

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAŞI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ



**MeetUP - aplicație pentru
rezervarea locurilor din localuri**

propusă de

Radu Mihai-Emilian

Sesiune: iunie, 2021

Coordonator științific,

Olariu Florin

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IASI

FACULTATEA DE INFORMATICA

MeetUP - aplicație pentru rezervarea locurilor din localuri

Radu Mihai-Emilian

Sesiune: iunie, 2021

Coordonator științific,

Florin Olariu

Avizat, Îndrumător Lucrare de Licență,

Florin Olariu

Data XX.XX.XXXX Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul Radu Mihai-Emilian, domiciliat în strada Dreptății, Numărul 34, Sat Moimești, Comuna Popricani, Județul Iași, născut la data de 31.10.1998, identificat prin CNP 1981031226763, absolvent al Universității „Alexandru Ioan Cuza” din Iași, Facultatea de Informatică, specializarea Română, promoția 2020, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art.326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul: *MeetUP - aplicație pentru rezervarea locurilor din localuri* elaborată sub îndrumarea dl. Olariu Florin, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consumând inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării fasificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată, ci reprezintă rodul cercetării pe care am întreprins-o.

Data,

XX.XX.XXXX

Semnătură student,

DECLARAȚIE DE CONSUMĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul *MeetUP - aplicație pentru rezervarea locurilor din localuri*, codul sursă al programelor și celealte conținuturi (grafice, multimedia, date de test etc.) care însotesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, XX.XX.XXXX

Absolvent,

Radu Mihai-Emilian

Introducere	6
Contribuții	8
Capitolul 1 - Descrierea problemei	10
Capitolul 2 - Abordări anterioare	11
Capitolul 3 - Descrierea soluției	13
3.1 - Arhitectura	13
3.2 - Bază de date	17
<i>3.2.1 - Motivația alegerii PostgreSQL</i>	<i>17</i>
<i>3.2.2. - Arhitectura bazei de date</i>	<i>19</i>
3.3 - Tehnologii	21
3.4 - Detalii de implementare	25
<i>3.4.1 - Modulul de înregistrare al administratorilor</i>	<i>25</i>
<i>3.4.2 - Modulul de autentificare</i>	<i>26</i>
<i>3.4.3 - Modulul Explore</i>	<i>28</i>
<i>3.4.4 - Modulul History și Modulul Reviews</i>	<i>33</i>
<i>3.4.5 - Modulul Friends</i>	<i>34</i>
<i>3.4.6 - Modulul Profile</i>	<i>35</i>
<i>3.4.6 - Back-end</i>	<i>36</i>
<i>3.4.7 - Servicii adiționale</i>	<i>41</i>
Concluzii	42
Direcții de viitor	43
Bibliografie	44

Introducere

Privind în retrospectivă asupra schimbărilor provenite de la lansarea primului iPhone în 2007 și, implicit, punerea la dispoziție a populației generale a unui dispozitiv conectat în permanentă la internet, putem afirma că faptul că societatea în care trăim astăzi reprezintă un sistem de procese și sisteme interconectate - de la sistemul medical, juridic și legislativ, până la cumpărături online, streaming de muzică și filme și Internet of Things; toate au la baza internetul. Luând cont de această afirmație, devine mult mai clar procesul rapid prin care trecem numit digitalizare.

Un mijloc important și definiitoriu prin care adopția și migrarea majorității serviciilor în mediul online este reprezentat de magazinele de aplicații - *App Store și Play Store*, care au deschis orizonturi noi de explorat pentru servicii și firme existente, cât și pentru start-up-uri. Inițial, posesorii de smartphones au gravitat către aplicații de divertisment - precum jocurile, reprezentând, astfel, categoria majoritară. Însă, în scurt timp, datorită faptului că din ce în ce mai mulți oameni ajungeau în posesia unui smartphone, iar rețelele de socializare erau în plin proces de adopție, platforme precum *Facebook*, *Instagram*, *Tinder*; au devenit nouă atracție.

În toată această perioadă, observându-se potențialul pe care îl oferă atât în materie de date cât și de putere de procesare și funcționalități, această nouă piață liberă apărută datorită telefoanelor inteligente, care are deja adopție de peste 45% din populația globală, a devenit plină de aplicații care ne facilitează viață de zi cu zi. Acum, putem folosi telefonul pentru a comanda mâncare (*FoodPanda*, *Glovo*), pentru a chama un taxi (*Uber*, *Bolt*), chiar și pentru a face plăti contactless (*Apple Pay*, *Google Pay*).

Observând această tendință, cât și din experiențe personale, am dorit să dezvolt aplicația de mobil MeetUP, pe care o să o prezint în lucrarea de licență, o aplicație ce are că scop simplificarea și digitalizarea ieșitului în oraș. Fie că e vorba de o ieșire cu prietenii sau o întâlnire de afaceri, am vrut prin soluția mea să schimb procesul prin care se fac rezervările, eliminând din ecuație apelul pe care trebuie să îl faci către un local sau căutarea îndelungată a unui loc la masă atunci când este aglomerat.

Aplicația dezvoltată de mine propune o soluție atât pentru clienți, cât și pentru deținătorii de localuri și personalul localului. Aplicația propune deținătorilor de afaceri o platformă în care își pot adauga localul, în care își pot defini personalul, meniul, programul și să adauge poze cu locația. De asemenea, cel mai importantă parte este definirea unor zone, cu mese pe baza unui grid. Aceste mese au drept caracteristici un nume, un număr de locuri și un chelner. Din punctul de vedere al chelnerilor, aceștia prin contul creat de către deținător, vizualizează și administrează fiecare masă la care sunt atribuiți - confirmă și finalizează rezervări, modifică și blochează anumite mese.

Utilizatorii principali ai aplicației - clienții, au la dispoziție o pagină în care, în funcție de distanță la care se află, le sunt afișate localurile deschise din proximitate. Aceștia mai au de asemenea posibilitatea de a-și seta profilul și de a face rezervări într-o locație la alegere la care își pot invita prietenii, pe care ii pot adaugă tot din aplicație.

În continuare, în lucrarea de licență *MeetUP - aplicație pentru rezervarea locurilor din localuri*, o să prezint motivația de a implementa această platformă în **capitolul 1 — Descrierea problemei**, după care voi face o analiză a soluțiilor deja existente în **capitolul 2 — Abordări anterioare**, finalul conține o analiză comprehensivă asupra modului de implementare, incluzând detalii arhitecturale și tehnologice despre produsul dezvoltat, în **capitolul 3 — Descrierea soluției**.

Contribuții

Contribuția pe care am avut-o în această lucrare constă în implementarea unei soluții alcătuită dintr-un client - o aplicație dezvoltată în *React Native* și două servere - un server ce expune un *API GraphQL* pentru a facilita comunicarea între front-end și baza de date relațională *PostgreSQL*; și un server de web în *React* pentru înregistrarea deținătorilor de localuri într-un mod securizat.

În dezvoltarea aplicației de mobil, am luat în considerare ghidul de dezvoltare a unei interfețe scris de către cei de la Apple [1]. Acest ghid aduce atât constrângeri cât și anumite principii ce fac experiența persoanelor care interacționează cu produsul final una cât mai ușoară și plăcută.

Pe lângă acest aspect pe care l-am luat în considerare în dezvoltarea interfeței, am luat în considerare și folosirea unor tehnologii moderne, aspect care aduce un aport semnificativ în validitatea și actualitatea produsului. *React Native* este framework-ul folosit pe partea de client, ce facilitează dezvoltarea de aplicații pe *Android* și *iOS* într-un singur limbaj, este dovedit drept un pionier în contextul mobile development-ului. *GraphQL* fiind o tehnologie destul de nouă, proiectată pentru construirea de API-uri scalabile, stabile, cu o arhitectură solidă, folosit pe partea de server, asigură sustenabilitatea și viteza necesară de răspuns de care orice aplicație ce se adresează unui număr mare de utilizatori are nevoie.

Una dintre contribuțiile importante pe care le-am avut în dezvoltarea aplicației a fost integrarea acestoria cu multe librării și tehnologii externe cu scopul de a avea un produs cât mai aproape de unul final. Pentru a ajunge la acest rezultat, a fost necesar să implementez un sistem automatizat de deploy astfel încât la fiecare modificare a codului și urcarea acestuia pe repository-ul de *Git*, modificările să se propage instant către client și server. Dezvoltând un script de automatizare pentru *Github Actions* folosind *Fastlane*, care se ocupă de distribuirea aplicației în *App Store*, iar alt script automat ce face publicarea modificărilor API-ului *GraphQL* și bazei de date *PostgreSQL* către hosting service-ul *Heroku*, am creat un mediu de dezvoltare continuu și ușor de înțeles pentru orice programator.

Pe lângă aceste tehnologii și sisteme de automatizare pe care le-am dezvoltat ce doresc să faciliteze procesul de dezvoltare și integrare continuă (denumit *CI/CD*), am integrat și diverse resurse și *API*-uri externe pentru experiență cât mai fluidă a utilizatorilor proiectului - precum *OneSignal* ca sistem de notificări pe client în timp real, *Apple Maps* și *Google Maps API* pentru localizare și calcularea distanței între client și local, *SendGrid* pentru trimiterea de email-uri automate, *BranchIO* pentru distribuirea de link-uri în-app, *LocationIQ* pentru transformarea coordonatelor de tip latitudine-longitudine în adresa reală și multe altele pe care le voi dezvolta în capitolele următoare.

Capitolul 1 - Descrierea problemei

Lucrând ca și chelner pe perioada studenției, unul dintre aspectele care împiedică desfășurarea optimă de servire a clienților era reprezentat de persoanele care sunau la telefon cu cererea de a face o rezervare. Asta presupune pentru un chelner într-o seară aglomerată pe lângă faptul că trebuie să memoreze comenziile, plătile, preferințele clienților și aspectele administrative precum stocul și produsele, să cunoască și faptul dacă o masă este liberă sau nu. Pe lângă apelurile telefonice de la potențiali vizitatori, de multe ori acestora li se alăturau persoanele care veneau fizic cu aceeași întrebare.

Rezervările pe care reușesc să le confirme personalul localului, însă, sunt tot limitate de eroarea umană. Personal, s-a întâmplat de multe ori să uit de faptul că la o masă am o rezervare, la care între timp s-a așezat altcineva și trebuia să rezolv situația. Faptul că toate aceste date erau stocate cu pixul pe foaie nu facilita deloc vizualizarea statisticilor, evaluarea performanței și determinarea nivelului de ocupare al localului.

Înând cont de experiențele personale, ideea de a crea un astfel de sistem a pornit încă de atunci. Însă, datorită contextului pandemic în care revenim la o oarecare normalitate e mai important decât oricând prezența unui astfel de sistem ușor de utilizat de către oricine pentru a ne putea proteja și a decide dacă dorim să petrecem câteva ore într-un loc aglomerat sau mai retras.

Astfel, pot afirma că această aplicație reprezintă o soluție viabilă pentru problema prezentată cu care se confruntă încă industria, lovită mai ales de contextul ultimului an. MeetUP își propune să vină în ajutorul restaurantelor, barurilor și cafenelelor în mod gratuit, aducând un beneficiu atât temporar, prin a facilita respectarea măsurilor de siguranță, cât și de lungă durată, prin sistemul robust prin care clienții pot foarte ușor să își rezerve o masă în locul lor preferat fără a mai fi necesară programarea telefonică.

Capitolul 2 - Abordări anterioare

După conturarea ideii subiectului, am constatat că pe piață actuală sunt prezente anumite soluții similare, ceea ce a presupus dezvoltarea unor particularități pentru a diferenția lucrarea mea de produsele lansate. Astfel, am făcut o analiză a modelului soluțiilor existente a două firme cu scopul de a găsi un mijloc inovator de abordare.



Figura 2.1 - OpenTable platform [2]

Prima analiză a fost făcută asupra produsului *OpenTable*, o soluție dezvoltată încă din 1998, deținută de Booking [2]. Aplicația este adoptată vast în Statele Unite deoarece este dezvoltată în California. Am observat ca prezența pe teritoriul România este mică, doar unele restaurante și cafenele din capitală folosind platforma. Prezența scăzută la nivel local este cauzată de necesitatea de a achiziționa echipament și de subșcripția lunară plătită către firma americană. Multe localuri nu au bugetul necesar să se integreze cu acest sistem, iar soluția propusă de mine este una ce nu presupune investiții suplimentare, deoarece necesită doar un smartphone pe care majoritatea angajaților îl dețin. Pe lângă asta, soluția propusă va fi gratuită, profitabilitatea fiind garantată de reclame și promoții.



Figura 2.2 - Eat App platform [3]

Eat App, cea de a doua opțiune la care am cercetat modelul de business, este o aplicație dezvoltată în 2015 [3], ce are drept clientelă principală Emiratele Arabe și Orientul Mijlociu. Acest model este mai ușor de implementat față de cel anterior, însă costurile mari de utilizare, lipsa de prezență teritorială și de suport, sunt un impediment mare în extinderea acestuia. De asemenea, acest model nu prezintă rolul de *chelner* pe care îl poate avea un utilizator, configurație care în lucrarea mea este luată în considerare.

Pe lângă diferențele de preț și echipament, soluția propusă de mine aduce un aspect social al acestui sistem prin implementarea modulului de *Prietenii*, modul prin care i se permite unui client să își invite prietenii. La rezervarea unei mese, în funcție de numărul de locuri, unui utilizator i se permite adăugarea prietenilor care au un cont creat pe platformă, ca mai apoi aceștia să primească o notificare și un email cu detaliile necesare, care sunt vizibile și în aplicație. Acest pas nu este unul obligatoriu, deci nu se distanțează de la conceptul de bază al soluțiilor prezentate anterior, însă vine ca un plus ce aduce un plus de personalitate și unicitate aplicației MeetUP.

Partea socială ce diferențiază lucrarea de soluțiile prezentate anterior este reflectată și în denumire. MeetUP sugerează o lejeritate și ideea de întâlnire între prieteni, pe când OpenTable reflectă rigiditate și industrialitate, iar EatApp poate reflecta faptul că este o aplicație strict pentru a consuma produse gătite - precum livrare de alimente sau carte de rețete digitală. Jovialitatea pe care am ilustrat-o în alegerea numelui proiectului este prezentă peste tot în interfață - de la animații și până la paleta cromatică.

Capitolul 3 - Descrierea soluției

Acest capitol cuprinde toate particularitățile de dezvoltare, cât și deciziile arhitecturale pe care le-am aplicat în realizarea produsului

3.1 - Arhitectura

Soluția dezvoltată are la bază patru elemente principale - aplicația de mobil scrisă cu ajutorul framework-ului React Native, site-ul de înregistrare al utilizatorilor cu rolul de “Owner”, API-ul pe care îl consumă cele două servere scris în GraphQL și baza de date relațională la care acesta din urmă realizează operații de tipul CRUD, reprezentată de PostgreSQL.

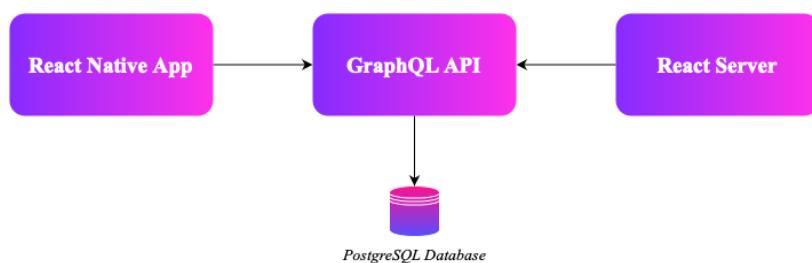


Figura 3.1.1 - Diagramă abstractă a serverelor

Pe departe cea mai simplă arhitectură o constituie server-ul de înregistrare al utilizatorilor de tip “Owner”. Aceasta este un simplu server web cu unicul scop de a limita accesul persoanelor care vor să adauge localuri pentru a evita încărcarea bazei de date cu date invalide care nu aduc nici un beneficiu utilizatorilor. Acest server este reprezentat de un singur view - ecranul de înregistrare, care folosește Apollo Client ca metodă de trimis și preluare a datelor din formularul din view către back-end. Acest proces este securizat prin utilizarea unui JWT care cripteză datele cu ajutorul unei chei stocate în fisierul .env.

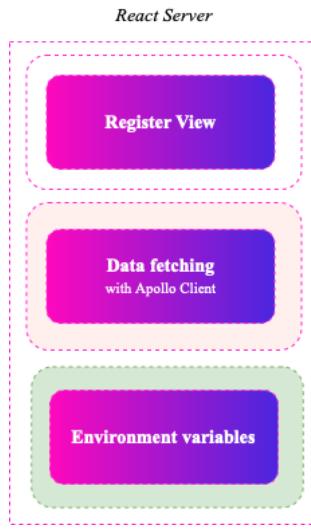


Figura 3.1.2 - Arhitectura site-ului de înregistrare

În contrast complet, back-end-ul este structurat pe trei subcategorii - partea logică, partea relațională și partea de servicii externe. Partea logică este alcătuită din modelarea făcută prin intermediul librăriilor Nexus (pentru validarea tipurilor), Prisma (pentru schema bazei de date și migrări) și GraphQL query types (Queries, Subscriptions, Mutations). Acestea alcătuiesc API-ul de GraphQL ce comunică cu baza de date prin al doilea layer, cel relațional. Aici se găsesc toate operațiile pe care le pune la dispoziție API-ul, cât și măsurile de validare de care dispune GraphQL în adiție a celor furnizate de Nexus. Asta garantează că indiferent de originea request-ului și tipul de date, putem găsi și preveni atacurile dinainte să afecteze baza de date Postgres.

În alăturarea acestor tehnologii se află servicii externe precum sistemul de notificări native implementat cu OneSignal, cel de trimitere a mail-urilor cu ajutorul SendGrid și cel de calculare și prelucrare a datelor de geolocație cu ajutorul LocalizationIQ.

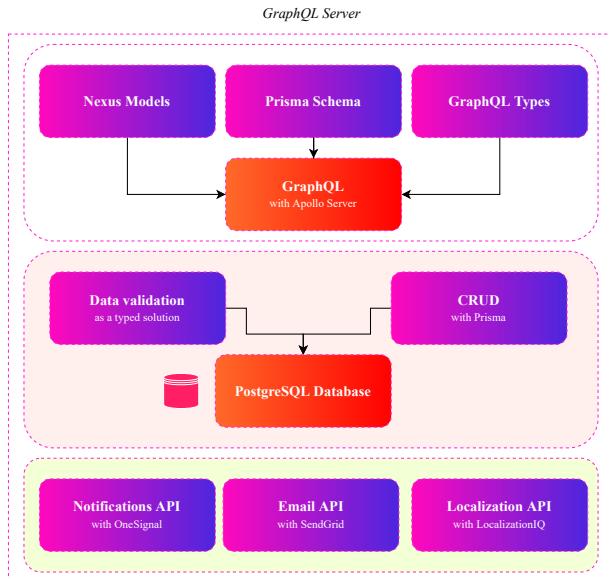


Figura 3.1.3 - Arhitectura back-end-ului

Arhitectura aplicației de mobil prezintă pe langă partile de vizualizare, data management și servicii externe, partea de permisiuni esențială în funcționarea corectă a aplicației. Fără acordarea acestor permisiuni, accesul către modulele importante este restricționată. Proiectul este structurat conform design pattern-ului MVC (Model View Controller). Primul strat reprezintă User Interface-ul, care poate fi de trei tipuri în funcție de rolul pe care îl are utilizatorul. Acest aspect de routing este rezolvat prin React Navigation care permite afișarea condițională de ecrane către utilizator. Partea de management de date este alcătuită din soluțiile de preluare și stocare a datelor primite de la back-end furnizate de către Apollo Client, ceea ce oferă dezvoltatorului accesul la un black-box în care se memorează răspunsurile primite, sub forma unui Redux Store. Pe client, spre deosebire de server, se substituie partea de mailing cu un serviciu de URL sharing, însă structura serviciilor externe rămâne similară.



Figura 3.1.4 - Arhitectura front-end-ului

3.2 - Bază de date

Soluția propusă de mine are drept bază de date PostgreSQL. Am decis să folosesc această bază de date după ce două dintre soluțiile încercate nu s-au pliat pe nevoile de dezvoltare. În următoarele subcapitole o să prezint motivația și arhitectura bazei de date.

3.2.1 - Motivația alegerii PostgreSQL

Inițial, am optat pentru o bază de date folosind MongoDB, apoi din cauza limitărilor pe care le avea versiunea pe care o foloseam, am făcut o migrare temporară către SQLite până la găsirea unei soluții mai potrivite. În final, am decis să folosesc PostgreSQL deoarece Object-Relational Mapping-ul de care dispune framework-ul în care am implementat server-ul, cât și soluția de hosting, dispun de suport complet. PostgreSQL, cunoscut și sub numele de Postgres, este cel mai avansat sistem de baze de date open source, cu cea mai bună medie de performanță, conform unui studiu din 2018.

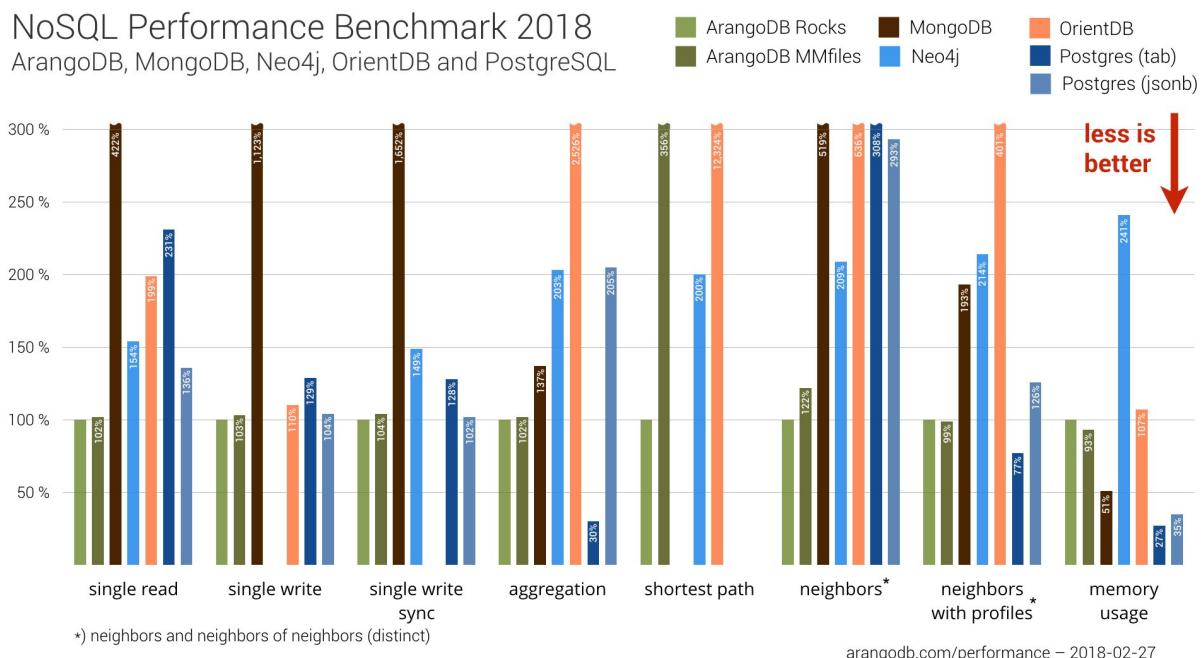


Figura 3.2.1 - Comparație a vitezelor bazelor de date [4]

Postgres beneficiază de o reputație puternică datorită fiabilității, integrității datelor, extensibilității, deciziilor arhitecturale și implicării în comunitatea open-source, ceea ce asigură livrarea consistentă a celor mai inovative și performante soluții. Această reputație este creata în peste 30 de ani de dezvoltare.

Postgres permite pe lângă definirea de tipuri standard SQL, definirea unor tipuri proprii de date. Toate categoriile suportate de tipuri de date ale acestei baze de date sunt:

- **Primitive:** Integer, Number, String, Boolean;
- **Structurate:** Date/Time, Array, Range, UUID
- **Document:** JSON, XML
- **Geometry:** Point, Line, Circle, Polygon

Un alt aspect important care a avut impact în alegerea acestei baze de date este integritatea datelor. Având o structură complexă arhitecturală, alegerea unei soluții de tip SQL era obligatorie, iar cea mai bună soluție din punctul de vedere al securității este Postgres, deoarece oferă validare a datelor de orice tip, inclusiv la nivel de JSON. Astfel, dacă se încearcă introducerea de date necorespunzătoare, se va returna mereu o eroare explicită. Din punctul de vedere al performanței, comparativ cu MongoDB, Postgres este deținătorul datorită sistemului dezvoltat de ei denumit *Multiversion Concurrency Control*, ce asigură accesul concurrent al interogărilor prin folosirea memoriei tranzacționale.

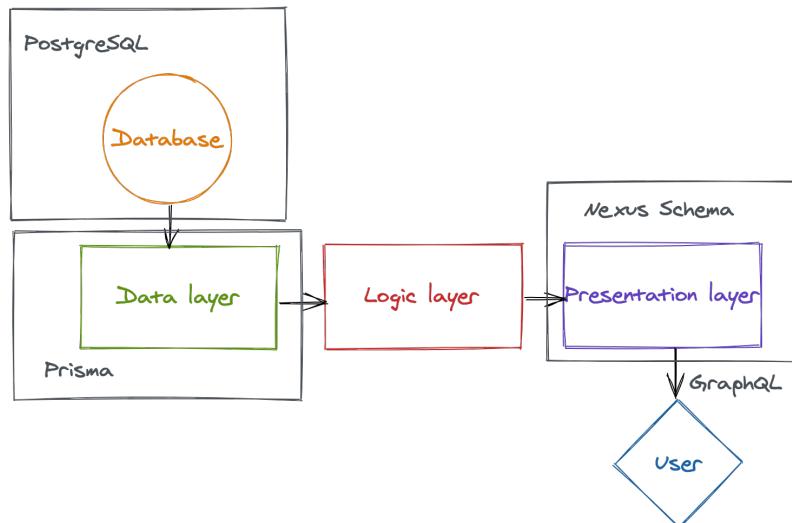


Figura 3.2.2 - Arhitectura logica a bazei de date [5]

In *Figura 3.2.2*, putem observa cum toate librăriile de NodeJS sunt utilizate pentru a permite a expune CRUD-ul către utilizatorii aplicației.

3.2.2. - Arhitectura bazei de date

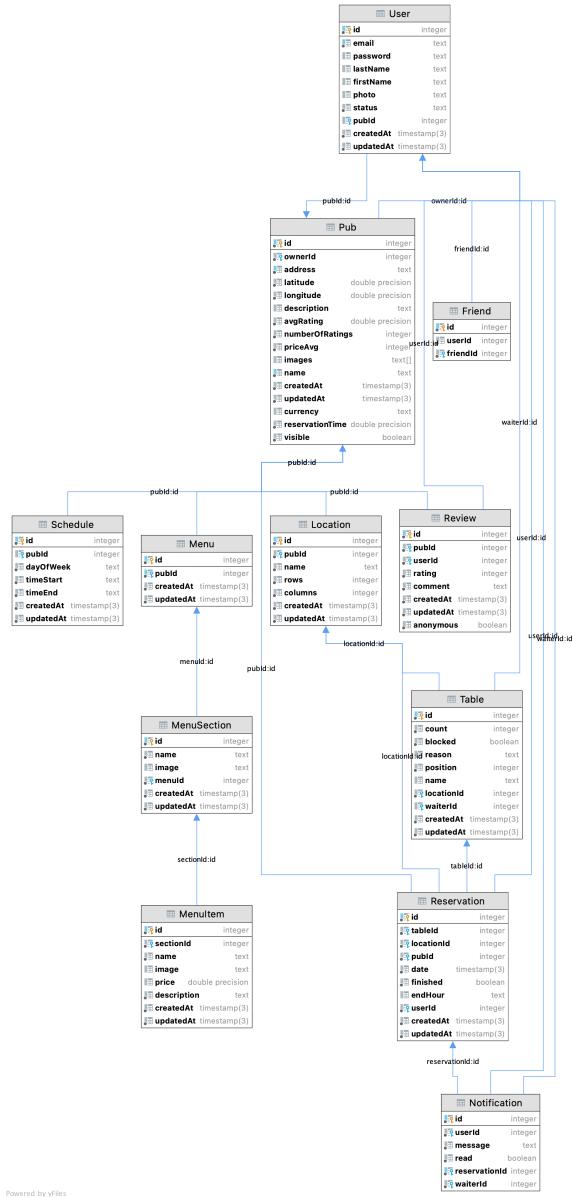


Figura 3.3.3 - Arhitectura bazei de date

Baza de date din spatele produsului final este alcătuită din 12 tabele, după cum se poate observa și în *Figura 3.3.3*. Toate tablele conțin o cheie primară, sub forma unui ID unic. Structura bazei de date este generată de către librăria Prisma. Cele mai importante

tabele sunt cele de *User*, de *Pub* și cel de *Reservation*. În jurul acestor tabele sunt orientate tot restul entităților. Spre exemplu, fiecare entitate de tip *Pub* are o relație de one-to-one către un *Menu*, fiind unite prin cheia secundară *pubId*. La rândul său, entitatea *Menu* prezintă o relație de one-to-many către *MenuSection*, ce reprezintă o categorie din meniul definit de către creatorul locației. Toate aceste relații co-dependente respectă forma normală 3, mai specific versiunea Boyce-Codd.

3.3 - Tehnologii

După cum am afirmat anterior, în implementarea aplicației MeetUP au fost utilizate tehnologii moderne care să asigure o experiență de utilizare eficace, stabilă, scalabilă, o modalitate de dezvoltare continuă și de extindere a conceptului de bază. Aceste tehnologii sunt:

- React
- React Native
- GraphQL
- Apollo Client & Apollo Server
- Prisma
- Nexus
- OneSignal
- BranchIO
- LocationIQ
- Firebase
- Google Cloud
- Fastlane
- Github Actions
- SendGrid
- Heroku

React este o librărie de JavaScript ce permite dezvoltatorilor să creeze aplicații web. Este o librarie bazată pe de componente ce au principii de life-cycle. Asta înseamnă că orice parte din produs este controlabilă și de sine statătoare. Această librărie este una dintre cei 3 competitori aflați pe piață la momentul actual pentru web, pe lângă Vue și Angular.

React Native este un framework de JavaScript ce vine ca soluție pentru a scrie aplicații de mobil native pentru iOS și Android. După cum sugerează și numele, este bazat pe React. Avantajele pe care le prezintă React Native sunt reprezentate de faptul că poți scrie o singura dată cod JavaScript, cu care dezvoltatorii de aplicații web sunt deja obișnuiți, atât pe iOS cât

și pe Android, timpul de dezvoltare reducându-se semnificativ; majoritatea codului scris în React este reutilizabil și pe mobile; oferă acces la codul nativ în cazul în care sunt necesare modificări complexe (ex: logare cu amprentă); dispune de o comunitate mare ce este la curent cu noile capabilități pe care le lansează fiecare platformă de mobile în parte.

GraphQL este un limbaj de interogare pentru API-uri (Application Programming Interface). Acest limbaj dispune de expunerea unei descrieri extensive a datelor, fiind strongly-typed, și permite clientului să ceară exact datele de care are nevoie fără a fi necesară implementarea unor endpoint-uri noi.

Apollo Client & Apollo Server reprezintă o platformă ce folosește GraphQL pentru a transfera date între server și client. În ultimul timp, implementarea unui server de GraphQL presupune implicit folosirea acestei platforme pentru dezvoltare. Această reușită se datorează faptului că este open-source și permite programatorului o instanțiere rapidă a unui proiect nou. Asemenea unui server Express (un framework des întâlnit pentru crearea unui API în NodeJS), acesta poate fi consumat de o varietate de clienți, cu toate că nu respectă standardele HTTP - GraphQL prin definiție dispune de un singur endpoint care are tipul “POST”. Această particularitate modifică structura originală de endpoint, datele cerute de client sunt cerute astfel în body-ul request-ului.

Prisma, așa cum menționam și în capitolul **Bază de Date**, este un Object Relațional Mapping ce facilitează modelarea datelor și preluarea acestora din baza de date, scris în NodeJS cu TypeScript. Un echivalent al acestui ORM este JPA, care furnizează aceeași funcționalitate pentru limbajul Java. Prisma dispune de funcționalități avansate precum paginare integrată, API tranzacțional, acces deschis prin interogări SQL la baza de date, tipuri de date native, suport pentru middleware (ex: autentificare, criptare), suport pentru JSON și multe altele. Un avantaj puternic pe care îl prezintă Prisma este API-ul ușor de înțeles pentru cei care cunosc limbajul SQL.

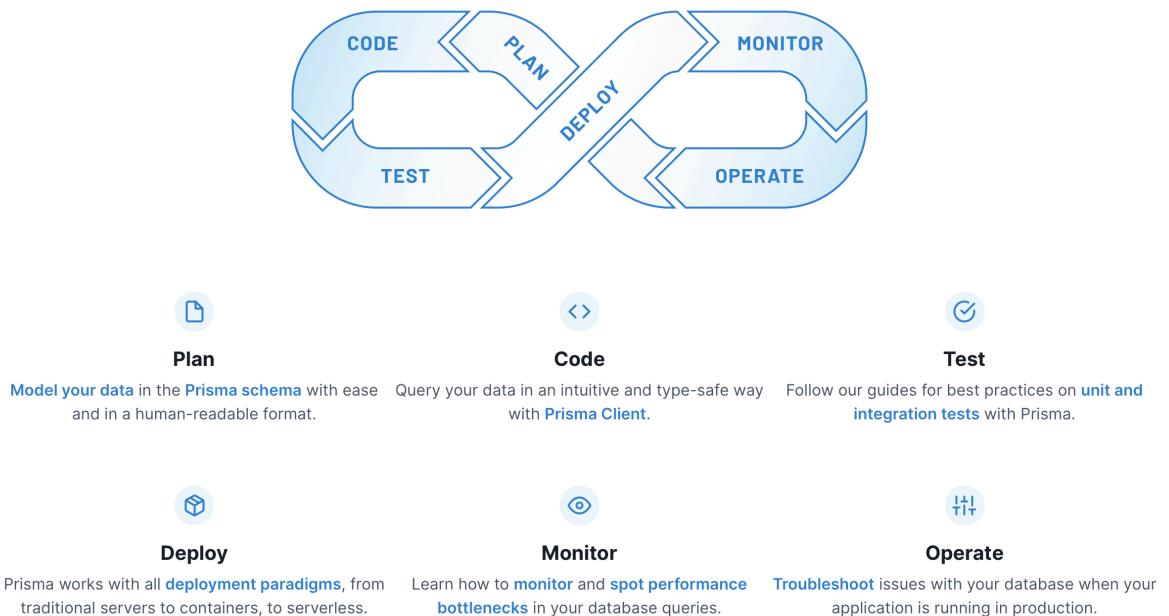


Figura 3.3.1 - Prisma în dezvoltarea aplicațiilor [6]

Nexus este o librărie ce ajută la modelarea schemei GraphQL în interfețe strongly-typed. Deoarece JavaScript este un limbaj de tip script, iar GraphQL este orientat-obiect, pentru a respecta standardele se folosește Nexus. Acest “layer” se asigură că datele provenite din baza de date prin intermediul Prisma sunt conforme cu schema definită de GraphQL sau cu datele cerute de client. În principiu, este o modalitate de a validare și extindere a tipurilor de date definite. În lucrarea prezentă, Nexus are rolul de a extinde modelul “Pubs” pentru a include distanța dintre un utilizator și un local care nu este stocată în baza de date.

OneSignal este un serviciu ce se ocupă cu trimiterea de notificări în timp real de tipul “push” către aplicațiile de mobil. Acest serviciu dispune de un API care este apelat de către server-ul de GraphQL și de un SDK care este integrat în aplicația de React Native unde clientul este identificat pe bază de email.

BranchIO este un serviciu ce permite deschiderea de link-uri web în aplicații de mobil. În mod normal, pe mobil atunci când deschizi un link dintr-un mesaj, sistemul de operare deschide o nouă pagină în browser-ul prestabilit pentru a vizualiza conținutul. Cu ajutorul serviciului BranchIO putem redirectiona utilizatorul către aplicația de mobil atunci când apăsăm pe un link generat din React Native.

LocationIQ este un API gratuit și open-source ce permite transformarea unor date de tipul latitudine, longitudine în adresă completă.

Firebase este un serviciu complet de dezvoltare în cloud a aplicațiilor. În cadrul proiectului suite de servicii oferita de Google a fost folosită ca modalitate de distribuire a aplicației, pentru stocarea imaginilor, ca modalitate de vizualizare a bug-urilor care cauzează crash-uri pe dispozitive și ca modalitate de studiu a statisticilor comportamentului utilizatorilor.

Google Cloud este o soluție de cloud extensivă ce ne permite cloud processing, cloud hosting, cloud functions și multe alte soluții complete de cloud. În MeetUP a fost folosit pentru semnarea cheilor cu care se face interacțiunea cu Firebase, pentru a ne asigura că toate request-urile sunt protejate de atacuri cibernetice.

Fastlane este o metodă de automatizare a procesului de distribuire de aplicații mobile. În general, acest proces de distribuire, fie către testare, fie către producție, consumă foarte mult timp și resurse. Fastlane este o soluție la această problema propunând o externalizare a procesului de distribuire prin crearea unui script ce face automat pașii pe care ar trebui să ii facă un programator în caz contrar.

Deoarece Fastlane permite mutarea procesului de distribuire din mâinile programatorului într-un proces automat, Github Actions este următorul pas pentru completarea acestui proces. Prin scrierea unui script ce rulează automat la fiecare push pe repository, în anumite condiții, putem integra Github Actions cu Fastlane și astfel să avem un build al aplicației distribuit fără a fi nevoie de intervenția programatorului.

SendGrid este o soluție de trimitere de mail-uri standardizate. Este folosit pe server-ul de GraphQL pentru a trimite mail-uri la înregistrarea utilizatorilor și la crearea rezervărilor.

Heroku este o platformă de hosting gratuit ce suportă o varietate de servere și baze de date.

3.4 - Detalii de implementare

Deoarece proiectul este împărțit în 3 aplicații diferite, am luat decizia să le descriu pe fiecare independent, de la cea mai simplă, la cea mai complexă. Prima dintre aceste aplicații este platforma de înregistrare pentru utilizatorii cu rolul de *Owner*.

3.4.1 - Modulul de înregistrare al administratorilor

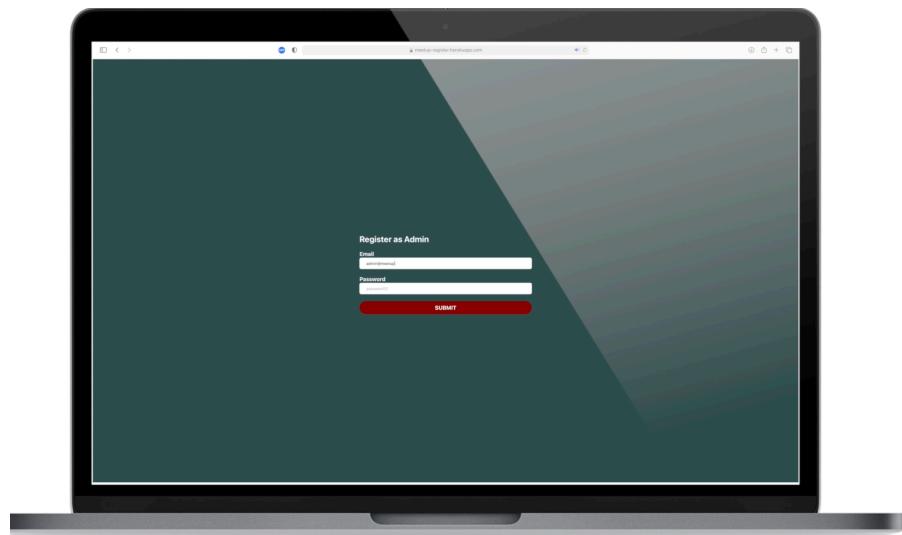


Figura 3.4.1 - Server-ul de înregistrare pentru utilizatorii cu rolul de *Owner*

Acest modul este cel mai simplu din toată aplicația, deoarece este un simplu server web pentru client, făcut cu ajutorul librăriei React și al Apollo Client pentru a consuma API-ul GraphQL. Particularitățile de care dispune acest modul sunt legate de securitatea implementată pentru a nu putea oricine să își creeze un cont cu acest rol.

```
if (secret && status === user_status.admin) {
    const secretStatus = verify(secret, APP_SECRET)
    if (!secretStatus) {
        handleError(errors.invalidKey)
    }
}
```

Figura 3.4.2 - Verificarea cheii primite de la client

Securitatea implementată este bazată pe criptarea datelor și verificarea acestora pe partea de server prin intermediul utilizării unui JSON Web Token. Astfel, numai anumite persoane desemnate prin primirea unui email pot accesa înregistrarea.

3.4.2 - Modulul de autentificare

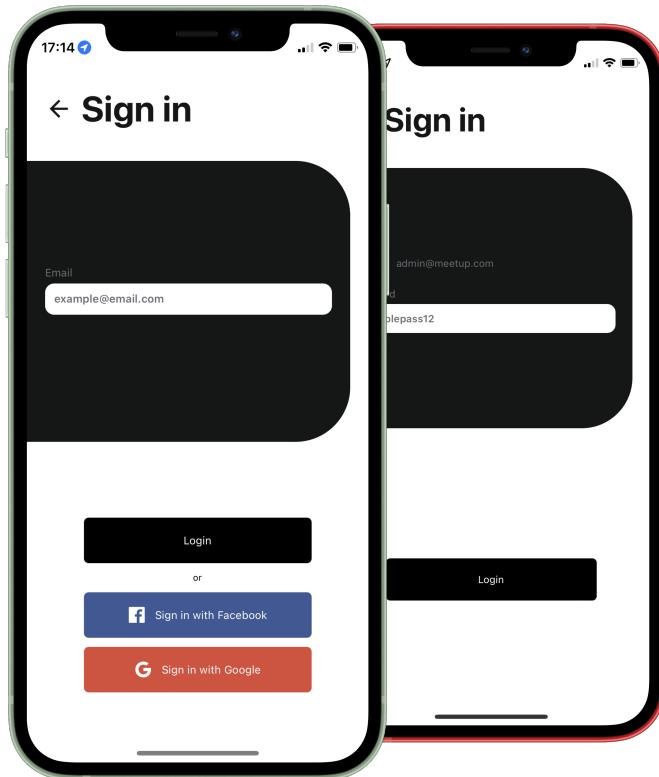


Figura 3.4.3 - Modulul de autentificare pe mobil

Aplicația mobilă este alcătuită din 6 module în care am urmărit să ofer utilizatorilor o experiență cât mai intuitivă și ușor de folosit. Primul dintre aceste module este cel de autentificare. Funcționalitatea pe care o aduce acest modul este reprezentată de faptul că permite utilizatorilor să se autentifice sau înregistreze dintr-un singur ecran, indiferent de rolul pe care îl au. În schița inițială a aplicației erau definite ecrane diferite pentru fiecare tip de utilizator, însă în timpul dezvoltării am realizat că e un mod ineficient de a aborda această problemă pentru că ar presupune că orice utilizator ar putea încerca să se autentifice cu un alt rol. Simplificarea a venit mutând logica necesară pe back-end.

Rezultatul este ilustrat în *Figura 3.4.4*, care este implementat astfel încât după apăsarea butonului de “Login”, în cazul în care autentificarea se face pe bază de input, să se facă un query la API-ul de GraphQL care verifică dacă email-ul introdus aparține unui cont. În caz pozitiv este preluat rolul utilizatorului existent, pe când în caz contrar se va afișa ecranul de înregistrare.

```

export const EXIST_QUERY = gql`query($email: String!) {
  exists(email: $email) {
    exist
    user {
      id
      email
      firstName
      lastName
      photo
    }
    hasPassword
  }
}`;

```

```

if (secret && status === user_status.admin) {
  const secretStatus = verify(secret, APP_SECRET)
  if (!secretStatus) {
    handleError(errors.invalidKey)
  }
}

```

```

const [existQuery, {called, loading, data, error: errorAPI}] =
useLazyQuery(
  EXIST_QUERY,
  {
    fetchPolicy: 'no-cache',
  },
);

```

Figura 3.4.4 - Implementarea verificării de către back-end a email-ului introdus de către utilizator prin sintaxă de GraphQL cu ajutorul Apollo Client

De asemenea, autentificarea cu ajutorul rețelelor de socializare este implementată folosind Facebook SDK și Firebase Google Auth, și după preluarea token-ului de către API-ul de GraphQL se reproduc aceeași pași de verificare a rolului pe care îl are user-ul. Câmpul de email prezintă validare cu ajutorul unei expresii regulate. După autentificare token-ul generat prin JWT se salvează local până la expirarea acestuia. Prin intermediul token-ului utilizatorul are acces la toate metodele pe care le expune back-end-ul.

3.4.3 - Modulul Explore

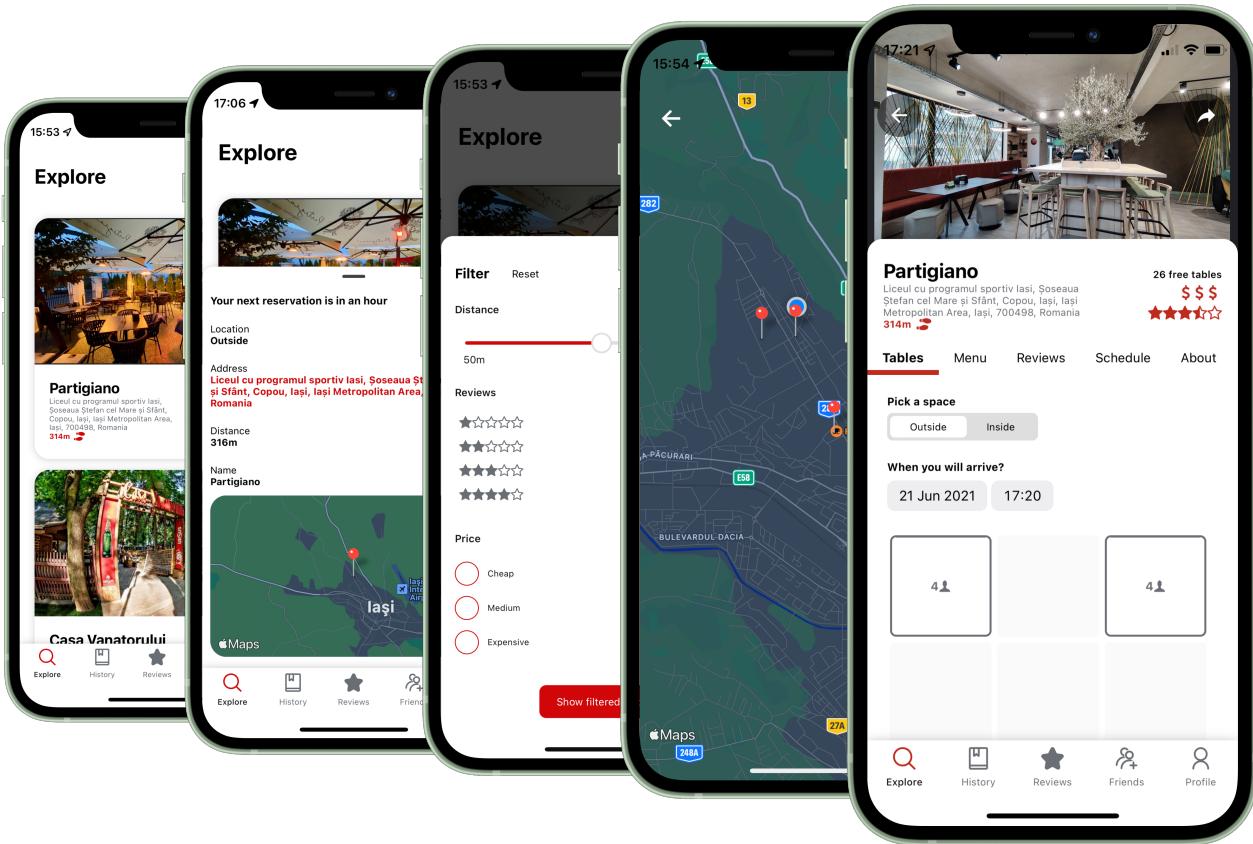
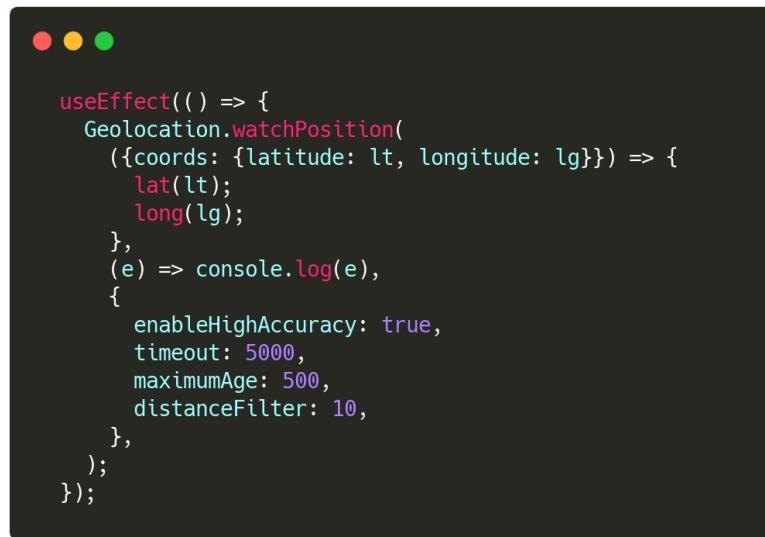


Figura 3.4.5 - Modulul Explore

Cel mai important modul din aplicație este cel de Explore, sau, în cazul în care utilizatorul are rolul Owner, My Pubs. Acest modul cuprinde majoritatea funcționalității pe care o cuprinde aplicația dezvoltată.

Din perspectiva unui client, acest modul afișează o listă de localuri aflate în apropiere ce au sunt deschise la ora curentă. Calculul proximității este făcut pe back-end, prin furnizarea de coordonate de tip latitudine-longitudine de către client. Astfel, back-end-ul returnează către client o listă filtrată, minimizând timpul de răspuns. În Figura 3.4.6 se află codul ce face rost din GPS-ul integrat pe device și îl salvează în mod global cu scopul de a fi trimis la server.



```
useEffect(() => {
  Geolocation.watchPosition(
    {coords: {latitude: lt, longitude: lg}}) => {
      lat(lt);
      long(lg);
    },
    (e) => console.log(e),
    {
      enableHighAccuracy: true,
      timeout: 5000,
      maximumAge: 500,
      distanceFilter: 10,
    },
  );
});
```

Figura 3.4.6 - Preluarea coordonatelor din GPS-ul device-ului

Acste rezultate pot fi filtrate după distanța la care sunt afișate - valoarea inițială fiind de maximum 5000 m, customizabilă print-un slider de la valoarea de 50 m până la 10000 m; după media recenziilor și după prețul stabilit prin compararea mediei de prețuri afișate în meniu. De asemenea, în această pagină în cazul în care un client are o rezervare apare un modal în care sunt afișate detaliile legate de aceasta.

Dispunem și de o mapă ce afișează punctele ce reprezintă locația precisă a localului. Această hartă este implementată folosind sistemul de hărți implicit de pe fiecare tip de device - Android cu Google Maps, iOS cu Apple Maps. Prin apăsarea unui punct de pe hartă clientului i se afișează un mic modal cu o galerie de fotografii ale localului și informații importante precum adresa, media recenziilor, numărul de mese libere la momentul actual și distanța în metri.

Prin apăsarea pe un astfel de card descriptiv pentru local, utilizatorul este trimis către o pagină în care îi sunt afișate informații despre local în ordinea - Tab-ul de mese, de meniu, de recenzii, de program și ultimul fiind tab-ul despre. De asemenea, i se oferă opțiunea de a

vizualiza pozele în detaliu printr-un ecran de tip galerie și să distribuie locația prin orice mijloc de mesagerie.

Prima secțiune, cea de mese, prezintă un selector de locație, definită de administratorul localului și un selector de timp limitat de constrângerile de program ale localului și de data curentă. După ce clientul selectează ora și zona, poate selecta dintr-o matrice de mese una la care dorește să plaseze rezervarea. Fiecare zonă poate avea definit un grid specific potrivit poziționării reale a locației. Fiecare masă are 4 atribute, din care clientului îi sunt vizibile doar două - numărul de locuri disponibil la masă și statusul acesteia după cum urmează:

- ■ masă disponibilă în intervalul selectat
- ■■ masa urmează să fie ocupată de o rezervare în curând
- ■■■ masă indisponibilă

După selectarea mesei, clientul are opțiunea de a invita prietenii pe care îi are în lista de prieteni. Dacă adaugă prieteni acestora le va apărea o notificare cu conținutul: “*Prietenul tău X a făcut o rezervare la localul X la ora X*”, însă acest pas nu este unul obligatoriu. După plasarea rezervării flow-ul de rezervare al clientului este complet iar utilizatorul cât și prietenii invitați nu mai pot plasa o rezervare până la finalizarea celei actuale.

Clientul poate vizualiza și celelalte detalii furnizate de administratorul localului în pagina de meniu, unde sunt afișate pe secțiuni produsele cu ingrediente și prețul în valuta definită. În ecranul de recenzii este afișat feedback-ul oferit de persoane care au plasat rezervări în trecut cu numărul de stele oferit. În ecranul de program este afișat pe zile programul localului, iar în ultimul ecran sunt afișate restul câmpurilor din tabela Pub.

Din punctul de vedere al administratorului localului interfața este una similară, cu abilități de adăugare, modificare și ștergere a datelor. În *Figura 3.4.7* sunt prezentate diferențele dintre interfața de client și cea de client.

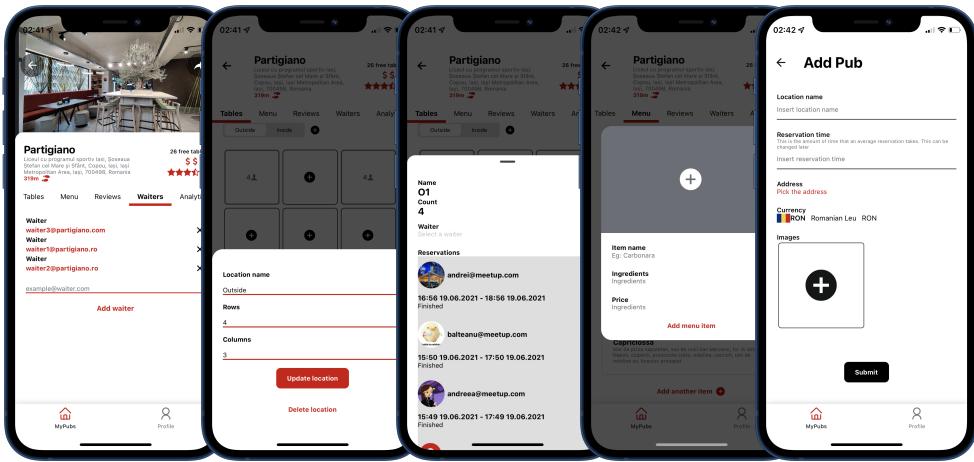


Figura 3.4.7 - Interfața de admin

Pe lângă toate elementele de care dispune utilizatorul de tip client, administratorului îi se adaugă mai multe opțiuni. Prima dintre aceste opțiuni este apariția a două ecrane noi - cel de chelneri și cel de adăugare a unui local.

Ecranul de chelneri este folosit pentru a defini personalul localului care vor fi repartizați la mese. La adăugarea unui chelner se crează un cont nou cu email-ul introdus ce va avea rolul de chelner și o interfață simplă alcătuită doar din ecranul "Tables". În cazul în care email-ul introdus nu există deja, se crează un cont nou fără parola, căruia, dacă este folosit la autentificare în termen de 30 de minute, îi se poate seta o parola de către personalul restaurantului. În cazul în care există un cont deja cu acest email se va afișa o eroare deoarece constrângerea este că un chelner face parte dintr-un singur local (relație one-to-one).

În adiție, administratorul mai dispune de un ecran nou în care îi se permite să adauge localuri. În acest ecran utilizatorul trebuie să introducă denumirea afacerii, apoi un număr ce reprezintă o valoare orientativă drept o medie a duratei unei rezervări. Dupa introducerea acestor valori, se selectează o locație pe hartă care este convertită prin LocationIQ API într-o adresă fizică care poate fi modificată în cazul în care nu este specifică. Se poate selecta și o valută diferită dacă cea primită de la API nu este cea folosită în locația respectivă. Este

obiigatorie adăugarea a minim o poză cu localul, însă pot fi introduse mai multe. Aceste poze sunt stocate folosind Firebase Storage.

Administratorul mai dispune de editarea, adăugarea și ștergerea zonelor definite, a meselor, a pozelor, a meniului și a personalului. De asemenea se pot modifica date ale localului precum vizibilitatea (dacă localul este afișat către clienți), valuta și adresa.

3.4.4 - Modulul History și Modulul Reviews

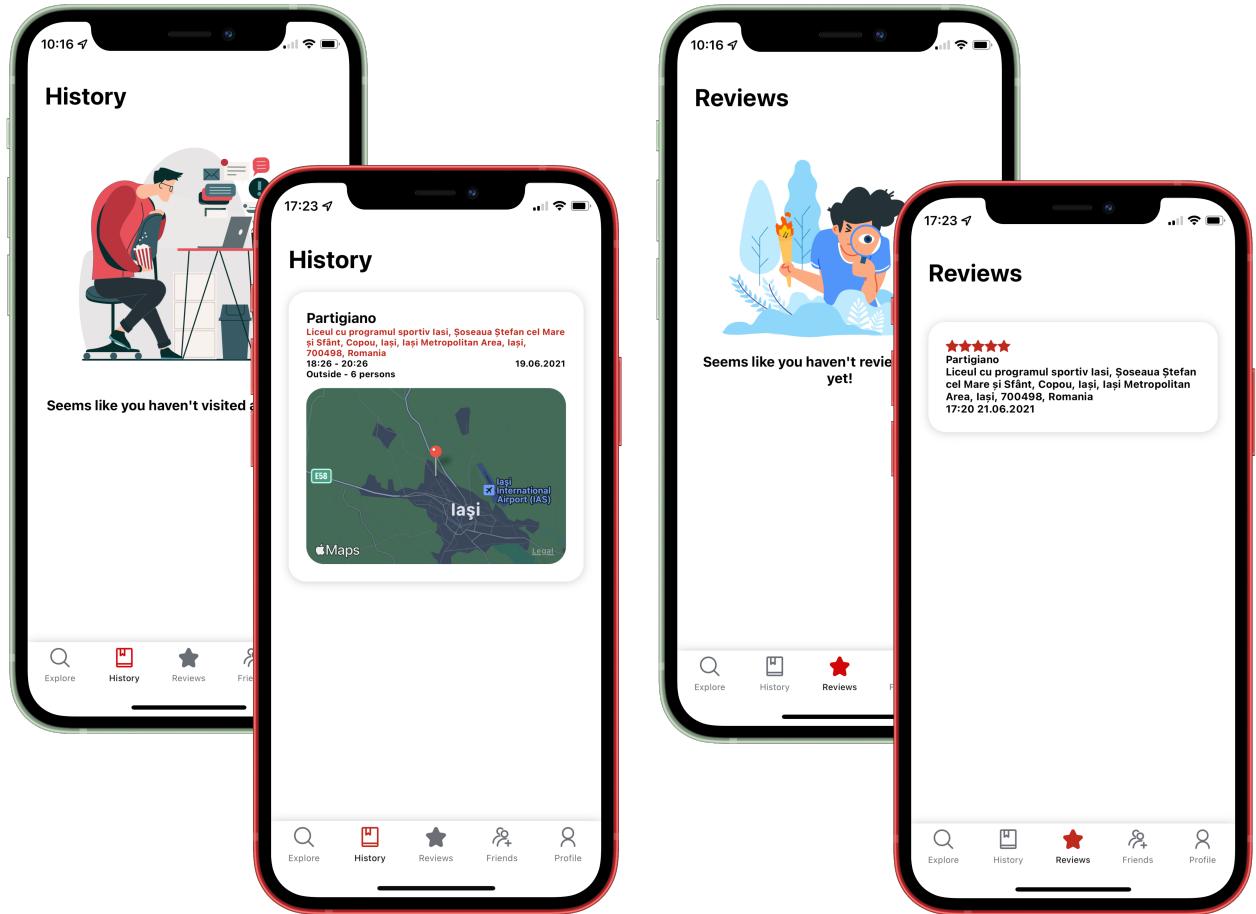


Figura 3.4.8 - Modulele History și Reviews

În aceste ecrane îi sunt afișate utilizatorului recenziile pe care le-a lăsat și istoricul rezervărilor. Aceste două module au doar rol informativ deoarece nu există nici o interacțiune pe care utilizatorul poate să o efectueze. Datele se preiau din query-ul “me” apelat cu ajutorul token-ului stocat în memoria telefonului. Aceste module nu sunt conectate prin subscriptions de API deoarece datele nu sunt dinamice, apar numai prin finalizarea use case-urilor permise unui utilizator.

3.4.5 - Modulul Friends

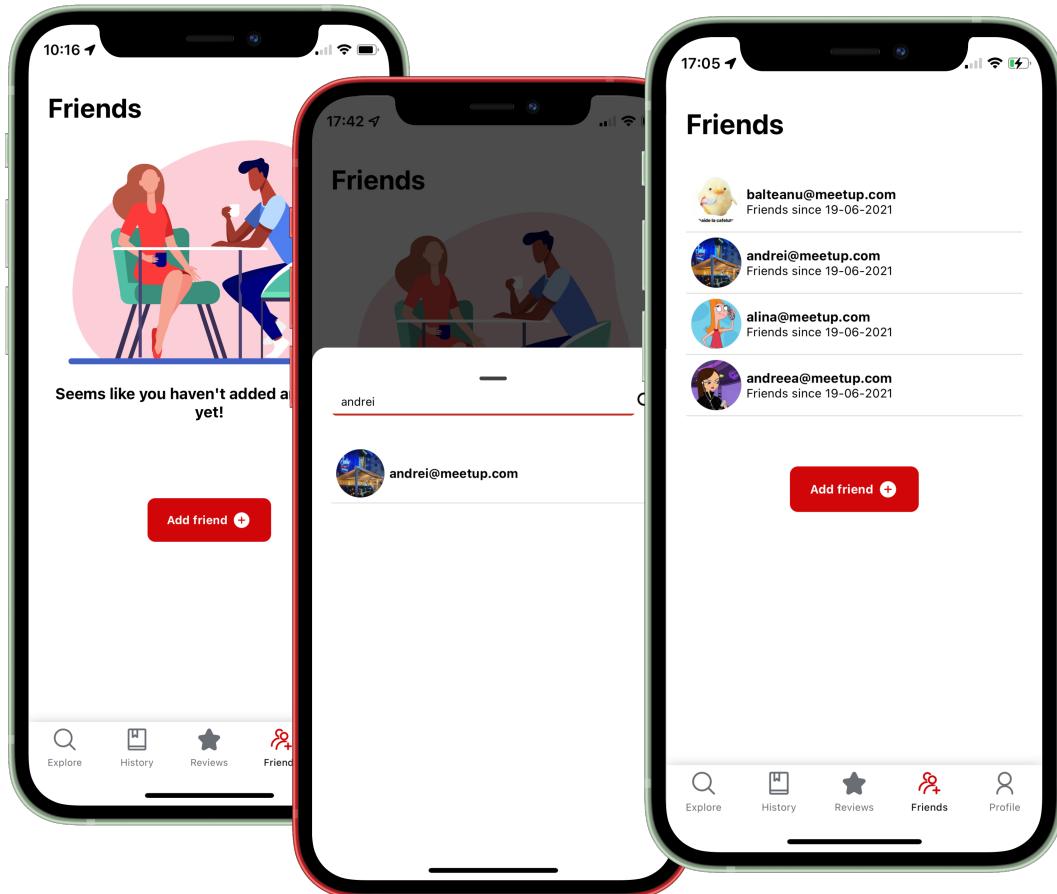


Figura 3.4.9 - Modulul Friends

Această secțiune a aplicației permite unui client adăugarea de prieteni în baza unui search pe câmpul *email*. În cazul în care un utilizator este înregistrat în platformă cu email-ul căutat, este afișat și se oferă opțiunea de a-l adăuga în lista de prieteni. Datele folosite în afișarea listei sunt preluate din request-ul de preluare a datelor utilizatorului.

3.4.6 - Modulul Profile

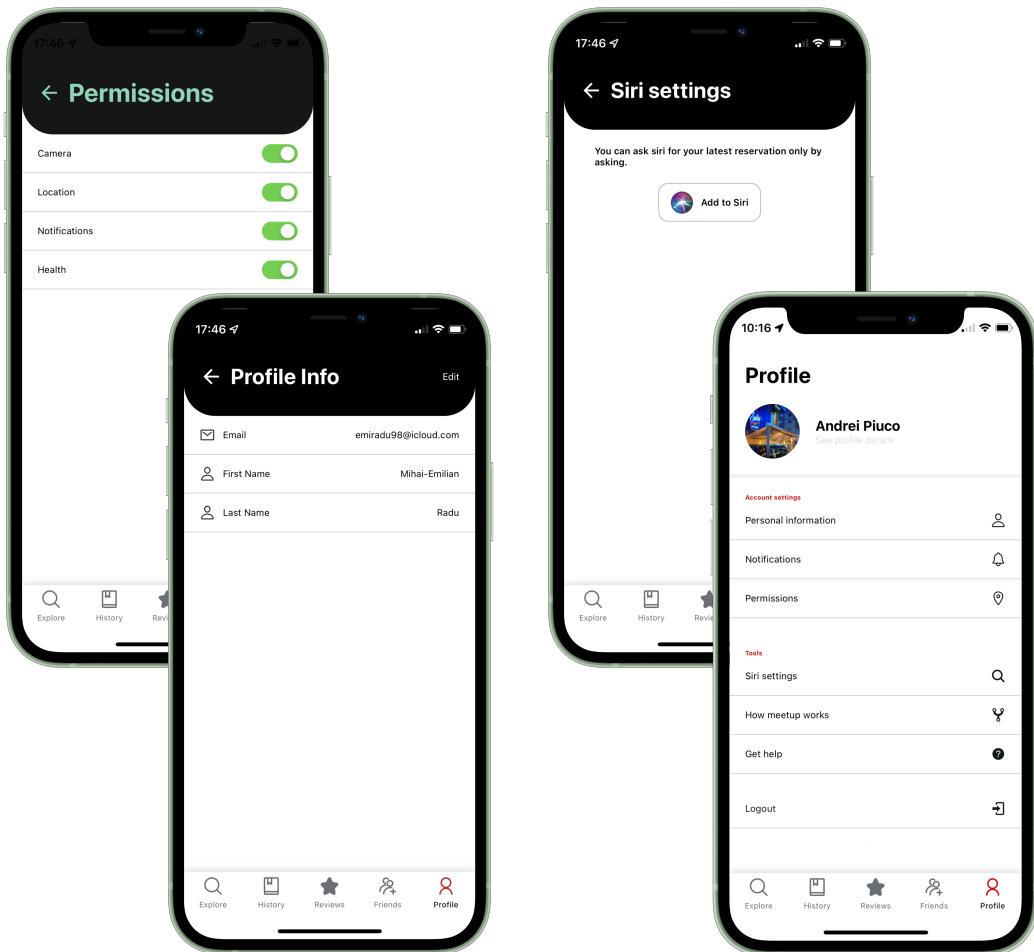


Figura 3.4.10 - Modulul Profile

Acest modul este comun pentru orice tip de utilizator, unde sunt permise modificări de permisiuni, de date personale, și este prezentată opțiunea de a adăuga o un *Siri Shortcut* [10] pe iOS pentru a vizualiza rezervarea activă și opțiunea de a se deloga.

3.4.6 - Back-end

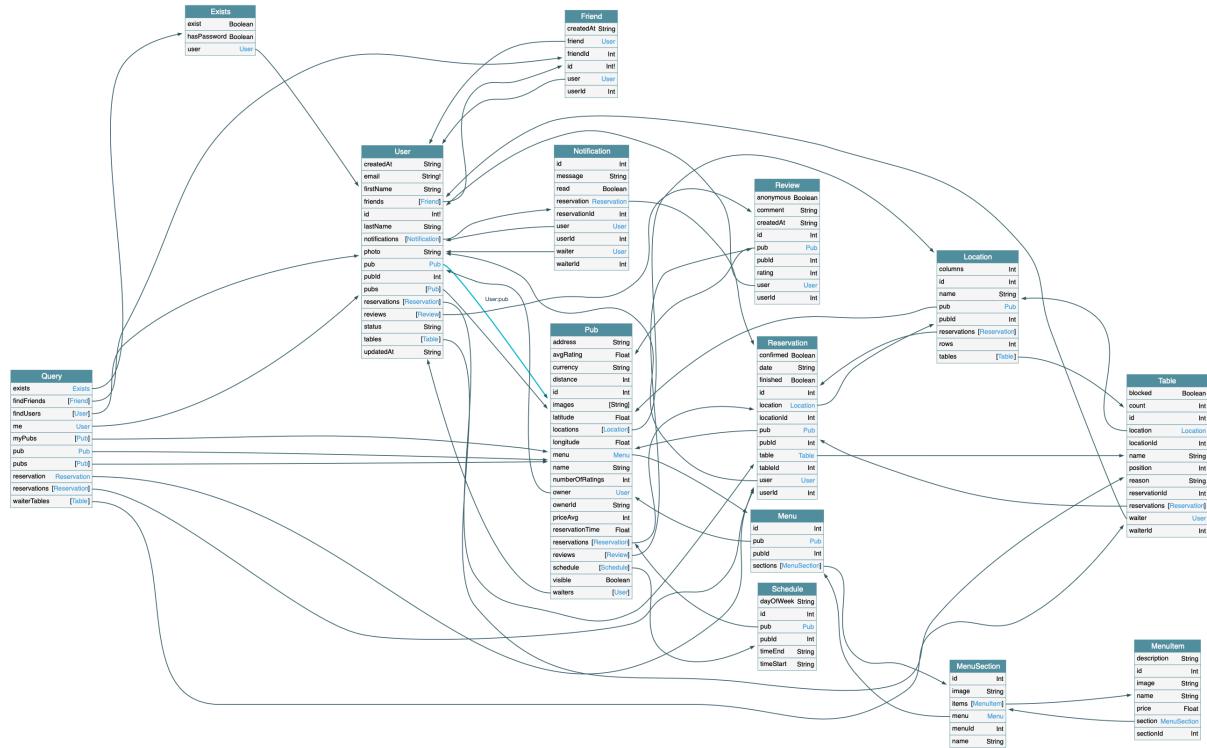


Figura 3.4.11 - Structura bazei de date sub forma de graf

Folosind GraphQL drept API, acesta transformă, după cum sugerează și numele, baza de date într-un graf din care poți accesa orice nod legat printr-o relație fără a fi nevoie de a mai scrie JOIN-uri. *Figura 3.4.11* este generată folosind GraphQL Voyager, o librărie ce ilustrează și validează structura definită în fișierul *schema.graphql*.

Cu toate că schema generată arată încurcată, prin selectarea unui nod în interfață Voyager sunt afișate toate câmpurile ce sunt în relație cu selecția făcută. Datorită acestei structuri, numărul de query-uri de pe client este unul minim, iar dacă apar pe viitor modificări în datele afișate de către client, nu este necesară intervenția developerilor de back-end, deoarece body-ul query-ului făcut de client este identic cu răspunsul primit de la server.

Pentru a veni în ajutorul front-end developerilor, GraphQL dispune de echivalentul documentației de API Swagger, denumit Playground. În schimb, comparativ cu Swagger, acesta nu necesită adnotări și definirea endpoint-urilor, ceea ce economisește timp și

minimizează eroarea umană în scrierea documentației. Acest Playground prezentat în *Figura 3.4.12*, afișează toate operațiunile definite de către developer, cu tipurile de date.

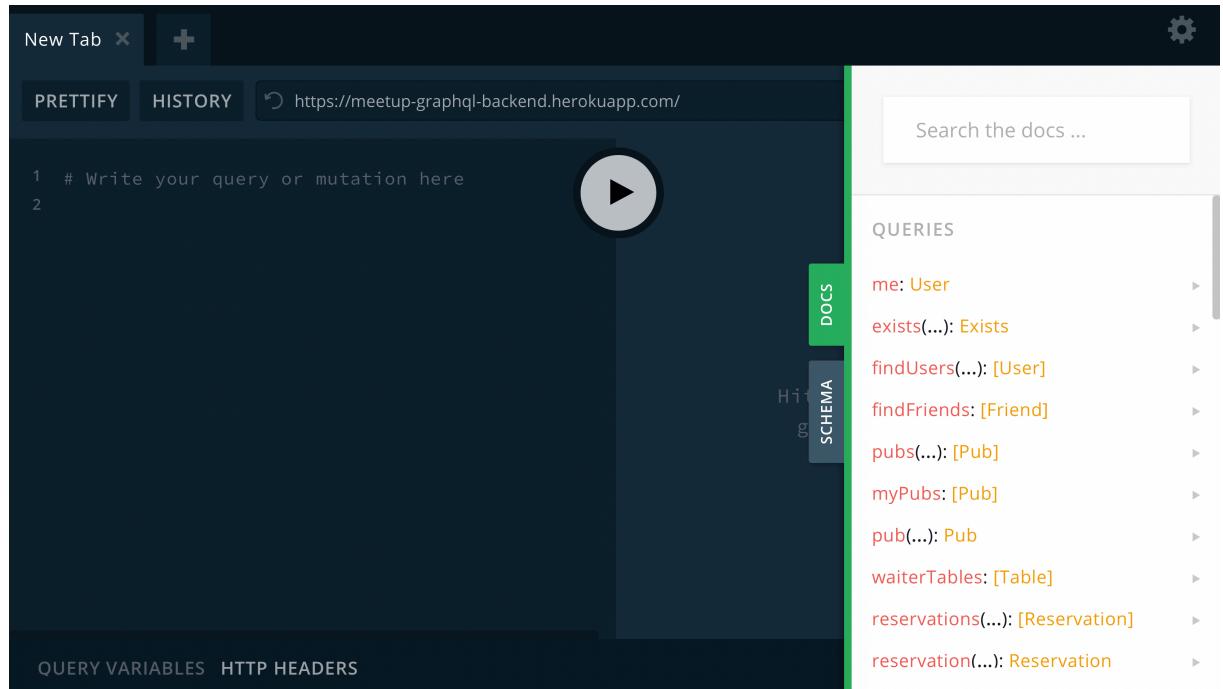


Figura 3.4.12 - Playground GraphQL

Operațiunile definite în dezvoltarea proiectului *MeetUP* sunt:

- Queries pentru obținerea datelor - echivalentul GET
- Mutations pentru manipularea datelor - echivalentul PATCH, PUT, POST, DELETE
- Subscriptions pentru updatarea datelor - echivalentul WebSockets

Această din urmă este operația ce se asigură ca atât clientul cât și personalul localului au datele actualizate cu privire la numărul de mese libere și rezervări.

Pentru a vizualiza sintaxa GraphQL, atașez *Figurile X și X* în care sunt prezentate linii de cod ce deschid server-ul, generează baza de date și crează o operațiune de tip mutație - adăugarea unui prieten.

```

export const server = new ApolloServer({
  schema: applyMiddleware(schema, permissions),
  context: createContext,
  playground: true,
  tracing: isDev(),
  introspection: true,
  debug: isDev(),
  cors: true,
  subscriptions: {
    onConnect: (_connectionParams, _websocket, connContext): SocketContext
  => {
    return {
      req: connContext.request,
      prisma,
      pubsub,
    }
  },
})

```

Figura 3.4.12 - Instanțierea serverului Apollo

```

export const friends = extendType({
  type: 'Mutation',
  definition(t) {
    t.field('addFriend', {
      type: 'Friend',
      args: {
        userId: nonNull(intArg()),
        friendId: nonNull(intArg())
      },
      async resolve(_parent, { friendId, userId }, ctx) {
        try {
          return ctx.prisma.friend.create({
            data: {
              friendId: friendId,
              userId
            },
            include: {
              friend: true
            }
          })
        } catch (e) {
          handleError(errors.userAlreadyExists)
        }
      }
    })
  }
})

```

```

model User {
  id          Int          @id @default(autoincrement())
  email       String       @unique
  password    String?
  lastName    String?
  firstName   String?
  photo       String?
  pubs        Pub[]
  reviews    Review[]
  status     String?      @default("CLIENT")
  pub        Pub?         @relation(name: "Walter", fields: [pubId], references: [id])
  pubId      Int?
  createdAt   DateTime    @default(now())
  updatedAt   DateTime    @updatedAt
  reservations Reservation[] @relation("UserReservation")
  tables     Table[]     @relation("TableWaiter")
  notifications Notification[] @relation("NotifiedUser")
  waiterNotifications Notification[] @relation("WaiterNotif")
  friends    Friend[]    @relation("Friend")
}

model Pub {
  id          Int          @id @default(autoincrement())
  ownerId     Int          @id @default(autoincrement())
  owner       User         @relation(name: "Owner", fields: [ownerId], references: [id])
  reservations Reservation[] @relation("PubReservation")
  schedule    Schedule[]
  address     String       @unique
  latitude    Float
  longitude   Float
  description String?
  visible    Boolean      @default(true)
  avgRating  Float        @default(0)
  numberofRatings Int        @default(0)
  reservationTime Float    @default(2.0)
  priceAvg   Int          @default(0)
  images     String[]
  name       String       @unique
  reviews    Review[]
  locations  Location[]
  createdAt  DateTime    @default(now())
  updatedAt  DateTime    @updatedAt
  currency   String?
  menu       Menu?
  waiters    Waiter[]    @relation("Waiter")
}

```

Figura 3.4.13 - Mutation addFriend și definirea structurii bazei de date folosind Prisma

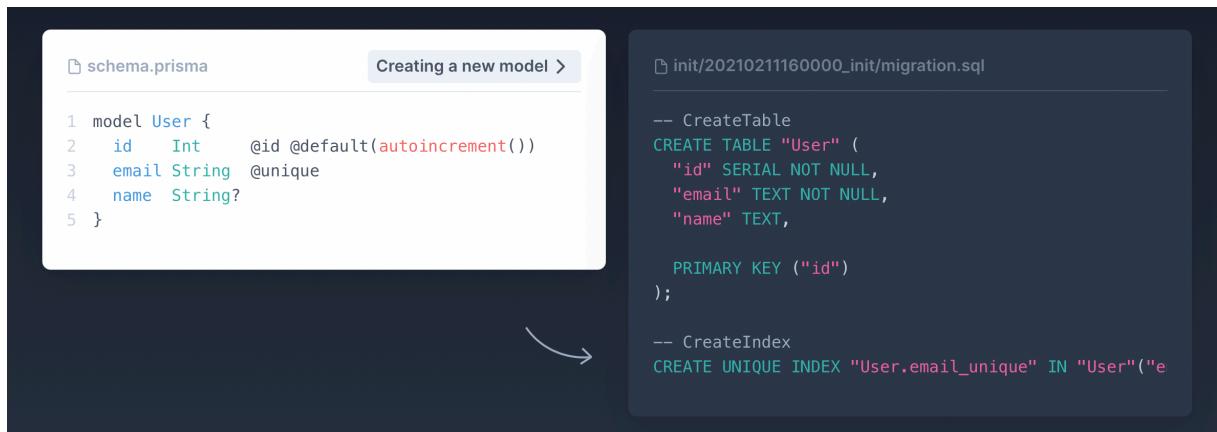
Figura 3.4.14 prezintă structura fișierului schema.prisma care realizează conexiunea între Apollo Server și baza de date Postgres. Aceasta conexiune se stabilește prin codul din Figura 3.4.15.



```
datasource db {
    provider = "postgresql"
    url      = env( "DATABASE_URL" )
}
```

Figura 3.4.15 - Stabilirea conexiunii între Prisma și Postgres folosind variabile din .env

La fiecare modificare a acestui fișier, se crează o migrare a bazei de date care este reversibilă. Această migrare se traduce în limbaj SQL conform Figurii 3.4.16.



The screenshot shows the Prisma IDE interface. On the left, there is a code editor for a file named 'schema.prisma' containing the following Prisma schema:

```
1 model User {
2     id    Int      @id @default(autoincrement())
3     email String  @unique
4     name  String?
5 }
```

A curved arrow points from the code editor towards a second panel on the right, which displays the generated SQL migration script:

```
-- CreateTable
CREATE TABLE "User" (
    "id" SERIAL NOT NULL,
    "email" TEXT NOT NULL,
    "name" TEXT,
    PRIMARY KEY ("id")
);

-- CreateIndex
CREATE UNIQUE INDEX "User.email_unique" IN "User"("e
```

Figura 3.4.16 - Transformarea schemei prisma în script de generare al bazei de date prin intermediul migrării [9]

O altă responsabilitate pe care le are back-end-ul este cea de a trimite notificări și email-uri și de a se integra cu alte servicii precum LocationIQ pentru a oferi clientului estimări și adrese corecte. Aceasta integrare cu servicii third-party a fost făcută conform *Figurii X*.

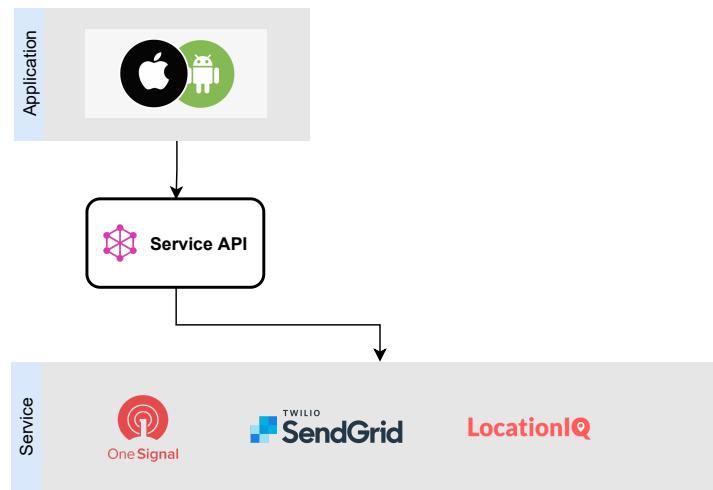


Figura 3.4.17 - Integrarea cu servicii third party de către back-end

3.4.7 - Servicii adiționale

Pe lângă stack-ul tehnologic menționat, intenția mea a fost să creez un produs cât mai apropiat de unul “gata de producție” prin implementarea unor scripturi ce automatizează procesul de distribuție, deoarece dezvoltarea unei aplicații de mobil este mai diferită și complicată decât a unui site web. De asemenea, am dorit și ca baza de date și API-ul să fie accesibile oricând, nefiind necesară pornirea unui server local de fiecare dată când se dorește utilizarea platformei *MeetUP*. Acst aspect de integrare cu servicii adiționale de distribuție automată este prezentat în *Figura 3.4.18*.

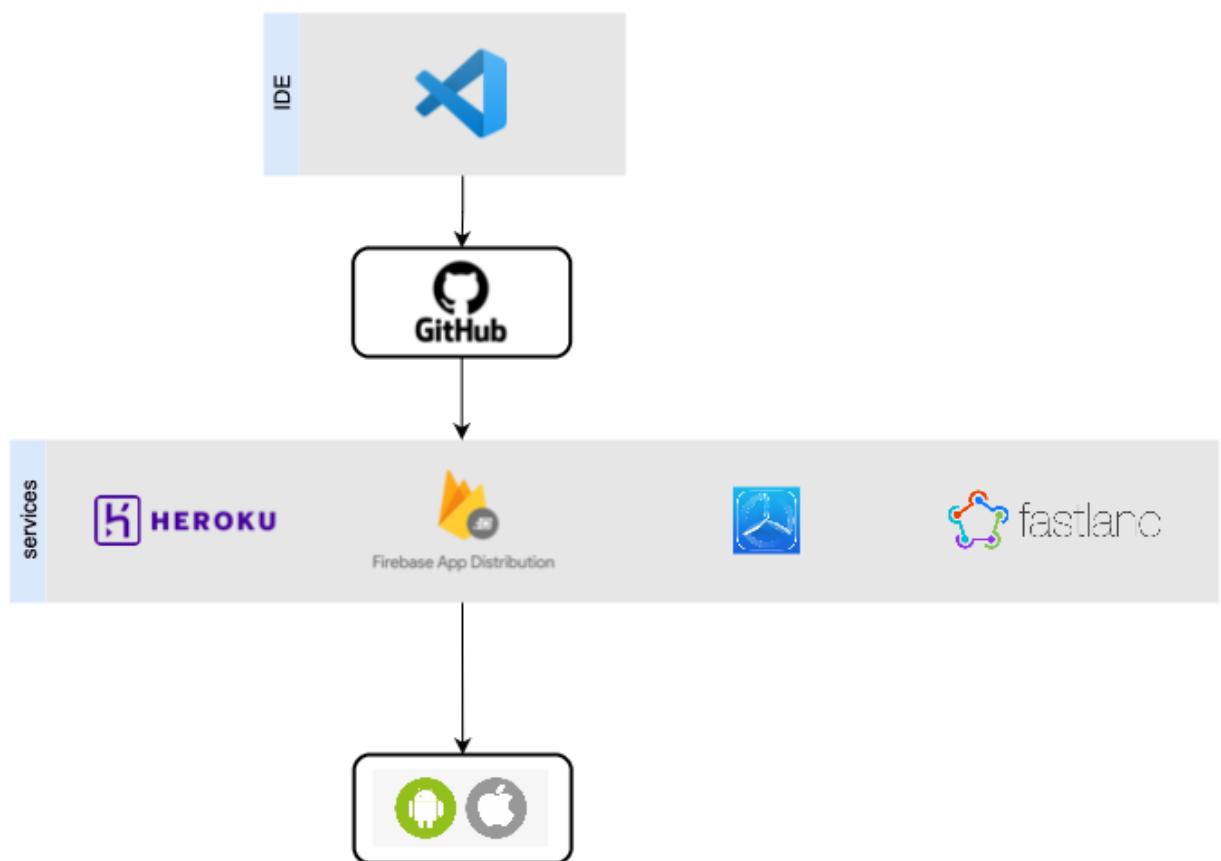


Figura 3.4.18 - Integrarea cu servicii de distribuție automate

Concluzii

Dezvoltarea platformei *MeetUP* valorifică o paletă largă de cunoștiințe acaparate pe perioada studiilor universitare. Prin cultivarea pasiunilor pe care le-am descoperit în cadrul facultății, aplicația dezvoltată cuprinde subiecte precum tehnologii web, baze de date, securitatea informației, structuri de date, rețele de calculatoare, programare orientată obiect, programare avansată, cloud computing, ingineria programării și algoritmica grafurilor. Cu toate că unele materii studiate își au simțită prezența mai mult decât altele, toate aceste cunoștiințe au dus la crearea unui produs viabil, de actualitate, o soluție simplă și intuitivă cu o arhitectură complexă și cu integrarea unor tehnologii moderne.

Toate aceste aspecte luate în considerare, pot afirma faptul că dezvoltarea acestei aplicații a reprezentat o provocare pe care am reușit să o depășesc și prin care am acaparat o înțelegere mai uniformă a tuturor pașilor dezvoltării unui produs.

MeetUP - aplicație pentru rezervarea locurilor din localuri, este produsul ce propune aducerea în era digitală a unui proces de zi cu zi, ce vine cu avantaje semnificative comparativ cu soluțiile existente, venind în ajutor atât clienților, cât și personalului și administratorului unei locații.

Direcții de viitor

Cu toate că lucrarea este una completă și pregătită pentru lansare, planurile pentru versiunea următoare sunt numeroase. Deja am observat modalități prin care pot îmbunătăți experiența utilizatorilor și un set de funcționalități noi. Printre acestea se numără:

1. Implementarea unui chat pentru prieteni pentru a dezvolta aspectul social al aplicației;
2. Finalizarea părții de statistici pentru administratori pentru a vizualiza cât mai concis și a face o analiză asupra fluxului de clienți;
3. Implementarea unui buton de “Stay more” atât pentru clienți cât și pentru personal, în cazul în care nu mai sunt rezervări în perioada dorită, pentru a nu fi necesară rezervarea de multiple ori a aceleiași mese;
4. Îmbunătățirea experienței utilizatorilor prin stilizarea mai modernă a anumitor module;
5. Implementarea unui sistem de testare automată cu JEST pentru a avea un proces complet automatizat;
6. Implementarea de Tag-uri pentru localuri, pentru a putea filtra între restaurante, baruri și specific culinar;
7. Logarea chelnerilor prin QR coduri printate pe ecuson;
8. Implementarea unei rezervări custom în cazul în care se dorește rezervarea unei mese mai mari decât cele de care dispun localul;

Bibliografie

- [1] **Human Interface Guidelines**, Apple Inc. - <https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/>
- [2] **OpenTable** - <https://en.wikipedia.org/wiki/OpenTable>
- [3] **Eat App** - https://en.wikipedia.org/wiki/Eat_App
- [4] "NoSQL Performance Benchmark 2018 – MongoDB, PostgreSQL, OrientDB, Neo4j and ArangoDB", Februarie 2018 - <https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/>
- [5] "Build production grade API with Prisma and GraphQL" - <https://blog.geographer.fr/prisma-graphql-api>
- [6] "Should I Use Apollo for GraphQL?" - autor Rajat S, februarie 2019 - <https://blog.bitsrc.io/should-i-use-apollo-for-graphql-936129de72fe>
- [7] **Documentația oficială GraphQL** - <https://graphql.org>
- [8] **Documentația oficială ReactJS** - <https://reactjs.org>
- [9] "How Prisma and GraphQL fit together" - <https://www.prisma.io/graphql>
- [10] "Run shortcuts with Siri, the Shortcuts app, or Siri Suggestions", Apple Inc. - <https://support.apple.com/en-us/HT209055>
- [11] **Documentația oficială React native** - <https://reactnative.dev>
- [12] "What is Prisma?" - <https://www.prisma.io/docs/concepts/overview/what-is-prisma>
- [13] **Documentația oficială Prisma** - <https://www.prisma.io>
- [14] **Documentația oficială OneSignal** - <https://documentation.onesignal.com/docs>
- [15] **Documentația oficială BranchIO** - <https://help.branch.io/developers-hub/docs/react-native>
- [16] **Documentația oficială Google Cloud** - <https://cloud.google.com/docs>
- [17] **Documentația oficială a librăriei React Native Firebase** - <https://rnfirebase.io>
- [18] **Documentația oficială a tool-ului Fastlane** - <https://docs.fastlane.tools>
- [19] **Documentația oficială Github Actions** - <https://docs.github.com/en/actions>
- [20] "Getting started with the SendGrid API" - <https://docs.sendgrid.com/for-developers/sending-email/api-getting-started>

- [21] "How Heroku Works", Heroku Architecture - <https://devcenter.heroku.com/articles/how-heroku-works>
- [22] "Hassle-free Database Migrations" - <https://www.prisma.io/migrate>
- [23] "The Ultimate Guide to Mobile API Security", autor Randall Degges, martie 2015 - <https://stormpath.com/blog/the-ultimate-guide-to-mobile-api-security>
- [24] "Push Notifications and In-App Messaging with OneSignal and Expo", autor George Deglin, mai 2020 - <https://blog.expo.io/push-notifications-and-in-app-messaging-with-onesignal-and-expo-3abf77b7f288>
- [25] "How to build a GraphQL API" - autor Matt Mcnamee, martie 2019 - <https://mcnamee.medium.com/how-to-build-a-graphql-api-73786f7b7c80>