUAL D'USUARI I/O INSTAL·LACIÓ

Les instruccions a seguir per instal·lar el projecte, un cop tens els requisits especificats a l'apartat 7.2 són els següents:

- Nodejs
    - Descargar desde: https://nodejs.org
- OpenSSL
    - Descargar versió v1.0.2 desde: http://slproweb.com/products/Win32OpenSSL.html
- MongoDB:
    - Descargar l'ultima versió desde: https://www.mongodb.com/download-center#community
    - Instal·lar per defecte en "C:\Program Files\MongoDB"
    - Afegir la carpeta bin en el path "C:\Program Files\MongoDB\Server\3.4\bin"
    - Crear la carpeta que contindran les dades del MongoDB a "D:\MongoDB" amb l'estructura següent:
        - Carpeta "*Data*"
        - Carpeta "*Logs*"
        - Fitxer de configuració (copiar el proporcionat del mongo *mongo.config*).
- Obrir el CMD amb privilegis d'administració i executar la comanda:

`mongod.exe --config D:\MongoDB\mongo.config --install`

- Iniciar el servei MongoDB server desde:
    - Serveis->MongoDB->Start
    - En el CMD amb privilegis d'administració i executar la comanda:

      `net start mongodb`
- Instal·lar les dependències globals:
    - Obrir el CMD amb privilegis d'administració, situart-te a la carpeta del projecte i executar:
        - `npm install --global --production windows-build-tools`
        - `npm install --global truffle`
        - `npm install --global ganache-cli`
- Instal·lar les dependències del projecte:
    - Obrir el CMD amb privilegis d'administració, situart-te a la carpeta del projecte i executar:
        - npm init
        - npm install express mongoose ejs morgan
        - npm install -g nodemon
        - npm install body-parser bcrypt-nodejs express-session
        - connect-flash cookie-parser passport passport-local
        - npm install stripe express pug body-parser
        - npm install --save web3
- Executar el projecte
    - Obrir el CMD amb privilegis d'administració, situart-te a la carpeta del projecte i executar la comanda:

```
ganache-cli account="0xbbe22d945c91709e3e7a110ac52404a711b012bd02e0ba8c26573d0277038af1,
100000000000000000000000", --
account="0xb5f843140dafa4b25a1c86c3a8708ccd545dcde426fdc4c0ad1d5da8450d7a6a,
100000000000000000000000", --
account="0x28bb6847c01825da07b076a8b22787494a40fd0c8b4e662a40f764a2132f0985,
100000000000000000000000", --
account="0xa25611b736cb36d5b971c293eca6a3f94e01894b9cd1c76ab7af64d13976008a,
100000000000000000000000", --
account="0x7cc2087b991fd589942ce53c619b22955c22e3e0cfe49b74fe417c669f259fff,
100000000000000000000000", --
account="0xca6128093e3ca570e715ac1b1cbbc74397ce77cea49236352650f8516dd215f7,
100000000000000000000000", --
account="0x560b668d8374998fa0585c955497b62425f250285b1e48b34697801476f4a685,
100000000000000000000000", --
account="0xb6dc849889742fa0819551d11d2c9c71e314301548f00f2b4e6ed5c552706636,
100000000000000000000000", --
account="0x61fb23537d2e6181e30ae4fdd15b51e1115d3793991080e1a3874d69b2647e7a,
100000000000000000000000", --
account="0x379b3155ced61fb0a3426ea25a3d021ee031393bf84fb260cd10c949e49ef144,
100000000000000000000000", --
account="0xbdf40c2c4048184a26e7916ee50454afe4b7903ee8a6712db17002f4c5a067b5,
100000000000000000000000", --
account="0x66c14ddb845e629975e138a5c28ad5a72a73552ea65b3d3ec99810c82751cc35,
100000000000000000000000" --unlock "0x5DC0Cc7EC79a88c5a90bcdA06CCf0F85829b87ca" --unlock
"0x7d7dABA5358BaEF1bd57Fb45A90Ec5bFb6EE74c8" --unlock
"0x6aB20665288cd8Dd3919d89F730aeba3f7DBb949" --unlock
"0x24B84237971f9ee9D6ff4fA2a34D84c2Cb18E995" --unlock
"0x2fF9AB3ca31A69907235e9a0aa2fd5551D381Da6" --unlock
"0xBfF17713DF340983ff5C264f336450ea83e235B6" --unlock
"0xBd92617BbA86D2F635E9Bb887F6BbafE9F3A6cf5" --unlock
"0x42B50388c6F3Ec68747adA8aC3fE5dCB21D6d5c0" --unlock
"0x1AA4490aC708d501A63E394Ca5212eaeaD82C3Ba" --unlock
"0x68Ea7fd2E7D65EC1C9B653b36CF204ca28462406" --unlock
"0xc0Eb469a948C5b7A163Df6e9bCa0a7115a74B7a9" --unlock
"0xfF1FbFe19950b12ACe5101dE1459d765e062Cb59" --gasLimit "9999999999"
```

- Obrir un altre CMD amb privilegis d'administració, situart-te a la carpeta del projecte i executar la comanda:

  npm run dev