

ACADEMIA DE STUDII ECONOMICE DIN BUCUREȘTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

LUCRARE DE LICENȚĂ

Coordonator științific:

Conf. Univ. Dr. Toma Cristian-Valeriu

Absolvent:

Huluba Laura-Alexandra

BUCUREȘTI

2022

ACADEMIA DE STUDII ECONOMICE DIN BUCUREȘTI
FACULTATEA DE CIBERNETICĂ, STATISTICĂ ȘI INFORMATICĂ ECONOMICĂ

Aplicație mobilă pentru gestionarea criptomonedelor

Coordonator științific:

Conf. Univ. Dr. Toma Cristian-Valeriu

Absolvent:

Huluba Laura-Alexandra

BUCUREȘTI

2022

Cuprins

1. Introducere	2
1.1. Motivarea alegerii temei	2
1.2. Contribuția originală	2
1.3. Conținutul lucrării	3
2. Tehnologii utilizate	5
2.1. JavaScript.....	5
2.2. TypeScript	5
2.3. React Native	6
2.4. Blockchain	8
2.5. REST	9
2.6. DApp.....	10
2.7. WalletConnect.....	11
2.8. Moralis.....	11
2.9. CoinGecko	11
2.10. NewsData	12
3. Arhitectura soluției și implementarea soluției.....	13
3.1. Prezentare generală a arhitecturii	13
3.2. Flux de date	13
3.3. Navigarea între ecranele aplicației	14
3.4. Prezentarea soluției implementate	16
4. Concluzii	35
Bibliografie.....	37
Lista tabelelor	38
Lista figurilor	38
Anexe.....	39

1. Introducere

1.1. Motivarea alegerii temei

Întrucât ne aflăm într-o perioadă în care atacurile cibernetice sunt într-o continuă creștere, a apărut necesitatea găsirii unei modalități cât mai sigure de a realiza o tranzacție. Astfel, odată cu apariția criptomonedei Bitcoin în anul 2009, a apărut și primul blockchain care a permis investitorilor să realizeze tranzacții sigure și imutabile. Pentru a putea deține diverse criptomonede apărute pe parcursul ultimilor ani, este nevoie de deținerea unui portofel virtual care să suporte criptomonedele deținătorului.

În acest moment se disting două tipuri de portofele pentru criptomonede, mai precis portofelele virtuale, cunoscute și sub numele de „hot wallet”, pentru telefon mobil, desktop sau web, dar și portofele fizice, cunoscute drept „cold wallet”, precum portofele hardware sau cele din hârtie („paper wallet”). În acest sens, au fost dezvoltate multiple portofele care să accepte jetoanele dezvoltate pe diverse blockchain-uri.[\[1\]](#)






Companiile au început să își adapteze procesele de plată, integrând beneficiile aduse de blockchain-urile existente, cum ar fi securitatea tranzacțiilor, viteza și numărul acestora, acestea funcționând într-o manieră descentralizată. Companii precum Microsoft, PayPal, Amazon și Tesla acceptă plățile în criptomonede pentru diverse produse și servicii oferite de aceștia.

Aplicația mobilă Paw Wallet permite utilizatorilor să își conecteze diverse portofele virtuale deja existente, precum MetaMask, Rainbow, Trust Wallet etc., oferindu-le astfel o interfață comună și îmbunătățită pentru toate aceste portofele.

1.2. Contribuția originală

În acest moment pe piață există o varietate imensă în ceea ce privește portofelele virtuale. În acest sens, câteva dintre cele mai cunoscute sunt MetaMask pentru jetoanele dezvoltate pe blockchain-ul de la Ethereum, Binance pentru BNB, Yoroi pentru jetoanele ADA și Avax Wallet pentru jetoanele Avalanche.

În tabelul următor sunt surprinse diferențele între portofelele virtuale menționate anterior, în comparație cu aplicația dezvoltată, Paw Wallet.

Emblemă	Ecran pentru învățare	Transfer	Grafice	Știri	Swap (Schimb valutar pentru criptomonede)	Adăugare jetoane prin intermediul unui robinet
 Paw Wallet	✓	✓	✓	✓	✗	✓
 MetaMask	✗	✓	✗	✗	✓	✓
 Binance	✗	✓	✓	✗	✓	✓
 Avax	✗	✓	✗	✗	✓	✓
 Yoroi	✗	✓	✗	✗	✓	✓

Tabel 1 - Comparație între aplicații

Conform tabelului, portofelele virtuale existente și cunoscute nu dețin un ecran dezvoltat special cu scopul de a ajuta utilizatorii să învețe despre portofelele virtuale, criptomonede sau despre trendurile existente pe piață. O altă diferență dintre aceste aplicații și Paw Wallet este faptul că în aplicația dezvoltată nu este posibil schimbul valutar, întrucât această funcționalitate este specifică doar rețelei principale, cunoscută sub numele de Mainnet.

Astfel, Paw Wallet se deosebește de restul aplicațiilor deja existente prin faptul că aduce noi funcționalități precum posibilitatea de învățare despre criptomonede direct din aplicație, citirea știrilor, oferind totodată și acces la grafice pentru a analiza piața.

1.3. Conținutul lucrării

În primul capitol al lucrării s-a prezentat motivația alegerii temei, mai precis dorința cunoașterii unui domeniu nou, a unei tehnologii noi, de mare interes în aceste zile, și anume

blockchain-ul. Totodată, în acest întâi capitol au fost prezentate diferențele între funcționalitățile aplicației dezvoltate și prezentate în această lucrare și aplicațiile deja existente pe piață, fiind și printre cele mai cunoscute portofele virtuale de criptomonede (Binance, Yoroi, Avax, MetaMask).

În cel de-al doilea capitol sunt prezentate tehnologiile utilizate în cadrul aplicației, mai precis limbajele de programare, mai precis JavaScript și TypeScript, limbajul platformă ReactNative, dar și API-urile utilizate pentru ca experiența utilizatorului să crească, precum NewsData, CoinGecko, Moralis, WalletConnect, aceste tehnologii și API-uri ajutând atât la dezvoltarea aplicației client, cât și a celei de server.

În cel de-al treilea capitol este prezentată arhitectura aplicației, fluxul de date între componentele aplicației, dar și fluxul de navigare între ecranele acesteia. În continuarea acestui capitol s-au descris ecranele existente în aplicație împreună cu câteva dintre metodele care realizează funcționalitățile portofelului virtual.

În final, în cel de-al patrulea capitol s-au prezentat concluziile lucrării, mai precis scopul acestei aplicații, sumarizând modurile de implementare a acesteia, prezentându-se noile funcționalități aduse în comparație cu cele a unui portofel virtual deja existent, dar și îmbunătățirile care pot fi aduse aplicației dezvoltate, în vederea realizării unei noi versiuni a aplicației.

2. Tehnologii utilizate

2.1. JavaScript

JavaScript este un limbaj de programare interpretat de browser, fiind considerat unul dintre limbajele omniprezente, putând fi utilizat atât în aplicația client, cât și în aplicația server. Totodată, JavaScript poate fi folosit nu numai pentru aplicațiile Web, dar și pentru dezvoltarea aplicațiilor mobile, a dronelor, Internet Of Things și multe alte altele. [\[2\]](#)

Limbajul JavaScript s-a schimbat odată cu apariția ECMAScript 2015, această actualizare adăugând clase, module, tipuri de declarări de variabile și multe altele, evoluând într-un mod pozitiv în scopul de a rezolva mai bine problemele apărute. [\[3\]](#)

În ceea ce privește DOM-ul („Document Object Model”), acesta este o interfață pentru documentele HTML. DOM afișează paginile web într-o manieră în care aplicațiile îi pot modifica structura, stilul și conținutul. Există două standarde specifice pentru DOM și anume „World Wide Web Consortium” (W3C) și „Web Hypertext Application Technology Working Group” (WHATWG).

DOM-ul unei pagini web este construit pe baza unor noduri. Elementele din DOM de tip ierarhie au relații între elemente de tip părinte sau copil sau frate. Pentru a modifica DOM-ul cu JavaScript, trebuie identificat exact nodul care trebuie modificat, cu ajutorul identificatorilor. Elementele din DOM pot fi găsite în patru moduri, în funcție de id, tag, clasă sau selector CSS. De pildă, găsirea elementului după id se face cu `document.getElementById(id)` [\[4\]](#)

- `document.getElementById(id)`

2.2. TypeScript

Între TypeScript și JavaScript există o dependență foarte mare, dar care poate crea confuzii. De pildă, compilarea acestui limbaj de programare – TypeScript - are loc în cadrul JavaScript.

Se spune că TypeScript este un superset al Javascript, adică orice cod JavaScript devine cod Typescript prin redenumirea extensiei, însă există cod scris în TypeScript care nu este cod JavaScript pentru că TypeScript are și sintaxă proprie. Fișierele corespunzătoare pentru TypeScript au extensia `.ts` sau `.tsx`, în timp ce fișierele corespunzătoare pentru JavaScript au

extensia .js sau .jsx, și totuși codul scris în fișierele tip JavaScript poate fi deja interpretat în cadrul TypeScript.

Un avantaj al TypeScript este posibilitatea de a detecta secvențe de cod care ar duce la excepții la rulare, cât și acele secvențe de cod care nu ar îndeplini scopul pentru care au fost scrise, astfel că, la scrierea programelor în TypeScript se încearcă trecerea tuturor acestor verificări făcute în mod automat. Totuși, se poate ca un program să compileze chiar dacă există unele erori de scriere.

De asemenea, este important editorul ales, astfel încât să permită folosirea serviciilor oferite de TypeScript, precum sugestiile în timp ce este scris codul. Sunt de preferat declarările de forma tip `“:Type”` față de declarările de tip `“as Type”`. Se recomandă și folosirea primitivelor tipurilor de variabile în locul obiectelor de tip wrapper, de pildă `“number”` în loc de `“Number”`. [\[5\]](#)

TypeScript este un limbaj de programare care trebuie convertit în limbaj JavaScript, numai după aceea putând fi executat în cadrul unui browser. Spre deosebire de alte limbaje de programare precum Java sau C++, TypeScript identifică erorile înainte ca programul să fie rulat, acesta fiind unul dintre principalele motive pentru care din ce în ce mai mulți programatori aleg TypeScript.

Adesea TypeScript și JavaScript sunt folosite în același proiect. De pildă, codul poate fi scris în TypeScript și să se folosească librării JavaScript. Pentru a face acest lucru, se folosesc fișiere de definire a tipului, prin care compilatorul TypeScript înțelege ce tipuri trebuie folosite cu API-urile specifice JavaScript. Aceste fișiere au extensia `“.ts”`, iar majoritatea acestor fișiere pot fi găsite gata făcute pentru mai mult de 7000 de librării JavaScript. Deci, aceste librării nu trebuie instalate, ci doar sunt necesare fișierele de definire a tipurilor. [\[6\]](#)

2.3. React Native

React Native este un limbaj platformă JavaScript care ajută la scrierea codului pentru aplicații mobile de tip iOS sau Android în mod simultan. React Native se bazează tot pe React, doar că este destinat pentru aplicații mobile, nu pentru browsere web. Folosind React Native se vor interpreta componente mobile pentru interfața cu utilizatorul și nu vizualizări web, ceea

ce ajută la obținerea unor aplicații mobile autentice, care pot utiliza de pildă camera foto a telefonului sau locația utilizatorului.

Un avantaj al React Native este că are metode deja definite pentru aplicațiile mobile, ce folosesc un mix între JavaScript, HTML și CSS. De asemenea, transformă codul în elemente UI native și are performanță mare. Atunci când “props” sau “state” sunt modificate, React Native declanșează actualizarea componentelor. În plus, oferă un avantaj mare pentru dezvoltarea aplicațiilor mobile, precum experiența îmbunătățită a programatorilor și potențialul mare de a realiza aplicații complexe, cu mai multe platforme de dezvoltare interconectate.

Practic, React Native se bazează în principal pe JavaScript și de aceea nu este nevoie să se ruleze din nou aplicația pentru a putea vedea modificările, ci doar se reîmprospătează pagina. Un alt avantaj al React Native este că pot fi folosite instrumentele de dezvoltare ale Chrome sau Safari și pentru dezvoltarea aplicațiilor mobile. În plus, se poate folosi orice editor de text pentru JavaScript și nu impune de pildă folosirea Xcode pentru dezvoltarea aplicațiilor iOS sau Android Studio pentru dezvoltarea aplicațiilor android.

React Native nu necesită resurse mari pentru a construi aplicații mobile, ci orice dezvoltator care știe să scrie cod în React Native poate folosi aceleași abilități pentru a dezvolta aplicații Web, iOS, cât și Android.

React Native folosește algoritmi prin care se folosește cel mai mic număr posibil de apelări către DOM. Utilizarea JavaScript XML (JSX) ajută la exprimarea declarativă a componentelor și la adăugarea întregii logici a proiectului într-un singur fișier.

O componentă React Native este un obiect de tip funcție, care se poate refolosi și care ajută la descrierea componentelor mobile native pe care vrem să le redăm. React Native are două metode pentru fluxul de date dintre componente: una pentru fluxul de date prin ierarhia componentelor, cât și una pentru menținerea stării. Mai precis, se folosește „props” pentru fluxul de date, care nu ar trebui modificate direct niciodată și pot fi accesate prin `this.props`. Se poate folosi modulul `PropTypes` pentru a face validări ale proprietăților și acoperă multe dintre primitivele și tipurile oferite de JavaScript. Pentru stare se poate folosi funcția „`setState`”, care îmbină obiectul cu starea curentă a componentei. [\[7\]](#)

2.4. Blockchain

Blockchain este un sistem prin care pot fi tranzacționate bunuri, nu doar bani, fără o terță parte, care ar putea fi bănci sau brokeri. De pildă, în cazul tranzacțiilor clasice de acțiuni reglementarea poate dura până la câteva săptămâni, deși actul de vânzare-cumpărare a avut loc aproape instantaneu. Blockchain ajută la realizarea reglementării aproape instantaneu, fără a fi necesari brokeri sau alte instituții financiare, iar cele două noțiuni de tranzacție și reglementare sunt identice în ideologia blockchain.

În blockchain datele inserate nu pot fi modificate sau, cu alte cuvinte, fiecare intrare este permanentă. Fiecare set de date inserat este copiat în fiecare nod al blockchainului. Blockchain a fost realizat cu scopul de a obține o descentralizare reală. Fiecare nod din blockchain sau bloc („block”) conține o referință a blocului anterior, astfel încât să nu poată fi modificată vreo tranzacție din blocul anterior. Bineînțeles, un nod conține și lista de tranzacții și informațiile necesare precum suma sau adresa celor implicați în tranzacție. În plus, tranzacțiile sunt ireversibile și datele de tranzacționare neschimbate, imutabile.

Blockchain a apărut odată cu Bitcoin, în anul 2009, iar ulterior au fost demonstrate avantajele pe care le poate aduce această tehnologie și în alte domenii față de criptomonede. De pildă, a apărut Ethereum sau platformele cloud ale Microsoft și IBM (Azure și respectiv Bluemix). A devenit o tehnologie atât de revoluționară, încât aproape orice banca sau entitate financiară a început să folosească blockchain. Alte domenii în care blockchain a început să își găsească utilitatea sunt: tranzacționarea de energie, predicția evoluției piețelor, lanțuri de aprovizionare, asigurări, logistică, sănătate sau chiar aplicații ale armatei sau ministerelor.[\[8\]](#)

Componentele generale a blockchainului sunt: o rețea între serverele participante care conectează și propagă tranzacțiile și blocurile tranzacțiilor verificate, mesaje tranzacțiilor, un set de reguli prin care se identifică un status valid de tranziție, o mașină care procesează tranzacțiile conform regulilor, un lanț de blocuri de tranzacții verificate și bineînțeles, un algoritm de descentralizare a nodurilor din cadrul lanțului.[\[9\]](#)

Astfel, blockchainul este o tehnologie revoluționară întrucât aduce un nivel economic în cadrul aplicațiilor Web, nemaiîntâlnit până în 2009, întrucât acesta reprezintă o nouă modalitate de oferire a serviciilor personalizate și guvernate în mod descentralizat care facilitează dezvoltarea economică. Acesta este văzut ca un sistem care pare a fi coordonat de milioane de angajați, dar care a cărei tehnologie revoluționară poate schimba percepția erei mașinilor inteligente.[\[10\]](#)

2.5. REST

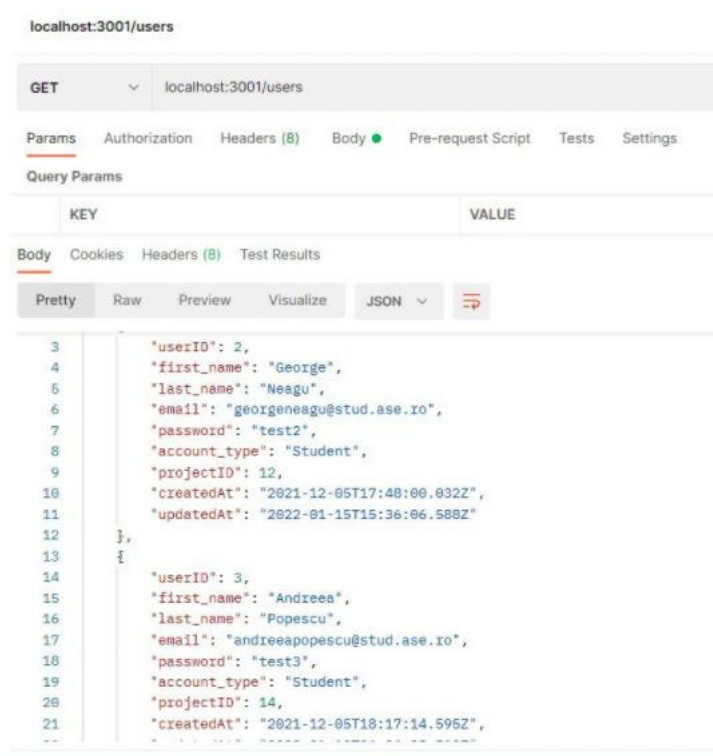
Arhitectura de tip REST – Representational state transfer este un stil creat pentru a ghida implementarea efectivă a aplicației, dar și a design-ului acesteia. Prin intermediul arhitecturii de tip REST, fiind bazată pe protocolul HTTP, se respectă standardele aplicațiilor informatice de pe web, facilitând comunicarea între acestea. [\[11\]](#)

O cerere HTTP este o cerere a unei resurse către un server, fiind așteptat înapoi un răspuns din partea server-ului. Această cerere se realizează prin intermediul unui URL, un URL unic care să poată identifica resursa cerută și durata de viață se încheie odată cu terminarea sesiunii dintre client și server.

Standardul HTTP definește opt metode prin care un client poate cere o resursă, câteva dintre acestea fiind GET, POST, PUT, DELETE. Ca răspuns a acestor apeluri de metode, serverul trimite înapoi un răspuns ce poate fi împărțit în trei, astfel:

- Cod de status – de exemplu, cod 200, semnificând că răspunsul a fost în regulă și resursa a fost găsită sau cod 400, intitulat ca și „Bad Request” sau cererea rea, aceasta semnificând că serverul nu va procesa cererea întrucât există malformații a cerere sau o rutare greșită către resursă
- Corpul – acesta reprezintă resursa efectivă cerută de client, corpul fiind regăsit între acolade și este un obiect JSON
- Antetele răspunsului – acestea sunt perechi cheie-valoare care descriu corpul răspunsului și răspunsul HTTP în general. [\[12\]](#)

Așa cum se observă în Figura 2-1, pentru a se realiza o cerere HTTP, este nevoie să se specifice tipul de metoda, în acest caz GET, adresa la care se regăsește resursa și identificatorul unic din cadrul URL-ului. Ca urmare a realizării cererii în cadrul aplicației POSTMAN, s-a primit ca răspuns un vector de obiecte JSON.



Figură 2-1 - Exemplu de cerere HTTP GET

2.6. DApp

Aplicațiile descentralizate sunt cunoscute sub numele de DApps. Aceste aplicații funcționează în cadrul unui blockchain sau a unei rețele de calculatoare. DApps sunt contruite în general pe platforma Ethereum și sunt folosite în diferite domenii, precum cel al jocurilor sau în sectorul financiar. Întrucât aceste aplicații sunt descentralizate, ele nu sunt supuse unei singure autorități. Astfel, diferența majoră dintre aplicațiile centralizate și cele descentralizate este că o aplicație centralizată este deținută de către o singură companie, în timp ce dApps funcționează în cadrul unui blockchain.[13]

Așadar, o aplicație descentralizată este o aplicație a cărei server rulează pe un sistem de rețea între rețea descentralizat, codul acestuia fiind gratuit. Niciun nod din cadrul blockchainului nu are control complet asupra aplicației.

Câteva dintre avantajele aplicațiilor descentralizate sunt:

- Este preferată de utilizatori întrucât aplicația nu este deținută de nicio entitate care ar putea folosi datele
- Aplicațiile descentralizate previn încălcarea cenzurării nete.[14]

2.7. WalletConnect

WalletConnect este un protocol folosit pentru comunicarea securizată dintre portofelele virtuale și DApps. Acest protocol are implementarea scrisă în TypeScript. Pentru a fi stabilită conexiunea, inițiatorul (DApp) trimite un cod encriptat și unic folosit doar cu scopul de a realiza operațiunea de „strângerea de mână” (handshake), cât și detaliile conexiunii către serverul de conexiune. Algoritmii de encriptare sunt AES-256-CBC și HMAC-SHA256. Ulterior, DAppul trimite în formatul standard al WalletConnect URI parametrii necesari conexiunii, și anume codul encriptat pentru handshake, URI și cheia, sub forma:

- `wc:{topic...}@{versiune...}?bridge={URI...}&key={cheia...}`

Urmează intervenția portofelului virtual de a citi URI-ul, utilizând fie un cod QR, fie o adresă care redirecționează direct către aplicație („deep link”), ceea ce va decodifica imediat codul pentru conexiune. Apoi portofelul virtual va afișa detaliile de conexiune, astfel încât utilizatorul să poată accepta sau respinge conexiunea. [\[15\]](#)

2.8. Moralis

Moralis ajută la construirea și rularea DApps pe baza de Ethereum sau Solana și asigură funcționarea lor și în viitor în cadrul oricărui blockchain. Toate opțiunile oferite de Moralis pot fi accesate printr-un SDK. Moralis face mai ușoară construirea proiectelor de blockchain, atât pentru programatorii începători, cât și pentru cei seniori.

Moralis folosește un așa-numit Moralis server și oferă și o baza de date care, de pildă, salvează adresa portofelului virtual a unui utilizator ce a folosit autentificarea portofelului în DApp. De asemenea, oferă și „Cloud Code”, care ajută la execuția codului de server al aplicațiilor DApp. Moralis server și DApp funcționează împreună prin intermediul Moralis SDK. Mai mult, prin Moralis SDK se poate face autentificarea utilizatorilor fie prin numele de utilizator și parolă, fie printr-un portofel virtual pentru criptomonede precum MetaMask. [\[16\]](#)

2.9. CoinGecko

CoinGecko oferă API-uri pentru aplicații realizate în domeniul criptomonedelor, precum: prețuri în timp real, volumul tranzacțiilor, date istorice, diferite categorii de criptomonede, etc. De pildă, prin endpoint-ul GET /simple/price, ce are ca parametri de

interogare „ids” și „vs_currencies”, sunt returnate prețurile criptomonedelor inserate în valuta dorită. Un alt endpoint important în realizarea aplicației este GET /coins/{id}/history prin care sunt returnate date istorice ale unei monede într-o anumită zi. [\[17\]](#)

Pentru a instala API-urile oferite de CoinGecko se folosește comanda:

- `npm install coingecko-api`

CoinGecko oferă și metode utile în realizarea aplicației precum Order, prin care se ordonează datele, sau Ping, prin care se verifică starea unui server de API-uri. Metoda coins.all() afișează toate monedele, în timp ce exchanges.all() afișează toate schimburile de monede. [\[18\]](#)

2.10. NewsData

NewsData este o platformă care caută și urmărește știrile din întreaga lume, ajutând și la analiza acestora. De asemenea, afișează și date istorice pentru ultimii doi ani din mii de surse, returnându-le atât în format JSON, cât și în format Excel. Căutarea se poate face atât prin simple cuvinte, cât și utilizând fraze întregi. Pentru rezultate mai precise, se poate specifica și perioada în care au apărut anumite știri, țara sau limba în care au fost scrise știrile.

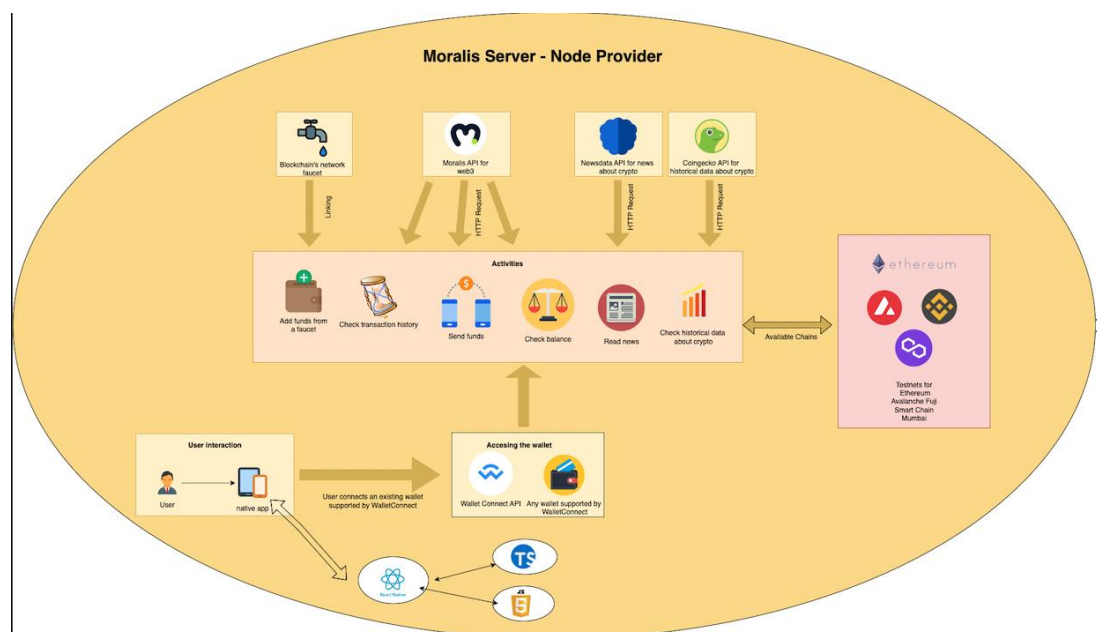
API-urile de tip REST oferite de NewsData folosesc chei de autentificare, care se obțin prin crearea unui cont pe platforma acestora. Cheia se pune ca parametru în URL, apiKey fiind singurul parametru obligatoriu din URL. Alți parametri posibili sunt: country, domain, language, page, etc. În răspuns se găsesc câmpuri precum status sau totalResults, cât și array-ul results, în care se găsesc informațiile efective despre știri. Un exemplu de endpoint este: [\[19\]](#)

- `https://newsdata.io/api/1/news?apikey=YOUR_API_KEY&language=en`

3. Arhitectura soluției și implementarea soluției

3.1. Prezentare generală a arhitecturii

Aplicația îmbină diferite tehnologii și module pentru a crește experiența utilizatorilor, dar și pentru a optimiza aplicația. Astfel, în cadrul limbajului platformă ReactNative, au fost folosite limbajele JavaScript și TypeScript pentru realizarea cererilor, dar și pentru crearea componentelor din cadrul aplicației.



Figură 3-1 - Arhitectura aplicației

3.2. Flux de date

Prin intermediul stilului de arhitectură REST, aplicația comunică cu cele patru servere din cadrul acesteia: WalletConnect, Moralis, NewsData și CoinGecko. În primul rând, pentru ca utilizatorul să se poată conecta la aplicație, acesta trimite o cerere de tip POST serverului WalletConnect, care odată cu verificarea datelor, stabilește o conexiune la distanță, permițând astfel o comunicare sigură între portofelele virtuale și aplicațiile de tip DApp (aplicații descentralizate). Datele trimise sunt encriptate simetric printr-o cheie comună între cele două entități. Conexiunea este inițializată de către una dintre entități prin deep linking și se stabilește atunci când cealaltă entitate aprobă această cerere. [15]

Deep links sau “link-urile profunde” sunt linkuri care trimit utilizatorii către un alt site sau către o altă aplicație instalată. Acestea sunt folosite pentru a trimite utilizatorii direct către

anumite locații din aplicație, economisind utilizatorilor timp și energie pentru a localiza ei înșiși o anumită pagină - îmbunătățind semnificativ experiența utilizatorului. Conectarea profundă face acest lucru specificând o schemă URL personalizată (linkuri universale iOS) sau o adresă URL de intenție (pe dispozitivele Android) care deschide aplicația utilizatorului dacă aceasta este deja instalată. [20]

Astfel, procesul prin care utilizatorul se poate conecta la portofelul virtual în aplicația de tip DApp este realizat prin intermediul API-ului WalletConnect, care la rândul său, prin intermediul Deep linkingului, redirecționează utilizatorul din aplicația dezvoltată în altă aplicație, o aplicație de portofel virtual deja existent, cum ar fi MetaMask sau Binance, prin care i se cer credențialele de conectare, verificarea amprenteii sau a Face ID-ului (recunoaștere facială). Odată cu această verificare, utilizatorului i se cere semnarea unei tranzacții care atestă că acesta a fost de acord să permită conectarea portofelului prin intermediul API-ului WalletConnect și încă o tranzacție prin care își exprimă acordul ca API-ul Moralis să verifice jetonul de conectare pe parcursul sesiunii în cadrul aplicației dezvoltate.

În cadrul aplicației, utilizatorul poate adăuga fonduri în portofelul său virtual bazat pe una dintre rețelele de Testnet a blockchainurilor Ethereum, Smart Chain, Mumbai sau Avalanche Fuji. Adăugarea de fonduri se realizează de asemenea prin Deep Linking, utilizatorul fiind redirecționat către unul dintre „robinetele” (Faucets) rețelei. Această rețea este identificată prin intermediul id-ului de lanț (chain).

Ca funcții generale, prin intermediul aplicației se pot transfera fonduri, se poate verifica istoricul tranzacțiilor și a fondurilor existente. Astfel, toate aceste procese sunt bazate pe API-ul Moralis, care are injectat Web3, pentru a avea acces la blockchain. Pentru le putea realiza, trebuie trimise id-ul lanțului, dar și adresa portofelului virtual conectat în aplicație.

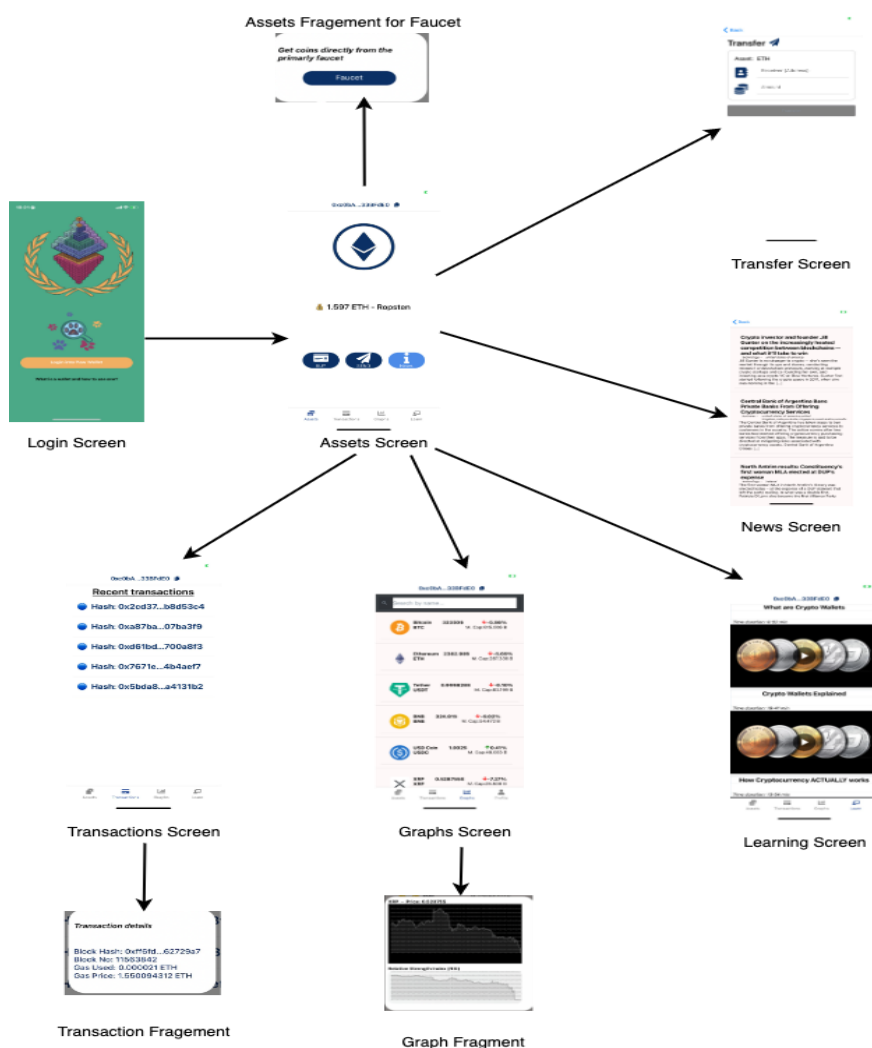
Pentru a prelua datele, aplicația trimite solicitări (cereri de tip GET) către API-urile NewsData și Coingecko, de fiecare dată când utilizatorul ajunge pe unul dintre ecranele de grafice sau de știri. Odată preluate, acestea se trimit mai departe spre a fi formate și afișate.

3.3. Navigarea între ecranele aplicației

Navigarea în cadrul aplicației s-a realizat prin intermediul modululelor „@react-navigation/material-bottom-tabs” și „@react-navigation/stack”. Astfel, ecranul de conectare

este cel dintâi, după cum se poate observa și în Figura 2-2, acesta jucând rolul unui proxy, pentru a limita accesarea aplicației fără un cont valid și nesuportat de API-ul WalletConnect.

Odată conectat în cadrul aplicației, utilizatorul este redirecționat către ecranul „Assets”, fiind ecranul principal a portofelului. Din acest ecran utilizatorul poate să deschidă fragmentul „Faucet” pentru a adăuga fonduri, să fie redirecționat către ecranele „Transfer” și „News”, prin intermediul controlului Stack, din modulul „@react-navigation/stack”, care permit accesarea ecranului anterior printr-un buton de „înapoi” (butonul „Back”), regăsit în partea stânga și superioară a ecranului.



Figură 3-2 - Navigarea între ecranele aplicației

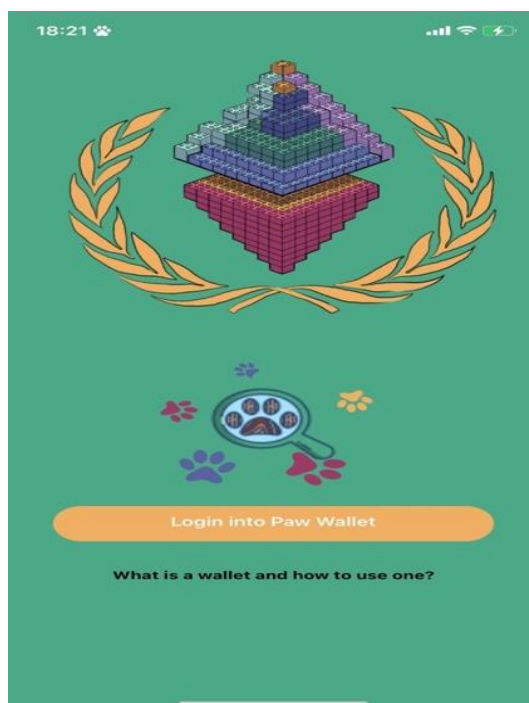
Tot de pe ecranul „Assets”, prin intermediul filelor (tab-urilor) din navigarea din partea inferioară a aplicației, utilizatorul poate accesa oricare dintre ecranele secundare, precum „Transactions”, „Graphs” sau „Learn”.

3.4. Prezentarea soluției implementate

Aplicația mobilă PawWallet este un portofel virtual pentru criptomonede, care îi permite utilizatorului să interacționeze cu unul dintre cele patru blockchainuri pentru care oferă suport, detectând Testnet-ul pe care acesta s-a conectat. Totodată, prin intermediul aplicației, spre deosebire de alte portofele virtuale existente, utilizatorul are posibilitatea să verifice fluctuațiile prețurilor criptomonedelor direct din aplicație, având acces direct la grafice, dar și posibilitatea de a-și face propria analiză tehnică bazată pe ultimele știri apărute.

Interfața grafică este alcătuită din componente atât dinamice, cât și statice, având șapte ecrane, menținute în fișiere scrise în JavaScript și TypeScript.

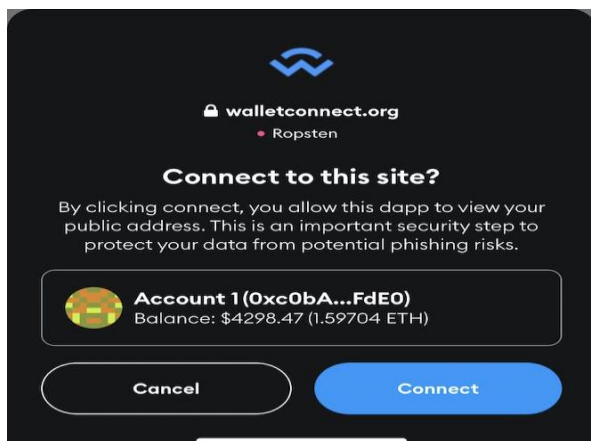
Ecranul de conectare „Login”, prezentat în Figura 3-3, ajută la conectarea în aplicație și se realizează prin intermediul a două API-uri. Prin intermediul API-ului WalletConnect, odată cu apăsarea butonului de conectare, un fragment de tip Dialog se deschide pentru a-i permite utilizatorului să aleagă un alt portofel deja existent, de exemplu MetaMask sau Rainbow, pentru ca apoi să conecteze portofelul ales în aplicația descentralizată dezvoltată.



Figură 3-3 - Ecranul „Login”

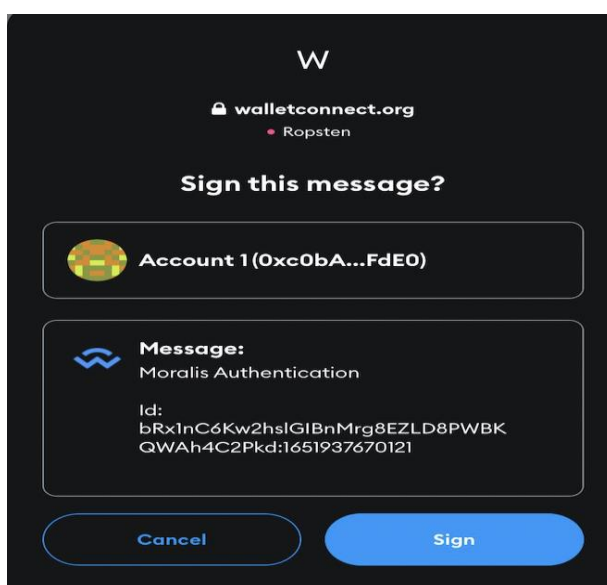
Astfel, prin intermediul Deep Linkingului și a API-ului Moralis, utilizatorul va fi redirectionat în aplicația aleasă, în care i se va cere semnarea unei tranzacții care atestă că a

fost de acord să conecteze portofelul virtual la o aplicație externă, așa cum se observă în Figura 3-4. În această notificare utilizatorul poate confirma totodată că adresa contului și rețeaua sunt cele dorite pentru conectarea în aplicație.



Figură 3-4 - Tranzacție de aprobare pentru conectarea prin intermediul API-ului WalletConnect

Prin intermediul metodelor din cadrul modului „react-moralis”, se verifică statusul conectării în aplicație. În cazul în care utilizatorul are o eroare de conectare, atunci se va face vizibil un buton, prin intermediul căruia se va deschide o componenta de tip Dialog pentru verificarea erorii de conectare, accesarea aplicației fiind mai departe restricționată.



Figură 3-5 - Tranzacție de aprobare pentru verificarea statusului autentificării și aprobarea de sincronizare a datelor prin intermediul API-ului Moralis

În cazul în care autentificarea a fost validată, utilizatorul va fi redirecționat către pagina principală. Totuși, în vederea accesării statusului conform căruia utilizatorul încă are portofelul virtual conectat, acesta trebuie să semneze o a doua tranzacție conform căreia îi permite API-ului să verifice, această tranzacție fiind prezentată în Figura 3-5.

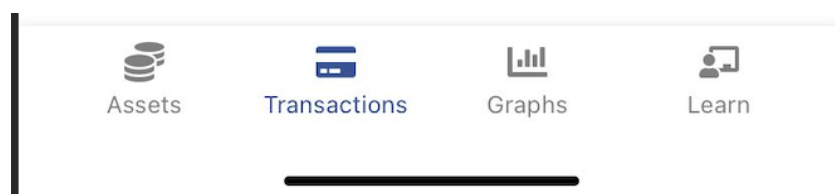
Pe ecranul de „Login” se află un control de tip LottieView, din cadrul modului “lottie-react-native”, care permite afișarea unei animații, aceasta aflându-se în format JSON, fiind utilizat cu scopul de a reda o interfață grafică cât mai plăcută utilizatorului. Acest control a fost setat să ruleze la nesfârșit și să pornească singur derularea imaginilor, prin setarea proprietăților „loop” și „autoplay”.

De asemenea, în cadrul ecranului se află un control de tip Text, care are atașat o metodă de „onPress”, reacționând deci la apăsarea textului, prin intermediul căreia utilizatorul va fi redirecționat către browserul setat principal. Mai precis, utilizatorul va fi redirecționat către un articol realizat de CoinDesk despre cum se utilizează un portofel virtual. Acest eveniment este posibil prin intermediul Deep Linkingului.

```
<Text  
  
  style={styles.infoButton}  
  
  onPress={() =>  
  
    Linking.openURL("https://www.coindesk.com/learn/your-first-crypto-wallet-how-to-use-it-and-why-you-need-one/")  
  
  }>  
  
  What is a wallet and how to use one?  
  
</Text>
```

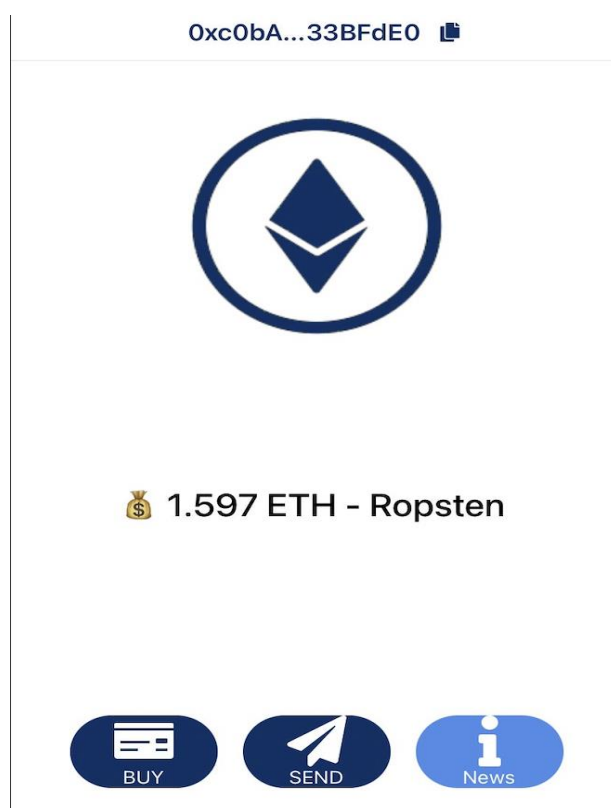
În acest ecran, au fost folosite și controale precum Image, prin care a fost atașată emblema portofelului virtual dezvoltat, controale de tip View pentru a încadra și fixa textul și imaginile, dar și controale de tip TouchableOpacity pentru a simula un buton, ce încadrează un text prin intermediul căruia utilizatorul se conectează în aplicație.

Odată conectat în aplicație, componenta de bază din React Native, App.tsx, face încărcarea unui meniu de tip Tabs Navigation, componenta „material-bottom-tabs”, din cadrul modulului „react-navigation”. Acesta conține, conform Figurii 3-6, patru butoane, Assets, Transactions, Graphs și Learn, la care, pe apăsarea unuia dintre ele, se va încarca un alt ecran.

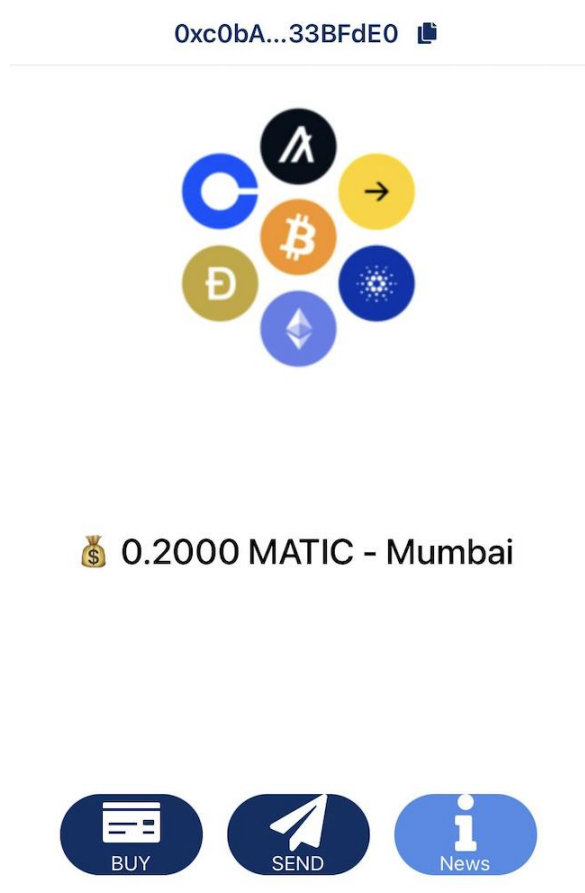


Figură 3-6 - Navigarea în aplicație prin intermediul Tab-urilor

Ecranul „Assets”, regăsit în Figura 3-7, este cel implicit, în cadrul căruia utilizatorul își poate vedea balanța pe care acesta o are în cont, în criptomoneda blockchainului ales, dar și rețeaua de Testnet pe care s-a conectat. Atât imaginea cu moneda corespunzătoare blockchainului, cât și simbolul monedei acestuia sunt încărcate dinamic, pe baza id-ului lanțului.



Figură 3-7 - Ecranul „Assets” pentru blockchain-ul Ethereum



Figură 3-8 - Ecranul „Assets” pentru blockchain-ul Polygon

În Figura 3-8, se observă cum o altă imagine a fost încărcată ca fundal al ecranului, imagine ce se încarcă dacă id-ul lanțului nu face parte din id-urile blockchainului Ethereum, fiind astfel mai mult o imagine generală și reprezentativă pentru cele mai cunoscute blockchainuri. De asemenea, balanța contului, în urma adresei schimbate, are un nou simbol, în acest caz fiind Matic, iar rețeaua conectată este Mumbai, specifică blockchainului Polygon.

```
export const getBalance = async (address) => {
  try {
    var balance;
    await web3.eth.getBalance(address, (error, wei) => {
      balance = web3.utils.fromWei(wei, 'ether')
    })
    return balance;
  } catch (error) {
    console.error(error);
  }
}
```

```

const [balance, setBalance] = useState(0);
const { walletAddress, chainId } = useMoralisDapp();

useEffect(() => {
  const fetchBalance = async () => {
    const sold = await getBalance(walletAddress);
    setBalance(sold);
  }
  fetchBalance();
}, [])

```

Pentru preluarea balanței s-au utilizat două metode, una pentru partea de client și una pentru server. În ceea ce privește metoda construită pe partea de server, aceasta este o cerere de GET, sub forma unei promisiuni („promise”), a cărei răspuns este convertit din „wei” în criptomoneda Ethereum, utilizând o metodă regăsită în cadrul modulului Web3. În client, este apelată metoda anterior descrisă, prin intermediul „hook”-urilor `useEffect` și `useState`, pentru a asigura ca întâi apelul acesteia se realizează, iar abia apoi se realizează afișarea răspunsului.

Tot în cadrul acestui ecran se regăsesc și trei butoane, numite „BUY”, „SEND” și „NEWS”, prin intermediul cărora utilizatorul poate adăuga jetoane în cont, să acceseze ecranul de transfer sau cel de știri.

Pe apăsarea butonului „BUY”, aplicația va încărca un fragment de tip Popover, din cadrul modulului „react-native-popover-view”, prin care se informează utilizatorul că va putea accesa un “robinet”, cunoscut sub denumirea de “faucet”, cel primar al Testnetului pe care îl utilizează. Prin intermediul butonului aflat în fragment, odată cu apăsarea acestuia, utilizatorul va fi redirecționat către browser, pe website-ul „robinetului”. Prin apelul metodei „getFaucet”, trimițând ca parametru id-ul lanțului, se va returna linkul potrivit Testnetului utilizatorului, folosit în Deep Linking, iar pentru a finaliza procesul de adăugare a fondurilor este nevoie să se introducă adresa contului, aceasta putând fi copiată direct din aplicație.

```

const handleGoToFaucet = () => {

  Linking.openURL(

    `${getFaucet(chainId)}`

  )
}

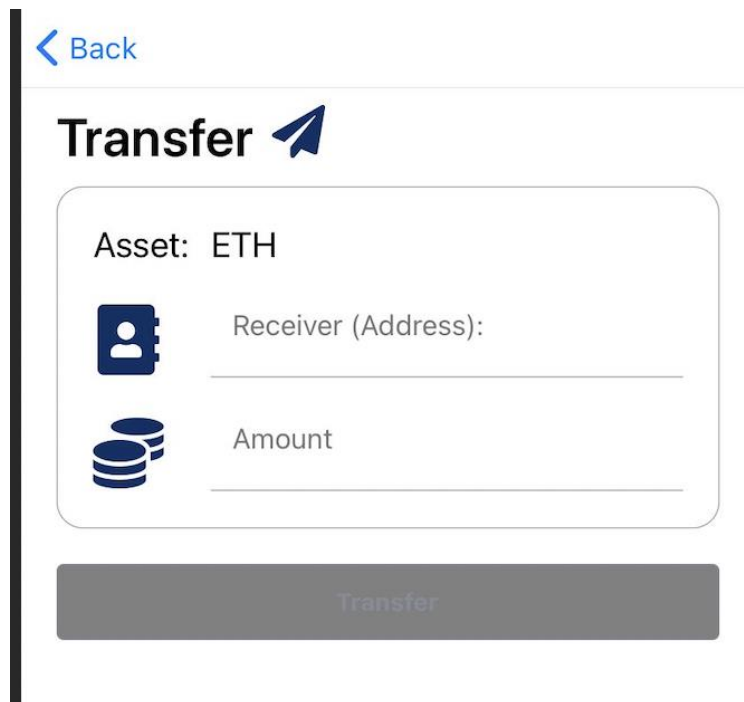
```

```
}
```

Pe baza id-ului trimis ca parametru, metoda accesează obiectul global care conține toate configurațiile implicite folosite în aplicație, precum simbolul, scannerul de pe care se citesc tranzacțiile efectuate, dar și URL-u către „robinetul” de alimentare de jetoane, cautând în elementele acestuia, id-ul primit și preluând astfel linkul corespunzător conform acestuia.

```
export const defaultConfig = {  
  
  "0x1": { symbol: "ETH", scanner: "https://etherscan.io/" },  
  
  "0x3": { symbol: "ETH", scanner: "https://ropsten.etherscan.io/", faucet:  
"https://faucet.metamask.io/" },  
  
  "0x4": { symbol: "ETH", scanner: "https://kovan.etherscan.io/", faucet:  
"https://faucet.kovan.network/" },  
  
  "0x2a": { symbol: "ETH", scanner: "https://rinkeby.etherscan.io/", faucet:  
"https://www.rinkeby.io/#faucet" },  
  
  "0x5": { symbol: "ETH", scanner: "https://goerli.etherscan.io/", faucet: "https://goerli-  
faucet.slock.it/" },  
  
  "0xa86a": { chainId: 43113, chain: "Avalanche Fuji", symbol: "AVAX", scanner: "  
https://testnet.snowtrace.io/", faucet: "https://faucet.avax-test.network/" },  
  
  "0x61": { chainId: 97, chain: "Smart Chain - Testnet", symbol: "BNB", scanner:  
"https://testnet.bscscan.com", faucet: "https://testnet.binance.org/faucet-smart" },  
  
  "0x13881": { chainId: 80001, chain: "Mumbai", symbol: "MATIC", scanner: "https://rpc-  
mumbai.matic.today", faucet: "https://faucet.polygon.technology/" },  
  
};  
  
export const getFaucet = (chain) => defaultConfig[chain]?.faucet;
```

În ecranul „Transfer” se surprinde funcționalitatea principală a unui portofel virtual și mai precis, transferul către un alt deținător de un cont cu o adresă validă în rețeaua aleasă.



Figură 3-9 - Ecranul „Transfer”

Ecranul este compus din controale din cadrul modulului „react-native”, acestea fiind mai precis View, TextInput, Text, SafeAreaView și ScrollView. Pentru a face acest ecran cât mai prietenos pentru utilizatori, așa cum se observă în Figura 3-9, au fost utilizate pictograme atât pentru evidențierea adresei necesare pentru transfer, cât și pentru TextInput-ul care trebuie completat cu valoarea tranzacției. Pentru a realiza transferul, este nevoie ca utilizatorul să introducă atât suma, cât și adresa dorită pentru ca butonul care declanșează transferul să se activeze.

```
const sendTransaction = async () => {
  try {
    const messageReceived = await sendCrypto(receiver, amount).then(() => {
      setMessage(messageReceived);
      showConfirmation();
    }).catch((error) => {
      setMessage(messageReceived);
      showErrorMessage();
    })
  } catch (err) {
    console.log(err)
  }
}

const showConfirmation = () =>
```


Mai precis, în acest obiect JSON se poate observa blocul hashului generat, numărul acestuia, de cine a fost inițializată tranzacția și către cine a fost trimisă, dar și informații precum gazul cumulat utilizat și prețul gazului.

Pentru a stiliza ecranul, s-au folosit diverse proprietăți de CSS, prin care a fost posibilă aranjarea textului cu spațieri, dar și schimbarea fonturilor și a culorilor utilizate în cadrul acestuia.

În ecranul „News”, utilizatorul poate citi ultimele știri apărute despre criptomonede, având astfel posibilitatea de a-și face propria analiză fundamentală, prin diferiți factori economici și financiari, pentru a evalua dacă o criptomonedă este sub sau deasupra valorii actuale. Acest ecran constă într-o listă de știri, redând fiecare dintre acestea ca un element separat, acestea având valori ale proprietăților specifice articolului. Aceste proprietăți includ titlul, categoria din care face articolul, țara sau țările de interes pentru articolul respectiv, dar și descrierea acestuia.

Pentru preluarea acestor date, se realizează un apel de cerere de metodă GET spre serverul NewsData, folosind modulului „axios”, utilizat pentru efectuarea unei promisiuni („promise”).

Știrile sunt introduse într-un control de tip „FlatList”, din cadrul modului „react-native”, iar fiecare element al listei este un control personalizat. Controlul personalizat, numit în acest caz „NewsItem”, conține la rândul său controale din cadrul modului „react-native”, prin intermediul cărora se afișează textul proprietăților (control de tip Text), dar se și aranjează și ordonează în cadrul ecranului (control de tip View). Toate aceste elemente au fost stilizate prin intermediul StyleSheet-ului din cadrul controlului personalizat, cu ajutorul CSS-ului, prin proprietăți precum „borderBottomWidth”, „borderBottomColor”, „margin” etc.

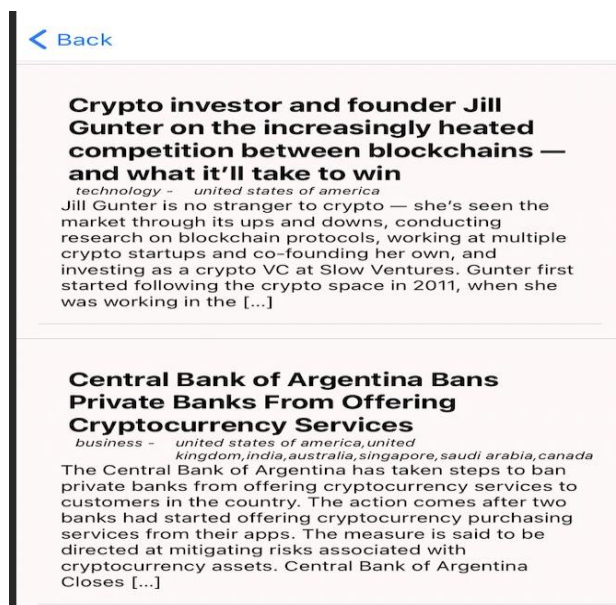
Pentru a face posibilă redirecționarea către articolul dorit, prin intermediul apăsării elementului din listă, controalele personalizate au fost încadrate în controale de tip TouchableOpacity. Astfel, apăsând pe unul dintre elementele listei, utilizatorul va fi redirecționat în browser, către sursa acestuia, unde va putea accesa și citi întregul articol.

```
const onPressGoToNews = (link) => {  
  
  Linking.openURL(  

```

```
`${link}`
))}
```

Figura 3-11 surprinde două știri, a căror titlu este scris cu bold pentru a evidenția utilizatorilor care este numele articolului. Sub titlu, se regăsesc etichetele prin care pot fi regăsite mai ușor articolele de interes a utilizatorului, dar și țările care sunt afectate de aceste anunțuri. Astfel, se pot filtra mai ușor știrile care ar putea influența piața financiară pentru a realiza o mai bună analiză care să îi protejeze astfel fondurile. În ceea ce privește descrierea articolului, o primă parte este surprinsă în fiecare control personalizat din cadrul FlatList-ului. Pentru a accesa întregul articol, utilizatorul trebuie să accese sursa originală a articolului.



Figură 3-11 - Ecranul „News”

Ecranul „Graphs” este constituit dintr-o listă, control de tip FlatList, din cadrul modulului „react-native” și un SearchBar din cadrul modulului “react-native-elements”. Elementele din cadrul listei sunt personalizate, construite pe baza răspunsului primit de la API-ul CoinGecko, prin intermediul unei cereri HTTP, mai precis al unei metode „GET”. Astfel, prin intermediul „hook”-urilor, din cadrul modulului „react”, se rețin valorile pe un model al ecranului pentru a se realiza apoi asocierea dintre valorile elementelor listei și rezultatul primit de la server.

În Figura 3-12 este prezentată lista criptomonedelor primite ca răspuns, în urma apelului către serverul CoinGecko. Elementele listei sunt personalizate, astfel încât a fost posibilă afișarea tuturor informațiilor necesare. Acest control personalizat este denumit „GraphListItem” și este compus din multiple alte controale generice pentru a reda atât un aspect plăcut al aplicației, cât și pentru a-i extinde funcționalitatea unui simplu element de listă. Pentru a evidenția capitalul criptomonedei pe piață, acestea au fost ordonate în ordine descrescătoare, în funcție de acest criteriu. Totodată, pentru a spori experiența utilizatorului, fiecare element al listei afișează schimbarea prețului în ultimele 24 de ore, simbolul criptomonedei, imaginea acesteia, dar și prețul actual.



Figură 3-12 - Ecranul „Graphs”

Searchbar-ul din cadrul ecranului îi permite utilizatorului să filtreze elementele listei după numele criptomonedelor. Controlul are atașat un eveniment de “live search”, adică o căutare în timp real, care face filtrarea instantanee asupra criptomonedelor, fără a fi nevoie ca utilizatorul să apese alt buton sau să apese tasta “Enter” de pe tastatura telefonului. Acest control se regăsește în Figura 3-13.

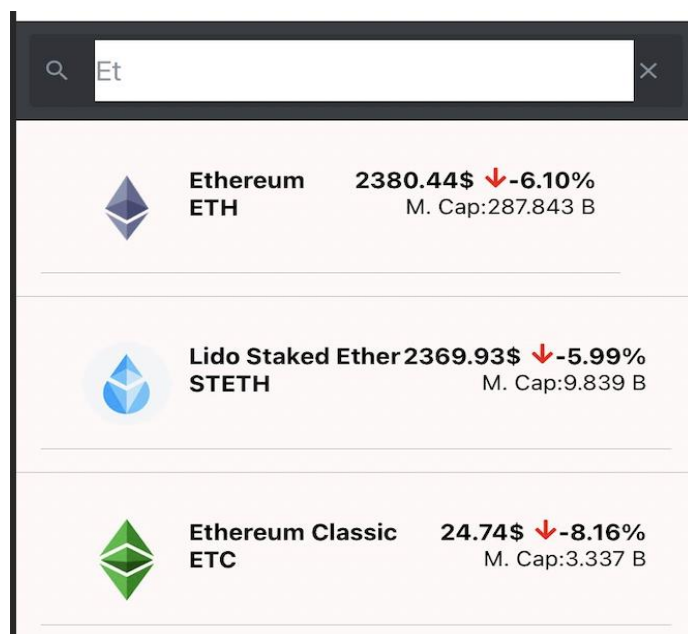
```
updateSearch = (event) => {
```

```

setSearch(event);
if(event){
  setFilteredData(data.filter(x => x.name.includes(event)))
}else if(event == "" || event == null || event == undefined){
  setFilteredData(data)
}
};

```

Metoda apelată pentru realizarea filtrării primește ca parametru o variabilă de tip „event” (eveniment), în care se regăsește textul introdus de la tastatură de către utilizator. Criptomonedele primite ca răspunsul în urma cererii către API-ul CoinGecko sunt păstrate în două useState-uri, unul dintre ele fiind denumit „filteredData”. Așadar, odată introdus textul, se filtrează datele pe baza numelui, fiind schimbată valoarea setată pe „filteredData”.



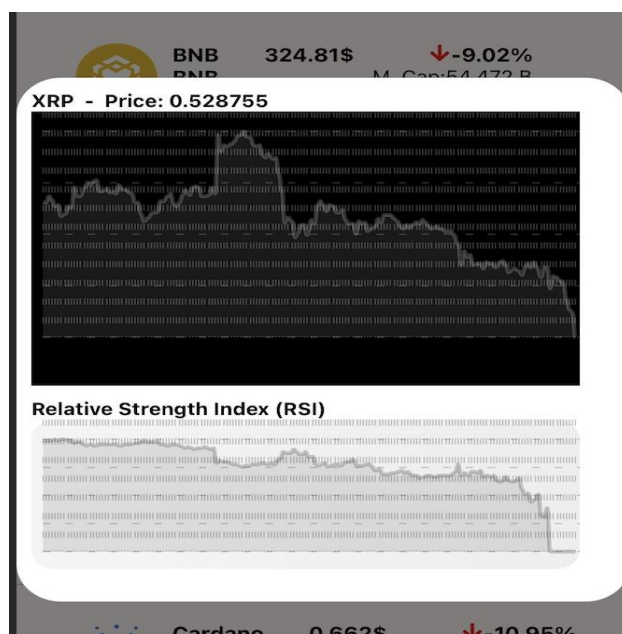
Figură 3-13 - Searchbar cu eveniment de “live search” atașat

Elementele listei personalizate sunt încadrate într-un control de tip Touchable Opacity, care îi permit utilizatorului să apese pe unul dintre elementele listei pentru a încarca graficul criptomonedei, dar și a indicatorul calculat.

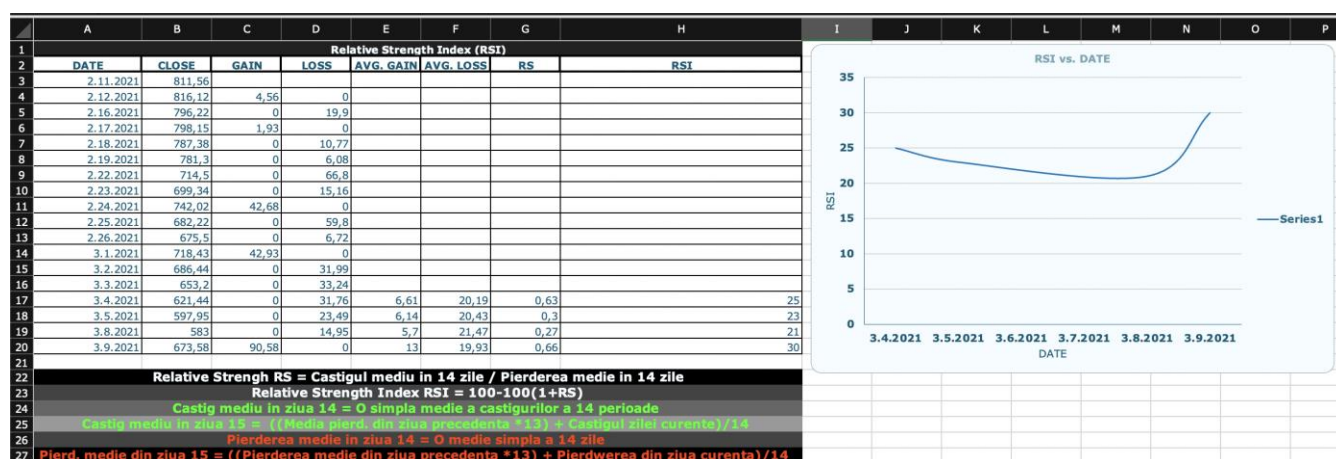
Indicatorul afișat în controlul de tip Popover, alături de grafic, este un indicator tehnic, numit Relative Strength Index, care măsoară raportul mediu dintre mișcările ascendente și descendente ale volatilității prețului criptomonedei și ia valori absolute, între 0 și 100. Scopul

acestui ecran este de a-i permite utilizatorului să-și creeze propria analiza tehnică. Astfel, acest indicator ajută utilizatorul să identifice sfârșitul unui trend.

Atât graficul, cât și indicatorul care se regăsesc în Figura 3-14, au fost construite utilizând controale din cadrul modulului „react-native-chart-kit”. Cele două grafice sunt grafice de tip LineChart, fiind personalizate prin intermediul configurării acestora, prin proprietatea „chartConfig”. Datele de intrare a graficelor se transmit prin intermediul proprietății „data”, prin precizarea setului de date, a căror valori au fost calculate odată cu încărcarea ecranului „Graphs” și salvate în două „useState”-uri.



Figură 3-14 - Graficul criptomonedei selectate și indicatorul calculat



Figură 3-15 - Indicatorul RSI calculat în Excel

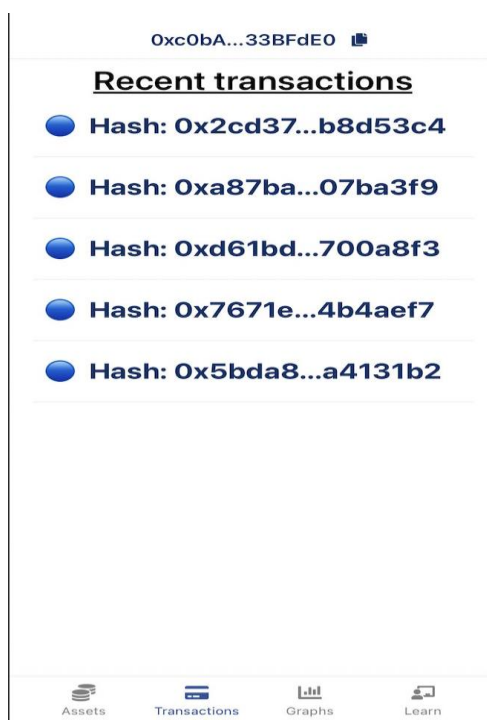
În Figura 3-15, s-a calculat indicatorul RSI pe baza formulei din cadrul acestui ecran, dorind să se evidențieze modalitatea de calculare a acestuia. Conform figurii, pentru a calcula acest indicator pe o zi, este nevoie să luăm în considerare datele pe multiple zile din trecut pentru a calcula totalul câștigurilor și totalul pierderilor. În baza acestor sume, comparându-le cu nivelul de toleranță ales, putem calcula valoarea RSI-ului. În cazul în care totalul pierderilor în modul este mai mare decât nivelul de toleranță ales, atunci se va returna direct „100”, reprezentând faptul că întreaga suma investită a fost pierdută. Altfel, se va returna media dintre totalul câștigurilor, împărțit la totalul pierderilor.

În cadrul ecranului „Transactions” , utilizatorul își poate accesa istoricul tranzacțiilor, dar și detaliile acestora. Astfel, prin intermediul unei cereri de tip POST, se preiau tranzacțiile aferente adresei primite ca parametru, la apelul funcției. Adresa care este utilizată pentru apelul cererii este preluată prin API-ul Moralis și afișată în permanență pe orice ecran. Apelul de metodă va întoarce tranzacțiile regăsite pe scannerul rețelei și a blockchainului corespunzător adresei. Mai precis, dacă utilizatorul și-a conectat un portofel virtual, bazat pe blockchainul de la Ethereum, atunci în cadrul URL-ului accesat de API se va regăsi adresa platformei Etherscan. Înregistrările obținute în urma apelului conțin informații precum hash-ul tranzacției, hash-ul blocului, numărul blocului din care acesta face parte, dar și gazul utilizat pentru realizarea tranzacției și prețul acestuia, valoarea acestora fiind convertită din Wei în jetonul Ethereum.

```
const { walletAddress, chainId } = useMoralisDapp();
const [data, setData] = useState([]);
useEffect(() => {
  const getTransactions = async () => {
    const transactionData = await getETHTransactions(walletAddress);
    setData(transactionData);
    console.log(transactionData[0])
  }
  getTransactions();
}, [])
```

În Figura 3-16 se pot observa tranzacțiile realizate cu portofelul conectat, regăsindu-se cinci tranzacții, încadrate într-un control de tip FlatList. Tranzacțiile sunt reprezentate la început cu bulină albastră pentru a ajuta utilizatorul să facă distincția între acestea.

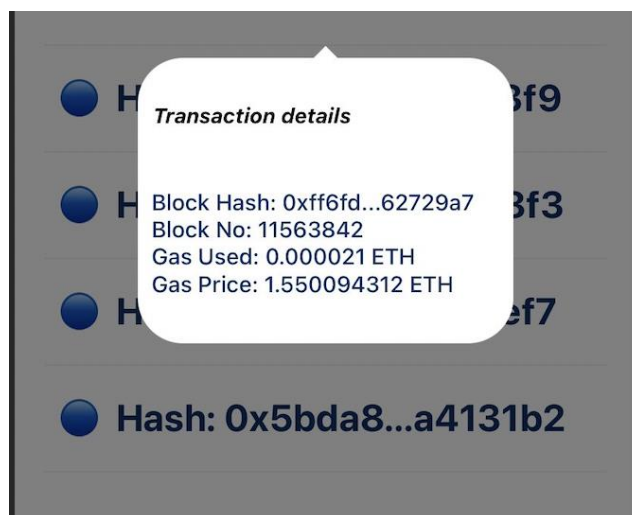
Ca interfață, ecranul este compus din controale de tip View, Text, FlatList, TouchableOpacity și Popover. Controlul de tip FlatList încarcă în mod dinamic datele primite, elementele listei fiind personalizate. Fiecare element în parte este încadrat într-un control de tip TouchableOpacity și unul de tip Popover. La atingerea elementului, se va deschide Popover-ul ce va conține informațiile adiționale tranzacției.



Figură 3-16 - Ecranul „Transactions”

Astfel, așa cum se observă în Figura 3-17, hash-ul blocului este afișat prima dată, fiind urmat de numărul acestui bloc, gazul utilizat, dar și prețul gazului. Toate aceste informații au fost formate prin intermediul CSS-ului, precum stilul fontului utilizat pentru titlul din cadrul Popover-ului, dar și culoarea utilizată pentru informațiile tranzacției.

Întrucât atât valoarea gazului consumat cât și prețul acestuia este returnat în Wei, a fost nevoie de realizarea unei conversii în criptomoneda Ethereum. Conversia fost realizată prin utilizarea metodei „Moralis.Unit.FromWei()”, din cadrul modulului „react-moralis”, aceasta primind ca parametrii valoarea în Wei și numărul de zecimale dorit pentru afișare.

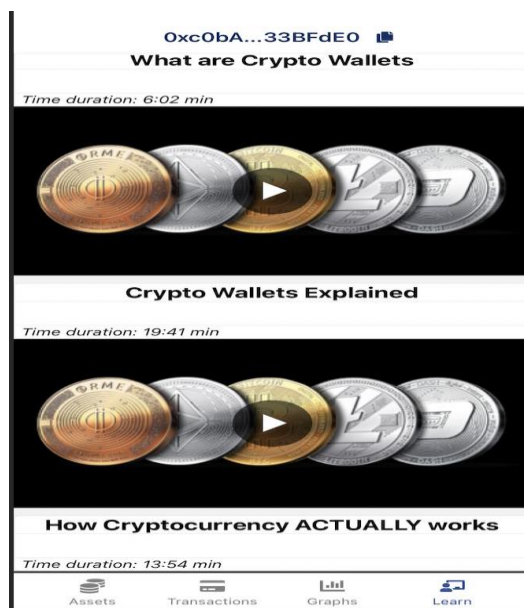


Figură 3-17 - Detaliile tranzacției alese

În ecranul „Learn”, utilizatorul are oportunitatea de a învăța despre criptomonede, portfelele de criptomonede, despre cum poate investi, cum să folosească și să interpreteze indicatorul pe care îl regăsește în ecranul “Graphs”, dar și cum poate să preconizeze prețurile criptomonedelor și să recunoască diverse trenduri.

Acest ecran a fost realizat cu scopul de a le oferi utilizatorilor o sursă rapidă și sigură de învățare, fără a fi nevoie ca aceștia să utilizeze alte aplicații. Deși numărul videoclipurilor este redus, utilizatorii pot învăța cele mai importante noțiuni direct din aplicație. Mai precis, având în vedere ca portofelele virtuale sunt relativ noi, iar mulți utilizatori nu cunosc încă modul în care îl pot utiliza, aceștia pot accesa un tutorial rapid pentru a-și extinde cunoștințele.

Ecranul este compus în mare din controale regăsite în modulul de „react-native”, precum View, Text, ScrollView, FlatList ș.a.m.d., dar și diverse controale din modulele „@ui-kitten/components” și „react-native-video-player”. Astfel, ecranul constă într-o listă de videoclipuri, propuse ca tutoriale, conform Figurii 3-18. Datele din listă sunt preluate dintr-un fișier de configurări, prin intermediul căruia se preiau informații precum URL-ul tutorialului, titlul și timpul acestuia. Prin apelarea metodei care returnează obiectul JSON cu configurațiile, odată cu încărcarea ecranului, se setează prin intermediul hook-urilor din React, useEffect și useState, datele ce vor fi folosite ca elemente ale listei.



Figură 3-18 - Ecranul „Learn”

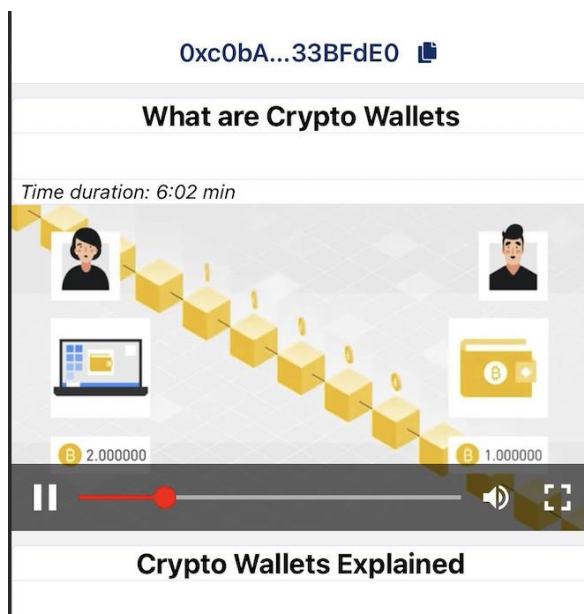
Prin intermediul controlului VideoPlayer i se permite utilizatorului să vizioneze videoclipurile direct în cadrul aplicației. Acestea sunt setate să nu pornească decât la comanda utilizatorului și să aibă volum. Toate acestea au fost setate prin proprietățile acestui control. Astfel, prin proprietatea „video” din control, s-a specificat adresa la care se regăsește videoclipul, prin proprietatea „thumbnail” s-a specificat poza cu care se afișează videoclipul, iar prin proprietățile „defaultMuted” și „autoplay”, care au fost setate pe fals, s-a indicat dacă videoclipul va avea sonor și dacă va porni automat la navigarea către acest ecran.

Pentru preluarea configurării create, se apelează metoda „fetchVideos()”, a cărui răspuns se salvează într-o variabilă de tip useState, din cadrul modulului „react”.

```
const [data, setData] = useState([]);
useEffect(() => {
  const fetchVideos = () => {
    const videoData = getVideos();
    setData(videoData);
  }
  fetchVideos();
}, [])
```

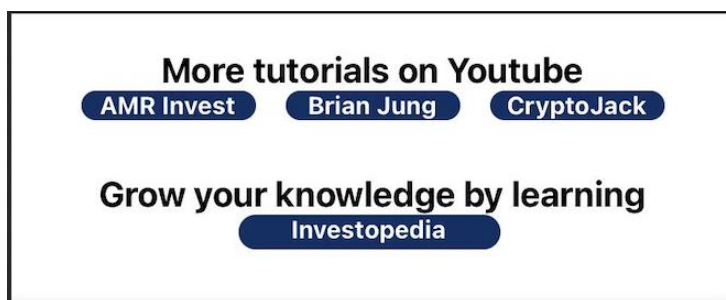
Așa cum se observă în Figura 3-19, utilizatorul poate derula videoclipul prin mișcarea cursorului către secvența dorită, având posibilitatea să oprească sau să pornească volumul

videoclipului. În cadrul acestui ecran, utilizatorul poate să vizioneze șase tutoriale oricând dorește.



Figură 3-19 - Videoclip în derulare în cadrul aplicației

La subsolul ecranului, conform Figurii 3-20, utilizatorul poate accesa atât diverse canale de Youtube, cât și să acceseze Websiteul „Investopedia” pentru a învăța mai mult despre subiectele de interes. Fiecare buton are atașat un eveniment de „onPress” care declanșează redirectionarea prin URL către Youtube sau browser, după caz.



Figură 3-20 - Subsolul ecranului „Learn”

Interfața ecranului „Learn” este simplistă, aceasta având doar informațiile necesare utilizatorului, mai precis titlu, durata videoclipului și videoclipul în sine. În ceea ce stilul ecranului, au fost folosite stilizări precum dimensiunea fontului (font-size), margine și aliniere.

4. Concluzii

În dezvoltarea aplicației Paw Wallet am utilizat tehnologii noi, moderne și din ce în ce mai căutate în domeniul tehnologiei informației, precum TypeScript, React Native și blockchain, deprinzând noi cunoștințe despre tehnologiile anterior menționate și extinzându-le pe cele cunoscute, precum arhitectura REST și limbajul JavaScript.

Prin implementarea acestei aplicații am dorit să le ofer noilor investitori de criptomonede un portofel virtual, prin intermediul căruia să aibă acces la diverse rețele și blockchainuri. Portofelul virtual dezvoltat le oferă utilizatorilor șansa de a avea o aplicație multifuncțională, prin intermediul căreia aceștia să poată învăța să folosească o astfel de tehnologie până a investi în criptomonede adevărate, utilizând jetoane de testare. Spre deosebire de alte aplicații de portofele, Paw Wallet oferă posibilitatea de adăugare de jetoane, prin redirecționarea directă către „robinetul” rețelei de Testnet, în browser.

Totodată, prin implementarea acestei soluții, am dorit să ofer posibilitatea de învățare a noțiunilor principale pe care orice investitor ar trebui să le cunoască, având astfel acces rapid și direct către resurse de învățare prin accesarea ecranului „Learn”. Pentru investitorii care doresc să aibă acces la ultimele știri și schimbări de pe piață, aceștia pot accesa ecranele „Graphs” și „News” în vederea realizării unei analize de piață.

Soluția implementată dorește să ofere o interfață intuitivă și simplă pentru a-i oferi utilizatorului o experiență cât mai plăcută, împreună cu diverse API-uri care asigură informații noi și sigure despre piață. API-urile au fost utilizate, de asemenea, și pentru a ușura implementarea funcționalității, acest fapt făcând-o ușor de întreținut în timp, dar și ușor de actualizat.

Aplicația poate să fie îmbunătățită în versiunile următoare prin realizarea unui transfer mai rapid între conturi, fiindcă în acest moment transferul se realizează în aproximativ un minut, prin adăugarea unor noi resurse de învățare precum noi tutoriale, canale de Youtube și chiar a unui dicționar a termenilor. În plus, în aplicație se pot adăuga în viitor blockchainuri adiționale care să fie suportate de aplicație, dar și posibilitatea de a avea un portofel virtual pe o rețea Mainnet.

Pentru a îmbunătăți securitatea aplicației, dar și experiența utilizatorului, ar trebui dezvoltat un portofel virtual propriu, astfel încât utilizatorul să nu mai fie nevoit să aibă o

aplicație adițională de portofel virtual, precum MetaMask, pe care să îl conecteze în aplicația Paw Wallet, evitând astfel folosirea celor două API-uri care să permită realizarea conexiunii și asigurarea menținerii sesiunii în cadrul aplicației, acestea funcționalități fiind realizate în acest moment prin intermediul API-urilor WalletConnect și Moralis.

În concluzie, implementarea soluției Paw Wallet a avut ca scop extinderea cunoștințelor acumulate în anii de studiu, prin utilizarea limbajului JavaScript, dar și învățarea unor noi tehnologii precum limbajul platformă React Native, prin intermediul căruia a fost posibilă implementarea soluției mobile într-un mod nativ, limbajul TypeScript, utilizat în mod special pentru a crește lizibilitatea codului, dar și noua tehnologie blockchain, utilizat în implementarea funcționalităților principale a portofelului virtual, precum transferul și obținerea balanței.

Bibliografie

- [1] D. Geroni, Different Types Of Crypto Wallets - Explained, <https://101blockchains.com/types-of-crypto-wallets/>, (Accesat la data de 14/06/2022).
- [2] A. Mardan, Full Stack JavaScript - Learn Backbone.js, Node.js and MongoDB, Apress, 2018.
- [3] J. Scherer, Hands-On JavaScript High Performance, Packt, 2020.
- [4] Z. Shute, Advanced JavaScript, Packt Publishing, 2019.
- [5] V. Dan, Effective TypeScript - 62 Specific Ways to Improve Your TypeScript, O'Reilly Media, 2020.
- [6] A. M. Yakov Fain, TypeScript Quickly, Manning Publications, 2020.
- [7] T. B. Ethan Holmes, Getting Started with React Native, Packt Publishing, 2015.
- [8] G. D. P. S. P. Bikramaditya Singhal, Beginning Blockchain. A Beginner's Guide to Building Blockchain Solutions, Apress, 2018.
- [9] D. G. W. Andreas M. Antonopoulos, Mastering Ethereum - Building Smart Contracts and dApps, O'Reilly, 2018.
- [10] M. Swan, Blockchain - Blueprint for a New Economy, O'Reilly, 2015.
- [11] Wikipedia, https://en.wikipedia.org/wiki/Representational_state_transfer, (Accesat la data de 14/04/2022).
- [12] M. A. S. R. Leonard Richardson, RESTful Web APIs, O'Reilly , 2013.
- [13] J. Frankenfield, Decentralized Applications(dApps), <https://www.investopedia.com/terms/d/decentralized-applications-dapps.asp>, 2021.
- [14] N. Prusty, Building Blockchain Projects, Packt, 2017.
- [15] WalletConnect, Documentation, <https://walletconnect.com/>, (Accesat la data de 22/05/2022).
- [16] Moralis, Introduction, <https://docs.moralis.io/introduction/readme>, (Accesat la data de 23/05/2022).
- [17] CoinGecko, Documentation, <https://www.coingecko.com/en/api/documentation>, (Accesat la data de 22/05/2022).

[18] OpenBase, Documentation, <https://openbase.com/js/coingecko-api/documentation>, (Accesat la data de 22/05/2022).

[19] NewsData, Documentation, <https://newsdata.io/>, (Accesat la data de 23/05/2022).

[20] Adjust, Deep Linking, <https://www.adjust.com/glossary/deep-linking/>.

Lista tabelelor

Tabel 1 - Comparație între aplicații	3
--	---

Lista figurilor

Figură 2-1 - Exemplu de cerere HTTP GET	10
Figură 3-1 - Arhitectura aplicației	13
Figură 3-2 - Navigarea între ecranele aplicației.....	15
Figură 3-3 - Ecranul „Login”	16
Figură 3-4 - Tranzacție de aprobare pentru conectarea prin intermediul API-ului WalletConnect	17
Figură 3-5 - Tranzacție de aprobare pentru verificarea statusului autentificării și aprobarea de sincronizare a datelor prin intermediul API-ului Moralis	17
Figură 3-6 - Navigarea în aplicație prin intermediul Tab-urilor	19
Figură 3-7 - Ecranul „Assets” pentru blockchain-ul Ethereum	19
Figură 3-8 - Ecranul „Assets” pentru blockchain-ul Polygon	20
Figură 3-9 - Ecranul „Transfer”	23
Figură 3-10 - Răspuns din partea server-ului după transfer	24
Figură 3-11 - Ecranul „News”	26
Figură 3-12 - Ecranul „Graphs”	27
Figură 3-13 - Searchbar cu eveniment de “live search” atașat	28
Figură 3-14 - Graficul criptomonedei selectate și indicatorul calculat	29
Figură 3-15 - Indicatorul RSI calculat în Excel	29
Figură 3-16 - Ecranul „Transactions”	31
Figură 3-17 - Detaliile tranzacției alese	32

Figură 3-18 - Ecranul „Learn”	33
Figură 3-19 - Videoclip în derulare în cadrul aplicației	34
Figură 3-20 - Subsolul ecranului „Learn”	34

Anexe

Assets.js

```
export default function Assets() {
  const { walletAddress, chainId } = useMoralisDapp();
  const [color, setColor] = useState("white");
  console.log("assets chainid" + chainId);
  const picture = ["0x1", "0x3", "0x4", "0x2a", "0x5"].indexOf(chainId) > -1 ? eth : coins;
  const handleGoToFaucet = () => {
    Linking.openURL(
      `${getFaucet(chainId)}`
    )
  }

  const navigation = useNavigation();

  const handleSendCrypto = () => {
    navigation.navigate("Send")
  }

  const handleGoToNews = () => {
    navigation.navigate("News")
  }

  return (
    <SafeAreaView style={styles.areaContainer}>
      <Image source={picture} style={{ width: 180, alignSelf: 'center', marginTop: 20 }} />
      <Balance chain={chainId} />
      <View style={styles.buttonsContainer}>
        <Popover popoverStyle={{ width: 250, height: 150, borderRadius: 30 }}
          from={() => {
            <TouchableOpacity style={styles.buttons} >
              <FontAwesomeIcon icon={faMoneyCheck} size={40} color={color} style={{ alignSelf: 'center' }} />
              <Text style={{ alignSelf: 'center', color: 'white' }} >BUY</Text>
            </TouchableOpacity>
          }} >
          <Text style={styles.faucetText}>Get coins directly from the primary faucet</Text>
          <TouchableOpacity onPress={() => handleGoToFaucet()} style={styles.faucetButton}>
            <Text style={{ alignSelf: 'center', color: 'white', padding: 5 }}>Faucet</Text>
          </TouchableOpacity>
        </Popover>
      </View>
    </SafeAreaView>
  )
}
```

```

    <TouchableOpacity style={styles.buttons} onPress={() => handleSendCrypto()}>
      <FontAwesomeIcon icon={faPaperPlane} size={40} color={color} style={{ alignSelf: 'center' }} />
      <Text style={{ alignSelf: 'center', color: 'white' }}>SEND</Text>
    </TouchableOpacity>
    <TouchableOpacity style={styles.infoButton} onPress={handleGoToNews}>
      <FontAwesomeIcon icon={faInfo} size={40} color={color} style={{ alignSelf: 'center' }} />
      <Text style={{ alignSelf: 'center', color: 'white' }}>News</Text>
    </TouchableOpacity>
  </View>
</SafeAreaView>
);
}

```

Balance.js

```

function Balance(props) {
  const [balance, setBalance] = useState(0);
  const { walletAddress, chainId } = useMoralisDapp();
  console.log(chainId)
  useEffect(() => {
    const fetchBalance = async () => {
      const sold = await getBalance(walletAddress, chainId);
      setBalance(sold);
      console.log(sold);
    }
    fetchBalance();
  }, [])

  return (
    <View style={styles.itemView}>
      <Text style={styles.crypto}>□ {parseFloat(balance).toFixed(4)} {getNativeByChain(props.chain)} - {getChain(props.chain)} </Text>
    </View>
  );
}

```

News.js

```

const News = () => {
  const [data, setData] = useState([]);
  useEffect(() => {
    const fetchMarketData = async () => {
      const marketData = await getMarketNews();
      setData(marketData.results);
    }
    fetchMarketData();
  }, [])
  return (
    <SafeAreaView style={styles.itemContainer}>

```

```

    <FlatList data={data} renderItem={({item}) => <NewsItem marketNews={item} /> } />
  </SafeAreaView>
);
};

```

NewsItem.js

```

const NewsItem = ({ marketNews }) => {
  const { category, country, description, link, title } = marketNews;

  const onPressGoToNews = (link) => {
    Linking.openURL(
      `${link}`
    )
  }

  return (
    <View style={styles.coinContainer}>
      <TouchableOpacity
        style={styles.coinContainer}
        onPress={onPressGoToNews(link)}
      >
        <View>
          <Text style={styles.title}>{title}</Text>
          <View>
            <View style={{ flexDirection: 'row' }}>
              <Text style={styles.newsDetails}>{category.join(",")} - </Text>
              <Text style={styles.newsDetails}>{country.join(",")}</Text>
            </View>
            <Text style={styles.coin} >{description}</Text>
          </View>
        </View>
      </TouchableOpacity>
    </View>
  );
}

```

Graphs.js

```

const Graphs = (props) => {
  const [search, setSearch] = useState(null);
  const [data, setData] = useState([]);
  const [filteredData, setFilteredData] = useState([]);
  useEffect(() => {
    const fetchMarketData = async () => {
      const marketData = await getMarketData();
      setData(marketData);
    }
  });

```

```

    setFilteredData(marketData);
  }

  fetchMarketData();
}, []);

updateSearch = (event) => {
  setSearch(event);
  if(event){
    setFilteredData(data.filter(x => x.name.includes(event)))
  }else if(event == "" || event == null || event == undefined){
    setFilteredData(data)
  }
};

return (
  <SafeAreaView style={styles.itemContainer}>
    <SearchBar
      placeholder="Search by name..."
      onChangeText={updateSearch}
      value={search}
      style={styles.searchbarStyle}
    />
    <FlatList data={filteredData} renderItem={({item}) => <GraphItem marketCoin={item} /> } />
  </SafeAreaView>
);
};

```

infuraBalance.js

```

const Web3 = require('web3')
const rpcURLforRopsten = "https://ropsten.infura.io/v3/ffc5617974314cec831f5b1810e40590"
const rpcURLforKovan = "https://kovan.infura.io/v3/9b6c4ef0857b450880c0d2dfe8770967"
const rpcURLforMumbai = "https://rpc-mumbai.maticvigil.com/"

export const getBalance = async (address, chainId) => {
  try {
    console.log(address)
    var web3;
    if (chainId == '0x13881') {
      web3 = new Web3(rpcURLforMumbai)
    } else if (chainId == '0x3') {
      web3 = new Web3(rpcURLforRopsten);
    } else if (chainId == '0x4') {
      web3 = new Web3(rpcURLforKovan);
    }

    var balance;

```

```

    await web3.eth.getBalance(address, (error, wei) => {
      console.log(wei)
      balance = web3.utils.fromWei(wei, 'ether')
    })
    return balance;
  } catch (error) {
    console.error(error);
  }
}

var Tx = require("ethereumjs-tx").Transaction
export const sendCrypto = async (receiver, amount) => {
  try {
    if(amount.includes(",")){
      amount = amount.replace(",", ".");
      console.log(amount)
    }
    const privateKey = /* cheia privată */
    var web3 = new Web3(rpcURLforRopsten);
    const SignedTransaction = await web3.eth.accounts.signTransaction({
      to: receiver,
      value: web3.utils.toHex(web3.utils.toWei(amount, 'ether')),
      gas: web3.utils.toHex(21000)
    }, privateKey );
    var responseToSend;
    web3.eth.sendSignedTransaction(SignedTransaction.rawTransaction).then((receipt) => {
      console.log(receipt);
      responseToSend = receipt;
    }).catch(err => console.log(err.message));
    return responseToSend;
  } catch (error) {
    console.error(error);
  }
}

```

retrieveNews.js

```

export const getMarketNews = async () => {
  try {
    const response = await
    axios.get("https://newsdata.io/api/1/news?apikey=pub_63872102fb8fa507ba957f27072faf4de867&q=crypto&language=en ");
    const data = response.data;
    return data;
  } catch (error) {
    console.error(error);
  }
}

```

```
}
```

retrievetransactions.js

```
export const getETHTransactions = async (address) => {
  try {
    console.log(address);
    const URL = "https://api-ropsten.etherscan.io/api?module=account&action=txlist&address=" + address +
"&sort=asc"
    const response = await axios.get(URL);
    const data = response.data.result;
    return data;
  } catch (error) {
    console.error(error);
  }
}
```

RecentTransaction.js – TransactionItem

```
const TransactionItem = ({ item }) => {
  return (
    <Popover popoverStyle={{ width: 250, height: 190, borderRadius: 30 }}
    from={{
      <TouchableOpacity>
        <Item
          hash={item.hash}
        />
      </TouchableOpacity>
    }} >
      <Text style={styles.info}>Transaction details</Text>
      <View style={styles.itemView}>
        <View style={{ flex: 1, justifyContent: "center" }}>
          <Text style={styles.text}>Block Hash: {getEllipsisTxt(item.blockHash, 7)}</Text>
          <Text style={styles.text}>Block No: {item.blockNumber}</Text>
          <Text style={styles.text}>Gas Used: {parseFloat(Moralis.Units.FromWei(item.gasUsed, 9))}</Text>
          <Text style={styles.text}>Gas Price: {parseFloat(Moralis.Units.FromWei(item.gasPrice, 9))}</Text>
        </View>
      </View>
    </Popover>
  );
};
```