

UNIVERSITATEA POLITEHNICA DIN BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL DE CALCULATOARE



PROIECT DE DIPLOMĂ

Aplicație mobile de recomandări și rezervări pentru
localurile unui oraș
"Uerto"

David-Gabriel Dunică

Coordonator științific:

As. Drd.Ing. DANIEL CALIN POPEANGA

Prof. Dr. Ing. ALEXANDRU BOICEA

BUCUREŞTI

2023

CUPRINS

1	Introducere	1
1.1	Context	1
1.2	Ipoteze	1
1.3	Obiective	1
1.4	Soluția propusă	1
1.5	Rezultatele obținute	2
1.6	Structura lucrării	2
2	Analiza Cerințelor	3
2.1	Punct de plecare	3
2.2	Descriere funcționalități	4
3	Studiu de Piață	5
3.1	Prezentarea problemelor	5
3.2	Aplicații similare	6
3.3	Analiza partilor interesante și SWOT	7
3.4	Tehnologii utilizate	8
4	Soluția Propusă	10
4.1	Arhitectura Aplicației	10
4.2	Backend	11
4.2.1	Baza de date	11
4.2.2	RestApi	15
4.2.3	Firebase Auth și Elemente de Securitate	19
4.2.4	Testare	20
4.3	Frontend	22

4.3.1	Flutter	22
4.3.2	Interfața Grafică și Experiența Utilizatorului	23
4.3.3	Componentele Aplicației	26
4.3.4	State Management în Redux	28
4.3.5	Google Maps API	30
5	Detalii de implementare	32
5.1	Algoritmii de Recomandări - Machine learning	32
5.1.1	Content Based Filtering	33
5.1.2	Collaborative Filtering	33
5.1.3	Regresie Liniara cu Coborâre în Gradient	34
5.2	Algoritm de Rezervări	34
6	Evaluare	37
6.1	Obiective	37
6.2	Utilitate	37
6.3	Performante	37
6.4	Îmbunătățiri ulterioare	38
7	Concluzii	39
	Bibliografie	41
	Anexe	42

SINOPSIS

[RO] Tema abordată în cadrul proiectului de diplomă este realizarea unei aplicații mobile de rezervări și recomandări a localurilor unui oraș. Punctul de plecare a fost digitalizarea rezervărilor. În momentul de față acesta reprezintă un impediment, deoarece multe localuri încă apelează la varianta telefonică, iar variantele digitale nu sunt automatizate complet. Scopul aplicației dezvoltate este de a diminua această problemă, grupând localurile după categorii, astfel această nișă va fi mult mai organizată. Dorința este de a simplifica mecanismul de căutare pentru utilizatori.

Pentru a îndeplini acest scop am realizat o aplicație mobilă care are ca principale funcționalități un sistem de căutare avansată, un model de machine-learning pentru recomandări și un algoritm pentru gestionarea rezervărilor.

Scopul îndeplinit în cadrul dezvoltării este să obțin o aplicație funcțională și performantă, ceea ce s-a și realizat. Această versiune este completă, fiind pregătită pentru deployment, însă trebuie luate în seamă și îmbunătățirile pe care doresc să le aduc în viitor, pentru ca funcționarea să fie optimă.

[EN] The topic addressed in the diploma project is the development of a mobile application for reservations and recommendations of places in a city. The starting point was the digitization of reservations. Currently, this represents a hindrance as many establishments still rely on phone bookings, and digital options are not fully automated. The goal of the developed application is to alleviate this problem by categorizing the venues and making this niche more organized. The desire is to simplify the search mechanism for users.

To achieve this goal, I have created a mobile application that includes key functionalities such as an advanced search system, a machine learning model for recommendations, and an algorithm for managing reservations.

The achieved objective in the development process is to obtain a functional and high-performance application, which has been successfully accomplished. This version is complete and ready for deployment. However, I also need to consider the future improvements that I intend to implement to ensure optimal functioning.

1 INTRODUCERE

1.1 Context

Din perspectiva proiectului meu de licență am dezvoltat un sistem automat de rezervări și recomandări pentru localurile unui oraș. Acest proiect poate avea o amplitudine foarte mare, deoarece se va axa pe o nișă de clienți, care a suferit foarte multe instabilități în ultimii ani, iar clientela clasică tende să fluctueze. De asemenea, sistemul de recomandări pentru localuri este un element inovator, fiind primul tip de acest fel de pe piață. Digitalizarea forțată a tuturor industrilor este un alt factor care stă la baza implementării mele.

1.2 Ipoteze

Punctul de plecare l-a constituit problema rezervărilor telefonice, care de cele mai multe ori crează un disconfort utilizatorilor, fiind prezente multe cazuri de neînțelegere între interlocutor și locutor. O altă problemă este lipsa de alternative și dificultatea cu care oamenii își găsesc noi locuri în care să își petreacă timpul liber. Majoritatea localurilor au o expunere redusă pe plan local, fiind afaceri familiale, care de cele mai multe ori nu au o clientelă pe măsura serviciilor.

1.3 Obiective

Obiectivul principal este de a dezvolta o platformă online 100% funcționabilă, care să deservească diminuării problemelor expuse. La finalul dezvoltării rezultatul obținut trebuie să fie o aplicație care rezolvă problema digitalizării rezervărilor și prezintă un sistem de recomandări care să îndrume utilizatorii. Intenționez ca platforma să aibă o interfață ușor de utilizat pentru a fi servit tuturor categoriilor de utilizatori. Un obiectiv secund, este ca aplicația să aibă o expunere sporită, astfel trebuie implementată pe mai multe dispozitive sau sisteme de operare.

1.4 Soluția propusă

Astfel, soluția pe care am găsit-o este dezvoltarea unei aplicații mobile, pentru ambele platforme Android și iOS. Pentru a digitaliza sistemul de rezervări, am gândit un backend și o bază de date care automatizează acest proces în totalitate. Mai mult, pentru sistemul de

recomandări, am venit cu o soluție formată dintr-un automat cu trei rutine de predicție, fiind vorba de trei algoritmi de machine learning, specializați în sistemele de recomandări. Această aplicație are funcționalitățile unei aplicații moderne, fiind vorba de autentificare și măsuri împotriva intruziunilor sau atacurilor.

1.5 Rezultatele obținute

Din punct de vedere al rezultatelor, aplicația funcționează exact ca în planul inițial. Am reușit să implementez atât sistemul de machine learning, cât și să găsesc o soluție pentru un algoritm de rezervări, cu ajutorul căruia am digitalizat complet acest proces. Atât backendul cât și frontendul funcționează în marametrii, având performanțe semnificative. Design-ul aplicației este unul intuitiv și simplist, fiind lipsit de bug-uri, iar latența provenită din conectivitatea la server este în normele acceptate.

1.6 Structura lucrării

În cele ce urmează voi prezenta structura lucrării, pentru o parcursere facilă a lucrării:

- Capitolul **“Analiza cerințelor”** cuprinde două subcapitole cu prezentarea problemei detaliate de la care am plecat, iar apoi, am explicitat lista funcționalităților aplicației dezvoltate, punând în balanță ideea inițială și rezultatele obținute.
- Capitolul **“Studiu de piață”** are ca scop analiza completă și complexă a factorilor de pot influența importanța acestui proiect. Capitolul începe cu identificarea și detalierea problemelor, fiind găsite soluții pentru acestea, ca mai apoi să fie analizate aplicațiile prezente pe piață, cu avantaje și dezavantaje. În cele ce urmează am prezentat o analiză a partilor interesante și o analiza SWOT, ca în final să prezint tehnologiile utilizate în cadrul dezvoltării.
- Capitolul **“Soluție propusa”** este divizat în trei. În prima parte este prezentată întreaga arhitectură a aplicației, ca următoarele două parti să prezinte pe rand partea de backend și de frontend. În acest capitol se găsesc dezvoltate ideile puse în aplicare, fiind un contrast între limbaj uzual și limbaj tehnic, folosind terminologia adecvată acestui domeniu.
- Capitolul **“Detalii de implementare”** prezintă pe rand algoritmii de recomandări și cel pentru rezervări.
- Capitolul **“Evaluare”** pune accentul pe rezultatele obținute, unde intervine spiritul autocritic, fiind evaluate îndeplinirea obiectivelor, utilității și a performanțelor, plus o secțiune de îmbunătățiri.
- Capitolul **“Concluzii”** încheie lucrarea printr-un rezumat ce sintetizează toate ideile importante ale aplicației.

2 ANALIZA CERINȚELOR

2.1 Punct de plecare

Recreerea reprezintă principalul factor de socializare interumană. Într-o lume în care accentul tehnologiei ne împinge din ce în ce mai mult să ne îndepărtem de metodele de socializare clasice, trebuie luată în calcul o reexaminare atentă a comportamentelor umane. Având în vedere că omul, prin natura lui, este o ființă care trăiește în comunitate, trebuie dezvoltate metode prin care această puncte dintre oameni să fie întărită, ci nu diluată sau, mai rău, distrusă. Mai mult pandemia din anii 2020 și 2021 a destabilizat și mai tare tendințele oamenilor de a ieși din casă cu scop recreativ, așa cum este prezentat și în studiul Ipsos [7], realizat anul trecut în Uniunea Europeană. În urma acestui studiu s-a dovedit că oamenii ies într-o proporție de 63% mai puțin la restaurant. Acest fapt se poate întâmplă din două cauze principale. Prima este reprezentată de comoditatea oamenilor de a rămâne acasă, iar cea de-a doua este reprezentată de prețurile care sunt din ce în ce mai mari, astfel ieșirile în oraș nu mai reprezintă o prioritate pentru utilizatori. Din această deducere, reiese faptul că nevoie de un sistem sau o aplicație care să vină în ajutorul utilizatorilor; această aplicație trebuie să ajute prin nenumărate metode de filtrare și chiar un sistem automat de recomandări, care să ajute oamenii în găsirea unor localuri pe placul acestora.

De asemenea, avansurile tehnologice resimțite în viațile oamenilor din ultimii ani, au avut ca efect secundar și accelerarea proceselor umane. Astfel trebuie luat în seamă și inconvenientul rezervărilor telefonice. Acestea trebuie automatizate, pentru a se alinia la standardele epocii. Chiar și sistemele de rezervări actuale nu îndeplinesc toate cerințele, deoarece sunt parțial automatizate: nu dispun de un sistem de disponibilități avansat, fiind nevoie de un operator uman care să sorteze și să manevreze aceste cereri.

Consider că la momentul actual aplicațiile prezente pe piață nu ating esențialul problemei, deoarece algoritmul de rezervări nu este complet automatizat. De asemenea, nicio aplicație nu vine în ajutorul utilizatorului pentru a prezența localurile într-un mod mai eficient, pentru a simplifica procesul de alegere. Totodată, la momentul actual nu există niciun algoritm de recoamandări pentru localuri.

Cele de sus fiind spuse, intenția proiectului meu este de a rezolva aceste două probleme primordiale, deoarece în urma studiilor precizate există o tendință destul de abruptă în cele ce înseamnă petrecutul timpului liber în afara propriei incinte, încercând astfel ameliorarea acestui fenomen post pandemic.

2.2 Descriere funcționalități

Din punct de vedere al funcționalităților, aplicația are sistem de autentificare, care poate menține utilizatorul logat, astfel nu este nevoie să logarea la fiecare intrare în aplicație. Totaodata există elemente de securitate, precum resetarea parolei prin intermediul email-ului și la înregistrare email-ul este verificat prin un email de validare. Odată ce utilizatorul este autentificat acesta poate să caute localuri atât prin intermediul sistemului de recomandări, cât și într-un mod manual, introducând numele în bara de search, sau printr-o căutare filtrată, ce constă în atributele unui local, distantă, zonă, rating, etc. După o astfel de căutare, localurile apar fie într-o listă, fie pe hartă, depinzând de preferință utilizatorului. La selectarea unui local, utilizatorul poate să vadă toate detaliile aferente și lista de activități la care se poate programa prin intermediul aplicației. Pentru a face o programare se selectează data și numărul de participanți, iar aplicația redă toate intervalele de disponibilitate. Odată ce o rezervare este realizată aceasta poate fi regăsită în secțiunea de rezervări a profilului, unde utilizatorul poate să o anuleze, iar cu o oră înainte de rezervare aplicația emite o notificare cu scopul de reamintire. Alte funcționalități introduse sunt: editarea profilului, posibilitatea de evaluare (rating) după o rezervare și categoria de favorite, unde utilizatorul poate adăuga localurile preferate.

Aceste funcționalități prezentate mai sus reprezintă punctul de plecare, pe care l-am urmat în dezvoltarea aplicației mele. Am considerat că aceste funcționalități se împart în trei categorii, fiecare reprezentând o gamă de probleme pe care le preîntâmpină.

Prima categorie este reprezentată de "Securitate". Aceasta are ca scop sporirea încrederii și totodată a profesionalismului pe care aplicația trebuie să îl ofere. Elemente precum autentificare, autorizare, modificarea parolei și a datelor utilizatorului conferă încredere. Astfel aliniem aplicația la cerințele moderne, fiind asemănătoare din acest punct de vedere cu celelalte aplicații. De asemenea, la acest capitol nu este recomandată vreo inovație, deoarece se pot ridica semne de întrebare; Alinarea cu alte aplicații din punct de vedere al autentificării oferă utilizatorilor o ușurință în manipularea aplicației.

Cel de-al doilea set de funcționalități este denumit "funcționalități principale". Acestea reprezintă funcționalitățile cheie ale aplicației prin care se deosebește de alte aplicații: sistemul de recomandări, algorimul de disponibilități, sistemul de rezervări automatizat și metodele de filtrare și căutare ale localurilor. Așa cum am spus scopul este strict funcțional și oferă utilizatorilor o alternativă performantă față de alți competitori pe piață.

În cele din urmă ultima gamă de funcționalități este reprezentată de cele "secundare". Printre ele se enumera și: integrarea Google Maps, posibilitatea de rating, drawer, animații. Acestea au scopul de a amplifica experiența utilizatorului într-un mod pozitiv. Este format din funcționalități diverse, minore, care completează ansamblul aplicației. De asemenea, designul simplist și modern este o funcționalitate secundară, deoarece are scopul dezvoltării experienței userilor.

3 STUDIU DE PIATĂ

Pentru a începe un studiu de piată trebuie să expun inițial problemele pe care intenționez să le rezolv prin intermediul acestei aplicații, ca mai apoi să evidențiez avantajele și dezavantajele principalilor competitori prezenți pe această nișă. La momentul de față am evidențiat patru mari probleme la care clientii sunt expuși dinainte ca aceștia să meargă fizic în locul respectiv.

3.1 Prezentarea problemelor

În primul rând este vorba de neînțelegерile bazate pe rezervările telefonice. Cea mai mare parte a rezervărilor se fac prin intermediu telefonic. Clientul trebuie să găsească pe internet numărul de telefon apoi să sune pentru a își face o rezervare. Prima problemă care poate să intervină este respingerea apelului telefonic de către local, din diferite motive precum neatenția angajațiilor. O altă problema este reprezentată de erorile umane care pot apărea: angajatul poate să confundă rezervarea și să o schimbe.

Prin intermediul aplicației noastre acest intermediar va fi înlocuit cu sistemul de rezervări integrat. Așadar erorile umane vor dispărea, iar mai mult de atât acest inconvenient reprezentat de apelul telefonic va fi de asemenea, înălăturat. O altă funcționalitate care vine în ajutorul utilizatorului este posibilitatea de a vizualiza toate intervalele disponibile la care se poate face rezervarea; acest lucru este în avantajul clientilor, deoarece în timpul unui apel telefonic de cele mai multe ori angajatul nu specifică toate intervalele la care se pot rezerva clientii, ci doar intervalele care ii avantajează.

Neîncrederea cauzată de site-urile actuale de rezervări online este cel de-al doilea inconvenient gasit. Aceasta este o altă problemă recurrentă în rândul rezervărilor online este necredibilitatea clientilor în site-urile actuale de rezervări. În momentul de față o mare parte din localurile cu un sistem de rezervări online își folosesc propriul website unde și-au dezvoltat un astfel de mecanism. Aici am întâlnit două mari probleme: prima este reprezentată de interfață grafică, care crează ambiguitate; multe din regulile de UI nu sunt respectate, iar la finalul rezervării aceste site-uri nu îți conferă o maximă credibilitate că rezervarea a avut loc. Cea de-a doua problema este nestandardizarea acestui mecanism. Utilizatorii trebuie să se adapteze, deoarece fiecare local are propriul model de rezervări online. Aplicația dezvoltată de mine oferă un standard, cu o interfață inteligență care reduce acesta lipsa de încredere în rezervările online.

În al treilea rând, lipsa de expunere a multor localuri din categoria de divertisment sau relaxare reprezintă o oportunitate pentru proiectul meu. Am sesizat de ceva timp că majoritatea arenelor sportive sau a locurilor de divertisment au o expunere foarte mică în ceea ce privește

marketingul. De cele mai multe ori acestea se bazează pe rețelele de socializare pentru expunere; Așadar aplicația noastă inovează față de alte aplicații asemănătoare prin introducerea acestei categorii de clientela. De asemenea, sistemul de recomandări al aplicației este antrenat să ofere utilizatorului alternative bazate pe preferințele sale în materie de relaxare, astfel automat aplicația va reprezenta o poartă de acces a unei cliente noi pentru acest tip de localuri.

Ultima problema, este reprezentată de lipsa unui îndrumător în ceea ce privește deciziile oamenilor atunci când doresc căutarea unui local. Acest element este inovativ, fiind implementat în cadrul acestei aplicații, cu scopul de a ajuta utilizatorul în găsirea localurilor care i se potrivesc, reducând astfel și timpul de căutare și decizei.

3.2 Aplicații similare

La momentul de față există pe piață din România trei aplicații asemănătoare de rezervări online, fiecare fiind axată pe o categorie de localuri; Niciuna dintre acestea nu a introdus un sistem de recomandări, având unicul scop principal de a face rezervări.

- **Stailer**

Stailer este o aplicație mobile care facilitează rezervarea la saloane de înfrumusețare. Aceasta este un proiect românesc apărut în urmă cu 3 ani, care își desfășoară momentan activitatea în marile orașe din București și are peste 50.000 de descărcări. Această aplicație le permite utilizatorilor să descopere saloane de înfrumusețare în orașul selectat, având posibilitatea de a își face rezervări online. Principalul punct forte al acestei aplicații este faptul că utilizatorul poate să caute stilistul dorit după nume, deși această poate să creeze confuzie, deoarece șansele că un nume să se repete sunt foarte mari. O altă particularitate a acestei aplicații este catalogul de inspirație, unde stilistii pot să posteze poze cu serviciile oferite. Punctele slabe ale acestei aplicații sunt faptul că nu poți vedea saloanele din imprejurime, ceea ce într-un oraș precum București este necesar; utilizatorul nu poate vedea istoricul rezervărilor, iar interfață cu utilizatorul este neprofesionistă, ceea ce crează neîncredere.

- **Bookingham**

Bookingham este o aplicație axată pe rezervări online pentru nișă restaurantelor. Este o aplicație destul de recent apărută care nu are o popularitate prea mare, având mai puțin de 1000 de descărcări. Avantajul esențial al acestei aplicații este reprezentat de multitudinea de categorii care definesc un restaurant, astfel sunt satisfăcute toate categoriile de utilizatori. Părtiile slabe ale acestei aplicații sunt faptul că nu pot căuta localurile după distanță față de utilizator, care din nou relevă un dezavantaj pentru utilizatorii care se află în metropole, și faptul că aplicația pune accentul doar pe restaurante; cafenelele și barurile sunt neglijate,

aliciatia prezentând restaurantele într-un mod destul de neconvenient, și anume într-o lista; astfel utilizatorul trebuie să caute prin acea lista localul potrivit, ceea de duce la pierderea interesului destul de rapid.

- **Ialoc**

Ialoc este o altă aplicație asemănătoare cu Bookingham, dar cu un număr de descarcări mult mai mare; a fost descărcată de peste 10000 de ori. Această aplicație are marele avantaj că utilizatorul poate seta aria dintr-un oraș în care să caute localele și chiar le poate vedea pe harta. Cu toate acestea aplicația Ialoc duce lipsa de o filtrare bazată pe categoria de restaurant; acestea sunt prezentate într-o lista, fapt care poate duce rapid la pierderea interesului utilizatorului. Totodată designul interfeței grafice lasă de dorit, fiind destul de neintuitiv și învechită.

Concluzia principală trasă din analizarea celorlalte aplicații asemănătoare de pe piață este că niciuna nu pune accentul pe interacțiunea cu utilizatorului; toate prezintă localurile într-un mod nestructurat și aleator. Aceste aplicații nu vin în sprijinul utilizatorului cu variante sau soluții. Consider că aplicația mea prin sistemul de recomandări va putea sparge această barieră. O altă remarcă este faptul că în continuare nu există o aplicație pe care se pot înscrie localuri precum escape-room, terenuri sportive, etc. Mai mult consider că nu este necesar ca fiecare categorie să aibă propria aplicație: Staier pentru saloane, Ialoc și Bookingham pentru restaurante; toate acestea pot face parte dintr-o singură aplicație, în care să se respecte niște reguri de UI pentru a nu confuză utilizatorul. Astfel mai multe categorii de localuri pot facilita de aceste servicii de automatizare a rezervărilor având totodată prilejul de a se promova prin intermediul acestei aplicații.

3.3 Analiza partilor interesate și SWOT

În cele ce urmează putem cataloga nevoia unei astfel de aplicații printr-o analiză a părților interesate, această fiind detaliată în Tabela 1. Mai mult pentru o examinare în ansamblu, care implică interacțiunea cu mediul, am realizat o analiză SWOT, ce expune punctele tari, dar și cele slabe, oportunitățile și riscurile la care se expune aplicația. Această este realizată în Tabela 3, sub capitolul Anexe.

Tabela 1: Analiza partilor interesatei

Parti interesate
Beneficiarii: Aceștia sunt reprezentați de persoane juridice, localurile prezente pe aplicație. Proiectul meu le oferă acestora serviciul de digitalizare al rezervărilor, și indirect publicitate, aducând constant clientela nouă.
Utilizatorii: Aplicația vine în ajutorul acestora prin două metode: prima este reprezentată de sistem de recomandări, care personalizează toate căutările pe baza profilului utilizatorului; iar cea de-a doua se referă la simplificarea procesului prin care aceștia își fac o rezervare.
Investitorii: Rezultatele financiare ale acestui proiect depind în mod direct de costurile asociate producției. Având în vedere faptul că această aplicație necesită costuri minime de dezvoltare și mențenanță, fiind vorba de un proiect IT, atragerea unor investitori poate fi facilitată de riscul mic la care aceștia se expun.
Angajații: Un astfel de proiect necesită și operare umană: astfel această aplicație poate genera noi locuri de muncă: în primul rând pentru a preveni propagarea unui bug este nevoie de un serviciu de call-center unde utilizatorii să apeleze în caz de probleme, iar în al doilea rând pentru a extinde proiectul este nevoie de angajați în domeniul IT.

3.4 Tehnologii utilizate

În ceea ce privește o aplicație mobile trebuie luate în considerare mai multe nivele tehnologice. Aceste nivele formează conceptul de full-stack, reprezentant o stivă a tehnologiilor utilizate în cadrul proiectului. Putem segregă această stivă în două categorii: partea de front-end a aplicației, reprezentată de interfață grafică ce se poate descărca pe mobil și partea de back-end ce constă în principal în baza de date, API-uri, business logic și securitate.

Pentru realizarea acestei aplicații am folosit pentru partea de front-end sdk-ul celor de la Google: Flutter. Motivul principal pentru care am utilizat acest framework în defavoarea dezvoltării native a aplicațiilor de mobile, deși acestea sunt ceva mai performante din punct de vedere al resurselor utilizate, este faptul că am înjumătățit timpul de realizare al părții de frontend. Având în vedere că piată sistemelor de operare pentru mobil este segregată în principal între iOS și Android, acest fapt însemna că pentru frontend era necesar să îmi aloc un timp cel puțin dublu de dezvoltare, deoarece trebuia să creez două aplicații independente pentru fiecare OS-uri. Flutter, precum și alte frameworkuri ca React Native sau Xamarin, facilitează dezvoltarea unei singure aplicații care poate fi rulată pe o multitudine de platforme, printre care și cele două menționate mai sus.

Am favorizat utilizarea sdk-ului Flutter, față de React Native, deoarece este o tehnologie în continuă dezvoltare, având o popularitate din ce în ce mai mare în ultimii 3 ani, devenind în momentul de față a două cea mai utilizată tehnologie de frontend; aşadar există o comunitate solidă care contribuie la extinderea bibliotecilor și funcționalităților deja existente. Pe de altă parte, resursele utilizate de dispozitive sunt mai reduse pentru aplicațiile realizate în Flutter, față de cele realizate în React. O astfel de comparație se poate observă în Tabela 2, extras

dintr-o publicație a siteului [11].

Tabela 2: Benchmark Native vs React vs Flutter

Framework	FPS	CPU[%]	Memory[MB]	Battery[mAh]
Native	60	2.4	58	49.7
React Native	58	11.7	139	79.01
Flutter	60	5.4	114	65.28

Se poate deduce din tabelul prezentat mai sus că pentru același număr de FPS, o aplicație realizată pe Flutter folosește jumătate din resursele procesorului utilizate de aceeași aplicație realizată cu React; memoria alocată cât și bateria utilizată fiind totodată mai mici.

Pe partea de management a datelor în aplicație am utilizat bibliotecile Redux prezente în Flutter. Cu ajutorul acestui tool am putut să gestionez mult mai ușor stările prin care trece aplicația, având datele într-un store, global, care poate fi accesat de oriunde din widget-tree.

În ceea ce privește partea de back-end am luat următoarele decizii: pentru baza de date am utilizat o baza de date relațională, în favoarea unei baze de date NoSQL, iar pentru business logic și API-uri am folosit mediul de execuție NodeJS, împreună cu framework-ul Express. Am considerat că o baza de date relațională este mai eficientă în proiectul meu pentru două motive principale; în primul rând, potrivit informațiilor expuse pe site-ul [2], acest tip de baze de date permite utilizarea de interogări complexe, un element esențial în realizarea sistemului de recomandări pe care intenționez să îl implementez. În cel de-al doilea rând, pentru a păstra o coerentă a datelor am nevoie că baza mea de date să aibă proprietățile ACID (atomicitate, consistentă, izolare și durabilitate) pentru a facilita utilizarea tranzacțiilor. De exemplu, pentru o rezervare în aplicație trebuie utilizate o tranzacție formată din mai multe query-uri de tip INSERT ; dacă din motive neprevăzute una din operații eşuează, celelalte automat trebuie anulate, pentru a păstra o consistentă a datelor.

Ca sistem de gestionare al bazei de date am utilizat MySQL în prima instanță datorită faptului că este o tehnologie open-source și totodată compatibilă cu mai multe sisteme de operare. Un mare avantaj al MySQL, remarcat pe [2], este viteză și scalabilitatea de care dispune. Astfel deși o baza NoSQL este scalabilă orizontal, acest SGBD-ul poate facilita o scalabilitate bazată pe clusteri.

Pe partea de dezvoltare a backend-ului am utilizat mediul de execuție NodeJS, în favoarea altor backend-uri precum Java sau PHP, în principal pentru vastă selecție de funcționalități și biblioteci de care dispune. Totodată, conform, NodeJS este o alegere populară în rândul unor mari companii globale, precum: Twitter, Netflix, PayPal sau LinkedIn, astfel se poate deduce capabilitatea acestei tehnologii. Biblioteca predominant utilizată în acest proiect este Express; această framework contribuie prin managementul cererilor HTTP, fiind utilizat propriu zis pentru crearea API-urilor.

4 SOLUȚIA PROPUȘĂ

4.1 Arhitectura Aplicației

Arhitectura acestei aplicații mobile este formată din două componente principale, și anume: partea de frontend, care este responsabilă de contactul cu utilizatorul, fiind reprezentată de aplicația fizică, mobilă, disponibilă pe telefon; și partea de backend ce gestionează traficul de informații server-utilizator, stocarea , integritatea și securitatea datelor.

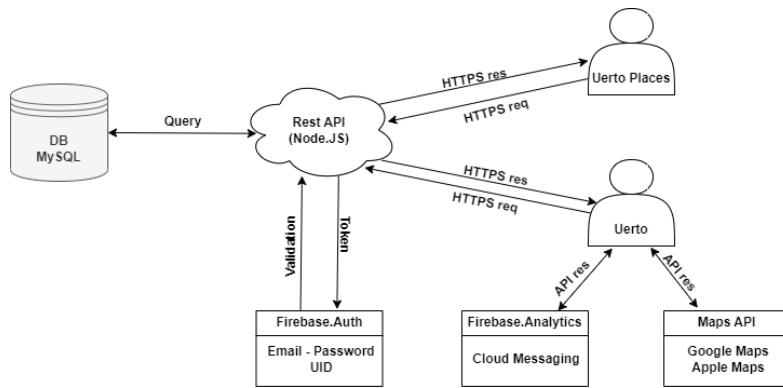


Figura 1: Diagrama sistemului

Așa cum se poate observa și în Figura 1 contactul dintre aplicația mobilă și server, reprezentat de RestAPI-ul realizat cu NodeJS se face prin protocolul HTTPS, cu ajutorul căruia datele sunt transferate pe server prin request în body sau parametrii, iar ca un call pe server să fie acceptat se trimit și refresh-tokenul utilizatorului care este verificat în Firebase.Auth, că mai apoi să se execute query-ul aferent operației, iar informațiile să fie furnizate înapoi către utilizator prin body-ul responsului HTTPS. Alte două API-uri utilizate direct din frontend sunt Api-ul Google Maps, astfel că localurile să poată fi disponibile pe harta implementată în aplicație; și API-ul Firebase.Analysis care conferă posibilitatea de notificări planificate, care să fie trimise către utilizatorui cu scop comercial.

4.2 Backend

4.2.1 Baza de date

SQL

Am considerat că o bază de date relațională este mai eficientă în proiectul meu pentru două motive principale; în primul rând, potrivit informațiilor expuse pe site-ul [2], acest tip de bază de date permite utilizarea de interogări complexe, un element esențial în realizarea sistemului de recomandări pe care intenționez să îl implementez. În cel de-al doilea rând, pentru a păstra o coerentă a datelor am nevoie ca baza mea de date să aibă proprietățile ACID (atomicitate, consistentă, izolare și durabilitate) pentru a facilita utilizarea tranzacțiilor. De exemplu, pentru o rezervare în aplicație trebuie utilizată o tranzacție formată din mai multe query-uri de tip INSERT ; dacă din motive neprevăzute una din operații eşuează, celelalte automat trebuie anulate, pentru a păstra o consistentă a datelor. Am împărțit diagrama bazei de date în două imagini, pentru a fi mai ușor de urmărit, astfel imaginea 3 reprezintă tabelele care stochează date referitoare la localuri, iar imaginea 2 cuprinde tabelele în care se găsesc informațiile despre utilizatori, și acțiunile acestora. Pentru a vedea întreaga bază de date se poate urmări Figura 11, ce se găsește în Anexe.

Pentru stocarea utilizatorilor am folosit tabela User care are mai multe atrbute de tip VARCHAR pentru date precum: nume, email, telefon, dar și idul de autentificare din Firebase și o variabilă de tip TINYINT numită nextStrategy, care contorizează următoarea strategie folosită în sistemul de recomandări. Tabela User_Rating este formată din relația many to many dintre User și Place și înregistrează ratingurile date de utilizatori după finalizarea unei rezervări. Tabela User_Rating_Require_Action stochează requesturi de evaluare a experienței, fiind tot formată din relația M:M dintre User și Reservation; înregistrările din această tabelă apar automat după finalizarea unei rezervări, și se sterg atunci când userul evaluează localului respectiv. Ultima tabelă din subcategoria user este User_Favourite_Place, care de asemenea, este formată din relația M:M dintre User și Place și care reprezintă localurile favorite ale utilizatorilor. Relațiile dintre aceste tabele se pot observa în Figura 2.

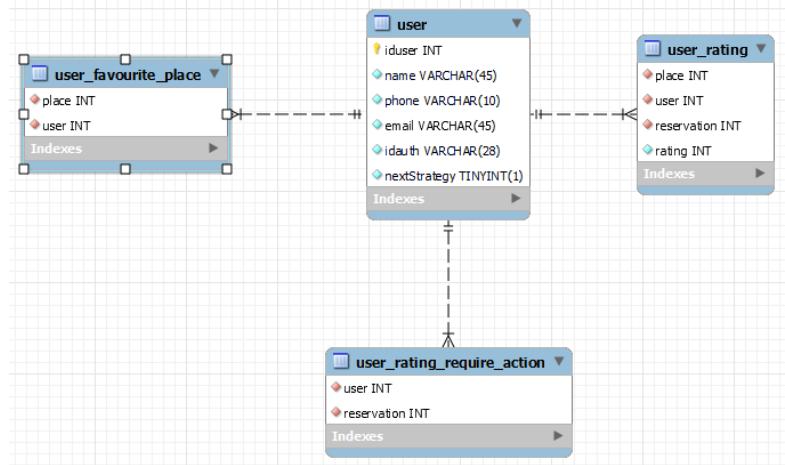


Figura 2: Schema tabelelor ce stocheaza datele utilizatorilor

Pentru stocarea localurilor am folosit tabela Place care stochează datele localurilor inclusiv coordonate geografice sau orele de operare. Tabelele Place_Filter_Restaurant și Place_Filter_Leasure au attribute TinyInt iar acestea iau valorile 1 sau 0 astfel formandu-se lista de atribute ce definesc localele respective. Tabela Activity înregistrează activitatiile localurilor, iar tabela Activity_Seating înregistrează toate subdiviziunile care se pot rezerva ale unei activități (de exemplu mese pentru activitatea “rezervă o masă” la un restaurant). Tabela Reservation stochează toate rezervările realizate pe aplicație, fiind M:M intre Place și User, iar statutul rezervării este dat de atributul status, care indică dacă o rezervare este în viitor, în curs sau este finalizată. Deoarece o rezervare poate să includă mai multe “Activity_Seating” am introdus și Tabela Activity_Arrangement care este formată din relația many to many dintre Activity_Seating și Reservation. Ultima tabela referitoare la localuri este Place_Blocked_Users care stochează utilizatorii blocați de localuri, în cazul în care acestia nu și-au onorat rezervările; de asemenea, această tabela este formata prin M:M intre User și Place. Pentru a vedea relațiile dintre tabelele ce stochează datele localurilor se poate consulta Figura 3.

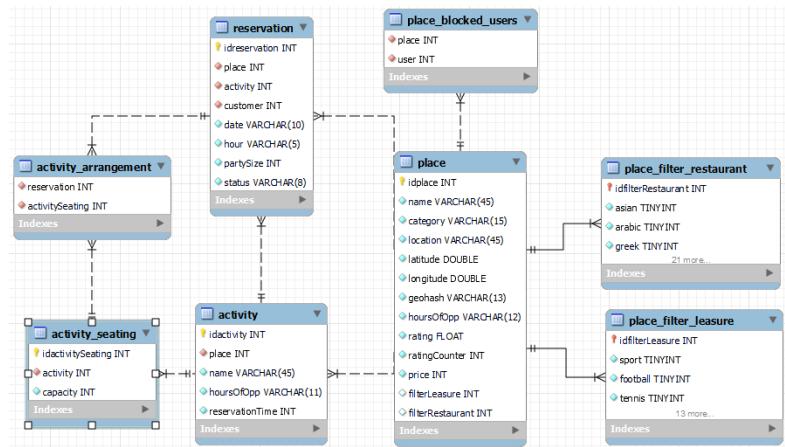


Figura 3: Schema tabelelor ce stocheaza datele localurilor

Utilizarea Constrângerilor și a Tranzacțiilor pentru Menținerea Consistenței

Pentru menținerea proprietăților ACID ale bazei de date, și în esență a întregii integrități a aplicației, am utilizat mecanisme de control pentru operațiile de write (INSERT, UPDATE, DELETE).

Constrângerile reprezintă primul nivel de securitate a datelor, astfel încât date ce nu respectă regulile de stocare nu pot fi salvate în baza de date, returnând erori în RestAPI. Principalele constrângeri utilizate au fost formate din cheie UNIQUE, ce redă faptul că intrările din acea coloană trebuie să fie unice; am utilizat această constrângere atât unic cât și pentru grupuri de attribute, de exemplu în tabela User_Rating_Require_Action am utilizat câmpul UNIQUE pentru perechea iduser și idreservation; FOREIGN KEY, ce este o cheie specifică bazelor de date SEQUEL ce stă la baza relațiilor dintre tabele. Astfel o intrare cu FOREIGN KEY reprezintă id-ul unei intrări din altă tabelă. Constanța NOT NULL este utilizată pentru a preciza că o intrare nu poate avea acel atribut null. De regulă am utilizat cheia primară ca fiind id-ul de identificare, ce are asignat și proprietatea auto-increment, astfel fiecare nouă intrare are un id unic și incrementat față de ultima intrare. Am utilizat valoare Default pentru predefinirea unei valori la crearea unei înregistrări în tabelă. De exemplu la crearea unui USER nextStrategy (ce reține idul următoarei rutine de căutare pentru algoritmul de recomandări) este initializat cu 0. Pentru a spori performanța de căutare pentru QUERYS am folosit indexarea tabelelor.

În general tabelele sunt indexate după id, care au și constrângerea de primary key, însă există și excepții cum ar fi tabela User-ilor care este căutată în general după două câmpuri: cheia primară – iduser sau după idauth, care este idul de autentificare provenit din Firebase, ce este de forma unui CHAR de 28 de caractere. Astfel am folosit două indexuri distincte, unul pentru iduser, iar celalalt pentru idauth.

Tranzacțiile reprezintă o secvență logică și atomică de operațiuni în limbaj SQL, în urma cărora se execută modificări asupra bazei de date. Acestea respectă proprietatiile ACID, specifice bazelor de date relationale, și anume: sunt atomice (toate operațiile trebuie să se execute cu succes), consistente (baza de date rămâne validă și consistentă după finalizarea tranzacției), izolate (de îndată ce o tranzacție începe, activitatea tabelelor este blocată) și durabile (tranzacția aduce modificări permanente).

În dezvoltarea aplicației am întâlnit mai multe cazuri prin care era necesară utilizarea de tranzacții pentru menținerea consistenței datelor. De exemplu atunci când se execută un request de PUT pentru o rezervare se întâmplă trei acțiuni: prima este de adăugare în tabela Reservation o nouă înregistrare. Apoi trebuie adăugată o nouă înregistrare în tabela Activity_Arrangement ce reprezintă masa asignată rezervării respective. În cele din urmă este nevoie să fie returnat și Id-ul noii înregistrări în Reservation, care este preluată cu ajutorul funcției LAST_INSERT_ID(). Pentru a nu avea „ghinionul” ca între queryuri să se înregistreze o nouă rezervare, am considerat util ca această instrucțiune să fie de asemenea, introdusă în

tranzacție, fiind utilă proprietatea de izolare. Deoarece cele trei acțiuni sunt dependente una de celalătă, eroarea uneia ar trebui să opreseaca executarea celorlalte două. Aici intervine tranzacția care asigură că datele se stochează în memorie doar dacă toate acțiunile sunt valide.

Tranzacțiile utilizează metodele : commit() și rollback(). Acțiunea de commit se execută după finalizarea tuturor operațiilor din tranzacție, fiind momentul în care se salvează toate modificările în baza de date. Rollback se execută când este întâmpinată o eroare, aducând baza de date la statusul ei dinainte de executarea tranzacției. Astfel este respectată consistența datelor.

Alte exemple de cazuri în care am utilizat tranzacții sunt: ștergerea unei rezervări, updatarea imaginilor unui local sau user, acțiunea de rating a unui local după o rezervare.

Stocarea resurselor

Pentru stocarea resurselor am considerat că stocarea lor în baza de date SQL este mult prea costisitoare. Astfel imaginile de profil ale utilizatorilor și imaginile localurilor sunt stocate în directorul „/images” de pe server. În baza de date am salvat calea către aceste imagini și numele fișierelor de format .jpeg.

Imaginiile userilor au fost încărcare în subdirectorul „/images/users”, iar pozele pentru localuri au fost introduse în directorul „/images/places”.

Pentru a păstra intimitatea utilizatorilor, am considerat că imaginile stocate pe server să poarte un nume criptat format dintr-un string de 13 caractere din intervalul [0, 9]; fiind format din data la momentul încărcării fișierului, plus extensia fișierului. Aceasta reprezintă un strat de protecție sporită, deoarece în cazul unui atac imaginile nu sunt denumite direct după utilizatorii care le-au încărcat, ci după data la care au fost postate în aplicație. Totodată se menține și unicitatea numerelor, fiind o variantă mai rapidă și necostisitoare de contorizare a numerelor fișierelor, fiindcă două imagini nu pot fi încărcate concomitent.

```
Date.now() + path.basename(file.originalname)
```

Pentru a afișa o poză de pe server se execută următoarele requesturi, în funcție dacă imaginea este a unui utilizator sau a unui local:

```
https://localhost:3000/images/profile-image/1686320187863.jpeg  
https://localhost:3000/images/places/1686590079326.jpg
```

4.2.2 RestApi

NodeJS

Partea de server ce constă în logica sa este realizată cu ajutorul framework-ului NodeJS. Am utilizat această tehnologie de backend datorită următoarelor avantaje prezentate și pe pagina oficială [3]: viteza de execuție, fiind unul din cele mai performante tehnologii pentru web la momentul actual; o altă calitate esențială pentru un server îndeplinită de NodeJS este scalabilitatea. De asemenea, fiind un framework foarte popular există o multitudine de resurse sub formă de biblioteci care pot să scurteze timpul de dezvoltare semnificativ. Am utilizat biblioteca "express" versiunea 4.18.2 pentru manevrarea mai simplă a requesturilor HTTPS, având o gamă largă de soluții pentru rutele "Absolute path"-ului din URL. Această bibliotecă permite utilizarea nenumarator funcții de tip middleware cu ajutorul cărora se pot introduce elemente de securitate sau se pot manipula erorile.

Pentru conectarea la baza de date am folosit biblioteca "mysql2" versiunea 2.3.3 care îmi asigură legătura printr-un pool care primește ca parametrii hostul, user-ul, numele și parola bazei de date. Acestea sunt securizate într-un fișier de tip .env, furnizat de biblioteca "dotenv" versiunea 16.0.3. RestAPI-ul execută instrucțiuni SQL prin intermediul pool-ului sau prin intermediul unei conexiuni directe pentru tranzacții.

Am utilizat biblioteca "https" versiunea 1.0.0 pentru utilizarea protocolului securizat hypertext transfer protocol. Astfel am criptat datele de transport cu ajutorul unui certificat SSL. Un astfel de certificat este format din două fișier de tip .pem, un certificat și o cheie privată. Acestea sunt date ca parametrii la funcția de creare a serverului https.createServer() și sunt necesare pentru algoritmul de criptare a protocolului. Prin intermediul acestui protocol putem preîntâmpina numeroase vulnerabilități de securitate precum falsificarea mesajului, atac data-thief sau eavesdropping. Mai multe informații despre problemele de securitate tratate se întâlnesc la subcapitolul 2.3.3.

Având în vedere că performanțele serverului pot fi scalate vertical, am utilizat "pm2", versiunea 5.3.0, care aşa cum reiese și din pagina oficială a bibliotecii [9] ajută la managementul proceselor atât în perioada de producție cât și în cea de exploatare. Aceasta folosește un sistem de echilibrare al taskurilor, care permite restartarea serverului fară a fi nevoie să închiderea sa temporara. Totodată acest pachet se bazează pe un model cluster, în care toate threadurile procesoarelor de pe sistem sunt utilizate în paralel, astfel timpul de așteptare pentru un request este constant chiar dacă se fac un număr ridicat de requesturi simultan.

Atunci când un request este primit de server primul lucru este verificată integritatea acestuia prin 2 funcții de middleware care în caz afirmativ execută acțiunea next, iar în caz negativ returnează un response de eroare cu status code din seria 400 (client errors). Aceste două funcții compun de partea de autorizare care verifică bearer-tokenul transmis prin headerul "autorization" al protocolului HTTPS. În cazul validării acestor funcții, se trece la următoarea funcție, care este propriu zis funcția aferentă acelei rute URL, în cadrul căreia se execută și

secvență SQL pentru modificarea bazei de date. În final în funcție de rezultat se dă response cu status code din seria 200, iar în body datele necesare, ori în caz de eroare se returnează status 400 și un mesaj cu textul erorii pentru identificare și tratare în frontend.

Conecțarea serverului la SQL, Firebase și memoria internă

Ca RestAPI-ul să fie perfect funcțional, este nevoie de o legătură între logica serverului (proiectul scris în JavaScript, cu ajutorul frameworkului NodeJS) și sistemele de stocare sau autorizare. În alte cuvinte trebuie realizată o conexiune între API și următoarele trei componente:

- Baza de date SQL
- Memoria internă a serverului
- Platforma Firebase

Aceste conexiuni s-au realizat cu ajutorul unor biblioteci dedicate, care au implementate în componența lor o serie de metode care simplifică acest proces. Având în vedere că, atât conectarea la baza de date relațională, cât și la Firebase necesită un număr de chei de acces sau parole, este necesară protejarea acestora, în cazul unor intruziuni. Am utilizat biblioteca “dotenv”, care gestionează variabilele de mediu utilizate în aceste două contexte. Este o modalitate sigură și eficientă pentru gestionarea acestei variabile, deoarece fișierul .env, în care sunt introduse aceste variabile nu face parte din codul sursă; mai mult poate fi adăugat în fisierul .gitignore pentru a împiedica încărcarea acestor informații sensibile pentru publicul larg.

Revenind la configurare, am creat în interiorul proiectului din NodeJS un director “/config” ce conține trei fisere, fiecare fiind asignat pentru conectarea unui serviciu.

• Baza de date SQL

Această conectare se face în fișierul “/config/db.js”. Am utilizat biblioteca “mysql2” pentru managementul queryurilor și al conectării bazei de date. Pentru a face legătura dintre server și baza de date se configuraază un pool. Un pool reprezintă un mecanism de gestionare și reutilizare pentru conexiunile dintre baza de date și API. Această variantă este mai eficientă, deoarece nu este necesară conectarea la baza de date pentru fiecare cerere. Conexiunile individuale sunt mult mai costisitoare, deoarece ar trebui create și închise după fiecare request. Pool-ul ajută la rezolvarea acestui inconvenient, prin crearea unui set prestabilit de conexiuni, care pot fi accesate cât timp serverul este funcțional.

Trebuie menționat că în cazul tranzacțiilor a fost necesară o conexiune separată față de cea din pool, astfel se deduce și complexitatea tranzacțiilor. Pentru a crea și închide o astfel de conexiune am utilizat următoarele metode :

```

const connection = await mysql.createConnection({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    database: process.env.DB_NAME,
    password: process.env.DB_PASSWORD,
});
connection.destroy();

```

Un pool se crează asemănător cu o conexiune, însă se utilizează metoda **createPool()**, și se adaugă încă trei atribute ce reprezintă coada de așteptare și limita de conexiuni simultane:

```

waitForConnections: true,
connectionLimit: 10,
queueLimit: 10

```

- **Memoria internă a serverului**

Așa cum am explicitat și în subcapitolul referitor la stocarea resurselor, pentru încărcarea de imagini este necesară o conexiune între API și accesul la memoria internă a dispozitivului. Astfel am utilizat biblioteca “multer” pentru stocarea imaginilor pe server, și biblioteca “fs” pentru ștergerea acestora.

Conecțarea la memoria de pe disc am realizat-o în fișierul “/config/storage.js”, folosind metoda **diskStorage()**, ce are ca parametrii destinația și numele noului fișier:

```

const storagePlace = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'images/places/')
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname))
  }
});
const uploadPlace = multer({storage: storagePlace});

```

- **Platforma Firebase**

Conecținea la Firebase este configurată în fișierul “config.firebaseio-config.js” și se realizează prin biblioteca “firebase-admin”. Pentru a inițializa această punte între serviciu și API este utilizată metoda **initializeApp()**, ce are ca parametru credentialele acelui proiect Firebase. Acestea includ mai multe chei, tokene și certificare, ce ajută în a decodifica token-urile de acces. Se poate observa mai jos fragmentul de cod ce initializează acest serviciu.

```

const serviceAccount = {
  "type": process.env.FB_TYPE,
  "project_id": process.env.FB_PROJECT_ID,
  "private_key_id": process.env.FB_PRIVATE_KEY_ID,
  "private_key": process.env.FB_PRIVATE_KEY,
  "client_email": process.env.FB_CLIENT_EMAIL,
  "client_id": process.env.FB_CLIENT_ID,
  "auth_uri": process.env.FB_AUTH_URI,
  "token_uri": process.env.FB_TOKEN_URI,
  "auth_provider_x509_cert_url": process.env.FB_AUTH_PROVIDER_X509_CERT,
  "client_x509_cert_url": process.env.FB_CLIENT_X509_CERT_URL
};

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});

```

Schedule

Un alt concept interesant adăugat în componenta serverului sunt acțiunile planificate în viitor. Am utilizat acest concept pentru modificarea statusului unei rezervări. De exemplu atunci când o rezervare începe sau se termină, statusul acesteia trebuie modificat din “accepted” în “on-going” în “finished”. Mai mult după finalul unei rezervări userul trebuie să primească o cerere de evaluare a experienței (rating). Acest concept l-am îndeplinit prin utilizarea bibliotecii “node-schedule”.

Pentru crearea unei astfel de acțiuni planificate în viitor este nevoie de o variabilă constantă ce ia valoarea metodei **schedule.scheduleJob()**. Această metodă are 2 parametrii: primul este un obiect **Date()**, ce reprezintă momentul de timp la care acțiunea să se execute, și o funcție care are în componenta sa acțiunea.

Pentru a închide un astfel de scheduleJob se execută metoda **cancel()**.

Trebuie menționat că acestea acțiuni planificate se pierd atunci când serverul este restartat sau oprit. Astfel o modificare ulterioare pe care o am în vigoare este încărcarea temporară a informațiilor joburilor în baza de date, până acestea se execută. În caz de oprire a serverului, prima acțiune după pornire este reluarea joburilor din baza de date. În acest mod se menține consistența datelor la nivel logic.

Scalabilitate

Termenul de scalabilitate este echivalent cu dezvoltarea serverului din punct de vedere al capacitatii de preluare a cererilor si de eficientizare a raspunsurilor acestora. Există mai multe metode prin care se pot spori performantele unui sistem, dintre care eu am utilizat două metode pentru eficientizarea requesturilor.

Cu ajutorul bibliotecii "pm2" din javascript am realizat o scalare pe baza de clustere. Practic această bibliotecă folosește un alt pachet: "cluster" care formează mai multe instanțe ale serverului NodeJS. Astfel se formează un sistem distribuit ce rulează pe fiecare fir de execuție al procesorului de pe sistem acest server. Primul fir este master, el se ocupă în primă instanță de pornirea serverului, iar restul sunt workers, preluând requesturile. Având în vedere că un sistem server are un număr de threaduri de ordinul zecilor, iar o medie de execuție a unui request este de 35 ms, se deduce că prin această scalare se pot primi aproximativ 2500 de requesturi / secunda ceea ce nu implică o latentă de așteptare.

De asemenea, această bibliotecă utilizează un sistem de load balancing. Astfel toate firele de execuție au o cantitate echilibrată de requesturi, ceea ce reduce timpul de așteptare până când requestul este preluat și executat.

Totodată acesta pachet vine cu un sistem de monitorizare a fiecărui fir de execuție, fiind posibilă restartarea sau oprirea individuală. O caracteristică importantă este restartarea întregului server, care se întâmplă secvențial thread cu thread; aşadar nu există o perioadă de idle, fiind posibilă preluarea pe N-1 threaduri cât timp durează restartul.

Ulterior se poate lua în calcul și o scalare verticală. Aceasta implică achiziționarea de noi mașini hardware. Această tranziție este utilă atunci când memoria sau lățimea de bandă nu mai satisfac cerințele de o bună funcționare.

4.2.3 Firebase Auth și Elemente de Securitate

Partea de auth, ce include autorizarea și autentificarea, este realizată cu ajutorul Firebase, funcționalitatea de Authentication. Am conectat proiectul din Firebase atât la partea de backend cât și la frontend. Funcțiile de login, logout și signin sunt implementate direct în interfață grafică, iar funcțiile de autorizare și verificare a tokenului sunt implementate în backend. Am utilizat biblioteca "firebase-admin" în proiectul din node; am inițializat aplicația cu ajutorul credențialelor din Firebase, care sunt de asemenea, ca în cazul bazei de date, stocate în fișierul .env. Firebase folosește un sistem de refresh-token ce persistă 30 min, fiind detaliat modul de utilizare în următorul articol [5], astfel se furnizează o siguranță sporită, regenerarea fiind automată în timp ce userul este logat la interfața grafică.

Funcțiile de autorizare:

- **decodeToken()**

preia bearer tokenul din headerul https și îl verifică cu ajutorul funcției VerifyIdToken(), din biblioteca prezentată mai sus. În cazul în care tokenul este valid, se execută next(), trecând la următoarea funcție de middleware. Dacă tokenul nu este valid, se returnează un response negativ.

- **UserAuthorization()**

verifică dacă tokenul este al unui utilizator care are permisiunea să execute acea acțiune. Această funcție compară id-ul de authenticare, decodat din token cu id-ul unde se va executa o operație în baza de date, astfel se realizează un strat de protecție suplimentar.

Alte elemente de securitate preîntâmpinate în partea de backend sunt:

- SQL injection, folosind queryuri parametrizate; astfel atacatorul nu poate să introducă alte secvențe de cod în queryurile deja existente. Acest tip de queryuri se formează prin înlocuirea parametrilor ce trebuie adăugați cu “?”, asemenea acestui exemplu extras din cod

```
const sql = 'UPDATE reservation SET status = ? WHERE idreservation = ?;'
```

Apoi se crează o lista cu toți parametrii

```
const params = [status, idreservation];
```

Ca în final să fie executat cu ajutorul funcției db.query(sql,params);

- Cu ajutorul protocolului HTTPS am preîntâmpinat atacuri de tip man-in-the-middle, în care atacatorul poate să preia, modifice sau să fure datele transmise din și către server. Aceste atacuri poartă următoarele denumiri: Message Forgery, Data Theft, Eavesdropping
- În ultimul rând am introdus și helmet că middleware, acesta fiind furnizat de biblioteca cu același nume. Aceast pachet ajută în securizarea protocolului HTTPS prin adăugarea mai multor headere.

4.2.4 Testare

Având în vedere natura serverului, requesturile pot returna diferite valori în funcție de cazul în care se întâlnesc acțiunile comandate. Astfel fiecare path duce către o metodă care în funcție de posibilitate returnează diverse rezultate sau chiar eroare. De exemplu metoda **decodeToken()** explicită la 4.2.3 poate să returneze un body cu “Unauthorize.”, “No token provided.” sau “Invalid token.” În caz de eroare. Astfel un request care trece prin ambele metode de autorizare și apoi se execută metoda propriu zisă poate să aibă până la 12 cazuri de returnare.

Pentru a verifica integritatea metodelor create în server este nevoie de o modalitate de testare atât rapidă cât și flexibilă; pentru a facilita multe cazuri. O regulă importantă în testare este de a introduce cât mai multe teste, astfel cazurile de bug-uri se diminuează.

Pentru testare am utilizat biblioteca “jest”, versiunea 29.5.0, care permite scrierea testelor pentru verificarea ulterioara. Un alt program util folosit în perioada de implementare a backendului este POSTMAN. Cu ajutorul acestui program puteam să execut requesturi către server, având și posibilitatea de a urmări performanțele acestuia, atât din punct de vedere a timpului, cât și pentru resurse utilizate.

Jest

Am utilizat biblioteca Jest pentru a testa cererile cu un număr mare de cazuri de intrare. Avantajul acestui mod de testare este că testele rămân salvate, fiind scrise manual. Pentru testare am creat un director „/_test_” care are în componenta să fiserele de testare cu extensia **.test.js**.

În cazul următor voi prezenta un test pentru metoda de creare a unui nou local:

```
describe("POST /places/create", () => {
  test("Create Successful", async () => {
    const response = await request(app).post('/places/create')
      .set('Authorization', 'Bearer ' + token)
      .send({
        "name" : "Hype Arena by Kiddo",
        // ...am redus din date
        "hours_of_opp" : "10:00-23:00"
      });
    expect(response.statusCode).toBe(201);
    expect(response.body.message).toBe("Place created successful");
  });
});
```

În urma executării comenzi: **npx jest places.test.js** se returnează ce teste dau eroare. Astfel am o viziune clară asupra cazurilor netratate sau tratate incorect.

Postman

Această platformă facilitează dezvoltarea și testarea aplicațiilor web, printr-un set de instrumente utile. Postman permite trimiterea de requesturi HTTPS personalizate către serverul local.

Am utilizat acest software pentru teste din timpul implementării propriu-zise, datorită volatilității și vitezei de testare. Postman permite dezvoltatorului crearea de requesturi personalizate într-un timp foarte scurt, ceea ce este perfect pentru perioada de implementare.

Mai mult, acest program include elemente avansate de monitorizare a cererilor HTTP. Astfel se poate analiza memoria utilizată sau timpii de procesare ai cererilor. Aceste funcționalități ajută dezvoltatorul în procesul de optimizare. Se poate observa un exemplu de diagramă a proceselor temporale ale unui request în Figura 4



Figura 4: Desfasurare temporală a unei cereri web

4.3 Frontend

4.3.1 Flutter

Flutter este un SDK pentru crearea de user-interface open-source creat de Google. Așa cum reiese din sursa [6], este utilizat pentru crearea de aplicații crossplatform printre care amintim: iOS, Android, Linux, macOS, Windows, web, etc. Engine-ul Flutter este scris în C++ și oferă suport de nivel înalt de randare. În cadrul proiectului meu am utilizat Flutter pentru implementarea crossplatform a unei aplicații de mobile cu suport Android și iOS. Limbajul de implementare al acestui framework este Dart (extensia .dart), fiind foarte asemănător cu Java.

În Flutter toate widgeturile ce compun ecranul sunt de fapt niște clase, care sunt interpretate de către engineul făcut în C++, care mai departe îl compilează ca o randare. Un proiect în Flutter poate fi constituit că un arbore de widgeturi. Pentru a lucra într-un proiect Flutter se modifică folderul lib; acesta conține și fișierul main.dart care se rulează la build. În acest fișier întâlnim metoda void main() în care se apelează metoda runApp() care are ca parametru un Widget; această metoda facilitează încărcarea acelui widget pe ecran. Astfel widgetul/ clasa MyApp se rindeaza și devine automat baza arborelui de widgeturi.

Clasa MyApp are implementată o metodă ce returnează tot un widget, build(), pentru a crea o aplicație trebuie să returnăm MaterialApp() aceastei clase îi putem da inițial valoare prin constructor pentru mai multe atribute esențiale în aplicație, precum numele aplicației, theme

(tema de fonturi și culori folosite) și routes care ține evidența tuturor ecranelor ce compun aplicația

4.3.2 Interfață Grafică și Experiența Utilizatorului

Structura proiectului

Aplicația este formată din 20 de ecrane individuale. În cadrul acestei secțiuni voi defalca și explica fiecare ecran și funcționalitățile acestora, punând accentul pe tranziția și legăturile dintre acestea.

Atunci când aplicația pornește, intră în pagina **init_page.dart**. Aceasta nu are o interfață propriu zisă, ci este formată din mai multe teste care trimit aplicația pe pagina potrivită, în funcție de starea în care se află după finalizarea funcției init (preluarea datelor de pe server). Inițial se verifică dacă funcția a fost finalizată. Atât timp cât încă nu este gata, aplicația rămâne pe un ecran de loading. După finalizare se testează dacă serverul este activ. Dacă nu, se intră într-o pagină de eroare, care specifică utilizatorului că există o problemă legată de server. Dacă nu este nicio problemă, se verifică dacă există user curent. În caz contrar se returnează ecranul de login, iar un caz afirmativ se verifică dacă emailul userului este validat. În cazul în care nu este validat aplicația intră în ecranul de validare. Apoi se verifică dacă în baza de date există datele userului, deoarece în caz contrar se intră pe ecranul de register phase 2. După aceste teste se mai verifică dacă utilizatorul a acceptat accesul la locație. În cazul în care nu mai există acces se intră pe ecranul aferent acestei cerințe. În final dacă toate aceste condiții sunt îndeplinite, se intră pe pagina main, însă dacă utilizatorul are un rate request, se intră pe pagina de evaluare a unei rezervări. Această schemă este exemplificată și în diagrama din Figura 5.

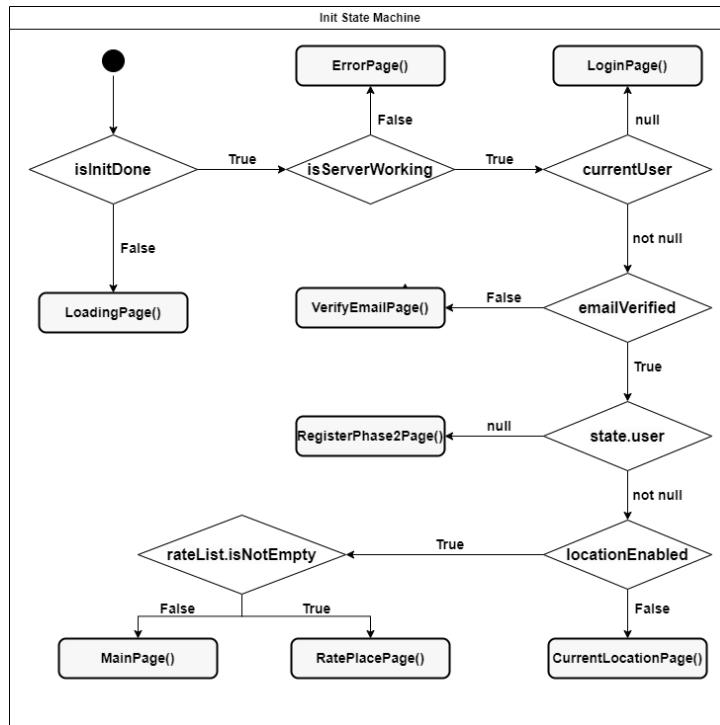


Figura 5: Diagrama de stări pentru init

În cele ce urmează voi face o scurtă prezentare a fiecărui ecran prezent în aplicație, iar capturile de ecran aferente se regăsesc în arhivă, Figurile 12, 13, 14 și 15:

- **InitPage()**: pagina explicată mai sus, este o pagina ce tratează starea inițială a aplicației.
- **ErrorPage()**: indică faptul că există o problemă de conectivitate la server.
- **LoadingPage()**: este formata dintr-un loading indicator și este utilizată cât timp se așteaptă finalizarea testării de stări init.
- **CurrentLocationPage()**: pe aceasta pagina se cere accesul la locația dispozitivului.
- **VerifyEmailPage()**: este o pagina din setul de ecrane din procesul de înregistrare; aceasta are un timer de 1 minut ce repermite apoi retrimiterea unui mail de verificare; îndată ce mailul este verificat aplicația trece la ecranul următor.
- **LoginPage()**: pagina de autentificare.
- **RegisterPage()**: pagina pentru înregistrare.
- **Register2Page()**: este o pagina din setul de ecrane pentru înregistrare, unde utilizatorul își adaugă datele personale și imaginea de profil.
- **MainPage()**: reprezintă pagina principala; prima pagina după autentificare, aceasta conține diverse widgeturi ce trimit utilizatorul către mare parte din funcționalități.
- **EditProfilePage()**: este o pagina pentru editarea datelor utilizatorului; aceasta pagina se accesează din drawer; în cardul acesta se întâlnește și funcționalitatea de signout.
- **ResetPasswordPage()**: se accesează din pagina de login și ajuta utilizatorul să își recupereze parola, prin trimiterea unui email de resetare.

- **PlaceCategoryPage()**: aceasta pagina este utilizata pentru alegerea localurilor și pentru filtrare.
- **PlaceCategoryDetailedFilterPage()**: este o pagina pe care sunt listate toate filtrele de tip atribut.
- **PlaceLocationFiltersPage()**: pe acesta pagina se alege locația în jurul căreia se vor căuta localuri.
- **PlaceResultListPage()**: este pagina pe care sunt afișate rezultatele căutării, de asemenea, aceasta este sub două forme: lista sau harta.
- **PlaceDetailsPage()**: este pagina pe care sunt prezentate detaliile localurilor selectate.
- **RatePlacePage()**: aplicația pornește cu acesta pagina dacă există rezervări anterioare la care nu s-a încercat efectuarea unei evaluări.
- **RecommanderSystemPage()**: este pagina principală a sistemului de recomandări.
- **RecommanderSystemResultPage()**: este pagina pe care sunt primite rezultatele algoritmilor de recomandări.
- **ReservationsFuturePage()**: pe acesta pagina se întâlnesc rezervările programate.
- **ReservationsPreviousPage()**: iar pe aceasta pagina se întâlnesc rezervările precedente.
- **SearchPlaceResults()**: este pagina pe care utilizatorii pot căuta localuri după nume.
- **SearchPlaceAllResults()**: iar pe aceasta pagina acestia văd toate rezultatele cu acel acronim; se aceseaza prin apăsarea textului „see more” .

Design receptiv

Având în vedere multitudinea de dispozitive telefonice de pe piață, trebuie luat în calcul că dimensiunea ecranului variază de la utilizator la utilizator, astfel o cerință esențială în dezvoltarea unei aplicații mobile moderne este utilizarea unui design receptiv.

Acest concept redă faptul că elementele ce compun ecranul, și anume widgeturile trebuie să aibă dimensiuni raportate la dimensiunea ecranului. Un alt aspect important este alinierea widgeturilor, astfel încât redimensionarea acestora să nu afecteze aliniamentul.

În Flutter acest concept este relativ ușor de realizat cu ajutorul clasei **MediaQuery**. Se pot utiliza metode precum **MediaQuery.of(context).size.[height sau width]** pentru aflarea dimensiunii ecranului, ca mai apoi elementele de compun imaginea să reprezinte fracțiuni din acele valori. Totodată această clasă are și variabila **aspectRatio** care ajută programatorul în setarea de paddinguri și aliniamente.

Experiența utilizatorului și elemente de design

Un factor esențial în reușită unui proiect este ca aspectul vizual să fie unul plăcut și funcțional. Designul unei aplicații reprezintă una din pietrele de temelie ale succesului, deoarece aceasta este legătura directă între utilizatori și proiect.

Aspectul aplicației mele este unul modern, fiind inspirat din alte aplicații din același segment de piață, cu un public cu precădere Tânăr și dinamic. Având în vedere că scopul aplicației este de a ajuta utilizatorii să descopere localuri într-un mod mai rapid, am optat pentru un design simplist, bazat pe forme regulate, și widgeturi standard, care să nu ofere ambiguitate, însă acestea au fost adaptate la cerințele tematicii de culori.

Am optat pentru o paletă de culori monocromă, având o culoare de contrast galben-verzui. Am utilizat această culoare pentru a da impresia de dinamicitate, redând de asemenea, scopul principal al aplicației. Totodată, paleta coloristică duce către unicitate și originalitate, putând fi imediat remarcată. Culorile utilizate și codul hex sunt prezentate în Figura 6.



Figura 6: Paleta de culori utilizată în aplicație

Pentru un aspect mai modern, am utilizat diverse animații, create cu widgeturi existente în Flutter, precum **AnimatedContainer**, **Builder**, **AnimatedIcon**, **AnimatedText** și așa mai departe. Acestea au ca parametrii o durată și tranziția, sub formă de elvis operator. Animațiile au scopul de a menține atenția utilizatorului atunci când intervine latența requesturilor de pe server.

Mai multe imagini reprezentative găsiți în secțiunea Anexe unde am extras o gamă de screenshot-uri din aplicație.

4.3.3 Componentele Aplicației

Pentru a crea o experiență mai bună utilizatorilor, aplicația trebuie să deservească toate nevoile acestora. Mai mult de atât, aplicația trebuie să vină în ajutorul userilor prin mici elemente automatizate ce simplifică procesul de utilizare. Exemple de acest fel sunt: utilizarea notificărilor (cu scop de reamintire sau atenționare), utilizarea locației curente (pentru a simplifica procesul de căutare a utilizatorilor), utilizarea imaginilor din galerie (atunci când se doresc încărcarea sau actualizarea de imagini) și așa mai departe.

Obtinerea de privilegii pentru utilizarea acestor servicii strâns legate de sistemul de operare, se face cu ajutorul supra scrierii de proprietăți din fisiere de baza: **AndroidManifest.xml** pentru Android și **Info.plist** pentru iOS.

Acces locație

Datorită diverselor modalități prin care utilizatorul poate să găsească localuri, una dintre variantele posibile este căutarea localurilor din proximitate. Acest tip de căutare este util în zone metropolitane de dimensiunea Bucureștiului, care se pot întinde pe sute de kilometri pătrați. O astfel de căutare pe o rază necesită coordonate în spațiu, ce reprezintă punctul de plecare al razei. De cele mai multe ori utilizatorii doresc să caute localuri în proximitate, astfel că introducerea locației curente poate deveni deranjanta după o perioadă de timp. Astfel că o funcționalitate este aceea că aplicația să verifice coordonatele spațiale ale telefonului, pentru a scuti userul de procesul de selectare al locației, numai dacă acesta intenționează să aleagă altă locație înapoi de cea curentă.

Pentru accesul la locația dispozitivului, utilizatorul trebuie să își dea acordul ca aceasta să fie furnizată. Aceasta cerere se face odată ce utilizatorul își crează un cont, sau atunci când aplicația detectează că acest drept a fost oprit din setările telefonului.

Am utilizat biblioteca **geolocator 9.0.2** din Flutter, pentru accesul locației dispozitivului. Pentru a verifica că se pot extrage coordonatele în aplicație, se utilizează metoda **isLocationServiceEnabled()**; aceasta cere utilizatorului dreptul de a permite extragerea coordonatelor. Apoi se apelează funcția **checkPermission()**, care verifică dacă utilizatorul a acceptat acest serviciu. Cu ajutorul metodei **getCurrentPosition()** se extrag latitudinea și longitudinea, la care sunt localizate telefonul.

Acces galerie și camera

Atât utilizatorii cât și localurile au nevoie de funcționalitatea de încărcare de imagini în aplicație. Aceasta oferă un aspect mai plăcut, dar totodată ajută ambii beneficiari în scopul de localizare și identificare.

Pentru accesul la galerie sau la camera dispozitivului este nevoie de un rând de privilegii acordate de către setările sistemelor de operare. De exemplu în IOS pentru accesarea camerei și a galeriei, plus pentru descărcarea acestora din memoria internă a telefonului, trebuie suprascrise setările de bază din fișierul **Info.plist** :

```
<key>NSCameraUsageDescription</key>
<string>To take user profile image</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>To choose user profile image</string>
<key>NSMicrophoneUsageDescription</key>
<string>Used to capture audio for image picker plugin</string>
```

De asemenea, am utilizat biblioteca **image_picker 0.8.8** pentru extragerea imaginilor din galerie. Aceasta este salvată local în aplicație până este trimisă, printr-o cerere aplicației

web, sub forma de fișier. Am preluat imaginiea din galerie cu ajutorul metodei **ImagePicker().getImage(source: ImageSource.gallery)**.

Acces notificări

O altă funcționalitate, pe care am considerat-o esențială pentru experiență utilizatorului, este utilizarea notificărilor. Acestea sunt folosite în scop de atenționare. Am utilizat acest serviciu în trei situații: se crează o rezervare, mai este o ora până la următoare rezervare și o rezervare își modifică statusul.

Pentru utilizarea notificărilor trebuie modificare fișierele de permisiuni ale sistemelor de operare pe care rulează aplicațiile. De exemplu în Android am adăugat două permisiuni în fișierul **AndroidManifest.xml** pentru aceste privilegii:

```
<meta-data  
    android:name="com.google.firebaseio.messaging.default_notification_channel_id"  
    android:value="channel_id 6"  
/>  
<intent-filter>  
    <action android:name="FLUTTER_NOTIFICATION_CLICK"/>  
    <category android:name="android.intent.category.DEFAULT"/>  
</intent-filter>
```

Apoi am utilizat biblioteca **fultter_local_notifications 14.0.0+1**, care initializează și configerează notificările. Am utilizat obiectele **AndroidInitializationSettings()** și **DarwinInitializationSettings()** pentru configurarea notificărilor pe cele două platforme tintă. Pentru afișarea unei notificări am utilizat metoda **show()**, iar pentru programarea unei notificări în viitor, utilă în cazul în care doream afișarea cu o ora înainte de o programare, am utilizat metoda **zonedSchedule()**.

4.3.4 State Management în Redux

Redux reprezintă o alternativă fiabilă pentru manageriera și centralizarea stăriilor prin care trece o aplicație, așa cum este prezentat în [4]. Acst tool funcționează după următorul principiu: există un Store global care încapsulează variabilele utilizare în reprezentarea UI-ului. UI-ul prin interacțiunea ușerului creează un Action. O acțiune la rândul ei creează un Dispatcher în Store. Aici intervine Epics; acest epics leagă acțiunea de rezultatul ei care poate să fie de success sau error (în cazul de utilizare a apiurilor, dacă nu se pot prelua datele necesare prin api). Dacă acțiunea este successful atunci se trece în starea de Reducer; în Reducer se pot face modificări la elementele din Store, astfel se creează această încapsulare de date. După ce se modifică în Reducer se modifică state-ul și se trece iar în starea de UI unde se modifică afișajul. Se poate înțelege acest concept și prin reprezentarea din Figura 7.

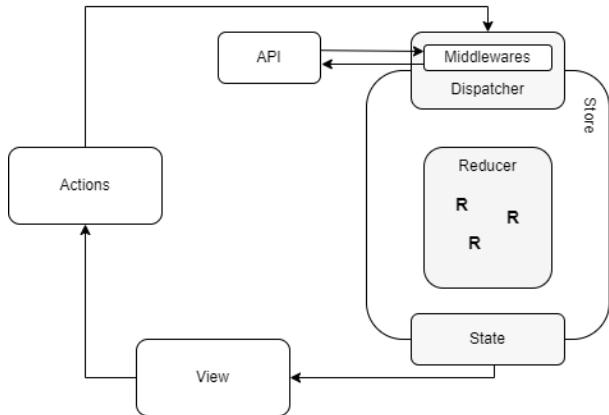


Figura 7: Model automat redux furnizat de la [4]

Deși folosirea unei structuri Redux în dezvoltarea unui proiect poate fi costisitoare, aceasta vine cu o serie de avantaje care își fac resimțită prezentă atunci când dimensiunea proiectului depășește pragul de proiect minor. Principalele avantaje ale acestei arhitecturi sunt:

- **imutabilitatea:** obiectele și variabilele stocate nu pot fi modificate decât intern.
- **consistentă:** datele stocate în store sunt consistente, fiind vorba de un singur loc unde acestea se modifica, șansele unei erori de stocare sunt reduse.
- **modularitate:** se referă la faptul că Redux facilitează o structură standard care permite reutilizarea codului.
- **accesibilitatea:** Așa cum reiese și din capitolul 4.3.2 structura widgeturilor în Flutter este arborescentă, astfel în mod convențional transferarea variabilelor prin widget-tree este costisitoare. Redux permite prin store accesul oricărei variabile stocate să fie accesibile oriunde, indiferent de nivelul din arbore la care se găsesc.

Pentru utilizarea tool-ului Redux am avut nevoie de următoarele pachete: redux: versiunea 5.0.0 flutter_redux: versiunea 0.9.0 redux_epics: versiunea 0.15.1 Totodată cu aceste pachete am mai utilizat 6 pachete care să mă ajute în generarea de cod pentru clasele și acțiunile componente automatului Redux. Prin executarea comenzii următoare în consola aceste pachete, prin tool-urile lor, identifică fișierele din folderele /models (pentru clase) și /actions (pentru acțiuni) și generează în index.g.dart, serializers.g.dart și index.freezed.dart clasele cu tot cu serializare și deserializare.

```
flutter pub run build_runner watch --delete-conflicting-outputs
```

Pachetele pentru generare clase sunt: built_value: 8.1.4 built_collection: 5.1.1 build_runner: 2.1.8 built_value_generator: 8.1.4 iar cele pentru generare acțiuni sunt: freezed: 1.1.1 freezed_annotation: 1.1.0.

Structura fișierelor din proiectul Flutter respectă împărțirea stărilor Redux; Așadar proiectul devine modular, fiind mult mai ușor de împărțit și gestionat. În continuare voi prezenta structura proiectului pe directoare, având în vedere și componențele acestora:

- **/lib/actions:** sub acest director sunt regăsite fișierele cu acțiuni. Acestea sunt activate prin metoda redux **dispatch()** și stau la baza schimbării stateului.
- **/lib/containers:** în acest folder sunt create widgeturi cu scopul de a face accesul la anumite date din store mai rapid. Aceste Widgeturi au incorporate diferite instanțe ale variabilelor din store.
- **/lib/data:** fișierele din acest director sunt responsabile de requesturile și response-urile API-ului. Conține funcții asincrone pentru fiecare request.
- **/lib/epics:** acest director conține un unic fișier cu același nume care reprezintă o metoda de middleware în manevrarea acțiunilor asincrone.
- **/lib/init și lib/mixin:** sunt două directoare cu un singur fișier în ele care au ca scop configurarea aplicației în stadiul de init.
- **/lib/models:** reprezintă un director ce conține toate clasele de obiecte care formează store-ul. Practic, în acest folder se definesc datele stocate în timpul rulării aplicației.
- **/lib/reducer:** Conține metodele de reducer, care sunt responsabile de modificarea datelor din store, atunci când se modifica statul, pe baza initializarii unei acțiuni.
- **/lib/presentation:** Conține toate ecranele care compun aplicația. În acest director sunt stocate toate fișierele responsabile de interfață grafică.
- **main.dart:** se apelează atunci când aplicația rulează.

4.3.5 Google Maps API

Am utilizat API-ul Google Maps pentru a conferi o imagine mai clară a poziționării localurilor, fiind mult mai convenabil punerea lor pe hartă, dar și pentru a ajuta utilizatorul să aleagă mai ușor zona de interes pentru filtrare. Pentru a folosi acest serviciu am importat în proiectul din Flutter următoarele biblioteci: `google_maps_flutter: 2.2.5` și `dart_geohash: 2.0.2`, iar de pe site-ul de developer Google am generat o cheie unică pentru utilizarea acestui API. Am adăugat în aplicație hartă cu ajutorul widgetului `GoogleMaps`, iar pentru a adăuga punctele cu localurile pe mapă am adăugat în constructor la atributul “`markers`” o structură de date de tip set cu toți markerii și coordonatele lor măsurate în latitudine și longitudine. Acest serviciu furnizat de Google oferă și posibilitatea de stilizarea a hărții, astfel am generat un String cu structură JSON care modifică proprietatiile hărții din punct de vedere estetic.

În componența aplicației am utilizat serviciul de hărți în trei situații distințe, utilizând două tipuri de hărți. În primul rând am utilizat hărțile pentru a oferi utilizatorilor o experiență cât mai autentică în cadrul aplicației, iar pe plan secundar este un scop funcțional.

Primul tip de hărți utilizate au fost hărțile dinamice, fiind utilizate de două ori în componența aplicației: pentru afișarea localurilor pe hartă, cu scopul de UX, și atunci când utilizatorul trebuie să își aleagă locația unde dorește să caute localuri, având un scop funcțional. În ambele cazuri trebuie folosit widgetul **GoogleMap**, din cadrul bibliotecii `google_maps_flutter`.

Pentru o consistență în design, platforma google vine în ajutorul programatorilor cu diverse varietăți de a customiza harta. Astfel în fisierul “`/lib/services/google_maps_style.dart`” se

întâlnește sub formă de String mapstyle-ul. Acesta are o structură de JSON, și se dă ca parametru atunci când harta se initializează.

Un ultim mod în care am utilizat harta a fost în modul static, pe pagina de datalii a unui local, unde este afișată o hartă cu un pin-marker asupra locației sale. Această hartă se afișează printr-un request HTTPS către "<https://maps.googleapis.com/maps/api/>".

De asemenea, designul hărții statice se poate modifica pentru menținerea temei aplicației. În același fișier prezentat mai sus se întâlnește și variabila String ce conține detaliile designului hărții, însă nu mai este prezentă aceeași structură de JSON, ci o structură parametrizată pentru requesturile HTTP.

În final trebuie menționat că acest serviciu RestAPI este singurul serviciu care nu este gratis. Google oferă un plafon lunar de 200\$ pentru consumul în platformă. Astfel, după un calcul rapid reiese că aplicația mea trebuie să depășească 100.000 requesturi pentru a trece peste acel plafon, ceea ce este mult peste asteptările actuale. De asemenea, trebuie menționat că o îmbunătățire ulterioară este adăugarea hărților Apple pentru utilizatorii IOS, acestea fiind gratuite, iar acest plafon de 100.000 cereri/lună s-ar rezuma la utilizatorii de Android.

5 DETALII DE IMPLEMENTARE

5.1 Algoritmii de Recomandări - Machine learning

Pentru sistemul de recomandări am utilizat mai multe strategii de predicție a localurilor, astfel încât rezultatele să fie cât mai diverse și axate pe preferințele utilizatorilor. Algoritmii utilizati sunt des folosiți în extragerea de seturi de date masive, folosind principii matematice pentru a prezice nivelul de compatibilitate cu utilizatorul.

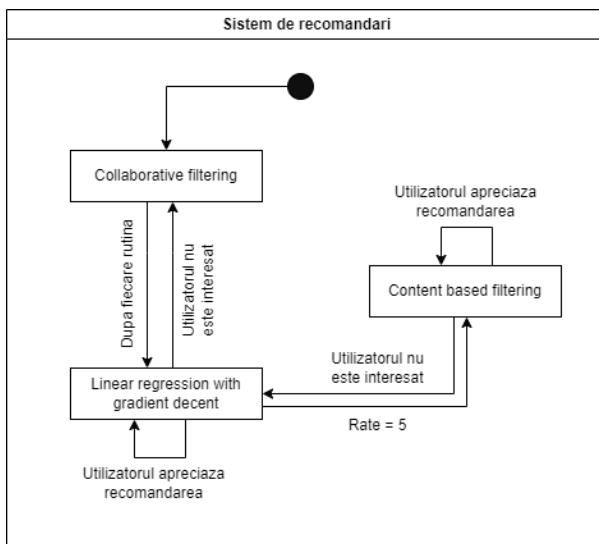


Figura 8: Diagrama starilor sistemului de recomandari

Cele trei strategii sunt următoarele: Collaborative filtering, ce se caracterizează prin recomandări bazate pe experiențele altor utilizatori (se caută utilizatori cu preferințe asemănătoare ca mai apoi să se prezică pe baza acestora sănsele ca noi localuri să fie plăcute și de utilizator), astfel aceast algoritm poate fi utilizat atunci când dorim să scoatem userul din zona sa de confort. A doua rutină de căutare este Content-based-filtering; aceasta caută noi localuri asemănătoare cu un local dat ca referință, astfel am utilizat acest algoritm după ce utilizatorul da un rating maxim. Al treilea algoritm este Linear-regresion-with-gradient-decent, care este cea mai echilibrată variantă, fiind o predicție bazată pe toate ratingurile acordate de utilizator, astfel se formează un vector de preferințe a atributelor, care ajută ulterior la predicție. Acești algoritmi au fost extrași din capitolul 9 al cărții [8]. În diagrama următoare am reprezentat automatul sistemului de recomandări: Se poate observa în Figura 8 comportamentul sistemului de recomandări. Așa cum am spus, inițial sistemul începe cu filtrare colaborativă, deoarece utilizatorul încă nu are o baza de rating pe care să se bazeze ceilalți algoritmi. După fiecare căutare colaborativă, următoarea va fi cu ajutorul rutinei de regresie liniară, iar dacă

utilizatorul are un comportament pozitiv cu căutările (), sistemul vă rămâne în acea stare, în caz contrar, va considera că rezultatele nu sunt de interes, astfel se va întoarce la căutarea colaborativa. Se ieșe din aceste stări dacă ultimul rating dat de user este maxim. În acest caz rutina va fi content-based pe baza aceluia local. Atunci când căutările content based nu mai sunt de interes algoritmul se întoarce la rutina de filtrare cu regresie liniară.

5.1.1 Content Based Filtering

Pentru utilizarea acestui algoritm avem nevoie de atributele ultimului local cu rating maxim. Acestea se extrag din baza de date apoi se formează un vector în funcție de existența atributelor sau nu (0 sau 1). Apoi se extrage din baza de date un număr de localuri pentru care se va calcula predicția. Pentru a pregăti datele se formează pentru fiecare local în parte un astfel de vector, iar mai apoi se normează datele.

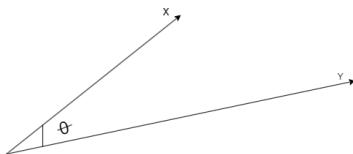


Figura 9: Cosine similarity

Pentru obținerea predicției content-based se folosește Cosine-similarity, ce reprezintă, așa cum se observă și în imaginea 9 de mai sus, diferența dintre unghiul format de cei doi vectori și 180 grade. Am utilizat biblioteca “cosine-similarity” din NodeJS pentru calcularea acestei valori pentru fiecare local în parte; acest pachet folosind formula 1.

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}} \quad (1)$$

5.1.2 Collaborative Filtering

Se bazează pe același principiu de predicție cu Content-based, însă nu se ia ca referință un vector format din atributele ultimului local cu rating maxim, ci se crează o matrice de atribută în raport cu ratingurile date de useri. Din acest punct de vedere metoda colaborativa consumă mai multe resurse, fiind mai lentă decât cea bazată pe un singur vector de atribută.

Pregătirea datelor pentru acest algoritm este diferită față de algoritmul content-based. Pentru acesta este nevoie de lista ratingurilor grupate după utilizatori, și ratingurile grupate după localuri. Principiul de predicție este executarea funcției **cosinesimilarity()** pentru fiecare pereche de ratinguri și localuri, astfel încât rezultatul se bazează pe posibilitatea ca un local să fie pe preferință utilizatorului, bazat pe ratingul altui utilizator cu care are în comun un rating asemănător.

5.1.3 Regresie Liniara cu Coborâre în Gradient

Algoritmul de regresie liniară cu coborâre în gradient este unul dintre cei mai simpli și eficienți algoritmi de machine learning utilizate pentru a realiza predicții cu ajutorul unui model liniar. În esență, algoritmul încearcă să găsească o linie dreaptă care se potrivește cel mai bine cu datele de antrenare; formulele și informațiile extrase sunt furnizate de pe site-ul [1]. Această linie dreaptă este reprezentată de o ecuație de forma $y = mx + b$. Acest algoritm utilizează lista de atribută a localurilor în care se face filtrarea, notată cu X , și lista de ratinguri oferite de utilizator, notată cu Y . Pentru calcularea scorurilor de predicție inițial am parcurs o serie de 750 de iterării în care am construit variabila theta după formula 2.

$$\theta = \theta - \frac{\alpha}{m \cdot ((X \cdot \theta - y)' \cdot X)'} \quad (2)$$

Din 50 în 50 de iterării se calculează pe baza lui X, Y , și theta: costul, eroarea și predicțiile intermediare, ilustrate prin ecuațiile următoare 3, 4, 5.

$$predictions = X \cdot \theta \quad (3)$$

$$sqrErrors = (predictions - y)^2 \quad (4)$$

$$J = \frac{1}{(2 \cdot m) \cdot \sum_i sqrErrors_i} \quad (5)$$

După finalul structurii repetitive se calculează predicția finală pentru toate localurile din lista X , după formula $X^*\theta$ și se salvează în variabila predictedRatings. Această listă este sortată și sunt excluse idurile localurilor care sunt deja evaluate de utilizator.

5.2 Algoritm de Rezervări

Înainte de a explica acest algoritm, trebuie explicată modalitatea de stocare în baza de date a rezervărilor. Pentru modularitate, am considerat 3 tabele utile acestui sistem. Prima este tabela de **activități**, care reprezintă generic un calup de mese/subdiviziuni ale acestei activități; spre exemplu pentru un restaurant putem avea activitățile de "rezervare pe terasă" sau "rezervare în interior", iar pentru o bază sportivă de pildă putem avea "rezervare teren tenis" sau "rezervare ședință echitație".

Următoarea tabelă este **activity_seating**, iar aceasta încapsulează subdiviziunile activităților. Exemplul curent este o continuare a exemplificării precedente: pentru activitatea "rezervare pe terasă", avem mai multe intrări în această tabelă care reprezintă mesele de pe terasă, și care au ca atribut în baza de date numărul disponibil de oameni la acea masă.

Rezervările se introduc în tabela **reservation**; o rezervare are două chei FOREIGN către activitatea dorită și către localul la care s-a făcut rezervarea, ora, data și numărul de oameni

pentru care este făcută rezervarea și o altă cheie FOREIGN către utilizatorul a cărei rezervare este.

De asemenea, mai există o tabelă **activity_arrangement** care leagă o rezervare de una sau mai multe subdiviziuni ale activității selectate.

Acum că am prezentat modalitatea de stocare din baza de date, rămâne de explicat logica serverului de a gestiona requesturile astfel încât să nu existe suprapunerি. Metodele care formează această logică sunt următoarele:

- **availability()** Se apelează prin requestul HTTP: <https://localhost:3000/places/availability?date=2023-03-07&idactivity=2&partySize=3>. Această metodă este responsabilă pentru pregătirea disponibilităților atunci când utilizatorul alege activitatea dată și numărul de oameni pentru care dorește rezervarea. Outputul acestei metode este o listă de obiecte de tipul:

```
{  
    'idactivityseating' : 12,  
    'hour' : '13:30'  
}
```

Care reprezintă ora și subdiviziunea la care sunt repartizați. Astfel algoritmul reușește să automatizeze sistemul de gestionare al activităților.

- **reservation()** Se apelează prin requestul HTTP: <https://localhost:3000/reservations/create> cu body ce include datele aferente rezervării. Această metodă nu are o logică prea avansată, având în vedere că funcția precedentă se ocupă de repartizare. Unica întrebuițare este de a verifica dacă disponibilitatea selectată nu a fost modificată, deoarece există o perioadă de timp între prezentarea disponibilităților și luarea deciziei de către utilizator.

Așa cum am spus și mai sus, metoda **availability()** este responsabilă de logica disponibilităților. Ideea principală de la care am plecat este crearea unor vectori de valori booleane în care fiecare element reprezintă 5 minute din timpul total pentru rezervări ale acelei activități. Astfel valoarea 1 reprezintă faptul că acel moment de timp este blocat, iar 0 reprezintă că acel interval de 5 minute este liber. De exemplu o matrice de disponibilități a unei activități la care există o rezervare care durează 30 minute de la ora 08:00, iar orele între care este valabilă activitatea sunt 07:30 și 11:00 arată în felul următor:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Această matrice se completează automat după preluarea rezervărilor din acea zi pentru acea activitate, după regulile următoare: Având în vedere că o diviziune de timp reprezintă 5 minute, lungimea vectorului se calculează după formula 6, unde **hoursOfOpp** reprezintă intervalul orar în care acea activitate permite rezervări și este de forma "HH:mm-HH:mm". O rezervare reprezintă adăugarea în vector de valori 1 în dreptul perioadei respective. Numărul de 1 adăugați se determină prin formula 7, iar indexul de start pentru adăugarea acestor valori se face cu formula 8. După aceste operații de modificare a valorilor pentru fiecare rezervare, matricea de disponibilități este completată.

$$\begin{aligned} totalTimeDivision = & (\text{parseInt}(\text{hoursOfOpp}.slice(6, 8)) \\ & - \text{parseInt}(\text{hoursOfOpp}.slice(0, 2))) \\ & * 12 + \text{parseInt}(\text{hoursOfOpp}.slice(9, 11))/5 + 1 \\ & - \text{parseInt}(\text{hoursOfOpp}.slice(3, 5))/5 \end{aligned} \quad (6)$$

$$activityTimeDivision = reservationTime/5 \quad (7)$$

$$\begin{aligned} startIndex = & (\text{parseInt}(\text{value.hour.slice}(0, 2)) - \text{parseInt}(\text{hoursOfOpp}.slice(0, 2))) \\ & * 12 + \text{parseInt}(\text{value.hour.slice}(3, 5))/5 \\ & - \text{parseInt}(\text{hoursOfOpp}.slice(3, 5))/5 \end{aligned} \quad (8)$$

Având în vedere că de cele mai multe ori o activitate are mai multe subdiviziuni, trebuie menționat că atunci când se caută disponibilitățile, există o listă de asemenea, vectori, fiecare vector reprezentând vectorul de disponibilități pentru fiecare subdiviziune.

Către utilizator se returnează toate orele la care își poate face rezervare. Acestea sunt întotdeauna din 15 în 15 minute. Algoritmul returnează ora în cazul în care găsește un număr de **activityTimeDivision** de 0 consecutivi începând cu indexul de start al orei.

O altă caracteristică a algoritmului este încercarea acestuia de a optimiza rezervările. Întotdeauna încearcă să găsească o disponibilitățि la subdiviziuni ale activității cu id-ul în ordine crescătoare, astfel întotdeauna va exista o continuitate. De asemenea, algoritmul tinde să încerce să lege o rezervare de alta, pentru a minimiza spațiile goale, în care se formează un timp mort.

6 EVALUARE

6.1 Obiective

Din punct de vedere al functionalitatilor, mă declar mulțumit; am reușit să implementez toate funcționalitățile dorite, inclusiv anumite elemente de securitate și scalabilitate, mărind performanțele aplicației. Atât algoritmii de recomandări cât și cel pentru rezervări functionaza fară probleme, astfel am reușit să rezolv problema de la care am plecat. De asemenea, am îndeplinit obiectivele estetice ale aplicației, implementând un design intelligent, bazat pe reguli fixe de UI, aplicația fiind funcțională ată pe Android cât și pe IOS. Din punct de vedere al performanțelor aplicația functionaza optim, iar serverul de asemenea; mai multe detalii referitoare la performanțe se găsesc mai jos, la subpunctul 6.3.

6.2 Utilitate

Din punct de vedere al utilității aplicația reprezintă un demo, fiind momentan doar în stadiu de proiect, fiind nevoie și de implementarea aplicației de gestionare pentru localuri; însă utilitatea acestei aplicații este destul de mare având în vedere că tratează o idee inovatoare, care ajută atât localurile prin digitalizarea și contorizarea rezervărilor, cât și utilizatorii care pot să își gestioneze mai ușor și rapid ieșirile în oraș, atât cu scop recreativ, cât și intelectual.

6.3 Performante

Performanțele aplicației sunt bune, fiind în limitele acceptate, atât pe partea de frontend cât și pe cea de backend. În imaginea 10 se pot observa graficele de performanță ale aplicației în Flutter, rulată pe emulator; se poate observa că rulează constant cu 60 imagini pe secundă, având mici depășiri de latență peste 16ms, ceea ce reprezinta un rezultat îmbucurător pentru o rulare virtuală, având resurse limitate. Din punct de vedere al serverului performanțele sunt pe măsură asteparilor, mare parte din requesturi sunt efectuare între 5-20 ms, ceea ce este un rezultat cu latență insesizabilă, iar cel mai costisitor din punct de vedere al timpului este algoritmul de recomandări care se execută în 170 ms, care de asemenea, este sub limita agrata de 200ms latență maximă pe request.

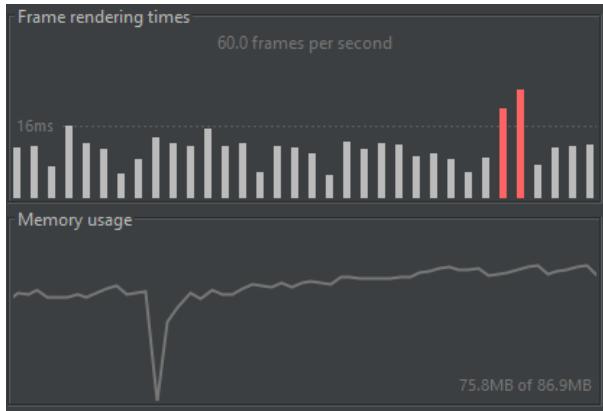


Figura 10: Performantele obtinute in emulator

6.4 Îmbunătățiri ulterioare

Intenționez continuarea dezvoltării acestui proiect și pe viitor, astfel, vor exista o serie de modificări care trebuie aduse în perioada următoare. În primul rând trebuie dezvoltată și aplicația pentru gestionarea localurilor. După finalizarea acestei aplicații, teoretic acest proiect poate fi pus în funcțiune, deoarece întreagă arhitectură ar fi finalizată.

Pentru deployment, serverul din NodeJS și baza de date trebuie încărcate în containere, cu ajutorul platformei Docker, ca mai apoi să fie încărcate pe o platformă de hostare pentru servere. Un alt aspect deja realizat este achiziționarea domeniului; am cumpărat domeniul **uerto.com** și **uerto.org** utilizând platforma NameCheap. Pentru partea de frontend, aplicațiile în format **.apk** și **.ipa** vor fi încărcate pe platformele AppStore și GooglePlay.

După lansare, primul pas este optimizarea arhitecturii de rezervări astfel încât serverul să nu fie suprasolicită. Acest proces este unul pe termen lung, având în vedere că fluxul de utilizatori noi nu este constant.

Pe viitor intenționez să dezvolt algoritmul de recomandări, utilizând o arhitectură mai optimă din punct de vedere al complexității de timp, iar ca funcționalități, tind să dezvolt acest sistem, astfel încât să ajute utilizatorii în găsirea unor programe pe mai multe ore, având ca input bugetul și tipul de activități dorite.

De asemenea, intenționez să adaug mai multe funcții de interacțiune între utilizatori, fiind începutul unui nou concept de social media, în care accentul se pune pe activități în comunitate.

7 CONCLUZII

Ca și concluzie, în cadrul proiectului meu de licență am dorit și reușit să implementez o soluție pentru un sistem automatizat de rezervări pentru localuri și un algoritm de recomandări. Problema inițială, de la care am plecat este reprezentată de lipsa digitalizării rezervărilor și lipsa de alternative în zone metropolitane mari, precum Bucuresti, unde anual există un tranzit imporant de oameni.

Pentru îndeplinirea acestui scop am dezvoltat o aplicație de mobil, utilizată atât pe Android cât și pe IOS. Am redus timpul de dezvoltare utilizând frameworkul Flutter care permite crearea unei aplicații care este randată pentru ambele sisteme de operare.

Datorită faptului că doresc dezvoltarea acestui proiect, am considerat scalabilitatea un factor cheie în arhitectura inițială. Astfel am creat un server în NodeJS legat la o bază de date relațională.

Aplicația mea reprezintă un avans față de tehnologiile prezente pe piață în acest moment, deoarece am introdus funcționalități unice ce ușurează experiența utilizatorilor.

Sistemul de recomandări functionază optim, utilizând trei rutine de predicție: Content-based filtering, Collaborative filtering și Linear regression. Acești algoritmi de machine learning compun un automat ce tranzitează dintr-o stare în alta în funcție de preferințele și ratingurile utilizatorilor.

Consider că aplicația dezvoltată de mine are un aport semnificativ în industria hospitalității și cea a divertismentului, deoarece prin sistemul de rezervări am reușit automatizarea acestui proces. De asemenea, această aplicație are scopul de a pune în competiție localurile, ceea ce nu poate decât să crească expunerea și rata de profitabilitate a acestora. Mai mult de atât, importanța aplicației este datorată metodelor de filtrare și recomandare, care ajută utilizatorii în alegeri cât mai pertinente. Astfel rata de utilizatori mulțumiți de serviciile furnizate crește, ceea ce este un dublu căștig. Din punct de vedere al evaluării aplicația funcționează în parametrii optimi, fiind o aplicație concepută pentru scalare.

În final, consider că, eforturile mele intelectuale depuse în ultimul an, în cadrul acestui proiect nu au fost de prisos, având la momentul actual un produs care îndeplinește toate cerințele initiale, bazate pe problemele existente pe piață. Astfel cred că aplicația, în ansamblul ei, a fost concepută și dezvoltată corect, luând în calcul resursele puse la dispoziție, având un plan clar în minte, cu targeturi realiste, nicidcum idealiste. În continuare voi dezvolta proiectul, având în plan continuarea arhitecturii în cadrul lucrării de disertație.

BIBLIOGRAFIE

- [1] Ankita Banerji. Manage user sessions. <https://www.analyticsvidhya.com/blog/2021/04/gradient-descent-in-linear-regression/>. Last accessed: 09 aprilie 2021.
- [2] Santiago Castro. 5 reasons why mysql is still the go-to database management system. <https://www.jobsity.com/blog/5-reasons-why-mysql-is-still-the-go-to-database-management-system>. Last accessed: 29 aprilie 2022.
- [3] Michael Dawson and Myles Borins. Node.js - quality with speed. <https://nodejs.org/en/blog/community/quality-with-speed>. Last accessed: 22 februarie 2017.
- [4] Michael Van den Bergh. React redux: Building modern web apps with the arcgis js api. <https://www.esri.com/arcgis-blog/products/3d-gis/3d-gis/react-redux-building-modern-web-apps-with-the-arcgis-js-api/>. Last accessed: 08 september 2017.
- [5] Firebase Official Documentation. Manage user sessions. <https://firebase.google.com/docs/auth/admin/manage-sessions>. Last accessed: 26 august 2022.
- [6] Filip Hracek. Your first flutter app. <https://codelabs.developers.google.com/codelabs/flutter-codelab-first#0>. Last accessed: 17 martie 2023.
- [7] Ipsos. Cum a influențat pandemia obiceiurile de cumpărare și ritualurile de mâncat în oraș. <https://www.ipsos.com/ro-ro/cum-influentat-pandemia-obiceiurile-de-cumparare-si-ritualurile-de-mancat-in-ora>. Last accessed: 5 februarie 2021.
- [8] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2 edition, 2014.
- [9] Alexandre Strzelewicz. pm2 - official library page. <https://www.npmjs.com/package/pm2>. Last accessed: 22 martie 2023.
- [10] Mirabela Tiron. Aeroporturile din românia au fost tranzitate în 2022 de 21 de milioane de pasageri, în creștere cu 87,5% față de anul anterior și triplu față de 2020. <https://www.zf.ro/companii/aeroporturile-din-romania-au-fost-tranzitate-in-2022-de-21-de-21551011>. Last accessed: 30 ianuarie 2023.

[11] Ihor Demedyuk & Nazar Tsybulskyi. Flutter vs react native vs native: Deep performance comparison. <https://inveritasoft.com/blog/flutter-vs-react-native-vs-native-deep-performance-comparison>. Last accessed: 29 iunie 2020.

ANEXE

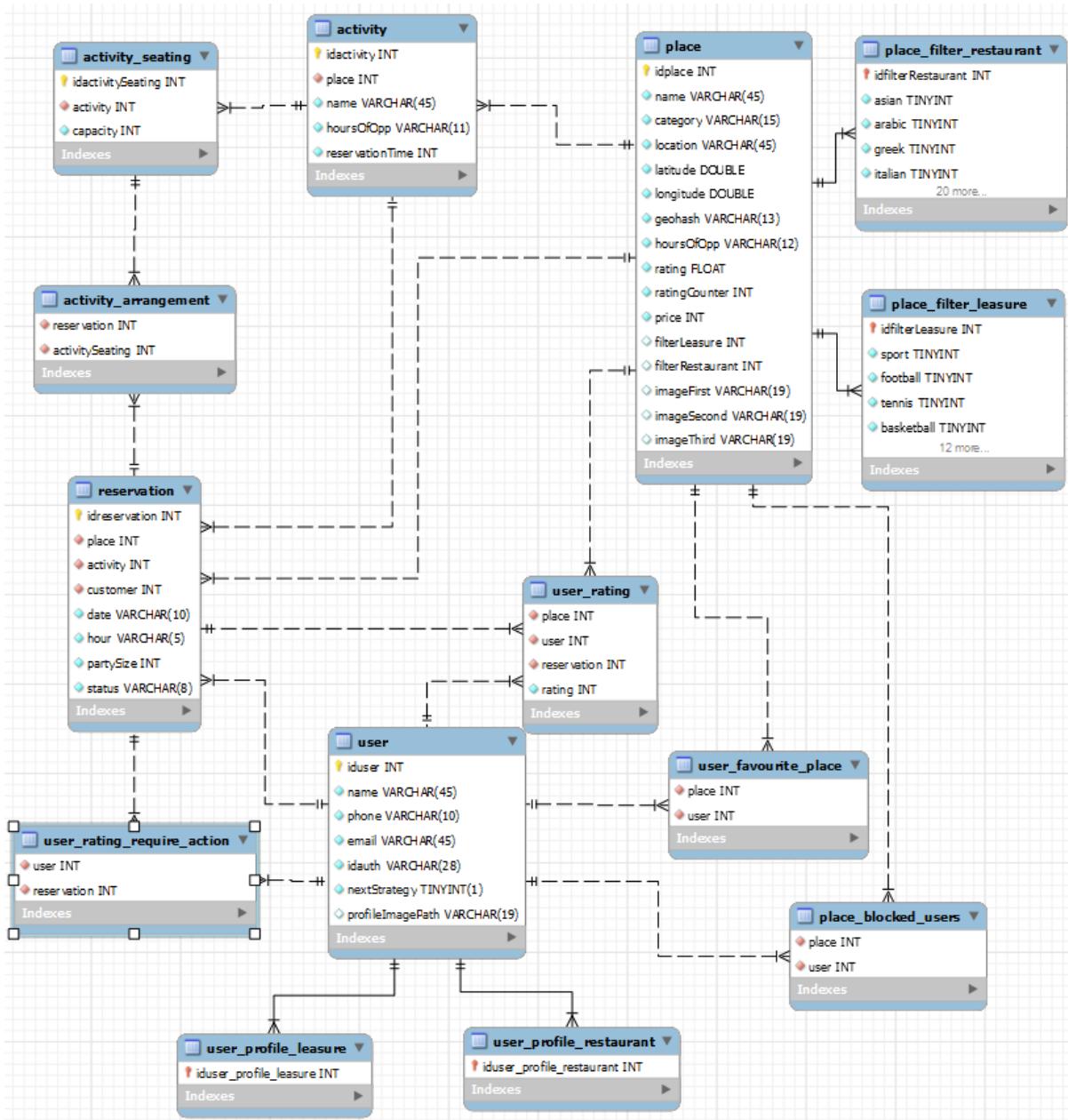


Figura 11: Diagrama basei de date

Tabela 3: Analiza SWOT

Analiza SWOT	
Strengths	Weaknesses
<ul style="list-style-type: none"> • Standardizarea sistemului de rezervări online. • Sistem de inteligență artificială pentru recomandarea altor localuri pe baza experiențelor anterioare. • Aplicația oferă fiecărui utilizator o experiență inedită pe baza profilului sau. • Utilizatorul poate căuta localuri folosind o gamă largă de query-uri, astfel eficacitatea găsirii unui loc pe plac crește. • Aplicația este valabilă pe toate platformele de mobile, astfel întreagă piață este acoperită. • Interfață grafică inteligență care conferă oricărei categorii de utilizator o experiență plăcută. • Aplicația are funcție de reminder pentru rezervări, astfel devine indirect un calendar personal pentru utilizator. 	<ul style="list-style-type: none"> • Există aplicații nișate pe anumite categorii precum Stailor pentru partea de saloane de înfrumusețare sau Eatapp pentru restaurante; aceste aplicații au anumite elemente particulare ca de exemplu pe aplicația Ialoc utilizatorul poate să se rezerve la o anumită masă a restaurantului. • Varianta inițială a aplicației nu are implementată posibilitatea de review pe baza de comentariu, ci doar pe baza de rating.
Opportunities	Threats
<ul style="list-style-type: none"> • Finalul pandemiei de COVID-19 a însemnat redeschiderea multor localuri cu o clientela incertă. • După doi ani de pauză o bună parte din oameni și-au regândit activitățile recreative. Un studiu realizat de Ipsos pe [7] spune că în Europa 58% din clienți frecventează mai rar localurile după pandemie. • Orașele cu populație în continuă creștere, precum București sau Cluj necesită o aplicație prin care utilizatorii să descopere mai ușor localuri, fiind o creștere cu 87% a tranzitantilor, asa cum reiese și din sursa [10]. • Tendință socială, din ultimii ani, de a digitaliza toate domeniile de activitate. • Scăderea forței de muncă în rândul joburilor fără o perspectivă de viitor. • Tendință socială de expunere tot mai acerbă, poate crea din acest proiect un instrument pentru monitorizarea activităților recreative. 	<ul style="list-style-type: none"> • O nouă criză medicală poate readuce carantină care să opreasă activitatea localurilor, deci aplicația își pierde din relevanță. • Mișcare de anti-digitalizare a unor conducători de localuri poate crea scepticism în prima instanță. • Criza financiară poate amâna o dorință de digitalizare la momentul actual.

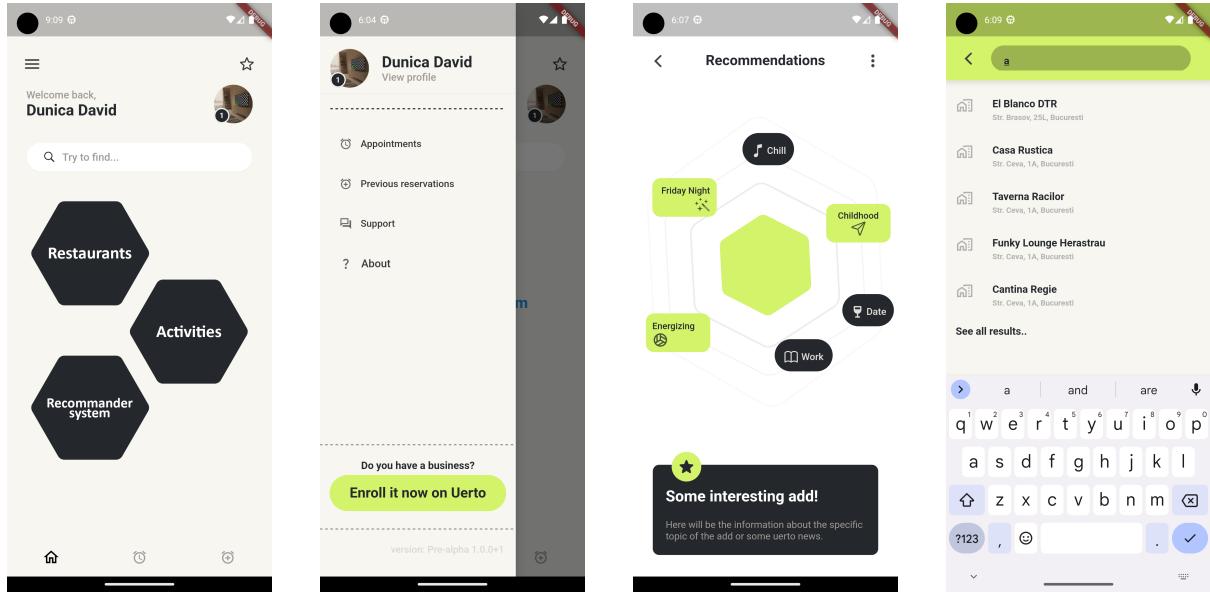


Figura 12: Main Page & Drawer & Recommendations Page & Search Page

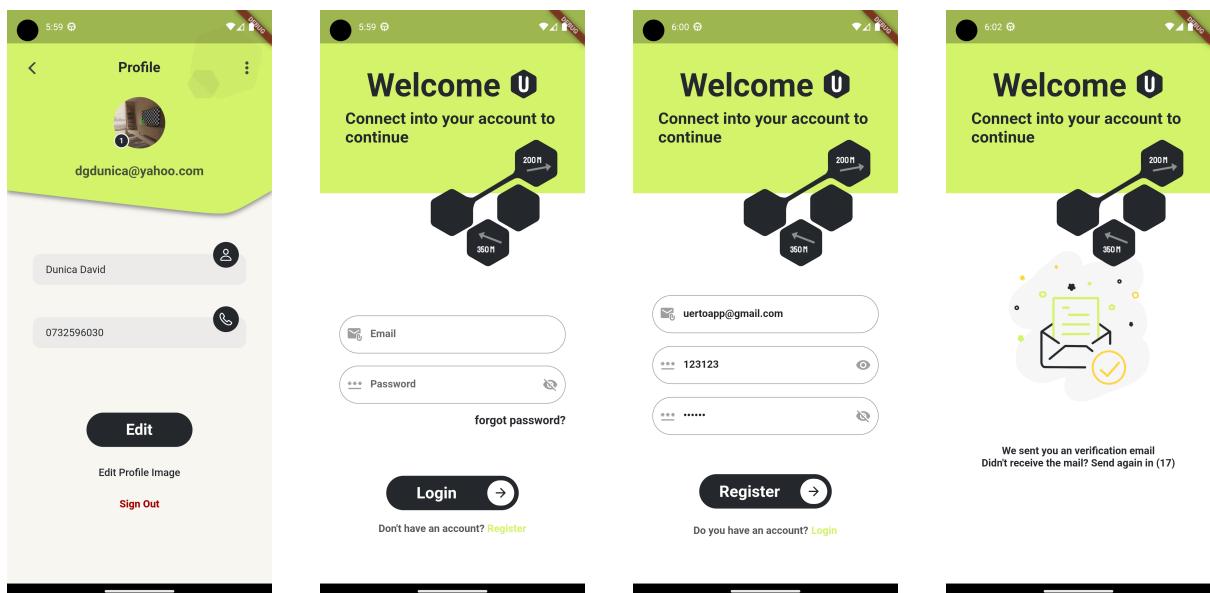


Figura 13: Edit Profile Page & Login Page & Register Page & Verify Email Page

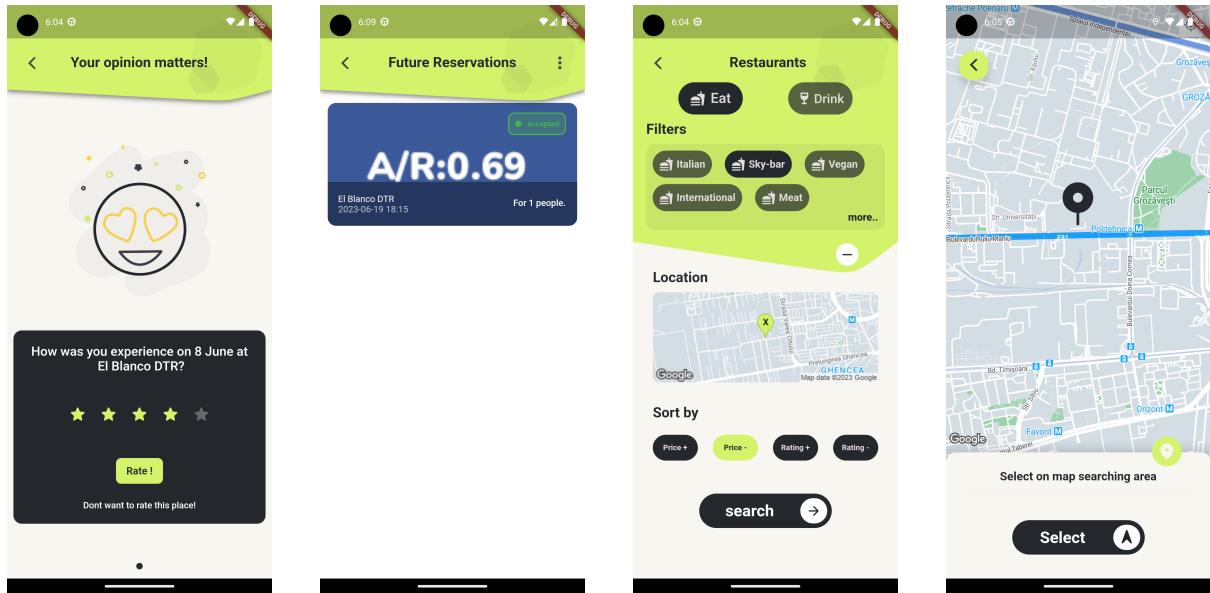


Figura 14: Rate Page & Reservations Page & Category Page & Location Pick Page

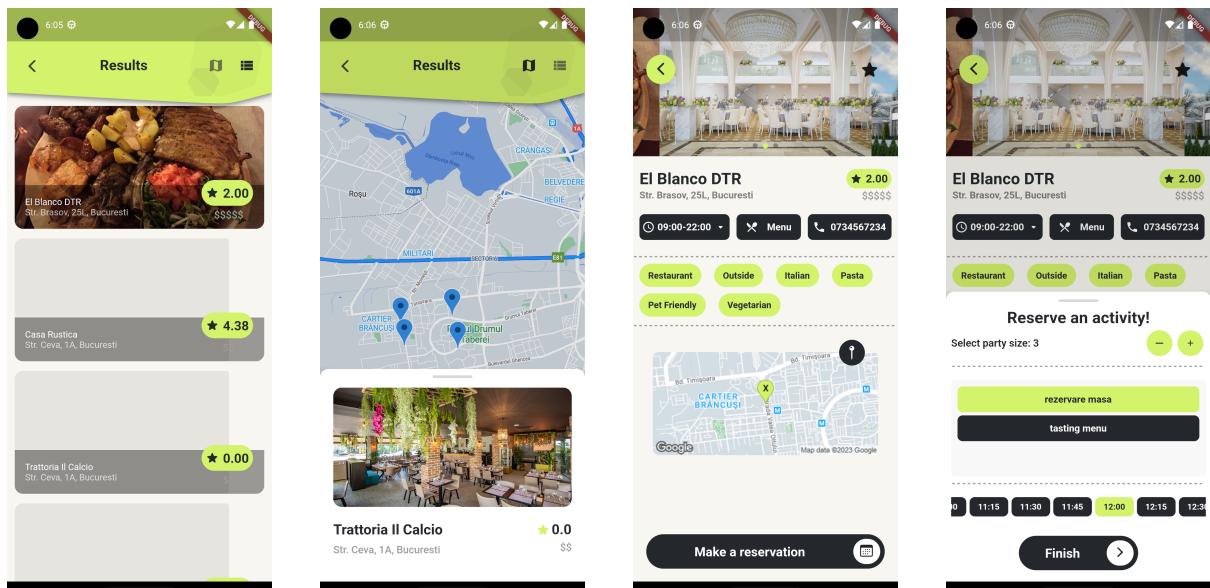


Figura 15: Results Page List & Results Page Map & Place Details Page & Place Details Page - reservation pop-up