



UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI
ELECTRONICĂ
DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA
INFORMAȚIEI



PROIECT DE DIPLOMĂ

Alin Marian Stanciu

COORDONATOR ȘTIINȚIFIC

Prof. univ. dr. ing. Costin Bădică

Septembrie 2021

CRAIOVA



UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE AUTOMATICĂ, CALCULATOARE ȘI
ELECTRONICĂ
DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA
INFORMAȚIEI



Platformă de e-business bazată pe tehnologia Blockchain

Alin Marian Stanciu

COORDONATOR ȘTIINȚIFIC

Prof. dr. ing. Costin Bădică

Septembrie 2021

CRAIOVA

„Învățătura este o comoară care își urmează stăpânul pretutindeni.”

Proverb popular

DECLARAȚIE DE ORIGINALITATE

Subsemnatul ALIN MARIAN STANCIU, student la specializarea CALCULATOARE din cadrul Facultății de Automatică, Calculatoare și Electronică a Universității din Craiova, certific prin prezenta că am luat la cunoștință de cele prezentate mai jos și că îmi asum, în acest context, originalitatea proiectului meu de licență:

- cu titlul Platformă de e-business bazată pe tehnologia Blockchain,
- coordonată de Prof. univ. dr. ing. Costin Bădică,
- prezentată în sesiunea Septembrie 2021.

La elaborarea proiectului de licență, se consideră plagiat una dintre următoarele acțiuni:

- reproducerea exactă a cuvintelor unui alt autor, dintr-o altă lucrare, în limba română sau prin traducere dintr-o altă limbă, dacă se omit ghilimele și referința precisă,
- redarea cu alte cuvinte, reformularea prin cuvinte proprii sau rezumarea ideilor din alte lucrări, dacă nu se indică sursa bibliografică,
- prezentarea unor date experimentale obținute sau a unor aplicații realizate de alți autori fără menționarea corectă a acestor surse,
- însușirea totală sau parțială a unei lucrări în care regulile de mai sus sunt respectate, dar care are alt autor.

Pentru evitarea acestor situații neplăcute se recomandă:

- plasarea între ghilimele a citatelor directe și indicarea referinței într-o listă corespunzătoare la sfârșitul lucrării,
- indicarea în text a reformulării unei idei, opinii sau teorii și corespunzător în lista de referințe a sursei originale de la care s-a făcut preluarea,
- precizarea sursei de la care s-au preluat date experimentale, descrieri tehnice, figuri, imagini, statistici, tabele et caetera,
- precizarea referințelor poate fi omisă dacă se folosesc informații sau teorii arhicunoscute, a căror paternitate este unanim cunoscută și acceptată.

Data,

Semnătura candidatului,

14.09.2021



UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare și Electronică
Departamentul de Calculatoare și Tehnologia Informației

Aprobat la data de
.....
Șef de departament,
Prof. dr. ing.
Marius BREZOVAN

PROIECTUL DE DIPLOMĂ

Numele și prenumele studentului/-ei:	Stanciu Alin Marian
Enunțul temei:	Platformă de e-business bazată pe tehnologia Blockchain
Datele de pornire:	Se dorește implementarea unei soluții care vine în ajutorul consumatorului de miere ecologică. Aplicația asigură transparența și încrederea pentru o cantitate de miere cu ajutorul tehnologiei blockchain și noțiunii de smart contract care face posibilă comunicarea între diferiți membri din lanțul de aprovizionare. Testarea soluției constă într-o demonstrație în timp real pentru cel puțin 2 membri.
Conținutul proiectului:	Capitolul 1: Introducere Capitolul 2: Analiza domeniului Capitolul 3: Proiectarea sistemului Capitolul 4: Implementare Capitolul 5: Testare și experimente
Material grafic obligatoriu:	Diagrame care descriu sistemul implementat Capturi de ecran din aplicație Prezentarea Power Point
Consultații:	Periodice
Conducătorul științific (titlul, nume și prenume, semnătura):	Prof. dr. ing. Costin Bădică 
Data eliberării temei:	07.10.2020
Termenul estimat de predare a proiectului:	06.09.2021

Data predării proiectului de către student și semnătura acestuia:	06.09.2021



UNIVERSITATEA DIN CRAIOVA
Facultatea de Automatică, Calculatoare și Electronică

Departamentul de Calculatoare și Tehnologia Informației

REFERATUL CONDUCĂTORULUI ȘTIINȚIFIC

Numele și prenumele candidatului/-ei: Stanciu Alin Marian
Specializarea: Calculatoare
Titlul proiectului: Platformă de e-business bazată pe tehnologia Blockchain
În facultate ☒
În producție ☐
În cercetare ☐
Altă locație:

Locația în care s-a realizat practica de documentare (se bifează una sau mai multe din opțiunile din dreapta):

În urma analizei lucrării candidatului au fost constatate următoarele:

Nivelul documentării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine X
Tipul proiectului		Cercetare X	Proiectare X	Realizare practică X	Altul
Aparatul matematic utilizat		Simplu <input type="checkbox"/>	Mediu X	Complex <input type="checkbox"/>	Absent <input type="checkbox"/>
Utilitate		Contract de cercetare <input type="checkbox"/>	Cercetare internă X	Utilare <input type="checkbox"/>	Altul
Redactarea lucrării		Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine X
Partea grafică, desene		Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună X
Realizarea practică	Contribuția autorului	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Mare <input type="checkbox"/>	Foarte mare X
	Complexitatea temei	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă X
	Analiza cerințelor	Insuficient <input type="checkbox"/>	Satisfăcător <input type="checkbox"/>	Bine <input type="checkbox"/>	Foarte bine X
	Arhitectura	Simplă <input type="checkbox"/>	Medie <input type="checkbox"/>	Mare <input type="checkbox"/>	Complexă X
	Întocmirea specificațiilor	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună X

	funcționale				
	Implementarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună X
	Testarea	Insuficientă <input type="checkbox"/>	Satisfăcătoare <input type="checkbox"/>	Bună <input type="checkbox"/>	Foarte bună X
	Funcționarea	Da X	Parțială <input type="checkbox"/>	Nu <input type="checkbox"/>	
Rezultate experimentale		Experiment propriu X		Preluare din bibliografie <input type="checkbox"/>	
Bibliografie		Cărți X	Reviste X	Articole X	Referințe web X
Comentarii și observații					

În concluzie, se propune:

ADMITEREA PROIECTULUI X	RESPINGEREA PROIECTULUI <input type="checkbox"/>
----------------------------	---

Data,

01.09.2021

Semnătura conducătorului științific,



REZUMATUL PROIECTULUI

Într-un viitor apropiat în care majoritatea activităților clasice vor fi înlocuite de roboți, mașini, utilaje, pe scurt vor fi automatizate, va fi nevoie de un sistem prin care să garantăm calitatea produselor. Avansând tehnologia pentru domeniul alimentar, va fi posibilă crearea produselor de consum zilnic într-un laborator, printr-un minim de efort și într-un timp foarte scurt. Într-o lume a vitezei, aceste activități care acum par normale, naturale și suficiente, vor deveni greu de realizat, procesat și populația va fi mult mai greu de satisfăcut.

O problemă importantă este calitatea produsului vândut într-un magazin. Consumatorul dorește să cumpere un produs calitativ, realizat prin procese și componente naturale, fără aditivi artificiali.

Producătorul are obligația de a expune pe eticheta produsului toate informațiile relevante pentru consumator, pentru ca acesta să poată lua o decizie în privința calității, pentru a evita posibile alimente la care corpul său prezintă reacții adverse și pentru a contura un raport calitate-preț.

Problema apare în momentul în care pe eticheta produsului există inconsistențe, datele despre anumite ingrediente folosite în producție nefiind specificate. Aceste ingrediente necunoscute pot fi omise din greșeli, dar sunt temporare și se remediază rapid sau apar implicit în procesele de producție și nu sunt menționate pe produsul final. A doua categorie de ”greșeală” este realizată pentru că legislația în vigoare interzice ca astfel de ingrediente să ajungă în hrana de zi cu zi a populației.

Există și cazuri când unui produs îi scade rata de vânzare pentru că are în compoziție un ingredient ”periculos și dăunător”, consumatorul evitând să cumpere astfel de produse.

Aceste scenarii ar trebui să nu existe într-o lume înconjurată de tehnologie, cu o mare parte din activități automatizate, cu inteligență artificială și acces la internet 24/7.

O primă soluție ar putea fi interzicerea utilizării unor ingrediente selectate și grupate într-o ”listă neagră” și accentuarea controlului calității produselor alimentare pe întreg procesul de producție.

Evident, printre aceste acțiuni de control pot interveni și alte interese de natură diferită de scopul inițial și astfel credibilitatea produsului respectiv scade pentru că în faza finală când produsul este în magazin, informația de pe etichetă poate fi coruptă.

Pentru această problemă propun spre implementare o soluție simplă, bazată pe tehnologia Blockchain, prin care consumatorului să îi fie transparent tot lanțul de aprovizionare (origine, transport, producție, ambalare, depozitare, vânzare).

Proiectul este reprezentat printr-o aplicație descentralizată, realizată pentru rețeaua Ethereum, utilizând suita Truffle.

Logica de business a aplicației este susținută de utilizarea unor smart contracts. Fiecare modelează relația între doi membri ai lanțului de aprovizionare și mediază transferul de informație dintre aceștia.

Pentru fiecare membru al lanțului de aprovizionare, există un smart contract care înregistrează pe rețeaua Ethereum tranzacții între diverși membri, citește date deja scrise sau emite evenimente declanșate în puncte cheie ale rulării aplicației.

Smart contract-urile sunt compilate, apoi se execută operația de deploy pe o rețea privată, locală, de blockchain, simulată cu ajutorul instrumentului Ganache și apoi sunt utilizate de front-end-ul aplicației destinate utilizatorului.

Termenii cheie: blockchain, ethereum, dapps, truffle suite, solidity, smart contract, lanț de aprovizionare, tranzacții, evenimente, rețea distribuită, visual studio code, remix IDE

MULȚUMIRI

În prezentul paragraf, doresc să adresez sincere mulțumiri către domul profesor coordonator Prof. univ. dr. ing. Costin Bădică pentru tot sprijinul oferit pe întreaga perioadă a dezvoltării proiectului.

Pe această cale mulțumesc familiei pentru suportul și încurajările făcute în timpul redactării proiectului.

De asemenea, mulțumiri adresez domnilor profesori din cei 4 ani de studii, pentru ceea ce m-au învățat teoretic și practic și pentru momentele frumoase din timpul orelor de curs, laborator și seminar.

PROLOG

CUPRINSUL

1	INTRODUCERE	1
1.1	SCOPUL.....	1
1.2	MOTIVAȚIA.....	1
2	ANALIZA DOMENIULUI	2
2.1	MODELE DE BUSINESS	2
2.2	EXEMPLE DE PROIECTE DE TIP LANȚ DE APROVIZIONARE DEZVOLTATE CU TEHNOLOGIA BLOCKCHAIN	5
2.3	DE CE BLOCKCHAIN-UL ETHEREUM PENTRU IMPLEMENTAREA PROIECTULUI	6
2.3.1	<i>EOS vs Ethereum</i>	8
2.3.2	<i>Ethereum vs Hyperledger Fabric</i>	9
2.4	PLATFORMA ETHEREUM	12
2.4.1	<i>Criptografie prin curbă eliptică</i>	16
2.4.2	<i>Algoritmi de consens</i>	20
2.4.3	<i>Smart contract</i>	22
2.4.4	<i>Solidity</i>	23
2.4.5	<i>Toolchain</i>	25
3	PROIECTAREA SISTEMULUI	27
3.1	CAZURI DE UTILIZARE/CERINȚE.....	27
3.2	SCHEMA BLOC A ÎNTREGULUI SISTEM	28
3.3	COMPONENTELE ARHITECTURALE	35
3.4	IMPLEMENTARE	39
3.4.1	<i>Structura unui proiect Truffle generat</i>	39
3.4.2	<i>Structura proiectului BioTrack</i>	40
3.4.3	<i>Schema de mapare design-contract inteligent</i>	42
3.5	TESTARE ȘI EXPERIMENTE	51
3.5.1	<i>Testare direct în Remix IDE online</i>	51
3.5.2	<i>Testare în consola Truffle</i>	55
4	TERMENI DE UTILIZARE	59
4.1	AUTORII	59
4.2	LICENȚA DE UTILIZARE	59
5	CONCLUZII	60
6	BIBLIOGRAFIE	61

REFERINȚE WEB	62
A. CODUL SURSĂ.....	64
B. CD/ DVD	65
INDEX	66

LISTA FIGURILOR

FIGURĂ 1: TRANZACȚIE ÎNTRE 2 PERSOANE [YES01]	2
FIGURĂ 2: CONCEPTELE CHEIE ALE BLOCKCHAIN-ULUI APLICATE ÎN BUSINESS [GUP17]	5
FIGURĂ 3: FUNȚIA DE TRANZIȚIE ÎNTRE DOUĂ STĂRI [BUT13]	13
FIGURĂ 4: MODELUL DE EXECUȚIE AL MAȘINII VIRTUALE CU POSIBILITĂȚILE DE EXCEPȚII	15
FIGURĂ 5: MERKLE PATRICIA TRIE - PREZENTARE SCHEMATICĂ.....	17
FIGURĂ 6: GRAFICUL UNEI CURBE ELIPTICE [NIK22].....	18
FIGURĂ 7: ASPECT REMIX-IDE.....	24
FIGURĂ 8: DIAGRAMA DE PROCES PENTRU LANȚUL DE APROVIZIONARE	29
FIGURĂ 9: INTERACȚIUNEA STUPINĂ-CENTRU DE COLECTARE-LABORATOR-PROCESATOR.....	31
FIGURĂ 10: CELE 2 POSIBILITĂȚI DISTINCTE DE CONTINUARE DE LA PROCESATOR	32
FIGURĂ 11: POSIBILITĂȚILE DE TRANSPORT	33
FIGURĂ 12: ACȚIUNILE UNUI CONSUMMATOR.....	34
FIGURĂ 13: ARHITECTURA DE SISTEM	35
FIGURĂ 14: DIAGRAMA CAZURILOR DE UTILIZARE.....	38
FIGURĂ 15: EXEMPLU DE EXECUȚIE PENTRU CREAREA UNUI PROIECT TRUFFLE.....	39
FIGURĂ 16: STRUCTURA PROIECTULUI TRUFFLE CREAT ÎNȚIAL	40
FIGURĂ 17: STRUCTURA PROIECTULUI BioTrack.....	40
FIGURĂ 18: CODUL JAVASCRIPT PENTRU 2_CONTRACTS_MIGRATION.JS	41
FIGURĂ 19: O SECVENȚĂ DIN FIȘIERUL DE CONFIGURARE CONFIG-TRUFFLE.JS	42
FIGURĂ 20: DIAGRAMA DE PROCESS PENTRU LANȚUL DE APROVIZIONARE SPECIALIZAT PE MIERE ECOLOGICĂ (ÎNCEPUT).....	42
FIGURĂ 21: CODUL SOLIDITY OBTINUT DUPĂ TRANSPUNEREA DESIGN-ULUI PENTRU TOWNHALL.....	43
FIGURĂ 22: MODIFICATORUL DE ACCES, CONSTRUCTORUL ȘI O FUNCȚIE DIN CONTRACTUL INTELIGENT TOWNHALL	44
FIGURĂ 23: EXEMPLU DE FUNCȚIE EXTERNĂ.....	45
FIGURĂ 24: INTERFAȚA PENTRU CONTRACTUL TOWNHALL	45
FIGURĂ 25: DIAGRAMA DE PROCESS PENTRU LANȚUL DE APROVIZIONARE SPECIALIZAT PE MIERE ECOLOGICĂ, CONTINUARE	47
FIGURĂ 26: DIAGRAMA DE PROCESS PENTRU LANȚUL DE APROVIZIONARE SPECIALIZAT PE MIERE ECOLOGICĂ, CONTINUARE	49
FIGURĂ 27: DIAGRAMA DE PROCESS PENTRU LANȚUL DE APROVIZIONARE SPECIALIZAT PE MIERE ECOLOGICĂ, CONTINUARE	50
FIGURĂ 28: COMPILAREA CONTRACTULUI INTELIGENT APIARY	51
FIGURĂ 29: O PARTE DIN ACȚIUNILE CONTRACTULUI APIARY	52
FIGURĂ 30: IMPLEMENTAREA CONTRACTULUI APIARY PE BLOCKCHAIN-UL ETHEREUM.....	52
FIGURĂ 31: EWM FACE REVERT TRANZACȚIEI CURENTE	53
FIGURĂ 32: MESAJUL PRIMIT CĂ TRANZACȚIA S-A REALIZAT CU SUCCESS DUPĂ ADĂUGAREA UNUI NOU TIP	53
FIGURĂ 33: RE-EXECUȚIA TRANZACȚIEI DUPĂ ADĂUGAREA TIPULUI DE MIERE ÎN STUPINĂ.....	54
FIGURĂ 34: ÎNREGISTRARE TOWNHALL, SOLICITARE CERTIFICAT DE PRODUCĂTOR ȘI VERIFICARE EXISTENȚĂ	54
FIGURĂ 35: UN APICULTOR ÎNREGISTRAT ÎN CADRUL PRIMĂRIEI.....	55

FIGURĂ 36: INSTRUMENTUL GANACHE	55
FIGURĂ 37: CONTRACTELE INTELIGENTE AU COMPILAT CU SUCCES.....	56
FIGURĂ 38: MIGRARE EXECUTATĂ CU SUCCESS (OBSERVAT ȘI ÎN GANACHE)	56
FIGURĂ 39: CREAREA INSTANTELOR APIARY ȘI TOWNHALL ÎN JAVASCRIPT.....	57
FIGURĂ 40: EROARE RAPORTATĂ DE MAȘINA VIRTUALĂ.....	57
FIGURĂ 41: COMENZILE NECESARE PENTRU A PUTEA PUNE LA VÂNZARE CANTITATEA DE MIERE	57
FIGURĂ 42: CÂTEVA INTEROGĂRI REALIZATE CU JAVASCRIPT	58

LISTA TABELELOR

TABEL 1: ETHEREUM VS EOS.....	9
TABEL 2 ETHEREUM VS HYPERLEDGER FABRIC.....	10
TABEL 3 EXEMPLU DE TRANSFER CU 7 ETHER	23
TABEL 4 TRANZACȚIA PENTRU CONSTRUCȚIA UNUI CONTRACT INTELIGENT	23

1 INTRODUCERE

1.1 Scopul

Problematica abordată în prezenta lucrare reprezintă posibila lipsă de încredere a unui client când achiziționează un produs dintr-o piață. Clientul are nevoie de o garanție validă a calității produsului, a originii acestuia precum și prelucrarea și condițiile de păstrare de-a lungul întregului lanț de aprovizionare, care trebuie să fie la cele mai înalte standarde.

Scopul aplicației practice este de a face realizabilă această necesitate a omului de a achiziționa produse ecologice, de calitate, având toată încrederea necesară datorită sistemului implementat cu tehnologia *blockchain*, caracterizat fiind de transparență, imutabilitate, anonimitate și securitate.

1.2 Motivația

Tehnologia blockchain nu are o istorie foarte mare, aceasta fiind inventată de către Satoshi Nakamoto în 2008 pentru a veni în ajutorul criptomonedei Bitcoin. Această tehnologie prezintă interes pentru mine și doresc să aprofundez acest domeniu de studiu. De asemenea, să înțeleg limbajul Solidity, cum funcționează un smart contract, ce reprezintă și ce particularități are rețeaua Ethereum. Aplicația pentru un lanț de aprovizionare utilizând blockchain-ul este o oportunitate importantă pentru studiul, aprofundarea și exercițiul meu cu tehnologia blockchain, rețeaua Ethereum și instrumentele auxiliare care simplifică dezvoltarea de aplicații descentralizate.

2 ANALIZA DOMENIULUI

2.1 Modele de business

De când oamenii au început să schimbe bunuri, de exemplu o unealtă de săpat cu o unealtă de vânat, chiar dacă nu era un echilibru creat, asistăm de fapt la un proces de tranzacționare. Pentru a umple dezechilibrul creat în schimbul unor obiecte de valori diferite, au fost introduși banii, ca metodă de plată. [GUP17] Odată cu evoluției și inovații în domenii precum telecomunicații, electronică, calculatoare și internet, s-au dezvoltat și noi metode de a tranzacționa.



Figură 1: Tranzacție între 2 persoane [YES01]

Dar problemele nu erau terminate, tranzacționarea fiind în continuare inefficientă și scumpă datorită unor limitări precum aria redusă pentru utilizarea banilor cash, timp foarte mare pentru efectuarea tranzacției, nevoia de o parte terță care să valideze, posibilitate ridicată pentru fraude, atacuri sau greseli care pot afecta afacerea și poate cea mai importantă limitare este lipsa accesului la un cont bancar sau la un sistem de plăți. A apărut astfel nevoia pentru un sistem care să stabilească încrederea, fără echipament suplimentar, fără taxe și toate să fie înregistrate într-un registru comun.

Tehnologia blockchain reprezintă o soluție pentru un registru distribuit cu numeroase avantaje în procesul de înregistrare al unor tranzacții dar și pentru monitorizarea bunurilor în cadrul unor business-uri [GUP17]. Trebuie să înțelegem că aceste bunuri sunt cuprinse în două categorii foarte importante. Prima, a bunurilor *tangibile* precum o casă, o mașină, bani cash, pământ și cea de-a doua, bunuri *intangibile*, având ca exemple proprietatea intelectuală, patente, drepturi de autor. Cu ajutorul acestei tehnologii, putem înregistra orice entitate care necesită urmărire/trasabilitate într-o afacere, totul cu un risc minimal și costuri reduse.

Unul dintre cele mai comune cazuri de utilizare pentru tehnologia blockchain este reprezentat de criptomonede, cum ar fi Bitcoin, în finanțe descentralizate dar și foarte multe alte criptomonede ca

alternativă pentru Bitcoin. Criptomonedele cu o atracție foarte mare pentru investitori au fost create cu diverse scopuri, având susținerea unor proiecte care se dezvoltă rapid și acest fapt oferă încredere cumpărătorilor. Prezint mai departe câteva referințe teoretice despre ce înseamnă criptomonedele și câteva date despre cele mai influente din piață la momentul scrierii lucrării, august 2021.

Criptomoneda este o valută digitală sau virtuală care este securizată prin metode de criptografie care o fac aproape imposibil de falsificat și în același timp este evitată dubla cheltuire, marea majoritate a criptomonedelor din piață fiind bazate pe blockchain. [CRY21]

La momentul redactării, august 2021, Bitcoin are în circulație mai mult de 18.8 milioane de Bitcoin cu o capitalizare de piață în jurul a 858.9 miliarde de dolari. Tehnica prin care este prevenită inflația (rata de creștere a prețurilor de-a lungul unei perioade de timp [CEY01]) și manipularea este numărul limitat de criptomonede Bitcoin, și anume 21 de milioane. Menținerea ratei de inflație la valori normale este foarte importantă pentru o creștere a valorii criptomonedei de-a lungul timpului.

În prezent, Bitcoin reprezintă un bun a cărui valoare crește odată cu reducerea numărului de monede disponibile pentru mină. Această monedă nu mai este folosită cu scopul direct de a fi tranzacționată, ci de a fi păstrată de-a lungul timpului pentru a crește valoarea. Astfel putem face o asemănare între ceea ce înseamnă aurul pentru omenire și ceea ce începe Bitcoin să devină. Se observă o creștere a companiilor mari din lume care încep să investească în această criptomonedă.

Proiectul pentru a rezolva problema mierii ecologice este dezvoltat mai departe cu ajutorul platformei Ethereum, cu instrumente și framework-uri special create pentru a facilita implementarea unor soluții tehnice.

Oamenii au realizat în scurt timp că blockchain-ul pentru Bitcoin a fost creat să funcționeze în spațiul criptomonedelor și aveau nevoie de ceva separat pentru a satisface nevoile existente într-o afacere. Trei caracteristici importante ale unui blockchain diferit de cel pentru Bitcoin sunt: [IBM17]

- *Bunuri vs criptomonedă*. Implementarea blockchain-ului ține cont de bunuri (tangibile sau intangibile) sub altă formă decât cea de token. Un exemplu este Everledger, un proiect care presupune utilizarea blockchain-ului pentru a urmări proveniența bunurilor de lux prin minimizarea numărului documentelor necesare, reducerea fraudei și evitarea supraevaluării sau plății multiple. Mai multe detalii despre acest caz de utilizare pot fi consultate la adresa <https://everledger.io/>.
- *Identitate vs anonimitate*. O particularitate importantă a unei rețele blockchain este anonimitatea, oricine având dreptul să vadă tranzacții, fiind uneori o problemă în timpul proiectării unei aplicații deoarece în contextul afacerilor este foarte importantă intimitatea

(privacy). De asemenea, o altă problemă pe care a trebuit să o gestionez la proiectarea lanțului de aprovizionare a fost să știu în orice moment de timp, pentru un membru, de la cine am primit bunuri sau alte intrări necesare desfășurării activității și către cine trimit bunurile mai departe sau informații suplimentare. În orice moment trebuia să știu de la cine primesc și către cine trimit, ca membru, dincolo de o simplă adresă (hash) asignată în cadrul blockchain-ului.

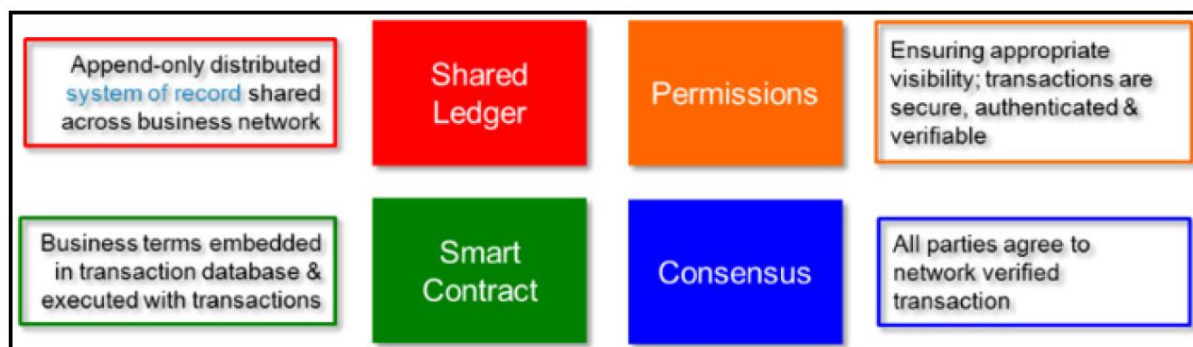
- *Restricționare vs proof-of-work.* Gestionarea unui lanț de aprovizionare presupune activități desfășurate exclusiv de un membru. Astfel, apare nevoia de a restricționa accesul altor membri la acțiunile unui membru. De exemplu, în cazul primăriei, văzută ca un membru separat, doar contul care a creat instanța smart contract-ului are dreptul de a modifica date interne din primărie. O altă metodă de verificare (în interiorul unei funcții) este utilizarea clauzei *require*(condiție), care face să eșueze tranzacția în cazul în care nu a fost îndeplinită condiția dintre paranteze.

Integrând tehnologia blockchain în soluția noastră, sunt evitate semnele de întrebare adresate către diferiți membri din lanțul de aprovizionare la un moment de timp și de aceea este foarte prețios, fiecare tranzacție bazându-se pe o altă tranzacție, toate entitățile participante fiind conștiente de orice interacțiune. Blockchain-ul pune la dispoziție aceste facilități folosindu-se de următoarele atribute cheie[GUP17]:

1. *Distribuit și durabil.* Registrul este distribuit, este actualizat în timp real după fiecare tranzacție către toate nodurile existente în rețea, fără dependențe față de unul sau mai multe entități ale lanțului de aprovizionare.
2. *Sigur și privat.* Siguranța că participanții sunt cu adevărat cei declarați și prevenirea accesului neautorizat sunt asigurate de sistemele/tehnicele de permisiuni și criptografie avansate. Permisunile reprezintă un mecanism foarte important în modelarea unui sistem multi-actor, pentru a evita probleme cauzate din neatenție, ajutate de algoritmi avansați de criptografie pentru a reduce la minimum posibilitatea fraudei și coruperii datelor înregistrate.
3. *Transparent.* Participanții în procesul de aprovizionare pot valida tranzacții și verifica identități sau drepturi de proprietate fără necesitatea unor terți care să intervină, totul datorită accesului la același registru cu datele înregistrate.
4. *Consens.* Pentru ca o tranzacție să fie validă, totii membri relevanți din rețea trebuie să fie de acord. Acest mecanism de validare este posibil prin utilizarea unor algoritmi de consens care stabilesc în ce condiții și situații o tranzacție este validă, executabilă.

5. *Flexibilitate*. Un smart contract conține una sau mai multe condiții care fac posibilă execuția pe blockchain a unei tranzacții. Capacitatea de extindere a unui smart contract face ușoară adaptarea regulilor business-ului oricărui proces în cadrul rețelei.

Pentru a înțelege mai bine avantajele unei implementări ale tehnologiei blockchain în business, sunt prezentate 4 concepte care stau la baza unei astfel de arhitecturi [GUP17].



Figură 2: Conceptele cheie ale blockchain-ului aplicate în business [GUP17]

1. *Registru distribuit*. Înregistrează toate tranzacțiile din rețea în registru și le face imutabile, toți participanții rețelei având drepturi de acces către datele scrise. Astfel este evitată informația duplicată cum apare în unele cazuri când se lucrează cu registrul clasic.
2. *Permiuni*. În ceea ce privește permiunile într-o rețea distribuită, există rețele permissive și non-permissive. Într-un blockchain permisiv, fiecare participant cu identitate unică are acces către detaliile unei tranzacții. Un blockchain non-permisiv are abilitatea de a restricționa accesul la detaliile unei tranzacții stocate intern, și participanții pot specifica ce informații își doresc să permită altor participanți să citească.
3. *Consens*. Într-o rețea în care participanții sunt foarte bine cunoscuți și de încredere, tranzacțiile pot fi verificate și înscrise în registru printr-o metodă/acord numită consens/acord.
4. *Smart contract*. Un smart contract este un set de reguli care guvernează logica unei tranzacții fiind stocat pe blockchain și executat automat ca parte a procesului de tranzacționare.

2.2 Exemple de proiecte de tip lanț de aprovizionare dezvoltate cu tehnologia blockchain

Există numeroase exemple de proiecte care încep să dezvolte o astfel de soluție, cum am menționat mai sus, Everledger, IBM FoodTrust, VeChain, Cargill blockchain, Honeysuckle White, lanț de aprovizionare și transport pentru ouă, proiect inițiat de compania Midwestern USA.

Blockchain-ul are capacitatea de a schimba viziunea despre afaceri, prin nivelul ridicat de încredere oferit (din proiectarea acestuia, orice înregistrare în blockchain nu mai poate fi ștersă sau modificată și pentru orice bun înscris în registrul distribuit, se cunoaște foarte bine locația/adresa, calculată la un moment de timp anterior, printr-o tranzacție).

Un exemplu foarte bun de proiect pus în practică este *IBM Food Trust* [IBM21], o rețea blockchain caracterizată de integritate, eficiență și încredere, atribute mult dorite într-un sistemul alimentar. O astfel de industrie se poate confrunta cu diverse provocări, cu transparența transportului între fiecare entitate, cu reducerea tensiunilor între lanțurile de aprovizionare globale. Printre segmentele de piață unde este aplicată soluția, se numără produsele proaspete (fructe și legume), fructe de mare, producție, logistică pentru alimente precum și restaurante care garantează mâncare sănătoasă clienților. Valoarea adăugată business-ului constă în eficiența lanțului de aprovizionare, încrederea în companie, siguranță și prospețime pentru alimente, evitarea fraudei alimentare precum și risipa acestora, și nu în ultimul rând, sustenabilitatea. Sistemul este proiectat să lucreze la cele mai înalte standarde și furnizează un ecosistem care permite conexiuni între multiple niveluri (gestionarea unor date din sisteme existente și utilizarea standardelor GS1 și Produce Traceability Initiative 128-PTI).

Soluția IBM Food Trust este construită pe blockchain-ul Hyperledger Fabric care permite membrilor și terților să creeze interogări private și să comunice cu platforma, toate acestea acoperind o mare parte din nevoile de business dintr-un lanț de aprovizionare.

Un alt exemplu este platforma *open-source* de la *Cargill* numită *Splinter* care este proiectată să permită oricărei organizații implicate într-un lanț de aprovizionare să interschimbe bunuri agro-alimentare de la un punct la un alt punct cu logistica și interacțiunile rezolvate de către blockchain. [JES15]

Chiar dacă utilizarea blockchain-ului pentru gestionarea unor lanțuri de aprovizionare este doar la început, există foarte multe cazuri de utilizare pentru acesta. În cadrul unui lanț de aprovizionare se pot utiliza senzori, inteligență artificială, subsisteme care să producă date, să le manipuleze și apoi să le înregistreze într-un registru distribuit.

2.3 De ce blockchain-ul Ethereum pentru implementarea proiectului

Decizia de a implementa un proiect cu o anumită tehnologie este foarte importantă deoarece trebuie mult studiate toate celelalte posibilități, analizate avantajele și dezavantajele în raport cu competiția și stabilit dacă ceea ce este ales cumulează și satisface un minim de proprietăți fundamentale pe care business-ul le solicită.

Pentru dezvoltarea acestei aplicații practice am început prin a studia ce este blockchain-ul, cum este reprezentat, cum lucrează și ce beneficii aduce pentru implementarea mea, o astfel de soluție software.

După, am continuat cu studiul pe diferite blockchain-uri, precum Bitcoin, Hyperledger Fabric și mai apoi am îndreptat atenția către Ethereum care s-a dovedit soluția cea mai convenabilă.

A fost o decizie greu de luat pentru că fiecare blockchain are particularitățile sale, de exemplu, la Hyperledger Fabric există noțiunea de autoritate, certificat dar la Ethereum acesta funcționalitate trebuie modelată cu ajutorul unui smart contract. Prezint mai departe câteva aspecte importante pentru fiecare blockchain în parte, scurtă descriere, avantaje, dezavantaje, și de ce am renunțat sau de ce am ales o platformă anume.

Potrivit unui studiu realizat de Gartner [AVI13], 2021 este anul creșterii sigure și constante pentru platformele blockchain permissive și se așteaptă ca în acest an, cazurile de utilizare ale acestora să își dubleze cifrele procentuale.

Trei mari trenduri care susțin dezvoltarea sunt următoarele:

1. Maturarea criptomonedelor și a monedelor digitale ale băncii centrale.
2. Încrederea în lanțurile de aprovizionare – coordonate parțial de ținte sociale, guvernamentale și de mediu. Câteva cazuri de utilizare:
 - *Context Labs*: (contextlabs.com) o platformă imutabilă care servește industria petrolului și gazelor naturale (incluzând producători și investitori) cu scopul de a reduce amprenta de carbon.
 - *Settlemint*: ([Settlemint.com](https://settlemint.com)) lucrează cu Colruyt în Belgia iar lanțul de aprovizionare verifică autenticitatea porcilor și a cărnii care provine de la aceștia sub etichetă „bio”.
 - *Copperwire*: (copperwire.io) lucrează cu o companie foarte mare din SUA care produce saltele organice și autentifică proveniența unor materiale naturale precum lâna, cauciucul sau bumbacul.
3. Straturi de abstractizare pentru blockchain cu exemple Chainlink, Copperwire sau Settlemint.

Conform descrierii realizate în [SAT08], Bitcoin dorește a fi o versiune online a banilor electronici, într-o rețea pur P2P(peer-to-peer) care să permită plățile online să fie trimise direct de la o parte la alta, fără intervenția unei instituții financiare.

2.3.1 EOS vs Ethereum

Alegerea unui blockchain potrivit pentru aplicația mea devine un proces tot mai dificil. Prima dată am analizat posibilitățile, acestea fiind EOS, Ethereum și Hyperledger Fabric [DMY20].

Începem prin analiza posibilităților enumerate mai sus.

Rețeaua EOS. EOS este un proiect de blockchain care permite utilizarea de contracte inteligente (*smart contracts*) [LAU21]. Principalele ținte ale creatorilor au fost să fie cel mai rapid, cel mai ieftin și cel mai scalabil blockchain care permite smart contracte din toate posibilitățile existente.

Blockchain-ul EOS este descentralizat iar tranzacțiile sunt verificate de către comunitate. Inițial, tokenii EOS au fost construiți peste blockchain-ul Ethereum, mai exact erau utilizați tokenii ERC-20. După lansarea rețelei principale în iunie 2018, toți tokenii au fost transferați acolo și toate transferurile se realizau tot pe EOS.

Rețeaua Ethereum. O platformă blockchain care permite oamenilor să trimită și să primească fonduri fără nevoia unei părți terțe, precum o instituție bancară.

Ethereum este primul proiect care a introdus noțiunea de un smart contract. Acesta este bazat pe condiții predefinite iar odată ce sunt îndeplinite, contractul execută operațiunile automat, fără acțiunea unui intermediar.

În continuare exemplific cum acționează un smart contract printr-un caz de utilizare simplu.

1. Mark deține o casă și decide să își facă o asigurare pentru aceasta, în caz de fenomene naturale extreme.
2. În schimbul utilizării unui broker clasic de asigurări, Mark decide să folosească un smart contract.
3. Pentru a utiliza contractul inteligent, Mark depune valoarea echivalentă pentru asigurare.
4. Contractul inteligent este capabil să caute prin toate ofertele disponibile și să aleagă cea mai potrivită poliță de asigurare a casei lui Mark.
5. Mark obține în cel mai scurt polița fără ajutorul sau intervenția unei părți terțe.

Acesta a fost doar un exemplu, dar aplicabilitatea unui smart contract poate fi extinsă spre domenii diferite, finanțe, energie, sănătate, lanțuri de aprovizionare precum și alegeri politice.

După această comparație (rezumată în Tabel 1), am considerat potrivit Ethereum în defavoarea EOS, chiar dacă EOS pare să aibă mai multe avantaje precum numărul mai mare de tranzacții per secundă, timpul mai mic pentru crearea unui nou bloc și lipsa unor costuri de tranzacționare. Ethereum are avantaje mult superioare din punctul de vedere al descentralizării și al nivelului foarte ridicat de siguranță, acestea două fiind proprietățile care m-au convins să aleg Ethereum.

Mai departe am continuat studiul pentru cel mai potrivit blockchain care să implementeze cel mai bine cerințele aplicației de monitorizare a mierii ecologice.

	ETHEREUM	EOS
SIMBOL CRIPTOMONEDĂ	Ether (ETH)	EOS
LIMBAJ CONTRACT INTELIGENT	Solidity	Web Assembly
TPS¹	~30	~4000
RATA MINARE BLOC NOU	10 – 15 secunde	0.5 secunde
ALGORITM DE CONSENS	PoW	Delegated PoS
COST TRANZACȚIONARE	Consumă <i>gas</i>	Doar resurse computaționale

Tabel 1: Ethereum vs EOS

2.3.2 Ethereum vs Hyperledger Fabric

Următorul blockchain analizat a fost Hyperledger Fabric, pe care îl compar în continuare cu Ethereum.

Hyperledger Fabric este un proiect important din familia blockchain-ului Hyperledger. Fundația Linux împreună cu IBM au dezvoltat *framework*-ul Fabric [GWY12]. Hyperledger Fabric vine cu registre, posibilitatea utilizării unui contract inteligent precum și protocoale specifice care ajută procesul de tranzacționare, având o arhitectură permisivă și privată.

Modelul flexibil din punctul de vedere al consensului este confirmat de un număr ridicat de algoritmi de consens disponibili, multe opțiuni de conectare, formate multiple de registre dar și multe alte posibilități. Toate acestea stau în spatele unei platforme ușor de extins cu o implementare

¹ Tranzacții per secundă

potrivită pentru orice fel de industrie. Industree precum sănătatea, lanțurile de aprovizionare, asigurări, media, finanțe, guverne, imobiliare utilizează deja tehnologia.

O comparație succintă este prezentată în Tabelul 2, fiind surprinse principalele diferențe dintre blockchain-ul Ethereum și Hyperledger Fabric.

	ETHEREUM	HYPERLEDGER FABRIC
TIP REGISTRU	Nepermisiv	Permisiv
DEZVOLTATOR	Comunitate dezvoltatori Ethereum	Fundația Linux
CRIPTOMONEDĂ	Ether (ETH)	Nu există
TPS	~30	~2000
ALGORITM DE CONSENS	Proof-of-work (PoW)	Pluggable Mechanism
CONTRACT INTELIGENT	Da	Da
LIMBAJ CONTRACT INTELIGENT	Solidity	NodeJS/GoLang/Java
TIP APLICAȚIE	Domenii diverse	Domenii diverse

Tabel 2 Ethereum vs Hyperledger Fabric

Sursa: <https://101blockchains.com/ethereum-vs-hyperledger-fabric/#prettyPhoto/0/>

Analizând prima linie din Tabel 2, observăm o diferență semnificativă între cele două tipuri de registre. Hyperledger Fabric este o platformă permisivă, adică nu este accesibilă publicului, fiind potrivită pentru afaceri și ceea ce își doresc acestea din punctul de vedere al intimității. De asemenea vine cu mecanisme care permit selectarea entităților care au drepturi de acces sau nu, în sistem. În schimb, platforma Ethereum este publică, fără a exista noțiunea de intimitate, dar poate fi ușor limitat accesul anumitor utilizatori, prin codul din contractul inteligent.

În ceea ce privește industria țintă a platformei Hyperledger, putem spune că este potrivită pentru dezvoltare transindustrială, adică poate fi folosită în aproape orice industrie cu multe și diverse scenarii. De asemenea, și Ethereum este foarte bun pentru dezvoltare transindustrială, chiar dacă este un registru distribuit public (o soluție adoptată a fost Ethereum Enterprise, care a creat o soluție pentru intimitatea dorită de mediile de afaceri).

Din punct de vedere al eficienței de tranzacționare, Hyperledger are avantajul unui număr mai mare de tranzacții pe secundă (TPS), > 2000. Acest avantaj vine și din utilizarea tranzacțiilor paralele pe care le poate executa blockchain-ul. Pe de altă parte, Ethereum cu un număr redus de noduri poate să ofere foarte multe tranzacții pe secundă, dar în condițiile unui blockchain public, acest număr este de aproximativ 20 TPS.

Platforma Fabric nu are un *token* nativ sau o criptomonedă în sistem, dar dacă o companie dorește, poate adăuga un *token*, cu efort suplimentar, însă. În schimb, Ethereum dispune de o criptomonedă nativă numită Ether, un *token* special numit *Gas* care face posibilă execuția unei tranzacții dar cu dezavantajul că are un cost relativ mare și crește o dată cu dimensiunea și numărul de noduri din rețea.

Dacă discutăm despre algoritmi de consens, identificăm ceva special la platforma Hyperledger, și anume că în realitate acesta oferă mai multe mecanisme dintre care se poate alege. Nativ, algoritmul Kafka este implementat, totuși există un mix de Solo (potrivit pentru dezvoltatori) și Raft (tolerant cu avarii și erori). Ethereum folosește ca mecanism de consens o variantă mai avansată a algoritmului *Proof-of-work*, fiind foarte robust și eficient dar cu neajunsul că are nevoie de multe resurse de calcul.

Ethereum a fost primul care a introdus noțiunea de contract inteligent, adică un sistem acționabil care automatizează procesul între doi membri, fiind foarte eficient. Limbajul nativ pentru scrierea unui contract inteligent este Solidity, un limbaj nou și relativ ușor comparativ cu alte limbaje de programare. Pe de altă parte, și Hyperledger pune la dispoziție mecanismul contractelor inteligente, fiind asemănător ca principiu cu cel al platformei Ethereum. În cazul Hyperledger, există 3 limbaje disponibile, care oferă flexibilitate dezvoltatorilor de contracte inteligente, mai exact GoLang, Java și NodeJS.

Ultimul criteriu de comparație, tipul de aplicație în lumea reală găsește Ethereum în avantaj, fiind foarte popular și deja folosit în mii de aplicații descentralizate. Dar și Hyperledger este într-o continuă dezvoltare, fiind preferat de companii cu un model de business în care au nevoie de mai mult control asupra participanților.

Am hotărât să aleg Ethereum și să continui dezvoltarea aplicației cu această platformă deoarece este foarte sigură, cu o perioadă de creștere și dezvoltare relativ mare raportându-ne la vechimea tehnologiei blockchain, cu utilizare foarte mare în lumea dezvoltatorilor de aplicații descentralizate. Chiar dacă există ideea că nu este potrivit pentru ceea ce vor companiile, există ca soluție Enterprise Ethereum, coordonată de Ethereum Alliance unde mai multe entități/companii se unesc și lucrează la soluții pentru afacerile lor.

Platforma deține o monedă care poate fi tranzacționată și aceasta reprezintă fundamentul unei funcționalități viitoare a aplicației, aceea de a se tranzacționa doar Ether în cadrul lanțului de aprovizionare, fără să mai existe în circulație bani clasici (poate doar la clientul care cumpără din magazin, dar și acolo doresc exclusivitate pentru criptomonedă cât mai rapid). Pentru a restricționa accesul altor membri decât creatorul unui smart contract am folosit mecanismele puse la dispoziție de limbajul Solidy, fără limitări din acest punct de vedere. Toate aceste aspecte confirmă că alegerea blockchain-ului Ethereum este potrivită pentru modelul de business al lanțului de aprovizionare.

2.4 Platforma Ethereum

Vitalik Buterin a propus prima variantă de *kernel* în noiembrie 2013 [GAV21]. Deși a avut o evoluție evidentă în ultimii ani, fundamentul nu s-a schimbat în ceea ce privește completitudinea Turing² și capacitatea nelimitată de stocare a tranzacțiilor rămâne neschimbată. Prima încercare de criptografie computațională (în 1992) a însemnat transmiterea valorii unui semnal prin Internet, faptă destul de mult criticată datorită lipsei de încredere la recepționarea semnalului, mai ales în cazul transimierii unei monede virtuale. În 2003, Vishnumurthy a exemplificat pentru prima dată conceptul de *proof-of-work* la nivel global. De această dată, tokenul a fost utilizat pentru a ține sub control tranzacțiile *peer-to-peer*³, oferind consumatori cu capacitatea de a efectua micro-plăți către furnizori pentru serviciile lor. În 2008, la 5 ani diferență, Nakamoto a introdus alt token cu abordare *proof-of-work* securizată, cu o extindere mai mare decât încercarea precedentă.

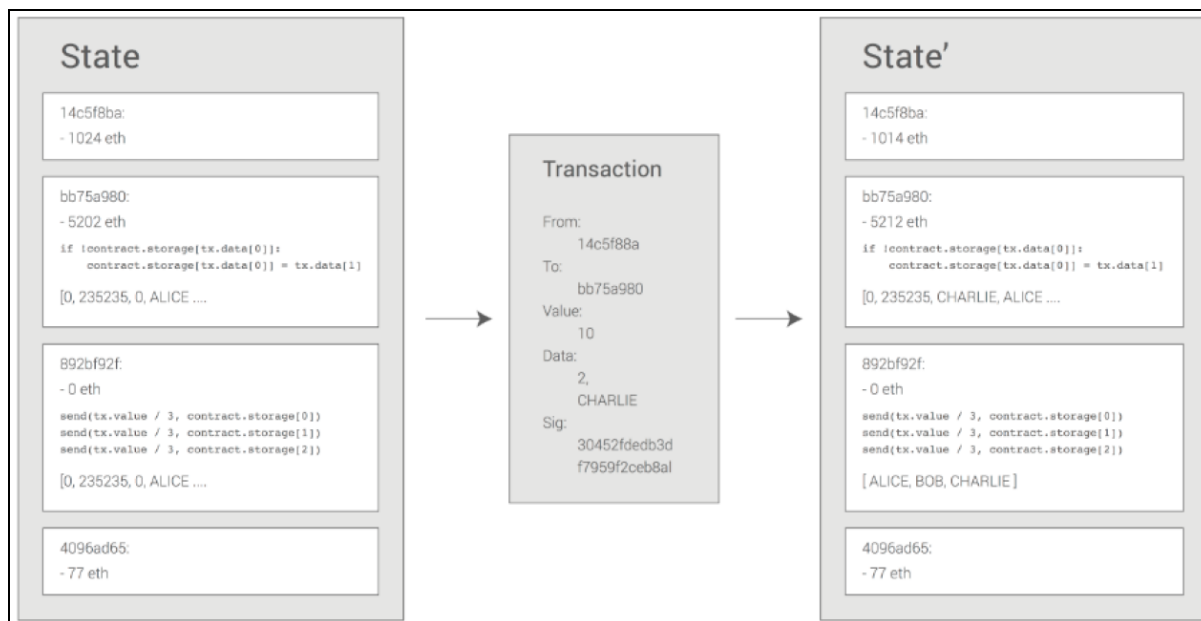
Ethereum privit ca un întreg, reprezintă o mașină de stări bazată pe tranzacții: începe cu o stare de geneză și execută tranzacții pentru a se transforma în starea curentă. Tranzacția reprezintă un arc valid între două stări (Figură 3). Este foarte important să fie validă deoarece există un număr mult mai mare de stări invalide decât cele valide. Tranzacțiile sunt unite în blocuri.

Tranzacțiile sunt grupate în blocuri iar blocurile sunt înlănțuite folosind împreună un hash criptografic ca mijloc de referință. Blocurile funcționează ca un jurnal, înregistrând o serie de

² Un sistem de calcul care poate determina fiecare funcție Turing calculabilă. Alternativ, un astfel de sistem este unul care poate simula o mașină universală Turing. Sursa: https://ro.ert.wiki/wiki/Turing_completeness

³ Conceptul peer-to-peer (P2P) a fost popularizat prin funcționalitatea de schimb/partajare de fișiere (în domeniul financiar, un cumpărător și un vânzător interacționează direct prin P2P). O rețeaua P2P a permis ca milioane de utilizatori de internet să se conecteze direct, să formeze grupuri și să colaboreze între ei pentru a funcționa ca supercomputere virtuale sau sisteme de fișiere. Sursa: <https://www.investopedia.com/terms/p/peertopeer-p2p-service.asp>

tranzacții împreună cu blocul anterior și un identificator pentru starea finală (deși aceasta nu se stochează pentru că este foarte mare).



Figură 3: Funcția de tranziție între două stări [BUT13]

De asemenea, punctează serii de tranzacții cu stimulente pentru noduri în scopul de a le „mina”. Această stimulare este realizată printr-o funcție de tranziție de stare, adăugând valoare unui cont nominalizat. *Minarea* este procesul de dedicare a efortului (de lucru) prin care este susținută o serie de tranzacții (un bloc) peste oricare alt bloc potențial concurent și se realizează datorită unei abordări criptografice.

Deoarece sistemul este descentralizat iar toate părțile au ocazia de a crea un nou bloc pe un bloc deja existent, structura rezultată este neapărat un arbore de blocuri. Pentru a forma un consens în ceea ce privește calea, de la rădăcină (blocul de geneză) până la frunze (bloc care conține cele mai recente tranzacții) prin această structură de arbore, cunoscută sub numele de blockchain, trebuie să fie un sistem de acord comun. Dacă există vreodată un dezacord între noduri cu privire la calea rădăcină-frunză și strică echilibrul blockchain-ului, apare un proces numit „fork”. Uneori, o cale urmează un nou protocol de la un anumit nivel din arbore (numărul blocului).

Conceptele de bază pentru Ethereum sunt tranzacția, blocul și stare. În continuare le analizăm pe fiecare în parte pentru a înțelege cum lucrează această platformă.

Starea globală este o mapare între adresă (identificator de 160 de biți) și starea contului (o structură de date serializată). Chiar dacă nu este stocat în blockchain, implementarea menține această mapare într-un arbore modificat Merkle Patricia având mai multe beneficii cum ar fi:

1. Nodul rădăcină al acestei structuri este criptografic dependent de toate datele interne și astfel hash⁴-ul său poate fi folosit ca o identitate sigură pentru întreaga stare a sistemului.
2. Fiind o structură de date imutabilă, permite oricărei stări anterioare (cunoscându-se hash-ul rădăcinii) să fie reapelată prin simpla modificare a hash-ului rădăcinii curente. Pentru că toate aceste hash-uri ale rădăcinilor sunt stocate în blockchain, se poate foarte ușor să se revină la stări mai vechi.

O tranzacție T, este o singură instrucțiune semnată criptografic creată de un actor extern domeniului Ethererum. Există două tipuri de tranzacții, acelea care rezultă în urma apelurilor de mesaje și acelea care sunt rezultate din crearea unor noi conturi, cu codul asociat (crearea unui contract).

Tranzacția rezultată după crearea unui contract are în plus câmpul *init*, o zonă nelimitată specificând codul EVM pentru inițializarea procedurilor, în contrast, apelul mesaj are câmpul *data* ce conține o zonă nelimitată specificând datele de intrare ale mesajului.

Un bloc din Ethereum este o colecție de informație relevantă (cunoscută sub denumirea de *header*) împreună cu informații despre tranzacții și un set de alte headere ale altor blocuri care au părintele egal cu prezentul părinte al părintelui blocului (asemenea blocuri se numesc *ommers*⁵).

Pentru a evita probleme de abuz în rețea și unele întrebări despre completitudinea Turing, Ethereum realizează toate calculele contra cost. În funcție de tipul de calcul există taxă diferită, asociată, în unități de *gas*. Fiecare tranzacție are asociată o sumă de gas specifică, *gasLimit*, și tranzacția este invalidă dacă soldul contului nu este suficient pentru operațiunea realizată în conformitate cu *gasPrice*, de asemenea, specificat în tranzacție.

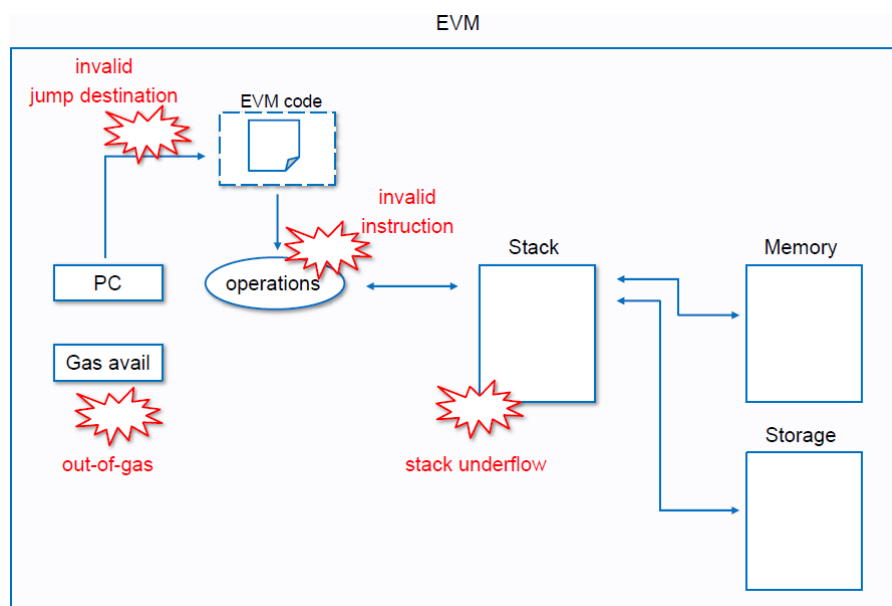
Există un număr de parametri intrinseci utilizați când un cont nou este creat: emițătorul, tranzactorul original, gas-ul disponibil, prețul gas-ului, specificul contractului, o matrice de octeți de lungime arbitrară, inițializarea codului EVM, adâncimea stivei apelului/mesajului/contractului, adresa de salt pentru contul creat și intern, permisiunea de a face sau nu modificări stării.

⁴ Un hash este o funcție matematică ce convertește o intrare de lungime arbitrară într-o ieșire criptată de lungime fixă. Funcțiile hash criptografice adaugă caracteristici de securitate funcțiilor hash tipice, ceea ce face mai dificilă detectarea conținutului unui mesaj sau a informațiilor despre destinatari și expeditori. Sursa: <https://www.investopedia.com/terms/h/hash.asp>

⁵ Termen de gen neutru care înseamnă „fratele părintelui” [GAV21].

EVM este o arhitectură simplă bazată pe stivă. Dimensiunea cuvântului mașină (și, astfel, dimensiunea stivei de elemente) este de 256 de biți. Aceasta a fost aleasă pentru a facilita schema hash Keccak-256 și calculele de criptografie prin curbe eliptice⁶.

Mașina virtuală nu urmează modelul de arhitectură von Neumann⁷. Codul de program decât să fie stocat într-o memorie general accesibilă, acesta este stocat separat într-o memorie ROM virtuală, accesibilă doar printr-o instrucțiune specializată. EVM poate arunca excepții din câteva motive precum stivă insuficientă sau instrucțiuni invalide (Figura 4).



Figură 4: Modelul de execuție al mașinii virtuale cu posibilitățile de excepții

De exemplu, pentru excepția *out-of-gas* (nu mai există gas pentru a continua tranzacția), starea curentă nu rămâne neschimbată. Mașina se oprește imediat și raportează o eroare către agentul care a executat operațiunea (un procesator de tranzacții sau progresiv, reducând mediul de execuție) care gestionează problema separat. Din punctul de vedere al ordinii octeților, EVM are ordinea *Big Endian*⁸.

⁶ Topic detaliat în capitolul 2.4.1

⁷ Arhitectura sistemului von Neumann este alcătuit din unitate de control, unitate aritmetico-logică, unite de memorie, registrii și unitate de intrare/ieșire. Este bazat pe conceptul programului stocat în calculator, unde datele de instrucțiuni și datele de program sunt stocate în aceeași memorie. Sursa:

<https://www.computerscience.gcse.guru/theory/von-neumann-architecture>

⁸ Big Endian: Cel mai semnificativ octet („capătul mare”) al datelor este plasat la octetul cu cea mai mică adresă. Restul datelor sunt plasate în ordine în următorii trei octeți din memorie.

Procesul de finalizare al unui bloc implică 4 etape:

1. Validarea (sau, dacă se minează, determinarea) ommerilor.
2. Validarea (sau, dacă se minează, determinarea) tranzacțiilor.
3. Aplicarea recompenselor.
4. Verificarea (sau, dacă se minează, un calcul valid) stării și nonce-ul blocului.

2.4.1 Criptografie prin curbă eliptică

Prima noțiune importantă din topicul de criptografie este Merkle Patricia⁹ trie [GAV21]. Arborele modificat Merkle Patricia furnizează o structură de date persistentă pentru maparea între date binare de lungime diferită.

Este definit ca o structură de date mutabilă pentru maparea între fragmente binare de 256 biți și lungimi arbitrare de date binare, adesea implementate ca o bază de date. Scopul principal este de a pune la dispoziție o valoare singulară care identifică un set de perechi cheie-valoare, care mai pot fi o secvență de 32 de octeți sau o secvență de biți vidă.

Într-o manieră similară cu trie radix¹⁰, când un trie este traversat de la rădăcină spre frunză, se poate construi o singură pereche cheie-valoare. Cheia este adunată prin traversare, dobândind un singur nibble¹¹ de la fiecare nod ramură (la fel ca în cazul unui trie radix). Spre deosebire de un trie radix, în cazul mai multor chei care partajează aceeași prefixare sau în cazul unei singure chei având un unic sufix, sunt puse la dispoziție două noduri de optimizare. Astfel, în timpul parcurgerii, se pot obține mai multe nibbles de la fiecare din celelalte două tipuri de noduri, extensie și frunză. Există trei tipuri de noduri în trie:

1. *Frunză.*
2. *Extensie.*

⁹ Practical Algorithm to Retrieve Information Coded in Alphanumeric, Journal of the ACM, 15(4):514–534, Octombrie 1968

¹⁰ Cunoscuți și sub denumirea de trie radix, sau arbori cu prefix compact și reprezintă o variantă optimizată a unui trie standard. Diferit de un arbore trie normal (poate reține un singur caracter), muchiile/pointerii unui arbore radix pot stoca o secvență a unui string (șir de caractere) sau doar un singur caracter. Sursa: <https://medium.com/basecs/compressing-radix-trees-without-too-many-tears-a2e658adb9a0>

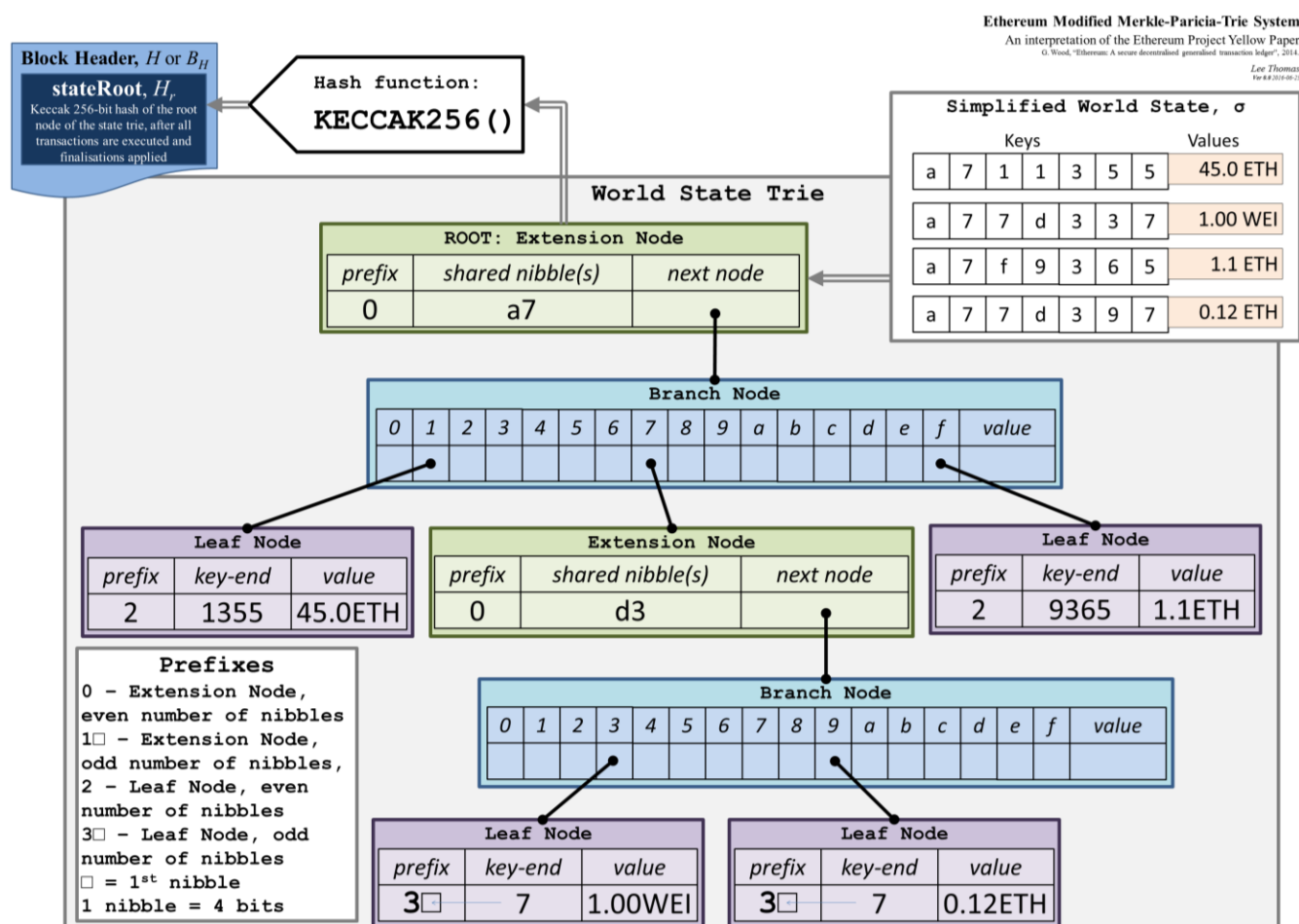
¹¹ O agregare de 4 biți, jumătate dintr-un byte/octet, Sursa: <https://www.chessprogramming.org/Nibble>

3. Ramură.

După ce am înțeles cum informația este stocată în memorie, continuăm analiza modului de criptare al datelor prin metoda curbei eliptice.

Criptografia prin curbă eliptică (ECC¹²) este una dintre cele mai puternice metode de securizare a datelor utilizată în zilele noastre [NIK22]. Istoria criptografiei poate fi împărțită în două epoci: epoca clasică și epoca modernă. Criptografia a trecut de la a schimba în siguranță coduri secrete în întreaga lume la a putea avea o comunicare sigură între oricare două părți, fără a avea teama că cineva ascultă/urmărește schimbul de chei.

Criptografia modernă se bazează pe ideea că acea cheie utilizată pentru a cripta datele personale poate fi făcută publică, în timp ce cheia care este folosită pentru a decripta datele personale poate fi păstrată privată.



Figură 5: Merkle Patricia trie - prezentare schematică

¹² Elliptic Curve Cryptography

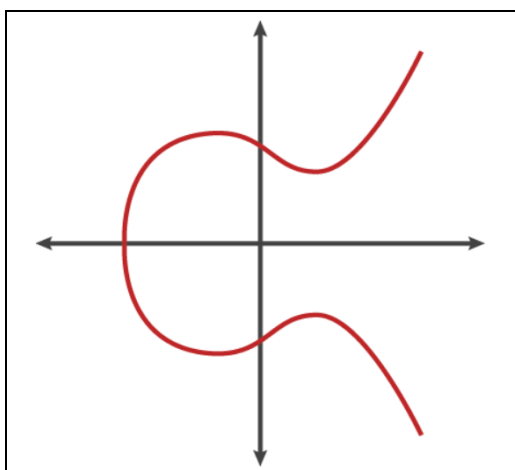
Sursa: <https://ethereum.stackexchange.com/questions/6415/eli5-how-does-a-merkle-patricia-trie-tree-work>

Ca atare, aceste sisteme sunt cunoscute ca sisteme criptografice cu cheie publică. Primul sistem și încă cel mai utilizat pe scară largă este cunoscut sub numele de RSA^{13} - numit după inițialele celor trei bărbați care au descris pentru prima dată algoritmul: Ron Rivest, Adi Shamir și Leonard Adleman.

Pentru ca un sistem criptografic cu cheie publică să funcționeze este nevoie de un set de algoritmi ușor de procesat într-o singură direcție, dar dificil de descifrat. În cazul RSA, algoritmul înmulțește simplu două numere prime. Dacă criptarea (înmulțirea) este un algoritm ușor, algoritmul său omolog (decriptarea) factorizează produsul în primele două componente obținute la procesul de criptare. Algoritmii care au această caracteristică - ușor într-o direcție, greu în cealaltă - sunt cunoscuți ca *Funcții Trapdoor*. Găsirea unei funcții Trapdoor potrivite este esențială pentru crearea unui sistem criptografic securizat cu cheie publică.

În 1985, algoritmii criptografici au fost propuși pe baza unei ramuri a matematicii numite curbe eliptice. O curbă eliptică este o mulțime de puncte care satisfac o ecuație matematică specifică. Ecuația unei curbe eliptice arată astfel:

$$y^2 = x^3 + ax + b$$



Figură 6: Graficul unei curbe eliptice [NIK22]

¹³ Algoritmul RSA este cel mai popular sistem de criptografie cu cheie publică. Securitatea sa se bazează pe faptul că factorizarea (decriptarea) este lentă, iar multiplicarea (criptarea) este rapidă. [NIK22]

O curbă eliptică are mai multe proprietăți interesante. Una dintre acestea este simetria orizontală. Orice punct de pe curbă poate fi reflectat pe axa x și rămâne aceeași curbă. O proprietate mai interesantă este că orice linie non-verticală va intersecta curba în cel mult trei locuri.

Pentru a evidenția caracterul criptografic al curbei eliptice, trebuie să ne restrângem la numere într-un interval fix, ca în cazul RSA. În loc să permitem orice valoare pentru punctele de pe curbă, ne restrângem la numere întregi într-un interval fix. Când calculăm formula pentru curba eliptică ($y^2 = x^3 + ax + b$), folosim același mecanism de a trece peste numere atunci când atingem maximul. Dacă alegem maximul ca număr prim, curba eliptică se numește curbă primă și are foarte bune proprietăți criptografice.

Ecuția pentru o linie pe curbă are aceleași proprietăți. Mai mult, produsul scalar poate fi calculat eficient. Se poate observa linia dintre două puncte ca o linie care se prelungește la margini până când atinge fiecare punct.

Un criptosistem de curbă eliptică poate fi definit prin alegerea unui număr prim ca maxim, a unei ecuații de curbă și a unui punct public pe curbă. O cheie privată este un număr, notat *priv*, iar o cheie publică este formată din produsul scalar dintre punctul public cu el însuși de *priv* ori. Calculul cheii private din cheia publică în acest tip de criptosistem se numește *funcția logaritmică discretă a curbei eliptice*. Aceasta se dovedește a fi funcția Trapdoor pe care o căutam. Funcția logaritm discret al curbei eliptice este problema grea care stă la baza criptografiei curbei eliptice.

Semnăturile criptografice sunt o parte esențială a blockchain-ului [MAA08]. Acestea sunt utilizate pentru a dovedi proprietatea unei adrese fără a expune cheia sa privată. Acesta este utilizat în principal pentru semnarea tranzacțiilor, dar poate fi utilizat și pentru semnarea de mesaje arbitrare.

O semnătură în criptografie, înseamnă un fel de dovadă de proprietate, validitate, integritate în anumite situații precum: dovada că avem cheia privată pentru o adresă (autentificare) sau asigurarea faptului că un mesaj nu a fost modificat din diferite motive.

Semnăturile ECDSA¹⁴ constau din două numere (întregi): r și s . Ethereum folosește, de asemenea, o variabilă suplimentară v (identificator de recuperare). Semnătura poate fi notată ca $\{r, s, v\}$. Pentru a crea o semnătură, avem nevoie de mesajul de semnat și de cheia privată (d_a) cu care să îl semnăm. Procesul de semnare rezumat este:

1. Calculăm un hash (e) din mesaj pentru a semna.

¹⁴ Elliptic Curve Digital Signature Algorithm

2. Generăm o valoare aleatorie sigură pentru k .
3. Calculăm punctul (x_1, y_1) pe curba eliptică înmulțind k cu constanta G a curbei eliptice.
4. Calculăm $r = x_1 \bmod n$. Dacă r este egal cu zero, revenim la pasul 2.
5. Calculăm $s = k^{-1}(e + rd_a) \bmod n$. Dacă s este egal cu zero, revenim la pasul 2.

În Ethereum, hash-ul este de obicei calculat cu $Keccak256("\x19Ethereum Signed Message:\n32" + Keccak256(message))$. Acest lucru asigură faptul că semnătura nu poate fi utilizată în scopuri externe Ethereum.

2.4.2 Algoritmi de consens

Un topic de o importanță majoră pentru o rețea blockchain precum Ethereum, care în esență este o bază de date distribuită este reprezentat de mecanismul de consens din cadrul sistemelor distribuite. Ca o definiție simplă, putem spune că un sistem distribuit este un set de noduri (în general computere) care realizează o sarcină comună, simultan comunicând între ele printr-o rețea. Cu această idee de sistem distribuit, putem spune că nodurile dintr-un sistem distribuit care convin asupra unei anumite valori reprezintă un consens distribuit [ISU19]. Cazurile comune de utilizare a consensului distribuit sunt:

- Alegerea unui lider între noduri.
- Luarea unei decizii de comitere.
- Replicarea datelor în noduri.

Pentru a obține consensul într-un sistem distribuit, fiecare nod ar trebui să urmeze același protocol atunci când comunică. Există trei condiții care ar trebui îndeplinite de un algoritm de consens pentru a obține un consens distribuit [ISU19]:

- *Acord*. Condiția acordului înseamnă că toate nodurile *non-faulty*¹⁵ ar trebui să fie de acord asupra aceleiași valori.

¹⁵ În situații practice, nodurile dintr-un sistem distribuit se pot bloca, funcționează defectuos sau pot fi piratate. Aceste noduri sunt noduri defecte și, prin urmare, nesigure. Deci, este mai greu să obții un consens într-un sistem distribuit atunci când există noduri defecte. Există două tipuri de eșecuri luate în considerare în sistemele distribuite: eșecuri de tip avarie și eșecuri bizantine [ISU19].

- *Valabilitate*. Valoarea cu care este de acord un nod ar trebui să fie o valoare sugerată de unul dintre nodurile din sistemul distribuit.
- *Finitudine*. Conform condiției de terminare, fiecare nod *non-faulty* va decide în cele din urmă o anumită valoare. Dacă un singur nod nu este de acord, consensul într-un sistem distribuit nu poate fi atins.

Nodurile din sistemele distribuite încearcă să atingă un obiectiv comun (poate prelucra un calcul mare), astfel apare nevoia de o coordonare între ele. Sistemele distribuite au nevoie de un protocol de consens pentru că ele trebuie să știe efectul acțiunilor lor asupra întregului sistem.

Mecanismele de consens (cunoscute și sub numele de protocoale de consens sau algoritmi de consens) permit sistemelor distribuite (rețele de computere) să lucreze împreună și să rămână în siguranță [PAU30].

Un mecanism de consens într-un sistem cripto-economic ajută, de asemenea, la prevenirea anumitor tipuri de atacuri economice. În teorie, un atacator poate compromite consensul controlând 51% din rețea. Mecanismele de consens sunt concepute pentru a face imposibil de realizat acest „atac de 51%”. Diferite mecanisme sunt concepute pentru a rezolva diferit această problemă de securitate [PAU30].

Ethereum utilizează pentru pentru moment protocolul de consens *Proof-of-work*. Dovada muncii (proof-of-work) este făcută de mineri, care concurează pentru a crea noi blocuri pline de tranzacții procesate. Câștigătorul împarte noul bloc cu restul rețelei și câștigă niște ETH proaspăt bătute. Cursa este câștigată de către computerul cui poate rezolva cel mai rapid un puzzle matematic - aceasta produce legătura criptografică între blocul curent și blocul care a fost înainte [PAU30]. Din punct de vedere al securității, Rețeaua este menținută în siguranță prin faptul că ar fi nevoie de 51% din puterea de calcul a rețelei pentru a frauda lanțul, ceea ce este foarte costisitor, cu investiții foarte mari în echipamente și energie fiind în mare pierdere.

Protocolul de consens *proof-of-work*, cunoscut și sub denumirea de Ethash solicită din partea minerilor un efort deosebit cu încercări și erori pentru a determina nonce-ul pentru bloc deoarece numai blocurile cu un nonce valid pot fi adăugate mai departe în lanț. Atunci când concurează pentru a crea un bloc, un miner va pune în mod repetat un set de date, pe care îl putem obține numai din descărcarea și rularea lanțului complet (așa cum o face un miner), printr-o funcție matematică. Aceasta este pentru a genera un mixHash care se află sub un *nonce* țintă, așa cum este dictat de dificultatea blocului. Cel mai bun mod de a face acest lucru este prin încercare și eroare [PAU08].

Minerii care creează cu succes un bloc sunt recompensați cu 2 ETH și toate taxele de tranzacție din cadrul blocului. Un miner poate obține, de asemenea, 1,75 ETH pentru un bloc unchi. Acesta este un bloc valid, creat simultan cu blocul, de către un alt miner. Acest lucru se întâmplă de obicei din cauza latenței rețelei. Deoarece minerii lucrează într-un mod descentralizat, este posibil ca două blocuri valide să fie minate în același timp. Aceasta creează un *fork* temporar. În cele din urmă, un lanț va deveni lanțul acceptat odată ce un bloc ulterior a fost extras și adăugat, făcându-l mai lung.

Ethereum are planuri de viitor pentru a face upgrade la un protocol de consens numit *Proof-of-Stake* (PoS) [PAU30]. *Proof-of-stake*-ul este realizat de validatori care au mizat ETH pentru a participa în cadrul sistemului. Un validator este ales la întâmplare pentru a crea blocuri noi, a le partaja cu rețeaua și a câștiga recompense. În loc să trebuiască o muncă intensă de calcul, trebuie pur și simplu o miză cu ETH în rețea. Acesta este ceea ce stimulează un comportament sănătos în rețea. Sistemul este păstrat în siguranță prin faptul că ar fi nevoie de 51% din totalul ETH pentru a frauda blockchain-ul dar și prin faptul că miza este anulată imediat pentru comportament rău intenționat [PAU30].

La un nivel global, *proof-of-stake* are același obiectiv final ca și *proof-of-work*: de a ajuta rețeaua descentralizată să ajungă la consens, în siguranță, dar cu unele diferențe de proces [PAU08]:

- PoS schimbă importanța puterii de calcul cu ETH mizat.
- PoS înlocuiește minerii cu validatori. Validatorii își pun în joc ETH pentru a activa capacitatea de a crea noi blocuri.
- Validatorii nu concurează pentru a crea blocuri, în schimb sunt aleși la întâmplare de un algoritm.
- Finalitatea este mai clară: la anumite puncte de control, dacă 2/3 validatori sunt de acord asupra stării blocului, acesta este considerat final. Validatorii trebuie să parieze întreaga lor miză pe acest lucru, așa că, dacă încearcă să se strecoare, își vor pierde întreaga miză.

2.4.3 Smart contract

Dacă blockchain-ul este o bază de date, atunci numim un contract inteligent ca o procedură stocată [STE20]. Token-ul care circulă în cadrul rețelei Ethereum este Ether, care se poate transfera cu ușurință. O schemă simplificată pentru acest proces este:

De la: Ionuț

Către: Andrei

Suma: 7 Ether

Data: „Transfer pentru cadoul Alexandrei”

Tabel 3 Exemplu de transfer cu 7 Ether

Această tranzacție descrie o transformare de stare foarte simplă. Soldul lui Ionuț este redus cu 7 Ether, iar soldul lui Andrei este mărit cu aceeași sumă. Fără implicarea contractelor inteligente, informațiile din câmpul de date sunt stocate permanent ca parte a tranzacției. Viitoarele entități nu vor trebui să se întrebe niciodată de ce Ionuț a plătit lui Andrei 7 Ether.

De asemenea, Ethereum are un fel special de tranzacție care nu este trimisă către nimeni:

De la: Ionuț

Către: null

Suma: 0 Ether

Data: ... (bytecode)

Tabel 4 Tranzacția pentru construcția unui contract inteligent

În tabelul 5 observăm tipul special de tranzacție utilizat pentru a crea un contract inteligent, fără a face un transfer între conturi. Un contract inteligent este un cont, cum este și al lui Ionuț sau Andrei. Singura diferență este că are o bucată de cod asociată. Codul respectiv este executat automat dacă cineva trimite o tranzacție în contul contractului inteligent. De aici rezultă termenul „procedură stocată” [STE20].

Contractele inteligente Ethereum au propriul lor sold cu Ether ca orice alt cont și au, de asemenea, stocare, pe care o pot folosi pentru a reține starea între diverse invocări de cod. Atunci când sunt invocate, pot manipula stocarea și interacționa cu alte conturi (inclusiv alte contracte inteligente) trimițând propriile tranzacții [STE20].

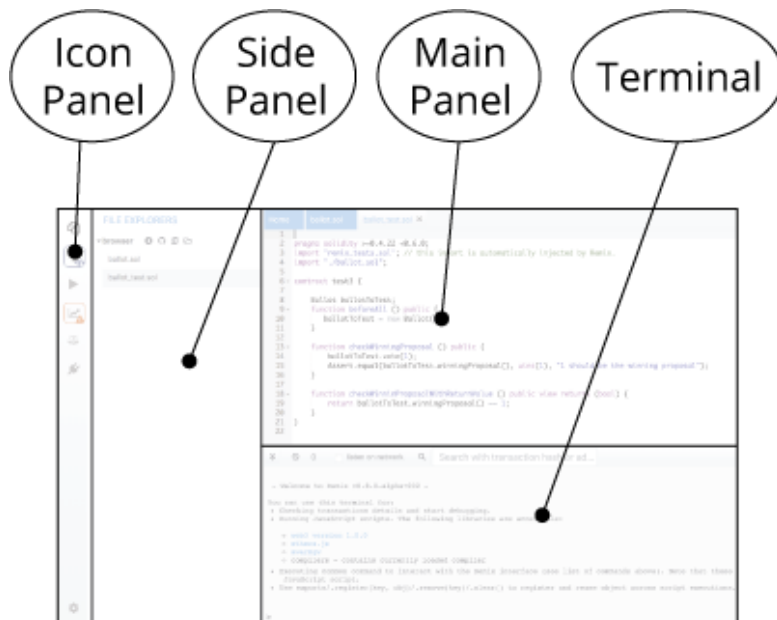
2.4.4 Solidity

Solidity este un limbaj de programare creat în 2014–2015 de Gavin Wood ca limbaj de programare complet Turing pentru a interacționa cu mașina virtuală Ethereum. Turing complet înseamnă un sistem care poate rezolva orice problemă de calcul cu suficientă memorie și putere

[TOM22]. Există sute de mii de dezvoltatori care folosesc limbajul de programare pentru a crea servicii bazate pe blockchain pentru un număr tot mai mare de cazuri de utilizare [CAL19].

Solidity este similar cu unul dintre cele mai comune limbaje de programare, JavaScript. Poate fi considerat ca un dialect al JavaScript-ului. Solidity are, de asemenea, caracteristici similare cu limbajele de programare C++ și Python. Este tastat static, cu suport pentru moștenire, biblioteci și tipuri complexe definite de utilizator. Deoarece Solidity este tastat static, utilizatorul specifică fiecare variabilă. Tipurile de date permit compilatorului să verifice utilizarea corectă a variabilelor. Tipurile de date din Solidity sunt de obicei clasificate fie ca tipuri valoare, fie ca tipuri referință. Principala diferență între tipurile valoare și tipurile referință este modul în care acestea sunt atribuite unei variabile și stocate în EVM (Ethereum Virtual Machine). În timp ce modificarea valorii într-o variabilă a unui tip valoare nu afectează valoarea unei alte variabile, oricine referă valorile modificate din variabilele tipului referință poate obține valori actualizate [CAL19].

Una dintre componentele cheie care face posibilă executarea codului Solidity este EVM. În spate, Solidity creează cod la nivel de mașină care este executat pe EVM. Un compilator este folosit pentru a descompune codul lizibil de la nivel înalt, pe care îl transformă în instrucțiuni citite de procesor. Diferite platforme oferă compilare Solidity gratuită (Figură 7), inclusiv compilatorul online Remix și un compilator descărcat de tip comandă pe un PC.



Figură 7: Aspect Remix-IDE

Sursa: https://remix-ide.readthedocs.io/en/latest/_images/a-layout1c.png

Funcțiile publice sunt similare cu API¹⁶-urile pe care le poate accesa oricine din lume. Oricine le poate apela în codul său. Funcțiile publice sunt concepute, în multe cazuri, pentru procese partajate pe o platformă pe care toți utilizatorii o utilizează. Un exemplu concret este funcția care actualizează valoarea unei variabile de stare din contract, funcție ce poate fi executată de orice alt participant la blockchain. Funcțiile private pot fi apelate numai din interiorul contractelor. Acestea conțin instrucțiuni care pot fi executate numai după ce au fost apelate de alte funcții, într-un lanț. Acest lucru face mai dificilă manipularea codului de către actori răuvoitori [CAL19].

Deoarece blockchain-ul Ethereum este imuabil, este imposibil să se schimbe datele și logica scrisă într-un contract inteligent. O modalitate de a rezolva acest lucru este de a utiliza un proxy pentru a indica spre alt contract care conține logica reală dorită. Acest lucru permite remedierea erorilor în timp ce se implementează o nouă versiune a contractului [CAL19].

Un resursă online unde se poate învăța Solidity prin practică interactivă poate fi accesată la adresa: <https://cryptozombies.io/>.

2.4.5 Toolchain

La fel ca toate celelalte procese de dezvoltare software, crearea de aplicații descentralizate în Ethereum vine cu suita sa de instrumente de dezvoltare. De fapt, există mai multe opțiuni. Pentru implementarea aplicației am ales să utilizez instrumentele următoare:

- *Ganache*. Blockchain personal pentru dezvoltarea rapidă a aplicațiilor distribuite Ethereum și Corda. Se poate utiliza Ganache pe întregul ciclu de dezvoltare, implementare și testare a aplicației descentralizate într-un mediu sigur și determinist. Mai multe informații pot fi consultate la adresa: <https://www.trufflesuite.com/docs/ganache/overview>
- *Visual Studio Code cu alternativa online, Remix*. Am utilizat Visual Studio Code ca mediu de dezvoltare integrat (IDE). O alternativă rapidă este IDE-ul online Remix (Figura 17). Remix IDE¹⁷ este o aplicație web și desktop open source. Încurajează un ciclu de dezvoltare rapid și are un set bogat de pluginuri cu UI intuitiv. Remix este utilizat pentru întreaga dezvoltare a contractelor inteligente. De asemenea, Remix IDE are module pentru testarea, depanarea și implementarea contractelor inteligente

¹⁶ Application Programming Interface

¹⁷ Integrated Development Environment

și multe altele. Mai multe informații la adresa: <https://remix-ide.readthedocs.io/en/latest/>.

- *Suita Truffle*. Un mediu de dezvoltare de clasă globală și un cadru de testare pentru blockchain-uri care folosesc mașina virtuală Ethereum (EVM), cu scopul de a ușura munca unui dezvoltator. Cu Truffle, putem obține:
 - Compilare, implementare și gestionare binară a contractelor inteligente.
 - Testare automată a contractelor pentru o dezvoltare rapidă.
 - Cadrul de implementare și migrare extensibil.
 - Administrare pentru implementarea în orice număr de rețele publice și private.
 - Consolă interactivă pentru comunicare directă cu un contract.

Mai multe pot fi consultate la adresa:

<https://www.trufflesuite.com/docs/truffle/overview>.

Lista completă disponibilă pentru dezvoltarea unei aplicații se poate consulta la adresa: <https://github.com/ConsenSys/ethereum-developer-tools-list>.

Acestea au fost instrumentele utilizate pentru dezvoltarea aplicației descentralizate, printre cele mai folosite în 2021 de către toți programatorii specializați dar și de cei care doresc să învețe. Pe lângă acestea, am folosit web3.js¹⁸.

¹⁸ O colecție de biblioteci care permit interacțiunea cu un nod Ethereum local sau la distanță folosind HTTP, IPC sau WebSocket. Sursa: <https://web3js.readthedocs.io/en/v1.4.0/>

3 PROIECTAREA SISTEMULUI

3.1 Cazuri de utilizare/Cerințe

În zilele noastre, tehnologia este adoptată din ce în ce mai mult de procese realizate de oameni, în diferite domenii de lucru. Rapiditatea adopției este datorată timpului de lucru economisit, eficienței și simplificării unui proces coordonat cu ajutorul tehnologiei.

Mai mult, înmulțirea segmentelor care folosesc tehnologii pentru automatizarea proceselor ridică probleme noi de transparență și încredere în sistem precum și de protecția datelor cu caracter personal.

Achiziționarea unui produs cu adevărat ecologic este foarte importantă, întreg lanțul de aprovizionare având nevoie de corectitudine, confidențialitate pentru datele cu caracter personal, imutabilitate pentru datele înregistrate dar și transparență în orice moment s-ar afla procesul.

O soluție care vine spre rezolvarea problemelor de mai sus este utilizarea tehnologiei blockchain cu toate avantajele unui ecosistem distribuit. Soluția este adaptată modelului de business specific unui lanț de aprovizionare deoarece tehnologia blockchain adaugă un plus de valoare acestuia prin perfecționarea modului de tranzacționare între entități. Îmbunătățirea tranzacționării este posibilă cu ajutorul unui registru distribuit în care participanți autorizați au acces la scrierea și citirea în timp real a informațiilor prelevate pe durata întregului proces/lanț de aprovizionare.

Managementul lanțului de aprovizionare descrie un sistem complex, care presupune multă documentație din partea fiecărui membru, documente, facturi care cu ușurință pot fi fraudate.

Mai jos prezint câteva exemple de lanțuri de aprovizionare care utilizează tehnologia blockchain [HAC17]:

- Livrarea de containere este un caz de utilizare pentru a reduce documentația și posibilitatea de fraudă printre mulții participanți la proces.
- Aprovizionarea cu medicamente contrafăcute reprezintă o problemă foarte mare, pentru că de acestea depinde viața unei ființe vii, și nu trebuie să existe nicio îndoială cu privire la calitatea unui astfel de produs, blockchain-ul aducând un plus de siguranță.
- Facilitatea de a urmări un produs de la origini până la vânzare, asigurând consumatorul că produsul achiziționat este calitativ, un vânzător poate să observe dacă un produs este potrivit

sau nu pentru raftul de produse ecologice și pentru consumator este disponibil un raport scurt cu privire la toți pașii prin care a trecut produsul de-a lungul propriului lanț de aprovizionare.

- Utilizarea Internet of Things prin echiparea cu senzori pe cât mai multe obiecte din lanțul de aprovizionare, colectează date în diferite momente și le înregistrează în blockchain.

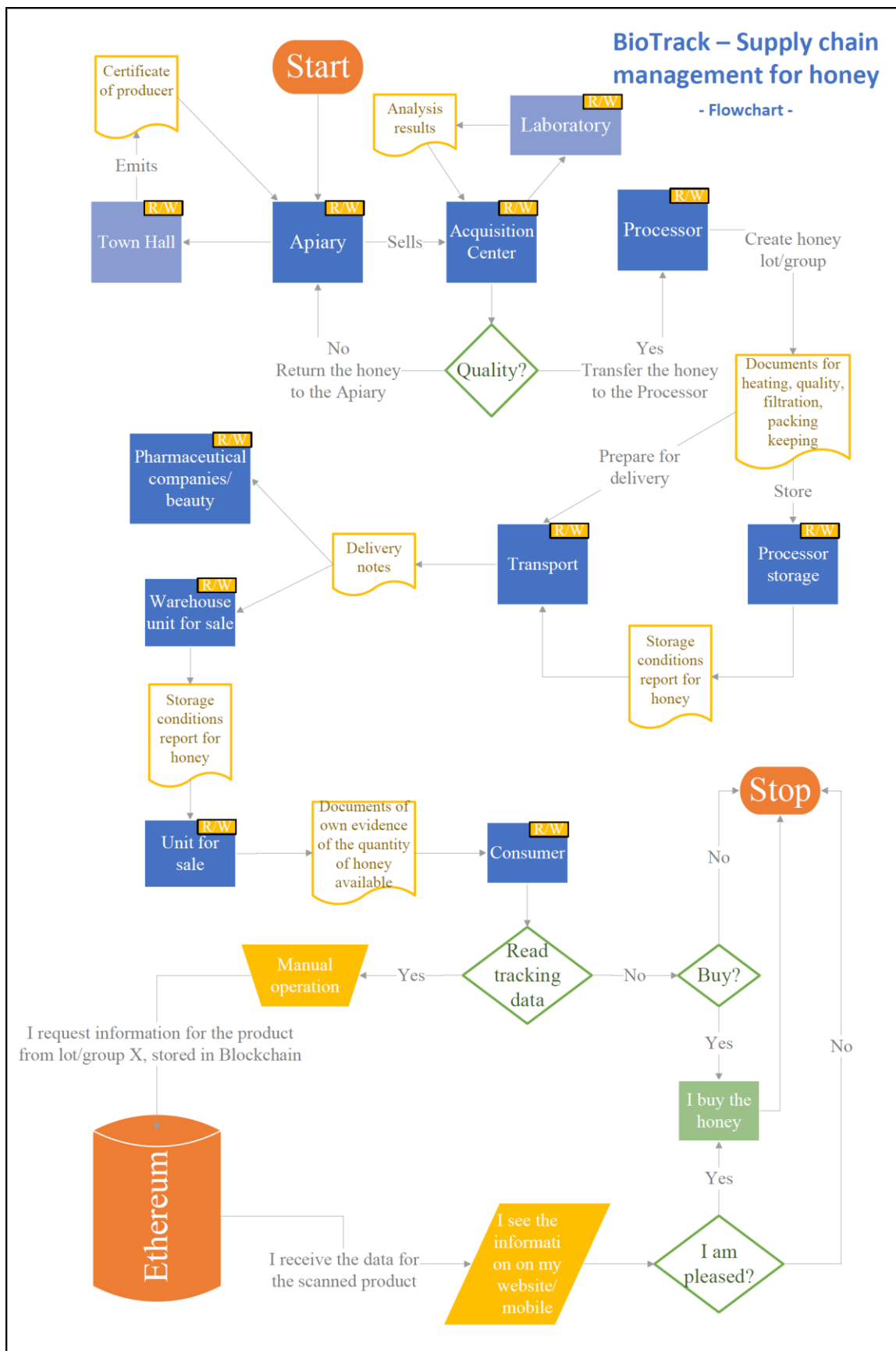
Termenul de "managementul lanțului de aprovizionare" a apărut la sfârșitul anilor 1980 și a început să fie folosit la nivel global după anii 1990. Înainte de această dată, întreprinderile au folosit termeni precum "logistică" și "gestionarea operațiunilor". În literatura de specialitate există mai multe definiții pentru un lanț de aprovizionare, prezentate fiind câteva, preluate din [FAL12]:

1. Un lanț de aprovizionare constă într-o multitudine de etape, implicate direct sau indirect pentru îndeplinirea nevoii clienților. Acesta nu include doar producătorii și furnizorii, ci și transportatorii, depozitele, comercianții și clienții.
2. Un lanț de aprovizionare este o rețea de organizații care cooperează cu scopul de a eficientiza fluxul de materiale dintre furnizorul inițial și consumator, de asemenea cu avantaje în sfera costurilor.

3.2 Schema bloc a întregului sistem

Aplicația propusă pentru proiectare oferă utilizatorilor posibilitatea urmăririi unui produs apicol, mai exact, mierea, de la stupină până la consumator. Soluția constă dintr-un model de management proiectat să satisfacă standardele și cerințele fiecarui membru din lanțul de aprovizionare.

În continuare este prezentată diagrama de process, cu fiecare membru, ce date sunt intră în fiecare proces, ce date ies din fiecare proces. Ulterior este explicat modul de desfășurare al întregului lanț de aprovizionare.



Figură 8: Diagrama de proces pentru lanțul de aprovizionare

Arhitectura aplicației presupune împărțirea lanțului de aprovizionare în mai multe module (Figură 8). Fiecare modul reprezintă un membru/actor din proces care interacționează direct sau indirect pentru a îndeplini scopul final, acela de a oferi consumatorului un raport cu toate etapele parcurse de o anumită cantitate de miere de la stupină până în momentul achiziționării.

Transpunerea unui astfel de modul pentru a putea lucra pe blockchain presupune scrierea unui smart contract care să cuprindă toată logica de business. Potrivit Investopedia, [FRA21], un contract inteligent este un contract care se execută de sine stătător, termenii acordului dintre cumpărător și vânzător (două entități) fiind direct transpuși prin cod. Codul și acordurile conținute în acesta există într-o rețea blockchain distribuită și descentralizată. Codul controlează execuția, iar tranzacțiile sunt urmărite și ireversibile.

Acest avantaj al tranzacțiilor urmărite și ireversibile consolidează nivelul de încredere garantat pentru soluția implementată, odată înregistrată o informație în urma execuției unui contract inteligent, informația devine imutabilă și astfel, fiecare membru poate verifica ce s-a tranzacționat la un moment dat.

Interfețele disponibile în cadrul sistemului prezintă caracteristici diferite care permit următoarea clasificare:

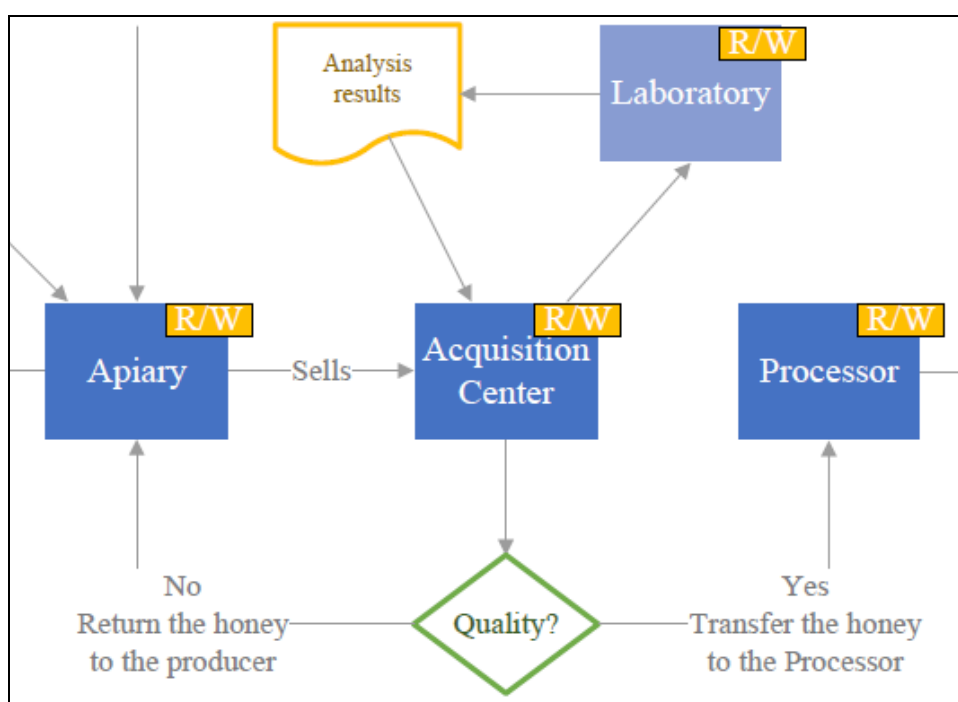
- *Interfețe de scriere/citire în/din blockchain.* Datele noi înregistrate de fiecare membru sunt scrise în registrul distribuit cu ajutorul unei interfețe de scriere, care modifică starea curentă. De asemenea, datele deja înregistrate pot fi accesate utilizând o interfață de citire.
- *Interfețe între cel puțin 2 membri din proces.* Aceste interfețe sunt utile pentru transferuri de bunuri între diferiți membri, generarea unor rapoarte, acțiuni, decizii.

Procesul de urmărire pentru o cantitate de miere începe chiar prin obținerea acesteia, direct de la o stupină. Pentru ca mierea să fie eligibilă mai departe în proces, apicultorul este responsabil să dețină un certificat de producător, valid. Aici intervine un nou membru din lanț, și anume, Primăria, văzut ca un membru generic ce poate fi particularizat mai departe prin adăugarea unor atribute precum numele localității, țara, etc. Fiecare primărie are un cont valid în rețeaua privată de blockchain prin intermediul căruia poate executa operația de emitere a unui certificat de producător nou, pentru contul solicitant.

Odată ce apicultorul are un certificat de producător valid, acesta poate să vândă mierea mai către un centru de colectare. Astfel, am introdus următorul membru al lanțului de aprovizionare, centrul de colectare miere, care are câteva acțiuni posibile de executat: colectarea mierii de la mai mulți apicultori, trimite spre verificare mierea și transferă mierea mai departe spre procesator. Mierea

colectată poate fi de mai multe tipuri, în stare solidă, cristalizată sau în stare lichidă, păstrată în condiții normale. Centrul de colectare este responsabil să trimită către laborator o probă din fiecare cantitate de miere primită de la apicultori. Laboratorul este următorul membru, foarte important, din lanțul de aprovizionare. Acesta analizează mierea primită și întoarce un raport către centrul de colectare. Raportul conține informații cu privire la tipul de miere, vechimea estimată, prețul analizei care se scade din suma returnată către stupină și cel mai important dacă respectiva cantitate este sau nu ecologică. Pasul imediat aparține centrului de colectare care analizează rezultatele rapoartelor și realizează următoarele acțiuni în funcție de miere (Figură 9):

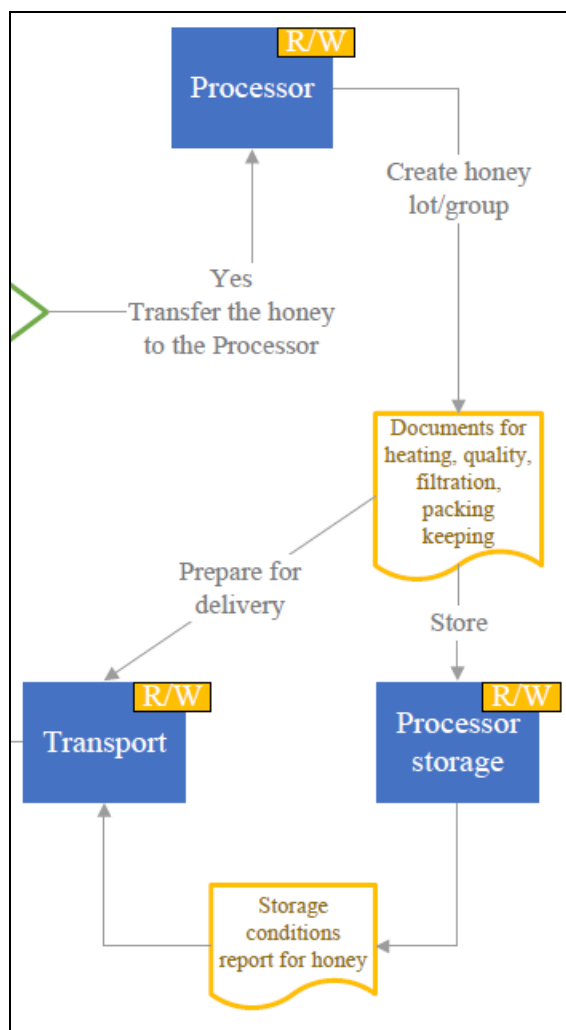
- Dacă rezultatul analizei dovedește că mierea este ecologică (într-un procent mai mare de 98%), atunci centrul de colectare transferă banii către stupină.
- Dacă rezultatul analizei arată că mierea nu este ecologică, atunci centrul de colectare returnează cantitatea de miere către stupină, în aceeași stare în care a primit-o, în cel mult 48 de ore.



Figură 9: Interacțiunea Stupină-Centrul de colectare-Laborator-Procesator

Dacă mierea este ecologică, centrul de colectare transferă mai departe mierea către Procesator. Acesta primește separat fiecare cantitate de miere și alcătuiește loturi, cu informații legate de stupină, centrul de colectare, tip dar și cantitate. Un lot de miere este separat, etichetat, împachetat și pregătit de transport sau pentru a fi depozitat o perioadă.

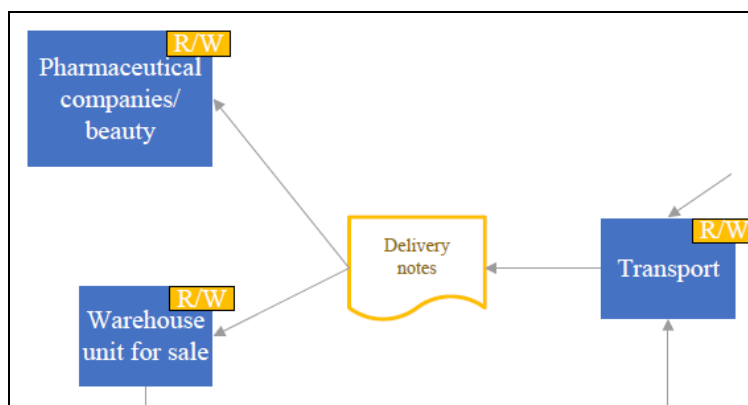
După ce loturile de miere sunt pregătite, acestea pot fi depozitate la procesator pentru o perioadă scurtă de timp, caz în care depozitul trebuie să emită un document/raport cu privire la condițiile de stocare pe întreaga perioadă. Cealaltă posibilitate este ca mierea să fie preluată direct de un transportator care să livreze mai departe în lanțul de aprovizionare.



Figură 10: Cele 2 posibilități distincte de continuare de la procesator

Transportatorul eliberează un raport cu privire la condițiile de transport, detalii despre călătorie și cantitatea de miere transferată.

Destinația finală este un alt membru al lanțului de aprovizionare. Acesta poate fi o companie farmaceutică sau de frumusețe, care folosește miere ecologică în procesul de producție sau un alt membru, depozitul unui magazin (Figură 11).



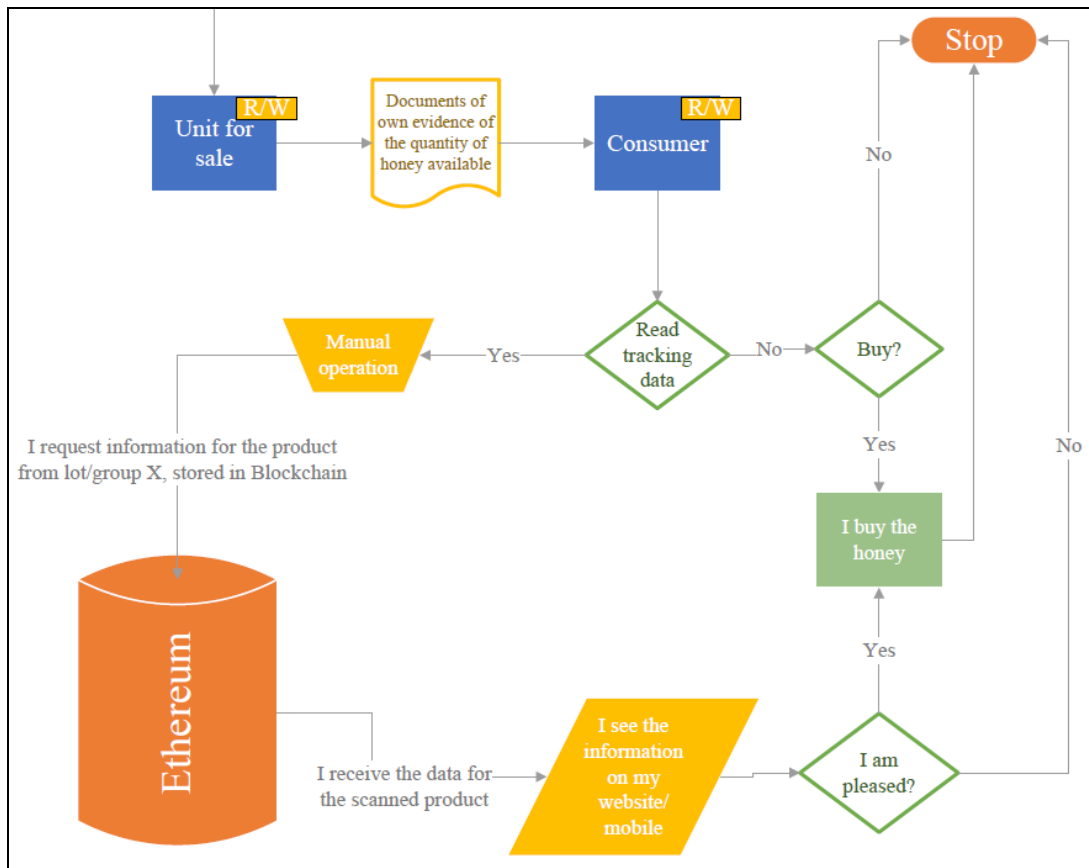
Figură 11: Posibilitățile de transport

Intern, depozitul magazinului trebuie să elibereze un raport despre condițiile de depozitare, perioada dar și despre stocul de miere disponibil care poate fi mutat direct în magazin.

Când mierea ajunge pe rafturile magazinului, aceasta este la ultimul pas înainte de interacțiunea cu un posibil cumpărător. În raportul magazinul se poate observa perioada de timp cât a stat pe raft și ce cantitate disponibilă a rămas din fiecare tip.

Datorită arhitecturii blockchain care este fundamentul aplicației, consumatorul are disponibilă opțiunea de a vizualiza date din întreg lanțul de aprovizionare (Figură 12). Dacă alege această opțiune, printr-o operațiune manuală, obține orice informație are nevoie. În soluția prezentată, consumatorul este limitat la a verifica doar un subset de proprietăți considerate importante la momentul proiectării aplicației.

Consumatorul poate alege să nu verifice istoricul provenienței produsului caz în care poate cumpăra mierea și tot procesul ajunge la final (tot în punctul final ajunge procesul și dacă nu cumpără produsul). Achiziționarea produsului poate fi influențată de rezultatul înregistrărilor, caz în care cumpărătorul își adresează întrebarea dacă este sau nu mulțumit de informațiile citite. În primul caz acesta cumpără mierea, mulțumit fiind de ceea ce a citit sau renunță la achiziția mierii, în ambele cazuri procesul ajungând la final.



Figură 12: Acțiunile unui consummator

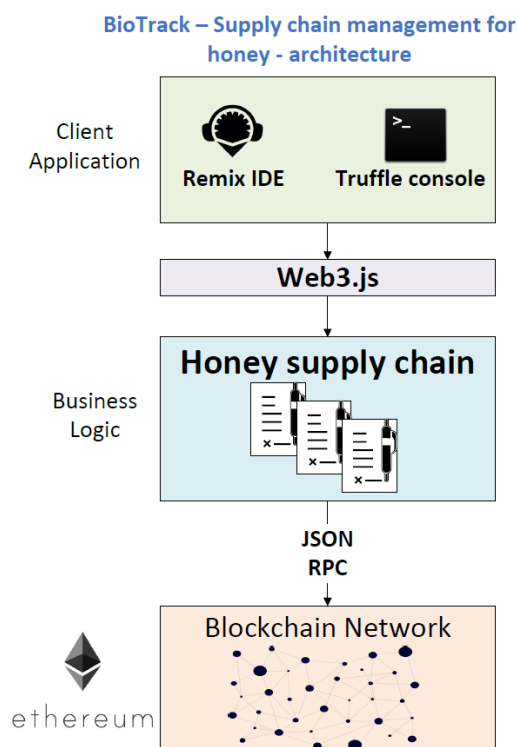
Considerând momentul de timp unic, X_1 , ca începutul procesului și momentul de timp X_2 ca finalul procesului, de la x_1 la x_2 avem Δt parcurs. Considerăm o cantitate fixă de miere, notată cu O și pentru starea mierii în momentul trecerii prin membrul i , notăm O_{M_i} cu $i = 0, 10$:

$$X_1 + \Delta t = O_{M_0} \rightarrow O_{M_1} \rightarrow O_{M_2} \rightarrow O_{M_3} \rightarrow O_{M_4} \rightarrow O_{M_5} \rightarrow O_{M_6} \rightarrow O_{M_7} \rightarrow O_{M_8} \rightarrow O_{M_9} \rightarrow O_{M_{10}} \rightarrow X_2$$

3.3 Componentele arhitecturale

Lanțul de aprovizionare pentru miere ecologică cuprinde mai mulți membri, din diverse domenii de activitate precum producători, procesatori, transportatori, clienți. Toți acești membri trebuie să interacționeze după reguli stabilite de la început pentru întregul lanț de aprovizionare, fără conflicte, cu pașii de urmat foarte bine definiți. Arhitectura întregului sistem este formată din următoarele componente principale:

1. *Rețeaua Ethereum* – unde contractele inteligente ale aplicației dezvoltate scrise cu ajutorul limbajului dedicat, Solidity, sunt implementate.
2. *Business Logic* – partea care conține logica aplicației, contractele inteligente, interfețele între membri, activitatea fiecărui membru, pe scurt, modul în care lanțul de aprovizionare funcționează ca un sistem unitar.
3. *Aplicația client* – poate fi mediul de dezvoltare online, Remix IDE, fiind utilizat pentru întreaga dezvoltare a contractelor inteligente (scriere, implementare pe blockchain-ul Ethereum, execuție, interogare pentru datele deja scrise în registru). Cealaltă variantă pentru aplicația client este consola Truffle în care poate fi scris cod JavaScript și astfel sunt executate contractele inteligente, practic face același lucru ca Remix IDE, însă utilizând o consolă text.



Figură 13: Arhitectura de sistem

Lanțul de aprovizionare ca sistem unitar implică mulți actori și multe activități. Toate acestea sunt prezentate în diagrama de mai jos (Figura 14). Actorii sistemului proiectat pentru mierea ecologică sunt următorii:

1. *Primăria.*

- ✓ Emite un certificat de producător valid pentru un producător care solicită un astfel de document.
- ✓ Gestionează numărul total de apicultori înregistrați la o anumită Primărie.

2. *Stupina.*

- ✓ Poate solicita un certificat de producător de la Primărie pentru a putea comercializa mierea mai departe către centre autorizate de colectare.
- ✓ Vinde mierea ecologică mai departe în lanțul de aprovizionare, către centrul de colectare.

3. *Centrul de achiziție.*

- ✓ Gestionează toate cantitățile de miere primite de la apicultorii certificați.
- ✓ Trimite o cantitate mică de miere de la fiecare apicultor către laboratorul de analize, care stabilește dacă mierea este ecologică sau nu.
- ✓ Analizează rezultatele primite de la membrul specializat (laboratorul).
- ✓ În funcție de rezultatele analizelor, centrul de achiziție poate lua decizia de a trimite mierea înapoi către apicultor dacă analizele au indicat că respectiva cantitate de miere nu are proprietățile necesare.
- ✓ În funcție de rezultatele analizelor, centrul de achiziție poate trimite întreaga cantitate de miere către procesator, dacă analizele au indicat că respectiva cantitate de miere are proprietățile necesare.

4. *Laboratorul.*

- ✓ Analizează cantitățile de miere primite de la centrul de achiziție.
- ✓ Returnează către centrul de achiziție un raport detaliat cu proprietățile necesare mierii ecologice.

5. *Procesatorul.* (Centrul de procesare)

- ✓ Combină mierea de același fel primită de la centrul de achiziție cu scopul de a împărți respectiva cantitate în loturi.
- ✓ Emite rapoarte cu privire la condițiile de păstrare, filtrare, împachetare, calitate, disponibile oricărei entități care dorește să le verifice.
- ✓ Pregătește loturile pentru distribuire mai departe în lanțul de aprovizionare. Mierea poate fi trimisă direct către un transportator autorizat sau poate fi depozitată local în depozitul procesatorului pentru un timp maxim de 48h. În cazul unui transport direct, transportatorul autorizat trebuie să emită o notă de livrare. În cazul depozitării locale, centrul de procesare trebuie să emită un raport cu privire la condițiile de păstrare ale mierii pentru toată perioada.

6. *Compania farmaceutică.*

- ✓ Primește loturile de miere de la transportator alături de o notă de livrare pentru respectiva cantitate de miere ecologică.

7. *Depozitul magazinului.*

- ✓ Primește loturile de miere de la transportator alături de o notă de livrare pentru respectiva cantitate de miere ecologică.

8. *Magazinul.*

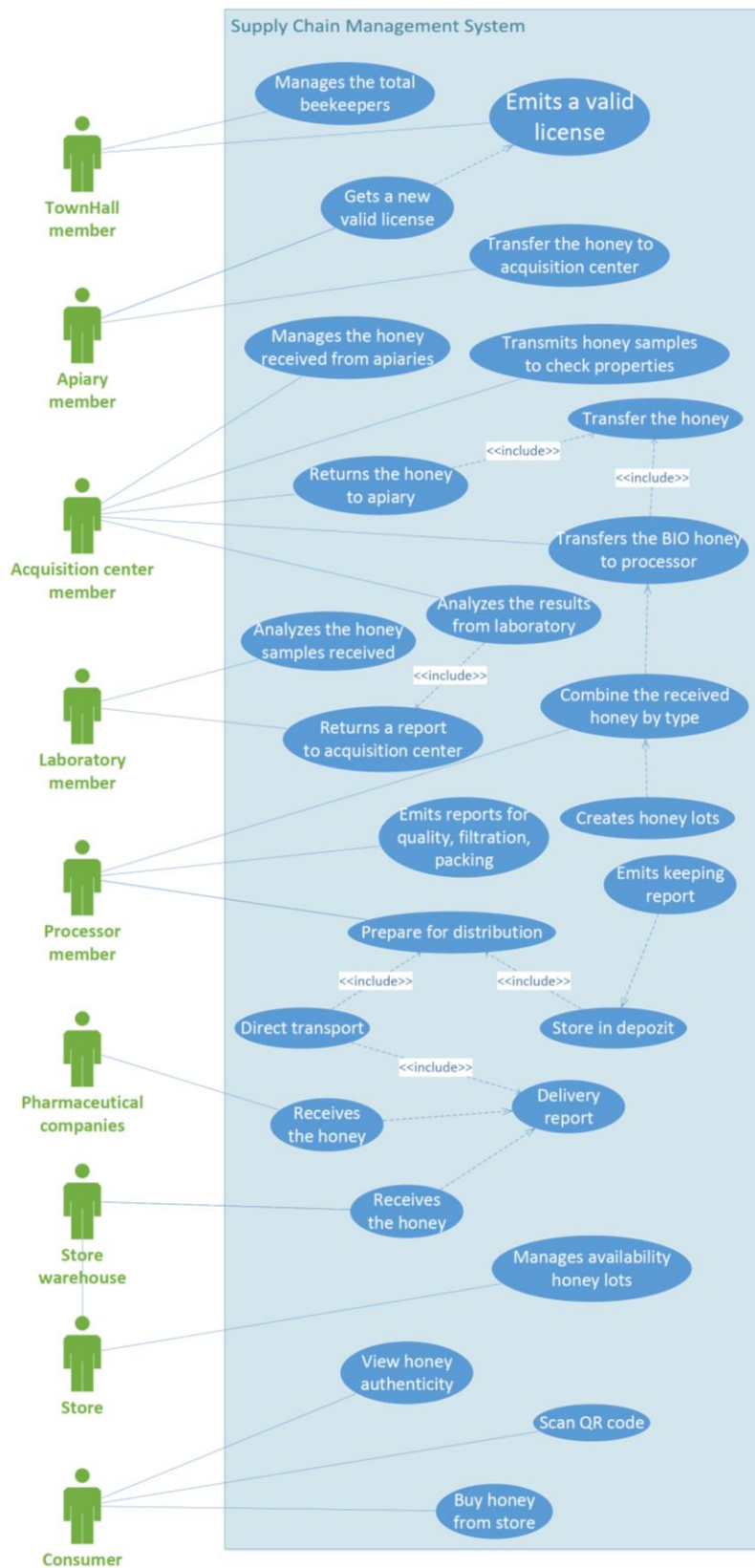
- ✓ Gestionează cantitatea de miere disponibilă (în loturi).

9. *Consumatorul.*

- ✓ Poate scana un cod QR care dezvăluie toate informațiile relevante de la stupină până în magazin pentru borcanul de miere dorit.
- ✓ Datorită proiectării lanțului de aprovizionare cu ajutorul blockchain-ului Ethereum (Figura 27), consumatorul are opțiunea de a urmări toți pașii anteriori pentru mierea ecologică pe care dorește să o cumpere și să verifice autenticitatea informațiilor furnizate direct de fiecare membru în parte.
- ✓ Dacă toate informațiile citite în urma scanării codului QR de pe eticheta borcanului de miere ecologică au satisfăcut cerințele consumatorului, acesta poate alege să achiziționeze respectiva cantitate de miere ecologică.

Supply chain management for ecological honey

- Actors & Use cases -



Figură 14: Diagrama cazurilor de utilizare

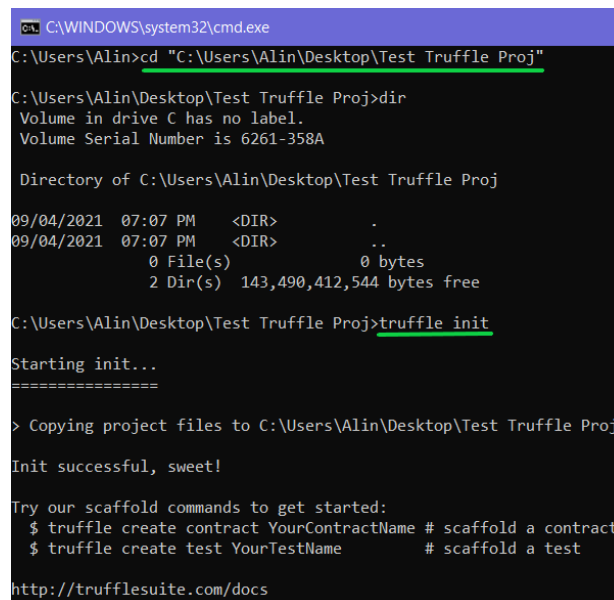
3.4 Implementare

Implementarea presupune parcurgerea unor noțiuni generale precum structura unui proiect *Truffle*, apoi structura proiectului BioTrack și în final analiza schemei de mapare design-contract.

3.4.1 Structura unui proiect *Truffle* generat

Structura implementării urmează diagrama de proces a lanțului de aprovizionare (Figura 8). Proiectul inițial a fost creat cu suita *Truffle*, utilizând câteva comenzi (Figura 15):

- Am creat directorul proiectului și am intrat în locația respectivă cu comanda *cd*.
- Am instalat modulele Truffle prin comanda: *npm install truffle --save-dev*. De asemenea se poate verifica versiunea de truffle instalată astfel: *truffle version*.
- Am rulat scriptul de inițializare pentru un proiect nou: *truffle init*.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\Alin>cd "C:\Users\Alin\Desktop\Test Truffle Proj"

C:\Users\Alin\Desktop\Test Truffle Proj>dir
Volume in drive C has no label.
Volume Serial Number is 6261-358A

Directory of C:\Users\Alin\Desktop\Test Truffle Proj

09/04/2021  07:07 PM    <DIR>        .
09/04/2021  07:07 PM    <DIR>        ..
               0 File(s)                0 bytes
               2 Dir(s)  143,490,412,544 bytes free

C:\Users\Alin\Desktop\Test Truffle Proj>truffle init

Starting init...
=====
> Copying project files to C:\Users\Alin\Desktop\Test Truffle Proj

Init successful, sweet!

Try our scaffold commands to get started:
$ truffle create contract YourContractName # scaffold a contract
$ truffle create test YourTestName        # scaffold a test

http://trufflesuite.com/docs
```

Figură 15: Exemplu de execuție pentru crearea unui proiect Truffle

După ultimul pas, scriptul a creat o structură de proiect cu mai multe directoare ca în Figura 16. Au fost create directoarele *contracts/* (unde stocăm toate fișierele Solidity cu extensia *.sol*), *migrations/* (fișiere JavaScript pentru implementarea contractelor în blockchain), *test/* (fișiere *.js* sau *.sol* care conțin teste pentru contracte) și fișierele *Migrations.sol* (un contract utilizat de truffle pentru a se asigura că procesul de implementare pe blockchain se desfășoară în ordinea corectă), *1_initial_migrations.js* (scriptul care implementează contractul de mai sus în rețeaua blockchain) și *truffle-config.js* (principalul fișier de configurare al proiectului truffle unde se pot defini rețelele pe care rulează contractele, *gas*-ul utilizat, adrese, etc).


```

Directory of C:\Users\Alin\Desktop\Test Truffle Proj
09/04/2021 07:08 PM <DIR>      .
09/04/2021 07:08 PM <DIR>      ..
09/04/2021 07:08 PM <DIR>      contracts
09/04/2021 07:08 PM <DIR>      migrations
09/04/2021 07:08 PM <DIR>      test
07/29/2021 12:40 AM          4,560 truffle-config.js
1 File(s)          4,560 bytes
5 Dir(s) 143,494,340,608 bytes free

```

Figură 16: Structura proiectului Truffle creat inițial

După această operațiune putem executa comanda *truffle compile* pentru a verifica dacă totul rulează corect, iar după execuție se generează fișierul *Migrations.json* în locația *build/contracts/* care conține foarte multe informații despre contractul compilat.

3.4.2 Structura proiectului BioTrack

```

build
├── contracts
│   ├── AcquisitionCenter.json
│   ├── Apiary.json
│   ├── IAcquisitionCenter.json
│   ├── IApiary.json
│   ├── ILaboratory.json
│   ├── ITownHall.json
│   ├── Laboratory.json
│   ├── Migrations.json
│   ├── PharmaceuticalAndBeauty.json
│   ├── Processor.json
│   ├── ProcessorStorage.json
│   ├── SaleUnit.json
│   ├── TownHall.json
│   ├── Transporter.json
│   └── WarehouseUnit.json
├── contracts
│   ├── artifacts
│   │   ├── AcquisitionCenter.sol
│   │   ├── Apiary.sol
│   │   ├── Laboratory.sol
│   │   ├── Migrations.sol
│   │   ├── PharmaceuticalAndBeauty.sol
│   │   ├── Processor.sol
│   │   ├── ProcessorStorage.sol
│   │   ├── SaleUnit.sol
│   │   ├── TownHall.sol
│   │   ├── Transporter.sol
│   │   └── WarehouseUnit.sol
│   ├── interfaces
│   │   ├── artifacts
│   │   │   ├── IAcquisitionCenter.sol
│   │   │   ├── IApiary.sol
│   │   │   ├── ILaboratory.sol
│   │   │   ├── IProcessor.sol
│   │   │   └── ITownHall.sol
│   └── migrations
│       ├── 1_initial_migration.js
│       └── 2_contracts_migration.js
├── node_modules
├── react-dapp
├── test
├── my-script.js
├── package-lock.json
├── package.json
└── truffle-config.js

```

Figură 17: Structura proiectului BioTrack

Dacă analizăm structura proiectului BioTrack (Figura 17), observăm directorul de *build/contracts* în care avem câte un json pentru fiecare contract inteligent. În directorul *contracts/* avem codul sursă pentru fiecare contract inteligent, adică logica de business pentru fiecare membru din lanțul de aprovizionare. În subdirectorul *contracts/artifacts/* sunt mai multe fișiere json cu multă informație generată în urma implementării pe blockchain precum și fișiere json cu metadata. Am creat eu un director separat pentru interfețe numit *interfaces/* unde regăsim codul sursă pentru interfețele necesare contractelor. De asemenea, și directorul *interfaces/* are un subdirector numit *artifacts/* ce conține aceleași tipuri de fișiere ca subdirectorul *artifacts/* din *contracts/*.

În directorul *migrations/* găsim fișierul *2_contracts_migration.js* care conține codul JavaScript ce realizează implementarea contractelor proiectului pe blockchain (Figura 18).



```
1 const AcquisitionCenter = artifacts.require("../contracts/AcquisitionCenter.sol");
2 const Apiary = artifacts.require("../contracts/Apiary.sol");
3 const Laboratory = artifacts.require("../contracts/Laboratory.sol");
4 const PharmaceuticalAndBeauty = artifacts.require("../contracts/PharmaceuticalAndBeauty.sol");
5 const Processor = artifacts.require("../contracts/Processor.sol");
6 const ProcessorStorage = artifacts.require("../contracts/ProcessorStorage.sol");
7 const SaleUnit = artifacts.require("../contracts/SaleUnit.sol");
8 const TownHall = artifacts.require("../contracts/TownHall.sol");
9 const Transporter = artifacts.require("../contracts/Transporter.sol");
10 const WarehouseUnit = artifacts.require("../contracts/WarehouseUnit.sol");
11
12
13 module.exports = function(deployer, network, accounts) {
14   deployer.deploy(AcquisitionCenter, {from: accounts[0]});
15   deployer.deploy(Apiary, {from: accounts[1]});
16   deployer.deploy(Laboratory, {from: accounts[2]});
17   deployer.deploy(PharmaceuticalAndBeauty, {from: accounts[3]});
18   deployer.deploy(Processor, {from: accounts[4]});
19   deployer.deploy(ProcessorStorage, {from: accounts[5]});
20   deployer.deploy(SaleUnit, {from: accounts[6]});
21   deployer.deploy(TownHall, {from: accounts[7]});
22   deployer.deploy(Transporter, {from: accounts[8]});
23   deployer.deploy(WarehouseUnit, {from: accounts[9]});
24 }
```

Figură 18: Codul JavaScript pentru *2_contracts_migration.js*

În directorul *node_modules/* sunt salvate toate pachetele din NPM necesare proiectului pentru a rula. Directorul *react-dapp/* conține șablonul pentru o viitoare interfață web cu fiecare membru din lanțul de aprovizionare. Directorul *test/* nu conține momentan niciun fișier, însă va conține fișiere .js sau .sol cu teste scrise pentru contractele inteligente. Fișierul *package.json* conține principalele caracteristici ale proiectului, cum ar fi versiunea de compilator Solidity utilizată, dependențe, licențe și multe alte informații relative la proiect. Fișierul *truffle-config.js* reprezintă principala sursă de configurare pentru proiectul truffle unde se pot defini rețelele pe care rulează contractele, gas-ul utilizat, adrese, porturi, etc. În proiect am definite 2 rețele după cum se observă în Figura 19, o rețea pe portul 7545 denumită *development*, clasică și încă o rețea pe portul 8545, denumită *develop*, având câteva opțiuni configurabile cum ar fi numărul de conturi, suma de Ether de la început în fiecare cont și altele.

```

32 networks: {
33   // Useful for testing. The `development` name is special - truffle uses it by default
34   // if it's defined here and no other network is specified at the command line.
35   // You should run a client (like ganache-cli, geth or parity) in a separate terminal
36   // tab if you use this network and you must also set the `host`, `port` and `network_id`
37   // options below to some value.
38   //
39   development: {
40     host: "127.0.0.1",      // Localhost (default: none)
41     port: 7545,            // Standard Ethereum port (default: none)
42     network_id: "*",       // Any network (default: none)
43   },
44   develop: {
45     port: 8545,
46     network_id: 20,
47     accounts: 20,
48     defaultEtherBalance: 5000,
49     blockTime: 3
50   }

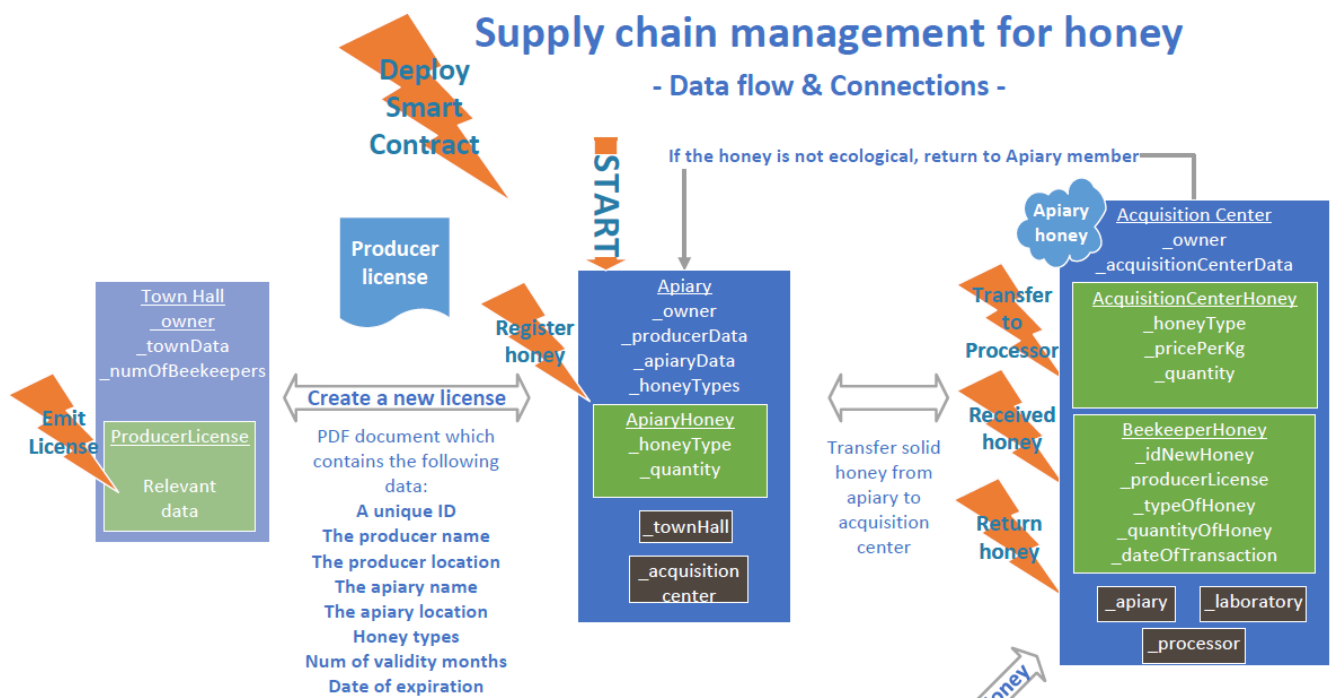
```

Figură 19: O secvență din fișierul de configurare config-truffle.js

Mai sus am descris pe scurt elementele unui proiect Truffle generat și tot ce regăsim în proiectul BioTrack în plus față de structura generată pentru a avea o înțelegere de ansamblu a structurii acestuia.

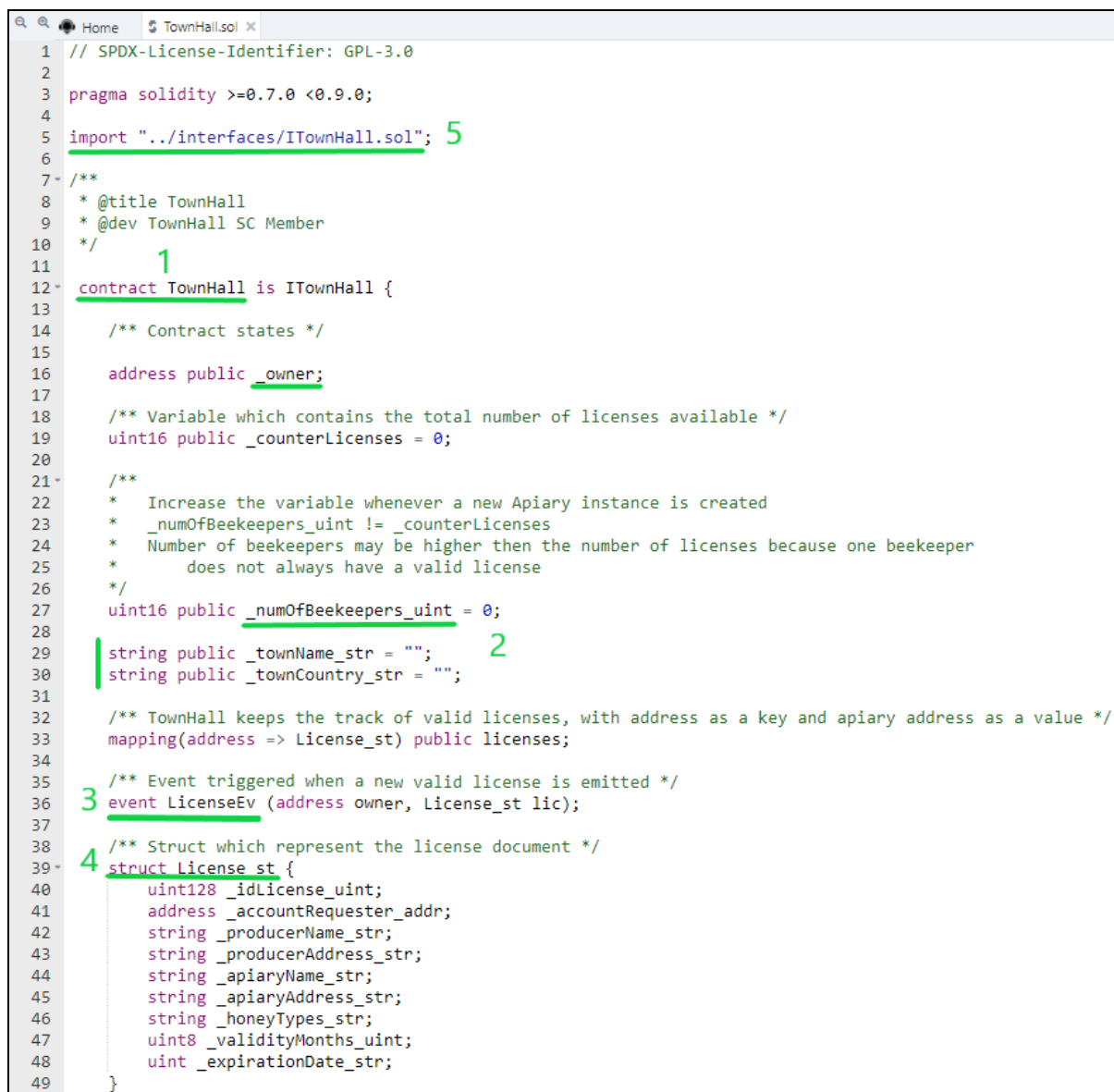
3.4.3 Schema de mapare *design-contract inteligent*

În Figura 20 sunt prezentați primii 3 membri din lanțul de aprovizionare, din punctul meu de vedere fiind foarte importanți deoarece ei sunt sursa, entitatea care certifică producătorul dar și centrul de achiziție al mierii ecologice, toate acestea aflându-se la începutul lanțului. Foarte important în verificarea calității mierii este *Laboratorul*, care decide pentru o cantitate de miere dacă este ecologică sau nu, deci are un impact major în desfășurarea proceselor ulterioare.



Figură 20: Diagrama de proces pentru lanțul de aprovizionare specializat pe miere ecologică (început)

Revenind la membri de la început, pentru exemplificare luăm componenta **TownHall** (*Primăria*). Observăm un dreptunghi de culoare albastră (puțin decolorată față de celelalte componente) ce reprezintă întregul contract inteligent care ține logica membrului (Figura 21, 1 cu verde). Culoarea diferă deoarece *Primăria*, ca membru din lanțul de aprovizionare, poate să nu intervină de fiecare dată când un apicultor dorește să vândă miere ecologică, ci doar atunci când are nevoie de emiterea unui certificat de producător. De-a lungul lanțului mai există un membru cu această caracteristică (Figura 25) – *Processor Storage* (Depozitul procesatorului). Mierea organizată în loturi de către *Processor* (Procesator) poate fi trimisă direct la *Transport* (Transportator) caz în care depozitul procesatorului nu intervine sau poate fi trimisă în depozit pentru maxim 24h, în această situație având o contribuție activă în lanțul de aprovizionare.



```

1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4
5 import "../interfaces/ITownHall.sol"; 5
6
7 /**
8  * @title TownHall
9  * @dev TownHall SC Member
10  */
11
12 1 contract TownHall is ITownHall {
13
14      /** Contract states */
15
16      address public _owner;
17
18      /** Variable which contains the total number of licenses available */
19      uint16 public _counterLicenses = 0;
20
21      /**
22       * Increase the variable whenever a new Apiary instance is created
23       * _numOfBeekeepers_uint != _counterLicenses
24       * Number of beekeepers may be higher then the number of licenses because one beekeeper
25       * does not always have a valid license
26       */
27      uint16 public _numOfBeekeepers_uint = 0;
28
29      2 string public _townName_str = "";
30      string public _townCountry_str = "";
31
32      /** TownHall keeps the track of valid licenses, with address as a key and apiary address as a value */
33      mapping(address => License_st) public licenses;
34
35      /** Event triggered when a new valid license is emitted */
36      3 event LicenseEv (address owner, License_st lic);
37
38      /** Struct which represent the license document */
39      4 struct License_st {
40          uint128 _idLicense_uint;
41          address _accountRequester_addr;
42          string _producerName_str;
43          string _producerAddress_str;
44          string _apiaryName_str;
45          string _apiaryAddress_str;
46          string _honeyTypes_str;
47          uint8 _validityMonths_uint;
48          uint _expirationDate_str;
49      }

```

Figură 21: Codul Solidity obținut după transpunerea design-ului pentru TownHall

Datele scrise în interiorul dreptunghiului reprezintă date cheie, generale, care definesc membrul. Acestea sunt date publice, accesibile tuturor și cuprind trăsături legate de proprietarul contractului inteligent (creatorul – o adresă hash de 256 biți), adrese și denumiri ale localității de care aparține *Primăria* dar și numărul total de apicultori dintr-o localitate (Figura 21, 2 cu verde). Tot în interiorul dreptunghiului există un dreptunghi mai mic, de culoare verde, ce se transpune mai departe în cod sub forma unei structuri (Figura 21, 4 cu verde).

Simbolul din diagramă asemănător unui *fulger* reprezintă în codul contractului un eveniment care este declanșat după ce o licență validă a fost emisă pentru un apicultor solicitant (Figura 21, 3 cu verde). De asemenea, observăm la începutul contractului inteligent un *import* (Figura 21, 5 cu verde), pentru o interfață. Interfața permite modificarea numărului de apicultori din localitate, funcția ce realizează această operațiune fiind parte din contractul *Primăriei*, dar în momentul apariției unui apicultor nou, această funcție poate fi apelată extern, direct din contractul *Stupinei*.

```

54  /** This modifier restrict the access to the creator of the smart contract (the owner) */
55  1  modifier restricted() {
56      require (
57          msg.sender == _owner,
58          "This function is restricted to the contract's owner"
59      );
60      _;
61  }
62
63  /** When a new instance is deployed, the owner address get the sender address */
64  2  constructor () {
65      _owner = msg.sender;
66      /* Perform some operations */
67  }
68
69  /** */
70  3  function UpdateTownHallLocation(
71      string memory townName_str,
72      string memory townCountry_str
73  )
74      public
75      restricted {
76
77          _townName_str = townName_str;
78          _townCountry_str = townCountry_str;
79  }

```

Figură 22: Modificatorul de acces, constructorul și o funcție din contractul inteligent TownHall

Mai departe observăm cum accesul la anumite resurse poate fi restricționat, fiind permise acțiuni doar unor entități stabilite de la început, în cazul nostru pentru modificatorul de acces *restricted* (Figura 22, 1 cu verde), funcțiile care îl utilizează pot fi apelate doar de proprietarul (creatorul) contractului inteligent. Am creat acest modificator de acces pentru a evita suprascrierea din greșeală sau cu intenție a unor date interne contractului, și am permis ca modificarea acestor date importante să fie realizată doar de organizația/entitatea/contul cu drepturi depline. Dacă un cont fără drepturi

încearcă să apeleze o funcție cu accesul restricționat, acesta primește înapoi mesajul de eroare „*This function is restricted to the contract's owner*“.

Din punctul de vedere al programatorului, eu „văd” un contract ca o clasă, cu date membre (numite stări ale contractului) și metode (pot fi publice, private, externe). Prin analogie cu o clasă care trebuie să fie instanțiată sub forma unor obiecte pentru a fi utilizată, așa regăsim și în Solidity, unde, pentru a exista un contract, acesta trebuie instanțiat. Această instanțiere este realizată printr-un *constructor* (Figura 22, 2 cu verde) și inițializează (de cele mai multe ori) stări globale ale contractului inteligent.

Funcțiile reprezintă o caracteristică foarte importantă pentru un contract inteligent, mai exact acțiunile desfășurate de contract, intern prin funcții private sau extern prin funcții publice. În contractul *Primăriei* există funcția publică *UpdateTownHallLocation(...)*, cu accesul restricționat pentru proprietar, având doi parametri, numele localității și țării din care face parte membrul. Astfel avem o acțiune permisă doar *Primăriei*, prin care aceasta își actualizează datele de identificare, adică stările interne ale contractului (Figura 22, 3 cu verde).

```
121  /** When a new Apiary is created, increase the number of total beekeeper in TownHall registers */
122  function UpdateTotalBeekeepers ()
123      external
124      override {
125
126      _numOfBeekeepers_uint += 1;
127  }
```

Figură 23: Exemplu de funcție externă

În Figura 23 am prezentat un exemplu de funcție externă, declarată în interfața contractului (Figura 24) și definită în cadrul contractului. Aceasta poate fi apelată de *Stupină* în momentul când un nou apicultor apare în localitate și astfel se incrementează variabila contor.

```
1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity >=0.7.0 <0.9.0;
4
5  interface ITownHall {
6      function EmitLicense(
7          address _accountRequester,
8          string memory _producerName,
9          string memory _producerAddress,
10         string memory _apiaryName,
11         string memory _apiaryAddress,
12         string memory _honeyTypes) external;
13
14         function UpdateTotalBeekeepers () external;
15     }
```

Figură 24: Interfața pentru contractul TownHall

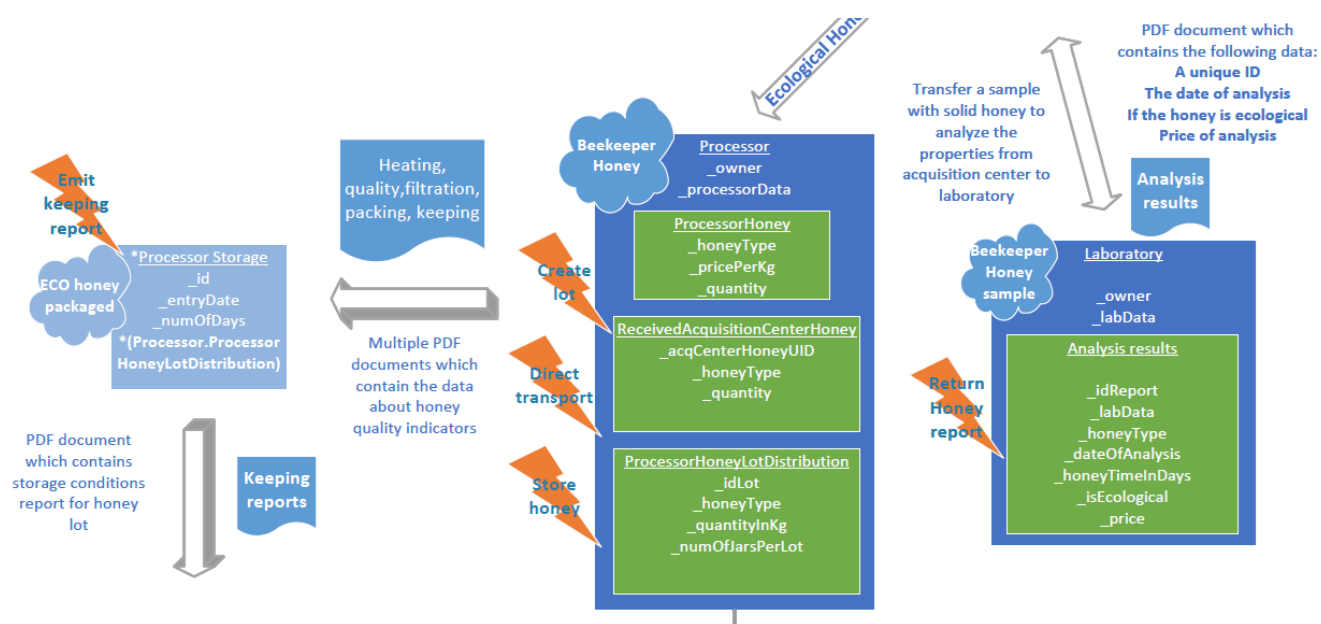
În acest mod se realizează transpunerea din design în cod Solidity pentru un contract inteligent. În continuare prezint caracteristicile importante ale contractelor membrilor, în cod existând adăugate variabile și metode suplimentare de la caz la caz, dar care pot fi analizate în repository-ul de GitHub al proiectului.

În Figura 20, observăm **Stupina**, un membru cheie pentru lanțul de aprovizionare deoarece reprezintă sursa în jurul căreia se dezvoltă întregul proces. Avem contractul Solidity numit *Apiary.sol* cu interfața *IApiary.sol*. Datele esențiale contractului sunt proprietarul acestuia, date despre producător (nume, locație), date despre stupină (denumire, locație) și două obiecte, instanțe ale contractelor *TownHall* și *Acquisition Center*. De asemenea, contractul conține o structură foarte importantă ce reține tipul de miere și cantitatea per tip în care fiecare stupina își ține evidența. Evenimentul caracteristic *Stupinei* este înregistrarea mierii, eveniment declanșat în momentul când proprietarul stupinei adaugă o nouă cantitate de miere ecologică, fiind disponibilă pentru vânzare. *Stupina* interacționează cu *Primăria* pentru a solicita un certificat de producător și cu *Centrul de achiziție* pentru a vinde mierea ecologică sau în cazul nefericit când mierea nu îndeplinește criteriile necesare, aceasta primește mierea returnată de *Centrul de achiziție*.

Centrul de achiziție este un membru cu multe responsabilități. Avem contractul *AcquisitionCenter.sol* cu interfața *IAcquisitionCenter.sol*. Datele esențiale contractului sunt proprietarul acestuia și datele despre centrul de achiziție (denumire, locație) precum și trei obiecte, instanțe ale contractelor *Apiary*, *Laboratory* și *Processor*. Contractul inteligent conține două structuri foarte importante, prima ce gestionează mierea centrului de achiziție și cea de-a doua care gestionează mierea primită de la apicultori. Prima structură poate reține tipul de miere, prețul per kilogram și cantitatea de miere aflată deja în gestiunea centrului de achiziție. A doua structură poate reține un identificator pentru noua cantitate de miere din centrul de achiziție, un identificator pentru mierea ajunsă de la o stupină, contul stupinei din care a ajuns mierea, tipul și cantitatea mierii, dacă mierea este sau nu ecologică și data când s-a realizat tranzacția. Evenimentele caracteristice *Centrului de achiziție* sunt primirea mierii de la *Stupină*, transferul mierii ecologice mai departe în lanț către *Procesator* și returul mierii ne-ecologice către *Stupina* furnizoare (decizie luată în urma analizelor trimise de *Laborator*). *Centrul de achiziție* interacționează ca receptor între acesta și *Stupină* când primește mierea, ca emițător când returnează mierea ne-ecologică, în ambele sensuri cu *Laboratorul*, când trimite o mostră de miere pentru analize și când primește raportul cu analizele mostrei de miere și ca emițător când trimite mierea ecologică mai departe în lanțul de aprovizionare către *Procesator*.

Laboratorul este membrul care decide pentru o cantitate de miere dacă este sau nu ecologică. Avem contractul *Laboratory.sol* cu interfața *ILaboratory.sol*. Datele esențiale contractului sunt proprietarul acestuia și datele despre laborator (denumire, locație). Contractul conține o structură

importantă care poate reține un identificator pentru analiza realizată, contul membrului solicitant, tipul de miere, numele și locația laboratorului care a realizat analiza, data analizei, dacă mierea este sau nu ecologică și prețul analizei. Am ales să rețin un identificator pentru ca fiecare analiză să fie unică și contul membrului solicitant pentru transparența relației centru de achiziție – laborator de analize. Evenimentul declanșat de *Laborator* este întoarcerea raportului cu analizele cantității de miere către membrul solicitant. *Laboratorul* interacționează doar cu *Centrul de achiziție* și ca receptor (primește de la acesta mostra de miere pentru analiză) și emițător (întoarce către acesta un raport privind calitatea mierii analizate).



Figură 25: Diagrama de proces pentru lanțul de aprovizionare specializat pe miere ecologică, continuare

Procesatorul este membrul care este responsabil de colectarea mierii de la centrele de achiziție, combinarea mai multor cantități de miere de același tip și împărțirea în loturi de miere care ulterior vor fi transportate direct spre alți membri din lanțul de aprovizionare sau păstrate cel mult 48h în *Depozitul procesatorului* și apoi livrate mai departe. Avem contractul inteligent *Processor.sol* și interfața *IProcessor.sol*. Datele esențiale contractului sunt proprietarul acestuia și datele despre procesator (denumire, locație). Contractul conține 3 structuri importante pentru logica de business, prima ce poate reține mierea de la *Procesator* prin tipul de miere, prețul per kilogram și cantitatea. A doua pentru a reține mierea primită de la *Centrul de achiziție* printr-un identificator al cantității de miere, tipul de miere și cantitatea. Cea de-a treia structură gestionează loturile de miere formate prin reținerea identificatorului de lot, o listă de identificatori pentru toate cantitățile de miere care s-au combinat anterior și din care a rezultat lotul curent, tipul și cantitatea de miere și un câmp cantitativ care conține numărul de borcane dintr-un lot. Dacă numărul este 0, înseamnă că lotul respectiv nu a fost împărțit în borcane. De asemenea, *Procesatorul* trebuie să emită un raport care să conțină un

număr unic de identificare raport, număr unic de identificare lot miere, condițiile de încălzire, filtrare, prelucrare, împachetare și data eliberării. Evenimentul declanșat de *Procesator* este crearea unui lot de miere după o pregătire ulterioară. *Procesatorul* interacționează cu *Centrul de achiziție* (ca receptor pentru mierea ecologică) și cu *Transportatorul* sau *Depozitul* acestuia, după caz, în rolul de emițător.

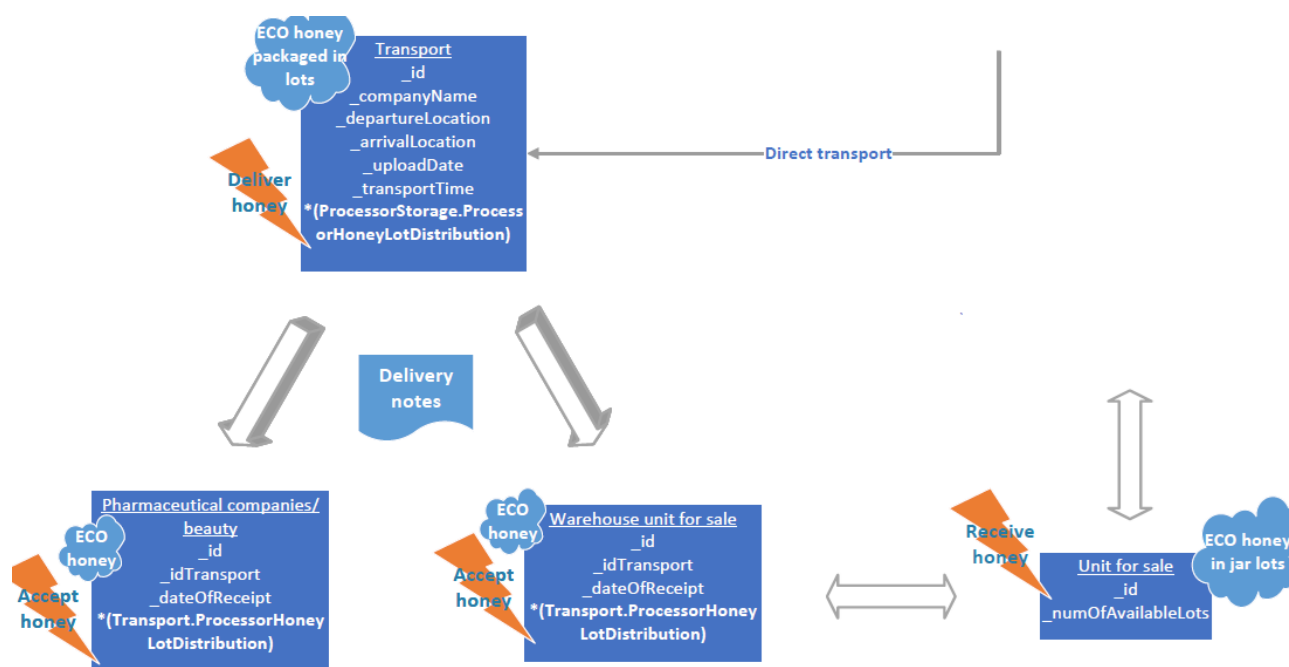
Depozitul procesatorului este un membru care poate să nu apară în anumite cazuri, pentru anumite loturi de miere. Decizia este luată de *Procesator* care poate trimite direct mierea la transportator sau poate să depoziteze mierea un timp limitat. Avem contractul inteligent *ProcessorStorage.sol* ce reține câteva date esențiale lanțului de aprovizionare precum un identificator pentru noua înregistrare de miere depozitată, data de intrare în depozit a mierii, numărul de zile în care mierea a stat în depozit (0, 1 sau 2 zile). Acest membru trebuie să emită un raport care să conțină un număr de identificare unic pentru raport, un număr unic de identificare pentru lotul de miere depozitat, condițiile de păstrare și data eliberării. Evenimentul declanșat de membru este emiterea raportului de păstrare în depozit. Interacțiunea *Depozitului* este dinspre *Procesator* dacă acesta decide să păstreze un timp mierea și spre *Transportator* pentru a transmite mierea ecologică mai departe în lanțul de aprovizionare.

În Figura 26 observăm ***Transportatorul*** care este reprezentat prin contractul inteligent *Transporter.sol*. Datele esențiale conținute de acesta sunt adresa proprietarului contractului și numele companiei de transport. Primește de la *Procesator* o cantitate (un număr de loturi de miere) pe care să o transporte mai departe fie către o companie farmaceutică fie spre depozitul unui magazin. *Transportatorul* trebuie să emită un raport (o notă de livrare) care să conțină un număr unic de identificare pentru transportul realizat, o listă cu loturile de miere transportate (cunoscând tipul de miere), locația de plecare, destinația, data încărcării, data descărcării, condițiile de transport și prețul transportului. Evenimentul declanșat de membru reprezintă momentul când *Transportatorul* face livrarea cu o anumită cantitate de miere către una din destinații.

Mai departe discutăm despre ***companiile farmaceutice*** sau cele pentru ***produse de cosmetică*** care primesc loturile de miere de la *Transportator* (loturile sunt primite direct în recipiente mari, nu în borcane). Contractul inteligent este *PharmaceuticalAndBeauty.sol* și conține date precum contul proprietarului acestuia și date despre companie (denumire, adresă) dar și date pentru trasabilitate și transparență cum ar fi numărul de identificare al transportului și data sosirii mierii. Evenimentul declanșat de membru reprezintă acceptul cantității de miere primită de la *Transportator*.

Depozitul magazinului poate fi a doua destinație a *Transportatorului*. Avem contractul inteligent *WarehouseUnit.sol* ce reține date despre contul proprietarului acestuia și date pentru trasabilitate și transparență cum ar fi numărul de identificare al transportului și data sosirii mierii la fel ca în cazul companiilor farmaceutice și de cosmetică. Acest membru trebuie să emită un raport care să

conțină un număr unic de identificare raport, o listă cu numerele unice de identificare ale loturilor de miere, durata de păstrare a mierii în depozit, data de intrare în depozit precum și informații despre condițiile de păstrare pe toată perioada. Există o permanentă interacțiune între *Depozit* și *Magazin* pe tot parcursul procesului de vânzare. Astfel introducem ultimul membru înainte de cumpărătorul final, **Magazinul**, ce reprezintă punctul de acces din care clientul cumpără mierea ecologică. Contractul inteligent *SaleUnit.sol* reține informații despre contul proprietarului acestuia, date despre magazin (denumire, locație) și numărul de loturi disponibile în magazin. Evenimentul declanșat de *Magazin* reprezintă momentul când acesta primește mierea din depozit, sub formă de loturi, urmând să expună borcanele pe rafturi către consumatori. *Magazinul* interacționează în permanență cu *Depozitul* pentru a ști câte loturi de miere mai sunt disponibile și din ce tip. Înainte de a pune pe rafturi anumite loturi de miere, *Magazinul* analizează raportul emis de *Depozit* și după această verificare a calității, produsele ajung pe rafturi (borcanele de miere).

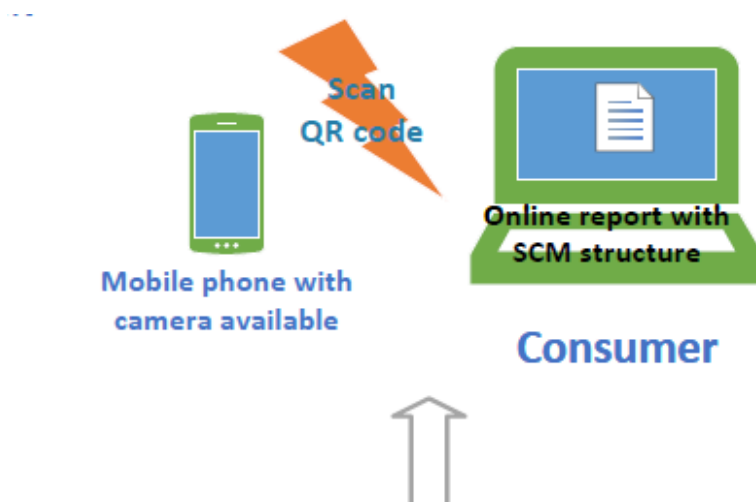


Figură 26: Diagrama de proces pentru lanțul de aprovizionare specializat pe miere ecologică, continuare

Membrul care beneficiază de avantajele unui lanț de aprovizionare bazat pe tehnologia blockchain este **Consumatorul**. Acesta nu are un contract inteligent asignat deoarece nu face parte din logica de business a lanțului, ci tot lanțul a fost proiectat pentru a obține transparența pe care o are disponibilă la momentul achiziției unei cantități de miere. *Consumatorul* final poate accesa istoricul provenienței mierii prin mai multe metode (nu sunt implementate încă, sunt planificate pentru dezvoltarea și extinderea aplicației în viitor). Scanarea codului QR din Figura 27 nu reprezintă un eveniment

asemănător ce alte evenimente anterioare, ci o exprimare a unei acțiuni pe care face *Consumatorul* asupra unui produs pe care dorește să îl achiziționeze.

Diferența apare deoarece evenimentele anterioare aveau drept consecință înregistrarea datelor caracteristice evenimentului în rețeaua blockchain dar acțiunea Consumatorului doar citește date înscrise deja, fără a modifica starea unor variabile. Tot pentru o dezvoltare viitoare, când consumatorul citește informații din blockchain, să se înregistreze acțiunea sa, iar aplicația să genereze grafice pentru utilizator, statistici pentru un panou de administrator cu mulțimea de clienți care a folosit aplicația de vizualizare a provenienței mierii, grafice săptămânale, lunare, semestriale, complet configurabile, care să conțină chiar și produsele scanate. O metodă este prin scanarea unui cod QR scris pe eticheta borcanului de miere care să direcționeze consumatorul către o aplicație și să îi afișeze toate informațiile relevante din lanțul de aprovizionare pentru lotul de miere din care face parte borcanul ales. O altă metodă de verificare este pe site-ul web destinat unui consumator, scrie codul înscris pe eticheta borcanului și primește un răspuns cu toate informațiile relevante cu privire la originea mierii însă cu dezavantajul că poate fi mai greu de verificat. În schimb, cu cealaltă metodă, pornește camera telefonului, scanează codul QR și obține toată informația necesară care îi confirmă că mierea este ecologică.



Figură 27: Diagrama de proces pentru lanțul de aprovizionare specializat pe miere ecologică, continuare

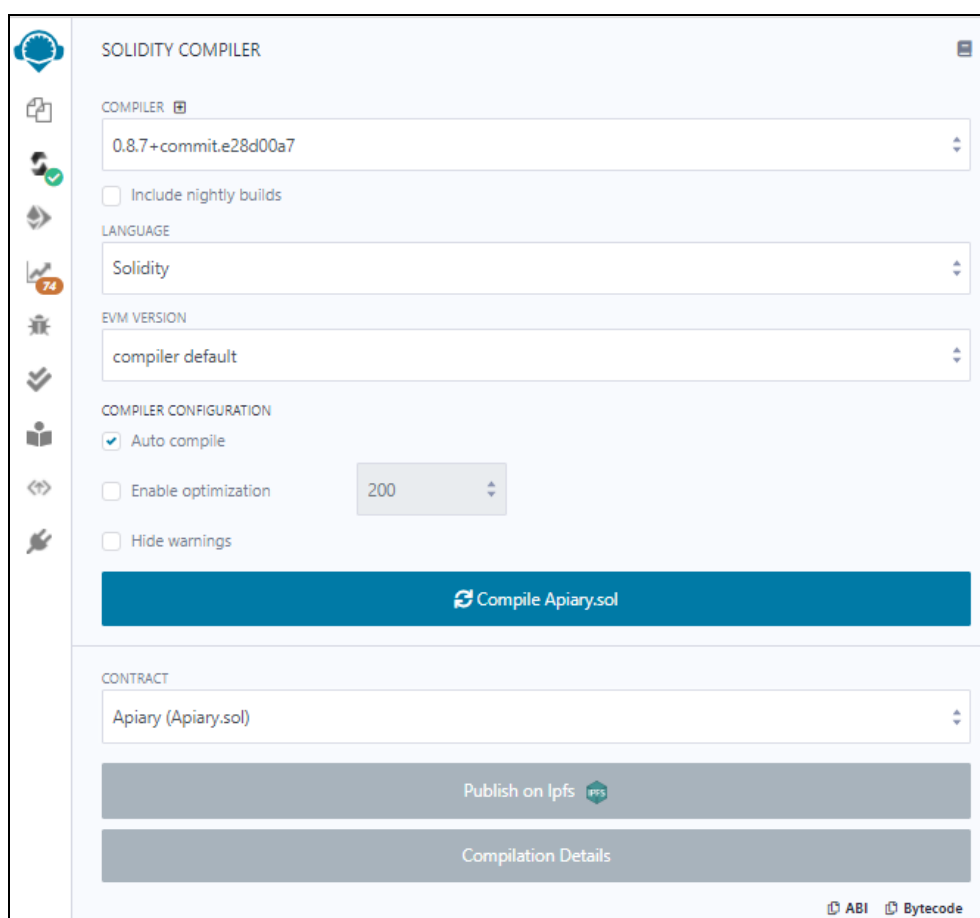
Mai sus am prezentat schema de mapare între design-ul aplicației și codul Solidity, în contextul utilizării tehnologiei blockchain. Am încercat să limitez accesul pentru anumite operațiuni pentru anumiți membri, în limita în care tot sistemul să rămână descentralizat, fiecare membru să nu fie privat, toate informațiile să fie transparente în fiecare punct în care s-ar afla lanțul de aprovizionare.

3.5 Testare și experimente

Am ajuns în momentul special care presupune testarea proiectului dezvoltat. Propun testarea prin două metode, prima, testare direct în mediul de dezvoltare online, Remix IDE, cu operația de implementare pe blockchain, testare cu date valide și observarea rezultatelor și a doua metodă prin intermediul consolei Truffle, cu o rețea blockchain privată, simulată cu ajutorul instrumentului Ganache.

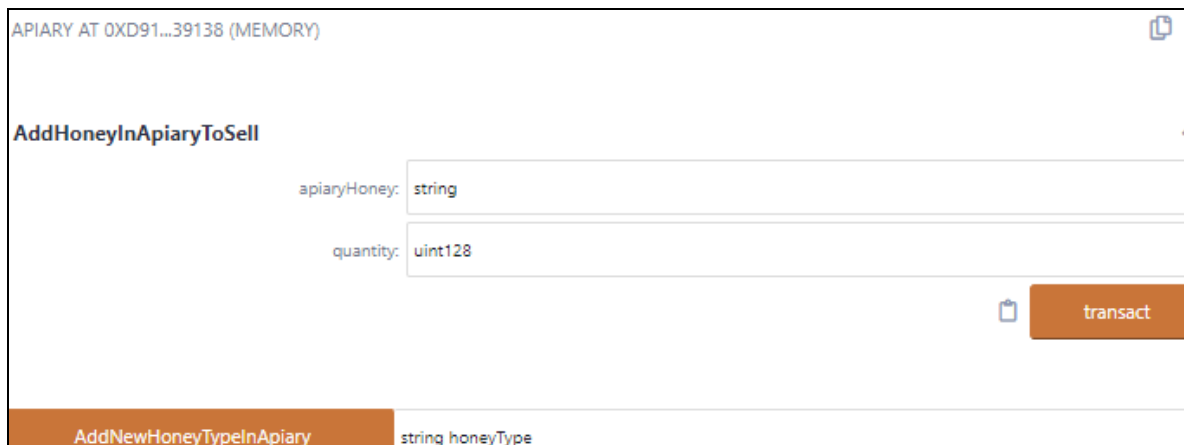
3.5.1 Testare direct în Remix IDE online

Testarea contractelor inteligente este un pas foarte important, orice cod scris trebuie la un moment testat pentru a valida că funcționalitatea dorită este executată cu succes. Exemplific câteva operațiuni importante pentru primele contracte din lanțul de aprovizionare, prin execuție în cadrul mediului de dezvoltare online, Remix IDE. Primul pas după ce am scris contractul este de a compila codul sursă. Observăm selectată ultima versiune de compilator, 0.8.7, limbajul de programare, Solidity, versiunea de mașină virtuală standard. Semnul verde observat pe bara lateral-stânga ne garantează că nu avem erori de compilare și nici *warning*-uri pentru contractul *Apiary*.



Figură 28: Compilarea contractului inteligent Apiary

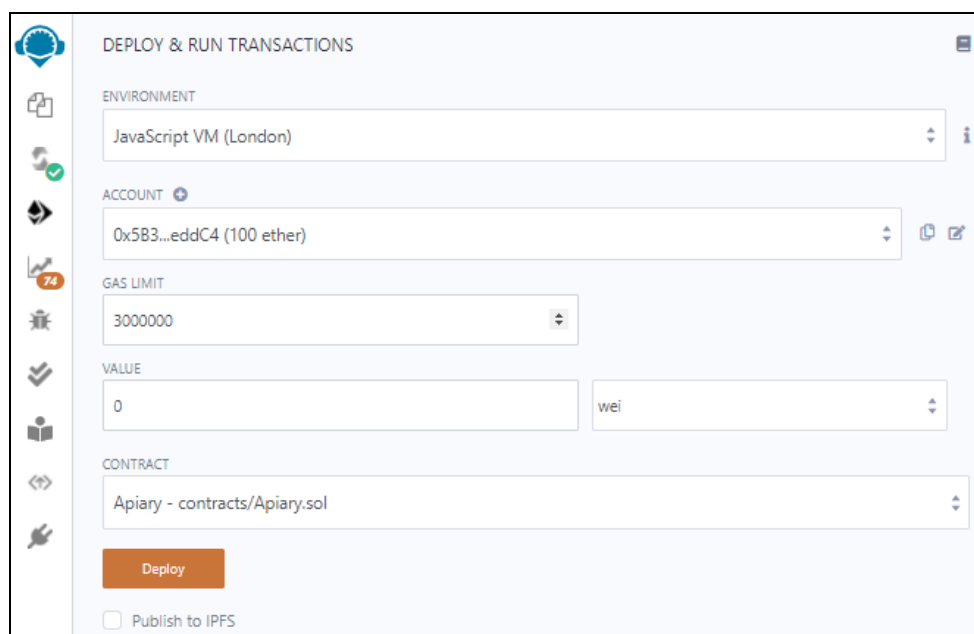
Pasul următor este acela de a implementa contractul inteligent pe rețeaua Ethereum pentru a putea executa acțiuni. Ca mediu de implementare am ales varianta de mașină virtuală JavaScript London, adresa unde se va face implementarea fiind aleasă de mediu, limita maximă de gas pe care o poate utiliza o tranzacție pentru a se executa (3000000). După ce apăsăm pe butonul de *Deploy*, obținem:



The screenshot shows the Remix IDE interface. At the top, it says 'APIARY AT 0XD91...39138 (MEMORY)'. Below that, the function 'AddHoneyInApiaryToSell' is selected. The function has two arguments: 'apiaryHoney' with a value of 'string' and 'quantity' with a value of 'uint128'. There is a 'transact' button on the right. At the bottom, there is a button labeled 'AddNewHoneyTypeInApiary' and a text input field containing 'string honeyType'.

Figură 29: O parte din acțiunile contractului Apiary

În dreptul fiecărui câmp putem completa cu datele pe care dorim să le scriem prin intermediul contractului. De exemplu, în Figura 29, putem scrie Salcâm la tipul de miere și 3000 la cantitatea de miere de care *Stupina* dispune pentru vânzare.

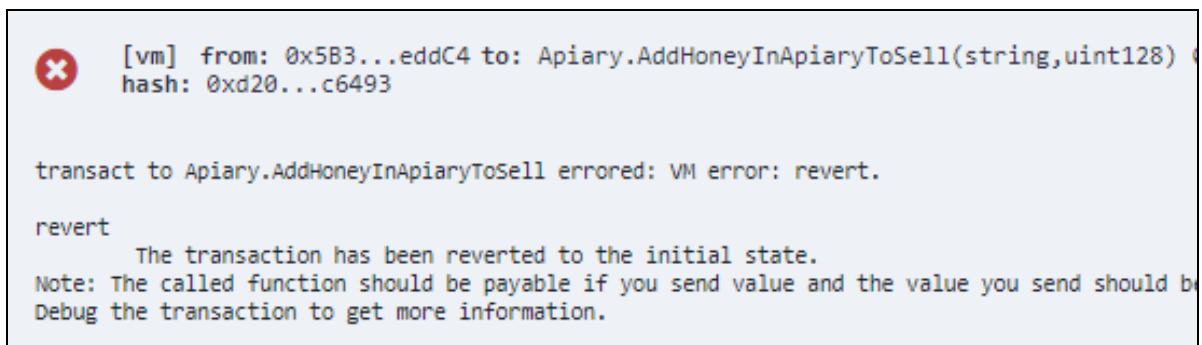


The screenshot shows the 'DEPLOY & RUN TRANSACTIONS' panel in the Remix IDE. It contains the following fields and options:

- ENVIRONMENT:** JavaScript VM (London)
- ACCOUNT:** 0x5B3...eddC4 (100 ether)
- GAS LIMIT:** 3000000
- VALUE:** 0 wei
- CONTRACT:** Apiary - contracts/Apiary.sol
- Buttons:** Deploy, Publish to IPFS (unchecked)

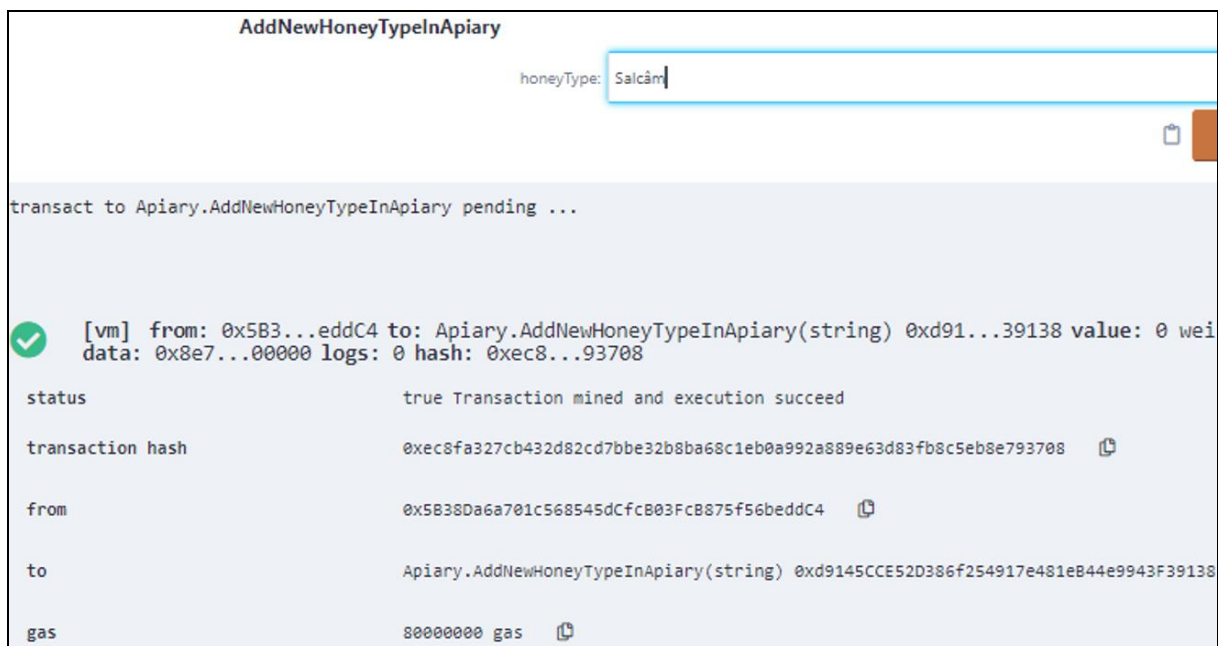
Figură 30: Implementarea contractului Apiary pe blockchain-ul Ethereum

După încercarea de tranzacționare cu datele completate în câmpuri, primim o eroare din partea EWM care ulterior aplică operația *revert* pentru tranzacție și se revine la starea de înainte a blockchain-ului.



Figură 31: EWM face revert tranzacției curente

În funcția care adaugă miere la vânzare am introdus un *assert(condiție, mesaj_eroare)* care, pentru a trece mai departe execuția codului, trebuie să fie îndeplinită condiția dintre paranteze. Condiția era ca în *Stupină* să fie disponibilă miere de tipul respectiv deoarece nu putem vinde un tip de miere care nu există. Astfel, prima dată trebuie să înregistrăm tipul de miere ca disponibil pentru vânzare în cadrul *Stupinei* (Figura 32). După ce am introdus datele și am apăsăat pe butonul de *transact*, observăm în consola text mesajul că tranzația a fost realizată cu succes și câteva date scrise în blockchain în urma execuției tranzacției (Figura 32).



Figură 32: Mesajul primit că tranzacția s-a realizat cu succes după adăugarea unui nou tip

După adăugarea tipului de miere în *Stupină*, putem pune la vânzare mierea respectivă, cu o cantitate precizată (al doilea parametru). Observăm că tranzacția s-a executat cu succes (Figura 33).

```
transact to Apiary.AddHoneyInApiaryToSell pending ...
```

✓ [vm] from: 0x5B3...eddC4 to: Apiary.AddHoneyInApiaryToSell(string,uint128) 0xd91...39138 value: 0 wei data: 0xfbf...0000 logs: 1 hash: 0xdf5...fd112

Figură 33: Re-execuția tranzacției după adăugarea tipului de miere în stupină

În continuare prezint cum o *Stupină* se înregistrează în cadrul unei *Primării*. Pentru a face înregistrarea și apoi solicitarea pentru un certificat de producător, trebuie să implementăm contractul *TownHall*, în mod asemănător cu *Apiary* (Figura 30). După creare, putem să înregistrăm stupina (Figura 34), dar mare atenție la adresa utilizată pentru că adresa unde este implementat contractul este diferită de adresa contului care a implementat contractul. Adresa contractului *TownHall* este *0xa1...AD95*, după apăsăm pe *RequestLicense* și din contractul *TownHall* verificăm licența pentru contractul de la adresa *0xd9...9138*, adică adresa unde este implementat contractul *Apiary*. În Figura 34 observăm certificatul creat, având câmpurile 1, 2, 3, 4 și 5 necompletate (sunt necesare tranzacții

RegisterAtTownHall

townHallContract_addr: 0xa131AD247055FD2e2aA8b156A11bdEc81b9eAD95

transact

RequestLicense

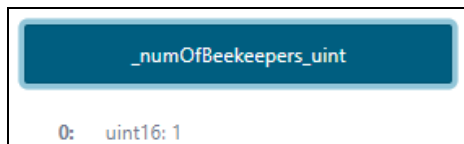
licenses

0xd9145CCE52D386f254917e481eB44e9943F39138

0: uint128: _idLicense_uint 0
1: address: _accountRequester_addr 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
2: string: _producerName_str
3: string: _producerAddress_str
4: string: _apiaryName_str
5: string: _apiaryAddress_str
6: string: _honeyTypes_str Salcâm
7: uint8: _validityMonths_uint 12
8: uint256: _expirationDate_str 1662408318

Figură 34: Înregistrare TownHall, solicitare certificat de producător și verificare existență

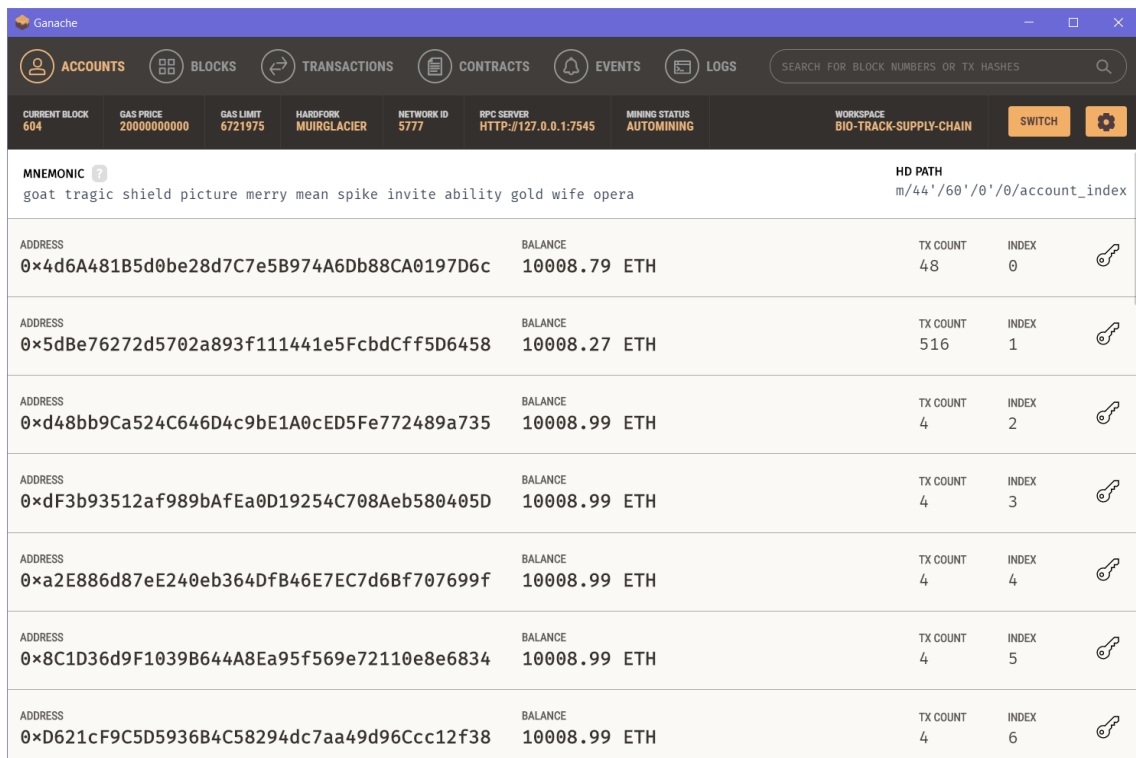
pentru actualizarea acestor date), câmpul 6 conține tipul de miere pentru care s-a obținut certificatul de producător, câmpul 7 reprezintă valabilitatea în luni și ultimul câmp indică data de expirare a acestuia scrisă în format Unix. Câmpul 0 este 0 deoarece este un contor al tuturor certificatelor emise de *Primăria* respectivă. Tot în cadrul contractului *TownHall* putem verifica numărul de apicultori din localitate, deoarece odată cu înregistrarea la *Primărie*, este apelată funcția externă din interfața *ITownHall.sol* care incrementează variabila responsabilă cu evidența stupinelor (Figura 35).



Figură 35: Un apicultor înregistrat în cadrul Primăriei

3.5.2 Testare în consola *Truffle*

Cea de-a doua modalitate de testare a unui contract inteligent este cu ajutorul consolei *Truffle*, folosită direct în *Visual Studio Code*. Analizăm aceleași cazuri de test de mai sus, cu mențiunea că este o rețea blockchain privată, simulată cu ajutorul instrumentului Ganache. Primul pas este de a porni Ganache și implicit rețeaua de blockchain privată (Figura 36).



Figură 36: Instrumentul Ganache

Continuăm cu pasul în care verificăm dacă există erori de compilare prin comanda *truffle compile --all* (Figura 37). Apoi utilizăm comanda *truffle migrate* pentru a implementa contractele în rețeaua locală privată dar pentru că această comandă este bine să o executăm la prima implementare, utilizez mai departe comanda *truffle migrate --reset* (Figura 38).

```
PS C:\Users\Alin\Desktop\Licenta\GitHub\BioTrack_dev\BioTrack\Impl> truffle compile --all

Compiling your contracts...
=====
> Compiling .\contracts\AcquisitionCenter.sol
> Compiling .\contracts\Apiary.sol
> Compiling .\contracts\Laboratory.sol
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\PharmaceuticalAndBeauty.sol
> Compiling .\contracts\Processor.sol
> Compiling .\contracts\ProcessorStorage.sol
> Compiling .\contracts\SaleUnit.sol
> Compiling .\contracts\TownHall.sol
> Compiling .\contracts\Transporter.sol
> Compiling .\contracts\WarehouseUnit.sol
> Compiling .\interfaces\IAcquisitionCenter.sol
> Compiling .\interfaces\IApiary.sol
> Compiling .\interfaces\ILaboratory.sol
> Compiling .\interfaces\IProcessor.sol
> Compiling .\interfaces\ITownHall.sol
> Artifacts written to C:\Users\Alin\Desktop\Licenta\GitHub\BioTrack_dev\BioTrack\Impl\build\contracts
> Compiled successfully using:
   - solc: 0.8.4+commit.c7e474f2.Emscripten.clang
```

Figură 37: Contractele inteligente au compilat cu succes

```
Replacing 'WarehouseUnit'
-----
> Blocks: 0                      Seconds: 0
> contract address: 0x5F49E2DCe6a42EEA4Bd067B8aa887D3bcEbBEb20
> block number: 616
> block timestamp: 1630876980
> account: 0xEaeEB6895F5E772471842db8caD6461D4391664f
> balance: 10008.98713886
> gas used: 128621 (0x1f66d)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.00257242 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost: 0.16742366 ETH

Summary
=====
> Total deployments: 11
> Final cost: 0.17236174 ETH
```

NAME	ADDRESS	TX COUNT	
WarehouseUnit	0x5F49E2DCe6a42EEA4Bd067B8aa887D3bcEbBEb20	0	DEPLOYED

Figură 38: Migrare executată cu success (observat și în Ganache)

Următorul pas este pornirea consolei *Truffle* prin comanda *truffle console* și obținerea unei instanțe pentru cele 2 contracte, *Apiary* și *TownHall* în cod JavaScript (Figura 39).

```
PS C:\Users\Alin\Desktop\Licenta\GitHub\BioTrack_dev\BioTrack\Impl> truffle console
truffle(development)> await Apiary.deployed().then(function(instance) {ap = instance;});
undefined
truffle(development)> await TownHall.deployed().then(function(instance) {th = instance;});
undefined
truffle(development)> █
```

Figură 39: Crearea instanțelor Apiary și TownHall în JavaScript

Dacă încercăm să adăugăm la vânzare miere de salcâm în acest moment, primim o eroare care anulează tranzacția în curs de execuție (Figura 40).

[illegible]

Figură 40: Eroare raportată de mașina virtuală

```
truffle(development)> await web3.eth.getAccounts(function(err, accounts, res) { accounts = res; });
[
  '0x4d6A481B5d0be28d7C7e5B974A6Db88CA0197D6c',
  '0x5dBe76272d5702a893f11441e5FcbbdCff5D6458',
  '0xd48bb9Ca524C646D4c9bE1A0cED5Fe772489a735',
]
truffle(development)> let transaction2 = await ap.AddNewHoneyTypeInApiary("Salcâm", {from: accounts[1]});
undefined
truffle(development)> let transaction1 = await ap.AddHoneyInApiaryToSell("Salcâm", 3000, {from: accounts[1]});
undefined
```

Figură 41: Comenzile necesare pentru a putea pune la vânzare cantitatea de miere

În continuare, pentru înregistrarea *Stupinei* la *Primărie*, verificarea certificatului de producător pentru *Stupina* emitentă și vizualizarea numărului de apicultori din localitatea de care aparține, folosim comenzile din Figura 42.

```

truffle(development)> let tr3 = await ap.RegisterAtTownHall(th.address, {from: accounts[1]});
undefined
truffle(development)> const certif = await ap.RequestLicense({from: accounts[1]});
undefined
truffle(development)> th.licenses(ap.address);
Result {
  '0': BN {
    negative: 0,
    words: [ 0, <1 empty item> ],
    length: 1,
    red: null
  },
  '1': '0x5dBe76272d5702a893f111441e5FcBdCff5D6458',
  '2': '',
  '3': '',
  '4': '',
  '5': '',
  '6': 'Salcâm',
  '7': BN {
    negative: 0,
    words: [ 12, <1 empty item> ],
    length: 1,
    red: null
  },
}
truffle(development)> th._numOfBeekeepers_uint()
BN { negative: 0, words: [ 1, <1 empty item> ], length: 1, red: null }

```

Figură 42: Câteva interogări realizate cu JavaScript

Observăm astfel că prin cele 2 metode este foarte simplă testarea contractelor inteligente. În funcție de preferință se poate alege una din metode. Personal aleg metoda a doua prin consola Truffle deoarece consider că este un control mai mare asupra comenzilor executate și această flexibilitate aduce multe avantaje în principal la depanarea erorilor sau dezvoltarea unor contracte inteligente complexe.

4 TERMENI DE UTILIZARE

4.1 Autorii

Autorii acestui proiect de licență sunt studentul Stanciu Alin Marian și coordonatorul științific, dl. Prof. dr. ing. Costin Bădică.

4.2 Licența de utilizare

Licența de utilizare se referă la autorii acestui proiect de licență. Ei au dreptul de utilizare.

5 CONCLUZII

Ca o concluzie la tot ce s-a desfășurat pe toată perioada dezvoltării aplicației descentralizate pentru un lanț de aprovizionare cu miere ecologică, prezint ce am realizat din proiect dar și puncte pentru dezvoltare în viitor. A fost o provocare foarte mare pentru dezvoltarea acestui proiect deoarece am folosit tehnologia blockchain alături de contracte inteligente, un limbaj nou de programare special pentru contracte (Solidiy), am studiat problemele generale în gestiunea lanțurilor de aprovizionare existente și am încercat să aduc o soluție cât mai potrivită acestora, toate acestea reprezentând provocări pe care le-am abordat într-o manieră științifică și iterativă pe toată perioada dezvoltării proiectului.

Puncte importante din soluția lanțului de aprovizionare realizate în cadrul proiectului sunt:

- Am studiat mai multe tipuri de lanțuri de aprovizionare cu ajutorul blockchain-ului, modul cum au fost gândite să rezolve problemele ridicate de anumite furnizarea anumitor produse (alimente, bunuri materiale, active), am comparat mai multe tehnologii blockchain pentru a stabili pe cea mai potrivită pentru soluția mea, ajungând la Ethereum.
- Am proiectat un sistem pentru lanțul de aprovizionare cu un total de 10 actori, foarte multe interfețe prin care aceștia comunică și de asemenea multe activități specifice fiecărui actor/membru din lanț (Figura 24).
- Am definit formatul și datele care sunt stocate în blockchain pentru a putea fi ulterior interogate de orice membrul din lanțul de aprovizionare.
- Am prezentat o schemă de mapare a design-ului în cod Solidity pentru aplicație.
- Am realizat transpunerea în cod Solidity a 60% din design-ul întregii aplicații.
- Aplicația a fost testată prin două variante: cu ajutorul Remix IDE online și local, cu ajutorul consolei Truffle și Ganache pentru simularea unei rețele blockchain.

Puncte importante pentru dezvoltarea/extinderea aplicației în viitor:

- Finalizarea implementării Solidity pentru întreg design-ul aplicației.
- Definirea unei interfețe grafice personalizate pentru fiecare actor/membru (web și mobil).
- Realizarea unor teste automate.
- Modificarea respectiv adăugarea unor detalii legate de gestiunea lanțului de aprovizionare.

6 BIBLIOGRAFIE

[GUP17] – *Blockchain for dummies*, IBM Limited Edition, autor Manav Gupta, Editura John Wiley & Sons, Inc., Hoboken, 2017, pp. 25-30.

[HAC17] – Blockchain in logistics and supply chain: Trick or treat? , autori Hackius, Niels; Petersen, Moritz, Conference Paper, 2017, pp. 7, 9.

[BUM20] – A blockchain use case in food distribution: Do you know where your food has been?, autori: Daniel Bumblauskasa, Arti Manna, Brett Duganb, Jacy Rittmerb, International Journal of Information Management 52 (2020) 102008, pp 1-3.

[BUT13] – Ethereum White Paper, A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM, autor Vitalik Buterin, 2013.

[GAV21] – Ethereum Yellow Paper, ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER Istanbul version 80085f7 - 2021-07-11, autor: Gavin Wood, 2021.

[SAT08] – Bitcoin: A Peer-to-Peer Electronic Cash System, autor Satoshi Nakamoto, 2008.

[TAK18] – Ethereum EVM illustrated - exploring some mental models and implementations, autor Takenobu T., Martie 2018, revizia 0.01.1

REFERINȚE WEB

[WHA01] – What is Blockchain for business?, 2021, disponibil on-line la adresa <https://www.ibm.com/topics/blockchain-for-business>, accesat la 7 august 2021.

[FAL12] – Studiu de caz - Servicii în gestiunea lanțurilor de aprovizionare, articol disponibil on-line la adresa: https://www.academia.edu/8869518/Lant_de_aprovizionare, accesat la 8 august 2021.

[FRA21] – Smart Contracts, articol Investopedia scris de Jake Frankenfield, revizuit de Erika Rasure, ultimul update 26 mai 2021, disponibil on-line la adresa: <https://www.investopedia.com/terms/s/smart-contracts.asp>, accesat la 8 august 2021.

[CRY21] – Cryptocurrency, articol Investopedia scris de Jake Frankenfield, revizuit de Michael Sonnenshein, ultimul update 9 august 2021, disponibil on-line la adresa: <https://www.investopedia.com/terms/c/cryptocurrency.asp>, accesat la 10 august 2021.

[CEY01] – Inflation: Prices on the Rise, autor Ceyda Oner, articol Finance and Development, disponibil on-line la adresa: <https://www.imf.org/external/pubs/ft/fandd/basics/30-inflation.htm>, accesat la 10 august 2021.

[STE21] - The 10 Public Companies With the Biggest Bitcoin Portfolios, autor Stephen Graves și Daniel Phillips, publicat pe 16 iulie 2021, disponibil on-line la adresa: <https://decrypt.co/47061/public-companies-biggest-bitcoin-portfolios>, accesat la 10 august 2021.

[YES01] – Transaction, Yeshanew Gonfa & Co., imagine, disponibilă online la adresa: <https://ygandco.com/index.php/services/transactions>, accesat la 10 august 2021.

[IBM17] – The difference between Bitcoin and blockchain for business, Blockchain education, autor: Matt Lucas, publicat pe 9 mai 2017, disponibil on-line la adresa: <https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-bitcoin-and-blockchain-for-business/>, accesat la 11 august 2021.

[IBM21] – Why IBM Food Trust?, pagină de prezentare generală IBM Food Trust, disponibil on-line la adresa: <https://www.ibm.com/blockchain/resources/food-trust/why-foodtrust/>, accesat la 15 august 2021.

[JES15] – Cargill open-sources Splinter, its ‘blockchain-like’ supply chain software, autor Jessica Pothering, creat la data de 18 ianuarie 2021, disponibil on-line la adresa: <https://agfundernews.com/splinter-cargill-open-sources-software-to-build-a-better-agrifood-supply-chain.html>, accesat la 15 august 2021.

[AVI13] – 3 Blockbuster Blockchain Trends in 2021, autor Avivah Litan, creat la data de 13 ianuarie 2021, disponibil on-line la adresa: <https://blogs.gartner.com/avivah-litan/2021/01/13/3-blockbuster-blockchain-trends-in-2021/>, accesat la 16 august 2021.

[LAU21] – EOS vs Ethereum - What's the Better Alternative?, autor Laura M., ultima actualizare la data de 5 ianuarie 2021, disponibil on-line la adresa: <https://www.bitdegree.org/crypto/tutorials/eos-vs-ethereum>, accesat la 18 august 2021.

[GWY12] – Hyperledger Fabric Vs Ethereum: Head-To-Head Battle, autor: Gwyneth Iredale, articol creat la data de 12 februarie 2021, disponibil on-line la adresa: <https://101blockchains.com/ethereum-vs-hyperledger-fabric/>, accesat la 19 august 2021.

[NIK22] – A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography, autor: Nick Sullivan, articol creat la data de 24 octombrie 2013, disponibil on-line la adresa: <https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/>, accesat la 22 august 2021.

[MAA08] – The Magic of Digital Signatures on Ethereum, autor: Maarten Zuidhoorn, articol Medium creat la data de 8 octombrie 2020, disponibil on-line la adresa: <https://medium.com/mycrypto/the-magic-of-digital-signatures-on-ethereum-98fe184dc9c7>, accesat la 22 august 2021.

[PAU30] – Consensus Mechanisms, autor: Paul Wackerow, ultima actualizare 30 iulie 2021, disponibil on-line la adresa: <https://ethereum.org/en/developers/docs/consensus-mechanisms/>, accesat la 22 august 2021.

[PAU08] – Proof-of-work (PoW), Paul Wackerow, ultima actualizare 8 august 2021, disponibil on-line la adresa: <https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/>, accesat la 22 august 2021.

[ISU19] – What is Consensus in Distributed Systems?, autor: Isuru Boyagane, articol Medium creat la data de 28 iulie 2019, disponibil on-line la adresa: <https://medium.com/@isuruboyagane.16/what-is-consensus-in-distributed-system-6d51d0802b8c>, accesat la 23 august 2021.

[STE20] – Smart Contracts Are Stored Procedures, autor: Steve Marx, articol creat la data de 20 august 2020, disponibil on-line la adresa: <https://smarx.com/posts/2020/08/smart-contracts-are-stored-procedures/#fn:5>, accesat la 23 august 2021.

[ETI18] – Introduction to Solidity Programming and Smart Contracts (For Complete Beginners), autor: Etienne Dusseault, articol Medium creat la data de 5 august 2018, disponibil on-line la adresa: <https://medium.com/coinmonks/introduction-to-solidity-programming-and-smart-contracts-for-complete-beginners-eb46472058cf>, accesat la 23 august 2021.

[TOM22] – How to Learn Solidity in 30 days, autor: Tom Teredo, articol Medium creat la data de 22 aprilie 2021, disponibil on-line la adresa: <https://medium.com/coinmonks/how-to-learn-solidity-in-30-days-78b02e503d23>, accesat la 23 august 2021.

[CAL19] – What Is Solidity and How Is It Used to Develop Smart Contracts?, autor: Calvin Ebun-Amu, articol creat la data de 19 aprilie 2021, disponibil on-line la adresa: <https://www.makeuseof.com/what-is-solidity/>, accesat la 23 august 2021.

A. CODUL SURSĂ

Link-ul către repository-ul GitHub:

[StanciuAlin/BioTrack: Supply chain for BIO honey using Ethereum \(github.com\)](https://github.com/StanciuAlin/BioTrack)

B. CD/ DVD

Versiunea electronică a aplicației, a acestei lucrări, precum și prezentarea finală a tezei.

INDEX

B

Bibliografie..... 21

C

CUPRINSULxiii

L

LISTA FIGURILOR..... xv

LISTA TABELELOR.....xvi

R

Referințe web 22