MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

IDENTITYCHAIN - SISTEM PENTRU ADMINISTRAREA IDENTITĂȚII DIGITALE PRIN BLOCKCHAIN

LUCRARE DE LICENŢĂ

Absolvent: **Mihai-Cătălin LOȘONȚI**

Coordonator ştiinţific: **Asis. ing. Cosmina IVAN**

2018

Cuprins

Capitolul 1. Introducere

În acest capitol se va face o introducere a sistemului IdentityChain, prin prezentarea contextului acestuia, motivând totodată ideea care a stat la baza dezvoltării sistemului. La finalul acestui capitol se va prezenta conținutul lucrării.

1.1. Contextul proiectului

Trăim într-o era în care tehnologia a devenit indispensabilă pentru viețile noastre, avansând atât de mult încat este regăsită pretutindeni. Avantajele tehnologiei sunt multiple pentru omenire, dar fiecare avantaj poate deveni în același timp și un dezavantaj. Ea ne tentează să ne publicăm online declarațiile și faptele personale, unde par în siguranță, dar nu sunt. Tehnologia le permite persoanelor răuvoitoare să le sustragă, ceea ce ar trebui să fie dificil, dar de fapt e ușor. Astfel au loc furturi de indentitate și accesul nepermis la conturi bancare sau alte conturi din sfera internetului.

Fiecare utilizator al internetului, este repezentat într-o formă sau alta de către conturile sale create pe diferite aplicații mobile sau web. Aceste conturi reprezintă identitatea digitală a utilizatorului. Cele mai multe conturi de utilizator sunt protejate prin intermediul unei parole. Într-un studiu recent[1], s-a descoperit că doar 38% dintre utilizatori își creează o parolă complexă pentru fiecare cont, în timp ce o persoană din șapte (14%) are o singură parolă pentru toate conturile sale. Aceşti utilizatori riscă să le fie compromise mai multe conturi simultan, în cazul unui furt de date. Cu toate acestea, gradul de risc nu este cu mult mai mic pentru cei care folosesc câteva parole pentru un număr mai mare de conturi (36%) sau pentru cei care folosesc variaţiuni ale aceluiaşi tipar (12%).

Datorită problemelor enumerate mai sus, în domeniul securității informatice, a apărut o disciplină numită Administrarea Identitatii Digitale (sau Identity Management) care dorește „accesul indivizilor la resursele potrivite, în momente potrivite, din motive corecte”. De asemenea, pe data de 25 mai 2018, a intrat în vigoare regulamentul european privind protecția datelor cu caracter personal, prescurtat GDPR (General Data Protection Regulation). Acest regulament prevede o serie de drepturi și obligațiuni pentru cetățenii Uniunii Europene și operatori. Pe scurt acest regulament împiedică prelucrarea și accesul la datele cu caracter personal ale utilizatorilor, fără consințământul acestora.

1.2. Motivația

Motivația realizării acestui proiect provine din dorința de a oferii o soluție de securizare a accesului pentru utilizatorii internetului, astfel proiectul propus va oferii acestora un sistem prin care identitatea digitală va fi administrată în manieră personală. Utilizatorii își pot crea o singură identitate digitală cu care se pot înregistra pe diferite platforme web, prin scanarea unui cod QR cu ajutorul telefonului mobil. Astfel individul are acces la mai multe conturi de utilizator, fară să mai fie nevoie de o parolă. Identitatea digitală a utilizatorlui va fi stocată sub formă criptată în blockchain, asigurând în acest mod securizarea sa.

[1] http://www.capital.ro/cum-sa-ti-protejezi-identitatea-digitala.html

În al doilea rând, familiarizarea cu tehnologia blockchain și utilizarea ei într-un proiect cu aplicabilitate practică este un alt motiv al alegerii proiectului.

1.3. Structura

În continuare se va prezenta conținutul lucrării. Structura documentului fiind precizată în lista următoare:

- În capitolul 1, **Introducere**, au fost prezentate contextul și motivația din spatele acestui proiect.
- În capitolul 2, **Obiectivele Proiectului**, sunt enumerate obiectivele principale, dar si cele secundare pe care proiectul trebuie să le atingă.
- În capitolul 3, **Studiu Bibliografic**, se vor prezenta conceptele esențiale din domeniul în care se încadrează tema sistemului propus, fiind explicate concepte precum: blockchain, indetitate digitală și smart contracts. De asemenea, aici se vor prezenta și câteva sisteme similare cu proiectul propus.
- În capitolul 4, **Analiză și Fundamentare Teoretică**, se va descrie arhitectura conceptuală a sistemului implementat. Totodată, se vor prezenta tehnologiile utilizate și motivele utilizării acestora în contextul dat. Pe urma se va prezenta o serie de cerințe funcționale și non-funcționale și se vor defini cazurile de utilizare ce se propun a fi implementate în acest sistem.
- În capitolul 5, **Proiectare de Detaliu și Implementare**, sunt descrise etapele de proiectare ale sistemului, pentru fiecare componentă și modul în care acesta funcționează.
- În capitolul 6, **Testare și Validare**, se vor prezenta o serie de teste la care a fost supus sistemul, precum și rezultatele acestora. Tot aici vor analiza rezultatele simulării unui caz real.
- În capitolul 7, **Manual de Instalare și Utilizare**, sunt descriși pașii necesari instalării sistemului și operațiile necesare ce trebuie efectuate pentru buna funcționare a acestuia, precum și un manual de utilizare.
- În capitolul 8, **Concluzii**, se vor prezenta concluziile desprinse din realizarea proiectului propus, contribuțiile aduse la implementarea acestuia, dar și o serie de idei prin care sistemul poate fi dezvoltat ulterior.

Capitolul 2. Obiectivele Proiectului

În cadrul acestui capitol se vor enumera și prezenta obiectivele principale ale proietului implementat, precum și obiectivele secundare care vor avea un rol important în buna funcționare a sistemului.

2.1. Obiective principale

Primul obiectiv principal, care a jucat un rol important în alegerea temei proiectului propus este **studiul tehnologiei blockchain**. Blockchain-ul este un registru public distribuit într-o rețea peer-to-peer, cu potențial de a crea noi baze pentru sistemele economice și sociale globale, reducând frauda financiară prin automatizarea proceselor care anterior erau consumatoare de timp și resurse umane. Deoarece tehnologia blockchain este tot mai răspândită, fiind folosită de către tot mai multe companii, înțelegerea modului de funcționare a acesteia este benefic în aplicarea și în dezvoltarea anumitor clase de aplicații.

Al doilea obiectiv este proiectarea și crearea unui sistem distribuit, folosind tehnologia blockchain, care să asigure **administrarea identitatii digitale**, fiind utilizabil în aplicații web. Deoarece, în zilele noastre, navigarea pe internet presupune tot mai multe riscuri, sistemul propus dorește să elimine cât mai multe dintre acestea, cu ajutorul tehnologiei blockchain. Componenta principală a sistemului va fi o aplicație pentru telefonul mobil, care va administra identitatea utilizatorului și care se va putea conecta la diferite aplicații web. Folosind tehnologia blockchain, utilizatorii sistemulului vor putea interacționa cu contracte smart, o nouă formă de contracte care poate acționa ca o completare sau înlocuire a contractelor legale, tradiționale, nemaifiind nevoie de o autoritate centrală pentru executarea lor. Pentru a integra sistemul într-o aplicație web, se va dezvolta o bibliotecă de JavaScript, pe care operatorii aplicațiilor web o vor putea integra în sistemele lor.

Ultimul obiectiv principal, este realizarea unei **aplicații demonstrative**, prin care să se exemplifice funcționalitățile sistemului. Astfel se va realiza o aplicație web pentru o casă de licitații, care utilizează un contract smart pentru partea financiară.

2.2. Obiective secundare

În lista următoare se vor prezenta obiectivele secundare, care sunt urmărite pentru realizarea unui sistem cât mai prietenos și util:

- **Recuperare identitate:** Al treilea obiectiv secundar este de a permite utilizatorilor sistemului să își recupereze identitatea și totodată accesul la adresa lor din blockchain, în cazul în care acestia își pierd sau schimbă telefonul mobil.
- **Notificarea utilizatorului:** Utilizatorul va fi notificat în cazul în care identitatea sa este cerută de o aplicație web sau în cazul în care se dorește realizarea unei tranzacții în blockchain din contul său. De asemenea, utilizatorul poate respinge o cererile primite.
- **Crearea unei interfețe utilizator intuitivă:** Ultimul obiectiv secundar al acestui proiect, este de a realiza o aplicație mobilă care să aibă o interfată utilizator cât mai prietenoasă și intuitivă.

- **Vizualizare tranzacții proprii:** Al doilea obiectiv secundar este de a oferii posibilitatea utilizatorilor să-și vizualizeze propriile tranzacții efectuate în blockchain. Acest lucru se va realiza în aplicația mobilă.
- **Transferul de criptomonede:** Criptomoneda este un tip de monedă digitală, folosită ca mijloc de plată. Primul obiectiv secundar este de a se permite transferarea criptomonedei *ether(ETH)* între utilizatorii sistemului, dar și între utilizatori și alte sisteme deja existente. Acest obiectiv reprezintă defapt, funcționalitatea sistemului de a fi un portofel electronic. Astfel, utilizatorii vor putea efectua tranzacții financiare.

Capitolul 3. Studiu Bibliografic

În acest capitol se vor prezenta conceptele esențiale din domeniul în care se încadrează tema sistemului propus. În urmatoarele subcapitole vor fi enunțate concepte precum: blockchain, smart contracts, identitate digitală. Tot în acest capitol vor fi prezentate sisteme similare cu proiectul propus.

3.1. Blockchain

Blockchain-ul [1] este, în esență, o bază de date distribuită de înregistrări sau de informații publice ale tuturor tranzacțiilor sau evenimentelor digitale care au fost executate și partajate între părțile participante. Blockchain-ul poate fi interpretat ca și un „registru”(*ledger* în engleză) public ce conține informații despre tranzacții. Fiecare tranzacție din registrul public este verificată prin consensul majorității participanților la sistem. Odată introduse, informațiile nu pot fi șterse sau alterate. Autorii din articolul [1] prezintă conceptul de blockchain printr-un exemplu din viața de zi cu zi: ”este mult mai usor să furi o prajitură dintr-un borcan cu prăjituri aflat într-un loc retras, decât dintr-un borcan aflat într-o piață publică, unde este observat de către mii de oameni.”

Primul blockchain distribuit a fost conceptualizat în anul 2008 de o persoană anonimă care s-a identificat cu numele de Satoshi Nakamoto. Aceași persoană anonimă a creat criptomoneda *Bitcoin* în anul 2009.

3.1.1. Arhitectura

Blockchain-ul este o listă de înregistrări în continuă creștere numite blocuri, care sunt legate între ele cu ajutorul criptografiei. Ca structură de date, un blockchain este o lista simplu înlănțuită, în care legăturile dintre blocuri se realizează printr-un *hash*.

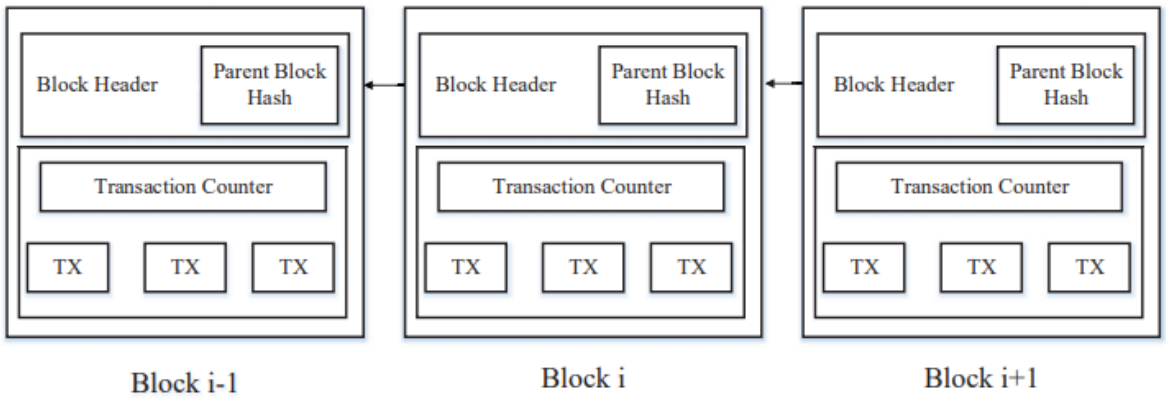

Figura 3.1. Exemplu de blockchain [2]

În figura 3.1 este prezentat un exemplu de blockchain ce conține o secvență continuă de blocuri. Fiecare bloc conține hash-ul blocului anterior, astfel realizându-se legatura dintre ele.

O funcție hash(sau funcție de dispersie) [3] este, în sens matematic, este o funcție definită pe o mulțime cu multe elemente (posibil infinită) cu valori într-o mulțime cu un număr fix. Funcțiile hash nu sunt inversabile. O funcție hash poate lega două sau mai multe chei de la aceeași valoare hash. În multe aplicații, este de dorit minimalizarea șansei apariției unor astfel de

coliziuni, ceea ce înseamnă că funcția hash trebuie să lege cheile de valorile hash cât mai uniform posibil. Deși ideea a fost concepută în anii 1950, proiectarea optimă a funcțiilor hash este încă un subiect activ de cercetare și discuție. Funcțiile hash sunt utilizate și ca sume de control sau funcții hash criptografice, dar nu trebuie confundate cu caracterele de verificare, amprentele numerice, funcțiile de randomizare, codurile de corectare a erorilor. Figura 3.2 prezintă grafic procesul prin care se crează o valoare hash dintr-o data de lungime arbitrară.

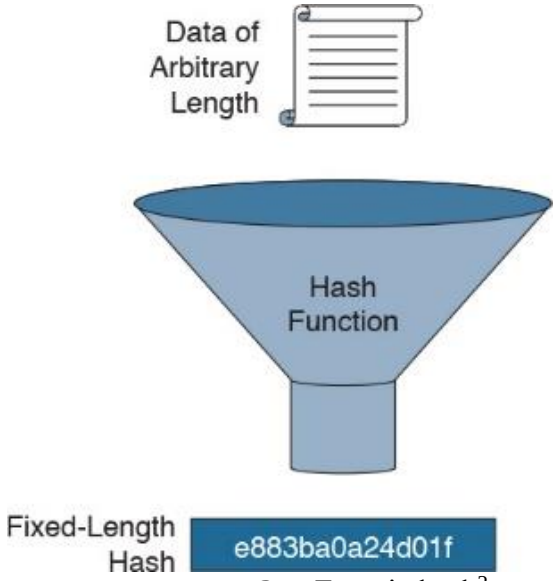

Figura 3.2. Funcție hash[2]

Hash-ul fiecărui bloc este format din datele din bloc plus hash-ul blocului anterior(parinte) astfel încât, dacă va exista o modificare într-un bloc parinte, valorile hash ale blocurilor copii vor fi și ele modificate.

Un bloc este format din:

- Antet(block header)
- Corp(block body)

În figura 3.3 este prezentată structura unui bloc. Antetul unui bloc cuprinde urmatoarele elemente:

- **Block Version:** indică ce set de regului de validare trebuie să urmeze blocul curent
- **Merkle Tree Root Hash:** valoarea hash a tuturor tranzacțiilor din blocul curent
- **Timestamp:** timpul curent, universal în secunde de la 1 Ianuarie, 1970
- **nBits:** ținta este pragul sub care trebuie să fie hash-ul, astfel încat blocul să fie valabil. nBits este forma codată a pragului țintă
- **Nonce:** un câmp de 4 octeți, care de obicei începe cu 0 și crește pentru fiecare calcul hash
- **Parent Block Hash:** o valoare hash de 256 de biți care indică blocul anterior

[2]http://ptgmedia.pearsoncmg.com/imprint_downloads/cisco/digitalstudyguide/9780134424064/ch29.html

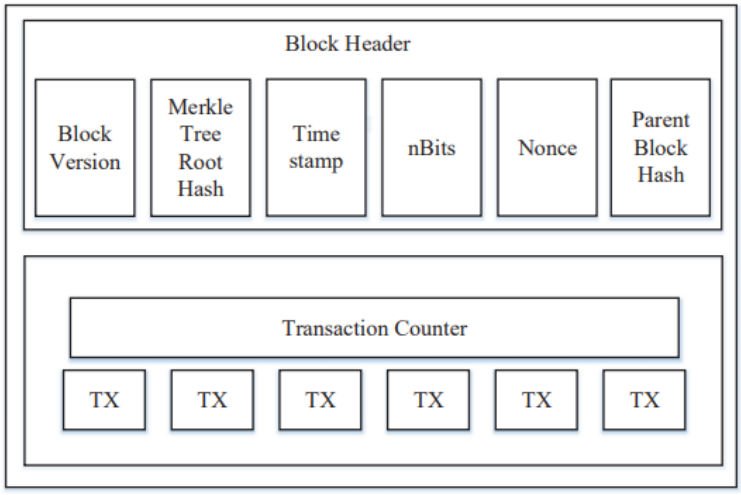

Figura 3.3. Structura unui bloc [2]

Corpul blocului este compus din lista de tranzacții și un contor de tranzacții. Numărul maxim de tranzacții pe care un bloc poate să le conțină depinde de dimensiunea blocului și de dimensiunea maxima a fiecărei tranzacții. Blockchain-ul folosește o criptografie asimetrică pentru mecanismul de validare a autentificării tranzacțiilor.

3.1.2. Semnătura digitală

Semnătură digitală bazată pe criptografia asimetrică este utilizată într-un mediu nesigur. Fiecare utilizator deține o pereche de chei, o cheie privată și o cheie publică. Cheia privată este confidențială și va fi folosită pentru semnarea tranzacțiilor. Tranzacțiile digitale semnate vor fi distribuite în întreaga rețea. Semnătura digitală tipică este implicată în doua faze:

- Faza de semnare
- Faza de verificare

În figura 3.4 este prezentat procesul de transmitere al unui mesaj folosind criptarea asimetrică.

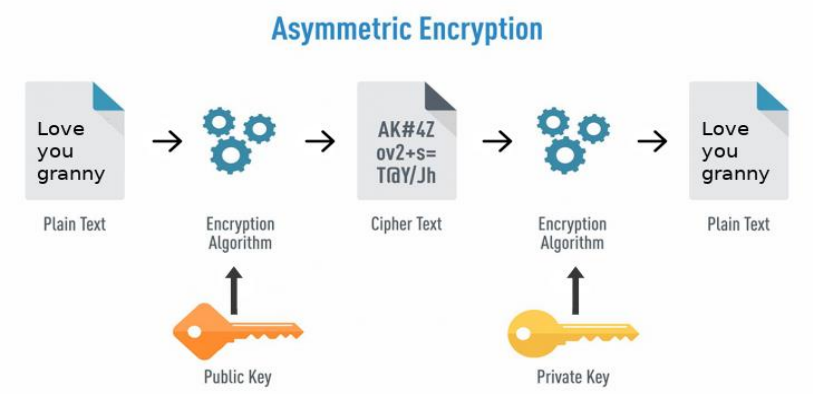

Figura 3.4. Criptarea asimetrică[3]

[3] https://hackernoon.com/symmetric-and-asymmetric-encryption-5122f9ec65b1

Vom expica procesul de autentificare printr-un exemplu, folosind doi actori: Bob si Alice. Alice dorește să îi trimită lui Bob un mesaj. În faza de semnare, Alice criptează mesajul cu cheia ei privată și trimite lui Bob mesajul original plus rezultatul criptării. În faza de verificare, Bob decriptează mesajul cu cheia publică a lui Alice și astfel verifică dacă datele au fost modificate sau nu.

3.1.3. Caracteristicile unui blockchain

Caracteristicile cheie ale unui blockchain sunt prezentate în lista de mai jos.

1. **Decentralizarea.** În sistemele convenționale de tranzacții centralizate, fiecare tranzacție trebuie să fie validată prin intermediul unei agenții centrale de încredere (de exemplu, banca centrală), adăugând în mod inevitabil costurile și blocajele de performanță ale serverelor centrale. În contrast cu modul centralizat, agenția centrală nu mai este necesară în blockchain. Algoritmii de consens din blockchain sunt utilizați pentru a menține consitența datelor în rețeaua distribuită.
2. **Persistența.** Tranzacțiile pot fi validate rapid, iar tranzacțiile nevalabile nu vor fi admise de catre mineri. Este aproape imposibil ca tranzacțiile să fie șterse sau retrase, odată ce acestea sunt incluse în blockchain. Blocurile care conțin tranzacții nevalide pot fi descoperite imediat.
3. **Anonimat.** Fiecare utilizator poate interacționa în blockchain cu ajutorul unei adrese generate, care nu dezvăluie identitatea reală a utilizatorului. De reținut este faptul că blockchain-ul nu poate garanta conservarea perfectă a confidențialității.

3.1.4. Clasificarea sistemelor blockchain

Sistemele actuale de blockchain-uri sunt clasificate în trei tipuri: blockchain-uri publice, blockchain-uri private și blockchain-uri de consorțiu(consortium blockchains). În blocurile publice, toate înregistrările sunt vizibile publicului, iar toată lumea ar putea lua parte la procesul de consens. În mod diferit, doar un grup de noduri pre-selectate pot participa la procesul de consens al unui blockchain de consorțiu. În privința blockchain-ului privat, numai nodurile care provin dintr-o organizație specifică vor putea să se alăture procesului de consens. Procesul de consens este un proces decizional de grup în care membrii grupului discută anumite idei și ajung la un acord în interesul întregului grup.

Un blockchain privat este privit ca o rețea centralizată, deoarece este controlat de o singură organizație. Blockchain-ul de consorțiu construit de mai multe organizații este parțial descentralizat, deoarece doar o mică parte a nodurilor va fi selectată pentru determinarea consensului. Comparația dintre cele trei tipuri de blockchain-uri este prezentată în tabelul 3.1. De asemenea se vor definii criteriile de diferențiere.

Tabel 3.1. Comparație între blockchain-uri [2]

Proprietate	Blockchain public	Blockchain de consorțiu	Blockchain privat
Determinarea consensului	Toți participanții	Set de noduri selectate	O organizație
Permisiunea de citire	Publică	Publică sau restricționată	Publică sau restricționată
Caracter imutabil	Aproape imposibil de falsificat	Poate fi manipulat	Poate fi manipulat
Eficiența	Mică	Mare	Mare
Centralizarea	Nu	Parțială	Da
Procesul de consens	Fără permisiune	Cu permisiune	Cu permisiune

- **Determinarea consensului:** în blockchain-ul public, fiecare nod ar putea lua parte la procesul de consens. Doar un set selectat de noduri este responsabil pentru validarea blocului în blockchain-ul de consorțiu. În ceea ce privește blockchain-ul privat, acesta este controlat de o singură organizație, iar organizația determină consensul final.
- **Permisiunea de citire:** tranzacțiile într-un blockchain public sunt vizibile publicului, în timp ce într-un blockchain de consorțiu sau privat, acest lucru depinde de administratori.
- **Caracterul imutabil:** deoarece într-un blockchain public, înregistrările sunt stocate de un număr mare de participanți, este aproape imposibil ca acestea să fie falsificate. În mod diferit, tranzacțiile într-un blockchain privat sau într-un bloc de consorțiu ar putea fi modificate cu ușurință, deoarece există un număr limitat de participanți.
- **Eficiența:** într-un blockchain public este nevoie de mult timp pentru propagarea tranzacțiilor și a blocurilor în rețaua distribuită. Ca urmare, distribuirea tranzacțiilor este limitată, iar latența ridicată. Cu mai puțini validatori, blockchain-urile de consorțiu și blockchain-urile private sunt mai eficiente.
- **Centralizarea:** diferența principală dintre cele trei tipuri de blockchain este că cel public este descentralizat, blockchain-ul de consorțiu este parțial centralizat, iar blockchain-ul privat este complet centralizat, deoarece este controlat de un singur grup.
- **Procesul de consens:** toată lumea ar putea să se alăture procesului de consens al blockchain-ului public. Față de blockchain-urile publice, atât blockchain-urile de consorțiu, cât și blockchain-urile private sunt autorizate.

Deoarece blockchain-ul public este deschis pentru utilizare, acesta atrage mulți utilizatori, iar comunitățile sunt active. Multe blockchain-uri publice apar zilnic. În ceea ce privește blockchain-ul de consorțiu, acesta este utilizat în aplicații pentru mediul de afaceri.

3.1.5. Procesul de minare

Mineritul [4] este mecanismul care permite blockchain-ului să fie o securitate decentralizată. Minerii validează o nouă tranzacție și o adaugă în registrul public al blockchain-ului. În medie, un bloc este minat odata la 10 minute. Minerii concurează pentru a rezolva o

problemă matematică dificilă, bazată pe un algoritm hash criptografic. Soluția găsită se numește Proof-Of-Work. Aceasta dovedește că un miner a petrecut timp și resurse pentru rezolvarea problemei, iar la final minerul va primi o recompensă sub forma unei criptomonede.

Scopul minării este de a gasi o valoare pentru **nonce**(definit în subsecțiunea 3.1.1) din care va rezulta o valoare hash al blocului mai mica decât campul **nBits(target).** Deci, se vor încerca miliarde de valori nonce înainte ca blocul să primească o valoare hash validă.

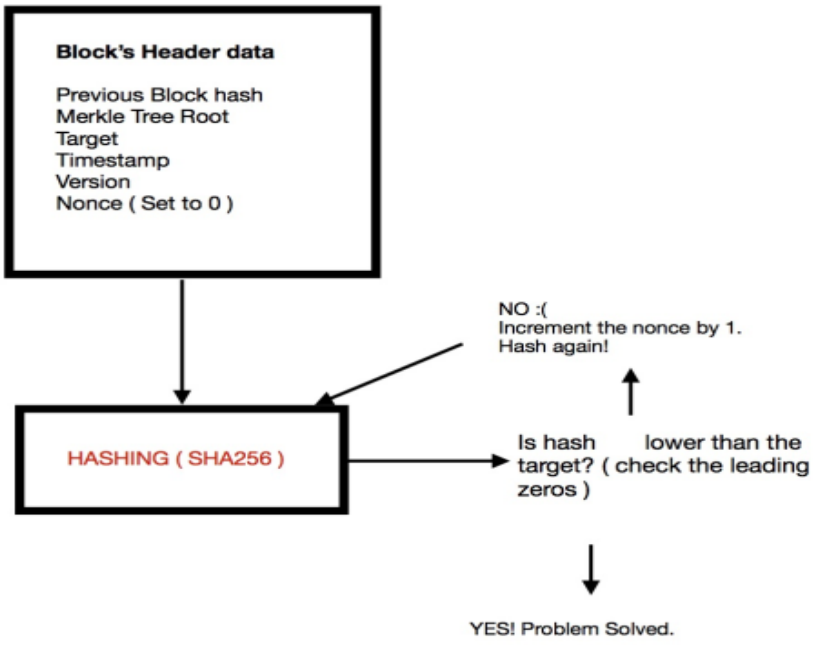

Figura 3.5. Procesul de minare [4]

3.2. Smart Contracts

Smart Contract(sau contract inteligent) [5] este un termen folosit pentru a descrie un cod de program informatic care este capabil să faciliteze, să execute și să impună negocierea sau executarea unui acord (adică contract) utilizând tehnologia blockchain. Întregul proces este automatizat și poate acționa ca o completare sau înlocuire a contractelor legale, în care termenii contractului smart sunt înregistrați într-un limbaj de calculator ca un set de instrucțiuni.

Contractele inteligente sunt doar programe informatice care acționează ca acorduri în care termenii acordului pot fi preprogramați cu capacitatea de a se auto-executa și de a se auto-aplica. Scopul principal al unui smart contract este de a permite unor părți anonime să facă schimburi comerciale sau să facă afaceri între ele, de obicei pe internet, fără a fi nevoie de un intermediar. Originea și istoria contractelor smart sunt mult mai vechi decât cele bitcoin-ul și datează încă din anii 1990. Termenul "smart contract" a fost folosit pentru prima dată în 1993 de către unul dintre presupușii creatori ai bitcoinului, Nick Szabo, și sa referit la programele de calculator automatizate care pot să execute termenii oricărui contract.

Figura 3.6. Smart Contract [5]

3.2.1. Comparație cu contractele tradiționale

Contractele fizice tradiționale, cum ar fi cele create astăzi de profesioniștii din domeniul juridic, sunt formate dintr-o cantitate vastă de documente tipărite și se bazează în mare măsură pe o organizație centrală pentru executarea lor. Acest tip de aplicare nu este doar foarte consumator de timp, ci este și foarte ambiguu. Dacă lucrurile se rătăcesc, părțile contractante trebuie să se bazeze pe sistemul judiciar public pentru a remedia situația, care poate fi foarte costisitoare și consumatoare de timp.

În schimb, contractele inteligente, create de programatori prin intermediul instrumentelor inteligente de dezvoltare, sunt în întregime digitale, fiind scrise în limbaje de programare, precum C++, Go, Python, Java, Solidity. Acest cod definește regulile și consecințele în același mod ca și un document juridic tradițional, indicând obligațiile, beneficiile și sancțiunile care ar putea fi datorate fiecărei părți în diferite circumstanțe. Codul poate fi apoi executat automat de un sistem de registru distribuit(distributed ledger system).

3.2.2. Modul de funcționare al unui smart contract

Pentru a înțelege modul în care funcționează contractele inteligente, este important mai întâi să facem distincția între codul contractului smart și modul în care se aplică acel cod. Contractele smart poate fi împărțit în două componente separate:

- **Smart Contract Code:** codul stocat, verificat și executat într-un blockchain
- **Smart Legal Contracts:** utilizarea codului unui smart contract ca supliment sau alternativă pentru contractele legale.

Pașii genereali prin care un smart contract functionează în registrul public distribuit:

1. **Codarea.** Deoarece contractele inteligente funcționează ca programe de calculator, este foarte important să facă exact ceea ce doresc părțile implicate. Acest lucru se realizează prin introducerea logicii corespunzătoare în scrierea lor. Codul se va comporta în moduri predefinite și nu va putea fi alterat.
2. **Registrul public.** Codul este apoi criptat și expediat la alte calculatoare printr-o rețea distribuită de registre. Dacă acest lucru se face prin intermediul unui blockchain public fară permisiuni, cum ar fi bitcoin, contractul este trimis în mod similar cu apariția unei noi tranzacții în blockchain.

3. **Execuția.** Un calculator din această rețea de registre distribuite primește codul și il execută, apoi verifica execuția codului de la alte calculatoare, ajungându-se la un acord cu privire la rezultate. Rețeaua va actualiza registrele distribuite pentru a înregistra executarea contractului. În acest tip de sistem, manipularea de către o singură parte nu este posibilă, deoarece controlul asupra executării contractului inteligent va fi realizat în rețeaua distribuită, fiind validat de catre mulți participanți.

3.2.3. Smart Contracts în identitatea digitală

Identitatea digitală auto-suverană(self-sovereign), oferită prin contracte smart [6], asigură accesul indivizilor la un internet în care elementul central este utilizatorul.

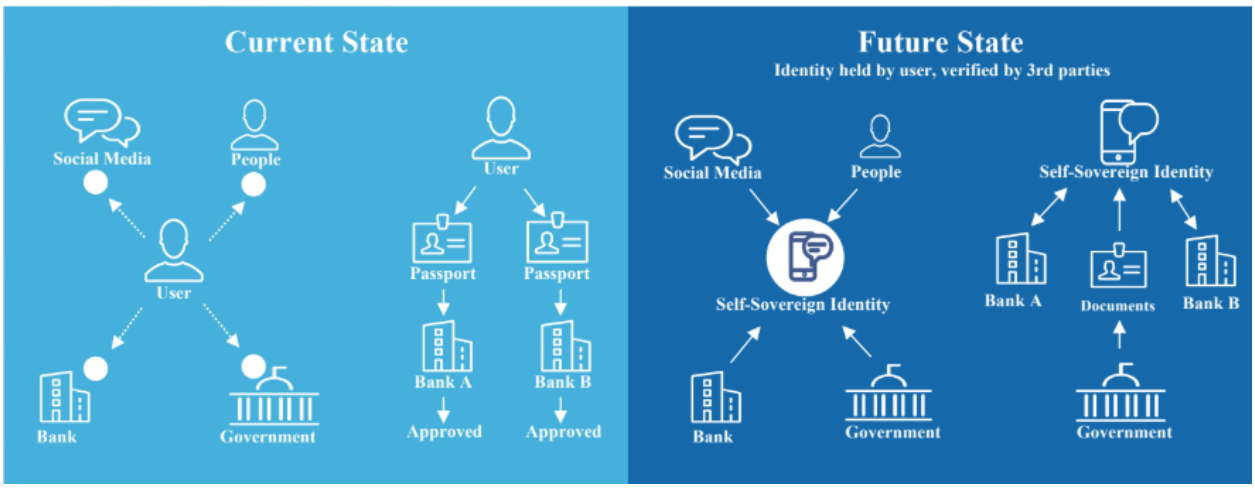

Figura 3.7. Diferențele între sistemele actuale și cele bazate pe smart contracts [6]

În figura 3.7 sunt prezentate diferențele între sistemele actuale care manipulează identitatea digitală a utilizatorului și cele care sunt bazate pe contracte smart.

Problemele sistemelor actuale:

- Procese Know Your Customer(KYC) care sunt costisitoare, consumatoare de timp si incolmplete
- Control limitat asupra eventualelor pierderi de date
- Răspunderea mare pentru protejarea datelor utilizatorilor prezintă un punct sigur de eșec și o țintă pentru hackeri

Beleficiile contractelor smart:

- Indivizii dețin și controlează datele cu caracter personal (de exemplu, vor putea să dezvăluie în siguranță datele personale către diverse părți terțe)
- Părțile terțe(third parties) nu vor trebui să dețină date sensibile pentru a verifica tranzacțiile, reducând răspunderea, facilitând în același timp un proces KYC fără divergențe
- Compatibilitate sporită, elasticitate și cooperare eficientă

Pentru a crea un sistem de manipulare a identității virtuale, bazat pe smart contracts, va fi necesară dezvoltarea de protocoale și standarde care să permită comunicarea și cooperarea eficientă între părțile implicate.

3.3. Identitate digitală

În lucrarea [7], identitatea este definită ca fiind un set de caracteristici și atribute care descriu o entitate, fiind pe urmă utilizată pentru a o identifica în mod unic. Identitatea digitală este reprezentarea unei entități într-un mediul sau context digital, specific. Aceasta este compusa din identificatori unici, atribute descriptive și declarații privind acordul folosirii datelor în sistemul utilizat.

Pentru a înțelege varietatea de platforme de identitate care au existat de-a lungul anilor, unele concepte de bază despre identitate vor vi definite în lista de mai jos:

1. **Identificatori.** Un identificator este o informație utilizată pentru a distinge o persoană sau o entitate în contextul unui sistem. O entitate poate avea mai mulți identificatori asociați cu diferite funcții. Un exemplu de identificator pentru o persoană, poate fi numele, CNP-ul, adresa de email sau numărul de telefon.
2. **Autentificarea.** Autentificarea unei entități se realizează prin demonstrarea unei asociații între utilizator și identitatea acestuia. Sistemul va cere utilizatorului informații secrete sau private cunoscute ca autentificatori. Autentificatorii pot include parole, coduri PIN și chei private. În cazul parolelor, sistemul le va compara cu informațiile secrete pe care le deține deja. Pentru cheile private, acest lucru este diferit, deoarece este nevoie doar de verificarea unei dovezi prin care să se demonstreze că utilizatorul cunoaste valoarea secretă, nefiind necesară transmiterea acesteia.
3. **Autorizarea.** Autorizarea este folosită pentru solicitarea permisiunii de a efectua o acțiune bazată pe un anumit identificator sau atribut. Un sistem va stabili autorizația utilizatorului prin verificarea unui set de reguli stabilite înainte de aprobarea acțiunilor solicitate. Identitatea autentificată a utilizatorului este stocată de sistem, iar odată autorizată, va putea avea acces la resursa dorită.
4. **Atribute.** Atributele sunt fragmente de date referitoare la descrierea unei entități. Valorile pot fi persistente sau temporare. Exemple de atribute ale unei persoane sunt: numele, data nașterii, sexul și adresa.
5. **Verificarea atributelor.** Verificarea atributelor este necesară pentru a stabili corectitudinea unei identități. În multe sisteme de identitate, autentificarea identității este urmată de procesul de verificare, pentru a confirma prezentarea identității corecte. Un exemplu de verificare a unui atribut este cererea unui cod generat de sistem si trimis prin mail utilizatorului, astfel fiind validată adresa de email a acestuia.

3.3.1. Identitatea auto-suverană

Identitatea auto-suverană(self-sovereign identity) este o abordare a indentității, axată pe utilizator, astfel încât fiecare utilizator poate să își administreze propria identitate. Propune ca indivizii să aibă controlul total și autonom asupra identității și datelor lor.

Principiile care încearcă să asigure controlul utilizatorilor asupra indetităților, prezentate în articolul [8], sunt:

- **Existenta.** Utilizatorii trebuie să aibă o prezență independentă în sistem, iar acesta trebuie să fie o prelungire a identității lor personale.
- **Controlul.** Uilizatorii sunt autoritatea supremă asupra identității lor, a modului în care aceasta este utilizată și a modului în care sunt dezvăluite datele.

- **Accesul.** Utilizatorii trebuie să acces usor la propriile date sau resurse, fară ca acestea să fie ascunse sau inaccesibile.
- **Transparența.** Sistemele care gestionează datele și algoritmii care le analizează trebuie să fie transparente, deschise și publice.
- **Persistența.** Identitățile trebuie să aibă capacitatea de a fi de lungă durată sau în permanentă, la discreția utilizatorului.
- **Portabilitatea.** Identitățile nu pot fi deținute de o singură entitate ci trebuie să fie distribuite între servicii.
- **Cooperarea.** Identitățile trebuie să aibă capacitatea de a funcționa între sisteme.
- **Consimțământul.** Schimbul de date ale utilizatorului trebuie făcut cu consimțământul și cu notificarea acestuia.
- **Minimalizarea.** Datele care sunt dezvăluite sau împărtășite ar trebui să fie reduse la minimum atunci când este posibil. Acest lucru este ajutat de introducerea tehnicilor de divulgare selectivă.
- **Protecția.** Drepturile utilizatorului trebuie să fie esențiale pentru sistem, independente și descentralizate.

Acește principii pot părea stricte, dar sunt necesare pentru crearea unui sistem cu adevărat auto-suveran. Regulamentul general privind protecția datelor (GDPR) în Europa, care a intrat în vigoare în luna mai a anului 2018, a elaborat legi cu privire la aceste principii, oferind o deosebită importanță contextului.

3.3.2. Administrarea identității digitale

În lucrarea [9] sunt prezentate 3 categorii de sisteme care gestionează identitatea digitală, disponibile pe scară largă: federalizate(*federated* în engleză), orientate pe utilizator și hibride.

Gestiunea identității **federalizate** permite utilizatorilor să acceseze mai multe servicii pe baza unei singure autentificări. În mod conceptual, acest sistem este format dintr-un grup de organizații de încredere, care împartășesc informații despre identitatea utilizatorilor, permițându-le acestora accesul la resursele lor. Utilizatorul se înscrie o singură dată pentru a accesa toate serviciile oferite de diferiți parteneri. Arhitectura identitații federalizate (FIA) constă în esență într-un furnizor de identitate (IdP) și un furnizor de servicii (SP). IdP-ul gestionează identitatea utilizatorului și efectuează procesul de autentificare pentru a valida identitatea acestuia. SP-ul oferă unul sau mai multe servicii utilizatorilor din cadrul federației.

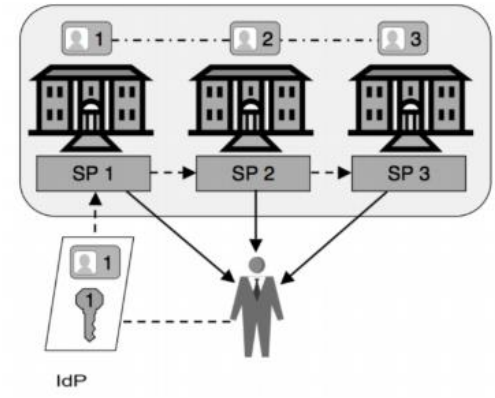

Figura 3.8. Arhitectura identității federalizate [9]

În acest sistemele **orientate pe utilizatori**, utilizatori pot alege ce informații din identitățile lor să fie folosite pentru fiecare aplicație. Acest sistem permite utilizatorilor să stocheze mai multi identificatori și credențiale pentru diferiți furnizori de servicii, într-un singur dispozitiv hardware, care ar putea fi un card inteligent(smart card) sau un dispozitiv personal portabil(de exemplu telefon mobil). Acest model este diferit față de cel federalizat deoarece se axează mai mult pe utilizator și nu pe întreprinderi sau organizații. O altă caracteristică particulară a acestui sistem este identitatea bazată pe atribute. Această abordare vizează rezolvarea problemelor legate de securitate și confidențialitate prin folosirea unei tehnici de acreditare bazată pe atribute (ABC). Tehnologia ABC are o viziune diferită asupra identității și autorizației permițând ca atributele să fie emise și stocate de către persoana vizată (individul). Mai mult decât atât, doar subsetul relevant al acestor atribute trebuie să fie folosit într-o instanță de verificare și de autorizare. Individul nu își poate schimba valorile atributelor de identificare, acest lucru oferind siguranță sistemelor care utilizează tehnologia ABC pentru luarea deciziilor de acces.

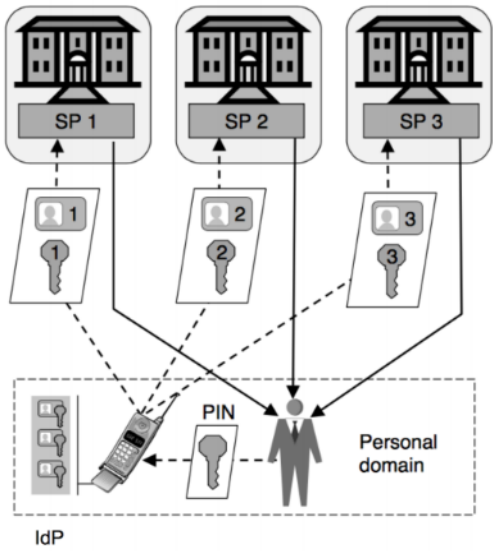

Figura 3.9. Arhitectura identității orientate pe utilizator [9]

Sitemele **hibride** oferă o alternativă atunci când atât abordările federalizate, cât și cele orientate pe utilizatori nu fac față cu ușurință în anumite circumstanțe. De exemplu, în scenariile de asistență medicală, procesele de schimb a atributelor nu pot fi complet orientate spre utilizator, deoarece în caz de accidente critice, utilizatorii nu își pot da consimțământul. Pe de altă parte, modelele federalizate ridică semne de întrebare privind confidențialitatea, deoarece înregistrările medicale pot fi disponibile fiecărei entități din cadrul cercului de încredere, chiar dacă nu există situații de urgență. Prin urmare, modelul hibrid propus permite utilizatorilor să își configureze și să urmărească accesul la înregistrările lor medicale, în timp ce furnizorii de identitate stochează și gestionează credențialele utilizatorilor.

3.3.3. Problemele actuale

Majoritatea tehnologiilor de management al identității suferă de o serie de probleme cauzate în principal de abordarea arhitecturală veche și de lipsa de caracteristici de securitate și de confidențialitate în tehnologiile actuale. Cele mai grave probleme regăsite în sistemele actuale, descrise în lucrarea [9] sunt:

- **Identificatorii globali.** Multe sisteme utilizează identificatori globali pentru a identifica utilizatorii, cum ar fi adrese de email, numere de telefon, CNP-uri. Identificatorii globali permit site-urilor să coreleze informațiile despre utilizatori, obținând pe urmă mai multe informații decât a fost permis inițial.
- **Stații de lucru nesigure.** Stația de lucru tipică folosită de un utilizator pentru acces la internet nu este un mediu sigur. Virușii și alte tipuri de malware pot infecta cu ușurință stația de lucru obținând controlul asupra tuturor activităților utilizatorului. În timp ce stația de lucru se află sub controlul malware-ului, activitatea utilizatorului poate fi urmărită cu ușurință, parolele introduse pot fi observate și chiar și atacuri complexe de tip "man-in-the-middle" pot fi instalate și folosite împotriva unor mecanisme puternice de autentificare.
- **Efectul Honeypot.** Centralizarea informațiilor personale poate fi foarte convenabilă din perspectiva gestionării datelor, dar un astfel de “depozit” creează o țintă foarte atractivă pentru atacatori. Efectul este același indiferent dacă informația este stocată pe servere guvernamentale, găzduită de furnizorii de identitate pe internet sau este păstrată pe stația de lucru a utilizatorului.

3.4. Sisteme similare

În cadrul acestei secțiuni se vor prezenta sisteme similare cu proiectul propus, aceste sisteme administrând identitatea digitala cu ajutorul blockchain-ului.

3.4.1. ShoCard

ShoCard [10] permite utilizatorilor și întreprinderilor să-și administreze identitatea într-un mod sigur și verificat, astfel încât orice tranzacție (fie că este vorba de conectare, de împărtășirea informațiilor personale sau de încheierea unor tranzacții financiare), poate fi realizată rapid și fără probleme.

Identitatea digitală creată este ușor de utilizat, fiind construită pe blockchain-ul public, cea ce înseamnă că întreprinderile nu dețin datele utilizatorului. Identitatea unui utilizator este criptată, și apoi scrisă în blockchain, unde poate fi apelată atunci când este necesar. Utilizatorii, oferă băncilor acces temporar la partea privată a înregistrărilor din blockchain, pentru a verifica identitatea. Odată verificată, banca își creează un registru propriu care poate fi consultat în viitor pentru a determina dacă o anumită persoană este aceea care pretinde a fi.

3.4.2. uPort

uPort [11] [12] este un framework open source pentru administrarea identității digitale care dorește să ofere "identitate descentralizată pentru toți". Principalul său caz de utilizare este administrarea identității (IdM) pentru aplicațiile descentralizate (DApps) pe platforma Ethereum și pentru aplicațiile centralizate tradiționale, cum ar fi e-mail sau serviciile bancare. uPort a fost dezvoltat de Consensys folosind blockchain-ul Ethereum. Acest framework constă din trei componente principale: contracte smart, biblioteci pentru dezvoltatori și o aplicație mobilă.

Identitatea uPort este compusă din două template-uri de contracte smart: controller și proxy. Pentru crearea unei noi identități, aplicația mobilă a utilizatorului uPort creează o nouă pereche de chei asimetrice, apoi trimite o tranzacție către blockchain-ul Ethereum pentru instanțierea unui contract de tip *Controller* cu o referință la cheia publică nou creată. Apoi, se instanțiază un nou contract *Proxy* care conține o referință la adresa contractului *Controller* creat

anterior. Doar contractul *Controller* poate invoca funcțiile contractului *Proxy*. Adresa proxy-ului formează identificatorul uPort unic (uPortID) al unui utilizator. Un utilizator este liber să creeze mai multe uPortID-uri care nu sunt dependente unele de altele.

În figura 3.10 sunt prezentate domeniile în care poate fi folosit framework-ul uPort.

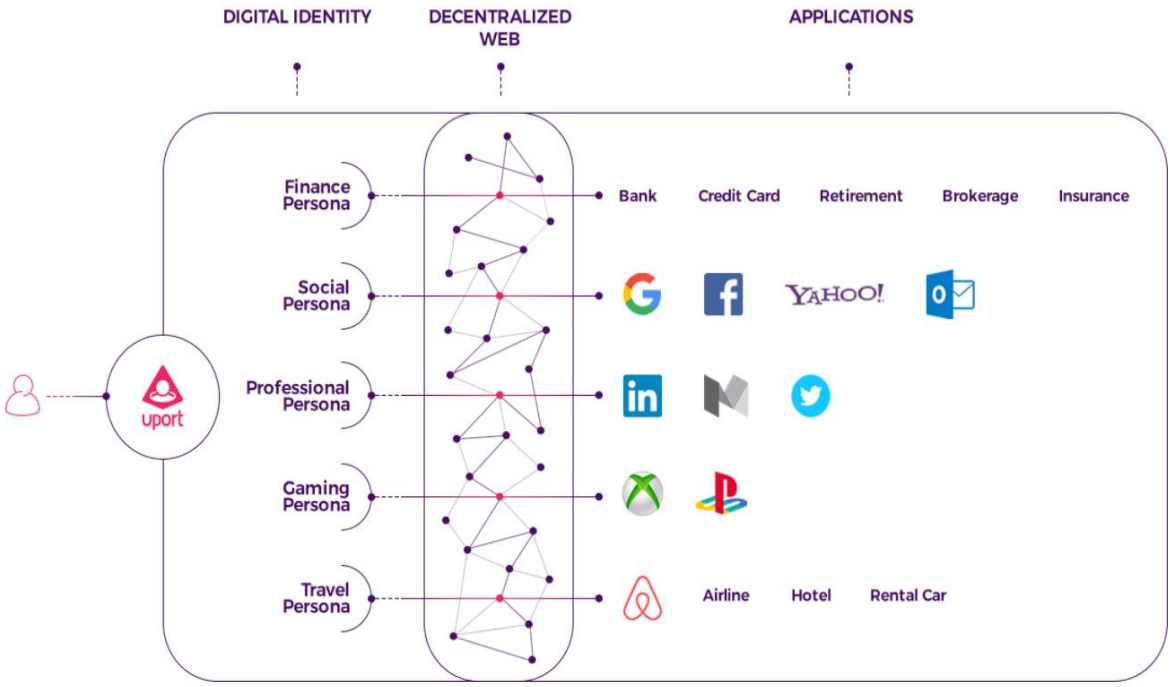

Figura 3.10. Domenile de aplicare al framework-ului uPort[4]

3.4.3. Bitnation

Bitnation [13] este o platformă de guvernare bazată pe blockchain care încearcă să stabilească conceptul de cetățenie mondială prin înregistrarea identității pe blockchain. Scopul său este acela de a furniza aceleași servicii pe care le oferă guvernele, dar într-un mod descentralizat și voluntar. Orice individ din întreaga lume poate deveni cetățean Bitnation prin semnarea constituției.

Programul de reacție în situații de urgență a refugiaților Bitnation (BRER) oferă un sistem de identificare denumit BlockHain Emergency ID (BE-ID). BE-ID permite unei persoane să primească un ID înregistrat în blockchain pentru cei care nu pot obține alte documente de identificare. Cu acest ID, oamenii vor putea beneficia de asistență socială și servicii financiare.

3.4.4. UniquID

UniquID [14] este un startup care urmărește să asigure managementul identității digitale și accesului la resurse utilizând informațiile biometrice. UniquID permite autentificarea

[4] https://media.consensys.net/state-change-36-the-uport-developer-alpha-2016-in-review-49a52be47121

dispozitivelor, a serviciilor cloud și a persoanelor. Oferă o gestionare sigură a identității, integrată cu amprentele digitale și alte biometrici pe dispozitive personale.

3.4.5. Cryptid

Cryptid [15] este un startup care dezvoltă o soluție prin care se elimină posibilitatea identificărilor false, adăugând factori de identificare și criptare. Cryptid preia datele utilizatorului dintr-un formular și le împachetează într-un format compact care poate fi citit de sistem, generând datele de identificare Cryptid. Toate datele sunt criptate cu o parola furnizată de utilizator, după care sunt transferate definitiv în blockchain. Clientul primește apoi un număr unic de identificare care indică informațiile din blockchain. Acest cod poate fi stocat pe aproape orice, de la dungi magnetice la coduri QR.

Capitolul 4. Analiză şi Fundamentare Teoretică

În cadrul acestui capitol, se va descrie arhitectura conceptuală a sistemului implementat și se vor prezenta tehnologiile utilizate și motivele utilizării acestora în contextul dat. Pe urma se vor prezenta cerințele funcționale și non-funcționale ale sistemului și se va defini o serie de cazuri de utilizare ce se propun a fi implementate în acest sistem.

4.1. Arhitectura conceptuală

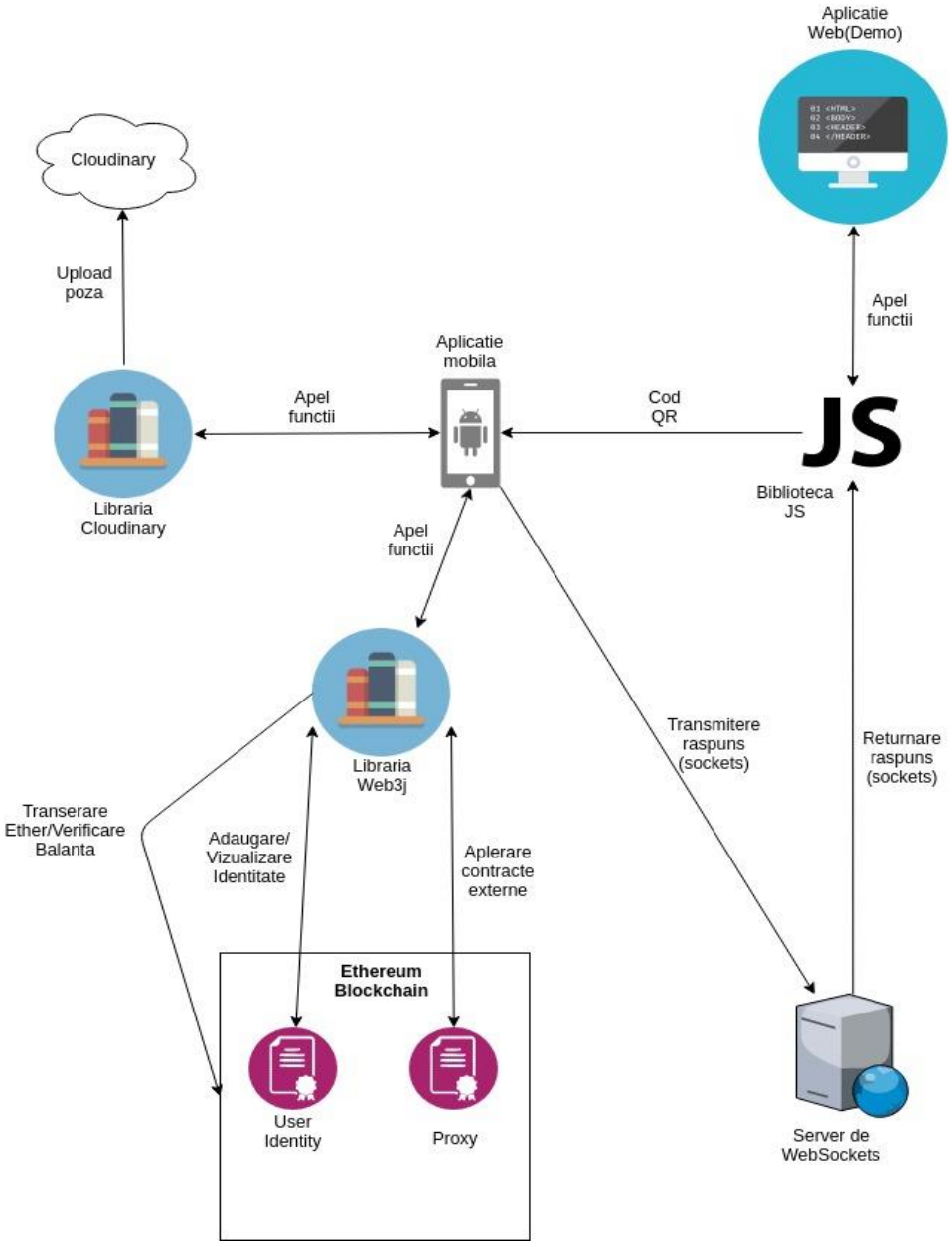

Figura 4.1. Arhitectura conceptuală a sistemului

În figura 4.1 este ilustrată arhitectura conceptuală a sistemului. După cum se poate observa din schemă, sistemul este format din 4 componente principale:

1. Aplicația mobilă
2. Biblioteca de JavaScript
3. Contractele Smart din Blockchain
4. Serverul de WebSockets

Aplicatia mobilă poate fi gazduită de orice telefon cu sistem de operare Android, cu o versiune egala sau mai mare decat 7.0 (Android Nougat). Majoritatea funcțiilor sistemului sunt integrate în această aplicație.

Biblioteca de JavaScript este salvată în baza de date a administratorului de pachete **npm** și poate fi folosită de orice aplicatie web scrisă în limbajul JavaScript, care utilizează acest administrator de pachete. Acească bibliotecă va permite operatorilor de aplicații web să poată integra funcționalitățile sistemului descris în aplicațiile lor.

A treia componentă principală a sistemului, **Contractele Smart** sunt gazduite de **Ethereum Blockchain,** iar funcțiile acestora pot fi apelate din orice mediu de programare prin biblioteci sau protocoale specifice. Contractele smart din blockchain stochează identitatea utilizatorilor și permit interacțiunea acestora cu alte contracte smart din blockchain-ul Ethereum.

Ultima componentă și anume **Serverul de WebSockets** este gazduit de platforma **Heroku**, o platforma care permite stocarea și rularea aplicatiilor web în cloud. Rolul serverului este de a realiza procesul de comunicare între aplicația mobilă și biblioteca de JavaScript aferentă sistemului, prin protocolul WebSockets.

4.2. Perspectiva Tehnologică

În aceast subcapitol se vor prezenta tehnologiile folosite în implementare precum și motivul alegerii acestor tehnologii în realizarea sistemului propus.

4.2.1. Android

Android [16] este o platformă software și un sistem de operare pentru dispozitive și telefoane mobile bazată pe nucleul Linux, dezvoltată inițial de compania Google, iar mai târziu de consorțiul comercial Open Handset Alliance. Android permite dezvoltatorilor să scrie cod în limbajul Java, controlând dispozitivul prin intermediul bibliotecilor Java dezvoltate de Google.

Caracteristici:

- Platforma este adaptabilă la configurații mai mari, VGA, biblioteci grafice 2D, biblioteci grafice 3D bazate pe specificația OpenGL ES 1.0 și configurații tradiționale smartphone.
- Software-ul de baze de date SQLite este utilizat în scopul stocării datelor.
- Android suportă tehnologii de conectivitate incluzând GSM/EDGE, 3G, 4G, CDMA, EV-DO, UMTS, Bluetooth și Wi-Fi.
- SMS și MMS sunt formele de mesagerie instant disponibile, inclusiv conversații de mesaje text.
- Navigatorul de web disponibil în Android este bazat pe platforma de aplicații open source WebKit.
- Android acceptă următoarele formate media audio/video/imagine: MPEG-4, H.264, MP3, AAC, OGG, AMR, JPEG, PNG, GIF.

- Android poate utiliza camere video/foto, touchscreen, GPS, accelerometru, și grafică accelerată 3D.
- Android are suport nativ pentru multi-touch

Android Studio [17] este mediul de dezvoltare integrat (IDE) oficial pentru sistemul de operare Android, construit pe software-ul IntelliJ IDEA al lui JetBrains și proiectat special pentru dezvoltarea aplicatiilor Android native în limbajul de programare JAVA.

Urmatoarele caracteristici motivează alegerea facută în detrimentul altui sistem de operare pentru dispozitive mobile:

- Editor de cod inteligent și editor vizual pentru desigul aplicatiei
- Sistem flexibil de constructie: cu ajutorul sistemului automat de construcție *Gradle*, se pot adauga rapid biblioteci externe în proiectul dezvoltat, iar în plus se pot genera versiuni multiple ale aceluiași proiect pentru diferite dispozitive mobile.
- Emulator rapid pentru testare, plus optiunea de rulare instantă pe dispozitiv fizic.
- Design responsive pentu diferite tipuri de ecrane.

4.2.2. Ethereum Blockchain

Ethereum [18] este o platformă bockchain distribuită, caracterizată prin funcționalitatea contractelor de tip smart(smart contracts). Ethereum oferă o mașină virtuală(Ethereum Virtual Machine) care poate să execute scripturi utilizând o retea publică de noduri. De asemenea Ethereum oferă o moneda viruala numită „ether” care poate fi transferată între conturi. „Gas-ul” este un mecanism intern al tranzacțiilor pentru alocarea de resurse rețelei si evitarea spam-urilor.

Contractele Smart sunt protocoale destinate să faciliteze, să verifice sau să impună negocierea sau executarea unui contract. Aceste contracte sunt niste programe scrise în limbajul Solidity(un limbaj de programare similar cu C și JavaScript), aflate la o anumită adresă în blockchain. Din punctul de vederea al limbajului de programare Solidity, un smart contract este o colecție de cod(functii/metode ale programului) și de date(starea programului). Un exemplu de smart contract este urmatorul:

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) {
        storedData = x;
    }

    function get() constant returns (uint) {
        return storedData;
    }
}
```

Figura 4.2. Exemplu de Contract Smart

Prima linie spune compilatorului că programul este scris pentru limbajul Solidity, versiunea 0.4.0 sau orice versiune mai mare decât atât. Linia uint storedData; declară o variabilă de stare de tipul uint(unsigned integer de 256 de biți). Prin funcția set putem modifica variabila de stare, iar prin functia get putem afla valoarea ei. Fiecare tranzacție care interactionează cu un Smart Contract este taxată cu o anumită cantitate de „gas". Creatorul tranzacției stabileste „gas price", iar acestă trebuie să platească numarul de „gas" necesar execuției tranzacției, înmultit cu „gas price".

Cele două Contracte Smart definite pentru sistemul propus sunt *UserIdentity* care stochează identitatea user-ului și contractul *Proxy*, care permite comunicarea cu alte Contracte Smart externe.

4.2.3. Limbajul Java

Java [19] este un limbaj de programare orientat-obiect, conceput de către James Gosling la Sun Microsystems (acum filială Oracle) la începutul anilor 90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile gen telefon, agenda electronică, etc. În felul acesta se creează o platformă unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat.

Limbajul Java este folosit pentru dezvoltarea aplicației mobile a sistemului propus deoarece, frameword-ul Android Studio impune acest limbaj.

4.2.4. Limbajul JavaScript

JavaScript [20] este un limbaj de programare orientat obiect bazat pe conceptul prototipurilor. Este folosit mai ales pentru introducerea unor funcționalități în paginile web, codul JavaScript din aceste pagini fiind rulat de către browser. Limbajul este binecunoscut pentru folosirea sa în construirea siturilor web. A fost dezvoltat inițial de către Brendan Eich de la Netscape Communications Corporation sub numele de Mocha, apoi LiveScript, și denumit în final JavaScript.

Avantajele limbajului JavaScript sunt:

- Mai puțină interacțiune din partea serverului
- Feedback imediat pentru vizitatori
- Interactivitate sporită
- Interfețe mai bogate

Limbajul JavaScrit a fost ales în realizarea bibliotecii JS a sistemului și pentru dezvoltarea serverului de WebSockets deoarece este singurul limbaj client-side pentru aplicațiile web.

4.2.5. Web3j

Web3j [21] este o bibliotecă modulară pentru JAVA si Android care permite lucrul și interacțiunea cu nodurile rețelei din Ethereum Blockchain. Scopul principal al folosirii acestei biblioteci în aplicația mobilă este de a generarea wrappere(clase in JAVA) pentru Contractele Smart. Acest lucru simplifică comunicarea cu Contractele Smart, apelul unei funcții dintr-un contract realizându-se printr-un apel al unei metode dintr-o clasă generată. Totodată, **web3j** este utilă pentru crearea și utilizarea portofelelor virtuale(transferare ether, interogare balanță).

4.2.6. Node.js

Node.js [22] este o platformă de server open source gratuită ce poate rula pe diferite sisteme de operare (Windows, Linux, Unix, Mac OS X, etc) și care permite rularea de JavaScript pe server. Node.js are o arhitectură bazată pe evenimente, capabilă de input/output asincron. Această achitectură vizează optimizarea capacității și a scalabilității aplicațiilor web cu multe operațiuni de input/output, precum și a aplicațiilor web care necesită comunicare în timp real (ex. Jocuri). Runtime-ul intepretează codul scris în JavaScript folosind engine-ul Javascript V8 de la Google.

Principalele avantaje ale platformei Node.js sunt:

- capacitatea de a rula in paralel mai multe cereri, chiar dacă cea inițială nu a fost finalizată
- simplitatea și vireza rapidă de execuție

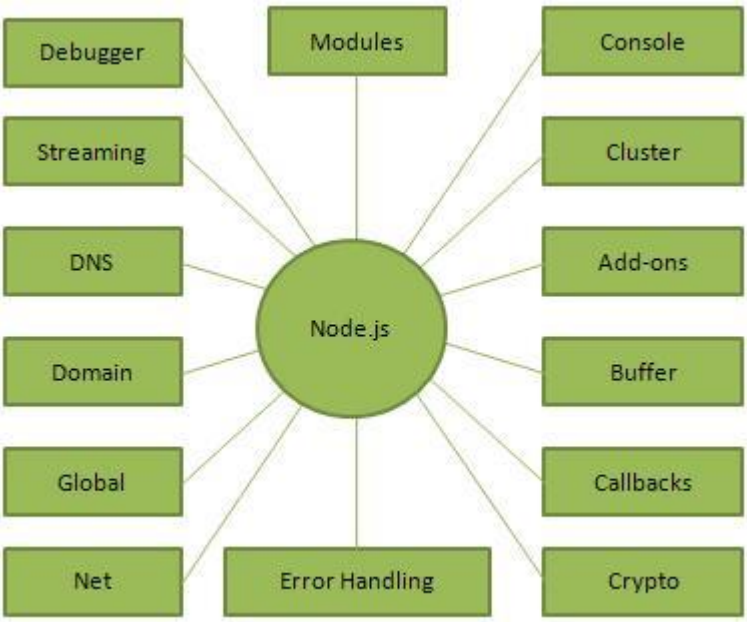

Figura 4.3. Componente NodeJS [22]

Node.js a fost ales în locul altei tehnologii deoarece limbajul folosit este JavaScrip, la fel ca și în dezvoltarea bibliotecii(pe care o pune la dispoziție sistemul creat) și deoarece comunicarea prin WebSockets este mult mai usoara în Node.js decât cu ajutorul altor tehnologii, pe platforma Heroku. Pentru a folosi comunicarea prin WebSockets, pe platforma Heroku, cu un alt framework, de exemplu Ruby On Rails, este necesara adaugarea de addon-uri spleciale.

4.2.7. Npm

Npm [23] este un manager de pachete pentru limbajul de programare JavaScript. Acesta este managerul implicit de pachete pentru platforma Node.js. Se compune dintr-un client pentru linia de comandă și o bază de date online a pachetelor publice și private numită registrul npm. Registrul este accesat prin intermediul clientului, iar pachetele disponibile pot fi căutate prin intermediul site-ului npm.

În sistemul prezentat, npm este folosit pentru publicarea bibliotecii de JavaScript și pentru instalarea dependințelor în serverul de Node.js. O alternativă ar fi administratorul de pachete Yarn, care oferă o securitate mai ridicată, dar nu este la fel de popular ca si npm. Npm a fost ales datorită numarului mare de pachete din care se poate alege și integrarea nativă cu Node.js.

4.2.8. WebSockets

WebSocket este un protocol de comunicații pentru calculatoare, oferind canale de comunicare full duplex printr-o singură conexiune TCP. Protocolul WebSocket a fost standardizat de IETF în 2011. WebSocket este un protocol TCP diferit de la HTTP. Ambele protocoale sunt localizate la nivelul 7 din modelul OSI și, prin urmare, depind de TCP la nivelul 4. Deși sunt diferite, WebSocket este compatibil cu protocolul HTTP. WebSockets permite interacțiunea între un client web (cum ar fi un browser) și un server web, facilitând transferul de date în timp real de la și către server. Protocolul WebSocket este utilizat deoarece permite serverului sa trimită date fără să aibă o cerere de la un client. În figura 4.4 este prezentat modul de funcționare al protocolului WebSocket.

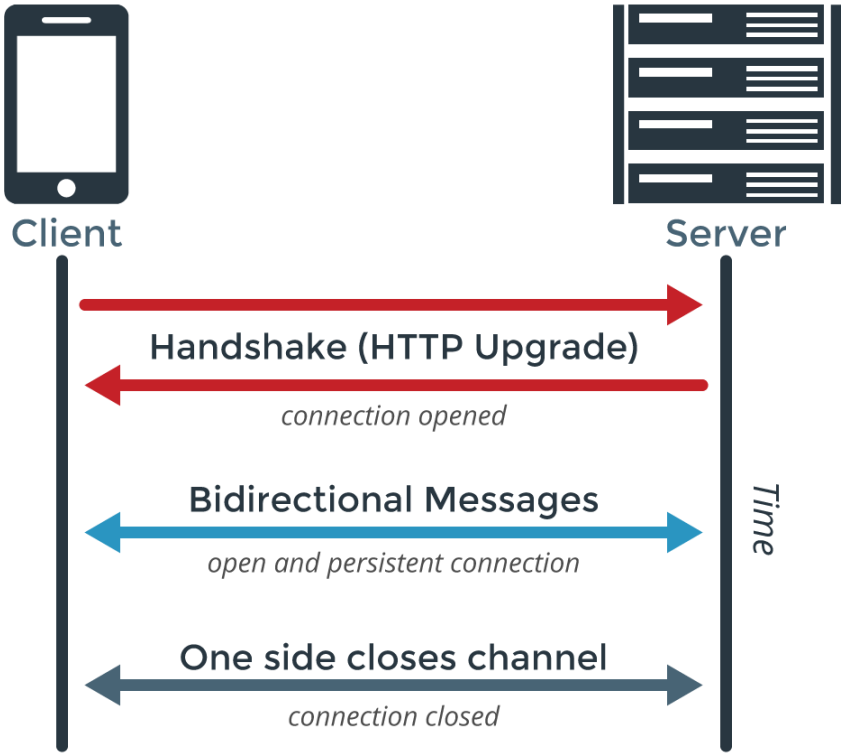

Figura 4.4. Protocolul WebSocket [5]

4.2.9. Cloudinary

Cloudinary [24] furnizează o soluție bazată pe cloud pentru stocare de imagini și conținut video. Cloudinary este o companie cu sediul central în Sunnyvale, California, cu

[5] https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/

un birou în Israel. Acesta permite utilizatorilor să încarce, să stocheze, să gestioneze, să manipuleze și să furnizeze imagini și video pentru site-uri și aplicații, cu scopul de a îmbunătăți performanța. Cloudinary este utilizat de mai mult de 120.000 de dezvoltatori de aplicații web și mobile la peste 3.000 de companii. În sistemul propus, Cloudinary este folosit pentru încarcarea imaginilor de profil ale utilizatorilor. Datorită faptului că este gratuit și deoarece permite o integrare rapidă și simplă într-un proiect de Android Studio, Cloudinary a fost folosit în locul altui serviciu de stocare de imagini(ex. Amazon S3).

4.3. Cerințele sistemului

În cadrul acestui subcapitol se vor prezenta cerințele funcționale si cerințele non-funcționale ale sistemului propus.

4.3.1. Cerințele funcționale

Funcționalitătile sistemului implementat sunt descrise in următoarea listă de cerinte funcționale:

CF1. **Crearea unei identități virtuale și salvarea ei în Ethereum Blockchain.** Identitatea utilizatorului va fi stocată în blockchain cu ajutorul unui Contract Smart. Doar utilizatorul poate avea acces propria identitate.

CF2. **Crearea unui portofel virtual pentru utilizatori.** Este un element obligatoriu în funcționarea sistemului deoarece adresa portofelului identifică unic fiecare utilizator în parte. Acest lucru se va realiza la începutul folosirii aplicației mobile.

CF3. **Editarea indetității virtuale.** Utilizatorii vor putea să-si modifice informațiile din identitatea virtuală. Acest lucru este util în cazul în care un utilizator îți schimbă, de exemplu numărul de telefon sau adresa de mail și dorește actualizarea lor în blockchain.

CF4. **Transferarea de ether.** Fiecare utilizator va putea să trimită sau să primească moneda digitală ether, totodată vor putea să vizualizeze balanța actuală a conturilor proprii. Trasferul se poate realiza prin introducerea manuală a adresei destinatare sau mai simplu, prin scanarea codului QR aferete ei.

CF5. **Vizualizare tranzacții proprii.** Aplicatia mobilă va permite utilizatorilor să vizualizeze o listă cu ultimele tranzacții proprii și să obțină informații despre fiecare tranzacție în parte.

CF6. **Distribuirea identitatii altor aplicatii web.** Utilizatorii vor putea să distribuie informatii din identitatea salvată in blockchain altor aplicatii web, care integrează biblioteca de JS a sistemului. Acest lucru are scopul de a usura inregistrarea sau logarea in alte aplicatii web.

CF7. **Interacțiunea cu Contracte Smart externe.** Cu ajutorul bibliotecii JS pe care o pune la dispoziție sistemul, utilizatorii vor putea interacționa cu Contracte Smart externe, efectuând tranzacții și apeluri către funcțiile acestora.

CF8. **Recuperarea identiății virtuale.** În cazul în care utilizatorul își schimbă telefonul mobil sau fisierul ce conține portofelul virtual este șters, el poate să își recupereze acel fișier și implicit identitatea virtuală prin introducerea cheiei private primite inițial la crearea contului.

4.3.2. Cerințele non-funcționale

Constrângeri ale serviciilor și funcțiilor pe care le pune la dispoziție sistemul, vor fi prezentate în lista de mai jos:

CNF1. **Securitatea.** Datorită faptului că datele din care este formată identitatea utilizatorului sunt salvate în blockchain, acestea nu pot fi accesate decât de către proprietarul acestora, astfel datele nu pot fi alterate sau furate de către persoane sau aplicații neautorizate. În plus portofelul virtual al unui utilizator este salvat în memoria externă a propriului telefon, fiind protejat de către o parolă setată de către utilizator.

CNF2. **Scalabilitatea.** Deoarece nu putem anticipa numarul de utilizatori care vor utiliza sistemul, ar trebuii ca funcționalitătile sistemului să nu depindă de acest fapt. Acest lucru se întamplă, deoarece sistemul nu depinde de o bază de date clasică, ci folosește blockchain-ul ca mediu de stocare, indetitățile utilizatorilor fiind indexate de către propria lor adresă a portofelului virtual, astfel timpul de acces nu este influențat.

CNF3. **Utilizabilitatea.** Pentru o buna utilizare a sistemului, acesta trebuie să aiba o interfață prietenoasă și intuitivă, care să se adapteze pe orice ecran al telefoanelor mobile. De asemenea, utilizatorul trebuie atenționat în timpul utilizării sistemului, prin mesaje și notificări.

CNF4. **Portabilitatea.** Aplicatia mobilă poate rula pe orice telefon cu sistemul de operare Android, mai mare sau egala cu versiunea 7.0. Aplicațiile web ce integrează biblioteca JS pot fi utilizate în orice sistem de operare care suportă managerul de pachete npm.

CNF5. **Performanță.** Timpul de acces la identitățile utilizatorilor va fi același indiferent de numarul acestora, deoarece datele sunt indexate in blockchain. Prin utilizarea protocolului WebSocket, transmisia datelor se efectuează într-un timp scurt, dar acesta depinde de rețeaua la care este conectat telefonul mobil.

CNF6. **Flexibilitate.** Datorită arhitecturii alese, modificarea unor funcționalități, ștergerea sau adăugarea altor noi, reprezintă o operație simplă. Deoarece Contractele Smart nu pot fi modificate după ce acestea au fost create, se vor crea contracte noi care vor comunica cu cele vechi, permițând astfel modificarea funcționalităților.

4.4. Cazuri de utilizare

4.4.1. Actori

În această secțiune se vor definii actorii principali ai sistemului implementat și se vor descrie rolurile și responsabilitățile acestora.

Sistemul propus are doi actori:

1. Utilizatorul
2. Administratorul unei aplicații web externe

Actorii sistemului propus sunt prezentați în Tabelul 4.1.

Tabel 4.1. Descrierea actorilor și a responsabilităților acestora

Nume	Descriere	Responsabilități
Utilizator	Simplu utilizator al aplicației mobile	Navigare în aplicația mobilă, utilizarea serviciilor oferite de sistem
Administrator aplicație web externă	Administratorul unei aplicații web ce implementează biblioteca de JS oferită de sistem	Crearea unei aplicații web ce implementează bilbioteca de JS oferită de sistem, apelarea funcțiilor acestei biblioteci cu scopul utilizării serviciilor pe care le pune la dispoziție sistemul

4.4.2. Descriere

Figura 4.3 reprezintă diagrama UML a cazurilor de utilizare. În acească figură se pot observa actorii sistemului și modul în care aceștia înteracționează cu cazurile de utilizare.

În subsecțiunile ce urmează se descriu mai in detaliu, fiecare caz de utilizare în parte, specificând actorul principal, precondițiile, postcondițiile, scenariul de succes și scenariile alternative, în caz de eroare.

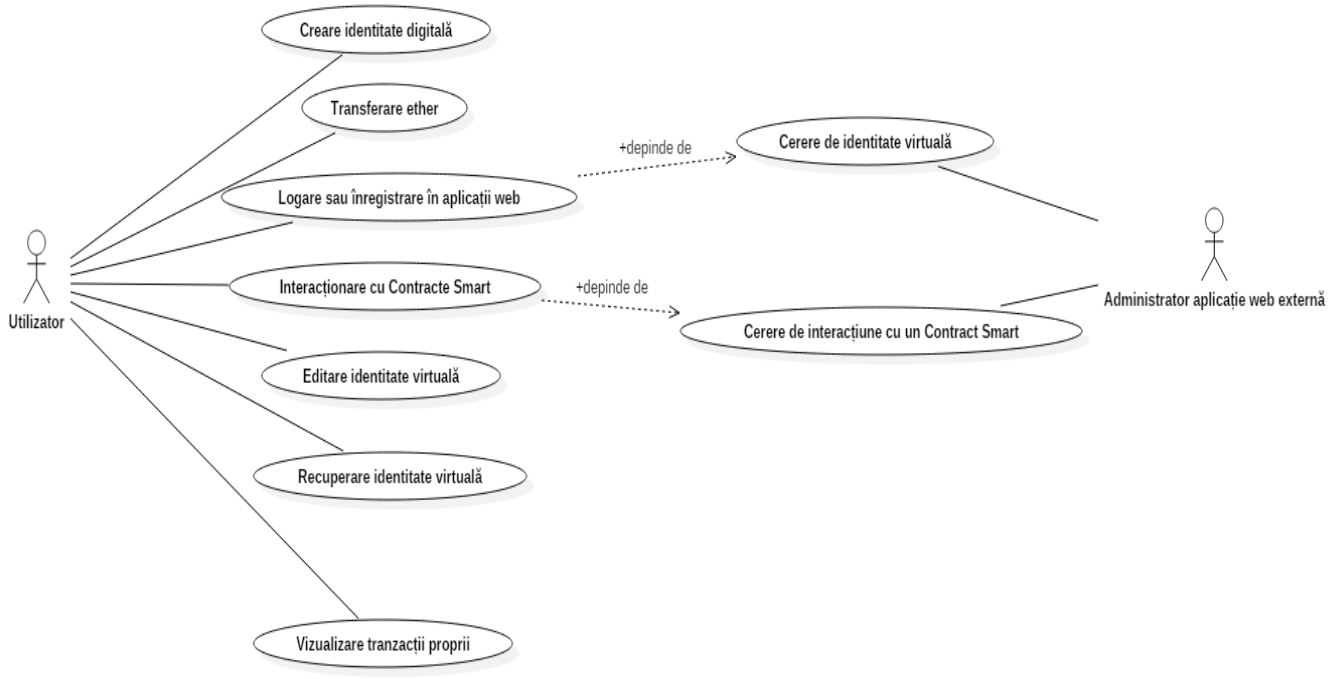

Figura 4.5. Diagrama UML a cazurilor de utilizare

4.4.2.1. Cazul de utilizare 1

Numele cazului de utilizare: Creare identitate virtuală
Actor principal: Utilizatorul
Precondiții:

1. Utilizatorul este pe pagina corespunzătoare în aplicația mobilă
2. Utilizatorul trebuie sa aibă în contul său suficient ether pentru efectuarea tranzacției

Postcondiții:

1. Identitatea virtuală a fost creată
2. Utilizatorul poate interacționa cu informațiile

Scenariul principal (de succes):

1. Utilizatorul introduce datele referitoare la propria sa identitate
2. Utilizatorul alege din telefon o imagine care va reprezenta poza sa de profil
3. Aplicatia mobila va incarca imaginea pe platforma Cloudinary si va returna url-ul aplicatiei
4. Sistemul salvază identitatea în blockchain și redirecționează utilizatorul pe ecranul principal

Scenarii alternative:

1. Utilizatorul introduce date invalide sau incomplete
    a. Aplicația va semnala erorile printr-un mesaj
    b. Utilizatorului i se va cere să reintroducă informațiile lipsă sau eronate
2. Utilizatorul nu are în cont suficient ether
    a. Aplicatia va afisa o alertă prin care i se cere utilizatorului să își alimenteze contul
3. Sistemul întâmpină o eroare
    a. Aplicația semnalează acest lucru printr-un mesaj

4.4.2.2. Cazul de utilizare 2

Numele cazului de utilizare: Editare identitate virtuală
Actor principal: Utilizatorul
Precondiții:

1. Utilizatorul este pe pagina corespunzătoare în aplicația mobilă
2. Utilizatorul trebuie sa aibă în contul său suficient ether pentru efectuarea tranzacției

Postcondiții:

1. Identitatea virtuală a fost actualizată
2. Utilizatorul poate interacționa cu informațiile

Scenariul principal (de succes):

1. Utilizatorul introduce datele pe care dorește să le modifice la propria sa identitate
2. Utilizatorul alege din telefon o imagine dacă dorește sa își modifice poza sa de profil
3. Aplicatia mobila va incarca imaginea pe platforma Cloudinary si va returna url-ul aplicatiei, doar daca utilizatorul dorește să o schimbe
4. Sistemul actualizează identitatea în blockchain și redirecționează utilizatorul pe ecranul principal

Scenarii alternative:
1. Utilizatorul introduce date invalide sau incomplete
    a. Aplicația va semnala erorile printr-un mesaj
    b. Utilizatorului i se va cere să reintroducă informațiile lipsă sau eronate
2. Utilizatorul nu are în cont suficient ether
    a. Aplicatia va afisa o alertă prin care i se cere utilizatorului să își alimenteze contul
3. Sistemul întâmpină o eroare
    a. Aplicația semnalează acest lucru printr-un mesaj

4.4.2.3. Cazul de utilizare 3

Numele cazului de utilizare: Recuperare identitate virtuală
Actor principal: Utilizatorul
Precondiții:
1. Utilizatorul este pe pagina corespunzătoare în aplicația mobilă

Postcondiții:
1. Identitatea utilizatorului a fost recuperată
2. Utilizatorul poate interacționa cu informațiile

Scenariul principal (de succes):
1. Utilizatorul introduce cheia privată aferetă portofelului său virtual
2. Sistemul crează fisierul pentru portofelul virtual și recuperează identitatea din blockchain
3. Sistemul va redirecționa utilizatorul pe ecranul principal

Scenarii alternative:
1. Utilizatorul introduce introduce o cheie privată invalidă
    a. Aplicația va semnala eroarea printr-un mesaj
    b. Utilizatorului i se va cere să reintroducă informațiile lipsă sau eronate
2. Sistemul întâmpină o eroare
    a. Aplicația semnalează acest lucru printr-un mesaj

4.4.2.4. Cazul de utilizare 4

Numele cazului de utilizare: Transferare ether
Actor principal: Utilizatorul
Precondiții:
1. Utilizatorul este pe pagina corespunzătoare în aplicația mobilă
2. Utilizatorul trebuie sa aibă în contul său suficient ether pentru efectuarea tranzacției

Postcondiții:
1. Balanța utilizatorului a fost actualizată
2. Utilizatorul poate vizualiza informații despre tranzacție

Scenariul principal (de succes):
1. Utilizatorul introduce sau scanează adresa la care dorește să transfere ether
2. Utilizatorul introduce suma pe care doreste să o transfere
3. Sistemul va îniția tranzacția
4. După terminarea tranzacției se va afișa un mesaj de succes

Scenarii alternative:

1. Utilizatorul introduce introduce o adresă sau o sumă invalidă
    a. Aplicația va semnala eroarea printr-un mesaj
    b. Utilizatorului i se va cere să reintroducă informațiile lipsă sau eronate
2. Utilizatorul nu are în cont suficient ether
    a. Aplicatia va afisa o alertă prin care i se cere utilizatorului să își alimenteze contul
3. Sistemul întâmpină o eroare
    a. Aplicația semnalează acest lucru printr-un mesaj

4.4.2.5. Cazul de utilizare 5

Numele cazului de utilizare: Logare/Înregistrare intr-o aplicație web
Actor principal: Utilizatorul și Administratorul aplicației web
Precondiții:

1. Utilizatorul este pe pagina corespunzătoare în aplicația web

Postcondiții:

1. Identitatea utilizatorului a fost transferată aplicației web
2. Aplicația web crează o sesiune pentru utilizator
3. Utilizatorul poate interacționa cu resursele din aplicația web

Scenariul principal (de succes):

1. Utilizatorului i se cere să scaneze un cod QR
2. Utilizatorul scanează codul QR și acceptă tansferul de identitate către aplicația web
3. Aplicația primește detele despre identitatea utilizatorului și le procesează

Scenarii alternative:

1. Utilizatorul nu acceptă transferul identității
    a. Aplicația web nu va primii identitatea
    b. Aplicația web va afisa un mesaj
2. Sistemul întâmpină o eroare
    a. Aplicația web semnalează acest lucru printr-un mesaj

4.4.2.6. Cazul de utilizare 6

Numele cazului de utilizare: Interacționare cu un contract smart extern
Actor principal: Utilizatorul și Administratorul aplicației web
Precondiții:

1. Utilizatorul este pe pagina corespunzătoare în aplicația web

Postcondiții:

1. Tranzacția creată a fost finalizată
2. Utilizatorul poate interacționa cu resursele din contractul smart extern

Scenariul principal (de succes):

1. Utilizatorului i se cere să scaneze un cod QR
2. Utilizatorul scanează codul QR și acceptă inițierea tranzacției
3. Aplicația web primește detele despre tranzacție
4. Tranzacția a fost finalizată cu succes
5. Aplicația web notifică utilizatorul cu privire la finalizarea tranzacției

Scenarii alternative:

1. Utilizatorul nu acceptă inițierea tranzacției

    a. Aplicația web va semnala acest lucru printr-un mesaj
2. Tranzacția a fost finalizată, dar nu cu succes
    a. Aplicatia web va afisa o eroare
3. Sistemul întâmpină o eroare
    a. Aplicația semnalează acest lucru printr-un mesaj

4.4.2.7. Cazul de utilizare 7

Numele cazului de utilizare: Vizualizare tranzacții proprii
Actor principal: Utilizatorul
Precondiții:

1. Utilizatorul este pe pagina corespunzătoare în aplicația web

Postcondiții:

1. Tranzacțiile sunt afisate în aplicația mobilă
2. Utilizatorul poate vizualiza informații despre fiecare tranzacție în parte

Scenariul principal (de succes):

1. Utilizatorul alege tab-ul *Home* din meniu sau apasă butonul *Refresh* din ecranul *Home*
2. Aplicația mobilă face un request HTTP pentru a prelua tranzacțiile și le afisează pe ecran

Scenarii alternative:

1. Requestul HTTP eșuează
    a. Nicio tranzacție nu va fi afisată pe ecran
2. Sistemul întampină o eroare
    a. Aplicația semnalează acest lucru printr-un mesaj

Capitolul 5. Proiectare de Detaliu și Implementare

În cadrul acestui capitol, se vor prezenta detaliile despre proiectarea și implementarea sistemului propus. De asemenea, se va evidenția aplicarea tehnologiilor descrise în capitolul 4, precum și conceptele esențiale din domeniul din care se încadrează tema proiectului, prezentate în capitolul 3.

Sistemul final este format din 4 componente principale și anume: aplicația mobilă, contractele smart din blockchain, biblioteca de JavaScript și serverul de WebSockets. În subcapitolele ce urmează, vor fi prezentate toate cele 4 componente ale sistemului propus.

La finalul capitolului se va descrie o aplicație web demonstativă, care folosește un smart contract, pentru testarea funcționalităților sistemului.

5.1. Arhitectura aplicației mobile

Aplicația mobilă este realizată în limbajul Java, fiind un proiect al IDE-ului Android Studio. Clasele proiectului au fost organizate în pachete, în funcție de rolul lor. Astfel au fost definite 7 pachete:

- **Activities**: conține toate activitățile aplicației
- **Fragments**: conține toate fragmentele aplicației
- **Services**: conține toate serviciile aplicației
- **Adapters**: conține clase ce extind *ArrayAdapter*
- **Dialogs**: conție clase pentru pop-up-uri
- **Interfaces**: conține interfețele definite
- **Contracts**: conține clasele mapate pe contractele smart din blockchain

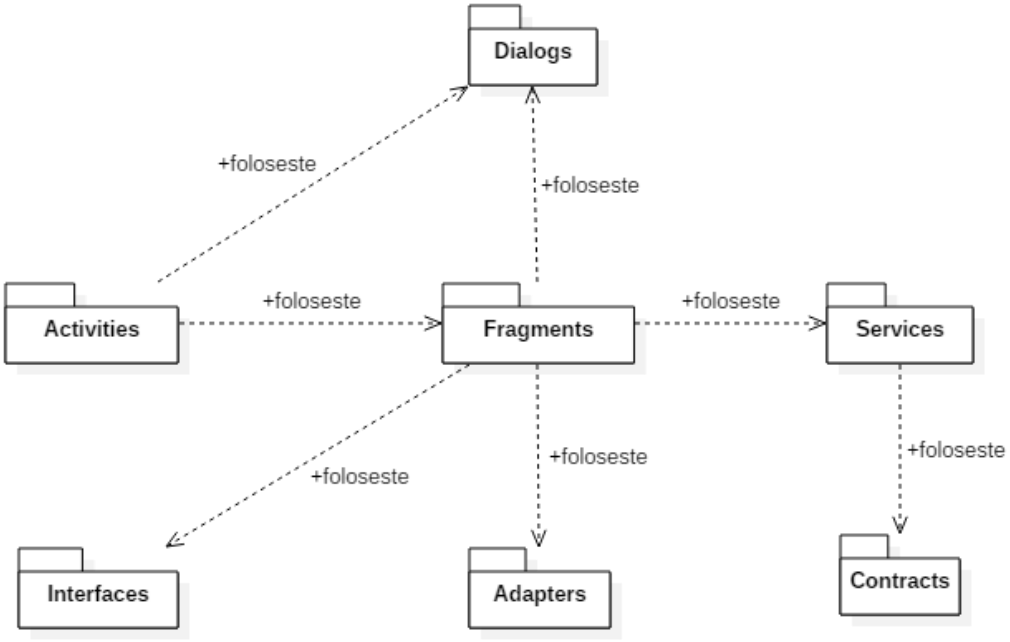

Figura 5.1. Diagrama de pachete a aplicației mobile

În figura 5.1 se poate observa diagrama de pachete a aplicației mobile și modul în care acestea interacționează între ele.

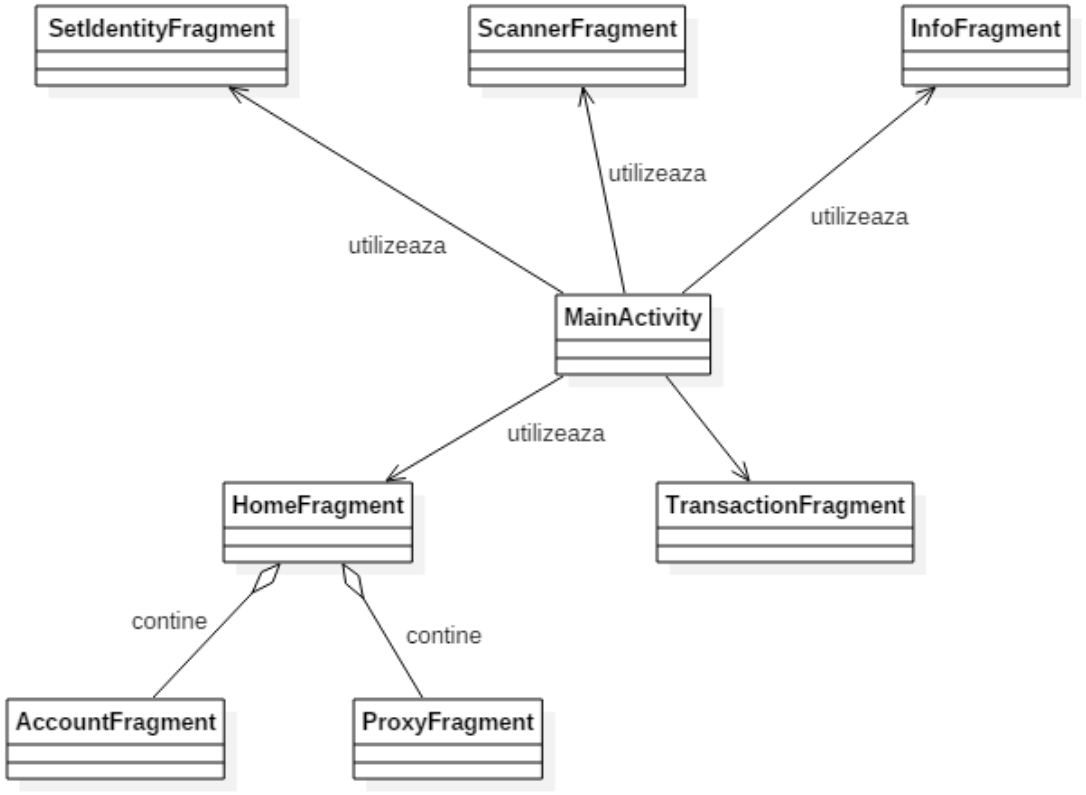

Figura 5.2. Diagrama de clase

În figura 5.2 sunt prezentate cele mai importante clase din aplicația mobilă, precum și relațiile dintre ele, într-un mod minimalist, sub forma unei diagrame de clase.

5.1.1. Descriere generală

Un proiect din Android Studio conține tot ceea ce definește spațiul de lucru pentru o aplicație, de la codul sursă și materiale, la codul de testare și elemente de configurare. Când se începe un proiect nou, Android Studio creează structura necesară executarea acestuia pe un telefon mobil cu sistem de operare Android sau pe emulatorul pe care îl pune la dispoziție Android Studio.

În figura 5.3 se observă structura generală al unui proiect realizat în Android Studio. Acest framework foloseste sistemul Gradle, care este un set de instrumente avansate pentru a automatiza și gestiona procesul de construire, permițând în același timp definirea de configurații personalizate. Gradle este utilizat și pentru adăugarea de biblioteci externe cu scopul de a fi folosite în dezvoltarea proeictului.

Un modul este o colecție de fișiere sursă și fișiere de configurație care permite împărțirea unui proiect în mai multe unități discrete de funcționalitate. Modulul *app* este modulul principal, fiind generat automat la crearea unui nou proiect Android Studio.

Acesta oferă un container pentru codul sursă, fișierele de resurse și setările de nivel ale aplicației(de exemplu, fișierul AndroidManifest).

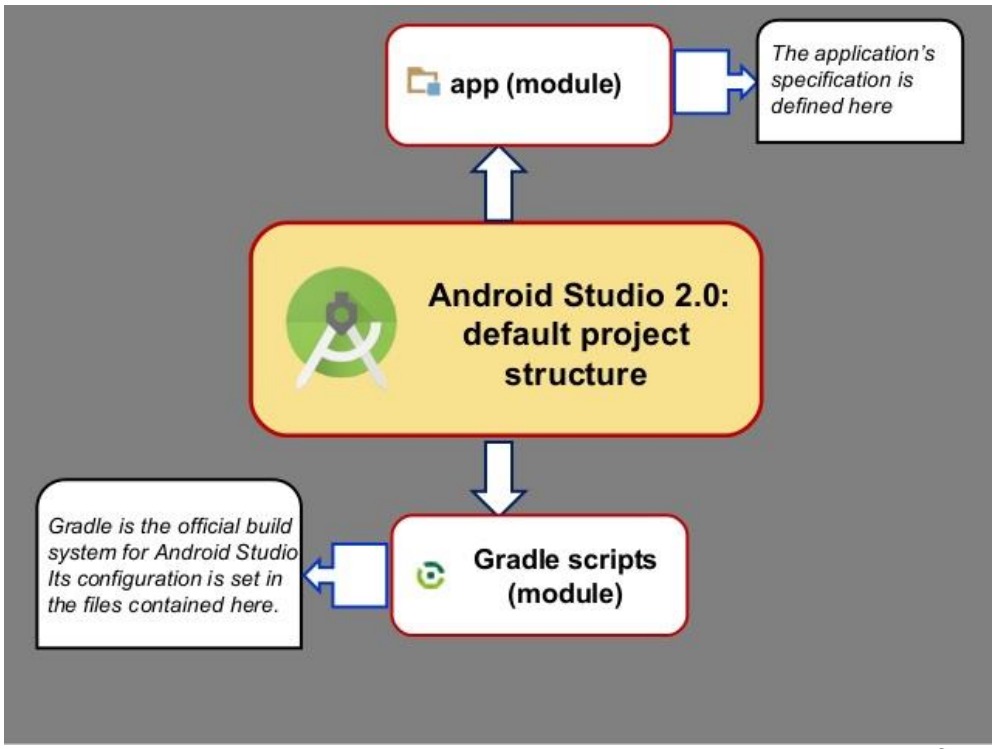

Figura 5.3. Structura generala al unui proiect Android Studio [6]

În directorul *manifests*, al modulului *app* se află fișierul de configurare *AndroidManifest*. Acest fișier descrie informațiile esențiale ale aplicației Android.

Printre multe alte lucruri, fișierul AndroidManiest trebuie să declare următoarele:

- Numele pachetului aplicației, care de obicei corespunde namespace-ului codului. Instrumentele de construire(build) Android îl utilizează pentru a determina locația entităților de cod în procesul de construire.
- Componentele aplicației, care includ toate activitățile, serviciile și furnizorii de conținut. Fiecare componentă trebuie să definească proprietăți de bază, cum ar fi numele clasei sale Java și momentul în care aceasa trebuie să pornească.
- Permisiunile de care are nevoie aplicația pentru a accesa părți protejate ale sistemului sau ale altor aplicații. De asemenea, declară toate permisiunile pe care alte aplicații trebuie să le aibă dacă doresc să acceseze conținut din această aplicație.
- Caracteristicile hardware și software pe care trebuie să le aibe un dipozitiv mobil care dorește să instaleze și să ruleze aplicația.

Directorul *java* al modulului *app* conține codul sursă al proiectului împreună cu module și fișiere de testare al acestuia. Codul sursă este organizat în funcție de dorințele programatorului. Clasele Java care definesc interața utilizator și care interacționează cu acesta sunt numite Activități(Activities). Activitățile din sistem sunt gestionate ca o stivă.

[6] https://pt.slideshare.net/VyaraGeorgieva1/android-studio-20-default-project-structure

Atunci când se pornește o nouă activitate, ea este plasată în partea superioară a stivei și devine activă(în desfășurare). Activitatea anterioară rămâne într-un nivel inferior al stivei și nu va mai reveni în prim plan până când noua activitate nu va fi oprită. Un fragment reprezintă o porțiune a interfeței utilizator dintr-o activitate. Se pot combina mai multe fragmente într-o singură activitate pentru a crea o interfață UI multiplă și se poate reutiliza un fragment în mai multe activități. Un fragment poate fi interpretat ca o secțiune modulară a unei activități, care are propriul ciclu de viață, primește propriile evenimente de intrare și pe care îl putem adăuga sau elimina în timp ce activitatea se desfășoară.

Directorul *res* al modulului *app* conține toate resursele aplicației. Resursele sunt fișierele suplimentare cu conținut static pe care codul Java le utilizează. Aceste resurse pot fi imagini, fișierele bitmap, definițiile layout-urilor, instrucțiunile de animație și multe altele. Fiecare resursă primește un ID, pentru a putea fi identificată atunci când este folosită. Subdirectorul *layout* conține fișiere XML care definesc interfața utilizator. Fiecare fișier layout trebuie să conțină exact un element rădăcină, care trebuie să fie un obiect de tip View sau ViewGroup. După definirea obiectului rădăcină, se pot adăuga obiecte suplimentare sau widget-uri ca elemente copil pentru a construi treptat o ierarhie de vizualizare. În figura 5.4 este exemplificat un fisier layout care conține ca element radacină un obiect de tip *LinearLayout*, iar ca elemente copii un obiect de tip *TextView* și un obiect de tip *Button*. Aceste elemente au și ele atribuite câte un ID, pentru a putea fi folosite în clasele Java.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Figura 5.4. Exemplu de fisier layout[7]

5.1.2. Activități

Aplicația mobilă conține două activități: *StartActivity* și *MainActivity*.

StartActivity este folosită pentru crearea unui cont pentru utilizator și pentru recuperarea de conturi cu ajutorul PrivateKey-ului.

În figura 5.5 se observă interfata grafică a activității de start. Dacă se apasă butoanele de *Create Account* sau *Import Account* vor apărea pop-up-urile prezentate în

[7] https://developer.android.com/guide/topics/ui/declaring-layout

figură. Crearea unui cont de utilizator presupune defapt crearea unei adrese în blockchain-ul Ethereum.

În memoria externa a telefonului se va salva un fisier de tip JSON, ce conține PrivateKey-ul si PublicKey-ul adresei noi create. Acest lucru se realizează cu ajutorul bibliotecii Web3j, prin funcția *WalletUtils.generateLightNewWalletFile* ce primește ca parametrii o parolă setată de utilizator și locul(path-ul) unde va fi salvat fisierul(folderul Wallets), în memoria externă a utilizatorului. După crearea fisierului wallet, utilizatorul va fi redirecționat către MainActivity unde va fi rugat să îți seteze identitatea virtuală. Deoarece setarea identității presupune realizarea unei tranzacții în blockchain, care costă o anumită valoare ether(aproximativ 0.02 ETH), utilizatorul trebuie întâi să își alimenteze contul nou creat. După setarea identității digitale, utilizatorul va putea folosi toate funcționalitățile pe care le pune la dispoziție sistemul.

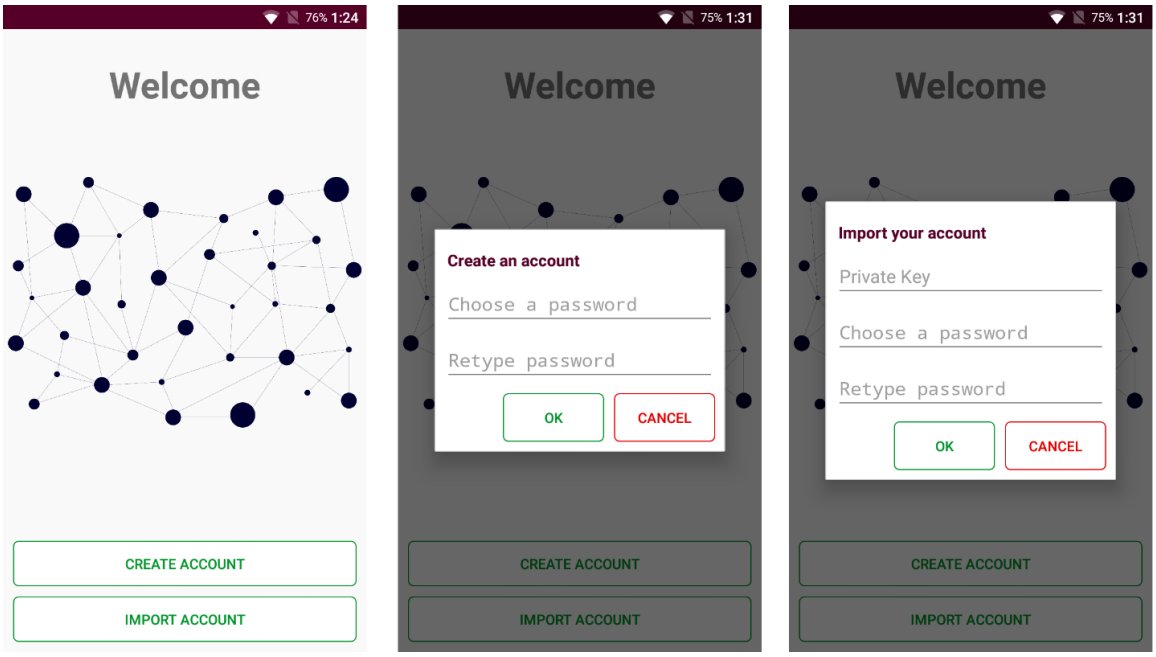

Figura 5.5. StartActivity

În cazul în care utilizatorul își schimbă telefonul, sau sterge fisierul wallet din memoria telefonului, va putea să își recupereze contul dacă a salvat PrivateKey-ul acestuia. Pentru a realiza acest lucru, utilizatorul va apăsa butonul de *Import Account* din StartActivity, va introduce PrivateKey-ul impreună cu o parolă aleasă de el. Prin intermediul librăriei Web3j se va genera un obiect de tip *KeyPair* care conține cheia publică și cheia privată a contului. Apoi cu ajutorul funcției *WalletUtils.generateWalletFile* ce primește ca parametrii parola setată de utilizator, perechea de chei și path-ul spre folderul Wallets, se va crea un fisier JSON la fel ca și la crearea contului nou. Astfel utilizatorul și-a recuperat contul și totodată identitatea.

MainActivity este activitatea principală, fiind pornită la deschiderea aplicației. Această activitate conține un meniu prezentat în figura 5.6 și un container pentru fragmentul ce va fi activ în funcție de opțiunea aleasă din meniu. MainActivity preia din blockchain identitatea utilizatorului și o salvează într-o variabilă de sesiune.

Meniul conține în partea de sus o prezentare a utilizatorului(imagine de profil, nume și email) și cele 5 elemente de meniu:

- Home: fragment corespunzător *Home*
- Send Ether: fragment corespunzător *Transaction*
- Camera: fragment corespunzător *Scanner*
- Settings: fragment corespunzător *SetIdentity*
- Info: fragment corespunzător *Info*

După alegerea unui element din meniu, utilizatorul va fi redirecționat către fragmentul corespunzător alegerii. La pornirea aplicației, utilizatorului i se va cere parola cu care și-a setat contul, precum se poate observa în figura 5.6. După întroducerea corectă a parolei, utilizatorul va fi redirecționat la fragmentul *Home*.

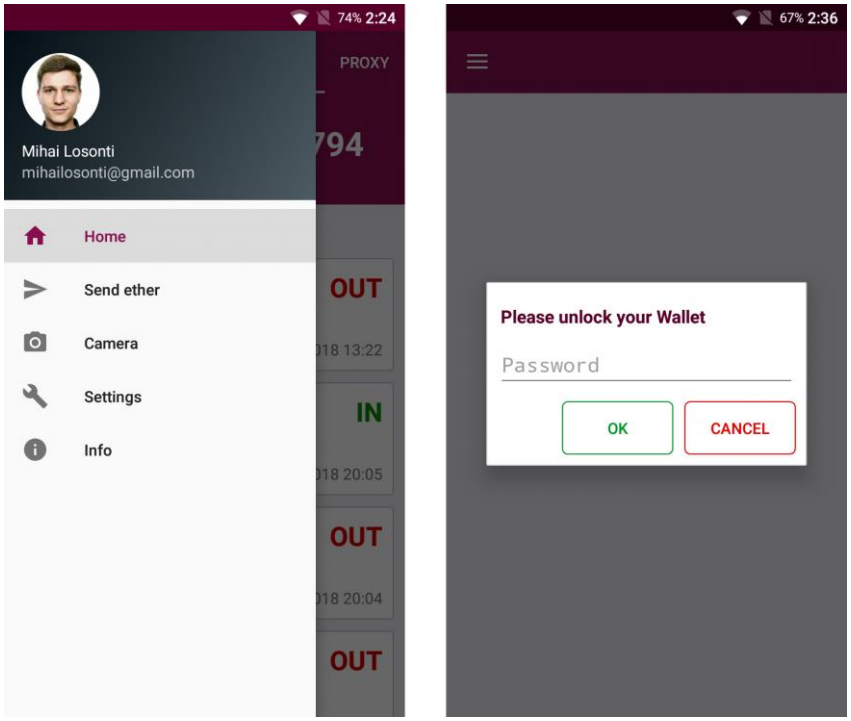

Figura 5.6. Meniul din MainActivity și Pop-up-ul pentru introducerea parolei

5.1.3. Fragmente

Aplicația mobilă conține 7 fragmente, dintre care doar unul este activ la un moment dat. Cele 7 fragmente sunt: HomeFragment, AccountFragment, ProxyFragment, TransactionFragment, ScannerFragment, SetIdentityFragment și InfoFragment. Pentru înlocuirea unui fragment cu altul, se crează un obiect de tip *FragmentTransaction* care adaugă în contrainerul din MainActivity fragmentul dorit. Fiecare fragment are o interfată utilizator proprie, definită printr-un fisier layout.

În subsecțiunile ce urmează se va prezenta fiecare fragment în parte.

5.1.3.1. HomeFragment

HomeFragment este un container de tip swipe care conține AccountFragment și ProxyFragment. Utilizatorul va putea trece de la un fragment la altul printr-un gest de tip

swipe right sau *swipe left*. În HomeFragment se inițializează cele două fragmente, fiind afișat prima data fragmentul Account.

5.1.3.2. AccountFragment

AccoutFragment conține informații despre contul utilizatorului precum balanța curentă și istoricul tranzacțiilor. Balanța curentă este preluată din blockchain cu ajutorul bibliotecii Web3j, prin intermediul funcției *ethGetBalance*, ce primeste ca paramentru adresa contului din blockchain. Pentru a afisa balanța în dolari, aplicația mobilă inițiază un request HTTP la adresa:
https://min-api.cryptocompare.com/data/price?fsym=ETH&tsyms=USD.

Acest serviciu va returna rata actuală dintre dolar și ether. Balanța în dolari va fi produsul dintre rata returnată de serviciu și balanța în ether.

Istoricul tranzacțiilor este reprezentat printr-un obiect de tip *ListView*, care afisează o listă de elemente. Tranzacțiile sunt preluate de la API-ul serviciului *Etherscan*, printr-un request HTTP. Inițial se preluează primele 25 de tranzacții, iar dacă utilizatorul ajunge la finalul listei, se mai preluează urmatoarele 25 de tranzacții. Tehnica folosită pentru preluarea și afișarea tranzacțiilor se numește infinite scroll[8]. Dacă utilizatorul selectează o tranzacție, va fi redirecționat cu ajutorul browser-ului pe pagina *Etherscan* corespunzătoare tranzacției selectate. Aici va putea vizualiza informații mai în detaliu despre tranzacție. Tranzacțiile marcate cu *OUT* au fost inițializate de către utilizator, iar cele marcate cu *IN* au ca adresă destinație utilizatorul. Butonul de *Refresh* al acestui fragment are ca scop reîmprospătarea balanței și a listei de tranzacții, adică efecuează din nou operațiile prezentate mai sus.

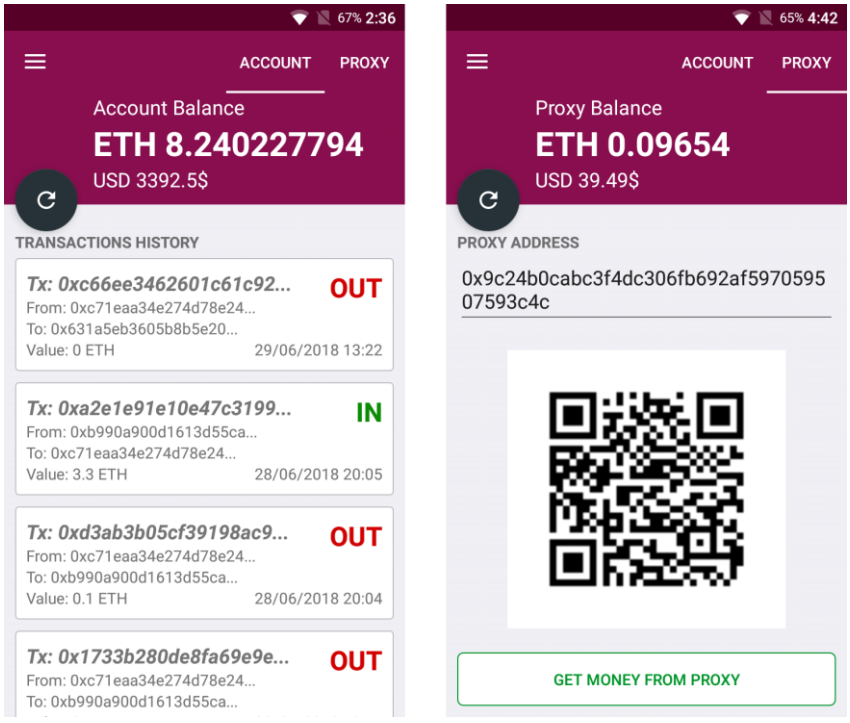

Figura 5.7. AccountFragment și ProxyFragment

[8] https://en.wiktionary.org/wiki/infinite_scroll

5.1.3.3. ProxyFragment

ProxyFragment este destinat adresei Proxy a utilizatorului, aceasta reprezentând locația din blockchain a contractului Proxy creat pentru utilizator în momentul setării identității digitale. Fiecare utilizator are un contract smart Proxy al său. Deoarece interacțiunea sistemului cu un contract smart extern se realizează prin contractul Proxy al utilizatorului, în anumite cazuri, contractul Proxy va primi ether. Astfel în acest fragment este afișată balanța curentă a contractului Proxy. Blanața este preluată din blockchain la fel ca și balanța contului în AccountFragment. ProxyFragment mai conține un câmp cu adresa contractului Proxy și o imagine sub formă de cod QR care este creată dinamic și conține adresa Proxy. Dacă utilizatorul dorește să transfere toți banii din contractul Proxy în contul său, va trebuii să apese butonul
Get money from Proxy. După apăsarea butonului se va inițializa o tranzacție, iar la finalul acesteia balanța contractului Proxy va fi 0. Butonul de *Refresh* este utilizat pentru reîmprospătarea balanței contractului Proxy.

5.1.3.4. ScannerFragment

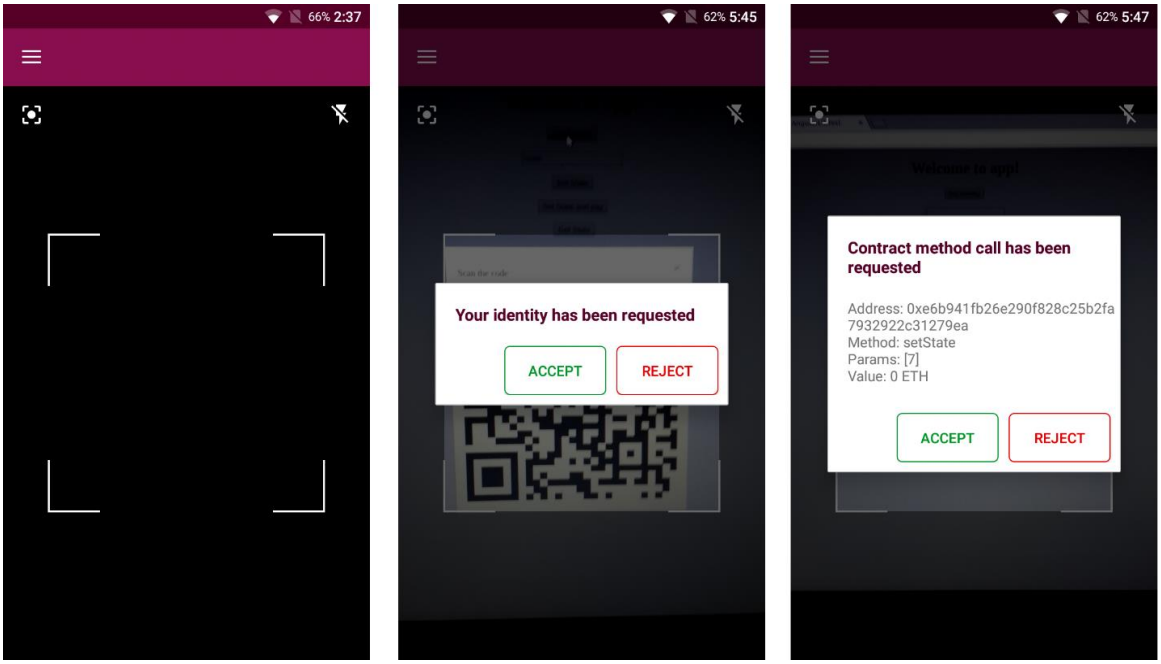

Figura 5.8. ScannerFragment

Acest fragment accesează camera telefonului și este folosit pentru scanarea codurilor QR generate de biblioteca de JavaScript a sistemului. Un cod QR reprezintă o cerere pentru identitatea utilizatorului(tip GET_IDENTITY) sau pentru apelarea unei metode dintr-un contract extern(tip CALL_CONTRACT). Pentru citirea codurilor QR se folosește libraria *code scanner*[9]. Mesajele ecodate în codurile QR sunt de tip JSON. Aplicația mobilă indetifică tipul cererii(GET_IDENTITY sau GET_CONTRACT) și trimite răspunsul prin protocolul WebSockets serverului sistemului. Apoi serverul va redirecționa răspunsul către biblioteca de JavaScript. Pentru fiecare cerere, pe ecranul

[9] https://github.com/yuriy-budiyev/code-scanner

telefonului mobil va apărea un pop-up care îi cere utilizatorului să își dea acordul pentru realizarea operației. Acest lucru poate fi observat în figura 5.8.

Pop-up-ul pentru apelarea unui contract extern conține urmatoarele informații:

- Adresa contractului
- Metoda apelată
- Parametrii transmiși
- Valoarea trimisă(în ether)

5.1.3.5. SetIdentityFragment

Acest fragment este utilizat pentru salvarea identității utilizatorului în blockchain-ul Ethereum. Identitatea utilizatorului va fi criptată inainte de a fi salvată în blockchain, folosind un algoritm PBE(Password Based Encryption), în care parola este PrivateKey-ul utilizatorului. Astfel, doar utilizatorul poate avea acces la propria identitate. În figura 5.9 se obeservă câmpurile ce alcătuiesc identitatea utilizatorului și anume:

- **First Name**: prenumele utilizatorului
- **Last Name**: numele de familie al utilizatorului
- **Email**: adresa de email a utilizatorului
- **Phone Number**: numarul de telefon al utilizatorului
- **Birthdate**: data în care s-a nascut utilizatorul
- **Picture**: poza de profil a utilizatorului

În cazul în care unele câmpuri nu sunt completate sau au un format invalid, aplicația mobilă va afisa un mesaj corespunzător.

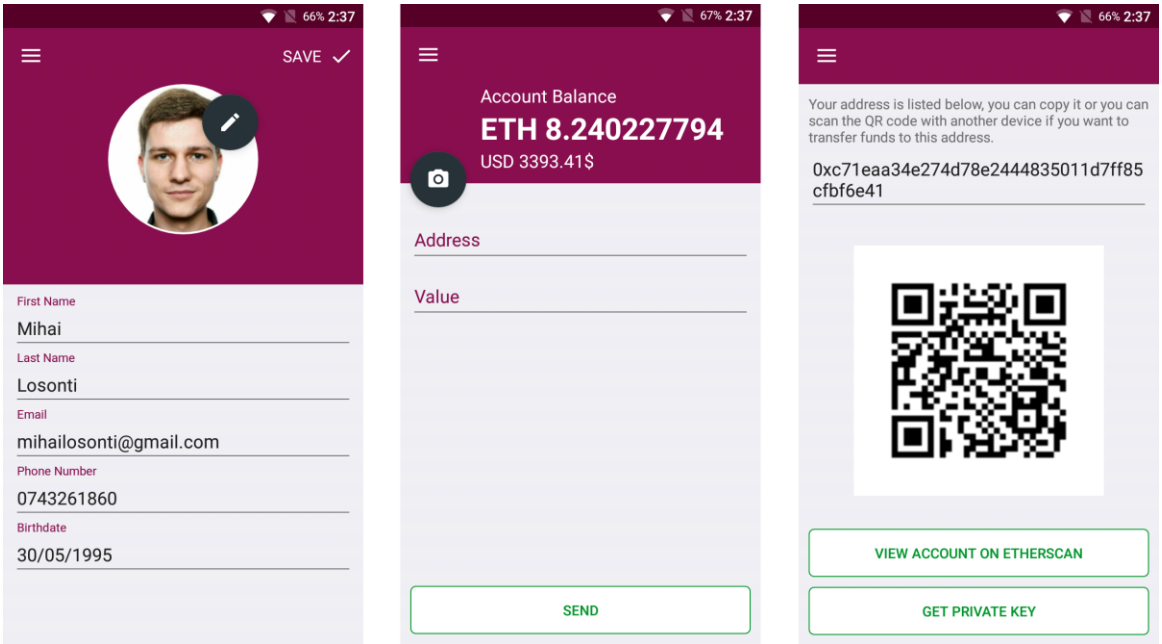

Figura 5.9. SetIdentityFragment, TransactionFragment și InfoFragment

Dacă utilizatorul dorește să își schimbe imaginea de profil, va apasa butonul de *Edit* de lângă poză și va selecta altă imagine din memoria telefonului. Aplicația va încărca noua imagine pe platforma Cloudinary, care va returna un url cu adresa imaginii. În blockchain se salvează url-ul imaginii. Pentru a actualiza în blockchain modificările

făcute, trebuie apăsat butonul *Save*, care va inițializa o tranzacție în blockchain. După terminarea tranzacției utilizatorul va fi înștiințat printr-un mesaj.

5.1.3.6. TransactionFragment

TransactionFragment este destinat transferării de ether din contrul utilizatorului în contul adresei destinație introduse de către acesta. Adresa poate fi introdusă manual sau scanată cu ajutorul camerei telefonului. Dacă se apasă butonul *Camera*, fragmentul Transaction va fi inlocuit de ScannerFragment, care va scana codul QR aferent adresei destianție. După scanare, fragmentul Scanner se închide și se revine la fragmentul Transaction unde câmpul pentru adresă va fi precompletat cu adresa scanată. Utilizatorul va introduce suma dorită și va apăsa butonul *Send*. În cazul în care unele câmpuri au rămas necompletate sau există o eroare, utilizatorul va fi înștiințat printr-un mesaj. Transferul de ether se realizează cu ajutorul funcției *Transfer.sendFunds*, din biblioteca Web3j. După terminarea tranzacției, un mesaj se va afișa pe ecran, iar tranzacția va apărea în lista de tranzacții din AccountFragment.

5.1.3.7. InfoFragment

Acest fragment conține informații despre contul utilizatorului din blockchain-ul Ethereum. În figura 5.9 se pot observa un câmp ce conține adresa contului și o imagine cu un cod QR care encodează această adresă. InfoFragment este util în momentul în care utilizatorul dorește să își alimenteze contrul cu ajutorul altui dispozitiv. De asemenea, InfoFragment mai conține două butoane: *View account on Etherscan* și *Get Private Key*. Primul buton va redirecționa utilizatorul pe pagina Etherscan destinată contului său. Aici, utilizatorul poate vizualiza balanța și propriile tranzacții. Al doilea buton va afisa un pop-up cu PrivateKey-ul utilizatorului. Este recomandat ca utilizatorul să își salveze PrivateKey-ul într-un loc sigur, pentru a putea să își recupere contrul în cazul pierderii acestuia.

5.2. Contractele Smart din blockchain

Pentru încărcarea contractelor smart in blockchain-ul ethereum s-a folosit IDE-ul Remix[10]. Cele două contracte ale sistemului(UserIdentity și Proxy) au fost încărcate în blockchain-ul Rinkeby[11], o rețea pentru testare(testnet în engleză), care are aceleași funcționalități ca și blockchain-ul Ethereum public(mainnet). Acest blockchain permite realizarea tranzacțiilor și testarea contractelor smart, fără a risipi bani reali. Pentru a fi apelabile din limbajul Java, pentru fiecare contract smart trebuie să se genereze cu ajutorul librăriei Web3j o clasă wrapper, specifică. Din aceaste clase rezultate se vor inițializa obiecte Java, fiind specificate în constructor adresa contractului din blockchain. Apoi metodele obiectelor inițializate vor coincide cu metodele conractelor smart din care s-au generat.

[10] https://remix.ethereum.org
[11] https://www.rinkeby.io

5.2.1. Contractul Proxy

O instanța a contractului Proxy este creată pentru fiecare utilizator al sistemului, care își setează identitatea în blockchain. În constructorul acestui contract este trimisă adresa proprietarului(owner-ului) și este salvată ca un atribut public.

Funcția principală a acestui contract este de a apela metodele altor contracte smart externe. Acest lucru se realizează prin metoda *callContract* ce primește ca parametrii adresa contractului extern, numele metodei și parametrii acesteia sub formă de octeți în variablia data și valoarea de ether(de obicei 0). Această metodă conține modificatorul *onlyOwner* care permite doar proprietarului contractului să o apeleze.

Metoda *pay* permite proprietarului să trimită ether contractului său Proxy, pentru ca aesta să poata trimite mai departe valoarea, prin metoda *callContract*.

Metoda *getBalance* returnează balanța curentă a contractului Proxy, iar metoda *sendMoneyToOwner* trimite toți banii din contract, proprietarului.

Contractul Proxy mai conține o metodă implicită, fără nume, care este apelată în momentul în care cineva trimite ether la adresa contracutlui Proxy.

```
contract Proxy {
    address public owner;

    constructor(address _owner) public {
        owner = _owner;
    }

    function pay() public payable {
    }

    function () public payable {
    }

    modifier onlyOwner() {
        require(tx.origin == owner);
        _;
    }

   function callContract(
       address contractAddress,
       bytes data,
       uint256 value) public payable onlyOwner returns(bool success) {
       assembly {
            success := call(gas, contractAddress, value, add(data, 0x20), mload(data), 0, 0)
        }
    }

    function getBalance() public view returns(uint256 balance) {
        balance = address(this).balance;
    }

    function sendMoneyToOwner() public payable {
        if (!owner.send(address(this).balance)) {
            revert();
        }
    }
}
```

Figura 5.10. Contractul Smart Proxy

5.2.2. Contractul UserIdentity

Acest contract conține toate identitățile utilizatorilor. Identitățile sunt salvate în variablila *users*, care este un șir de elemente ce au ca și cheie adresa utilizatorilor, iar ca valoare o structură (*User)*, ce conține identitatea și contractul Proxy al acestora. Deoarece identitatea este criptată, aceasta este salvată într-o variabilă de tip string(șir de caractere). Variabila *msg* conține informații despre utilizatorul care apelează metodele contractului smart: *msg.sender* reprezintă adresa din blockchain a utilizatorului, iar *msg.value* reprezintă valoarea de ether pe care acesta a trimis-o.

Metoda *setUser* salvează în variabila users, identitatea celui care a inițializat tranzacția și instanțiează un contract Proxy pentru utilizator, în cazul în care nu a fost instanțiat. Astfel identitatea utilizatorului a fost salvată în blockchain.

Metoda *getUser* returnează identitatea utilizatorului, în formă criptată. În aplicația mobilă, identitatea va fi decriptată cu ajutorul PrivateKey-ului utilizatorului.

Prin metoda *callContract* se trimite ether către contractul Proxy, dacă este necesar și se apelează metoda callContract din Proxy, pentru apelul unui contract smart extern.

Metoda *getProxyBalance* returnează balanța contractului Proxy, iar *getMoneyFromProxy* transferă banii din Proxy-ul utilizatorului în contul acestuia.

```
contract UserIdentity {
    struct User {
        string identity;
        Proxy proxy;
    }

    mapping(address => User) internal users;

    function setUser(string identity) public {
        users[msg.sender].identity = identity;
        if (users[msg.sender].proxy == address(0)) {
            Proxy proxy = new Proxy(msg.sender);
            users[msg.sender].proxy = proxy;
        }
    }

    function getUser() public view returns(string identity, Proxy proxy) {
        identity = users[msg.sender].identity;
        proxy = users[msg.sender].proxy;
    }

    function getProxyBalance() public view returns(uint256 balance) {
        balance = users[msg.sender].proxy.getBalance();
    }

    function callContract(address contractAddress, bytes data) public payable {
        users[msg.sender].proxy.pay.value(msg.value)();
        users[msg.sender].proxy.callContract(contractAddress, data, msg.value);
    }

    function getMoneyFromProxy() public payable {
       users[msg.sender].proxy.sendMoneyToOwner();
    }
}
```

Figura 5.11. Contractul Smart UserIdentity

5.3. Serverul de WebSockets

Serverul de WebSockets este scris în limbajul JavaScript, fiind un proiect al platformei Node.js. Serverul va accepta conexiuni la portul prestabilit(constanta *PORT*), cu ajutorul framework-ului *express.js*[12]. Pe urmă se crează constanta *wss*, care va permite conexiuni prin protocolul WebSockets la acest server. Pentru sesiunile de comunicare se va genera câte un ID pentru fiecare conexiune. Aceste conexiuni vor fi salvate în variabila *connections*. Dacă se primește un mesaj de tip GET_ID, conexiunea va fi savlată și va primi un ID, trimițând clientului un mesaj de tip RECEIVE_ID. Mesajele ce au atribuite un ID, vor fi trimise clientului cu ID-ul respectiv.

```
const express = require('express');
const WebSocket = require('ws');
const PORT = process.env.PORT || 5000;

const server = express().listen(PORT, () => console.log(`Listening on ${ PORT }`));

const wss = new WebSocket.Server({ server });
var connections = {};
var id = 1;

wss.on('connection', (ws) => {
  console.log('Client connected');
  ws.on('close', () => console.log('Client disconnected'));
  ws.on('message', (message) =>  {
    message = JSON.parse(message);
    console.log('received: %s', message.type);
    if (message.type == 'GET_ID') {
      connections[String(id)] = ws;
      ws.send(JSON.stringify({ type: 'RECEIVE_ID', id: id }));
      id++;
    }
    else if (message.id) {
      var socket = connections[String(message.id)];
      socket.send(JSON.stringify(message));
      if (message.type != 'TRANSACTION_STARTS') {
        delete connections[String(message.id)];
      }
    }
  });
});
```

Figura 5.12. Codul sursă al serverului de WebSockets

O sesiune de comunicare între biblioteca de JavaScript și aplicația mobilă se realizează în următorii pași:

1. Biblioteca de JavaScript va trimite un mesaj serverului, de tip GET_ID.
2. Serverul va salva conexiunea și va trimite bibliotecii de JavaScript ID-ul sesiunii, printr-un mesaj de tip RECEIVE_ID.
3. Biblioteca va genera un cod QR cu cererea către aplicația mobilă și ID-ul sesiunii.
4. Aplicația mobilă va scana codul QR și va trimite răspunsul serverului, împreună cu ID-ul.

[12] https://expressjs.com

5. Serverul va prelua conexiunea corespunzătoare ID-ului și va trimite răspunsul bibliotecii de JavaScript.

Conexiunile încheiate vor fi șterse din variabila *connections*, pentru a nu încărca memoria serverului. În cazul în care un mesaj este de tip TRANSACTION_STARTS, conexiunea nu va fi stearsă deoarece, serverul va mai primii un mesaj de tip TRANSACTION_ENDS, care indică incheierea tranzacției din blockchain.

Serverul este găzduit de platforma *Heroku*, fiind disponibil la adresa: ws://identity-management-server.herokuapp.com.

5.4. Biblioteca de JavaScript

Biblioteca de JavaScript permite integrarea sistemului în diferite aplicații client web. Este scrisă în limbajul JavaScript și poate fi instalată folosind managerul de pachete *npm*. Biblioteca pune la dispoziție o funcție *getIdentity* și o clasă *Contract*.

Funcția *getIdentity* permite preluarea identității utilizatorului. Funcția va genera un cod QR cu o cerere de tip GET_IDENTITY către aplicația mobilă. În cazul în care utilizatorul își dă acordul la trimiterea identității, aceasta va fi trimisă prin protocolul WebSockets serverului, iar serverul o va redirecționa funcției. Funcția *getIdentity* returnează un *Promise*[13], adică un obiect ce reprezintă eventuala finalizare (sau defecțiune) a unei operații asincrone și valoarea rezultată a acesteia. Dacă utilizatorul nu dorește trimiterea identității, funcția *getIdentity* va semnala acest lucru printr-un mesaj.

```
getIdentity().then((identity) => {
  console.log(identity);
}).catch((error) => {
  console.log('Error', error)
});
```

Figura 5.13. Apel funcție getIdentity

Clasa *Contract* permite interacțiunea cu contractele smart externe. Constructorul clasei primește 2 parametrii: adresa contractului în blockchain și ABI-ul(Application Binary Interface) acestuia. ABI-ul unui contract reprezintă lista cu funcțiile contractului și parametrii acestora, în format JSON. ABI-ul este generat de către compilatorul Solidity.

```
var contract = new Contract(abi, '0xe6b941fb26e290f828c25b2fa7932922c31279ea');
```

Figura 5.14. Inițializare obiect de tip Contract

După înstanțierea unui obiect de tip *Contract* se pot apela următoarele metode ale acestuia:

- send(methodName, ...params)
- sendWithValue(value, methodName, ...params)
- call(from, methodName, ...params)

[13] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Funcția *send* va apela o metoda a unui contract smart, ce va modifica starea contractului, realizând o tranzacție. Această funcție primește ca parametrii numele metodei contractului smart și parametrii cu care se dorește să se efectueze apelul. La fel ca și în funcția *getIdentity*, se va genera un cod QR pentru aplicația mobilă, doar că aici nu se va returna un *Promise*, ci se va emite event-uri la începutul tranzacției(event de tip *start*), la sfârșitul acesteia(event de tip *end*) și în momentul în care utilizatorul respinge inițierea tranzacției(event de tip *reject*), dacă este cazul.

```
let tx = contract.send('setState', 10);
tx.on('start', () => console.log('start'));
tx.on('end', (result) => console.log('end', result));
tx.on('reject', (result) => console.log('rejected', result));
```

Figura 5.15. Aplel funcție send

Funcția *sendWithValue* realizează același lucru ca și funcția *send*, doar că aici se trimite și ether, valoarea trimisă fiind specificată prin parametrul *value*.

```
var tx = contract.sendWithValue('0.1', 'setStateWithPay', 20);
tx.on('start', () => console.log('start'));
tx.on('end', (result) => console.log('end', result));
tx.on('reject', (result) => console.log('rejected', result));
```

Figura 5.16. Aplel funcție sendWithValue

Funcția *call* va apela tot o metodă a contractului smart, doar că nu va inițializa o tranzacție, deoarece metoda apelată nu va modifica starea contractului smart. De obicei se apelează metode care returnează o informație. Această funcție nu generează un cod QR, adresa utilizatorului fiind specificată în parametrul *from*.

5.5. Aplicația web demonstrativă

Aplicația web demonstrativă a fost dezvoltată pentru a ilustra funcționalitățile sistemului propus. Framework-ul utilizat pentru crearea aplicației este Angular [14]. Aplicația simulează o casă de licitații online, care folosește criptomoneda ether ca metodă de plată.

Pentru a ilustra comunicarea dintre aplicația mobilă a sistemului și contractele smart externe, s-a creat un contract numit *ActionHouse*, care este folosit de către aplicația demonstrativă pentru a permite licitarea produselor. Fiecare produs este indentificat unic printr-un ID.

La încărcarea contractului în blockchain, se va salva în variabila *owner*, adresa din blockchain a proprietarului casei de licitații. Astfel, folosind metoda contractului *transferFundsToOwner*, se va transfera la final toată suma licitată în ether(pentru toate produsele), proprietarului.

În figura 5.17 se poate observa codul sursă al contractului *ActionHouse*. Pentru a licita un produs, se apelează funcția *bid*, ce primeste ca parametru ID-ul produsului. În cazul în care produsul a mai fost licitat anterior, suma veche licitată va fi transferată vechiului licitator. Apelul acestei funcții necesită îndeplinirea următoarelor condiții:

- Suma licitată trebuie să fie mai mare decât ultima sumă

[14] https://angular.io

- Persoana care licitează trebuie să fie diferită de ultimul licitator

```
contract ActionHouse {

    struct Product {
        address lastBidder;
        uint256 lastBid;
    }

    mapping(uint => Product) internal products;
    address public owner;

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

    function bid(uint id) payable public {
        require(msg.value > products[id].lastBid && msg.sender != products[id].lastBidder);

        if (products[id].lastBid > 0) {
            products[id].lastBidder.transfer(products[id].lastBid);
        }
        products[id].lastBid = msg.value;
        products[id].lastBidder = msg.sender;
    }

    function getProduct(uint id) public view returns (address lastBidder, uint256 lastBid) {
        lastBidder = products[id].lastBidder;
        lastBid = products[id].lastBid;
    }

    function transferFundsToOwner() public {
        require(msg.sender == owner);

        owner.transfer(address(this).balance);
    }
}
```

Figura 5.17. Contractul Smart ActionHouse

Funcția *getProduct*, primeste ca parametru ID-ul unui produs și returnează adresa ultimei persoane care a licitat și ultima sumă licitată, pentru produsul cu ID-ul respectiv. În cazul în care nu s-a licitat niciodată acel produs, se va returna 0 pentru ambele câmpuri.

Aplicația web demonstrativă este găzduită de platforma *Heroku*, fiind disponibilă la adresa: http://identity-chain-demo.herokuapp.com.

Capitolul 6. Testare şi Validare

Scopul acestui capitol este de a testa și a valida corectitudinea sistemului, prin efectuarea operațiilor principale în aplicația mobilă. De asemenea, se va analiza comportamentul sistemului prin integrarea acestuia într-o aplicație web demonstrativă.

6.1. Testara funcționalităților aplicației mobile

Pentru a testa funcționalitățile aplicației mobile se vor folosi două conturi de utilizator. Primul cont de utilizator utilizat, va fi cel personal, cu adresa din blockchain: 0xc71eaa34e274d78e2444835011d7ff85cfbf6e41.

Se va crea al doilea cont de utilizator, pentru testare. După crearea contului în aplicația mobilă, adresa din blockchain generata pentru acesta este: 0x77f6d00290788aa4b7ea82b6dbf6d6f41dfbea16.

Deoarece contul de testare nu are încă o identitate setată, iar balanța acestuia este 0 ETH, prima operație prin care se va testa aplicația mobilă este de transfera valoarea de 1 ether(ETH) din contul personal, în contul de test. Balanța actuală a contului personal este de: 7.539853706 ETH. După efectuarea tranzacției balanța contului personal este de: 6.539832706 ETH, deoarece s-a adăugat o taxă de 0.000021 ETH pentru realizarea tranzacției. În figura 6.1 se observă detaliile tranzacției efectuate.

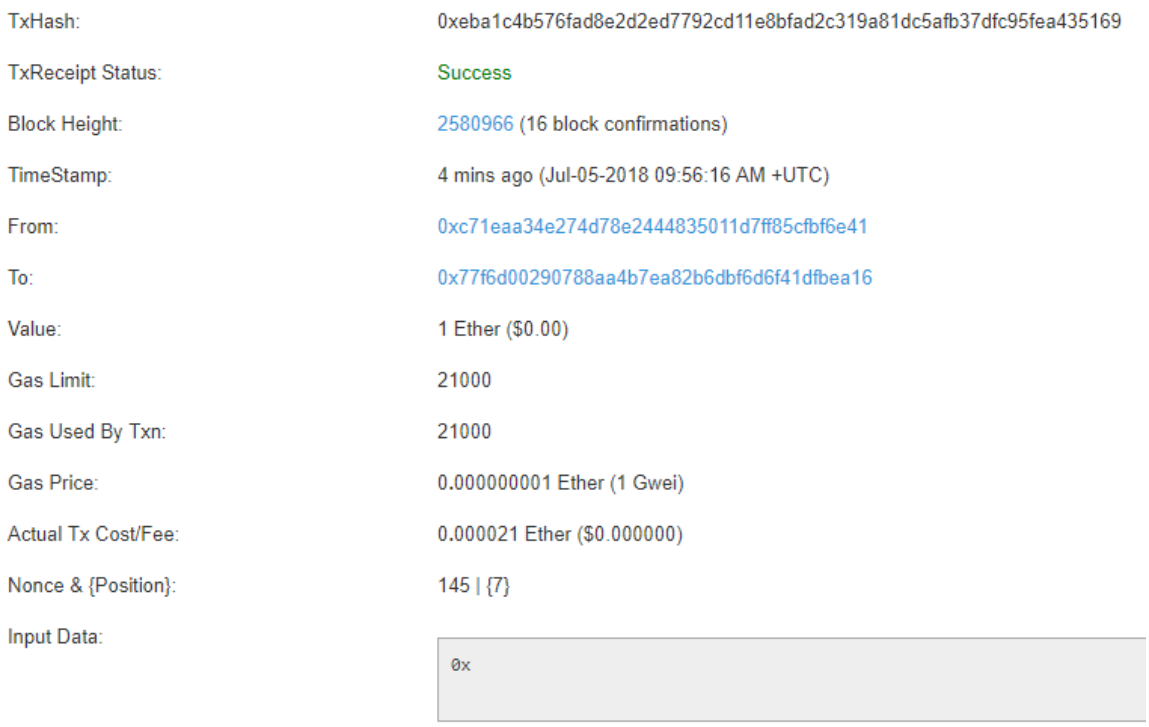

Figura 6.1. Detalii tranzacție efectuată

Contul de test va avea acum balanța de 1 ETH și se poate seta identitatea. În figura 6.2 se observă noua balanță a contului de test, tranzacția prin care s-a primit valoarea de 1 ETH și identitatea precompletată, pe care dorim să o salvăm în blockchain.

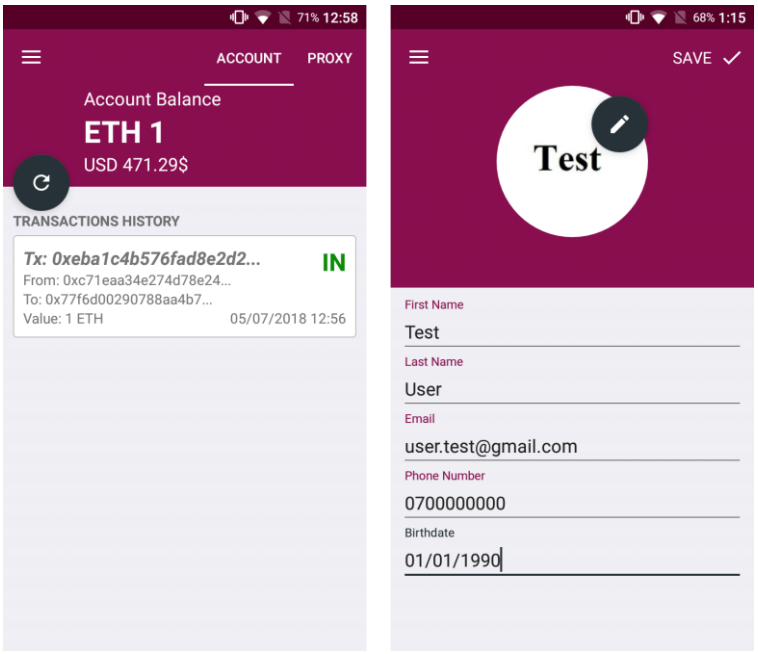

Figura 6.2. Detalii cont de test

După setarea identității în contul de test, balanța acestuia este de 0.98916588 ETH, deoarece s-a perceput o taxă de 0.01083412 ETH pentru realizarea tranzacției.

6.2. Testara prin simularea unui scenariu real

În continuare se va testa integrarea sistemului într-o aplicație web demonstrativă și comunicarea cu un contract smart extern.

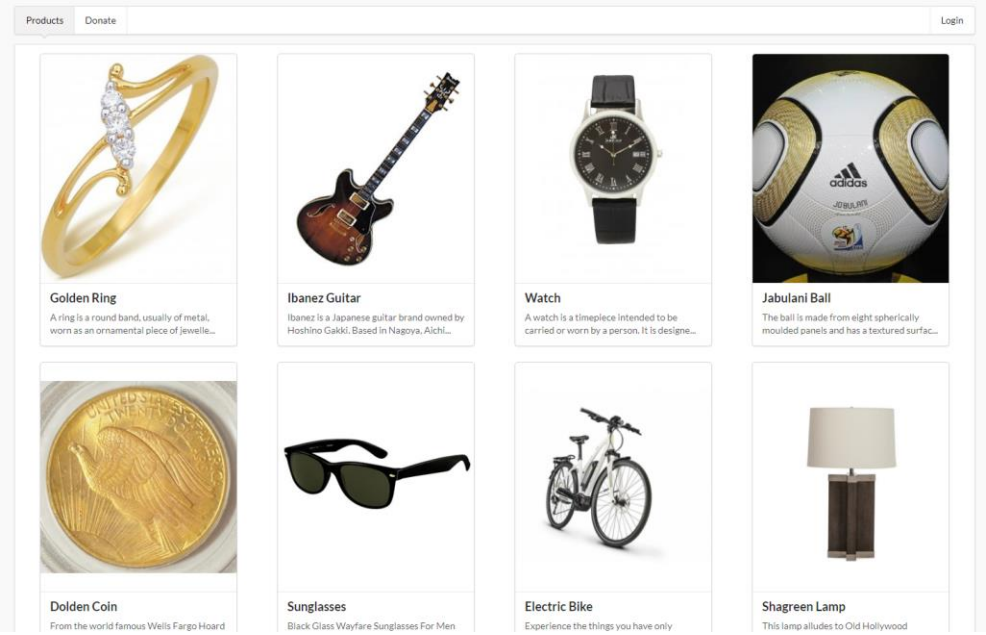

Figura 6.3. Aplicația web demonstrativă

Aplicația demonstrativă simulează o casă de licitații online, prin care se permite licitarea în timp real folosind moneda digitală ether.

Prima funcționalitate testată va fi transferarea identității uilizatorului de test, din blockchain, în aplicația demonstrativă. Pentru a efectua această operație se apasă butonul de *Login*, din aplicația demonstrativă și se va scana codul QR rezultat cu aplicația mobilă. După această operație se observă că utilizatorul a fost conectat, profilul său fiind vizibil în meniul *Profile*, precum se poate observa în figura 6.4.

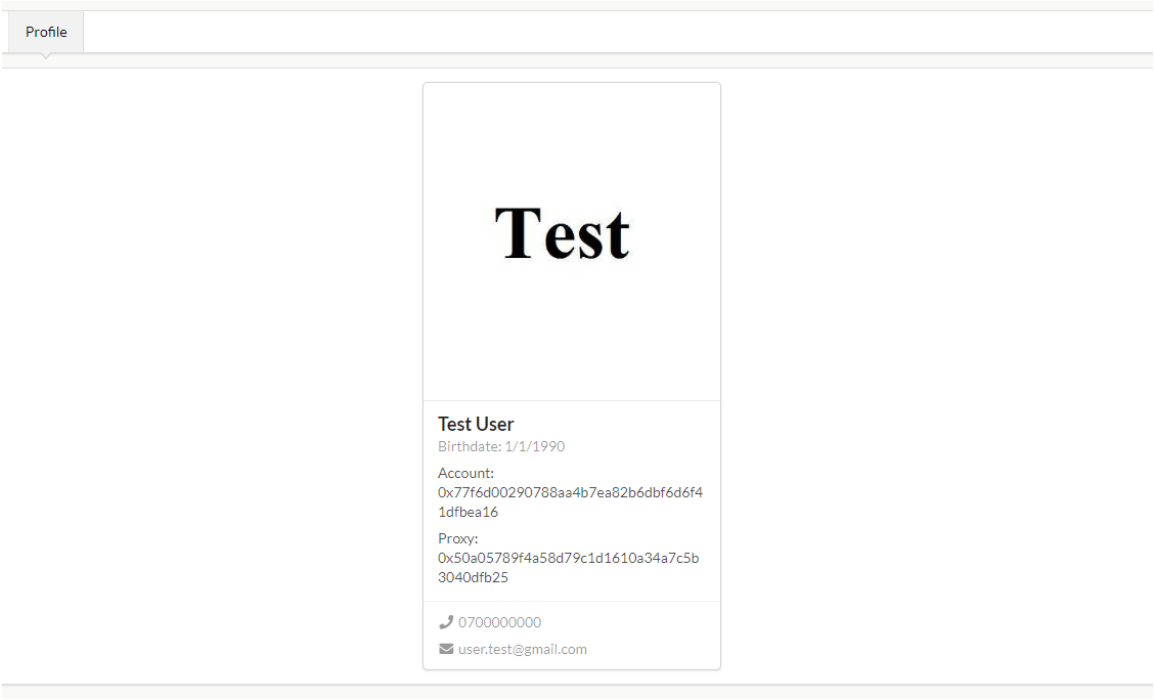

Figura 6.4. Meniul Profile din aplicația demonstrativă

În continuare vom licita un produs. Se alege produsul dorit, se introduce suma licitată(0.1 ETH) și se apasă butonul *Bid*. Se va genera un cod QR care trebuie scanat cu aplicația mobilă.

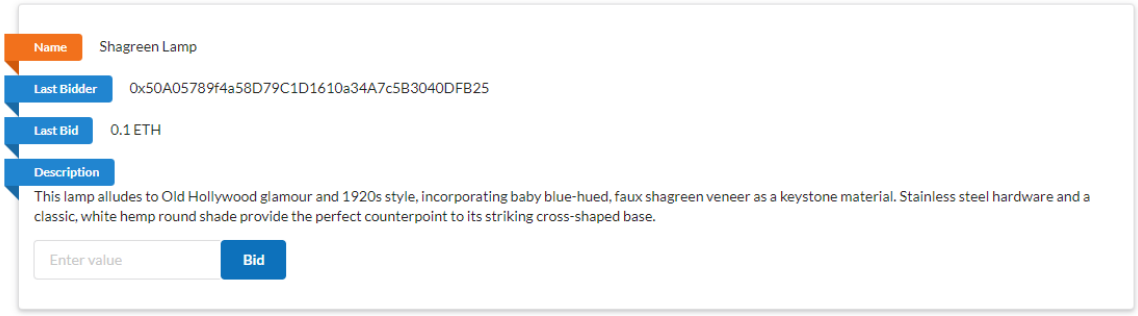

Figura 6.5. Prima licitare

În urma efectuării acestei operații valoarea de (0.1 ETH) din contul utilizatorului de test a fost retrasă. În figura 6.5 se poate observa că ultimul utilizator care a licitat este contractul Proxy al utilizatorului de test.

Pe urmă, ne vom conecta la aplicație cu contul personal și vom licita același produs cu suma de 0.2 ETH. În figura 6.6 se observă că suma licitată a fost modificată și că ultimul utilizator care a licitat este contractul Proxy al contului personal.

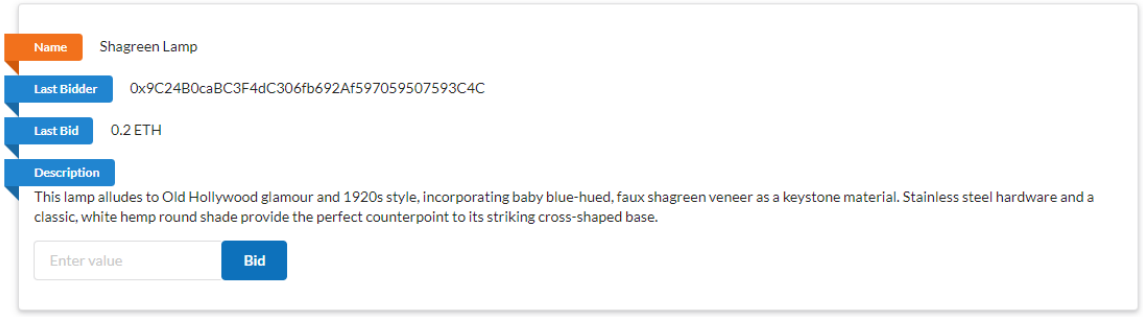

Figura 6.6. A doua licitate

De asemenea în contul Proxy al utilizatorului de test au fost restituiți cei 0.1 ETH licitați anterior. De aici utilizatorul poate să ii transfere în contul său principal, apăsând butonul *Get Money From Proxy*.

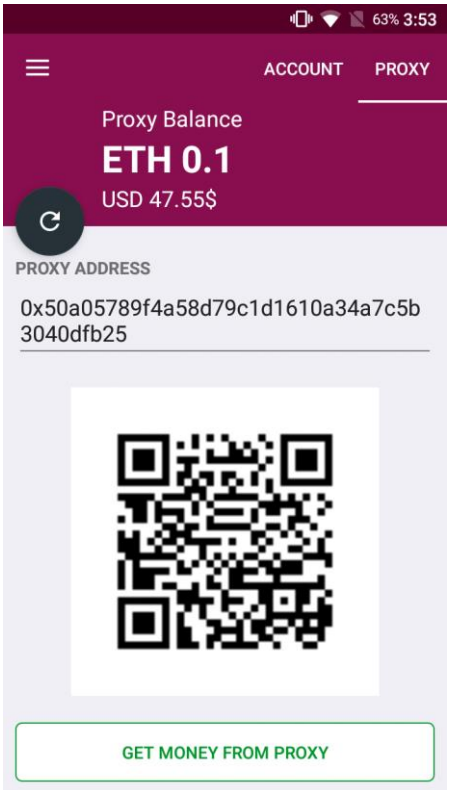

Figura 6.7. Proxy-ul utilizatorului de test

Capitolul 7. Manual de Instalare si Utilizare

În capitolul curent sunt descrișii pașii necesari instalării sistemului și operațiile necesare ce trebuie efectuate pentru buna funcționare a acestuia. De asemenea, se va prezenta manualul de utilizarea al aplicației mobile.

7.1. Instalarea și rularea

7.1.1. Instalare aplicație mobilă

Pentru a instala aplicația mobilă pe un telefon cu sistem de operare Android, sunt necesari următorii pasi:

1. Copiere aplicație în memoria telefonului

Aplicația cu extensia .apk trebuie copiată în memoria telefonului mobil pentru a fi instalată. Acest lucru se poate realiza prin conectarea telefonului mobil la PC cu ajutorul unui cablu USB.

2. Bifare ”Unknown Sources”

Deoarece aplicația mobilă nu este disponibilă în Google Play, trebuie să bifăm opțiunea ”Unknown Sources” din meniul Setări -> Securitate.

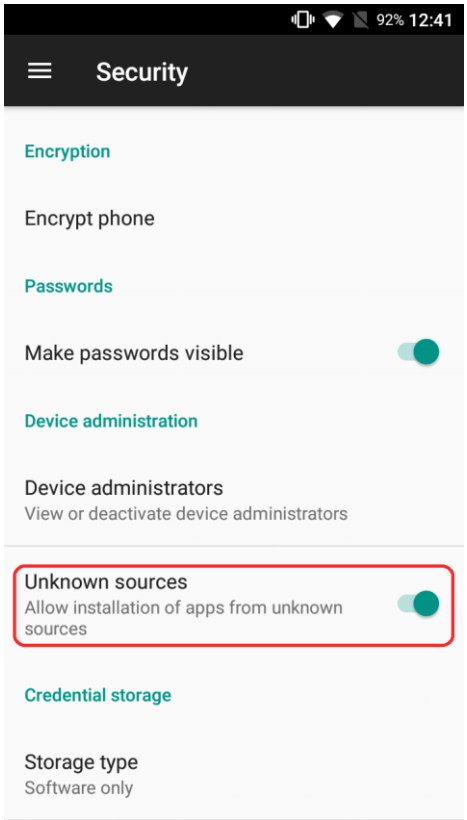

Figura 7.1. Bifare Unknown Sources

3. Instalare Aplicație

Se navigează spre locația unde a fost copiat fisierul .apk în memoria telefonului mobil și se execută. Se acceptă cererea de instalare și se asteaptă terminarea procesului. După acest pas aplicația este instalată și poate fi utilizată.

7.1.2. Instalare bibliotecă de Javascript

Pentru a instala și a integra biblioteca de Javascript a sistemului, într-un proiect, sunt necesari următorii pasi:

1. Instalare Node.js

Node.js este necesar pentru adăugarea bibliotecii de JavaScript într-un proiect web, dar și pentru instalarea modulelor și dependințelor acesteia. Node.js poate fi descărcat de pe site-ul oficial https://nodejs.org, ca executabil, pentru sistemele de operare Windows. Putem verifica dacă Node.js a fost instalat corect, în Command Prompt, prin tastarea comenzii ”node -v”. Dacă se afisează versiunea, atunci acesta a fost instalat cu succes.

2. Descărcare Bibliotecă

În Command Prompt se navighează spre ruta proiectului la care se dorește folosirea biliotecii și se tastează ”npm install identity-chain-js --save”. După descărcarea bilbiotecii cu ajutorul managerului de pachete *npm*, aceasta ar trebuii să apară în directorul *node_modules.*

3. Importare Biliotecă

Pentru a folosi funcțiile biliotecii într-un fișier(de exemplu o componentă a unui proiect Angular), se adaugă la început următoarea linie de cod:

import { getIdentity, Contract } from ”identity-chain-js”

7.2. Manual de utilizare

La prima pornirea a aplicației mobile, ecranul telefonului va arăta ca și în figura 7.2. Pentru a crea un cont de utilizator nou, se apasă butonul *Create Account* și se setează o parolă pentru protecție.

După crearea contului, utilizatorul este nevoit să își seteze identitatea, dar această operație necesită o valoare de ether pentru efectuarea tranzacției în blockchain (aproximativ 0.002 ETH). Deoarece sistemul folosește un blockchain public de testare, contul utilizatorului poate fi alimentat folosind serviciul: https://faucet.rinkeby.io.

După alimentarea contului, utilizatorul poate să își seteze identitatea, accesând meniul *Settings*.

Pentru a transfera ether spre o altă adresă, se selectează meniul *Send Eher*, iar pe pagiana acestuia se va putea introduce sau scana adresa destinație, impreună cu valoarea care se dorește să fie trimisă.

Interacțiunea cu aplicațiile web se realizează selectând opțiunea *Scanner* din meniu. Se va accesa camera telefonului, pentru a putea permite aplicației scanarea codurilor QR.

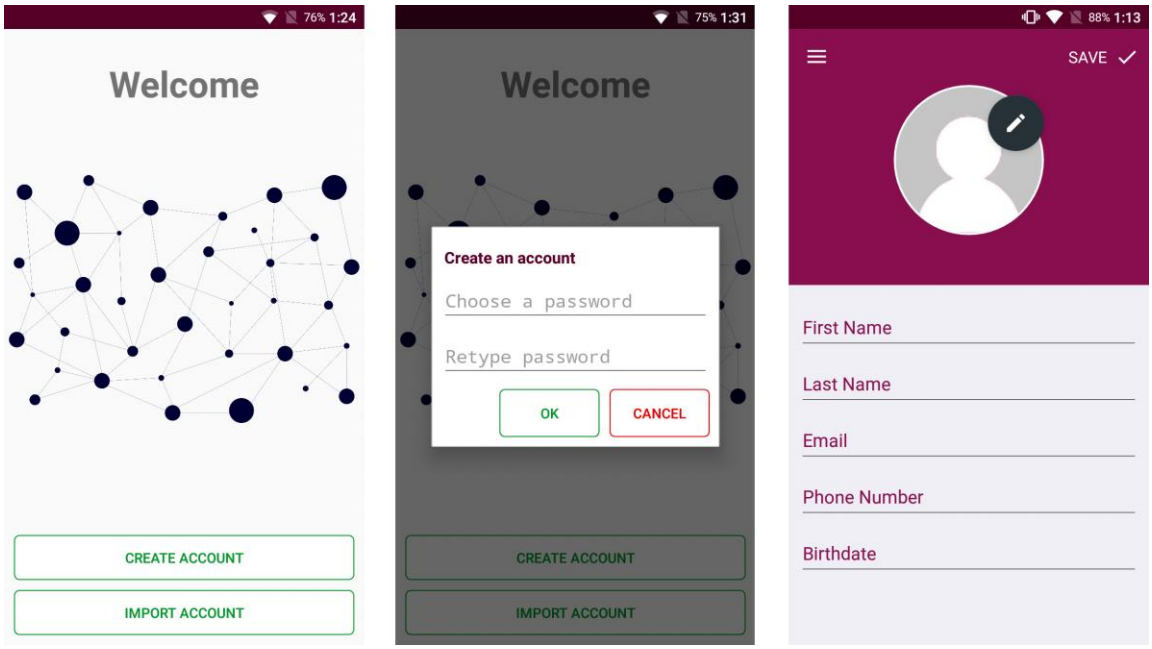

Figura 7.2. Ecranul de start, pop-up creare cont, ecran setare identitate

Meniul *Home* și *Info* ilustrate în figura 7.3 oferă utilizatorilor informații despre conturile lor, precum: blanță, istoric tranzacții efectuate în blockchain, adresă cont, cheie privată, informații despre contractul *Proxy*.

Apelarea funcțiilor din biblioteca de JavaScript este exemplificată în secțiunea 5.4 al capitolului 5.

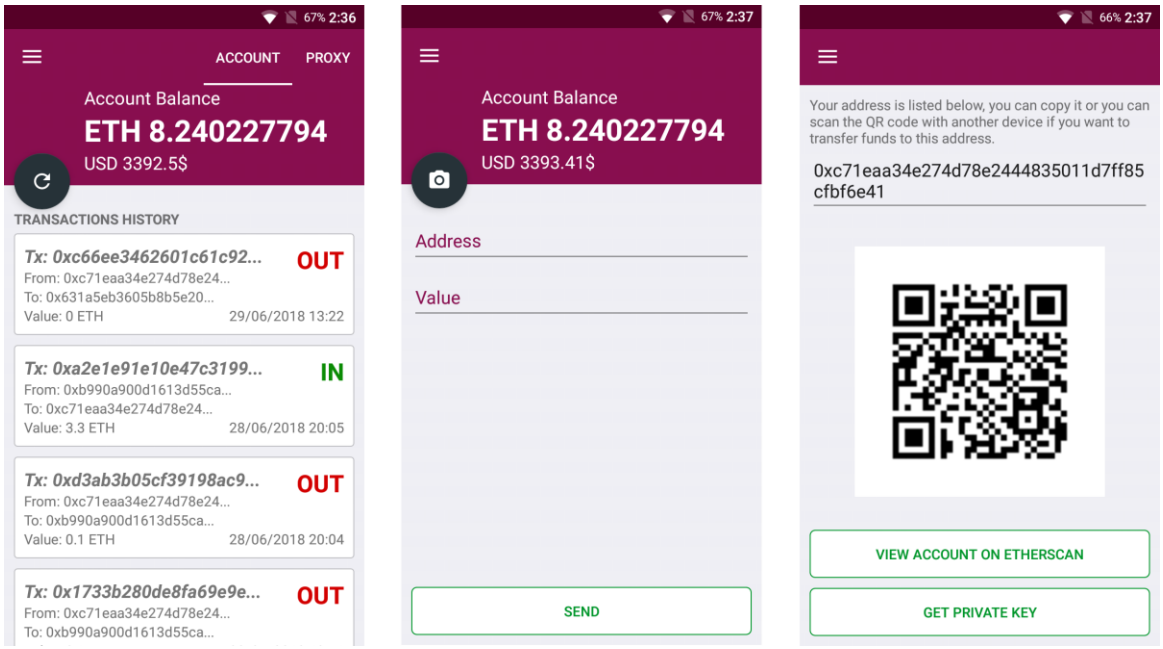

Figura 7.3. Ecranele Home, Send Ether și Info

Capitolul 8. Concluzii

În cadrul acestui capitol se vor prezenta contribuțiile personale aduse la proiect, precum și o retrospectivă a obiectivelor ce au fost atinse.

De asemenea, la finalul acestui capitol se vor enumera cateva posibilități de dezvoltare ulterioară, care să îmbunătățească funcționalitățile sistemului.

8.1. Realizarea obiectivelor propuse

Primul obiectiv principal a fost studiul tehnologiei blockchain. Cunoștințele despre aceasta tehnologie au fost dobândite din resursele bibliografice ale lucrării și din alte surse de pe internet.

Al doilea obiectiv principal a presupus proiectarea și crearea unui sistem distribuit, folosind tehnologia blockchain, care să asigure administrarea identitatii digitale, fiind utilizabil în aplicații web. Acest obiectiv a fost atins în totalitate, IdentityChain fiind un sistem robust, capabil să administreze identitatea utilizatorilor folosind tehnologia blockchain, într-un mediu securizat. Accesul la aplicații web se realizează folosind biblioteca de JavaScript oferită de sistem.

Ultimul obiectiv principal, realizarea unei aplicații demonstrative, prin care să se exemplifice funcționalitățile sistemului, a fost finalizat și el, prin crearea unei aplicații web care simulează o casă de licitații online.

Totodată, obiectivele secundare propuse în capitolul 2 au fost atinse. Sistemul rezultat oferă o interfată utlizator simplă și eficientă, cu funcționalități multiple.

8.2. Contribuții personale

O primă contribuție personală la acest proiect, ar fi: crearea unei aplicații mobile care să permită managementul identității utilizatorilor. Această aplicație poate fi rulată pe telefoanele mobile cu sistem de operare Android, fiind scrisă în limbajul Java. Pentru accesul la blockchain, s-au identificat și integrat cele mai bune tool-uri și biblioteci, folosite în perioada actuală. O altă contribuție a fost crearea și încărcarea contractelor smart în blockchain. Contractele *UserIdentity* și *Proxy* sunt elementele cheie ale sistemului, stocând identitatea utilizatorilor și permițându-le acestora să interacționeze cu alte contracte smart.

Dezvoltarea bibliotecii de JavaScript și a server-ului de WebSockets, care permit integrarea sistemului într-o aplicație web, sunt alte două contribuții personale aduse proiectului.

8.3. Dezvoltări ulterioare

Proiectul realizat are capacitatea de a înlocui metodele tradiționale de protecție ale conturilor de utilizator, nemaifiind nevoie de o parolă pentru securitate. Astfel, printr-o simplă scanare a unui cod QR, utilizatorul se va putea conecta la aplicația web dorită, fiind identificat unic de catre adresa sa din blockchain. În următoarele rânduri se va prezenta o listă de dezvoltări ulterioare ce au ca scop îmbunătățirea funcționalităților sistemului:

- Crearea unei aplicații mobile pentru telefoanele cu sistem de operarea IOS. Deoarece pe piața actuală de smarphone-uri, telefoanele mobile cu sistem de operare IOS ocupă aproape 20%, crearea unei aplicații mobile pentru aceste telefoane va mării numărul de utilizatori ai sistemului.
- Permiterea efectuării de tranzacții în blockchain-ul Bitcoin. Deoarece criptomoneda bitcoin este cea mai poplară și cea mai valoroasă în momentul de fată, realizarea de tranzacții cu această monedă digitală va atrage mai mulți utilizatori.
- Permiterea utilizatorilor să se conecteze la aplicații web, atunci când acestea sunt accesate de pe telefonul mobil, prin redirecționare către aplicația mobilă a sistemului, fără să se mai afiseze un cod QR.
- Accesul aplicațiilor web la câmpurile din identitate specificate de utilizator. La momentul de fată, când o aplicație web cere identitatea unui utilizator, iar acesta acceptă, toate informațiile despre identitatea utilizatorului sunt trimise. Această funcționalitate va permite utilizatorilor să aleagă ce informații vor fi accesate.

Bibliografie

[1] Michael Crosby, Nachiappan, Pradhan Pattanayak, Sanjeev Verma and Vignesh Kalyanaraman, ”BlockChain Technology”, 16 Oct. 2015. [Online]. Available: http://scet.berkeley.edu/wp-content/uploads/BlockchainPaper.pdf.

[2] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang, ”An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends,” 2017. [Online]. Available: https://www.researchgate.net/publication/318131748_An_Overview_of_Blockchain_Technology_Architecture_Consensus_and_Future_Trends.

[3] Hash Function, [Online]. Available: http://mathworld.wolfram.com/HashFunction.html.

[4] Cosset Damien, ”Blockchain: What is Mining?”, [Online]. Available: https://dev.to/damcosset/blockchain-what-is-mining-2eod.

[5] ”The Ultimate Guide to Understanding Blockchain Smart Contracts”, 18 Apr. 2017, [Online]. Available: https://www.blockchaintechnologies.com.

[6] Nick Szabo, ”Smart Contracts: 12 Use Cases for Business & Beyond”, White Paper, Dec. 2016, [Online]. Available: https://digitalchamber.org/wp-content/uploads/2018/02/Smart-Contracts-12-Use-Cases-for-Business-and-Beyond_Chamber-of-Digital-Commerce.pdf

[7] Zachary Diebold, ”Self-Sovereign Identity using Smart Contracts on the Ethereum Blockchain”, Dissertation, University of Dublin, Trinity College, May 2017.

[8] Christopher Allen, ”The Path to Self-Sovereign Identity”, 25 Apr. 2016, [Online]. Available: http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html.

[9] Sesaria Kikitamara, ”Digital Identity Management on Blockchain for Open Model Energy System”, Master’s Thesis, 23 Apr. 2017.

[10] ShoCard, [Online]. Available: https://shocard.com.

[11] uPort. [Online]. Available: https://www.uport.me.

[12] Paul Dunphy and Fabien A. P. Petitcolas, ”A First Look at Identity Management Schemes on the Blockchain” .

[13] Bitnation, [Online]. Available: https://tse.bitnation.co.

[14] UniquID, [Online]. Available: https://uniquid.com.

[15] Cryptid, [Online]. Available: http://cryptid.xyz.

[16] Android, [Online]. Available: https://www.android.com.

[17] Android Studio, [Online]. Available: https://developer.android.com/studio.

[18] Ethereum, [Online]. Available: https://www.ethereum.org.

[19] Java - Overview, [Online]. Available: https://www.tutorialspoint.com/java/java_overview.htm.

[20] JavaScript - Overview, [Online]. Available:

https://www.tutorialspoint.com/javascript/javascript_overview.htm.

[21] Web3j, [Online]. Available: https://docs.web3j.io.

[22] Node.js Introduction, [Online]. Available: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm.

[23] Npm, [Online]. Available: https://www.npmjs.com.

[24] Cloudinary, [Online]. Available: https://cloudinary.com.

[25] RFC 4627, [Online]. Available: https://tools.ietf.org/html/rfc4627.

Anexa 1. Glosar

Termen	Descriere
API	Application Programming Interface
APK	Android Application Package
Activitate	Clasă Java ce definește interața utilizator, pentru aplicațiile Android
Android	Sistem de operare pentru dispozitive și telefoane mobile
Blockchain	O listă în continuă creștere de înregistrări, numite blocuri, care sunt legate între ele și securizate prin criptografie
Ethereum	O platformă și sistem de operare open-source, distribuit, pe bază de blockchain, ce oferă posibilitatea implementării contractelor smart
Fragment	Subcomponenta unei activități Android
HTTP	Hypertext Transfer Protocol
Heroku	Platformă pentru găzduirea aplicațiilor web
IOS	iPhone OS – sistem de operare pentru dispozitivele mobile produse de firma Apple
JS	JavaScript
JSON	JavaScript Object Notation
Node.js	Framework pentru dezvoltarea aplicațiilor de tip server-side
Smart Contract	Un program informatic destinat să faciliteze, verifice, sau să aplice negocierea sau executarea unui contract.
WebSockets	Protocol de comunicație full-duplex
XML	Extensible Markup Language

Anexa 2. Lista figurilor

Anexa 3. Lista tabelelor