

LUCRARE DE DISERTAȚIE

**Conducător științific:**

**Profesor dr. Bildea Ștefan Teodor**

**Absolvent:**

**Botez Bogdan**

**BRAȘOV, 2022**

*Universitatea Transilvania din Brașov*

*Facultatea de Științe Economice și Administrarea Afacerilor*

*Masterat: Sisteme informatice pentru afaceri*

Aplicație pentru turism comunitar

**Conducător științific:**

**Profesor dr. Bildea Ștefan Teodor**

**Absolvent:**

**Botez Bogdan**

**BRAȘOV, 2022**

CUPRINS

[1. INTRODUCERE 3](#_Toc105938513)

[2. TEHNOLOGII UTILIZATE 5](#_Toc105938514)

[2.1. REACT NATIVE 5](#_Toc105938515)

[2.2. JAVASCRIPT 6](#_Toc105938516)

[2.3. EXPO 6](#_Toc105938517)

[2.4. API 7](#_Toc105938518)

[2.5. LIBRĂRII 8](#_Toc105938519)

[2.5.1 Expo Google App Auth 9](#_Toc105938520)

[2.5.2 Expo Facebook 9](#_Toc105938521)

[2.5.3. Axios 9](#_Toc105938522)

[2.5.4. Expo Barcode Scanner 10](#_Toc105938523)

[2.5.5. React Native Ratings 10](#_Toc105938524)

[3. PREZENTAREA SISTEMULUI 11](#_Toc105938525)

[3.1. CERINȚE FUNCȚIONALE 11](#_Toc105938526)

[3.2. COMPONENTE 11](#_Toc105938527)

[3.2.1. Pagina Login 12](#_Toc105938528)

[3.2.2. Pagina Questionnaire 13](#_Toc105938529)

[3.2.3. Pagina Welcome 15](#_Toc105938530)

[3.2.4. Pagina Scanner 16](#_Toc105938531)

[3.2.5 Pagina Discount 18](#_Toc105938532)

[3.2.6 Pagina Review 18](#_Toc105938533)

[3.2.7. Meniul aplicației 20](#_Toc105938534)

[3.2.8. Pagina Entities 21](#_Toc105938535)

[3.2.9. Pagina EntityInformation 22](#_Toc105938536)

[3.2.10. Pagina Transactions 23](#_Toc105938537)

[3.2.11. Participanți 25](#_Toc105938538)

[3.2.12. Măsurători 25](#_Toc105938539)

# 1. INTRODUCERE

Prima parte a acestei lucrări va conține informații cu privire la conținutul aplicației și motivele alegerii acestei teme.

Obiectivul principal al aplicației „Hai la noi” este de a forma o comunitate prin intermediul căreia toate părțile implicate să aibă de câștigat. Comunitatea amintită anterior va fi formată din locații de interes din industria HoReCa cât și monumente istorice, muzee, puncte de atracție turistică pe care o comunitate turistică (cum ar fi centrul vechi al orașului Brașov) le are de oferit atât localnicilor cât și turiștilor care aleg să îi treacă pragul. Dacă acordăm puțină atenție către turismul de la nivelul municipiului Brașov, împreună cu tot ce implică acesta, putem observa că este o lipsă totală de interes pentru dezvoltarea la nivel comunitar a acestui sector care poate aduce foarte multe beneficii regiunii noastre, cel mai important fiind planul economic. Prin crearea unei comunități, atât locațiile participante cât și clienții vor avea parte de beneficii substanțiale, costul acestora fiind neglijabil. Nu este necunoscut faptul că anumiți factori externi au scăzut atât economia locală cât și cât și cea globală, aici făcând referire atât la criza pandemică instaurată la începutul anului 2020 și resimțită ulterior cât și la războiul din Ucraina. Deoarece factorii externi amintiți anterior au adesea puterea de a dezbina oamenii, cred că este necesară crearea unui nucleu care va aduce un bine comun. În contextul actual, locațiile au nevoie de clienți, iar clienții au nevoie de promoții. Soluția pe care am găsit-o este aplicația „Hai la noi” care oferă clienților reduceri la locațiile participante, iar la rândul lor, acestea vor primi statistici cu ajutorul cărora pot atrage clienți noi. Principalul avantaj al aplicației „Hai la noi” este faptul că i se oferă clientului o diversitate de locații în care poate primi reduceri, totul fiind posibil după parcurgea a numai trei pași simpli. În primul rând, utilizatorul va trebui să își creeze un cont nou cu ajutorul conturilor oferite de platformele „Google” și „Facebook”. Crearea unui cont nou implică completarea anonimă a unui scurt chestionar, acesta fiind modul în care se face colectarea datelor. După ce contul a fost creat urmează autentificarea, ultimul pas fiind scanarea unui cod QR aflat pe afișele postate în cadrul locațiilor participante. După această scanare, clientul va fi eligibil pentru obținerea unei reduceri. Discount-ul va fi aplicat de către personalul locației numai după ce clientul le va arăta acestora mesajul care atestă că scanarea a fost realizată. În continuare, clientul va putea adăuga o recenzie pentru locația vizitată. Această acțiune aduce un mare beneficiu comunității, deoarece obligă locațiile să ofere servicii cât mai bune pentru a se asigura că vor avea o notă cât mai mare, notă care îi poate atrage pe clienți mult mai ușor. Totodată serviciile foarte bune oferite de locațiile participante vor spori și numărul de clienți interesați, în acest mod se reușește creșterea imaginii comunității de care aminteam anterior. Tot în cadrul aplicației, clienții vor avea posibilitatea de a vedea o listă cu locațiile partenere cât și detalii despre acestea. Fiecare vizită completă (o vizită completă implică scanarea unui cod QR) va fi adăugată în istoricul clientului. Acest istoric va conține lista tuturor vizitelor făcute de client din momentul creării contului.

Celălalt actor din cadrul proiectului, și anume locația, va putea accesa statisticile oferite de site-ul web dedicat. Prin intermediul acestor statistici, administratorii vor avea o privire de ansamblu asupra afacerii lor. Aceștia vor cunoaște cu exactitate profilul clientului care le calcă pragul, fapt care le poate schimba strategia de marketing pentru a crește veniturile afacerii. Tot în cadrul site-ului, clienții vor avea la dispoziție o hartă care va conține toate locațiile partenere pentru o informare mai ușoară cu privire la localizarea acestora.

# 2. TEHNOLOGII UTILIZATE

2.1. REACT NATIVE

React Native este un framework open-source UI JavaScript creat de Meta Platforms, Inc. și lansat în anul 2015. Acesta permite dezvoltarea de aplicații native IOS, Android, Android TV, macOS, tvOS, Windows și UWP. Framework-ul React Native este bazat pe React, librăria JavaScript folosită pentru dezvoltare UI, dar în loc să vizeze site-urile web, acesta este folosit pentru dezvoltarea aplicațiilor mobile native. Cu alte cuvinte, acest framework poate fi folosit pentru dezvoltarea aplicațiilor mobile care arată și se simt „nativ”.

Similar cu modul de funcționare al React pentru Web, aplicațiile React Native sunt scrise folosind un amestec de JavaScript și JSX. Pentru o mai bună înțelegere a modului în care React Native funcționează, trebuie afirmat faptul că framework-ul apelează API-urile native de randare în Objective-C (pentru IOS) și Java (pentru Android). Astfel, aplicația randează folosind componente reale de UI pentru mobil, motiv pentru care oferă o înfățișare nativă. În acest moment, pentru aplicații mobile, framework-ul suportă doar dezvoltarea în IOS și Android, însă există potențial pentru extindere spre mai multe sisteme de operare în viitor.

Cel mai mare avantaj al React Native este faptul că acesta permite utilizarea aceluiași cod pentru rularea aplicațiilor pe IOS și Android, rezolvând o problemă majoră pentru dezvoltarea aplicațiilor mobile, și anume necesitatea dezvoltării a câte o aplicație în parte pentru fiecare sistem de operare pentru telefoane mobile. În plus, React nu folosește firul principal de UI, astfel menținând performanța ridicată a aplicațiilor. Un alt avantaj este ciclul update-urilor în React Native, acesta fiind făcut prin intermediul props-urilor sau state-urilor, framework-ul randează din nou view-urile din care acestea fac parte.

2.2. JAVASCRIPT

JavaScript este un limbaj de scripting orientat pe obiecte care a fost lansat în anul 1995 de către Netscape, fiind inspirat de limbajul de programare Java. Acest limbaj de programare poate fi folosit atât pentru partea de server cât și cea de client, fiind creat pentru dezvoltarea interactivă a paginilor web.

Acest limbaj de programare are la bază limbajul de programare C, asemenea majorității limbajelor de programare moderne. Implementarea lui JavaScript este compusă din trei componente separate:

1. ECMAScript – limbaj care determină sintaxa, instrucțiunile, operatorii, obiectele, tipurile și cuvintele-cheie.
2. Document Object Model (DOM) – interfață de programare pentru documente web. Aceasta transformă pagina în noduri și obiecte astfel încât programele să poată schimba structura, stilul și conținutul documentului.
3. Browser Object Model (BOM) – include proprietățile și metodele necesare limbajului de programare JavaScript pentru interacțiunea cu aplicația.

2.3. EXPO

Expo este o platformă open-source creată peste React Native, fiind concepută pentru realizarea de aplicații native pentru Android, IOS și Web. React Native nu oferă toate API-urile JavaScript de care programatorii au nevoie, ci doar cele mai primitive funcții. Expo își propune să rezolve această problemă, îmbunătățind React Native prin oferirea unei game largi de API-uri JavaScript menite să rezolve cele mai comune nevoi ale programatorilor. Pe scurt, Expo reprezintă un set de biblioteci native de calitate deja ambalate într-o singură librărie și anume: ExpoKit. În unele cazuri bibliotecile oferite de Expo sunt deja existente în React Native, fiind doar integrate în ExpoKit.

Expo este format din două componente principale: Expo CLI și Expo Client. Expo CLI este un instrument dedicat programatorilor, fiind folosit pentru crearea de proiecte, vizualizarea log-urilor, publicare etc. Pe de altă parte, Expo Client este o aplicație mobile care permite deschiderea proiectelor în timpul dezvoltării, eliminând nevoia trecerii prin XCode sau Android Studio. Totodată, aplicația Expo Client poate fi folosită de către alți utilizatori pentru a accesa o aplicație care a fost publicată în prealabil prin intermediul Expo CLI pe dispozitive IOS sau Android. O altă funcționalitate a aplicației este posibilitatea creării de aplicații independente pentru publicarea ulterioară a acestora în Play Store sau Magazin Play.

Utilizarea Expo oferă o multitudine de avantaje printre care:

* Crearea unui proiect este foarte ușoară .
* Distribuirea aplicațiilor este la îndemână, fiind posibilă prin intermediul unui cod QR.
* Nu este necesar un build pentru rularea aplicațiilor.
* Un proiect poate fi rulat de mai multe persoane în același timp în timp ce programatorul lucrează la acesta.

2.4. API

Pentru acest proiect partea de server, și implicit baza de date, este accesată prin intermediul unor endpoint-uri API. Aplication Programming Interface (API), permite comunicarea dintre două sisteme, în cazul de față baza de date și aplicația mobile. Un API oferă în esență limbajul și contractul pentru modul în care două sisteme interacționează. Fiecare API are documentație și specificații care determină modul în care informațiile pot fi transferate. La fel cum o pagină web este randată, API-urile pot folosi solicitări HTTP pentru a obține informații de la o aplicație web sau un server web.

API-urile sunt de obicei clasificate ca SOAP sau REST și ambele sunt folosite pentru a accesa serviciile web. SOAP se bazează exclusiv pe XML pentru a furniza servicii de mesagerie, în timp ce REST oferă o metodă mai ușoară, folosind URL-uri în cele mai multe cazuri pentru a primi sau trimite informații. REST folosește patru metode HTTP 1.1 diferite (GET, POST, PUT și DELETE) pentru a efectua sarcini. Spre deosebire de SOAP, REST nu trebuie să folosească XML pentru a oferi răspunsul. Serviciile web bazate pe REST pot returna următoarele tipuri de date: Command Separated Value (CSV), JavaScript Object Notation (JSON) și Really Simple Syndication (RSS). Ideea este că în urma unui apel API, rezultatul poate veni într-o formă care este ușor de analizat și manipulat de limbajul în care se dezvoltă aplicația.

Comunicarea efectivă dintre un API și un alt sistem este realizată prin intermediul unui endpoint API. Pentru API-uri, un endpoint poate include o adresă URL a unui server sau serviciu. Fiecare endpoint este locația de unde API-urile pot accesa resursele de care au nevoie pentru a-și îndeplini funcția.

API-urile funcționează folosind apeluri și răspunsuri. Când un API solicită informații de la un server web, acesta va primi un răspuns în funcție de rezultatul returnat.

2.5. LIBRĂRII

Acest subcapitol va conține detalii despre librăriile importante folosite în decursul dezvoltării aplicației. Pentru siguranța autentificării native, acest modul folosește autentificatiorul PKCE.

2.5.1 Expo Google App Auth

Această bibliotecă oferă integrarea autentificărilor prin intermediul unui cont Google pentru aplicațiile Expo folosind un sistem securizat de browser web. Acest modul funcționează în felul următor: în momentul unei autentificări cu succes, se va returna un token de acces, iar prin intermediul acestui token se pot realiza apeluri către API-ul Google, putând fi utilizate API-urile REST oferite de Google direct prin apeluri HTTP.

Pentru a putea fi utilizată metoda de autentificare oferită de Google, trebuie creat în prealabil un proiect în Consola pentru Dezvoltatori Google, prin intermediul acestuia fiind creat și un ID de client Oauth 2.0. De asemenea, programatorii vor fi nevoiți să înregistreze un set separat de ID-uri de client pentru o aplicație autonomă.

2.5.2 Expo Facebook

Acest modul oferă integrare Facebook, cum ar fi autentificarea prin intermediul unui cont Facebook pentru aplicațiile dezvoltate în React Native. Expo expune un API nativ minim, deoarece Graful API oferit de Facebook poate fi accesat direct prin apeluri HTTP.

Pentru a putea fi folosită această librărie, programatorii trebuie să creeze o aplicație prin intermediul paginilor de dezvoltator Facebook, și implicit API-ul oferit de aceștia. Prin înregistrarea unei aplicații, se va genera un ID unic al acesteia, ID care va fi folosit ulterior ca parametru pentru apelarea unei funcții prin intermediul căreia un utilizator se va putea autentifica în aplicație cu contul Facebook.

2.5.3. Axios

Axios este un client HTTP bazat pe promisiuni pentru node.js și browser. Este izomorf, cu alte cuvinte, poate rula în browser și nodejs cu același cod de bază. Pe partea de server folosește modulul HTTP nativ node.js, în timp ce pe partea de client folosește XMLHttpRequests.

Această bibliotecă oferă următoarele caracteristici:

* Opțiunea de XMLHttpRequests din browser.
* Opțiunea de http request prin node.js
* Acceptă API-uri promise.
* Interceptează cererea și răspunsul
* Posibilitatea transformării datelor de apel și a rezultatelor
* Anularea apelurilor
* Transformarea rezultatelor automate pentru date de tip JSON

Acest pachet a fost folosit pentru cereri API în vederea legăturii dintre client și server. Cu alte cuvinte, „Axios” facilitează conexiunea dintre aplicație și baza de date, atât pentru popularea acesteia cât și în cazul preluării de date în vederea afișării acestora

2.5.4. Expo Barcode Scanner

Expo oferă prin intermediul acestui model o componentă React care randează un vizor pentru camera dispozitivului prin intermediul căruia se vor scana codurile de bare care vor apărea în cadru. Prin intermediul acestui pachet open-source a fost rezolvată problema scanării codurilor QR. Această librărie facilitează atât cererea permisiunii pentru utilizarea camerei telefonului cât și decriptarea codului QR și returnarea unui șir de caractere care a fost folosit pentru verificarea existenței în baza de date a unei locații căreia îi aparține acest cod.

2.5.5. React Native Ratings

Pentru afișarea rating-urilor primite de locații și alegerii unei note de către clienți în momentul adăugării unei recenzii, a fost folosit pachetul React Native Ratings care s-a dovedit a fi foarte folositor în momentul implementării fiind totodată foarte ușor de utilizat de către client.

# 3. PREZENTAREA SISTEMULUI

Scopul lucrării este acela de a crea vehiculul software prin care consumatorii își înregistrează în platformă interacțiunea cu locația. Aplcația a fost dezvoltată utilizând React Native, acesta fiind un framework bazat pe JavaScript care permite crearea de aplicații mobile native atât pentru IOS cât și pentru Android. Motivul alegerii acestui framework este dat de faptul că React Native permite utilizarea aceluiași cod pentru rularea aplicațiilor pe IOS și pe Android. În continuare va fi prezentat sistemul aplicației *„Hai la noi”* , fiind detaliate componentele importante din care acesta este alcătuit.

3.1. CERINȚE FUNCȚIONALE

* Autentificare
* Creare cont
* Ieșire din cont
* Completare chestionar
* Scanare cod QR
* Adăugare recenzie
* Adăugare imagini
* Vizualizare listă locații
* Vizualizare detalii locații
* Vizualizare istoric utilizator

3.2. COMPONENTE

Aplicația *Hai la noi* este formată din mai multe ecrane, acestea fiind reprezentate în cod sub forma unor fișiere de tip JavaScript. Părțile importante din modul în care acestea au fost implementate vor fi explicat în continuare.

3.2.1. Pagina Login

Aceasta este pagina de pornire a aplicației, prin intermediul căreia utilizatorul își va putea crea un cont nou sau se va putea autentifica în contul deja existent. UI-ul acestei pagini este format dintr-un container custom, în interiorul căruia se află o componentă de tip Image în care se afișează local logo-ul aplicației, în continuare fiind plasate titlurile și două componente Button reprezentând modalitatea de creare de cont/autentificare prin intermediul conturilor Google sau Facebook. Spre exemplu pentru autentificarea prin intermediul contului Google, în momentul apăsării unui buton va fi apelată funcția necesară, denumită *handleGoogleSignIn* prin intermediul proprietății *onPress*. În această funcție sunt mai întâi configurate id-urile necesare pentru apelarea API-ului oferit de Google, pentru ca în continuare, cererea să returneze un răspuns prin intermediul căruia se rețin tipul autentificării (succes sau eșuat) și un obiect de tip utilizator în care se rețin datele acestuia. Dacă autentificarea prin Google a avut succes, în continuare se va apela o funcție care are rolul de a verifica existența unui cont în baza de date în funție de adresa de email a utilizatorului. Verificarea este făcută prin intermediul funcției *isUserRegisteredGoogle* (Fig. 1) cu ajutorul apelării unui API, funcția returnând valoarea *true* în cazul în care există deja un cont cu adresa respectivă de email sau valoarea *false* în caz contrar. În cazul în care contul căutat în baza de date este inexistent, pe ecran va apărea o componentă de tip *Alert* în care utilizatorul va trebui să își aleagă limba dorită (Engleză sau Română) pentru completarea chestionarului care reprezintă modalitatea prin care se face colectarea datelor. După apăsarea pe butonul care reprezintă limba dorită, utilizatorul va fi redirecționat către pagina *Questionnaire* (vezi 3.2.2.). Revenind la funcția de verificare a existenței contului, pe ramura de adevăr, utilizatorul va fi redirecționat prin intermediul funcției *persistLogin* spre pagina *Welcome (vezi 3.2.3.).*

Datele utilizatorului sunt reținute global prin intermediul hook-ului *CredentialsContext* și a lirăriei *Async Storage*. Datele sunt salvate într-un state prin intermediul hook-ului custom amintit mai sus. Înainte de a fi redirecționat către pagina *Welcome,* credențialele utilizatorului, și anume adresa de email, numele și ID-ul unic sunt reținute pentru a putea fi folosite ulterior în toate celelalte pagini ale aplcației. În momentul în care utilizatorul alege să se delogheze, se va șterge elementul care reține credențialele din AsyncStorage pentru ca datele turistului să nu rămână salvate.

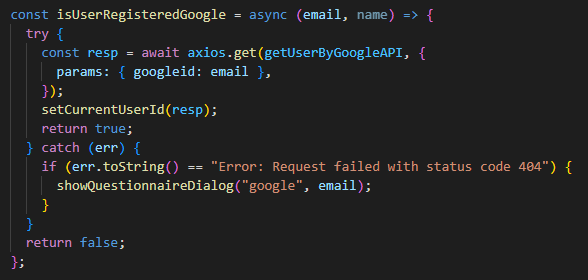


Fig. 1. Funcție care verifică dacă există un cont Google în baza de date

3.2.2. Pagina Questionnaire

Această pagină conține o funcționalitate vitală pentru economia proiectului, prin intermediul acesteia fiind făcută colectarea datelor pentru crearea statisticilor ulterioare prin intermediul site-ului web dedicat locațiilor participante. Chestionarul este format din 6 întrebări de tip demografic. Pentru prima întrebare, utilizatorul va răspunde prin intermediul componentei *Alert* prin care selectează limba dorită, parametrii fiind transmiși din pagina *Login* în pagina curentă prin intermediul *route.params*. Parametrii transmiși sunt preluați, iar în funcție de limba selectată, o variabilă va reține întrebările dintr-un fișier JavaScript unde întrebările și răspunsurile sunt salvate sub formă JSON. Elementele din JSON au o formă standard care trebuie respectată pentru rularea corectă a aplicației, aceasta fiind alcătuită din următoarele proprietăți:

* Question – șir de caractere care conține textul întrebării.
* Type (simple/dropdown) – folosită pentru a face diferența între tipurile de întrebări.
* Answers – array format din șiruri de caractere care conțin textul răspunsurilor.
* Property – șir de caractere; folosită ca o cheie pentru salvarea întrebărilor sub aceiași formă în baza de date, indiferent de limba aleasă.

Prima întrebare din cadrul chestionarului este țara de proveniență pentru limba engleză, respectiv județul de domiciuliu pentru limba română. În ambele cazuri, răspunsurile sunt afișate în interiorul unei componente *DropDownPicker*, aceasta aparținând unei librării open-source. După selectarea unei opțiuni din listă, va apărea un buton denumit *Next* care în momentul apăsării va apela funcția *handleNext*  (Fig. 2.) care va duce utilizatorul la următoarea întrebare.

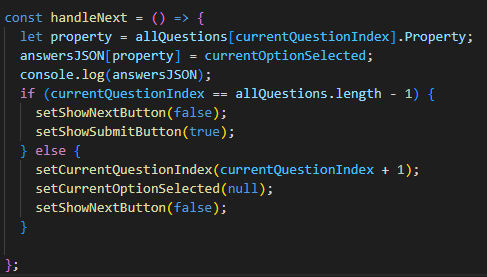


Fig. 2. Funcția care schimbă următoarea întrebare

Celelate întrebări sunt de tip simple, acestea având între 3 și 6 opțiuni de răspuns, acestea fiind următoarele:

* Care este vârsta ta?
* Care este scopul vizitei tale în Brașov?
* Câte nopți petreci în Brașov?
* Cât de des iei masa în Brașov?

Pentru afișarea întrebării corecte și a răspunsurilor aferente ale acesteia, a fost folosită o variabilă counter care a fost incrementată cu o unitate după fiecare apăsare a butonului Next. În momentul în care utilizatorul selectează un răspuns pentru o întrebare de tip simple, background-ul răspunsului va fi colorat, acest lucru fiind făcut cu ajutorul unui state în care este reținut id-ul răspunsului selectat. Când se ajunge cu valoarea counter-ului la numărul de întrebări aflate în JSON-ul în care sunt salvate întrebările, după alegerea unei opțiuni, pe ecran va deveni vizibil un buton denumit Submit, care prin apăsare apelează funcția postNewUser care are ca parametrii variabilele loginType și email. Primul parametru reține tipul de cont pe care utilizatorul dorește să îl creeze (Facebook sau Google), iar al doilea reține adresa de email a acestuia. În funcție de tipul de cont, se va face o cerere API de tip POST prin intermediul căreia se vor salva datele contului nou creat în baza de date. Între aceste date se află și un fișier de tip JSON care reține după cum spuneam mai sus, proprietatea întrebării cu răspunsul dat de utilizator.

3.2.3. Pagina Welcome

Aceasta este cea mai simplă pagină dintre toate cele prezente în cadrul aplicației. În momentul autentificării, utilizatorul va fi redirecționat către această pagină care conține două componente de tip *Text* prin intermediul cărora sunt afișate un mesaj de bun venit și numele utilizatorului. Sub acestea se află o componentă de tip *Button* cu textul *Apasa pentru scanare*, care în momentul apăsării va apela o funcție care va redirecționa utilizatorul spre pagina *Scanner*, funcționalitățile acesteia fiind detaliate în subcapitolul următor.

3.2.4. Pagina Scanner

Aceasta este pagina care se ocupă de scanarea codurilor QR ale locațiilor participante. Pentru început, după randarea paginii, prin intermediul hook-ului *useEffect* se cere/verifică permisiunea pentru folosirea camerei de filmat a telefonului. Statusul permisiunii va fi reținut în cadrul unui state, iar în funcție de acest state va fi afișat un mesaj de atenționare (în cazul în care nu au fost acceptate permisiunile) sau o fereastră prin intermediul căreia se poate scana un cod QR. Pentru această scanare este nevoie de simpla încadrare a unui cod QR în fereastră care va apela funcția *handleBarCodeScanned* (Fig. 3.). Această funcție va seta mai întâi un state cu valoarea *true* pentru a opri funcționalitatea de scanare, mai apoi va fi verificat cu ajutorul parametrului *data,* care conține valoarea codului QR decriptată, și a funcției *verifyLocationRegistered* (Fig. 4.)existența unei locații cu acel cod QR în baza de date. Valoarea returnată de această funcție va fi reținută de o variabilă, iar în funcție de această variabilă va fi afișat mesajul *Codul scanat nu apartine unei locatii partenere* sau în cazul în care codul scanat este bun, va apela funcția *showConfirmDialog* (Fig. 5.). Această funcție returnează o componentă de tip *Alert*  care va avea două butoane cu textele *Da* și *Nu*. Pentru apăsarea butonului afirmativ, utilizatorul va fi redirecționat către pagina *Discount* (vezi 3.2.5.), iar pentru celălalt caz va fi închisă caseta de alertă. După ce a fost făcută o scanare, pentru a seta valoarea *false* pentru state-ul care oferă sau oprește opțiunea unei scanări, utilizatorul va trebui să apese pe o componentă de tip *Button* cu titlul *Apasa pentru o noua scanare*.

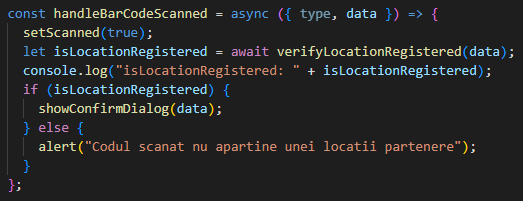


Fig. 3. Funcție care acționează după scanarea unui cod QR

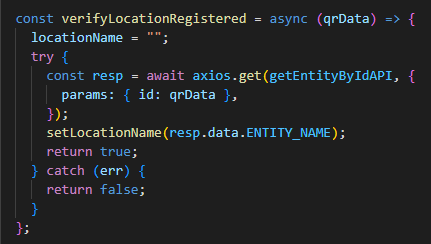


Fig. 4. Verificarea existenței locației în baza de date

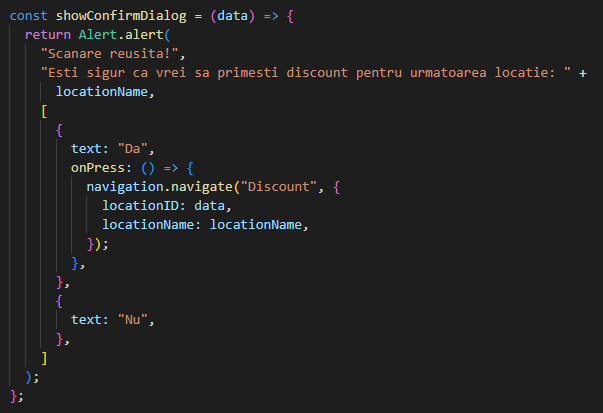


Fig. 5. Căsuță de alertă pentru acceptul/refuzarea unui discount

3.2.5 Pagina Discount

Funcționalitatea acestei pagini este de a oferi turiștilor posibilitatea de a primi un discount pentru o anumită locație după scanarea codului QR care aparține acesteia, însă se mai ocupă și de adăugarea unei vizite în baza de date. La momentul randării, prin intermediul hook-ului *useEffect* se va reține într-o variabilă data și ora locale ale locației utilizatorului, iar pe urmă va fi apelată funcția *postTransaction* care are rolul de face o cerere API de tip *post*, prin intermediul căreia sunt transmise către server datele necesare pentru adăugarea unei tranzacții dintre utilizator și locație. Pentru ca turistul să poată primi discount-ul cuvenit, acesta va trebui să arate această pagină unui membru din staff-ul locației, acest lucru fiind semnalat prin componentele de tip *Text* care sunt afișate prin intermediul paginii. După ce un membru din staff-ul locației a luat la cunoștință faptul că turistul trebuie să primească un discount pentru ce a comandat, turistul poate folosi în continuare aplicația prin adăugarea unei recenzii. Această funcționalitate poate fi obținută prin apăsarea unei componente de tip *Button* cu textul *Adauga un review* care în momentul apăsării va redirecționa utilizatorul către pagina *Review*, aceasta fiind detaliată în subcapitolul următor.

3.2.6 Pagina Review

Această pagină va fi folosită de utilizatori în momentul în care vor dori să adauge o recenzie pentru o locație pe care aceștia au vizitat-o. Prima componentă vizibilă în pagină va fi elementul *Rating* (Fig. 6.), oferit de către pachetul open-source *React Native Ratings*. Prin intermediul acestei componente, utilizatorii vor putea adăuga o notă de la 1 la 5 care va fi salvată prin intermediul unei funcții care setează valoarea unui state și a proprietății *onFinishRating* care obține valoarea notei selectate de către utilizator. Sub aceasta se află o componentă *View* care are ca și copil o componentă *TextInput* prin intermediul căreia utilizatorii vor putea adăuga un mesaj. Acest mesaj va fi salvat sub forma unui șir de caractere și va fi setat unui state prin intermediul unei funcții.

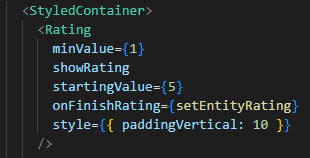


Fig. 6. Componenta Rating

Mai departe se va afla o componentă *Button* care la apăsare va apela funcția *postReview* care va face o cerere API de tip *post* prin intermediul căreia recenzia va fi adăugată în baza de date, alături de data și ora la care a fost efectuată. Înainte, însă, de a face cererea API, se verifică dacă utilizatorul a selectat o notă pentru locație, această acțiune fiind una obligatorie. Această verificare va fi făcută prin intermediul unei condiții *if* în care se verifică dacă variabila care reține nota este nulă, iar în caz afirmativ va fi apelată o funcție care va afișa un mesaj de alertă prin care utilizatorului i se atrage atenția asupra faptului că selectarea notei este obligatorie. Înainte de a adăuga recenzia, utilizatorii mai pot adăuga o poză cu bonul primit prin apăsarea butonului intitulat *Adauga o poza*. Apăsarea acestui buton va apela funcția *openCamera* (Fig. 7.). Această funcție va cere întâi permisiunea pentru a folosi camera telefonului prin intermediul librăriei *ImagePicker*, aceasta fiind oferită de Expo. Dacă utilizatorul a acceptat permisiunea utilizării camerei telefonului, imaginea va fi reținută de o variabilă denumită *result*, prin intermediul căreia, ulterior va fi setată unui state valoarea uri a imaginii. Colectarea de bonuri va fi folosită pentru o funcționalitate viitoare a aplicației.

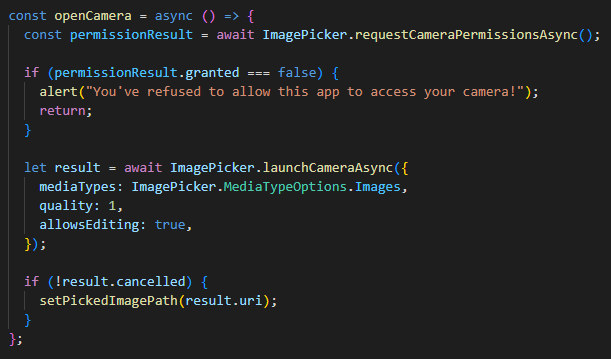


Fig. 7. Funcție pentru salvarea unei poze

3.2.7. Meniul aplicației

Meniul aplicației a fost creat prin intermediul unei componente de tip *Drawer Navigator*. Această componentă randează un sertar de navigare care poate fi deschis prin glisarea telefonul pe ecran din partea stângă spre dreapta sau prin apăsarea butonului *hamburger* aflat în partea din stânga sus a ecranului. Pentru aplicația „Hai la noi” meniul nu poate fi accesat din toate paginile. Paginile care nu au acces asupra acestuia sunt: *Login, și Questionnaire*. Limitarea accesului în meniul aplicației a fost făcut pentru funcționarea ideală a aplicației. Prin intermediul meniului, utilizatorul va putea accesa următoarele pagini: *Welcome*, *Entities* (vezi 3.2.8)și*Transactions (vezi 3.2.10)*. Totodată, delogarea se va putea face tot prin intermediul meniului, la finalul acestuia aflându-se opțiunea *Ieșire din cont.* Acest meniu al aplicației oferă și funcționalitatea de navigare între pagini.

3.2.8. Pagina Entities

Această pagină este oferă utilizatorului o listă compusă din locațiile participante, la care acesta poate primi discount. Lista de locații este salvată prin intermediul unui state. La momentul randării, prin hook-ul *useEffect* se apelează funcția *getAllEntities* care va seta în state JSON-ul returnat de cererea API de tip *GET.* Lista obținută va fi mai apoi randată prin intermediul unei componente *FlatList* (Fig. 8.).



Fig. 8. Randarea paginii Entities

Aceasta va conține un separator de elemente, mai exact un *View* care este un spațiu gol menit să delimiteze elementele din listă. Componentele de tip *FlatList* necesită ca elementele ce vor fi randate să aibă o cheie unică, iar pentru asta a fost folosită proprietatea *keyExtractor* care va oferi fiecărui element o cheie unică numerică în funcție de poziția acestora din listă. Pentru randarea fiecărei locații în parte va fi folosită proprietatea *renderItem* care primește ca parametru elementul curent din listă, pentru ca mai departe să fie afișată proprietatea *ENTITY\_NAME*, proprietate preluată din baza de date. Fiecare element al listei va avea ca părinte o componentă de tip *TouchableOpacity* care are rolul de a oferi utilizatorului opțiunea de a apăsa pe o locație pentru a afla mai multe informații despre

aceasta prin intermediul paginii *EntityInformation* care va fi detaliată în subcapitolul următor.

3.2.9. Pagina EntityInformation

Această pagină va conține următoarele detalii despre o locație selectată:

* Logo-ul locației.
* Numărul total de vizite prin intermediul aplicației.
* Numărul total de turiști care au vizitat locația.
* Numărul total de recenzii oferite de utilizatori.
* Nota (între 1 și 5) locației afișată prin stele.
* Listă de recenzii.

De fiecare dată când valoarea state-ului care reține numele locației va fi schimbată, implicit și la prima randare, vor fi apelate următoarele funcții: *getAllEntities, getLast10Reviews* și *getEntityById*. Prima dintre acestea va face o cerere API de tip *GET* pentru a reține toate locațiile din baza de date. După ce s-a primit rezultatul, se va apela o funcție care îl va avea ca parametru, rolul acesteia fiind de a căuta în listă locația cu numele transmis în momentul navigării de la o pagină la alta. După ce numele va fi setat se revine la apelarea celei de-a doua funcții din *useEffect* și anume cea care se ocupă de preluarea recenziilor locației selectate. Prin intermediul unei cereri API de tip *GET* se preia rezultatul și se setează ulterior valoarea unui state cu datele obținute. Mai departe, se revine la ultima funcție din corpul *useEffect* care, printr-o cerere API setează valoarea unui state cu logo-ul locației selectate, logo care va fi folosit ulterior pentru afișare. După cum aminteam mai sus, după logo-ul aplicației, urmează câteva date despre vizite/turiști iar ulterior prin intermediul unei componente *Rating* se afișează scorul mediu al locației, nota provenită din recenziile utilizatorilor. Această componentă are setată valoarea *true* pentru proprietatea *readonly* pentru ca utilizatorii să nu îi poată schimba valoarea din greșeală, fapt care ar putea duce la confuzie.

În continuare, sub forma unei componente *FlatList* sunt afișate recenziile oferite de utilizatori, fiecare dintre acestea fiind formate din: nota oferită, data la care a fost adăugată și comentariul (dacă acesta există).



Fig. 9. Componentă pentru afișarea listei de recenzii

3.2.10. Pagina Transactions

Ultima pagină din componența aplicației „Hai la noi” este pagina tranzacțiilor, aceasta fiind după cum îi spune și titlul, *Istoric utilizator*, modalitatea prin care turiștii își vor putea vedea vizitele făcute în trecut. Pentru ca lista de tranzacții să fie actualizată la fiecare randare a paginii, va fi folosit hook-ul *useEffect* fără niciun parametru. În corpul acestuia va fi apelată funcția *getAllTransactionsByUserID* care va returna din baza de date, sub formă de JSON, toate vizitele făcute de utilizatorul curent. Asemeni tuturor listelor afișate în aplicație, și pentru istoricul vizitelor a fost folosită o componentă *FlatList*. Fiecare element din listă poate fi apăsat, iar în momentul apăsării se apelează funcția *pressHandler* care primește ca parametrii numele locației și id-ul acesteia. În această funcție se apelează mai întâi două funcții diferite care au rolul de a reține numărul total de vizite făcute de utilizator la locația selectată și numărul total de recenzii adăugate de acesta aceleiași locații. Aceste date sunt obținute din baza de date prin intermediul unor cereri API de tip *GET*. Mai departe sunt comparate numărul de vizite cu numărul de recenzii, iar în cazul în care există mai multe vizite decât review-uri se apelează funcția *showAlert* (Fig. 10.) având primul parametru valoarea *true*, altfel se va trimite ca parametru valoarea *false*.



Fig. 10. Funcție pentru afișarea unei alerte după apăsarea pe o vizită

În funcție de parametrul boolean denumit *isReviewable,* vor fi setate mesajul și titlul componentei *Alert* folosită pentru afișare. Această căsuță de alertă va avea două butoane denumite *Da* și *Nu*. Pentru butonul nu, se va închide caseta de alertă. Prin apăsarea butonului *Da*,însă, în funcție de cazul în care ne aflăm se pot întâmpla următoarele: fie utilizatorul va fi redirecționat spre pagina *Review* pentru a adăuga o recenzie, fie va fi redirecționat spre pagina *EntityInformation*.

3.2.11. Participanți

Deoarece aplicația nu a fost lansată în producție până la momentul realizării acestui proiect de cercetare, voi ilustra utilizarea acesteia folosind date simulate. Datele simulate au fost generate în Python și constau din 2000 de turiști virtuali care au fost trimiși la plimbare prin Brașov. Fiecare turist a efectuat între 1 și 10 vizite la restaurant și între 1 și 3 vizite la obiective turistice. Atât la restaurant cât și la obiectivele turistice s-a acordat rating aleatoriu între 1 și 5 in 30% dintre vizite. De exemplu, dacă un turist a făcut 6 vizite la restaurant atunci 2 dintre acestea implică și rating, în cazul în care turistul a făcut 3 vizite la muzee, doar una dintre acestea implică și rating.

Vizitatorii virtuali au completat în mod aleator chestionarul, rezultând o distribuție uniformă a vizitatorilor în contextul diferitelor atribute considerate: grup vârstă, motivul vizitei, numărul de nopți petrecute, tipul vizitatorului (localnic, român, străin).

3.2.12. Măsurători

Datele virtuale corespund datelor care s-ar putea strange în timp cu ajutorul aplicației mobile care face obiectul acestei lucrări de disertație. Pentru a ilustra impactul utilizării acestei aplicații mobile prezentăm diversele analize bazate pe datele sintetice descrise mai sus.

Datele înregistrate măsoară traficul și consumul. Vizitatorii sunt grupați în turiști și turiști activi (grup care conține și localnicii). Măsurătorile sunt de mai multe tipuri:

* Măsurători individuale - locație și locație vs categorie. De exemplu un restaurant comparat cu categoria tuturor restaurantelor. Metrici folosite: comparație la săptămână a traficului în locație/categorie. Analiza profilului și a cotei de piață pe segmente date de atributele colectate.
* Măsurători la nivel de categorie
  + Vizite vs rating – istoric
  + Ratinguri pentru ultima săptămână

**BIBLIOGRAFIE**

Va conține minim 30 surse distincte. Se vor indica numai acele surse care au fost utilizate în lucrare. Sursele vor fi aranjate în ordine alfabetică; mai întâi autorii, ordonați după numele de familie, fiind urmați de lucrările fără autor (rapoarte, statistici, documente concepute de diferite instituții) şi de paginile web. Modul de scriere este inserat în modelul de conținut.

Exemplu de bibliografie:

1. Băcanu, B. (2009), *Practici de management strategic*, Polirom, București.

2. Bronner, F., Hoog, R. (2011), Vacationers and eWOM: Who Posts, and Why, Where, and What?*, Journal of Travel Research*, 50, 1, pp. 15-26.

3. Filip, R., Iamandi, E. (2012), *Management internațional*, Academia de Studii Economice, București.

4. Jeannet, J., Hennessey, D. (1998), *Global Marketing Strategies*, Houghton Mifflin, Boston.

5. Miraz, M.H., Habib, M. (2016), ICT Adoption in Small and Medium Enterprises: An Empirical Evidence of Service Sectors in Bangladesh, *Journal of Economics, Business and Management*, 4, 8, pp. 482-485.

6. Sahut, J.M. (2009), The impact of Internet on pricing strategies in the tourism industry, *Journal of Internet Banking and Commerce*, 14, 1, pp. 1158-1162.

7. \*\*\* (2017), *Bilanțul contabil şi contul de profit şi pierdere,* SC Mara SRL.

8. \*\*\* (2016), *European indicators on sustainability*, Eurostat.

9. \*\*\* (2014), *Sinteza datelor privind investițiile străine*, Banca Națională a Romȃniei.

10. http://www.aippimm.ro/categorie/programe/proiecte-proceduri-implementare-2016/

11. http://dce.gov.ro/Materiale%20site/OUInternatioanliz.pdf

12. https://www.listafirme.ro/continental-logistics-srl-19005084/

**ANEXE**

Nu se numerotează. Pot conține chestionare, tabele de dimensiuni mai mari, imagini, documente preluate de la firmă etc.