UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA

DE

MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

Allbank fintech – aplicație Android

Coordonator științific

Prof. dr. Alin Ștefănescu

Absolvent

Manolache Andrei

București, iunie 2021

Cuprins

Introducere

„Fintech” - o combinație a termenilor „Financial” și „Technology” - este o denumire ce descrie orice tehnologie ce revoluționează felul în care se efectuează serviciile financiare. Conceptul urmărește automatizarea și eficientizarea modului în care are loc interacțiunea utilizatorului cu produsele bancare la care acesta are acces prin intermediul unei aplicații software. Multe din produsele fintech sunt proiectate astfel încât să faciliteze felul în care orice entitate (simplul utilizator, companie, instituție) folosește serviciile bancare și care îi permit cu ușurință și în siguranță să facă operațiunile dorite. Toate aceste noutăți fac parte din evoluția serviciilor financiare, de la modalitățile prin care un utilizator poate intra online pentru a-și verifica cheltuielile, la posibilitatea de a analiza performanța investițiilor în timp real și până la tehnologii care permit instituțiilor să ia decizii rapide de creditare.

Inițial, conceptul de fintech desemna tehnologiile de back-end folosite în cadrul instituțiilor bancare, însă odată cu dezvoltarea internetului și a tehnologiilor web, acesta a cunoscut o orientare către nevoile consumatorului de rând și include mai multe subdomenii precum: Blockchain, Fundraising, Home Banking etc.

Un studiu [1] susține că 64% dintre persoane a folosit o soluție de tip fintech, o creștere de la 16% în 2015 și 33% în anul 2017. Principalele arii de utilizare sunt „Money transfer ad payments”, „Savings and Investments” și „Budgeting and financial planing”, după cum reiese din figura 1.

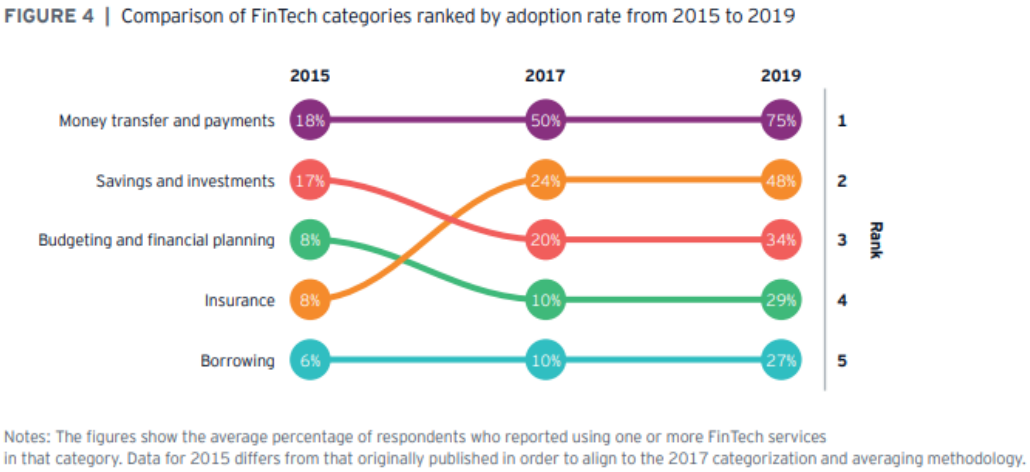

Figura 1 Comparația adopției produselor fintech în perioada 2015-2019

Ținând cont de evoluția pe care acest domeniu a cunoscut-o în ultimii ani și de faptul că este într-o continuă dezvoltare, lucrarea va prezenta o aplicație mobilă care să faciliteze accesul utilizatorului la datele financiare ale acestuia, dezvoltând mai multe funcționalități: administrarea facilă a conturilor bancare din cadrul mai multor bănci și a tranzacțiilor corespunzătoare, obținerea unor statistici detaliate bazate pe tranzacțiile efectuate, gestionarea bugetului cheltuielilor pe o anumită perioada de timp etc.

În primul capitol vor fi prezentate aspectele teoretice, limbajele și tehnologiile utilizate la implementarea aplicației. În următorul capitol este prezentată structura proiectului. Sunt descrise arhitectura, componentele și funcționalitățile sistemului, schema bazei de date și este motivată necesitatea unei astfel de aplicații fintech în contextul progresului tehnologiilor web. Capitolul trei prezintă funcționalitățile aplicației și modurile în care aceasta poate fi folosită, acompaniate de imagini care să ilustreze scenariile de utilizare, urmat de concluzia lucrării, ce recapitulează componentele de structură ale aplicației.

1. Preliminarii și noțiuni teoretice

1.1. PSD2

PSD2 [2] este o directivă europeană, adoptată de Parlamentul European în 2015, ce urmărește să creeze un sistem de plăți integrat la nivel european, sprijinind inovația tehnologică și creșterea nivelului de securitate al plăților digitale. Obiectivele acestei reglementări sunt încurajarea concurenței și inovării în domeniul serviciilor financiare. Astfel, se urmărește externalizarea serviciilor bancare prin care aplicații terțe, autorizate de utilizatori, să utilizeze conturile și datele clienților, prin intermediul unor API-uri puse la dispoziție de instituțiile bancare.

Această directivă europeană stă la baza lucrării, aplicația fiind dezvoltată integrând API-urile puse la dispoziție de bănci. Fiecare entitate bancară trebuie să facă publice o serie de trei API-uri:

- AIS – Account Information Service – se ocupă de preluarea informațiilor despre conturile utilizatorilor. Include diferite rute pentru accesarea detaliilor (nume, cod iban, monedă, bilanț etc) unui anumit cont sau a tuturor conturilor accesibile, precum și lista de tranzacții corespunzătoare
- PIS – Payment Initiation Service – se ocupă de inițierea de tranzacții către un alt cont bancar
- PIIS - Payment Instrument Issuer Service – verifică dacă un anumit cont bancar deține suficiente fonduri

1.2. Android

Android este un sistem de operare, bazat pe o versiune a nucleului Linux, destinat dispozitivelor mobile, ce permite dezvoltatorilor să scrie cod în limbajul Java. Pentru compilarea și rularea programului, este folosit kit-ul de dezvoltare software Android SDK. Este cel mai răspândit sistem de operare destinat dispozitivelor mobile, cuprinzând aproximativ 85% din piața globală [3].

Platforma Android a fost lansată în 2007 de către Open Handset Alliance [4] și chiar dacă aplicațiile găzduite de această platformă sunt dezvoltate în Java, interpretarea lor nu este făcută de JVM [5], ci de o altă mașină virtuală numită ART (Android Runtime) ce înlocuiește DVM (Dalvik Virtual Machine) începând cu versiunea de Android 5.0. [6]

1.3. Java

Java [7] este un limbaj de programare orientat-obiect popular încă de la apariția sa de la mijlocul anilor ’90. Notorietatea sa este dată de o serie de avantaje precum:

- Portabilitate – datorită JVM, limbajul poate rula pe majoritatea sistemelor de operare
- Versatilitate – programele dezvoltate în Java au diferite destinații: aplicații web, sisteme software (pe majoritatea platformelor), servere etc
- Robustețe – sunt eliminate surse care pot genera ușor erori precum pointerii, este introdusă administrarea automată a memoriei (apariția garbage collector-ului)
- Scalabilitate – extinderea sistemelor dezvoltate în Java se realizează cu ușurință prin prisma proiectării sale

Pentru ca Java să fie un limbaj independent de platformă, a fost introdus conceptul de JVM (Java Virtual Machine) care este un mediu de execuție ce se ocupă de compilarea codului (transformarea surselor în byte-code), urmat de interpretarea lui de mediul Java.

Ușurința cu care sunt dezvoltate sisteme software complexe și puterea acestui limbaj reprezintă principalele motive pentru care sistemul de operarea Android permite dezvoltatorilor să scrie codul în limbajul Java.

1.4. Node.js

Node.js [8] reprezintă o tehnologie ce permite executarea codului Javascript prin intermediul motorului V8 [9] în alte medii în afară de web browser. Permite dezvoltarea facilă

și rapidă a serverelor web sau a altor instrumente de rețea în Javascript, fapt pentru care se bucură de o popularitate ridicată.

Tehnologia este bazată pe o arhitectură bazată pe evenimente, instrucțiunile putând fi executate asincron. Aceste decizii de proiectare a sistemului sunt menite să îmbunătățească randamentul și scalabilitatea aplicațiilor online.

1.5. Express

Express este o bibliotecă folosită în proiectele realizate în Node.js ce include funcționalități minimale pentru a dezvolta aplicații web [10]. Asigură mecanisme de:

- manipulare a request-urilor prin definirea rutelor și verbelor HTTP
- adăugare de middleware (software intermediar ce se execută înaintea altor blocuri de cod) în cadrul procesului de gestionare a request-urilor
- analizare facilă a request-urilor HTTP pentru prelucrarea de cookie-uri, parametrilor sau corpului cererii

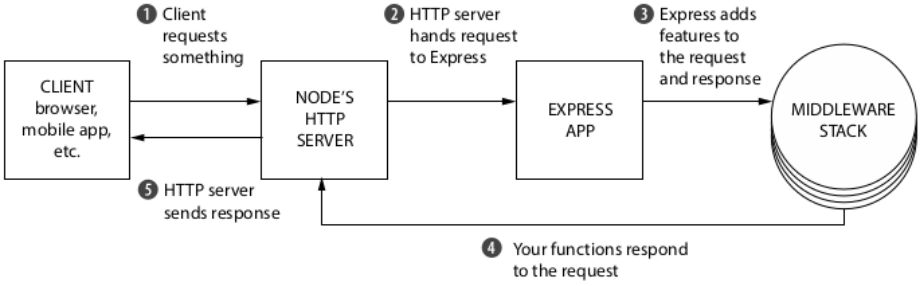

Figura 1.5.1 Arhitectura framework-ului Express [11]

În figura 1.5.1 este prezentată arhitectura bibliotecii Express, prezentând succesiunea de instrucțiuni prin care este trecut și filtrat fiecare request HTTP. Clientul trimite o cerere către server care este redirecționată către componenta Express, fiind prelucrat în prealabil de middleware-urile puse pe ruta respectivă. Se execută funcția corespunzătoare rutei Express, urmată de trimiterea unui răspuns HTTP înapoi către aplicația client.

1.6. Android Jetpack

Android Jetpack prezintă un set de biblioteci conceput pentru a ajuta programatorii să urmeze cele mai bune practici în procesul de dezvoltare a codului, astfel încât să fie unul fiabil și consistent pe dispozitivele Android.

Printre conceptele de bază al unei aplicații Android este componenta Activity ce este responsabilă de afișarea conținutului pe ecran prin elementele vizuale, precum și înregistrarea comenzilor date de utilizator. O aplicație poate include mai multe activități, însă doar una poate fi afișată pe ecran. Ciclul de viață ale unei activități este prezentat în figura 1.6.1, trecând prin mai multe stări, în funcție de context:

- onCreate() – apelată atunci când activitatea este creată
- onStart() – apelată atunci când urmează să fie atașată
- onResume() – se apelează când activitatea urmează să fie vizibilă
- onPause() – apelată atunci când o nouă activitate este creată și îi ia locul activității vechi
- onStop() – apelată atunci când nu se mai utilizează activitatea curentă în defavoarea alteia
- onDestroy() – apelată atunci când activitatea nu mai este folosită și este distrusă, iar memoria se eliberează

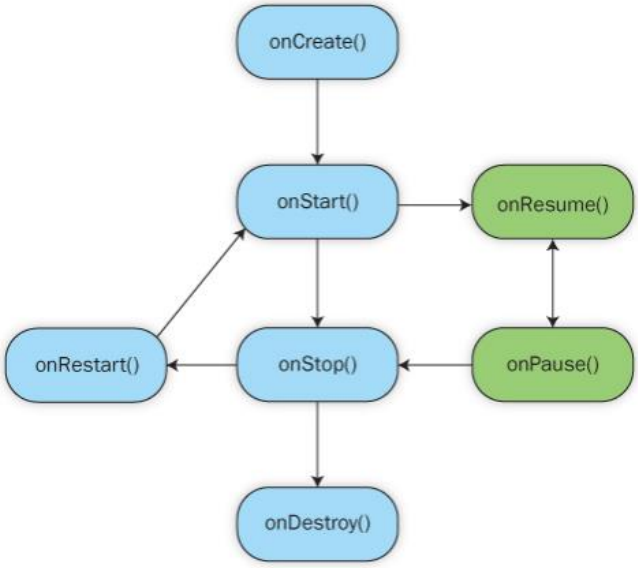

Figura 1.6.1 Ciclul de viață al unei activități [12]

Un Fragment reprezintă o porțiune din interfața unei activități, ciclul de viață al acestuia depinzând de cel al activității. Principalul avantaj al folosirii fragmentelor este faptul că e mai puțin costisitor față de o activitate din punctul de vedere al resurselor și poate fi reutilizat ușor în alt context. În figura 1.6.2 este prezentată activitatea în aplicația Android, punând în evidență faptul că mai multe fragmente pot fi atașate unei singure activități.

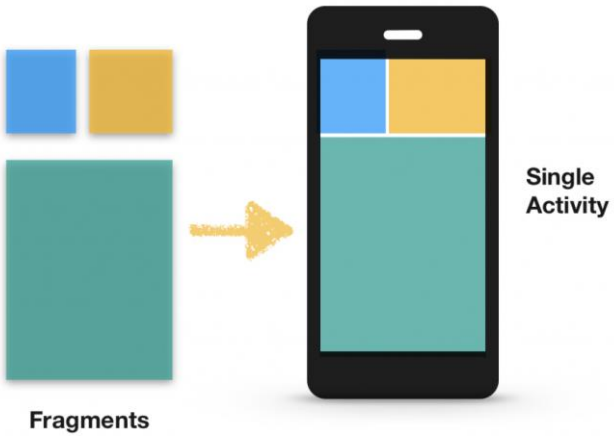

Figura 1.6.2 Distribuirea fragmentelor într-o activitate

Introducerea Navigation Component a permis dezvoltarea mult mai rapidă a aplicațiilor Android, aceasta permițând implementarea navigării prin ecranele aplicației într-un mod facil.

Pentru aceasta, dezvoltatorul trebuie să atașeze secvențial fragmentele într-o singură activitate și să definească graf-ul de navigare între destinațiile fragment, asemănător cu cel din figura 1.6.3.

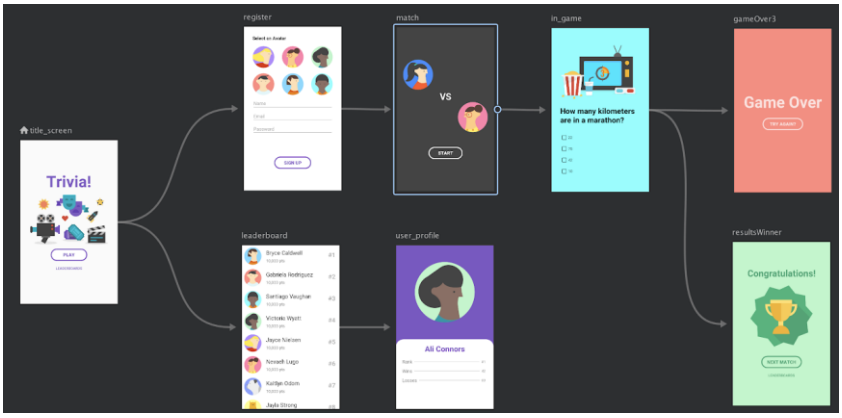

Figura 1.6.3 Graful de navigare

Navigation Component asigură navigarea dintre 2 fragmente adiacente definite în graf, precum și alte mecanisme precum definirea animațiilor dintre acestea, apăsarea butonului back ce întoarce utilizatorul către fragmentul precedent, administrarea stivei de fragmente etc.

Room [13] este o librărie de tip ORM (Object-relational Mapper) ce se ocupă de interacțiunea cu baza de date în mecanisme precum corelarea directă a tabelelor în obiecte cu care se poate lucra ușor sau asigurarea persistenței codului în ciuda modificărilor schemei din baza date. Aceasta acționează ca un nivel de abstractizare peste baza de date SQLite utilizată de Android, astfel obținându-se un mod simplificat de a converti și prelucra informații între tabele relaționale și obiecte.

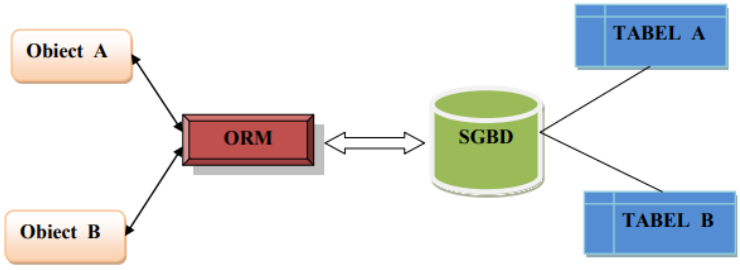

Figura 1.6.4 Legătura dintre un SGBD și aplicație

În figura 1.6.4 este ilustrată legătura pe care o face un ORM dintre o bază de date și logica aplicației. Se observă că tabelele A și B se asociază cu câte un obiect corespunzător unei înregistrări din fiecare tabel.

Material Design Components [14] reprezintă o altă bibliotecă ce se concentrează pe stilizarea aplicației. Ea oferă un set bine definit de stiluri și recomandări astfel încât elementele vizuale să fie vizibile și ușor accesibile, iar imaginile, butoanele și graficele sunt sugestive și anticipează funcționalitățile oferite. Funcționalitățile sunt concepute și distribuite de-a lungul ecranelor astfel încât să fie maximizată interacțiunea cu sistemul, iar navigarea prin aplicație să fie fluidă.

1.7. Sequelize

„Sequelize este o librărie de tip ORM [15] ce oferă posibilitatea de a introduce instrucțiuni din terminal folosind comanda *sequelize-cli*. Folosind-o, librăria poate genera direct tabelele entitate, creând inițial obiectele model, urmate de crearea tabelelor prin realizarea unei migrații. Pentru a asigura dezvoltarea continuă a aplicației, dar și menținerea suportului pentru versiuni mai vechi ale acesteia, Sequelize oferă posibilitatea de a memora mai multe migrații pentru baza de date, astfel încât să se asigure persistența acesteia. Biblioteca oferă suportul necesar stabilirii tipurilor de date corespunzătoare câmpurilor și a relațiilor între tabele (one-to-many, one-to-one, many-to-many).

Sequelize accelerează procesul de dezvoltare al aplicațiilor, făcând codul consistent, ușor de interpretat și dezvoltat, abstractizează motorul bazei de date (astfel încât se poate face ușor trecerea între două baze de date diferite) și asigură suport pentru versionarea bazelor de date prin intermediul migrațiilor.

1.8. Retrofit

Retrofit [16] este o librărie destinată platformei Android ce permite aplicațiilor să autorizeze request-uri ce respectă standardul REST [17] către servere web externe sau să interacționeze cu servicii API. Include serializarea și deserializarea conținutului prin folosirea

librăriei GSON pentru obiectele de tip json, dar se poate utiliza și un convertor personalizat pentru alte tipuri precum XML.

Pentru utilizarea acestui framework, este necesară definirea a trei componente:

- Instanța de Retrofit – este necesară crearea unui obiect Retrofit folosind pattern-ul Builder care are nevoie de URL-ul de acces la serviciul web extern și metoda de serializare și convertire a răspunsului primit
- Interfața – cu ajutorul ei sunt definite endpoint-urile la care se accesează serviciile externe
- Clasele model – folosite pentru a prelua informațiile din răspunsul primit

Figura 1.8.1 Schema unui endpoint Retrofit

În figura 1.8.1 am ilustrat structura unui endpoint Retrofit. Structura acestuia este:

1. Metoda este adnotată cu tipul metodei HTTP (GET în exemplu) și este precizat și numele endpoint-ului, alături de parametri
2. Este precizată numele metodei, alături de tipul de date primit ca răspuns după ce a fost prelucrat de convertor-ul obiectului Retrofit (în cazul de față o lista de obiecte de tip UserData)
3. Sunt precizați parametrii din endpoint, atât cei din query, cât și cei din path. Adițional, pot fi definiți și parametrii din antet

Clientul de Retrofit lucrează asincron, deci request-ul se face de pe un alt thread, iar cererea făcută către server poate să nu se realizeze din cauza unor erori tehnice (caz în care se execută metoda onFailure), respectiv poate să aibă loc (caz în care se execută metoda onResponse) și se prelucrează răspunsul primit.

1.9. OAuth2

OAuth2 [18] este un protocol de autorizare ce permite unor aplicații accesul limitat la datele utilizatorului din cadrul unui serviciu terț, fără a dezvălui aplicației client datele de acces ale acestuia. Pentru a autoriza accesul la API-ul folosit, aplicația trebuie să se înregistreze la serviciile serverului web extern, furnizând detalii despre ea (numele aplicației, domeniul, URL-ul de callback la care să fie redirecționat răspunsurile din partea server-ului etc), iar acesta furnizează aplicației client un set de date de identificare (de obicei id și parolă) pe care aceasta le folosește când inițiază procesul de autorizare pentru un utilizator. În momentul în care autorizarea s-a terminat cu succes, serverul extern generează un token de acces limitat pe care aplicația client îl va folosi să acceseze resursele serviciului terț. Fiecare token are un set de permisiuni restrânse care sunt autorizate de utilizator în momentul autentificării acestuia.

Există diferite tipuri de autorizare: tipul implicit sau autorizarea cu cod de acces. Autorizarea cu cod de acces este cea mai întâlnită deoarece a fost concepută să fie folosită de back-end-ul aplicațiilor deoarece nu este expusă public parola aplicației.

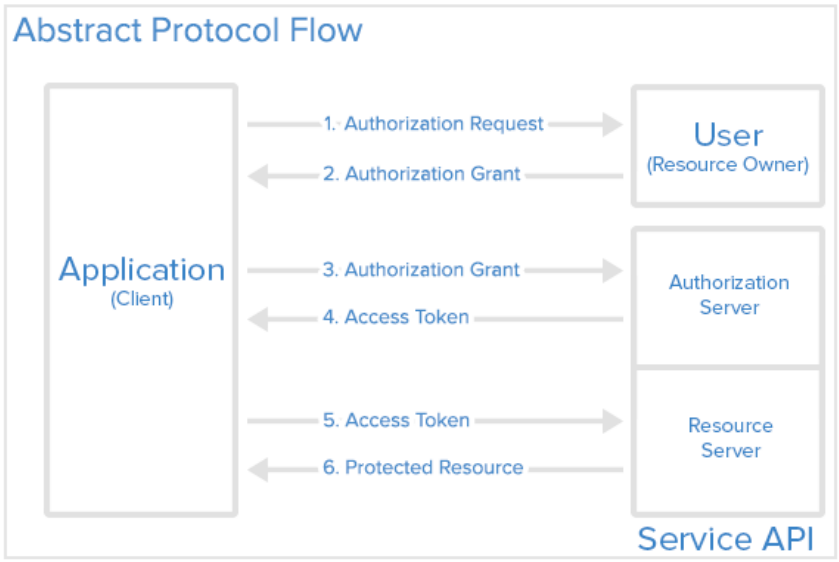

Figura 1.9.1 Schema OAuth2 de autorizare cu cod [19]

În figura 1.9.1 este prezentată schema OAuth2 cu cod de autorizare. Aplicația client inițiază procesul de autorizare, furnizând id-ul și parola primite în momentul înregistrării acesteia, iar utilizatorul se autentifică și permite accesul aplicației la resursele sale. Serviciul extern redirecționează aplicația către URL-ul cu care s-a înregistrat și îi oferă codul de

autorizare. În continuare, back-end-ul aplicației trimite un request către serverul extern în care schimbă codul de autorizare pe un token de acces. Astfel, s-a terminat procesul de autorizare, iar aplicația poate să acceseze resursele utilizatorului folosind token-ul primit.

1.10. Cod QR

Codul QR este un mod ușor de codificare a informației sub forma unei matrice formată din pătrate albe și negre. În general, acesta este folosit pentru partajarea de conținut text, deschiderea unei pagini web noi, conectarea la o rețea wireless etc. Există multe generatoare și cititoare de coduri QR. În această lucrare, este inclusă funcționalitatea de generare și citire a unui cod QR, iar pentru aceasta, am folosit librăriile *com.google.android.gms:play-services-vision:20.1.3* (dezvoltată de Google), respectiv *androidmads.library.qrgenearator: QRGenearator:1.0.3* [20] care este o bibliotecă open-source.

1.11. Json Web Token

Json Web Token [21] este un standard de transmitere a informației, semnată digital, sub forma de obiect JSON, alcătuit din 3 blocuri, ilustrate în figura 1.10.1:

1. Header – Obiect json cu 2 proprietăți: tipul algoritmului folosit la criptarea conținutului și tipul tokenului
2. Payload – Obiect Json cu mai multe proprietăți standard sau puse de programator, în funcție de necesități, precum Issuer (entitatea care a semnat token-ul), Subject (subiectul token-ului), Exp (momentul de tip la care expiră tokenul), Iat (momentul de timp la care expiră valabilitatea token-ului) etc
3. Semnătura – Este un cod MAC folosit pentru verificarea integrității token-ului. Se obține din concatenarea antetului și a payload-ului (în format Base64UrlEncoding, despărțite prin ‘.’), iar asupra textului obținut se aplică algoritmul specificat în antet cu cheia secretă aleasă. Cele 3 părți se codifică în formatul Base64UrlEncoding, iar rezultatele se concatenează și se despart prin ‘.’

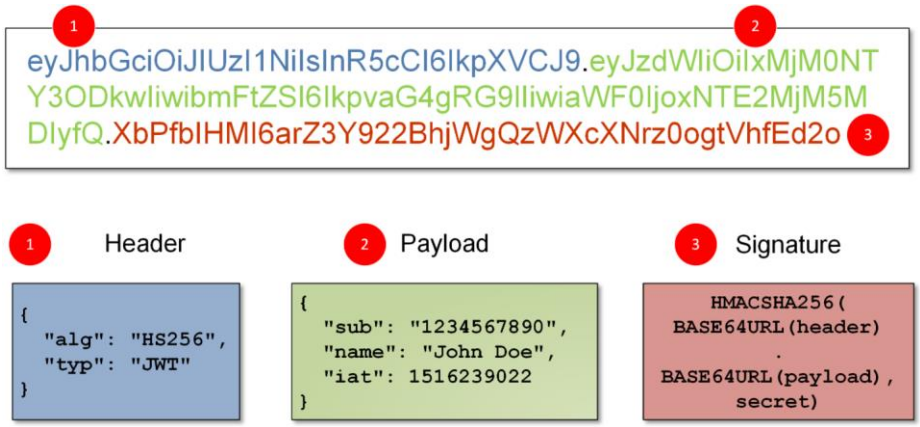

Figura 1.10.1 Structura unui Json Web Token [22]

Pentru partea de validare a token-ului, se obține algoritmul de criptare (prin decodificarea antetului), se concatenează primele 2 blocuri (antetul si payload-ul), despărțite prin '.', după care rezultatul se trece prin algoritmul de criptare cu cheia secretă. Dacă semnătura coincide cu rezultatul final, atunci token-ul este valid.

Puterea conceptului de JWT este dată de puterea algoritmilor prin care are loc semnarea conținutului. Având un JWT, conținutul payload-ului este vizibil pentru toată lumea care-l deține, deci nu este recomandat ca acesta să conțină date sensibile (parole, numărul cardului de credit etc), deci partea de stabilire a integrității sale se realizează prin validarea semnăturii.

2. Descrierea aplicației

2.1. Conceptul și utilitatea aplicației

Aplicația prezentată în această lucrare este destinată dispozitivelor mobile ce rulează pe platforma Android și are rolul de a facilita accesul utilizatorului la propriile date bancare, precum și obținerea unor informații detaliate cu privire la gestionarea acestora.

În prezent, entitățile bancare pun la dispoziția clienților accesul la produse de tip mobile banking care le oferă posibilitatea de a realiza operațiunile financiare folosind dispozitivele mobile (consultarea bilanțului curent, generarea unui extras de cont, efectuarea unei plăți noi etc), acest lucru necesitând instalarea unei aplicații.

În cazul în care o persoană are conturi deschise la mai multe bănci, este necesară instalarea mai multor aplicații, ceea ce înseamnă mai puțin spațiu de stocare neocupat în memoria internă a telefonului și reducerea resurselor rămase disponibile, lucru care va duce la performanțe scăzute. Astfel, aplicația a fost concepută pentru a evita acest inconvenient, mizând pe faptul că utilizatorul va putea gestiona conturile din cadrul mai multor bănci prin intermediul acesteia. Utilitatea ei este dată de micșorarea spațiului de stocare ocupat deoarece nu mai este necesară instalarea a numeroase alte aplicații pe dispozitiv, precum și de reducerea numărului de pași în momentul în care o persoană dorește să realizeze operațiuni din mai multe conturi bancare, folosind câte o aplicație diferită pentru fiecare instituție financiară, iar utilizatorul va câștiga timp.

O altă funcționalitate o reprezintă gestionarea cheltuielilor care se împarte în două componente: cea de economisire și cea de detaliere prin statistici și grafice. Aplicația îi oferă utilizatorului date despre cheltuielile făcute, având posibilitatea de a le personaliza (filtrare după numele băncii, data și categoria tranzacției la care s-a efectuat etc). Astfel, aplicația capătă și o latură de controlare a cheltuielilor, complementară cu prima funcționalitate amintită unde se punea accentul pe ușurarea accesibilității conturilor bancare.

Aplicația dezvoltată în acest scop îndeplinește funcționalitățile prezentate și are următoarele opțiuni:

- Filtrarea tranzacțiilor după tip (cheltuieli sau venituri) și accesarea detaliilor
- Obținerea de statistici (totalul și media zilnică a cheltuielilor și veniturilor, diferența dintre venituri și cheltuieli) filtrate după perioadă și bancă
- Vizualizarea de grafice cu totalul cheltuielilor într-o anumită perioadă de timp, împărțită pe categorii și afișarea acestora
- Posibilitatea de creare a unor obiective de economisire, clasificate pe categorii
- Adăugarea de noi tranzacții în două moduri: introducerea datelor destinatarului (cu listă de sugestii în funcție de datele introduse) sau prin scanarea de cod QR
- Generarea unui cod QR cu datele necesare unei tranzacții care poate fi salvat în telefon sau distribuit prin alte aplicații

2.2. Descrierea componentelor aplicației

2.2.1. Structura proiectului

Aplicația este formată din două proiecte folosind tehnologii diferite. Partea de client a fost realizată în IDE-ul Android Studio, folosind tehnologiile specifice platformei Android: Java, Jetpack Compose, Material Design, Room, Retrofit, iar componenta serverului a fost dezvoltată în Visual Studio Code cu ajutorul Javascript, NodeJS și Sequelize. Cele două proiecte sunt modularizate, componentele fiind grupate pe baza funcționalităților oferite.

Structura aplicației Android este:

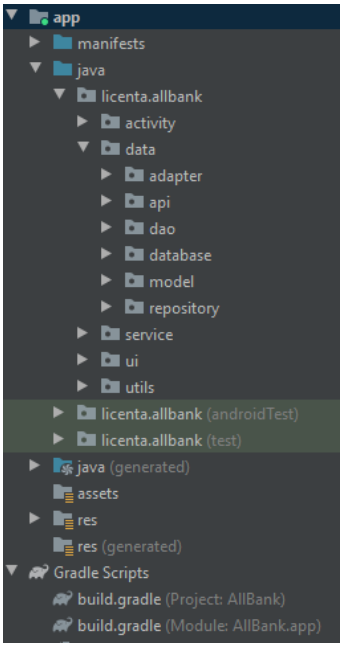

Figura 2.2.1.1 Structura componentei Android

Proiectul este împărțit în directoare și fișiere după cum urmează:

- „manifests” conține informațiile esențiale de configurare a aplicației (numele, imaginea, activitățile, permisiunile necesare)
- folderul „java” conține pachetul “licenta.allbank” cu următoarea structură
- „activity” conține activitățile folosite în proiect
- „data” conține alte subdirectoare cu componente specifice interacțiunii și modelării datelor: „adapter”, „api”, „dao”, “database”, „model”, „repository”
- „service” conține serviciile ce se ocupă cu logica aplicației și interacțiunile acesteia cu API-urile folosite
- „ui” conține fragmentele, iar fiecăruia îi este asociat un layout (partea vizuală), respectiv un ViewModel care încapsulează logica fragmentului și accesează baza de date locală în vederea obținerii de date necesare
- „utils” conține clasele utilitare folosite în serviciile aplicației
- folderul „res” include componentele de stilizare ale view-urilor (animații, fonturi, imagini, layout-uri, tema aplicației) și fișiere de configurare precum certificatele de securitate

- fișierul „build.gradle” în care se regăsesc informații despre bibliotecile si dependențele folosite, versiunea de Java utilizată și de Android

Componenta serverului prezintă următoarea structură:

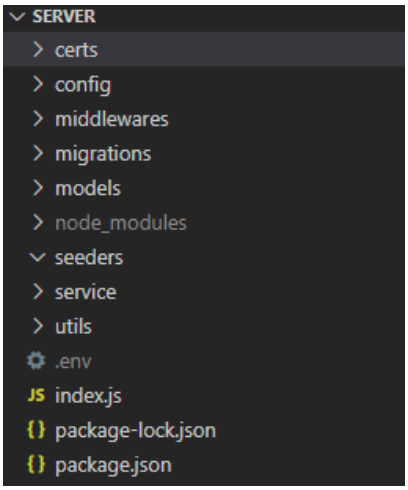

Figura 2.2.1.2 Structura componentei server

Este împărțită în foldere și fișiere după cum urmează:

- „certs” include certificatele de securitate folosite în utilizarea API-urilor
- „config” dispune datele de acces către baza de date a serverului
- „middlewares” conține middleware-urile folosite în cadrul requesturilor primite de la aplicația mobilă, cu rolul de a le verifica autenticitatea și integritatea
- „migrations” este folderul generat de Sequelize la crearea entităților ce substituie tabelele din baza de date și conțin informații despre acestea (nume, tip), legătura cu câmpurile din altă entitate, în funcție de versiunea bazei de date
- „models” este de asemenea generat de Sequelize la crearea entităților model. Ele conțin câmpurile claselor-tabel și asocierile dintre tabele (one-to-one, one-to-many etc.)
- „service” conține serviciile, grupate pe funcționalități, care se ocupă de logica backend-ului și interacționează cu baza de date

- „utils” are fișiere utilitare cu configurații și funcții folosite în cadrul serviciilor
- „index.js” este fișierul principal prin care se pornește serverul

2.2.2. Arhitectura sistemului

Arhitectura întregului sistem este una de tip client-server situată pe două nivele, bazată pe request-urile făcute de utilizatori prin aplicația de Android către componenta serverului, iar în cazul în care cererea făcută este una care necesită și implicarea serverelor bancare, atunci serverul aplicației se va comporta ca un intermediar, procesând cererea, pe care o va trimite în continuare către instituțiile financiare.

Această separare a nivelurilor a apărut în mod natural. Serverul aplicației are rolul de a memora resursele specifice aplicației (datele utilizatorilor) într-o bază de date și asigură administrarea și consistența ei, de aceea trebuie să se afle într-o entitate diferită de cea a clientului. Datorită faptului că resursele (datele bancare) se află distribuite și în afara sistemului, este nevoie de un interlocutor în acest proces, de aceea serverul răspunde cererilor de interogare venite din partea clientului, dar tot el trebuie să fie cel care interpretează și prelucrează request-urile făcute de aplicația de Android și le redirecționează în continuare către entitățile bancare pentru a putea asigura consistența sistemului, dar și partea de securitate (autenticitatea și confidențialitatea).

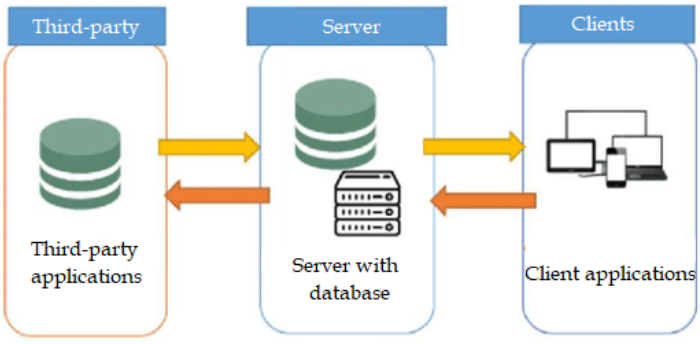

Figura 2.2.2.1 Diagrama arhitecturii aplicației

În figura 2.2.2.1, se observă schema de comunicare între componentele aplicației. Clientul (aplicația de Android) este cel care interoghează baza de date stocată pe server,

respectiv acesta comunică cu serviciile externe third-party (în cazul de față API-urile băncilor) pentru a putea asigura funcționalitățile necesare.

2.2.3. Arhitectura clientului Android

Arhitectura clientului de Android a fost dezvoltată folosind principiile pattern-ului Model-View-ViewModel [23] care urmărește separarea celor trei componente funcționale astfel încât să existe o relație de independență între ele:

- Model – înglobează tipurile și clasele care se ocupă de persistența și validarea datelor în cadrul aplicației. Ele nu trebuie să implementeze un comportament particular pentru contextul în care sunt folosite, ci reprezintă un tip de date abstract cu care are loc manipularea informației de-a lungul aplicației
- View – include elementele vizuale ale aplicației (de la animații și fonturi până la butoane și casete text cu care interacționează utilizatorul) prin intermediul cărora sunt declanșate acțiunile din ViewModel
- ViewModel – reprezintă o punte de comunicare între View și Model, prin care sunt accesate serviciile aplicației și notificate componentele grafice cu privire la orice schimbare prin folosirea pattern-ului Observer

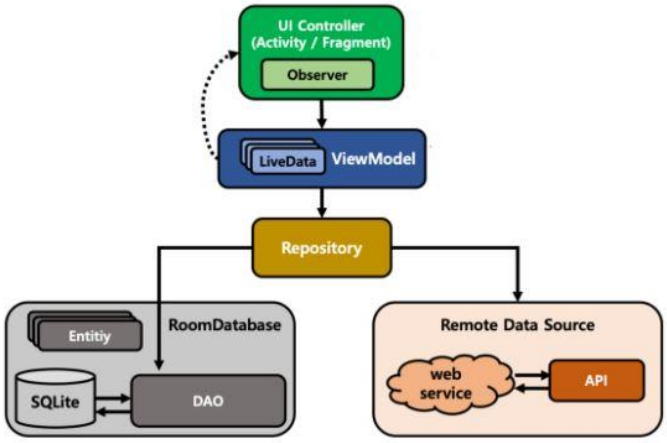

Figura 2.2.3.1 Arhitectura pattern-ului MVVM [23]

În figura 2.2.3.1 este ilustrată diagrama arhitecturii MVVM așa cum este recomandată de dezvoltatorii ecosistemului Android. În activitate sau fragment se declară o variabilă ViewModel care face legătura între interfața grafică și cea logică a aplicației. Pentru obținerea acestui comportament, se utilizează pattern-ul Observer [24] care este atașat unei variabile de tipul LiveData [25] din clasa ViewModel. Componenta Observer este cea care notifică sistemul că a apărut o modificare a variabilei LiveData și este declanșat un callback în care dezvoltatorul specifică noile instrucțiuni (actualizarea interfeței grafice, interogarea bazei de date etc.). Clasa ViewModel definită include un obiect Repository care face legătura cu baza de date locală sau preia datele printr-un serviciu web extern. După cum este menționat în capitolele anterioare, am folosit Room - un Object Relational Mapper (ORM) – menit să ușureze lucrul cu baza de date. De interogarea acesteia se ocupă interfața de tip DAO [26] prin care au loc comenzile de modificare, inserare sau ștergere.

Concret, voi ilustra printr-un exemplu cum a fost implementată arhitectura MVVM în partea practică a lucrării: pentru fragmentul în care sunt afișate conturile bancare ale unui utilizator, am definit o clasă ViewModel (AccountsViewModel) care are în componența sa un obiect repository „AccountRepository”, iar acesta are la rândul său un obiect de tip DAO prin intermediul căruia are loc interacțiunea cu baza de date.

În fragment este folosit un RecyclerView care afișează toate conturile bancare ale utilizatorului curent. De aceea, se folosește un obiect de tipul AccountsViewModel pentru a le a prelua din baza de date locală, iar cu ajutorul Observerului, se actualizează lista când are loc o modificare:

```
accountsViewModel.getBankingAccounts().observe(getViewLifecycleOwner
(), newBankingAccountsList ->
accountAdapter.setAccounts(newBankingAccountsList));
```

Clasa AccountsViewModel conține drept câmpuri o listă de obiecte Account de tip model și o instanță a clasei AccountRepository de tip repository:

```
public class AccountsViewModel extends AndroidViewModel {
    private final AccountRepository accountRepository;
    private LiveData<List<Account>> allAccounts;

    public AccountsViewModel(@NonNull Application application) {
        super(application);
```

```
        accountRepository = new AccountRepository(application);
        allAccounts = accountRepository.getAllAccounts();
    }

}
```

Clasa AccountsRepository include un obiect AccountDao de tip Dao care se ocupă de interogarea bazei de date:

```
public class AccountRepository {
    private AccountDao accountDao;
    private LiveData<List<Account>> accountList;

    public AccountRepository(Application application) {
        AccountsRoomDatabase accountsRoomDatabase =
AccountsRoomDatabase.getDatabase(application);
        accountDao = accountsRoomDatabase.accountDao();
        accountList = accountDao.getAllAccounts();
    }
    public LiveData<List<Account>> getAllAccounts() {
        return accountList;
    }
}
```

Interfața AccountDao este adnotată în acest sens și sunt declarate interogările către baza de date:

```
@Dao
public interface AccountDao {
/* Funcție care interoghează baza de date locală și returnează toate
conturile utilizatorului care au fost sincronizate cu aplicația */
@Transaction
@Query("SELECT * " +
        "FROM accounts_table")
LiveData<List<Account>> getAllAccounts();
}
```

În acest fel, am definit o arhitectură a sistemului scalabilă și flexibilă, care implică separarea componentelor astfel încât să fie independente, prin utilizarea pattern-ului Model-View-ViewModel așa cum este definit în documentația oficială de Android.

2.2.4. Baza de date

Pentru a putea gestiona informațiile despre utilizatorii aplicației, serverul se folosește de o bază de date relațională MySQL. Avantajele acesteia sunt faptul că asigură o securitate bună și este scalabilă pentru un număr mare de date (spre deosebire de SQLite unde nu prezintă funcții de gestionare a bazei de date și securitate la nivelul de acces, fiind recomandată pentru baze de date de dimensiuni reduse).

Serviciile de back-end din server interacționează cu baza de date a acestuia în diferite contexte (verificarea faptului că un utilizator are acces la o anumită resursă, accesarea tuturor conturilor bancare ale unei persoane, adăugarea unei sume maxime pe care să o cheltuiască etc). Pentru o mai bună gestionare a bazei de date, am folosit ORM-ul Sequelize.

Schema bazei de date de pe server este următoarea:

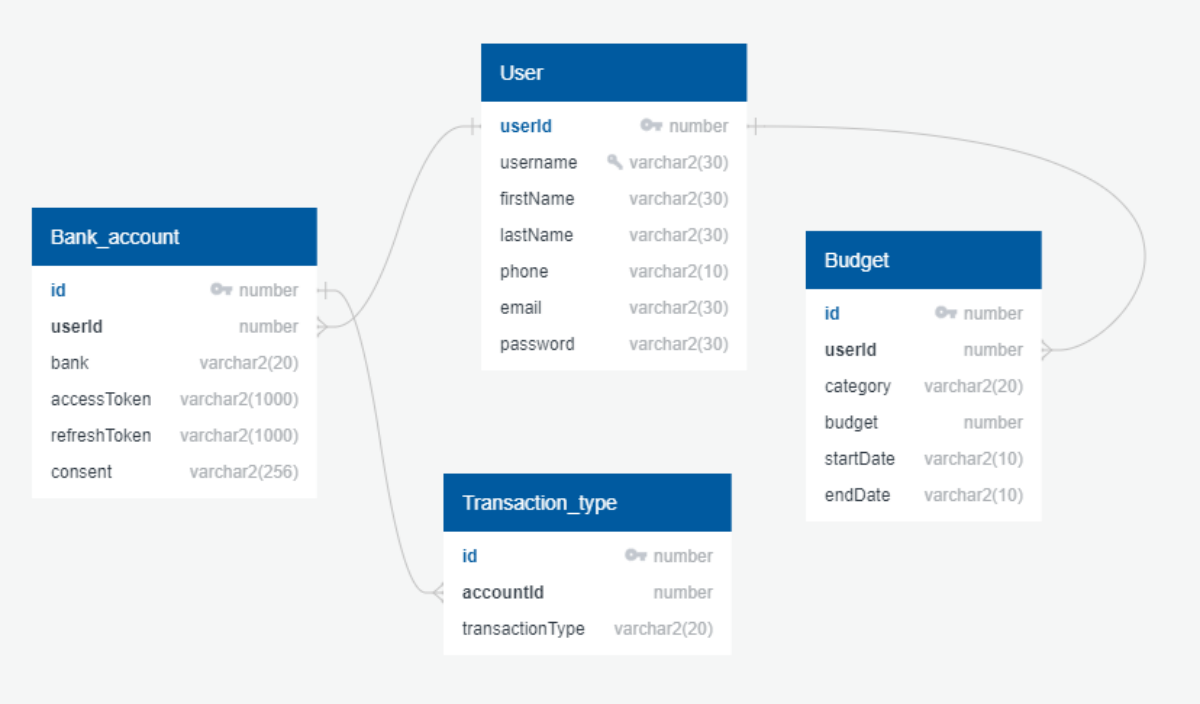

Figura 2.2.4.1 Diagrama bazei de date

În diagrama 2.2.4.1 este reprezentată schema bazei de date a serverului. Se identifică tabelele:

- User – memorează informații cu privire la utilizatorii aplicației odată ce se înregistrează în aplicație
- Bank_account – reprezintă conturile bancare ale utilizatorului și se află în relație many-to-one cu tabela User deoarece un utilizator poate avea conturi bancare la mai multe bănci (de ex. BCR și BT)
- Budget – reprezintă tabelul cu bugetele obiectiv pentru cheltuieli pe care și le pot seta utilizatorii. Se află în relație many-to-one cu tabela User
- Transaction_type – este tabela care memorează categoria setată de utilizator pentru fiecare tranzacție făcută

Pentru a asigura confidențialitatea utilizatorilor, aplicația reține doar datele necesare pentru a asigura funcționalitățile. De aceea, nu sunt memorate informații precum detaliile despre un anumit cont bancar (codul iban, monedă etc) sau despre tranzacțiile făcute de pe contul respectiv (data, suma, numele persoanei care a primit/trimis suma de bani). Pentru accesarea detaliilor despre un anumit cont bancar, este suficient să fie memorate token-ul de acces și refresh token-ul pe care le returnează modulul de autentificare al băncii, iar acestea sunt suficiente pentru a putea prelua datele utilizatorului pentru contul bancar respectiv.

În cadrul aplicației, după ce o persoană s-a autentificat, se execută automat un request către server pentru a prelua informațiile bancare ale utilizatorului. Odată primit răspunsul de la server, datele sunt memorate de aplicație într-o bază de date locală de tip SQLite, iar pentru a facilita accesul la ele, am utilizat ORM-ul Room care prezintă numeroase avantaje. De exemplu, datorită folosirii librăriei, se elimină din dezavantajele utilizării SQLiteDatabase precum convertirea manuală a datelor extrase din baza de date în obiecte și invers, problema mentenanței interogărilor odată cu modificarea schemei bazei de date, iar codul este mai ușor de integrat în arhitectura MVVM, devenind flexibil și ușor de întreținut. Pe de altă parte, această abordare aduce și dezavantaje precum scăderea vitezei de procesare datorită abstractizării nivelului de date.

Schema bazei de date de pe aplicația de Android unde sunt reținute doar detalii despre conturile și tranzacțiile utilizatorului:

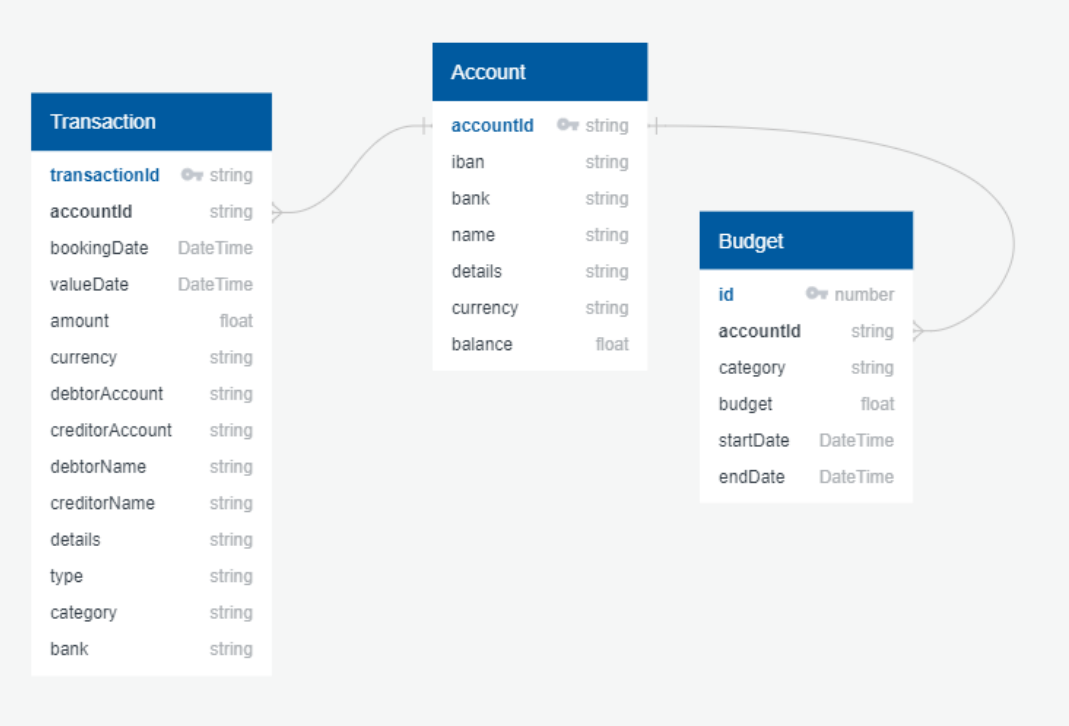

Figura 2.2.4.2 Diagrama bazei de date de pe clientul de Android

În diagrama 2.2.4.2 este reprezentată schema bazei de date a clientul de Android. Se identifică tabelele:

- Account – memorează toate conturile bancare regăsite în profilul unui utilizator de la o anumită bancă. De exemplu, un utilizator are un profil la BCR pentru care are deschise mai multe conturi bancare
- Budget – reprezintă tabelul cu bugetele obiectiv pentru cheltuielile făcute pe care le-a setat utilizatorul.
- Transaction– este tabela care memorează toate cheltuielile făcute de utilizator corespunzătoare tuturor conturilor pe care acesta le-a adăugat în aplicație

Tabela „Account” reține detalii despre conturile utilizatorului precum iban-ul, moneda în care a fost creat, numele contului, bilanțul, banca la care este deschis etc. Pentru fiecare cont, serverul returnează și lista tranzacțiilor corespunzătoare care sunt adăugate în tabela „Transaction”. Răspunsul de la server conține si lista cu bugetele pe care și le-a creat o persoană, iar acestea sunt adăugate în tabela „Budget”.

În sistemul propus, fiecărui tabel îi corespunde o clasă model generată automat de Sequelize sau definită manual pentru aplicația Android. Pentru generarea schemei bazei de date s-a ținut cont de principiile normalizării [27] astfel încât să nu existe câmpuri multiple sau redundanță între acestea, fiecare tabelă are o cheie primară etc.

2.2.5. Securitatea

Asigurarea unui nivel de securitate ridicat pentru protecția datelor este o problemă importantă în proiectarea unei aplicații web, în special în cazul produselor fintech. Pentru aceasta, toate informațiile care sunt stocate în baza de date nu trebuie menținute în clar, ci sunt prelucrate.

Parolelor li se aplică o funcție hash bcrypt [28] deoarece aceasta nu este o funcție inversabilă. Metoda introduce avantajul că nu oferă unui atacator accesul direct la parole, iar efectul hash-ului nu poate fi inversat. În plus, pentru evitarea unui atac de tip dicționar în care adversarul deține perechi precalculate pentru parole uzuale și hash-urile corespunzătoare, complementar se utilizează tehnica de salting ce presupune adăugarea unui text ales aleatoriu, iar asupra noului text se aplică metoda bcrypt.

Pentru restul câmpurilor, este obligatorie folosirea unei metode de criptare și nu a unei funcții hash deoarece, în cadrul aplicației, este nevoie de datele obținute în text clar. De aceea, am folosit metoda de criptare asimetrică RSA [29] care se bazează pe o pereche de chei publică și privată. Ideea metodei de criptare este că oricine poate cripta un text deoarece se folosește cheia publică, dar decriptarea se poate face doar de posesorul cheii private.

De asemenea, pentru a asigura securitatea transmiterii informației pe canalul de comunicare între client și server, ea se realizează prin protocolul HTTPS. Acesta este alcătuit din protocolul HTTP căruia i se adaugă protocolul criptografic TLS [30] care este direct responsabil de confidențialitatea datelor, prevenind un atac de tip man-in-the-middle sau previne un atacator pasiv să preia datele cu caracter sensibil (parole, numărul cardului de credit etc). Atât serverul, cât și aplicația client dețin câte o pereche de certificate digitale TLS care asigură identitatea entităților în momentul în care este inițiată conexiunea, urmată de schimbul de chei Diffie-Hellman [31] pentru a genera o cheie de sesiune secretă folosită la criptarea

informațiilor pe canalul de comunicație (URL-ul, anteturile, parametrii etc) prin tehnica de criptare simetrică.

Pentru a asigura integritatea datelor de-a lungul canalului de comunicare și a împiedica un adversar activ să altereze mesajele fără ca părțile să detecteze schimbarea, este folosit un mecanism de tip HMAC [32], mai precis HMAC-SHA256. Acesta se ocupă și de autenticitatea mesajelor prin folosirea unei chei secrete cunoscută doar de cele două părți, din care se generează 2 chei distincte. Ideea mecanismului constă în aplicarea succesivă a funcției hash SHA-256 peste mesaj și prima parte a cheii, respectiv generarea mesajului HMAC final prin aplicarea funcției peste mesajul obținut la prima rundă și a doua parte a cheii. Codul HMAC generat nu reprezintă o criptare a mesajului, ci acesta trebuie să însoțească corpul unui request, iar destinatarul verifică integritatea conținutului acestuia prin repetarea pașilor și compararea cu HMAC-ul primit. În cazul aplicației, a fost urmată schema de semnătură a request-urilor HTTP [33]. Fiecare cerere către server (fie că este vorba de serverele bancare sau serverul aplicației) este însoțită de anteturile *date, digest* și *signature*.

Pentru asigurarea accesului autorizat la resursele serverului, fiecare utilizator primește un token de acces după autentificarea în aplicație. În continuare, pentru fiecare cerere pe care utilizatorul o trimite la server, este adăugat în antetul de autorizare tokenul primit pentru ca serverul să verifice identitatea astfel încât numai persoanele autorizate să aibă acces. Pentru generarea acestuia, am folosit Json Web Token [21]. În momentul în care utilizatorul se autentifică cu succes, este generat un nou token pe care acesta să-l folosească în următoarele request-uri pentru a-și dovedi identitatea. Pentru a-i putea gestiona valabilitatea, în payload este adăugat câmpul expiresIn setat pe 600 secunde, iar pentru generarea lui, trebuie determinată semnătura, iar acest lucru se face prin semnarea conținutului cu algoritmul HMAC-SHA256. La următoarele cereri către server, se verifică token-ul atașat în header-ul de autorizare astfel încât acesta să fie corect și valabil.

Verificarea autorizării și integrității mesajelor se realizează prin intermediul unor middleware-uri, mai precis unul de verificare a integrității datelor, respectiv unul de autorizare. Fiecare request, odată ajuns la server, este filtrat prin intermediul acestor servicii. În cazul în care token-ul de acces al utilizatorului a expirat sau conținutul request-ului a fost alterat pe canalul de comunicare, serviciile vor semnala o eroare, iar utilizatorul primește un mesaj corespunzător.

2.2.6. Integrarea API-urilor

Utilizarea API-urilor în cadrul sistemului presupune adăugarea unor anteturi de securitate pentru a satisface cerințele din documentațiile lor. Trimiterea de cereri către serverele bancare presupune adăugarea în antet a header-ului *web-api-key* ce reprezintă un cod unic de identificare pentru fiecare aplicație înregistrată în portalul de dezvoltatori al sistemelor bancare. Prin folosirea acestui header, serverele instituțiilor financiare verifică sursa fiecărui request și se asigură că este una autorizată să trimită cereri.

După cum este menționat și în subcapitolul anterior, fiecare request este semnat digital, prin generarea și adăugarea headerelor *digest, date, signature* folosite la verificarea integrității conținutului. În acest fel, este prevenit scenariul în care un atacator activ modifică pe canalul de comunicație datele trimise în request (de exemplu, pentru crearea unei noi tranzacții, atacatorul poate modifica codul iban către care este trimisă suma în propriul cod iban și nu s-ar semnala nicio eroare).

3. Funcționalitățile aplicației

Aplicația este concepută astfel încât să răspundă nevoilor utilizatorilor de a-și gestiona propriile finanțe. De aceea, principalele funcționalități ale aplicației se încadrează în ușurința administrării conturilor bancare din cadrul mai multor bănci, respectiv a tranzacțiilor corespunzătoare și obținerea detaliată de statistici bazate pe tranzacțiile efectuate și a cheltuielilor într-o anumită perioada de timp.

Fără a restrânge din generalitate, lucrarea utilizează API-urile puse la dispoziție de BCR [34] și Banca Transilvania [35] prin care pot fi accesate datele bancare ale utilizatorilor lor, însă funcționalitățile aplicației pot fi extinse cu ușurință prin integrarea serviciilor externe oferite și de alte instituții financiare (ING, BRD, Raiffeisen etc). În acest fel, aplicația ar acoperi o gamă mai largă de produse bancare și ar îmbunătăți experiența utilizatorului.

Odată ce aplicația a fost lansată, se deschide prima pagină (figura 3.1) unde persoana are două opțiuni: înregistrarea unui nou cont de utilizator sau autentificarea.

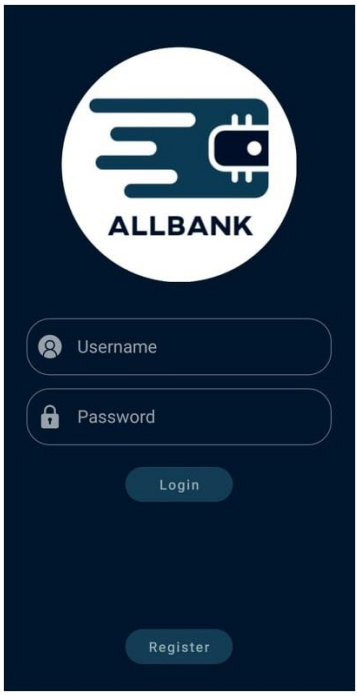

Figura 3.1 Ecranul principal al aplicației

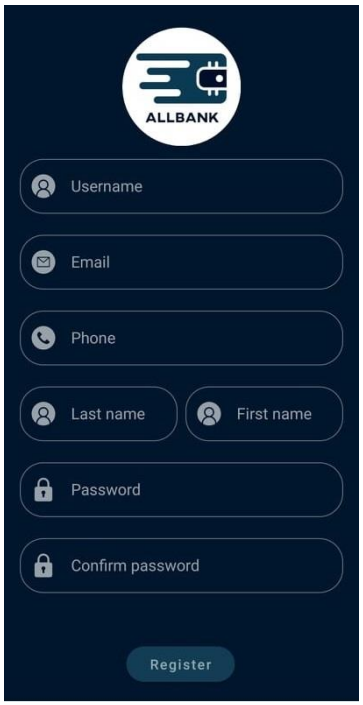

Figura 3.2 Ecranul de înregistrare al unui utilizator

Pentru a putea utiliza aplicația, o persoană este obligată să se înregistreze, introducând date precum numele de utilizator, nume, prenume, adresă de e-mail, telefon și parolă. În continuare, noul utilizator se poate autentifica introducând o parte din datele de înregistrare (numele de utilizator și parola), iar în caz de succes, va fi întâmpinat de un ecran de încărcare cât timp serverul se ocupă de preluarea informațiilor bancare de pe serverele externe și afișarea lor în cadrul aplicației.

Primul ecran este cel principal (figura 3.3), în care sunt prezentate toate tranzacțiile efectuate de utilizator din conturile bancare sincronizate cu aplicația. Utilizatorul are posibilitatea de a le filtra, in funcție de tipul lor (încasări, cheltuieli sau ambele). În plus, este prezentată balanța tuturor conturilor, alături de totalul cheltuielilor și veniturilor din ultima lună.

Figura 3.3 Ecranul principal al aplicației

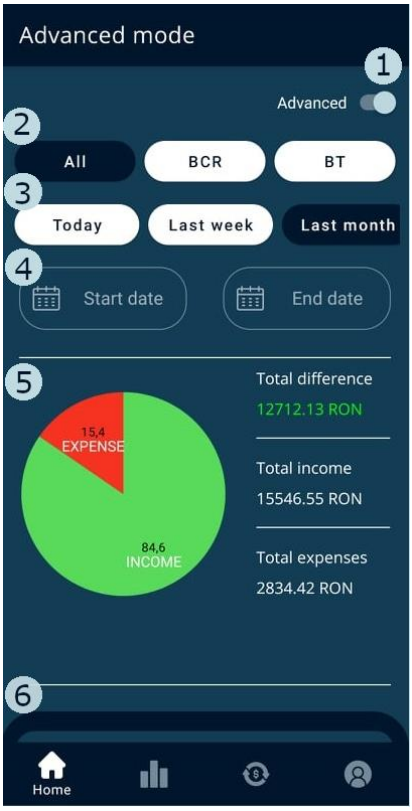

Figura 3.4 Ecranul modului avansat al cheltuielilor

Pentru a obține informații detaliate despre cheltuielile făcute, este definit un mod avansat (figura 3.4) care poate fi accesat prin glisarea butonului din partea de sus. Odată apăsat, se afișează pe ecran mai multe elemente și opțiuni folosite la filtrarea datelor:

1. Butonul de comutare între cele două ecrane: cel principal și al modului avansat
2. Butoane folosite la filtrarea cheltuielilor în funcție de banca aleasă (implicit sunt selectate toate)
3. Butoane folosite la filtrarea cheltuielilor în funcție de perioadă. Reprezintă o scurtătură pentru intervale de timp generice (o zi, ultima săptămână, ultima lună, ultimele trei luni, ultimul an). Implicit este selectată opțiunea de ultima lună
4. Casete text în care se introduc două date calendaristice pentru filtrarea cheltuielilor. Când au fost completate amândouă câmpurile, se dezactivează filtrul selectat cu intervalele generice, iar când se alege din nou opțiunea de interval generic, se dezactivează casetele acestea
5. Diagramă radială ce prezintă procentual cheltuielile și veniturile în funcție de filtrările alese (perioada de timp și numele băncii) . În partea dreaptă sunt afișate veniturile, cheltuielile și diferența dintre cele două în intervalul de timp ales. Pentru mai multe detalii, se poate da click pe tipul ales (cheltuieli sau venituri), iar în partea dreaptă se vor actualiza datele: este afișat totalul cheltuielilor sau veniturilor, respectiv media lor zilnică
6. De fiecare dată când se alege o nouă variantă de filtrare sau când se selectează o categorie din diagramă, în partea de jos a ecranului este afișată o listă cu tranzacțiile efectuate ce se actualizează constant

În partea de jos a ecranului, este afișată o bară de navigare cu mai multe pictograme, fiecare având diferite destinații. Cea de-a doua opțiune este corespunzătoare funcționalității de statistică pe baza categoriilor de cheltuială. Aceasta își propune obținerea unui mai bun control al plăților efectuate, grupate în funcție de categorie, și presupune două componente: primirea de statistici detaliate a cheltuielilor, respectiv gestionarea bugetului acestora într-o anumită perioadă de timp.

Figura 3.5 Ecranul statisticilor cheltuielilor grupate pe categorii

Figura 3.6 Continuarea ecranului statisticilor cheltuielilor grupate pe categorii

Primul ecran, prezentat în figurile 3.5 și 3.6, conferă utilizatorului posibilitatea de a obține detalii exclusiv asupra cheltuielilor, filtrate pe categorii. Implicit, se deschide primul fragment, cel de statistici (reprezentat prin elementul 1), respectiv fragmentul în care se pot vizualiza bugetele setate de utilizator (elementul 2).

Acesta oferă mai multe opțiuni de personalizare a filtrării cheltuielilor:

3. Butoane folosite la filtrarea plăților în funcție de tipul categoriei (mâncare, casnice, distracție etc). În mod implicit sunt selectate toate categoriile
4. Butoane folosite la filtrarea cheltuielilor în funcție de perioadă. Reprezintă o scurtătură pentru intervale de timp generice (o zi, ultima săptămână, ultima lună, ultimele trei luni, ultimul an). Implicit este selectată opțiunea de ultima lună
5. Casete text în care se introduc două date calendaristice pentru filtrarea cheltuielilor. Când au fost completate amândouă câmpurile, se dezactivează filtrul selectat cu intervalele generice, iar când se alege din nou opțiunea de interval generic, se dezactivează casetele

6. Diagramă unde sunt prezentate procentual cheltuielile în funcție de filtrările alese (perioada de timp și categorie). În partea dreaptă este afișată legenda cu numele categoriei și culoarea sa în grafic
7. Sunt prezentate statistici pentru cheltuieli în funcție de filtrările alese (perioada de timp și categorie). Acestea includ media zilnică, săptămânală și lunară a plăților
8. Este afișat un graf cu bare, împărțit în intervale egale (calculate automat în funcție de mărimea perioadei de timp aleasă) în care sunt afișate cheltuielile grupate pe intervale de timp și pe categorie.
9. De fiecare dată când se alege o nouă variantă de filtrare sau când se selectează o categorie din diagramă, în partea de jos a ecranului este afișată o listă cu tranzacțiile efectuate ce se actualizează constant

Cel de-al doilea ecran al funcționalității este destinat gestionării bugetelor cheltuielilor. Pentru o bună gestionare a acestora, un utilizator are posibilitatea să seteze un buget maxim pentru o categorie de plăți pe o perioadă de timp.

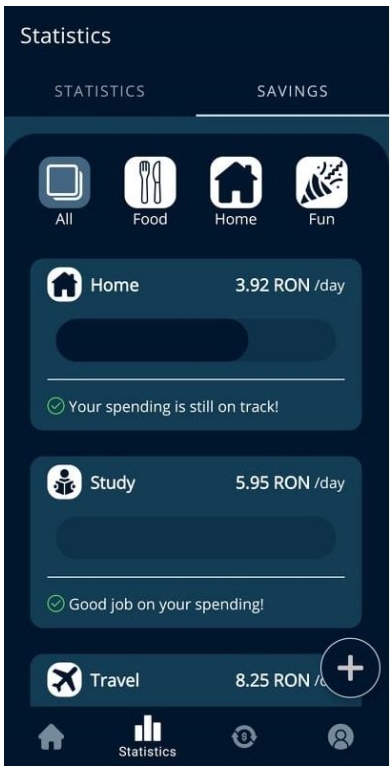

Figura 3.7 Ecranul cu bugetele setate de utilizator, grupate pe categorii

Figura 3.8 Ecranul de adăugare a unui nou buget

Figura 3.9 Ecranul de confirmare al budgetului adăugat

Ecranul prezentat în figura 3.7 prezintă o listă de componente vizuale în care sunt afișate toate bugetele țintă setate de utilizator. În cadrul unui astfel de element, sunt prezentate tipul categoriei, suma medie zilnică ce poate fi cheltuită, este afișată o bară de progres și un mesaj corespunzător, în funcție de suma plăților făcute.

În partea de sus este afișată o listă de filtrare a bugetelor setate, în care utilizatorul poate să personalizeze căutările în lista menționată, în funcție de categoria aleasă. Implicit sunt afișate toate categoriile, dar este permisă selectarea mai multe categorii în același timp. Pentru a anula acest efect, este necesară alegerea opțiunii corespunzătoare pentru toate categoriile.

Utilizatorul poate să adauge ușor un nou buget țintă, prin intermediul butonului din dreapta jos, urmând a se deschide un nou fragment, prezentat în figura 3.8. Trebuie alese:

1. Categoria (implicit e selectată opțiunea „Others” – „Altele”)
2. Datele de start și final ale perioadei. Data minimă aleasă este ziua curentă

3. Suma maximă pe care utilizatorul dorește să o cheltuiască în perioada aleasă pentru categoria respectivă

Odată apăsat butonul, este afișat ecranul de confirmare (Figura 3.9) cu următoarele detalii:

1. Categoria aleasă, alături de o bară progres a sumei cheltuite raportată la suma maximă setată de utilizator
2. Suma totală, suma rămasă și cât a fost cheltuit în total pentru categoria respectivă
3. Intervalul de timp pentru care a fost setat bugetul
4. Statistici pentru cheltuiala medie zilnică, săptămânală și lunară

Utilizatorul confirmă noul buget creat prin apăsarea butonului, iar în caz de succes, este redirecționat către meniul statisticilor.

De asemenea, acesta are posibilitatea să vizualizeze informații despre un anumit buget apăsând pe el, deschizându-se un ecran asemănător cu cel de confirmare a bugetului din figura 3.9 sau să editeze un buget, ținând lung apăsat și apărând un ecran ca în figura 3.8. În cazul ultimei variante, procesul de editare este asemănător cu cel al adăugării unui nou buget.

A treia destinație din meniul barei de navigare este adresată funcționalității creării de noi tranzacții. Sunt posibile trei opțiuni:

1. Crearea unei noi plăți introducând manual codul iban, numele destinatarului, suma și detaliile acesteia
2. Crearea unei noi tranzacții prin scanarea unui cod QR
3. Crearea de cod QR ce reprezintă suma pe care să o primească utilizatorul într-un cont de-al său, care poate fi salvat în telefon sau distribuit prin alte aplicații. Pentru generarea lui, sunt introduse codul iban al contului, suma și detaliile de plată

Prima variantă se deschide implicit în momentul navigării către cea de-a treia destinație.

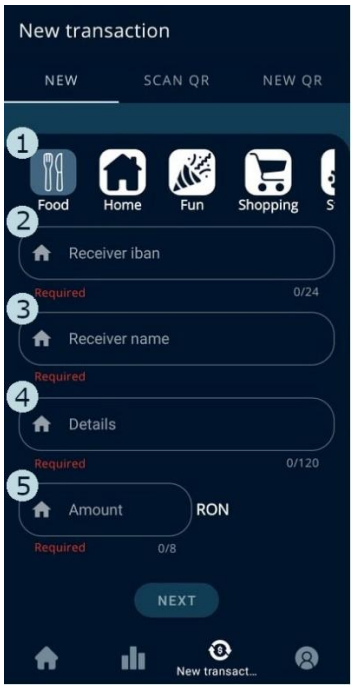

Figura 3.10 Ecranul de adăugare unei noi plăți folosind prima metoda

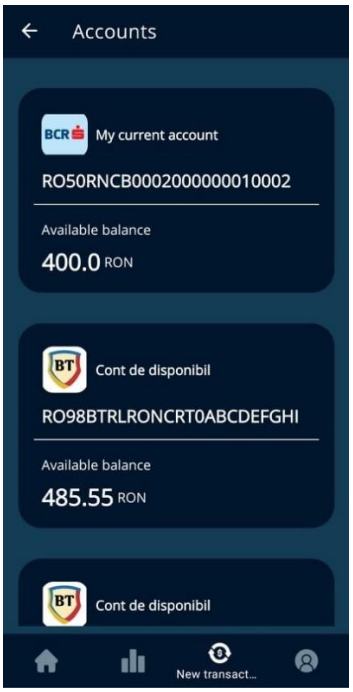

Figura 3.11 Meniul ce afișează conturile utilizatorului

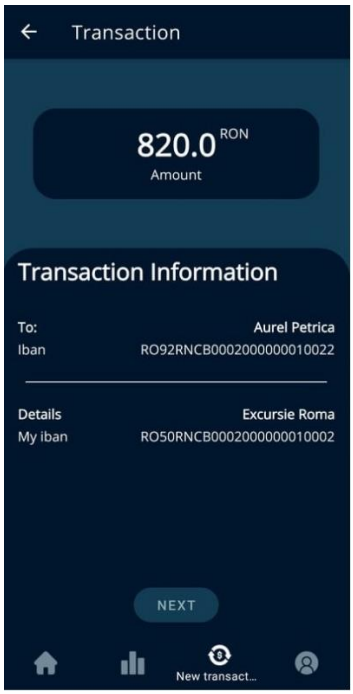

Figura 3.12 Ecranul cu detalii despre noua tranacție

Utilizatorul are de completat mai multe casete text:

1. Tipul categoriei tranzacției, ales dintr-o listă predefinită
2. Codul iban al destinatarului. Pe măsură ce utilizatorul completează în căsuța text codul iban, este afișată o listă de sugestii cu numele și iban-ul destinatarilor pentru care a fost adăugată o plată anterior. Odată ce utilizatorul apasă pe o astfel de sugestie, se completează automat și căsuța text cu numele destinatarului
3. Numele destinatarului. Asemănător cu caseta text descrisă anterior, pe măsură ce se completează numele destinatarului, este afișată o listă de sugestii cu numele și iban-ul destinatarilor pentru care a fost creată o plată anterior. Odată

ce utilizatorul apasă pe o astfel de sugestie, se completează automat și căsuța text cu contul iban

4. Detaliile tranzacției
5. Suma pe care utilizatorul vrea să o transfere

După apăsarea butonului de confirmare, utilizatorul este redirecționat către un nou ecran (figura 3.11) în care sunt afișate toate conturile bancare ale acestuia. El are de ales contul din care se va face plata, apăsând pe acesta, se încarcă noul ecran (figura 3.12) în care vor fi afișate toate detaliile tranzacției. După ce verifică din nou că a introdus toate datele corect, utilizatorul este nevoit să confirme autorizarea tranzacției la nivelul aplicației (figura 3.13).

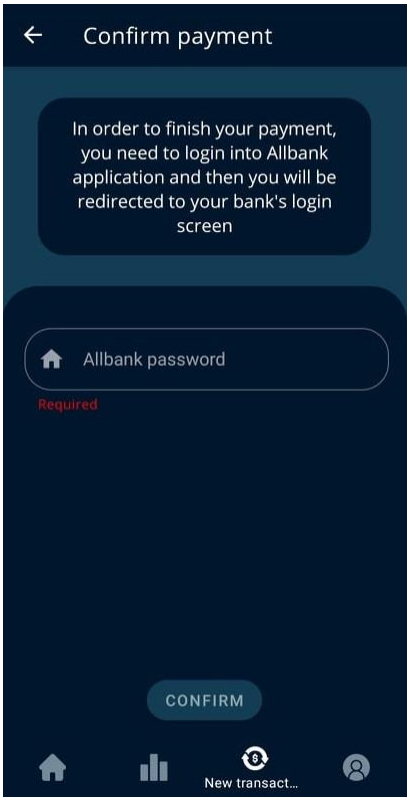

Figura 3.13 Confirmarea tranzacției

Pentru a face acest lucru, trebuie să dovedească identitatea sa prin reintroducerea parolei, urmată de autentificarea în cadrul băncii corespunzătoare contului din care are loc plata. În final, după ce utilizatorul a urmat cu succes acești pași, este autorizată tranzacția din meniul instituției bancare, după care este redirecționat înapoi în aplicație și primește un mesaj corespunzător.

A doua metoda este destinată creării unei noi plăți în baza scanării unui cod QR. Utilizatorul alege opțiunea „Scan QR” din meniul de deasupra, moment în care se activează camera foto a telefonului (figura 3.14).

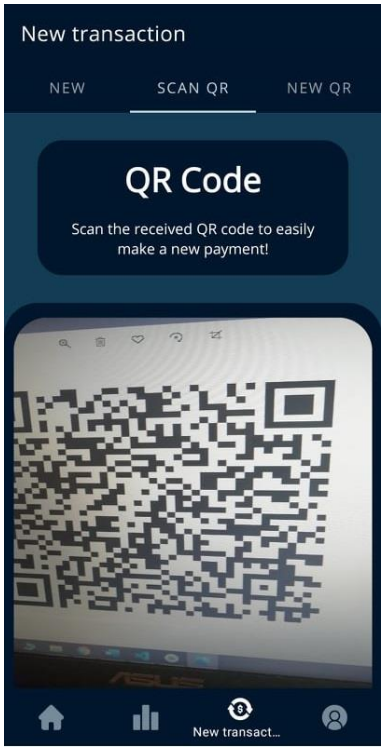

Figura 3.14 Ecranul de scanare a unui cod QR

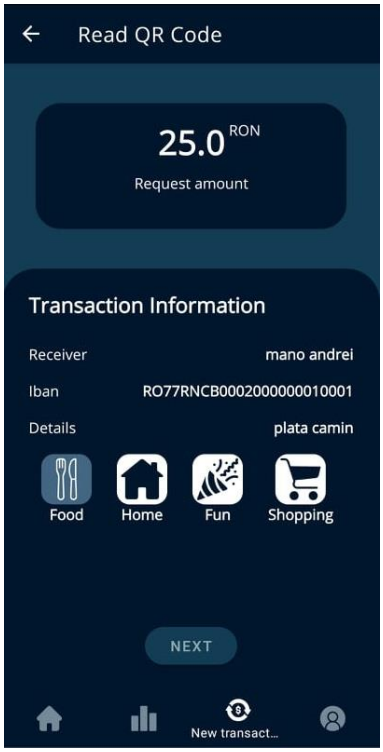

Figura 3.15 Ecranul cu detaliile despre o plată după scanarea codului QR

După ce este scanat codul QR, sunt afișate detaliile tranzacției (nume destinatar, codul iban al contului, detaliile tranzacției) într-un nou ecran (figura 3.15). Utilizatorul confirmă datele și este redirecționat către mecanismul de autorizare a plății în cadrul aplicației, similar cu procesul descris anterior.

A treia metodă o reprezintă generarea de cod QR corespunzător unei plăți către utilizator. În acest fel, o persoană poate genera ușor un șablon al unei tranzacții care poate fi utilizată nelimitat.

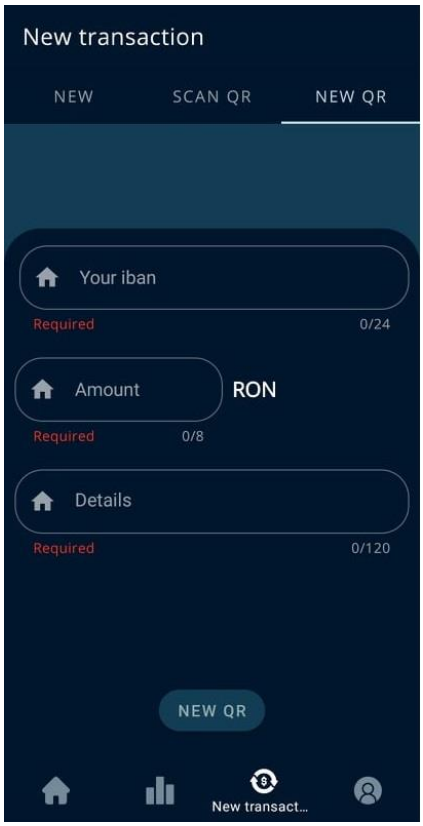

Figura 3.16 Ecranul de generare a unui nou cod QR

Figura 3.17 Ecranul cu noul cod QR generat

Utilizatorul introduce detalii precum suma, codul iban al contului în care să fie încasată și detaliile plății (figura 3.16). În continuare, el confirmă datele introduse și este redirecționat către noul ecran (figura 3.17) unde este afișată o imagine cu noul cod QR generat, alături de două opțiuni:

- Salvarea fotografiei în memoria internă a telefonului
- Distribuirea pozei prin intermediul altor aplicații (Facebook, Whatsapp, Messenger, Gmail etc)

A patra destinație din bara de navigare duce în meniul profilului utilizatorului (figura 3.18). De aici, el are posibilitatea de a-și edita datele personale, de a vedea detalii și accesa termenii și condițiile aplicației, de a se deconecta sau să gestioneze conturile bancare ale aplicației, prin apăsarea butonului corespunzător.

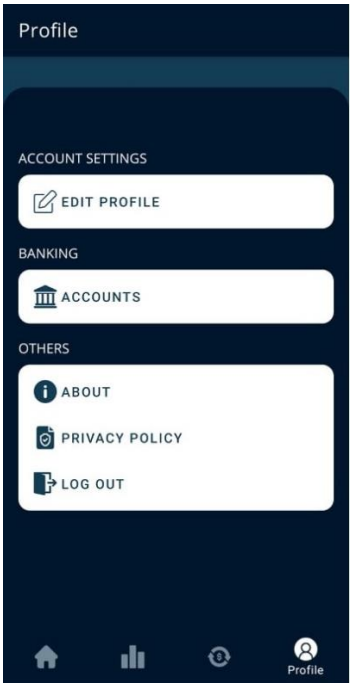

Figura 3.18 Ecranul cu meniul profilului

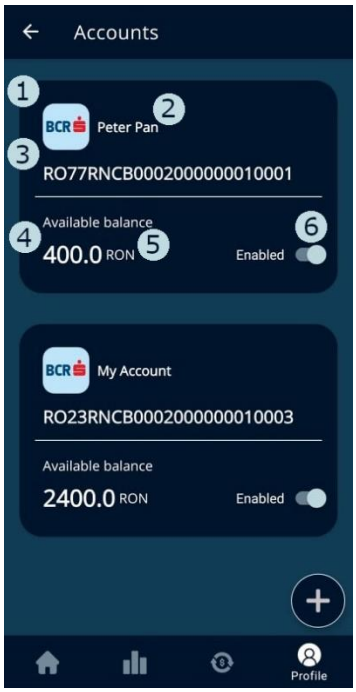

Figura 3.19 Ecranul cu conturile utilizatorului

Figura 3.20 Ecranul de adăugare a unui nou cont bancar

Ultima opțiune asigură administrarea conturilor din cadrul tuturor băncilor pe care utilizatorul le-a sincronizat cu aplicația. Este afișată lista cu toate conturile bancare (figura 3.19), iar pentru fiecare componentă, sunt prezentate:

1. Banca unde este deschis contul bancar, reprezentată printr-o imagine
2. Numele contului bancar
3. Codul iban al contului
4. Suma rămasă disponibilă
5. Moneda în care a fost deschis contul
6. Buton ce activează sau dezactivează contul bancar. Implicit, toate conturile sunt active, iar în cazul dezactivării unuia dintre ele, acesta și tranzacțiile sale nu vor figura în cadrul statisticilor generate de aplicație

Utilizatorul poate adăuga noi conturi bancare în aplicație prin apăsarea butonului destinat din partea de jos a ecranului. Este redirecționat către un nou ecran (figura 3.19) unde are de ales entitatea bancară emitentă a contului, apoi se autentifică cu datele de acces primite de la bancă. În caz de succes, utilizatorul primește un mesaj în acest sens și este redirecționat înapoi în aplicație.

Concluzii

Allbank reprezintă o aplicație fintech ce își propune să rezolve problema administrării greoaie a conturilor bancare prin reunirea acestora într-o singură aplicație. Prin funcționalitățile implementate, utilizatorul poate să vadă ușor detalii și statistici despre toate tranzacțiile efectuate. În plus, prezintă și o componentă destinată economisirii prin posibilitatea setării unei sume maxime pe care utilizatorul să o cheltuiască într-o perioadă de timp.

Aplicația a fost proiectată respectând standardele actuale. Sistemul este unul de tip client-server, serviciile web fiind implementate folosind principiile REST bazate pe protocolul HTTPS. Serverul se ocupă cu găzduirea și interogarea bazei de date, precum și interacțiunea cu API-urile oferite de bănci. Aplicația client este dezvoltată pe platforma Android și folosește arhitectura MVVM ce se ocupă de separarea componentelor astfel încât să fie independente.

Securitatea aplicației este obținută prin folosire protocolului de comunicare HTTPS care asigură confidențialitatea datelor de-a lungul canalului de comunicare, iar integritatea informației s-a realizat prin mecanisme de tipul MAC. Pentru accesarea în siguranță a API-urilor și comunicației client-server, s-au folosit certificatele digitale pentru asigurarea identității. Informațiile din baza de date sunt memorate în formă criptată, cu excepția parolelor cărora le-au fost aplicate funcții hash.

Fiind o aplicație care rezolvă o problemă de natură financiară, constant vor fi nevoite actualizări și îmbunătățiri. În urma a avansului tehnologiei și al atacurilor cibernetice, este necesară conformarea sistemelor de securitate cu ultimele standarde în vederea protejării datelor utilizatorilor. În plus, ca urmare a integrării unor API-uri, acestea pot suferi modificări care pot altera mecanismele utilizate în cadrul aplicației, deci trebuie urmărite ultimele schimbări sau comunicate ale instituțiilor financiare cu privire la actualizări. De asemenea, cum acest domeniu este puternic reglementat de instituțiile europene, trebuie respectate normele legislative aflate în vigoare. Nu în ultimul rând, aplicația poate fi îmbunătățită prin adăugarea de funcționalități complementare. De exemplu, este vorba de o componentă bazată pe inteligența artificială care să modeleze comportamentul utilizatorului și să genereze predicții cu privire la viitoarele tranzacții ale acestuia. Alte funcționalități care ar îmbunătăți aplicația

sunt: posibilitatea de a împărți o notă de plată între mai mulți persoane, modificarea interfeței astfel încât să devină mai sugestivă în interacțiunea cu utilizatorul, posibilitatea de generare a unui fișier PDF cu statisticile obținute în aplicație sau crearea unui extras de cont care să cuprindă toate tranzacțiile utilizatorului.

Pentru asigurarea funcționalităților aplicației, cea mai importantă și complexă componentă a reprezentat-o integrarea API-urilor bancare. Aceste servicii web externe oferă posibilitatea accesării în siguranță a datelor bancare ale unui utilizator astfel încât acestea să nu fie supuse niciunui risc, precum și prelucrarea lor după nevoi. În plus, pentru mai buna înțelegere a acestor informații, interfața prezintă grafice și statistici detaliate ce pot fi ușor personalizate în funcție de necesități. Simplificarea unor mecanisme financiare face parte din subiectul lucrării, astfel că se oferă posibilitatea utilizatorului să reunească conturile bancare din cadrul mai multe instituții financiare într-o singură aplicație. În plus, pentru ușurarea realizării de tranzacții, se oferă posibilitatea de generare a unui cod QR, drept plată șablon, care poate fi salvat și distribuit. Cum aplicația se încadrează în categoria fintech-urilor, aceasta permite utilizatorului să controleze mai ușor plățile făcute și să-și seteze sumele maxime pe care poate să le cheltuiască, în funcție de categorie, prin crearea unor bugete.

Astfel, următoarele obiective, formulate inițial, s-au transpus în rezultatul final al acestei lucrări:

- Posibilitatea sincronizării mai multor conturi bancare ale unui utilizator în cadrul unei singure aplicații și gestionarea facilă a acestora
- Oferirea de analize detaliate despre cheltuielile și veniturile utilizatorului, statistici care pot fi ușor configurate în funcție de preferințe
- Implementarea unor funcționalități destinate economisirii și controlului cheltuielilor

Bibliografie

[1] EY, „EY global fintech adoptation index," 2019. [Interactiv]. Available: https://assets.ey.com/content/dam/ey-sites/ey-com/en_gl/topics/banking-and-capital-markets/ey-global-fintech-adoption-index.pdf. [Accesat 24 Mai 2021].

[2] „PSD2 directive on payment services," [Interactiv]. Available: https://data.consilium.europa.eu/doc/document/PE-35-2015-INIT/en/pdf. [Accesat 4 Iunie 2021].

[3] International Data Corporation (IDC), „Smartphone Market Share," [Interactiv]. Available: https://www.idc.com/promo/smartphone-market-share. [Accesat 9 Iunie 2021].

[4] „Open Handset Alliance," [Interactiv]. Available: http://www.openhandsetalliance.com/oha_faq.html. [Accesat 8 Iunie 2021].

[5] V. Bill, Inside the Java Virtual Machine, McGraw-Hill, Inc., 1996.

[6] A. Frumușanu, „A Closer Look at Android RunTime (ART) in Android L," AnandTech, 2014.

[7] Oracle, „Java Documentation," [Interactiv]. Available: https://docs.oracle.com/en/java/. [Accesat 8 Iunie 2021].

[8] „Node.js," [Interactiv]. Available: https://nodejs.org/en/about/. [Accesat 8 Iunie 2021].

[9] „V8 Engine," [Interactiv]. Available: https://v8.dev/. [Accesat 8 Iunie 2021].

[10] „Express/Node introduction," [Interactiv]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction. [Accesat 9 Iunie 2021].

[11] E. Mahn, Express in Action, 2016.

[12] „Android Activity Lifecycle," [Interactiv]. Available: https://justforchangesake.wordpress.com/tag/activity-life-cycle/. [Accesat 12 Iunie 2021].

[13] „Room," [Interactiv]. Available: https://developer.android.com/training/data-storage/room. [Accesat 9 Iunie 2021].

[14] Google, „Material Design," [Interactiv]. Available: https://material.io/. [Accesat 1 Iunie 2021].

[15] „Sequelize," [Interactiv]. Available: https://sequelize.org/master/. [Accesat 4 Iunie 2021].

[16] „Retrofit," [Interactiv]. Available: https://square.github.io/retrofit/. [Accesat 4 Iunie 2021].

[17] R. T. Fielding, „Chapter 5: Representational State Transfer (REST)," în *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, 2000.

[18] „The OAuth 2.0 Authorization Framework," [Interactiv]. Available: https://datatracker.ietf.org/doc/html/rfc6749.

[19] DigitalOcean, „An Introduction to OAuth2," [Interactiv]. Available: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2. [Accesat 12 Iunie 2021].

[20] „QRGenerator," [Interactiv]. Available: https://github.com/androidmads/QRGenerator. [Accesat 9 Iunie 2021].

[21] „JSON Web Token," [Interactiv]. Available: https://jwt.io/. [Accesat 3 Iunie 2021].

[22] „JWT (JSON Web Token) (in)security," [Interactiv]. Available: https://research.securitum.com/jwt-json-web-token-security/. [Accesat 12 Iunie 2021].

[23] Google, „Guide to app architecture," [Interactiv]. Available: https://developer.android.com/jetpack/guide. [Accesat 1 Iunie 2021].

[24] Google, „Observer pattern," [Interactiv]. Available: https://developer.android.com/reference/java/util/Observer. [Accesat 1 Iunie 2021].

[25] Google, „LiveData," [Interactiv]. Available: https://developer.android.com/topic/libraries/architecture/livedata. [Accesat 1 Iunie 2021].

[26] Google, „DAO interface," [Interactiv]. Available: https://developer.android.com/training/data-storage/room/accessing-data. [Accesat 1 Iunie 2021].

[27] E. Codd, „A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM,* vol. 6, nr. 13, pp. 377-387, 1970.

[28] N. Provos și D. Mazières, „A Future-Adaptable Password Scheme," în *Proceedings of 1999 USENIX Annual Technical Conference*, Monterey, 1999.

[29] R. Rivest, A. Shamir și L. Adleman, „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM,* vol. 21, nr. 2, pp. 120-126, 1978.

[30] R. Khare și S. Lawrence, „Upgrading to TLS Withing HTTP/1.1," [Interactiv]. Available: https://datatracker.ietf.org/doc/html/rfc2817. [Accesat 6 Iunie 2021].

[31] W. Diffie și M. Hellman, „New Directions in Cryptography," *IEEE Transactions on Information Theory,* vol. 22, nr. 6, pp. 644-654, 1976.

[32] H. Krawczyk, M. Bellare și R. Canetti, „HMAC: Keyed-Hashing for Message Authentication," 1997. [Interactiv]. Available: https://datatracker.ietf.org/doc/html/rfc2104. [Accesat 5 Iunie 2021].

[33] A. Backman, J. Richer și M. Sporny, „Signing HTTP Messages,” [Interactiv]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-message-signatures. [Accesat 3 Iunie 2021].

[34] Erste Group, „Our APIs,” [Interactiv]. Available: https://developers.erstegroup.com/docs/apis/bank.bcr. [Accesat 12 Iunie 2021].

[35] Banca Transilvania, „API products,” [Interactiv]. Available: https://apistorebt.ro/bt/sb/api-products. [Accesat 12 Iunie 2021].