

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și tehnologia informației
SPECIALIZAREA: Tehnologia informației

Aplicație de investiții bazată pe Blockchain

LUCRARE DE DIPLOMĂ

Coordonator științific
Ş.l.dr.ing. George-Emil Vieriu

Absolvent
Veronica Bradu

Iași, 2020

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII LUCRĂRII DE DIPLOMĂ

Subsemnatul(a) BRADU VERONICA,
legitimat(ă) cu CI seria VS nr. 764906, CNP 2980725375473
autorul lucrării APLICAȚIE DE INVESTIȚII BAZATĂ PE BLOCKCHAIN

elaborată în vederea susținerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea IULIE a anului universitar 2019-2020, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 – Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data

15.07.2020

Semnătura



Cuprins

Introducere.....	1
1. Contextul proiectului	1
2. Motivația alegerii temei.....	2
3. Structura lucrării.....	2
Capitolul 1. Fundamentarea teoretică și documentarea bibliografică pentru tema propusă	3
1.1. Tehnologii utilizate pentru implementarea aplicației	3
1.1.1. Blockchain.....	3
1.1.1.1. Arhitectura.....	3
1.1.1.2. Procesul de minare	5
1.1.1.3. Mecanisme de consens	6
1.1.1.4. Caracteristicile unui blockchain	7
1.1.1.5. Semnătura digitală	8
1.1.1.6. Clasificarea sistemelor blockchain.....	8
1.1.1.7. Ethereum Blockchain	9
1.1.2. Smart Contracts	10
1.1.2.1. Modul de funcționare	11
1.1.2.2. Caracteristici.....	11
1.1.2.3. Truffle Suite	11
1.1.3. Limbajul Solidity.....	12
1.1.4. HTML, CSS și JavaScript	12
1.1.5. Bootstrap	13
1.1.6. Node.js.....	13
1.1.7. Web3.js.....	13
1.1.8. Electron	13
1.1.9. React.....	14
1.2. Referințe la teme/subiecte conceptual similare	15
1.2.1. Fundition	15
1.2.2. WeiFund	15
Capitolul 2. Proiectarea aplicației	16
2.1. Proiectarea componentei blockchain.....	16
2.2. Proiectarea aplicație client.....	16
2.2.1. Interfața grafică	17
2.2.2. Integrare componentei blockchain cu aplicația client	17
2.3. Diagrame UML	17
2.3.1. Diagrama state-machine	17
2.3.2. Diagrama de activitate.....	18
2.3.3. Diagrama user-case	19
2.3.4. Diagrama de secvență.....	19
2.3.5. Diagrama de clase	20
2.4. Avantajele și dezavantajele metodelor alese	22
Capitolul 3. Implementarea aplicației	23

3.1. Implementarea componentei blockchain	23
3.2. Implementarea aplicației client	25
3.3. Integrare componente blockchain cu aplicația client.....	26
3.4. Interfață cu utilizatorul.....	26
3.4.1. Înregistrarea/Autentificarea utilizatorului.....	26
3.4.2. Crearea unui proiect.....	28
3.4.3. Investirea într-un proiect.....	29
3.4.4. Adăugarea transferurilor pentru un proiect.....	30
3.5. Dificultățile întâmpinate	30
Capitolul 4. Testarea aplicației și rezultate experimentale	31
4.1. Testarea componente blockchain.....	31
4.2. Testarea aplicației client	32
4.3. Performanța aplicației	33
Concluzii	34
Bibliografie	35
Anexe	37
Anexa 1. Structura unui contract inteligent	37
Anexa 2. Interacțiunea cu blockchain-ul prin intermediul Web3.js	39

Aplicație de investiții bazată pe Blockchain

Veronica Bradu

Rezumat

Fundamentele teoretice ale proiectului dezvoltat oferă o soluție pentru multe din ariile cotidiene, domeniul fiind însă restrâns la sectorul creării și investirii în proiecte, într-o manieră transparentă, incoruptibilă și modernă. Tehnologiile utilizate sunt puternice, de actualitate și oferă o perspectivă largă de aplicabilitate și scalabilitate.

Regulile aplicației sunt definite în contractele inteligente, instrumente esențiale ce permit imutabilitatea operațiilor și un cost al tranzacțiilor scăzut. Interfața cu utilizatorul este prezentată sub forma unei aplicații desktop, în care fluiditatea proceselor și design-ul modern conferă un mediu plăcut de utilizare. Mecanismul de implementare a fost abstractizat, însă s-a pus accentul pe detaliile relevante pentru utilizator.

Componenta principală a proiectului a fost construită folosind principiile tehnologiei Blockchain, o tehnologie în curs de maturizare, însă cu un potențial ridicat. S-au folosit unelte puternice pentru crearea unei rețele locale private, care simulează toate operațiile ce pot fi executate pe rețeaua principală, însă fără a exista un cost. Interfața cu utilizatorul este definită în cadrul restrâns al aplicației desktop, deoarece focusul acestuia trebuie să fie strict direcționat asupra procesului de creare de proiecte sau investire, orice distrajere fiind eliminată. Cele două componente se îmbină perfect, cu ajutorul *API*-ului, ce facilitează interacțiunea cu rețeaua privată locală, și care a fost folosit în cadrul metodelor proiectate, ce permit prezentare informațiilor în interfața cu utilizatorul.

Obiectivele propuse inițial au suferit ușoare modificări deoarece procesul de dezvoltare nu se desfășoară într-o manieră limitată, ci flexibilă. Un utilizator se poate loga folosind o cheie privată validă, un portofel securizat sau își poate crea un cont, informațiile acestuia putând fi ulterior encriptate și salvate. Procesul de investire presupune transmiterea banilor din contul utilizatorului, în contul proiectului. Utilizatorul poate investi de nenumărate ori, însă în limita existenței banilor necesari pentru a se valida transferul. Banii acumulați din investiții nu pot fi folosiți în alte scopuri decât în cele care presupun buna derulare a activității descrise de proiect. Astfel, investitorii au puterea de a alege în ce direcție se cheltuiesc banii.

Introducere

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

1. Contextul proiectului

Evoluția exponențială a utilizării tehnologiei a reprezentat un pas extraordinar în definirea și dezvoltarea societății, în special, pe plan economic și social. Astfel, cu miliarde de oameni conectați la internet, tehnologia a ajuns să însemne un conglomerat de oportunități care pot fiexploataate intelligent pentru a facilita viața oamenilor.

Această împrejurare a introdus și un mediu în care oamenii își pot unifica ideile și resurselor în scopul creării unor produse și servicii. În acest sens, au fost create platforme pentru a aduce simple vizuini la stadiu realizabil, prin încurajarea persoanelor care au avut o idee, au expus-o, și au fost susținuți de alte persoane care au crezut în visul lor.

Platformele de investiții au reprezentat o formă foarte bună pentru a oferi finanțare, astfel în anul 2015, peste 34 de miliarde de dolari americanii au fost strânși la nivel mondial [1]. În ciuda impactului adus, acest tip de platformă a fost criticat ca fiind un sistem fraudulos și, conform rapoartelor de încredere, banii investiți au fost direcționați către alte campanii decât cele cărora li se adresaseră inițial, sau au fost blocați de entitatea care deținea platforma. O soluție pentru aceasta problemă este implementarea platformei bazate pe Blockchain¹. Prin această tehnologie, investitorii pot ști cu îi trimit banii și cum sunt cheltuiți. Acest lucru poate fi obținut prin folosirea unui contract intelligent², care implementează un set de reguli pentru blocarea fondurilor din Blockchain până la o anumită dată sau până este atins un obiectiv. În funcție de rezultat, fondurile vor fi date spre dezvoltarea proiectului sau vor fi restituite în siguranță către contribuabili. Mai mult decât atât, costurile de tranzacționare se micșorează considerabil, înlătura existența unui intermediar între investitor și fondatorul proiectului și oferă securitate.

Deși este încă la început, tehnologia blockchain poate fi aplicată oricărei industrii, iar marile corporații au început integrarea tehnologiei în produsele proprii [2], spre exemplu:

- **Amazon Web Services** s-a asociat cu **Kaleido** pentru a oferi o platformă de întreprindere blockchain *full-stack* pe *cloud* care integrează serviciile blockchain cu serviciile AWS³.
- **Google** dezvoltă un registru distribuit pentru a permite terților să trimită și să verifice tranzacțiile. De asemenea, intenționează să ofere un serviciu complet de blockchain pentru dezvoltatori, pe care companiile le pot utiliza pe propriile lor servere.
- **ZhongAn** de la **Alibaba** susține blockchain pentru a procesa și stoca datele despre pacienți într-o manieră sigură, pentru peste 100 de spitale din China.

¹ Blockchain, tehnologie care permite înregistrarea tranzacțiilor într-un mod incoruptibil.

² Contract intelligent (*smart contract*, în engleză) sau program pe calculator, destinat să faciliteze, verifice, sau să efectueze negocierea sau executarea unui contract.

³ AWS, serviciul web Amazon, este o platformă care oferă soluții flexibile, fiabile, scalabile, ușor de utilizat și rentabile.

- **Walmart, Kroger, Nestle și Unilever**, s-au asociat cu **IBM** pentru a utiliza blockchain pentru creșterea siguranței alimentelor, printr-o urmărire îmbunătățită a lanțului de aprovizionare.

2. Motivația alegerii temei

Motivația realizării acestui proiect este reprezentată de dorința de a oferi o soluție securizată și eficientă pentru procesul de investiții și finanțare a proiectelor. Tema aleasă va da posibilitatea realizării acestui obiectiv într-o manieră personal structurată și implementată, ținând cont de problemele actuale și conferind o rezolvare pentru eliminarea lor.

Proiectul va fi implementat folosind tehnologia Blockchain, caracterizată prin incoruptibilitate și mecanism rapid de transfer, securizată prin design, reprezentând un exemplu de sistem de calcul distribuit⁴ cu toleranță ridicată de tip bizantin⁵, atrăgătoare esențiale pentru tema propusă.

Protocolul electronic de investiții actual necesită un sistem centralizat care să controleze întreaga procedură de la modul de efectuare a tranzacțiilor la rezultatele și monitorizarea proiectelor. Între timp, tehnologia blockchain oferă un sistem descentralizat care se deschide pe întreaga rețea de participanți și elimină multe din problemele actuale. Aplicarea tehnologiei blockchain în protocolul de investiții electronice printr-o arhitectură adecvată poate insuflare caracteristici precum autenticitatea tranzacțiilor, transparența acestora, și folosirea resurselor colectate în scopul în care au fost formulate inițial.

3. Structura lucrării

În continuare, va fi prezentat conținutul lucrării. Structura acesteia se bazează pe o listă de obiective, personal formilate, în vederea documentării și rezolvării problemei propuse, prezentată în continuare:

- În *Introducere*, au fost descrise contextul și motivația care au constituit fundamentul realizării acestui proiect.
- În capitolul 1, *Fundamentarea teoretică și documentarea bibliografică pentru tema propusă*, vor fi descrise și analizate tehnologiile utilizate la proiectarea soluției, și se va face referință la aplicațiile similare conceptual cu proiectul propus.
- În capitolul 2, *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 3, *Implementare aplicației*, va fi prezentată într-o manieră exhaustivă implementarea aplicației. Mai mult decât atât, vor fi descrise și dificultățile întâmpinate și modalitatea de rezolvare a acestora.
- În capitolul 4, *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, atât din perspectiva incipientă, cât și concluzionată și se vor contura noi posibile funcționalități, ce pot fi implementate ulterior.

⁴ Un sistem distribuit este o colecție de calculatoare independente care apar utilizatorilor sistemului ca un singur calculator.

⁵ Toleranță de tip bizantin este toleranță la atacatori sau la calculatoare necooperante.

Capitolul 1. Fundamentarea teoretică și documentarea bibliografică pentru tema propusă

În acest capitol se vor prezenta concepte esențiale din sfera temei alese, precum: blockchain, contracte inteligente și semnătura digitală. De asemenea, se va face și o analiză a sistemelor similare cu proiectul propus.

1.1. Tehnologii utilizate pentru implementarea aplicației

1.1.1. Blockchain

Un blockchain este, în esență, o serie de înregistrări imutabile de date, în continuă creștere, numite blocuri, care este gestionată de un grup de computere, și nu deținută de o singură entitate. Fiecare dintre aceste blocuri de date este securizat și legat de celelalte folosind principii criptografice, formând un lanț [3]. Blockchain-ul conține informații publice despre toate tranzacțiile sau evenimentele digitale care au fost executate sau partajate între entitățile participante. Rețeaua blockchain nu are autoritate centrală - este însăși definiția unui sistem democratizat. Deoarece este un registru (*ledger*⁶, în engleză) partajat și imuabil, informațiile din acesta sunt deschise pentru toată lumea. Prin urmare, orice este construit pe blockchain este, prin natura sa, transparent, iar toți cei implicați sunt responsabili pentru acțiunile lor.

Primul blockchain a fost elaborat în anul 2008 de o persoană anonimă care s-a identificat cu numele de Satoshi Nakamoto [4]. De asemenea, în 2009, Nakamoto a dezvoltat criptomoneda⁷ Bitcoin⁸, folosită pentru tranzacțiile în rețea.

1.1.1.1. Arhitectura

Ca structură de date, un blockchain este o listă simplu înlănțuită, în care legăturile dintre blocuri se fac printr-un hash⁹.

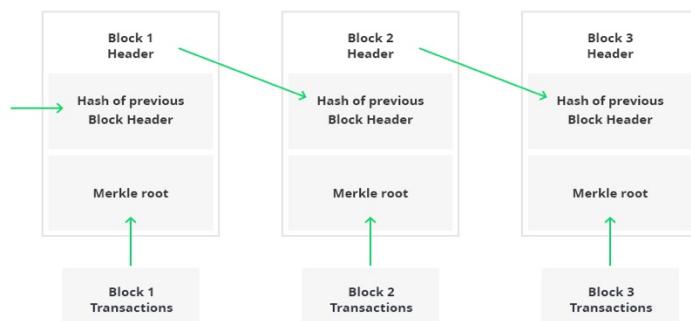


Figura 1.1. Exemplu de blockchain [5]

⁶ Ledger, asemănător cu o baza de date, unde tranzacțiile confirmate sunt înregistrate. Platformele care utilizează tehnologia blockchain nu folosesc o baza de date centralizată și fiecare nod deține o copie a ledger-ului.

⁷ Criptomoneda este un tip de monedă digitală, virtuală.

⁸ Bitcoin este un sistem de plată electronică descentralizat (controlul și puterea sunt distribuite între participanți) și o monedă digitală.

⁹ Un hash este secvența de numere, de lungime fixă, obținută prin aplicarea unei funcții hash (descrișă în acest capitol).

În Figura 1.1. este ilustrat un exemplu de blockchain, ce conține o secvență continuă de blocuri. Primul bloc se numește *genesis block*, bloc de geneză, referit și ca blocul 0. Fiecare bloc din lanț conține hash-ul blocului anterior, exceptând primul bloc, astfel legătura dintre ele este formată.

O funcție hash (sau funcție de dispersie) [6] este, în domeniul matematicii, o funcție definită pe o mulțime cu multe elemente (posibil infinită) cu valori într-o mulțime cu un număr fix. Reversibilitatea funcțiilor hash este exclusă.

O funcție hash poate lega două sau mai multe chei la aceeași valoare hash. Astfel, de cele mai multe ori, se dorește minimalizarea ratei de apariție a unor astfel de coliziuni, adică, funcția hash trebuie să lege cât mai uniform posibil cheile de valorile. Conceptualizarea acestei idei a luat naștere încă din anii 1950, însă proiectarea optimă a funcțiilor hash este încă un subiect activ de dezbatere și cercetare. Funcțiile hash sunt utilizate și ca funcții hash criptografice sau sume de control, însă nu trebuie confundate cu caracterele de verificare, amprentele numerice, funcțiile de randomizare, coduri de corectare a erorilor. Figura 1.2. ilustrează mecanismul de conversie a unei valori de lungime arbitrară într-o valoare hash de lungime fixă.

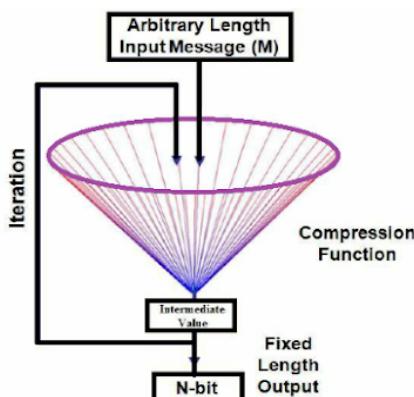


Figura 1.2. Funcția de dispersie (funcția hash) [7]

Hash-ul fiecărui bloc este partajat cu blocul următor din lanț, astfel, orice încercare de a modifica informațiile unui bloc va produce schimbarea valorii hash, care nu va mai coincide cu valoarea hash salvată în blocul următor, și astfel se poate detecta un posibil atac fraudulos.

Un bloc este format din mărimea blocului, antet (sau *header*) și corp (sau *body*) [8].

Antetul unui bloc reprezintă metadatele unui bloc, ocupând în jur de 80 bytes, și este format din următoarele elemente:

- versiunea blocului: valoare pe 4 bytes, precizează setul de reguli de validare necesare blocului curent; frecvent, este definită de structura internă a datelor din bloc [9].
- valoarea hash a blocului anterior: valoare hash de 32 bytes.
- timestamp: în secunde, valoare pe 4 bytes, variabilă indispensabilă mecanismului de urmărire a creației sau a modificărilor aduse unui bloc. Aceasta descrie timpul la care s-a înregistrat un eveniment, în stilul Unix, momentul initial de referință fiind 1 ianuarie 1970.
- *nBits* sau pragul țintă al dificultății curente [10]: valoarea pe 4 bytes sub care trebuie să fie hash-ul, astfel încât blocul să fie valid (acceptat de rețea).

- rădăcina arborelui Merkle¹⁰: rezultatul unui arbore Merkle; constituie un mod eficient de a garanta că toate tranzacțiile incluse în sarcina utilă sunt incluse în bloc. Deoarece orice utilizator poate însuși să obțină rădăcina Merkle din toate tranzacțiile, în cazul în care această rădăcină Merkle este egală cu cea din antetul blocului, utilizatorul poate fi sigur că toate tranzacțiile sunt corecte [11].
- nonce [12]: un număr ales în mod întâmplător, unic, întreg, format din 4 bytes și reprezintă o parte centrală a algoritmului *Proof of work*, detaliat în capitolul următor, pentru blockchain-uri și criptomonede. Minerii concurează între ei pentru a găsi un nonce care produce un hash cu o valoare mai mică sau egală cu cea stabilită de dificultatea rețelei. Aceștia testează și resping milioane de nonce-uri în fiecare secundă. Dacă un miner găsește un astfel de nonce, numit „*nonce de aur*”, atunci câștigă dreptul de a adăuga acel bloc în blockchain și de a primi recompensa blocului. În prezent, nu există nicio modalitate de a accelera procesul de a găsi nonce-ul corect. Astfel, minerii operează doar prin încercare și eroare, până când găsesc un „*nonce de aur*”.

Corpul blocului conține toate tranzacțiile confirmate în bloc, și un contor de tranzacții. Când un miner construiește un bloc, validează tranzacțiile. Adică verifică dacă expeditorul are de fapt suficienți bani pentru a cheltui. El poate citi cu ușurință aceste informații din blockchain. Numărul maxim de tranzacții conținute într-un bloc depinde atât de dimensiunea blocului aflat în discuție, cât și de dimensiunea maximă a fiecărei tranzacții [13].

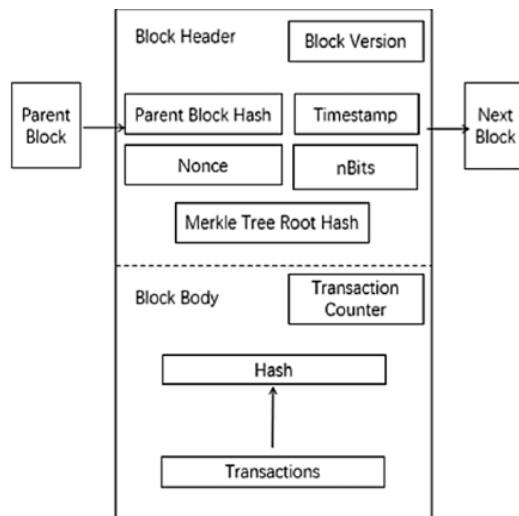


Figura 1.3. Structura unui bloc [14]

1.1.1.2. Procesul de minare

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 reusește să adauge noul bloc primește o recompensă pentru timpul și resursele implicate în proces. În medie, un bloc este minat la fiecare 10 minute [15].

¹⁰ Arborele Merkle, sau arbore binar, este o structură de date care este utilizată pentru a rezuma și a verifica integritatea seturilor mari de date.

Obiectivul [16] este de a găsi o valoare pentru *nonce* (definit în capitolul *Arhitectura*), care va duce la valoarea hash-ului mai mică decât *pragul ţintă al dificultății curente* (*nBits*). Deci, nodul de exploatare va încerca milioane sau miliarde de valori nonce înainte de a obține un hash valid.

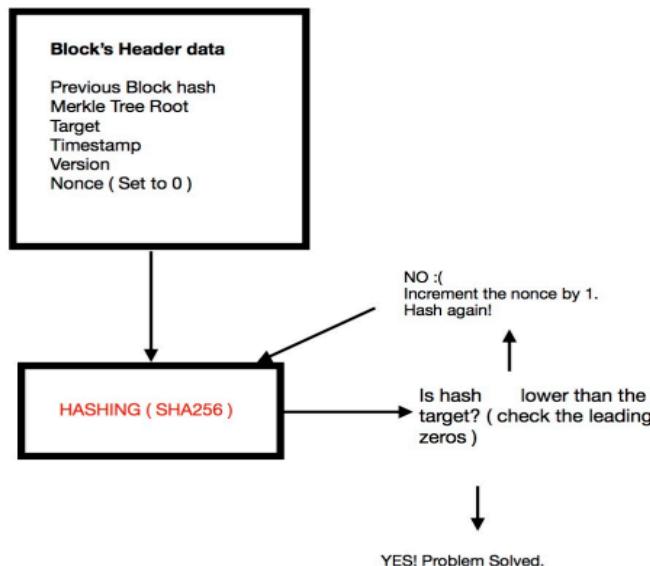


Figura 1.4. Procesul de minare [17]

1.1.1.3. Mecanisme de consens

Un mecanism de consens este un mecanism tolerant la erori, care este utilizat în sistemele de calculatoare și blockchain pentru a obține acordul necesar asupra unei valori de date unice sau a unei singure stări a rețelei între procesele distribuite sau sistemele cu *multi-agent*, cum ar fi cu criptomonede. Este util în monitorizarea înregistrărilor, printre altele lucruri. În orice sistem centralizat, cum ar fi o bază de date care deține informații cheie despre permisele de conducere într-o țară, un administrator central are autoritatea de a menține și actualiza baza de date. Sarcina de a face orice operație- cum ar fi adăugarea, ștergerea sau actualizarea numelor persoanelor care s-au calificat pentru anumite licențe, este îndeplinită de o autoritate centrală care rămâne singura responsabilă cu păstrarea înregistrărilor autentice.

Lanțul de blocuri publice care funcționează ca sisteme descentralizate, de auto-reglare, funcționează la scară globală, fără nicio autoritate unică. Acestea implică contribuții de la sute de mii de participanți care lucrează la verificarea și autentificarea tranzacțiilor care au loc pe blockchain și la activitățile de minare a blocurilor. Într-o astfel de stare de schimbare dinamică a blockchain-ului, acești ledgeri publici distribuiți au nevoie de un mecanism eficient, corect, în timp real, funcțional, fiabil și sigur pentru a se asigura că toate tranzacțiile care apar pe rețea sunt autentice și toți participanții stabilesc un consens privind starea ledger-ului. Această sarcină importantă este realizată de mecanismul consensului, care este un set de reguli care decid asupra contribuților de la diversi participanți la blockchain. Există diferite tipuri de algoritmi de mecanism de consens, care funcționează pe principii diferite:

- **Dovada muncii, Proof of Work (POW)**, este un algoritm utilizat de cele mai populare rețele de criptomonede, precum bitcoin și litecoin. Este necesar ca un nod participant să demonstreze că munca depusă și înregistrată de aceștia îi califică să primească dreptul de a adăuga noi tranzacții la blockchain. De asemenea, acest proces are nevoie de un timp mai lung de procesare, determină un consum uriaș de energie pentru mineri și încurajează

utilizarea mining pool-urilor¹¹, care transformă blockchain-ul într-un sistem centralizat, contrazicând fundamental descentralizat. Mai mult decât atât, mărimea acestor grupări poate fi într-un raport majoritar, acestia putând aproba tranzacții frauduloase foarte usor. Cu alte cuvinte, dacă un singur miner sau un grup de mineri poate obține 51% din puterea computatională, acestia pot controla blockchain-ul. În scopul rezolvării acestei probleme, a fost propusă o tehnică numită *Proof of stake (POS)*, sau Dovada mizei.

- **Dovada mizei**, este un alt algoritm care a evoluat ca o alternativă, cu consum redus de energie și costuri, la algoritmul *POW*. Aceasta implică un proces de alegere a unui nod, în mod aleatoriu, pentru a valida blocul următor. Pentru a deveni validator, un nod trebuie să depoziteze o anumită cantitate de monede în rețea. Mărimea depozitului determină posibilitatea unui nod de a crea următorul bloc. Validatorul își va pierde o parte din depozit în cazul în care acesta aprobă o tranzacție frauduloasă. Atâtă timp cât miza pusă în joc este mai mare decât taxa tranzacției, pe care ar urma să o primească, validatorul poate reprezenta o entitate de încredere.

1.1.1.4. Caracteristicile unui blockchain

Caracteristicile de bază ale unui blockchain sunt prezentate în lista de mai jos [18]:

- **Persistență**, 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, permitând astfel tranzacțiile sigure. Cei care validează tranzacțiile și le adaugă în bloc se numesc mineri.
- **Transparentă**, fiind un registru deschis. Ledger-ul reprezintă înregistrarea tranzacțiilor efectuate și este vizibilă pentru toată lumea, de aceea se numește evidență sau registru deschis. Nicio persoană sau o organizație nu se încarcă cu tranzacții. Fiecare nod din rețeaua blockchain are aceeași copie a înregistrării.
- **Descentralizare**, adică nu are autoritate centrală pentru a controla rețeaua, aşa cum există în modelul client-server. Blockchain oferă o rețea *peer-to-peer*¹². Această caracteristică a blockchain permite tranzacțiilor să implice doar două părți, expeditorul și receptorul. Astfel, se elimină cerința de entitate centrală de încredere (de exemplu, banca monetară), deoarece toată lumea din rețea este în măsură să autorizeze tranzacțiile mult mai rapid și la un cost scăzut.
- **Anonimat**. Datele confidențiale ale utilizatorului nu sunt necesare pentru a interacționa cu blockchain-ul, o singură adresă fiind suficientă.

¹¹Mining pool, sau grup de mineri de criptomonede care își unesc resursele computaționale, într-o rețea, pentru a maximiza probabilitatea de a adăuga un bloc în lanț.

¹²Peer-to-peer (P2P), se traducere din limba engleză prin de la egal la egal, este o arhitectură de rețea pentru aplicațiile distribuite care împarte sarcinile la mai mulți parteneri.

1.1.1.5. Semnătura digitală

Semnătura digitală [19] este o schemă matematică pentru demonstrarea autenticității mesajelor sau documentelor digitale. O semnătură digitală validă oferă unui destinatar un motiv de a crede că mesajul a fost creat de către un expeditor cunoscut (autentificare), că expeditorul nu poate nega că a trimis mesajul (non-repudiere) și că mesajul nu a fost modificat în tranzit (integritate).

Semnătura digitală se bazează pe criptografia¹³ asimetrică. Criptografia asimetrică se referă la un tip de criptografie prin care cheia care este utilizată pentru criptarea datelor este diferită de cea care este folosită pentru a decripta datele. Cunoscută și sub denumirea de criptografie cu chei publice, utilizează chei publice și private pentru a cripta și a decripta date. Cheia privată este folosită pentru semnarea tranzacțiilor și este confidențială. Tranzacțiile sunt semnate și distribuite în rețea pentru a fi verificate.

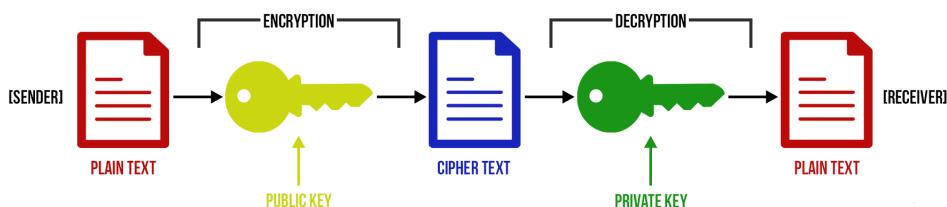


Figura 1.5. Criptarea asimetrică [20]

Spre exemplu, în criptografia cu cheie publică [21], Alice și Bob nu doar că au chei diferite, ci fiecare are două chei. O cheie este privată și nu trebuie partajată cu nimeni. Celalată cheie este publică și poate fi partajată cu oricine. Atunci când Alice dorește să-i trimită un mesaj sigur lui Bob, ea criptează mesajul folosind cheia publică a lui Bob și îi trimit rezultatul. Bob își folosește cheia privată pentru a decripta mesajul. Atunci când Bob dorește să-i trimită un mesaj sigur lui Alice, el criptează mesajul folosind cheia publică a lui Alice și îi trimit rezultatul. Alice își folosește cheia privată pentru a decripta mesajul. Eva poate intercepta ambele chei publice și mesajele criptate, dar nu poate decripta mesajele deoarece nu are nici una dintre cheile private.

1.1.1.6. Clasificarea sistemelor blockchain

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

1) Blockchain public

Acestea permit oricărei persoane să participe ca utilizator, miner, dezvoltator sau membru al comunității. Toate tranzacțiile care au loc pe blockchain-uri publice sunt complet transparente, ceea ce înseamnă că oricine poate examina detaliile tranzacției.

- 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.

¹³ Criptografia, ramură a matematicii care se ocupă cu securizarea informației.

- 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 token¹⁴ asociat cu acestea, care este de obicei conceput pentru a stimula și recompensa participanții din rețea.

2) **Blockchain privat**

Cunoscute și sub numele de blockchain-uri autorizate, dețin o serie de diferențe notabile față de blockchain-urile publice:

- Participanții au nevoie de consimțământ pentru a se alătura rețelelor
- Tranzacțiile sunt private și sunt disponibile numai participanților la ecosistem cărora li s-a acordat permisiunea de a se alătura rețelei
- Blockchain-urile private sunt mai centralizate decât blockchain-urile publice

Blockchain-urile private sunt valoroase pentru întreprinderile care doresc să colaboreze și să partajeze date, dar nu doresc ca datele lor comerciale sensibile să fie vizibile pe un blockchain public. Aceste lanțuri, prin natura lor, sunt mai centralizate; entitățile care conduc lanțul au un control semnificativ asupra participanților și a structurilor de guvernare. Blockchain-urile private pot sau nu să aibă un token implicat în lanț.

3) **Consortiu / Blockchain hibrid**

Blockchain-urile din consorțiu sunt uneori considerate un caz particular al blockchain-urilor private. Principala diferență între ele este că blockchain-urile consorțialui sunt guvernate de un grup și nu de o singură entitate. Această abordare are toate avantajele unei blockchain private și ar putea fi considerată o subcategorie a blockchain-urilor private.

- Acest model de colaborare oferă unele dintre cele mai bune cazuri de utilizare pentru beneficiile blockchain-ului, reunind un grup de întreprinderi care lucrează împreună, dar și concurează între ele.
- Aceștia sunt capabili să fie mai eficienți, atât individual, cât și colectiv, prin colaborarea la anumite aspecte ale activității lor.
- Participanții la blockchain-urile consorțialui ar putea include pe oricine de la băncile centrale, la guverne, să furnizeze lanțuri de aprovizionare.

1.1.1.7. Ethereum Blockchain

Ethereum [22] este a doua cea mai mare platformă de criptomonede, prin capitalizarea pieței, imediat după Bitcoin. Este un blockchain *open-source* descentralizat, cu funcționalitate inteligentă a contractului.

Componentele speciale ale platformei

- **Mașină virtuală Ethereum (EVM)**: Mașina virtuală Ethereum este platforma de calcul descentralizată care constituie nucleul platformei Ethereum.
- **Contract intelligent**: o bucată de cod persistentă pe blockchain-ul Ethereum care are un set de date și funcții executabile. Aceste funcții se execută atunci când tranzacțiile sunt

¹⁴ Token, unitate de valoare stabilită de o organizație privată care utilizează sistemul blockchain.

- efectuate cu anumiți parametri de intrare. Pe baza parametrilor de intrare, funcțiile vor executa și interacționa cu datele din contract și din afara acestuia.
- **Ether:** Ether este numele monedei utilizate în cadrul rețelei Ethereum. Minerii sunt răsplătiți cu eteri pentru furnizarea puterii de calcul pentru găsirea unui bloc nou și confirmarea unei noi tranzacții a contractelor inteligente. Eterul este utilizat pentru a plăti calculele în cadrul EVM.
 - **Gaz:** se referă la valoarea (prețul) necesară pentru a efectua cu succes o tranzacție sau a executa un contract intelligent pe platforma Ethereum Blockchain. Este, de asemenea, numele pentru cripto-combustibilul care este consumat atunci când codul este executat de EVM. Gazul este plătit cu titlu de taxă de execuție pentru fiecare operațiune efectuată pe un Blockchain Ethereum.
 - **Limita de gaz:** limita de gaz reprezintă cantitatea maximă de gaz pe care un utilizator este dispus să o plătească pentru o execuție intelligentă a tranzacției contractuale.

1.1.2. Smart Contracts

Smart Contract, este un termen folosit pentru a descrie un protocol special destinat să implementeze logica de negociere și performanță a unui contract, utilizând tehnologia blockchain. Acesta conține toate detaliile despre clauzele contractului și execută automat toate acțiunile definite inițial, fiind o substituție modernă a contractelor clasice și având o sferă largă de aplicabilitate. Ele permit efectuarea tranzacțiilor credibile, între două sau mai multe entități, fără a mai fi necesară validarea tranzacțiilor de către un intermediar (sau parte terță). Aceste tranzacții sunt ireversibile și monitorizate [23].

În 1994, criptograful american Nick Szabo a publicat un articol în care a conturat conceptul de contracte inteligente, definite ca reguli de tranzacție care pot fi citite de mașinile de calcul și care creează un contract cu termeni prestabiliri [24].

Principiul de bază poate fi comparat cu cel al automatelor, acestea executând doar instrucțiunile primite. La început, bunurile și termenii contractului sunt programati și introduși în blockchain. Mai apoi, acesta este distribuit și copiat de mai multe ori, între nodurile platformei.

Odată cu apariția unui eveniment, contractual este executat, se verifică termenii contractului și o tranzacție este efectuată sau nu. Diferența dintre interacțiunea cu un contract intelligent și interacțiunea cu un RESTful API¹⁵, este că nu există un singur server care efectuează acțiunile solicitate. În schimb, este o rețea de calculatoare, toate verificându-se reciproc, asigurându-se că niciun nod nu încearcă să fraudeze sistemul.



Figura 1.6. Smart Contract [25]

¹⁵ RESTful API este o interfață de programare a aplicațiilor (API), care folosește cereri HTTP pentru a insera, modifica, returna și a șterge date.

1.1.2.1. Modul de funcționare

Contractele inteligente Ethereum nu sunt compatibile cu limbajele de programare convenționale. În schimb, limbajele specifice contractului intelligent, cum ar fi Solidity și Vyper, pot fi utilizate pentru a defini o funcție de contract logic. Un compilator este apoi utilizat pentru a converti această funcție de contract logic în bytecode și a le distribui pe blockchain.

Când o funcție de contract logic este transformată în bytecode, transferă și ceva numit *Application Binary Interface* (ABI) [26]. O parte interesantă care intenționează să utilizeze contractul nu interacționează direct cu computerul. Întrucât interacționează folosind dispozitive de intrare/ieșire, cum ar fi tastaturi, mouse-uri și monitoare, acestea trebuie construite în interfețe prietenoase cu oamenii.

Pentru a interacționa cu contractul intelligent, biblioteca *Web3.js* este furnizată, care ajută funcțiile contractului intelligent să fie utilizate în browserele web. Dacă se dorește utilizarea acestor funcții în Python, există și *Web3.py*, iar pentru mediile mobile, se pot utiliza implementări Swift și Kotlin.

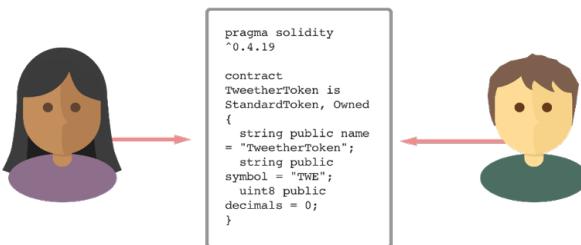


Figura 1.7. Smart Contract scris în limbajul Solidity

1.1.2.2. Caracteristici

Înainte de a analiza utilitatea contractelor inteligente, vom rezuma punctual caracteristicile lor. Din perspectivă tehnologică, contractele inteligente au următoarele caracteristici:

- Oricine poate distribui contractele.
- Oricine poate verifica contractele - nu doar proprietarul.
- Executarea codului poate fi automatizată.
- Este dificil de falsificat/modificat.

Astfel, contractele inteligente sunt cele mai potrivite pentru situațiile în care transparența este o cerință fundamentală, când intermediarii doresc a fi înălțurați și unde sunt suportate costurile tranzacției la un preț scăzut.

1.1.2.3. Truffle Suite

Truffle Suite [27] 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 Suite este formată din 3 componente, dedicate mecanismului de dezvoltare pe platforma Ethereum, și anume:

- **Truffle**, cel mai popular framework pentru dezvoltarea contractelor inteligente. Acesta facilitează management-ul unui contract intelligent, având facilitatea de a compila, migra și testa contracte în rețea.
- **Drizzle**, colecția de biblioteci front-end, care facilitează dezvoltarea UI a Dapps.¹⁶
- **Ganache**, blockchain-ul Ethereum local (acționează ca un simulator), scris în Javascript și distribuit ca un pachet Node.js prin intermediul managerului de pachete npm, care poate fi configurat în funcție de cerințe. Aceasta crează pentru configurația standard 10 conturi virtuale, fiecare având 100 etheri alocati. Spre deosebire de versiunea desktop a aplicației, unde există limitări, în special în privința numărului de conturi, care trebuie să fie cel mult egal cu 100, versiunea CLI¹⁷ permite o configurare flexibilă.

Avantaje:

- Existența unui singur nod în rețeaua Ethereum
- Nu trebuie setat niciun miner deoarece toate tranzacțiile trimise clientului ganache vor fi minate instant (timpul necesar minării unui bloc este de 0 secunde)
- Nodul este local pe computer-ul dezvoltatorului și nu afectează rețeaua principală de Ethereum

1.1.3. Limbajul Solidity

Solidity [28] 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 limbajul de programare complexe definite de utilizator. Se poate utiliza pentru a crea contracte pentru utilizări precum votare, crowdfunding, licitații și portofele cu mai multe semnături.

1.1.4. HTML, CSS și JavaScript

HTML oferă structura de bază a site-urilor, care este îmbunătățită și modificată de alte tehnologii precum CSS (controlă prezenta, formatarea și aspectul) și JavaScript (folosit pentru a controla comportamentul diferitelor elemente). HTML (HyperText Markup Language), este un limbaj de marcăre care folosește etichete pentru a identifica diferite tipuri de conținut și scopurile pe care le servesc fiecare pentru pagina web.

Cascading Style Sheets (CSS) este un limbaj „al foilor de stil” utilizat pentru a descrie prezenta unui document scris în HTML sau XML (incluzând dialecte XML precum SVG, MathML sau XHTML). CSS descrie modul în care elementele trebuie redate pe ecran sau pe alte suporturi.

JavaScript (sau JS) este un limbaj de programare folosit cel mai adesea pentru scripturile client dinamice pe paginile web, dar este de asemenea folosit adesea pe server, folosind un mediu de rulare, cum ar fi Node.js.

¹⁶ Dapp, aplicație descentralizată, aplicație computerizată care rulează pe un sistem de calcul distribuit.

¹⁷ Command line interface.

1.1.5. Bootstrap

Bootstrap este un *framework open-source* pentru CSS pentru construirea aplicațiilor *responsive*, cu finisaj modern. Este ușor de integrat și este extrem de flexibil, având o colecție impresionantă de şabloane gratuite și profesionale.

1.1.6. Node.js

Node.js este o platformă proiectată pe runtime-ul JavaScript din Chrome pentru a construi cu ușurință aplicații de rețea rapide și scalabile. Node.js folosește un model neblocant de I/O, bazat pe evenimente, care îl face ușor și eficient, perfect pentru aplicațiile în timp real, cu flux intensiv de date, care rulează pe dispozitive distribuite.

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 următoarul *API*, și folosește un mecanism de notificare a evenimentelor Node.js, care ajută serverul să obțină un răspuns din apelul API anterior.
- Foarte rapid - Fiind construit pe motorul JavaScript V8 al Google Chrome, biblioteca Node.js este foarte rapidă în executarea codului.
- *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.
- Fără *buffering* - aplicațiile Node.js nu „tamponează” niciodată date. Aceste aplicații pur și simplu emit datele în bucăți.

1.1.7. Web3.js

Ethereum JavaScript API, adică o colecție de librării care permit interacțiunea cu un nod ethereum local sau remote, utilizând o conexiune HTTP [29]. Cele mai utilizate funcții din Web3.js sunt umătoarele [30]:

- `web3.eth.getAccounts()` – obținerea tuturor adreselor (conturilor) valabile.
- `web3.eth.sendTransaction()` – folosită pentru a trimite etheri, sau altă monedă validă platformei, dintr-un cont în altul, sau unui contract.
- `web3.eth.estimateGas()` – estimează gas-ul consumat de tranzacția curentă.
- `web3.eth.contract()` – încarcă contractul în aplicația descentralizată.
- `web.utils.toWei()` – convertește etherii în wei (unitate de valoare folosită de contractele inteligente).

1.1.8. Electron

Electron este un framework *open-source*, care permite dezvoltatorilor să utilizeze tehnologii web pentru a construi aplicații desktop. La baza sa, Electron este format din trei componente. Înglobează biblioteca de redare Chromium (cunoscută sub numele de libchromiumcontent), care este fundamentalul *open-source* pentru browserul Google Chrome. Cu

suport pentru cele mai noi standarde web, un motor JavaScript performant și îmbunătățirea constantă a instrumentelor pentru dezvoltatori. Este o concepție greșită comună că Electron include Chrome în întregime; pentru a păstra Electron cât mai ușor posibil, acesta conține biblioteca de redare Chromium.

A doua componentă este Node.js, popularul mediu de rulare JavaScript construit pe V8 (același motor JavaScript care alimentează libchromiumcontent și Chrome). Acesta aduce un ecosistem *open-source* vibrant: nenumărate module de pe npm oferă soluții testate pentru sarcini comune de dezvoltare și integrare ușoară cu aproape orice serviciu.

A treia componentă este un strat gros de C++, extinzând setul obișnuit de API-uri disponibile cu implementări native pentru operațiuni comune ale sistemului de operare, cum ar fi trimitera notificărilor native, accesarea preferințelor sistemului și crearea de dialoguri native. În plus, Node.js permite utilizarea modulelor native - componente native scrise în C, C++, Rust sau Objective-C - pentru a se asigura că dezvoltatorul nu este niciodată limitat la JavaScript.

Combinând Chromium și Node.js și aruncând un număr mare de API-uri implementate nativ, Electron permite dezvoltatorilor să construiască aplicații desktop puternice, cu puțin efort. Toate cele trei tehnologii sunt disponibile pe multiple platforme, care acceptă Windows, macOS și Linux. Dezvoltatorii cu experiență în realizarea de software pentru desktop sunt deseori surprinși de cât de mici sunt echipele din spatele aplicațiilor Electron populare: Codul Visual Studio, produs de Microsoft, a fost inițial construit de doar șapte ingineri, aplicația desktop a lui Slack cu doar doi și aplicația pentru desktop a lui GitHub cu patru.

1.1.9. React

React este o bibliotecă JavaScript dezvoltată pentru crearea de interfețe de utilizare rapide și interactive pentru aplicații web și mobile. Este o bibliotecă front-end, bazată pe componente, responsabilă numai de aspectul vizual al aplicației. În arhitectura *Model View Controller* (MVC), stratul de vizualizare este responsabil de modul în care arată și se simte aplicația. React a fost creat de Jordan Walke, un inginer software la Facebook [31].

Caracteristici:

- JSX este o extensie de sintaxă a JavaScript. Se folosește cu React pentru a descrie cum ar trebui să arate interfața utilizator. Folosind JSX, putem scrie structuri HTML în același fișier care conține cod JavaScript. Acest lucru face ca codul să fie mai ușor de înțeles și depanat, deoarece evită utilizarea structurilor complexe JavaScript DOM¹⁸.
- React păstrează în memorie o reprezentare ușoară a DOM-ului „real” și care este cunoscut sub numele de „virtual” DOM (VDOM). Manipularea DOM-ului real este mult mai lentă decât manipularea VDOM, deoarece nimic nu este afișat pe ecran. Când starea unui obiect se schimbă, VDOM schimbă doar acel obiect din DOM real, în loc să actualizeze toate obiectele. Se compară apoi starea anterioară și apoi se actualizează doar acele obiecte din DOM real, în loc să actualizeze toate obiectele. Acest lucru face ca lucrurile să se miște rapid, în special în comparație cu alte tehnologii front-end care trebuie să actualizeze fiecare obiect, chiar dacă doar un singur obiect se schimbă în aplicația web.

¹⁸ DOM (Document Object Model) tratează un document XML sau HTML ca o structură arbore în care fiecare nod este un obiect reprezentând o parte a documentului.

1.2. Referințe la teme/subiecte conceptual similare

Crowdfunding-ul, cu caracterul descentralizat al fondatorilor, proiectelor și finanțatorilor, oferă o oportunitate unică pentru blockchain de a oferi beneficii unui set larg de utilizatori. Deși în mod tradițional nu este vorba despre o zonă discutată în cercurile de dezvoltare a blockchain-ului, oportunitățile legate de utilizarea blockchain-ului în crowdfunding, inclusiv ajutorarea antreprenorilor care mențin controlul asupra proiectelor și creșterea transparenței care conectează fondurile la rezultatele proiectului, sunt semnificative și merită cu siguranță analize suplimentare.

Platformele celebre de crowdfunding precum GoFundMe, Kickstarter sau Patreon ar trebui să își regândescă arhitectura și strategia de business deoarece au fost dezvoltate platforme care s-au adaptat noilor cerințe tehnologice și care propun un cost redus al tranzacțiilor și o transparență crescută.

1.2.1. Fundition

Fundition [32] este o platformă de ultimă generație, de crowdfunding de la egal la egal, descentralizată, construită pe blockchain-urile Steem & Tron. Scopul acesteia este de a înlocui vechile modele centralizate (cum ar fi Kickstarter, Indiegogo, Patreon sau GoFundMe) și să ofere o modalitate pentru persoane și organizații, fie să strângă fonduri, fie să creeze proiecte semnificative. Platforma Fundition oferă utilizatorilor numeroase avantaje, inclusiv accesul la marea bază de utilizatori de aplicații Steem & Tron, care crește exponențial în fiecare lună, și upvote bazate pe merite, în cadrul aceluia sistem, care se traduce în monedă reală. Susținătorii sunt, de asemenea, răsplătiți pentru aderare, iar gamificarea¹⁹ din cadrul platformei oferă interacțiune și un flux de venituri pentru toate părțile implicate (oricât de mic).

Una dintre caracteristicile esențiale este de a permite sponsorilor să doneze în așa fel încât fondurile să fie eliberate pe toată durata de viață a unui proiect.

1.2.2. WeiFund

WeiFund [33] este o platformă *open-source* ce conține un set de instrumente pentru derularea campaniilor de crowdfunding pe blockchain-ul Ethereum. Este deosebit de util pentru lansarea de tokeni care pot fi utilizate în aplicații și protocoale noi. Contractele WeiFund implementează mecanica de bază a crowdfundingului - contribuții cu rambursări în cazul în care obiectivul campaniei nu este îndeplinit - cu *hook-uri* personalizabile pentru emiterea de tokeni către contribuabili, după cum este necesar. Aplicația web WeiFund ușurează începerea acceptării contribuților și are *hook-uri* care permit reutilizarea interfaței utilizatorului cu contracte personalizate proprii, dacă este necesar. Cu un accent puternic pe securitatea și cele mai bune practici ale contractelor inteligente, WeiFund ușurează lansarea unei campanii sigure care funcționează în interfețele de utilizator.

¹⁹ Gamificare, definită ca un set de activități și procese de rezolvare a problemelor prin utilizarea sau aplicarea caracteristicilor elementelor de joc.

Capitolul 2. Proiectarea aplicației

Aplicația este formată din două componente: aplicația client desktop Electron și componenta blockchain. Comunicarea celor două componente s-a facut prin intermediul unui Ethereum Javascript API, întreg mecanismul fiind descris în capitolele următoare.

Aplicația a fost dezvoltată pe un computer cu sistemul de operare *macOS Catalina*, cu un procesor quad-core *Intel Core i5*, și poate fi folosită pe majoritatea sistemelor de calcul care dispun de o arhitectură modernă și un sistem de operare performant. Cele două componente fundamentale ale aplicației vor fi analizate detaliat în capitolul următor.

2.1. Proiectarea componentei blockchain

Componenta blockchain a aplicației a fost dezvoltată folosind limbajul Solidity și cuprinde regulile unui mecanism de finanțare automatizat și securizat, incluse în contractele inteligente. Principalele funcționalități ale sistemului sunt descrise la nivelul acestor contracte, și anume:

- Crearea unui proiect nou sau unui cont de utilizator nou, dar și vizualizarea proiectelor existente, sau a detaliilor contului curent.
- Investirea într-un proiect. Un cont poate investi orice sună și de un număr nelimitat de ori, însă în limita existenței banilor necesari pentru a se valida transferul.
- Inițializarea transferului de către proprietarul contului care dorește să investească.
- Votarea pozitivă sau negativă a acțiunilor pe care creatorul proiectului dorește să le întreprindă cu banii acumulați. Aceste acțiuni pot fi aprobată doar dacă un procent majoritar de investitori votează pozitiv.

2.2. Proiectarea aplicație client

Componenta client a fost dezvoltată ca o interfață de prezentare facilă interacțiunii cu orice utilizator, accentul fiind pus pe fluența mecanismului și designul modern, dar simplist. Principalele funcționalități ale aplicației sunt:

- Autentificarea utilizatorului folosind o cheie, autentificarea prin încărcarea unui fișier *wallet*, care conține datele securizate, și o parolă, sau crearea unui nou cont și salvarea ulterioară a datelor encriptate.
- Vizualizarea proiectelor înregistrate pe blockchain și a detaliilor fiecărui proiect. Maniera de prezentare a acestora este definită sub forma componentei *responsive* și reutilizabile de tip *Card*.
- Crearea proiectelor și a acțiunilor pe care proprietarul unui proiect le dorește a fi făcute cu banii de la investitori.
- Investirea într-un proiect, vizualizarea investițiilor facute și a istoricului evenimentelor din fiecare investiție.
- Vizualizarea detaliilor contului și mecanismul de *logout*

2.2.1. Interfața grafică

Pentru interfața grafică a fost folosit *framework*-ul Electron, care permite construirea aplicațiilor desktop multi-platformă, folosind pe partea de backend Node.js și Chromium pe front-end. S-a urmărit simplitatea și claritatea suprafeței de control, rapiditatea dezvoltării și flexibilitatea răspunsului la evenimente utilizatorului, astfel framework-ului React respectă toate criteriile formulate inițial.

React oferă o gamă largă de componente reutilizabile și randarea acestora este rapidă, astfel orice schimbare este fluid afișată, atribute esențiale atât pentru dezvoltatori, cât și pentru utilizatori.

2.2.2. Integrare componenței blockchain cu aplicația client

Pentru a integra cele două componente principale ale aplicației se folosește librăria Web3.js, care simplifică mecanismele contractelor inteligente, reducându-le comportamentul la simple obiecte JavaScript.

Conecțarea cu blockchain-ul se poate face odată ce am definit un *provider*, entitate care comunică cu node-ul Ethereum și transferă date, din cod mașină în comenzi Web3.

2.3. Diagrame UML

UML (Unified Modeling Language) este un limbaj de modelare ce descrie vizual reprezentarea structurii unei aplicații, arhitectura sau comportamentul unui sistem, modelarea structurilor de date sau a proceselor de *business logic*, sau construirea unei specificații detaliate pentru un sistem. O diagramă reprezintă prezentarea grafică a elementelor și a relațiilor descrise într-un sistem.

2.3.1. Diagrama state-machine

Diagrama *state-machine* descrie diferențele stării unei entități. De asemenea, aceasta modelează natura dinamică a sistemului și poate arăta modul prin care o entitate răspunde unei varietăți de evenimente, schimbându-si comportamentul.

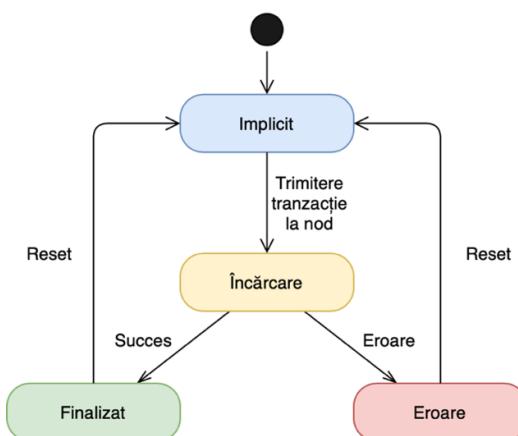


Figura 2.1. Diagrama state-machine

2.3.2. Diagrama de activitate

Diagrama de activitate arată controlul execuției de la o activitate la alta, fiind folosită pentru modelarea proceselor care descriu un anumit caz de utilizare.

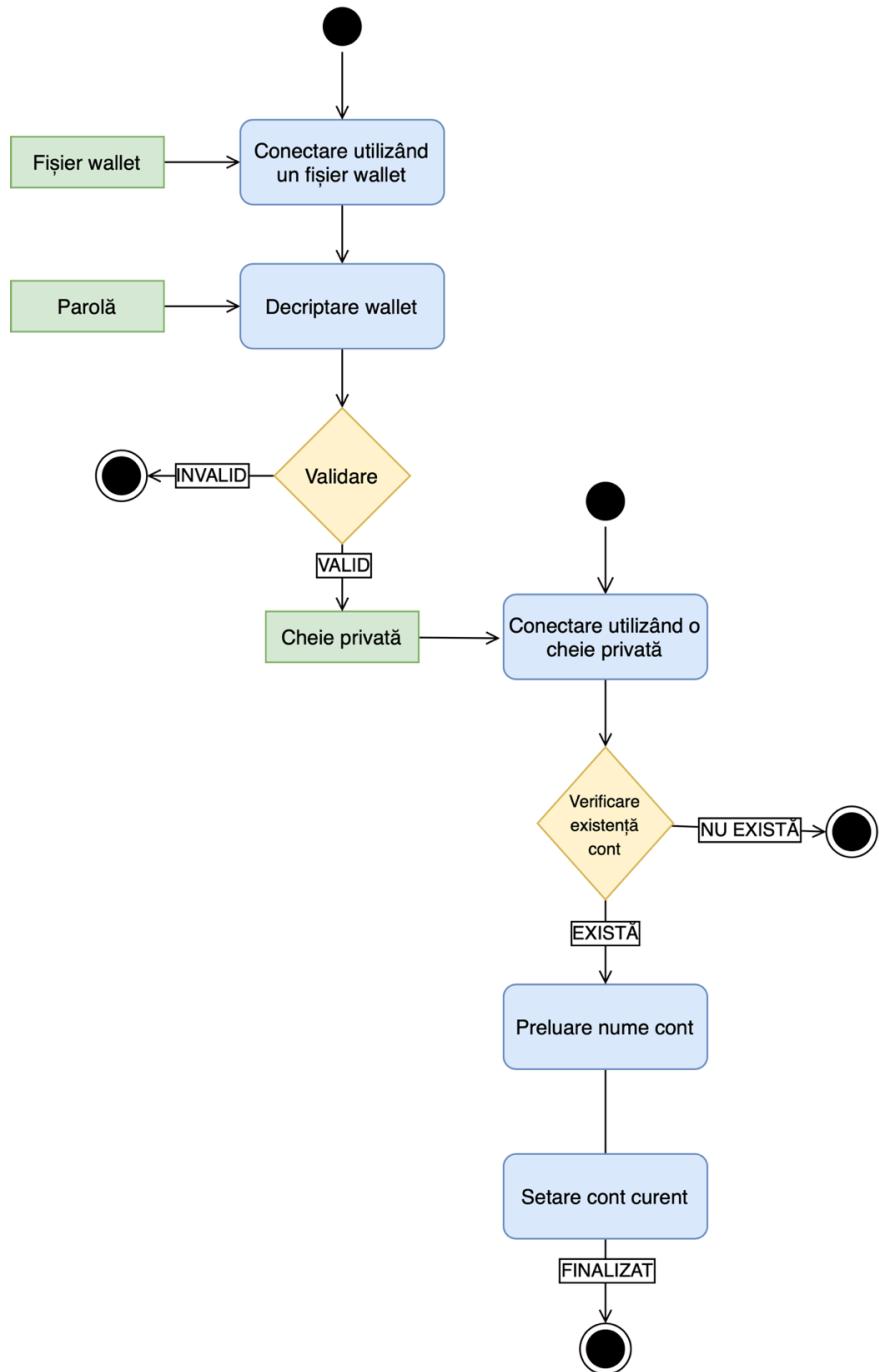


Figura 2.2. Diagrama de activitate corespunzătoare procesului de autentificare

2.3.3. Diagrama user-case

Diagrama cazurilor de utilizare (user-case diagram) descrie modul de utilizare a unui sistem informatic, a modului de interacțiune dintre utilizator și sistem, și a relațiilor dintre funcționalitățile sistemului.

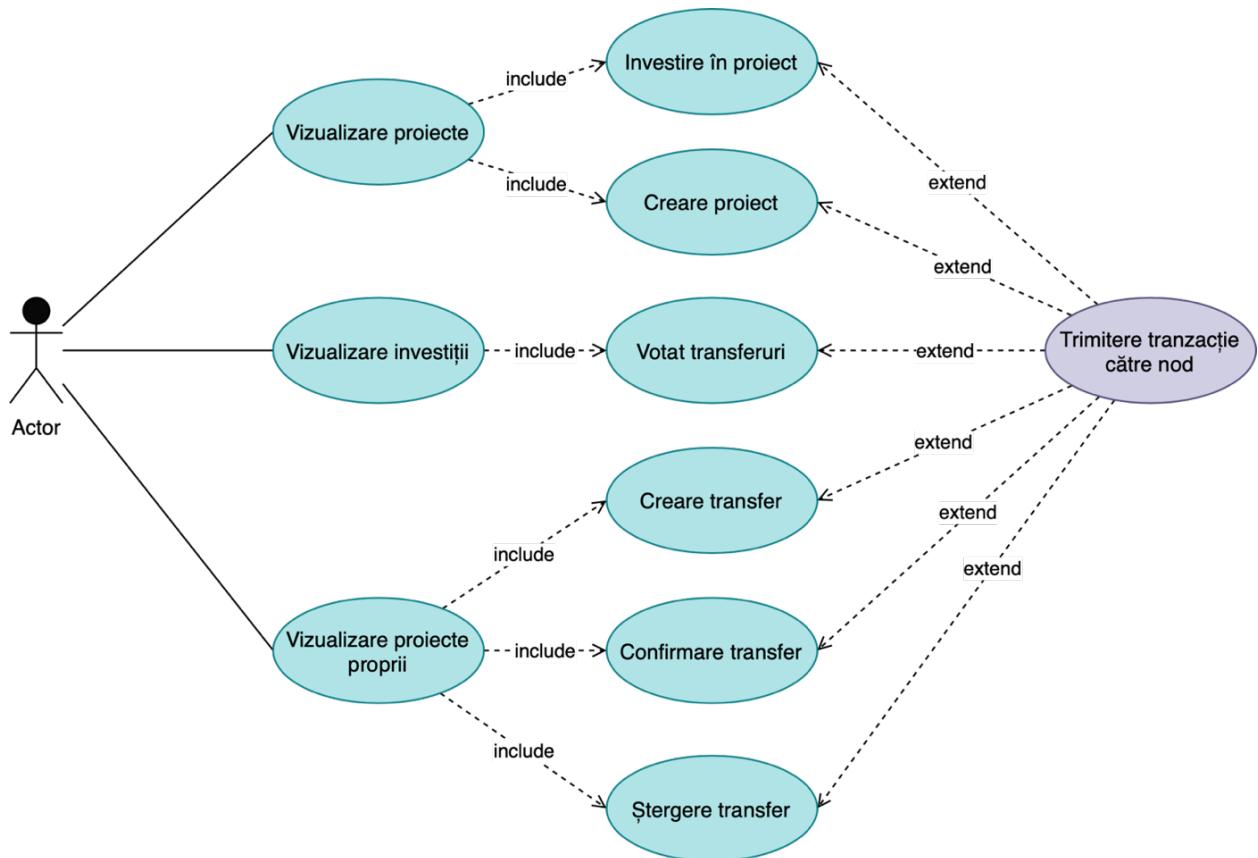


Figura 2.3. Diagrama Use-Case corespunzătoare funcționalităților proiectate în aplicație

2.3.4. Diagrama de secvență

Diagrama de secvență prezintă interacțiunile care au loc între obiectele unui sistem, într-o ordine cronologică.

În Figura 2.4. a fost descris, într-o manieră simplificată, procesul de investire și secvența de apeluri care au loc în urma acțiunii utilizatorului.

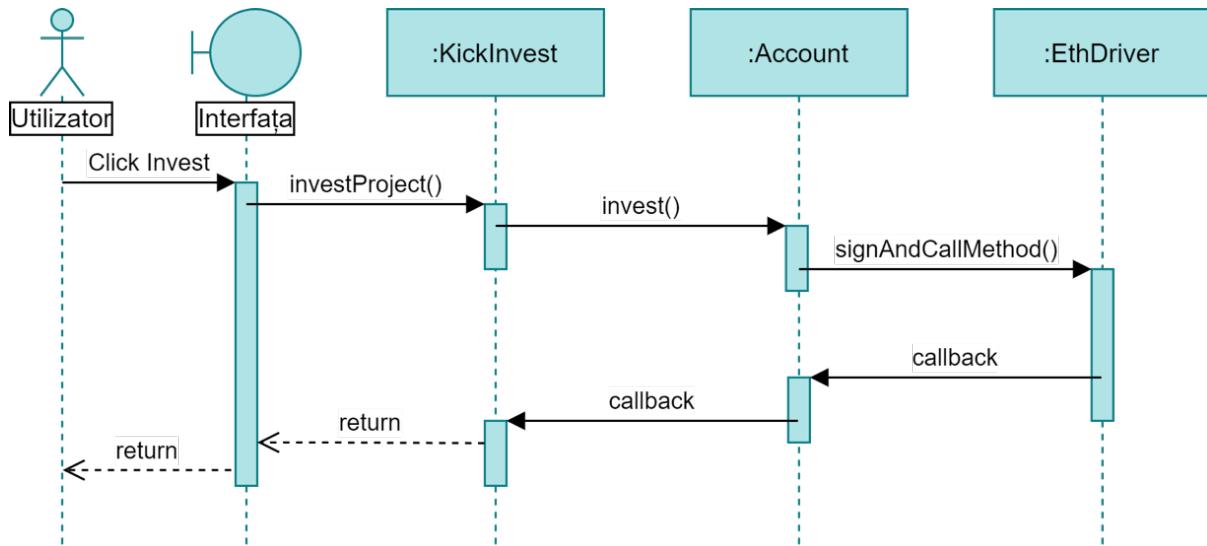


Figura 2.4. Diagrama de secvență pentru procesul de investire

2.3.5. Diagrama de clase

Diagrama de clase este utilizată pentru a modela structura unui sistem, conținând clasele și interfețele, obiectele și relațiile care se stabilesc între acestea.

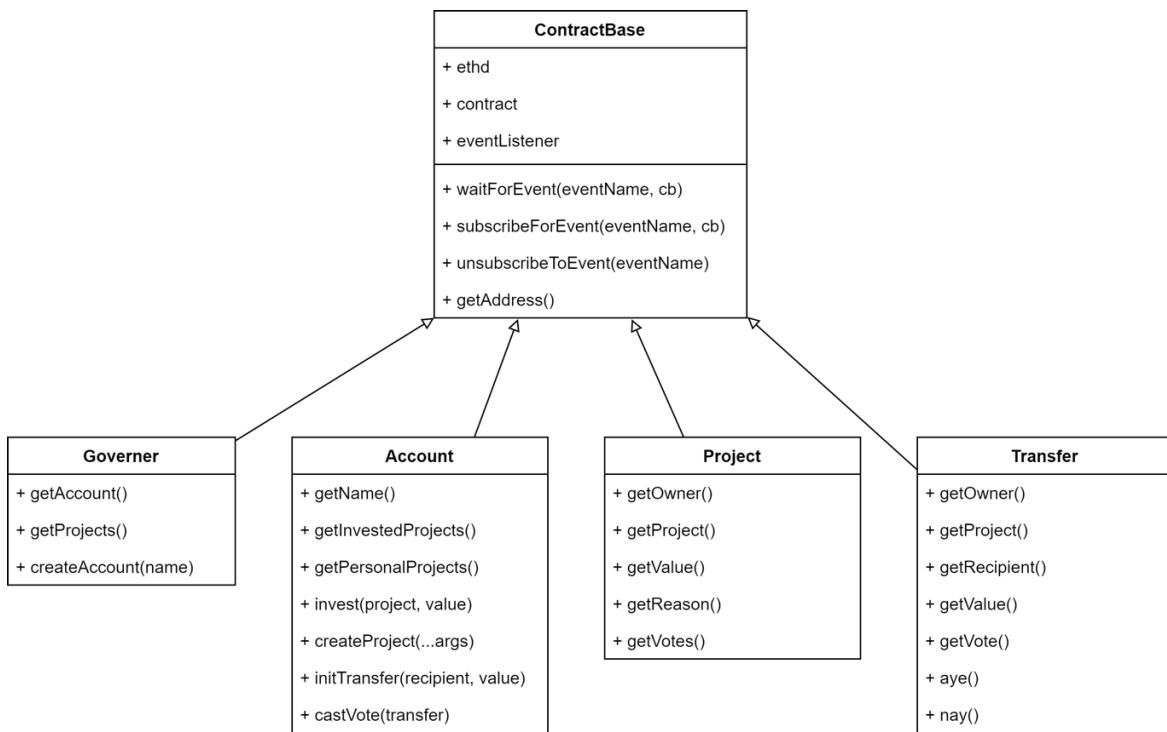


Figura 2.5. Diagrama de clase

În Figura 2.5. a fost generată diagrama contractelor, ce conține doar metodele cele mai importante, pentru a putea face posibilă ilustrarea acestora într-o manieră lizibilă. S-a folosit

pachetul *sol2uml* descărcat cu ajutorul managerului de pachete node (*npm*) pentru generarea diagramei următoare.

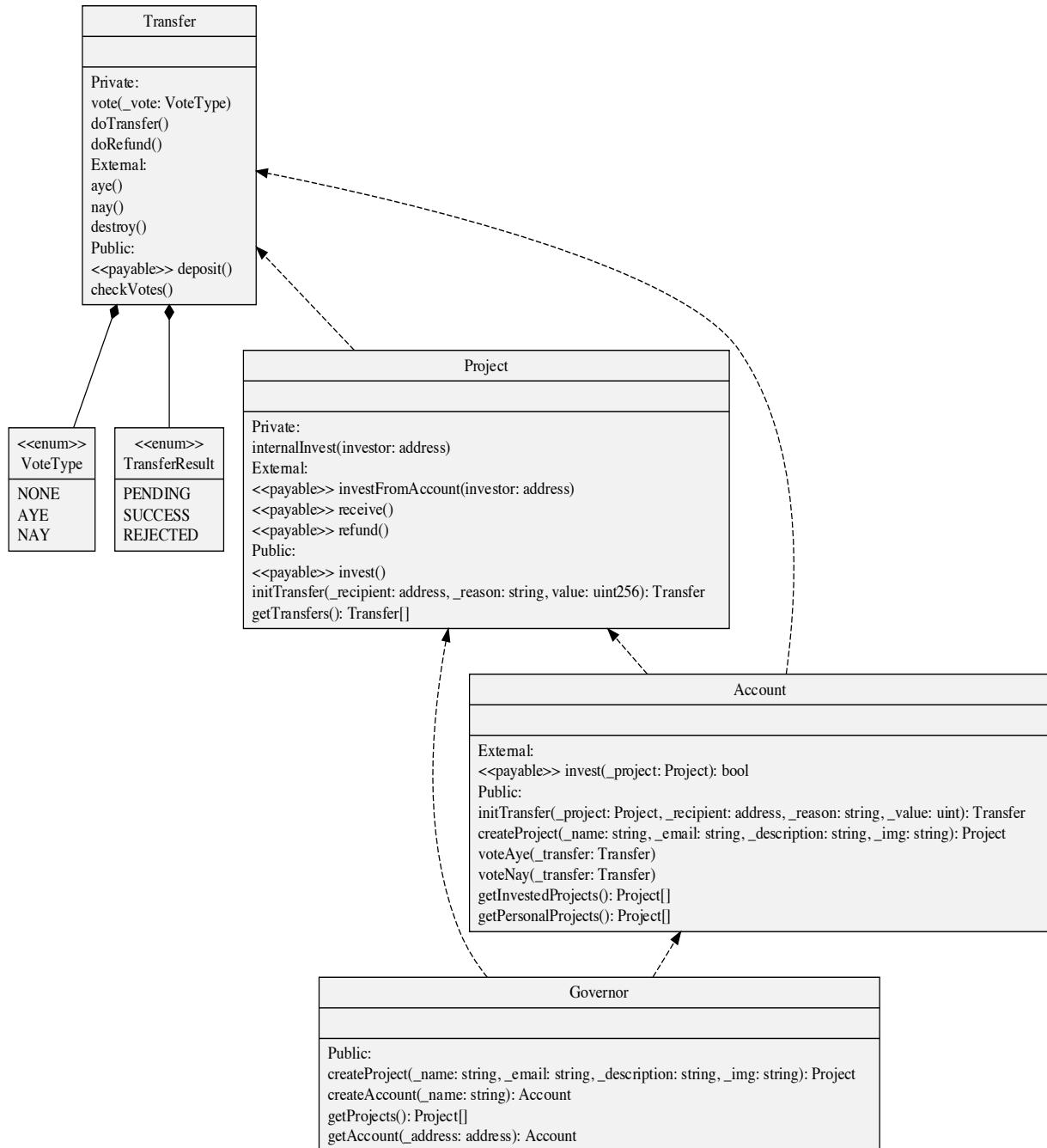


Figura 2.6. Diagrama contractelor

2.4. Avantajele și dezavantajele metodelor alese

Grație tehnologiei blockchain, metoda de proiectare aleasă oferă un grad ridicat de transparentă și securitate, iar costul tranzacțiilor este unul redus, comparativ cu sistemele rudimentare folosite în prezent. Sfera de aplicabilitate este una extinsă și protocolul de aderare în rețea nu solicită oferirea datelor cu caracter personal.

Deși această tehnologie pare a fi metoda ideală pentru un sistem incoruptibil, este greu de scalat din cauza mecanismelor de consens, iar unele probleme de calcul consumă prea multă energie. De asemenea, aceasta reprezintă o soluție mai securizată decât alte platforme, însă are încă numeroase vulnerabilități, cum are fi:

- atacul 51% (o entitate/grup deține mai mult de 51% din controlul rețelei)
- *double-spending*, posibil în cadrul rețelelor cu vulnerabilitate la atacul 51%, și este prevenit prin utilizarea mecanismelor de consens, explicitate în capitolul anterior
- atacul DDoS²⁰, în care nodurile sunt atacate cu cereri similare, aglomerând rețeaua și doborând-o
- *cryptographic cracking*, deoarece algoritmii sau computațiile cuantice sunt extrem de avansate și pot găsi soluția criptografică.

Această tehnologie necesită mai mult timp pentru maturizare și rezolvarea graduală a problemelor, însă rămâne o tehnologie pe care orice dezvoltator o poate experimenta cu fascinație.

Luând în considerare capitolul performanței și al securității, s-a ales o aplicație desktop, multi-platform, în detrimentul unei aplicații web din următoarele motive:

Psihologic, oferă un cadru mai izolat, mult mai profesional și determină utilizatorul să își canalizeze atenția într-o singură direcție, evitând distragerile oferite de plenitudinea de tab-uri care pot fi deschise într-un browser. De asemenea, acestea oferă o interfață cu utilizatorul mult mai prietenoasă și responsive.

Aplicațiile desktop, cel puțin pentru sistemul de operare macOS, sunt trecute prin mai multe filtre de analiză și trebuie să respecte criterii destul de stricte pentru a putea fi instalate și utilizate.

Incărcarea facilă a unui portofel digital în aplicație constituie un alt avantaj esențial pentru alegerea acestei implementări.

²⁰ DDoS (Distributed denial-of-service), atac malicioz destinat perturbării traficului normal al unui server prin aglomerarea unei ţinte sau a întregii infrastructuri a rețelei.

Capitolul 3. Implementarea aplicației

3.1. Implementarea componentei blockchain

Componenta blockchain conferă funcționalitatea primordială a aplicației, deoarece operațiile executate în cadrul acesteia au loc sub constrângerea unor reguli bine definite, care, încărcate în rețea, devin imutabile.

Aceste reguli sunt încapsulate în contactele inteligente, iar pentru implementarea, validarea, testarea și folosirea lor am folosit următoarele unelte:

- a) **Solidity**, limbaj de programare special proiectat pentru scrierea contractelor inteligente. Pentru a fi compilate contractele se poate folosi compilatorul solidity (*solc*), însă truffle simplifică acest mecanism, odată cu procesul de migrare, prezentat în cadrul acestui subcapitol.

În continuare se va face o descriere a funcționalității fiecărui contract, astfel:

- Governor: folosit pentru crearea proiectelor și conturilor, dar și returnarea acestora.
- Account: contract necesar memorării proiectelor create sau proiectelor în care s-a investit dintr-un cont; această metodă este cea mai rapidă pentru salvarea informațiilor la nivel de blockchain, aflându-se și într-un cadrul imutabil, iar transmiterea informațiilor pentru prelucrări și afișări ulterioare este foarte ușoară.
- Project: utilizat pentru definirea mecanismelor de investire într-un proiect, inițializare și manipulare a transferurilor, și definirea regulilor de votare.
- Transfer: conține regulile de votare, transfer sau restituire, și a verificării procentului voturilor date de către investitorii, atunci când proprietarul proiectului a adăugat o nouă tranzacție.

- b) **Ganache command line interface**, deși se poate folosi și aplicația desktop, care afișează o interfață modernă, însă oferă un cadrul de configurare mai limitat. Unealta Ganache permite crearea unui blockchain ethereum privat, pentru a executa comenzi, a rula teste și a inspecta starea rețelei și a controla modul în care operează. Mai mult decât atât, oferă abilitatea de a simula toate acțiunile care pot fi executate pe rețeaua principală, însă fără a exista un cost.

Pentru instalare și configurare s-au folosit următoarele comenzi:

```
npm install -g ganache-cli  
ganache-cli -d -e 1000000 --gasPrice 0
```

Fiind scris în JavaScript și distribuit ca un pachet Node.js prin intermediul npm, poate fi instalat local sau global, prin intermediul primei comenzi descrise mai sus.

Odată instalat, pornirea lui se face utilizând cea de-a doua comandă, având multiple opțiuni de configurare. Parametrii comenzi specifice următoarele lucruri:

- *-d* sau *--deterministic*, generează deterministice adresele pe baza unei mnemonici predefinite
- *-e* (1000000) sau *--defaultBalanceEther*, cantitatea de etheri asignată fiecărui cont. Valoarea implicită este 100, însă s-a folosit o valoare mai mare deoarece se pot crea mai

multe tranzacții și se poate testa mecanismul pentru un timp mai îndelungat, evitând astfel necesitatea încărcării frecvente a contului.

- -g sau --gasPrice (0), valoarea prețului gas-ului în wei este egală cu 0, deoarece comenziile se execută în blockchainul privat și nu există mineri.
- În urma executării comenziilor descrise anterior se generează conturile și cheile private, aceste informații fiind mereu aceleași deoarece s-a inclus parametrul pentru generarea deterministică a informațiilor.

Figura 3.1. ilustrează rezultatul obținut în urma rulării comenzi, iar pentru logarea în aplicație poate fi folosită oricare din cheile private, sau se pot utiliza alte metode, care urmează să fie descrise în capitolul următor.

```

braduveronik — node /usr/local/bin/ganache-cli -d -e 1000000 --gasPrice 0 — 97x52
Ganache CLI v6.9.1 (ganache-core: 2.10.2)

Available Accounts
=====
(0) 0x90F8bf6A479f320ead074411a4B0e7944Ea8c9C1 (1000000 ETH)
(1) 0xFcfcf8FDE72ac11b5c542428B35EEF5769C409f0 (1000000 ETH)
(2) 0x22d491Bde2303f2f43325b2108D26f1eAbA1e32b (1000000 ETH)
(3) 0xE11BA2b4D45Eaed5996Cd0823791E0C93114882d (1000000 ETH)
(4) 0xd03ea8624C8C5987235048901fB614fDcA89b117 (1000000 ETH)
(5) 0x95ceD938F7991cd0fcba48F0a06a40FA1aF46EBC (1000000 ETH)
(6) 0x3E5e9111Ae8eB78Fe1CC3bb8915d5D461F3Ef9A9 (1000000 ETH)
(7) 0x28a8746e75304c0780E011BEd21C72cD78cd535E (1000000 ETH)
(8) 0xAca94ee8bd5ffEE41947b4585a84BdA5a3d3DA6E (1000000 ETH)
(9) 0x1dF62f291b2E969fb0849d99D9Ce41e2F137006e (1000000 ETH)

Private Keys
=====
(0) 0x4f3edf983ac636a65a842ce7c78d9aa706d3b113bce9c46f30d7d21715b23b1d
(1) 0x6cbed15c793ce57650b9877cf6fa156bef513c4e6134f022a85b1ffdd59b2a1
(2) 0x6370fd033278c143179d81c5526140625662b8daa446c22ee2d73db3707e620c
(3) 0x646f1ce2fad0e6dee05c7e8e5543bde65e86029e2fd9fc169899c440a7913
(4) 0xadd53f9a7e588d003326d1cbf9e4a43c061aadd9bc938c843a79e7b4fd2ad743
(5) 0x395df67f0c2d2d9fe1ad08d1bcbb6627011959b79c53d7dd6a3536a33ab8a4fd
(6) 0xe485d098507f54e7733a205420dfddbe58db035fa577fc294ebd14db98767a52
(7) 0xa453611d9419d0e56f499079478fd72c37b251a94bfde4d19872c44cf65386e3
(8) 0x829e924fdf021ba3dbbc4225edfece9aca04b929d6e75613329ca6f1d31c0bb4
(9) 0xb0057716d5917badef911b193b12b910811c1497b5bada8d7711f758981c3773

HD Wallet
=====
Mnemonic: myth like bonus scare over problem client lizard pioneer submit female collect
Base HD Path: m/44'/60'/0'/0/{account_index}

Gas Limit
=====
6721975

Call Gas Limit
=====
9007199254740991

Listening on 127.0.0.1:8545
eth_blockNumber
eth_getBlockByNumber
eth_blockNumber
eth_blockNumber

```

Figura 3.1. Rezultatul configurării blockchain-ului privat folosind *Ganache-CLI*

- c) **Truffle**, mediu de dezvoltare și *framework* de testare pentru blockchain.
Următoarele comenzi au fost folosite pentru a configura acestă unealtă:

```

npm install -g truffle
truffle init
truffle migrate

```

Instalarea se face similar cu cea pentru ganache, initializarea presupune crearea 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.

3.2. Implementarea aplicației client

Ecosistemul Node.js este foarte generos din punct de vedere al multitudinii modulelor care pot fi folosite pentru a simplifica munca dezvoltatorilor.

Pentru aplicația client s-a folosit framework-ul React pentru manipularea datelor și a modului de afișare, și framework-ul Electron, care facilitează construirea aplicațiilor *cross-platform*. Acestea se instalează folosind managerul de pachete Node.js (npm).

Odată instalate, în fisierul Main.js trebuie inclus modulul electron și se pot crea și configura elemente ca: fereastra browser și interfața acestaia, și eventuale schimbări în funcție de sistemul de operare pe care este folosită aplicația, mecanismul de afișare a paginii și adresa serverului.

Pentru integrarea Electron cu React se folosesc următoarele ajustări ale fișierului package.json:

```
"scripts": { ...  
    "start": "BROWSER=none react-scripts start"  
    "electron": "electron ."  
}
```

Mai exact, s-a dezactivat afișării aplicației în fereastra browserului atunci când este executată comanda **npm run start**, iar pentru randarea în Electron se folosește comanda **npm run electron**.

În Electron, modulele legate de interfața grafică (dialogul, meniu, etc.), sunt disponibile doar în cadrul procesului principal (*main*), nu și în procesul de randare (*renderer*). Pentru a le utiliza din procesul de randare, modulul *ipc* este necesar pentru a trimite mesaje *inter-process* procesului principal. Folosind modulul *remote* rezolvă această problemă, prin facilitare modalității de invocare a metodelor din obiectul procesului main, fără a trimite explicit mesaje *inter-process*.

```
const { dialog } = window.require('electron').remote;  
const app = window.require('electron').remote.app;
```

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 componentă 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.

În continuare va fi prezentat mecanismul bazat pe stări folosit în momentul în care un utilizator dorește să investească într-un proiect și apare un ecran modal.

```

const [modalShow, setModalShow] = React.useState(false);

<Card onClick={() => setModalShow(true)}>

<ProjectModal
    show={modalShow}
    onHideAction={setModalShow}
/>

```

De asemenea, folosirea efectelor este utilizată pentru a managerui orice efect advers care este generat de o schimbare a datelor componentei sau ca efect al randării acesteia.

De exemplu, odată cu schimbarea stării variabilei *showAddAccount*, se apelează funcția *ResetFields*.

```

useEffect(() => {
    ResetFields();
}, [showAddAccount]);

```

3.3. Integrare componentei blockchain cu aplicația client

Lianțul dintre componenta blockchain și aplicația client este reprezentat de *Web3.js*, disponibil prin managerul de pachete *Node.js (npm)*. Clasele în care s-a construit logica de comunicare cu blockchain-ul sunt:

- **EthDriver** – conține o instanță *Web3* și contul curent și definește o serie de funcții pentru extragerea informațiilor pentru contul curent și pentru apelul funcțiilor din contracte.
- **ContractBase** – clasă de bază pentru contractele *Governor*, *Account*, *Project* și *Transfer*, care definește o instanță a *EthDriver*-ului, un contract și un event listener, și metode care așteaptă primirea unor evenimente.
- **EthFactory** – necesară atunci când se dorește furnizarea *abi*-ului și construirea clasei specificate. *ABI*-ul tuturor claselor este generat din fișierele *.json* obținute în urma migrării, cu ajutorul unui script *Python (gen_config.py)* care generează fișierul de configurare (*kickinvest-cfg.json*), unde se găsește acest *abi*.
- **KickInvest** – interfața dintre contracte și interfața grafică, conține toate funcțiile indinspensabile proiectului: prelucrarea fișierului de configurare menționat anterior, toate metode pentru cont și proiecte, și metode de encriptie, decriptie și hash-uire a unei informații.

3.4. Interfața cu utilizatorul

3.4.1. Înregistrarea/Autentificarea utilizatorului

Procesul de autentificare este unul flexibil, utilizatorul având posibilitatea de a alege între autentificarea cu o cheie privată validă, prin intermediul unui portofel aflat pe computerul personal

și o parolă, sau prin crearea unui cont de utilizator și exportarea datelor într-un portofel care conține aceste date encriptate, și o parolă, folosită pentru decriptarea datelor.

În figurile următoare sunt ilustrate cele 3 tipuri de autentificări, ce conțin reguli de validare pentru câmpurile în care se introduc datele.

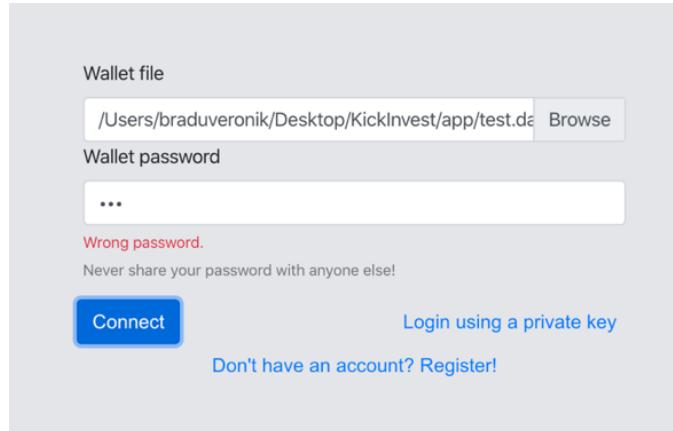


Figura 3.2. Autentificarea prin intermediul portofelului și a parolei

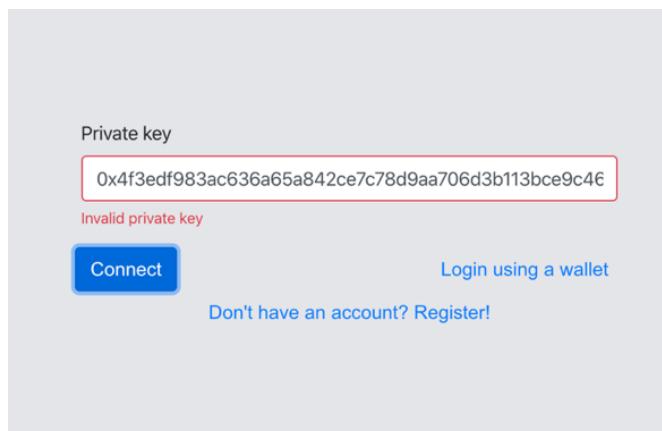


Figura 3.3. Autentificarea cu o cheie privată validă

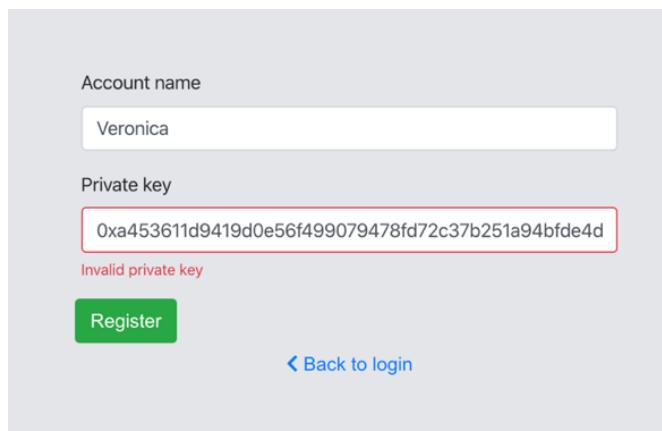


Figura 3.4. Crearea unui cont de utilizator

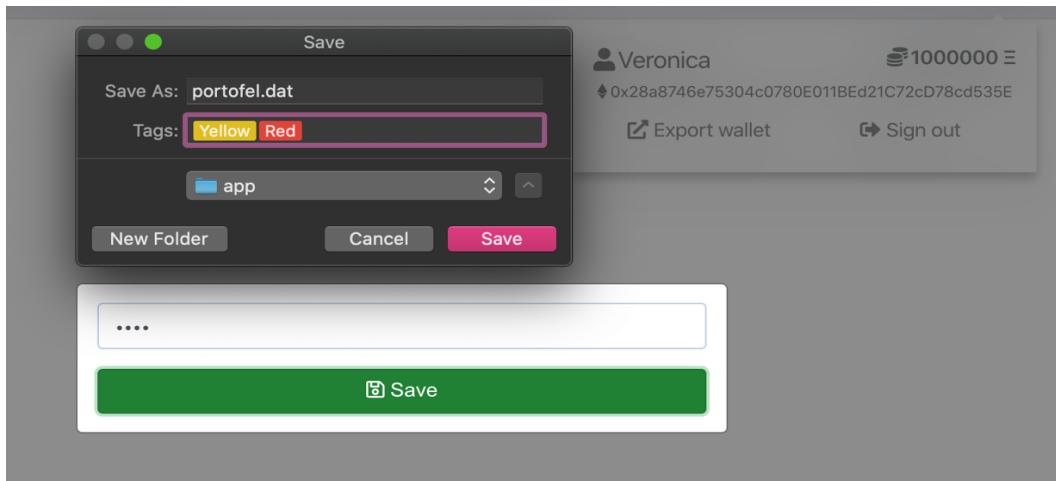


Figura 3.5. Salvarea contului creat într-un fisier portofel

Pentru encriptia și decriptia datelor s-a folosit modulul *crypto*, întreg mecanismul fiind descris de funcțiile următoare:

```
encrypt: function(key, data, alg = this.defaultAlg) {
  const salt = crypto.randomBytes(16);
  const cipher = crypto.createCipheriv(alg, key, salt);
  return Buffer.concat([salt, cipher.update(data), cipher.final()]);
}

decrypt: function(key, data, alg = this.defaultAlg) {
  const salt = data.slice(0, 16);
  data = data.slice(16);
  const decipher = crypto.createDecipheriv(alg, key, salt);
  return Buffer.concat([decipher.update(data), decipher.final()]);
}
```

3.4.2. Crearea unui proiect

Crearea unui proiect presupune autentificarea în aplicație și completarea detaliilor pentru proiect.

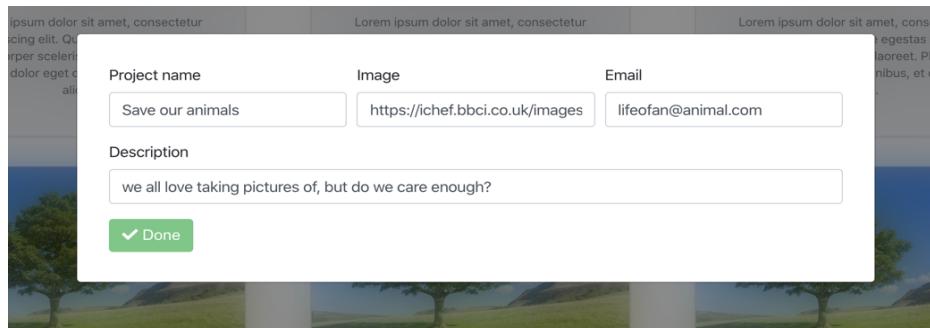


Figura 3.6. Crearea unui proiect

3.4.3. Investirea într-un proiect

Investirea într-un proiect presupune selectarea proiectului, introducerea sumei pe care utilizatorul dorește să o investească și confirmarea tranzacției.

```
function internalInvest(address investor) private {
    if (investors[investor] == 0) {
        numberOfInvestors++;
    }
    investors[investor] += msg.value;
    account += msg.value;
    emit NewInvestor(investor);
}

function invest() public payable {
    internalInvest(msg.sender);
}

function initTransfer(
    address payable _recipient,
    string memory _reason,
    uint256 value
) public onlyOwner returns (Transfer) {
    require(account >= value, "Not enough money");
    Transfer t = new Transfer(this, _recipient, _reason);
    t.deposit.value(value)();
    transfers.push(t);
    account -= value;

    // Emit NewTransfer log
    emit NewTransfer(msg.sender, t);
    return t;
}
```

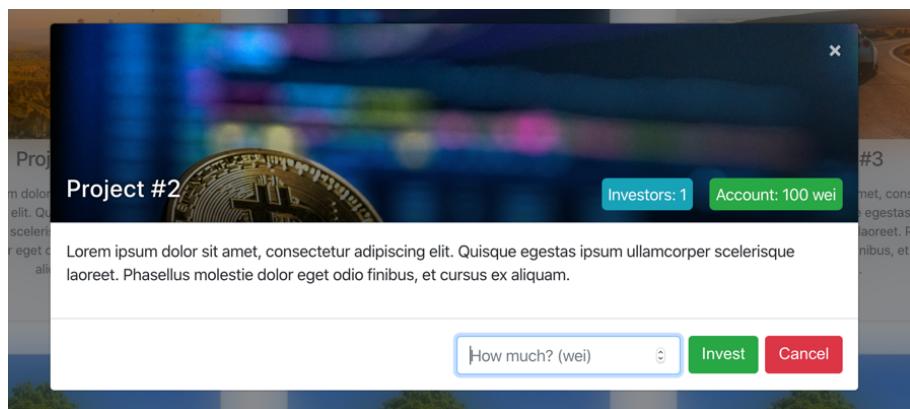


Figura 3.7. Investirea într-un proiect

3.4.4. Adăugarea transferurilor pentru un proiect

Un proiect care are investitori are nevoie de aprobarea acestora atunci când se dorește efectuarea oricărei acțiuni în cadrul proiectului.

Acest proces implică verificarea procentului de voturi raportat la numărul de investitori și a tipului voturilor majoritare: un procent majoritar pozitiv determină aprobarea acțiunii și transferarea banilor pentru rezolvarea ei, altfel, acțiunea este anulată.

```
function checkVotes() public onlyOwner {
    // Se verifica daca au votat mai mult de 50% din investitori
    uint investorNumber = project.getNumberOfInvestors();
    if (
        voteByCategory[getValueFor(VoteType.AYE)] >=
        ( investorNumber / 2 + 1)
    ) {
        doTransfer();
    }
    else if (
        voteByCategory[getValueFor(VoteType.NAY)] >=
        ( investorNumber / 2 + 1)
    ) {
        doRefund();
    }
    emit TransferStatus(result);
}
```

3.5. Dificultățile întâmpinate

Dificultățile apărute în cadrul proiectului au provenit atât din limitările limbajului Solidity, un limbaj încă nou și cu multe posibilități 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 Electron, și modelarea cerințelor definite inițial într-un limbaj în curs de dezvoltare. Documentația explicită a ajutat foarte mult la rezolvarea tuturor neclarităților și problemelor apărute pe parcurs.

S-a urmărit obținerea unei soluții cu un design atractiv pentru utilizator, cu abstractizare mecanismelor, însă cu transparență rezultatelor.

Capitolul 4. 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, sau, ideal, dacă se adoptă un stil de dezvoltare *test-driven*²¹.

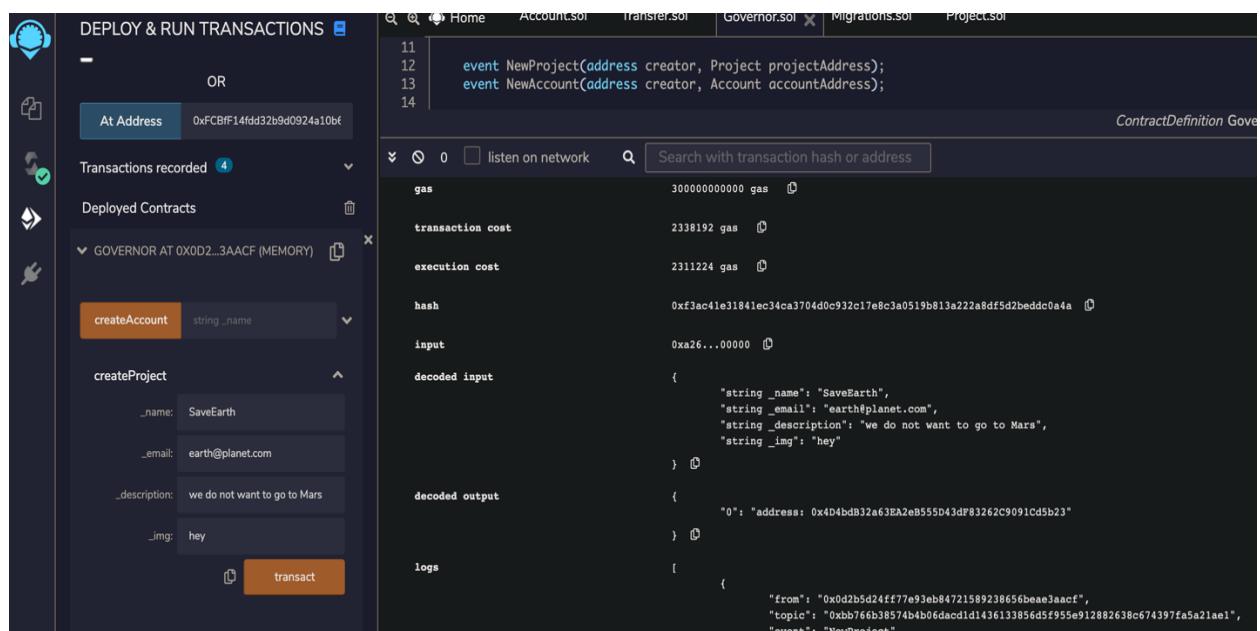
De asemenea, acest proces salvează bani, timp și uneori și vieți, deoarece în urma lor, codul este îmbunătățit și utilizatorii se pot bucura de soluțiile oferite într-un mod sigur și la standardele așteptate.

4.1. Testarea componentei blockchain

Testarea componentei blockchain s-a făcut pe platforma *Remix*, care este un instrument puternic, *open-source*, 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, iar testarea funcționalităților se poate face și cu ajutorul colecției de biblioteci oferite de *Web3.js*. S-a urmărit însă prezentarea rezultatelor într-o manieră mai detaliată, efectuată mai rapid, într-un cadru cu design modern.

După încărcarea contractelor, acestea se compilează și se testează funcționalitatea metodelor.

În Figura 4.1. se poate observa faptul că contractul *Governor* a fost compilat cu succes și că se testează funcționalitatea metodei *createProject*.



The screenshot shows the Remix IDE interface. On the left, there's a sidebar titled "DEPLOY & RUN TRANSACTIONS" with various icons. Below it, a dropdown says "At Address" with the address "0xFCBfF14fd32b9d0924a10b6". Under "Transactions recorded", there are four entries. In the center, under "Deployed Contracts", there is a section for "GOVERNOR AT 0X0D2...3AACF (MEMORY)". Inside this section, there are two tabs: "createAccount" and "createProject". The "createProject" tab is active, showing fields for "_name" (SaveEarth), "_email" (earth@planet.com), "_description" (we do not want to go to Mars), and "_img" (hey). Below these fields is a "transact" button. To the right of the sidebar, the main area shows the contract code in Solidity:

```

11 event NewProject(address creator, Project projectAddress);
12 event NewAccount(address creator, Account accountAddress);
13
14

```

Below the code, there are sections for "gas", "transaction cost", "execution cost", "hash", "input", "decoded input", "decoded output", and "logs". The "gas" section shows values like "300000000000 gas". The "hash" section shows the hash value "0xf3ac41e31841ec34ca3704d0c932c17/e8c3a0519b813a22a8df5d2beddc0a4a". The "input" section shows "0xa6...00000". The "decoded input" section shows a JSON object with fields like "string _name: SaveEarth", "string _email: earth@planet.com", etc. The "decoded output" section shows a JSON object with a single entry "0": "address: 0x4D4bdB32a63EA2eB555D43dP83262C9091Cd5b23". The "logs" section shows an array of log entries with details like "from: 0xd2b5d24ff77e93eb84721589238656beae3aacf", "topic: 0xbb766b38574b4b06dacd1d1436133856d5f955e912882638c674397fa5a21ae1", and "event: NewProject".

Figura 4.1. Testarea metodei *createProject* din contractul *Governor*

În Figura 4.2. se poate observa numele proiectului creat în pasul precedent. Urmând niște pași bine explicați în documentație, orice metodă se poate testa sau i se poate face *debug*.

²¹ Test-driven development (TDD), proces de dezvoltare software care se bazează pe cicluri scurte de dezvoltare: mai întâi se formulează cazurile de test care definesc un simplu aspect al funcționalității, se scriu testele, iar apoi se scrie codul care va face testul să treacă și tot aşa.

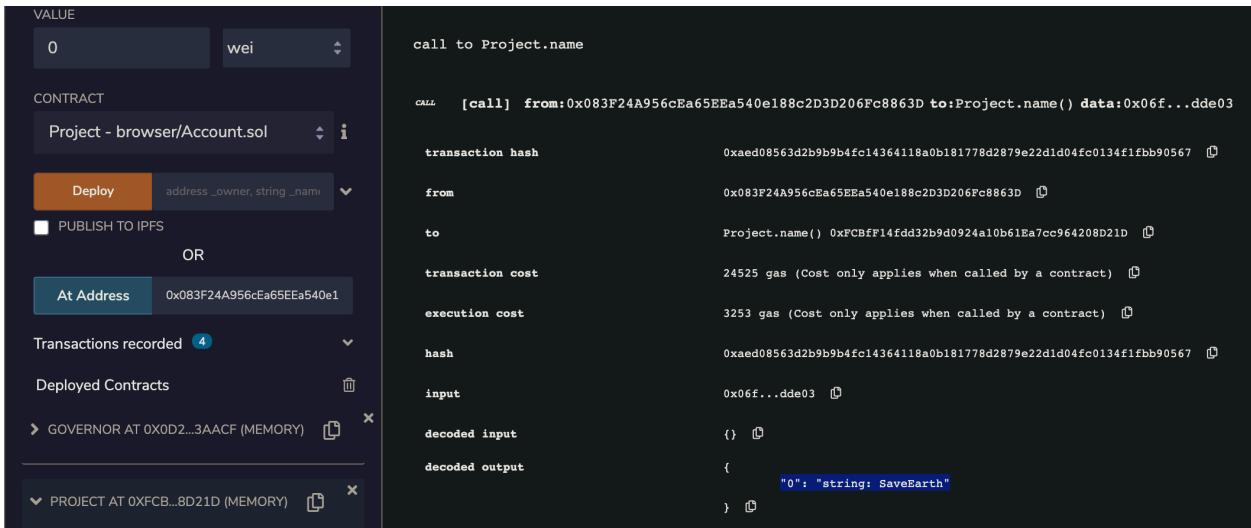


Figura 4.2. Verificarea creării proiectului folosind metoda getName din contractul Project

4.2. Testarea aplicației client

Testarea aplicației client presupune în primul rând validarea codului pentru a ne asigura că nu există nicio eroare majoră. Această validare este urmată de o verificarea a logicii fiecarei componente și a interacțiunilor dintre acestea.

Syntax Validator checks for mistakes and errors

```

1 const Web3 = require("web3");
2 const fs = require("fs");
3 const crypto = require("crypto");
4
5
6 // https://medium.com/@anned20/encrypting-files-with-nodejs-a54a073
7 const CryptoHelper = {
8   defaultAlg: 'aes-256-ctr',
9
10  encrypt: function (key, data, alg = this.defaultAlg) {
11    const salt = crypto.randomBytes(16);
12    const cipher = crypto.createCipheriv(alg, key, salt);
13    return Buffer.concat([salt, cipher.update(data), cipher.fin
14  }],
15
16  decrypt: function (key, data, alg = this.defaultAlg) {
17    const salt = data.slice(0, 16);
18
19    data = data.slice(16);
20
21  }
22}

```

Code is syntactically valid.

Figura 4.3. Validarea sintactică a codului Javascript cu ajutorul platformei Esprima

Mai mult decât atât, datele introduse de utilizator au fost verificate și s-au transmis mesaje explicite de eroare pentru a ajuta utilizatorul să înțeleagă de ce acțiunea nu s-a efectuat corespunzător, și în același timp.

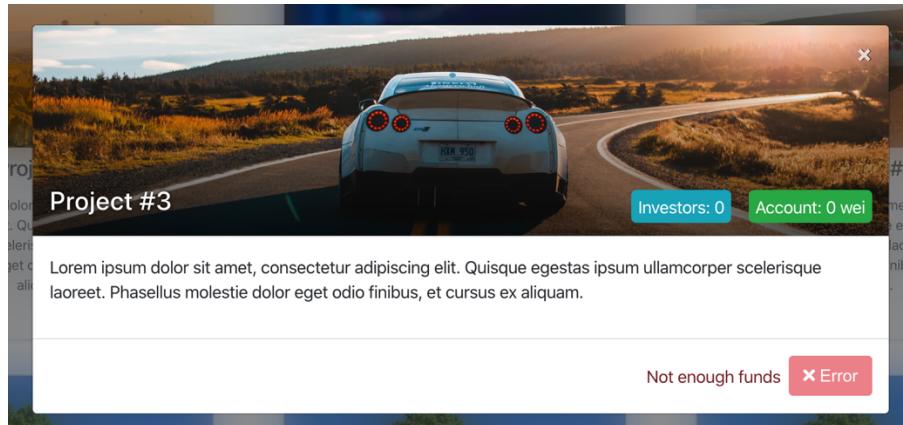


Figura 4.4. Validarea datelor introduse de utilizator

4.3. Performanța aplicației

Pentru a oferi utilizatorului o experiență unică în cadrul aplicației dezvoltate, cerințele au fost analizate din punct de vedere al performanței.

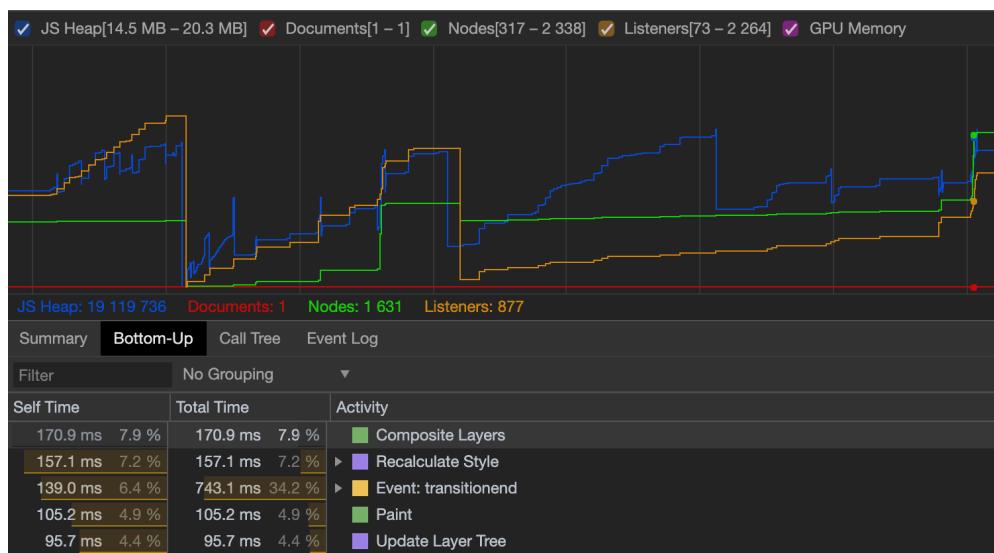


Figura 4.5. Analiza performanței memoriei

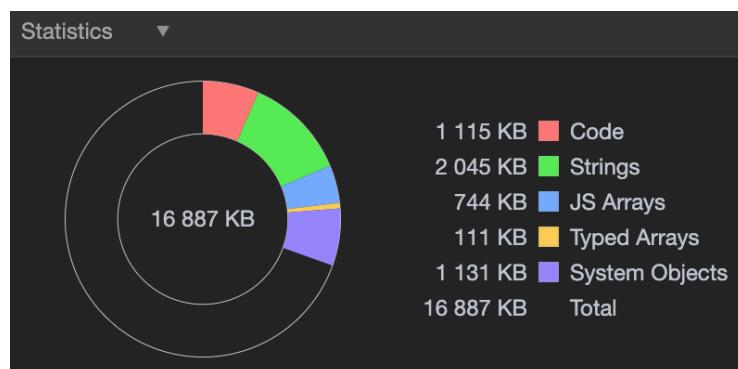


Figura 4.6. Memoria mașinii virtuale JavaScript

Concluzii

Ideea care a constituit fundamentalul acestei lucrări este reprezentată de dorința de a oferi o platformă securizată și eficientă pentru procesul de investiții în proiecte, prin construirea unei aplicații desktop *cross-platform*. De asemenea, s-au luat în calcul aspecte precum: ușurința și fluiditatea mecanismelor pe care utilizatorul le experimenteză, design-ul modern și minimalist.

În momentul actual, protocolul electronic de investiții este construit pe principiile unui sistem centralizat, în care o singură entitate controlează întregul proces: de la modul de efectuare a tranzacțiilor la rezultate și monitorizarea proiectelor. Ca o soluție la această problemă, tehnologia blockchain oferă un sistem descentralizat care se deschide pe întreaga rețea de participanți și elimină multe din problemele curente.

Utilizarea tehnologiei *Blockchain* în protocolul de investiții electronice ar satisface multe din cerințele pieței, oferind transparență ridicată asupra autenticității tranzacțiilor, costurilor și monitorizarea folosirii fondurilor acumulate.

Cosider că sfera de aplicabilitatea a tehnologiei este una densă, rezolvarea multor probleme actuale se poate plia pe arhitectura propusă spre rezolvarea problemei inițial formulate. Multe din mariile companii au început integrarea acesteia în propriile produse sau au luat baza teoretică, pe care au adăugat soluția proprie.

Prin elaborarea acestei lucrări am punctat anumite aspecte dorite a fi implementate în societate, iar prin aplicația dezvoltată am oferit un cadru de ajutor celor care doresc să propună un proiect, și celor ce văd potențialul unui proiect și se decid să investească. Societatea se îndreaptă spre principii noi, spre experimentarea noilor tehnologii și concepte, lucru firesc în mecanismul de evoluție. Sper, ca pe viitor, popularitatea tehnologiei *Blockchain* să fie folosită cât mai frecvent, în toate ariile cotidiene, grație plenitudinii de avantaje pe care le oferă.

Pentru dezvoltarea ulterioară a proiectului, intenționez să construiesc versiunea mobile a aplicație, deoarece acest sector este în continuă dezvoltare. Construirea unei aplicații hibride, care să funcționeze fluid și performant pe majoritatea sistemelor de operare de pe *smartphone*-uri, se poate face foarte ușor, modificând implementarea facută cu *React*, utilizând *React Native*. Soluția propusă este deja construită ca o aplicație desktop *cross-platform*, varianta web a acesteia poate fi ușor obținută prin mici configurații.

De asemenea, în cazul scalării aplicației, se dorește implementarea unor mecanisme de filtrare a datelor și a detecției proiectelor cu caracter imoral și înlăturarea acestora, folosind inteligența artificială.

Bibliografie

- [1] "Cambridge Judge Business School: Cambridge Centre for Alternative Finance," 24 July 2015. [Online]. Available: <https://www.jbs.cam.ac.uk/faculty-research/centres/alternative-finance>.
- [2] "Companii care au adoptat blockchain," [Online]. Available: <https://moonwhale.io/mainstream-blockchain-adoption/>.
- [3] A. Narayanan, J. Bonneau, E. Felten, A. Miller and S. Goldfeder, Bitcoin and cryptocurrency technologies: a comprehensive introduction, Princeton University Press, 2006.
- [4] A. J. B. E. F. A. M. a. S. G. Narayanan, Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction., Princeton University Press, 2016.
- [5] "Blocuri înlanțuite," [Online]. Available: <https://d32myzxfy112w.cloudfront.net/>.
- [6] "Hash Function," [Online]. Available: <https://mathworld.wolfram.com/HashFunction.html>.
- [7] "Exemplu funcție hash," [Online]. Available: <https://lh3.googleusercontent.com/>.
- [8] "Componența unui bloc," [Online]. Available: <https://medium.com/datadriveninvestor/what-is-a-block-in-the-blockchain>.
- [9] "Elementele unui bloc," [Online]. Available: <https://learnmeabitcoin.com/beginners/blocks>.
- [10] "Pragul țintă al dificultății curente," [Online]. Available: <http://man.hubwiz.com/docset/Bitcoin.docset>.
- [11] "Arborele Merkle," [Online]. Available: http://stst.elia.pub.ro/news/RCI_2009_10.
- [12] "Nonce-ul blocului," [Online]. Available: <https://coincentral.com/what-is-a-nonce-proof-of-work/>.
- [13] "Corpușul blocului," [Online]. Available: <https://forum.ethereum.org/discussion/1757/maximum-block-size>.
- [14] "Structura unui bloc," [Online]. Available: https://www.researchgate.net/profile/Sonali_Chadel.
- [15] "Bitcoin," [Online]. Available: <https://bitcoin.org/ro/cum-functioneaza>.
- [16] "Procesul de minare," [Online]. Available: <https://dev.to/damcosset/blockchain-what-is-mining-2eod>.
- [17] "Minarea unui bloc," [Online]. Available: <https://res.cloudinary.com/practicaldev/>.
- [18] "Caracteristicile unui blockchain," [Online]. Available: <https://www.geeksforgeeks.org/features-of-blockchain/>.
- [19] "Semnătura digitală," [Online]. Available: https://ro.wikipedia.org/wiki/Semnătură_digitală.
- [20] "Criptarea asimetrică," [Online]. Available: <https://hackernoon.com/>.
- [21] "Criptarea cu cheie publică," [Online]. Available: <http://users.utcluj.ro/~jim/SSA/Resources/Laboratory>.
- [22] "Ethereum," [Online]. Available: <https://coinmarketcap.com>.

- [23] "Contractele inteligente," [Online]. Available: <https://cointelegraph.com/explained/smart-contracts-explained>.
- [24] "Conceptul de contract intelligent," [Online]. Available: <https://www.etla.fi/wp-content>.
- [25] "Mecanismul unui contract intelligent," [Online]. Available: <https://rubygarage.s3.amazonaws.com>.
- [26] "Application Binary Interface," [Online]. Available: <https://medium.com/haechi-audit>.
- [27] "Truffle Suite," [Online]. Available: <https://pthomann.pl/truffle-suite-ethereum-developer-toolbox-truffle>.
- [28] "Solidity," [Online]. Available: https://www.tutorialspoint.com/solidity/solidity_overview.html.
- [29] "Web3.js," [Online]. Available: <https://web3js.readthedocs.io/>.
- [30] "Web3.js," [Online]. Available: <https://medium.com/blockcentric/getting-started-with-web3-js>.
- [31] "React," [Online]. Available: <https://www.simplilearn.com/what-is-react-article>.
- [32] "Fundition," [Online]. Available: <https://purplepaper.fundition.io>.
- [33] "Weifund," [Online]. Available: <https://weifund.readthedocs.io/>.

Anexe

Anexa 1. Structura unui contract intelligent

```
// SPDX-License-Identifier: AFL-1.1
pragma solidity ^0.5.16;

import "./Project.sol";
import "./Account.sol";

contract Governor {
    // Governor owner
    address public owner;

    event NewProject(address creator, Project projectAddress);
    event NewAccount(address creator, Account accountAddress);

    // list of all active contracts
    Project[] public projectList;

    // list of all active accounts
    mapping(address => Account) private accountList;

    constructor () public {
        owner = msg.sender;
    }

    modifier onlyOwner {
        require(msg.sender == owner,
            "Only the owner is allowed to call this function");
    }

    modifier onlyAccount {
        require(accountList[msg.sender] != Account(0),
            "Only a registered account is allowed to call this function");
    }

    function createProject(
        string memory _name,
        string memory _email,
        string memory _description,
        string memory _img
    ) public returns(Project) {
        Project p = new Project(msg.sender, _name, _email, _description, _img);
        projectList.push(p);
        emit NewProject(msg.sender, p);
        return p;
    }
}
```

```
function createAccount(string memory _name) public returns(Account) {
    Account a = new Account(msg.sender, _name);
    accountList[msg.sender] = a;
    emit NewAccount(msg.sender, a);
    return a;
}

function getProjects() public view returns(Project[] memory) {
    return projectList;
}

function getAccount(address _address) public view returns(Account) {
    return accountList[_address];
}
```

Anexa 2. Interacțiunea cu blockchain-ul prin intermediul Web3.js

```

class EthDriver {
    constructor(url) {
        this.w3 = new Web3(url);

        this.auth = null;
        this.selectedAccount = 0;
        this.wallet = []
    }

    isConnected() {
        return this.w3.currentProvider.connected;
    }

    authenticate(password) {
        this.auth = CryptoHelper.hash(password).substr(0, 32);
    }

    getWallet() {
        let addressArray = [];
        this.wallet.forEach((account) => {
            addressArray.push({
                name: account.name,
                account: account.address
            });
        });

        return addressArray;
    }

    addAccount(privateKey, name) {
        const acc = this.w3.eth.accounts.privateKeyToAccount(privateKey);
        acc.name = name;
        this.wallet.push(acc);
    }

    saveWallet(file) {
        // Get an array of all private keys currently loaded.
        let keysArray = [];
        this.wallet.forEach((account) => {
            keysArray.push({
                name: account.name,
                privateKey: account.privateKey
            });
        });

        // Serialize the array
        const walletStringBuffer = Buffer.from(JSON.stringify(keysArray));

        // Encrypt the data
        const encryptedWallet = Buffer.concat([Buffer.from('wal'), '\0', CryptoHelper.encrypt(this.auth, walletStringBuffer)]);

        // Save encrypted data to file
        fs.writeFileSync(file, encryptedWallet);
    }
}

```

```

loadWallet(file) {
    // Read encrypted data from file
    // fs.existsSync
    const encryptedWallet = fs.readFileSync(file);

    // Decrypt & deserialize data
    const keysArray = JSON.parse(CryptoHelper.decrypt(this.auth, encryptedWallet).toString());

    // Load private keys
    keysArray.forEach((obj) => {
        this.addAccount(obj.privateKey, obj.name);
    });
}

getCurrentAccount() {
    return this.wallet[this.selectedAccount].address;
}

changeCurrentAccount(i) {
    if (i >= this.wallet.length) {
        throw "Out of bounds.";
    }
    this.selectedAccount = i;
}

getBalance(address = null) {
    if (address === null) {
        address = this.getCurrentAccount();
    }
    return this.w3.eth.getBalance(address);
}

async signAndCallMethod(contractAddress, contractFunction, value = 0, gasDelta = 0) {
    const functionAbi = contractFunction.encodeABI();

    let estimatedGas;
    let nonce;
    const w3 = this.w3;

    const account = this.getCurrentAccount();

    let gasAmount = await contractFunction.estimateGas({ from: account });
    gasAmount += Math.ceil(gasAmount * 0.15);
    // estimatedGas = gasAmount.toString(16);
    console.log("GAS: " + gasAmount);

    const _nonce = await w3.eth.getTransactionCount(account);
    nonce = _nonce.toString(16);

    const txParams = {
        to: contractAddress,
        // gas: 1500000,
        // gas: Math.ceil(gasAmount + gasAmount * 0.1),
        gas: gasAmount + gasDelta,
        nonce: '0x' + nonce,
        data: functionAbi,
        value: value
    };

    const tx = await this.wallet[this.selectedAccount].signTransaction(txParams);

    const serializedTx = tx.rawTransaction;

    return new Promise((resolve, reject) => {
        w3.eth.sendSignedTransaction(serializedTx).on('receipt', receipt => {
            // console.log("receipt: " + receipt);
            resolve(true);
        }).on('error', exc => {
            console.log(exc);
        });
    });
}

```