

LUCRARE DE DISERTAȚIE



WEB 3.0: Dezvoltarea aplicațiilor decentralizate pe blockchain-ul Ethereum.

Noțiuni teoretice, dezvoltare și implementare

Autor: © Sergheenco Daniel

Coordonator: ș.l. dr. ing. Mătășaru Daniel





PREFAȚĂ

Lucrarea dată este realizată cu scopul de a obține calificarea de master în domeniul Rețelelor de Comunicații. Trend-urile actuale indică faptul că informația se dorește a fi tot mai decentralizată și pentru acest lucru se apelează adesea la tehnologia blockchain, ceea ce reprezintă o bază de date distribuită în rețeaua internet. În acest sens, mi-am propus să fac studiul de caz pentru a dezvolta o aplicație decentralizată, utilă în domeniul medical.

AUTOR: Daniel Sergheenco

<https://www.linkedin.com/in/daniel-s-376a288a/>

www.sergheenco.com

Coordonator: ș. I. dr. Ing. Mătășaru Daniel





Cuprins

1. Introducere	5
1.1. Motivație	5
1.2. Obiective	6
1.3. Cerințe funcționale pentru aplicația pe blockchain	6
2. Bazele Tehnologiei Blockchain. Concepte Teoretice	7
2.1. Conceptul „Sisteme distribuite”	7
2.1.1. Problema generalilor Bizantini (Capitol trebuie reformulat sau scos).....	7
2.2. Consensul în sistemele distribuite	8
2.3. Funcțiile HASH și Arborele Merkle	9
2.4. Tehnologia blockchain	11
2.4.1. Structura unui bloc	12
2.4.2. Minarea	12
2.5. Blockchain-ul Ethereum.....	13
2.5.1. WEB3.0: Decentralizarea	13
2.5.2. Arhitectura Ethereum.....	15
2.5.3. Portofelul Ethereum și interfața clientului.....	18
2.5.3.1. Metamask	19
2.6. Contracte inteligente (Smart Contracts).....	19
2.6.1. Concepte de bază	19
2.6.2. Limbajul Solidity pentru programarea contractelor inteligente Ethereum	20
2.6.3. NFT – Token nefungibil	21
2.7. Conceptul Dapps (Aplicații decentralizate)	22
2.7.1. Dezvoltarea aplicațiilor decentralizate	22
2.7.2. Utilitare: Node.js, React, Web3.js	23
2.7.3. Framework Truffle.....	24
2.7.4. Simulatorul Ganache.....	25
3. Studiu de caz: Aplicație decentralizată pentru prescripții medicale electronice în rețeaua Ethereum	25
3.1. Introducere	25
3.2. Mediul de dezvoltare	26
3.2.1. Instalare Node.js	26



3.2.2.	Instalare Truffle	27
3.2.3.	Instalare și configurarea Ganache	27
3.2.4.	Instalare și configurarea Metamask	28
3.3.	Smart Contractul „PrescriptionNFT.sol”	29
3.3.1.	Rulare smart contractului pe blockchain.....	33
3.4.	Aplicația „Interfața Medicului”	35
3.4.1.	Arhitectură Software	36
3.5.	Aplicația „Interfața Farmacistului”	38
3.5.1.	Arhitectură Software	39
3.6.	Aplicația „Interfața Pacientului”	41
3.6.1.	Arhitectură Software	42
4.	Concluzii	44
5.	Bibliografie	45
6.	Rezumat	46



1. Introducere

1.1. Motivație

De la apariția tehnologiei blockchain, companiile private s-au concentrat pe integrarea acesteia în activitățile pe care le desfășoară pentru a-și crește eficiența. Această tehnologie nu este dedicată doar sectorul privat. Din perspectiva sectorului public și al îmbunătățirii serviciilor furnizate cetățenilor, implementarea unor tehnologii de tip blockchain ar putea aduce beneficii în ramuri precum managementul identității (documente de identitate, acte de căsătorie, pașapoarte, permis de conducere etc.), securizarea datelor în aviație, optimizarea sistemului de colectare a taxelor, gestionarea registrelor medicale, gestionarea în mod integrat a parcursului școlar al cetățenilor, asigurarea securității cibernetice, securizarea lanțului de aprovizionare, managementul proprietății și cadastrarea terenurilor agricole.

Mai multe studii și sondaje ale firmelor specializate în consultanță au evidențiat utilitatea blockchain-ului la nivel global. Astăzi, conform unui studiu al Forumului Economic Mondial, tehnologia blockchain este folosită de cel puțin 40 de bănci centrale și peste 200 de programe publice din 45 de țări din întreaga lume.

Tehnologia blockchain prezintă numeroase OPORTUNITĂȚI pentru domeniul sănătății, dar care încă nu dispune de o soluție globală fiabilă în prezent. Un transfer al informațiilor din domeniul sănătății într-un sistem bazat pe blockchain poate facilita cooperarea dintre organizații, întrucât acesta are potențialul de a reduce/elimina interacțiunea și costurile intermediarilor. De asemenea, tehnologia blockchain poate transforma serviciile medicale, plasând pacientul în centrul ecosistemului sănătății și contribuind la creșterea nivelului de siguranță, confidențialitate și interoperabilitate a datelor din domeniul medical.

Partajarea și schimbul dosarelor pacienților între medici și organizații medicale este un proces complicat, pe fondul aspectelor administrative și de confidențialitate. În plus, este dificil de urmărit utilizarea și distribuirea datelor din dosar. Tehnologia blockchain poate reprezenta o soluție viabilă la această problemă, întrucât permite înregistrarea fiecărui pas al procesului și poate furniza dovezi în cazul unui comportament defectuos. Pe lângă managementul datelor pacienților, blockchain prezintă un potențial considerabil pentru utilizarea în sectorul medical, deschizând calea către numeroase aplicații care ar putea face industria medicală mai accesibilă, mai sigură și mai eficientă:

- managementul datelor demografice privind sănătatea populației;
- asigurarea securității datelor utilizate pentru studiile clinice;
- urmărirea traseului comercial al medicamentelor.



1.2. Obiective

- Studiul Tehnologiei Blockchain din punct de vedere tehnic;
- Studiul limbajelor de programare utilizate pentru Aplicațiile pe blockchain;
- Utilizarea instrumentelor și framework-urilor utilizate în blockchain-ul Ethereum;
- Dezvoltarea unei aplicații decentralizate (Dapp);
- Identificarea avantajelor și dezavantajelor aplicațiilor decentralizate.

1.3. Cerințe funcționale pentru aplicația pe blockchain

Prescripții medicale pe Blockchain reprezintă un ansamblu de aplicații rulate pe blockchain (SmartContract) cât și pe server WEB (orientate către client).

Aplicațiile au scopul de a elimina complet riscurile și costurile implicate în fraudarea prescripțiilor medicale prin înlocuirea celor clasice, cu cele electronice. Astfel se facilitează următoarele aspecte:

- Prevenirea falsificării prescripțiilor medicale
- Prevenirea fraudării cantităților de medicamente prescrise
- Scris lizibil indiferent de medic și prescripție
- Eliminarea efemerității prescripțiilor
- Asigurarea unui istoric al tratamentelor

Rolurile definite pentru acest scenariu de proiect sunt următoarele:

Medicul - prescrie medicamente prin executarea unui contract inteligent care simbolizează o rețetă validă, pe baza metadatelor, cum ar fi ID-ul medicului, numele pacientului, cantitatea, doza și data de expirare.

Pacient - Primește un jeton (NFT*) reprezentând o rețetă valabilă eliberată de un medic. Utilizează o rețetă la o farmacie autorizată prin trimiterea jetonului în portofelul public al farmaciei.

Farmacie - Eliberează medicamente după ce a primit jetonul de la un Pacient și este independent de plată în ceea ce privește primirea jetonului sau a monedei fiduciare ca plată pentru prescripție medicală. Verifică validitatea prescripției verificând blockchain-ul pentru o semnătură între pacient și medic.

**NFT – Non-Fungible Token. Un jeton nefungibil este o unitate de date unică dintr-un registru contabil informatic denumit blockchain.*

2. Bazele Tehnologiei Blockchain. Concepte Teoretice

2.1. Conceptul „Sisteme distribuite”

Sistemele centralizate tradiționale constau din două componente principale: clientul și serverul. În cea mai simplă configurare, clientul este cel care face o solicitare pentru a executa un program, iar un server este cel care o face. Așa a funcționat web 1.0, cel pe care am început să-l numim World Wide Web. De exemplu, plasarea unei solicitări de căutare pe motorul de căutare Google unde acesta returnează un set de link-uri web și rezultate rezumate. În acest sistem, dacă doi clienți doresc să comunice între ei, trebuie să plaseze o cerere prin intermediul serverului, care servește ca intermediar. Un al doilea exemplu ar putea fi, dacă se trimite un mesaj din aplicația client de pe mobil, acest mesaj este trimis către serverul WhatsApp, care apoi notifică aplicația destinatarului. Odată ce mesajul este vizualizat, aplicația client trimite înapoi un semnal de confirmare către serverul WhatsApp. Așa funcționează actualul internet și îl numim web 2.0, care în special este diferențiat prin apariția rețelelor de socializare.

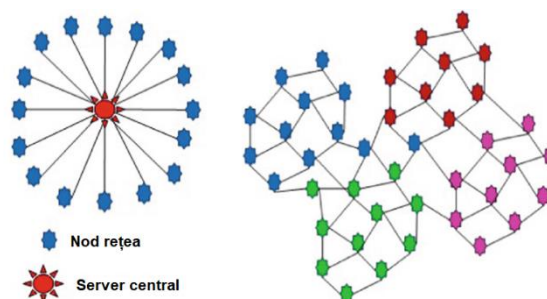


Fig.1.1 Structura unui sistem centralizat și a unui sistem distribuit

În ambele exemple, putem vedea că sistemul centralizat funcționează foarte bine. Problema este că aceste servere centralizate sunt în general deținute de organizații comerciale și pot fi influențate de o entitate criminală sau de o autoritate centrală pentru a scurge date private în timp ce clienții comunică. Pentru a depăși acest defect fundamental, a intrat în practică rețelele peer-to-peer (web 3.0) (de exemplu, BitTorrent). Acestea au fost sisteme distribuite, în care fiecare nod poate fi un client sau un server sau ambele și nu se pot distinge de alte noduri (Figura 1.1). Chiar dacă aceste sisteme erau bune la confidențialitate, s-au confruntat cu provocări precum Problema generalilor bizantini.

2.1.1. Problema generalilor Bizantini

Presupunem că o navă pirat este atacată. La bordul navei pirat se află 200 de piraiți, înconjuși de șase nave armate a câte 50 de războinici fiecare, care au ancorat, înconjurând nava piratilor. Fiecare navă de armată este comandată de un căpitan. Cei 300 de războinici îi pot învinge cu ușurință pe cei 200 de piraiți de la bordul navei de pirat. Cu toate acestea, dacă nu atacă toți simultan, există un risc foarte real ca războinicii să fie depășiți numeric de piraiți și

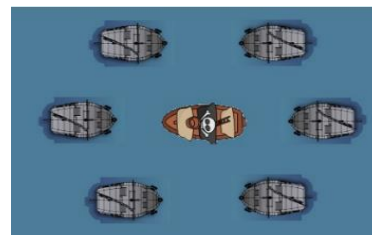


Fig.1.2 Navă pirat înconjurată de 6 nave armate



să piardă lupta. Deci, cum pot căpitani să cadă de acord în același timp să atace nava piratilor? În zilele noastre, am avea nevoie pur și simplu de un apel vocal sau video-conferință rapid de grup, iar căpitani ar trebui să fie de acord să atace la ora 22:00.

Cu toate acestea, în vremurile întunecate, lucrurile erau puțin mai complicate:

- mesajul de atac de la 22:00 putea fi transmis doar de un marinar pe o barcă mică. El trebuie să navigheze în jurul fiecărei nave ale armatei, vizitând pe rând fiecare căpitan pentru a confirma.
- Orice căpitan poate fi un trădător și poate fi în ligă cu pirații din nava pirat.

Strategia pierzătoare:

Căpitanul 1 decide să atace la 22:00. El își trimite marinarul cu mesajul (atac 22:00) pentru a-l livra căpitanului 2. La sosire, căpitanul 2 citește mesajul, notează ora atacului și trimite mesajul mai departe. Îl trimite pe marinar să împărtășească mesajul căpitanului 3. Cu toate acestea, avem o problemă. Căpitanul 3 este un trădător. El vrea ca atacul să eșueze. Deci, când primește mesajul, îl rupe și îl înlocuiește cu un nou mesaj care spune atac la 21:00. Marinarul continuă fără să știe. Căpitanul 4 primește acum un mesaj care spune că atacă la 21:00. El notează ora, semnează mesajul spunând „21:00 atac” și trimite acest lucru către Căpitanul 5, care apoi trimite același mesaj către Căpitanul 6. Acum, mesajul este răspândit la toată lumea, dar avem o problemă. Căpitanul necinstit a perturbat rezultatul. Acum avem trei căpitani (4, 5 și 6) cu 150 de războinici care atacă nava piratilor la ora 21:00. Așteptându-se ca alții să li se alăture, ei sunt în schimb depășiți numeric de cei 200 de pirați. Pirații victorioși ies acum din nava pirat și își unesc forțele cu trădătorul - Căpitanul 3. Deodată, cei doi căpitani rămași (1 și 2) au doar 100 de războinici și se trezesc că luptă cu 200 de pirați plus 50 de trădători. Din păcate, pirații și trădătorii câștigă.

Soluția:

Teoretic, problema și-a găsit soluția în zilele noastre prin crearea unei liste înlățuite cu mesajul fiecărui căpitan, astfel încât, dacă cineva ar vrea să modifice mesajul, ar trebui să rescrie tot istoricul mesajelor într-un timp foarte scurt.

2.2. Consensul în sistemele distribuite

O problemă fundamentală în sistemele de calcul distribuite și multi-agent este obținerea fiabilității generale a sistemului în prezența unui număr de procese defecte. Acest lucru necesită adesea coordonarea proceselor pentru a ajunge la un consens sau a conveni asupra unor valori ale datelor necesare în timpul calculului. Aplicațiile din lumea reală care necesită adesea consens includ: cloud computing, sincronizarea ceasului, PageRank, rețelele de energie inteligente, estimarea stării, controlul Aeronavelor (și a mai multor roboți/agenți în general), echilibrarea sarcinilor de calcul în sistemele distribuite, inclusiv și într-un blockchain.

Problema consensului necesită acordul între un număr de procese (sau agenți) pentru o singură valoare a datelor. Unele dintre procese (agenți) pot eșua sau nu pot fi de încredere în alte moduri, așa că protocoalele de consens trebuie să fie tolerante la erori sau rezistente. Procesele trebuie să-și expună cumva valorile candidate, să comunice între ele și să convină asupra unei singure valori de consens.

Problema consensului este o problemă fundamentală în controlul sistemelor multi-agent. O abordare pentru generarea consensului este ca toate procesele (agenții) să cadă de acord asupra unei valori majoritare. În acest context, o majoritate necesită cel puțin unul mai mult de jumătate din voturile disponibile (unde fiecare proces primește un vot). Cu toate acestea, unul sau mai multe procese defecte pot denatura rezultatul final, astfel încât să nu se ajungă la un consens sau să se ajungă încorect.

Protocoalele care rezolvă problemele de consens sunt concepute pentru a face față unui număr limitat de procese defecte. Aceste protocoale trebuie să îndeplinească o serie de cerințe pentru a fi utile. De exemplu, un protocol banal ar putea avea toate procesele să scoată valoarea binară 1. Acest lucru nu este util și, prin urmare, cerința este modificată astfel încât ieșirea trebuie să depindă cumva de intrare. Adică, valoarea de ieșire a unui protocol de consens trebuie să fie valoarea de intrare a unui proces. O altă cerință este ca un proces să poată decide asupra unei valori de ieșire o singură dată și această decizie este irevocabilă. Un proces este numit corect într-o execuție dacă nu se confruntă cu un eșec.

2.3. Funcțiile HASH și Arborele Merkle

Criptografia este arta și știința de a ascunde un mesaj. Este mai mult o artă decât o știință. Știința aici acționează doar ca un simplu instrument pentru a transforma o imaginație artistică într-un algoritm matematic. În această capitole, ne vom concentra doar pe două concepte specifice din acest subiect imens de vast, care ne vor ajuta să înțelegem adevărata esență a unui blockchain. Acestea sunt funcția hash și arborele Merkle. Acestea sunt folosite pentru a verifica integritatea, și nu neapărat pentru a ascunde ceva. După cum putem vedea în figura 1.3, o funcție hash este o mapare unidirecțională.

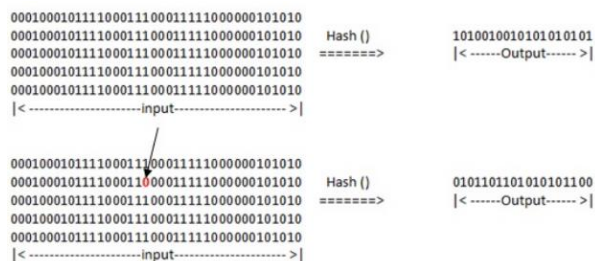


Fig.1.3 Funcție HASH cu intrare și ieșire

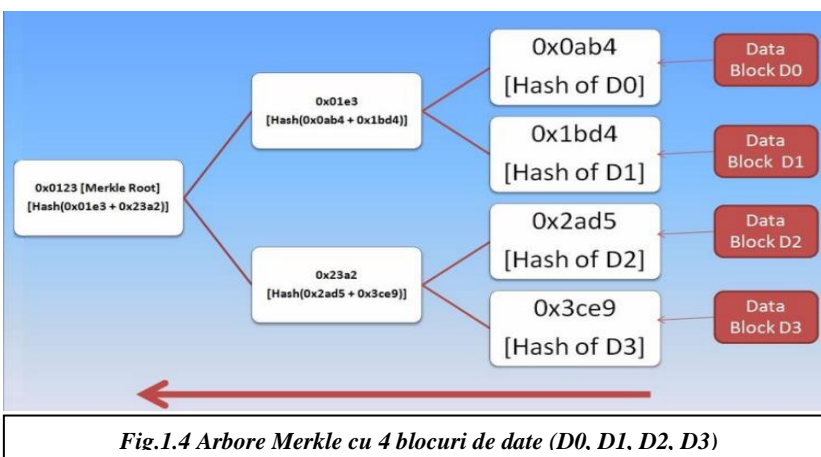
Aceasta înseamnă că poate lua o intrare (intrarea este de obicei o secvență mare de biți; poate fi un film, o melodie, o carte electronică, o imagine sau orice date digitale) și să producă o valoare de dimensiune fixă ca ieșire, adesea mult mai mic decât dimensiunea de intrare. Cu toate acestea, dacă schimb doar un bit în această intrare, ieșirea va fi complet diferită. Aceasta este proprietatea numărului unu a funcției hash și face ca funcția hash să fie foarte rezistentă la coliziuni. A doua proprietate este că nu există nicio modalitate de a afla intrarea dacă am doar ieșirea. Deci, dacă am

intrarea pentru o funcție hash, pot obține întotdeauna o ieșire, indiferent cât de mare este intrarea. Cu toate acestea, dacă am doar o ieșire a unei funcții hash, nu există nicio modalitate de a reconstrui intrarea din aceasta, deoarece funcțiile hash sunt unidirecționale. De asemenea, hash-ul de ieșire este o secvență de biți aleatoriu cu lungime fixă, dar, atunci când este afișat pe terminale, este convertit într-un format hexazecimal și arată alfanumeric.

Următorul lucru pe care trebuie să-l înțelegem este ce este un arbore Merkle. Un arbore Merkle este o structură de date în care fiecare strat este o combinație de hash-uri din stratul său copil. În general, un arbore Merkle este reprezentat folosind un arbore hash binar, unde fiecare nod are cel mult doi copii. Ramurile sunt hashurile combinate ale celor doi copii. Acest proces de re-hashing a concatenării nodurilor copil pentru a crea nodul părinte este efectuat până când se ajunge la vârful arborelui, numit hash rădăcină sau rădăcină Merkle.

În Figura 1.4 avem un arbore Merkle cu 4 blocuri de date.

Să presupunem că avem două valori hash (2ad5 și 3ce9). 0x reprezintă o notație hexazecimală, dar voi omite prefixul pentru simplitate. Pentru a obține următoarea valoare din arbore, le combin într-un singur hash și obțin valoarea 23a2.



Deci, combinând 2ad5 și 3ce9 putem obține întotdeauna valoarea hash 23a2, dar dacă avem doar 23a2 nu există nicio modalitate de a afla valorile originale. Acum fac același hashing cu 0ab4 și 1bd4 și le combin pentru a obține valoarea 01e3 și apoi hashing 01e3 și 23a2 pentru a obține valoarea rădăcină 0123. Deci, această valoare rădăcină va fi o reprezentare a acestei structuri de date, dar nu ar exista modalitate prin care să ne întoarcem și să aflăm toate valorile individuale din arbore, să nu mai vorbim de blocurile de date originale D0, D1, D2 și D3. Ar fi foarte greu.

Să presupunem că sunt proprietarul blocului de date D2 din diagrama precedentă. Mai dobândesc, din consensul distribuit, hash-ul rădăcină, care în situația noastră este 0123. Rog sistemul distribuit să-mi demonstreze că înregistrarea mea D2 este în arbore. Ceea ce îmi returnează serverul sunt hashurile 3ce9, 01e3 și 0123. Folosind aceste informații, dovada este oarecum după cum urmează:

1. Hash de D2, pe care îl calculăm pentru a obține 2ad5
2. Hash de [2ad5 + 3ce9] din care calculăm 23a2
3. Hash de [01e3 + 23a2] din care calculăm 0123

Deoarece cunoaștem rădăcina hash din consensul nostru distribuit, dovada validează că $2ad5$ există în arbore. La fel și datele din D2. În plus, sistemul de la care ați obținut dovada, ne arată că este o autoritate, deoarece este capabil să furnizeze hashuri valide, astfel încât să puteți ajunge de la D2 la hash-ul rădăcină cunoscută 0123. Orice sistem care pretinde că vă validează cererea nu vă poate oferi hashe-urile intermediare. Deoarece nu îi dai sistemului hash-ul rădăcină, îi spui doar să-ți dea dovada, consensul distribuit pur și simplu nu poate inventa demonstrația pentru că nu cunoaște hash-ul rădăcină. Mai mult, pentru a verifica dovada, vi se dezvăluie foarte puține informații despre arbore. În plus, pachetul de date necesar pentru această demonstrație este foarte mic, ceea ce face eficientă trimiterea printr-o rețea și efectuarea calculului probei.

2.4. Tehnologia blockchain

Din perspectiva unui dezvoltator de software, un blockchain este un arbore Merkle gigant, în care fiecare bloc nou încorporează hash-ul blocului anterior, precum și hash-ul rădăcină a blocării prezente, formând în cele din urmă un lanț de hash numit blocuri. S-ar putea să nu fie foarte evident, dar o astfel de structură arborescentă Merkle nu necesită un server central; fiecare bloc poate proveni de la clienți separați fizic. Prin urmare, un blockchain este, de asemenea, un sistem distribuit. În Figura 1.5 avem reprezentată structura unui blockchain, compus în general din blocuri ce conțin HASH-uri, indici de timp (timestamp) și valori de validare (nonce).

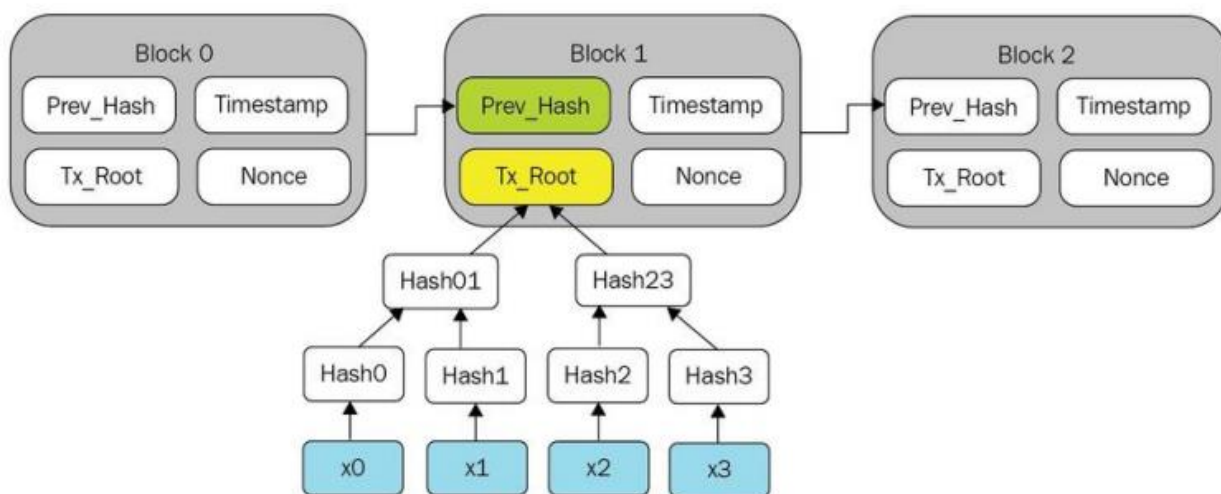


Fig.1.5 Blockchain ca un arbore Merkle gigant

2.4.1. Structura unui bloc

Un bloc este elementul de bază al unui blockchain. Vom lua ca exemplu un bloc din bitcoin elementele cărui sunt reprezentate în Figura 1. În primul rând, avem un număr magic, numit și ID-ul rețelei bitcoin.

Field	Description	Size
Magic no	value always 0xD9B4BEF9	4 bytes
Block size	number of bytes following up to end of block	4 bytes
Block header	consists of 6 items	80 bytes
Transaction counter	positive integer VI = VarInt	1 - 9 bytes
Transactions	the (non empty) list of transactions	<Transaction counter>-many transactions

Fig.1.6 Componentele unui bloc din Bitcoin

Are o lungime de patru octeți și este un număr arbitrar care semnalează că acesta este un bloc bitcoin. Numărul magic nu este ceva specific bitcoin. Toate nodurile comunică folosind protocolul de control al transmisiei. În TCP, diferite tipuri de pachete de date folosesc numere magice diferite pentru a se identifica. În contextul nostru, un bloc este de fapt o secvență de 0 și 1. Când orice mașină citește o astfel de secvență de date și întâlnește versiunea binară a 0xD9B4BEF9, va identifica datele ca un bloc bitcoin. Deci, acest număr este același pentru toate blocurile bitcoin.

În continuare, avem dimensiunea blocului care este, de asemenea, de patru octeți și ne spune cât de lung este acest bloc cu toate tranzacțiile sale. Din iulie 2017, dimensiunea unui întreg bloc bitcoin poate fi de maximum 1 MB. Avem apoi antetul blocului, care are 80 de octeți. Următorul este contorul de tranzacții, care este un număr întreg care ne spune câte tranzacții are acest bloc și în continuare avem o listă de tranzacții, care conține pur și simplu toate tranzacțiile care sunt în acest bloc.

2.4.2. Minarea

Minarea este procesul de adăugare a blocurilor în lanț. Ceea ce face minerul (care deține putere de calcul) este să ia rădăcina hash Merkle a blocului curent și apoi să adauge un nonce (o valoare) pe care o ghidește pornind de la zero. Ei vor distribui aceste date concatenate pentru a obține un nou hash și vor compara acest rezultat cu ținta. Dacă acest nou hash este mai mic decât ținta, care este practic un număr de 256 de biți specificat de protocolul bitcoin, atunci ei au terminat de rezolvat puzzle-ul și primesc recompensa pentru minerit. Cu toate acestea, dacă noua valoare hash



este mai mare decât valoarea țintă, trebuie să crească nonce; adică creșteți nonce de la zero la unu și adăugați-l cu hash-ul rădăcinii Merkle, luați din nou hash-ul pentru a obține o altă valoare hash nouă, complet diferită și comparați-o cu ținta pentru a verifica dacă această nouă valoare hash este mai mică decât valoarea țintă. Dacă noul hash este acum mai mic, atunci, din nou, au rezolvat acest puzzle; în caz contrar, ei continuă să repete întregul proces prin creșterea valorii nonce. Așadar, este important să realizați că, cu cât ținta este mai mică, cu atât este mai dificil să găsiți o valoare hash mai mică decât valoarea țintă.

2.5. Blockchain-ul Ethereum

Ethereum este o platformă și sistem de operare open-source, distribuit, pe bază de blockchain, ce oferă posibilitatea implementării contractelor smart. Ethereum oferă o mașină virtuală descentralizată de tip Turing-complete, numită Ethereum Virtual Machine (EVM), care poate executa script-uri folosind o rețea internațională de noduri publice.

2.5.1. WEB3.0: Decentralizarea

Web3 este un protocol pentru o nouă versiune a internetului descentralizată și publică bazată pe blockchain. În Web3, datele nu sunt deținute de entități centralizate, ci sunt partajate între utilizatori. Web3 cuprinde aplicații descentralizate (Dapps) și finanțe descentralizate (DeFi) bazate pe blockchain cum ar fi criptomonede, active și jetoane (NFT). Oricine este capabil să creeze și să se conecteze cu diferite aplicații Dapp fără permisiunea unei entități centrale. Aplicațiile din Web3 fie rulează pe blockchain, rețele descentralizate ale mai multor noduri (servere) peer-to-peer, fie o combinație a celor două care formează un protocol cripto-economic.

Paul Baran, în textul său clasic despre rețelele de comunicații distribuite (august 1964), a distins trei tipuri de arhitecturi care ar fi aplicabile rețelelor de comunicații, așa cum este descris în Figura 1.7.

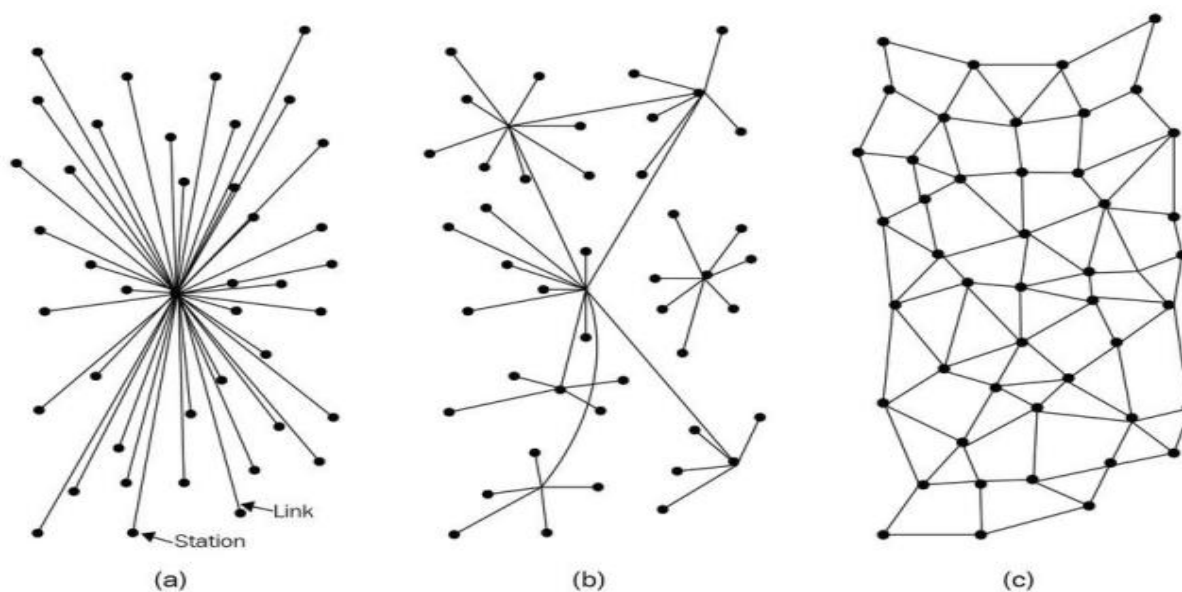


Fig.1.7 Arhitectură centralizată(a), decentralizată(b), distribuită(c)

Descentralizarea constă din cel puțin trei axe importante: axa arhitecturală, axa politică și axa logică. Un sistem precum Ethereum poate fi centralizat, fie descentralizat de-a lungul acestor trei axe ortogonale (Figura 1.8). De asemenea, uneori se poate întâmpla că este greu să tragi o linie și să spui că acest sistem este centralizat sau descentralizat în ansamblu. Să începem cu ceea ce înțelegem prin axa arhitecturală. Dacă un sistem este centralizat în axa arhitecturală și scot câteva noduri/terminale de calcul din sistem, s-ar prăbuși. Dimpotrivă, într-un sistem care este descentralizat pe axa arhitecturală, chiar dacă aș elimina majoritatea nodurilor/terminalelor de calcul, sistemul ar funcționa în continuare conform intenției. Deci, în acest sens, Ethereum este descentralizat pe axa arhitecturală, deoarece putem elimina multe noduri de calcul din sistem, dar întreaga rețea va funcționa în continuare. Cu toate acestea, dacă aveți un site web pe care îl găzduiți pe un server sau o rețea de servere în care fiecare are nevoie de toate celelalte și eliminați un server, serverele și site-ul web nu vor mai funcționa, în mod evident, deoarece acesta era un sistem centralizat pe axa arhitecturală.

A doua axă este axa politică. De exemplu, un sistem centralizat pe axa politică ar putea fi o companie sau o întreprindere în care avem un CEO și poate avem un consiliu de administrație, iar ei decid cursul companiei. Cu toate acestea, Ethereum este descentralizat pe axa politică, pentru că nici măcar Fundația Ethereum nu poate face pe toți să-și folosească clienții sau să oblige pe toți să urmeze un protocol. Acum, ultima axă de discuție este axa logică. Deci, un sistem care este centralizat pe axa logică acționează ca o singură entitate, ca un obiect monolitic, în timp ce un sistem care este descentralizat pe axa logică poate fi împărțit în mai multe părți, iar fiecare dintre acele părți s-ar comporta și funcționa conform intenției. De exemplu, protocolul BitTorrent și sistemul BitTorrent sunt descentralizate pe axa logică, deoarece putem împărți sistemul în mai multe părți, iar computerele din acele părți împărțite vor putea în continuare să folosească protocolul BitTorrent așa cum este prevăzut. Cu toate acestea, aici este locul în care Ethereum nu

este descentralizat. De fapt, Ethereum este centralizat când vine vorba de axa logică, deoarece acesta acționează ca un singur computer. Îți execuți contractele pe Ethereum și pe acest computer uriaș care rulează pe blockchain. Deci, în acest sens, Ethereum este centralizat. Dacă am împărți Ethereum-ul în două părți diferite, atunci acestea nu ar mai funcționa așa cum s-a intenționat, deoarece întreaga idee a lui Ethereum este că ar trebui să fie un singur computer - ar trebui să acționeze ca un singur computer. În acest fel, Ethereum nu este centralizat pe axa logică.

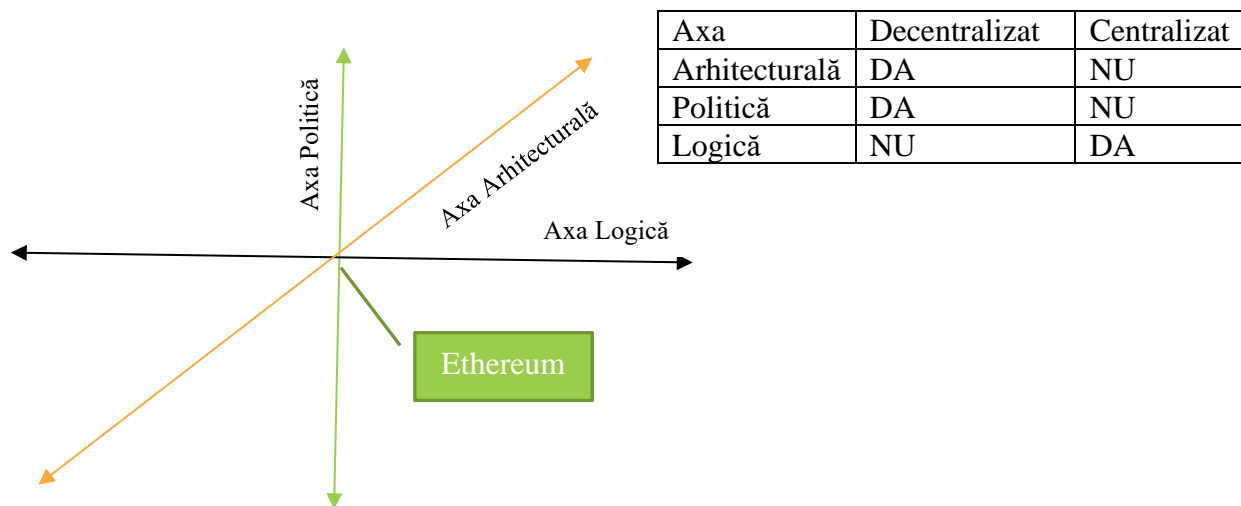


Fig.1.8 Decentralizarea pentru Ethereum

2.5.2. Arhitectura Ethereum

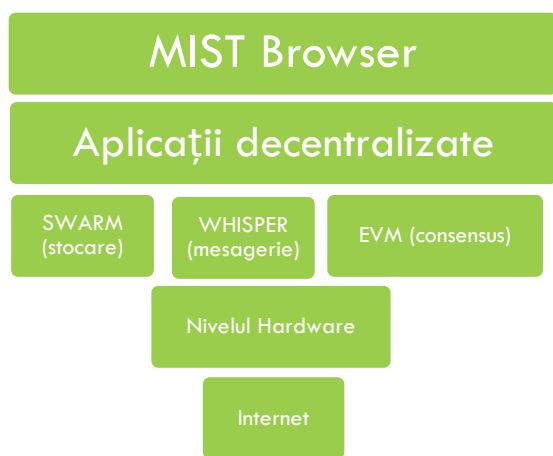


Fig.1.9 Sistemul Ethereum

Sistemul Ethereum este unul complex și are structura prezentată în figura 1.9.

Browserul MIST, este straturat orientat spre utilizator al platformei Ethereum. Mist este instrumentul de alegere pentru a naviga și a utiliza aplicații descentralizate. În figura 1.10 putem vedea care ar fi analogia browserului MIST cu Aplicații din Sistemele bazate pe server (WEB 2.0).

Următorul strat: aplicațiile descentralizate (DApps). Acestea sunt practic aplicații care sunt executate printr-o rețea descentralizată peer-to-peer și au un cod sursă deschis. Poate fi un program simplu bazat pe logică care adaugă două numere sau poate fi ceva destul de complex, cum ar fi un serviciu de microblogging.

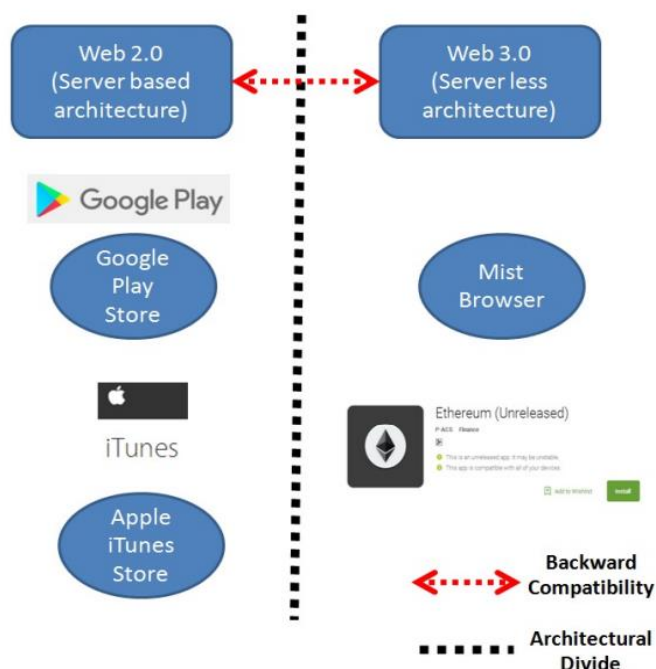


Fig.1.10 Analogie MIST Browser cu aplicații bazate pe server

Mist și DApps fac în principal parte din stiva de front-end al Ethereum. Să trecem acum la mecanismele backend. În primul rând, începem cu middleware-ul. Middleware-ul Ethereum are o ierarhie plată, deoarece nu există o autoritate centrală, cum ar fi un server. Este alcătuit din trei componente principale de importanță egală:

- SWARM: stocarea de date.
- Whisper: Sistemul de comunicare.
- EVM: Mediul de rulare.

O arhitectură fără server în web 3.0, permite o abordare mult mai modulară, în care diferite dispozitive de calcul și diferite protocoale se ocupă de sarcini specifice; unele pe partea clientului și altele în noduri specializate implementate pe o rețea peer-to-peer. Prin urmare, toată logica de date care cuprinde jurnalele despre ceea ce se salvează, cine îl salvează și rezolvă conflictele sunt gestionate de contractele inteligente pe blockchain; Fișierele statice sunt servite prin SWARM, iar comunicația prin WHISPER.

Figura 2.8 ilustrează modul în care aceste module funcționează la unison cu aplicațiile decentralizate și blockchain-ul subiacent într-o arhitectură fără server. O astfel de configurație încurajează și inovația. Pe măsură ce interfețele sunt detașate de date, oricine poate dezvolta o nouă interfață pentru aceeași aplicație, creând un ecosistem mai vibrant și mai competitiv.

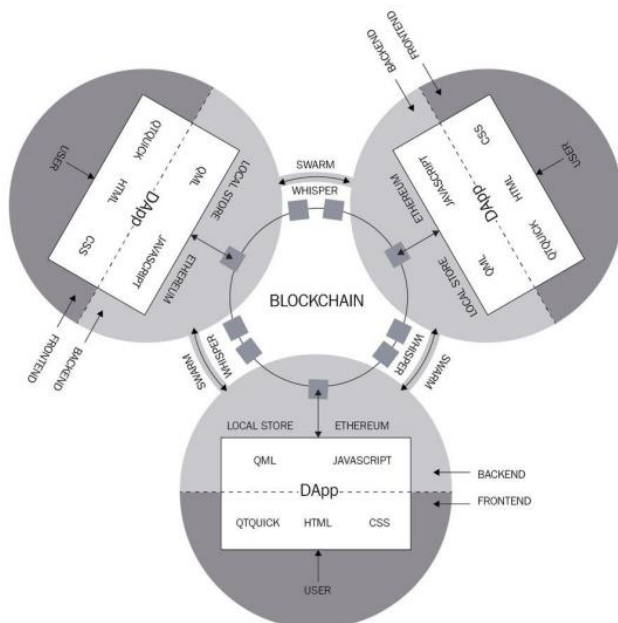


Fig.1.11 Arhitectura decentralizată

EVM este interpretorul bazat pe stivă, care are o matrice de octeți de memorie și stocare cheie-valoare. Figura 1.12 ilustrează o astfel de stivă ca un proces LIFO (ultimul intrat, primul ieșit) cu operațiunile sale cruciale precum pop și push:

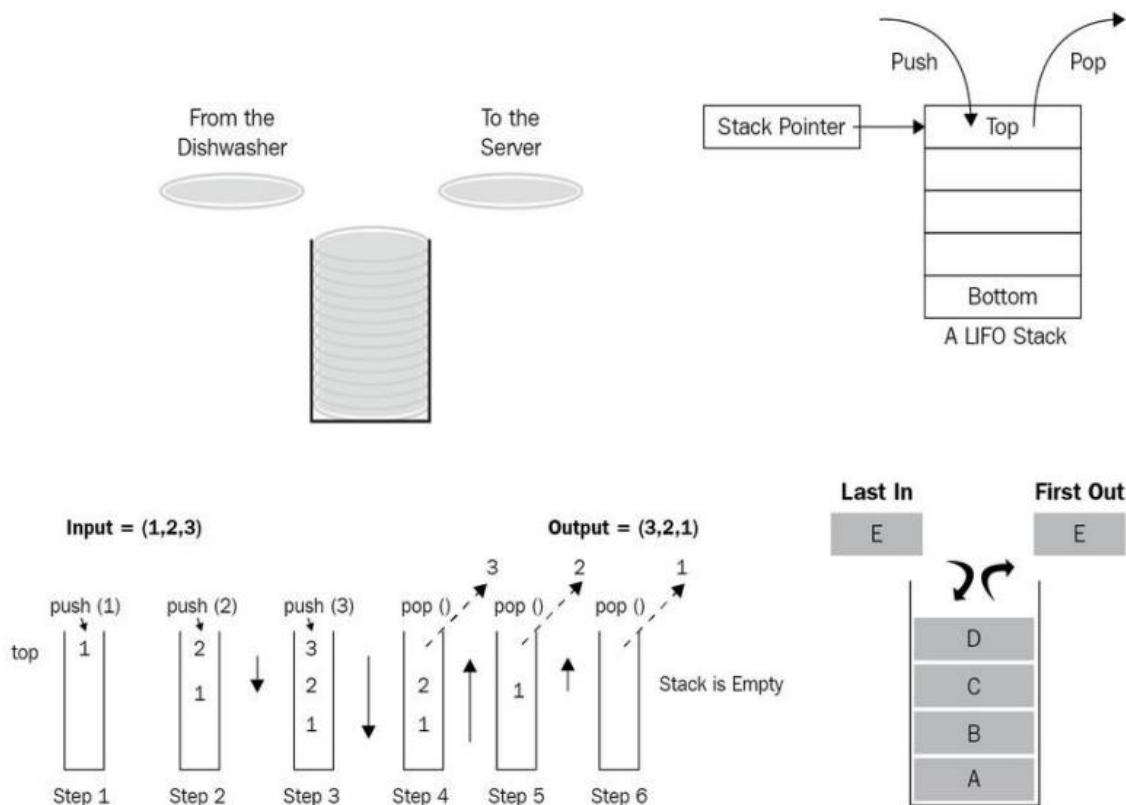


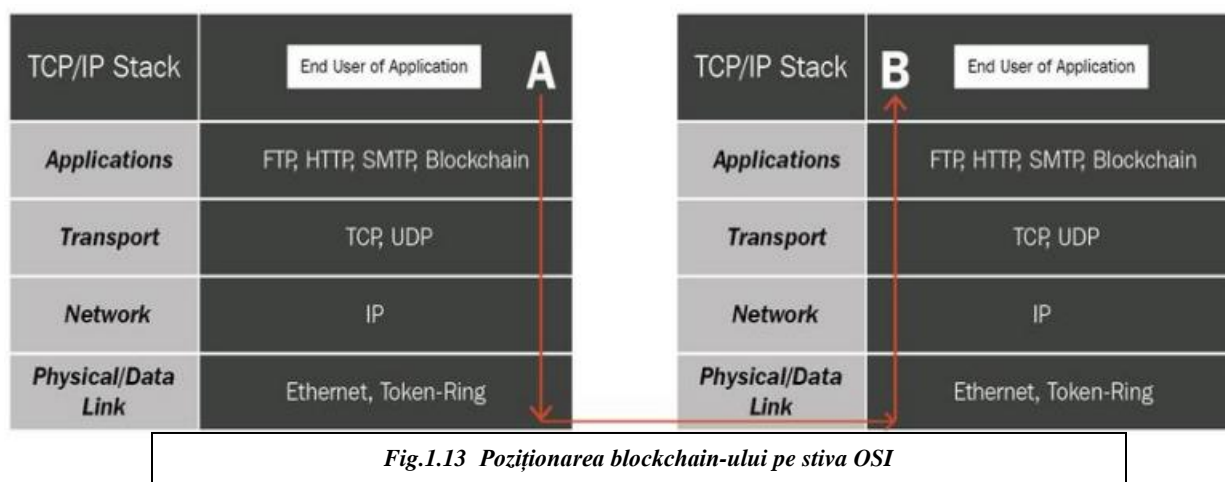
Fig.1.12 Stiva operațională din EVM

Elementele din stiva EVM au o lungime de 32 de octeți și toată stocarea cheie-valoare are, de asemenea, 32 de octeți. Contractele inteligente, care sunt codificate în limbaje de nivel înalt, rulează pe blockchain prin EVM, care generează coduri operaționale la nivel de mașină (opcodes) în timpul rulării. Aceste opcode au acces la trei tipuri de spațiu pentru stocarea datelor:

- Stiva, un container LIFO în care valorile pot fi încărcate și afișate.
- Memorie, o matrice de octeți care poate fi extinsă la infinit.
- Stocarea pe termen lung a contractului este un magazin cheie-valoare. Spre deosebire de stiva și memorie, care se resetează după terminarea calculului, stocarea persistă pe termen lung.

Nivelul Hardware

CPU-urile și GPU-urile, sunt folosiți ca clienți hardware pentru Ethereum. Să discutăm acum stratul de internet al stivei tehnologice Ethereum. Internetul, așa cum știm, este în principal rețeaua de suprafață. Rețeaua de suprafață constă din multe straturi logice, prin care trebuie să treacă un mesaj sau date pentru a ajunge de la punctul A la punctul B, care poate fi separat geografic și fizic. Fiecare strat logic are un protocol standard, pe care un mesaj în tranzit trebuie să-l urmeze, altfel i-ar fi interzis să circule pe web. Internet engineering task force (IETF) este o comunitate deschisă care recomandă, într-o manieră pur voluntară, un teren comun de comunicare pe internetul de suprafață, propunând suita protocol de control al transmisiei/protocol al internet. Figura 1.13 reprezintă TCP/IP și poziția blockchain-ului ca tehnologie pe acesta. Linia direcționată indică modul în care un mesaj se deplasează de la punctul A la B:



Deci, vedem că un blockchain este o tehnologie de nivel de aplicație în stiva TCP/IP, care oferă un protocol de transfer de valori. FTP este folosit pentru a transfera fișiere, HTTP este folosit pentru a transfera text hiperlinkat, SMTP este folosit pentru a transfera e-mailuri, iar blockchain este folosit pentru transferul digital nativ de valori și stări de execuție.

2.5.3. Portofelul Ethereum și interfața clientului

Un client Ethereum se referă la orice nod care poate analiza și verifica blockchain-ul Ethereum și poate executa contracte inteligente pe deasupra. Scopul principal al interfețelor client este de a autoriza acreditările utilizatorului și de a efectua diferite operațiuni. Un portofel oferă un serviciu de trimitere, stocare și primire de fonduri. Principalii parametri ai unui bun serviciu de portofel sunt securitatea și încrederea. Utilizatorii trebuie să simtă că fondurile lor sunt în siguranță și că administratorul portofelului nu va fura fondurile. Astfel de portofele sunt cunoscute ca portofele HOT, deoarece sunt întotdeauna conectate la rețea și sunt adesea vulnerabile la hack-uri și exploatare. Portofelele mai sigure includ portofelele de hârtie și portofelele hardware, care sunt

adesea numite portofele COLD. Acestea stochează practic jetoanele în modul offline. Un portofel de hârtie este stocat ca coduri QR scanate reprezentând adresa de primire, împreună cu o parolă, care trebuie reținută. Cu cuvinte simple, imprimați cheile pe o hârtie pentru a nu fi sparte. Portofelele hardware sunt criptate pe un dispozitiv compatibil USB, care se deschide numai cu o parolă.

2.5.3.1. Metamask

MetaMask este un portofel de criptomonede software folosit pentru a interacționa cu blockchain-ul Ethereum. Permite utilizatorilor să-și acceseze portofelul Ethereum printr-o extensie de browser sau o aplicație mobilă, care poate fi apoi folosită pentru a interacționa cu aplicații descentralizate. În figura 1.13 este reprezentată interfața metamask conectată la un portofel dintr-o rețea locală. MetaMask Wallet funcționează pe două chei, una privată și una publică. Cea publică este folosită pentru identificarea utilizatorului, iar cea privată incryptează informația oferind acces doar utilizatorului care deține cheia. MetaMask folosește tehnologia Infura. În ceea ce privește securitatea, portofelul MetaMask oferă setări HD scurte pentru setări deterministe ierarhice. Folosind această caracteristică specială, MetaMask oferă fraze de bază care sunt în esență utile în resetarea conturilor în cazul pierderii informațiilor sau parolelor.

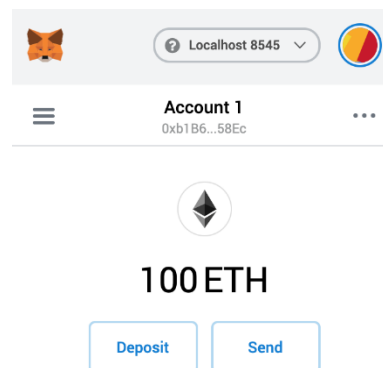


Fig.1.14 Interfața Metamask

2.6. Contracte inteligente (Smart Contracts)

2.6.1. Concepte de bază

Un „contract inteligent” este pur și simplu un program care rulează pe blockchain-ul Ethereum. Este o colecție de cod (funcțiile sale) și date care se află la o anumită adresă din blockchain-ul Ethereum.

Contractele inteligente sunt un tip de cont Ethereum. Aceasta înseamnă că au un sold și pot trimite tranzacții prin rețea. Cu toate acestea, ele nu sunt controlate de un utilizator, ci sunt implementate în rețea și rulează așa cum a fost programat. Conturile de utilizator pot interacționa apoi cu un contract inteligent prin transmiterea tranzacțiilor care execută o funcție definită în contractul inteligent. Contractele inteligente pot defini reguli, cum ar fi un contract obișnuit, și le pot aplica automat prin intermediul codului. Contractele inteligente nu pot fi șterse implicit, iar interacțiunile cu acestea sunt ireversibile.

2.6.2. Limbajul Solidity pentru programarea contractelor inteligente Ethereum

Solidity este un limbaj de nivel înalt orientat pe obiecte pentru implementarea contractelor inteligente. Solidity este un limbaj cu paranteze conceput pentru a viza mașina virtuală Ethereum (EVM). Este influențat de C++, Python și JavaScript.

Solidity este tipizat static, acceptă moștenire, biblioteci și tipuri complexe definite de utilizator, printre alte caracteristici.

Cu Solidity putem crea contracte pentru utilizări precum vot electronic, crowdfunding, licitații și portofele cu semnături multiple.

Mai jos este prezentat un exemplu simplu de un smart contract, care stochează o dată de tip întreg.

```
pragma solidity >=0.4.16 <0.9.0;  
  
contract StocareData {  
    uint Data;  
  
    function set(uint x) public {  
        Data = x;  
    }  
  
    function get() public view returns (uint) {  
        return Data;  
    }  
}
```

Primul rând specifică faptul că codul sursă este scris pentru versiunea Solidity 0.4.16 sau o versiune mai nouă a limbii până la, dar fără a include versiunea 0.9.0. Acest lucru este pentru a ne asigura că contractul nu este compilabil cu o nouă versiune de compilator (încărcare), unde s-ar putea comporta diferit. Pragma-urile sunt instrucțiuni comune pentru compilatori despre cum să trateze codul sursă (de exemplu, pragma o dată).

Un contract în sensul Solidity este o colecție de cod (funcțiile sale) și date care se află la o anumită adresă pe blockchain-ul Ethereum. Linia `uint Data;` declară o variabilă de stare numită `Data` de tip `uint` (întreg fără semn de 256 de biți). Vă puteți gândi la el ca pe un singur slot într-o bază de date pe care îl puteți interoga și modifica apelând funcțiile codului care gestionează baza de date. În acest exemplu, contractul definește funcțiile `set` and `get` care pot fi utilizate pentru a modifica sau a prelua valoarea variabilei.

Pentru a accesa un membru (cum ar fi o variabilă de stare) al contractului curent, de obicei nu se adugă prefixul `.this`, îl accesați direct prin numele său. Spre deosebire de alte limbi, omiterea acesteia nu este doar o chestiune de stil, ci are ca rezultat o modalitate complet diferită de a accesa membrul.



Acest contract permite oricui să stocheze un singur număr care este accesibil de oricine din lume fără o modalitate (fezabilă) de a vă împiedica să publicați acest număr. Oricine ar putea apela set din nou cu o valoare diferită și poate suprascrie numărul dvs., dar numărul este încă stocat în istoricul blockchain-ului.

2.6.3. NFT – Token nefungibil

Un **jeton nefungibil** este o unitate de date unică dintr-un registru contabil informatic denumit blockchain. Jetoanelor nefungibile le corespund fișiere de diverse formate, în funcție de natura procesului de creație: fotografii, înregistrări audio, videoclipuri etc. Deși fișierele în sine pot fi copiate, jetoanele nefungibile care le sunt asociate sunt consemnate și monitorizate în permanență în registrele blockchain din care fac parte, oferindu-le cumpărătorilor dovada dreptului de proprietate.

Dreptul de proprietate și unicitatea unui jeton nefungibil pot fi verificate cu ajutorul registrului blockchain din care face parte. NFT-urile au metadate procesate cu ajutorul unei funcții hash criptografice, un algoritm care generează o secvență unică alcătuită din 40 de litere și cifre.

Un token este pur și simplu un contract inteligent sau o bucată de cod pe Ethereum. Un token ERC-721 este creat prin scrierea unei bucăți de cod într-un limbaj de programare smart contract, cum ar fi Solidity, care urmează același șablon de bază sau cod de bază.

Odată ce șablonul de bază este urmat, puteți decide asupra detaliilor unice despre jetonul pe care îl creați, cum ar fi proprietarul, numele jetonului, simboluri etc. Puteți chiar să programați funcționalități suplimentare în NFT, dar adevărata distracție este modul în care NFT interacționează cu alte contracte inteligente.

ERC-721 este un standard gratuit, care descrie cum să construiți jetoane nefungibile sau unice pe blockchain-ul Ethereum. În timp ce majoritatea jetoanelor sunt fungibile (fiecare jetoane este la fel ca orice alt jetoane), jetoanele ERC-721 sunt toate unice.

ERC-721 definește o interfață minimă pe care un contract inteligent trebuie să o implementeze pentru a permite gestionarea, deținerea și tranzacționarea token-urilor unice. Nu impune un standard pentru metadatele jetoanelor și nu restricționează adăugarea de funcții suplimentare.

2.7. Conceptul Dapps (Aplicații decentralizate)

O aplicație se califică drept DApp (pronunțată ca Dee App) dacă îndeplinește următoarele patru criterii:

- Aplicația trebuie să fie open source, adică cod sursă liber disponibil publicului. În plus, aplicația trebuie să funcționeze autonom fără ca nicio entitate majoră să controleze jetoanele, adică protocolul cererii va fi pe deplin reglementat de consensul utilizatorilor.
- Aplicația nu trebuie să aibă un singur punct de eșec, cu toate log-urile de date și operațiuni stocate criptografic într-un spațiu public blockchain.
- Aplicația trebuie să folosească un anumit tip de token nativ pentru diverse procese și acces și orice contribuție de valoare trebuie să fie recompensat în astfel de jetoane.
- Aplicația ar trebui să folosească unele standarde și algoritmi criptografici ca dovadă de valoare adusă de un nod la generarea de jetoane.

2.7.1. Dezvoltarea aplicațiilor decentralizate

Figura 1.15 reprezintă o Aplicație decentralizată Ethereum la un nivel înalt. Dacă observați, fiecare browser client comunică cu propria instanță a aplicației. Nu există un server central la care toți clienții să se conecteze. Aceasta înseamnă că fiecare persoană care dorește să interacționeze cu o aplicație descentralizată va avea nevoie de o copie completă a blockchain-ului care rulează pe computerul/telefonul său și așa mai departe. Aceasta înseamnă că, înainte de a putea folosi o aplicație, trebuie să descarcăm întregul blockchain și apoi să începem să folosim aplicația. Acest lucru ar putea suna ridicol la început, dar are avantajul de a nu se baza pe un singur server central care ar putea dispărea mâine sau de a începe să ne percepe comisioane de la terți.

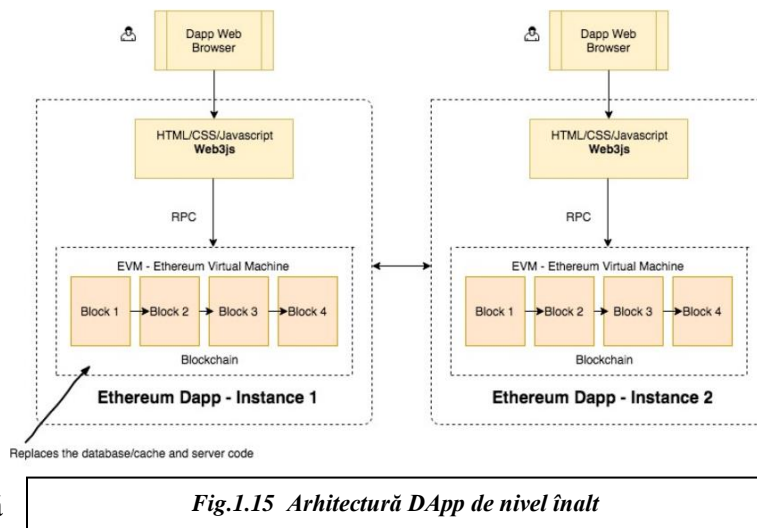


Fig.1.15 Arhitectură DApp de nivel înalt

Pentru a construi DApp-uri bazate pe web, Ethereum vine cu o bibliotecă JavaScript la îndemână numită web3.js, care se conectează la nodul nostru blockchain folosind apeluri de procedură la distanță (RPC). Deci, putem include această bibliotecă în cadrele noastre JavaScript preferate, cum ar fi ReactJS sau AngularJS, și să începem să dezvoltăm DApp-ul nostru.



2.7.2. Utilitare: Node.js, React, Web3.js

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. Adesea folosit pentru aplicații în timp real, cum ar fi chat, fluxuri de știri și notificări push. 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. Noțiuni de bază în Node.js 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.

Unul dintre principalele avantaje ale Node.js, conform creatorului său Ryan Dahl, este că nu blochează I / O (Input / Output). Dacă un proces necesită un număr semnificativ de cicluri de procesor, aplicația este blocată. Acest lucru poate cauza o defecțiune. Suporterii modelului Node.js susțin că timpul de procesare a procesorului este mai puțin preocupat de numărul mare de procese mici pe care se bazează codul site-ului. Popularitatea aplicațiilor javascript este în plină expansiune în ultimii ani, iar Node.js contribuie cu siguranță la această creștere. Dacă ne uităm la statistici, vom vedea că există mai multe pachete Node în lume decât date Ruby similare. Al doilea factor: pachetele de noduri cresc mai repede decât Ruby, Python și Java.

ReactJS ne permite să cream repede un frontend scalabil și ușor de utilizat pentru aplicațiile web. Este unul dintre framework-urile open-source cele mai populare în rândul programatorilor dar și al mediului de afaceri, mulțumită avantajelor sale în dezvoltarea aplicațiilor web:

- Permite dezvoltarea de aplicații la scară largă, în scurt timp și la un nivel superior de calitate.
- Permite scrierea de cod curat, modular și reutilizabil.
- Este ideal pentru proiecte ce pot fi sparte în componente separate.
- Aplicațiile web construite în ReactJS sunt flexibile, se pliază pe cerințele SEO și sunt ușor de scalat și de menținut.
- Este ușor să se treacă la React Native și să se creeze aplicații mobile cu un aspect nativ.

Web3.js oferă legături JavaScript către Ethereum, care pot fi apoi folosite pentru a construi interfețe intuitive cu utilizatorul folosind stiva web. Oferă o abstractizare pentru a ascunde lucrările interne complexe ale smart contracte pe blockchain. Cu cuvinte simple, o aplicație web implementată în JavaScript comunică cu un nod Ethereum sau interacționează cu un contract inteligent pe blockchain folosind biblioteca Web3.js. (Figura 1.16 ilustrează această interacțiune)

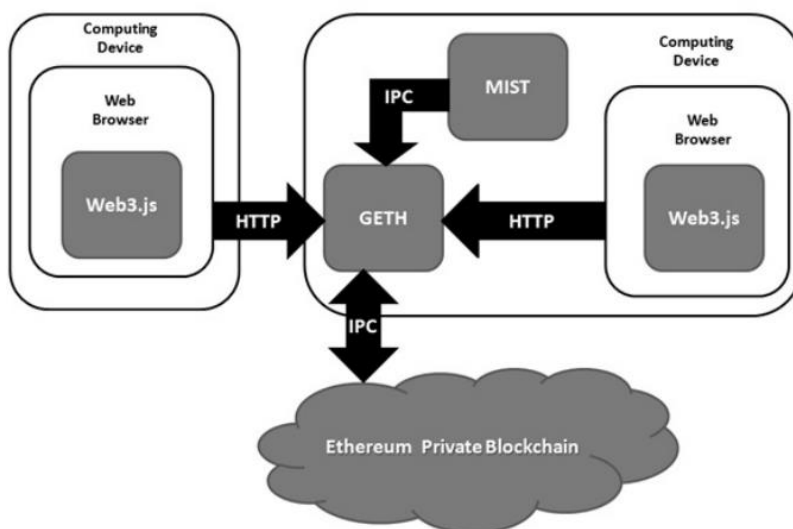


Fig.1.16 Web3.js interacționează cu un blockchain privat prin geth

Web3.js utilizează apeluri de procedură la distanță (RPC).

2.7.3. Framework Truffle

Truffle este un mediu de dezvoltare de clasă mondială, un cadru de testare pentru blockchain-uri care utilizează Ethereum Virtual Machine (EVM).

Truffle pune la dispoziția dezvoltatorului:

- Compilarea, implementarea și gestionarea binară a contractelor inteligente încorporate.
- Depanare avansată cu puncte de întrerupere și analiză variabilă.
- Implementări și tranzacții prin MetaMask pentru a vă proteja mnemonicul.
- Execuție de script extern care execută scripturi într-un mediu Truffle.
- Consolă interactivă pentru comunicare directă prin contract.
- Testare automată a contractelor pentru dezvoltare rapidă.
- Cadru de implementare și migrare scriptabil, extensibil.
- Managementul rețelei pentru implementare în orice număr de rețele publice și private.
- Gestionarea pachetelor cu NPM, folosind standardul ERC190 .

2.7.4. Simulatorul Ganache

Ganache este un blockchain personal pentru dezvoltarea rapidă a aplicațiilor distribuite Ethereum și Corda. Putem utiliza Ganache pe tot parcursul ciclului de dezvoltare; permițându-ne să dezvoltăm, să implementăm și să testăm aplicațiile decentralizate într-un mediu sigur și determinist.

Ganache vine în două variante: UI și CLI. Ganache UI este o aplicație desktop care acceptă atât tehnologia Ethereum, cât și Corda.

3. Studiu de caz: Aplicație decentralizată pentru prescripții medicale electronice în rețeaua Ethereum

3.1. Introducere

Prescripții ETH - este un sistem descentralizat de management al identității și ERP (Sistem gestiune financiară) farmaceutic construit pe o rețea de consorțiu Ethereum autorizată. Acest proiect implementează un standard de token nefungibil (NFT) pentru a elimina complet riscul și costurile implicate asociate falsificării prescripțiilor medicale. Prin crearea unui standard de contract inteligent descentralizat care definește metoda de proprietate și transferabilitate a unei rețete medicale se dorește:

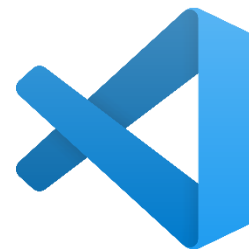
- Eliminarea posibilității de rețete contrafăcute/falsificate.
- Crearea unei perspective de reglementare asupra cantității, concentrației pentru eliberarea de medicamente.
- Crearea unei înregistrări imuabile a mișcării, cantității și tipului de prescripție medicală.

Prescripții ETH realizează acest obiectiv prin tokenizarea prescripțiilor farmaceutice într-un standard de contract de soliditate ERC-721. Acest token nefungibil conține metadata care descriu ID-ul medicului, cheia publică a pacientului, ID-ul medicamentului și cantitatea unității care trebuie eliberată.

3.2. Mediul de dezvoltare

Proiectul Prescripției ETH are următoarele dependențe tehnice:

- Visual Studio Code (IDE – Mediul de dezvoltare)
- Node.js (mediu de execuție JavaScript)
- React.js (bibliotecă Javascript)
- NPM (modul node.js)
- WEB3 (bibliotecă JavaScript utilizată pentru interacțiunea cu blockchain-ul)
- Ganache (Ganache este un blockchain personal pentru dezvoltarea rapidă a aplicațiilor distribuite Ethereum și Corda)
- Truffle (Framework de dezvoltare și compilare a contractelor inteligente Ethereum)



3.2.1. Instalare Node.js

Node.js este un mediu de execuție JavaScript de fundal multiplatformă cu sursă deschisă, care rulează pe motorul V8 și execută cod JavaScript în afara unui navigator web. Acesta se instalează cu pachetul descărcat de la adresa oficială: <https://nodejs.org/en/download/>

Instalarea se realizează utilizând interfața de instalare, acceptând condițiile și configurând căile de instalare.

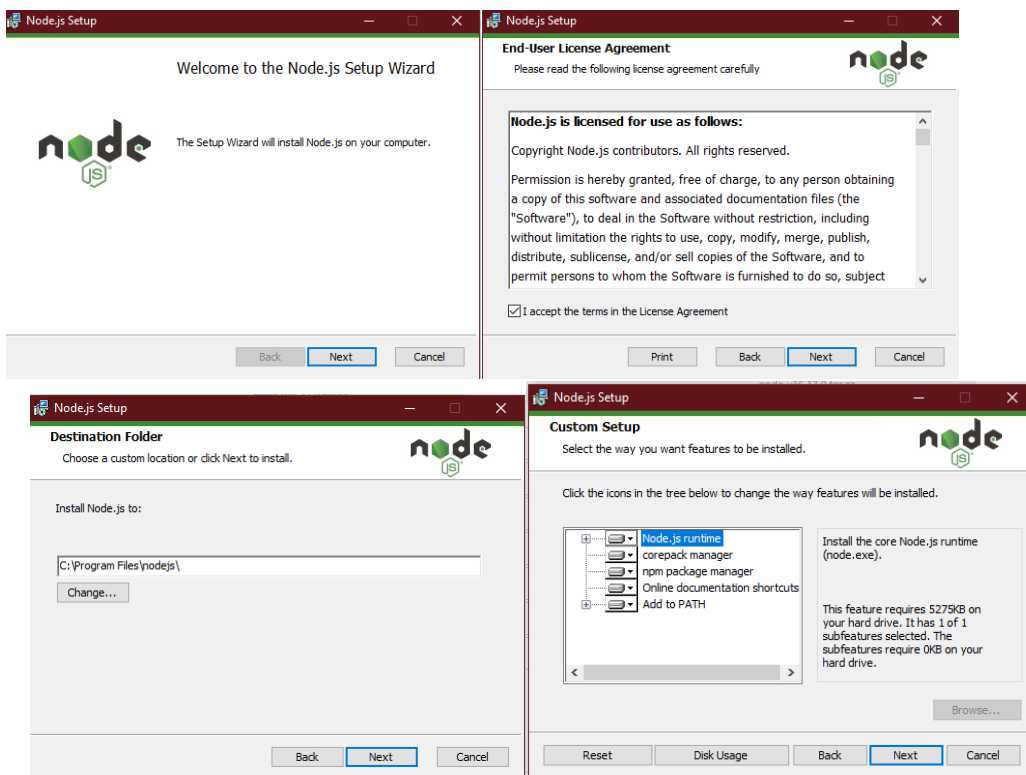


Fig.2.1 Instalare Node.js

De regulă, node.js va fi mapat automat la variabilele de environment al sistemului de operare, astfel acesta va putea fi accesat din orice director, de exemplu:

➤ `node -version` (returnează versiunea de node.js utilizată)

De asemenea avem nevoie și de modulul npm - un manager de pachete pentru limbajul de programare JavaScript menținut de npm, Inc.

Acesta se instalează rulând comanda:

➤ `npm install -g npm`

3.2.2. Instalare Truffle

Pachetul truffle se instalează utilizând modulul npm, rulând în linia de comandă:

➤ `npm install truffle -g`

3.2.3. Instalare și configurarea Ganache

Instalea Ganache, la fel ca și node.js se face printr-o interfață de instalare descărcat de la adresa: <https://trufflesuite.com/ganache/>



După instalare, configurăm un nou blockchain local:

1) Creem un spațiu de lucru (New Workspace)

2) Edităm opțiunile de configurare:

- *Nume proiect*
- *Locația fișierul de configurare (truffle-config.js)*
- *Serverul (localhost – 127.0.0.1)*
- *Portul (7545)*
- *ID-ul rețelei (5777)*
- *Minare automata – Pornit*
- *Generare a 10 portofele cunoscute cu câte 100 Unități fiecare)*
- *Configurarea comisionului de rețea (Gas price)*

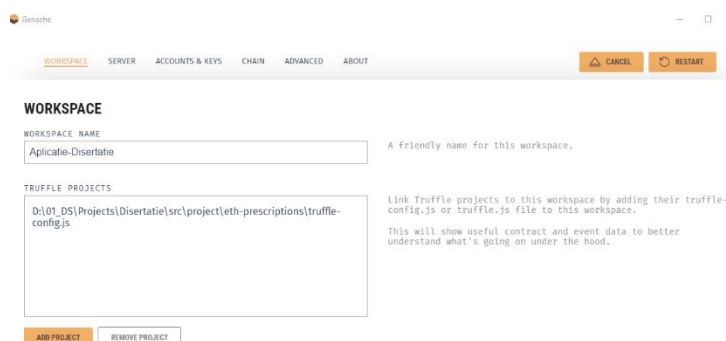
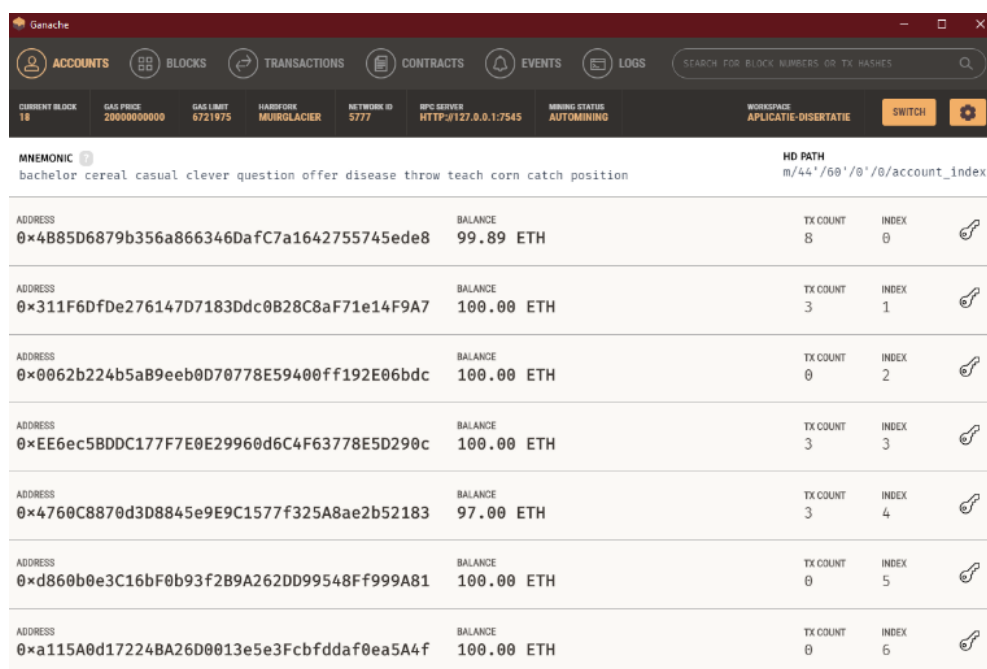


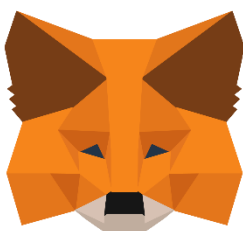
Fig.2.2 Configurator Ganache



Ganache			
ACCOUNTS BLOCKS TRANSACTIONS CONTRACTS EVENTS LOGS			
SEARCH FOR BLOCK NUMBERS OR TX HASHES			
CURRENT BLOCK 18	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MUROLACIER
NETWORK ID 5777		RPC SERVER HTTP://127.0.0.1:7545	MINE STATUS AUTOMINING
WORKSPACE APLICATIE-DISERTATIE			SWITCH
MNEMONIC bachelor cereal casual clever question offer disease throw teach corn catch position		HD PATH m/44'/60'/0'/0/account_index	
ADDRESS 0x4B85D6879b356a866346DafC7a1642755745ede8	BALANCE 99.89 ETH	TX COUNT 8	INDEX 0
ADDRESS 0x311F6DfDe276147D7183Ddc0B28C8aF71e14F9A7	BALANCE 100.00 ETH	TX COUNT 3	INDEX 1
ADDRESS 0x0062b224b5aB9eeb0D70778E59400ff192E06bdc	BALANCE 100.00 ETH	TX COUNT 0	INDEX 2
ADDRESS 0xEE6ec5BDDC177F7E0E29960d6C4F63778E5D290c	BALANCE 100.00 ETH	TX COUNT 3	INDEX 3
ADDRESS 0x4760C8870d3D8845e9E9C1577f325A8ae2b52183	BALANCE 97.00 ETH	TX COUNT 3	INDEX 4
ADDRESS 0xd860b0e3C16bF0b93f2B9A262DD99548Ff999A81	BALANCE 100.00 ETH	TX COUNT 0	INDEX 5
ADDRESS 0xa115A0d17224BA26D0013e5e3Fcbfddaf0ea5A4f	BALANCE 100.00 ETH	TX COUNT 0	INDEX 6

Fig.2.3 Interfața Ganache

3.2.4. Instalare și configurarea Metamask



Extensia de browser Metamask poate fi descărcată direct de pe website-ul oficial: <https://metamask.io/>

După instalare se accesează setările pentru a face legătura portofelului electronic, cu blockchain-ul local creat în ganache.

Metamask > Setari > Retele > Adauga Reta

Se adaugă rețeaua conform cerințelor (Nume rețea, RPC URL, Chain ID, Currency Symbol)

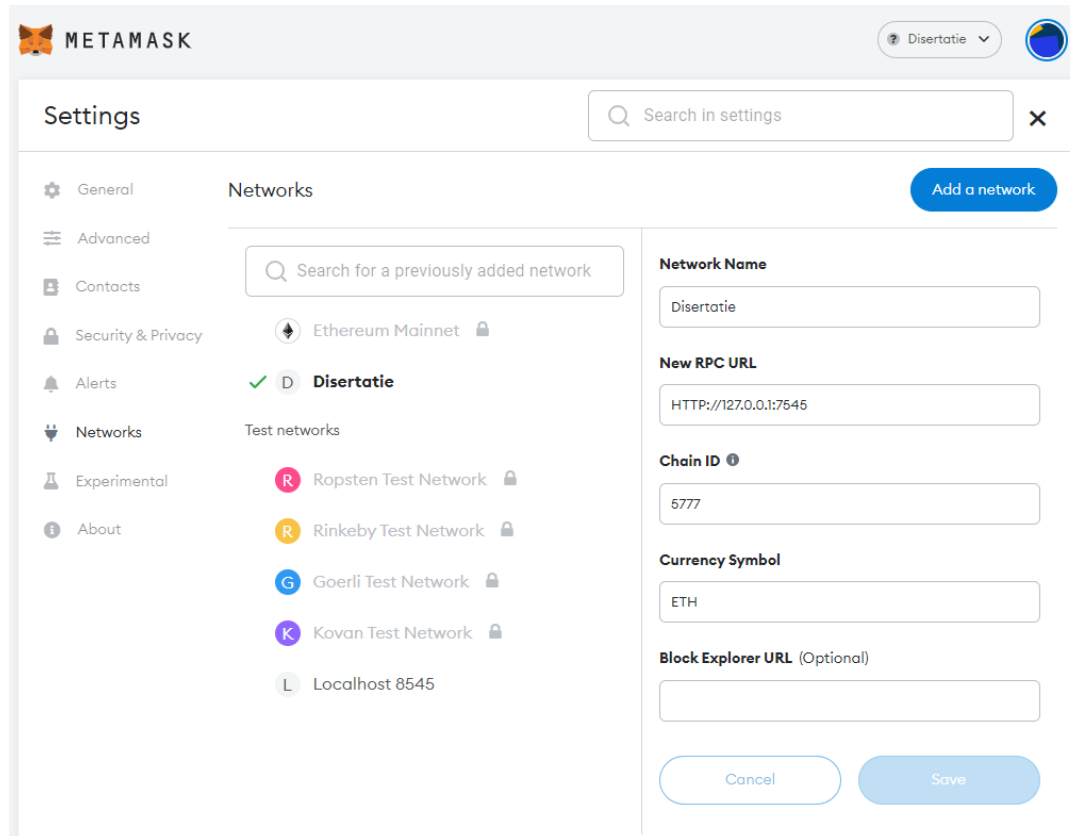


Fig.2.5 Configurare metamask

3.3. Smart Contractul „PrescriptionNFT.sol”

Fișierul PrescriptionNFT.sol este un smart-contract, scris în limbaj solidity, care formalizează tokenizarea prescripțiilor medicale. Tokenul primește o intrare de la un medic, care este oficializată ca metadate. Aceste intrări sunt:

- Doctor - adresa ETH a unui medic
- Cheia publică a pacientului - acționează ca un mecanism de verificare pentru farmacist pentru a preveni transferul necinstit
- Numele medicamentului - numele medicamentului
- Brand Name - marca specifică a produsului farmaceutic
- Dozaj - un număr care reprezintă sarcina utilă a pastilei care urmează să fie eliberată de farmacist
- Unitate de dozare - aceasta va fi unitatea de măsură oficializată (adică mililitri)
- Data Filled - data la care a fost completată comanda de către farmacist



- Cantitate totală - cantitatea totală de pastile care urmează să fie eliberată de către farmacist pacientului
- Data expirării - data la care expiră prescripția. Verificăm data de expirare când pacientul încearcă să completeze rețeta

În continuare se va detalia conținutul smart-contractului:

- 1) Declarea contractului. Acesta moștenește metodele unui contract NFT ERC721

```
- contract PrescriptionNFT is ERC721 {  
-     using SafeMath for uint256;  
-     //COD  
- }
```

- 2) Definirea structurilor pentru metadata:

```
struct PrescriptionMetadata {  
    //adresa medicului care prescrie  
    address doctor;  
    //adresa pacientului care primește  
    address prescribedPatient;  
    //Codul de bare al medicamentului  
    string pzn;  
    //Numele medicamentului  
    string medicationName;  
    //Dozarea  
    uint8 dosage;  
    //Unitatea de masura  
    string dosageUnit;  
    //Numarul de pastile  
    uint8 numPills;  
    //data la care s-a prescris  
    uint256 dateFilled;  
    //Data expirării prescripției  
    uint256 expirationTime;  
}  
  
//Metadata pentru o prescripție  
struct Prescription {  
    PrescriptionMetadata metadata;  
    address owner;  
    bool filled;  
}
```

```
//Metadatele pentru validarea medicului  
struct Doctor {  
    string name;  
    bool isValid;  
}
```

3) Maparea variabilelor globale (Maparea este un tip de referință ca matrice și structuri).

Maparea poate avea doar tip de stocare și sunt utilizate în general pentru variabilele de stare.

Exemplu:

```
mapping (uint256 => Prescription) public prescriptions;
```

4) Validarea adresei medicului:

```
constructor (address[] memory doctorAddresses, string[] memory doctorNames)  
public payable {  
    owner = msg.sender;  
    require(doctorAddresses.length == doctorNames.length);  
    // Crearea unui tablou cu adresele valide ale medicilor  
    for (uint i = 0; i < doctorAddresses.length; i++) {  
        approvedDoctors[doctorAddresses[i]] = Doctor(doctorNames[i], true);  
    }  
}
```

5) Funcția pentru crearea unei prescrieri

```
function prescribe(  
    address _patientAddress,  
    string memory _pzn,  
    string memory _medicationName,  
    uint8 _dosage,  
    string memory _dosageUnit,  
    uint8 _numPills,  
    uint256 _dateFilled,  
    uint256 _expirationTime) public doctorIsApproved(msg.sender) {  
    uint256 newTokenId = totalTokens;  
    prescriptions[newTokenId].filled = false;  
    prescriptions[newTokenId].metadata = PrescriptionMetadata(  
        msg.sender,  
        _patientAddress,  
        _pzn,
```



```
        _medicationName,  
        _dosage,  
        _dosageUnit,  
        _numPills,  
        _dateFilled,  
        _expirationTime  
    );  
    issuedTokensIndex[newTokenId] = issuedTokens[msg.sender].length;  
    issuedTokens[msg.sender].push(newTokenId);  
    _mint(_patientAddress, newTokenId);  
}
```

6) Funcția pentru anularea prescripției:

```
function cancelPrescription(uint256 _tokenId) public payable  
doctorIsApproved(msg.sender){  
    require(_tokenId < totalTokens);  
    Prescription memory p = prescriptions[_tokenId];  
    require(p.metadata.doctor == msg.sender);  
    require(p.filled == false);  
    removeToken(p.metadata.prescribedPatient, _tokenId);  
    destroyToken(p.metadata.doctor, _tokenId);  
}
```

7) Funcția pentru Utilizarea Prescripției

```
function fillPrescription(address _pharmacyAddress, uint256 _tokenId) public  
//hasNotExpired(_tokenId)  
{  
    transfer(_pharmacyAddress, _tokenId);  
    prescriptions[_tokenId].filled = true;  
}
```

8) Alte funcții de operare caracteristice unui NFT:

- Mint, Transfer, balanceOf, ownerOf, removeToken, addToken, destroyToken.

Codul complet este regăsit în proiect, găzduit pe github, la locația:

Proiect -> contracts -> token -> PrescriptionNFT.sol



3.3.1. Rulare smart contractului pe blockchain

Atât compilarea fișierelor .sol, cât și migrarea pe blockchain se realizează cu instrumentele de la truffle.

În primă fază este creat sau editat fișierul truffle-config.js din folderul rădăcină al proiectului.

Aici configurăm rețeaua blockchain-ului, compilatorul, cât și alte dependențe:

```
networks: {  
  development: {  
    host: "127.0.0.1",      // Localhost (default: none)  
    port: 7545,            // Standard Ethereum port (default: none)  
    network_id: "*",      // Any network (default: none)  
  }  
}
```

Compilarea:

Este necesar un director (build/contracts) în rădăcina proiectului pentru artifactele obținute la compilare.

Comanda utilizată la compilare: `> truffle --compile`

Aceasta va compila toate fișierele .sol din directorul "/contracts".

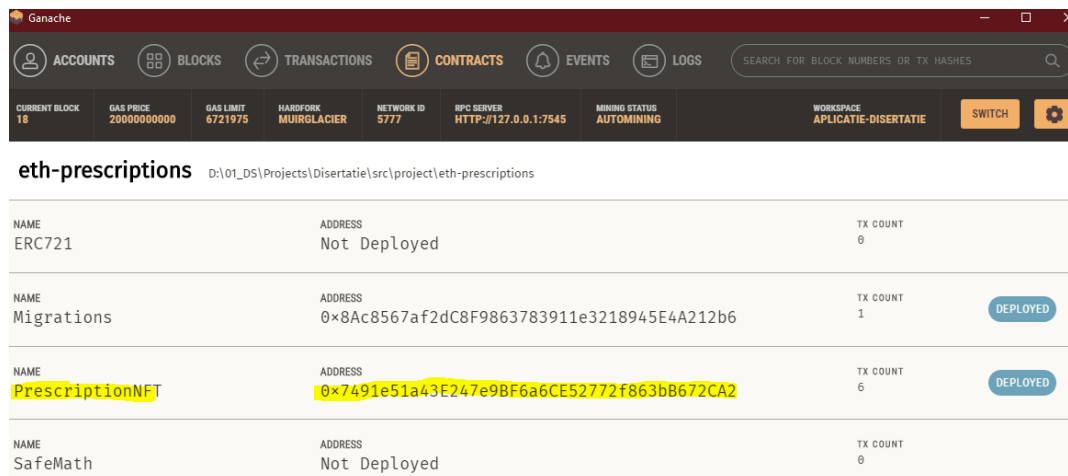
Migrarea

Funcția **deploy** este responsabilă de încărcarea smart-contractului în rețea. Pe lângă aceasta specificăm și adresele medicilor autorizați pentru a face prescripții. Codul de execuție poate fi editat din fișierul migrations/2_deploy_contracts.js.

Pentru a încărca smart-contract-ul în blockchain, se utilizează comanda: `> truffle --migrate`

Rezultat:

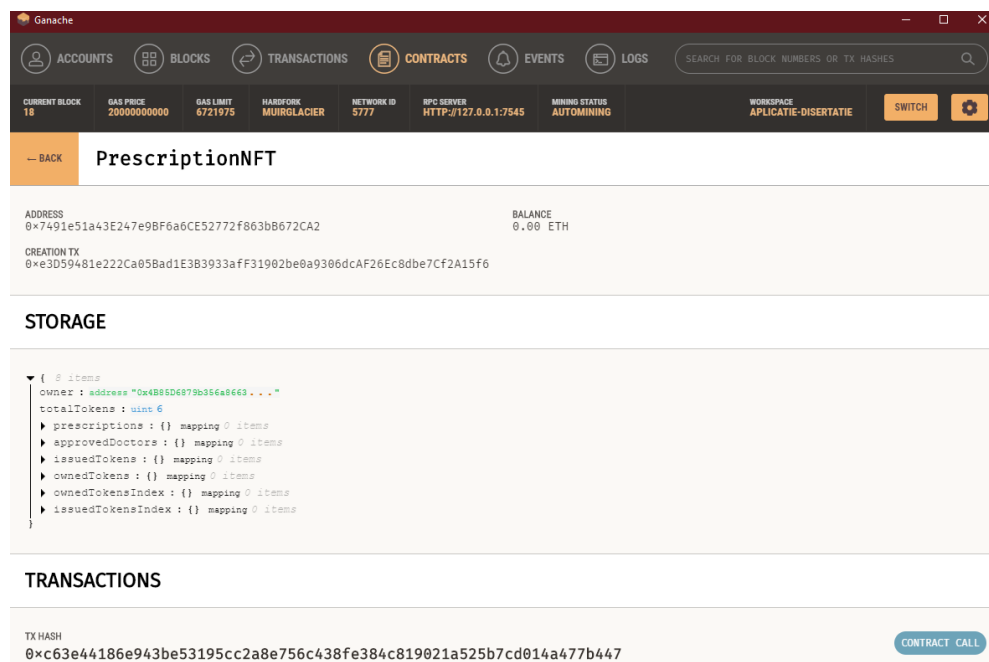
S-a creat un smart-contract (PrescriptionNFT) (figura 2.6) la adresa 0x7491e51a43E247e9BF6a6CE52772f863bB672CA2



NAME	ADDRESS	TX COUNT	
ERC721	Not Deployed	0	
Migrations	0x8Ac8567af2dC8F9863783911e3218945E4A212b6	1	DEPLOYED
PrescriptionNFT	0x7491e51a43E247e9BF6a6CE52772f863bB672CA2	6	DEPLOYED
SafeMath	Not Deployed	0	

Fig.2.6 Vizualizare cntracte smart

Se poate vizualiza atât metadatele, metodele publice, cât și istoricul tranzacțiilor (fig. 2.7):



PrescriptionNFT

ADDRESS: 0x7491e51a43E247e9BF6a6CE52772f863bB672CA2
BALANCE: 0.00 ETH

CREATION TX: 0xe3D59481e222Ca058ad1E3B3933affF31902be0a9306dcAF26Ec8dbe7Cf2A15f6

STORAGE

```

{
  0 items
  owner : address "0x4B85D6975b356a8663..."
  totalTokens : uint 6
  prescriptions : {} mapping 0 items
  approvedDoctors : {} mapping 0 items
  issuedTokens : {} mapping 0 items
  ownedTokens : {} mapping 0 items
  ownedTokensIndex : {} mapping 0 items
  issuedTokensIndex : {} mapping 0 items
}
  
```

TRANSACTIONS

TX HASH: 0xc63e44186e943be53195cc2a8e756c438fe384c819021a525b7cd014a77b447
CONTRACT CALL

Fig.2.6 Vizualizare metadate și istoric cntracte smart

3.4. Aplicația „Interfața Medicului”

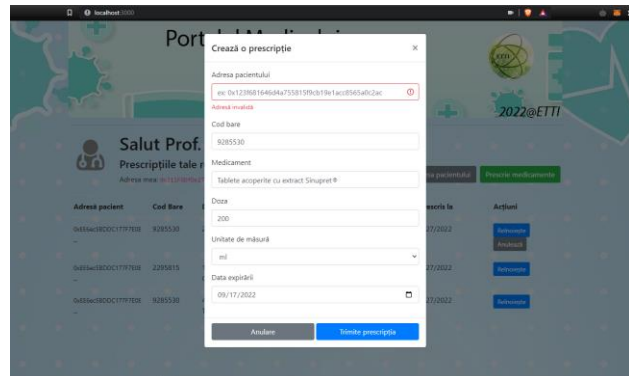
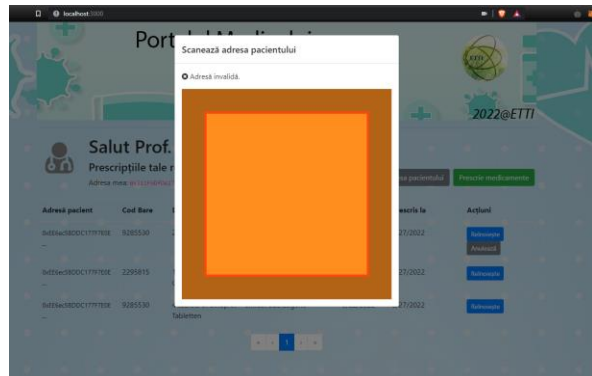


Fig.2.7 Interfața medicului

Funcționalități:

- 1) Logare în platformă cu portofelul Metamask utilizând WEB3.
- 2) Vizualizare istoric prescripții medicale.
- 3) Scanare QR-cod – adresa pacientului.
- 4) Prescriere medicamente.
- 5) Anularea prescripțiilor, până ca acestea să fie utilizate.
- 6) Reînnoirea prescripțiilor pacienților

3.4.1. Arhitectură Software

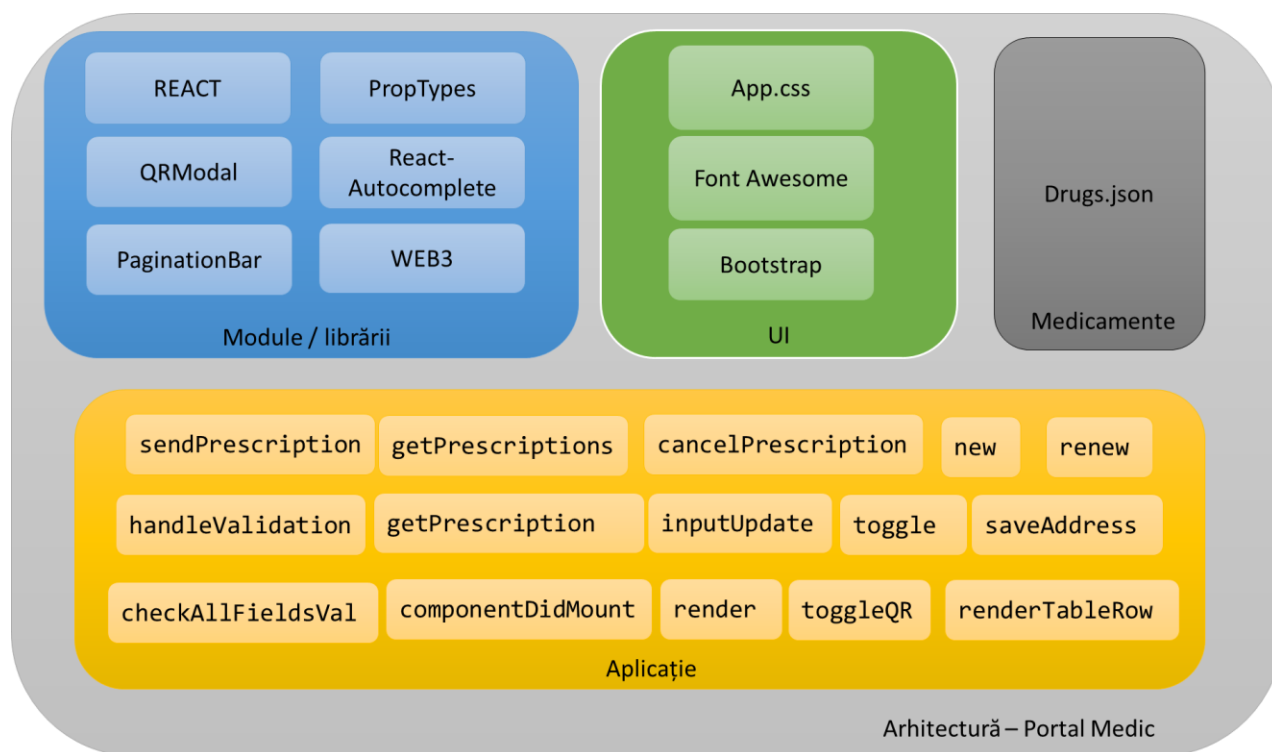


Fig.2.8 Arhitectura Software pentru aplicația medicului

Arhitectura aplicației poate fi vizualizată în figura 2.8, iar codul sursă este disponibil online, pe repoitoriul github din anexă.

Module / Librării:

- **React:** bibliotecă JavaScript open-source pentru construirea de interfețe de utilizator.
- **PropTypes:** exportă o serie de validatoare care pot fi utilizate pentru a se asigura că datele pe care le primiți sunt valide
- **QRModal:** componentă pentru scanarea codurilor QR dintr-o aplicație bazată pe browser web.
- **React-Autocomplete:** componentă pentru introducerea de text îmbunătățită de un panou de opțiuni sugerate.
- **PaginationBar:** Componentă ușoară pentru paginarea paginilor aplicației pe react
- **WEB3:** este un cadru simplu și puternic pentru construirea de aplicații moderne Ethereum folosind React.



Stilizare:

- **App.css:** Conține clasele de stiluri personalizate definite implicit pentru aplicația medicului.
- **Font Awesome:** este un set de instrumente pentru fonturi și pictograme bazat pe CSS și Less.
- **Bootstrap:** este un framework CSS gratuit și 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.

Listă medicamente:

- **Drugs.json:** Fișier ce conține toate medicamentele, inclusiv date despre acesta în format json.

Funcțiile aplicație:

Aplicația definește 2 clase care moștenesc metodele unei componente REACT:

- ModalForm (definește metodele utilizate în ferestrele pop-up):
 - *sendPrescription:* funcția apelată la tastarea butonului de trimitere prescripție.
 - *handleValidation:* validează prezența adresei pacientului și a datei expirării prescripției.
 - *inputUpdate:* funcție apelată la un anumit eveniment care ar necesita actualizarea formularului.
 - *checkAllFieldsVal:* validează datele introduse în formularul prescripției.
 - *render:* generează și afișează în pagină codul HTML cu datele dinamice.
- App (definește metodele utilizate în paginile statice):
 - *getPrescriptions:* interoghează blockchain-ul pentru a obține istoricul prescripțiilor.
 - *cancelPrescription:* este utilizată atunci când medicul dorește să anuleze o prescripție, din butonul disponibil.
 - *new:* actualizează lista cu prescripții după realizarea acesteia.
 - *renew:* crează o prescripție nouă, completând formularul în baza uneia mai veche.
 - *getPrescription:* returnează toate datele despre o singură prescripție.
 - *toggle:* deschide fereastra cu formularul prescripției.
 - *saveAddress:* completează automat câmpul pentru adresa pacientului, cu adresa scanată.
 - *componentDidMount:* funcție de inițializare, cu rolul de logare prin metamask.
 - *render:* generează și afișează în pagină codul HTML cu datele dinamice.
 - *toggleQR:* deschide fereastra cu scannerul QR-codului.

- *renderTableRow*: generează și afișează în pagină tabelul în cod HTML cu date dinamice.
- *render*: generează și afișează în pagină codul HTML cu datele dinamice.

3.5. Aplicația „Interfața Farmacistului”

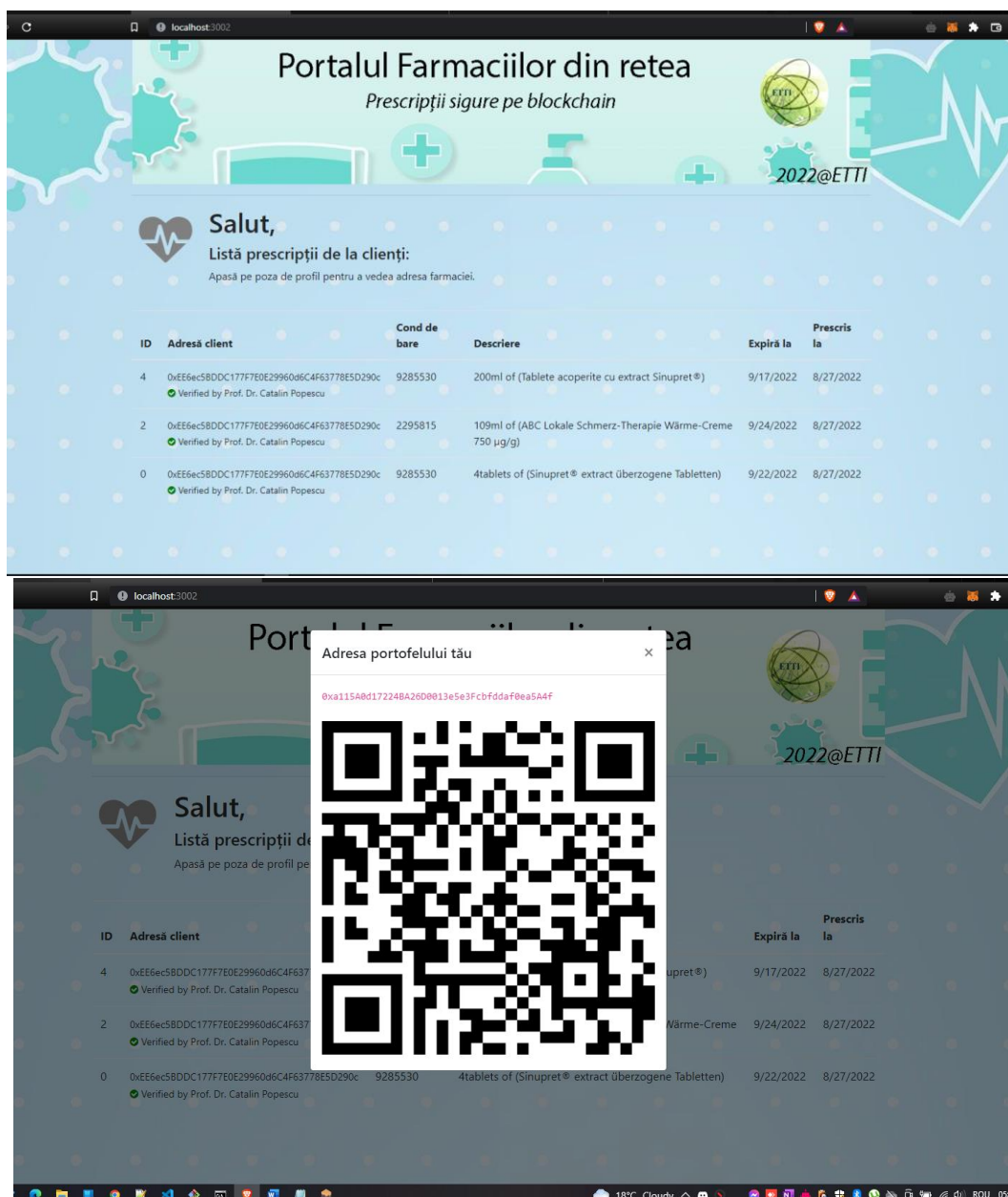


Fig.2.9 Interfața farmacistului

3.5.1. Arhitectură Software

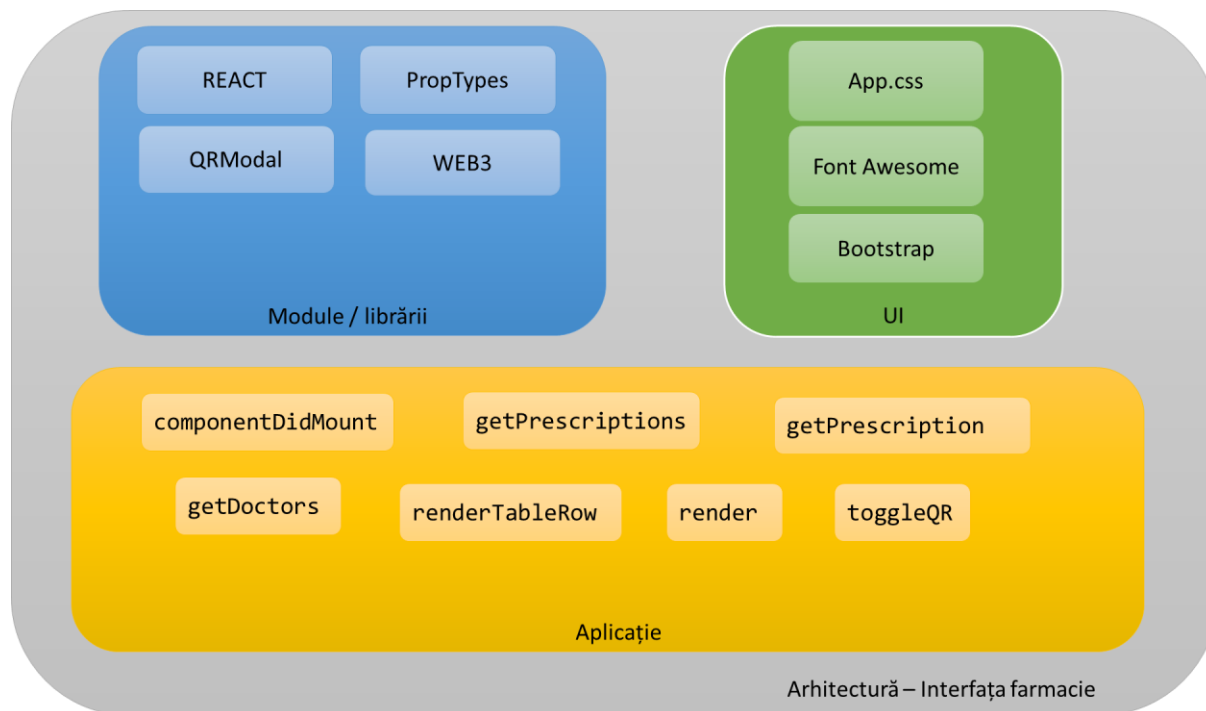


Fig.2.10 Arhitectura Software pentru aplicația farmaciei

Arhitectura aplicației poate fi vizualizată în figura 2.10, iar codul sursă este disponibil online, pe repoziitoriul github din anexă.

Module / Librării:

- **React:** bibliotecă JavaScript open-source pentru construirea de interfețe de utilizator.
- **PropTypes:** exportă o serie de validatoare care pot fi utilizate pentru a se asigura că datele pe care le primiți sunt valide
- **QRModal:** componentă pentru scanarea codurilor QR dintr-o aplicație bazată pe browser web.
- **WEB3:** este un cadru simplu și puternic pentru construirea de aplicații moderne Ethereum folosind React.



Stilizare:

- **App.css:** Conține clasele de stiluri personalizate definite implicit pentru aplicația medicului.
- **Font Awesome:** este un set de instrumente pentru fonturi și pictograme bazat pe CSS și Less.
- **Bootstrap:** este un framework CSS gratuit și 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.

Funcțiile aplicație:

Aplicația definește 2 clase care moștenesc metodele unei componente REACT:

- QRModal (definește metodele utilizate în fereaștrele pop-up):
 - *render:* generează și afișează în pagină codul HTML cu adresa și QR-ul farmaciei.
- App (definește metodele utilizate în paginile statice):
 - *componentDidMount:* funcție de inițializare, cu rolul de logare prin metamask.
 - *getPrescription:* returnează toate datele despre o singură prescripție.
 - *getPrescriptions:* interoghează blockchain-ul pentru a obține istoricul prescripțiilor.
 - *render:* generează și afișează în pagină codul HTML cu datele dinamice.
 - *getDoctors:* crează o listă cu medicii autorizați.
 - *toggleQR:* deschide fereaștra cu scannerul QR-codului.
 - *renderTableRow:* generează și afișează în pagină tabelul în cod HTML cu date dinamice.
 - *render:* generează și afișează în pagină codul HTML cu datele dinamice.

3.6. Aplicația „Interfața Pacientului”

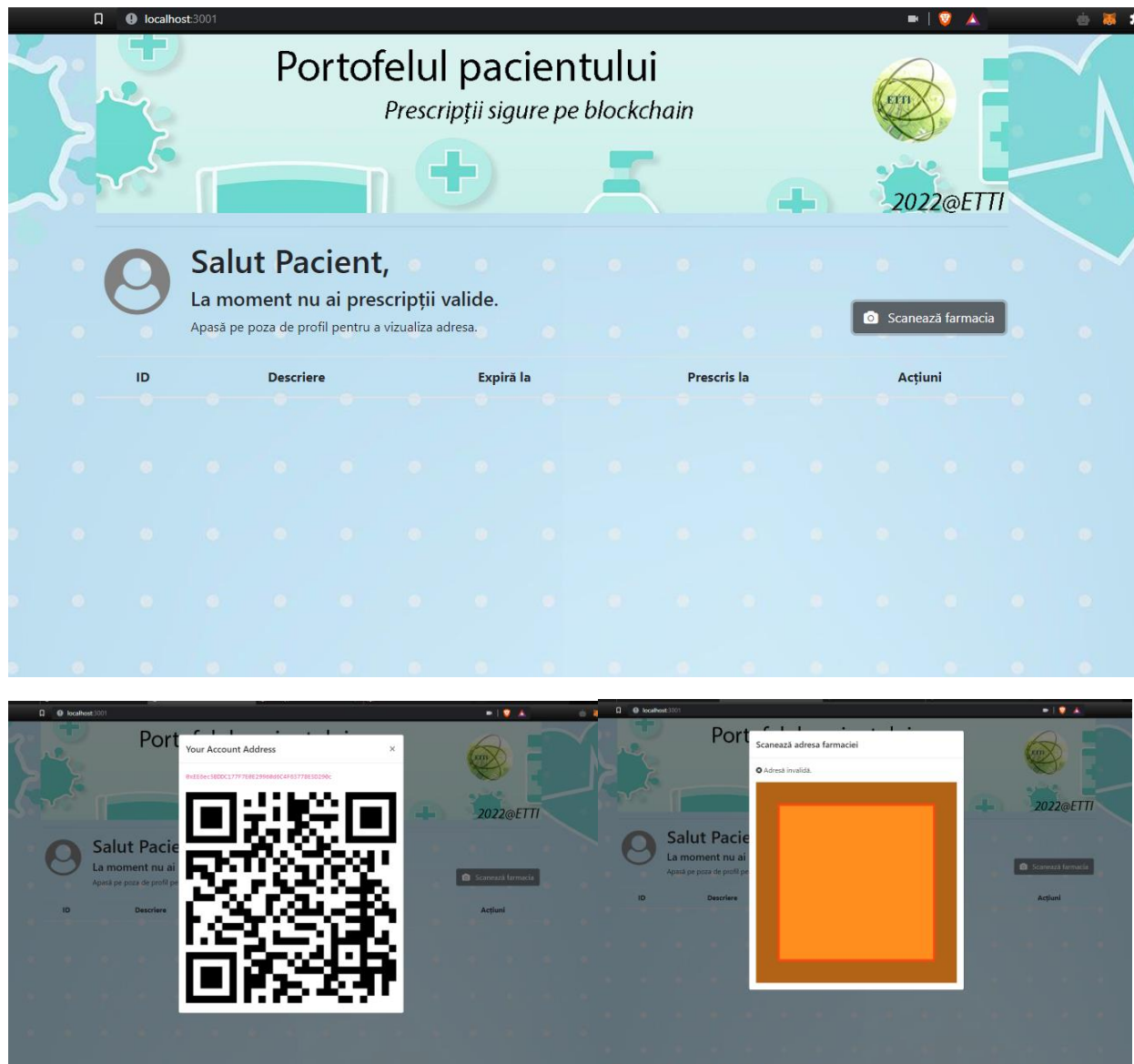


Fig.2.11 Interfața aplicației pacientului

3.6.1. Arhitectură Software

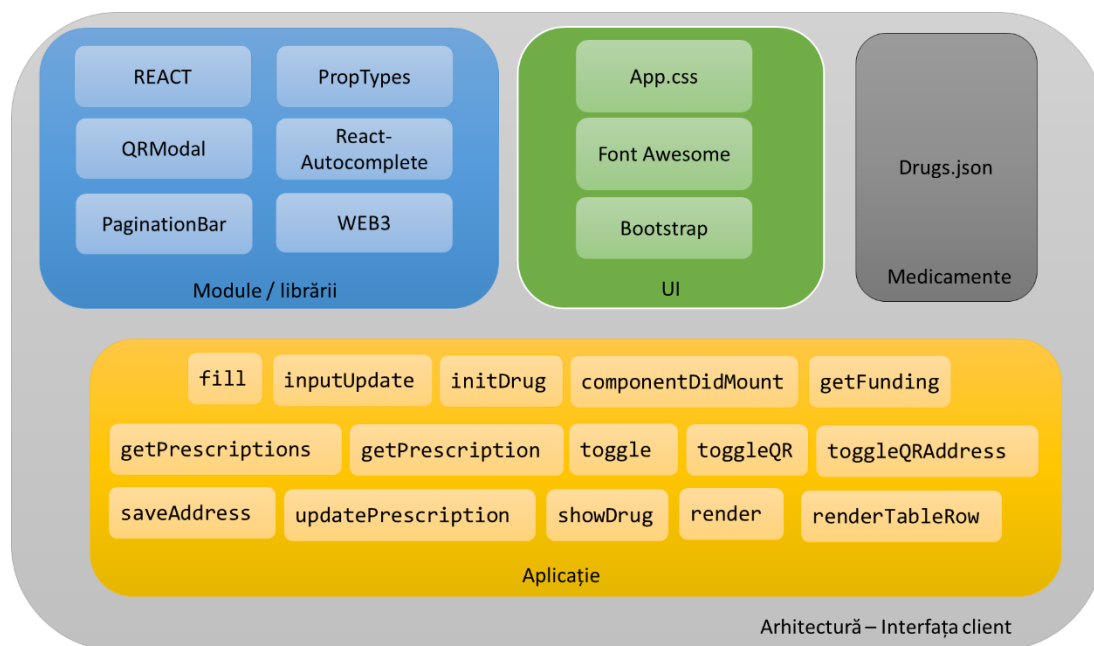


Fig.2.12 Arhitectura Software pentru aplicația clientului

Arhitectura aplicației poate fi vizualizată în figura 2.12, iar codul sursă este disponibil online, pe repositoryul github din anexă.

Module / Librării:

- **React:** bibliotecă JavaScript open-source pentru construirea de interfețe de utilizator.
- **PropTypes:** exportă o serie de validatoare care pot fi utilizate pentru a se asigura că datele pe care le primiți sunt valide
- **QRModal:** componentă pentru scanarea codurilor QR dintr-o aplicație bazată pe browser web.
- **React-Autocomplete:** componentă pentru introducerea de text îmbunătățită de un panou de opțiuni sugerate.
- **PaginationBar:** Componentă ușoară pentru paginarea paginilor aplicației pe react
- **WEB3:** este un cadru simplu și puternic pentru construirea de aplicații moderne Ethereum folosind React.



Stilizare:

- **App.css:** Conține clasele de stiluri personalizate definite implicit pentru aplicația medicului.
- **Font Awesome:** este un set de instrumente pentru fonturi și pictograme bazat pe CSS și Less.
- **Bootstrap:** este un framework CSS gratuit și 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.

Listă medicamente:

- **Drugs.json:** Fișier ce conține toate medicamentele, inclusiv date despre acesta în format json.

Funcțiile aplicație:

Aplicația definește 3 clase care moștenesc metodele unei componente REACT:

- ModalForm (definește metodele utilizate în ferestrele pop-up):
 - *Fill:* funcție pentru trimitere prescripție la farmacie,
 - *inputUpdate:* funcție apelată la un anumit eveniment care ar necesita actualizarea formularului.
 - *render:* generează și afișează în pagină codul HTML cu datele dinamice.
- DrugModal (definește metodele utilizate pentru afișarea detaliilor medicamentelor):
 - *initDrug:* funcție pentru afișare date despre medicament.
 - *render:* generează și afișează în pagină codul HTML cu datele dinamice.
- App (definește metodele utilizate în paginile statice):
 - *componentDidMount:* funcție de inițializare, cu rolul de logare prin metamask.
 - *getFunding:* funcție pentru interogarea portofelului clinetului în blockchain.
 - *getPrescriptions:* interoghează blockchain-ul pentru a obține istoricul prescripțiilor.
 - *getPrescription:* returnează toate datele despre o singură prescripție.
 - *cancelPrescription:* este utilizată atunci când medicul dorește să anuleze o prescripție, din butonul disponibil.
 - *toggle:* deschide fereastra cu formularul prescripției.
 - *toggleQR:* deschide fereastra cu scannerul QR-codului.
 - *toggleQRAddress:* deschide fereastra cu adresa pacientului și QR-codul.
 - *showDrug:* funcție pentru butonul "info" al medicamentelor.
 - *toggleDrug:* deschidere/închidere fereastră "informații medicament".



- *saveAddress*: completează automat câmpul pentru adresa pacientului, cu adresa scanată.
- *fill*: funcție apelată la tastarea butonului "Utilizează".
- *updatePrescription*: funcție apelată după finalizarea tranzacției pentru actualizarea stării prescripției.
- *renderTableRow*: generează și afișează în pagină tabelul în cod HTML cu date dinamice.
- *render*: generează și afișează în pagină codul HTML cu datele dinamice.

4. Concluzii

Această lucrare expune o aplicație practică pe blockchain, ce nu are legătură cu finanțele și specula pe monezi virtuale. Am reușit să demonstrez, că într-un portfel electronic se pot deține date valoroase, precum o prescripție medicală, astfel încât aceasta să nu poată fi falsificată, să existe transparență în transferurile acestora și totul să nu depindă de un server central.

Lucrul efectuat asupra aplicației practice, mi-a consolidat cunoștințele de dezvoltare web, a tehnologiilor precum node.js, react.js, web3.js; a pus o bază în abilitățile de a programa un smart-contract în limbaj solidity, inclusiv compilarea și executarea acestuia pe un blockchain în rețeaua locală.

Avantajul care îl aduce rețeaua Ethereum este de a oferi infrastructura necesară dezvoltatorilor pentru a-și concentra eforturile pe găsirea de utilizări inovatoare pentru aplicațiile digitale. Acest lucru ar putea permite implementarea rapidă a aplicațiilor dApp într-o varietate de industrii, inclusiv bancare și finanțe, jocuri, rețele sociale și cumpărături online.

Ca dezavantaje utilizarea dApps este încă în stadii incipiente, prin urmare, este experimentală și predispusă la anumite probleme și necunoscute. În cazul aplicației expuse în lucrare, nu am putut include date personale, precum Nume, Prenume, CNP Client, astfel prescripția fiind total anonimă. Acest lucru ar putea fi soluționat prin îmbinarea Smart-Contractului cu o aplicație web centralizată, administrată de o autoritate competentă.

De regulă, aplicațiile decentralizate trebuie să fie OPEN Source (cod liber pentru dezvoltatori), prin urmare codul sursă aferent lucrării este disponibil pe github [8] .



5. Bibliografie

1. Buletin Special Blockchain SRI: <https://www.sri.ro/assets/files/publicatii/buletin-special-cyber-blockchain.pdf>
2. „Ethereum Smart Contract Development” - Mayukh Mukhopadhyay
© 2018 Packt Publishing ISBN 978-1-78847-304-0
3. „Learn Blockchain by Building One” - Daniel van Flymen, New York, NY, USA
ISBN-13 (pbk): 978-1-4842-5170-6
4. Documentația publică ETHEREUM: <https://ethereum.org/en/developers/docs/intro-to-ethereum/>
5. Wikipedia – Enciclopedie liberă (Consensus, NFT, Ethereum, Dapps) -
[https://en.wikipedia.org/wiki/Consensus_\(computer_science\)](https://en.wikipedia.org/wiki/Consensus_(computer_science))
6. Limbajul solidity - <https://docs.soliditylang.org/en/v0.8.15/>
7. Standardul ERC721 - <https://erc721.org/>
8. Cod sursă aplicație - <https://github.com/DanielSergheenco/eth-prescriptions>
9. Documentație REACT.js - <https://reactjs.org/docs/getting-started.html>
10. Documentație NODE.js – <https://nodejs.org/en/docs/>
11. Documentație WEB3.js - <https://web3js.readthedocs.io/en/v1.7.5/>
12. Instrumente blockchain local - <https://trufflesuite.com/docs/>
13. Metamask – portofel electronic - <https://metamask.io/>
14. Aplicații decentralizate - <https://invatatiafaceri.ro/dictionar-financiar/aplicatii-descentralizate-dapps/>
15. Suport dezvoltare - <https://stackoverflow.com/>
16. „Building Ethereum DApps: Decentralized Applications on the Ethereum Blockchain 1st Edition” - Roberto Infante - ISBN-13: 978-1617295157
17. Prezentare DEMO -
https://www.youtube.com/watch?v=DJbFknG_gAE&fbclid=IwAR0cA9Qbg98AU9fsBKdqDSg8-vM1FidrzBA1jgFtvpV3IwEJKCW4K40rOFQ



6. Rezumat

Lucrarea „WEB 3.0: Dezvoltarea aplicațiilor decentralizate pe blockchain-ul Ethereum” urmărește studiul tehnologiei blockchain pentru a dezvolta o aplicație decentralizată, utilă în domeniul medical. Aceasta începe cu un capitol intitulat Introducere, în care se prezintă motivația dezvoltării aplicațiilor propuse, cerințele funcționale, precum și obiectivele propuse spre a fi urmărite în vederea finalizării proiectului.

Capitolul 2 este intitulat „Bazele Tehnologiei Blockchain. Concepte Teoretice” și cuprinde noțiuni despre Sistemele distribuite, Problema Generalilor Bizantini, informații tehnice precum Funcțiile Hash și Arborele Merkle, Structura unui bloc, Arhitectura blockchain-ului Ethereum. Un subcapitol important este dedicat metodelor de implementare a aplicațiilor decentralizate cu ajutorul contractelor inteligente (smart-contract) în care se prezintă limbajul de programare „Solidity” și metodele de compilare, executare a acestuia pe blockchain-ul Ethereum. De asemenea s-au pus în evidență framework-urile și librăriile necesare pentru dezvoltarea aplicațiilor decentralizate într-un mediu local cu sistem de operare Windows (Node.js, React.js, WEB3.js, Truffle, Ganache).

Capitolul 3 reprezintă un studiu de caz „Aplicație decentralizată pentru prescripții medicale electronice în rețeaua Ethereum” unde s-au prezentat în primă fază instalarea și configurarea instrumentelor necesare la dezvoltare. S-a detaliat modul de implementare a unui smart-contract în limbaj solidity, respectiv s-au descris metodele (funcțiile) ce le poate executa, împreună cu structurile de date utilizate. De asemenea s-a pus în evidență modul de compilare și executare a acestuia pe un blockchain local Ganache. S-au descris funcționalitățile a 3 aplicații web orientate către utilizator, implementate cu Node.js, React și WEB3 : Aplicația Medicului, Aplicația Pacientului, Aplicația Farmacistului.

În ultimul capitol s-au prezentat concluziile cu privire la gradul de finalizare al obiectivelor propuse inițial, cunoștințele noi însușite precum și avantajele și dezavantajele aplicațiilor decentralizate.