Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Facultatea de Automatică și Calculatoare
Domeniul: Ingineria Sistemelor
Specializarea: Automatică și Informatică Aplicată

Utilizarea tehnologiei blockchain în contextul industriei 4.0

PROIECT DE DIPLOMĂ

Absolvent:
Bogdan Iacob

Coordonator:
Ș.l. dr. ing. **Silviu-Florin Ostafi**

Iași, 2021

Cuprins

Utilizarea tehnologiei blockchain în contextul industriei 4.0

Rezumat

Baza teoretică a proiectului dezvoltat oferă o soluție pentru multe domenii, atât aplicații industriale cât și aplicații cotidiene, domeniul fiind însă restrâns la sectorul industrial și interacțiunea cu potențialii clienți într-un mod transparent, incoruptibil și modern. Tehnologiile utilizate sunt eficiente și de actualitate și oferă o perspectivă largă a aplicabilității și scalabilității. Principiile de funcționare ale aplicației sunt definite în contractele inteligente, instrumentele necesare care permit imuabilitatea operațiunilor și costul redus al tranzacțiilor. Interfața utilizatorului este afișată sub forma unei aplicații web, în care se oferă procese fluide și design modern, deci, un mediu de utilizare plăcut. Modul de implementare a fost abstractizat, dar accentul a fost pus pe detalii relevante pentru utilizator.

Componenta principală a proiectului a fost construită folosind principiile tehnologiei Blockchain, o tehnologie care se afla în curs de maturizare, dar care are un potențial mare. Instrumentele utilizate au oferit posibilitatea de a simula toate operațiunile posibile pe rețeaua principală. Interfața de utilizare este definită în cadrul restrâns al aplicației, deoarece accentul trebuie să fie strict orientat în procesul de interacțiunea utilizatorului cu elementul fizic reprezentativ.

Digitalizarea și globalizarea au adus o eră emergentă a sistemelor de fabricație prin conectarea consumatorilor la capacitățile producătorilor. Indiferent de sectorul din producție despre care vorbim, inovația tehnologică este esențială pentru viitorul oricărei afaceri de succes. Industry 4.0 reprezintă noul val în evoluția tehnologică a producției, convingând tot mai multe companii să răspundă noilor standarde stabilite de o piață aflată într-o continuă schimbare, prin transformarea unităților de producție în "fabrici inteligente". Cu toate acestea, asigurarea încrederii între elementele sistemului într-un scenariu de fabricare inteligentă poate reprezenta o provocare, în special atunci când există mai multe părți implicate. Scopul prezentei lucrări este propunerea unei soluții în cazul stabilirii încrederii dintre fabricile inteligente și potențialii clienți. Prezenta lucrare descrie o abordare descentralizată de gestionare a informațiilor de fabricare generate prin intermediul unei aplicații web folosind tehnologia Blockchain. Sunt descrise, de asemenea, și etapele obținerii unui sistem descentralizat, menit sa simuleze modul de interacțiune dintre un client și o astfel de organizație.

Introducere

În acest capitol se va face o descriere introductivă a aplicației, prin prezentarea contextului acesteia, motivând conceptul care a stat la baza studierii și dezvoltării acestui proiect. De asemenea, va fi definită și structura acestei lucrări, care, conține documentarea bazelor teoretice ale temei alese, pașii parcurși și rezultatele obținute după implementarea soluției dezvoltate.

Contextul proiectului

Digitalizarea producției aduce organizațiile într-o zonă de competitivitate ridicată, datorită transformării fabricilor în unități conectate și auto-organizate. Astfel de centre de producție prezintă o eficiență crescută prin fabricarea produselor într-un timp mai scurt și la costuri reduse, ceea ce determină automat și un profit mai ridicat. Transparența oferită de descentralizare poate rezolva problema determinării încrederii într-o rețea de fabricare inteligentă, fiecare componentă având în mod inerent încredere în propriile procese puse în aplicare. În mod tradițional, această încredere a fost stabilită între clienți și producători prin negocieri ample de contracte, recunoașterea performanței istorice anterioare, certificări actualizate și audituri pentru a asigura conformitatea. Această „taxă de încredere", sau mai degrabă costul asociat asigurării încrederii între toate părțile într-un lanț de aprovizionare al unui ansamblu de produse, este încorporat în prețul final de vânzare perceput clientului.[1]

Motivația alegerii temei

Motivația realizării acestui proiect este reprezentată de dorința de a oferi o soluție parțială a acestei probleme, și anume folosirea platformelor bazate pe cerere prin conectarea clienților la capabilitățile de producție a fabricilor. Fără capabilități fizice, aceste tipuri de platforme pot produce valoare adăugată colaboratorilor, clienților și furnizorilor de servicii prin împărtășirea informației asociate procesului de fabricație, creând un grad ridicat de încredere. Soluția ce ne permite acest mecanism de descentralizare este conceptul de shared ledger numit Blockchain.

Acest "registru distribuit" nu este deținut de nicio autoritate centrală și orice participant autorizat poate scrie și vedea ce conține. Evenimentele legate de fabricant sunt distribuite în întreaga rețea de noduri, iar datele stocate în blockchain sunt imuabile, deci stabilesc datele de proveniență și fabricație. Orice producător poate face accesibile datele către orice alt participant în această rețea.

Aplicarea tehnologiei Blockchain în acest context printr-o arhitectură adecvată poate insufla caracteristici precum autenticitatea tranzacțiilor, transparența acestora, și folosirea resurselor colectate în scopul în care au fost formulate inițial. În cazul unui client, acest proces poate ajuta la stabilirea unei reputații și determinarea parțială a încrederii între cele două părți.

Structura lucrării

În continuare, va fi introdus conținutul lucrării și structura sa ce se bazează pe o listă de obiective stabilite în vederea documentării și rezolvării problemei propuse, după cum urmează:

- În Introducere, au fost descrise contextul și motivația care au constituit fundamentul realizării acestui proiect.
- În capitolul 1, Fundamentarea teoretică și noțiunile de bază din domeniul Blockchain
- În capitolul 2, Analiza tehnologiilor utilizate la proiectarea soluției, și se va face referință la aplicațiile similare conceptual cu proiectul propus.
- În capitolul 3, Proiectarea aplicației, vor fi ilustrate principalele componente ale proiectului și a interacțiunilor dintre ele, reprezentate prin diagrame UML și va fi descrisă platforma pe care va fi executată aplicația. Mai mult decât atât, se vor analiza avantajele și dezavantajele metodei alese.
- În capitolul 4, Implementare aplicației, va fi prezentată într-o manieră detaliată realizarea aplicației și vor fi descrise dificultățile întâmpinate și modalitatea de rezolvare a acestora.
- În capitolul 5, Testarea aplicației și rezultate experimentale, va fi prezentată maniera de testare a aplicației, ilustrând rezultatele experimentate și performanța sistemului.
- În capitolul final, Concluzii, va fi analizat proiectul și se vor contura noi posibile funcționalități, ce pot fi implementate ulterior.

I. Blockchain - noțiuni de bază

În acest capitol se vor prezenta concepte de bază din domeniul temei alese, se vor acoperi elementele de esențiale ale infrastructurii de bază din spatele tehnologiei Blockchain, în special în legătură cu evoluțiile recente ale construirii Blockchain 4.0. Scurta descriere este furnizată pentru a introduce termenii folosiți pe parcursul lucrării.

1.1. Blockchain

Blockchain este un mecanism software distribuit care oferă un sistem cu o listă în continuă creștere a tranzacțiilor active de încredere, fără a fi nevoie de o autoritate centrală. Astfel de tranzacții sunt stocate într-un lanț de blocuri care sunt în esență structuri de date legate, care conțin un lot de tranzacții valide și verificate. Poate fi privit ca un registru în continuă creștere, care păstrează o înregistrare permanentă a tuturor tranzacțiilor care au avut loc în ordine cronologică și al căror conținut este imuabil. Fiecare bloc constă dintr-un hash imuabil al blocului anterior la care este conectat, și care formează în cele din urmă o verigă de blocuri care conțin date care pot fi asociate în mod unic cu un activ fizic, cum ar fi o persoană sau o proprietate fizică. Această bază de date distribuită rulează pe mai multe servere (noduri) pe o întreagă rețea, fiecare nod verificând securitatea și integritatea introducerii datelor în blocurile din această rețea peer-to-peer. Deoarece nu există un sistem central, încrederea este distribuită pe nodurile din rețeaua blockchain.

Blockchain 4.0: Această generație s-a concentrat în principal pe servicii precum public ledger și bazele de date distribuite în timp real. Acest nivel are o integrare perfectă a aplicațiilor bazate pe industria 4.0. Folosește contractul inteligent care elimină necesitatea contractelor pe hârtie.[2]

1.1.1. Arhitectura

Ca arhitectura, un Blockchain este o structură de date de tipul unei liste simple înlănțuite, în care legătura dintre blocuri se face printr-un hash.

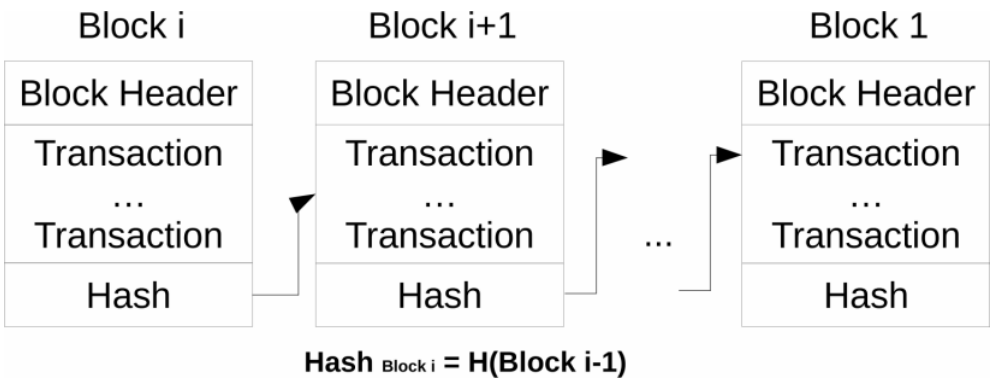

Fig. I.1 Exemplu de înlănțuire de blocuri, preluat din [2]

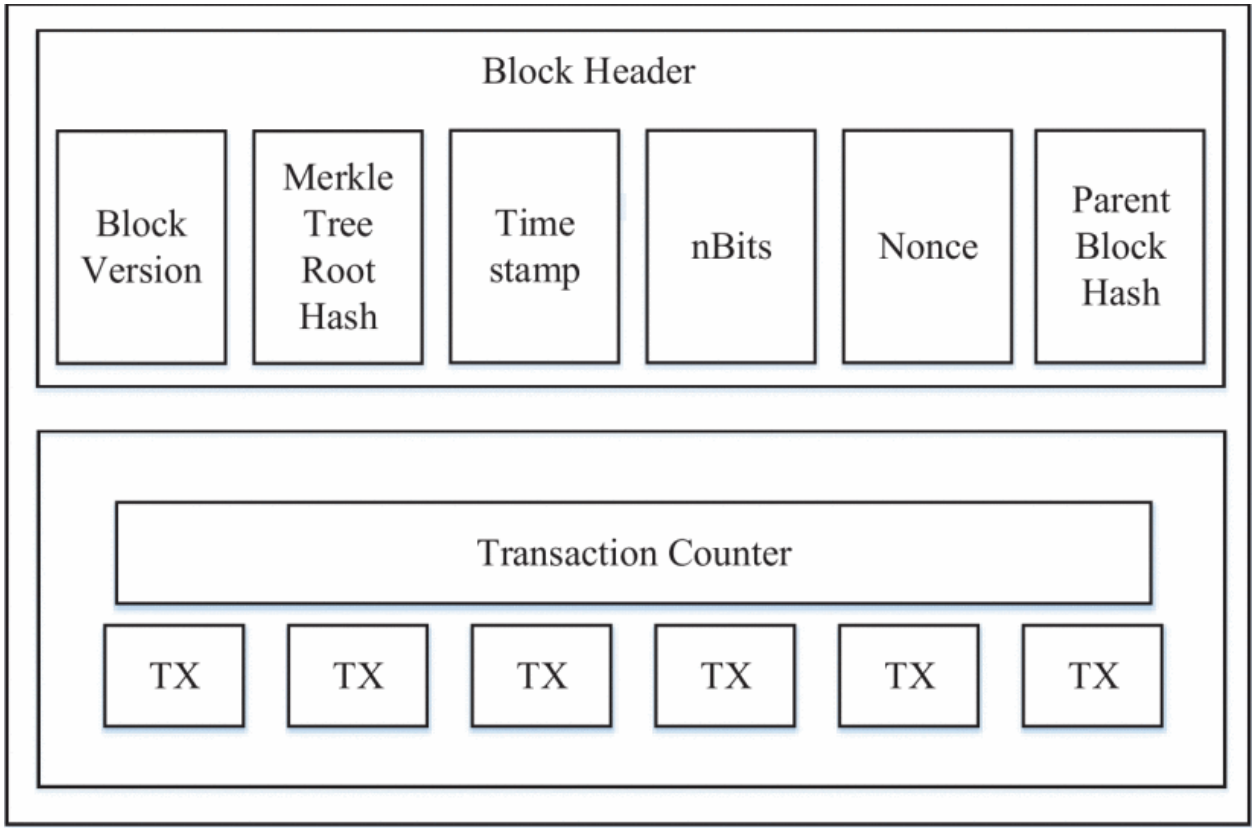

Fig. I.2 Structura unui bloc, preluat din [2]

Un bloc într-un blockchain este o înregistrare a unora sau a tuturor tranzacțiilor recente care au avut loc în rețea. O tranzacție poate conține înregistrări ale evenimentelor semnificative și relevante legate de activul urmărit. O structură de bloc conține informații despre cele mai recente tranzacții, dimensiunea blocului, un contor de tranzacții care păstrează o urmă a instanței de blocuri adăugate la blockchain, un antet de bloc care păstrează informații pe hash-ul criptografic al blocului anterior și actual, marcajul de timp curent și un „număr folosit o singură dată” sau un număr aleatoriu (Nonce) care ajută la generarea de hash-uri valide pentru blocurile ulterioare.[3]

Funcțiile criptografice hash sunt algoritmi de criptare ce au ca intrare un mesaj de orice lungime și returnează la ieșire un mesaj (numit hash) scurt, cu o lungime fixă. Acest hash poate fi utilizat, spre exemplu, în semnăturile digitale. Pentru ca o funcție hash să fie eficientă, un atacator nu ar trebui să poată găsi două mesaje care să producă același hash.

Funcțiile hash criptografice sunt funcții neinversabile, adică mesajul hash de la ieșire nu poate fi utilizat pentru reconstituirea mesajului inițial. Acestea sunt folosite pentru a verifica autenticitatea datelor primite de la surse necunoscute sau pentru a adăuga încă un nivel de securitate. Printre cei mai cunoscuți algoritmi de criptare sunt MD5 (Message Digest), Whirlpool, SHA-1, SHA-2, SHA-3 (Secure Hash Algorithm) și HMAC. Pentru această lucrare, de interes este algoritmul SHA-256.

În criptografie, SHA-256 și SHA-512 sunt funcții noi de tip hash, calculate cu mesaje de 32 și respectiv 64 de biți. Ele folosesc cantități diferite de schimburi și constante aditive, însă structurile lor sunt identice, diferind doar numărul de runde.

SHA-256 are dimensiunea de ieșire pe 256 de biți, dimensiunea stării interne pe 256 de biți, dimensiunea blocului de 512 biți, dimensiunea maximă a mesajului pe care o poate gestiona este de $2^{64} - 1$, lungimea cuvântului este de 32 de biți, iar numărul de runde aplicate este de 64, precum și operațiile aplicate hashului sunt +, și, sau, xor, shr și rot. Lungimea hash-ului este întotdeauna aceeași, indiferent de cât de mare este conținutul pe care îl utilizați pentru a genera hash-ul: indiferent de datele de intrare rezultatul va fi întotdeauna o succesiune de 40 de litere și numere.[4]

$$
\begin{aligned}
Ch(X,Y,Z) &= (X \wedge Y) \oplus (\overline{X} \wedge Z),\\
Maj(X,Y,Z) &= (X \wedge Y) \oplus (X \wedge Z) \oplus (Y \wedge Z),\\
\Sigma_0(X) &= RotR(X,2) \oplus RotR(X,13) \oplus RotR(X,22),\\
\Sigma_1(X) &= RotR(X,6) \oplus RotR(X,11) \oplus RotR(X,25),\\
\sigma_0(X) &= RotR(X,7) \oplus RotR(X,18) \oplus ShR(X,3),\\
\sigma_1(X) &= RotR(X,17) \oplus RotR(X,19) \oplus ShR(X,10),
\end{aligned}
$$

Fig. I.3 Operațiile algoritmului SHA-256 , preluat din [4]

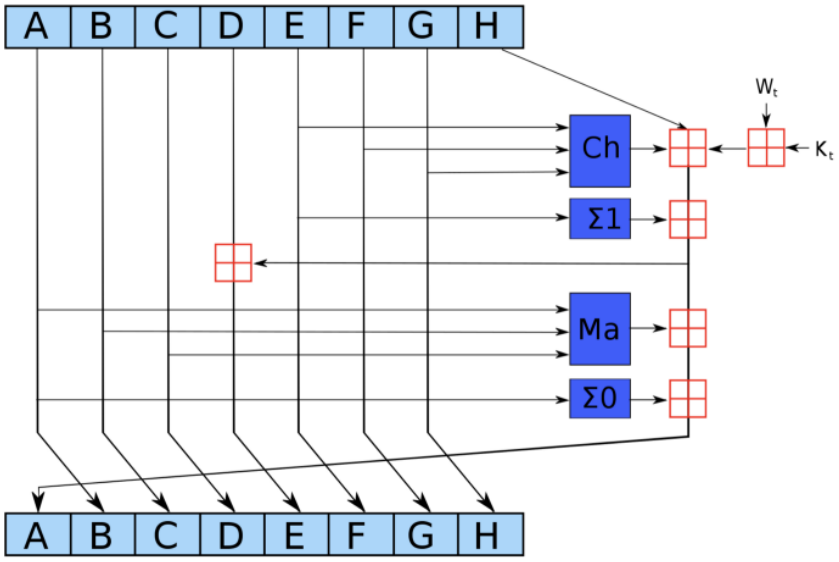

Fig. I.4 O iterație a algoritmului SHA-256, preluat din [5]

Ieșirea din fiecare etapa acționează ca o intrare pentru etapa următoare și acest proces continuă până când rămân ultimii biți ai mesajului și rezultatul ultimei etape a blocului de mesaj ne va da rezultatul, adică hash-ul pentru întregul mesaj. Lungimea ieșirii este de 256 de biți. Algoritmul de hash SHA-256 este în prezent unul dintre cei mai folosiți algoritmi de hash, deoarece nu a fost încă spart și hash-urile sunt calculate rapid.[5]

Mineritul este una dintre cele mai importante paradigme în ideea sistemelor descentralizate și distribuite. Mineritul în contextul blockchain-ului implică un mecanism pentru stabilirea consensului cu privire la starea blockchain-ului pentru a se asigura că este sigur.

Procesul de minerit este crucial pentru înțelegerea paradigmei blockchain și a imuabilității conținutului său de date. Mineritul este procesul de rezolvare a unei probleme matematice dificile, bazate pe un algoritm de hash criptografic. Minerii validează tranzacțiile noi și le înregistrează în registrul global (Blockchain). Cel care reușește să adauge noul bloc primește o recompensă pentru timpul și resursele implicate în proces.

Caracteristicile unui Blockchain:

- **Persistența**. Datele stocate în Blockchain sunt, în teorie, imuabile și nu pot fi modificate ușor. De asemenea, datele sunt adăugate la bloc după ce sunt aprobate de toată lumea din rețea, permițând astfel tranzacții sigure. Cei care validează tranzacțiile și le adaugă în bloc se numesc mineri.
- **Transparența**. Fiind un registru deschis, acesta conține înregistrarea tranzacțiilor efectuate în mod vizibil pentru toată lumea, de aceea se numește evidență sau registru deschis. Niciun participant nu se încarcă cu tranzacții. Fiecare nod din rețeaua Blockchain are aceeași copie a înregistrării.
- **Descentralizarea**. Nu există o autoritate centrală pentru a controla rețeaua, așa cum există în modelul client-server. Blockchain oferă o rețea peer-to-peer. Această caracteristică a Blockchain permite tranzacțiilor să implice doar două părți, expeditorul și receptorul.
- **Anonimatul**. Datele confidențiale ale utilizatorului nu sunt necesare pentru a interacționa cu Blockchain-ul, o singură cheie privată și o adresă fiind suficiente.

Cerintele unui Blockchain:

Smart Contracts: Este un protocol ce permite efectuarea tranzacțiilor în absența terților și care face tranzacțiile ireversibile și trasabile.

Securitatea datelor: respectarea securității este o cerință majoră și esențială a tehnologiei blockchain, din punct de vedere legal.

Tokenization: Este unul dintre cele mai importante aspecte care trebuie incluse în blockchain. Facilitează reprezentarea digitală a bunurilor, serviciilor și drepturilor cu ajutorul jetoanelor. Permite schimbul de valori și încredere pentru diferiți utilizatori fără implicarea autorității centrale.

Stocare de date descentralizată: este o cerință de bază a sistemului distribuit.

Imuabilitate: Toate înregistrările din rețea nu trebuie modificate în registrul partajat. Aceasta permite integritatea datelor stocate.

Consens: Tranzacțiile ar trebui actualizate numai atunci când toți utilizatorii verificați din rețea sunt de acord pentru același lucru.

Blocuri tipizate: este necesar pentru contractul inteligent și pentru plata de mare viteză în tranzacțiile comerciale. Deci, formatarea datelor diferitelor tipuri de blocuri include timpul, algoritmul de consens, numărul de tranzacții pe blocuri și tipurile sale de date de conținut.

Fragmentare: este necesar pentru separarea conținutului peste subseturi de noduri astfel încât nu toate nodurile să aibă nevoie de toate sarcinile de procesare sau orice sarcină.

Gestionarea drepturilor de acces: Criptografia bazată pe criptare privată și publică și bazele de date distribuite cu identificarea utilizatorului sunt necesare pentru a atribui și gestiona drepturile de acces.

Standarde utilizate pentru gestionarea blockchain-urilor permise: imuabilitatea rețelei blockchain permite accesul la date într-o ordine specifică. Certificatele publice sunt disponibile în blockchain public, dar fără a avea cheia privată, autorizarea nu poate fi acordată utilizatorilor. Deci, toate datele ar trebui gestionate în ordinea elementelor de date, cum ar fi IP ul utilizatorului, numele, codul acestuia și limbajul XML. Toate acestea sunt publicate consorțiului cu procesul de comunicare.

Formatarea standard a datelor: În sistemul blockchain, este de asemenea necesar să standardizați formatele de date în ceea ce privește interfețele de programare a aplicațiilor (API). Fiecare organizație din rețeaua blockchain trebuie să utilizeze același format de date sau API-uri pentru a comunica în aceeași rețea.

Actualizare: nevoia de actualizare a datelor în registrul distribuit este cea mai importantă pentru înregistrări. Într-o rețea peer-to-peer, datele trebuie să fie structurate și actualizate sistematic pentru fiecare nod care tranzacționează în cadrul rețelei.

Criptare P2P între nodurile blockchain: Criptarea este necesară pentru a asigura tranzacțiile dintre nodurile finale ce se pot lega împreună în protocolul blockchain.

Development operation: Pasul principal în producția sistemului este selectarea platformelor care necesită mai puțin timp și complexitatea configurării.[2]

1.1.2. Mecanisme de consens

Algoritmii de consens sunt esențiali pentru funcționarea sistemelor descentralizate și distribuite, scopul acestora este să obțină consistența datelor între toate nodurile unui sistem descentralizat. Responsabilitatea pentru verificarea și analiza datelor pe un sistem descentralizat poate fi asumată de orice nod arbitrar sau atribuit din rețea. Pentru a rezolva această problemă într-un mediu lipsit de încredere, ar avea sens ca fiecare nod al rețelei să poarte exact aceeași copie a înregistrării fiecărei tranzacții. În acest fel, atâta timp cât majoritatea utilizatorilor nu intră în conflict, sistemul descentralizat va menține un mediu de încredere pentru ca părțile să lucreze împreună fără a fi nevoie de un terț. Acest proces de a permite participanților la rețea să acționeze ca verificatori pentru tranzacții în schimbul unor recompense se numește *mining*.

Dovada muncii (Proof-of-Work) a fost primul algoritm de consens blockchain. Conceput de Satoshi Nakamoto pentru utilizarea în blocul Bitcoin, implementarea PoW o putem observa astăzi în operațiunile de *mining* și consumul de energie pe care îl vedem în întreaga lume. În PoW, minerii rezolvă probleme pentru a crea blocuri.

Dovada mizei (Proof of stake) propusă de King și Nadal pentru Peercoin nu se bazează pe procesele miniere tradiționale, ci mai degrabă propune selectarea verificatorilor pe baza factorilor aleatori ai parametrilor „miza", cum ar fi "bogăția" sau reputația combinate cu valori hash etc.

Dovada de autoritate (eng., Proof of Authority) este propusă ca o modalitate de implementare a consensului într-o rețea privată în care participanții sunt verificați și conștienți de identitatea celuilalt. Sistemul este condus de N noduri de încredere (autorități) care au autoritatea de a verifica tranzacții (sau date) noi și de a emite blocuri noi pentru a fi adăugate la blockchain. Se presupune că un număr minim de autorități trebuie sa fie onest (adică N / 2 + 1 autorități). Dacă există un conflict sau o fraudă, este posibil ca o anumită autoritate să fie oprită din procesele de analiză de date. Avantajul acestei rețele comparativ cu alte rețele de test disponibile este imunitatea la atacurile de tip spam.

Toleranța defectelor bizantine (Byzantine Fault Tolerance) este proprietatea unui sistem care este capabil să reziste clasei de eșecuri derivate din problema generalilor bizantini. Aceasta înseamnă că un sistem BFT este capabil să continue să funcționeze chiar dacă unele dintre noduri eșuează sau acționează rău intenționat.

Dovada greutatii (Proof-of-Weight) este o clasificare largă a algoritmilor de consens bazați pe modelul consensului Algorand. Ideea generală este că în cazul în care, în PoS, procentajul de jetoane deținute în rețea reprezintă probabilitatea de a "descoperi" următorul bloc, într-un sistem PoWeight este utilizată o altă valoare relativ ponderată.

Dovada delegată a mizei (DPOS) Diferența majoră dintre PoS și DPOS este că PoS este direct democratic, în timp ce DPOS este reprezentativ democratic. Părțile interesate își aleg delegații pentru a genera și valida blocuri. Cu semnificativ mai puține noduri pentru validarea blocului, blocul ar putea fi confirmat rapid, ducând la confirmarea rapidă a tranzacțiilor. Între timp, parametrii rețelei, cum ar fi dimensiunea blocului și intervalele de blocare, ar putea fi reglați de delegați. În plus, utilizatorii nu trebuie să se îngrijoreze de delegații necinstiți, deoarece ar putea fi votați cu ușurință.

Ripple este un algoritm consens care utilizează subrețele de încredere colectivă în cadrul unei rețele mai mari. În rețea, nodurile sunt împărțite în două tipuri: *server* pentru procesul de consens participant și *client* pentru transferul de fonduri. Fiecare server are o listă unică de noduri (UNL). UNL este important pentru server. Atunci când se determină dacă se introduce o tranzacție în registru, serverul interogheaza nodurile din UNL și dacă acordurile primite au atins 80%, tranzacția este ambalată în registrul. Pentru un nod, registrul va rămâne corect atâta timp cât procentul de noduri defecte în UNL este mai mic de 20%.[3]

Tabelul 1. Compararea algoritmilor de consens

Numele algoritmului						
Proprietate	PoW	PoS	PBFT	PoA	DPoS	Ripple
Economiseste energie	Nu	Partial	Da	Da	Partial	Da
Vulnerabilitate	< 25% puterea de calcul	< 51% din miza	< 33.3% replici	< 51% autorități	< 51% validatori	< 20% noduri corupte

1.1.3. Clasificarea sistemelor Blockchain

Sistemele actuale de Blockchain-uri sunt clasificate în trei tipuri: Blockchain public, privat și de consorțiu/hibrid.

1) **Blockchain public** Acesta prevede o platforma deschisa ce permite oricărei persoane să participe ca utilizator. Toate tranzacțiile care au loc pe Blockchain-uri publice sunt complet transparente, ceea ce înseamnă că oricine poate vedea detaliile tranzacției, citi sau scrie altele.

- Lanțurile de blocuri publice sunt concepute pentru a fi complet descentralizate, fără ca nimeni să controleze tranzacțiile înregistrate în Blockchain sau ordinea în care sunt procesate.
- Blockchain-urile publice pot fi rezistente la cenzură, deoarece oricine este deschis să se alăture rețelei, indiferent de locație, naționalitate, etc. Acest lucru face extrem de greu pentru autorități să le închidă.
- În cele din urmă, Blockchain-urile publice au toate un token asociat cu acestea, care este de obicei conceput pentru a stimula și recompensa participanții din rețea.

2) **Blockchain privat**. Este un tip de sistem blockchain care este configurat pentru a facilita partajarea și schimbul privat de date între un grup de persoane.Acesta deține o serie de diferențe notabile față de Blockchain-urile publice și anume:

- Participanții au nevoie de permisiune pentru a se alătura rețelelor.
- Tranzacțiile sunt private și sunt disponibile numai participanților cărora li sa acordat permisiunea de a se alătura rețelei.
- Blockchain-urile private sunt mai centralizate decât Blockchain-urile publice.

3) **Blockchain de consorțiu/hibrid.** BUn blockchain de consorțiu poate fi considerat un blockchain parțial privat și permis, în care nu o singură organizație, ci un set de noduri predeterminate sunt responsabile pentru consens și validarea blocurilor.

Această abordare are toate avantajele unei Blockchain private și ar putea fi considerată o subcategorie a Blockchain-urilor private.

- Aceștia sunt capabili să fie mai eficienți, atât individual, cât și colectiv, prin colaborarea la anumite aspecte ale activității lor.
- Participanții la Blockchain-urile de consorțiului ar putea include pe oricine de la băncile centrale, la guverne, să furnizeze lanțuri de aprovizionare.[3]

Tabelul 2. Comparație între blockchain public, blockchain consorțiu și blockchain privat

Proprietate	Blockchain public	Blockchain hibrid	Blockchain privat
Determinarea consensului	Toți minerii	Set select de noduri	O organizație
Permisiune de citire	Public	Public sau restricționat	Public sau restricționat
Imutabilitate	Imposibil de modificat	Poate fi modificat	Poate fi modificat
Eficiență	Mica	Mare	Mare
Centralizat	Nu	Parțial	Da
Procesul de consens	Fără permisiune	Cu permisiune	Cu permisiune

1.2. Ethereum Blockchain

Ethereum este a doua cea mai mare platformă de criptomonede, prin capitalizarea pieței, imediat după Bitcoin. Este un Blockchain open-source descentralizat, cu funcționalitatea contractului inteligent. Mașină virtuală Ethereum (EVM) este platforma de calcul descentralizată care constituie nucleul platformei Ethereum. Este mediul în care sunt executate contractele, acest protocol exista cu scopul de a menține operațiile neîntrerupte și imuabile [6].

1.3. Rețeaua de test Kovan

Ca rețea de test ethereum, am folosit Kovan testnet. Aceasta are la bază implementarea algoritmului de consens dovada de autoritate. Dovada de autoritate (eng., Proof of Authority) este propusă ca o modalitate de implementare a consensului într-o rețea privată în care participanții sunt verificați și conștienți de identitatea celuilalt. Sistemul este condus de N noduri de încredere (autorități) care au autoritatea de a verifica tranzacții (sau date) noi și de a emite blocuri noi pentru a fi adăugate la blockchain. Se presupune că un număr minim de autorități trebuie sa fie onest (adică N / 2 + 1 autorități).

Dacă există un conflict sau o fraudă, este posibil ca o anumită autoritate să fie oprită din procesele de analiză de date. Avantajul acestei rețele comparativ cu alte rețele de test disponibile este imunitatea la atacurile de tip spam [7].

1.4. Smart Contracts

Contractul inteligent (eng. Smart Contracts) este un protocol care permite efectuarea tranzacțiilor în absența terților și care face tranzacțiile ireversibile și trasabile.

Smart Contract, este un termen folosit pentru a descrie un protocol special care are ca scop utilizarea tehnologiei blockchain în logica de negociere și performanță a unui contract. Conține toate detaliile condițiilor contractuale și efectuează automat toate operațiunile definite inițial. Este o alternativă modernă la contractul clasic și are o gamă largă de aplicabilitate. Acestea permit tranzacții de încredere între două sau mai multe entități fără a fi nevoie ca un intermediar (sau o terță parte) să verifice tranzacția. Aceste tranzacții sunt ireversibile și pot fi monitorizate.

Principiile de bază pot fi comparate cu automatele, acestea funcționând doar cu instrucțiuni primite. La început, bunurile și condițiile contractuale sunt programate și introduse în Blockchain, apoi distribuite și reproduse de mai multe ori între nodurile platformei.

Odată ce evenimentul se produce, contractul este executat și termenii sunt verificați și tranzacția este executată sau nu. Diferența dintre interacțiunea cu contractul inteligent și interacțiunea cu un API RESTful, constă în faptul ca nu există o singură execuție de server. Acțiunea solicitată mai degrabă este o rețea de calculatoare, care se verifică reciproc, asigurându-se că niciun nod nu încearcă să înșele sistemul.[6]

Modul de funcționare

Contractele inteligente Ethereum nu sunt compatibile cu limbajele de programare convenționale. În schimb, limbajele specifice contractelor, cum ar fi Solidity, pot fi folosite pentru a defini funcțiile contractului. Apoi, compilatorul converteste această funcție logică din contract în bytecode și o distribuie pe blockchain. Când o funcție de contract logic este convertită în bytecode, va transmite, de asemenea, un Application Binary Interface (ABI). Pentru a interacționa cu contracte inteligente, se poate utiliza biblioteca Web3.js, care ajută contractul inteligent fie utilizată într-un browser web sau o aplicație.

```
pragma ton-solidity >= 0.35.0;
pragma AbiHeader expire;

contract HelloWorld {
    function HelloWorld() public pure returns (string) {
        tvm.accept();

        return 'Hello World!';
    }
}
```

Figura I.5. Smart Contract scris în limbajul Solidity

1.5. Oracol

Oracolele sunt programe ce permit rețelei blockchain să interacționeze cu elemente din afara sa. Oracolul poate fi scris personalizat pentru a interacționa cu contractele inteligente existente pe blockchain. Deoarece contractul inteligent se execută de sine stătător, oracolele pot furniza datele necesare pentru calcul de către contractele inteligente [6], [8].

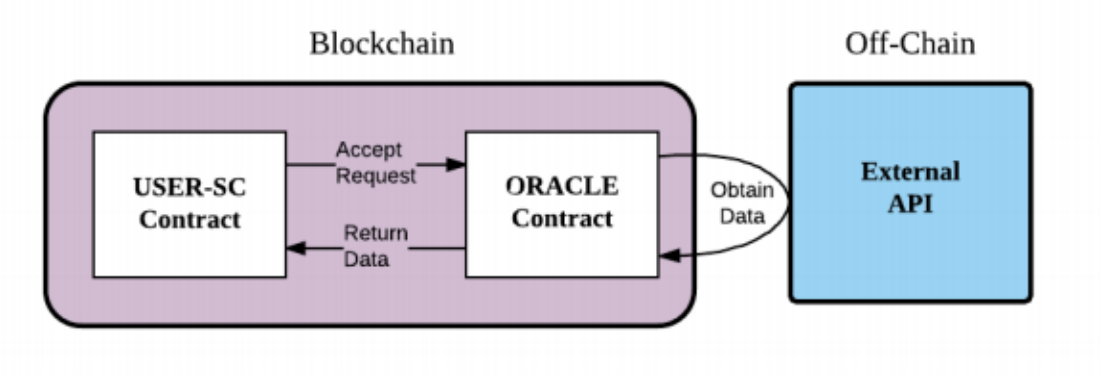

Figura I.6 Comportamentul unui oracol ideal, preluat din [8]

1.6. Chainlink

Chainlink este o rețea descentralizată de Oracole ce oferă intrări și ieșiri de încredere pentru contracte inteligente pentru orice Blockchain. Chainlink extinde în mare măsură capacitățile contractelor inteligente, permițând accesul la date din lumea reală și la calcule din afara lanțului (eng. off-chain), menținând în același timp garanțiile de securitate și fiabilitate inerente tehnologiei blockchain.

Metoda descentralizată a rețelelor de oracole Chainlink permite contractelor inteligente să utilizeze și să creeze fără probleme o serie de servicii descentralizate care accelerează dezvoltarea aplicațiilor descentralizate, permit funcționalitatea încrucișată și armonizează tehnologiile.

Modul de functionare

ChainLink Core - software-ul nodului de bază este responsabil pentru interfața cu blockchain-ul, planificarea și echilibrarea activității între diferitele sale servicii externe. Joburile efectuate de nodurile ChainLink sunt formatate ca sarcini. Fiecare sarcină este un set de specificații mai mici de lucrări, cunoscute sub numele de sarcini secundare, care sunt procesate ca un pipeline. Fiecare sarcina secundară are o operație specifică pe care o efectuează, înainte de a-și transmite rezultatul pe următoarea sarcina și, în cele din urmă, atinge un rezultat final.

Software-ul de nod ChainLink vine cu câteva sub-sarcini încorporate, inclusiv cereri HTTP, analiză JSON și conversie în diferite formate blockchain. Dincolo de tipurile de sub-sarcini încorporate, sub-sarcinile personalizate pot fi definite prin crearea de adaptoare.

Adaptoarele sunt servicii externe cu un API REST minim. Modelând adaptoarele într-o manieră orientată spre servicii, programele pot fi manipulate cu ușurință prin simpla adăugare a unui API intermediar în fața programului. În mod similar, interacțiunea cu API-urile complexe poate fi simplificată în sub-sarcini individuale parametrizate.[8]

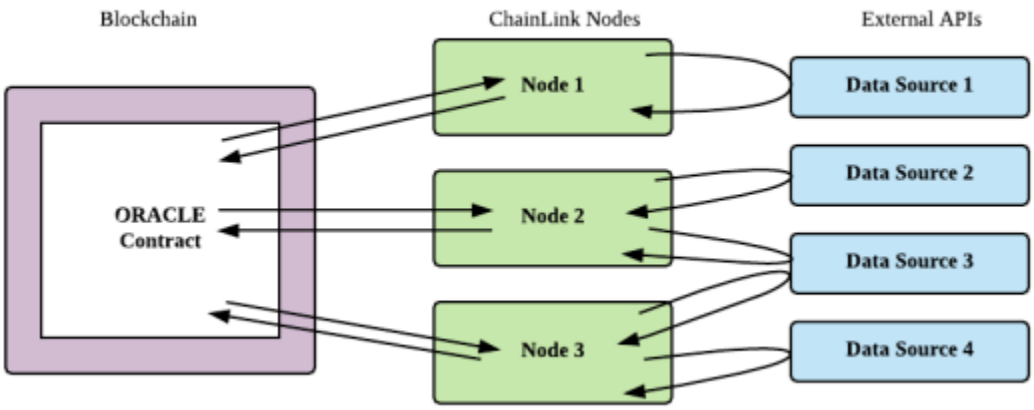

Figura I.7 Distribuirea informațiilor prin chainlink, preluat din [8]

II. Tehnologii utilizate pentru implementarea aplicației

2.1. Limbajul Solidity

Solidity este un limbaj de programare la nivel înalt pentru implementarea contractelor inteligente. Solidity este puternic influențat de C++, Python și JavaScript și a fost proiectat pentru a viza mașina virtuală Ethereum.

Solidity este scris static, acceptă moștenirea, bibliotecile și limbajele de programare complexe definite de utilizator. Se poate folosi pentru a crea contracte pentru utilizări în votare, crowdfunding, licitații și portofele cu mai multe semnături.[9]

2.2. HTML, CSS și JavaScript

HyperText Markup Language (HTML) este un limbaj de marcare utilizat pentru crearea paginilor web ce pot fi afișate într-un browser (sau navigator). Scopul HTML este mai degrabă prezentarea informațiilor – paragrafe, fonturi, tabel, descrierea semanticii documentului.[10]

CSS sau Cascading Style Sheets este un standard pentru formatarea elementelor unui document HTML. CSS se poate utiliza și pentru formatarea elementelor XHTML, XML și SVG. Acesta permite separarea și prezentarea vizuală a conținutului unei pagini web, inclusiv culorile și fonturile disponibile. Separarea elementelor unei pagini îmbunătățește accesibilitatea paginii și permite o mai bună flexibilitate și un control în specificațiile caracteristicilor de prezentare.[11]

JavaScript (JS) este un limbaj de programare orientat obiect bazat pe conceptul prototipurilor. Este folosit mai ales pentru introducerea unor funcționalități în paginile web, codul JavaScript din aceste pagini fiind rulat de către browser. Limbajul este binecunoscut pentru folosirea sa în construirea siturilor web, dar este folosit și pentru accesul la obiecte încapsulate (embedded objects) în alte aplicații, folosit adesea pe server, utilizând un mediu de rulare, cum ar fi Node.js.

Cea mai des întâlnită utilizare a JavaScript este în scriptarea paginilor web. Programatorii web pot îngloba în paginile HTML script-uri pentru diverse activități cum ar fi verificarea datelor introduse de utilizatori sau crearea de meniuri și alte efecte animate.

Browserele rețin în memorie o reprezentare a unei pagini web sub forma unui arbore de obiecte și pun la dispoziție aceste obiecte script-urilor JavaScript, care le pot citi și manipula. Arborele de obiecte poartă numele de Document Object Model sau DOM.[12]

2.3. Bootstrap

Bootstrap este un framework CSS opened-source, direcționat către dezvoltarea web front-end receptivă și mobilă. Conține șabloane de design bazate pe CSS și JavaScript pentru tipografie, formulare, butoane, navigare și alte componente de interfață. Este ușor de integrat și este foarte flexibil, având o librărie vastă de șabloane gratuite și profesionale.[13]

2.4. Node.js, Web3.js și Express

Node.js este o platformă de dezvoltare open source pentru rularea javascript pe partea de server. Nodul este util pentru dezvoltarea de aplicații care necesită o conexiune permanentă de la browser către server. Node.js este proiectat să funcționeze pe un server dedicat HTTP și să utilizeze un fir cu un proces pe o singură unitate de timp, este bazată pe eveniment și rulează asincron. Codul construit pe această platformă nu corespunde modelului tradițional de recepționare, procesare, trimitere, așteptare și primire. În schimb, Nodul procesează cererile primite în stivă de evenimente persistente, trimite cereri mici unul câte unul și nu așteaptă răspunsuri.[14]

Câteva dintre caracteristicile importante care fac din Node.js prima alegere a arhitecților software sunt:

- Asincron și event-driven - Toate API-urile bibliotecii Node.js sunt asincrone, adică neblocante. În esență, un server bazat pe Node.js nu așteaptă niciodată ca un API să returneze date. Serverul trece la următorul API, și folosește un mecanism de notificare a evenimentelor Node.js, care ajută serverul să obțină un răspuns din apelul API anterior.
- Single-threaded, dar foarte scalabil - Node.js folosește un model single-thread, event looping. Mecanismul de evenimente ajută serverul să răspundă într-un mod care nu blochează și face ca serverul să fie puternic scalabil, spre deosebire de serverele tradiționale care creează fire limitate pentru a gestiona cererile.
- Foarte rapid - Fiind construit pe motorul JavaScript V8 al Google Chrome, biblioteca Node.js este foarte rapidă în executarea codului.
- Fără buffering - aplicațiile Node.js nu „tamponează” niciodată date. Aceste aplicații pur și simplu emit datele în bucăți.

Express.js, sau pur și simplu Express, este un framework de aplicații web back-end pentru Node.js, lansat ca software gratuit și open-source sub licența MIT. Este conceput pentru a construi aplicații web și API-uri. A fost numit de facto cadrul standard de server pentru Node.js. Express este un cadru de aplicații web minimal și flexibil Node.js care oferă un set robust de caracteristici pentru aplicații web și mobile.[15]

Avantaje:

- Realizează rapid și ușor dezvoltarea aplicațiilor web Node.js.
- Ușor de configurat și personalizat.
- Permite să definiți rutele aplicației pe baza metodelor HTTP și a adreselor URL.
- Include diverse module middleware pe care le puteți utiliza pentru a efectua sarcini suplimentare la cerere și răspuns.
- Permite să crearea unui server REST API.
- Ușor de conectat cu baze de date precum MongoDB, Redis, MySQL

Biblioteca web3.js este o bibliotecă JavaScript open source (GNU Lesser General Public License versiunea 3) construită de Ethereum Foundation și include funcții pentru a comunica cu un nod Ethereum prin protocolul JavaScript Object Notation - Remote Procedure Call(JSON-RPC). Cu alte cuvinte, este o bibliotecă JavaScript care permite unui dezvoltator să interacționeze cu blockchain-ul Ethereum.[16]

Cele mai utilizate funcții din Web3.js sunt umătoarele:

- web3.eth.getAccounts() – obținerea tuturor adreselor (conturilor) valabile
- web3.eth.sendTransaction() – folosită pentru a trimite etheri, sau altă monedă validă
- web3.eth.estimateGas() – estimează gas-ul consumat de tranzacția curentă
- web3.eth.contract() – încarcă contractul în aplicația descentralizată
- web.utils.toWei() – convertește etherii in wei (unitate de valoare folosită de contractele inteligente)
- web3.eth.getBalance() - permite obținerea soldului ETH al unei adrese la un anumit bloc
- web3.eth.signTransaction - permite să semnarea unei tranzacții

2.5. Truffle Suite

Truffle Suite este pachetul de instrumente dedicat dezvoltatorilor Ethereum, concentrându-se pe contractele inteligente. Acest proiect dezvoltat foarte activ este în prezent unul dintre cele mai populare cadre pentru implementarea, testarea și rularea aplicațiilor descentralizate bazate pe Ethereum.

Truffle este un mediu de dezvoltare de clasă mondială, un cadru de testare și pipline pentru blockchain-uri folosind Ethereum Virtual Machine (EVM), cu scopul de a face viața de dezvoltator mai ușoară. Truffle este cel mai popular framework pentru dezvoltarea contractelor inteligente. Acesta facilitează management-ul unui contract inteligent, având facilitatea de a compila, migra și testa contracte în rețea. [17]

2.6. MongoDB

MongoDB este o bază de date NoSQL orientată spre documente, utilizată pentru stocarea volumului de date. În loc să folosească tabele și rânduri ca în bazele de date relaționale tradiționale, MongoDB folosește colecții și documente. Documentele constau din perechi cheie-valoare care reprezintă unitatea de bază a datelor din MongoDB. Colecțiile conțin seturi de documente și funcții, care este echivalentul tabelelor de baze de date relaționale.[18]

2.7. Docker

Docker este o platformă software pentru construirea de aplicații bazate pe containere - medii de execuție mici și ușoare care utilizează în comun sistemul kernelului de operare, dar care se execută în mod izolat unele de altele. Docker oferă posibilitatea de a împacheta și rula o aplicație într-un mediu izolat, numit container. Izolarea și securitatea vă permit să rulați mai multe containere simultan pe o anumită gazdă. Containerele sunt ușoare și conțin tot ce este necesar pentru a rula aplicația, ignorand configurația ce este instalată în prezent pe gazdă. [119]

2.8. Infura și Metamask

Infura oferă instrumentele și infrastructura prin intermediul careia permit dezvoltatorilor să își dezvolte cu ușurință aplicația blockchain de la testare la implementare la scară - cu acces simplu și fiabil la Ethereum și IPFS.[20]

Metamask este o extensie de browser ce îți permite să rulezi o aplicație descentralizată.[21]

2.9. Referințe la teme/subiecte conceptual similare

O prezentare amănunțită a modului în care tehnologia blockchain ar putea fi aplicată atât în domeniul industriei, cât și în alte domenii importante, precum agricultura și sănătatea, este descrisă în articolul **Blockchain for Industry 4.0: A Comprehensive Review** [2].

În domeniul alimentației, lanțul de magazine american Walmart, în colaborare cu IBM, au creat o soluție descentralizată pentru asigurarea siguranței alimentelor. Sistemul creat este bazat pe tehnologia open source Hyperledger Fabric și permite urmărirea procesului lanțului de aprovizionare cu alimente. Fiecare nod de pe Blockchain reprezintă o entitate care a manipulat alimentele în drum spre magazin, făcând mult mai ușor și mai rapid să se vadă dacă una dintre ferme a vândut un lot cu probleme într-o anumită locație. Hyperledger Fabric este o implementare plug and play a tehnologiei blockchain, concepută ca o bază pentru dezvoltarea aplicațiilor blockchain cu un grad mare de scalabilitate și un grad flexibil de permisiuni**.**

În **Implementation of smart contracts for blockchain based IoT applications** este propusă o rețea descentralizată de tipul Sensing as a Service, bazată pe Blockchain. Scopul acestui studiu este crearea unei aplicații descentralizate bazată pe rețeaua blockchain ethereum, care să permită utilizatorilor să cumpere și să vândă cu ușurință datele senzorului IoT, utilizând un jeton (engl. Token) personalizat ca monedă de plată.[22]

Un exemplu de implementare strâns legat de tema aleasă este **A case study for blockchain in manufacturing: 'FabRec': A prototype for peer-to-peer network of manufacturing nodes**, autorii acestei lucrări descriind dezvoltarea unei aplicații ce oferă transparență între fabricanți și clienți. Spre deosebire de autorii acestei lucrări, soluția descrisa de mine prezinta o soluție completă și ușor scalabilă cât și mai sigură întrucât informațiile din exterior sunt verificate în mediul chainlink înainte de a fi adaugate in bloc. [23]

În Automotive autorii lucrării **A Review on Blockchain Technologies for an Advanced and Cyber-Resilient Automotive Industry** prezintă o abordare conceptuala a unui blockchain pentru industria auto, cu o perspectivă a scenariilor principale și strategiilor de optimizare pentru proiectare și implementarea acestor aplicații.[24]

Astfel de implementări, deși nu sunt abordate încă la scară largă, pot oferi o idee despre cum transparența datelor ar putea oferi servicii/produse de o calitate superioară.

III. Proiectarea aplicației

Aplicația este formată din trei componente: aplicația web client/administrator componenta Blockchain și componenta fizica. Comunicarea componentelor s-a făcut prin intermediul unui Ethereum Javascript API, întreg mecanismul fiind descris în capitolele următoare. Aplicația a fost dezvoltată pe un computer cu sistemul de operare Ubuntu 18.04, cu un procesor quad-core Intel Core i5 cu o memorie ram de 8GB, și poate fi folosită pe majoritatea sistemelor de calcul care dispun de o configurație similară sau mai buna și un sistem de operare performant. Cele trei componente fundamentale ale aplicației vor fi analizate detaliat în capitolul următor.

Pentru a simula un sistem descentralizat, voi crea o rețea blockchain restransa conținând un computer și un arduino care acționează ca participanți la blockchain reprezentând diferiți furnizori de servicii de fabricație. Unul dintre noduri este conectat la un Arduino care este interfațat cu rețeaua blockchain printr-un Oracle, care permite computerului să citească și să scrie evenimente la costuri reduse.

3.1. Proiectarea componentei Blockchain

Componenta Blockchain a aplicației a fost dezvoltată folosind limbajul Solidity și cuprinde regulile unui mecanism de automatizat și securizat, incluse în contractele inteligente.

3.2. Proiectarea componentei Client

Componenta client a fost creată ca o interfață de prezentare ușor de folosit și prietenoasă cu orice tip utilizator, accentul este pus pe fluiditatea mecanismului și pe designul modern, dar simplu. Principalele funcționalități ale aplicației sunt:

- Autentificarea utilizatorului
- Vizualizarea fabricilor listate
- Crearea unui profil reprezentativ unei fabrici

3.2.1. Interfața grafică

Pentru interfața grafică a fost folosit framework-ul ReactJS, utilizând pe partea de backend Node.js alături de libraria express. S-a urmărit simplitatea și claritatea mediului de control, rapiditatea dezvoltării și flexibilitatea răspunsului la evenimente utilizatorului, astfel framework-ului React respectă toate criteriile formulate inițial și prin natura designului său oferă o gamă largă de componente reutilizabile. Randarea acestora este rapidă, astfel orice schimbare este fluid afișată, atribute esențiale atât pentru dezvoltatori, cât și pentru utilizatori.

3.2.2. Integrare componentei Blockchain cu aplicația client

Biblioteca Web3.js este utilizată pentru a integra componentele principale ale aplicației, care simplifică mecanismul contractelor inteligente și reduce comportamentul acestora la simple obiecte de tip JavaScript. Odată ce definim un furnizor, o entitate, putem stabili o conexiune cu

blockchain-ul care comunică cu node-ul Ethereum și transferă date, din cod mașină în comenzi Web3.

3.3. Proiectarea componentei fizice

Configurarea este menită să demonstreze modul în care un contract inteligent poate verifica anumite evenimente, o comandă trimisă către un dispozitiv fizic, care în cazul nostru este o simplă lumină LED atașată la o placă de dezvoltare Arduino.

Atât adaptorul extern, cât și API-ul ce se conectează la interfața serială a plăcii Arduino Uno, sunt scrise în NodeJs și reprezintă un strat suplimentar în conectarea oracolului la lumea exterioară.

3.4. Diagrame UML

Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele și specificații pentru software. Limbajul a fost creat de către consorțiul Object Management Group (OMG) care a mai produs printre altele și standardul de schimb de mesaje între sisteme CORBA. UML a fost la bază dezvoltat pentru reprezentarea complexității programelor orientate pe obiect, al căror fundament este structurarea programelor pe clase, și instanțele acestora (numite și obiecte). Cu toate acestea, datorită eficienței și clarității în reprezentarea unor elemente abstracte, UML este utilizat dincolo de domeniul IT.

UML oferă o largă gamă de diagrame pentru modelarea diferitelor situații în cadrul unui proiect de dezvoltare software.

3.4.1. Diagrama state-machine

Diagramele de stare sunt grafuri, în care nodurile sunt stări și arcele direcționate sunt tranziții, având ca etichete numele evenimentelor care le-au provocat. Se precizează acțiunile rezultate din aceste schimbări. Starea corespunde unui interval de timp în care obiectul satisface anumite condiții, efectuiază anumite acțiuni sau așteaptă un eveniment.

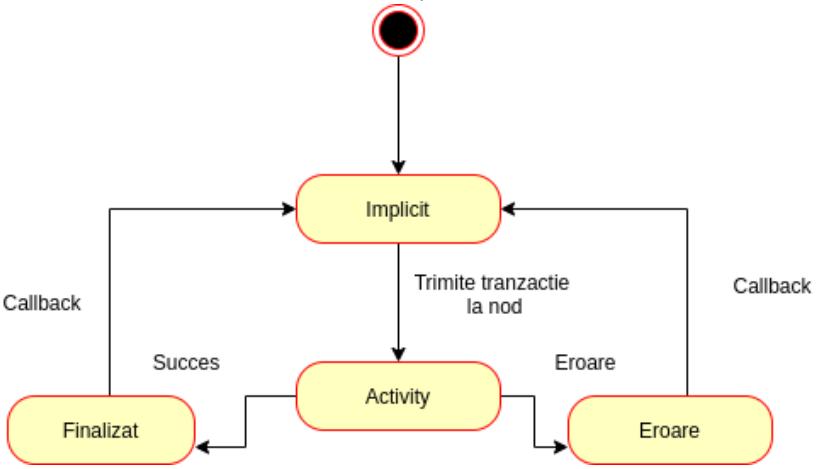

Fig. III.1 Diagrama state-machine

3.4.2. Diagrama use-case

Datorită simplității ei este utilizată în special în discuțiile dintre firma dezvoltatoare de software și clienți sau utilizatori. Diagramele use case reprezintă interacțiunea dintre elementele exterioare unui sistem (numite și actori) și sistem. În cazul acestor diagrame se prezintă acțiunea desfășurată de sistem la interacțiunea actorului, însă modul în care sistemul desfășoare acea acțiune nu trebuie să fie reprezentat într-o astfel de diagramă (conceptul blackBox).

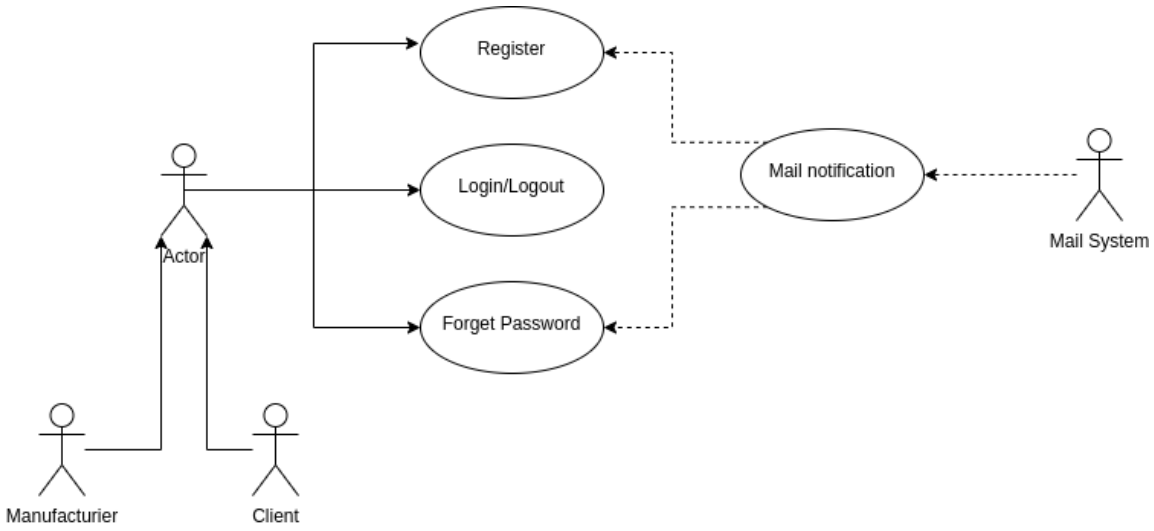

Fig. III.2 Diagrama use-case corespunzătoare funcționalității de înregistrare și conectare

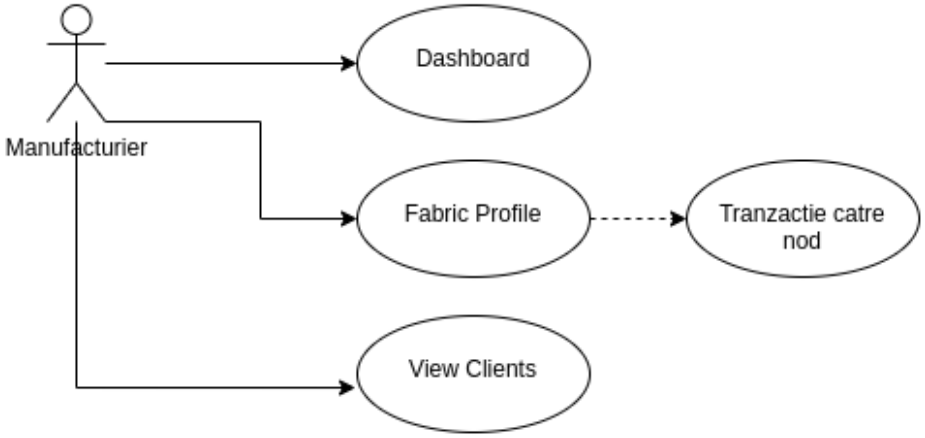

Fig. III.3 Diagrama use-case corespunzătoare funcționalității proiectate pentru fabricant

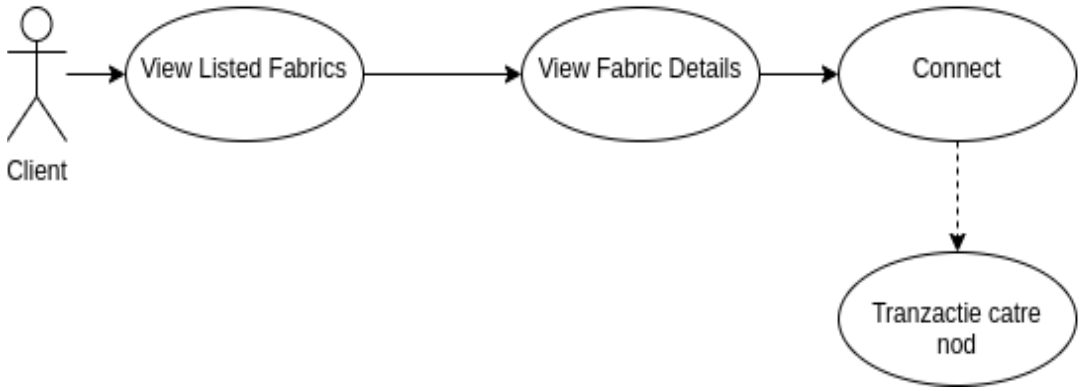

Fig. III.4 Diagrama use-case corespunzătoare funcționalității proiectate pentru client

3.4.3. Diagrama de secvență

Este utilizată în special în timpul dezvoltării de subsisteme, și ajută la identificarea relațiilor dintre obiecte pe parcursul efectuării unei anumite operații.

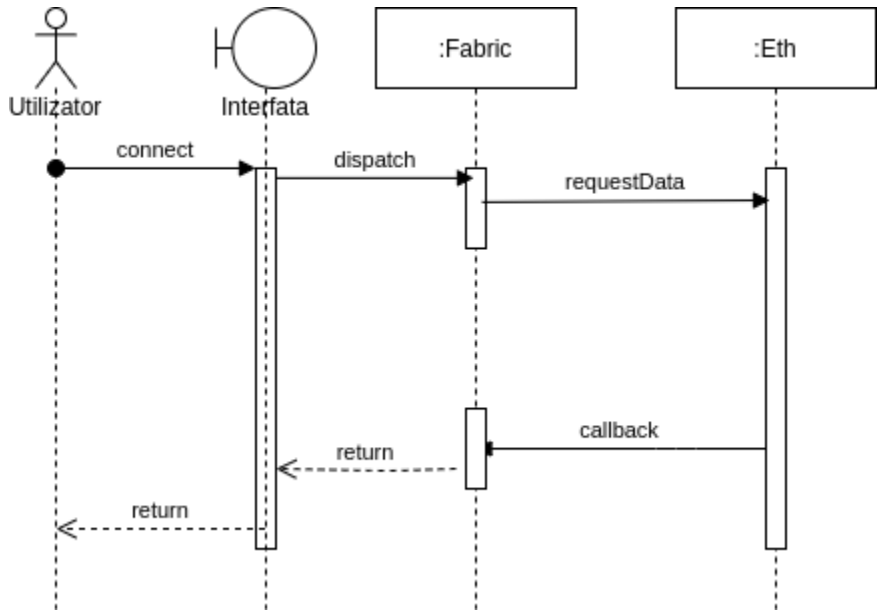

Fig. III.5 Diagrama de secvență pentru procesul de comanda

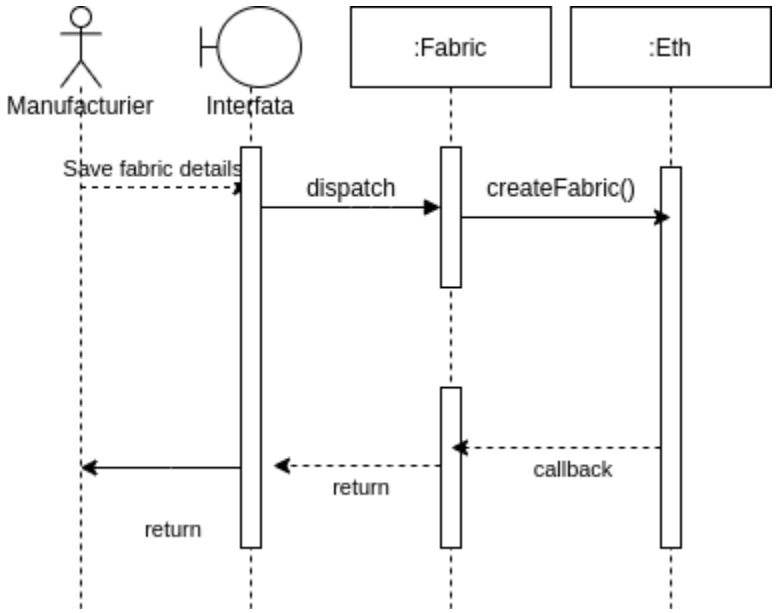

Fig. III.6 Diagrama de secvență pentru procesul de creare a profilului fabricantului

3.4.4. Diagrama de clase

Este folosită pentru reprezentarea vizuală a claselor și a interdependențelor, taxonomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte, și a legăturilor concrete dintre acestea.

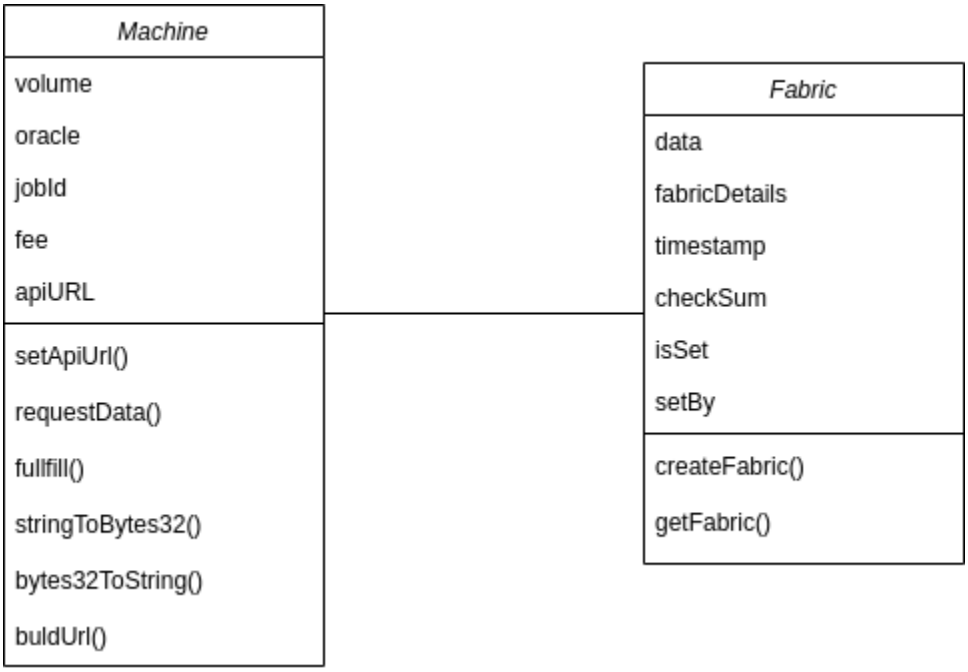

Fig. III.7 Diagrama de clase pentru contractul inteligent

3.5. Avantajele și dezavantajele metodelor alese

Datorită tehnologiei blockchain, metoda de proiectare aleasă oferă un grad ridicat de transparență și securitate, iar costul tranzacției este mai mic în comparație cu sistemele uzuale utilizate în prezent. Deși această tehnologie pare a fi o metodă ideală pentru un sistem incoruptibil, este greu de scalat din cauza mecanismelor de consens, iar unele probleme de calcul consumă prea multă energie. Este de asemenea o soluție mai sigură decât alte platforme, dar susceptibil la alte vulnerabilități, cum ar fi:

- atacul 51% (o entitate/grup deține mai mult de 51% din controlul rețelei).
- double-spending, posibil pentru rețelele cu vulnerabilitate la atacul 51%, și este prevenit cu ajutorul mecanismelor de consens.
- cryptographic cracking, deoarece algoritmii sau computațiile cuantice sunt extrem de avansate și pot găsi soluția criptografică.

IV. Implementarea aplicației

Această lucrare se concentrează pe dezvoltarea unei aplicații descentralizate bazată pe Chainlink și contracte inteligente și este implementată folosind protocolul și rețeaua ethereum prin care utilizatorii și furnizorii de servicii pot coopera într-un mod descentralizat și sigur. Scopul acestei aplicații este de a crea o piață descentralizată pentru fabricanți și clienți.

Fiecare participant la rețea va avea diferite roluri și responsabilități care le permit să interacționeze cu alți membri ai rețelei. Un participant poate fi reprezentat de un nod, o mașină sau un agent reprezentând organizația; orice participant este identificat pe rețea cu o adresă unică. Utilizatorii, cum ar fi designerii care caută un furnizor de servicii, pot fi participanții care accesează informații precum verificarea disponibilității unui un anumit producător sau care comandă un serviciu după cum se poate observa în Fig. IV.1.

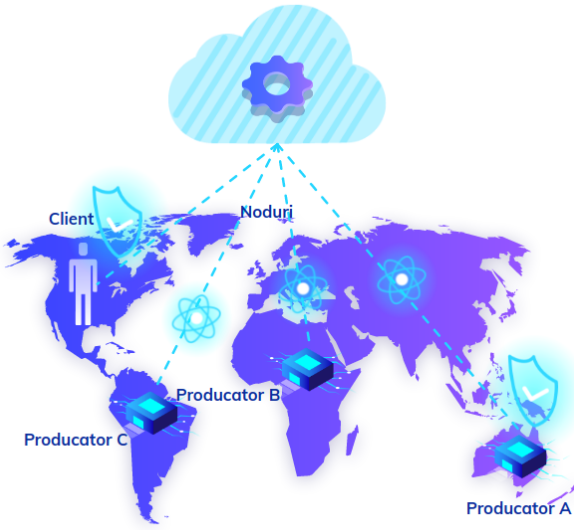

Fig. IV.1 Prezentare generală a participanților în rețea

4.1. Implementarea componentei Blockchain

Componenta Blockchain conferă funcționalitatea principală a aplicației, deoarece operațiile executate în cadru acesteia au loc sub constrângerea unor reguli bine definite, care, încărcate în rețea, devin imutabile. Aceste reguli sunt încapsulate în contactele inteligente.

Solidity este un limbaj de programare specializat creat pentru scrierea contractelor inteligente. Pentru a fi compilarea contractelor se poate folosi doar compilatorul solidity, însă suita truffle ușurează acest mecanism, odată cu procesul de migrare, prezentat în cadrul acestui subcapitol.

Contractul Fabric este folosit pentru crearea profilului fabricii salvând informațiile în blockchain aflându-se și într-un cadru imutabil, iar transmiterea informațiilor pentru prelucrări și afișări ulterioare este foarte ușoară. Acesta conține mecanismul de comanda pentru elementul

fizic cat și posibilitatea de a retrage fondurile primite. Funcția de creare a profilului fabricii este descrisă în codul de mai jos.

```
contract Fabric is ChainlinkClient{
    bytes32 private check;
    address private oracle;
    bytes32 private jobId;
    uint256 private fee;
    string private apiUrl;

  constructor() public {
        struct MyFabricEntry {
        string data;
        string fabricDetails;
        uint256 timestamp;
        bytes32 checkSum;
        bool isSet;
        address setBy;
    }
    mapping(bytes32 => MyFabricEntry) public myFabric;
    function createFabric(
        bytes32 _checksum,
        string memory _data,
        string memory _fabricDetails
    ) public {
        require(!myFabric[_checksum].isSet);

        myFabric[_checksum].isSet = true;
        myFabric[_checksum].fabricDetails = _fabricDetails;
        myFabric[_checksum].timestamp = block.timestamp;
        myFabric[_checksum].setBy = msg.sender;

        emit NewEntry(
            _checksum,
            _data,
            _fabricDetails,
            msg.sender
        );
    }
```

Definirea unui contract în solidity

```
var Fabric = artifacts.require("./Fabric.sol");
module.exports = function(deployer) {
  deployer.deploy(Fabric);
}
```

Migrarea contractului

4.2. Implementarea aplicației client

React folosește componente care pot fi reutilizate pentru a ușura munca dezvoltatorului sau pentru a obține o interfață fluidă. Acestea returnează doar un element DOM, care este randat. În fiecare componetă se pot defini stări, prin declararea unei variabile de stare și a metodei ce va schimba această variabilă în urmă unei acțiuni și inițializarea primei stări a variabilei. În urma apariției unui eveniment și schimbarea stării variabilei de stare, se produce adesea și o schimbare la nivelul interfeței grafice.

Componenta de rutare este poate cea mai importantă componentă din React Router. Responsabilitatea sa de bază este de a reda unele elemente de interfață atunci când calea sa se potrivește cu adresa URL curentă. Componenta de rutare din cadrul aplicației client:

Definirea rutelor:

```
ReactDOM.render(
 <BrowserRouter>
   <Switch>
     <PrivateRoute path="/admin" component={AdminLayout} />
     <PrivateRoute path="/user" component={UserLayout} />

     <AuthRoutes path="/auth" component={AuthLayout} />
     <Redirect from="/" to="/auth" />
     <Redirect from="/admin" to="/admin/index" />
     <Redirect from="/user" to="/user/index" />

   </Switch>
 </BrowserRouter>,
 document.getElementById("root")
);
```

Definirea rutelor modulului Root

Implementarea serviciilor ce interacționează cu o bază de date

Pentru implementarea modulului de interacțiune cu baza de date am folosit Mongoose ce oferă o soluție simplă, bazată pe scheme, pentru a modela datele aplicației. Include type casting, validarea și crearea de interogări. Un model este o clasă cu care se pot construi documente. Aplicația se folosește de o baza de date nerelațională întrucât este mai ușor scalabilă în viitor și cu performanțe mai bune în comparație cu alte soluții.

```
const mongoose = require('mongoose');
const UserSchema = new mongoose.Schema({
 name: {
   type: String,
   required: true,
 },
 email: {
   type: String,
   required: true,
 },
 password: {
   type: String,
   required: true,
 },
 accountType: {
   type: String,
   required: true,
 },
 accountConfirmation: {
   type: Boolean,
   default: false,
 },
});
const User = mongoose.model('User', UserSchema);
module.exports = User;
```

Definirea unui model utilizând mongoose.

4.3. Implementarea componentei fizice

Arhitectura la nivel de producător, furnizor de servicii de fabricație poate avea sub proprietatea sa mai multe mașini de fabricație care oferă capacitatea și capacitatea necesare pentru a-și servi clienții. Fiecare dintre aceste mașini cu o identitate unică va avea capacitatea de a scrie direct „evenimente” critice în blockchain-ul privat la care participă producătorul. Evenimentele înregistrate de mașină sunt transferate geamănului său virtual, fiind astfel atașate bazei de date blockchain.

Am utilizat Docker pentru a rula nodul chainlink versiunea 0.8.18, pentru a fi cât mai apropiat de un environment gata de implementare. Imaginea este compusă din doua componente, un server web și o bază de date. Conexiunea dintre nod și adaptorul extern, se face printr-un bridge, ce permite interacțiunea oracolului cu informațiile din exterior. Conexiunea la rețeaua blockchain se realizează printr-un endpoint Infura.

Atât adaptorul extern, cât și API-ul ce se conectează la interfața serială a plăcii Arduino Uno, sunt scrise în NodeJs și reprezintă un strat suplimentar în conectarea oracolului la lumea exterioară.

```
// the setup function runs once when you press reset or power the board
#include <ArduinoJson.h>
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  Serial.begin(9600); // Begen listening on port 9600 for serial
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
}

// the loop function runs over and over again forever
void loop() {

   if(Serial.available() > 0) // Read from serial port
    {
      char ReaderFromNode; // Store current character
      ReaderFromNode = (char) Serial.read();
      convertToState(ReaderFromNode); // Convert character to state
    }
  delay(1000);
}

void convertToState(char chr) {
  if(chr=='o'){
    digitalWrite(LED_BUILTIN, HIGH);
    delay(100);
    DynamicJsonDocument jDocument(200);

      jDocument["Masina"] = "Simens";
      jDocument["Tip"] = "CNC";
      jDocument["stare"] = "ocupat";
      serializeJson(jDocument, Serial);;
      Serial.println();
  }
  if(chr=='f'){
    digitalWrite(LED_BUILTIN, LOW);
    delay(100);
    DynamicJsonDocument jDocument(200);

      jDocument["Masina"] = "Simens";
      jDocument["Tip"] = "CNC";
      jDocument["stare"] = "liber";
      serializeJson(jDocument, Serial);;
```

```
      Serial.println();
  }
}
```

Definirea interacțiunii modulului hardware

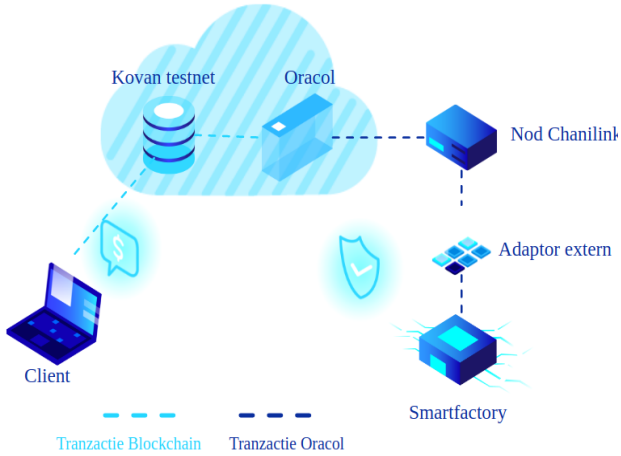

Fig. IV.2. Cursul tranzacțiilor în cazul unei cereri realizate cu succes.

Aceasta la momentul interogării returnează un JSON cu datele sale și starea actuală, care este returnat printr-un callBack drept rezultat al contractului. Procesul este descris schematic în Fig. IV.2, și reprezintă parcursul cererii formulate în contractul inteligent în cazul în care tranzacția este realizată cu succes.

API-ul reprezintă infrastructura informațională a fabricii, acesta este responsabil cu manipularea stării echipamentului fizic.

```
function readSerial () // for reading
{
    arduinoSerialPort.on('data', async function(data)
    {
        console.log(String(data));
    });
        console.log('Recived');
}
app.get('/:action', function (req, res) {
    var action = req.params.action || req.param('action');
    if(action == 'on'){
        arduinoSerialPort.write("o");
        readSerial();
        sleep(2000);
        return res.send(JSON.parse(data));
    }
    if(action == 'off') {
        arduinoSerialPort.write("f");
        readSerial();
```

```
        sleep(2000);
        return res.send(JSON.parse(data));
    }
        return res.send('Action: ' + action);
});
```

Definirea API-ului de control al modulului hardware

4.4. Integrare componentei Blockchain cu aplicația client

Liantul dintre componenta Blockchain și aplicația client este reprezentat de Web3.js, disponibil prin managerul de pachete Node.js (npm).

Acesta consta într-un serviciu dezvoltat în node.js care furnizează mijloacele necesare interactionarii cu contractul dezvoltat. Truffle oferă posibilitatea creării directoarelor contacts (care va stoca toate fișierele .sol), migrations (care va conține contractele migrate) și test (pentru eventuale teste ulterioare).

Migrarea contractelor include și compilarea acestora, astfel acestea nu pot fi folosite în rețea dacă conțin erori de sintaxă.

Serviciul reprezinta modul de comunicare dintre contracte și interfața grafică, conține toate funcțiile indispensabile proiectului: prelucrarea obiectul de configurare ABI al tuturor claselor este generat din fișierele .json obținute în urma migrării, cu ajutorul unui script, și metode de encripție, decripție și hashing a unei informații.

```
app.post("/api/blockchain", async (req, res) => {
    const id_Fabric = req.body.id;
    const email = req.body.email;
    const fabricDetails = req.body.details;

    email_hashed = Buffer.from(email).toString("base64");

    if (email && id_Fabric) {
      const dataToBeHashed = email + "_" + id_Fabric;

      const address =
        "0x" +
        crypto
          .createHash("sha256")
          .update(dataToBeHashed + "test")
          .digest("hex");

      const gasPrice = await web3.eth.getGasPrice();
      console.log('address,id:',address,id_Fabric);

      const gasEstimate = await contract.methods
        .createFabric(address, email_hashed, fabricDetails)
        .estimateGas({ from: web3.eth.defaultAccount });
```

```
    contract.methods
      .createFabric(address, email_hashed, fabricDetails)
      .send({
        from: web3.eth.defaultAccount,
        gasPrice: gasPrice,
        gas: gasEstimate,
      })
      .on("transactionHash", async function (txn_hash) {
        console.log('THX\n', txn_hash);
                  res.status(200).send({ txn_hash });
      })
      .on("receipt", function (receipt) {
        console.log("receipt", receipt,typedef(receipt));
      })
      .on("confirmation", function (confirmationNumber, receipt) {
        console.log("confirmationNumber", confirmationNumber);
      })
      .on("error", function (error, receipt) {
        throw error;
      });
  } else {
    res.sendStatus(401);
  }
         User.findOne({_id: id_Fabric}), function (err, user) {
          if (err) return console.error(err);
          console.log(user);}
});
```

Definirea componentei de creare a profilului fabricii

4.5. Interfața cu utilizatorul

4.5.1. Înregistrarea/Autentificarea utilizatorului

Autorizarea și autentificarea joacă ambele roluri vitale în configurarea mecanismului de securitate. Autorizarea este procesul de a permite utilizatorilor să acceseze obiecte de sistem pe baza identităților lor.

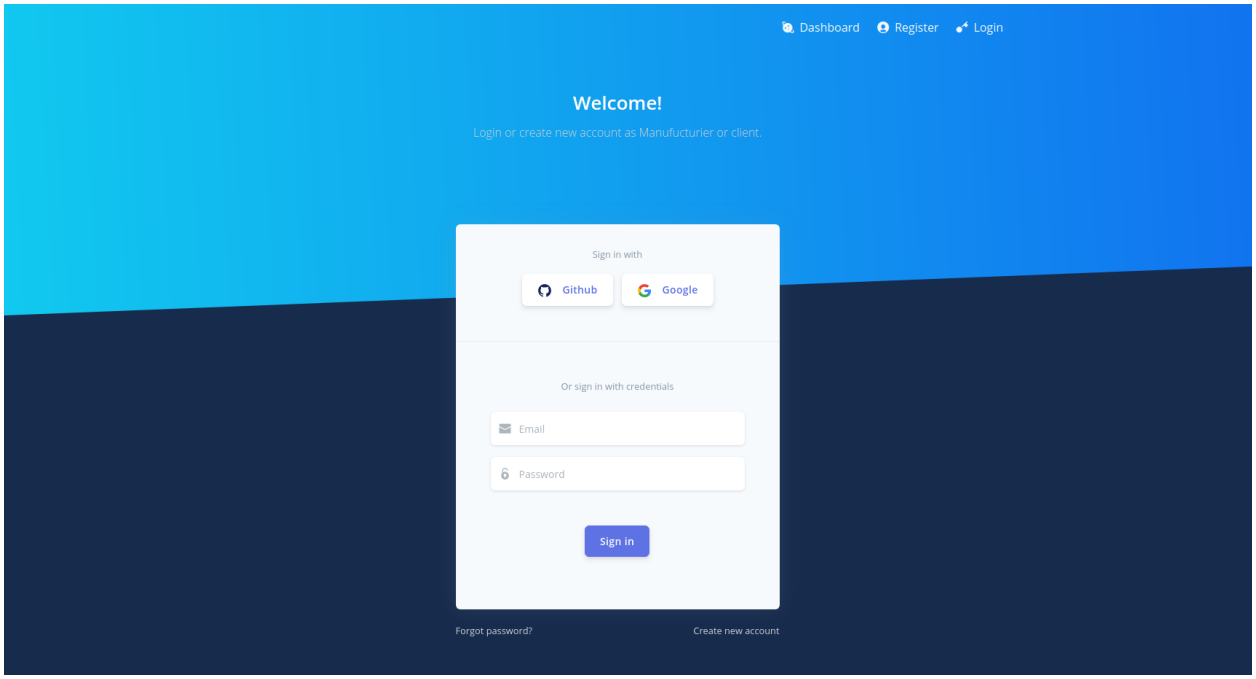

Fig. IV.3 Formularul de logare

```
const Login = props => {

   const [email, setEmail] = useState("");
   const [password, setPassword] = useState("");
   const [error, setError] = useState("");

   const tryLogin = async () => {
       const response = await login(email, password);
       const {data} = response;
       if (data.success) {
           setError("");
           localStorage.setItem("token", data.token);
           localStorage.setItem("user", JSON.stringify(data.user));
           props.history.push("/");
       } else {
           setPassword("");
```

```
            setError(data.msg);
        }
    }
```

Definirea formularului de logare

4.5.2. Crearea unei fabrici digitale

Crearea unui profil de fabricant presupune autentificarea în aplicație și completarea detaliilor fabricii alături de adresele echipamentelor fizice.

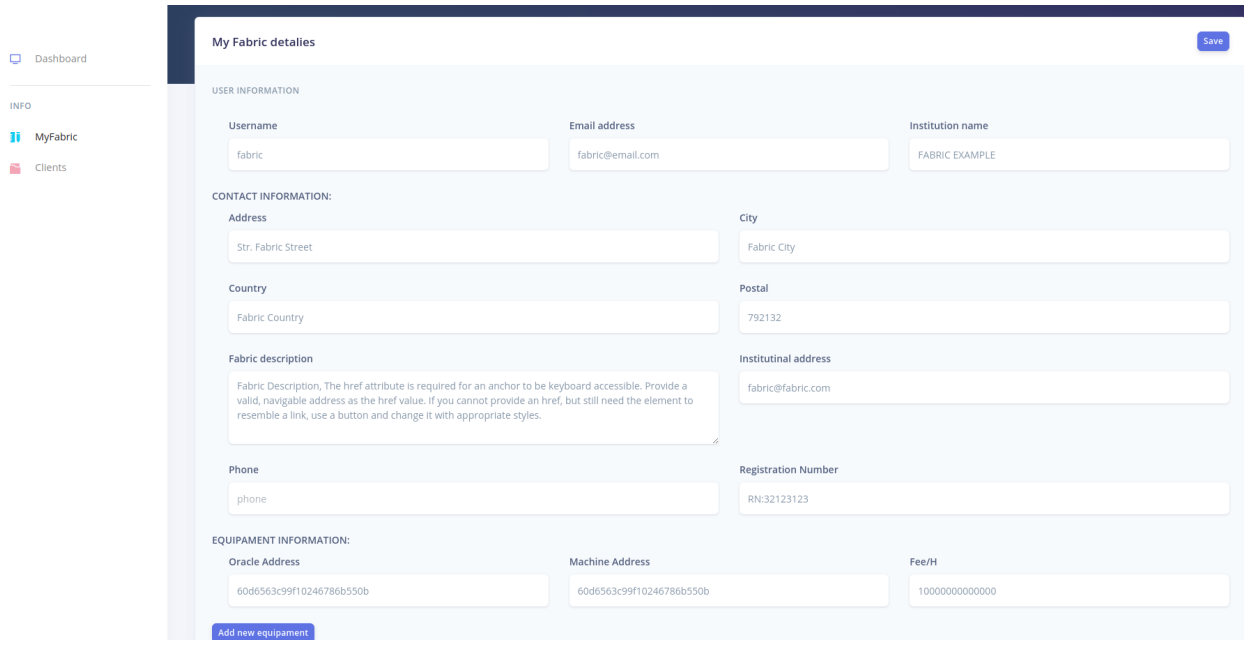

Fig. IV.4 Generarea profilului fabricii

```
const createFabric = async () => {
       const response = await edit(user._id,name, email, iname, address,
city,country,postal,description,semail,phone,rn,oracle,jobId,fee);
       const { data } = response;
       const detalies =iname+ "+"+address+ "+"+city+ "+"+country+ "+"+postal+
"+"+description+"+"+semail+"+"+phone+"+"+rn;

       if (data.success) {
           user = {...user, name,
email,iname,address,city,country,postal,description,semail,phone,rn,oracle,jobI
d,fee}
           localStorage.setItem("user", JSON.stringify(user));
           const blockchain = await createFabricBlockchain(user._id,
```

```
email,detalies);
            const { data } = blockchain;
            if (data.success) {
                console.log(blockchain);
                localStorage.setItem("blokchain", JSON.stringify(blockchain));
            }else{
                console.log('Am întâmpinat o eroare la crearea blocului');
            }
            props.history.push("/admin/);
        }
    }
```

Definirea procesului de generare a profilului fabricantului

4.5.3. Crearea configurației hardware

Pentru efectuarea conexiunii cu mediul fizic, am dezvoltat o punte de legătură între elementul de control al stării plăcuței și mediul chainlink.

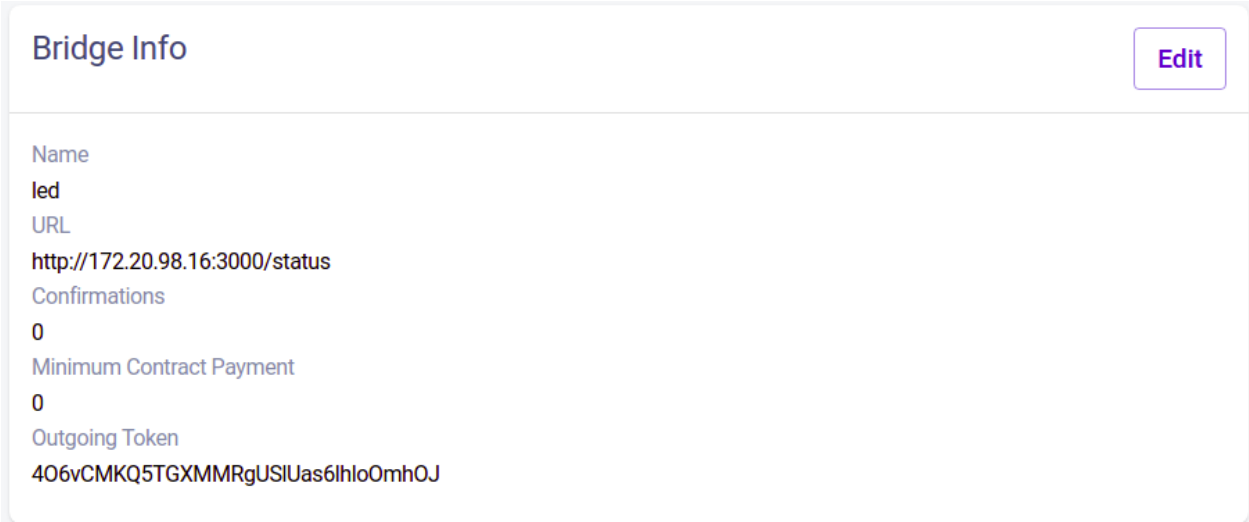

Fig. IV.5 Conexiunea de tip punte dintre elementul de control și chainlink

Acesta permite prin intermediul unor instrucțiuni predefinite sa preia informatiile primite, le prelucrează și le trimite înapoi apelantului sub forma unei tranzacții.

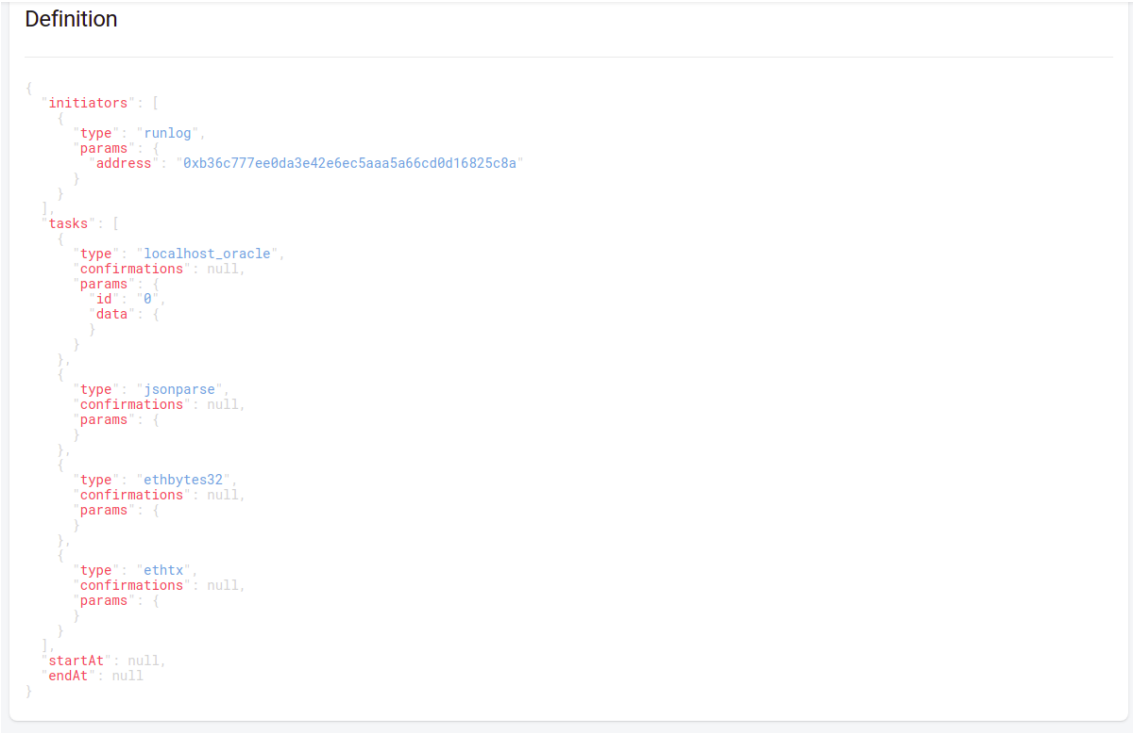

Fig. IV.6 Definiția Jobului alcătuită din subsarcini

Adaptorul extern oferă flexibilitate crescută în operarea cu diverse platforme, acesta poate fi configurat astfel incat sa intruneasca cererile oricărui API modern.

```
const { Requester, Validator } = require('@chainlink/external-adapter')
const customError = (data) => {
  if (data.Response === 'Error') return true
  return false
}
const customParams = {
  endpoint: false
}

const createRequest = (input, callback) => {
  const validator = new Validator(callback, input, customParams)
  const jobRunID = validator.validated.id
  const endpoint = validator.validated.data.endpoint || 'on'
  const url = `http://0.0.0.0:3000/${endpoint}`//url-ul api ului de control
  const params = {
  }
  const config = {
    url,
    params
```

```
  }
  Requester.request(config, customError)
    .then(response => {
      console.log('Console log of the
result',String(response.data.result['data']));
      callback(response.status, Requester.success(jobRunID, response))

    })
    .catch(error => {
      callback(500, Requester.errored(jobRunID, error))
    })
}
```

Definirea adaptorului extern

4.5.4. Plasarea unei comenzi

Plasarea unei comenzi presupune selectarea fabricii, și confirmarea tranzacției.

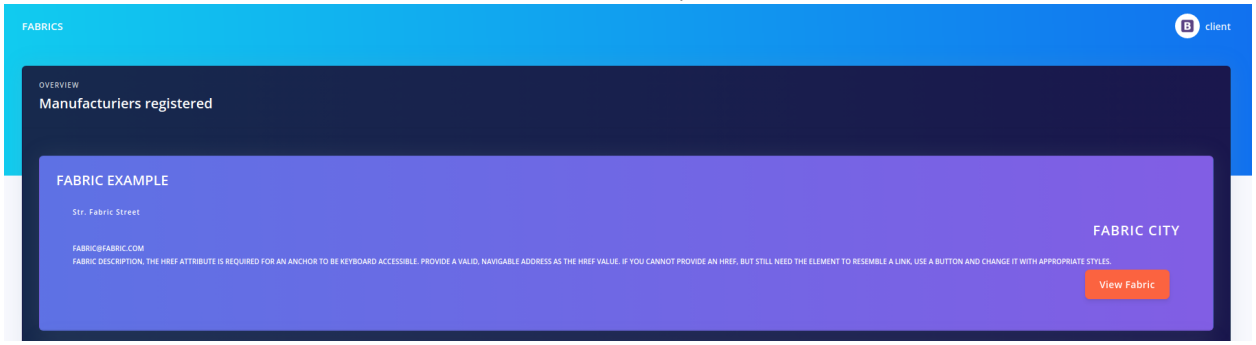

Fig. IV.7 Meniul principal al clientului

```
const Fabrics = props  => {
 const [users, setUsers] = useState([]);
 const [_id, setId] = useState("");

 const fabricDetails = async () => {
   const response = await viewDetails(_id);
   const { data } = response;
   console.log(response);
   if (data.success) {
       localStorage.setItem("selectedFabric", _id);
       console.log(response);
       props.history.push("/user/fabric-profile");
```

```
    }
}
    useEffect(() => {
        const runAsync = async () => {
            const response = await getAllFabrics();
            const {data} = response;
            console.log(data.users);
            if (data.success) {
                setUsers(data.users);
            }
        }
        runAsync();
    }, []

    );
```

Implementarea funcției de vizualizare a fabricanților disponibili

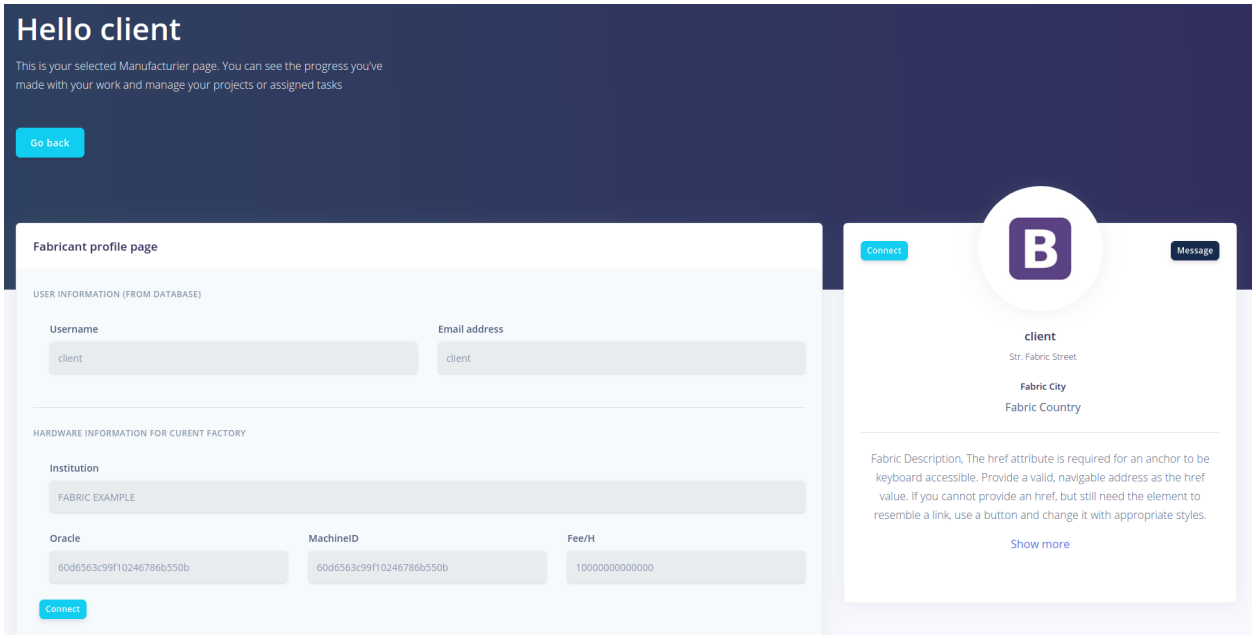

Fig. IV.8 Profilul fabricantului ales pentru plasarea comenzi

```
 const sendRequest = async () => {
            const response = await
requestHardware(user.email,fabric._id,fabric.oracle,fabric.jobId,fabric.fee
);
            const { data } = response;
            var options = {
```

```
            "headers": { "content-type": "application/json" },
            "url": "http://localhost:8100/api/blockchain/request",
            "body": JSON.stringify(data)
        };
```

Implementarea funcției de conectare la componenta hardware a fabricantului selectat

4.6. Dificultățile întâmpinate

Dificultățile apărute în cadrul proiectului au provenit atât din limitările limbajului Solidity, un limbaj încă nou și în continuu proces de îmbunătățire, din modul de implementare a aplicației și proiectare sistemului de comunicare între componente.

Una din provocările care a accelerat interesul pentru proiectarea acestei soluții a fost familiarizarea cu noile concepte din suita tehnologiei Blockchain, framework-ul React, și modelarea cerințelor pentru echipamentele fizice. S-a urmărit obținerea unei soluții cu un design atractiv pentru utilizator, cu abstractizare mecanismelor, însă cu transparența rezultatelor.

V. Testarea aplicației și rezultate experimentale

Testarea unei aplicații reprezintă un mecanism extrem de important, deoarece identifică erorile sau defectele din sistemul dezvoltat. Performanțele se îmbunătățesc considerabil dacă etapa de testare începe cât mai devreme.

5.1. Testarea componentei Blockchain

Testarea componentei Blockchain s-a făcut inițial pe platforma Remix, care este un IDE open-source pentru solidity, care permite testare contractelor din browser, odată ce acestea au fost încărcate. De asemenea, testarea compilării este făcută și în momentul în care contractele sunt migrate cu ajutorul truffle, iar testarea funcționalităților se poate face prin intermediul API-ului Ethereum cu ajutorul colecției de biblioteci oferite de Web3.js.

Fig. V.1 Tranzactia creata in momentul crearii unui nou profil de fabricant

Tranzacțiile create pot fi verificate folosind Etherscan, aceasta unealta oferind detalii despre tranzacție printre care și statusul și datele acesteia, după cum este prezentat în imaginea de mai jos .

Fig. V.2 Tranzactie reușită Ethereum pe rețeaua de test Kovan

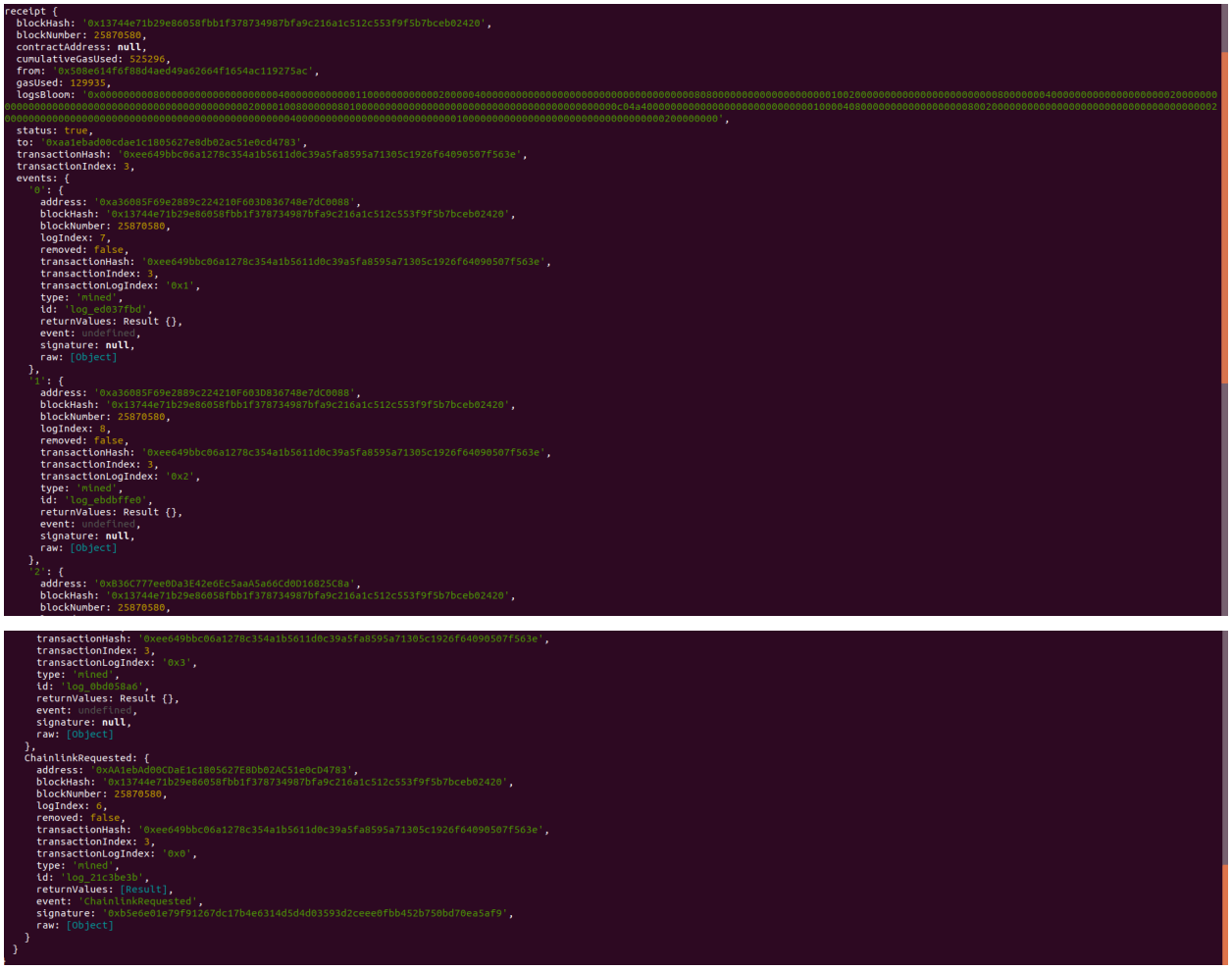

Fig. V.3 Tranzactia creata la momentul conectarii clientului la capabilitatile fizice descrise de fabricant

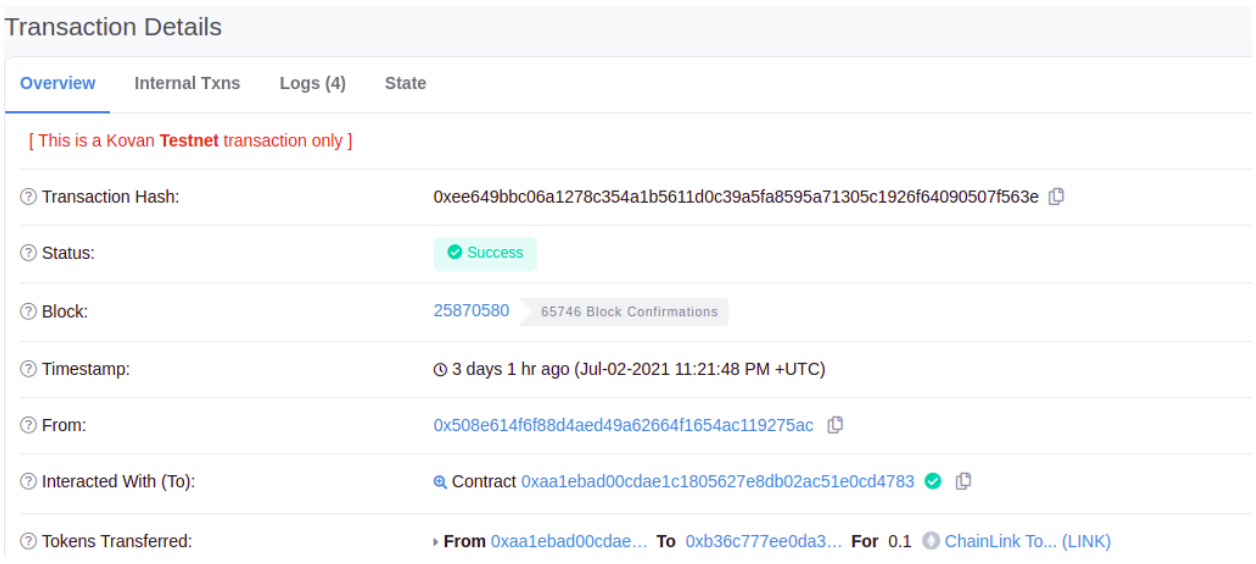

Fig. V.4 Interacțiune reușită, costul cerut de fabricant a fost transferat în momentul conectării

5.2. Testarea componentei fizice

Rezultatele preliminare constau în dezvoltarea și publicarea cu succes a contractului inteligent pe rețeaua de test Kovan, precum și a nodului Chainlink și al Oracolului. Performanțele soluției abordate variază în funcție de algoritmul de consens utilizat.

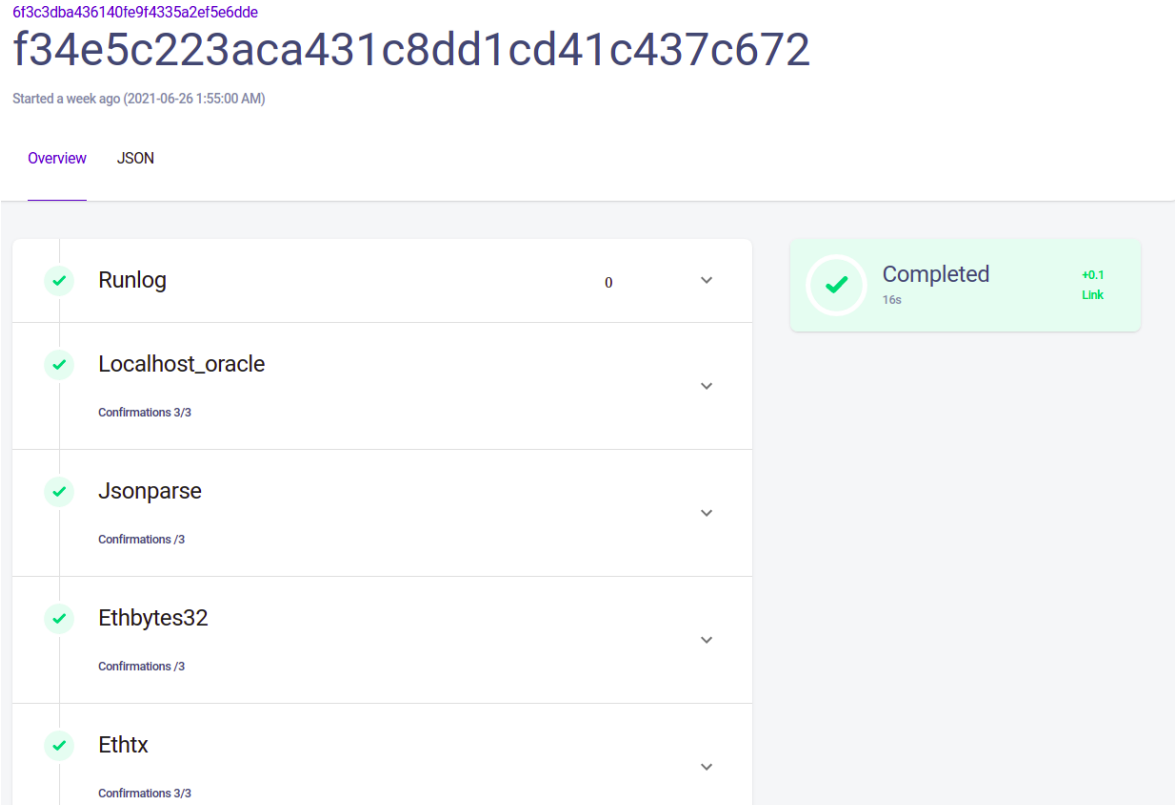

Fig. V.5 Tranzactie reușita în mediul Chainlink

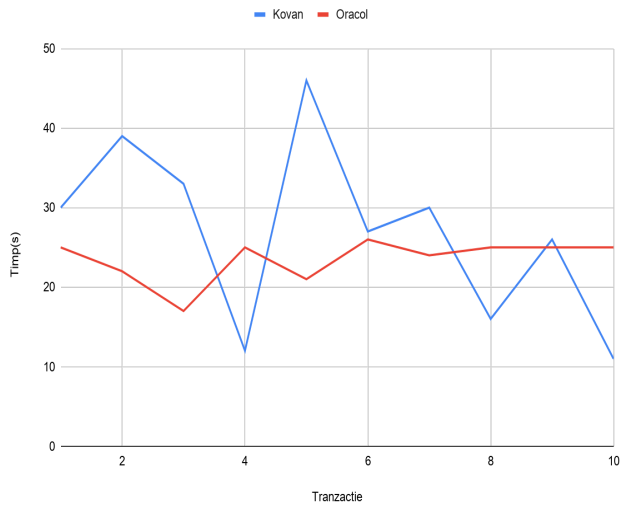

Fig. V.6. Timpul de procesare a unei tranzacții și timpul de procesare a cererii către oracol

În cazul rețelei Kovan testnet bazată pe PoA, după efectuarea a 10 cereri am obținut rezultatele prezentate în Fig.6 și tabelul 1 rezultate . (Fig. V.6, Tabelul 3)

Răspunsul oracolului este independent de procesarea cererii în blockchain, acesta fiind strict influențat de algoritmul de consens folosit și de mărimea blockchain-ului. Timpul de răspuns al oracolului depinde în mod direct de complexitatea adaptorului extern utilizat și timpul de răspuns al mașinii interogate.

Tabelul 3. Analiza timpului de procesare a tranzacției și a cererii

	std	***mean***	***median***
Kovan testnet	11.3 s	27s	28.5s
Oracol	2.7s	23.5s	25s

Aceste rezultate ne evidențiază faptul că blockchain nu este potrivit în mod special pentru transferul de date în timp real și luarea deciziilor. Însă am demonstrat că sistemul blockchain asigură corectitudinea provenienței datelor și transparența lor la costuri reduse, sacrificând performanța rețelei în mod deliberat.

5.3. Testarea aplicației client

Testarea aplicației client presupune în primul rând validarea codului, verificarea logicii fiecărei componente și a interacțiunilor dintre acestea. Din analiza performanțelor aplicației a reieșit că:

- Aplicația client oferă răspunsuri rapide și utilizează resurse limitate.
- Din totalul de 3 secunde înregistrate în mediul de test, răspunsul total al paginii a fost mai mic de 0.5s.

Aceste rezultate au fost înregistrate în profilul fabricantului în momentul creării profilului și include interacțiunea cu metoda de creare a tranzactiei ce adaugă în bloc detaliile fabricantului.

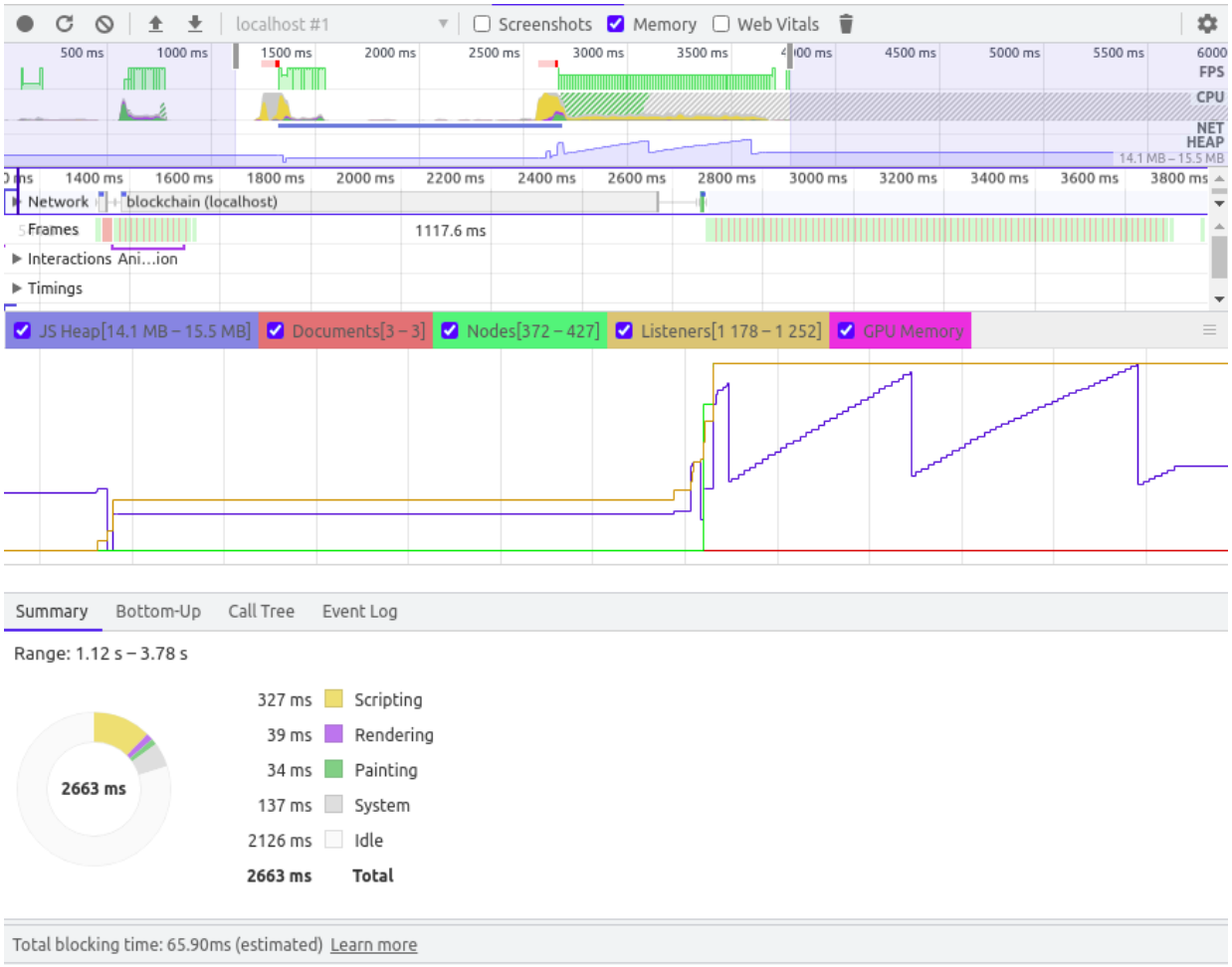

Fig. V.7 Analiza performanței memoriei

Concluzii

În aceasta lucrare am prezentat modul de utilizare a tehnologiei blockchain în viziunea celei de-a IV-a revoluții industriale prin crearea unei aplicații descentralizate menită să elimine costul asociat asigurării încrederii între părțile implicate și să genereze transparenta între furnizorul de servicii și client.

Pentru a simula un sistem descentralizat:

- am creat o rețea blockchain restransa conținând un computer și un arduino care acționează ca participanți la blockchain reprezentând diferiți furnizori de servicii de fabricație;
- am utilizat rețeaua de test kovan reprezentând totodată și componenta blockchain aceasta funcționează ca mediu de interacțiune între componenta fizică și componenta client cât și ca mediu de stocare transparent și imuabil.
- am fluidizat procesul prin crearea unei aplicații client cu un design-ul modern și minimalist care controlează întregul proces: de la modul de efectuare a tranzacțiilor la rezultate și monitorizarea lor. Interfața aplicație client este o aplicație web, aceasta poate fi ușor de folosit pe orice device modern.

Tehnologia Blockchain și-a arătat potențialul de transformare a industriei tradiționale cu caracteristicile sale cheie: descentralizare, persistență, anonimat și imuabilitate Utilizarea tehnologiei Blockchain satisface multe din cerințele pieței, oferind transparență ridicată asupra autenticității tranzacțiilor, costurilor și monitorizarea asigurând creșterea încrederii între clienți și fabricanți.

Pentru dezvoltarea ulterioară a proiectului, intenționez să dezvolt un mijloc de integrare facil al mijloacelor de fabricație deoarece acest sector este în continuă dezvoltare.

Bibliografie

[1] SE, Burkhard Blechschmidt, Carsten Stocker, Cognizant Technology Solutions, Innogy. *How Blockchain Can Slash the Manufacturing Trust Tax*. Accessed 6 July 2021.

[2] Bodkhe, Umesh, et al. "Blockchain for Industry 4.0: A Comprehensive Review." *IEEE Access*, vol. 8, 2020, pp. 79764–800, doi:10.1109/access.2020.2988579.

[3] Zheng, Zibin, et al. "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends." *2017 IEEE International Congress on Big Data (BigData Congress)*, IEEE, 2017, http://dx.doi.org/10.1109/bigdatacongress.2017.85.

[4] SHA-256. *The Cryptographic Hash Function SHA-256*. The cryptographic hash function SHA-256.

[5] "SHA-2." *Wikipedia*, 30 June 2021, https://en.wikipedia.org/wiki/SHA-2.

[6] "Ethereum." *Ethereum*, https://ethereum.org/en/. Accessed 6 July 2021.

[7] *Kovan Testnet*. https://kovan-testnet.github.io/website/. Accessed 6 July 2021.

[8] *Blockchain Oracles for Hybrid Smart Contracts*. https://chain.link/. Accessed 6 July 2021.

[9] *Solidity — Solidity 0.8.4 Documentation*. https://docs.soliditylang.org/en/v0.8.4/. Accessed 6 July 2021.

[10] "HTML." *Wikipedia*. N.p., 29 June 2021. Web. 7 July 2021.

[11] "CSS." *Wikipedia*. N.p., 6 July 2021. Web. 7 July 2021.

[12] "JavaScript." *Wikipedia*. N.p., n.d. Web. 7 July 2021.

[13] Otto, Mark and Bootstrap contributors. *Introduction*. https://getbootstrap.com/docs/5.0/getting-started/introduction/. Accessed 6 July 2021.

[14] Node.js. "Node.Js." *Node.Js*, https://nodejs.org/en/. Accessed 6 July 2021.

[15] *Express - Node.Js Web Application Framework*. https://expressjs.com/. Accessed 6 July 2021.

[16] *Web3.Js - Ethereum JavaScript API — Web3.Js 1.0.0 Documentation*. https://web3js.readthedocs.io/en/v1.3.4/. Accessed 6 July 2021.

[17] "Sweet Tools for Smart Contracts." *Truffle Suite*, https://www.trufflesuite.com/. Accessed 6 July 2021.

[18] *The MongoDB 4.4 Manual — MongoDB Manual*. https://docs.mongodb.com/manual/. Accessed 6 July 2021.

[19] "Empowering App Development for Developers." *Docker*, https://www.docker.com/. Accessed 6 July 2021.

[20] "Ethereum API." *Infura*, https://infura.io/dashboard. Accessed 6 July 2021.

[21] *MetaMask*. https://metamask.io. Accessed 6 July 2021.

[22] Papadodimas, Georgios, et al. "Implementation of Smart Contracts for Blockchain Based IoT Applications." *2018 9th International Conference on the Network of the Future (NOF)*, IEEE, 2018, http://dx.doi.org/10.1109/nof.2018.8597718.

[23] Angrish, Atin, et al. “A Case Study for Blockchain in Manufacturing: ‘FabRec’: A Prototype for Peer-to-Peer Network of Manufacturing Nodes.” *Procedia Manufacturing*, vol. 26, 2018, pp. 1180–92, doi:10.1016/j.promfg.2018.07.154.

[24] Fraga-Lamas, Paula, and Tiago M. Fernandez-Carames. “A Review on Blockchain Technologies for an Advanced and Cyber-Resilient Automotive Industry.” *IEEE Access*, vol. 7, 2019, pp. 17578–98, doi:10.1109/access.2019.2895302.

Anexe

```
const express = require('express');
const app = express();
var SerialPort = require("serialport");
const sleep = ms => new Promise(res => setTimeout(res, ms));
var port = 3000;
var arduinoCOMPort = "/dev/ttyUSB0";
var arduinoSerialPort = new SerialPort(arduinoCOMPort, {
 baudRate: 9600
});

arduinoSerialPort.on('open',function() {
  console.log('Serial Port ' + arduinoCOMPort + ' is opened.');
});

function readSerial () // for reading
{
    arduinoSerialPort.on('data', async function(data)
    {
        console.log(String(data));

    });
        console.log('Recived');
}
app.get('/:action', function (req, res) {

   var action = req.params.action || req.param('action');

    if(action == 'on'){
        arduinoSerialPort.write("o");
        readSerial();
        sleep(2000);
        return res.send(JSON.parse(data));
    }
    if(action == 'off') {
        arduinoSerialPort.write("f");
        readSerial();
        sleep(2000);
        return res.send(JSON.parse(data));
    }
```

```
    return res.send('Action: ' + action);

});

app.listen(port, function () {
  console.log('Oracle app listening on port http://0.0.0.0:' + port + '!');
});
```

Exemplu implementare API de manipulare a elementului hardware

```
app.post("/api/blockchain", async (req, res) => {
    const id_Fabric = req.body.id;
    const email = req.body.email;
    const fabricDetails = req.body.details;

    email_hashed = Buffer.from(email).toString("base64");

    if (email && id_Fabric) {
      const dataToBeHashed = email + "_" + id_Fabric;

      const address =
        "0x" +
        crypto
          .createHash("sha256")
          .update(dataToBeHashed + "test")
          .digest("hex");

      const gasPrice = await web3.eth.getGasPrice();
      console.log('address,id:',address,id_Fabric);

      const gasEstimate = await contract.methods
        .createFabric(address, email_hashed, fabricDetails)
        .estimateGas({ from: web3.eth.defaultAccount });

      contract.methods
        .createFabric(address, email_hashed, fabricDetails)
        .send({
          from: web3.eth.defaultAccount,
          gasPrice: gasPrice,
          gas: gasEstimate,
        })
```

```
        .on("transactionHash", async function (txn_hash) {
          console.log('THX\n', txn_hash);
                    res.status(200).send({ txn_hash });
        })
        .on("receipt", function (receipt) {
          console.log("receipt", receipt,typedef(receipt));
        })
        .on("confirmation", function (confirmationNumber, receipt) {
          console.log("confirmationNumber", confirmationNumber);
        })
        .on("error", function (error, receipt) {
          throw error;
        });
    } else {
      res.sendStatus(401);
    }
           User.findOne({_id: id_Fabric}), function (err, user) {
            if (err) return console.error(err);
            console.log(user);}
  });

app.get("/api/blockchain", async (req, res) => {
    const email = req.body.email;
    const id_Fabric = req.body.id;

    if (email && id_Fabric) {
      const dataToBeHashed = email + "_" + id_Fabric;

      const address =
        "0x" +
        crypto
          .createHash("sha256")
          .update(dataToBeHashed + "test")
          .digest("hex");

      contract.methods
        .getFabric(address)
        .call({ from: web3.eth.defaultAccount })
        .then(function (result) {
          formatedResult = formatBlockchainData(result);
          console.log(formatedResult);
```

```
          res.status(200).send(data);
        });
    } else {
      res.sendStatus(401);
    }
  });
```

Exemplu implementare serviciul de interacțiune cu blockchain

```
{
      "inputs": [
        {
          "internalType": "address",
          "name": "_oracle",
          "type": "address"
        },
        {
          "internalType": "string",
          "name": "_jobId",
          "type": "string"
        }
      ],
      "name": "requestFabricData",
      "outputs": [
        {
          "internalType": "bytes32",
          "name": "requestId",
          "type": "bytes32"
        }
      ],
      "stateMutability": "nonpayable",
      "type": "function"
    }
```

Exemplu implementare funcție requestFabricData în abi