



# **testDEX: Un DEX simple en la xarxa Elrond**

**Sergio González Rubio**

Màster universitari de Ciberseguretat i Privadesa  
Sistemes de blockchain

**Consultor:** Josep Lluís de la Rosa Esteva

**Professor:** Victor Garcia Font

1 de juny de 2022



Aquesta obra està subjecta a una llicència de  
Reconeixement-NoComercial-SenseObraDerivada  
[3.0 Espanya de Creative Commons](#)

## FITXA DEL TREBALL FINAL

<b>Títol del treball:</b>	<i>testDEX: Un DEX simple en la xarxa Elrond</i>
<b>Nom de l'autor:</b>	<i>Sergio González Rubio</i>
<b>Nom del consultor/a:</b>	<i>Josep Lluís de la Rosa Esteva</i>
<b>Nom del PRA:</b>	<i>Victor Garcia Font</i>
<b>Data de lliurament (mm/aa-aa):</b>	<i>juny/2022</i>
<b>Titulació o programa:</b>	<i>Màster universitari de Ciberseguretat i Privadesa</i>
<b>Àrea del treball final:</b>	<i>Sistemes de blockchain</i>
<b>Idioma del treball:</b>	<i>Català</i>
<b>Paraules clau:</b>	<i>Elrond, blockchain, EGLD, smart contract, swap</i>
<b>Resum del treball:</b>	L'adveniment Bitcoin i posteriorment d'Ethereum varen definir les bases de la tecnologia <i>blockchain</i> . Cal destacar que han suposat una disruptió tecnològica, econòmica i fins i tot política, despertant un interès acadèmic creixent i nous casos d'ús. En aquest context, es presenten reptes per aconseguir una descentralització plena que sigui capaç de substituir els sistemes centralitzats tradicionals de transaccions electròniques i moviment de capitals. Alt consum d'energia, baixa taxa de transaccions, escalabilitat, seguretat o interoperabilitat són alguns dels reptes plantejats per les tecnologies actuals de <i>blockchain</i> que resol Elrond. Aquest treball presenta un exemple d'implementació de <i>swaps</i> entre criptovalors amb l'objectiu de comprovar que la proposta de <i>blockchain</i> pública d'Elrond ha vençut les limitacions actuals de Bitcoin i Ethereum.
<b>Abstract:</b>	The advent of Bitcoin and later of Ethereum described the foundations of blockchain technology. It should be noted that they have led to a technological, economical and even political disruption, arousing growing academic interest and new use cases. In this context, we are being challenged to achieve full decentralization that is capable of replacing traditional centralized systems of electronic transactions and movements of capital. High energy consumption, low transaction rates, scalability, security or interoperability are some challenges of current blockchain technologies that Elrond solves. This paper presents an example of the implementation of swaps between tokens in order to verify that Elrond's public blockchain proposal has overcome the current limitations of Bitcoin and Ethereum.

# Índex

<b>1</b>	<b>Introducció</b>	<b>6</b>
1.1	Context i justificació del treball . . . . .	6
1.2	Objetius del treball . . . . .	8
1.3	Enfocament i mètode seguit . . . . .	8
1.4	Planificació del treball . . . . .	9
1.5	Breu sumari de productes obtinguts . . . . .	11
1.6	Breu descripció dels altres capítols de la memòria . . . . .	11
<b>2</b>	<b>Revisió de la tecnologia d'Elrond</b>	<b>12</b>
2.1	Visió general de l'arquitectura . . . . .	12
2.2	Entitats . . . . .	13
2.3	Cronologia . . . . .	14
2.4	<i>Secure Proof of Stake</i> . . . . .	14
2.5	<i>Adaptive State Sharding</i> . . . . .	15
2.6	Transaccions entre <i>shards</i> . . . . .	17
2.7	<i>La màquina virtual d'Elrond</i> . . . . .	19
2.8	Execució de contractes intel·ligents en l'arquitectura amb <i>shards</i> . . . . .	21
2.9	Xarxes Elrond disponibles . . . . .	21
2.10	Muntar un node validador . . . . .	22
2.11	Carteres . . . . .	24
2.12	Maia Exchange . . . . .	27
2.13	Altres punts destacables . . . . .	28
<b>3</b>	<b>testDEX</b>	<b>31</b>
3.1	Anàlisi . . . . .	31
3.1.1	DEX i AMM . . . . .	31
3.1.2	Les variants de DEX i AMM a testDEX . . . . .	34
3.1.3	Requeriments funcionals . . . . .	34
3.1.4	Requeriments no funcionals . . . . .	35
3.2	Disseny . . . . .	35
3.2.1	Casos d'ús . . . . .	35
3.2.2	<i>Smart contract</i> . . . . .	37
3.2.3	Arquitectura . . . . .	42
3.2.4	Pantalles . . . . .	43
3.3	Implementació . . . . .	44
3.3.1	Repositoris disponibles . . . . .	44
3.3.2	Aspectes a destacar de la implementació . . . . .	44
3.3.3	<i>Correcció de la constant k per l'error introduït pel redondeig</i> . . . . .	45
3.4	Posada en producció . . . . .	46
3.4.1	Desplegament del contracte intel·ligent . . . . .	46
3.4.2	Desplegament de la dApp . . . . .	48

3.5	Proves . . . . .	48
3.5.1	Sobre el contracte intel·ligent . . . . .	48
3.5.2	Sobre la dApp . . . . .	53
3.5.3	Sobre el rendiment de la <i>devnet</i> d'Elrond . . . . .	53
<b>4</b>	<b>Conclusions</b>	<b>60</b>
<b>5</b>	<b>Glossari</b>	<b>62</b>
<b>6</b>	<b>Bibliografia</b>	<b>63</b>
<b>7</b>	<b>Annexos</b>	<b>66</b>

## Índex de figures

1	Diagrama de Gantt del projecte (creat amb GNOME Planner 0.14.6). . . . .	10
2	Temporalització del projecte. . . . .	10
3	<i>Secure Proof of Stake.</i> (Font: Modificació d'imatge dins l'apartat corresponent a SPoS a [1]). . . . .	16
4	Estructura d'un bloc en la xarxa Elrond. (Font: Imatge dins l'apartat corresponent a SPoS a [1]). . . . .	16
5	La xarxa d'Elrond configurada amb un (a), dos (b) i tres (c) <i>shards</i> . . . . .	18
6	Redundància als <i>shards</i> entre èpoques. (Font: Captura de [2]). . . . .	19
7	Exemple d'execució de transacció entre dos <i>shards</i> . (Font: Fig. 4 de [2]). . . . .	20
8	Exploradors de les xarxes disponibles a Elrond. . . . .	22
9	Mètodes d'accés al moneder web. . . . .	25
10	Maia DeFi Wallet. . . . .	25
11	Maia per iPhone. . . . .	26
12	Maia Exchange. . . . .	28
13	Tipus de DEX. (Font: [3]). . . . .	31
14	Interaccions del propietari amb el contracte intel·ligent. . . . .	35
15	RF 5. Operacions de compra i venda de <i>tokens</i> ( <i>swaps</i> ). . . . .	36
16	Parells donats d'alta preparats per a intercanvis. . . . .	36
17	Valor actual de la constant <i>k</i> i preus de compra-venda dels parells de <i>tokens</i> . . . . .	36
18	Requeriment funcional 10. . . . .	37
19	Estructura del contracte intel·ligent. . . . .	38
20	Arquitectura de testDEX. (Font: Elaboració pròpria inspirada en [1]). . . . .	42
21	Finestres “Login” i “Claim”. . . . .	43
22	Finestres “Fund” i “Trade”. . . . .	43
23	Els dos tipus de transferències d'actius en el codi font de la dApp. . . . .	45
24	Funció emprada en les transferències de la Fig. 23. . . . .	46
25	Consultes a l'API REST de la devnet d'Elrond. . . . .	46
26	Error acumulat en la constant <i>k</i> . . . . .	47
27	Exemple d'implementació de l'ajustament de la constant <i>k</i> . . . . .	47
28	<i>Snippets</i> en l'IDE Visual Studio Code. . . . .	49
29	<i>Snippets</i> amb trucades a mètodes emprant <i>erdpy</i> . . . . .	50
30	Execució de proves per afegir un fons de liquiditat. . . . .	53
31	Fons de liquidesa creats. . . . .	54
32	Recuperació de fons de liquidesa i beneficis generats. . . . .	54
33	<i>Swaps</i> realitzats segons allò descrit en la secció 3.1.1. . . . .	55
34	Resultat del <i>script</i> de Python executat per a 5 transaccions. . . . .	56

35	Fons de liquidesa ABC-xEGLD abans i després d'executar el <i>script</i> . . . . .	57
36	Resultat del <i>script</i> de Python executat per a 1000 transaccions (es mostren només els darrer <i>hashes</i> ). . . . .	57
37	Resultat del <i>script</i> de Python executat per a 2000 transaccions (es mostren només els darrer <i>hashes</i> ). . . . .	57
38	Resultat del <i>script</i> de Python executat per a 5 transaccions amb un remitent que no és el propietari del <i>smart contract</i> però que es troba al mateix <i>shard</i> . . . . .	58
39	Resultat del <i>script</i> de Python executat per a 2000 transaccions amb un remitent que no és el propietari del <i>smart contract</i> però que es troba al mateix <i>shard</i> . . . . .	58
40	Resultat del <i>script</i> de Python executat per a 5 transaccions amb un remitent que no és el propietari del <i>smart contract</i> i que a més no es troba al mateix <i>shard</i> . . . . .	58
41	Resultat del <i>script</i> de Python executat per a 2000 transaccions amb un remitent que no és el propietari del <i>smart contract</i> i que a més no es troba al mateix <i>shard</i> . . . . .	59

# 1 Introducció

## 1.1 Context i justificació del treball

El *white paper* de **Bitcoin** fou presentat l'any 2008 per una o diverses persones ocultes rere el pseudònim de Satoshi Nakamoto [4]. Tot i que existien intents anteriors<sup>1</sup>, fou la primera solució “creïble” [8] que va permetre transferir fons sense la intervenció de tercera parts (cosa que inclou a bancs centrals encarregats d'emetre la moneda). Gràcies a la implementació d'un programari client –lliure i de codi obert– s'estableix una xarxa d'igual a igual (*peer-to-peer*) on es connecten nodes –no controlats– que verifiquen i emmagatzemem en una base de dades pública la comptabilitat dels moviments entre usuaris de la xarxa. Les operacions s'agrupen en blocs, que s'enllacen i xifren per assegurar que no hi hagi modificacions malicioses posteriors. Els càlculs per realitzar el xifratge tenen cert nivell de dificultat i comporten una despesa energètica considerable, per aquest motiu s'incentiva els nodes “miners” amb una recompensa quan aconsegueixen crear un nou bloc a la cadena. Per això es coneix aquesta tecnologia com a *blockchain*<sup>2</sup>. Així mateix, com afirma V. Buterin [8], l'altra gran aportació és la manera com aplica el concepte de *Proof-of-Work* (PoW) perquè els nodes arribin a un consens per validar les transaccions solucionant certs problemes (com, per exemple, el de la doble despesa). Amb PoW es pretén evitar que nodes de la xarxa tinguin comportaments indesitjats. Com ja s'ha remarcat, a Bitcoin els càlculs de xifratge per crear un nou bloc comporten molta feina computacional, però la validació d'un bloc ja creat requereix un esforç molt inferior. Si dos nodes distribueixen simultàniament diferents versions del següent bloc, el que tingui la cadena més llarga serà el que els nodes acceptin com a vàlid i descartaran la resta [4]. Finalment, el protocol de Bitcoin també disposa d'un llenguatge de *scripting*, encara que amb limitacions importants (per exemple, no és Turing complet i no té estat [8]).

El 2014 V. Buterin va presentar el *white paper* d'**Ethereum** [8] (la xarxa es va posar en producció el 30 de juliol de 2015 [11]). Igual que passa amb Bitcoin, s'utilitza el concepte PoW com a mecanisme de consens entre nodes, però s'incorpora un llenguatge anomenat “Solidity” (que sí que és Touring complet) per a la creació de “*smart contracts*” i d'aplicacions descentralitzades (o “dApps”). Els contractes intel·ligents són programes desplegats (o guardats) a la cadena de blocs que s'executen automàticament quan es compleixen certes condicions. A més, una dApp és una aplicació que funciona sense la necessitat de servidors centrals (gràcies a la tecnologia descrita). En el moment en què redacto aquestes línies, la versió d'Ethe-

---

<sup>1</sup>Per exemple el “*b-money*” de W. Dai (1998) [5], el “*Reusable Proofs of Work*” de H. Finney (2005) [6] o el “*Bit gold*” de N. Szabo (2008) [7].

<sup>2</sup>No és l'objectiu del present treball explicar els sistemes de *blockchain*, però una gran exposició gràfica la realitza A. Brownworth [9][10].

reum 2.0 “Serenity” encara no està completament desenvolupada. Aquest nou *fork* ha d’introduir millores com substituir el PoW per *Proof-of-Stake* (PoS), per reduir el consum d’energia, o com aplicar tècniques de *sharding* per augmentar l’eficiència (la versió actual de la xarxa no arriba a suportar les 20 transaccions per segon [12]).

Amb Bitcoin i Ethereum consolidats, “The Elrond Team” va publicar el 19 de juny de 2019 el *white paper* d’una nova solució de cadena de blocs pública sota el títol “**Elrond: A Highly Scalable Public Blockchain via Adaptive State Sharding and Secure Proof of Stake**”. Entre els reptes que plantejaren [1][2]:

- Descentralització plena.
- Seguretat robusta de les transaccions, prevenint qualsevol vector d’atac conegit.
- Alta escalabilitat, arribant al nivell de rendiment d’algun dels serveis equiparables amb arquitectura centralitzada.
- Eficiència a tots els serveis de xarxa amb el mínim consum energètic i esforç computacional.
- Millora de l’emmagatzematge i la sincronització de dades.
- Interoperabilitat entre cadenes de blocs des del disseny.

Tot i la data de publicació del *white paper*, l’equip d’Elrond assegura que el seu *mainnet* és actiu des de l’any 2018. A més, citen com a aconseguides en la data de redacció del present treball les següents fites [1][13]:

- Primera arquitectura de *blockchain* en producció amb fragmentació d’estat (*state sharding*).
- 1,5k TPS (escalable a més de 100k TPS<sup>3</sup>), latència de 6s i cost de \$0,001 per transacció.
- Maiar App<sup>4</sup> (moneder mòbil d’Elrond), Elrond Web Wallet<sup>5</sup> i Maiar DeFi Wallet<sup>6</sup>.
- *Smart Contracts, Staking & Delegation, Tokens*.
- Maiar DEX<sup>7</sup>.
- DeFi 2.0: Préstecs, sintètics.

---

<sup>3</sup>S’han fet proves a la *testnet* amb pics de més de 260k TPS [1].

<sup>4</sup><https://maiar.com/>

<sup>5</sup><https://wallet.elrond.com/>

<sup>6</sup>Per aconseguir el *plugin* per al teu navegador: <https://getmaiar.com/defi>

<sup>7</sup><https://maiar.exchange/>

- Validat mitjançant múltiples auditories per part de l'empresa Trail of Bits<sup>8</sup> i d'altres.

Referent al darrer punt, he cercat informació sobre quines auditories s'han fet i la informació que he trobat és escassa (pareix que per qüestions de seguretat han estat majoritàriament auditories internes [14]). Sí que apareix un acord de col·laboració per emprar les eines de l'empresa Runtime Verification<sup>9</sup> i que s'han ofert recompenses a *white-hat hackers* [15][16].

Tot i que hi ha altres projectes prometedors com Solana, Avalanche, Tron o Tezos [17], en la meva opinió, l'evolució d'Elrond des de l'aparició del seu *white paper* dona peu a parar el nostre interès acadèmic en aquest projecte i no en els altres. És per aquest motiu que es proposa desenvolupar un **DEX** molt simple que permeti fer intercanvis entre diferents criptomonedes emprant el protocol *automated market makers* (AMM) [18] amb la intenció de **posar a prova la xarxa d'Elrond**.

## 1.2 Objetius del treball

A nivell molt genèric, els quatre grans objectius que pretenc assolir són:

- L'anàlisi i estudi de la tecnologia de *blockchain* d'Elrond des d'un punt de vista acadèmic.
- La definició d'una proposta d'arquitectura per desenvolupar dApps en la xarxa d'Elrond.
- El desenvolupament d'un exemple pràctic usant l'arquitectura anterior.
- La definició de proves per comprovar característiques de la xarxa d'Elrond.

Vull destacar en aquest apartat que no he trobat cap altre treball acadèmic sobre la xarxa d'Elrond en llengua catalana. S'ha de destacar que el contingut de la secció 2 és una tasca de selecció, síntesi i traducció de les fonts citades en les respectives subseccions (bàsicament, [1] i [2]). Òbviament, l'autor no va participar en la creació de la xarxa d'Elrond i, per tant, l'autoria del conceptes que s'exposen és de The Elrond Team.

## 1.3 Enfocament i mètode seguit

Després de revisar ràpidament l'estat de l'art de les tecnologies *blockchain*, ha estat una aposta personal fer servir Elrond. És un projecte novedós i sé que, per aquest motiu, disposaré de menys bibliografia que amb Bitcoin

---

<sup>8</sup><https://www.trailofbits.com/>

<sup>9</sup><https://runtimeverification.com/>

o Ethereum. Aquest darrer fet espero que, més que ser una cosa negativa, em permeti fins i tot trobar nous objectius a mesura que vagi investigant. Així mateix, una bona planificació del projecte és ben necessària, on vull destacar que la fase d'investigació es realitzarà de forma paral·lela a la resta.

Les **fases** que es desenvoluparan són les següents:

- **Plantejament del problema:** Es duu a terme una entrevista amb Josep Lluís de la Rosa Esteva i es pacta desenvolupar un **DEX** emprant el protocol AMM amb el propòsit de provar i comprovar algunes de les característiques de la tecnologia d'Elrond. Com s'ha comentat, previ a l'entrevista, va haver-hi una feina d'investigació ràpida sobre l'estat de l'art en els sistemes *blockchain*.
- **Pla de treball:** Es definiran els recursos necessaris per fer el projecte, les tasques a fer i la seva temporalització.
- **Revisió de la tecnologia d'Elrond:** Es descriurà a tall de resum la tecnologia i s'enumeraran els recursos que hi ha disponibles per desenvolupar en aquesta xarxa.
- **testDEX:**
  - **Anàlisi:** Definició dels requeriments i dels models relacionats amb aquests.
  - **Disseny:** Realització dels models que defineixen el disseny del sistema i l'arquitectura.
  - **Implementació:** Escriptura del codi font.
  - **Posada en producció:** Desplegament en producció del projecte.
  - **Proves:** La fase de proves es durà a terme de forma paral·lela a la implementació i a la de posada en producció. S'han de definir uns criteris de qualitat mínims abans de posar l'aplicatiu en producció (on es continuarà realitzant proves).
- **Redacció de la memòria:** Es farà de forma paral·lela mentre es duen a terme les fases anteriors.
- **Creació del vídeo de la presentació final:** Necessari per defensar el treball.
- **Investigació i formació:** Pel plantejament inicial del treball, es durà a terme a la vegada que es realitzen la resta de fases.

## 1.4 Planificació del treball

Els recursos necessaris per desenvolupar el projecte són mínims: Un ordinador personal amb el seu sistema operatiu i un editor de codi (*Visual Studio*

*Code*<sup>10</sup>). Per altra banda, les tasques a realitzar han estat enumerades en el punt anterior i seran descrites en el seu apartat corresponent. Gràficament en un diagrama de Gantt queden de la següent forma:

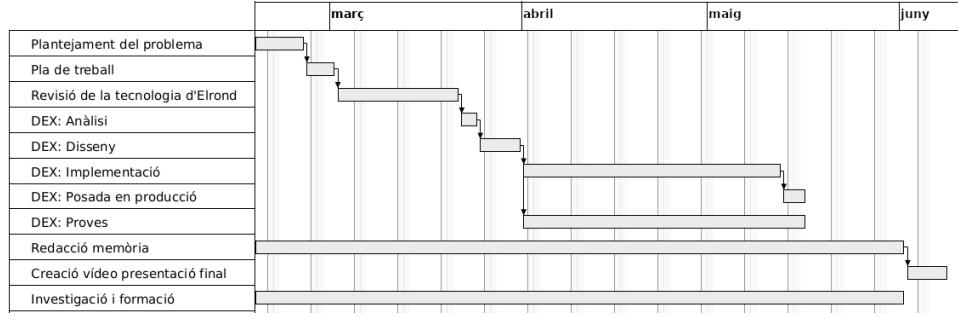


Figura 1: Diagrama de Gantt del projecte (creat amb GNOME Planner 0.14.6).

La següent figura presenta la temporalització de les fases del projecte, s'ha de considerar que un dia de feina correspon a una jornada de quatre hores:

1	Plantejament del problema	17/02/22	24/02/22	6d
2	Pla de treball	25/02/22	01/03/22	3d (coincideix amb l'entrega de la PAC 1)
3	Revisió de la tecnologia d'Elrond	02/03/22	21/03/22	14d
4	DEX: Anàlisi	22/03/22	24/03/22	3d
5	DEX: Disseny	25/03/22	29/03/22	3d (coincideix amb l'entrega de la PAC 2)
6	DEX: Implementació	30/03/22	12/05/22	32d (la PAC 3 caurà en mig de la implementació)
7	DEX: Posada en producció	13/05/22	16/05/22	2d
8	DEX: Proves	01/04/22	16/05/22	Durant les fases d'implementació i proves
9	Redacció memòria	17/02/22	31/05/22	Durant tot el projecte (coincideix amb l'entrega de la PAC 4)
10	Creació vídeo presentació final	01/06/22	08/06/22	6d
11	Investigació i formació	17/02/22	01/06/22	Durant tot el projecte

Figura 2: Temporalització del projecte.

<sup>10</sup>Existeix el plugin “Elrond IDE”: <https://marketplace.visualstudio.com/items?itemName=Elrond.vscode-elrond-ide>

## **1.5 Breu sumari de productes obtinguts**

“No hay que entrar en detalle: la descripción detallada se hará en el resto de capítulos.”

## **1.6 Breu descripció dels altres capítols de la memòria**

“Explicación de los contenidos de cada capítulo y su relación con el trabajo en global.”

## 2 Revisió de la tecnologia d'Elrond

### 2.1 Visió general de l'arquitectura

Elrond és una cadena de blocs<sup>11</sup> pública i d'alt rendiment. Altres punts a destacar són [1][19]:

1. **Entitats (usuaris i nodes)**: Els usuaris despleguen transaccions a la xarxa, en forma de transferència de valor o executant un *smart contract*. Per altra banda, els nodes són dispositius en la xarxa que executen el programari d'Elrond i processen les transaccions.
2. **Validadors**: Nodes de la xarxa Elrond amb almenys 2500 EGLD en *staking* (o bloquejats<sup>12</sup>) que processen les transaccions i asseguren la xarxa per la seva participació en el mecanisme de consens per validar els blocs (seran recompensats amb les tarifes –*fees*– de les transaccions).
3. **Shards<sup>13</sup>**: Particions de la xarxa d'Elrond que permeten escalar-la. La feina de cada *shard* és gestionar una part de l'estat i processar paral·lelament la corresponent part de la transacció.
4. **Adaptive State Sharding**: Divisió i unió dinàmica de *shards* segons el nombre de validadors disponible i càrrega de la xarxa. S'aplica a tots els nivells (transacció, dades i xarxa) de forma adaptativa.
5. **Secure Proof of Stake**: Els blocs són validats per consens entre els validadors del grup de consens (valgui la redundància), que és completat en dues passes de comunicació emprant una modificació de les signatures múltiples de Boneh-Lynn-Shacham (BLS) [20]. El grup de consens és seleccionat aleatoriament i només és possible conèixer la seva composició amb una ronda d'antelació.
6. **Alta resiliència**: Capacitat de recuperar-se d'atacs maliciosos pel canvi de nodes entre els *shards* (en cada “època” un terç dels nodes seran reubicats per prevenir connivències entre els mateixos).
7. **Font d'aleatorietat segura**: Utilitzant la signatura BLS, cosa que la fa no esbiaixada i impracticable.

---

<sup>11</sup>En el present treball empraré indistintament “blockchain” i “cadena de blocs” (traducció del terme proposada pel Termcat).

<sup>12</sup>El Termcat encara no ha traduït el terme, és una traducció pròpia que penso que fa entendre el concepte.

<sup>13</sup>No hi ha entrada al Termcat pel terme “shard”, jo l'he traduït per “fragment” (empraré ambdues paraules indistintament en el present treball).

8. **Elrond WASM VM:** Màquina virtual específica que permet executar contractes intel·ligents escrits en qualsevol llenguatge de programació que es permeti compilar amb *WebAssembly*<sup>14</sup>.
9. **Contractes intel·ligents:** Que s'executaran emprant l'*Adaptative State Sharding*. Elrond recomana que siguin escrits en Rust però, com s'ha dit en el punt anterior, es poden utilitzar altres llenguatges. Una característica única, comparant per exemple amb Ethereum, és que es poden modificar els *smart contracts* després de ser desplegats [21] (per exemple per corregir errors).
10. **Execució ràpida de transaccions cross-shard:** Gestionada de forma nadiua a nivell de protocol fent servir un algorisme d'expedició (*dispatching*) i un algorisme d'encaminament.
11. **Metachain:** Cadena de blocs que s'executa en un *shard* especial. El seu rol principal no és processar transaccions sinó que és notificar i autenticar les capçaleres dels blocs processats.
12. **Elrond Gold (EGLD):** Criptovalor (*token*) nadiu de la xarxa Elrond que serveix de mitjà de pagament per a les transaccions. Ajuda en el desplegament de dApps, en l'execució de contractes intel·ligents i també s'empra com a mecanisme de pagament de recompenses per als validadors.

## 2.2 Entitats

Bàsicament, en la xarxa Elrond hi ha dos tipus d'entitats: usuaris i nodes [1]. Un **usuari** és qualsevol ens que gestioni un dels **comptes** –o més– de la xarxa Elrond. Un parell de **claus criptogràfiques** (una pública i una privada) li permetran enviar **transaccions signades** fent servir la xarxa. Els comptes tenen associat una quantitat d'EGLD que es coneix com a **balanç** i a més tenen un espai d'emmagatzematge per a valors arbitraris (com per exemple informació sobre **tokens creates pels usuaris**). Els comptes s'identifiquen de forma unívoca per una **adreça** que coincideix amb la clau pública de l'usuari (32 bytes fent servir la representació Bech32). Normalment, els usuaris gestionen els seus parells de claus emprant unes aplicacions informàtiques que s'anomenen **carteres** (o *wallets*)<sup>15</sup>.

Per altra banda, els **nodes** són dispositius connectats a la xarxa d'Elrond que realitzen les operacions sol·licitades pels seus usuaris. Els nodes poden ser passius (*observers*) o actius (*validators* i *fishermen*). Els validadors s'encarreguen del consens, d'afegir blocs i mantenir l'estat, essent premiats

---

<sup>14</sup><https://webassembly.org/>

<sup>15</sup>En el present treball empraré els termes “cartera” o “moneder” indistintament per referir-me al concepte de “wallet”.

per la seva contribució. Els **validadors** són identificats de forma única per una clau pública BLS de 96 bytes. Per garantir el correcte funcionament dels nodes, els validadors han de tenir en *staking* com a mínim 2500 EGLD. Sense bloquejar aquesta quantitat d'EGLD, els nodes poden fer d'**observadors** (però no rebran cap recompensa). Aquests darrers són membres passius de la xarxa que poden actuar com a interfície de lectura i retransmissió. Poden ser complets (*full*), mantenint tota la història de la cadena de blocs, o lleugers (*light*), mantenint només 2 èpoques de l'historial de la cadena de blocs. Finalment, trobem els **pescadors** (*fishermen*). La seva tasca és verificar la validesa dels blocs després d'haver estat proposats, detectant així actors maliciós [19]. Rebran també una recompensa i aquest rol pot ser exercit per observadors o validadors que no formin part de la ronda de consens en curs.<sup>16</sup>

### 2.3 Cronologia

En la xarxa d'Elrond s'organitza el temps en **èpoques** (*epochs*) que es subdivideixen en **rondes** (*rounds*) i [1]. La primera ronda de la primera època (*genesis round*) és especial i serveix per inicialitzar la xarxa.

Una **època** és una seqüència de rondes consecutives en què la configuració de la xarxa no canvia. El nombre de rondes en una època es calcula perquè aquesta darrera duri 24 hores (això està definit en la configuració actual i és modificable). Quan hi ha un canvi d'època s'aprofita per adaptar la topologia de la xarxa segons el nombre de validadors disponibles i càrrega de treball. En aquest precís moment també s'aprofita per acomplir amb altres tasques per tancar l'època anterior (com calcular les recompenses per als validadors).

Com es pot intuir, les **rondes** tindran una durada fixa que (actualment és configurada en 5 segons). Per l'arquitectura en *shards* de la xarxa, en cada ronda només es podrà afegir un bloc a la cadena de blocs del *shard*. Òbviament, si no s'arriba a consens o quan el líder del grup de consens designat és fora de línia i no pot proposar un bloc, pot haver-hi rondes en què no s'afegeixi cap bloc a la *blockchain*.

S'ha de remarcar que els sistemes que implementen PoS solen dividir el temps de forma similar [2].

### 2.4 Secure Proof of Stake

El funcionament *Secure Proof of Stake* (SPoS) d'Elrond es pot desgranar en les següents passes [1][2]:

---

<sup>16</sup>No es diu explícitament en la bibliografia però investigant pels grups de Telegram d'Elrond (concretament en <https://t.me/ElrondValidators>) vaig descobrir que el concepte de “fisherman” encara no està implementat.

1. La font d'aleatorietat per seleccionar els validadors per al consens es calcula a partir del bloc anterior, que és signat pel líder de consens –també conegut com a proposador de blocs o *block proposer*– de la ronda que acaba. Això implica que aquesta font d'aleatorietat no podrà ser coneguda amb més d'una ronda d'antelació.
2. Se selecciona el grup de consens, compost per validadors i un únic *block proposer*. Una vegada coneguda la font d'aleatorietat, el procediment de tria del grup és determinista (triga menys de 100 ms<sup>17</sup>). S'ha de considerar que per triar els nodes es té en compte la quantitat d'EGLD en *staking* i una qualificació individual –revisada al final de cada època– que es basa en el comportament passat. Per exemple, es davallarà la seva puntuació si no proposa el bloc perquè és fora de línia, es detecta una activitat maliciosa, etc. Aquesta “meritocràcia” anima els propietaris dels nodes a tenir-los en bon funcionament.
3. El líder de consens (o validació) produceix el bloc per a la nova ronda. Si, pel motiu que sigui, no s'ha creat un bloc en una finestra de temps s'utilitzarà la font d'aleatorietat de l'últim bloc per seleccionar un nou grup de consens.
4. El líder de consens<sup>18</sup> envia el bloc que acaba de proposar als validadors.
5. Aquests darrers components del grup validen i també signen el bloc rebut, basant-se en una modificació de *Practical Byzantine Fault Tolerance* (pBFT).
6. Els validadors trameten les signatures al *block proposer*.
7. El proposador agrega les signatures i distribueix el bloc.
8. El *hash* del bloc, la signatura, les proves d'inclusió i el nombre de *shard* són enviats a la *Metachain*.

Aquestes passes es poden veure resumides en la Fig. 3.

## 2.5 Adaptive State Sharding

El *sharding* va sorgir originàriament en el camp de les bases de dades com un mètode per distribuir les dades entre múltiples màquines amb l'objectiu de suportar alt rendiment amb grans volums d'informació [22]. Elrond empra aquesta tècnica d'escalat horitzontal per particionar tant la xarxa com

---

<sup>17</sup>El SPoS d'Elrond es basa en la premissa que un actor malèvol només té el temps que dura una ronda per adaptar-se i intentar influir en el bloc que es proposarà.

<sup>18</sup>Per aclarir del tot aquest punt, en la bibliografia emprada, líder de consens, líder de validació i proposador de bloc (*block proposer*) fan referència al mateix concepte.

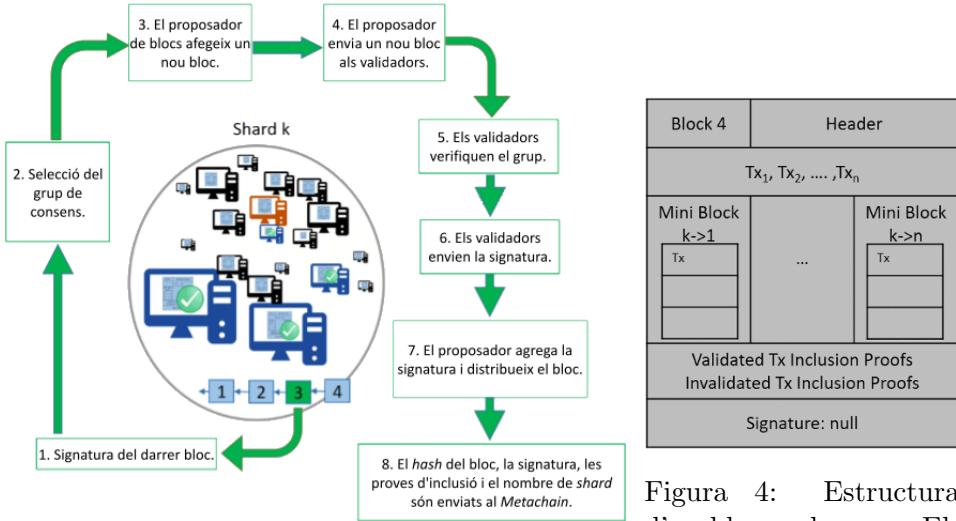


Figura 3: *Secure Proof of Stake*. (Font: Modificació d'imatge dins l'apartat corresponent a SPoS a [1]).

Figura 4: Estructura d'un bloc en la xarxa Elrond. (Font: Imatge dins l'apartat corresponent a SPoS a [1]).

l'estat i processament de les transaccions amb la finalitat que diferents nodes desenvolupin la seva tasca en paral·lel (amb el consegüent augment de l'eficiència). S'ha de destacar també que s'empren els tres tipus principals de *sharding* [1]:

- **Network sharding**<sup>19</sup>: Gestiona com els nodes es distribueixen i agrupen entre els diferents *shards*. S'ha de tenir en compte que les comunicacions dins del *shard* són més ràpides que fer una propagació (*broadcast*) dels missatges a tota la xarxa. Un punt molt important a destacar és que si un atacant arriba a controlar un *shard* sencer suposaria un greu problema de seguretat.
- **Transaction sharding**: S'ocupa la forma en què les transaccions s'assignen als *shards* encarregats del seu processament. Les transaccions són assignades a un *shard* de forma determinista emprant les adreces de les transaccions.
- **State sharding**: Cada *shard* només manté una part de l'estat. Els dos mètodes anteriors, si no es combinessin amb aquest, haurien d'emmagatzemar una còpia sencera de tot l'estat. Així, si els comptes implicats en una transacció resideixen en *shards* diferents, l'execució de

<sup>19</sup>Aquest concepte es podria traduir al català com “fragmentació de la xarxa” però he preferit deixar el terme anglès. El mateix passa amb *transaction sharding* i *state sharding*, que es podrien traduir respectivament per “fragmentació de les transaccions” i “fragmentació de l'estat”.

la transacció implicaria l'intercanvi de missatges entre nodes per modificar els respectius estats. Com ja s'ha dit, per augmentar la tolerància a atacs maliciósos, quan acaba una època es redistribueixen entre els diferents *shards* un subconjunt dels nodes.

Com a resultat d'aquesta fusió s'aconsegueix [1]:

- **Escalabilitat sense afectar a la disponibilitat:** En la Fig. 5a podem observar la xarxa amb només un *shard*. Les figures 5b i 5c mostren una arquitectura amb dos i tres *shards*<sup>20</sup>. Sense entrar en detall, es pot veure que augmentant o disminuint el nombre de *shards* –gràcies a la forma en què s'assignen les adreces al seu fragment– s'evita temps d'inactivitat per qüestions de configuració.
- **Expedició (*dispatching*) ràpida i traçabilitat:** La Fig. 5c mostra com es calcula el *shard* de destí de forma determinista (les adreces de color blau aniran al 0, les verdes a l'1 i les grogues al 2).
- **Eficiència i adaptabilitat:** El sistema permet processar transaccions en paral·lel i adaptar-se a la càrrega de treball i/o estat de la xarxa. Però s'ha de destacar que la distribució dels *shards* hauria de ser tan equilibrada com fos possible. Si ens fixem en la Fig. 5c veiem que no és equilibrada (ja que el nombre de *shards* no és una potència de 2).

Una altra cosa a destacar és que els nodes guarden una **còpia de l'estat dels seus germans (redundància)** per aportar **tolerància a fallades** (Fig. 6).

Finalment, s'ha de remarcar el fet que els **nodes es barregin entre *shards* al final de cada època** per evitar la connivència entre nodes maliciós. Això no es fa amb tots els nodes sinó que només es redistribuirà –de forma determinista i uniforme– un nombre controlat de validadors (aquesta tasca la realitza la *metachain* emprant una font d'aleatorietat procedent del bloc anterior de la metacadena). Es fa així, i no reorganitzant tots els nodes, per maximitzar la seguretat amb la mínima introducció de latències en el sistema. Els nodes triats es col·locaran als seus nous *shards* en una llista d'espera durant tota l'època actual fent la resincronització amb el nou fragment. Després el node es pot convertir en un validador elegible i unir-se al *shard* efectivament.<sup>21</sup>

## 2.6 Transaccions entre *shards*

Com a exemple de com s'executen les transaccions entre diferents *shards* i com es comuniquen amb la *metachain*, emprarem una **arquitectura amb**

---

<sup>20</sup>En [2] es pot trobar una funció en pseudocodi amb la que calcular el nombre òptim de *shards*.

<sup>21</sup>Aquest paràgraf és un resum molt breu. Per referències més detallades consultar [2].

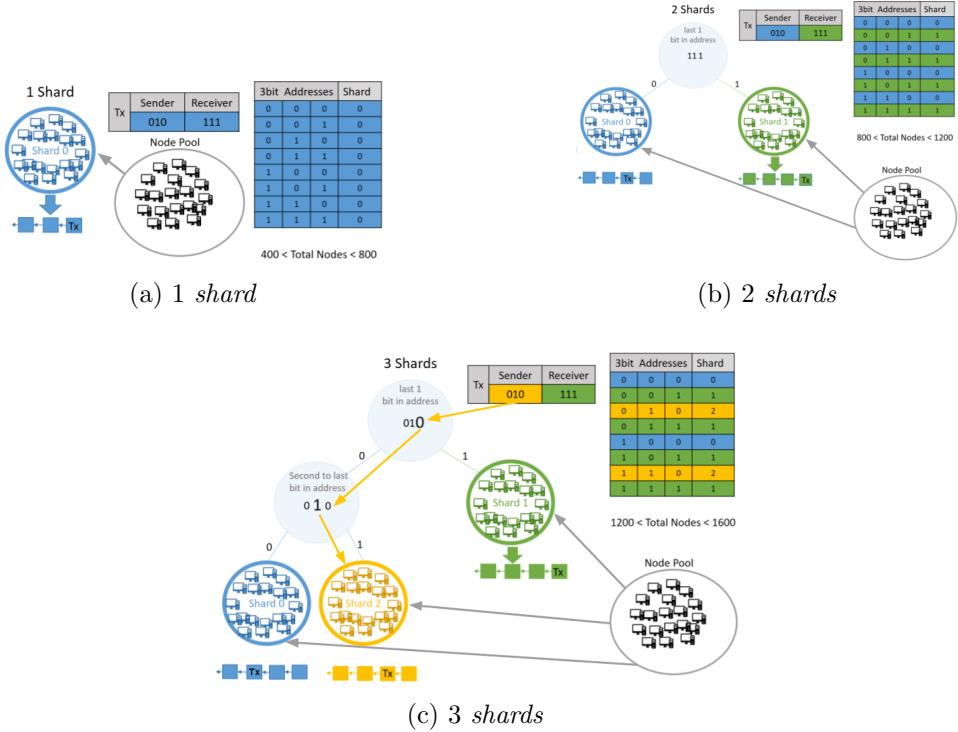


Figura 5: La xarxa d'Elrond configurada amb un (a), dos (b) i tres (c) *shards*. (Font: Modificació d'imatge dins l'àpartat *Adaptive State Sharding* a [1]).

**només dos *shards*.** Imaginem que un usuari des del seu *wallet* amb una adreça dins del *shard 0* envia EGLD a l'adreça d'altre *wallet* que es troba en el *shard 1* (Fig. 7). L'estrucció dels blocs estarà formada per [2]:

- **Capçalera (header):** Conté la informació referent al bloc (*nonce*, *round*, *proposer*, *validators*, *timestamps*, etc.).
- **Llista de miniblocs:** Per a cada *shard* hi haurà un minibloc que contindrà transaccions a executar. S'ha de destacar que un minibloc és la unitat atòmica de processament, això significa que o es processen totes les transaccions del minibloc o no es processa cap (en aquest darrer cas, es posposarà l'execució al següent *round*). Dins d'un mateix bloc, poden aparèixer diversos miniblocs amb el mateix emissor i receptor (no hi ha cap limitació). Per exemple, en aquest cas simple amb només dos *shards*, un bloc en el *shard 0* contindrà bàsicament tres tipus de miniblocs:
  - Minibloc 0: Conté les transaccions on les adreces del remitent i del destinatari són al *shard 0*.
  - Minibloc 1: Conté les transaccions on les adreces del remitent són al *shard 0* i les del destinatari al *shard 1*.

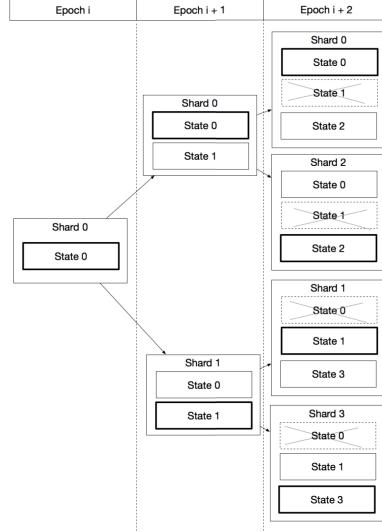


Figura 6: Redundància als *shards* entre èpoques. (Font: Captura de [2]).

- Minibloc 2: Conté les transaccions on les adreces del remitent són al *shard* 1 i les del destinatari al *shard* 0.

Observem en la Fig. 7 un exemple de transacció asíncrona amb dos *shards* implicats. Després d’arribar la transacció al *shard* 0 (**step 1**), la capçalera del bloc i els miniblocs s’envien a la *metachain*. Aquesta darrera dona fe<sup>22</sup> d’aquest bloc del *shard* 0 creant un nou bloc a la metacadena que conté la següent informació sobre tots els miniblocs: identificador del *shard* remitent, identificador del *shard* receptor i *hash* del minibloc (**step 2**). El *shard* 1 veu el minibloc a la *metachain* i obté el seu *hash* del metabloc (**step 3**), per després demanar el minibloc al *shard* 0 i executar-lo altra vegada (**step 4**). Finalment, el *shard* 1 envia el resultat a la *metachain* que valida la transacció creuada (**step 5**). En aquest punt la transacció ja es pot considerar finalitzada.

## 2.7 La màquina virtual d’Elrond

La màquina virtual d’Elrond executa WebAssembly (Wasm)<sup>23</sup>, un llenguatge de baix nivell amb format binari compacte anomenat *bytecode* [23]. El fet que executi Wasm implica que es pugui escriure contractes intel·ligents en qualsevol llenguatge que sigui possible compilar cap a *bytecode* (C, C++,

<sup>22</sup>“Donar fe” o “autenticar” són les traduccions al català que he trobat més adients pel verb anglès “to notarize”. El verb “notarizar” existeix en espanyol, però no existeix “notaritzar” en català. En qualsevol cas, empraré aquest darrer terme en algunes ocasions entre cometes per no generar confusions.

<sup>23</sup><https://webassembly.org/>

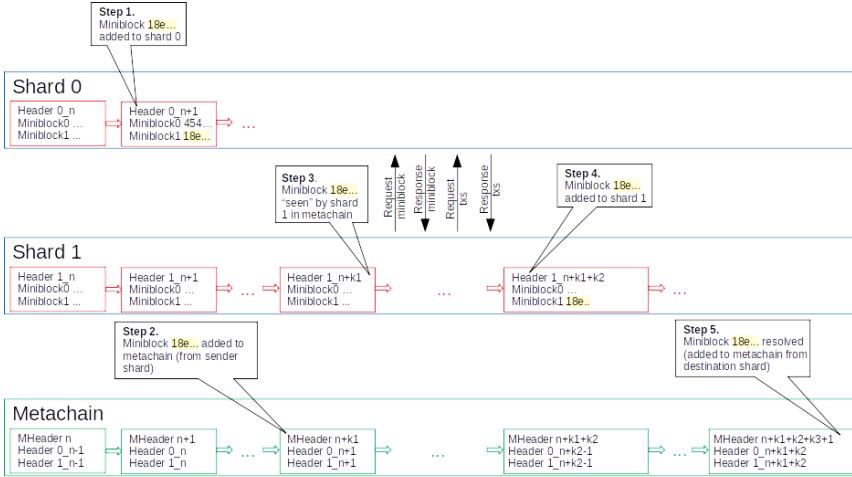


Figura 7: Exemple d'execució de transacció entre dos *shards*. (Font: Fig. 4 de [2]).

C#, Rust, Go, TypeScript, etc.). Tot i això, Elrond recomana als desenvolupadors que emprin Rust i els hi faciliten un *framework*<sup>24</sup> per a aquest llenguatge, així com un *plugin* per a l'IDE de Microsoft Visual Studio Code<sup>25</sup>. Entre les característiques a destacar de la màquina virtual d'Elrond [1][24]:

- **Sense estat:** Això significa que, quan un *smart contract* sigui executat, es guardarà la informació en una estructura de dades transitòria, en lloc d'escriure directament en l'estat. Quan acabi l'execució, si aquesta és exitosa, l'API aplicarà els canvis a l'emmagatzematge i/o a la cadena de blocs.
- **Execució fora de procés:** La màquina virtual serà executada en un procés independent i el node en si mateix en un altre, tot i que compartiran informació a través de canonades (*pipes*) anònimes a memòria principal. A més, el *bytecode* s'executarà en un entorn aïllat i la memòria del procés de la màquina virtual serà inaccessible.
- **Motor d'execució ràpida:** Com a motor d'execució s'empra una versió modificada per Elrond de Wasmer<sup>26</sup>, una implementació de Wasm<sup>27</sup> de codi obert escrita en Rust per entorns servidor. Per les característiques de Wasmer, l'execució dels contractes intel·ligents es fa a una velocitat gairebé nadiua.

<sup>24</sup><https://github.com/ElrondNetwork/elrond-wasm-rs>

<sup>25</sup><https://marketplace.visualstudio.com/items?itemName=Elrond.vscode-elrond-ide>

<sup>26</sup><https://wasmer.io/>

<sup>27</sup>Realitzada per Wasmer Inc. Aquesta empresa també ha creat un gestor de paquets que permet els desenvolupadors compartir mòduls empaquetats de codi Wasm.

- **Trucades asíncrones entre contractes:** Els contractes intel·ligents poden executar trucades entre ells fent servir l'API de la màquina virtual. Com hem vist, la xarxa d'Elrond es fragmenta adaptativament, el que pot fer que es truqui a un *smart contract* que es trobi en un altre *shard*. En aquest cas l'execució serà asíncrona. Si ambdós estan en el mateix fragment, l'execució serà síncrona. Tot això es fa de forma transparent per al desenvolupador.

En el cas general, els contractes intel·ligents seran compilats generant un arxiu WASM que serà desplegat en alguna de les xarxes d'Elrond.

## 2.8 Execució de contractes intel·ligents en l'arquitectura amb *shards*

Altre punt important a destacar és l'execució dels *smart contracts* en aquesta arquitectura fragmentada. Elrond ho solventa amb una solució amb execució asíncrona entre els *shards* [2].

El procés comença quan l'usuari crea una transacció per executar un contracte intel·ligent. Si el contracte intel·ligent no és ubicat en el mateix *shard*, el cost de la transacció es lleva del compte del remitent i s'afegeix a un minibloc –segons correspongui a l'adreça del receptor– del seu *shard*. La transacció és “notaritzada” per la *metachain* i després processada pel *shard* de destí. Al fragment de destí, la transacció es tracta com una invocació del mètode del contracte intel·ligent, ja que és on es troba (l'adreça de destí és la del *smart contract*). Per a la trucada del contracte intel·ligent, es crea un compte temporal que suplanta el compte del remitent, amb el saldo del valor de la transacció i es crida el contracte intel·ligent. Després de l'execució, el contracte intel·ligent pot retornar resultats que afecten diversos comptes en diferents *shards*. Els resultats que afecten els comptes del *shard* del contracte intel·ligent s'executen a la mateixa ronda, en cas contrari es crearan transaccions *Smart Contract Result* (SCR). En aquest darrer cas, es creen miniblocs SCR per a cada *shard* de destí que posteriorment són “notaritzats” per la *metachain* (de la mateixa manera que hem vist a la secció 2.6 per a les transaccions entre *shards*). Finalment, són processats pels *shards* respectius (on resideixen els comptes de destí). En el cas que un contracte intel·ligent truqui dinàmicament a un altre que es trobi en un *shard* diferent, aquesta trucada es desaria com a resultat intermedi i es tractaria de la mateixa manera que per als comptes. Aquesta solució necessitarà **almenys 5 rondes per completar-se**, però té els avantatges de què **no es necessita cap bloqueig ni moure estats entre *shards***.

## 2.9 Xarxes Elrond disponibles

En Elrond tenim tres xarxes disponibles:

- **Mainnet:** És la xarxa en producció d'Elrond. És a dir, es fan transaccions reals amb el pertinent cost econòmic. En el moment de redactar la present secció, té més de 3000 nodes validadors distribuïts en 3 *shards* i en una *metachain*.
  - **Devnet:** És una xarxa pública de proves mantinguda per la comunitat d'Elrond on qualsevol desenvolupador pot provar els seus contractes intel·ligents i dApps en un entorn real. Té uns 300 nodes validadors distribuïts en 3 *shards* i en una *metachain*. **En el present treball empraré aquesta xarxa**, en detriment de *testnet*, per tenir més estabilitat.
  - **Testnet:** També pública i mantinguda per la comunitat d'Elrond, però s'empra per fer proves de futures millores i rendiment [25] (això implica que el *blockchain* es pot reiniciar<sup>28</sup>). Té uns 2000 nodes reparats en 3 *shards* i en una *metachain*.

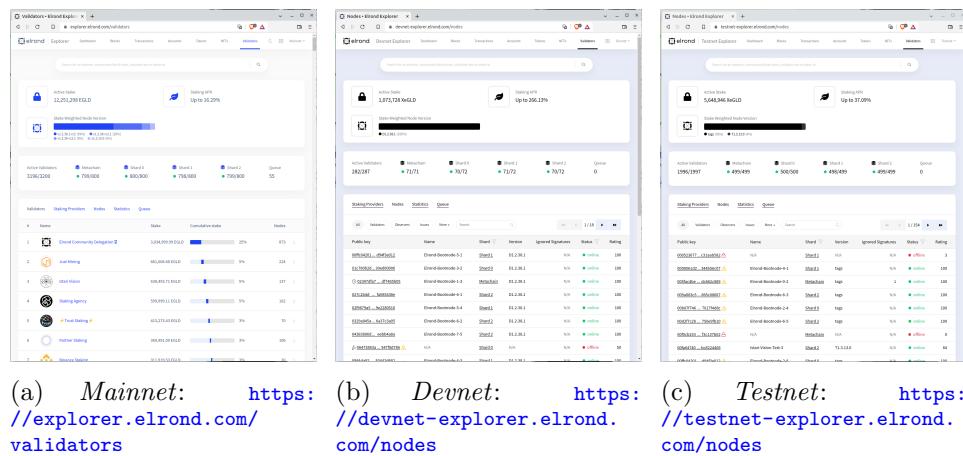


Figura 8: Exploradors de les xarxes disponibles a Elrond.

## 2.10 Muntar un node validador

Com ja s'ha deixat entreveure, la **xarxa d'Elrond** és composta pels seus nodes i per la interconnectivitat entre ells. Com s'ha dit en la secció 2.2, per **node** s'entindrà qualsevol instància del programari de codi obert desenvolupat per Elrond per a aquesta tasca<sup>29</sup>. A més a més, qualsevol persona o entitat responsable de la gestió d'un o més d'aquests nodes es coneix com a

<sup>28</sup>En el grup de Telegram “*Elrond Validators Announcements*” (<https://t.me/ElrondValidatorsAnn>) es va anunciar que es faria una ronda de gènisi per la *devnet* (reinici) per a dia 29 de març de 2022.

<sup>29</sup>Programari disponibile a: <https://github.com/ElrondNetwork/elrond-go>.

“**operador de nodes**”. Com també s’ha vist, la xarxa és dissenyada per ser segura i per poder balancejar la seva càrrega. Així, quan un nou node s’uneix a la xarxa aporta més seguretat i eficiència. I la xarxa premiarà els nodes per la seva aportació, creant-se així una espècie de simbiosi.

**Elrond és una xarxa descentralitzada de *blockchain***, és a dir, nodes de procedència desconeguda s’uneixen per crear seqüencialment blocs amb una cadència determinada. Els blocs contenen operacions que es realitzaren a petició dels usuaris de la xarxa pagant unes taxes (*fees*) per l’execució d’aquestes. Entre les operacions es troben la transferència de *tokens* entre comptes o l’execució de contractes intel·ligents (totes les operacions prenen la forma de transaccions). Per tant, es defineix així un llibre de comptabilitat distribuït que no depèn de cap entitat central.

Els nodes que estan autoritzats a prendre part en l’algorisme de consens s’anomenen **validadors**. Com que són els que veritablement produeixen i validen blocs, són els únics nodes recompensats amb EGLD per la seva contribució. Com ja vàrem dir en la secció 2.2, per assegurar el bon comportament dels validadors han de tenir bloquejats 2500 EGLD (*stake*). Els nodes sense *stake* s’anomenen **observadors** (no participen en el consens i no guanyen recompenses). Si el validador no funciona correctament (desconnexions de la xarxa en mig del procés de consens, accions malicioses, etc.) serà “multat” perdent EGLD (*stake slashing*) o fins i tot, en casos greus, se li llevarà l’estatus de validador (el node passarà a l’estat “jailed” i no es triarà per al consens si no paga una multa de 2,5 EGLD). Si això succeeix també davallarà la seva qualificació (*rating score*). Com que aquesta puntuació defineix la fiabilitat del validador, serà considerada per l’algorisme de selecció de nodes per triar el grup de consens. Així, s’afavoreix l’elecció de nodes amb qualificacions altes, també se’ls hi premiarà amb més EGLD.

Els requeriments mínims del sistema són:

- 4 CPU dedicades (Intel/AMD amb *flags* SSE4.1 i SSE4.2 activats).
- 8 GB de RAM.
- Disc dur SSD amb 200 GB d’espai lliure.
- Connexió a Internet de 100 Mbit/s sempre activa (mínim 4 TB/mes).
- Sistema operatiu Linux<sup>30</sup>/MacOS.

Com podem observar, són característiques d’un PC/Mac convencional en l’actualitat (i no d’ordinadors amb potents targetes gràfiques dedicades com passa amb el PoW de Bitcoin). No és l’objectiu del present treball, però guies com-es-fa<sup>31</sup> per instal·lar el programari per muntar un node validador en la xarxa Elrond es poden trobar a [1] o [26].<sup>32</sup>

---

<sup>30</sup>Es recomana Ubuntu 20.04.

<sup>31</sup>Traducció del Termcat del terme “*howto*”.

<sup>32</sup>Aquesta secció torna a ser una síntesi de l’apartat “*Validators*” de [1].

## 2.11 Carteres

Com ja s'ha comentat en la secció 1.1 existeixen diferents carteres que es poden fer servir per enviar, rebre i emmagatzemar EGLD de forma segura.

- **Moneder web.** Existeix una versió per a cadascuna de les xarxes:<sup>33</sup>
  - *Mainnet*: Accesible a <https://wallet.elrond.com/>.
  - *Devnet*: Accesible a <https://devnet-wallet.elrond.com/>.
  - *Testnet*: Accesible a <https://testnet-wallet.elrond.com/>.

Tampoc és l'objectiu del present treball explicar com crear un moneder, però una guia com-es-fa es pot trobar en [1]. Quan es crea s'haurà d'introduir una contrasenya<sup>34</sup> i se'ns donaran 24 paraules secretes (que permeten generar la clau privada). El procés finalitzarà descarregant un fitxer json (*keystore file*). Com es veu en la Fig. 9, podem accedir al nostre moneder amb aquest arxiu json i introduint la contrasenya que hem creat (nombre 1 en la Fig. 9). Una altra opció és generar un fitxer “PEM” (*Privacy Enhanced Mail*) amb la nostra clau privada i emprar-lo per accedir (nombre 3 en la Fig. 9), que es pot generar emprant la utilitat d'Elrond “erdpy” (s'hauran d'introduir les 24 paraules secretes separades per un espai):

```
erdpy --verbose wallet derive testdev-wallet.pem --mnemonic
```

Finalment, es pot iniciar sessió amb un moneder de maquinari (per exemple amb un Ledger Nano S) o llegint un codi QR amb l'aplicació mòbil Maiar (nombres 2 i 4 en la Fig. 9, respectivament).

- **Maiar DeFi Wallet.** Es tracta d'una extensió per a navegadors de la família de Chrome<sup>35</sup>. El funcionament i prestacions és similar al moneder web (fent la comparació amb la xarxa Ethereum, seria com l'extensió “Metamask”<sup>36</sup>). En la Fig. 10b es pot comprovar que també es pot canviar entre les tres xarxes disponibles (llista desplegable a dalt a la dreta que diu “Main”).
- **Maiar**<sup>37</sup>. DApp que implementa una cartera digital per dispositius iOS i Android. Permet enviar o rebre diners emprant el número de telèfon mòbil o un *herotag* (una espècie de nom d'usuari que es pot emprar en lloc de la teva adreça). És completament descentralitzada,

---

<sup>33</sup>Per a la *devnet* i la *testnet* es disposa d'una opció “Faucet” per obtenir xEGLD (que no té cap cost en moneda fiat i et permetrà poder relalitzar transaccions).

<sup>34</sup>Ha de tenir almenys nou caràcters, una lletra majúscula, un símbol i un nombre.

<sup>35</sup><https://chrome.google.com/webstore/detail/maiar-defi-wallet/>

<sup>36</sup><https://metamask.io/>

<sup>37</sup><https://maiар.com/>

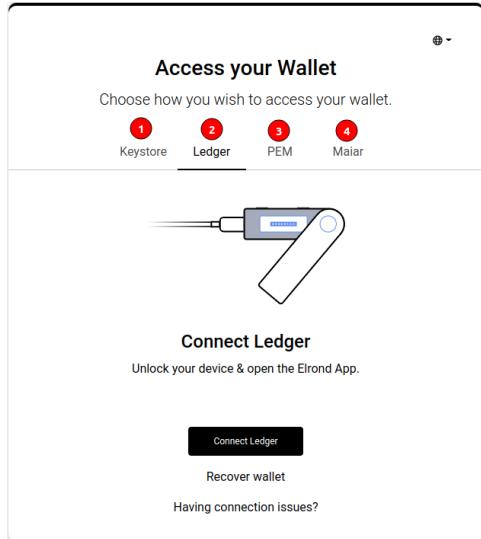


Figura 9: Mètodes d'accés al moneder web.

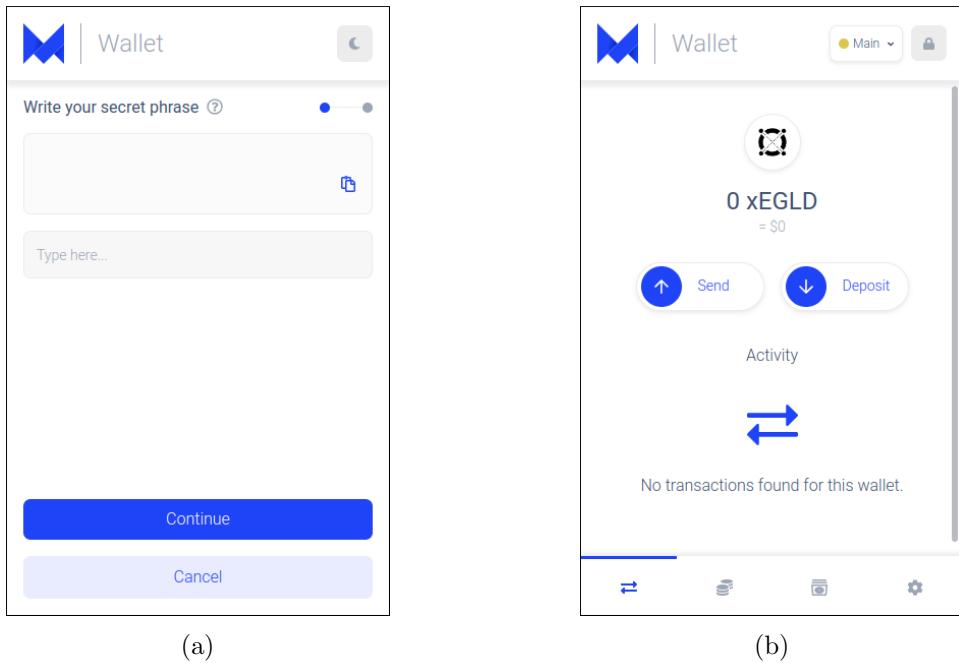


Figura 10: Maia DeFi Wallet.

sense custòdia i Elrond no té accés als fons de l'usuari en cap moment. Es pot treballar amb criptoactius com Elrond Gold (EGLD), Binance (BNB), Ethereum (ETH) o Bitcoin (BTC). Com no pot ser d'altra forma, es disposen de mecanismes de recuperació (una frase) per rescatar els fons en cas de pèrdua o robatori del nostre telèfon

intel·ligent. No he trobat forma de connectar a la *devnet* o *testnet* des de Maiar, penso que déu ser així perquè és principalment dissenyada per treballar amb la *mainnet*. Òbviament, aquesta dApp és pensada per comprar criptoactius a través de pagaments amb targeta bancària o transferències SEPA (s'han d'emprar passarel·les en ambdós casos). Curiosament, no permet fer *swaps* entre criptoactius (només es poden fer des de Maiar Exchange).



Figura 11: Maiar per iPhone.

- **Webhooks.** Són enllaços que criden<sup>38</sup> a la cartera de l'usuari perquè iniciï sessió o ompli un formulari per realitzar una transacció de pagament amb els arguments proporcionats. Un cop realitzada l'acció, l'usuari és redirigit a un URL amb informació d'estat d'èxit o d'error. Un exemple seria [1]:

<https://wallet.elrond.com/hook/login?callbackUrl=https://example.com/>

- **Ledger.** És l'opció recomanada per Elrond si es fa feina amb grans quantitats d'EGLD. Podem emmagatzemar els nostres criptoactius en moneders de maquinari com són els dispositius *Ledger Nano S* o *Ledger Nano X*.

---

<sup>38</sup>En català balear fem ús del verb “cridar” en lloc del verb “trucar”. Així diem per exemple “cridar per telèfon” o “cridar el mètode d'un objecte”.

## 2.12 Maiar Exchange

Maiar Exchange<sup>39</sup> és un DEX (*Decentralized EXchange*) basat en *liquidity pools* –fons de liquiditat– creat per Elrond emprant la seva arquitectura escalable. Permet els usuaris efectuar la compravenda de criptoactius (*trading*). A més a més, els usuaris poden convertir-se en proveïdors de liquiditat aportant els seus actius (guanyant *tokens* MEX per cada transacció d’intercanvi realitzada segons el parell de *tokens* aportats com a reserva). MEX és el *token* nadiu de Maiar Exchange. A part de tenir la funció d’incentivar els usuaris perquè proporcionin liquiditat, també serà emprat en processos de governança. Així, ens trobem davant d’un Exchange de *criptoactius* sense intermediaris, cosa que abarateix els costos de les transaccions i dels *swaps*.

Entre el que podem fer amb Maiar Exchange vull destacar [27][28]:

- **Swapping** (intercanvi): Consisteix a intercanviar un criptoactiu per un altre. El mecanisme d’intercanvi consisteix en permutar els actius de l’usuari amb els *liquidity pools* (fons de liquiditat), i no directament amb altres participants en el mercat. Els preus dels actius es fixen a partir d’una fórmula matemàtica del que s’anomena *Automated Market Making*. En lloc d’utilitzar un llibre de comptabilitat com en un intercanvi tradicional, els actius tenen uns preus calculats segons un algorisme. Elrond empra la fórmula  $x * y = k$ , on  $x$  és la quantitat d’un token al *liquidity pool* i  $y$  és la quantitat de l’altre.  $k$  és una constant fixa, el que significa que la liquiditat total del grup sempre romandrà en la mateixa proporció. Hi ha una comissió del 0,3% per intercanviar tokens (el 0,25% pels proveïdors de liquiditat proporcional a la seva contribució a les reserves de liquiditat i amb la resta es compraran MEX que es cremaran). En la data de redacció d’aquesta secció, els actius amb els que es poden fer intercanvis són MEX, USDC, RIDE, CRU, ZPAY, ISET, AERO, EFFORT i WAM contra EGLD. Això significa que hi ha fons de liquiditat MEX-EGLD, USDC-EGLD, etc.
- **Liquidity Pools**: Són reserves de tokens que es troben en contractes intel·ligents del DEX i estan disponibles perquè els usuaris facin intercanvis. Aquests tokens els proporcionen proveïdors de fons de liquiditat amb els quals els usuaris de la plataforma poden negociar. Els proveïdors són incentivats amb recompenses (tokens MEX) pels intercanvis que es produueixen en els seus *pools* (segons s’ha explicat en el punt anterior). S’ha de destacar que els proveïdors de liquiditat en qualsevol moment poden retirar els fons que han dipositat (independentment de les condicions del mercat).

---

<sup>39</sup><https://maiar.exchange/>

- **Farming**<sup>40</sup>: Consisteix en el fet que els usuaris bloquegin la seva liquiditat per obtenir recompenses. Si posem els nostres actius en un *liquidity pool* obtindrem les recompenses descrites en els apartats anteriors (això és el **farming** i és el que s'ha descrit en el punt d'adalt). Però hi ha també l'opció de posar els nostres *liquidity pools* en **staking**, el que significa que no els podrem retirar durant un temps definit a canvi d'aconseguir una recompensa major.

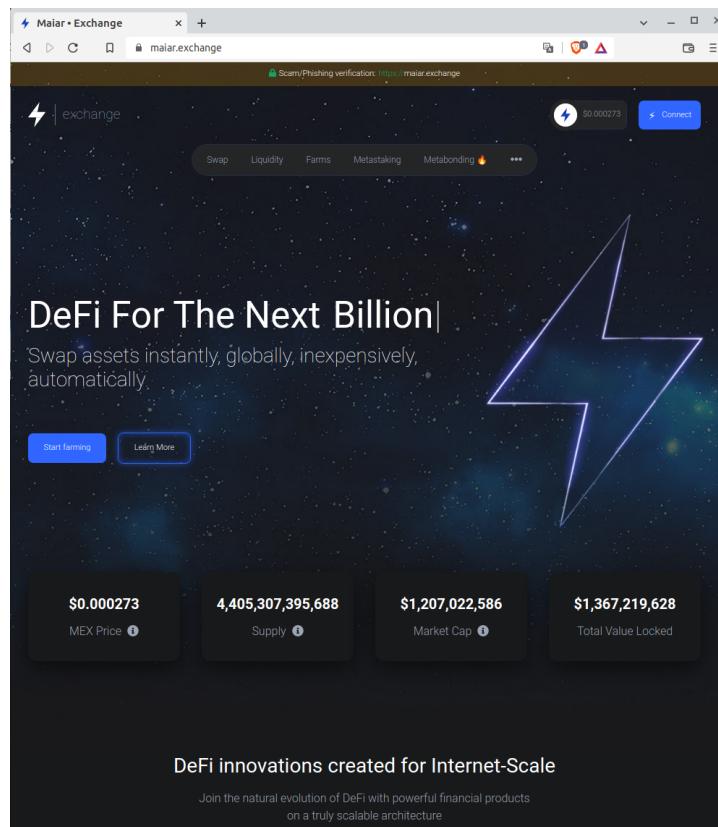


Figura 12: Maiar Exchange.

## 2.13 Altres punts destacables

- **Camps d'una transacció:**

Els camps d'una transacció sense signar en la xarxa Elrond són [1]:

- **nonce** (*number*): Nombre de seqüència del compte. És obligatori.

---

<sup>40</sup>No m'he atrevit a traduir aquest terme, però un mot que m'agrada del català és “conrear”.

- **value** (*string*): El valor a transferir. És obligatori (pot ser 0).
- **receiver** (*string*): Adreça del receptor (format *bech32*). És obligatori.
- **sender** (*string*): Adreça del remitent (format *bech32*). És obligatori.
- **gasPrice** (*number*): El preu del *gas* que s'utilitzarà en la transacció. És obligatori.
- **gasLimit** (*number*): El nombre màxim d'unitats de *gas* assignades per a la transacció. És obligatori.
- **data** (*string*): Informació arbitrària sobre la transacció, codificada en base64 . No és obligatori.
- **chainID** (*string*): Identificador de cadena. ‘D’ per a la xarxa *devnet*, ‘T’ per a *testnet* i ‘1’ per a la *mainnet*. És obligatori.
- **version** (*number*): La versió de la transacció (p. ex. 1). És obligatori.

En les transaccions que han de ser signades, amb la clau pública del remitent (amb l'algorisme Ed2519), s'ha d'afegir el següent camp:

- **signature** (*string*): Signatura digital que consta de 128 caràcters hexadecimals (64 bytes en representació en brut).

Hi ha disponibles diferents eines per realitzar la signatura des de la *shell* de Linux i des del codi font: *erdpy*, *erdwalletjs-cli* o *elrond-core-js*.

- **Decimals per treballar amb EGLD:** Un EGLD és configurat per tenir 18 decimals. Així, per exemple 2,5 EGLD tendrien un valor de  $2,5 * 10^{18} = 25000000000000000$  wei (seguint l'estàndard de 18 decimals dels tokens ERC20). L'usuari quan creï tokens ESDT podrà definir el seu nombre de decimals (amb un màxim de 18).
- **Costos de processament** [1]: Una transacció en la xarxa Elrond té un cost de processament (*processing cost*), que s'expressa com una quantitat d'unitats de *gas*. Quan es llança una transacció s'ha de proporcionar un *gasLimit*, que és el cost màxim que estem disposats a assumir. El **consum real de gas** (*actual gas consumption* o també *used gas*) és la quantitat real que s'ha consumit del *gasLimit* requerida perquè la xarxa processi la transacció. Per calcular aquest darrer, la xarxa el divideix en dos components d'utilització del *gas* [1]:
  - **Moviment de valors i tractament de dades:** Això seria per exemple una transferència d'EGLD entre adreces. El cost es calcula amb la fórmula:

```

tx.gasLimit =
    networkConfig.erd_min_gas_limit +
    networkConfig.erd_gas_per_data_byte * lengthOf(tx.data)
Nota:
networkConfig.erd_min_gas_limit <= tx.gasLimit i
tx.gasLimit <= networkConfig.erd_max_gas_per_transaction

```

- **Execució de contracte intel·ligent:** Una trucada a un *smart contract* requeriria aquest i l'anterior. El cost és més difícil de determinar, ja que depèn del codi font concret del contracte intel·ligent. Es solen emprar simulacions i estimacions.

Finalment, la **tarifa de processament** (*processing fee*) es calcula amb respecte de l'*actual gas consumption* i dels seus dos components. Per a un moviment de valors i tractament de dades s'especifica en la transacció un *gas price per gas unit* (que ha de ser igual o superior al paràmetre de la xarxa *erdmin\_gas\_price*). Per a l'execució d'un contracte intel·ligent el *gas price per gas unit* es calcula respecte a un altre paràmetre de xarxa anomenat *erd\_gas\_price\_modifier*:

```

value_movement_and_data_handling_price_per_unit = tx.GasPrice
contract_execution_price_per_unit =
    tx.GasPrice * networkConfig.erd_gas_price_modifier

```

- **ESDT (Elrond Standard Digital Token):** La xarxa Elrond admet de forma nadiua l'emissió de *tokens* creats pels usuaris (des de codi, des d'un contracte intel·ligent o des de la *web wallet*). Això implica que no és necessari emprar contractes com els de tipus ERC20 de la xarxa Ethereum. Ja que el suport és nadiu, no es requereix processament extra per part de la màquina virtual (amb el que són tan eficients com el mateix EGLD).
- **NFT (Non-Fungible Token) i SFT (Semi-Fungible Token):** Suport nadiu d'NFT i SFT afegint metadades i atributs a sobre de l'ESDT. Així, són bastant similars a aquests darrers però amb atributs extra (*NFT Name, Quantity, Royalties, Hash, Attributes* i *URI*).
- **Local Testnet:** Es pot configurar una *testnet* en local per fer proves i *debugging* del codi. Es pot emprar amb l'eina *erdpy* i conté nodes validadors, nodes observadors, un *seed node* i un *Elrond Proxy*.
- **API REST:** Té dues capes a les quals es pot accedir públicament:
  - **<https://gateway.elrond.com>:** La de més baix nivell. Gestiona l'encaminament de les peticions de forma transparent segons els mecanismes de fragmentació que s'han descrit en els apartats anteriors.
  - **<https://api.elrond.com>:** La de més alt nivell (empra els serveis de la capa anterior). Aporta serveis com un mecanisme de memòria cau, cerques històriques amb Elasticsearch<sup>41</sup>, etc.

---

<sup>41</sup><https://www.elastic.co/>

## 3 testDEX

### 3.1 Anàlisi

#### 3.1.1 DEX i AMM

Un **Decentralized exchange (DEX)** és un mercat on els negociants de criptomonedes fan transaccions directament entre ells –d’igual a igual (*peer-to-peer*)– sense haver de lliurar la gestió dels seus fons a un intermediari [3]. És a dir, són dissenyats per eliminar qualsevol autoritat de supervisió en els intercanvis (*swaps*) de criptoactius, evitant que s’hagin d’enviar dades de caràcter personal (noms, adreces, etc.). Per executar les ordres d’intercanvi sense intermediaris s’empren contractes intel·ligents. Això xoca amb el concepte de **Centralized exchange** on la gestió és responsabilitat d’una organització, com ara un banc o qualsevol altra corporació, que requerirà que els seus usuaris estiguin identificats i que custodiaran els actius dels usuaris (òbviament cercant un benefici econòmic).

En el present treball ens centrarem en una versió simplificada de DEX fent servir *Automated Market Makers* (AMM). Els diferents tipus de DEX es poden observar en la figura 13.

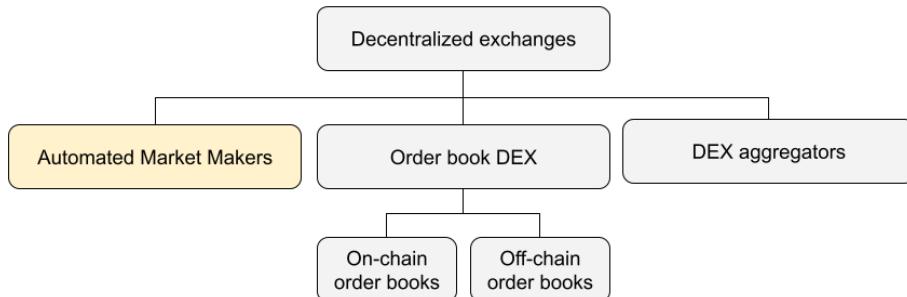


Figura 13: Tipus de DEX. (Font: [3]).

En [29] es descriu que una de les formes de prescindir del llibre d’ordres d’un *Exchange* tradicional és emprant AMM. Aquests protocols fan servir una fórmula matemàtica per calcular el preu de l’actiu i un fons de liquiditat per a cada parell de criptoactius (tal qual hem vist en la secció 2.12). Quan es vulgui fer un intercanvi entre criptoactius s’interactuarà directament amb un contracte intel·ligent que determinarà el preu de compra de l’actiu destí segons l’estat del fons de liquiditat. La fórmula per calcular això, també vista en la secció 2.12, és:

$$x * y = k$$

Amb  $x$  la quantitat d’un token al fons de liquiditat,  $y$  la quantitat de l’altre i  $k$  una constant fixa per mantenir la proporcionalitat entre el parell de tokens.

Imaginem que volem intercanviar una quantitat del primer token  $t_x$  i volen conèixer la quantitat  $t_y$  que rebrem a canvi:

$$(x + t_x) * (y - t_y) = k$$

$$\begin{aligned} (y - t_y) &= \frac{k}{x + t_x} \\ t_y &= y - \frac{k}{x + t_x} \\ t_y &= \frac{y * x + y * t_x - k}{x + t_x} \\ t_y &= \frac{y * t_x}{x + t_x} \end{aligned}$$

Si per contra volem donar  $t_y$  i rebre  $t_x$ :

$$\begin{aligned} (x - t_x) * (y + t_y) &= k \\ (x - t_x) &= \frac{k}{y + t_y} \\ t_x &= x - \frac{k}{y + t_y} \\ t_x &= \frac{x * y + x * t_y - k}{y + t_y} \\ t_x &= \frac{x * t_y}{y + t_y} \end{aligned}$$

Aquesta situació seria utòpica on no se pagaria cap taxa (*fee*). Si afegim aquesta comissió, les fórmules d'abans quedarien de la següent forma:

$$\begin{aligned} t'_x &= x - \frac{k}{y + t_y * (1 - fee)} = \frac{x * y + x * t_y * (1 - fee) - k}{y + t_y * (1 - fee)} \\ t'_x &= \frac{x * t_y * (1 - fee)}{y + t_y * (1 - fee)} \\ t'_y &= y - \frac{k}{x + t_x * (1 - fee)} = \frac{y * x + y * t_x * (1 - fee) - k}{x + t_x * (1 - fee)} \\ t'_y &= \frac{y * t_x * (1 - fee)}{x + t_x * (1 - fee)} \end{aligned}$$

Així, si es realitzes el *swap*, el que quedaría acumulat en el contracte intel·ligent seria:

$$\begin{aligned} acumulat_x &= t_x - t'_x \\ acumulat_y &= t_y - t'_y \end{aligned}$$

Per exemple, imaginem que tenim un fons de liquiditat amb el parell xEGLD-UOC –“UOC” és un ESDT creat per mi– amb 10 xEGLD ( $x$ ) i 10000 UOC ( $y$ ). Així tenim que la constant  $k$  queda com:

$$k = x * y = 10 * 10000 = 100000$$

Imaginem que volem canviar 1 xEGLD ( $t_x$ ) per UOC ( $t_y$ ) i que hi ha una comissió del 0,5%:

$$t_y = \frac{y * t_x}{x + t_x} = \frac{10000 * 1}{10 + 1} = 909,090909091$$

$$t'_y = \frac{y * t_x * (1 - fee)}{x + t_x * (1 - fee)} = \frac{10000 * 1 * (1 - 0,005)}{10 + 1 * (1 - 0,005)} = 904,956798545$$

$$acumulat_y = t_y - t'_y = 909,090909091 - 904,956798545 = 4,134110546$$

$$\begin{aligned} k &= (x + t_x) * (y - t_y) = (10 + 1) * (10000 - 909,090909091) = \\ &= 11 * 9090,909090909 = 99999,999999999 \end{aligned}$$

Si seguim i ara canviem 1000 UOC ( $t_y$ ) per xEGLD ( $t_x$ ):

$$\begin{aligned} t_x &= \frac{x * t_y}{y + t_y} = \frac{11 * 1000}{9090,909090909 + 1000} = 1,09009009 = \\ &= \frac{x * t_y * (1 - fee)}{y + t_y * (1 - fee)} = \frac{11 * 1000 * (1 - 0,0005)}{9090,909090909 + 1000 * (1 - 0,0005)} = \\ &= 1,089599034 \end{aligned}$$

$$acumulat_x = t_x - t'_x = 1,09009009 - 1,089599034 = 0,000491056$$

I la constant  $k$  queda com:

$$\begin{aligned} k &= (x - t_x) * (y + t_y) = (11 - 1,09009009) * (9090,909090909 + 1000) = \\ &= 9,90990991 * 10090,909090909 = 100000,000000908 \end{aligned}$$

A tot això se li ha d’afegir una cosa més: **la comissió que pagarà l’usuari a la xarxa Elrond**. En la secció 2.13 s’ha vist que emprar la xarxa Elrond duu associat uns costos pel moviment de valors i tractament de dades (transferim una quantitat de tokens al contracte intel·ligent) i per execució de contracte intel·ligent (el *smart contract* que implementa l’AMM). Això encarirà l’operació a l’usuari i les operacions que es realitzen dins aquest contracte influiran en el cost.

Finalment, es pot observar que **el redondeig provoca que la constant  $k$  es vagi desviant del seu valor original**. El codi del contracte intel·ligent haurà de corregir aquest problema.

### 3.1.2 Les variants de DEX i AMM a testDEX

Per assolir els objectius del projecte no fa falta arribar a desenvolupar un DEX de l'estil de Maiar Exchange, Uniswap<sup>42</sup> o PancakeSwap<sup>43</sup>. Òbviament, un projecte d'aquesta envergadura seria poc realista per una limitació clara de temps. Per exemple, el protocol AMM no és exempt de problemes que s'haurien de tractar [30]. Així en aquesta versió inicial faré una sèrie d'adaptacions per ajustar-me al temps disponible:

- El propietari del contracte intel·ligent serà l'únic que pot definir *liquidity pools*.
- El propietari del contracte serà, per tant, l'únic que rebi les comissions pels *swaps*.
- Tots els parells tindran com a un dels seus components xEGLD.

### 3.1.3 Requeriments funcionals

Els requeriments funcionals responen a la pregunta “què ha de fer un sistema”. Pel present projecte s'especifiquen els següents:

- RF 1: El sistema permetrà el propietari del contracte intel·ligent dipositar un parell xEGLD-ESDT en el fons de liquiditat des del seu *wallet*.
- RF 2: El sistema permetrà el propietari del contracte intel·ligent recuperar un parell xEGLD-ESDT del fons de liquiditat cap al seu *wallet*, que prèviament haurà dipositat.
- RF 3: El sistema permetrà el propietari del contracte definir una taxa (“*fee*”) que serà gravada als usuaris quan intercanviïn *tokens*.
- RF 4: El sistema permetrà el propietari recuperar els guanys acumulats al seu contracte intel·ligent resultant del pagament de l'anterior taxa cap al seu *wallet*.
- RF 5: El sistema permetrà els usuaris fer intercanvis (*swaps*) entre *tokens* de la seva cartera amb altres del fons de liquiditat (si existeix el parell adient).
- RF 6: El sistema permetrà els usuaris conèixer els parells donats d'alta al DEX.
- RF 7: El sistema permetrà els usuaris consultar si un parell és disponible per fer *swaps* (cap dels dos components pot tenir valor 0).

---

<sup>42</sup><https://uniswap.org/>

<sup>43</sup><https://pancakeswap.finance/>

- RF 8: El sistema permetrà els usuaris consultar la constant  $k$  d'un parell en un moment donat.
- RF 9: El sistema mostrerà els preus dels actius en el moment actual (tant de compra com de venda).
- RF 10: El sistema enregistrarà les transaccions.

### 3.1.4 Requeriments no funcionals

Els requeriments no funcionals responen a la pregunta “com ha de fer un sistema”. Pel present projecte s'especifiquen els següents:

- RNF 1: La informació emprada en el sistema viatjarà per la xarxa emprant protocols segurs.
- RNF 2: El sistema serà accessible des de qualsevol ubicació a través d'Internet.
- RNF 3: Els usuaris del sistema el podran emprar utilitzant qualsevol moneder vàlid de la xarxa Elrond.

## 3.2 Disseny

### 3.2.1 Casos d'ús

Tindrem un usuari privilegiat (el propietari del contracte intel·ligent) i usuaris genèrics. El primer, obviament, podrà realitzar també les operacions dels segons. El sistema permetrà només el propietari interactuar amb el contracte intel·ligent de la següent forma:<sup>44</sup>

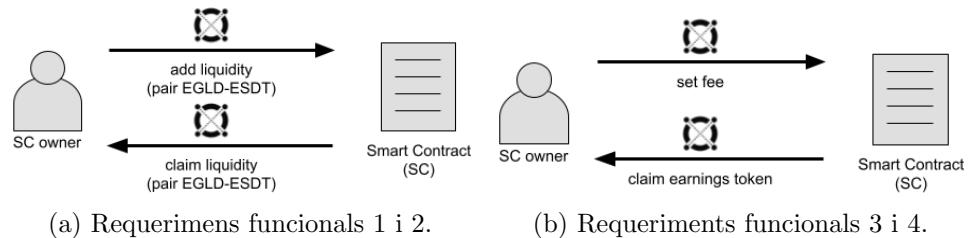


Figura 14: Interaccions del propietari amb el contracte intel·ligent.

Amb les interaccions anteriors es podran **definir *liquity pools*** i una **tasa (fee)** que es cobrarà per les operacions de *swap*. Allò següent a definir seran els intercanvis en si mateixos. El criptoactiu de referència serà xEGLD

<sup>44</sup>He imitat els diagrames explicatius de la documentació d'Elrond [1] i per això no he fet servir diagrames de casos d'ús d'UML. El contingut de les figures es troba en anglès ja que és l'idioma que he empatat en el codi font.

(tots els parells el tindran). Així es podrà passar una determinada quantitat d'un token ESDT a xEGLD. Això també es pot anomenar "**comprar ESDT amb EGLD**". A la inversa, podrem passar xEGLD a ESDT, que dit d'altra forma seria "**vendre ESDT per EGLD**". La següent figura ho mostra:

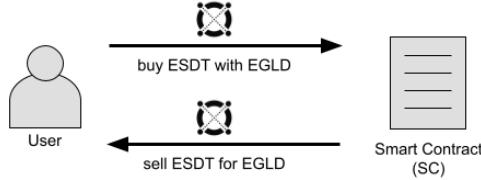


Figura 15: RF 5. Operacions de compra i venda de *tokens* (*swaps*).

Per altra banda, serà necessari que el sistema mostri als usuaris els **parells disponibles per fer intercanvis i si es pot operar amb ells en un moment donat** (requeriments funcionals 6 i 7).

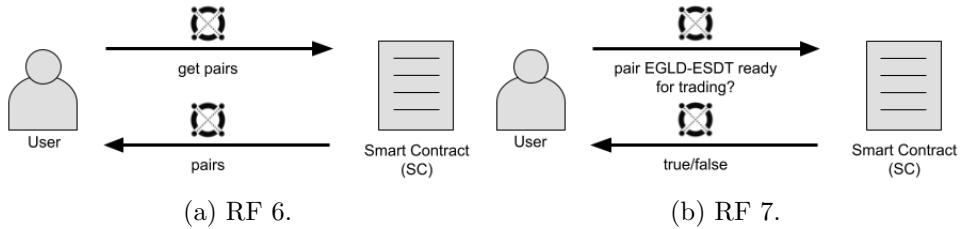


Figura 16: Parells donats d'alta preparats per a intercanvis.

A més, el sistema mostrarà la **constant  $k$  del protocol AMM** (que s'ha de mantenir constant) i el **preu dels actius en el moment actual**. Internament es treballa amb nombres sencers on es destinen les 18 posicions a la dreta per decimals, però realment són nombres sencers. Això farà que la constant  $k$  pugui fluctuar i que estigui sotmesa a correccions per mantenir-la en el seu valor inicial. Així mateix, els usuaris hauran de conèixer en temps real els preus de les parelles de tokens. Els casos d'ús sobre aquests punts es mostren en la Fig. 17.



Figura 17: Valor actual de la constant  $k$  i preus de compra-venda dels parells de *tokens*.

Finalment, pel que fa al RF 10 i als **requeriments no funcionals**, no

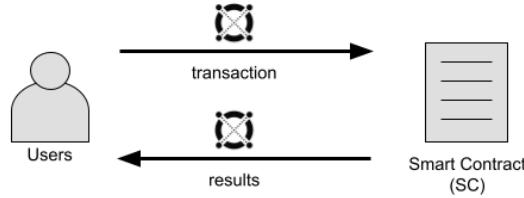


Figura 18: Requeriment funcional 10.

cal afegir res ja que formen part de les característiques intrínseques de les tecnologies d'Internet i de la xarxa Elrond.

### 3.2.2 Smart contract

Totes les dades dels *smart contracts* de la xarxa Elrond són disponibles públicament, tot i que pot ser complicat cercar manualment l'emmagatzematge del contracte i per això es solen definir *getters* públics<sup>45</sup>. Quan es desplegui a la xarxa se li assignarà automàticament una adreça amb el format que s'ha descrit en seccions anteriors.

En la Fig. 19 es mostren les propietats i mètodes del contracte intel·ligent que emprarem (anomenat “TestDEX”). Cal dir, que en argot específic de *blockchain* seria més apropiat anomenar a una propietat “emmagatzematge”.

La descripció dels “emmagatzematges” és la següent:

- **liquidity\_token(TokenIdentifier)**: Emmagatzema la liquiditat dels *tokens* (la meitat del parell). Per als identificadors del token dels que volguem guardar la informació, tindrem un valor del tipus BigInt<sup>46</sup> associat que defineix la seva quantitat acumulada en el contracte intel·ligent. Disposa d'un *getter* **getLiquidityToken** (amb el mateixos paràmetres).
- **liquidity\_egld(TokenIdentifier)**: Guarda la liquiditat en EGLD dels *tokens* (l'altra part del parell). Per als identificadors del token dels que volguem guardar la informació, tindrem un valor del tipus BigInt associat que defineix la quantitat d'EGLD. Disposa d'un *getter* **getLiquidityEgld** (amb el mateixos paràmetres).
- **initial\_k(TokenIdentifier)**: Per a cada parell *token-EGLD* emmagatzema un BigInt amb la seva constant *k* del moment en què es va donar d'alta. Disposa d'un *getter* **getInitialK** (amb el mateixos paràmetres).

<sup>45</sup>Quan s'implementin les propietats i mètodes públics en el lleguatge Rust s'anotaran amb #[endpoint] o #[view].

<sup>46</sup>Estructura de dades que representa un nombre sencer molt gran sense signe.

TestDEX
<pre>+ liquidity_token: BigInt + liquidity_egld: BigInt + initial_k: BigInt + fee: u32 + earnings: BigInt  + init(u32) + addLiquidityToken(): SCResult + claimLiquidityToken(TokenIdentifier): BigInt + addLiquidityEgld(TokenIdentifier): SCResult + claimLiquidityEgld(TokenIdentifier): BigInt + status(TokenIdentifier): Status + calculateK(TokenIdentifier): BigInt + priceEgldToken(TokenIdentifier, BigInt): BigInt + priceEgldTokenNumerator(TokenIdentifier, BigInt): BigInt + priceEgldTokenDenominator(TokenIdentifier, BigInt): BigInt + priceEgldTokenNoFee(TokenIdentifier, BigInt): BigInt + priceEgldTokenNoFeeNumerator(TokenIdentifier, BigInt): BigInt + priceEgldTokenNoFeeDenominator(TokenIdentifier, BigInt): BigInt + feeEgldToken(TokenIdentifier, BigInt): BigInt + priceTokenEgld(TokenIdentifier, BigInt): BigInt + priceTokenEgldNumerator(TokenIdentifier, BigInt): BigInt + priceTokenEgldDenominator(TokenIdentifier, BigInt): BigInt + priceTokenEgldNoFee(TokenIdentifier, BigInt): BigInt + priceTokenEgldNoFeeNumerator(TokenIdentifier, BigInt): BigInt + priceTokenEgldNoFeeDenominator(TokenIdentifier, BigInt): BigInt + feeTokenEgld(TokenIdentifier, BigInt): BigInt + ratio(TokenIdentifier, BigInt): BigInt + swapEgldForToken(TokenIdentifier): SCResult + swapTokenForEgld(TokenIdentifier): SCResult</pre>

Figura 19: Estructura del contracte intel·ligent.

- **fee:** Guarda la taxa –amb un valor del tipus  $u32^{47}$ – que s’aplicarà quan es faci un intercanvi. S’inicialitza quan es creï el contracte intel·ligent. Disposa d’un *getter* **getFee** (amb el mateixos paràmetres).
- **earnings\_egld(TokenIdentifier):** Emmagatzema les taxes (*fees*) – $BigInt$ – cobrades pels diferents *tokens*. Disposa d’un *getter* **getEarnings** (amb el mateixos paràmetres).

Pel que fa als mètodes:

- **init(u32):** Mètode que realitza la tasca de constructor del contracte intel·ligent. Se li passa la taxa que es cobrarà en els intercanvis.
- **addLiquidityToken() ->SCResult:** Afegeix liquiditat a la part “*token*” del parell. Retorna un *enum* anomenat *SCResult*, forma pre-determinada de retornar opcionalment un error (*Err(SCError)*) o una confirmació de resultat satisfactori (*Ok(T)*). El token i la quantitat les agafarà de la mateixa transacció dins de la xarxa Elrond. Mètode que només pot ser cridat pel propietari del contracte.
- **claimLiquidityToken(TokenIdentifier) ->SCResult:** Envia els fons de liquiditat d’un *token* determinat aportats pel propietari del

---

<sup>47</sup>Sencer sense signe representat amb 32 bits.

contracte intel·ligent a l'adreça del seu moneder. Se li ha de passar com a paràmetre l'identificador del *token* del qual volem reclamar la liquiditat. Mètode que només pot ser cridat pel propietari del contracte. Retorna un BigUint equivalent a la quantitat transferida. Només es torna la part “*token*”. El parell quedarà en estat “Funding”.

- **addLiquidityEgld(TokenIdentifier) ->SCResult**: Afegeix liquiditat a la part EGLD del parell. Se li ha de passar com a paràmetre l'identificador d'EGLD. La quantitat l'agafarà de la mateixa transacció dins de la xarxa Elrond. Mètode que només pot ser cridat pel propietari del contracte.
- **claimLiquidityEgld(TokenIdentifier) ->SCResult**: Envia els fons de liquiditat en EGLD del parell d'un determinat *token* aportats pel propietari del contracte intel·ligent a l'adreça del seu moneder. Se li ha de passar com a paràmetre l'identificador del token del que volem guardar la informació. Mètode que només pot ser cridat pel propietari del contracte. Retorna un BigUint equivalent a la quantitat transferida. El parell quedarà en estat “Funding”.
- **status(TokenIdentifier) ->Status**: Mostra l'estat d'un parell. Tindrà dos possibles estats: “Successful” i “Funding”. El primer significa que està preparat per fer intercanvis i el segon indica que encara s'han d'afegeir fons a algun dels components del parell. Se li ha de passar com a paràmetre l'identificador del *token* del que volem retornar aquesta informació. Retorna o bé “Successful” o bé “Funding”.
- **calculateK(TokenIdentifier) ->BigUint**: Calcula la constant *k* per a un parell en un moment donat de temps. Se li ha de passar com a paràmetre l'identificador del token que identifica el parell del que volem retornar aquesta informació (el que no sigui EGLD). Retorna la citada constant.
- **claimEarnings(TokenIdentifier) ->BigUint**: Envia els beneficis acumulats, resultants d'aplicar la taxa als intercanvis d'un *token* determinat, al contracte . Se li ha de passar com a paràmetre l'identificador del *token* del que volem transferir els beneficis. Mètode que només pot ser cridat pel propietari del contracte. Retorna un BigUint equivalent a la quantitat transferida.
- **priceEgldToken(TokenIdentifier, BigUint) ->BigUint**: Calcula el preu d'una certa quantitat d'un *token* determinat en EGLD aplicant la taxa (*fee*). Se li ha de passar com a arguments l'identificador del *token* del que volen calcular el preu i la citada quantitat. Retorna el preu com a nombre sencer.

- **priceEgldTokenNoFee(TokenIdentifier, BigUint)->BigUint**: Igual que l'anterior però al preu retornat no se li aplica la taxa.
- **feeEgldToken(TokenIdentifier, BigUint) ->BigUint**: Calcula la taxa que es pagarà per una certa quantitat d'un *token* determinat en EGLD. Se li ha de passar com a arguments l'identificador del *token* del que volen calcular el preu i la citada quantitat. Retorna aquest cost com a nombre sencer.
- **priceTokenEgld(TokenIdentifier, BigUint) ->BigUint**: Calcula el preu d'una certa quantitat d'EGLD en un *token* determinat. Se li ha de passar com a arguments l'identificador del *token* amb el que volem pagar i la quantitat d'EGLD que volem obtenir. Retorna el preu com a nombre sencer. S'aplica la taxa (*fee*).
- **priceTokenEgldNoFee(TokenIdentifier, BigUint) ->BigUint**: Igual que l'anterior però al preu retornat no se li aplica la taxa.
- **feeTokenEgld(TokenIdentifier, BigUint) ->BigUint**: Calcula la taxa que es pagarà per una certa quantitat d'EGLD en un *token* determinat. Se li ha de passar com a arguments l'identificador del *token* i la citada quantitat. Retorna aquest cost com a nombre sencer.
- **ratio(TokenIdentifier) ->BigUint**: Retorna un nombre sencer amb la relació que hi ha entre els *tokens* que formen el parell. Si la relació és 0, perquè el primer és més petit que l'altre, es retornarà 1. S'emprarà per corregir la constant *k* per evitar l'error introduït pel redondeig de nombres sencers. Se li ha de passar com a argument el *token* que identifica el parell.
- **swapEgldForToken(TokenIdentifier) ->SCResult**: Per intercanviar *token* per EGLD. La quantitat d'EGLD s'agafarà de la transacció. S'enviarà la quantitat del *token* al moneder de l'usuari que faci la compra. Les taxes pagades s'acumularan al contracte intel·ligent. S'haurà de passar com a argument l'identificador del *token* que es vol comprar. Les taxes es llevaran de la quantitat que ha de rebre l'usuari com a resultat de l'intercanvi
- **swapTokenForEgld(TokenIdentifier) ->SCResult**: Compra d'EGLD amb *token*. El *token* i la quantitat d'EGLD s'agafaran de la transacció. S'enviarà la quantitat d'EGLD al moneder de l'usuari que faci la compra. Les taxes pagades s'acumularan al contracte intel·ligent (es llevaran de la quantitat que ha de rebre l'usuari com a resultat de l'intercanvi).

Com ja s'ha dit, Elrond VM es basa en WebAssembly (WASM). A l'especificació se li **ha llevat el suport a les operacions de coma flotant**

[24]. Això ha provocat que s'hagin d'haver adaptat les fórmules que desenvolupades en la secció 3.1.1. D'aquesta forma, els corresponents mètodes del contracte intel·ligent en generaran dos de nous: un mètode que retorni el numerador i un altre que retorni el denominador. La idea de fer aquesta separació és fer les operacions de divisió a la dApp (que sí que permet operar amb decimals). Això té certa importància a nivell visual per a l'usuari, ja que es més comprensible llegir “1.21 xEGLD” que no pas “121000000000000000000 wei”. A part d'això, es generava també el problema de què la taxa (*fee*) no podia ser un nombre decimal. Això ha implicat que, a part de separar en numerador i denominador, que transformin les citades equacions de la següent forma per a què aquesta comissió sigui un nombre enter:<sup>48</sup>

$$t'_x = \frac{x * t_y * (1000 - fee)}{1000 * y + t_y * (1000 - fee)}$$

$$t'_{x_{numerador}} = x * t_y * (1000 - fee)$$

$$t'_{x_{denominador}} = 1000 * y + t_y * (1000 - fee)$$

$$t'_y = \frac{y * t_x * (1000 - fee)}{1000 * x + t_x * (1000 - fee)}$$

$$t'_{y_{numerador}} = y * t_x * (1000 - fee)$$

$$t'_{y_{denominador}} = 1000 * x + t_x * (1000 - fee)$$

Amb aquests canvis, *fee* podrà ser un nombre enter. Com ja hem vist, un valor de 5 aplicarà una taxa del 0,05% o un valor de 30 aplicarà una comissió del 0,3%.

En resum, per separar el numerador i el denominador i aplicant el citat canvi per a què la taxa sigui un nombre enter, s'han hagut d'afegir al contracte intel·ligent:

- Pel mètode priceEgldTokenNumerator apareixen els nous:
  - priceEgldTokenNumerator
  - priceEgldTokenDenominator
- Pel mètode priceEgldTokenNoFee apareixen els nous:
  - priceEgldTokenNoFeeNumerator
  - priceEgldTokenNoFeeDenominator
- Pel mètode priceTokenEgld apareixen els nous:
  - priceTokenEgldNumerator
  - priceTokenEgldDenominator

---

<sup>48</sup>Idea extreta dels exercicis que desenvolupen el material didàctic descrit a [29].

- I, finalment, del mètode priceTokenEgldNoFee apareixen els nous:

- priceTokenEgldNoFeeNumerator
- priceTokenEgldNoFeeDenominator

### 3.2.3 Arquitectura

Una **dApp** ens permetrà escriure en la blockchain i llegir l'emmagatzematge –o estat– del contracte intel·ligent descrit en l'apartat anterior. Com es pot comprovar en la Fig. 20 **per escriure en la cadena de blocs** la dApp haurà de llençar una **transacció** que invoqui a un mètode del contracte. Si per contra només necessitem **llegir l'estat** del contracte intel·ligent, emprarem una **API** aportada per Elrond per invocar certs mètodes del contracte, cosa que no farà pas cap canvi en la cadena de blocs.

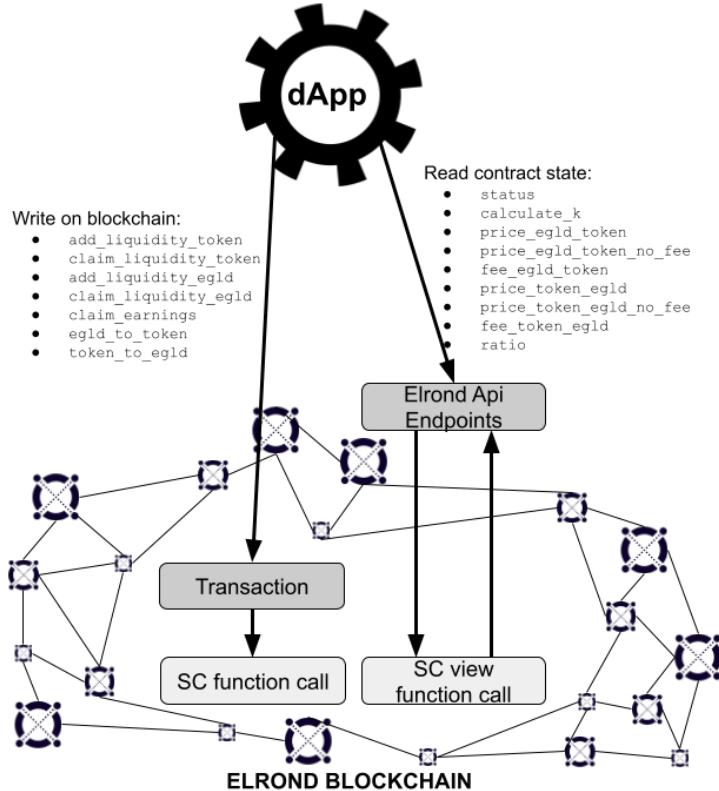
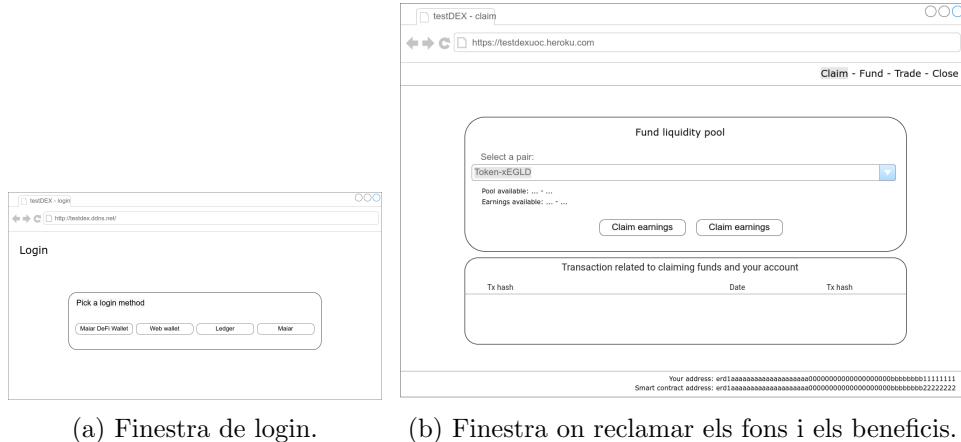


Figura 20: Arquitectura de testDEX. (Font: Elaboració pròpia inspirada en [1]).

Pot pareixer una descripció molt breu, però és que és veritablement una arquitectura molt senzilla.

### 3.2.4 Pantalles

Per accedir a la dApp s'haurà de triar un dels mètodes habilitats per fer-ho a la xarxa Elrond (Fig. 21a).

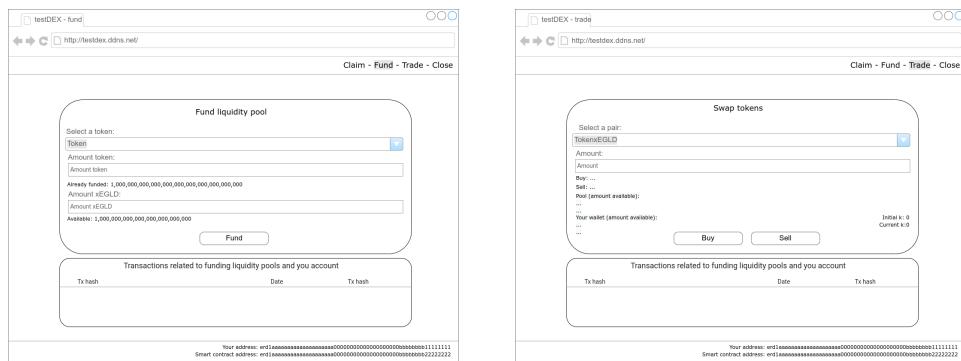


(a) Finestra de login.

(b) Finestra on reclamar els fons i els beneficis.

Figura 21: Finestres “Login” i “Claim”.

Només per a l’usuari que sigui el propietari del contracte, li apareixerà en el menú d’adalt a la dreta les opcions “Claim” i “Fund”. En la pantalla “Fund” podran crear els diferents fons de liquidesa del contracte intel·ligent (Fig. 22a) i en la finestra “Claim” es podran reclamar els fons i els beneficis obtinguts (Fig. 21b). S’ha de destacar que els fons una vegada creats, ja no es poden modificar a no ser que es realitzin *swaps* en la finestra de “Trade”. També es poden reclamar a la finestra “Claim” i a partir d’aquí es poden tonar a crear.



(a) Finestra on es provisionen els *liquidity pools*.

(b) Finestra on es poden realitzar els *swaps*.

Figura 22: Finestres “Fund” i “Trade”.

En l’opció del menú “Trade” trobem la finestra amb el cor de l’aplicació.

plicació (Fig. 23b). Es podran seleccionar el parells dels quals s'han donat d'alta fons de liquidesa i realitzar compres o vendes. Es mostrà les dades referents a les operacions que es faran (quantitat de *tokens* que rebrem si els comprem amb xEGLD pitjan el botó “Buy” o quantitat de xEGLD que rebrem si venem el citat *token* prement el botó “Sell”). També apareixeran les quantitats disponibles del parell dins del *pool* i dins de la *wallet* de l'usuari. Abaix a la dreta es pot observar que es veurà l'estat de la constant *k* en el moment actual (“*Current k*”) i quan es va inicialitzar el fons (“*Initial k*”)

### 3.3 Implementació

#### 3.3.1 Repositoris disponibles

He dividit el projecte en quatre repositoris a Github:

- **TFM\_latex**<sup>49</sup>: El present document amb tots els recursos necessaris per generar-lo.
- **TFM\_smart\_contract**<sup>50</sup>: Contracte intel·ligent del projecte escrit en Rust (es pot consultar en l'arxiu “testdex.rs”). Es troben també altres recursos necessaris per compilar-lo, així com d'altres per fer-ne proves.
- **TFM\_dapp**<sup>51</sup>: Aquí es troba la dApp que interacciona amb el contracte intel·ligent del punt anterior. Es basa en la plantilla “dapp-template” aportada per Elrond<sup>52</sup>.
- **TFM\_stats**<sup>53</sup>: Es tracta d'un *script* escrit en Python per comprovar el rendiment de la xarda d'elrond. Es llançaran peticions emprant *erdpy* al contracte intel·ligent descrit.

#### 3.3.2 Aspectes a destacar de la implementació

En primer lloc, vull destacar les diferents formes en què s'ha interactuat des de la dApp amb la *blockchain*:

- **Transferències ESDT**: Són transaccions signades. Tenen la característica de què el valor de la transacció (camp “*value*”) serà de 0 xEGLD. El token, la quantitat del mateix el mètode que es trucarà del *smart contract* i els respectius paràmetres aniran dins del camp “*data*” (al principi d'aquest camp s'indicarà que és aquest tipus de transferència amb el text “*ESDTTransfer*”). Es pot observar en la Fig. .

---

<sup>49</sup>[https://github.com/sergiogrubio/TFM\\_latex](https://github.com/sergiogrubio/TFM_latex)

<sup>50</sup>[https://github.com/sergiogrubio/TFM\\_smart\\_contract](https://github.com/sergiogrubio/TFM_smart_contract)

<sup>51</sup>[https://github.com/sergiogrubio/TFM\\_dapp](https://github.com/sergiogrubio/TFM_dapp)

<sup>52</sup><https://github.com/ElrondNetwork/dapp-template>

<sup>53</sup>[https://github.com/sergiogrubio/TFM\\_stats](https://github.com/sergiogrubio/TFM_stats)

- **Transferències de xEGLD:** També són transaccions signades. En el camp “*value*” anirà el valor de la transacció expressat en “wei” d’EGLD. En el camp “*data*” es posarà el mètode que es trucarà del *smart contract* i els respectius paràmetres.
- **Queries al smart contract:** No duen assiat una transacció signada. Es demana l’execució d’un mètode del contracte intel·ligent que no duu associat cap transferència d’actius, simplement consulta l’estat del mateix. En la Fig. ?? es pot observar un exemple.
- **Consulta a l’API REST de la devnet d’Elrond**<sup>54</sup>: S’ha comentat de passada la seva existència a la secció 2.13. Permet consultar dades sobre l’estat dels diferents elements de la devnet d’Elrond (òbviament, existeixen les respectives versions per a les altres xarxes). La seva utilitat m’ha semblant determinant. Per exemple, l’empro per consultar els actius del monedor d’un client o els actius disponibles en el contracte intel·ligent. Un exemple d’ús es pot consultar en la Fig. 25.

```
const swapTokenForEgld = async (
  pValue: string,
  pToken: string,
  pAddress: string,
  pGas: number
) => {
  const transaction = {
    value: '0',
    data: `'ESDTTransfer' +
      '@' + // token identifier in hexadecimal encoding
      hexEncodeStr(pToken) +
      '@' + // value to transfer in hexadecimal encoding
      hexEncodeNumber(pValue) +
      '@' + // name of method to call in hexadecimal encoding
      hexEncodeStr('swapTokenForEgld'),
    receiver: pAddress,
    gasLimit: pGas
  };

  const sessionId = await myTransactions([transaction]);

  if (sessionId != null) {
    setTransactionSessionId(sessionId);
  } else {
    console.log('swapTokenForEgld error sessionId = null');
  }
};
```

(a) Exemple de transferència ESDT.

```
const swapEgldForToken = async (
  pValue: string,
  pToken: string,
  pAddress: string,
  pGas: number
) => {
  const transaction = {
    value: pValue,
    data: `swapEgldForToken' + '@' + hexEncodeStr(pToken),
    receiver: pAddress,
    gasLimit: pGas
  };
  const sessionId = await myTransactions([transaction]);

  if (sessionId != null) {
    setTransactionSessionId(sessionId);
  } else {
    console.log('swapEgldForToken error sessionId = null');
  }
};
```

(b) Exemple de transferència xEgld.

Figura 23: Els dos tipus de transferències d’actius en el codi font de la dApp.

### 3.3.3 Correcció de la constant *k* per l’error introduït pel redondeig

Com s’ha comentat en la part de disseny, a la màquina virtual d’Elrond se li ha llevat el suport a les operacions de coma flotant. Com s’ha dit també, això provocarà que es vagi acumulant un error en la constant *k* que podria desvirtuar la natura del protocol AMM. Així com està implementat el *smart*

<sup>54</sup> <https://devnet-api.elrond.com/>

```

import { refreshAccount, transactionServices } from '@elrondnetwork/dapp-core';
import {
  Address,
  ContractFunction,
  ProxyProvider,
  Query
} from '@elrondnetwork/erdjs';

// TODO: change any for an interface
export const myTransactions = async (pTransactions: any[]) => {
  const { sendTransactions } = transactionServices;

  await refreshAccount();

  const { sessionId /*, error*/ } = await sendTransactions({
    transactions: pTransactions,
    redirectAfterSign: false
  });

  return sessionId;
};

```

Figura 24: Funció emprada en les transferències de la Fig. 23.

```

import { ApiNetworkProvider } from '@elrondnetwork/erdjs-network-providers';
export class CustomNetworkProvider extends ApiNetworkProvider {
  async getTakesAddress(string) {
    return await this._doGetGeneric('accounts/$address/tokens');
  }

  async getTokenDataAddress(string, token:string) {
    return await this._doGetGeneric('accounts/$address/tokens/$token');
  }

  async isOwner(string, addressOwner:string) {
    const { ownerAddress } = await this._doGetGeneric('accounts/$addressSC');
    return ownerAddress === addressOwner;
  }

  export const getProvider = () => {
    return new CustomNetworkProvider(providerAddress, {
      timeout: 1000
    });
  };
}

```

Figura 25: Consultes a l'API REST de la devnet d'Elrond.

*contract*, aquest error es va acumulant en el fons del cotntracte intel·ligent. Òbviament, és un error lleu que farà que el propietari del pool tingui més benefici. Mostra d'aquest error que es va acumulant es pot veure en la Fig. 26

He perdut temps intentant solucionar el problema i he arribat a diverses solucions on la  $k$  es va ajustant quan passa per sopa o per sobre del valor original. El problema d'aquestes solucions és que no les he pogudes demostrar matemàticament (cosa no apropiada per a un treball acadèmic). En la Fig. 27 es pot veure comentat un exemple d'aquestes proves (n'hi ha més al repositori de Github corresponent).

## 3.4 Posada en producció

### 3.4.1 Desplegament del contracte intel·ligent

Començarem compilant el contracte intel·ligent. Executarem dins el directori on es trobi el codi font del contracte intel·ligent:<sup>55</sup>.

```
erdpy contract build
```

Amb això generarem el fitxer “testdex.wasm” que desplegarem a la *devnet* d'Elrond amb la comanda:

<sup>55</sup>També es pot fer amb els *snippets*, que seran explicats en la secció 3.5.1.

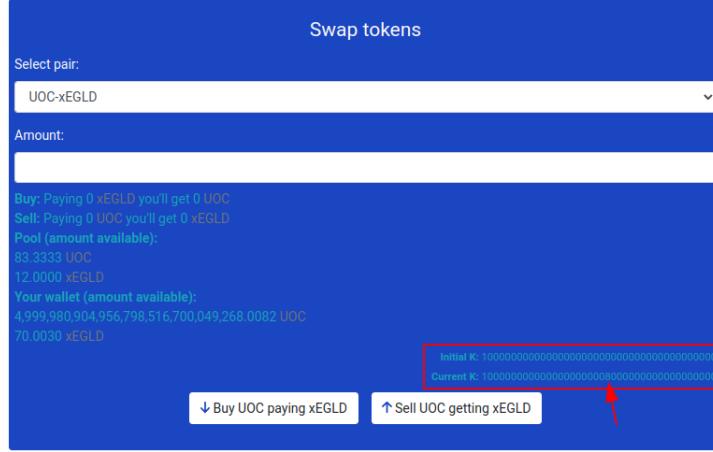


Figura 26: Error acumulat en la constant  $k$ .

(a) Mètode “swapTokenForEgld”

```

    // Swap egld for token
    int quantity;
    string token; // token that identifies the pair
    string wallet; // wallet for subtracted egld to the customer's wallet

    #endregion (swap)(dfToken)
    #payable(*)
    #region swap_for_token(string token, string tokenIdentifier) : ScResult<> <> {
        var self = self.wallet(token);
        var state = self.state(token);

        require(!state.isFrozen);
        require(state.status == "Successful",
            "Pair still funding!");
        }

        // egld paid for token with fees
        var payment = self.call_value() egld_value(); // GOLD
        var amount = self.get_egld_with_fees();
        let tokens_fee = self.get_egld_with_fees(token, &payment);
        // token bought with egld without fees
        let token_no_fee = self.get_price_egld_token_no_fee(token, &payment);
        let earnings_token = token_no_fee - tokens_fee;
        let token_no_fee = token_no_fee - tokens_fee;
        let caller = self.blockchain().get_caller();

        self.liquidity[token].update(liquidity_egld * liquidity_gold + &payment);
        self.liquidity[token].update(liquidity_token - liquidity_token * token_no_fee);
        self.earnings[token].update(earnings * earnings - token_no_fee);

        // Adjusting K constant
        // I correct it adjusting the token side of the pair
        // if now k is smaller than the last correction
        // is to get it from the tokens egld to the customer's wallet
        // let now_k = self.calculate_k(liquidity);
        // let new_k = now_k * self.get_egld_with_fees();
        // let new_k = self.get_egld_with_fees();
        if now_k < initial_k {
            self.liquidity[token].get();
            let new_liq_token = self.liquidity[token].get();
            let earnings_token = self.earnings[token].get();
            let final_correction = new_liq_token * liquidity_egld / now_liq_token;
            let amount_correction = new_liq_token_corrected * new_liq_token;
            let final_correction;
            if amount_correction > earnings_token {
                final_correction = amount_correction;
            } else if amount_correction < earnings_token {
                final_correction = earnings_token;
            }
            self.liquidity[token].update(liquidity_token * liquidity_token + final_correction);
            self.earnings[token].update(earnings * earnings + final_correction);
        } else if now_k > initial_k {
            self.liquidity[token].get();
            let new_liq_token = self.liquidity[token].get();
            let earnings_token = self.earnings[token].get();
            let new_k = self.calculate_k(liquidity);
            let new_liq_token = self.get_egld_with_fees();
            let new_liq_token_corrected = new_liq_token * liquidity_egld;
            let amount_correction = new_liq_token * liquidity_egld;
            let final_correction;
            if amount_correction > earnings_token {
                final_correction = amount_correction;
            } else if amount_correction < earnings_token {
                final_correction = earnings_token;
            }
            self.liquidity[token].update(liquidity_token * liquidity_token + final_correction);
            self.earnings[token].update(earnings * earnings + final_correction);
        }
    }

    // send token bought tokens for all customer address
    self.send(caller, token, token_no_fee, 4);
}

```

(b) Mètode “swapEgldForToken”.

Figura 27: Exemple d'implementació de l'ajustament de la constant  $k$ .

```
erdpy contract deploy --pem=~/.wallet/wallet1.pem" \
--recall-nonce --gas-limit=100000000 --project=. \
--proxy="https://devnet-gateway.elrond.com" \
--chain="D" --arguments 0x05 --send
```

D'entre els paràmetres, s'ha de destacar que dins de “/wallet/wallet3.pem” es troba la clau privada de l'usuari de la devnet d'Elrond que desplegarà el

contracte intel·ligent a la xarxa (que serà el propietari) i amb “–arguments 0x05” li passem en hexadecimal l’argument necessari pel mètode que fa de constructor (un valor de 5 implica una comissió del 0.05%). Aquest comanda mostrerà per la consola l’adreça del contracte intel·ligent dins de la *devnet* d’Elrond, en el nostre cas:

```
erd1qqqqqqqqqqqpgq578zh88hskf9efwzyhkf64el7d6ve3lrsn2qwkvmmt2
```

### 3.4.2 Desplegament de la dApp

He donat d’alta dins del servei [noip.com](https://noip.com) el nom testdex.ddns.net per a què apunti a un VPS (*Virtual Private Server*) a OVH<sup>56</sup>. Al servidor en clonat el repositori amb l’ordre:

```
git clone git@github.com:sergiogrubio/TFM_dapp.git
```

Dins del directori clonat he executat:

```
npm run build  
serve -s build
```

La dApp estarà disponible durant els temps d’avaluació d’aquest TFM al URL: <http://testdex.ddns.net/>.

## 3.5 Proves

### 3.5.1 Sobre el contracte intel·ligent

Les proves sobre el contracte intel·ligent tenen una especial importància, ja que errades en el seu disseny i/o implementació poden provocar pèrdues econòmiques. Per testejar el contracte intel·ligent allò primer que he fet és definir **“interaccions” mitjançant snippets** que empren l’eina *erdpy*. En gerga d’Elrond, es crea un fitxer d’interacciones (en el nostre cas anomenat “devnet.snippets.sh”). Després fent clic amb el botó dret al *plugin*<sup>57</sup> d’Elrond a Visual Studio Code podrem accedir als *snippets* (Fig. 28).

Hi ha definit un *snippet* per a cada mètode del contracte intel·ligent, a més de dos addicionals per compilar-lo i desplegar-lo a la xarxa d’Elrond. Ja s’han comentat anteriorment els mètodes del contracte intel·ligent i els respectius snippets permeten executar-los (i així comprovar si el comportament és l’esperar). Vull parar l’atenció en els que penso que són il·lustratius del funcionament d’*erdpy*:

- **Trucades a mètodes del contracte intel·ligent que impliquen transferència d’actius (*erdpycontractcall*):**

---

<sup>56</sup><https://www.ovhcloud.com/>

<sup>57</sup>Com configurar l’entorn per desenvolupar en la xarxa Elrond es pot consultar en [31] i [32].

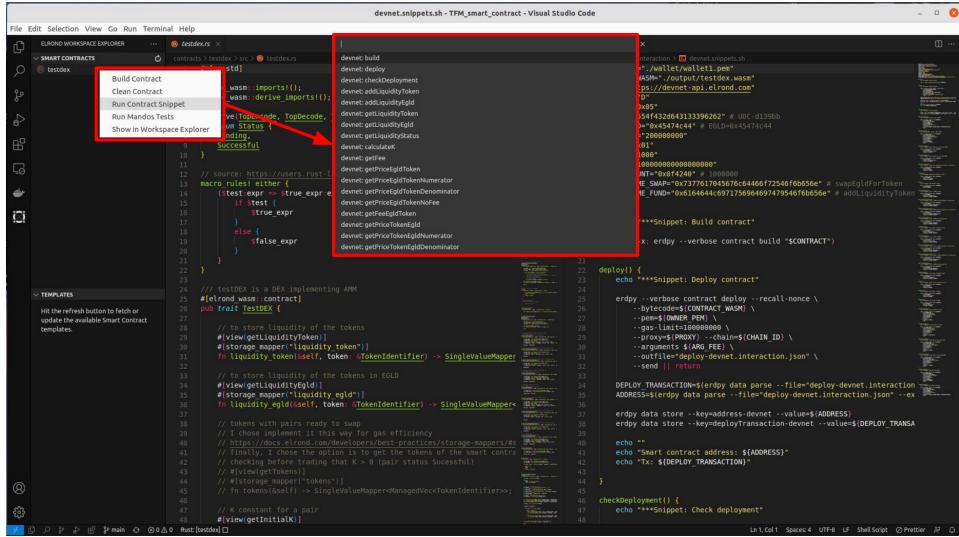


Figura 28: *Snippets* en l'IDE Visual Studio Code.

- **Transferència d'EGLD del client cap al contracte** (*snippet “addLiquidityEgld”*): En la Fig. 30a es pot veure un exemple de com transferir EGLD al contracte intel·ligent. S’ha de destacar que l’opció “--value” inclourà la quantitat d’EGLD a traspassar en wei i l’opció “--function” especificarà el mètode del contracte amb la lògica que tractarà el citat moviment (els argument que se li passen es definiran en “--arguments” i estaran codificats en hexadecimal). Dins el mètode del contracte intel·ligent es podrà recuperar la quantitat d’EGLD amb:

```
let payment = self.call\_value().egld\_value();
```

- **Transferència de tokens ESDT del client cap al contracte** (*snippet “addLiquidityToken”*): Aquest cas és un exemple de com transferir tokens ESDT (Fig. 30b). Aquí no s’especificarà opció “`--value`” i en l’opció “`--function`” es posarà el text “ESDT-Transfer”. En l’opció arguments –codificat en hexadecimal– es passarà el nom del token, la quantitat del mateix i el nom del mètode del contracte amb la lògica per gestionar la trucada. Ja dins el mètode del contracte intel·ligent, el token i la quantitat es podran recuperar amb:

```
let (payment, token) =  
    self.call_value().payment_token_pair();
```

- **Transferència d'EGLD/token del contracte cap al client** (*snippet “claimLiquidityToken”*): Aquí es cridarà un mètode que generarà una transferència –tant d'EGLD com de tokens ESDT–

cap a algun *wallet* (Fig. 29c). L'opció “--value” s'especificarà amb valor 0 i en l'opció “--function” es posarà el mètode del contracte amb la lògica que tractarà el citat moviment els argument que se li passen es definiran en “--arguments” i estaran codificats en hexadecimal).

- **Trucades a mètodes del contracte intel·ligent sense transferència d'actius (erdpy contract query):** En la Fig. 29d es pot vure aquest cas. Serà suficient amb especificar el mètode que es crida i els arguments codificats en hexadecimal. Com que no es tracta d'una trasacció que modifiqui la *blockchain*, és molt més ràpida.

```
addLiquidityEgld() {
    echo "****Snippet: Funding a pool (EGLD)"
    erdpy --verbose contract call ${ADDRESS} \
        --pem=${OWNER_PEM} \
        --gas-limit=${GAS_LIMIT} \
        --function="addLiquidityEgld" \
        --proxy=${PROXY} \
        --chain=${CHAIN_ID} \
        --recall-nonce \
        --value ${AMOUNT2} \
        --arguments ${TOKEN} \
        --send
}
```

(a) “addLiquidityEgld”.

```
addLiquidityToken() {
    echo "****Snippet: Funding a pool (token)"
    erdpy --verbose contract call ${ADDRESS} \
        --pem=${OWNER_PEM} \
        --gas-limit=${GAS_LIMIT} \
        --proxy=${PROXY} \
        --chain=${CHAIN_ID} \
        --function="ESDTTransfer" \
        --recall-nonce \
        --arguments ${TOKEN} ${TOKEN_AMOUNT} ${METHOD_NAME_FUND} \
        --send
}
```

(b) “addLiquidityToken”.

```
claimLiquidityToken() {
    echo "****Snippet: Claim TOKEN of a pair EGLD-TOKEN"
    erdpy --verbose contract call ${ADDRESS} \
        --pem=${OWNER_PEM} \
        --gas-limit=${GAS_LIMIT} \
        --function="claimLiquidityToken" \
        --proxy=${PROXY} \
        --chain=${CHAIN_ID} \
        --recall-nonce \
        --value 0 \
        --arguments ${TOKEN} \
        --send
}
```

(c) “claimLiquidityToken”.

```
claimEarningsToken() {
    echo "****Snippet: Claim earnings of a token"
    erdpy --verbose contract call ${ADDRESS} \
        --pem=${OWNER_PEM} \
        --gas-limit=${GAS_LIMIT} \
        --function="claimEarnings" \
        --proxy=${PROXY} \
        --chain=${CHAIN_ID} \
        --recall-nonce \
        --value 0 \
        --arguments ${TOKEN} \
        --send
}
```

(d) “getFee”.

Figura 29: *Snippets* amb trucades a mètodes emprant *erdpy*.

S'han executat els següents **snippets** per validar el contracte intel·ligent<sup>58</sup>:

- Compilar el contracte intel·ligent: *Snippet* “build”.
- Desplegar el contracte intel·ligent: *Snippet* “deploy”. Com a paràmetre si li passa la taxa (*fee*) que es cobrarà en els *swaps*.
- Comprovar que el contracte s'ha desplegat correctament: *Snippet* “check-Deployment”.

<sup>58</sup>S'ha de destacar que en el resultat del *snippet* apareixerà un enllaç cap a l'explorador de transaccions de la xarxa d'Elrond on es podrà comprovar si ha funcionat correctament o ha aparegut algun error (òbviament, es provaran situacions que haurien de generar error i en aquest cas el comportament anòmal és que funcioni correctament).

- Afegir un fons de liquiditat: *Snippets* “addLiquidityToken” (Fig. 30a) i “addLiquidityEgld” (Fig. 30b). Es torna a afegir fons al parell per comprovar que no deixa. Finalment, s’intenta crear un fons des d’una cartera que no és la propietaria del contracte intel·ligent per comprovar que dona error (dins del fitxer *devenet.snippets.sh* s’ha de canviar la constant *OWNER\_PEM*=“./wallet/wallet1.pem” per *OWNER\_PEM*=“./wallet/wallet2.pem”).
- Obtenir la liquiditat del *pool*: *Snippets* “getLiquidityToken” i “getLiquidityEgld”. Òbviament, els valors obtinguts hauran de coincidir amb les quantitats que s’han especificar en la creació del fons en el punt anterior.
- Obtenir l’estat del fons de liquiditat: *Snippet* “getLiquidityStatus”. Executat abans de constituir el fons i després de constituir-lo (ha de donar valors diferents).
- Calcular la constant K actual: *Snippets* “calculateK” i “getRatio”. Per l’error de rendodeig en la divisió entre nombres sencers que s’ha comentat abans, anirà fluctuant un poc per sobre i per sota del valor de la constant del moment de constituir el fons. S’ha comprovat fent *swaps* que així és.
- Comprovar que la taxa (*fee*): *Snippet* “getFee”. La taxa ha de ser la mateixa que es va definir en el moment de desplegar el contracte intel·ligent.
- Calcular el preu en token de l’EGLD (amb i sense taxa): *Snippets* “getPriceEgldToken”, “getPriceEgldTokenNumerator”, “getPriceEgldTokenDenominator”, “getPriceEgldTokenNoFee” i “getFeeEgldToken”<sup>59</sup>. S’ha comprovat que es generen els mateixos preus que en l’exemple descrit en la secció 3.1.1.
- Calcular el preu en EGLD del token (amb i sense taxa): *Snippets* “getPriceTokenEgld”, “getPriceTokenEgldNumerator”, “getPriceTokenEgldDenominator”, “getPriceTokenEgldNoFee” i “getFeeTokenEgld”<sup>60</sup>. S’ha comprovat que es generen els mateixos preus que en l’exemple descrit en la secció 3.1.1.
- Comprovar els beneficis obtinguts pels swaps realitzats: *Snippets* “getEarningsEgld” i “getEarningsToken”. També s’han comprovat que complien amb l’estudi teòric fet a la secció 3.1.1.

---

<sup>59</sup>Hi ha tres *snippets* per coincidir amb la implementació que s’ha exposat en la secció 3.5.1.

<sup>60</sup>Hi ha tres *snippets* per coincidir amb la implementació que s’ha exposat en la secció 3.5.1.

- Comprovar que el propietari del contracte rep els beneficis d'un token concret quan els reclama: Snippet “claimEarningsToken()”. S'ha d'emprar també el snippet “getEarningsToken” per esbrinar la quantitat a rebre i després d'executar-lo snippet ha d'aparèixer a la cartera del propietari del *smart contract*.
- Comprovar que el propietari del contracte rep els beneficis en EGLD quan els reclama: Snippet “claimEarningsEgld()”. S'ha d'emprar també el snippet “getEarningsEgld” per esbrinar la quantitat a rebre i després d'executar-lo ha d'aparèixer a la cartera del propietari del *smart contract*. Per disseny, l'EGLD que es recuperarà serà el generat amb tots els *swaps* (independentment del parell).
- Comprovar que el propietari del contracte rep la part d'EGLD d'un fons de liquiditat quan la reclama: Snippet “claimLiquidityEgld()”. S'ha d'emprar també el snippet “getLiquidityEgld” per esbrinar la quantitat a rebre i després d'executar-lo ha d'aparèixer a la cartera del propietari del *smart contract*. A diferència del que passa amb “claimEarningsEgld()”, aquí només es rebrà l'EGLD d'un parell en concret.
- Comprovar que el propietari del contracte rep la part del token d'un fons de liquiditat quan la reclama: Snippet “claimLiquidityToken()”. S'ha d'emprar també el snippet “getLiquidityToken” per esbrinar la quantitat a rebre i després d'executar-lo ha d'aparèixer a la cartera del propietari del *smart contract*.
- Comprovar que es poden intercanviar EGLD per token: Snippet “swapEgldForToken()”. S'ha comprovat que es generen els mateixos resultats que els descrits en l'exemple de la secció 3.1.1. Així mateix, s'emprarà el snippet “getPriceEgldToken” per conèixer la quantitat de token que hem de rebre a la cartera.
- Comprovar que es poden intercanviar token per EGLD: Snippet “swapTokenForEgld()”. S'ha comprovat que es generen els mateixos resultats que els descrits en l'exemple de la secció 3.1.1. Així mateix, s'emprarà el snippet “getPriceTokenEgld” per conèixer la quantitat d'EGLD que hem de rebre a la cartera.

Finalment, Elrond disposa d'una eina anomenada **Mandos i d'una testnet** (local), que permeten **automatitzar les proves** en un entorn controlat ([1], seccions “*Mandos tests reference*” i “*Setup a Local Testnet*”). A part de per una qüestió merament de temps (són eines específiques que requereixen un temps d'aprenentatge i no podia abarcar-ho tot), per l'objectiu de posar a prova el rendiment de la xarxa Elrond he triat realitzar les proves directament a la devnet amb els *snippets*. Com s'ha dit, els test de rendiment es faran en aquesta xarxa.

(a) “addLiquidityEgld”.

(b) “addLiquidityToken”.

Figura 30: Execució de proves per afegir un fons de liquiditat.

### 3.5.2 Sobre la dApp

S'han comprovar manualment els casos d'ús descrits en la secció a la dAPP.

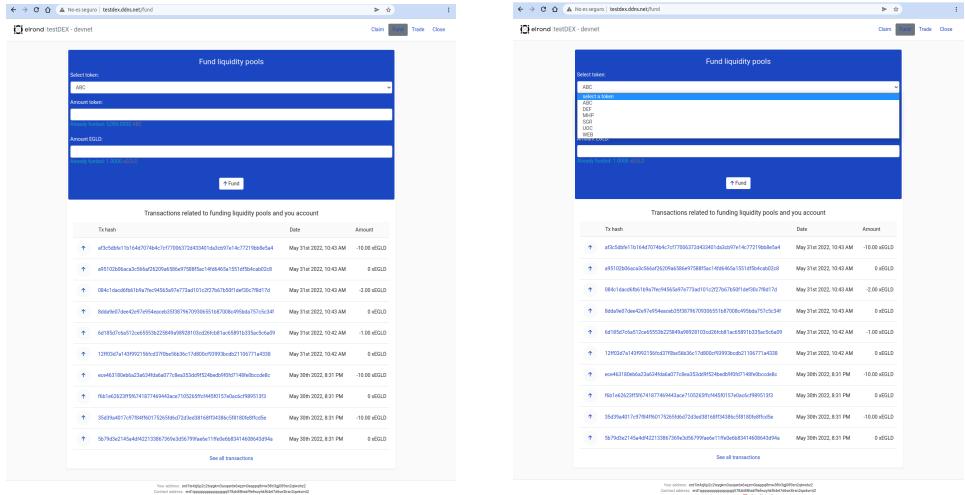
1. S'ha afegit liquiditat: En la finestra que apareix en l'opció “Fund” en la llista desplegable apareixen els *tokens* disponibles als moneder de l'usuari que ha accedit a testDEX (Fig. 31b). El fons quan estigui definit no se li podrà afegir més quantitat d'actius. En qualsevol cas, si es tria el token que defineix el fons apareixerà la quantitat d'actius que té assignat (Fig. 31a).
  2. S'han realitzat *swap* confirmant que els resultats coïncideixen amb l'anàlisi teòric fet a la secció 3.1.1. En la figura ?? es poden comprovar que coincideixen preus calculats, valors transferits i beneficis obtinguts.
  3. S'han recuperat beneficis i fons. En la Fig. 32 es pot veure l'estat inicial del fons i com queda després de reclamar els beneficis i els fons en sí mateixos.
  4. En totes les finestres de la dApp es mostraran les transaccions relacionades. Filtre les transaccions segons els mètodes del contracte intel·ligent trucats.

Un darrer comentari és que he decidit no limitar els nombres que l'usuari pot introduir als quadres de text amb les quantitats. Així es pot provar el comportament del contracte intel·ligent en casos extrems.

M'ha gués agradat automatitzar el procediment de proves amb algun *framework* de React però no he tingut temps material per formar-me.

### 3.5.3 Sobre el rendimiento de la *devnet* d'Elrond

Un dels objectius del present treball era definir proves per comprovar les característiques de la xarxa Elrond. Per altra banda, en la secció 2 s'ha



(a) Estat del fons ABC-xEGLD.

(b) *Tokens* al wallet del client.

Figura 31: Fons de líquides a crests



Figura 32: Recuperació de fons de liquidesa i beneficis generats

fet una revisió de la tecnologia d'Elrond, en la 2.9 s'han comentat les característiques bàsiques de la *devnet*, en la 3.2.3 l'arquitectura de testDEX i en la 3.3.2 les diferents formes en què ens hem comunicat a la *blockchain*. Amb la intenció de mesurar el temps que triga en realitzar les peticions s'ha realitzat un script en Python (que com he dit es troba en el repositori TFM\_stats).

Amb tot això, les transaccions per minut pot ser un bon indicador per valorar el rendiment de la devnet. Com s'ha vist, si el moneder que envia la transacció es troba la mateix *shard* que el contracte intel·ligent la transacció serà molt més ràpida que si es troba en un altre *shard*. Des de Python he trobat una solució per enviar transaccions en bloc, però per recuperar-les no he trobat forma de disparar automàticament una funció que tracti la resposta. Ho he implementat amb un bucle que demana les dades de les transaccions segons el seu *hash* a l'API d'Elrond fins que totes tenen l'estat

Swap tokens

Select pair: UOC-xEGLD

Amount: 1

Buy UOC paying xEGLD you'll get 0.0009 xEGLD  
Sell UOC getting xEGLD you'll get 0.0009 UOC  
Post (amount available): 0.0009 xEGLD  
0.0009 UOC  
Your wallet (amount available): 0.000999999999999999 UOC  
0.000999999999999999 xEGLD

↓ Buy UOC paying xEGLD    ↑ Sell UOC getting xEGLD

(a) Preu d'un xEGLD front a UOC.

Swap tokens

Select pair: UOC-xEGLD

Amount: 1000

Buy UOC paying xEGLD you'll get 0.0009 xEGLD  
Sell UOC getting xEGLD you'll get 0.0009 UOC  
Post (amount available): 0.0009 xEGLD  
0.0009 UOC  
Your wallet (amount available): 0.000999999999999999 UOC  
0.000999999999999999 xEGLD

↓ Buy UOC paying xEGLD    ↑ Sell UOC getting xEGLD

(c) Preu de mill xEGLD front a xEGLD.

Swap tokens

Select pair: UOC-xEGLD

Amount: 1

Buy UOC paying xEGLD you'll get 0.0009 xEGLD  
Sell UOC getting xEGLD you'll get 0.0009 UOC  
Post (amount available): 0.0009 xEGLD  
0.0009 UOC  
Your wallet (amount available): 0.000999999999999999 UOC  
0.000999999999999999 xEGLD

↓ Buy UOC paying xEGLD    ↑ Sell UOC getting xEGLD

(b) Resultat de *swap* d'un xEGLD per UOC.

Claim earnings and liquidity pools

Select pair: UOC-xEGLD

Post available: 0.0009 xEGLD - 10,000.0000 UOC  
Change available: 0.0009 xEGLD - 10,000.0000 UOC

↓ Claim earnings    ↓ Claim pool

(d) Beneficis després de fer un *swap* d'un xEGLD per UOC i seguit de 1000 UOC per xEGLD.

Figura 33: *Swaps* realitzats segons allò descrit en la secció 3.1.1.

a “*success*”. Això s’ha de tenir en compte perquè els resultats que es mostren per aquest motiu tenen un retard afegit.

El que fa el *script* és enviar una transacció que truca el mètode del contracte intel·ligent “addLiquidityEgld” i passant-li com a paràmetre “ABC-109739”. A més la transacció enviarà un valor de 0.01 xEGLD. Amb això el que estem fent és fundar amb xEGLD el fons ABC-xEGLD. En la Fig. 34 es veu un exemple d’execució per a un total de 5 transaccions (5 transaccions executades en 2,896 segons).

La prova “bona” la faré enviant 1000 transaccions. Això implica que s’enviaran 10 xEGLD al fons. En la Fig. 35 es pot observar l’estat del fons abans i després de l’execució i en la Fig. ?? el resultat (1000 transaccions en 3 minuts i 40 segons).

A partir de les 2500 transaccions el *script* falla, però per a 2000 ha donat el resultat de realitzades en 7 minuts i 7 segons (Fig. 37).

Una darrera prova és fer la transacció amb un moneder que es trobi en un altre *shard*. Com que no serà el propietari del contracte intel·ligent, aquest respondrà que no pot fundar el fons. Això implica que la màquina virtual no executarà pràcticament codi. Per aquest motiu he tornat a llançat el test amb un wallet que no es propietari del contracte intel·ligent però que es troba al mateix *shard*, per a 5 transaccions (Fig. 38) ha trigat 1 segond

(1852 ms) i per a 2000 ha tardat 7 miunts i 3 segons (Fig. 39).

Dit tot això, la prova ja des d'un compte que no es propietari del contracte i que no es troba al mateix *shards* ha trigat 31 segons per a 5 transaccions (Fig. 40) i per a 2000 transaccions 7 minuts i 31 segons (Fig. 41).

Núm. trans.	Shard	Owner	Figura	Temps (ms)
5	0	sí	34	2151
1000	0	sí	36	220959
2000	0	sí	37	427788
5	0	no	38	1852
2000	0	no	39	423468
5	2	no	40	31094
2000	2	no	41	451136

```
Starting parallel 2022-05-31 22:44:28
Funding ABC-xEGLD liquidity pool with 0.05 xEGLD among 5 transactions
please wait a few seconds
5 transactions were sent
-----
Hash: a4984f404700732c441d9a184e919f8accc6b62563149ab6dd08b0f76e48ae3973 - Status: success
Elrond age: 2022-05-31 22:44:28 2022-05-31 22:44:30 - Start time: 2022-05-31 22:44:28 - End time: 2022-05-31 22:44:30 - Diff (end t. - start t.): 1326
Epoch: 760 - Round: 913045 - Source shard: 0 - Destination shard: 0
-----
Hash: 1a30bhv177dbd0199d73082a765c0838801d765eb660535d95a002ae00000ae2c - Status: success
Elrond age: 2022-05-31 22:44:28 2022-05-31 22:44:30 - Start time: 2022-05-31 22:44:28 - End time: 2022-05-31 22:44:30 - Diff (end t. - start t.): 1539
Epoch: 760 - Round: 913045 - Source shard: 0 - Destination shard: 0
-----
Hash: d3f7407f7e471d45a0cef0a388c522e029291d57ecf5b64d038574f02354ec6a - Status: success
Elrond age: 2022-05-31 22:44:28 2022-05-31 22:44:30 - Start time: 2022-05-31 22:44:28 - End time: 2022-05-31 22:44:30 - Diff (end t. - start t.): 1737
Epoch: 760 - Round: 913045 - Source shard: 0 - Destination shard: 0
-----
Hash: d3e73664cad209eaaab1e42124219d3998b3e9be3e30a3a56a2b364988ae76d - Status: success
Elrond age: 2022-05-31 22:44:28 2022-05-31 22:44:30 - Start time: 2022-05-31 22:44:28 - End time: 2022-05-31 22:44:30 - Diff (end t. - start t.): 1966
Epoch: 760 - Round: 913045 - Source shard: 0 - Destination shard: 0
-----
Hash: e6ad025a46fcfb22d3eadhb6f6eeccc723a7d78f7320265766772c82f8e4a76f9 - Status: success
Elrond age: 2022-05-31 22:44:28 2022-05-31 22:44:30 - Start time: 2022-05-31 22:44:28 - End time: 2022-05-31 22:44:31 - Diff (end t. - start t.): 2151
Epoch: 760 - Round: 913045 - Source shard: 0 - Destination shard: 0
5 transactions executed successfully in 0 minutes 2 seconds (2151 ms).
They were sent from this computer at 2022-05-31 22:44:28.
They arrived to an Elrond devnet node at 2022-05-31 22:44:28.
This computer detected querying the Elrond API that they ended at 2022-05-31 22:44:31.
```

Figura 34: Resultat del *script* de Python executat per a 5 transaccions.

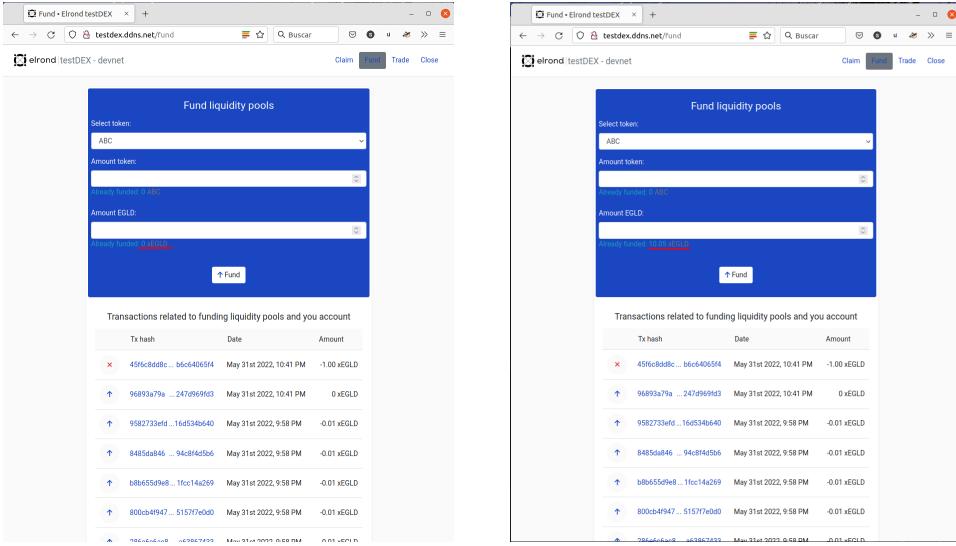


Figura 35: Fons de liquidesa ABC-xEGLD abans i després d'executar el *script*.

```

Hash: 30116e0a26633148af4ff739861ff51c143592eee5b4d97a52e639730362066 - Status: success
Elrond age: 2022-05-31 22:49:40 2022-05-31 22:49:48 - Start time: 2022-05-31 22:49:40 - End time: 2022-05-31 22:53:21 - Diff (end t. - start t.): 220391
Epoch: 760 - Round: 913098 - Source shard: 0 - Destination shard: 0
-----
Hash: 64452978ed875f06cd752930bb3a0ba9ehd515d5c958de3a1a485dc27aa03c - Status: success
Elrond age: 2022-05-31 22:49:40 2022-05-31 22:49:48 - Start time: 2022-05-31 22:49:40 - End time: 2022-05-31 22:53:21 - Diff (end t. - start t.): 220583
Epoch: 760 - Round: 913098 - Source shard: 0 - Destination shard: 0
-----
Hash: 6b0d0f689bd3d971794b57c291fd5090e31cc5cedf17e603b4e647348a6010 - Status: success
Elrond age: 2022-05-31 22:49:40 2022-05-31 22:49:48 - Start time: 2022-05-31 22:49:40 - End time: 2022-05-31 22:53:21 - Diff (end t. - start t.): 220766
Epoch: 760 - Round: 913098 - Source shard: 0 - Destination shard: 0
-----
Hash: d278bbf74dbb5fa9a3736f672a46eb5c2b517eaec2e093e35c2ba3a3e681c7 - Status: success
Elrond age: 2022-05-31 22:49:40 2022-05-31 22:49:48 - Start time: 2022-05-31 22:49:40 - End time: 2022-05-31 22:53:21 - Diff (end t. - start t.): 220959
Epoch: 760 - Round: 913098 - Source shard: 0 - Destination shard: 0
1000 transactions executed successfully in 1 minutes 49 seconds (220959 ms).
They were sent from this computer at 2022-05-31 22:49:40.
They arrived to an Elrond devnet node at 2022-05-31 22:49:48.
This computer detected querying the Elrond API that they ended at 2022-05-31 22:53:21.

```

Figura 36: Resultat del *script* de Python executat per a 1000 transaccions (es mostren només els darrer *hashes*).

```

Hash: f224b94f74dd793fe8b66571bfdfcd1d070fd8a26974c9804bbc00216a0599a1 - Status: success
Elrond age: 2022-05-31 22:57:56 2022-05-31 22:58:00 - Start time: 2022-05-31 22:57:56 - End time: 2022-05-31 23:05:02 - Diff (end t. - start t.): 426878
Epoch: 760 - Round: 913180 - Source shard: 0 - Destination shard: 0
-----
Hash: e98fc9bb2f5728408b5f1400fd4f62c862ebce83723f3bb616c1872ad599698 - Status: success
Elrond age: 2022-05-31 22:57:56 2022-05-31 22:58:00 - Start time: 2022-05-31 22:57:56 - End time: 2022-05-31 23:05:03 - Diff (end t. - start t.): 427119
Epoch: 760 - Round: 913180 - Source shard: 0 - Destination shard: 0
-----
Hash: 17844580c72131704789cffb2affac81230500ec2c37e53de2a8d79efb8602154 - Status: success
Elrond age: 2022-05-31 22:57:56 2022-05-31 22:58:00 - Start time: 2022-05-31 22:57:56 - End time: 2022-05-31 23:05:03 - Diff (end t. - start t.): 427346
Epoch: 760 - Round: 913180 - Source shard: 0 - Destination shard: 0
-----
Hash: f7b642a497c5d249e8866af31d35e7c68a8837f6acfabbf83c4056a1a99ffd51 - Status: success
Elrond age: 2022-05-31 22:57:56 2022-05-31 22:58:00 - Start time: 2022-05-31 22:57:56 - End time: 2022-05-31 23:05:03 - Diff (end t. - start t.): 427543
Epoch: 760 - Round: 913180 - Source shard: 0 - Destination shard: 0
-----
Hash: 77dad0bf3e0b9810a773103d3a576e459fcdd973263150e1f059fd7d7a4a - Status: success
Elrond age: 2022-05-31 22:57:56 2022-05-31 22:58:00 - Start time: 2022-05-31 22:57:56 - End time: 2022-05-31 23:05:03 - Diff (end t. - start t.): 427788
Epoch: 760 - Round: 913180 - Source shard: 0 - Destination shard: 0
2000 transactions executed successfully in 7 minutes 7 seconds (427788 ms).
They were sent from this computer at 2022-05-31 22:57:56.
They arrived to an Elrond devnet node at 2022-05-31 22:57:56.
This computer detected querying the Elrond API that they ended at 2022-05-31 23:05:03.

```

Figura 37: Resultat del *script* de Python executat per a 2000 transaccions (es mostren només els darrer *hashes*).

```

sergi@dragonfruit:~/Escritorio/TFM/TFM_stats$ python3 stats.py
Starting parallel 2022-05-31 23:08:29
Funding ABC-xEGLD liquidity pool with 0.05 xEGLD among 5 transactions
Please wait a few seconds
5 transactions were sent
-----
Hash: b9a4e4640d78c5eb5ac6c53cf4b2493seed5c696c6900de398a927394f934b - Status: success
Elrond age: 2022-05-31 23:08:29 2022-05-31 23:08:30 - Start time: 2022-05-31 23:08:29 - End time: 2022-05-31 23:08:30 - Diff (end t. - start t.): 1069
Epoch: 760 - Round: 913285 - Source shard: 0 - Destination shard: 0
-----
Hash: 8770ce5eeecd201400dd6446c316d32a3165d6a9ba5b4844b01100bb13ee3e8 - Status: success
Elrond age: 2022-05-31 23:08:29 2022-05-31 23:08:30 - Start time: 2022-05-31 23:08:29 - End time: 2022-05-31 23:08:30 - Diff (end t. - start t.): 1273
Epoch: 760 - Round: 913285 - Source shard: 0 - Destination shard: 0
-----
Hash: 1d08edd0894f1bd5789d845be3588862199f5a1c7c6aaef82d91f115d8fffafe - Status: success
Elrond age: 2022-05-31 23:08:29 2022-05-31 23:08:30 - Start time: 2022-05-31 23:08:29 - End time: 2022-05-31 23:08:30 - Diff (end t. - start t.): 1479
Epoch: 760 - Round: 913285 - Source shard: 0 - Destination shard: 0
-----
Hash: d4989aa3b5f5c69f8893dd9aab412f62e15b9f8496ebcdccda2db706c90d7645 - Status: success
Elrond age: 2022-05-31 23:08:29 2022-05-31 23:08:30 - Start time: 2022-05-31 23:08:29 - End time: 2022-05-31 23:08:30 - Diff (end t. - start t.): 1666
Epoch: 760 - Round: 913285 - Source shard: 0 - Destination shard: 0
-----
Hash: 55395b77e5771d2c9d9796ceb7ce53684c4699a944565d1f873b865f64735b - Status: success
Elrond age: 2022-05-31 23:08:29 2022-05-31 23:08:30 - Start time: 2022-05-31 23:08:29 - End time: 2022-05-31 23:08:30 - Diff (end t. - start t.): 1852
Epoch: 760 - Round: 913285 - Source shard: 0 - Destination shard: 0
5 transactions executed successfully in 0 minutes 1 seconds (1852 ms).
They were sent from this computer at 2022-05-31 23:08:29.
They arrived to an Elrond devnet node at 2022-05-31 23:08:29.
This computer detected querying the Elrond API that they ended at 2022-05-31 23:08:30.

```

Figura 38: Resultat del *script* de Python executat per a 5 transaccions amb un remitent que no és el propietari del *smart contract* però que es troba al mateix *shard*.

```

Hash: 1e80a7f081c75fe73e076f83a17c4d5b940942db888514ef8852c6ef4bc21e27 - Status: success
Elrond age: 2022-05-31 23:12:27 2022-05-31 23:12:30 - Start time: 2022-05-31 23:12:27 - End time: 2022-05-31 23:19:29 - Diff (end t. - start t.): 422377
Epoch: 760 - Round: 913325 - Source shard: 0 - Destination shard: 0
-----
Hash: 05d0d69885dc0c271101e5f62a4e14f1f938a239b9f927686675b75def6f974 - Status: success
Elrond age: 2022-05-31 23:12:27 2022-05-31 23:12:30 - Start time: 2022-05-31 23:12:27 - End time: 2022-05-31 23:19:29 - Diff (end t. - start t.): 422603
Epoch: 760 - Round: 913325 - Source shard: 0 - Destination shard: 0
-----
Hash: 33f7d7f595aaace3488b73fa15d6b07e73e118674a4c7023fa977c868fc2c - Status: success
Elrond age: 2022-05-31 23:12:27 2022-05-31 23:12:30 - Start time: 2022-05-31 23:12:27 - End time: 2022-05-31 23:19:30 - Diff (end t. - start t.): 422828
Epoch: 760 - Round: 913325 - Source shard: 0 - Destination shard: 0
-----
Hash: d0dfaf2b3d9pbde32d9a1lc1d388fce545d3372d538d2d4635f8eb13fd4ad7a - Status: success
Elrond age: 2022-05-31 23:12:27 2022-05-31 23:12:30 - Start time: 2022-05-31 23:12:27 - End time: 2022-05-31 23:19:30 - Diff (end t. - start t.): 423051
Epoch: 760 - Round: 913325 - Source shard: 0 - Destination shard: 0
-----
Hash: 063d9480d56b56ff6c5f1a61b6a1640699656544dalc1eb50785544ba - Status: success
Elrond age: 2022-05-31 23:12:27 2022-05-31 23:12:30 - Start time: 2022-05-31 23:12:27 - End time: 2022-05-31 23:19:30 - Diff (end t. - start t.): 423268
Epoch: 760 - Round: 913325 - Source shard: 0 - Destination shard: 0
-----
Hash: b0d65bd66c080b0fc4e12f3e57b1e509f21a40d90421b0b74711c999be040d9 - Status: success
Elrond age: 2022-05-31 23:12:27 2022-05-31 23:12:30 - Start time: 2022-05-31 23:12:27 - End time: 2022-05-31 23:19:30 - Diff (end t. - start t.): 423468
Epoch: 760 - Round: 913325 - Source shard: 0 - Destination shard: 0
2000 transactions executed successfully in 7 minutes 3 seconds (423468 ms).
They were sent from this computer at 2022-05-31 23:12:27.
They arrived to an Elrond devnet node at 2022-05-31 23:12:27.
This computer detected querying the Elrond API that they ended at 2022-05-31 23:19:30.

```

Figura 39: Resultat del *script* de Python executat per a 2000 transaccions amb un remitent que no és el propietari del *smart contract* però que es troba al mateix *shard*.

```

sergi@dragonfruit:~/Escritorio/TFM/TFM_stats$ python3 stats.py
Starting parallel 2022-05-31 23:32:41
Funding ABC-xEGLD liquidity pool with 0.05 xEGLD among 5 transactions
Please wait a few seconds
5 transactions were sent
-----
Hash: 612c8b7bccb51feff9c641f1fe50bf157f5bcf7bc4d7552111810def154f9b44a4 - Status: success
Elrond age: 2022-05-31 23:32:41 2022-05-31 23:33:12 - Start time: 2022-05-31 23:32:41 - End time: 2022-05-31 23:33:12 - Diff (end t. - start t.): 30281
Epoch: 760 - Round: 913532 - Source shard: 2 - Destination shard: 0
-----
Hash: d994d7aa8080f63b57269e0781136681289f521494542f8d1ec288aa3115b6e2 - Status: success
Elrond age: 2022-05-31 23:32:41 2022-05-31 23:33:12 - Start time: 2022-05-31 23:32:41 - End time: 2022-05-31 23:33:12 - Diff (end t. - start t.): 30481
Epoch: 760 - Round: 913532 - Source shard: 2 - Destination shard: 0
-----
Hash: 07b788905e8d3882d169934f83b1dddf155f24964233e836c2aeab58c07e136 - Status: success
Elrond age: 2022-05-31 23:32:41 2022-05-31 23:33:12 - Start time: 2022-05-31 23:32:41 - End time: 2022-05-31 23:33:12 - Diff (end t. - start t.): 30679
Epoch: 760 - Round: 913532 - Source shard: 2 - Destination shard: 0
-----
Hash: 2a3f2f48c072d0f473383cb7969710849930e0f8061b683471ac46601b5d020e - Status: success
Elrond age: 2022-05-31 23:32:41 2022-05-31 23:33:12 - Start time: 2022-05-31 23:32:41 - End time: 2022-05-31 23:33:12 - Diff (end t. - start t.): 30877
Epoch: 760 - Round: 913532 - Source shard: 2 - Destination shard: 0
-----
Hash: 0d07b2572590b7053bb01bdab280393c5fa53bbb7df33ddfe8cf29f41471d9 - Status: success
Elrond age: 2022-05-31 23:32:41 2022-05-31 23:33:12 - Start time: 2022-05-31 23:32:41 - End time: 2022-05-31 23:33:13 - Diff (end t. - start t.): 31094
Epoch: 760 - Round: 913532 - Source shard: 2 - Destination shard: 0
5 transactions executed successfully in 0 minutes 31 seconds (31094 ms).
They were sent from this computer at 2022-05-31 23:32:41.
They arrived to an Elrond devnet node at 2022-05-31 23:32:41.
This computer detected querying the Elrond API that they ended at 2022-05-31 23:33:13.

```

Figura 40: Resultat del *script* de Python executat per a 5 transaccions amb un remitent que no és el propietari del *smart contract* i que a més no es troba al mateix *shard*.

```

Hash: 1845821181b7fbf20f302265b078dc212091233ea8163c30593b23384c910 - Status: success
Elrond age: 2022-05-31 23:37:49 2022-05-31 23:38:18 - Start time: 2022-05-31 23:37:49 - End time: 2022-05-31 23:45:19 - Diff (end t. - start t.): 450032
Epoch: 760 - Round: 913583 - Source shard: 2 - Destination shard: 0
-----
Hash: 78fe10d5f0afbc61e3df73c011613d35535c31e03a8ad0070693fffe991252d2d - Status: success
Elrond age: 2022-05-31 23:37:49 2022-05-31 23:38:18 - Start time: 2022-05-31 23:37:49 - End time: 2022-05-31 23:45:19 - Diff (end t. - start t.): 450274
Epoch: 760 - Round: 913583 - Source shard: 2 - Destination shard: 0
-----
Hash: 012cb828dbac430b0e52a40c31b701b5636cd0b4b24efdf3266568beb7625f5 - Status: success
Elrond age: 2022-05-31 23:37:49 2022-05-31 23:38:18 - Start time: 2022-05-31 23:37:49 - End time: 2022-05-31 23:45:19 - Diff (end t. - start t.): 450482
Epoch: 760 - Round: 913583 - Source shard: 2 - Destination shard: 0
-----
Hash: db100068b7747240d21f713188bf64af3fc2988b20741baedaae742f9b6d7 - Status: success
Elrond age: 2022-05-31 23:37:49 2022-05-31 23:38:18 - Start time: 2022-05-31 23:37:49 - End time: 2022-05-31 23:45:20 - Diff (end t. - start t.): 450715
Epoch: 760 - Round: 913583 - Source shard: 2 - Destination shard: 0
-----
Hash: 5f9b11879ea0fdefb285592e8990b711b71e0072ba72f47a8c304d461d95766ddc - Status: success
Elrond age: 2022-05-31 23:37:49 2022-05-31 23:38:18 - Start time: 2022-05-31 23:37:49 - End time: 2022-05-31 23:45:20 - Diff (end t. - start t.): 450937
Epoch: 760 - Round: 913583 - Source shard: 2 - Destination shard: 0
-----
Hash: 150f44b20e934a2a87ae761ch37441f6de1923b93710baeda1319344f8901385 - Status: success
Elrond age: 2022-05-31 23:37:49 2022-05-31 23:38:18 - Start time: 2022-05-31 23:37:49 - End time: 2022-05-31 23:45:20 - Diff (end t. - start t.): 451136
Epoch: 760 - Round: 913583 - Source shard: 2 - Destination shard: 0
2880 transactions executed successfully in 7 minutes 31 seconds (451136 ms).
They were sent from this computer at 2022-05-31 23:37:49.
They arrived to an Elrond devnet node at 2022-05-31 23:37:49.
This computer detected querying the Elrond API, then they ended at 2022-05-31 23:45:20.

```

Figura 41: Resultat del *script* de Python executat per a 2000 transaccions amb un remitent que no és el propietari del *smart contract* i que a més no es troba al mateix *shard*.

## 4 Conclusions

Deu n’hi do per arribar fins aquí. Era allunyat de la programació de fa més de 10 anys hi ha estat un camí de pedres (en gran part perquè m’he tornat un cotxer rovellat). Tampoc tenia gaire idea de *blockchain* (no he cursat l’assignatura). En qualsevol cas, que hagi pogut finalitzar el present treball és bona mostra de què **l’entorn d’Elrond és força amigable per al programador** (punt important per a la seva adopció). Elrond ha avançat molt ràpid i, tot i que han desenvolupat molts de continguts per a aprendre la seva tecnologia, podrien estar millor (sobretot he trobat a faltar exemples amb codif font executable del que es comenta a [1]). Com que és un projecte novedós, no trobes gaire codi a Github, no hi ha cap curs a Udemy o altres plataformes i un llarg etcètera de punts on Ethereum guanya de llarg. Per contra, basta provar testDEX, veus que les transaccions van força més ràpides que a Ethereum.

Penso que el present treball revisa la tecnologia d’Elrond amb un mínim de rigor acadèmic, que s’ha realitzat un exemple pràctic (molt millorable per la meva inexperiència com a programador) i que s’ha provat mínimament el funcionament de la xarxa Elrond. Deu ser un dels primers, no només en català, sinó que també en altres idiomes. En la secció 1.2 no he citat el meu objectiu personal, que era bàsicament aprendre. Penso que això sí que s’ha assolit al 150%.

Com a crítica personal, la meva falta de coneixements i d’experiència en el desenvolupament de programari ha fet que em devii uns quants dies. Inicialment, en el diseny vaig plasmar que volia fer també un gràfic sobre l’evolució dels preus però faig haver de retallar el projecte un poc per falta de temps (he de dir que n’he dedicat més del que esperava). Sí que ara tinc clara una línia de formació que he de seguir per suprir les meves carències. El meu perfil és més de sistemes i de sistemes gestors de bases de dades, però m’està fascinant el món de les cadenes de blocs.

comentar tema menys nodes devnet comparat a testnet/producció

Como línies de treball futures vull formar-me en React i Rust i refer tot el codi (o tal vegada fer-ne un de nou relacionat amb NFT). La part d’estudi del rendiment és també només una pinzellada, i podria també formar-me per intentar realitzar auditories de sistemes de *blockchain*. En allò referent a potencials treballs de fi de grau o de màster, ara que ho he tocat, penso que la meva idea original de fer un bon material didàctic amb exemples per aprendre a desenvolupar en la tecnologia Elrond penso que seria força interessant (i molta gent ho agrairia).

La meva conclusió final, dins de la meva humil ignorància de nouvingut al camp, és que la xarxa Elrond està encara infravalorada.

•

“Este capítulo tiene que incluir:

- Una descripción de las conclusiones del trabajo: Qué lecciones se han aprendido del trabajo?.
- Una reflexión crítica sobre el logro de los objetivos planteados inicialmente: Hemos logrado todos los objetivos? Si la respuesta es negativa, por qué motivo?
- Un análisis crítico del seguimiento de la planificación y metodología a lo largo del producto: Se ha seguido la planificación? La metodología prevista ha sido la adecuada? Ha habido que introducir cambios para garantizar el éxito del trabajo? Por qué?
- Las líneas de trabajo futuro que no se han podido explorar en este trabajo y han quedado pendientes.”

## **5 Glossari**

“Definición de los términos y acrónimos más relevantes utilizados dentro de la Memoria.”

## 6 Bibliografia

### Referències

- [1] The Elrond Team, “Docs · the internet scale blockchain,” 2022. [Online]. Available: <https://docs.elrond.com/>
- [2] ——, “Elrond a highly scalable public blockchain via adaptive state sharding and secure proof of stake,” 2019. [Online]. Available: <https://elrond.com/assets/files/elrond-whitepaper.pdf>
- [3] cointelegraph.com, “What are decentralized exchanges, and how do dexs work?” [Online]. Available: <https://cointelegraph.com/defi-101/what-are-decentralized-exchanges-and-how-do-dexs-work>
- [4] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
- [5] W. Dai, “b-money,” 1998. [Online]. Available: <http://www.weidai.com/bmoney.txt>
- [6] H. Finney, “Rpow - reusable proofs of work,” 2005. [Online]. Available: <http://fennetic.net/irc/finney.org/~hal/rpow/>
- [7] N. Szabo, “Unenumerated: Bit gold,” 12 2008. [Online]. Available: <https://unenumerated.blogspot.com/2005/12/bit-gold.html>
- [8] V. Buterin, “Ethereum: A next-generation smart contract and decentralized application platform,” 2014. [Online]. Available: [https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum\\_White\\_Paper\\_-\\_Buterin\\_2014.pdf](https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_White_Paper_-_Buterin_2014.pdf)
- [9] A. Brownworth, “Blockchain 101 - a visual demo - youtube,” 11 2016. [Online]. Available: [https://www.youtube.com/watch?v=\\_160oMzbIY8](https://www.youtube.com/watch?v=_160oMzbIY8)
- [10] ——, “Blockchain 101 - part 2 - public / private keys and signing - youtube,” 12 2017. [Online]. Available: <https://www.youtube.com/watch?v=xIDLakeras>
- [11] S. Tual, “Ethereum launches — ethereum foundation blog,” 7 2015. [Online]. Available: <https://blog.ethereum.org/2015/07/30/ethereum-launches/>
- [12] D. Mechkaroska, V. Dimitrova, and A. Popovska-Mitrovikj, “Analysis of the possibilities for improvement of blockchain technology,” 2018 26th Telecommunications Forum,

*TELFOR 2018 - Proceedings*, 2018. [Online]. Available: [https://www.researchgate.net/publication/330585021\\_Analysis\\_of\\_the\\_Possibilities\\_for\\_Improvement\\_of\\_BlockChain\\_Technology](https://www.researchgate.net/publication/330585021_Analysis_of_the_Possibilities_for_Improvement_of_BlockChain_Technology)

- [13] L. Mincu, “The maiar defi wallet: A powerful web extension for internet-scale defi · elrond,” 9 2021. [Online]. Available: <https://elrond.com/blog/maiar-defi-wallet/>
- [14] B. Mincu, “(11) beniamin mincu on twitter: Everything we've built at elrond is prepared for the mainnet launch, pending security audits. spent the whole day with our team in an internal hacking sprint taking the network apart. everything else matters only if elrond is secure. heavy security audits ongoing. / twitter,” 11 2019. [Online]. Available: <https://twitter.com/beniaminmincu/status/1200835749412777984>
- [15] D. Nelson, “Elrond will pay you 60,000 usd to break its blockchain - coindesk,” 6 2020. [Online]. Available: <https://www.coindesk.com/tech/2020/06/10/elrond-will-pay-you-60000-to-break-its-blockchain/>
- [16] B. Mincu, “Battle of yields: 100,000 usd maiar exchange incentivized dex competition · elrond,” 9 2021. [Online]. Available: <https://elrond.com/blog/battle-of-yields-announcement/>
- [17] Baro Virtual - Coinmonks, “Ethereum and solana competitors,” 9 2021. [Online]. Available: <https://medium.com/coinmonks/ethereum-and-solana-%D1%81ompetitors-2638afd96ef2>
- [18] Cryptopedia Staff, “What is an automated market maker (amm)?” 3 2021. [Online]. Available: <https://www.gemini.com/cryptopedia/amm-what-are-automated-market-makers>
- [19] The Elrond Team, “Technology built for internet scale · elrond,” 2022. [Online]. Available: <https://elrond.com/technology/>
- [20] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the weil pairing,” *Journal of Cryptology* 2004 17:4, vol. 17, pp. 297–319, 7 2004. [Online]. Available: <https://link.springer.com/article/10.1007/s00145-004-0314-9>
- [21] J. Ćwirko, “Elven tools - tips on buying nfts on the elrond blockchain.” [Online]. Available: <https://www.elven.tools/docs/tips-on-buying-nfts-on-the-elrond-blockchain.html>
- [22] MongoDB, “Sharding — mongodb manual,” 2021. [Online]. Available: <https://docs.mongodb.com/manual/sharding/>
- [23] M. Foundation, “Webassembly — mdn,” 2 2021. [Online]. Available: <https://developer.mozilla.org/en-US/docs/WebAssembly>

- [24] B. Mincu, “The elrond virtual machine - smart contracts at internet scale,” 12 2020. [Online]. Available: <https://elrond.com/blog/elrond-virtual-machine-arwen-wasm-vm/>
- [25] ———, “After exceeding 10k tps in testnet, elrond is going open source,” 6 2019. [Online]. Available: <https://medium.com/elrondnetwork/after-exceeding-10k-tps-in-testnet-elrond-is-going-open-source-61dd25e93fd9>
- [26] J. Penaflor, “How do i setup my elrond validator?” 12 2019. [Online]. Available: <https://medium.com/@jeff.penaflor1983/how-do-i-setup-my-elrond-validator-52a35aec43f2>
- [27] Everstake, “Bringing defi to the next billion with maiar exchange,” 9 2021. [Online]. Available: <https://medium.com/everstake/bringing-defi-to-the-next-billion-with-maiar-exchange-869101bf555a>
- [28] The Elrond Team, “Welcome to maiar dex — maiar exchange docs,” 2021. [Online]. Available: <https://docs.maiar.exchange/>
- [29] J. L. de la Rosa Esteva, *Finances descentralitzades (DeFi)*, V. G. Font, Ed. Barcelona: FUOC, 2021.
- [30] V. Buterin, “Improving front running resistance of  $x^*y=k$  market makers,” 03 2014. [Online]. Available: <https://ethresear.ch/t/improving-front-running-resistance-of-x-y-k-market-makers/1281>
- [31] Elrond Network, “#1. elrond ide presentation & tutorial,” 10 2020. [Online]. Available: <https://youtu.be/bXbBfJCRVqE>
- [32] H. Ton, “How to setup your development environment to create smart contracts with rust on elrond blockchain network,” 11 2021. [Online]. Available: <https://tinyurl.com/4s5yzmrs>

## **7 Annexos**

“Listado de apartados que son demasiado extensos para incluir dentro de la memoria y tienen un carácter autocontenido (por ejemplo, manuales de usuario, manuales de instalación, etc.)

Dependiente del tipo de trabajo, es posible que no haya que añadir ningún anexo.”