

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Booking App cu arhitectură serverless

propusă de

Leleu Alexandru-Ioan

Sesiunea: *iulie, 2019*

Coordonator științific

Lect. Dr. Anca Ignat

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Booking App cu arhitectură serverless

Leleu Alexandru-Ioan

Sesiunea: *iulie, 2019*

Coordonator științific
Lect. Dr. Anca Ignat

Avizat,
Îndrumător Lucrare de Licență
Titlul, Numele și prenumele

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul(a)

domiciliul în

născut(ă) la data de, identificat prin CNP,
absolvent(a) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de
..... specializarea, promoția
....., 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:

.....
.....
.....elaborată sub îndrumarea dl. / d-na
....., pe care urmează să o susțină 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, consimțind 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 falsificării 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 azi,

Semnătură student

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Booking App cu arhitectură serverless*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc 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, data

Absolvent, Leleu Alexandru-Ioan

(semnătura în original)

Cuprins

1. Motivație	2
2. Introducere.....	4
2.1. Obiectiv.....	4
2.2. Structura lucrării.....	5
3. Contribuții.....	6
4. Tehnologiile utilizate	7
4.1. React-Native	7
4.2. Firebase	10
4.3. Concluzii.....	17
5. Prezentarea aplicației.....	18
5.1. Experiența utilizatorului.....	18
5.2. Despre „BarbersApp”.....	19
5.3. Descrierea aplicației și funcționalități	19
6. Baze de date nerelaționale actualizate în timp real.....	26
7. Concluziile finale	31
8. Bibliografie.....	32

1. Motivație

Alegerea acestei teme pentru lucrarea de licență a pornit de la o simplă întrebare, primită din partea coordonatorului meu științific, Conf. Dr. Anca Ignat, citez „Poți să faci ceea ce îți place ție să faci?”.

Așadar am hotărât să hrănesc pasiunea pentru mobile development prin crearea unei aplicații ce vine în ajutorul bărbierilor de pretutindeni de a se conecta cu clienții lor. Am optat pentru folosirea de tehnologii actuale, framework-uri noi atât pentru a-mi ușura munca și pentru a se plia pe cerințele mele dar și cu scopul de a dobândi noi cunoștințe și a descoperi noi posibile căi pentru crearea unei aplicații de succes.

Cum a apărut ideea acestei aplicații?

Frizerul căruia îi sunt fidel de ceva timp este destul de aglomerat și pentru a putea avea parte de serviciile lui trebuie să îmi rezerv un loc cu ceva timp înainte. Acest proces poate dura destul de mult având în vedere faptul că nu poți vedea locurile disponibile și poți repeta întrebarea "Alt interval orar mai este disponibil?" de nenumărate ori. Discutând cu el și analizând părțile ce le-ar putea rezolva o aplicație am început să pun pe picioare "BarbersApp".

"BarbersApp" este o platforma ce se focusează pe ușurarea interacțiunii dintre frizeri și clienții acestora. Utilizatorii acestei aplicații pot intra în contact mult mai ușor cu frizerii preferați și își pot rezerva un interval orar mult mai rapid.

Lucrând la acest proiect am observat că piața încă nu este destul de exploatată, în special în țara noastră și cred cu tărie că poate aduce un plus acesteia și chiar ar fi apreciată de către utilizatori.

"BarberApp" oferă servicii pentru rezervări online în maniera real-time, asigurând unicitatea pentru fiecare interval orar, un sistem prin care fiecare utilizator își poate căuta mai ușor frizerii din apropiere, posibilitatea de a personaliza propriul cont și de a oferi o gamă proprie de servicii dar și un sistem de notificări pentru a reaminti clienților data programării.

De ce am optat pentru realizarea unei aplicații mobile?

În anul 2018, 58% dintre accesările unui website au venit de pe un dispozitiv mobile, acest procentaj fiind în continuă creștere. Dispozitivele mobile dețin 42 de procente din timpul total petrecut online. Este destul de cunoscut faptul că multe dintre companiile IT își îndreaptă atenția către dezvoltarea aplicațiilor mobile din mai multe motive:

1. Aplicațiile pot fi mult mai interactive deoarece pot accesa diversele servicii și caracteristici ale dispozitivului (camera, gps și locații, senzori și multe altele).

2. Au abilitatea de a funcționa offline cu avantajul că utilizatorii pot accesa informații oricând și oriunde.

3. Interfețele oferite de către o aplicație mobile sunt mult mai intuitive și mult mai ușoare de utilizat în completarea diverselor sarcini deoarece dispun de ecrane tactile, userii putând naviga rapid prin diferite comenzi.

Prin urmare mi-am îndreptat atenția către dezvoltarea mobile pentru a dobândi noi cunoștințe specifice acestui domeniu destul de amplu și pentru a folosi anumite tehnologii destul de apreciate de către comunitate.

2. Introducere

2.1. Obiectiv

Dezvoltarea unei aplicații de succes nu este deloc ușoară.

Aplicația ta, fie Android, iOS sau Web are nevoie de un server pentru a stoca anumite date și de a satisface cererile clienților.

Ca programator te confrunți destul de des cu probleme, în special la nivelul serverului, mai ales când programul tău trebuie să suporte cereri concurente de la un număr destul de mare de utilizatori. Din acest moment apar diverse nevoi privind scalabilitatea, infrastructura și securizarea datelor. Pentru a soluționa toate aceste obstacole am ales să urmez o arhitectura serverless.

De ce am ales o arhitectura serverless?

În primul rând această arhitectură este o paradigmă de programare, un concept destul de nou, preluat din ce în ce mai mult de către companii de succes deoarece vine cu o serie de avantaje și tool-uri interesante.

Cei mai comuni și cunoscuți furnizori ale acestor servicii sunt Amazon (AWS Lambda service) și Google (Firebase). Termenul e destul de confuz deoarece sugerează faptul că aplicațiile noastre nu mai au nevoie de suportul unui server însă esența acestei arhitecturi stă în faptul că ne oferă soluții rapide pentru stocarea datelor noastre, pentru hosting și rutare, pentru configurare și infrastructura pregătită unui număr mare de utilizatori, servicii/funcții ce sunt rulate în cloud de către diverse servere ce nu necesită configurări din partea noastră.

Așadar am hotărât să dezvolt o aplicație mobile ce folosește diverse tool-uri oferite de Firebase precum, autentificare, stocare a datelor într-o manieră real-time și un sistem de notificări.

Am ales să folosesc tehnologii actuale ce se pliază pe necesitățile mele oferind suport atât pentru partea de front-end cât și cea de back-end. Aplicația este construită cu ajutorul framework-ului React-native și este suportată în momentul de față doar de către device-urile Android. Stocarea datelor se face într-o manieră real-time, asigurând integritatea datelor. Servicii precum Google Maps și FCM notifications au fost folosite pentru a facilita și a oferi o experiență plăcută utilizatorilor.

Scopul acestei lucrări a fost de a dobândi noi cunoștințe în domeniul dezvoltării aplicațiilor mobile, ce avantaje și dezavantaje implică o arhitectură serverless, utilizarea unei baze de date nerelațională și gestionarea datelor într-o maniera real-time.

2.2. Structura lucrării

Lucrarea cuprinde 3 capitole în care sunt prezentate etapele creării acestei aplicații mobile, detalii cu privire la tehnologiile și noțiunile folosite, respectiv modul în care putem utiliza corect serviciile oferite de platforme precum Firebase, Google sau Facebook.

- 1. Tehnologiile utilizate** - acest capitol cuprinde detalii despre tehnologiile folosite în obținerea unei aplicații mobile cu o arhitectură serverless. Vom discuta atât despre uneltele folosite pentru realizarea interfeței, a structurii unei aplicații native dar și despre serviciile folosite pentru stocarea informațiilor în maniera real-time. Limbajul folosit este Javascript atât pe partea de front-end cât și pe back-end. Pentru dezvoltarea aplicației am folosit React-Native, pentru realizarea interfeței am folosit diverse librării și componente dezvoltate de comunitate iar datele sunt salvate într-o baza de date nerelațională (NoSQL Firebase Realtime Database). Printre serviciile folosite se numără și Google Maps, Here Api pentru interacțiunea cu hărțile disponibile dar și informații cu privire la locația clienților. Aplicația dispune și de un sistem de notificări la distanță prin intermediul platformei FCM messages.
- 2. Prezentarea aplicației** - acest capitol cuprinde detalii în special despre partea funcțională, prezentând componentele aplicației, modul de utilizare și pachetul de servicii pus la dispoziție clienților. Vom insista și asupra modului în care putem îmbina diversele componente ce alcătuiesc interfața aplicației pentru obținerea unei experiențe a utilizatorului cât mai plăcută, precum și tehnici de utilizare și soluționare a diverselor instrumente oferite de diverși producători.
- 3. Baze de date nerelationale actualizate în timp real** – cuprinde detalii cu privire la modul de funcționare a unei baze de date NoSQL, ce beneficii sunt aduse la pachet de către platforma Firebase și ce plusuri aduc aplicației noastre. Vom discuta pe larg diferențele față de o baza de date relațională și în ce situații e bine să o folosim.

3. Contribuții

În realizarea acestei lucrări am avut contribuții pe următoarele planuri:

- Pe plan teoretic:
 - Stabilirea și definirea structurii bazei de date, prin crearea obiectelor și relației dintre ele pe baza unor identificatori unici.
 - Studiarea metodelor folosite pentru scrierea și citirea datelor dintr-o bază de date, sincronizată în timp real pentru orice client.
 - Definirea logicii unui serviciu automat pentru sistemul de notificări al aplicației.
 - Stabilirea unui sistem de programări online ce suportă cereri concurente oferind unicitate fiecarui interval orar.
 - Identificarea cerințelor și tehnologiilor necesare implementării unei astfel de aplicații ce oferă posibilitatea de a face programări la un anumit bărbier și de a interacționa cu acesta.
- Pe plan practic:
 - Analiza soluțiilor și alegerea optimă din punct de vedere al implementării și performanței.
 - Implementarea aplicației pe partea de client, stabilirea interfeței, alegerea unui design potrivit temei alese.
 - Integrarea aplicației cu baza de date oferită de platforma Firebase.
 - Integrarea aplicației și folosirea serviciilor de autentificare oferite de Firebase și Facebook.
 - Folosirea serviciilor de localizare și de interacțiune cu o hartă oferite de Google și Here Api.

4. Tehnologiile utilizate

Pentru dezvoltarea acestei lucrări am folosit React-Native, un framework JavaScript open-source destinat construirii unei aplicații native, atât pentru iOS cât și pentru Android. Este bazat pe biblioteca React dezvoltată de Facebook, ce vine în ajutorul randării interfețelor, folosind și împărțind aceleași principii destinate programării web. Această aplicație este 100% funcțională pentru dispozitive Android iar pentru a putea fi folosită și de dispozitive iOS este necesară doar configurarea anumitor componente specifice acestei platforme și ajustarea interfeței în funcție de dimensiunile dispozitivului. Pentru a contura un design potrivit acestei teme (programări online) am folosit diverse componente dezvoltate de comunitate. Am folosit și servicii pentru Autentificare, stocarea datelor și a fișierelor oferite de Firebase iar pentru interacțiunea cu o hartă și gestionarea locației curente am folosit funcționalități oferite de Google Maps și Here API. Aplicația include totodată și un sistem dinamic de notificări (FCM messaging) pentru a reaminti utilizatorilor data și serviciul programat.

4.1. React-Native¹

React-Native este un framework open-source dezvoltat de Facebook, Instagram și comunitatea de programatori, ce ne ajută să creăm aplicații captivante doar cu ajutorul unui singur limbaj de programare: JavaScript, aplicații ce vor fi suportate de ambele platforme (Android/iOS).

Similar React-ului pentru dezvoltarea web, aplicațiile React-Native sunt scrise folosind aceleași mix-uri JavaScript și markup-uri XML, cunoscute sub numele JSX (JavaScript XML). Partea interesantă stă în faptul că React-Native creează o legătură între componentele noastre scrise cu JavaScript și API-urile native scrise în Objective-C (pentru iOS) sau Java (pentru Android) astfel încât aplicația noastră să randeze componente mobile ce vor arăta și funcționa ca la orice altă aplicație nativă. Mai mult decât atât oferă posibilitatea accesării diverselor servicii, precum: cameră, microfon, locație, senzori de mișcare, de proximitate etc.

Care sunt avantajele și de ce să folosim acest framework?

„Învăță o singură dată, scrie peste tot”

¹ <https://facebook.github.io/react-native/>



- Este un mare plus în special pentru dezvoltatorii obișnuiți să lucreze pe Web cu React, deoarece vor putea scrie aplicații mobile cu performanță și aspectul unei aplicații native, folosind principii și instrumente familiare.
- Design-ul și aspectul se simt mult mai productive dacă ești capabil să vizualizezi ceea ce creezi. În această privință React-Native oferă o caracteristică foarte utilă numită „live reload” ce este o previzualizare live a modificărilor precedente de la nivelul codului.
- Performanța crescută, comparativă cu cea a aplicațiilor scrise în Java sau Objective-C.
- O altă caracteristică excelentă a acestui framework sunt „Componentele gata de aplicat” (Ready-made components). În loc să scrii cod de la zero poți refolosi, poți aplica componente gata făcute. Ele sunt excelente pentru a obține forme simple de funcționalități și fac procesul de dezvoltare mai intuitiv și mai rapid.
- Nu trebuie să ne facem griji cu privire la stabilitatea și fiabilitatea aplicațiilor construite cu React- Native. Aceasta simplifică interschimbarea datelor astfel încât elementele copilului nu pot afecta datele părintelui. Dacă dezvoltatorul dorește să schimbe orice obiect, ar trebui să-și modifice „starea” componentei și să aplice actualizările în mod corespunzător - adică numai componentele permise vor fi actualizate. Marile companii precum Facebook, Instagram, Airbnb și multe altele folosesc React-Native pentru aplicațiile lor! Aceasta, în sine, spune foarte mult despre calitatea, stabilitatea și fiabilitatea aplicațiilor create folosind React-Native.

În concluzie React-Native este un framework interesant ce permite dezvoltatorilor web să creeze aplicații mobile robuste folosind cunoștințele lor existente în JavaScript și oferă o dezvoltare mobilă mai rapidă și o partajare mai eficientă a codurilor pe iOS, Android și pe Web, fără a sacrifica experiența utilizatorului sau calitatea aplicațiilor.

Deoarece aplicația „**BarbersApp**” este suportată în momentul de față doar de device-urile Android am să enumăr pe scurt etapele configurării și pregătirii mediului de dezvoltare doar pentru această platformă:

1. În primul rând trebuie să avem instalat **Node.js**, un mediu de execuție open-source, multi-platformă, care permite dezvoltatorilor să creeze tot felul de instrumente și aplicații

în JavaScript. După care vom instala **Watchman** oferit de Facebook pentru a urmări schimbările fișierelor și a reconstrui aplicația noastră.

2. Trebuie să instalăm **React Native Command Line Interface** (CLI) prin rularea următoarei comenzi în terminal.

```
npm install -g react-native-cli
```

Această comandă utilizează Node Package Manager (NPM) pentru a aduce toate uneltele necesare și pentru a le instala global.

3. Va fi necesară și instalarea platformei **Android Studio** ce necesită o serie de configurări suplimentare (JDK – minim versiunea 8, Android SDK, Android SDK Platform) și pregătirea unor device-uri virtuale (Android Virtual Device) unde ne vom testa aplicația în modul „live reload” pentru diferite versiuni de Android.

4. După toate aceste configurări putem genera un nou proiect prin rularea unei simple comenzi folosind React-Native command line interface:

```
react-native init BarbersApp
```

Această comandă va crea un nou proiect cu numele „BarbersApp” (vezi figura 1) ce va conține tot ce este necesar pentru a construi și rula o aplicație nativă. În acest nou proiect vor fi prezente o serie de fișiere/foldere dintre care cele mai importante sunt:

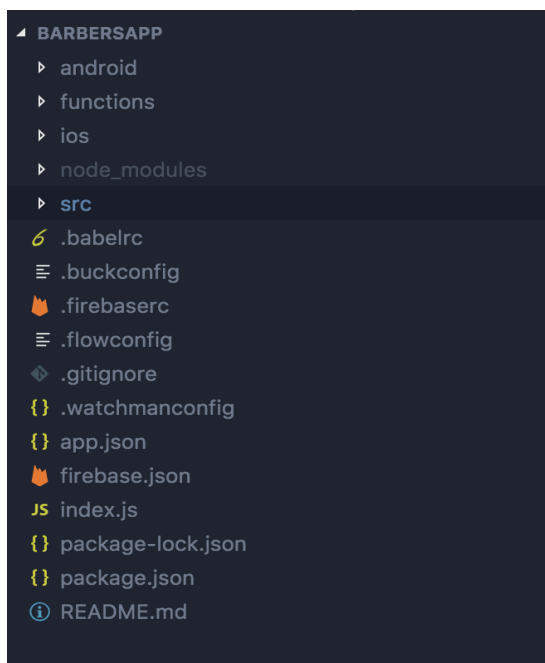
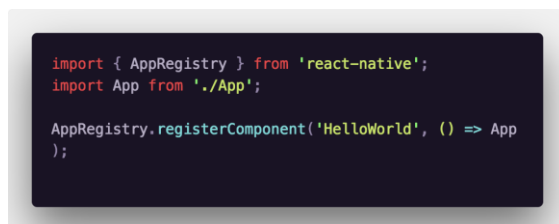


Figura 1: structura unui proiect React-Native

- **node_modules**: este un folder ce cuprinde pachetele necesare framework-ului React-Native și nu numai
- **index.js** este punctul de intrare și fișierul executat de unealta CLI
- **src** este folderul ce cuprinde componentele aplicației noastre (fișiere Js, imagini, video-uri)
- **ios** este un folder ce conține un proiect Xcode și codul necesar bootstrap-ului
- **android** este un folder ce conține codul necesar platformei Android, unde ne vom configura componentele native precum și alte pachete specifice.

Fișierul care randează efectiv aplicația noastră este `index.js` (vezi figura 2) aflat în folderul principal (`barbersapp`).



```
import { AppRegistry } from 'react-native';
import App from './App';

AppRegistry.registerComponent('HelloWorld', () => App
);
```

Figura 2: Structura fisierului de baza `index.js`

`AppRegistry`² este punctul de intrare pentru rularea aplicațiilor React-Native. Componenta principală „App” sau oricare altă componentă trebuie înregistrată folosind `AppRegistry.registerComponent` astfel încât sistemul nativ să poată încarcă pachetul aplicației noastre și să ruleze aplicația pornind `AppRegistry.runApplication`. Vom avea nevoie de un dispozitiv pentru rularea acesteia. Poate fi unul fizic, conectat printr-un cablu USB sau unul virtual (Android virtual device) configurat cu ajutorul platformei Android Studio. Dacă totul a fost configurat corect, putem vedea aplicația rulând pe unul dintre device-urile prezentate mai sus doar prin tastarea unui simple comenzi în folderul proiectului nostru.

```
React-native run-android
```

4.2. Firebase³

Firebase este o platformă de dezvoltare a aplicațiilor mobile și web care oferă dezvoltatorilor o multitudine de instrumente și servicii pentru ai ajuta să pună pe picioare aplicații de înaltă calitate.

Instrumentele oferite acoperă o mare parte a serviciilor pe care programatorii ar trebui să le construiască ei înșiși de la zero, printre care se numără analize ale activității și folosirii aplicațiilor, autentificare, baze de date, stocarea fișierelor, sistem de notificări și multe altele.

Acestea sunt găzduite/menținute în cloud, scalate și optimizate fără efort prea mare din partea programatorilor.

Când spun "găzduit/mentinut în cloud", vreau să spun că produsele au componente backend care sunt întreținute și exploatate integral de Google. SDK-urile (software development kit) clienților, furnizate de Firebase interacționează direct cu aceste servicii backend, fără a fi

² <https://facebook.github.io/react-native/docs/appregistry>

³ <https://console.firebase.google.com/>

necesară stabilirea niciunui middleware între aplicație și serviciu. Deci, dacă utilizați una dintre opțiunile bazei de date Firebase, scrieți în mod normal un cod pentru a interoga baza de date în aplicația dumneavoastră client.

Acest lucru este diferit de dezvoltarea tradițională a aplicațiilor, care implică, de obicei, scrierea atât a software-ului frontend, cât și a software-ului backend. Codul frontend doar invocă punctele finale API expuse de backend, iar codul backend face de fapt munca. Cu toate acestea, cu produsele Firebase, backend-ul tradițional este ușor ocolit, punând accentul pe partea de client unde se va stabili întreaga logică a aplicației, aceasta abordare conturând arhitectura **serverless**.

Accesul administrativ la fiecare dintre aceste produse este asigurat de consola Firebase.

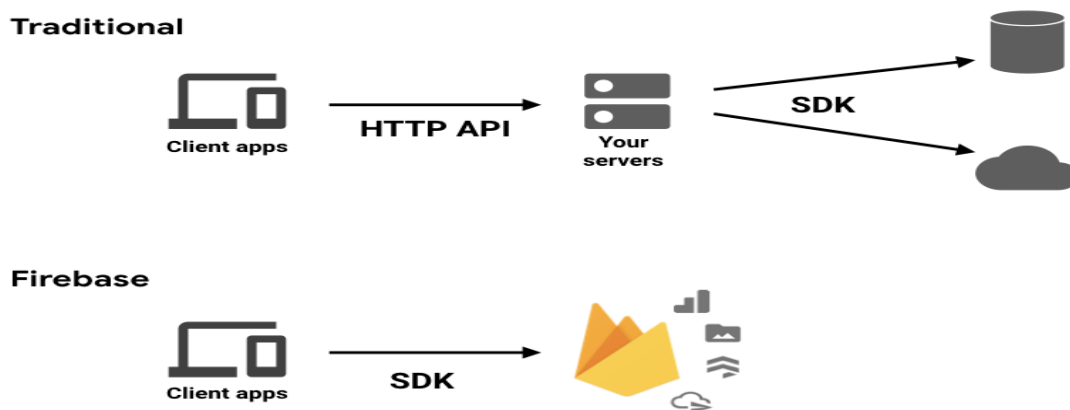
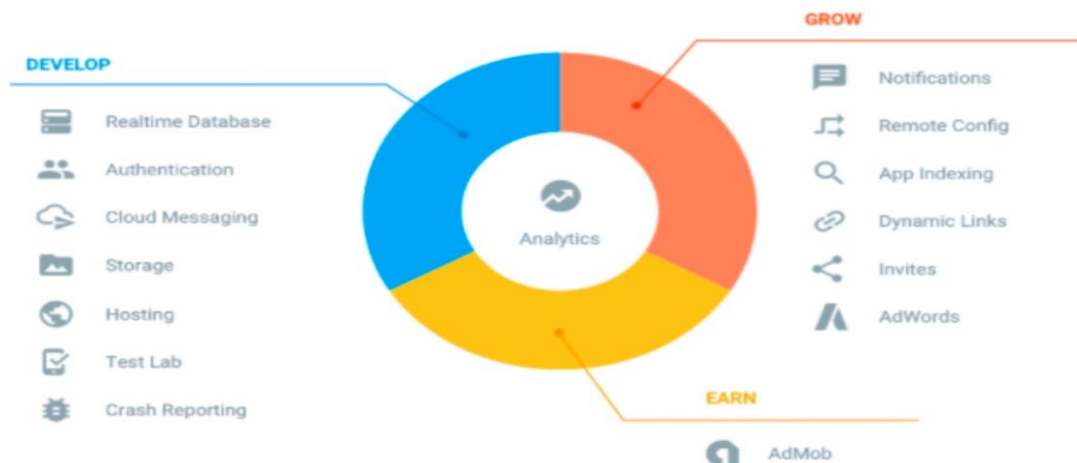


Figura 3: Diferențe dintre arhitectura tradițională și arhitectura serverless (Firebase)

Pentru ce tipuri de aplicații este bun Firebase?

Nu există nicio limită pentru tipurile de aplicații care pot fi ajutate de produsele Firebase. Există doar limite la platformele pe care se poate utiliza. iOS și Android sunt țintele principale pentru SDK-urile Firebase, și există un suport sporit pentru web, Flutter, Unity și C ++. De asemenea, trebuie să știți că există un SDK de administrare disponibil pentru o varietate de limbi, care să fie utilizat cu orice componente de backend pe care le-ați putea solicita.

Din cauza felului în care funcționează produsele Firebase, unii oameni ar putea să numească Firebase o „platformă ca serviciu” sau un „backend ca serviciu”. Nu m-am simțit niciodată confortabil înclinând Firebase complet într-una din aceste arii. Firebase se pretează ambelor arii și cuprinde servicii ce pot fi divizate în două mari grupe: **dezvoltare** și **creștere**.



Printre serviciile folosite în procesul de dezvoltare a aplicației „**BarbersApp**” se numără:

- Authentication - pentru logarea și identificarea unui user
- Realtime Database - baza de date NoSQL în timp real
- Cloud Storage - spațiu de stocare a fișierelor
- Cloud Functions - metode apelate în funcție de anumite evenimente (Pub/Sub)
- Cloud Messaging - sistem de notificări programate dinamic

Toate instrumentele prezentate mai sus vor fi detaliate în secțiunile următoare în care vom prezenta modul în care pot fi utilizate, ce plusuri aduc și impactul pe care l-au avut asupra aplicației „**BarbersApp**”.

4.2.1 Firebase Authentication⁴

Are grijă de conectarea și identificarea utilizatorilor. Acest produs este esențial pentru a asigura configurarea corectă a unora dintre celelalte produse, mai ales dacă trebuie să restricționați accesul la datele per utilizator (pe care aproape orice aplicație le va dori să le facă).

Un lucru important în ceea ce privește autentificarea Firebase este că facilitează efectuarea logărilor sigure, ceea ce este foarte dificil de implementat corect pe cont propriu. Aceste servicii suportă autentificare cu parole, număr de telefon și identități provenite de la diverși furnizori precum: Google, Facebook sau Twitter.

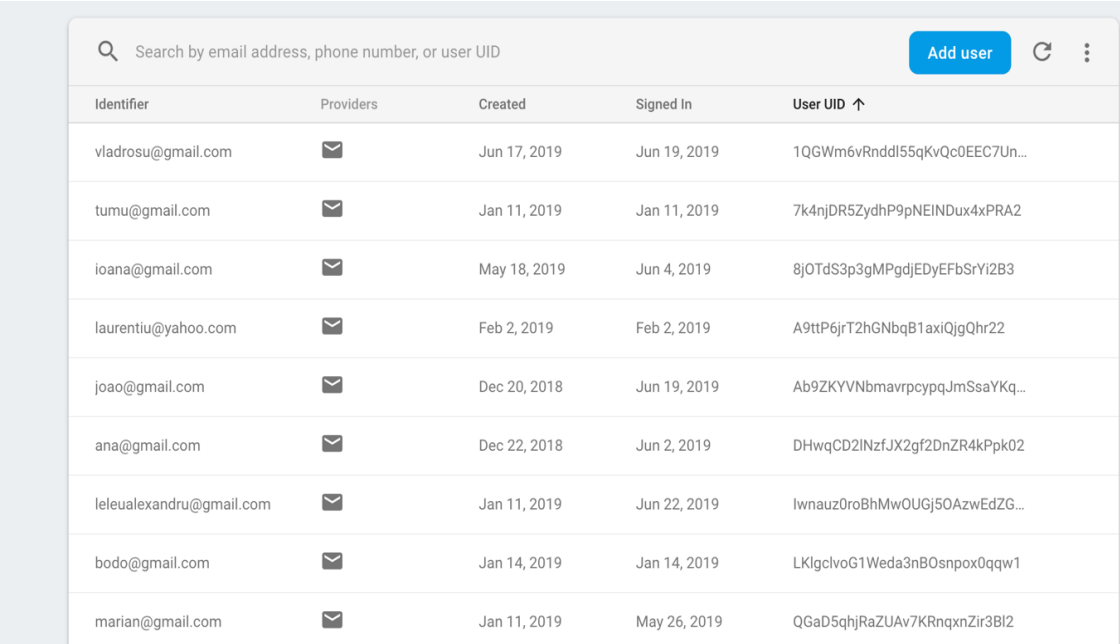
⁴ <https://firebase.google.com/docs/auth>

Utilizatorii aplicației „BarberApp” se pot conecta pe baza unui email și a unei parole. Verificările se fac doar la nivelul clientului, prin constrângerea formei email-ului și a complexității parolei. Nu au fost activate și metodele de validare a existenței email-ului folosit. O parte dintre utilizatori (clienții propriu-zisi ai bărbierilor) beneficiază și de logarea mai rapidă folosind un cont valid de Facebook.

Cum funcționează?

Pentru a înregistra un utilizator în aplicația noastră, primim mai întâi datele de autentificare de la utilizator. Aceste acreditări pot fi adresa de e-mail/parola utilizatorului sau un token OAuth de la un furnizor precum Facebook, în cazul de față. Apoi, trimitem aceste acreditări la Fișierul de autentificare Firebase SDK. Serviciile backend vor verifica aceste acreditări și vor returna un răspuns clientului.

După conectarea reușită cu succes, putem accesa informațiile de bază ale profilului și putem controla accesul utilizatorului la datele stocate în alte produse Firebase. De asemenea, putem utiliza token-ul de autentificare furnizat pentru a verifica identitatea utilizatorilor din propriile servicii backend.



Identifier	Providers	Created	Signed In	User UID ↑
vladrosu@gmail.com	📧	Jun 17, 2019	Jun 19, 2019	1QGwM6vRnddl55qKvQc0EEC7Un...
tumu@gmail.com	📧	Jan 11, 2019	Jan 11, 2019	7k4njDR5ZydhP9pNEINDux4xPRA2
ioana@gmail.com	📧	May 18, 2019	Jun 4, 2019	8jOTdS3p3gMPgdjEDyEFbSrYi2B3
laurentiu@yahoo.com	📧	Feb 2, 2019	Feb 2, 2019	A9ttP6jrT2hGNbqB1axiQjgQhr22
joao@gmail.com	📧	Dec 20, 2018	Jun 19, 2019	Ab9ZKYVNbmavrcypqJmSsaYKq...
ana@gmail.com	📧	Dec 22, 2018	Jun 2, 2019	DHwqCD2INzfJX2gf2DnZR4kPpk02
leleualexandru@gmail.com	📧	Jan 11, 2019	Jun 22, 2019	lwnauz0roBhMwOUGj5OAzwEdZG...
bodo@gmail.com	📧	Jan 14, 2019	Jan 14, 2019	LKlgclvoG1Weda3nB0snpox0qqw1
marian@gmail.com	📧	Jan 11, 2019	May 26, 2019	QGaD5qhjRaZUAv7KRnqxnZir3BI2

Figura 4: Utilizatorii aplicației BarbersApp autentificati pe baza unui email si a unei parole

Pentru a crea un cont nou și pentru a loga un utilizator existent avem nevoie doar de două metode specifice autentificării cu email și parolă:

```
firebase.auth().createUserWithEmailAndPassword(email, password)
firebase.auth().signInWithEmailAndPassword(email, password)
```

După logarea fiecărui utilizator se creează o sesiune unică pentru fiecare device ce este configurată fără termen de expirare (nu va necesita autentificare la fiecare accesare a aplicației), dezactivându-se automat la delogare.

4.2.2 Realtime Database⁵

Firebase Realtime Database este o baza de date NoSQL în care datele pot fi stocate ca obiecte în format JSON. Deși baza de date utilizează un arbore JSON, datele stocate pot fi reprezentate ca anumite tipuri native care corespund tipurilor de JSON disponibile pentru a vă ajuta să scrieți cod mai ușor de întreținut.

Ceea ce este cu adevărat special la aceste baze de date este că oferă actualizări în timp real ale datelor. Utilizând SDK-ul client asignat putem configura un "listener" pentru locația datelor pe care aplicația noastră dorește să le utilizeze, iar ascultătorul este invocat cu acele date în mod repetat, de fiecare dată când se observă o modificare. Acest lucru permite să păstrăm afișarea aplicației în stare proaspătă, fără a fi nevoie să efectuăm cereri datelor de interes.

Atunci când utilizatorii nu mai au acces la internet, seturile SDK ale bazei de date în timp real utilizează cache-ul local pe dispozitiv pentru a afișa și schimba modificările. Când dispozitivul este conectat, datele locale sunt sincronizate automat.

Baza de date Realtime poate fi integrată și cu Firebase Authentication pentru a oferi un proces de autentificare simplu și intuitiv (folosirea token-ului de la autentificare ca și cheie unică în structura bazei de date).

După cum se poate observa în Figura 5 de mai jos, baza de date a aplicației „BarbersApp” cuprinde 4 ramuri principale:

- Appointments
- Barbers
- Customers
- Notifications

Avem două obiecte principale (barbers și customers) ce cuprind informații specifice fiecărui tip de utilizator și mapări între diferite chei unice pentru realizarea logicii programărilor dar și a notificărilor. Fiecare cont are asignat token-ul unic obținut în urma autentificării (atât prin email sau contul de facebook).

⁵ <https://firebase.google.com/docs/database>

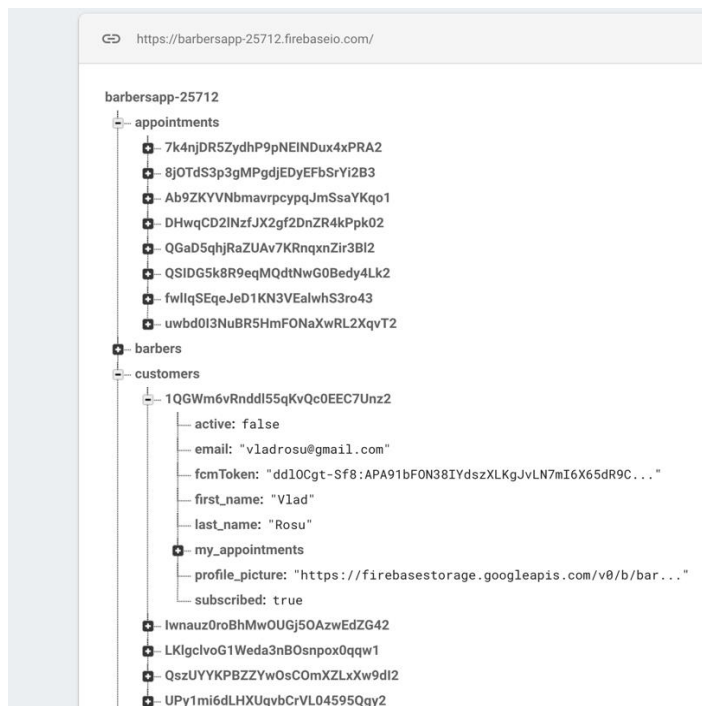


Figura 5: Structura bazei de date a aplicației „BarbersApp”

Citirea și scrierea într-o bază de date realtime se face folosind metode ale modului `firebase.database()` ce creează o referință către serviciile expuse de Firebase.

Câteva exemple:

a) Vom putea insera noi date pentru un utilizator cu id-ul „newclient” în obiectul de baza „customers” astfel:

```
firebase.database().ref('customers'+',newclient').set({datele noastre})
```

b) Vom putea accesa și „asculta” eventualele modificări prin crearea unui „listener” la o anumită locație folosind metodele modului `firebase.database.Reference`: `on()` sau `once()`.

Pe lângă acestea mai putem folosi metode pentru ștergere, actualizare a datelor, putem crea query-uri mai complexe și diverse logici pentru sortarea datelor în funcție de anumite valori.

Când lucrăm cu date ce ar putea fi corupte de modificări concurente, cum ar fi în cazul aplicației „BarbersApp” cererea simultană a rezervării aceluiași interval orar de cel puțin doi clienți, putem utiliza o operație de tranzacție. Putem oferi acestei operații o funcție de actualizare și un apel de încheiere opțional de finalizare. Funcția de actualizare preia starea curentă a datelor ca argument și returnează noua stare pe care dorim să o scriem. Dacă un alt client scrie la aceeași locație înainte ca noua noastră valoare să fie scrisă cu succes, funcția de actualizare este chemată din nou cu noua valoare curentă, iar scrierea este reluată.

4.2.3 Firebase Storage⁶

Firebase Storage este o soluție autonomă, de sine stătătoare, pentru încărcarea conținutului generat de utilizatori, precum imaginile și videoclipurile de pe un dispozitiv iOS și Android, precum și pe Web. Firebase Storage este conceput special pentru a scala aplicațiile, pentru a asigura securitatea și flexibilitatea rețelei.

Firebase Storage folosește un sistem simplu folder/file_name pentru a structura datele sale.

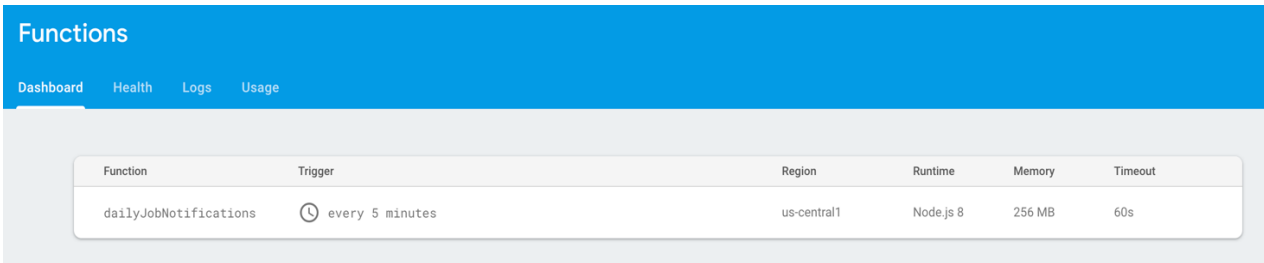
„BarbersApp” se folosește de acest serviciu doar pentru stocarea imaginii de profil a utilizatorilor.

4.2.4 Cloud Functions

Cloud Functions este un alt produs Google Cloud care funcționează bine cu alte produse Firebase și Cloud. Folosind SDK-urile Firebase pentru funcțiile Cloud, putem scrie și implementa coduri care rulează pe infrastructura Google "serverless", care răspunde în mod automat la evenimente provenite de la alte produse Firebase, precum modificări în baza de date, noi autentificări, job-uri ce rulează cu o anumită frecvență și multe altele⁷.

Când spun "serverless", nu sugerez lipsa serverelor. Cu o arhitectură „serverless”, există încă servere în joc, pur și simplu nu trebuie să știm prea multe despre ele. Nu furnizăm, întreținem, scalăm ci trebuie doar să scriem și să implementăm cod valid (functii stateless) și Google va face restul.

„BarbersApp” folosește un astfel de serviciu pentru verificarea utilizatorilor ce trebuie să primească notificări înainte de data programării (reminder ce trebuie trimis cu o oră înainte de fiecare programare confirmată).



The screenshot shows the Google Cloud Functions dashboard. At the top, there's a blue header with the word 'Functions'. Below it, there are tabs for 'Dashboard', 'Health', 'Logs', and 'Usage'. The 'Dashboard' tab is selected. Below the tabs, there's a table with the following columns: Function, Trigger, Region, Runtime, Memory, and Timeout. The table contains one entry: 'dailyJobNotifications' with a trigger of 'every 5 minutes' (indicated by a clock icon), region 'us-central1', runtime 'Node.js 8', memory '256 MB', and timeout '60s'.


Function	Trigger	Region	Runtime	Memory	Timeout
dailyJobNotifications	 every 5 minutes	us-central1	Node.js 8	256 MB	60s

Figura 6: Funcția *dailyJobNotifications* ce este rulată la fiecare 5 minute pentru a verifica utilizatorii ce trebuie să primească notificări

⁶ <https://firebase.google.com/docs/storage>

⁷ <https://github.com/firebase/functions-samples/>

Folosind **Firestore Cloud Messaging** (FCM) se realizează o conexiune între servere și device-urile ce permit primirea notificărilor.

FCM poate trimite mesaje instantaneu sau în viitor în fusul orar local al utilizatorului. Poate trimite date personalizate ale aplicațiilor, cum ar fi setarea priorităților, sunetelor și datele de expirare, precum și urmărirea evenimentelor personalizate.

Așadar utilizatorii „BarbesApp” beneficiază de remindere cu o zi sau o ora înainte de data programării.

4.3. Concluzii

Toate tehnologiile utilizate în această aplicație au avut un rol esențial, fiecare dintre acestea contribuind la realizarea produsului final. Partea dificilă a fost dată de modelarea și stabilirea relațiilor între entități (obiectele JSON) precum și păstrarea integrității serviciului de programări. De asemenea efortul pentru redarea funcționalităților într-un mod intuitiv și captivant nu e de neglijat întrucât diversele componente native aveau nevoie de ajustări în funcție de dimensiunile dispozitivelor. Pot spune că dezvoltarea mobile este mai costisitoare și mai interesantă deoarece implică o serie de configurări specifice device-urilor și o atenție sporită la detalii privind interacțiunea cu userii. Mai mult decât atât, instrumentele oferite de Firebase au adus un plus aplicației din punct de vedere al performanței și dinamicității, oferind datele în timp real.

5. Prezentarea aplicației

În acest capitol voi descrie pe larg procesul de dezvoltare al aplicației „BarbersApp” insistând pe partea funcțională și tehnicile folosite în obținerea produsului dorit.

Așadar voi începe cu prezentarea funcționalităților principale, ce servicii și beneficii oferă utilizatorilor dar și modul în care putem folosi aplicația.

5.1. Experiența utilizatorului

Experiența utilizatorului reprezintă modul în care se simte o persoană atunci când interacționează cu un produs software și este unul dintre factorii decizionali pentru succesul unei aplicații. Practic, prima impresie și prima interacțiune cu produsul joacă un rol extrem de important în păstrarea utilizatorului și în atragerea celor noi. De aceea trebuie acordată o atenție sporită design-ului și modului în care sunt prezentate informațiile pentru a ușura pe cât posibil funcționalitățile existente. Experiența utilizatorului este acoperită de mai mulți factori, o parte dintre ei fiind controlabili de designeri și dezvoltatori, alții fiind reprezentați de preferințele utilizatorului. Acești factori includ uzabilitatea, accesibilitatea, performanța, estetica, designul, utilitatea, simplitatea, ergonomia, cromatică și marketingul.

Experiența utilizatorului se concentrează pe creșterea satisfacției, prin îmbunătățirea modului în care oamenii interacționează cu site-urile, aplicațiile și dispozitivele din viața noastră. Cu alte cuvinte, UX (user experience) face ușor de utilizat lucrurile complexe.

Experiența utilizatorului este o practică ce stă la intersecția dintre știința comportamentală, dezvoltarea software-ului indiferent de platforma și cunoștințele asupra domeniului în care activează. Este o abordare umană cu scopul de a studia cum pot oamenii să înțeleagă tehnologia și cum pot oferi cele mai bune experiențe mobile posibile, considerând următoarele:

- În mediul online, 75% din factorii care afectează prima impresie sunt legați de design. Aceste prime impresii sunt extrem de importante. Emoțiile utilizatorilor neimpresionați sunt adesea neiertătoare.
- 50% dintre utilizatorii mobile dezinstalează o aplicație în urma unei experiențe nefericite sau a unei performanțe scăzute.

Așadar, pentru a respecta cerințele legate de experiența utilizatorului, informația oferită acestuia trebuie să respecte următoarele:

- Aplicația trebuie să livreze conținut ușor de digerat conform cu ceea ce pretinde că trebuie să ofere;

- Aplicația trebuie să fie intuitivă și ușor de navigat;
- Designul trebuie să fie atractiv, inteligent, responsive și ușor de urmărit;
- Utilizatorii nu trebuie să aiba probleme în găsirea informațiilor cheie în timp ce folosesc aplicația;
- Conținutul trebuie să fie accesibil oricui;
- Aplicația trebuie să ofere servicii credibile ce rulează conform așteptărilor.

5.2. Despre „BarbersApp”

"BarbersApp" este o aplicație care oferă toate instrumentele necesare unui frizer pentru a se conecta cu clienții existenți și pentru a câștiga alții noi. Clienții pot găsi cu ușurință magazinul/frizeriile din apropiere într-un mod plăcut și interactiv prin sistemul de hărți pus la dispoziție.

Aplicația suportă două tipuri de conturi destinate „clienților” și „barberilor sau barbershop-urilor”.

Clienți au posibilitatea:

- De a-și personaliza propriul cont;
- De a vedea locațiile celor mai apropiați frizeri;
- De a accesa topul celor mai apreciați frizeri;
- De a-și rezerva un interval orar în funcție de serviciul ales;
- De a vedea programările confirmate și de a primi notificări din timp;

„Barberii” au posibilitatea:

- De a-și personaliza propriul cont și de a specifica orarul în care pot activa, precum și timpul în care se află în pauză;
- Pot activa/dezactiva serviciile pentru o perioadă;
- Pot vedea intervalele orare rezervate pentru următoarele 7 zile;
- Pot oferi detalii utile clienților precum: numărul de telefon, adresa, lista serviciilor, lista prețurilor.

Astfel prin folosirea aplicației poți vedea intervalele și serviciile disponibile pentru diverși „barberi” din apropiere într-un mod mult mai rapid decât tradiționala programare telefonică.

5.3. Descrierea aplicației și funcționalități

„BarbersApp” este o aplicație mobilă, nativă, ce este funcțională în momentul de față pe dispozitivele Android, având un design adecvat temei în care se încadrează (booking app).

La accesarea aplicației prima pagină este cea de selectare a tipului de cont dorit urmată de cea de logare/înregistrare (figura 7, 8, 9 și 10).

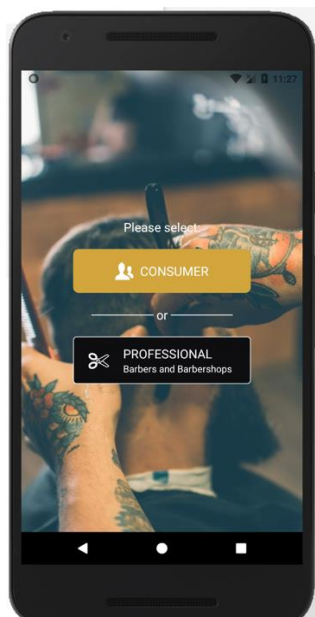


Figura 7

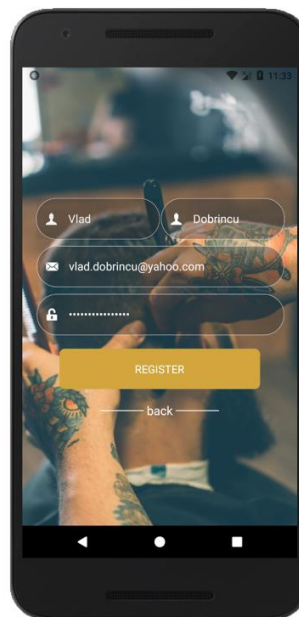


Figura 8

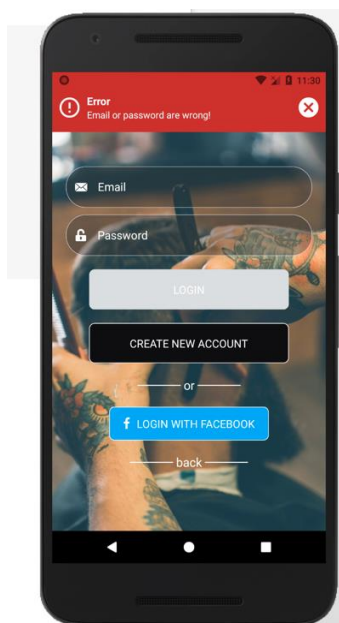


Figura 9

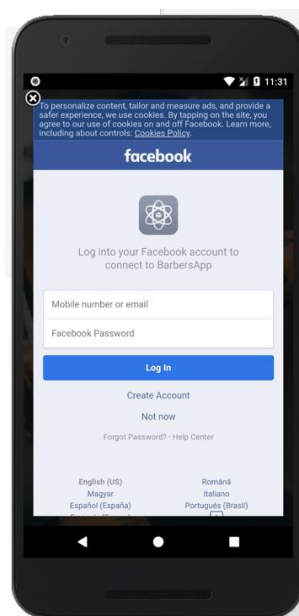


Figura 10

Fiecare utilizator poate opta pentru un cont de „client” sau „barber/barbershop” și va putea accesa aplicația doar prin completarea unui email valid și a unei parole de dificultate medie ori folosind informațiile contului de Facebook.

Acum vom continua cu prezentarea funcționalităților oferite utilizatorilor de tip „barber” sau „barbershop”. Fiecare nou utilizator de tip „barber” va trebui să completeze o

gamă de informații cu privire la locația sediului, numărul de telefon precum și intervalul orar în care va fi disponibil (figura 11, 12).

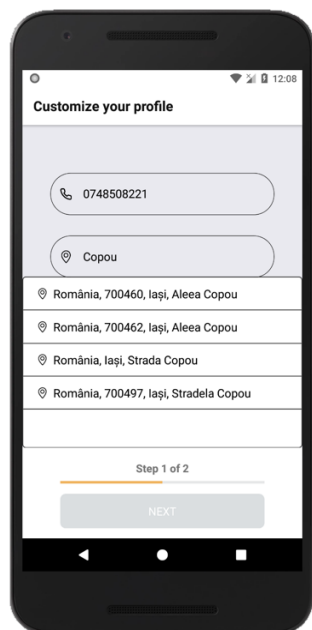


Figura 11

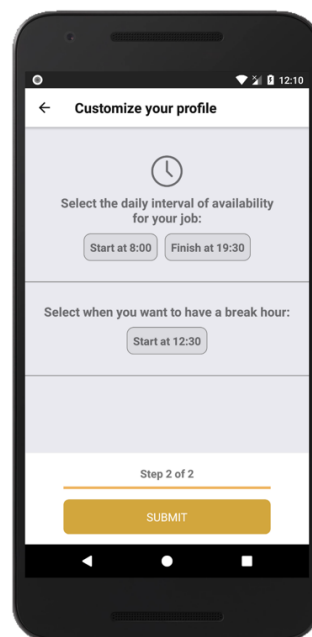


Figura 12

Informațiile precizate mai sus vor fi cerute spre completare după prima logare putând fi mai apoi schimbate în secțiunea de setări a aplicației (figura 13). Fiecare utilizator își poate configura propriul profil după bunul plac. Datele oferite vor fi vizibile posibililor clienți. Fiecare „barber” are acces la un sistem prin care poate verifica rezervările pentru următoarele 7 zile, intervalele orare ocupate precum și serviciile pentru care au optat clienții (figura 14).

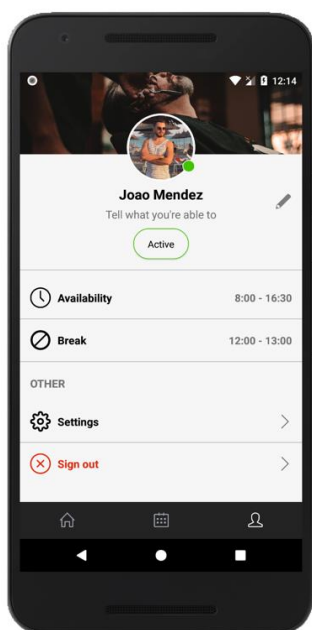


Figura 13

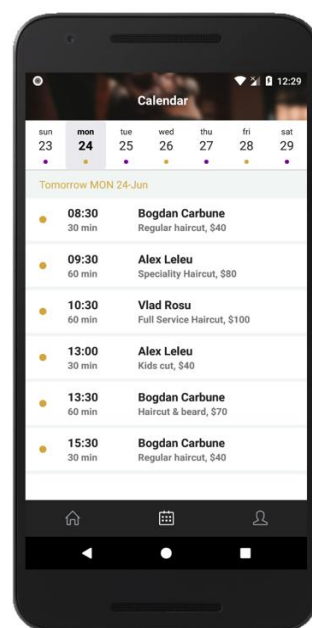


Figura 14

Datele vor fi actualizate în timp real iar o cerere pentru a fi salvată cu succes va trebui să treacă de o serie de validări și verificări pentru a asigura integritatea și unicitatea fiecărei

rezervări. Dar despre logica acestui sistem de programări și restricțiile stabilite vom discuta în secțiunile ce urmează.

Acum vom prezenta funcționalitățile întâlnite de utilizatorii ce optează pentru crearea unui cont de „consumer/client”. Înainte de a se loga trebuie să confirme accesul aplicației la locația dispozitivului deoarece „BarbersApp” oferă o listă a celor mai apropiați frizeri în funcție de locația acestuia.

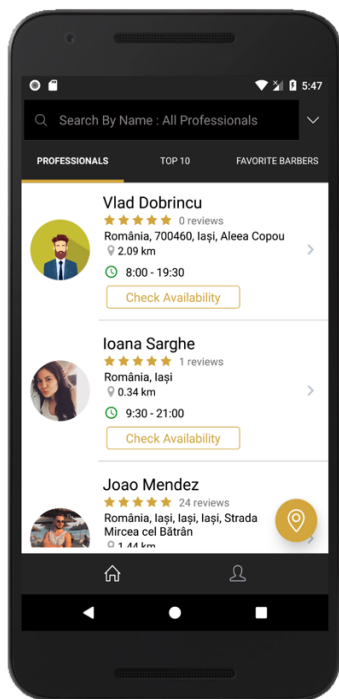


Figura 15

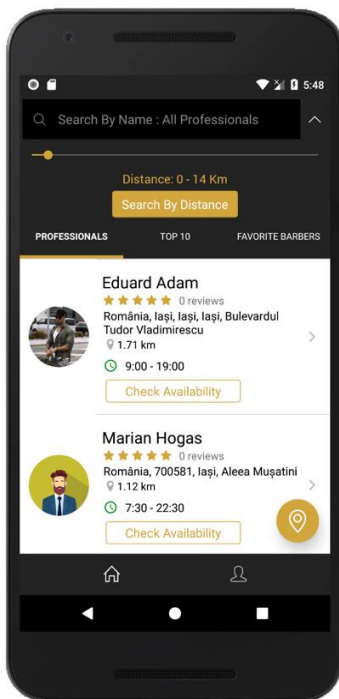


Figura 16

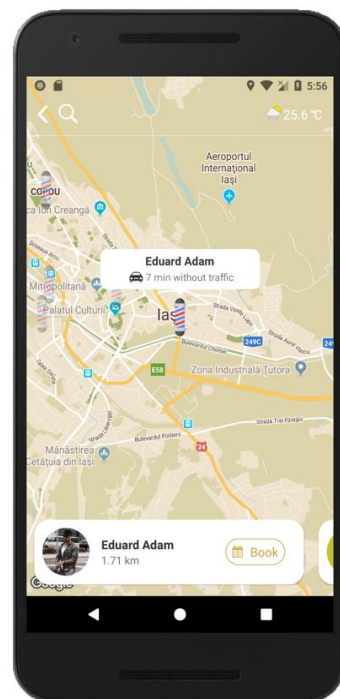


Figura 17

Prima pagina (Home) cuprinde o secțiune de filtrare a datelor putând astfel selecta bărbierii după numele lor sau distanța față de utilizatorul curent (figura 16). Userii au acces, de asemenea, la diverse informații precum locația detaliată a saloanelor, distanța față de acestea și programul de activitate.

Google Maps

Google Maps este cunoscut pentru funcționalitățile pe care le are: vizionarea de hărți 2D, 3D sau street view, consultarea traficului în timp real în anumite zone, oferirea de indicii în privința alegerii unui drum pentru a ajunge la o anumită destinație sau chiar utilizarea unui sistem de navigație GPS cu îndrumare vocală, dar ceea ce este interesant în cazul de față este opțiunea Latitude/Longitudine prin care utilizatorul își poate înregistra locația. Vizualizarea bărbierilor se realizează pe hărțile 2D, așa cum poate fi observat și în Figura 17, utilizatorii putând urmări într-un mod plăcut și intuitiv locațiile celor mai apropiați bărbieri precum și diverse informații despre trafic.

Here Api⁸

Am folosit acest pachet de servicii pentru a ajuta utilizatorul în completarea și setarea unei locații prin metodele de autocompletare a posibilelor străzi. Am folosit și un serviciu de conversie într-un sistem de geo-coordonate (latitudine/longitudine) pentru a putea fi instanțiată pe mapă. Am folosit acest Api și pentru preluarea informațiilor despre trafic și distanțe.

Pentru integrarea instrumentelor prezentate mai sus am folosit componente dezvoltate de comunitate (exemple: *react-native-maps*⁹, *react-native-heremaps*) ce sunt compatibile ecosistemului React-Native. Acestea necesită anumite configurări și dependențe specifice platformei Android iar partea dificilă a constat în specificarea versiunilor necesare, a meta datelor și a proprietăților specifice fiecărui provider.

Mai mult decât atât pentru a putea accesa locația unui dispozitiv, fișierele locale dreptul de a primi notificări de la o anumită aplicație este necesară includerea unor permisiuni din partea utilizatorului în *AndroidManifest.xml*.

Fișierul Manifest

Fiecare proiect Android include un fișier manifest, *AndroidManifest.xml*, ce permite definirea structurii și meta datei aplicației, componentele și cerințele acesteia. Acesta include noduri pentru diverse componentele (activități, servicii, notificari ș.a.), filtre pentru intenții (intent filters) și permisiuni ce determină modul cum acestea interacționează.

Fișierul manifest permite totodată specificarea de meta dată cu privire la aplicația curentă (iconițe și teme folosite), precum și noduri de nivel înalt ce includ setări de securitate, teste și cerințe hardware ale aplicației.

Manifestul este format dintr-o rădăcină reprezentată de eticheta `<manifest>` ce specifică prin atributul *xmlns:android* elementele de securitate ce sunt utilizate în cadrul fișierului, prin attributele *android:versionCode* și *android:versionName* versiunea aplicației, iar prin *android:package* pachetul principal al aplicației.

Următoarea listă oferă o perspectivă asupra nodurilor ce pot fi specificate în fișierul manifest și rolul acestora în construcția aplicației.

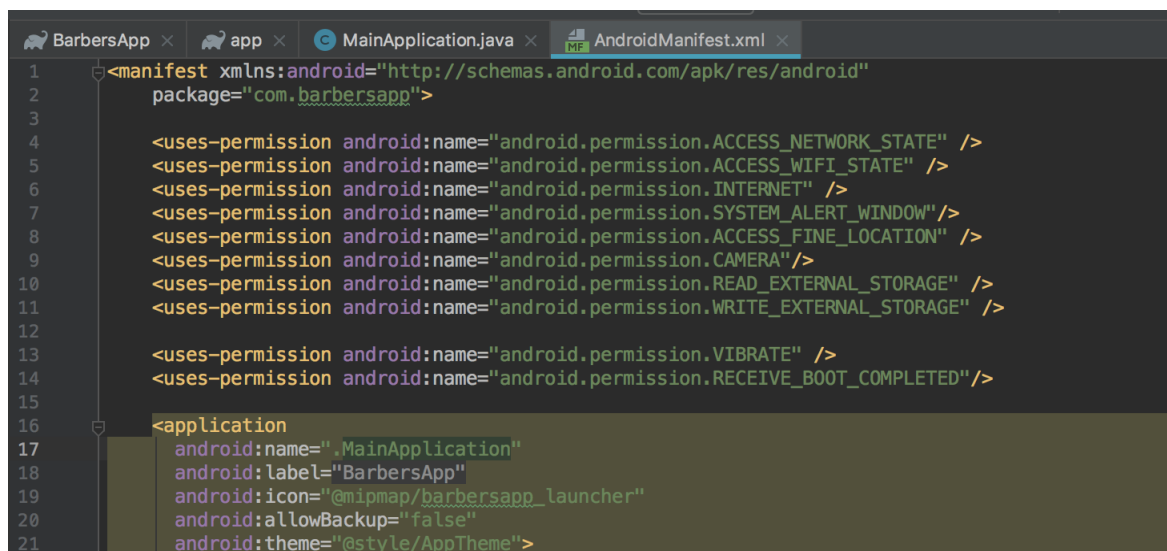
- **uses-sdk** marchează versiunea SDK ce trebuie să fie disponibilă pe un dispozitiv Android pentru ca aplicația să funcționeze precum și versiunea pe care s-a realizat și testat aplicația

⁸ <https://developer.here.com/documentation#geocoder>

⁹ <https://github.com/react-native-community/react-native-maps>

- **uses-configuration** specifică o combinație dintre mecanismele de intrare suportate de aplicație, acestea făcând referire la tastatura disponibilă, la modul de navigare într-o fereastră, și la modul de atingere.
- **uses-feature** ilustrează cerințele hardware ale aplicației și împiedică instalarea acesteia pe un dispozitiv ce nu respectă aceste condiții
- **supports-screens** enumeră dimensiunile ecranelor pe care aplicația le suportă
- **application** este un nod singular ce specifică meta data aplicației precum și componentele acesteia, constituind un container pentru noduri precum activity, service, receiver sau provider
- **uses-permission** stabilește permisiunile hardware de care are nevoie aplicația pentru a putea fi utilizată așa cum este așteptat și vor fi afișate utilizatorului înainte de a fi instalată
- **permission** este utilizată pentru restricționarea accesului unei alte aplicații la o componenta a sa. Pentru a avea acces la aceste elemente protejate, celelalte aplicații vor trebui să includă în manifestul lor noduri de tip uses-permission.
- **instrumentation** oferă un cadru de testare pentru componentele aplicației la momentul execuției. Acestea oferă mijloace de monitorizare a aplicației și a interacțiunii cu resursele sistemului.

Așadar pentru folosirea serviciilor furnizate de Google Maps, accesului la fișierele locale și primirea de notificări, a fost necesară includerea următoarelor permisiuni:



```

1 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2   package="com.barbersapp">
3
4   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
5   <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
6   <uses-permission android:name="android.permission.INTERNET" />
7   <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
8   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
9   <uses-permission android:name="android.permission.CAMERA"/>
10  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
11  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
12
13  <uses-permission android:name="android.permission.VIBRATE" />
14  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
15
16  <application
17    android:name=".MainApplication"
18    android:label="BarbersApp"
19    android:icon="@mipmap/barbersapp_launcher"
20    android:allowBackup="false"
21    android:theme="@style/AppTheme">

```

Figura 18: Structura fișierului AndroidManifest.xml pentru aplicația BarbersApp

Acum să revenim la celelalte funcționalități oferite utilizatorilor de tip „customer”. Aplicația „BarbersApp” oferă posibilitatea de a vedea topul celor mai apreciați frizeri precum și posibilitatea de a avea o listă proprie a celor preferați. Cei ce doresc să beneficieze de servicii

pot vedea lista completă a rezervărilor și pot selecta un interval liber în funcție de posibilitățile fiecăruia.

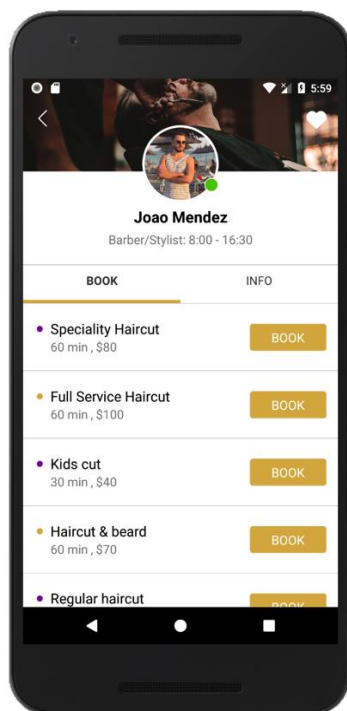


Figura 19

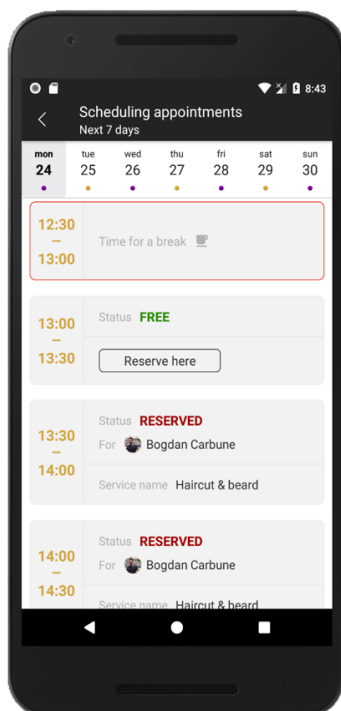


Figura 20

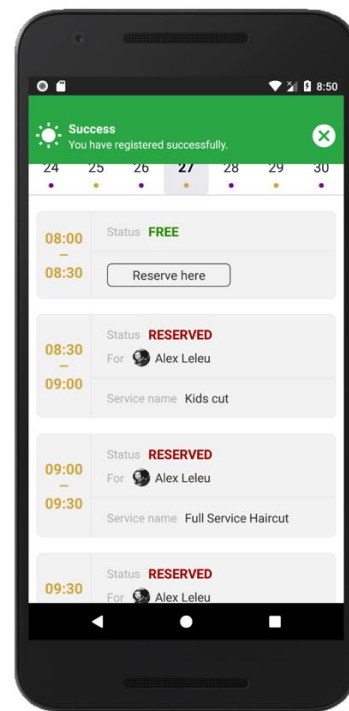


Figura 21

Fiecare slot disponibil reprezintă 30 de minute de activitate iar numărul total este creat dinamic în funcție de intervalul orar selectat de către barber (figura 20). Înainte de a fi confirmată, o cerere trece printr-o serie de verificări pentru a menține consistența datelor (figura 21). Dacă un serviciu necesită 60 de minute cererea confirmată va ocupa două sloturi succesive disponibile. Orice cerere confirmată va fi automat actualizată pentru toți ceilalți utilizatori menținând unicitatea pentru fiecare slot disponibil și evitând modificările concurente. Acest lucru este posibil datorită capacității bazei de date NoSQL și instrumentelor oferite de Firebase pentru a menține datele actualizate în timp real.

Exemplu: pentru a menține „fresh” datele specifice secțiunii pentru programări trebuie să atasăm un *listener* entității ce conține toate aceste informații, listener ce conține un „callback” ce va fi apelat și va întoarce datele updatate după fiecare modificare. (figura 22)

```

52     initializeAppointmentsList = () => {
53         const { barberInfo, listDate, appointmentsList } = this.props;
54         firebase
55             .database()
56             .ref("appointments")
57             .child(barberInfo.barberId)
58             .child(listDate)
59             .on("value", () => {
95                 // ...
96             });
97     };

```

Figura 22

6. Baze de date nerelaționale actualizate în timp real

Contextul actual

În prezent, bazele de date tradiționale sunt puse la încercare din ce în ce mai mult de noile tipuri de aplicații care le folosesc. Aceste tipuri de aplicații utilizează de regulă o cantitate mare de date complexe. Dacă în trecut pentru o mare perioadă de timp bazele de date relaționale esențiale pentru aplicațiile web sau mobile erau MySQL, astăzi aceste baze de date relaționale întâmpină multe dificultăți în lucrul cu cantități mari de date dar și probleme privind scalabilitatea și menținerea acestora. Pe piața în care activează MySQL au pătruns furnizorii de soluții de baze de date cloud. Aceste baze de date cloud poartă numele de NoSQL - Not only SQL și sunt baze de date non-relaționale. Dezvoltarea NoSQL și NewSQL amenință din ce în ce mai mult monopolul MySQL.

Diversele baze de date NoSQL existente azi pe piață prezintă diferite abordări. Ceea ce au în comun este faptul că nu sunt relaționale, adică nu au o structură bazată pe tabele și relații între acestea.

Principalul avantaj este acela că permit lucrul eficient cu date nestructurate precum email, multimedia, procesoare de text, accesând mult mai rapid anumite secvențe de date prin referențiere.

În prezent există multe companii care au dezvoltat propriile baze de date NoSQL. Cele mai populare sunt cele dezvoltate de către companiile mari Web, precum Amazon și Google, din nevoia de a procesa cantități mari de date. Acestea au dezvoltat Dynamo și Big Table ce stau la baza multor alte baze de date NoSQL existente acum pe piață. Astăzi avem multe opțiuni precum MongoDB, Cassandra, Redis, Couchbase, DynamoDB și Cosmos DB câștigând în popularitate, în creșterea comunității de utilizatori și adăugând rapid mai multe caracteristici. Există, de asemenea, patru tipuri de baze de date NoSQL:

- Key value: datele sunt stocate ca nume de atribut sau chei cu valori (ex.: Firebase)
- Document: conține diverse perechi diferite „cheie-valoare”
- Graph: utilizat pentru a stoca date referitoare la conexiuni sau rețele
- Column: datele sunt stocate în coloane în loc de rânduri

Unul dintre motivele apariției NoSQL constă în nevoia aplicațiilor de a manipula mai rapid cantități mari de date pentru a putea rămâne competitive. Cantitatea de informație digitală la nivel mondial este măsurată în exabytes. Conform unui studiu realizat de Universitatea Southern California cantitatea de date adăugată în 2006 a fost de 161 de exabytes. Doar un an mai târziu, în 2007 capacitatea totală s-a ridicat la 295 de exabytes,

reprezentând o creștere substanțială. Altfel spus, există o cantitate mare de informație în lume și aceasta crește exponențial. De aici survine și nevoia de baze de date web, în cloud, ce suportă cantități mari de date.

Conform unui studiu realizat de 451 Group Research intitulat MySQL vs. NoSQL and NewSQL între 2009 și 2011 s-a înregistrat o scădere în utilizarea MySQL de la 82% la 73%. Studiul a fost efectuat asupra unui eșantion compus din 347 de utilizatori de baze de date opensource.

Când să folosim o baza de date SQL în schimbul uneia NoSQL?

1. Atunci când lucrăm cu query-uri, interogări și rapoarte complexe deoarece cu SQL putem construi scripturi ce prelucrează și prezintă datele oferind o gamă largă de metode și funcții. Rularea interogărilor în NoSQL este posibilă, dar mult mai lentă în momentul de față.
2. Atunci când avem o aplicație de tranzacționare ridicată. Bazele de date SQL se potrivesc mai bine tranzacțiilor grele sau complexe, deoarece sunt mai stabile și asigură integritatea datelor.
3. Când trebuie să asigurăm respectarea ACID (Atomicity, Consistency, Isolation, Durability) sau să definim exact cum interacționează tranzacțiile cu o bază de date. Nu anticipați o mulțime de schimbări sau de creștere.
4. Dacă nu lucrăm cu un volum mare de date sau cu multe tipuri de date, NoSQL ar fi suprasolicitat.

Când să folosim o baza de date NoSQL în schimbul uneia SQL?

1. Dacă adăugăm în mod constant noi funcții, tipuri de date și este dificil să anticipăm modul în care aplicația va crește în timp.
 - Modificarea unui model de date SQL poate fi dificilă și necesită modificări de cod. Mult timp este investit în proiectarea modelului de date, deoarece modificările vor avea impact asupra tuturor sau asupra majorității straturilor din aplicație.
 - În NoSQL, lucrăm cu o schemă foarte flexibilă sau fără schemă predefinită. Procesul de modelare a datelor este iterativ și adaptabil. Schimbarea structurii sau schemei nu va afecta ciclurile de dezvoltare sau nu va crea nicio perioadă de întrerupere a aplicației.

2. Dacă nu suntem preocupați 100% de consecvența datelor și integritatea datelor nu este obiectivul nostru de top. Acest lucru este legat de cerința SQL de mai sus pentru conformitatea cu ACID. De exemplu, cu platformele de social media, nu este important ca toată lumea să vadă noua postare exact în același timp, ceea ce înseamnă că nu consistența datelor este o prioritate.
3. Dacă avem o mulțime de date, multe tipuri diferite, iar nevoile vor crește numai în timp. NoSQL facilitează stocarea tuturor tipurilor de date împreună și fără a trebui să se investească timp în definirea tipului de date pe care îl stocăm în avans.
4. Dacă datele trebuie să fie scalate. După cum sa discutat mai sus, NoSQL oferă o flexibilitate mult mai mare și capacitatea de a controla costurile pe măsură ce datele au nevoie de schimbări.

De ce am ales să folosesc NoSQL Firebase Realtime database?

1. Deoarece aplicația „BarbersApp” nu necesită o atenție sporită asupra conceptelor ACID iar serviciile furnizate nu stăteau la baza unor tranzacții și interogări complexe asupra bazei de date.
2. Aplicația „BarbersApp” avea nevoie de suportul diverselor tipuri de date (Dates, String, Number, Ids/Tokens) și accesarea rapidă a acestora prin referință.
3. Baza de date oferită de Firebase menține datele actualizate pentru toți userii asigurați, lucru foarte important în păstrarea consistenței pentru sistemul de programări.
4. Capacitatea sincronizării datelor în maniera real-time a adus un plus de valoare modului de interacțiune cu aplicația (experiența utilizatorului îmbunătățită).
5. Am decis să evit concepul de stocare a datelor în tabele deoarece structura sub forma de obiecte JSON încapsulate se îmbină perfect cu structura datelor pe partea de client (logica de pe frontend).
6. Această bază de date se îmbină perfect cu alte servicii precum: Authentication și Cloud Messaging deoarece au suport pentru a comunica între ele. Funcția rulată în cloud pentru trimiterea mesajelor clienților asigurați „ascultă” modificările asupra bazei de date.

Saltul de la SQL la Firebase NoSQL database:

Ca și noi utilizatori Firebase sunt sigur că încă tindem să găsim soluții pentru toate posibilitățile de interogare pe care le-am folosit și învățat în facultate, atunci când lucrăm cu bazele de date MySQL sau Oracle. Bazele de date SQL utilizează tabele pentru a stoca date,

aceste tabele au coloane și rânduri ce conțin diferite constrângeri și elemente de legătură între ele.

Dacă într-o tabelă voi avea stocate elemente de forma (id, firstName, lastName) și voi introduce mai târziu și elemente ce conțin emailul, NU voi putea face acest lucru deoarece coloana/atributul respectiv nu există.

Firebase (bazele de date NoSQL în general) nu au acest tip de restricții, oferindu-mi posibilitatea de a modifica tipul și forma unui entități fără prea mare efort (lucru destul de des întâlnit în procesul de dezvoltare al acestei aplicații). Practic, putem adăuga orice informație pe care dorim la nodul utilizatorului. Acest lucru este foarte bun pentru flexibilitate, dar nu ne ajută cu adevărat să asigurăm integritatea datelor.

Mai mult decât atât Firebase a simplificat procesul de preluare a datelor specifice din baza de date prin interogări. Interogările sunt create prin legarea uneia sau mai multor metode de filtrare.

Firebase are 4 funcții principale pentru ordonarea datelor:

- *orderByKey()*
- *orderByChild(„child”)*
- *orderByValue()*
- *orderByPriority()*

De reținut este faptul că vom primi date dintr-o interogare doar prin utilizarea metodelor *on()* sau *once()*, adică atașarea unui „listener” la entitatea dorită.

De asemenea, putem utiliza aceste funcții avansate de interogare pentru a restricționa și mai mult datele noastre:

- *startAt(‘value’)*
- *endAt(‘value’)*
- *equalTo(‘child_key’)*
- *limitToFirst(10)*
- *limitToLast(10)*

În Firebase, interogarea implică doi pași. Mai întâi, crearea unei referințe la cheia parentală și apoi utilizarea unei funcții de selectare/preluare.

```
const db = firebase.database();

const firebaseRef = db.child(`child`);

firebaseRef.orderByChild("user").equalTo("GeekyAnts").on("child_added",
  function(snapshot) {
    console.log(snapshot.key);
  }
);
```

Figura 23: Exemplu de interogare

În concluzie analizând diferențele și beneficiile aduse de ambele tipuri de baze de date am ales să folosesc una nerelațională deoarece se plia mai bine pe nevoie mele.

7. Concluziile finale

Consider ca alegerile făcute asupra temei, tehnologiilor și mediului de lucru au fost benefice și au adus câștiguri atât pe partea teoretică cât și pe cea practică.

Lucrând la acest proiect am dobândit cunoștințe multiple în ceea ce privește dezvoltarea mobile și dezvoltarea folosind o arhitectură serverless, logica fiind manipulată aproape în întregime pe partea de client. De asemenea, un alt scop a fost folosirea unor suite de tehnologii cât mai noi și de actualitate în dezvoltarea aplicației, iar React-Native și Firebase se încadrează în această categorie.

Recomand folosirea acestor tehnologii și nu e de mirare faptul că sunt foarte apreciate și folosite din ce în ce mai mult de companii de renume precum: Instagram, Facebook, SoundCloud, Walmart, Bloomberg.

Alegerea temei nu a fost una întâmplătoare deoarece m-am confruntat și am auzit de nenumărate ori din partea mai multor fizeri dorința existenței unei platforme speciale pentru suportul precesului de programări și promovarea saloanelor. Cred cu tărie că o aplicație de acest gen poate avea un impact destul de mare deoarece această piață nu este exploată îndeajuns, în special în țara noastră. Așadar, m-am gândit la o serie de funcționalități și îmbunătățiri asupra versiunii curente ce ar fi necesare lansării în producție:

- Posibilitatea „barberilor” de a-și configura propriul set de servicii;
- Integrarea unui sistem de plată cu cardul;
- Posibilitatea „consumerilor” de a filtra „barberii” în funcție de prețuri și gama de servicii oferite;
- Extinderea numărului de zile destinate rezervărilor;
- Opțional: generalizarea aplicației și folosirea logicii pentru rezervări și în alte domenii precum programării la doctorul de familie, la psiholog etc.

8. Bibliografie

1. <https://github.com/facebook/react-native>
2. <https://hackernoon.com/react-native-how-to-setup-your-first-app-a36c450a8a2f>
3. <https://docs.nativebase.io/Components.html>
4. <https://rnfirebase.io/>
5. <https://github.com/react-native-community/react-native-maps>
6. <https://www.pluralsight.com/guides/push-notifications-with-firebase-cloud-messaging>
7. <https://developers.facebook.com/>
8. <https://github.com/react-native-community/react-native-maps>
9. <https://firebase.google.com/docs/auth>
10. <https://console.firebase.google.com/project/barbersapp-25712/overview>