



UNIVERSITATEA DIN
BUCUREŞTI



FACULTATEA DE
MATEMATICA ŞI
INFORMATICA

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

TAKEMEOUT

Absolvent
Bejenariu David-Cosmin

Coordonator științific
Conf. Univ. Dr. Boriga Radu-Eugen

Bucureşti, iunie 2023

Rezumat

Cu toții am fost nevoiți să urmăm direcțiile unei hărți pentru a ajunge într-un anumit loc, existând și situații în care ne-a fost greu să corelăm reprezentarea direcțiilor de pe hartă cu realitatea. În același timp, au fost momente în care ar fi fost foarte convenabil să ne aducem aminte de restaurantele favorite situate în apropierea noastră, sau orice altă locație de interes despre care am aflat din diverse surse.

În acest fel a apărut conceptul **TakeMeOut**: o aplicație destinată dispozitivelor iOS ce încorporează o experiență completă de căutare, memorare și **notificare** a punctelor de interes, împreună cu generarea de direcții ce folosește sistemul clasic de hărți, dar și **realitatea augmentată** pentru îmbunătățirea procesului de orientare.

Abstract

Everybody had to follow the directions of a map at some point to get to a certain place, including times when corresponding between map directions and reality became quite tricky. There were also moments when recalling a great restaurant nearby, or any other place we might have heard about, would have been highly convenient.

Thus, this is how **TakeMeOut** came into view: an app for the iOS devices that offers a full point of interest search, preserve and **advise** experience, along with wayfinding that makes use of the standard mapping system and **augmented reality** to enhance the process.

Cuprins

1 Introducere	5
1.1 Motivație	5
1.2 Preliminarii	5
1.2.1 Căutarea, salvarea și semnalarea punctelor de interes	6
1.2.2 Generarea direcțiilor folosind harta	6
1.2.3 Generarea direcțiilor folosind realitatea augmentată	6
1.3 Domenii abordate	6
1.4 Aplicații similare	6
1.4.1 Apple Maps	6
1.4.2 Google Maps	7
1.4.3 TripAdvisor	7
1.5 De ce „TakeMeOut”?	7
1.6 Structura lucrării	8
2 Tehnologii folosite	10
2.1 Swift și Xcode	10
2.2 Apple SDKs	11
2.2.1 UIKit	11
2.2.2 MapKit	13
2.2.3 Core Location	15
2.2.4 ARKit și RealityKit	16
2.2.5 User Notifications	19
2.2.6 AVFoundation	19
2.3 Framework-uri	19
2.3.1 Firebase	19
2.3.2 Core Data	20
2.4 Altele	21
2.4.1 Cocoa Pods - Floating Panel	21
2.4.2 Foursquare Places API	22

3 Arhitectura și funcționalitatea aplicației	23
3.1 Baza de date	23
3.2 Models	24
3.3 Views	25
3.4 View Controllers	25
3.5 Utilities	31
3.6 Autentificarea utilizatorilor	32
3.7 Mecanismul de trimitere a notificărilor	32
3.8 Salvarea și ștergerea locațiilor favorite	33
3.9 Afisarea instrucțiunilor de mers	33
3.9.1 Folosind harta	33
3.9.2 Folosind realitatea augmentată	37
4 Prezentarea aplicației	41
4.1 UI/UX	41
4.2 Înregistrare și Autentificare	42
4.2.1 Autentificarea cu Google	43
4.3 Pagina principală	44
4.3.1 Vizualizarea locațiilor salvate	44
4.4 Secțiunea de profil	45
4.4.1 Adăugarea/modificarea fotografiei de profil	46
4.5 Căutarea punctelor de interes	46
4.6 Vizualizarea punctelor de interes	47
4.6.1 Adăugarea/eliminarea unui punct de interes	48
4.7 Primirea notificărilor	49
4.8 Afisarea rutelor de direcționare	50
4.9 Urmarea instrucțiunilor de mers	50
4.9.1 Folosind harta	51
4.9.2 Folosind realitatea augmentată	52
5 Concluzii	53
5.1 Impresii și provocări întâmpinate	53
5.2 Perspective	54
Bibliografie	55

Capitolul 1

Introducere

1.1 Motivație

La baza dezvoltării aplicației stau toate oportunitățile ratate de a vizita anumite locații despre care am auzit din diferite surse, locații pe care le-am salvat într-o listă de „Locuri de vizitat”, dar pe care nu am ajuns niciodată să le bifez pe motiv că am uitat de existența lor. În urma unui sondaj realizat printre cunoșcuți, am venit cu ideea proiectului „TakeMeOut” - o aplicație care te anunță când ești aproape de un punct de interes.

În același timp, datorită pasiunii pentru tehnologie încă din copilărie și conștient fiind de puterea dispozitivelor pe care le folosim zilnic, mi-am dorit să exploatez funcționalitățile acestora, cu precădere cele ale senzorului LiDAR¹. Astfel, integrând capacitatele GPS²-ului, Wi-Fi-ului, Bluetooth-ului, magnetometrului, barometrului, arhitecturii celulare, busolei, camerelor de filmat și senzorului LiDAR, ce sunt încorporate în majoritatea dispozitivelor mobile³, am adus îmbunătățiri semnificative proiectului meu prin adăugarea direcțiilor de mers ce folosesc atât sistemul clasic de hărți, cât și realitatea augmentată.

1.2 Preliminarii

Aplicația își îndeplinește obiectivul prin următoarele 3 funcționalități principale:

- Căutarea, salvarea și semnalarea punctelor de interes;
- Generarea direcțiilor folosind harta;
- Generarea direcțiilor folosind realitatea augmentată.

¹LiDAR = Light Detection And Ranging

²GPS = Global Positioning System

³cu excepția senzorului LiDAR, ce se găsește doar la anumite dispozitive de la Apple

1.2.1 Căutarea, salvarea și semnalarea punctelor de interes

Aplicația permite utilizatorului să caute locații noi prin intermediul barei de căutare puse la dispoziție. Acesta poate introduce atât numele complet al unei locații (ex. *Palatul Parlamentului*), adresa acesteia (ex. *Calea 13 Septembrie, 2-4, București*), cât și fracțiuni de nume (ex. *palatul*) sau doar cuvinte cheie care să sugereze o anumită categorie de locații (ex. *muzeu*). Aceasta poate selecta mai departe o locație din tabelul de rezultate pentru a o vizualiza. Odată ce o locație este în modul de vizualizare, utilizatorul poate vedea detaliile acesteia, având posibilitatea de a o adăuga la lista de favorite.

Lista locațiilor salvate este vizibilă pe pagina principală a aplicației, iar odată ce aplicația intră în background, utilizatorul va primi o notificare de fiecare dată când una dintre locațiile salvate se află în apropierea sa.

1.2.2 Generarea direcțiilor folosind harta

Din modul de vizualizare a locației, utilizatorul poate primi direcții de mers dacă este îndeajuns de aproape de aceasta. Direcțiile pe hartă au în prim plan harta centrată pe locația curentă a utilizatorului și care urmează linia de mers, împreună cu instrucțiunile de orientare.

1.2.3 Generarea direcțiilor folosind realitatea augmentată

De-a lungul sesiunii de orientare, utilizatorul poate opta pentru a primi instrucțiuni cu realitatea augmentată, ce proiectează obiecte indicative de-a lungul rutei de mers, în spațiul real, cu ajutorul camerei de filmat și senzorului LiDAR.

1.3 Domenii abordate

Domeniile care stau la baza acestei lucrări sunt:

- **Dezvoltarea aplicațiilor mobile** - aplicația „TakeMeOut”, realizată în **Swift** și disponibilă pe dispozitivele iOS, fiind obiectul principal al acestei lucrări;
- **Realitatea augmentată** - utilizată în îmbunătățirea experienței de orientare.

1.4 Aplicații similare

1.4.1 Apple Maps

Apple Maps este aplicația de hărți pe care compania Apple o pune la dispoziție tuturor dispozitivelor sale în mod implicit. Printre funcționalitățile sale se numără căutarea,

vizualizarea și salvarea locațiilor și punctelor de interes, generarea direcțiilor de mers prin intermediul mai multor mijloace de transport (pedestrian, mașină, transport în comun). Pentru fiecare locație se pot vizualiza detalii precum adresă, fotografii, descriere, program de funcționare, recenzii, date de contact și alte facilități. Aplicația permite și vizualizarea punctelor de interes în formatul *Street View*.

1.4.2 Google Maps

Google Maps este serviciul de la Google care implementează aceleași funcționalități ca cel detaliat anterior, dar care datează de mai mult timp și este disponibil pe toate sistemele de operare mobile care suportă serviciile Google, fiind cea mai folosită aplicație din categoria sa. O funcționalitate cu care aplicația aceasta vine în plus este orientarea utilizatorilor folosind realitatea augmentată.

1.4.3 Tripadvisor

Tripadvisor este o platformă online, dar și o aplicație mobilă, care oferă informații și recenzii despre hoteluri, restaurante și alte atracții turistice. Recenziile sunt scrise de călători și pot fi accesate de toți utilizatorii.

Aplicația mobilă Tripadvisor permite căutarea punctelor de interes din apropiere, consultarea recenziilor scrise de ceilalți utilizatori, dar și vizualizarea altor detalii precum fotografii și date de contact. Printre funcționalitățile aplicației se mai numără rezervarea camerelor de hotel, compararea prețurilor și verificarea disponibilității, planificarea unui itinerariu, salvarea locurilor favorite și obținerea direcțiilor de mers.

1.5 De ce „TakeMeOut”?

Aplicația poate fi considerată un „safe space” al locațiilor preferate și îmbină cele mai semnificative funcționalități regăsite în cele trei aplicații menționate anterior pentru a oferi o experiență completă în acest sens. Ideea acesteia nu se bazează pe a ajunge într-un punct oarecare prin toate căile posibile, ci mai degrabă pe a sugera activități noi utilizatorilor, prin a le reaminti de locurile pe care și-ar dori să le viziteze, fie că este vorba de o ieșire în oraș sau un city break. Aceasta este o funcționalitate ce nu se regăsește în niciuna dintre aplicațiile enumerate mai sus.

În același timp, posibilitatea de a vizualiza ruta de mers într-un mod cât mai natural cu ajutorul indicatoarelor plasate în spațiul real prin intermediul realității augmentate ușurează și îmbunătățește în mod considerabil procesul de orientare. Reiterând, direcțiile de mers sunt valabile numai pentru atracțiile situate într-o anumită rază față de locația utilizatorului, astfel încât să fie considerate în apropiere și să promoveze mersul pe jos în vederea ajungerii la destinație.

În plus, aplicația are o interfață relativ simplificată în comparație cu celelalte servicii similare, toată acțiunea desfășurându-se pe un singur dashboard intuitiv care se adaptează în funcție de nevoile utilizatorului.

1.6 Structura lucrării

Această lucrare elaborează aspecte legate de utilitatea, detaliile de implementare și funcționare ale aplicației „TakeMeOut”. Dacă prezentul capitol introductiv a descris utilitatea aplicației, următoarele capitole vor dezvolta celelalte aspecte.

Astfel, **capitolul 2** va aduce detalii privind tehnologiile folosite în realizarea aplicației. Atât partea de client (frontend), cât și cea de server (backend) sunt mai mult sau mai puțin implementate în limbajul **Swift**, împreună cu framework-urile **Core Data** și **Google Firebase** pentru persistarea datelor și autentificarea utilizatorilor. Tot în acest capitol se vor introduce și toate celelalte SDK⁴-uri și API⁵-uri folosite în dezvoltarea funcționalităților aplicației:

- **UIKit** pentru generarea, afișarea și coordonarea componentelor de UI⁶, introducerea de acțiuni asupra lor și gestionarea răspunsului la acțiunile utilizatorului;
- **MapKit** pentru afișarea hărții interactive, căutarea și generarea datelor despre locații, împreună cu rutele și instrucțiunile de orientare către acestea;
- **Core Location** pentru aflarea poziționării punctelor de interes, a rutelor către acestea, dar mai ales a utilizatorului, la care se adaugă controlul și buna gestionare a situațiilor în care locația acestuia se modifică;
- **ARKit și RealityKit** pentru adăugarea direcțiilor cu realitate augmentată;
- **UserNotifications** pentru trimiterea notificărilor către utilizator privind locațiile salvate din apropiere;
- **AVFoundation** pentru adăugarea direcțiilor audio sub formă de text rostit;
- **Cocoa Pods** pentru gestionarea pachetelor externe, precum **Floating Panel**;
- **Foursquare API** pentru obținerea de date și detalii auxiliare cu privire la locațiile căutate sau salvate.

Capitolul 3 aduce detalii privind arhitectura aplicației, corelând fiecare parte a acesteia cu funcționalitatea, respectiv componenta vizuală pe care o implementează. Toată acțiunea aplicației, exceptând partea de autentificare, are loc în dashboard-ul principal,

⁴SDK = Software Development Kit

⁵API = Application Program Interface

⁶UI = User Interface

ce are la bază un floating panel care își adaptează conținutul în funcție de acțiunea utilizatorului. Atât procesarea datelor și cererilor din partea utilizatorului, cât și afișarea lor au loc într-o componentă comună numită *ViewController* (*View* = frontend, *Controller* = backend), la care se adaugă clasele *Helper*, *Manager* pentru backend, respectiv *Model* și *View* pentru frontend.

În cel de-al **4-lea capitol** este prezentată, sub formă de demo bazat pe capturi de ecran, funcționalitatea aplicației din perspectiva utilizatorului. Fiecare componentă va fi expusă într-un subcapitol separat, urmând a se evidenția cazuri speciale și alte aspecte cheie.

Ultimul capitol vine ca o încheiere a acestei lucrări, în care se pune accentul pe experiența de dezvoltare, provocările care au apărut pe parcursul dezvoltării aplicației, dar și direcțiile în care aceasta poate fi îmbunătățită atât intern, cât și extern.

Capitolul 2

Tehnologii folosite

2.1 Swift și Xcode

Swift este un limbaj de programare compilat pentru sistemele de operare iOS, iPadOS, macOS, watchOS, tvOS și mai nou visionOS, dar și pentru aplicații Linux, lansat de Apple în anul 2014. Este open source și are rolul de a-și înlocui predecesorul, Objective-C, limbaj cu care este compatibil și suportă integrarea. Este preferat în rândul dezvoltatorilor pentru sintaxa curată și consistentă a codului ce introduce safeguards pentru prevenirea erorilor, dar și pentru viteza de execuție de care dă dovadă, fiind *de 2,6 ori mai rapid decât Objective-C și de 8,4 ori mai rapid decât Python*, conform Apple[21].

Împreună cu Xcode, IDE¹-ul de la Apple destinat dezvoltării de aplicații pentru dispozitive aparținând acestui ecosistem, experiența de iOS development devine flexibilă și permite separarea interfeței utilizatorului de restul codului care face aplicația să funcționeze[25]. De regulă, o aplicație iOS este alcătuită din 3 straturi[25]:

- Codul care asigură **funcționalitatea** aplicației;
- **Interfața** care poate fi stilizată vizual în Xcode;
- Codul care face legătura cu **componentele hardware** ale dispozitivului prin unul sau mai multe framework-uri de la Apple, pe care le vom acoperi în secțiunea 2.2.

Principalul dezavantaj al acestui limbaj de programare este suportul limitat la ecosistemul Apple, spre deosebire de alternative precum Flutter, Xamarin sau React Native care suportă platforme multiple.

Cu toate acestea, Swift rămâne principala opțiune pentru dezvoltarea aplicațiilor iOS: fiind un framework nativ, compilatorul contribuie la optimizarea performanței și la reducerea dimensiunii aplicațiilor[24], ceea ce îl face considerabil mai rapid față de celelalte framework-uri. În plus, Swift permite accesul la celelalte SDK-uri care asigură comunicarea cu componentele hardware ale dispozitivului, oferă extra suport pentru accesul

¹IDE = Integrated Development Environment

persoanelor cu dizabilități la funcționalitățile aplicațiilor și este foarte bine optimizat în dezvoltarea de componente UI reutilizabile și personalizabile.

2.2 Apple SDKs

Aproape orice aplicație iOS trebuie să comunice cu diverse componente hardware, precum camera sau ecranul[25]. Pentru a nu fi nevoie să scriem propriul cod care să asigure această comunicare, Apple pune la dispoziție o serie de SDK-uri și librării ce facilitează această sarcină.

2.2.1 UIKit

UIKit[22] oferă o multitudine de funcționalități pentru dezvoltarea aplicațiilor, inclusiv componente care pot fi folosite pentru a construi arhitectura de bază a aplicațiilor iOS/iPadOS și tvOS. Acest framework asigură arhitectura paginilor și componentelor în procesul de creare a interfeței vizuale, infrastructura de gestionare a evenimentelor și intrărilor single și multi-touch, fiind principala sursă de control a interacțiunilor dintre utilizator, sistem și aplicație.

UIKit oferă, de asemenea, suport pentru animații, documente, desenare și printare, gestionarea textului și afișajului, căutare, extensii ale aplicațiilor, administrarea resurselor și preluarea de informații privind dispozitivul curent, conform documentației oficiale Apple[22]. În plus, UIKit permite personalizarea suportului pentru accesibilitate și adaptarea interfeței vizuale în funcție de limbă, țară sau regiuni culturale.

În dezvoltarea aplicației „TakeMeOut” au fost folosite următoarele componente aparținând framework-ului UIKit:

- **UILabel** - cel mai comun mod de a afișa text drept componentă vizuală în interfață;
- **UITextField** - asemănător *UILabel*, aduce câteva funcționalități în plus, precum editarea textului (optional) și interpretarea sa: numerele de telefon și link-urile vor fi evidențiate, iar apăsând pe ele se va declanșa acțiunea specifică (inițierea unui apel/accesarea site-ului corespunzător);
- **UITextView** - reprezintă un câmp text în cadrul unui formular care poate răspunde la acțiunile utilizatorului prin adăugarea de acțiuni țintă (*target actions*): spre exemplu, validarea textului scris după ce utilizatorul a terminat de completat și afișarea unui mesaj de eroare dacă acesta este invalid;
- **UIButton** - este componenta de buton, căreia îl se poate atribui text și/sau o imagine și care răspunde la acțiuni asemănător câmpurilor text, prin adăugarea de target actions;

```
button.addTarget(self, action: #selector(buttonPressed), for:  
    .touchUpInside)
```

- **UIImage** și **UIImageView** - sunt componentele care afișează imagini în interfață, *UIImageView* fiind componenta principală ce are la bază un *UIImage*; la fel ca cele menționate anterior, și imaginile pot răspunde la acțiunile utilizatorului prin adăugarea unui *UITapGestureRecognizer*, care apelează o metodă când utilizatorul atinge imaginea respectivă, simulând astfel comportamentul unui buton;

```
let tapGesture = UITapGestureRecognizer(  
    target: self,  
    action: #selector(imageSelected))  
image.isUserInteractionEnabled = true  
image.addGestureRecognizer(tapGesture)
```

- **UIStackView** - este o componentă UI ce are rolul de a grupa alte subcomponente, în mod implicit vertical, asemănător unui *<div>* din HTML;
- **UIScrollView** - view-urile clasice (cu excepția *UITableView* și *UICollectionView*) nu sunt scrollable, ele trebuie să incorporeze într-un *UIScrollView* pentru a putea include mai mult conținut ce poate fi derulat;
- **UITableView**

UITableView se ocupă de reprezentarea informațiilor sub formă tabelară, implicit vertical, unde fiecare celulă a tabelului este de tipul **UITableViewCell**, o clasă care permite personalizarea prin extinderea ei. Această componentă UI este ceva mai complexă, întrucât necesită un *data source* din care să își extragă datele pe care să le atribuie fiecărei celule, astfel că o clasă care este data source pentru un *UITableView* trebuie să extindă protocolul *UITableViewDataSource*, unde sunt definite două metode ce atribuie tabelului un număr de rânduri, dar și datele corespunzătoare fiecărei celule din tabelă.

În plus, *UITableView* are nevoie de un *delegate*, la fel cum are nevoie de un data source. **Delegate** este un alt mod prin care Swift gestionează relația între componente, mai complexă și personalizabilă în comparație cu action target și *UITapGestureRecognizer*. Prin delegate, o componentă anunță delegatul său când un trigger are loc, iar delegatul are rolul de a controla acțiunea respectivă. Este personalizabil deoarece fiecare componentă își poate defini propriul protocol pentru delegatul său, în care fiecare metodă răspunde la o acțiune diferită. Spre exemplu, protocolul *UITableViewDelegate* cere gestionarea acțiunii în care utilizatorul selectează una dintre celulele tabelului.

- **UICollectionView** - asemănător *UITableView*, se ocupă de reprezentarea informațiilor sub forma unei colecții, implicit orizontal, unde fiecare obiect al colecției este de tipul **UICollectionViewCell**; la fel ca *UITableView*, această componentă are nevoie de un data source și un delegate;
- **UISearchBar** - este componenta vizuală corespunzătoare unei bare de căutări, asemănătoare unui *UITextField*, dar care aduce funcționalități în plus precum controlul tastaturii, posibilitatea afișării unui buton de cancel și controlul altor componente din exterior care pot fi influențate de activarea sau dezactivarea barei de căutare; datorită complexității interacțiunilor, acest obiect are nevoie de o clasă delegate;
- **UIImagePickerController** - este o componentă UI folosită pentru selectarea sau editarea de imagini și are două opțiuni de selectare a imaginilor: prin realizarea de poze instantanee, caz în care va prezenta un view al camerei și prin alegerea de poze din biblioteca utilizatorului, caz în care aceasta va fi prezentată; conformarea la protocolul delegate al acestei componente presupune definirea comportamentului prin care este selectată o anumită imagine.

UIKit permite și extinderea componentelor deja existente, prin adăugarea sau modificarea anumitor proprietăți și configurații. Spre exemplu, view-urile personalizate *CustomButton* și *DetailTableViewCell* extind clasele *UIButton* și *UITableViewCell* pentru a le adăuga noi proprietăți care pot fi refolosite în mai multe părți ale aplicației, evitând rescrierea codului.

2.2.2 MapKit

MapKit este un framework dedicat dezvoltatorilor care doresc să încorporeze serviciile de mapping puse la dispoziție de Apple. Acesta este folosit atât ca un instrument de afișare a hărții universale ce cuprinde în mod implicit toate informațiile necesare corespunzătoare locațiilor, cât și ca API prin care se pot obține detalii ale locațiilor precum adresă și rute de direcționare, dar și ca mecanism de control al interacțiunilor utilizatorului cu harta, punctele de interes și trăsăturile geografice, conform documentației oficiale Apple[14].

MapKit pune la dispoziție o serie de componente principale care au fost folosite în dezvoltarea aplicației „TakeMeOut” pentru a asigura o experiență cât mai intuitivă de navigare pe hartă, căutare de locații și generare de direcții de mers. Acestea sunt:

- **MKMapView**

MKMapView este o interfață de hartă încorporabilă, similară celei pe care aplicația *Hărți* de la Apple o pune la dispoziție[15]. Este folosit pentru a afișa informații ale hărții universale și pentru a manipula conținutul său prin intermediul aplicației. *MKMapView* suportă 3 stiluri de afișare diferite: *standard* - ce permite o vizualizare îmbogățită a

conținutului în 2D sau 3D, *hybrid* - ce permite vizualizarea hibrid din satelit a hărții și *imagery-based* - ce permite vizualizarea pe bază de imagini a hărții.

În plus, *MKMapView* suportă interacțiunile standard precum schimbarea poziției și a nivelului de zoom, atât prin configurarea lor folosind metoda *setRegion()* care primește ca parametru un obiect de tip *MKCoordinateRegion* ce are la bază o coordonată și un span², dar și prin gesturi tactile de tip flick - scroll și pinch - zoom in and out. Există suport și pentru o varietate de adnotări folosind, în exemplul nostru, *MKPointAnnotation* și *MKPolyline*.

- **MKMapItem** - ține evidența unui punct de interes de pe hartă și include informații semnificative precum adresa completă și locația sa în sistemul de coordonate sub forma unui obiect de tip *MKPlacemark*, dar și alte detalii cum ar fi denumirea, categoria (muzeu, cafenea, parc etc.), număr de telefon pentru contact și URL către website unde este cazul;
- **MKPointAnnotation** - este un tip de overlay folosit de MapKit pentru a indica un anumit punct de pe hartă; acesta primește ca parametru o coordonată pe care să o fixeze și un nume (optional), reprezentarea sa vizuală fiind sub forma unui pin căruia î se poate atașa un titlu;
- **MKUserLocation** - este o simplă adnotare care indică locația curentă a utilizatorului pe hartă;
- **MKLocalSearch**

MKLocalSearch inițiază căutarea de locații (prin apelul metodei *start*) pe hartă pornind de la un request ce primește ca intrare, în cazul nostru, o denumire text și o regiune sub tipul *MKCoordinateRegion* și returnează o listă de *MKMapItems* care se potrivesc descrierii date și se află cât mai aproape de regiunea stabilită, mergând în afara span-ului setat dacă nicio locație din acesta nu se potrivește cu denumirea dată.

Pentru a returna rezultatele așteptate, *MKLocalSearch* folosește o combinație de geocoding, adică translatarea unei adrese în coordonate geografice, procesare de limbaj natural pentru procesarea cererilor și o varietate de algoritmi de căutare și indexare, dar și algoritmi de machine learning pentru îmbunătățirea acurateței și relevanței rezultatelor.

- **MKDrections**

Acest obiect, la fel ca *MKLocalSearch*, inițiază căutarea de rute (prin apelul metodei *calculate*) pornind de la un request ce primește ca intrare două locații, sursă și destinație, ambele de tipul *MKMapItem*, împreună cu alte detalii precum tipul transportului care, în cazul nostru, este pedestrian dar și un atribut boolean care indică dacă răspunsul poate conține mai multe rute alternative, în cazul nostru având valoarea *true*.

²un span definește ce procentaj din hartă este vizibil

Răspunsul este o listă de rute de tipul *MKRoute* și asigură cele mai scurte și sigure rute existente prin folosirea unor algoritmi foarte bine optimizați de găsire a unui drum minim între două noduri ale unui graf, luând în calcul conexiunile drumurilor, restricțiile și condițiile de trafic plus mulți alți factori.

- **MKRoute**

Este un obiect returnat exclusiv ca răspuns al unui request de tip *MKDrections* și definește geometria rutei[17] dintre un punct sursă și un punct destinație, sub forma unor segmente liniare (sau curbe) numite *polylines*.

O rută este împărțită în mai mulți pași, sub forma unui obiect de tip *MKRoute.Step*. Acest obiect corespunde unei singure instrucțiuni ce trebuie urmată de-a lungul navigării între două puncte și conține propriul polyline între pasul curent și pasul următor, informații relevante precum instrucțiunea propriu-zisă (ex. *În 200 de metri virează la stânga.*), distanța până la următoarea instrucțiune, mijocul de transport și sfaturi adiționale de siguranță, dacă este cazul.

Același obiect conține, de regulă, și alte detalii ale rutei precum numele acesteia, distanța totală și timpul de călătorie estimat.

- **MKPolyline** - definește geometria unei rute între două puncte oarecare, fiind interpolat cu alte puncte ce definesc orice viraj, oricât de mic, astfel încât oricare două puncte consecutive sunt pe o linie dreaptă; un polyline are ca date suplimentare și distanța totală, ce poate fi calculată de asemenea foarte ușor aplicând distanța euclidiană între fiecare pereche de puncte consecutive;
- **MKMapCamera** - este un obiect ce definește și afectează felul în care este prezentată harta către utilizator, prin plasarea unui viewpoint virtual deasupra hărții[16]; cu ajutorul camerei putem specifica locația acesteia pe hartă, direcția în care să indice busola, ce procentaj din hartă este vizibil prin cameră, adică lungimea perpendiculariei dintre cameră și planul hărții.

2.2.3 Core Location

Core Location[13] asigură servicii de determinare a localizării geografice, altitudinii, orientării sau poziționării relative la cel mai apropiat dispozitiv *iBeacon* a unui dispozitiv, conform documentației oficiale Apple. SDK-ul adună datele folosind toate componentele semnificative ale telefonului, de la Wi-Fi, GPS, Bluetooth până la magnetometru, barometru și hardware celular.

Folosirea serviciilor din *Core Location* se realizează prin instantierea unui obiect de tip *CLLocationManager* ce are rolul de a configura, porni, administra și opri serviciile de localizare. Funcționalitățile managerului pot fi accesate doar de către o clasă delegate ce

extinde protocolul *CLLocationManagerDelegate* către care să raporteze orice schimbare care poate apărea în locația curentă a utilizatorului pentru ca delegatul să o poată gestiona corespunzător.

Printre capacitatele managerului se numără:

- **Actualizarea locației** - măsoară cu un grad semnificativ de acuratețe orice schimbare în locația curentă a utilizatorului; orice clasă delegate trebuie neapărat să implementeze metoda care gestionează aceste schimbări;
- **Monitorizarea regiunilor** - detectează când utilizatorul intră sau ieșe dintr-o regiune de interes ce trebuie definită anterior și generează un eveniment care trebuie mai departe gestionat prin metoda delegate corespunzătoare;
- **Detectarea și localizarea beacon-urilor din apropiere**;
- **Orientarea compasului** - raportează orice schimbare în orientarea compasului și generează un eveniment care poate fi gestionat prin metoda delegate corespunzătoare.

Pentru a putea folosi serviciile de localizare, aplicația cere autorizare, iar sistemul îndeamnă utilizatorul să accepte sau să respingă această cerere.

2.2.4 ARKit și RealityKit

Realitatea augmentată (*AR = Augmented Reality*) descrie experiențe la nivelul utilizatorilor prin care obiecte 2D sau 3D sunt adăugate la lumea reală surprinsă prin camera dispozitivului într-un mod în care aceste elemente oferă impresia de apartenență la lumea reală. **ARKit** combină monitorizarea mișcării dispozitivului, captarea scenelor folosind camera, procesarea avansată a scenelor, avantajele display-ului și senzorul LiDAR, unde este cazul, pentru a simplifica construcția unei experiențe AR, conform documentației oficiale Apple[4]. Experiențele AR pot fi create folosind atât camera de pe spate a dispozitivului, cât și cea din față, cu toate că aplicația noastră folosește exclusiv camera de pe spate.

RealityKit este folosit pentru a implementa simulare și interpretare 3D de înaltă performanță[18]. Framework-ul folosește în mare măsură informațiile furnizate de ARKit pentru integra cu ușurință obiecte virtuale în lumea reală.

Prin combinația celor două SDK-uri, am făcut posibilă interpretarea direcțiilor de mers și plasarea în spațiul real surprins cu camera de pe spatele dispozitivului a unor obiecte de tip ancoră care să proiecteze aceeași direcții ca și cum ar fi niște indicatoare în lumea reală.

Din toată suita de funcționalități pe care ARKit și RealityKit le pune la dispoziție oricărui dezvoltator, am dus această sarcină la final folosind o serie de componente aduse împreună prin diverse computații care vor fi elaborate în următorul capitol. Acestea sunt:

- **ARView**

ARView este o componentă vizuală care permite afișarea de experiențe AR folosind RealityKit, conform documentației oficiale Apple[20]. *ARView* folosește output-ul camerei din configurația acestuia pentru a crea o scenă ce permite amplasarea graficelor 3D interpretate care sunt adăugate sub forma unor ancore (*Anchor Entity*).

De asemenea, *ARView* poate fi folosit atât pentru a configura opțiunile de interpretare, trăsăturile mediului care face obiectul scenei și modul camerei, cât și pentru a gestiona interacțiunile utilizatorului sub formă de input tactil sau de la tastatură, a găsi entități plasate la un anumit punct și a accesa statistici și vizualizări care ajută la debugging-ul aplicației.

- **ARSession**

În mod implicit, pe lângă obiectul scenă, un *ARView* rulează într-o sesiune care poate fi pornită, oprită și configurată. *ARSession* este obiectul care controlează sarcinile majore asociate cu fiecare experiență AR, precum preluarea datelor ce țin de monitorizarea mișcării dispozitivului, controlul și redarea în timp real a imaginii camerei și analiza imaginii surprinse de cameră[7]. Toate aceste rezultate sunt procesate pentru a crea o legătură între spațiul lumii reale surprins de dispozitiv și spațiul virtual în care este plasat conținutul AR.

Pentru a putea rula o sesiune, acesteia îi trebuie atribuită întâi o configurație. *ARConfiguration* este clasa de bază pentru diferite opțiuni care au același rol, de a stabili legătura între spațiul real și cel virtual[5]. Pentru a obține imagini în direct de la cameră, ARKit se ocupă în spate de pipeline-ul de capturare a imaginii. În funcție de configurația aleasă, determină camerele care captează imagini și care flux de la cameră va fi afișat de aplicație.

Aplicațiile ce folosesc realitatea augmentată recunosc regiunile de interes din lumea reală. La rulare, ARKit generează un *ARAnchor* pentru obiectul pe care îl recunoaște, ceea ce permite aplicației să ia ca referință detalii precum dimensiune și locație. Configurația aleasă determină tipul obiectelor din lumea reală pe care ARKit le recunoaște și le pune la dispoziție aplicației.

ARKit pune la dispoziție 4 tipuri de configurații: *ARPositionalTrackingConfiguration*, *AROrientationTrackingConfiguration*, *ARWorldTrackingConfiguration* și *ARGeoTrackingConfiguration*.

ARPositionalTrackingConfiguration este folosit în principal în scenariile de realitate virtuală întrucât captează dispozitivul cu 6 grade de libertate - 3 grade de rotație (roll, pitch, yaw) și 3 grade de translație (x, y, z) - prin rularea camerei la cea mai mică rezoluție și rată a cadrelor posibile[6].

AROrientationTrackingConfiguration captează dispozitivul doar cu 3 grade de libertate și anume cele 3 grade de rotație, astfel încât iluzia oferită de realitatea augmen-

tată persistă la rotirea dispozitivului, dar este compromisă atunci când dispozitivul își schimbă poziția.

ARWorldTrackingConfiguration urmărește mișcarea dispozitivului cu cele 6 grade de libertate. Acest tip de tracking creează experiențe AR imersive, astfel că un obiect virtual pare că rămâne în același loc relativ la lumea reală, chiar dacă utilizatorul înclină dispozitivul pentru a se uita deasupra obiectului sau sub acesta, sau îl mișcă în împrejur pentru a observa obiectul din mai multe perspective[8]. O sesiune de world-tracking pune la dispoziție mai multe metode ca aplicația să recunoască și să interacționeze cu elementele din spațiul real, fie că este vorba de detectarea planelor orizontal și vertical, detectia de imagini, obiecte sau a poziției 3D a trăsăturilor din lumea reală ce corespund unui punct atins pe ecranul dispozitivului folosind raycasting. Această configurație este folosită în dezvoltarea experienței AR a aplicației „TakeMeOut”.

ARGeoTrackingConfiguration este cea mai avansată configurație, ce permite amplasarea de ancore geografice care sunt recunoscute de ARKit prin corelarea acestora în mod direct cu spațiul real, prin diversi algoritmi de machine learning antrenați să recunoască împrejurimile din jurul ancorelor, pe care le compară cu imaginile surprinse în direct de camera dispozitivului. Această configurație pare cea mai potrivită pentru aplicația noastră. Cu toate acestea, având în vedere necesitatea unor date actualizate sub formă de imagini ce surprind punctele de reper din împrejurimi (clădiri, fațade, semne de circulație, indicatoare etc.), pe care ARKit le descarcă pentru a corela cu coordonatele GPS specifice ancorelor geografice, *ARGeoTrackingConfiguration* este disponibil într-un număr redus de locații, în principal în SUA și câteva orașe mari din restul lumii, fiind de asemenea restricționat la străzile mai semnificative.

- **Anchor Entity**

Anchor Entities sunt principalele componente ale scenei din *ARView*. Ele sunt folosite pentru a controla modul în care RealityKit amplasează obiectele virtuale în scenă. Conform documentației oficiale Apple[19], RealityKit amplasează ancorele pe baza proprietății *target*. Spre exemplu, putem configura o astfel de entitate să stea pe o suprafață orizontală detectată sau la o anumită poziție în lumea virtuală, cum se întâmplă în aplicația noastră.

- **Model Entity**

Un *Anchor Entity* este alcătuit din unul sau mai multe **Model Entities**. Acestea conțin reprezentarea geometrică a obiectelor ce urmează a fi atașate ancorei și adăugate în scenă. Un *Model Entity* poate lua orice formă, de la o formă geometrică simplă (ex. cerc) până la modele mai complexe (ex. mașină). Pentru a controla felul în care un obiect apare în lumea virtuală, trebuie specificate mesh-ul (matrița) și materialele. Mai sunt detalii ce merită menționate, însă doar obiectele geometrice de tip sferă fac obiectul acestei lucrări.

2.2.5 User Notifications

Aplicația folosește **User Notifications**[23] pentru trimitera de notificări locale către useri. Acestea primesc un conținut sub formă de *UNMutableNotificationContent* și inițiază trimitera notificării în momentul în care se atinge un factor declanșator: în cazul nostru, atunci când o locație salvată se află în imediata apropiere a utilizatorului. Acest lucru este posibil folosind următoarele clase ale framework-ului:

- **UNUserNotificationCenter** - este obiectul central care gestionează activitățile ce presupun trimitera de notificări în cadrul aplicației și se ocupă, în cazul nostru, de cererea permisiunilor din partea utilizatorului și lansarea efectivă a notificărilor atunci când este declanșat trigger-ul;
- **UNNotificationRequest** - este obiectul care se ocupă de programarea unei notificări ce primește conținutul acesteia cuprinzând un titlu și un body.

2.2.6 AVFoundation

AVFoundation[9] (*Audio-Video Foundation*) este printre primele SDK-uri puse la dispoziție de Apple și are ca scop prelucrarea asset-urilor audiovizuale, controlul camerelor, procesarea audio și configurarea interacțiunilor audio.

„TakeMeOut” folosește AVFoundation exclusiv pentru generarea și redarea direcțiilor de mers sub formă de text rostit, pentru care folosește componenta *AVSpeechSynthesizer* prin apeluri ale metodei *speak()*, ce primește ca parametru un *AVSpeechUtterance* reprezentat de un string pe care synthesizer-ul îl va rosti.

2.3 Framework-uri

Pe lângă SDK-urile Apple ce asigură majoritatea funcționalităților aplicației atât pe partea de client, cât și ce pe cea de server, adresez această secțiune framework-urilor de backend care fac posibilă cu mare ușurință administrarea și autentificarea utilizatorilor, precum și persistarea informațiilor relevante.

2.3.1 Firebase

Firebase este framework de tip *Backend-as-a-Service* (BaaS) lansat de Google în anul 2012, ce cuprinde o serie de servicii de cloud computing în backend. Firebase face parte din categoria bazelor de date *NoSQL*, ce rețin datele sub formă de documente JSON. Printe serviciile pe care le pune la dispoziție se numără gestionarea și autentificarea utilizatorilor, administrarea unei baze de date în cloud în timp real, hosting pentru aplicațiile web, testarea aplicațiilor și trimitera de notificări la distanță.

Acstea funcționalități sunt disponibile prin crearea unui proiect pe platformă, care trebuie să înregistreze bundle ID-ul proiectului deja existent în Xcode pentru autorizarea request-urilor. Odată autorizat, proiectul generează un API key care poate fi folosit în cadrul proiectului principal pentru a folosi serviciile Firebase. Firebase se integrează cu Swift prin instalarea unor pachete adiționale și introducerea lor în spațiul de lucru.

Din multitudinea de servicii propuse de acest framework, doar partea de autentificare și control al utilizatorilor face obiectul acestei lucrări.

- **Autentificarea utilizatorilor**

În urma activării serviciului de Autentificare, Firebase creează în mod automat un tabel de utilizatori ce păstrează informațiile de bază: adresa de email, numărul de telefon³, parola criptată, modalitatea de autentificare, data creării, data ultimei autentificări și un identificator unic (*User UID*).

Auth.auth() este obiectul care controlează autentificarea. Înregistrarea se face apelând metoda *createUser()* ce primește ca parametri, în cazul autentificării cu email, două string-uri reprezentând adresa de email și parola, iar login-ul se realizează apelând metoda *signIn()* care primește aceiași parametri. Pentru validări optionale, se pot verifica metodele de autentificare asociate unei adrese de email apelând metoda *fetchSignInMethods()*. Cel mai des folosit atribut al acestei clase este însă *currentUser* care returnează obiectul *User* reprezentând utilizatorul conectat în sesiunea curentă sau *nil* dacă nu este niciun utilizator conectat.

Firebase suportă autentificare prin furnizori nativi (adresă de email și număr de telefon), furnizori 3rd party (Google, Apple, Facebook, Microsoft, GitHub etc.) și furnizori personalizați (OpenID Connect, SAML). Pe lângă email și parolă, „TakeMeOut” implementează și autentificarea cu Google care folosește *GoogleAuthProvider* pentru a genera un credential pe baza contului Google asociat unui utilizator. Acest credential este folosit mai departe pentru a crea și autentifica un nou utilizator ce are implicit adresa de email asociată contului Google și un token unic în loc de parolă.

2.3.2 Core Data

Core Data este un object graph și persistence framework[26] lansat de Apple pentru sistemele de operare macOS în 2005 și mai târziu, în 2009, pentru iOS. Conform documentației oficiale Apple[10], este folosit pentru a memora datele permanente ale aplicației cu scopul folosirii lor în offline, dar și pentru caching-ul datelor temporare. Spre deosebire de Firebase, Core Data este un framework *SQLite*, ce permite organizarea datelor după modelul relațional entity-attribute.

Pentru a folosi Core Data, proiectului i se adaugă fișierul *DBModel* care permite vizualizarea în Xcode și adăugarea de entități și attribute, respectiv a relațiilor dintre ele,

³Adresa de email și numărul de telefon sunt memorate în funcție de modalitatea de autentificare.

urmând ca IDE-ul să genereze modele specifice fiecărei entități. Structura bazei de date utilizată în acest proiect este detaliată în următorul capitol.

Pentru obținerea, introducerea și modificarea datelor din Core Data au fost folosite următoarele obiecte:

- **NSManagedObjectContext**

Este contextul bazei de date, ce conține un grup de obiecte model în relație, reprezentând o stare coerentă intern a uneia sau mai multor depozite de date[12]. Modificările aduse obiectelor din context rămân în memoria contextului până când acestea sunt salvate într-unul din depozitele Core Data prin apelul metodei *save()*. Din obiectele aparținând unei contexte se pot extrage date prin apelul metodei *fetch()* și șterge date prin apelul metodei *delete()*.

Contextul este accesibil prin instanțierea unei container persistent (*NSPersistentContainer*) reprezentând modelul bazei de date.

- **NSFetchRequest**

Este instanță care colectează criteriile necesare extragerii - și optional, sortării - unei grupuri de obiecte administrate dintr-un depozit de date persistent[11]. Un request conține de regulă numele entității în care face căutarea, un predicat care are rolul de a filtra datele și optional un sir de descriptori care specifică ordinea în care să fie sortate datele.

2.4 Altele

Această secțiune este dedicată altor librării ce au un rol semnificativ în asigurarea funcționalității aplicației.

2.4.1 Cocoa Pods - Floating Panel

Floating Panel[1] este o componentă UI simplă și ușor de folosit, scrisă în Swift 4.2 și compatibilă cu versiuni mai noi de iOS și iPadOS. Floating Panel este disponibil dezvoltatorilor prin **Cocoa Pods**, un manager de dependențe pentru proiecte în Swift și Objective-C.

Printre avantajele sale se numără simplitatea, fiind un view controller de tip container, controlul gesturilor și animațiilor, abilitatea de a derula conținutul din interiorul panoului și personalizarea comportamentului și layout-ului care permite mai multe stări de ancorare (tip, half, full).

„TakeMeOut” folosește în mare măsură Floating Panel pentru afișarea tuturor informațiilor și funcționalităților disponibile, adaptând conținutul său în funcție de acțiunile utilizatorului într-un mod foarte intuitiv și fluid, prin apelarea metodei *set()* care primește ca parametru un view controller al cărui view îl afișează.

Panoul poate comuta foarte ușor între stări prin apelarea metodei *move()*, care îl mută din starea de ancorare curentă în cea dată ca parametru metodei. În plus, acțiunile asupra panel-ului pot fi gestionate prin delegatul său care trebuie să respecte protocolul *FloatingPanelControllerDelegate*.

2.4.2 Foursquare Places API

Places API[2] este serviciul pus la dispoziție de Foursquare prin care se pot căuta puncte de interes din toată lumea și extrage informații suplimentare despre acestea. Prin crearea unui cont și a unui proiect pe platformă, dezvoltatorilor li se atribuie un API key prin care au acces la endpoint-urile specifice.

„TakeMeOut” folosește date din Places API pentru a îmbogăți secțiunea de detalii pentru fiecare locație, întrucât MapKit, prin intermediul componentei *MKMapItem*, furnizează un număr redus de detalii ce se rezumă la nume, adresă și categorie, optional website și număr de telefon pentru contact. Cu Places API, folosim inițial endpoint-ul *getPlaceId()*, pentru a face matching între numele și coordonatele deja obținute din MapKit și locația corespunzătoare din baza de date Foursquare. Cu acest id putem mai departe să apelăm endpoint-ul *getPlaceDetails()*, din care extragem descrierea, informațiile de contact (website, număr de telefon), programul de funcționare, o serie de imagini, rating-ul și rata de popularitate.

Capitolul 3

Arhitectura și funcționalitatea aplicației

Aplicația respectă o arhitectură **MVC** (Model-View-Controller), unul dintre cele mai populare pattern-uri folosite în dezvoltarea de software. Acesta, după cum indică și denumirea, are la bază 3 componente principale: **Model**, **View** și **ViewController**.

Dată fiind această structură a aplicației, nu există un proiect separat care se ocupă de sarcinile de backend, respectiv un proiect care configurează partea de frontend, ambele fiind gestionate în cadrul aceluiași proiect realizat în Swift.

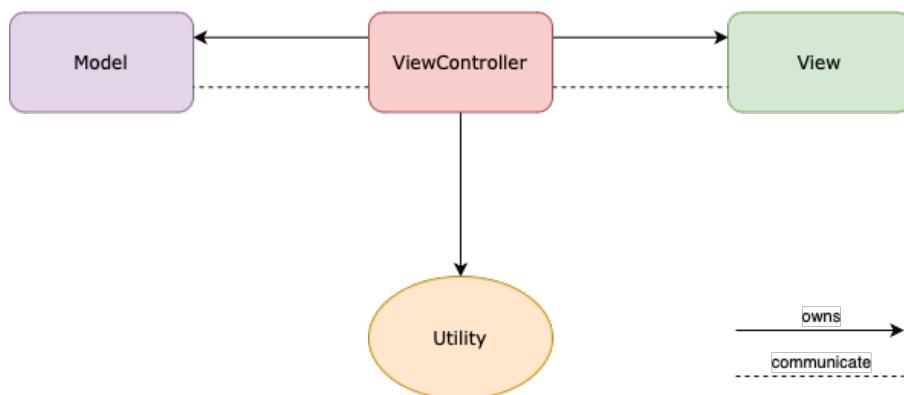


Figura 3.1: Arhitectura MVC

3.1 Baza de date

După cum am elaborat în capitolul precedent, baza de date este o combinație între serviciile oferite de framework-urile Firebase și Core Data. Astfel, Firebase reține tabela principală de useri în care sunt reținute informațiile de bază, urmând ca Core Data să țină date adiționale despre useri (nume complet, adresă de email, fotografie de profil) în propria tabelă *User*.

Pe lângă informațiile despre utilizatori, tabela *Location* reține informații despre locațiile salvate de utilizatori: denumire, coordonată (latitudine și longitudine), categorie și adresă.

Între cele două tabele din Core Data există o relație de tip *one-to-many*, astfel încât un utilizator poate salva mai multe puncte de interes, iar o locație poate fi salvată de un singur utilizator. Nu a fost implementată o relație *many-to-many* între cele două entități întrucât ne interesează strict lista de locații salvate de un user, nu și lista de useri care au salvat o locație. În modelul nostru, relația introduce câmpul adițional *savedLocations* în tabela *User*, reprezentând lista locațiilor salvate de utilizator, și câmpul *savedBy* în tabela *Location*, reprezentând utilizatorul care a salvat locația respectivă.

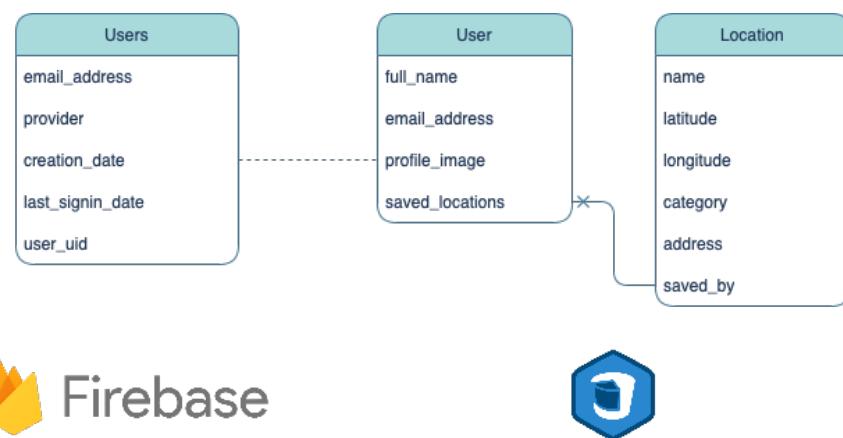


Figura 3.2: Baza de date

3.2 Models

Modelele au rolul de a defini informațiile afișate în cadrul aplicației. În cazul nostru a fost nevoie de două astfel de componente:

- **FavouriteModel** - această clasă modelează o locație extrasă din entitatea *Location* din baza de date și conține un obiect de tip *MKMapItem* creat doar cu numele și coordonata locației; deoarece acest tip de date este în general returnat de MapKit, nu permite crearea unui obiect identic folosind toate atributele și în schimb reținem în model adresa și categoria separat, păstrând totuși map item-ul pentru a-l putea folosi mai departe în computații;
- **LocationDetailsModel** - această clasă modelează informațiile suplimentare ale unei locații, extrase cu ajutorul Foursquare Places API, astfel că acesta conține descrierea, programul de funcționare, un atribut boolean care specifică dacă locația este deschisă în momentul curent, un sir de imagini, rating-ul, rata de popularitate, numărul de telefon și URL-ul către website.

3.3 Views

View-urile au rolul de a defini modul în care informațiile sunt afișate în cadrul aplicației. Pe lângă view-urile specifice view controllerelor, aplicația „TakeMeOut” mai definește câteva view-uri personalizate.

- **LocationTableViewCell** - este un view responsabil de structura și layout-ul unei celule aparținând tabelului de locații salvate de user și conține o iconă reprezentativă pentru categoria locației, titlul locației, distanța față de user și adresa;
- **RouteTableViewCell** - este un view responsabil de structura și layout-ul unei celule aparținând tabelului de rute disponibile între user și o locație, conținând lungimea totală a rutei ca distanță și timp, împreună cu un buton care lansează sesiunea de orientare către locația respectivă;
- **DetailTableViewCell** - este un view responsabil de structura și layout-ul unei celule aparținând tabelului de detalii suplimentare ale unei locații, conținând doar un label ce precizează tipul informației și încă un label pentru informația propriu-zisă;
- **CustomButton** - este un layout de buton reutilizabil ce afișează titlul acțiunii împreună cu o iconă reprezentativă pentru acțiunea respectivă și primește ca parametri titlul acțiunii, numele care indică iconă folosită și culoarea butonului.

3.4 View Controllers

View controller-ele sunt componentele principale ale aplicației și cuprind mare parte din logica și afișarea acesteia. După cum sugerează numele, aici este construit view-ul reprezentat prin adăugarea elementelor de UI și definirea constrângerilor de dimensionare și poziționare între ele, elemente care mai apoi sunt populate cu date fie în cadrul aceluiași view controller, fie în alte view controller delegate. Configurarea layout-ului unui view controller, precum și a altor funcționalități se realizează în cadrul metodelor standard *viewDidLoad()*, *viewDidLayoutSubviews()*, *viewWillAppear()* și *viewDidAppear()* care se apelează în această ordine la invocarea acestuia.

Tot aici sunt implementate extensiile specifice protocoalelor delegate, dacă este cazul, și alte metode de gestionare a interacțiunilor utilizatorilor cu propriile elemente de UI.

Aplicația folosește un *UINavigationController* pentru a comuta între view controller prin metoda *pushViewController()*. Aceasta este folosit, în cazul nostru, pentru a naviga între view controller-ul principal și cele două pentru autentificare (login și register), comutarea între restul view controllerelor realizându-se prin schimbarea conținutul afișat de floating panel.

View controllerele ce constituie aplicația noastră sunt:

- **LoginViewController** - definește mecanismul de afișare și control al paginii de autentificare și conține câmpurile necesare, metodele target prin care sunt gestionate interacțiunile cu acestea și un buton către pagina de înregistrare; prin apăsarea butonului de sign in, este apelată metoda *loginUser()* din extensia aceleiași clase ce se găsește în helper-ul de autentificare;
- **RegisterViewController** - definește mecanismul de afișare și control al paginii de înregistrare și conține câmpurile necesare, metodele target prin care sunt gestionate interacțiunile cu acestea, label-uri care indică în timp real dacă un câmp este completat invalid și un buton către pagina de autentificare; prin apăsarea butonului de register, este apelată metoda *createUser()* din extensia aceleiași clase ce se găsește în helper-ul de autentificare;
- **MainViewController**

MainViewController este baza aplicației, de altfel singura pagină cu care utilizatorul interacționează dacă este autentificat, în caz contrar fiind redirectionat la pagina de autentificare, respectiv înregistrare. Acest view controller are 3 componente principale care se integrează foarte intuitiv și se adaptează în funcție de acțiunile utilizatorului: bara principală, harta și panoul adaptiv.

Bara principală este afișată în partea de sus a paginii și conține numele aplicației ca titlu informativ, un buton sub formă de săgeată care are rolul de a actualiza poziția utilizatorului pe hartă și a o recentra și butonul cu imaginea de profil - sau o pictogramă sugestivă, în cazul în care utilizatorul nu a configurat o imagine de profil - care afișează secțiunea de detalii ale profilului. În timpul unei sesiuni de orientare a utilizatorului către o locație, această bară este înlocuită de una extinsă ce are rolul de a fișa instrucțiunile direcțiilor de mers.

Harta este un *MapView* cu centrul în locația curentă a utilizatorului ce permite vizualizarea fără derulare a împrejurimilor pe o rază de aproximativ 500 de metri. Harta se adaptează la acțiunea curentă: când utilizatorul a selectat o locație, harta își schimbă centrul în coordonatele locației respective, iar atunci când se generează rutile dintre 2 puncte, harta este redimensionată astfel încât să cuprindă cele două puncte și rutile dintre ele.

În modul de afișare a direcțiilor de mers, hărții i se aplică o cameră care își actualizează nordul în funcție de direcția de mers și este redimensionată astfel încât să aibă în prim plan punctul reprezentând locația utilizatorului, progresul acestuia și instrucțiunile de mers. Tot în acest mod, prin selectarea opțiunii de vizualizare în realitate augmentată, harta este înlocuită de *ARView*. Se poate comuta foarte ușor între cele două moduri prin apăsarea unui buton.

Panoul adaptiv este un floating panel în care sunt afișate majoritatea informațiilor și controalelor din aplicație. Acesta își schimbă conținutul în funcție de acțiunile utilizatorului și comută între cele 3 stări de ancorare în funcție de tipul conținutului, dar și gesturi: poate fi tras în sus pentru a afișa mai mult conținut sau în jos pentru a ascunde din informații și a putea vedea mai mult din hartă.

View controller-ul principal implementează și o mulțime de **protocole delegate**: *CLLocationManagerDelegate*, *ARSessionDelegate*, *SearchViewControllerDelegate*, *MKMapViewDelegate* și *FloatingPanelControllerDelegate*.

Dintre acestea, cea mai importantă este pe deosebire implementarea protocolului delegat pentru *CLLocationManager* întrucât MainViewController este responsabil de tratarea tuturor cazurilor de modificare a locației și orientării utilizatorului sau intrare, respectiv ieșire dintr-o regiune circulară. Vom reveni la implementarea acestui protocol în secțiunile care dezvoltă implementarea funcționalităților cheie.

În *MKMapViewDelegate* este definit modul în care sunt afișate rutele dintre 2 puncte din perspectiva aspectului polyline-urilor. În *FloatingPanelControllerDelegate* este controlat comportamentul panel-ului atunci când acesta își modifică starea de ancorare.

SearchViewControllerDelegate este un protocol personalizat prin care MainViewController decide ce se întâmplă cu un rezultat al căutării de locații atunci când acesta este selectat. Astfel, în momentul în care o locație este selectată, conținutul panel-ului se schimbă astfel încât să afișeze detaliile locației, iar pe hartă este atașat un pin la coordonata locației cu denumirea acesteia, coordonată care devine și centrul hărții.

- **FavouritesViewController**

View controller-ul destinat afișării de locații salvate de către utilizator este unul dintre tipurile de content din panoul adaptiv și controlează un tabel care încarcă aceste locații. Punctele de interes sunt preluate din baza de date prin apelarea metodei *loadFavourites()* la fiecare încărcare a acestui view, întrucât locații pot fi adăugate sau șterse între timp.

Prin aderarea la protocolul *UITableViewDataSource*, controller-ul specifică numărul de rânduri al tabelului și conținutul celulelor. Pentru layout-ul celulelor din tabel este folosit view-ul personalizat *LocationTableViewCell* care este populat cu datele din sirul de modele *FavouriteModel*, care este întâi sortat crescător după distanța de la locația curentă a user-ului, astfel că locațiile cele mai apropiate vor apărea primele în listă.

- **SearchViewController**

SearchViewController este responsabil de căutarea locațiilor folosind cuvinte cheie, denumiri sau adrese. Acesta este alcătuit dintr-o bară de căutare de tip *UISearchBar* și un tabel care afișează rezultatele căutării.

Acesta este conținutul afișat în mod implicit în floating panel la deschiderea aplicației. Dacă nu se exercită nicio acțiune asupra barei de căutare, controller-ul aduce în view FavouritesViewController, iar în momentul în care se acționează asupra acesteia, tabelul locațiilor favorite este înlocuit de tabelul care va afișa rezultatele căutării pe moment ce utilizatorul tastează. Acest lucru este posibil prin implementarea metodei *searchBar()* care se apelează la fiecare modificare adusă textului din bara de căutare. În cadrul acestei metode, se realizează un request de tip *MKLocalSearch* care primește ca parametri textul curent din search bar și regiunea curentă, urmând să actualizeze celulele afișate în tabelul de rezultate.

```
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) {  
    if !searchText.isEmpty {  
        let request = MKLocalSearch.Request()  
        request.naturalLanguageQuery = searchText  
        request.region = (owner?.mapView.region)!  
  
        let search = MKLocalSearch(request: request)  
        search.start { response, _ in // se execută asincron  
            guard let response = response else {  
                return  
            }  
  
            self.locations = response.mapItems  
            self.tableView.reloadData()  
        }  
    }  
}
```

SearchViewController implementează două protocoale delegate/data source: *UISearchBarDelegate* pentru a specifica comportamentul celorlalor componente de UI (în special floating panel) atunci când bara este activată sau dezactivată și *UITableViewDelegate*, respectiv *UITableViewDataSource* pentru a popula tabelul cu date și a defini afișarea acestora, dar și pentru a gestiona selectarea unei celule din tabel, urmând ca controller-ul să trimită datele locației selectate mai departe către DetailsViewController.

- **DetailsViewController**

DetailsViewController este view controller-ul chemat prin selectarea unei locații fie din FavouritesViewController, fie din SearchViewController și, după cum sugerează

numele, schimbă conținutul floating panel-ului astfel încât să afișeze detaliile relevante ale punctului de interes selectat, disponibile în modelul LocationDetailsModel. Tot aici este disponibil, în funcție de cât de aproape este utilizatorul de locația respectivă, un buton care trimite către RoutesViewController și încă un buton care permite adăugarea, respectiv eliminarea acesteia din lista de favorite.

La încărcarea acestui view controller sunt preluate detaliile locației prin apelarea metodei *getPlaceDetails()* din PlacesManager, care apelează mai departe Foursquare Places API și parsează datele obținute în LocationDetailsModel. DetailsViewController implementează protocoalele *UICollectionViewViewDelegate* și *UICollectionViewDataSource* pentru a afișa imaginile sub formă de galerie orizontală derulantă.

- **RoutesViewController**

RoutesViewController este view controller-ul chemat prin selectarea butonului „Get Directions” din DetailsViewController și schimbă conținutul floating panel-ului pentru a afișa, sub formă de tabel, rutele disponibile între locația curentă și cea selectată. În același timp, harta este redimensionată pentru a cuprinde cele două puncte și proiectează cu ajutorul atributului *MKPolyline* rutele respective.

Datele despre rute sunt afișate în cadrul unui tabel ce folosește layout-ul celulei RouteTableViewCell pentru a afișa pe câte un rând distanța și durata fiecărei rute. În urma selectării unui rând din tabel, polyline-ul de pe hartă este evidențiat prin schimbarea culorii. Apăsând butonul „GO” din dreptul unei rute, este chemat DirectionsViewController care începe sesiunea de orientare.

- **DirectionsViewController**

DirectionsViewController este inițiat prin selectarea unei rute din RoutesViewController și modifică atât panoul adaptiv, cât și bara principală cu scopul afișării instrucțiunilor de mers pentru a ajunge la o locație, în partea de sus având efectiv instrucțiunea curentă, iar în panel o descriere a rutei împreună cu opțiunile de a trece la orientarea folosind realitatea augmentată sau de a încheia ruta, moment în care utilizatorul este trimis înapoi la DetailsViewController.

Prin apăsarea butonului „View in AR”, harta este înlocuită de output-ul camerei de pe spatele telefonului, prin care sunt proiectate în lumea reală indicatoare ce simulează ruta de mers folosind realitatea augmentată. În modul AR, butonul care comută între cele două moduri primește altă culoare, iconiță și text: „Map View”.

Întregul proces de afișarea a direcțiilor este detaliat în secțiunile următoare.

- **ProfileViewController**

ProfileViewController este un alt tip de content afișat în floating panel și reprezintă secțiunea de profil a utilizatorului, în care sunt afișate fotografia de profil, numele complet, adresa de email și un buton care a întrerupt sesiunea de autentificare.

Prin apăsarea butonului de logout, este apelată metoda *logoutUser()* din extensia aceleiași clase ce se găsește în helper-ul de autentificare.

Acest view controller permite și schimbarea fotografiei de profil, prin prezentarea unui *UIImagePickerController* în momentul în care utilizatorul apasă pe poza curentă. Acest controller este configurat să selecteze o anumită fotografie din biblioteca utilizatorului, iar în metoda delegate actualizează baza de date cu noua imagine aleasă, ca mai apoi să o modifice în timp real și în aplicație.

```

extension ProfileViewController: UIImagePickerControllerDelegate,
    < UINavigationDelegate {
    func presentPhotoPicker() {
        let picker = UIImagePickerController()
        picker.sourceType = .photoLibrary
        picker.delegate = self
        picker.allowsEditing = true

        present(picker, animated: true)
    }

    func imagePickerController(_ picker: UIImagePickerController,
        < didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any])
    {
        picker.dismiss(animated: true)

        if let editedImage = info[UIImagePickerController.InfoKey.editedImage] as?
            UIImage {
            profileImage.image = editedImage
            mainVC!.profileImage.image = editedImage

            if Auth.auth().currentUser != nil {
                let currentUser = DataManager.shared.getUser(email:
                    < (Auth.auth().currentUser?.email)!)
                if currentUser != nil {
                    let jpegImageData = editedImage.jpegData(compressionQuality: 1.0)
                    DataManager.shared.updateProfileImage(profileImage:
                        < jpegImageData!, user: currentUser!)
                }
            }
        }
    }
}

```

3.5 Utilities

Directorul de **utilități** aduce extensii ale claselor ViewController deja existente sub formă de **Helpers** menite să separe codul de gestionare a componentelor UI și interacțiunilor cu acestea de restul metodelor care contribuie la implementarea funcționalităților aplicației. Tot aici se găsesc și clasele **Manager** care se ocupă de conexiunea cu baza de date și API-urile folosite, respectiv preluarea, modificarea și ștergerea datelor din acestea. Vom reveni la o prezentare mai detaliată a acestora în secțiunile ce detaliază implementarea funcționalităților cheie. Utilitățile folosite în dezvoltarea aplicației „TakeMeOut” sunt:

- **DataManager** - folosește *DataManager* din Core Data pentru a manipula baza de date; prin intermediul contextului din *NSPersistentContainer*, se pot realiza operațiile de extragere a datelor prin *NSFetchRequest*, actualizare și ștergere a cămpurilor și înregistrărilor, împreună cu operația de salvare a contextului actual;
- **PlacesManager** - are rolul de a apela Places API de la Foursquare pentru a obține în primă instanță id-ul unic al unei locații, ca mai apoi să extragă imaginile și restul detaliilor prin alte request-uri care se realizează asincron; tot aici se găsește și implementarea request-ului către MapKit pentru extragerea rezultatelor în urma unei căutări inițiate de SearchViewController;
- **CategoriesHelper** - cuprinde un dicționar care atribuie fiecărei categorii de locație denumirea iconicei de sistem corespunzătoare pentru a facilita afișarea acestieia în cadrul aplicației;
- **LocationHelper** - cuprinde extensii ale claselor MainViewController și SearchViewController care implementează funcții de calculare a distanței între două puncte reprezentate ca *MKMapItem* și de actualizare a locației curente a utilizatorului; tot aici se găsește metoda care inițiază trimiterea notificărilor către utilizator la schimbarea locației, când aplicația este în background, sub numele *notifyForUpdatingLocation()*;
- **DirectionsSessionHelper** - cuprinde o extensie a clasei MainViewController ce definește metodele auxiliare folosite în afișarea și configurarea direcțiilor de mers către o anumită locație;
- **AuthHelper** - cuprinde extensii ale claselor LoginViewController, RegisterViewController și ProfileViewController care implementează funcțiile de autentificare, creare de utilizator și deconectare.

Următoarele secțiuni detaliază implementarea funcționalităților cheie ale aplicației, care combină toate elementele menționate până în acest punct.

3.6 Autentificarea utilizatorilor

La login, utilizatorului îi este solicitată introducerea adresei de email și a parolei. La apăsarea butonului de sign in este apelată metoda `loginUser()` din AuthHelper. Această metodă verifică întâi existența unui cont cu adresa de email dată prin apelarea metodei `Auth.auth().fetchSignInMethods()`: în cazul în care acest email nu este înregistrat în baza de date din Firebase, este afișată o alertă `UIAlertController` care anunță utilizatorul cu privire la invaliditatea combinației de email și parolă. Dacă email-ul există, se încearcă în continuare autentificarea cu email-ul și parola prin apelarea metodei `Auth.auth().signIn()`: în cazul unei erori este afișată o alertă identică, iar dacă autentificarea s-a realizat cu succes, utilizatorul este trimis către view-ul principal.

Utilizatorul are și opțiunea de autentificare folosind un cont Google. La apăsarea butonului corespunzător este inițiat flow-ul standard de autentificare de la Google. Dacă validarea contului Google s-a realizat cu success, instanța de autentificare `GIDSignIn` întoarce un rezultat ce conține un obiect user de tip `GIDGoogleUser` și un token unic de identificare. Se încearcă în continuare autentificarea în Firebase tot prin apelarea metodei `Auth.auth().signIn()`, dar care ia ca argumente token-ul de identificare și un alt token de acces aparținând instanței user. Dacă și autentificarea s-a realizat cu succes, înainte de a trimite user-ul în dashboard, se verifică dacă există contul asociat în Core Data, iar în caz contrar se creează un user nou folosind numele complet și adresa de email preluate tot din instanța `GIDGoogleUser`.

La înregistrare, utilizatorul are de completat câmpurile corespunzătoare numelui complet, adresei de email, parolei și încă un câmp pentru confirmarea parolei. Toate aceste câmpuri sunt validate în frontend prin expresii `regex`, iar introducerea de date în format invalid într-unul dintre câmpuri va determina afișarea unei erori imediat sub acesta, butonul de register neputând fi apăsat. Dacă toate câmpurile sunt valide, se poate apăsa butonul de register care apelează metoda `createUser()`. Aceasta validează încă o dată câmpurile și trece prin aceeași verificare ca la login privind adresa de email: dacă aceasta este deja înregistrată se afișează o alertă corespunzătoare, iar în caz contrar se încearcă în continuare crearea contului prin apelarea metodei `Auth.auth().createUser()`. Dacă aceasta s-a realizat cu succes, se autentifică utilizatorul și se creează o evidență a sa și în Core Data folosind numele și adresa de email din câmpurile complete.

3.7 Mecanismul de trimitere a notificărilor

Odată ce aplicația intră în background, location manager va anunța view controller-ul principal când utilizatorul își schimbă locația. MainViewController verifică în metoda delegate ca starea aplicației să fie background, după care verifică dacă utilizatorul a dat permisiune aplicației să trimită notificări. Dacă toate aceste verificări se fac cu succes,

este apelată metoda *notifyForUpdatingLocation()* din LocationHelper. Aici sunt preluate locațiile salvate de utilizator din Core Data și se verifică, pentru fiecare locație pentru care nu s-a trimis deja notificare, dacă aceasta se află într-o rază de 1 km, caz în care se realizează un *UNNotificationRequest* ce primește drept conținut un titlu sugestiv și un body text reprezentând numele punctului de interes din apropiere.

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
    ↵ [CLLocation]) {
    ...
    if UIApplication.shared.applicationState == .active {
        ...
    } else if Auth.auth().currentUser != nil {
        if canSendNotifications && applicationStarted {
            notifyForUpdatingLocation(CLLocation(latitude: locValue.latitude,
                ↵ longitude: locValue.longitude))
        }
    }
}
```

Metoda folosește un dicționar ce reține drept chei coordonatele locațiilor la care s-a trimis notificare pentru a se asigura că nu se trimit un spam de notificări identice în aceeași sesiune, având în vedere că locația utilizatorului se actualizează frecvent.

3.8 Salvarea și ștergerea locațiilor favorite

În secțiunea de detalii ale unei locații, utilizatorul poate adăuga o locație în lista de favorite apăsând butonul „Add to Favourites” ce apelează metoda *addToFavourites()*. Aceasta preia mai întâi user-ul curent din Core Data, creează o locație nouă folosind constructorul din Data Manager ce primește ca parametri numele, coordonata, categoria și adresa acesteia, adăugând ca foreign key user-ul curent.

Dacă o locație este deja salvată în lista de favorite, același buton din secțiunea de detalii se transformă în „Remove from Favourites”. Prin apăsarea acestuia se apelează metoda *removeFromFavourites()* prin care se inițiază operația de ștergere a unei locații, definită în DataManager.

3.9 Afisarea instructiunilor de mers

3.9.1 Folosind harta

Începând din DetailsViewController, apăsarea butonului „Get Directions” apelează metoda *renderWalkingDirections()*. Această metodă preia coordonata specifică locației

curentă a utilizatorului și realizează un request de tip *MKDrections* configurat să genereze rute pedestriene multiple, dacă există, care primește ca sursă un *MKMapItem* reprezentând locația curentă, iar ca destinație un alt *MKMapItem* reprezentând locația selectată. Metoda *calculate()* întoarce în mod asincron o posibilă eroare și un răspuns ce conține o listă de *MKRoutes* reprezentând rutile calculate. În continuare, metoda proiectează traiectoriile rutelor pe map view cu ajutorul metodei *addOverlay()*, configuraază dimensiunea și poziționarea hărții astfel încât să cuprindă cele două puncte și rutele dintre ele, apoi cheamă *RoutesViewController*.

```
@objc func renderWalkingDirections() {
    // locația curentă
    guard let locValue: CLLocationCoordinate2D =
        mainVC!.locationManager.location?.coordinate else {
        return
    }

    let request = MKDirections.Request()
    request.source = MKMapItem(placemark: MKPlacemark(coordinate: locValue))
    request.destination = currentLocation?.mapItem
    request.transportType = MKDirectionsTransportType.walking
    request.requestsAlternateRoutes = true

    let directions = MKDirections(request: request)
    directions.calculate { [self] (response, error) in // se execută asincron
        ... // se tratează eroarea, dacă există
        for route in response.routes {
            mainVC!.mapView.addOverlay(route.polyline)
            mainVC!.mapView.setVisibleMapRect(
                route.polyline.boundingMapRect.offsetBy(dx: 0, dy: 500),
                edgePadding: UIEdgeInsets(top: 120, left: 50, bottom: 320, right:
                    50),
                animated: true
            )
        }
        ... // este configurat RoutesViewController
        mainVC!.panel.set(contentViewController: routesVC)
    }
}
```

Din *RoutesViewController*, apăsarea butonului „GO” din dreptul unei rute din tabel apelează metoda *getWalkingDirections()* care preia indexul rutei din atributul *tag* al butonului, șterge de pe hartă traiectoriile corespunzătoare celelalte rute și cheamă *DirectionsViewController*.

În *DirectionsViewController* se salvează pașii rutei selectate în *MainViewController* sub variabila *steps*, și se ține evidența pasului curent cu *stepId*, care este inițializat cu

1. Speech synthesizer-ul din MainViewController este configurat să rostească un mesaj inițial prin care este specificat numele locației către care se îndreaptă user-ul.

Întrucât pasul 0 dintr-o rută este chiar pasul de la care pleacă utilizatorul, se va afișa, respectiv rosti de către speech synthesizer instrucțiunea corespunzătoare pasului 0, motiv pentru care *stepId* este inițializat cu 1.

În continuare, în metoda delegate a managerului de localizare care gestionează schimbarea locației, este verificată constant apropierea de următorul pas. Având în vedere că atributul *polyline* al fiecărui pas reține coordonata la care începe, se poate verifica foarte ușor când utilizatorul se apropiie de un change point. Astfel, la apropierea de următorul pas, se afișează în bara de sus și este rostită instrucțiunea specifică acestui pas, după care se incrementează *stepId*.

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
    ↵ [CLLocation]) {
    ...
    if areDirectionsEnabled {
        let currentLocation = CLLocation(latitude: locValue.latitude, longitude:
            ↵ locValue.longitude)
        let changePoint = CLLocation(latitude:
            ↵ steps[stepId].polyline.coordinate.latitude, longitude:
            ↵ steps[stepId].polyline.coordinate.longitude)
        ...
        if currentLocation.distance(from: changePoint) <= 5.0 {
            ...
            directionsLabel.text = steps[stepId].instructions
            speechSynthesizer.speak(AVSpeechUtterance(string:
                ↵ steps[stepId].instructions))

            if stepId < steps.count - 1 {
                stepId += 1
            }
        }
    }
}
```

Pentru a actualiza traекторia de mers astfel încât să se șteargă pe măsură ce este parcursă, metoda delegate ține evidența mai multor puncte de pe traекторie (*turn points*) prin variabila *turnId*. Aceasta este folosită mai departe pentru a actualiza constant distanța totală parcursă ca fiind suma distanțelor Euler dintre fiecare două puncte consecutive până la punctul curent, la care se adaugă și distanța Euler de la punctul curent la punctul reprezentând locația curentă pentru un plus de precizie. Acest lucru este posibil deoarece oricare două turn points consecutive sunt pe o linie dreaptă. Aceste turn points sunt preluate din polyline-ul fiecărui pas, care este definit prin mulțimea punctelor de curbă. Distanța parcursă este calculată sub formă de progres (ce procentaj din distanța totală

a fost parcurs) în metoda *calculateProgress()* din DirectionsSessionHelper. Această metodă apelează mai departe *getTraveledDistance()* care calculează distanța parcursă după algoritmul explicitat anterior.

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
    ↵ [CLLocation]) {
    ...
    if areDirectionsEnabled {
        ...
        let turnPoint = CLLocation(latitude:
            ↵ (selectedPolyline?.coordinates[turnId].latitude)!, longitude:
            ↵ (selectedPolyline?.coordinates[turnId].longitude)!)

        if currentLocation.distance(from: turnPoint) <= 5.0 {
            if turnId < (selectedPolyline?.coordinates.count)! - 1 {
                turnId += 1
            }
        }

        let renderer = mapView.renderer(for: selectedPolyline!) as?
            MKPolylineRenderer
        renderer?.strokeStart = calculateProgress(to: locValue)
    }
}
```

În același timp, metoda delegate este responsabilă și de ajustarea camerei care trebuie să rămână centrată pe locația curentă și rotită astfel încât polyline-ul să fie orientat către nordul geografic. Acest lucru este realizat prin apelarea metodei *setCamera()*, ce primește configurația de cameră cu centrul în coordonata locației curente și orientarea la direcția (unghiul) dintre coordonata curentă și cea corespunzătoare următorului turn point.

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
    ↵ [CLLocation]) {
    ...
    if areDirectionsEnabled {
        ...
        mapView.setCamera(
            MKMapCamera(lookingAtCenter: locValue, fromDistance: 750, pitch: 0,
            ↵ heading: radiansToDegrees(radians: calculateBearing(from: locValue,
            ↵ to: turnPoint.coordinate))),
            animated: true
        )
        ...
    }
}
```

Acest proces se repetă până când se ajunge la ultimul pas, moment în care sesiunea de orientare poate fi opriță.

3.9.2 Folosind realitatea augmentată

În DirectionsViewController, sesiunea de realitate augmentată rulează în paralel cu cea de direcționare pe hartă, în cazul în care dispozitivul suportă *ARWorldTrackingConfiguration*. Aceasta este configurată în metoda *setUpARSession()* la încărcarea view controller-ului. În această metodă este creată o configurație de tip *ARWorldTrackingConfiguration* pe care sesiunea din *ARView*-ul aparținând view controller-ului principal începe să o ruleze. Tot aici sunt reținute poziția și orientarea curente ale dispozitivului pe post de origine a sistemului imaginar.

Originea și orientarea spațiului virtual sunt stabilite ca fiind coordonata și orientarea curente ale dispozitivului din momentul în care a început execuția sesiunii și nu se mai schimbă până la rularea altei sesiuni. Astfel, la plasarea obiectelor virtuale în spațiu se ține cont de cele două sisteme de coordonate care se formează, păstrând convenția mâinii drepte (axa Y este axa verticală, XZ este planul orizontal, +Ziese din palmă și +X urmează direcția degetului mare): sistemul real, unde -Z indică nordul și centrul este locația curentă a utilizatorului și sistemul imaginar, unde -Z indică orientarea dispozitivului și centrul este locația utilizatorului din momentul în care a început rularea sesiunii.

```
func setUpARSession() {
    let configuration = ARWorldTrackingConfiguration()
    mainVC!.arView.session.run(configuration)

    guard let locValue: CLLocationCoordinate2D =
        mainVC!.locationManager.location?.coordinate else {
        return
    }
    guard let locHeading: CLLocationDirection =
        mainVC!.locationManager.heading?.magneticHeading else {
        return
    }

    mainVC!.startCoordinate = locValue
    mainVC!.startHeading = locHeading
}
```

La fel ca la afișarea pașilor pe hartă, și în AR se afișează din start indicatoarele pentru primul pas. În schimb, la atingerea unui pas nou, se vor afișa indicatoarele pentru pasul următor pasului nou pentru a păstra continuitatea direcțiilor; astfel, la început se vor afișa de fapt indicatoarele pentru primii doi pași (0 și 1). În metoda delegate a managerului de localizare, în momentul în care se ajunge la pasul următor, se verifică încă o

dată dacă device-ul suportă *ARWorldTrackingConfiguration* (chiar dacă am verificat În DirectionsViewController, metoda delegate este în MainViewController și rulează indiferent dacă această configurație este suportată, deci este nevoie de încă o verificare) și apelează metoda *addAnchors()* din DirectionsSessionHelper pentru pasul următor.

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations:
    ↪ [CLLocation]) {
    ...
    if currentLocation.distance(from: changePoint) <= 5.0 {
        if ARWorldTrackingConfiguration.isSupported {
            if stepId < steps.count - 1 {
                addAnchors(forStep: stepId + 1)
            }
        }
        ...
    }
}
```

În *addAnchors()*, se iau toate punctele de curbură ce alcătuiesc polyline-ul pasului respectiv pentru a se plasa ancore virtuale în dreptul lor. Totuși, traекторiile pot fi atât linii drepte, cât și curbe astfel că ancorele nu vor fi la distanțe egale și vor apărea destul de rar. Pentru a remedia această situație și a păstra continuitatea rutei, se preiau toate aceste puncte de curbură și se adaugă puncte adiționale la distanțe egale între 2 puncte consecutive prin apelarea metodei *interpolatePoints()*.

Pentru fiecare punct din totalitatea punctelor deja existente și generate se calculează poziția relativă a punctului față de originea sistemului imaginar prin metoda *getRelativeDistance()*, după care se apelează metoda *addAnchor()* care ia drept parametru poziția calculată.

```
func addAnchors(forStep stepIndex: Int) {
    let routeCoordinates = steps[stepIndex].polyline.coordinates
    var lineCoordinates: [CLLocationCoordinate2D] = []

    for i in 0 ..< routeCoordinates.count {
        lineCoordinates.append(routeCoordinates[i])
        if i < routeCoordinates.count - 1 {
            lineCoordinates.append(contentsOf: interpolatePoints(start:
                ↪ routeCoordinates[i], end: routeCoordinates[i + 1])))
        }
    }
    for coordinate in lineCoordinates {
        let position = getRelativePosition(from: startCoordinate!, to: coordinate)
        addAnchor(at: position)
    }
}
```

Poziția relativă este un vector 3D care indică poziția ancorei ce urmează să fie poziționată, în relație cu originea sistemului virtual. Metoda *getRelativeDistance()* preia coordonata din spațiul real și o transpune în sistemul virtual. Pentru aceasta, calculează distanța dintre origine (coordonata specifică locației dispozitivului la începerea sesiunii) și punctul de ancorat împreună cu unghiul dintre dreapta determinată de cele două puncte și axa Z („nordul”). Folosind distanța și unghiul, se determină coordonatele Z (3.1) și X (3.2) prin aplicarea formulelor trigonometrice (în ARKit, rotația în jurul axei Y se realizează invers acelor de ceasornic, motiv pentru care coordonata Z trebuie înmulțită cu -1):

$$Z = -\text{haversineDistance} \cdot \cos(\varphi) \quad (3.1)$$

$$X = \text{haversineDistance} \cdot \sin(\varphi) \quad (3.2)$$

Distanța dintre cele două puncte este calculată folosind formula lui Haversine[27], care ține cont de curbura Pământului - 6371 km (3.3, 3.4), în metoda *getHaversineDistance()*, iar unghiul folosind formula clasica de unghi relativ la nordul geografic[3] (bearing - 3.5, 3.6, 3.7) în metoda *calculateBearing()*:

$$a = \sin\left(\frac{\Delta\text{lat}}{2}\right)^2 + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin\left(\frac{\Delta\text{lon}}{2}\right)^2 \quad (3.3)$$

$$\text{haversineDistance} = 2 \cdot \arctan\left(\frac{\sqrt{a}}{\sqrt{1-a}}\right) \cdot 6371 \quad (3.4)$$

$$y = \sin(\Delta\text{lon}) \cdot \cos(\text{lat}_2) \quad (3.5)$$

$$x = \cos(\text{lat}_1) \cdot \sin(\text{lat}_2) - \sin(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \cos(\Delta\text{lon}) \quad (3.6)$$

$$\text{bearing} = \arctan(y, x) \quad (3.7)$$

Deoarece unghiul este calculat relativ la nordul geografic, adică axa Z a spațiului real, pentru a obține unghiul final (3.8), scădem unghiul reprezentând orientarea dispozitivului la începutul sesiunii:

$$\varphi = \text{bearing} - \text{startHeading} \quad (3.8)$$

Pentru adăugarea ancorei, se creează întâi modelul acesta care este alcătuit din 2 sfere concentrice, una mai mică, de culoare albastră și cealaltă mai mare, transparentă. Pentru fiecare sferă se creează un *ModelEntity* ce primește ca matrice o sferă generată de *MeshResource* și materialul corespunzător. Cele 2 modele se adaugă la ancorea creată cu *AnchorEntity* și poziționată la punctul dat ca parametru, după care ancorea este adăugată la scena AR cu ajutorul metodei *scene.addAnchor()*.

```
func addAnchor(at position: SIMD3<Float>) {
    let innerSphere = ModelEntity(
        mesh: MeshResource.generateSphere(radius: 0.66),
        materials: [UnlitMaterial(color: #colorLiteral(red: 0, green: 0.3, blue: 1.4,
            alpha: 1))]
    )
    let outerSphere = ModelEntity(
        mesh: MeshResource.generateSphere(radius: 1),
        materials: [SimpleMaterial(color: #colorLiteral(red: 1, green: 1, blue: 1,
            alpha: 0.25), roughness: 0.3, isMetallic: true)]
    )
    let anchor = AnchorEntity(world: position)

    anchor.addChild(innerSphere)
    anchor.addChild(outerSphere)

    arView.scene.addAnchor(anchor)
}
```

Capitolul 4

Prezentarea aplicației

4.1 UI/UX

Aplicația „TakeMeOut” are un design modern, asemănător aplicațiilor native Apple și reușește să asigure utilizatorilor o experiență cât mai intuitivă de navigare, având un parcurs simplificat în care este nevoie de un număr minim de interacțiuni pentru a folosi funcționalitățile acesteia. De asemenea, animațiile componentelor UI și viteza de execuție contribuie la naturalețea navigării.

Pentru a economisi din resursele hardware necesare afișării, aplicația suportă și dark mode.

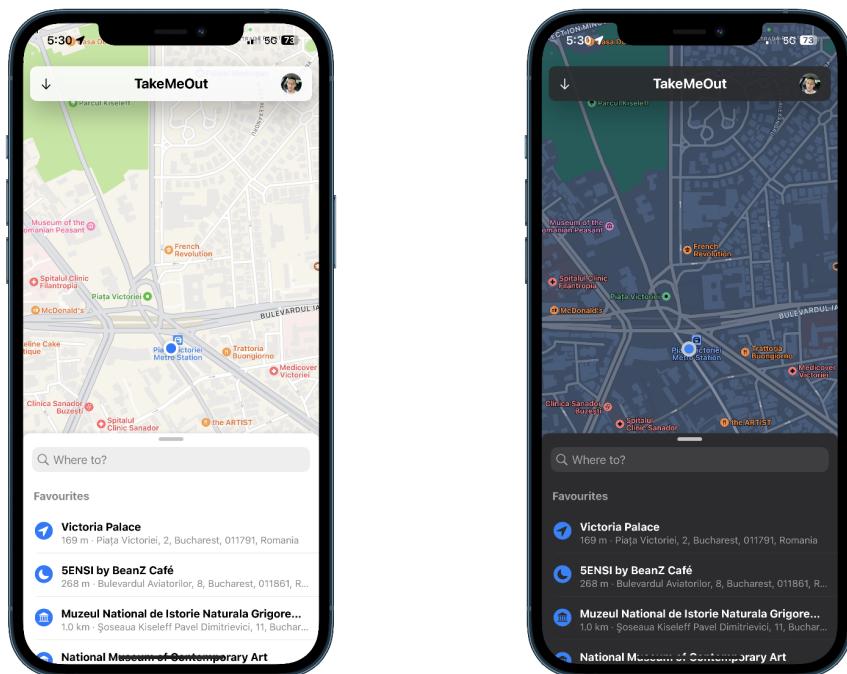


Figura 4.1: Vizualizarea paginii principale în light mode și dark mode

4.2 Înregistrare și Autentificare

În cazul în care utilizatorul nu este autentificat, acesta este redirectionat către pagina de autentificare. Aici, are opțiunea de a se autentifica folosind un cont deja existent, prin introducerea adresei de email și a parolei. Dacă credențialele sunt valide și corespund unui cont existent în baza de date, autentificarea se realizează cu succes, iar în caz contrar este afișată o alertă sugestivă.

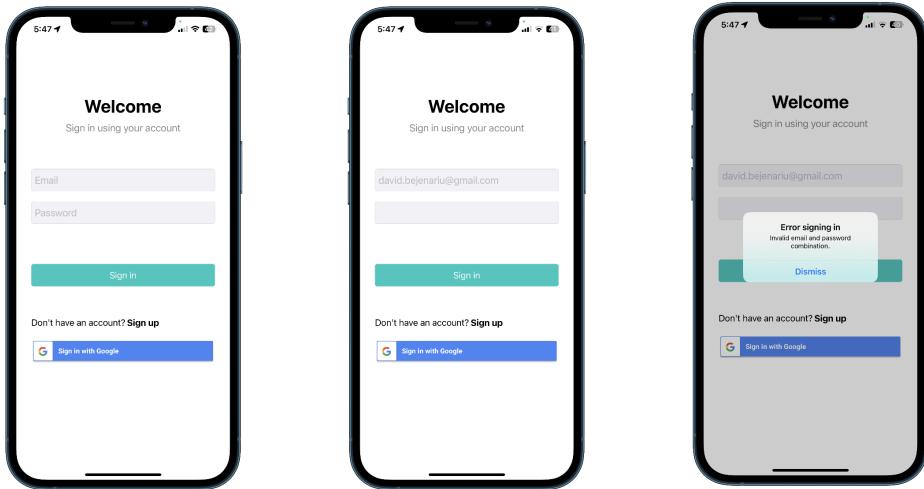


Figura 4.2: Procesul de autentificare și eroarea la validarea credențialelor

Dacă utilizatorul nu detine încă un cont, poate apăsa pe „Sign up” pentru a fi redirectionat la pagina de înregistrare. Aici, va trebui să introducă numele complet, adresa de email, parola și confirmarea acesteia. Toate câmpurile sunt validate, astfel că procesul de creare a contului nu poate continua până când nu sunt introduse informații valide.

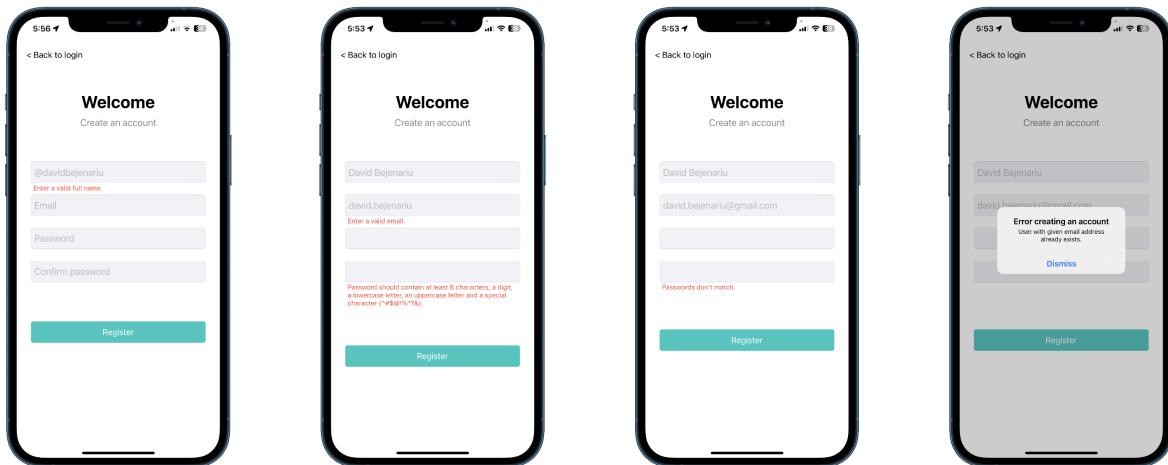


Figura 4.3: Erori la validarea câmpurilor de înregistrare

Odată apăsat butonul „Register”, dacă adresa de email nu este deja înregistrată se creează contul cu succes și utilizatorul este autentificat, iar în caz contrar se afișează o alertă sugestivă.

4.2.1 Autentificarea cu Google

Utilizatorul poate opta pentru autentificarea folosind un cont de Google. Prin apăsarea butonului „Sign in with Google”, acesta este trimis într-un view separat în care trebuie să autorizeze un cont de Google. Dacă acest pas este realizat cu succes, utilizatorul este autentificat și redirecționat către dashboard-ul aplicației.

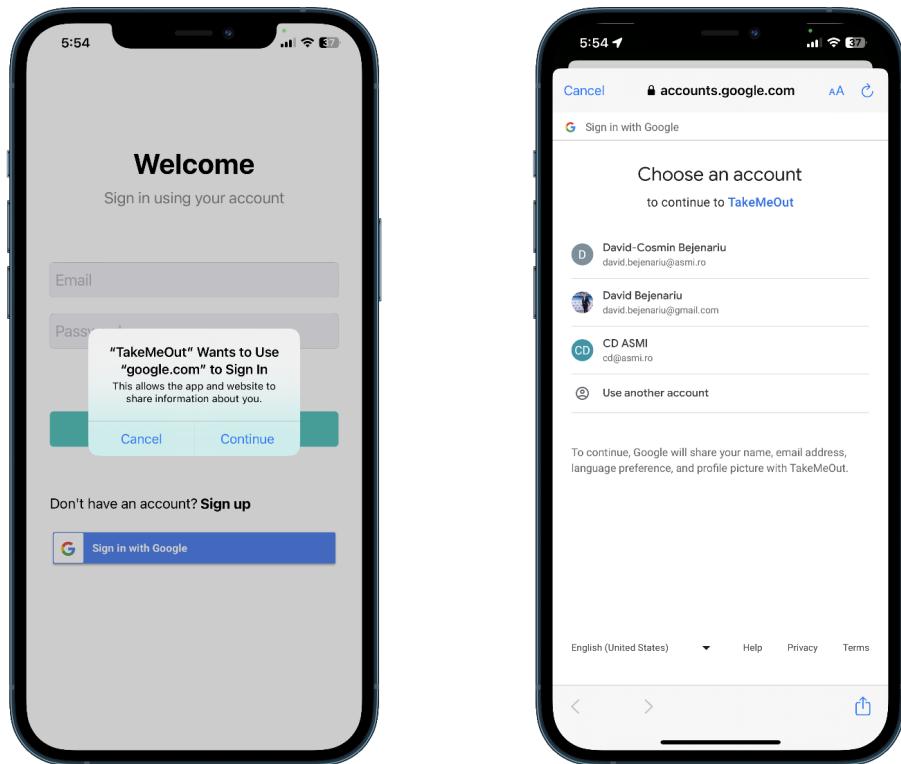


Figura 4.4: Autentificarea cu Google

4.3 Pagina principală

Dacă utilizatorul este autentificat, aplicația se deschide în dashboard. Aici, poate explora harta centrată în locația curentă a dispozitivului și folosi butonul în formă de săgeată din bara de sus pentru a recentra harta. Dacă aplicația este la prima rulare după instalare, îi va solicita utilizatorului acordul cu privire la folosirea camerei, locației curente și trimiterea de notificări.

4.3.1 Vizualizarea locațiilor salvate

În panoul de jos apar în mod implicit locațiile salvate, sau un mesaj sugestiv în cazul în care nu are locații salvate. Panoul poate fi extins pentru a vedea toate locațiile salvate, sau minimizat pentru a putea vedea mai mult din conținutul hărții. Locațiile apar în listă în ordinea crescătoare a distanței față de utilizator.

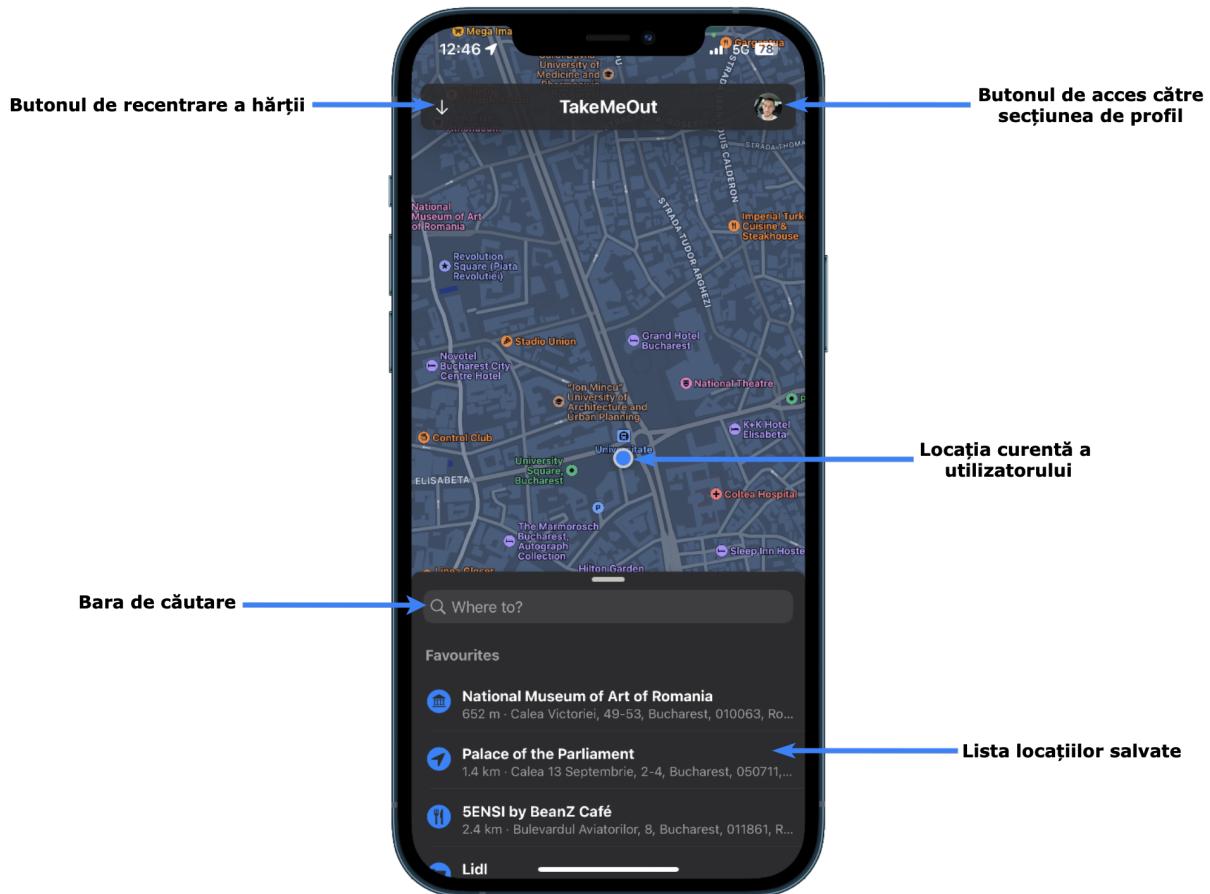


Figura 4.5: Pagina principală

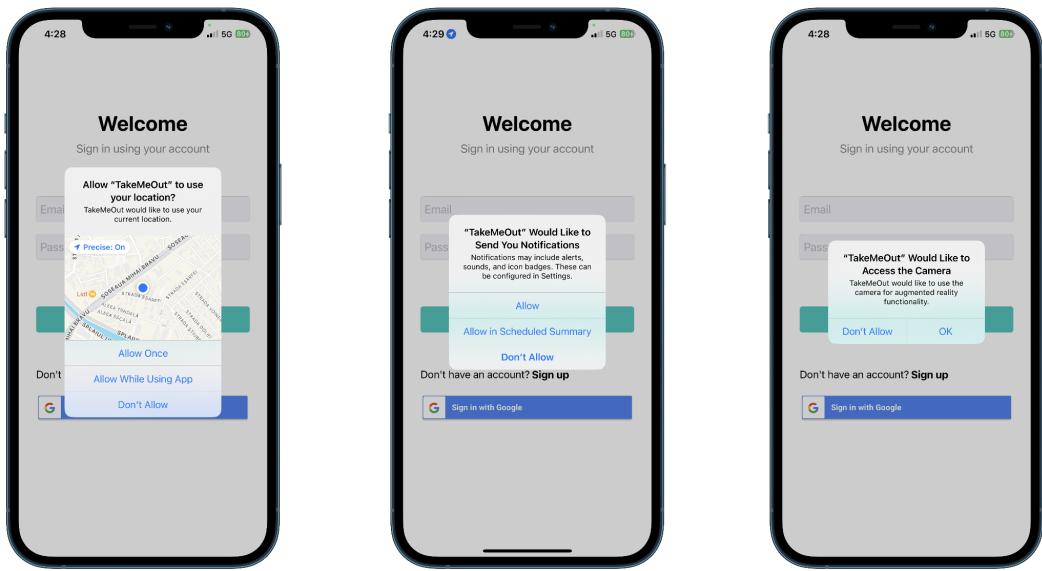


Figura 4.6: Cererea autorizării privind trimiterea de notificări și folosirea camerei și locației curente

4.4 Secțiunea de profil

Prin apăsarea iconiței de profil din partea dreaptă a barei de sus, utilizatorul accesează secțiunea de profil, care este vizibilă în panoul de jos. Aici sunt afișate informațiile asociate contului său. Prin apăsarea butonului „Logout”, utilizatorul este deconectat și redirectionat către pagina de autentificare.



Figura 4.7: Secțiunea de detalii ale profilului și schimbarea imaginii de profil

4.4.1 Adăugarea/modificarea fotografiei de profil

Prin apăsarea imaginii de profil, aplicația redirecționează către un view în care utilizatorul poate explora întreaga galerie de imagini pentru a adăuga sau schimba fotografia de profil. Odată selectată o imagine, aceasta poate fi ajustată astfel încât să cuprindă conținutul dorit în frame-ul pătrat disponibil. Apăsând „Choose”, aplicația modifică instantaneu imaginea. Prin selectarea butonului „Cancel” se poate reveni la secțiunea de profil.

4.5 Căutarea punctelor de interes

Accesarea barei de căutare din panoul de jos determină extinderea panoului, afișarea tastaturii și eliminarea afișajului locațiilor favorite, dacă există. Utilizatorul poate taste orice cuvânt cheie, denumire sau adresă specifică unei locații, iar aplicația va afișa instantaneu rezultatele căutării pe măsură ce acesta tastează. Prin apăsarea butonului „Cancel” se revine la vizualizarea inițială.

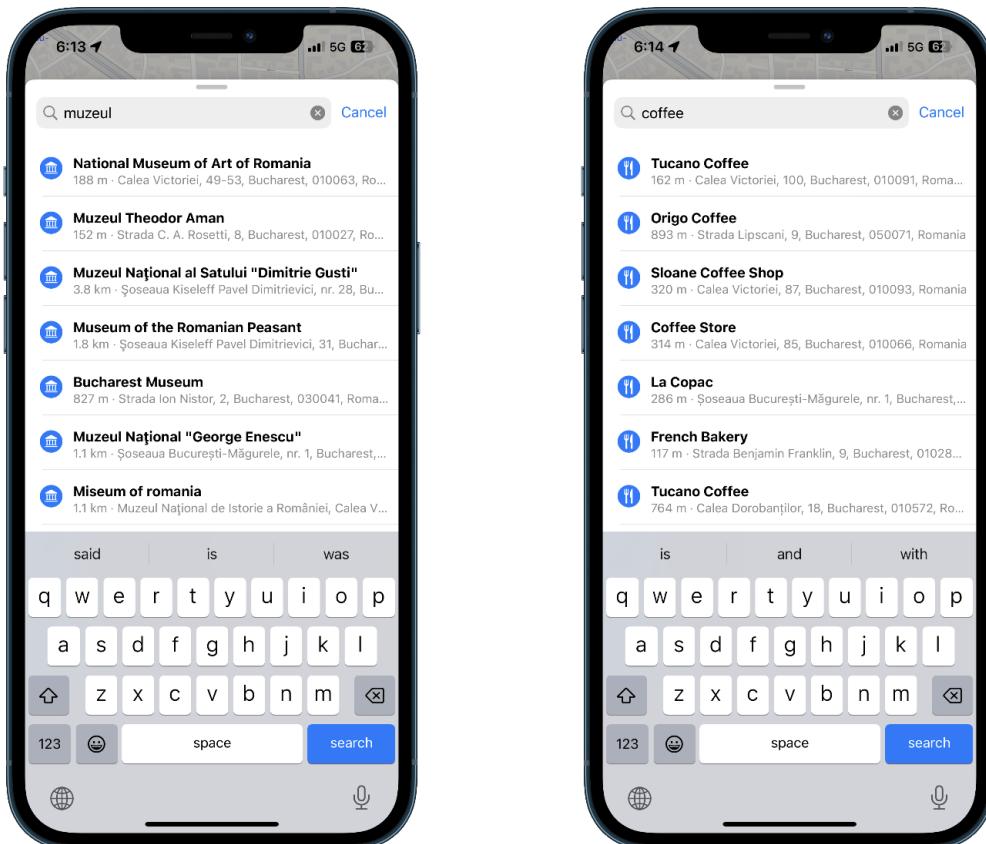


Figura 4.8: Folosirea barei de căutare pentru a explora locații noi

4.6 Vizualizarea punctelor de interes

Selectarea unei locații fie din lista de favorite, fie din lista rezultatelor unei căutări determină vizualizarea detaliilor acesteia în panoul de jos. Aici vor fi afișate, în funcție de disponibilitate, denumirea, adresa completă, descrierea, programul de funcționare, imaginile, rating-ul, popularitatea, numărul de telefon și website-ul specifice punctului de interes selectat. De asemenea, este afișată distanța dintre locație și utilizator împreună cu butonul „Get Directions” de generare a direcțiilor în cazul în care aceasta se află în range-ul maxim. În regiunile suportate, locațiile pot fi vizualizate direct pe hartă în format 3D.

Panoul poate fi extins dar și derulat pentru a permite vizualizarea tuturor detaliilor. Galeria poate fi, de asemenea, derulată pentru vizualizarea tuturor imaginilor. Butonul „X” din partea de dreapta sus a panoului determină închiderea vizualizării și revenirea la cea inițială.

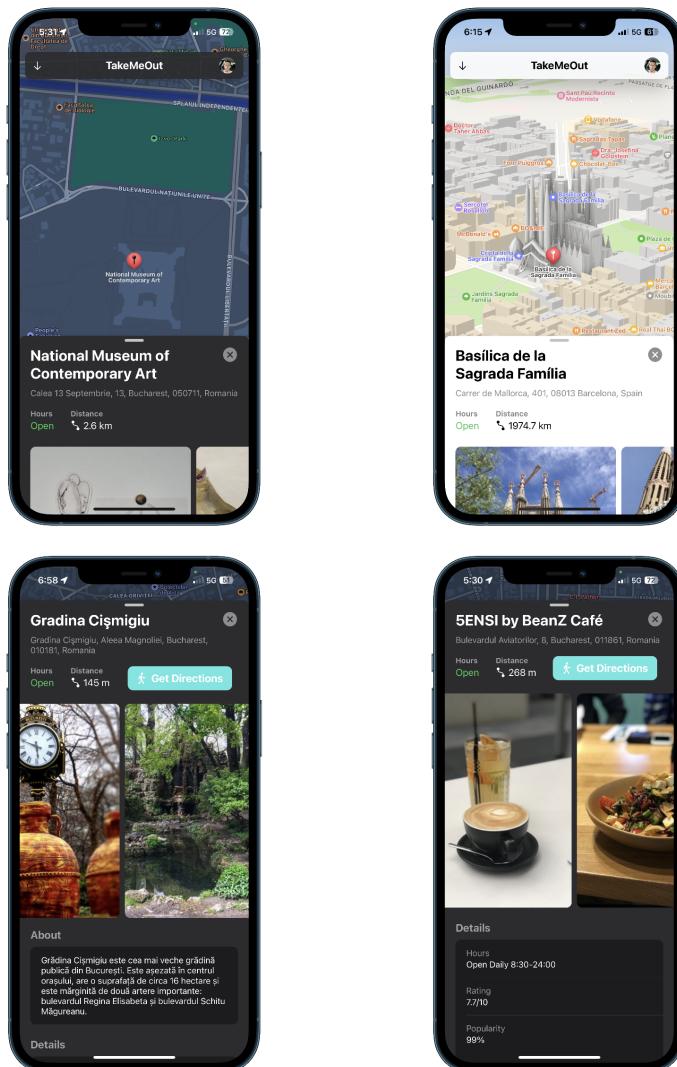


Figura 4.9: Vizualizarea punctelor de interes și a detaliilor acestora

4.6.1 Adăugarea/eliminarea unui punct de interes

Ultimul element prezent în vizualizarea detaliilor este un buton care permite adăugarea locației selectate din lista de favorite, respectiv ștergerea acesteia dacă este deja salvată. Apăsarea butonului determină acțiunea corespunzătoare, butonul schimbându-și afișarea și blocând alte posibile interacțiuni. Dacă utilizatorul se întoarce în vizualizarea principală după această acțiune, poate observa actualizarea instantanee a listei de locații salvate.

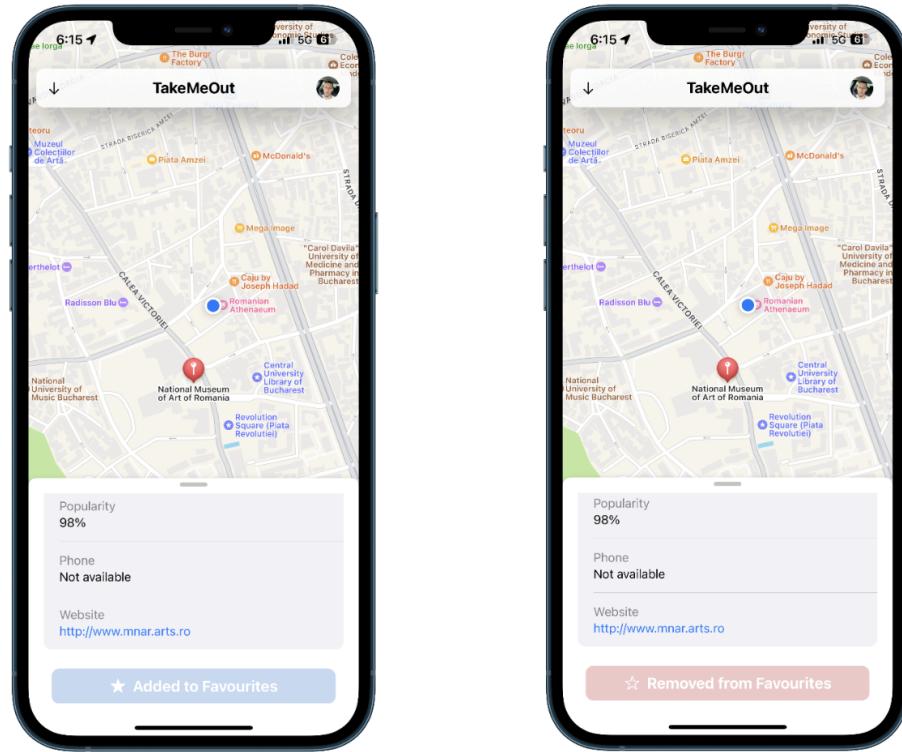


Figura 4.10: Adăugarea/eliminarea unui punct de interes

4.7 Primirea notificărilor

Dacă aplicația este în background, utilizatorul va primi câte o notificare pentru fiecare punct de interes salvat în lista de favorite ce se află în apropiere. Notificările vor apărea și pe celelalte dispozitive asociate, spre exemplu Apple Watch.



Figura 4.11: Primirea de notificări pe dispozitivul mobil și ceas

4.8 Afisarea rutelor de direcționare

Prin apăsarea butonului „Get Directions” este prezentată vizualizarea rutelor de mers disponibile până la locația selectată. Rutele sunt prezentate atât pe hartă prin traectoriile corespunzătoare, dar și în panou sub forma unei liste derulante ce specifică durata estimată și distanța totală a fiecărei rute. Prin selectarea unui element din listă este evidențiată ruta corespunzătoare.

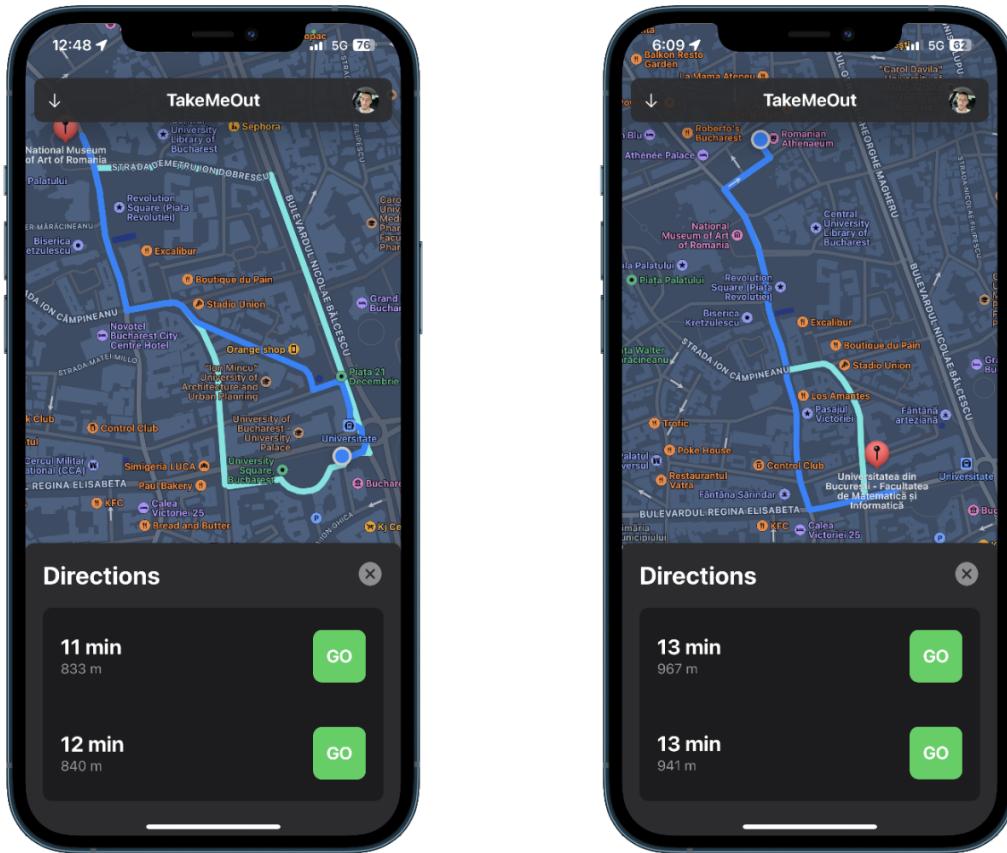


Figura 4.12: Afisarea rutelor de mers către un punct de interes

4.9 Urmarea instrucțiunilor de mers

Selectarea butonului „GO” din dreptul unei rute determină prezentarea unei noi vizualizări a direcțiilor de mers folosind ruta selectată. Bara de sus se extinde pentru a putea afișa instrucțiunea curentă, iar conținutul panoului de jos este modificat pentru a putea afișa descrierea rutei, împreună cu cele două butoane prin care se poate schimba modul de afișare sau termina sesiunea de orientare. Apăsarea butonului „End route” determină întoarcerea la vizualizarea detaliilor locației selecțiate.

4.9.1 Folosind harta

Această vizualizare afișează în mod implicit direcțiile de orientare pe hartă, care este redimensionată și recentrată pentru a atrage atenția asupra rutei, a cărei traекторii este orientată spre nord și ajustată pe măsură ce utilizatorul merge de-a lungul acesteia. Utilizatorul poate urmări traectoria de mers proiectată pe hartă împreună cu instrucțiunile care se actualizează la fiecare pas pentru a ajunge la destinație.

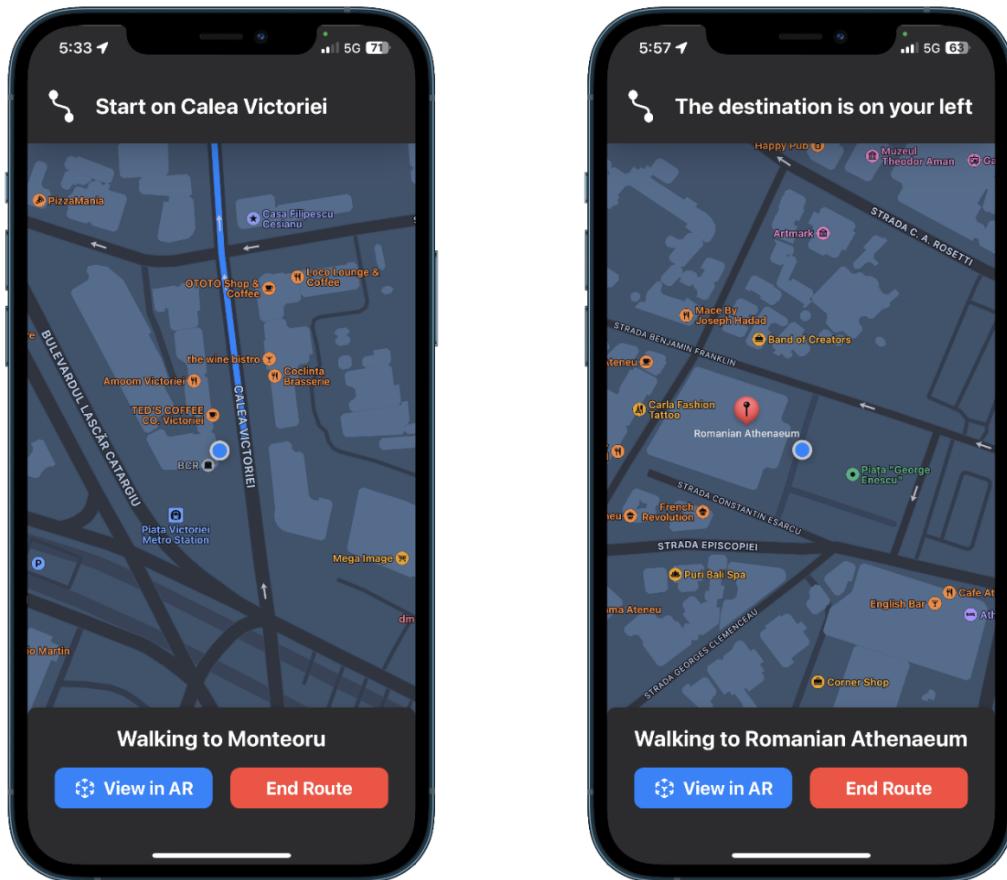


Figura 4.13: Urmarea instrucțiunilor de mers folosind harta

4.9.2 Folosind realitatea augmentată

Prin apăsarea butonului „View in AR”, harta este înlocuită de vizualizarea generată de output-ul camerei din spatele dispozitivului. Aici, traiectoria de mers este proiectată folosind realitatea augmentată, sub formă de sfere plasate la distanțe egale. Utilizatorul poate urmări traiectoria definită de ancorele vizuale plasate în lumea reală împreună cu instrucțiunile care se actualizează la fiecare pas pentru a ajunge la destinație. Apăsarea butonului „Map View” determină întoarcerea la vizualizarea standard a hărții.

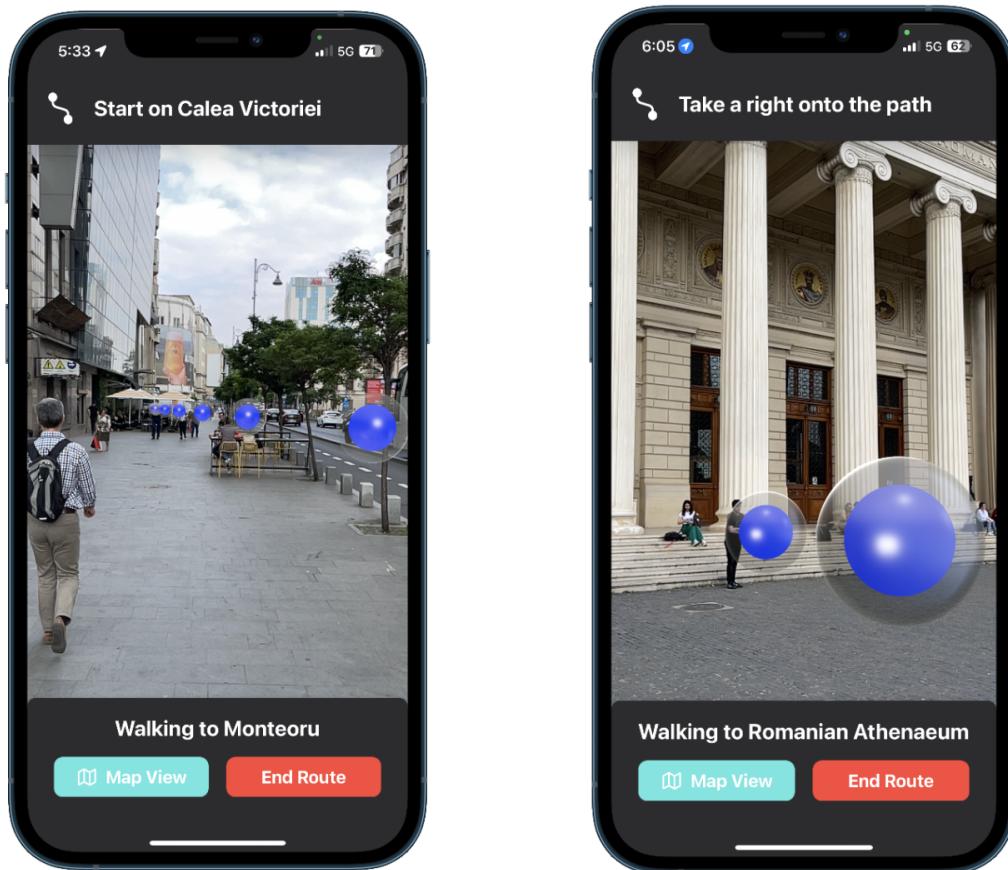


Figura 4.14: Urmarea instrucțiunilor de mers folosind realitatea virtuală

Capitolul 5

Concluzii

Prezenta lucrare descrie ideea, utilitatea, modul de implementare și funcționalitatea aplicației „TakeMeOut”. Concepță pentru utilizatorii de dispozitive iOS, aceasta propune un mod mai simplu și mai eficient de salvare a locațiilor de interes, împreună cu orientarea specifică aplicațiilor de hărți existente, la care se adaugă realitatea augmentată pentru o experiență cât mai palpabilă și un plus de siguranță.

5.1 Impresii și provocări întâmpinate

Dezvoltarea de aplicații iOS este un subdomeniu pe care am vrut de mult să îl explorez. Fără vreun fel de cunoștințe anterioare despre Swift sau Objective-C, am reușit să duc la bun sfârșit o aplicație complexă, ce folosește o întreagă suită de biblioteci și concepte avansate ale limbajului de programare. La acestea se adaugă suportul redus pentru acest limbaj de programare, întrucât este relativ nou în comparație cu cele consacrate și se modifică considerabil cu fiecare versiune, deoarece Apple actualizează frecvent bibliotecile astfel încât să pună la dispoziție dezvoltatorilor cele mai noi funcționalități.

Cea mai mare provocare a fost partea de afișare a instrucțiunilor specifice direcțiilor de mers, în special a celor ce folosesc realitatea augmentată. A fost nevoie de zile întregi de trial and error cu diferite computații pentru a putea plasa ancorele vizuale corect, la care se adaugă testarea în diferite medii și condiții, fiind genul de aplicație pe care nu o poți testa integral din confortul biroului de acasă. Așa cum sugerează și numele aplicației, am fost scos din casă cu multiple ocazii pentru a îmbunătăți funcționalitatea acesteia.

La final, satisfacția vine când te plimbi pe stradă și poți folosi propria aplicație pe telefon pentru a primi direcții de mers către un anumit loc, ca mai apoi să testezi funcționalitățile AR.

5.2 Perspective

Cu toate că aplicația își atinge scopul și poate fi considerată un produs finit, există câteva aspecte care ar îmbunătăți semnificativ experiența de utilizare. Acestea cuprind:

- **Îmbunătățirea preciziei cu care sunt plasate ancorele în spațiul virtual** - datorită serviciilor GPS care stau la baza localizării utilizatorului și a altor condiții care pot afecta detecția configurației AR, există șanse ca indicațiile vizuale să nu fie afișate în concordanță cu direcția de mers; însă până la îmbunătățirea serviciilor, nu putem decât să instruim utilizatorul să evite folosirea acestei funcționalități când condițiile nu sunt prielnice (lumină slabă, semnal GPS slab);
- **Recalcularea rutei în cazul unei erori din partea utilizatorului** - momentan, dacă utilizatorul se abate de la ruta stabilită, aplicația nu reușește să recalculeze ruta pentru a redresa utilizatorul, din motiv că un astfel de request este destul de costisitor și reduce considerabil performanța aplicației;
- **Trimiterea notificărilor și după ce aplicația este oprită** - acest feature presupune rularea unui serviciu care face aceeași verificare pe care o face și aplicația la trimiterea notificărilor locale, însă pe un server care să ruleze serviciul respectiv constant;
- **Recomandarea unor puncte de interes pe baza celor salvate** - aplicația ar putea trimite ocazional, sub formă de notificări, sugestii de locații noi, asemănătoare celor salvate, pe baza recenziilor din partea altor utilizatori;
- **Un range maxim personalizabil** - ar presupune ca utilizatorii să își configureze propriul range în care un punct de interes să fie considerat în apropiere și pentru care să primească notificări și direcții de mers;
- **Pe partea de UI/UX**, aplicația ar putea veni în viitor cu animații mai sugestive și suport pentru vizualizarea în landscape; de asemenea, ar putea fi adăugat suport pentru iPadOS și watchOS;
- În ceea ce privește **suportul utilizatorilor**, autentificarea poate fi îmbunătățită prin adăugarea unei confirmări adiționale a contului după crearea acestuia, dar și a unei opțiuni de resetare a parolei; de asemenea, pot fi adăugate opțiunile de autentificare cu conturile de Apple și Facebook, precum și salvarea credențialelor în Keychain.

Lista de mai sus demonstrează că aplicația „TakeMeOut” are multe direcții de dezvoltare și poate deveni alegerea principală a utilizatorilor când vine vorba de gestionarea punctelor de interes favorite și urmarea direcțiilor de mers către acestea.

Bibliografie

- [1] iOS Example, *A easy-to-use Floating Panel UI component for iOS*, 2018, URL: <https://iosexample.com/a-easy-to-use-floating-panel-ui-component-for-ios/> (accesat în 20.8.2018).
- [2] Foursquare, *Places API*, 2023, URL: <https://location.foursquare.com/developer/reference/places-api-overview> (accesat în 9.6.2023).
- [3] Maths at home, *How to Calculate Bearings*, 2023, URL: <https://mathsathome.com/calculating-bearings> (accesat în 9.6.2023).
- [4] Apple Inc., *ARKit*, 2023, URL: <https://developer.apple.com/documentation/arkit> (accesat în 7.6.2023).
- [5] Apple Inc., *ARKit ARConfiguration*, 2023, URL: <https://developer.apple.com/documentation/arkit/arconfiguration> (accesat în 7.6.2023).
- [6] Apple Inc., *ARKit ARPositionalTrackingConfiguration*, 2023, URL: <https://developer.apple.com/documentation/arkit/arpositionัltrackingconfiguration> (accesat în 7.6.2023).
- [7] Apple Inc., *ARKit ARSession*, 2023, URL: <https://developer.apple.com/documentation/arkit/arsession> (accesat în 7.6.2023).
- [8] Apple Inc., *ARKit ARWorldTrackingConfiguration*, 2023, URL: <https://developer.apple.com/documentation/arkit/arworldtrackingconfiguration> (accesat în 7.6.2023).
- [9] Apple Inc., *AVFoundation*, 2023, URL: <https://developer.apple.com/documentation/avfoundation> (accesat în 7.6.2023).
- [10] Apple Inc., *Core Data*, 2023, URL: <https://developer.apple.com/documentation/coredata> (accesat în 7.6.2023).
- [11] Apple Inc., *Core Data NSFetchedRequest*, 2023, URL: <https://developer.apple.com/documentation/coredata/nsfetchedrequest> (accesat în 7.6.2023).
- [12] Apple Inc., *Core Data NSManagedObjectContext*, 2023, URL: <https://developer.apple.com/documentation/coredata/nsmanagedobjectcontext> (accesat în 7.6.2023).

- [13] Apple Inc., *Core Location*, 2023, URL: <https://developer.apple.com/documentation/corelocation> (accesat în 7.6.2023).
- [14] Apple Inc., *MapKit*, 2023, URL: <https://developer.apple.com/documentation/mapkit> (accesat în 7.6.2023).
- [15] Apple Inc., *MapKit MapView*, 2023, URL: <https://developer.apple.com/documentation/mapkit/mkmapview> (accesat în 7.6.2023).
- [16] Apple Inc., *MapKit MKMapView*, 2023, URL: <https://developer.apple.com/documentation/mapkit/mkmapcamera> (accesat în 7.6.2023).
- [17] Apple Inc., *MapKit MKRoute*, 2023, URL: <https://developer.apple.com/documentation/mapkit/mkroute> (accesat în 7.6.2023).
- [18] Apple Inc., *RealityKit*, 2023, URL: <https://developer.apple.com/documentation/realitykit> (accesat în 7.6.2023).
- [19] Apple Inc., *RealityKit AnchorEntity*, 2023, URL: <https://developer.apple.com/documentation/realitykit/anchorentity> (accesat în 7.6.2023).
- [20] Apple Inc., *RealityKit ARView*, 2023, URL: <https://developer.apple.com/documentation/realitykit/arview> (accesat în 7.6.2023).
- [21] Apple Inc., *Swift*, 2023, URL: <https://www.apple.com/swift/> (accesat în 7.6.2023).
- [22] Apple Inc., *UIKit*, 2023, URL: <https://developer.apple.com/documentation/uikit> (accesat în 7.6.2023).
- [23] Apple Inc., *User Notifications*, 2023, URL: <https://developer.apple.com/documentation/usernotifications> (accesat în 7.6.2023).
- [24] Bacancy Technology, *Flutter Vs Swift: Which One Will Rule The Mobile App Development Market?*, 2023, URL: <https://www.bacancytechnology.com/blog/flutter-vs-swift> (accesat în 7.6.2023).
- [25] Wallace Wang, „Understanding iOS Programming”, în *Beginning iPhone Development with Swift 5* (2019), pp. 1–20, DOI: [10.1007/978-1-4842-4865-2_1](https://doi.org/10.1007/978-1-4842-4865-2_1).
- [26] Wikipedia, *Core Data*, 2023, URL: https://en.wikipedia.org/wiki/Core_Data (accesat în 7.6.2023).
- [27] Wikipedia, *Haversine formula*, 2023, URL: https://en.wikipedia.org/wiki/Haversine_formula (accesat în 9.6.2023).