**QUICKPLATE – APLICAȚIE MOBILĂ PENTRU GESTIONAREA REZERĂRILOR ȘI COMENZILOR LA RESTAURANT**

LUCRARE DE LICENŢĂ

|  |  |  |
| --- | --- | --- |
|  | Absolvent: | **Ioan-Octavian Stanciu** |
|  |  |  |
|  | Coordonator științific: | **Dr. Ing. Lia-Anca Hangan** |

**2023**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | |  | |
| DECAN, |  | | DIRECTOR DEPARTAMENT, | |
| **Prof. dr. ing. Liviu MICLEA** |  | **Prof. dr. ing. Rodica POTOLEA** | |

Absolvent: **Ioan-Octavian STANCIU**

**QUICKPLATE – APLICAȚIE MOBILA PENTRU GESTIONAREA REZERVARILOR ȘI COMENZILOR LA RESTAURANT**

1. **Enunțul temei:** *Proiectul își propune proiectarea și dezvoltarea unei aplicații mobile care oferă utilizatorilor posibilitatea de a rezerva o masă la un restaurant dorit.*
2. **Conținutul lucrării:** *Introducere, Obiectivele proiectului, Studiu bilbiografic, Analiza și Fundamentare Teoretică, Proiectare de detaliu și Implementare, Testare și validare, Manual de instalare și utilizare, Concluzii, Bilbiografie, Anexa 1 - Glosar de termeni, Anexa 2 – Lista figurilor și tabelelor.*
3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare
4. **Consultanți**:
5. **Data emiterii temei:** 1 noiembrie 2022
6. **Data predării:** 8 iulie 2023

|  |  |
| --- | --- |
| Absolvent: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |
| Coordonator științific: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**Declarație pe propria răspundere privind**

**autenticitatea lucrării de licență**

Subsemnatul(a)**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**, legitimat(ă) cu \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ seria \_\_\_\_\_\_\_ nr. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
CNP \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, autorul lucrării \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ din cadrul Universității Tehnice din Cluj-Napoca, sesiunea \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ a anului universitar \_\_\_\_\_\_\_\_\_\_, declar pe propria răspundere că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării, și în bibliografie.

Declar că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

În cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

|  |  |  |
| --- | --- | --- |
| Data  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Nume, Prenume  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
|  |  | Semnătura |

Cuprins

[Capitolul 1. Introducere 1](#_Toc137147276)

[Capitolul 2. Obiectivele proiectului 3](#_Toc137147277)

[2.1. Obiectivul principal 3](#_Toc137147278)

[2.2. Obiective specifice 3](#_Toc137147279)

[Capitolul 3. Studiu bibliografic 5](#_Toc137147280)

[3.1. Avantajele iOS comparativ cu Android 5](#_Toc137147281)

[3.2. Aplicații mobile de tipul food-delivery 5](#_Toc137147282)

[3.2.1. Aplicația Zomato 6](#_Toc137147283)

[3.2.2. Aplicația UberEats 6](#_Toc137147284)

[3.2.3. Aplicația Foodpanda 7](#_Toc137147285)

[3.2.4. Asemănări și diferente intre Zomato, UberEats și Foodpanda 7](#_Toc137147286)

[3.3. Software de administrare al restaurantelor 8](#_Toc137147287)

[3.3.1. Toast POS 8](#_Toc137147288)

[3.3.2. Square POS 8](#_Toc137147289)

[3.3.3. Comparație intre aplicațiile Toast și Square 8](#_Toc137147290)

[Capitolul 4. Analiză și fundamentare Teoretică 10](#_Toc137147291)

[4.1. Analiza problemei 10](#_Toc137147292)

[4.2. Cerințe funcționale 11](#_Toc137147293)

[4.3. Cerințe non-funcționale 12](#_Toc137147294)

[4.4. Modelul de date al aplicației 13](#_Toc137147295)

[4.5. Starile unei comenzi 13](#_Toc137147296)

[4.6. Cazuri de utilizare 14](#_Toc137147297)

[4.6.1. Inregistrare 14](#_Toc137147298)

[4.6.2. Rezervarea unei mese 15](#_Toc137147299)

[4.6.3. Comanda și plata consumației 17](#_Toc137147300)

[4.6.4. Procesarea comenzilor 18](#_Toc137147301)

[4.7. Algoritmul de echilibrare a comenzilor per chelner 20](#_Toc137147302)

[Capitolul 5. Proiectare de detaliu și implementare 21](#_Toc137147303)

[5.1. Proiectarea și implementarea arhitecturii 21](#_Toc137147304)

[5.2. Proiectarea și implementarea bazei de date 22](#_Toc137147305)

[5.2.1. Entitatea Food 23](#_Toc137147306)

[5.2.2. Entitatea MyUser 23](#_Toc137147307)

[5.2.3. Entitatea Order 24](#_Toc137147308)

[5.2.4. Entitatea Restaurant 25](#_Toc137147309)

[5.2.5. Entitatea Table 25](#_Toc137147310)

[5.3. Structura proiectului 25](#_Toc137147311)

[5.3.1. Structura fisierelor 26](#_Toc137147312)

[5.3.2. Modulul Extras 28](#_Toc137147313)

[5.3.3. Modulul LoginStates 31](#_Toc137147314)

[5.3.4. Modulul Features 33](#_Toc137147315)

[Capitolul 6. Testare și validare 44](#_Toc137147316)

[Capitolul 7. Manual de instalare și utilizare 47](#_Toc137147317)

[7.1. Resursele necesare 47](#_Toc137147318)

[7.1.1. Resurse hardware 47](#_Toc137147319)

[7.1.2. Resurse software 47](#_Toc137147320)

[7.2. Manual de instalare 47](#_Toc137147321)

[7.3. Manual de utilizare 48](#_Toc137147322)

[7.3.1. Manual de utilizare pentru client 50](#_Toc137147323)

[7.3.2. Manual de utilizare pentru ospătar și bucătar 53](#_Toc137147324)

[Capitolul 8. Concluzii 53](#_Toc137147325)

[8.1. Contributiile propria 54](#_Toc137147326)

[8.2. Dezvoltari ulteriorare 55](#_Toc137147327)

[Bibliografie 56](#_Toc137147328)

[Anexa 1. Glosar de termeni 57](#_Toc137147329)

[Anexa 2. Lista figurilor și tabelelor 58](#_Toc137147330)

[Anexa 3. Exemple de cod 58](#_Toc137147331)

# Introducere

Acest proiect își propune dezvoltarea și implementarea unei aplicații mobile de gestionare a rezervărilor și comenzilor la restaurant pentru a ușura munca angajaților și de a le oferi clienților o posibilitate mai ușoară de a avea un loc asigurat într-un local în timpul orelor mai aglomerate, în special la final de săptămână.

Odată cu dezvoltarea rapidă urbană, anumite orașe au ajuns să fie din ce în ce mai aglomerate, iar această problemă reprezintă un impediment pentru mulți oameni. Devine foarte sâcâitor când traficul este mare și drumul de zece minute crește la a dura patruzeci, când suni la doctor și spune că nu mai are niciun loc liber pentru o programare în plus sau cozile la magazin sunt atât de lungi încât anumite persoane fac alegerea de a pleca.

Cu cât se aproprie mai mult sfârșitul săptămânii, cu atât oamenii încep să frecventeze restaurantele și localurile din ce în ce mai des, fapt datorat dorinței de a socializa și de a se relaxa alături de prieteni și familie în urma a cinci zile de mers la serviciu. Deși trăim în secolul tehnologiei, majoritatea oamenilor preferă să nu opteze pentru efectuarea unei rezervări la localul dorit ci să meargă fizic cu speranța de a prinde un loc liber la o masă pentru a se bucura de compania celorlalți.

Cu toate acestea, în ziua de astăzi oricine poate căuta cu ușurință pe internet un local la care să își petreacă seara, poate vizualiza meniul restaurantului pentru a vedea dacă felurile de mâncare sunt pe placul său, poate efectua un apel telefonic pentru o posibilă rezervare sau chiar să se uite pe hartă pentru a vedea distanța și timpul rutei ce trebuie parcursă până la destinație.

În plus, angajații de la restaurante și localuri sunt foarte schimbători deoarece de obicei nu stau mult la noul lor loc de muncă, motivul principal fiind salariul mic comparativ cu cel din țările străine. Un alt motiv ce se poate datora plecării acestora este volumul de muncă împărțit. Unii muncesc mai mult decât alții, iar în cazuri ca acesta pot intervenii conflicte între angajați și poate duce la un mediu toxic.

Cu toate acestea, nu există un singur sistem care să se ocupe de remedierea acestor probleme, soluția fiind achiziționarea anumitor produse care, dupa îmbinarea lor într-un singur sistem, să comunice cât mai eficient și simplu pentru a păstra o scalabilitate cât mai mare și să fie pe placul tuturor utilizatorilor ce vor beneficia de el pentru a-și ușura zilele libere de final de săptămână.

În secolul 21, smartphone-urile joacă un rol important în viața majorității populației. Cu ajutorul acestora se pot face poze, înregistra video-uri, căutări pe internet, adăugări de notițe, poți comunica cu persoanele apropiate doar prin interacțiunea cu ecranul. Nu a fost niciodată mai ușor pentru un om să afle informații noi datorită telefoanelor mobile ce oferă posibilitatea de căutare pe internet. Ce odată era dificil de aflat datorită timpului pierdut în care un individ trebuia să meargă la bilbiotecă și să caute prin carți informația dorită, acuma persoana respectivă are acces la mult mai multe surse de informare, toate cuprinse în același dispozitiv. Odată cu evoluția tehnologiei, se dorește să existe o aplicație pentru fiecare lucru cu scopul de a automatiza pentru a reduce timpul, stresul și a maximiza pe cât mai mult posibil eficiența și precizia, lucruri care sunt limitate de către ființa umană.

Acestea fiind spuse, aplicația QuickPlate vine cu rezolvarea acestor probleme prin oferirea unui echilibru a împărțirii volumului de muncă angajaților și oferirea posibilității clienților de a putea rezerva o masă, toate fiind încorporate în același sistem. Toate sunt ușor de realizat datorită unei interfețe grafice simple și intuitive ce oferă o experiență plăcută utilizatorului și convingătoare de a folosi aplicația și în viitorul apropriat.

QuickPlate oferă clienților șansa de a putea vizualiza restaurantele din proximitatea lor, de a rezerva o masă la un restaurant dorit și de a alege metoda de plată nefiind nevoie deloc de interacțiunea cu vreun angajat al localului respectiv. Acest lucru ajută la economisirea timpului pierdut de așteptare a unui ospătar, de preluare a comenzii și de cerere în achitarea notei de plată. Din punct de vedere a angajaților, aplicația ajută la fluidizarea stărilor comenzilor, acestea fiind trimise la bucătărie printr-o simplă atingere a ecranului, iar volumul de muncă este egal împărțit între toți chelnerii.

# Obiectivele proiectului

## Obiectivul principal

Obiectivul principal al acestui proiect este de a dezvolta o aplicație mobilă iOS care are rolul de a ajuta utilizatorii în a rezerva o masă la un restaurant dorit pentru a evita aglomerația și de a reduce riscul în a nu găsi o masă liberă. Aceștia pot observa pe harta integrată restaurantele din proximitatea lor pentru a aproxima distanța dintre ei și destinație, o listă cu toate restaurantele pentru a căuta restaurantul dorit mai repede și pentru a rezerva o masă, și o pagină de profil pentru a confirma sosirea și restaurantele favorite.

Dintotdeauna a existat problema aglomerației în localuri care impune o dificultate în rezervarea unei mese în timpul săptămânii. Deși trăim în „secolul tehnologiei” nu există nici până în ziua de astăzi o aplicație lipsită de probleme care să rezolve acest lucru. Acest proiect are rolul de a economisi timpul oamenilor în rezervarea unei mese la un restaurant și de a nu depinde de eroare umană care poate să intervină când se efectuează o rezervare prin intermediul unui apel telefonic.

## Obiective specifice

**Înregistrarea unor noi utilizatori:** Aplicația oferă posibilitatea noilor utilizatori de a-și putea face un cont nou pe aplicație cu scopul de a profita de tot ce oferă aceasta. Cu ajutorul acestei funcționalități, utilizatorul va fi capabil de a putea naviga prin aplicație, de a-și alege restaurantele preferate, de a rezerva o masă și multe altele. Bineînțeles, în funcție de rolul ales funcționalitățile pot să difere de la un utilizator la altul.

**Confirmarea adresei de email:** La fiecare creare de cont nou, utilizatorul va primi pe adresa de email introdusă un email de confirmare pentru a finaliza înregistrarea și de a se asigura că într-adevar el este cel care dorește acest lucru.

**Vizualizarea restaurantelor pe hartă:** Acest proiect dispune de o hartă integrată ce oferă posibilitatea utilizatorilor de a putea vizualiza restaurantele din proximitatea locației acestuia pentru a aproxima distanța ce trebuie parcursă în cazul în care la acel restaurant se dorește să se meargă.

**Lista restaurantele:** Utilizatorul va avea acces la o listă interactivă cu toate restaurantele din aplicație pentru a le vizualiza într-un mod organizat și de a putea să caute un restaurant dorit. În plus, fiecare restaurant are un buton interactiv cu care utilizatorul poate interacționa pentru a adăuga unul sau mai multe restaurante în lista de favorite.

**Meniul unui restaurant:** În fiecare pagină de detalii a fiecărui restaurant se află meniul acestuia pentru a lăsa utilizatorii să vizualizeze felurile de mâncare și băuturile. Pe lângă aceasta, din pagina respectivă se poate rezerva o masă la restaurantul respectiv prin alegerea unei zile și ore.

**Lista de restaurante favorite:** Această listă se află în pagina de profil pentru a avea acces mai ușor și rapid direct la restaurantele preferate adăugate anterior, acest lucru economisind timp utilizatorului în căutarea și găsirea restaurantului pe care l-a accesat în trecut și ar vrea din nou să reserve o masă la acesta.

**Lista rezervărilor:** Pe pagina de profil se pot găsi toate rezervările făcute de utilizator la restaurantele alese arătând numărul de persoane, ora și ziua fiecăreia.

**Comandă din aplicație:** Când utilizatorul o să confirme sosirea la restaurant, acesta o să fie redirecționat la o pagină specială în care îi este prezentat meniul și poate să comande folosind aplicația, reducând timpul de așteptare a unui chelner pentru aducerea unui meniu și de a lua comanda.

**Modalități de plată:** Dupa ce utilizatorul a terminat de comandat și dorește să plătească, i se oferă mai multe posiblități de plată precum, cash, card sau folosind aplicația. Acest lucru ajută la eficiența atât a angajaților restaurantului cât și a clientului.

# Studiu bibliografic

## Avantajele iOS comparativ cu Android

iOS, sistemul de operare a telefoanelor iPhone de la Apple [1], renumit pentru designul său elegant și interfața grafică intuitivă a adunat o mulțime de oameni curioși. Unul dintre principalele motive pentru a alege iOS este integrarea sa perfectă cu ecosistemul hardware produs de Apple. Cum Android este folosit de mai mulți producători de telefoane inteligente, interfața utilizator diferă de la unul la altul substanțial, utilizatorii hotărându-se mai greu în a lua decizia ce telefon să își achiziționeze.

Conform unui blog [2], datorită faptului că iOS beneficiază de faptul că este o platformă cu sursă inchisă, procesul de ecranizare este mai strict la iPhone și utilizatorii ce folosesc acest dispozitiv sunt limitați la aplicațiile de pe App Store Apple, lucru ce ajută la prevenirea descărcării de viruși. Pe de altă parte, utilizatorii de Android au acces la mai mulți distribuitori de aplicații, dar nu toate aplicațiile oferite de aceștia sunt verificate.

Din punct de vedere a limbajelor de programare, aplicațiile iOS se dezvoltă folosind Swift, un limbaj unic creat de Apple, cu scopul de a fi la fel de rapid ca C, ușor de citit și sigur de scris în comparație cu aplicațiile Android ce se dezvoltă folosind Java care are o sintaxă lungă și mai dificilă de înțeles, iar compilatorul nu este la fel de bine optimizat făcând construirea și rularea aplicațiilor să consume mai mult timp.

Când vine vorba de fragmentare, Apple are un mare avantaj întrucât este o variație mai mică între dispozitivele lor în comparație cu telefoanele ce folosesc Android. O experiență cât mai plăcută și consistentă îi este oferită utilizatorului de iOS datorită funcțiilor de bază pe acest tip de dispozitiv, inclusiv experiența culorilor și animațiilor. În contrast cu acesta, aplicațiile ce rulează Android trebuie dezvoltate și testate cu mai multă atenție din cauza atâtor tipuri de dispozitive ce au ecrane diferite, timpii de rulare diferă, rezoluția și multe altele.

Cu toate acestea, conform unei statistici [3] făcute în anul 2023, iOS deține din piața globală la finalul anului 2022 doar 27.58% pe când Android deține 71.72% ceea ce înseamnă că majoritatea utilizatorilor folosesc Android. Acest fapt se poate datora faptului că, pe lângă faptul că sunt mult mai mulți dezvoltatori ce folosesc acest sistem de operare, utilizatorilor li se oferă și mai multe alegeri când vine vorba de achiziționarea unui telefon nou.

## Aplicații mobile de tipul food-delivery

HORECA [4] este un acronim care se referă la industria ospitalității, formată din hoteluri, restaurante și cafenele. Această industrie este foarte importantă pentru economia globală și reprezintă un sector în continuă dezvoltare, cu o varietate de servicii și produse destinate satisfacerii nevoilor turiștilor și clienților. HORECA este responsabilă pentru crearea de locuri de muncă, pentru promovarea turismului și pentru generarea de venituri. De asemenea, această industrie este influențată de factori economici și de stilul de viață al oamenilor, ceea ce o face mereu în schimbare și adaptare.

Industria livrărilor de alimente a evoluat semnificativ în ultimii ani datorită popularității tot mai mari a smartphone-urilor și a dezvoltării aplicațiilor mobile. Până în prezent, livrarea produselor alimentare s-a limitat în mare măsura la comenzile telefonice sau online, necesitând adesea timp și efort suplimentar pentru a finaliza procesul de comandă și livrare. Odată cu apariția aplicațiilor de livrare a alimentelor, clienții pot comanda mâncare cu un click de pe un telefon mobil și le pot livra chiar la ușă, oferind o experiență rapidă și convenabilă. Aceste aplicații nu numai că au simplificat procesul de comandă și livrare, dar au deschis și calea pentru noi afaceri și servicii de livrare de alimente, oferind clienților acces la o gamă mai largă de opțiuni de luat masă. Astăzi, aplicațiile de livrare a alimentelor sunt folosite în întreaga lume și evoluează constant.

Potrivit unui raport realizat de Statista, în anul 2020, peste 1,5 miliarde de oameni din întreaga lume au comandat mâncare online, iar acest număr este într-o continuă creștere. În State Unite ale Americii, de exemplu, peste 60% dintre consumatori comandă mâncare online cel puțin o dată pe săptămână.

Conform unui blog[5], în anul 2022 industria de livrare a mâncării era estimată la o valoare de aproximativ de 760 de miliarde de dolari dintre care 300 erau doar din livrarea mâncării, nu a alimentelor. În același timp, numărul de utilizatori a aplicațiilor de livrare a mâncării a crescut enorm cu un număr de peste 3 miliarde.

Această creștere semnificativă a comenzilor online a schimbat modul în care restaurantele își gestionează afacerile. În loc să folosească meniuri tradiționale în format fizic, multe restaurante au început să treacă la meniuri virtuale, care sunt disponibile pe site-ul lor sau prin intermediul aplicațiilor de tipul livrării de mâncare. Acest lucru a permis oamenilor să parcurgă meniul în mod facil și să aleagă mâncarea preferată fără a fi nevoiți să se afle fizic la restaurant.

În plus, comanda online de mâncare a făcut posibilă și creșterea popularității serviciilor de livrare la domiciliu, cum ar fi Uber Eats [6], Deliveroo [7] sau Glovo [8]. Aceste servicii permit oamenilor să comande mâncare de la o varietate de restaurante diferite și să o primească la ușa lor într-un timp foarte scurt.

### Aplicația Zomato

Potrivit unei statistici[9] din anul 2022, Zomato a fost cea mai folosită aplicație de tipul food-delivery din întreaga lume cu un număr de descărcări de peste 54 de milioane și cu un număr de utilizatori activi de 32.1 milioane în fiecare lună.

Zomato este un lanț de restaurante indiene cu acoperire internațională și companie de tipul food-delivery fondata de Deepinder Goyal și Pankak Chaddah în anul 2008 [10]. În anul 2022, compania avea 3800 de angajați, iar veniturile erau aproximate la un total de 890 de milioane de dolari în anul 2023.

O funcționalitate ce scoate Zamato în evidență este abilitatea de a-i ajuta pe utilizatori să descopere restaurante noi. Aplicația se folosește de preferințele utilizatorului, căutările recente și istoricul comenzilor sau rezervărilor făcute. Având toate acestea la dispoziție, sistemul recomandă un nou loc unde utilizatorul ar putea lua cina sau de unde ar putea comanda.

### Aplicația UberEats

UberEats este o aplicație de livrare a mâncării lansată în anul 2014. Este folosită în peste 6000 de orașe și 45 de țări, iar valoarea acesteia era estimată la undeva în jurul sumei de 8,3 miliarde de dolari în anul 2021 având în continuare o creștere destul de rapidă. Este o extensie a aplicației Uber ce oferă utilizatorilor posibilitatea de a comanda un mijloc de transport prin intermediul telefoanelor sau tabletelor doar prin câteva atingeri ale ecranului. UberEats are și un abonament de 10 dolari per luna, numit Eats Pass, care renunță la taxa de livrare pentru toate comenzile și oferă utilizatorului un discount de 5% pentru orice comandă de peste 15 dolari.

Potrivit statisticii menționate și la subcapitolul anterior [11], aceasta a fost a doua cea mai folosită aplicație cu 46,8 milioane de descărcări. Majoritatea utilizatorilor folosesc UberEats pentru marea varietate de restaurante din aplicație, pentru interfața grafică placută și ușor de folosit și pentru posibilitatea de a putea lăsa recenzii. De asemnea, aplicația mai oferă promoții, reduceri sau chiar și oferte exclusive. Aceste lucruri au reușit să își mențină cât mai mulți din clienți și să îi atragă pe unii noi pentru a-și mări numărul de utilizatori.

Un motiv pentru care anumiti utilizatori prefera aceasta aplicație este livrarea rapida. UberEats foloseste infrastructura și reteaua logistica a aplicației Uber, care este cunoscuta pentru serviciile sale eficiente de transport. Acest lucru le permite să livreze alimente rapid, adesea în 30 pana la 60 de minute. Capacitatea de a primi rapid mancarea este un factor cheie în popularitatea UberEats, în special pentru persoanele cu program incarcat sau cei care prefera să manance acasa.

### Aplicația Foodpanda

Foodpanda este o aplicație de livrare a mâncării și a cumpărăturilor deținută de multinaționala germană Delivery Hero. A fost fondată în anul 2012, iar la un an după a fost lansata în Bangladesh și Romania, la acel moment compania având peste 20.000 de angajați. Aplicația a avut cel mai mare succes în Asia, fiind la momentul actual cea mai folosită aplicație în acea zonă, exceptând China. [12]

Această aplicație, în comparație cu celelalte, nu iese cu nimic în evidență întrucât are toate funcționalitățile de bază pe care ar trebui să le aibă o aplicație din această categorie: interfața ușor de înțeles și intuitivă, o gamă variată și diversă de restaurante, disponibilitate la nivel internațional, mai multe metode prin care poți să achiți comanda și multe altele.

Foodpanda a avut succes mare din cauza că fondatorii au ales foarte bine publicul țintă și anume cei din Asia unde, la momentul lansării, nu se știa de o asemenea aplicație. Pe lângă aceasta, în toată lumea în anul 2012 aplicațiile de acest tip nu aveau popularitatea pe care o aveau acum 5 ani.

### Asemănări și diferențe între Zomato, UberEats și Foodpanda

Zomato, UberEats și Foodpanda sunt toate foarte cunoscute ca aplicații de tipul food-delivery și deși oferă servicii similare, există mici diferențe între acestea:

* **Restaurantele partenere**: Zomato și Foodpanda au o varietate mai mare de restaurante în toate zonele în care sunt disponibile, pe când UberEats tinde să aibă mai multe parteneriate cu lanțurile de restaurante mai populare
* **Interfața și experiența utilizator**: Zomato oferă un sistem robust de descoperire și recenzie a restaurantelor, în timp ce UberEats și Foodpanda se concentrează mai mult pe experiența de livrare a alimentelor
* **Recomandări personalizate:** Zomato și Foodpanda oferă recomandări personalizate bazate pe preferințele utilizatorului, istoricul de comenzi și istoricul căutărilor pe când UberEats se focusează mai mult pe promovarea restaurantelor mai populare sau care sunt în trending
* **Funcționalități unice**: Zomato se remarcă prin funcțiile sale complete de descoperire a restaurantelor, inclusiv informații detaliate, recenzii, evaluări și posibilitatea de a rezerva mese la restaurantele partenere. UberEats și Foodpanda se concentrează în primul rând pe livrarea alimentelor, dar pot oferi funcții suplimentare, cum ar fi urmărirea în timp real a comenzilor.

## Software de administrare al restaurantelor

### Toast POS

Toast este un software de administrare a restaurantelor fondat în anul 2012. Cu un număr de aproximativ 3172 de angajati, de 2,4 miliarde de dolari de venituri și 62.000 [13] de restaurante ce folosesc acest software, face ca acesta să fie cel mai folosit în categoria sa. Software-ul este folosit doar pe device-urile ce folosesc sistemul de operare Android și oferă un sistem „all-în-one”.

Inițial a fost o aplicație de consum cu focusul pe plăți mobile, promoții și un aspect social care mai târziu toate acestea au fost integrate cu sistemul de POS al restaurantelor.

Acest software se remarcă prin mai multe funcționalități:

* **Comanda și checkout la masă**: Toast poate să accepte comenzi și plăți direct la masă prin intermediul ei. Acest lucru poate să fie rău din moment ce nu poate procesa plăți fără o conexiune la internet.
* **Rapoarte**: Programul permite fiecărui restaurant să genereze rapoarte referitoare la costuri, numărul de alimente din inventar și altele. Cu ajutorul acestora, cel care se ocupă de administrarea restaurantului poate să ia anumite decizii în funcție de rezultatele din rapoarte.
* **Actualizarea meniului**: Fiecare restaurant poate să își facă modificări la meniu, iar acest lucru se actualizează pentru toți utilizatorii. Poate să fie un lucru bun mai ales pentru acele restaurante ce își actualizează meniul zilnic sau în fiecare sezon.

Prețul de bază pentru a folosi Toast este de 110 de dolari per luna, dar pe masură ce se adaugă servicii, prețul poate să atingă suma de 165 de dolari, cel din urmă adăugând Sling scheduling, Payroll & Team Management, Toast Pay Card, Payout și altele.

### Square POS

Square POS este un sistem de puncte de vânzare care oferă companiilor posibilitatea să își eficientizeze operațiunile și să accepte plăți cu ușurință. Dezvoltat de Square Inc., o companie lider de tehnologie financiară, Square POS oferă o gamă largă de caracteristici și beneficii concepute pentru a spori eficiența. Companiile pot procesa plațile rapid și în siguranță. Fie ca acceptă carduri de credit și de debit, plăți fără contact sau portofele mobile, Square oferă soluții hardware și software ușor de utilizat pentru a facilita tranzacțiile fără întreruperi.

Dincolo de procesarea plăților, Square POS oferă capabilități robuste de gestionare a stocurilor. Companiile pot urmări cu ușurință nivelurile stocurilor, pot gestiona variațiile articolelor și pot primi alerte pentru stocul scăzut. Aceste caracteristici permit companiilor să își optimizeze inventarul, să evite epuizarea stocurilor și să se asigure că au la dispoziție produsele potrivite pentru a satisface cerințele clienților.

### Comparație între aplicațiile Toast și Square

Conform unui blog [14] publicat de Forbes în 2023, alegerea dintre Toast și Square ar trebui să fie făcută în funcție de nevoia restaurantului de a folosi POS sau nu. Toast e o alegere excelenta pentru restaurantele cu mai multe locații, drive-throughs, baruri pe când Square este o opțiune mai bună dacă se dorește un sistem POS pentru a ajuta cu verificarea, statul de plată, scanarea codului de bare și altele. Acest lucru se datorează deoarece Toast are funcționalități care se concentrează mai mult pe nevoile unui restaurant pe când Square își extinde serviciile pentru mai multe tipuri de afaceri.

În Tabelul 3.1 indica diferentele esentiale intre cele 2:

Tabel 3.1 - Diferente intre Toast și Square

|  |  |  |
| --- | --- | --- |
|  | **Toast** | **Square** |
| Taxe lunare | De la $0 pana la $165 | Versiune gratis sau $60 lunar |
| Taxa la plată | 2.99% plus $0.15 cu versiunea gratis  2.49% plus $0.15 cu un plan plătit | 2.6% plus $0.10 per tranzactie |
| Cost hardware | Inclus în plata lunară | Primul cititor de carduri este gratis după care fiecare costă $10  Terminalele costă începând cu suma de $299 |
| Relații clienti | Suport telefonic la orice oră în orice zi, blog | Chat live, blog |
| Procesarea plăților încorporată | Da | Da |

# Analiză și fundamentare Teoretică

## Analiza problemei

Ideea implementării unui astfel de proiect a plecat de la o problemă reală cu care se confruntă tot cei care doresc să petreacă timp cu prietenii în oraș, și anume aglomerația barurilor și restaurantelor, mai ales în weekend. Aceasta este o problemă foarte mare, mai ales în marile orașe românești precum Cluj-Napoca, București, Timișoara, Iași, Constanța.

După studierea pieței aplicațiilor mobile destinate aducerii funcționalitătilor de a rezolva această problemă s-a constatât că nu există nicio aplicație mobilă care să conțină toate caracteristicile necesare. Așadar, s-a împărțit problema în 2 categorii: cum poate fi de folos aplicația atât clientului care vrea să rezerve o masă cât și restaurantului pentru a ușura aglomerația?

În primul rând, trebuie să existe un public țintă, iar acesta este format de utilizatorii cu vârsta cuprinsă în intervalul 18-35 de ani. Mai multe statistici arată că aceștia sunt cei mai frecvenți utilizatori ai aplicațiilor de tipul food-delivery sau de rezervare a unei mese.

În al doilea rând, nu numai clienții trebuie să profite de aplicație, dar și restaurantele la care se pot rezerva mese pentru a simplifica procesul de comandă. În acest fel, chelnerul nu trebuie să meargă la masă pentru a duce meniul, să revină pentru a prelua comanda, apoi să aștepte până când comanda este preparată pentru a o servii, iar la final pentru a genera bonul.

În ambele situații prezentate anterior, fiecare utilizator trebuie să aibă un cont. În momentul actual, număr aplicațiilor care nu dispun de o astfel de funcționalitate este foarte mic. Bineînteles, dupa ce fiecare se autentifică fiecare, o să fie redirecționați către interfețe utilizator destinate rolurilor lor: una pentru client, iar cealaltă pentru chelner și bucătar.

Clientul trebuie mai întâi să vadă locația restaurantului pe hartă pentru a știi cât de lung este drumul până la destinație. De asemenea, trebuie să poată alege un restaurant după anumite caracteristici, iar cea mai bună este nota de rating. Acest lucru poate să îl determine pe utilizator să afle mai multe informații despre restaurant precum meniul, prețurile și să vadă anumite poze cu localul respectiv.

S-a luat în considerare și cazul în care pe hartă sunt afișate prea multe restaurante și devine foarte aglomerat și greu de urmărit. Ca rezolvare, s-a implementat o listă cu toate restaurantele în care utilizatorul are și opțiunea de a căuta după nume un restaurant știut. Aceasta îl ajută pe utilizator să vizualizeze într-un mod mai organizat și mai ușor de interacționat comparativ cu harta. Din această listă, clientul poate intra pe pagina de detalii a restaurantului pentru vizualizarea meniului cu scopul de a-l ajuta să aleagă unde sa mănânce. În cazul în care acesta este mulțumit, poate să rezerve o masă din pagina de detalii. În plus, clientul își poate adăuga restaurante la lista de favorite pentru o navigare extrem de simplificată pe viitor.

Dacă fiecare utilizator are cont atunci trebuie să aibă și o pagină de profil. Clientul are trei secțiuni:

* Secțiunea în care observă datele personale și poate să se deconecteze din aplicație
* Secțiunea în care poate observa toate restaurantele favorite pe care le-a adăugat pe parcursul folosirii aplicației
* Secțiunea în care își poate vizualiza rezervările

În momentul în care clientul își confirmă sosirea la restaurantul respectiv, acesta este redirecționat către un ecran special din care poate comanda, iar după cel puțin o comandă plasată poate cere nota de plată.

Pe de altă parte, chelnerul și bucătarul au doar ecranul de profil în care sunt două secțiuni:

* Secțiunea în care fiecare își poate vizualiza datele personale și se pot deconecta
* Secțiunea în care vin comenzile

După ce toate acestea au fost decise, s-a început un deSign In Figma, unealtă de design folosită pentru proiectare și prototipizare a interfețelor utilizator și a experiențelor acestora, bazată pe cloud care folosește instrumente de editare vectoriale, biblioteci de design, prototipuri interactive și capabilități de transfer al dezvoltatorilor. În Figura 4.5 se poate observa prototipizarea și proiectarea ecranelor dezvoltate în unealta Figma.

A screenshot of a phone

Description automatically generated with medium confidence

Figură 4.1 - Design prototipizat în Figma

## Cerințe funcționale

Fiind o aplicație ce se ocupă de gestionarea meselor și rezervărilor la restaurante, utilizatorii trebuie să aibă roluri diferite, atât ca și client cât și ca angajat. Datorită acestui fapt, a trebuit adăugată o metodă de securitate pentru conturile duplicate și pentru a verifica într-adevăr dacă utilizatorul respectiv dorește să își folosească propria adresă cu acest scop.

Ca un client să poată să își facă o rezervare, trebuie să își aleagă restaurantul dorit, așadar o listă interactivă prin care își poate îndeplini această dorință este perfectă. De asemenea, clientului îi trebuie oferită opțiunea de a alege metoda de plată prin care dorește să achite consumația. Din moment ce aplicația are ca scop fluidizarea interacțiunii client-ospătar, nu se poate ca nota de plată să fie cerută în aceeași modalitate clasică și anume de a chema chelnerul la masă.

Acestea fiind enumerate, următoarele cerințe funcționale reprezintă funcționalitătile aplicației QuickPlate:

* **Înregistrarea utilizatorului având rol de client sau de angajat**. Aplicația permite noilor utilizatori să își aleagă rolul la crearea contului. Acest lucru le oferă funcționalități diferite în funcție de alegerea facută. Cei cu rolul de client vor beneficia de o hartă pe care sunt puse restaurantele, o listă cu toate restaurantele și o pagină de profil pe când cei înregistrați ca și chelner sau bucătar vor avea doar pagina de profil.
* **Confirmarea adresei de email**. Când un utilizator nou dorește să își creeze un cont pe aplicație, acesta v-a trebui să introducă o adresă de email ce va fi folosită și pentru autentificare. Acesta v-a primi un mail de confirmare pe adresa de email folosită pentru a verifica dacă într-adevar el este cel care dorește crearea contului. Acest lucru este o metodă simplă de securitate ce inspiră încredere unui nou client.
* **Vizualizarea restaurantelor.** Când utilizatorul a ales rolul de client acest v-a putea să vizualizeze restaurantele din aplicație atât pe harta integrată cât și intr-o lista. Acest lucru îl ajută în a aproxima distanța de la locația sa până la restaurantul ales și de a căuta mult mai repede restaurantul în comparație cu harta.
* **Rezervarea unei mese.** Utilizatorul cu rolul de client poate să rezerve o masă direct din pagina de detalii a restaurantului. Acesta trebuie să aleagă masă în funcție de numărul de clienți doriți și ora și ziua în care dorește să meargă.
* **Metodele de plata.** Clientului i se oferă posibilitatea de a plăti prin mai multe moduri cum ar fi cash, cu cardul sau folosind direct aplicația. Acestă funcționalitate ajută la economisirea timpului de așteptare a chelnerului de a-l atenționa că se dorește plata comenzii.
* **Echilibrarea comenzilor per chelner.** Aplicația dispune de o funcționalitate care vine în ajutorul chelnerilor de a nu-i lăsa să accepte prea multe comenzi, lucru ce poate să îi încarce prea mult comparativ cu colegii lor. Aceștia au un număr limitat de comenzi ce pot fi acceptate și acest număr decrementează atunci când o comandă ori este anulată ori este livrată.

## Cerințe non-funcționale

În timp ce cerințele funcționale ale unui sistem definesc comportamentul și funcționalitatea sa, cerințele non-funcționale se concentrează pe calitățile sistemului. Aceste cerințe sunt la fel de importante ca și cele funcționale deoarece acestea pot afecta în mod semnificativ experiența utilizatorilor și pot determina gradul de satisfacție al acestora.

Următoarele cerințe prezentate sunt cele non-funcționale ale aplicației QuickPlate:

* **Securitatea**. Aceasta este pe departe cea mai importantă cerință non-funcțională din moment ce aplicația QuickPlate reține informații despre utilizatori. Din fericire, Firebase are mecanisme implementate ce se ocupă de acest lucru precum înregistrarea folosind adresa de email și regulile pentru accesul la baza de date.
* **Utilizabilitatea.** Acestă cerință este prima cu care utilizatorul se întâlnește deoarece reprezintă gradul de satisfacție la interacțiunea acestuia cu sistemul. Flow-ul trebuie să fie fluent și cursiv, butoanele să fie sugestive, fiecare schimbare de ecran să fie intuitivă, iconițele să sugereze ce reprezintă, toate acestea ajută la ridicarea nivelului de satisfacție a utilizatorului pentru a-l determina pe acesta la refolosirea aplicației pe viitor.
* **Extensibilitate.** Posibilitatea de a extinde funcționalitățile dezvoltate este un lucru crucial în dezvoltarea aplicațiilor. Ce o să fie implementat o să stea la baza funcționalităților mai complicate și o să ofere cel puțin minimul de implementare ce o să fie necesar pentru o dezvoltare mai complexă.
* **Portabilitatea.** Această cerință este validă datorită faptului că este destinată dispozitivelor ce ruleaza sistemul de operare iOS sau iPadOS, iar versiunea este cel puțin 16.0. De exemplu, dacă aplicația rulează pe un iPhone 14 cu versiunea iOS 16.3 atunci poate rula și pe un iPhone X cu versiunea iOS 16.0.

## Modelul de date al aplicației

Principalele entități care formează modelul de date sunt:

* **Utilizatorii**: Cea mai importantă entitate existentă în aplicație. Aceasta trebuie să rețină date esențiale despre utilizator precum numele, prenumele, parola, adresa de email, dar și alte informații precum restaurantele preferate, comenzile plasate și altele.
* **Restaurantele**: Entitate de bază în aplicație, cu ajutorul căreia utilizatorul își poate alege locul unde vrea să servească masă, să rezerve o masă sau să adauge în lista de restaurante preferate. De asemenea, cu ajutorul acesteia clientul poate vizualiza și aproxima distanța de la propria locație până la un restaurant găsit pe hartă. Cele mai importate date pe care trebuie să le rețină sunt numele, orele și zilele în care acesta este deschis, media evaluărilor efectuate de utilizatori și adresa locației.
* **Comenzile**: Entități folosite de către toate tipurile de utilizator în momentul în care un client plasează o comandă, în urma efectuării unei rezervări la o masă. Aceasta trebuie să conțină detalii despre prețul total, ce feluri de mâncare s-au comandat și cantitățile fiecăreia.
* **Felul de mâncare**: Entitate ce conține numele, ingredientele și prețul unui fel de mâncare cuprins într-un meniu, aceasta fiind necesară în special pentru cazul în care un client dorește să plaseze o comandă.
* **Meniul**: Este format din mai multe feluri de mâncare și este creat cu scopul de a corespunde unui restaurant pentru a-i permite utilizatorului să se hotărască dacă într-adevăr dorește o rezervare sau nu la acel restaurant.
* **Mesele**: Entitate folositoare clienților în a-i ajuta să rezerve o masă la un restaurant dorit. Conține informații precum numărul de persoane disponibile, dacă este sau nu deja rezervată și numele restaurantului de care aparțin.

## Stările unei comenzi

Cu scopul de a fluidiza plasarea și servirea unei comenzi a fost implementat ca o comandă să treacă prin diferite stări. Acest lucru este benefic atât clientului cât și angajaților.

* **În așteptare**: Aceasta este starea inițială a fiecărei comenzi plasate. O comandă se află în această stare atunci când utilizatorul plasează comanda și este trimisă la angajații restaurantului
* **În preparare**: Această stare apare în momentul în care chelnerul acceptă comanda și este trimisă bucătarului
* **Pregatită**: Această stare apare atunci când bucătarul confirmă terminarea preparării comenzii
* **Servită**: Comanda se află în ultima stare după ce aceasta a fost servită utilizatorului, confirmată de către chelner

De asemenea, când clientul plasează comanda, aceasta este salvată în baza de date, iar în cazul în care chelnerul o anuleaza în starea doi sau patru atunci va fi ștearsă din Firestore.

În Figura 4.6 se poate observa o diagramă de stări a unei comenzi:

A picture containing text, diagram, circle, screenshot

Description automatically generated

Figură 4.2 - Diagramă de stări a unei comenzi

## Cazuri de utilizare

În această secțiune vor fi prezentate patru cazuri de utilizare a aplicației QuickPlate fiecare având o descriere și o digramă de desfășurare, dar mai întâi se vor prezenta cele trei tipuri de utilizatori ce pot folosi această aplicație:

* **Clientul**. Acesta o să fie cel mai important și o să beneficieze de majoritatea funcționalităților ce le oferă aplicația. Acesta o să poată să vizualizeze harta cu restaurantele, să rezerve o masă, să comande și să plătească.
* **Ospătarul**. Acest tip de utilizator o să se ocupe de gestionarea comenzilor, prin confirmarea și servirea acestora. Din punct de vedere a numărului de funcționalități de care acesta beneficiază, nu este unul mare întrucât scopul lui nu necesită folosința majorității acestora.
* **Bucătarul**. Ca și rolul de ospătar, funcționalitățile nu sunt numeroase, iar gestionarea comenzilor este singura acțiune pe care acesta o poate face.

### Înregistrare

Această acțiune este efectuată de fiecare utilizator ce are sau nu un cont în aplicația QuickPlate. Daca are atunci trebuie doar să se autentifice, iar în caz contrar să se înregistreze.

**Actor:** Clientul

**Descriere:** Clientul dorește să profite de funcționalitățile aplicației QuickPlate

**Precondiții:**

* Utilizatorul are o conexiune bună la internet
* Utilizatorul are aplicația instalată

**Postcondiții:**

* Utilizatorul o să fie redirecționat la pagina principală a aplicației

**Scenariul de succes:**

* Utilizatorul deschide aplicația și este redirecționat la pagina de Sign In
* Utilizatorul nu are cont, dar dorește să își facă fiind redirecționat la pagina de Sign Up
* Utilizatorul completează toate câmpurile cu datele dorite
* Nu există nicio dată greșită sau deja folosită de alt utilizator
* Utilizatorul este redirecționat înapoi la pagina de Sign In
* Utilizatorul își introduce credențialele
* Credențialele sunt valide
* Utilizatorul este redirecționat la pagina principală

**Scenarii alternative:**

* Utilizatorul are cont, își introduce credențialele, dar sunt incorecte și îi este afișat un mesaj de eroare
* Utilizatorul nu are cont și trebuie să își creeze unul, iar cel puțin o dată introdusă este invalidă
* Utilizatorul nu are o conexiune bună la internet, iar verificările datelor nu se pot efectua

A picture containing text, screenshot, font, design

Description automatically generated

Figură 4.3 - Diagramă de flow pentru inregistrare

### Rezervarea unei mese

Această acțiune este destinată doar utilizatorului cu rol de client întrucât ospătarul și bucătarul se ocupă doar de gestionarea comenzilor.

**Actor:** Clientul

**Descriere:** Clientul dorește să rezerve o masă la restaurantul dorit

**Precondiții:**

* Utilizatorul are o conexiune bună la internet
* Utilizatorul are aplicația instalată
* Utilizatorul are cont cu rolul de „Client”
* Utilizatorul este autentificat cu acel cont în aplicație

**Postcondiții:**

* Rezervarea apare în lista de rezervări de pe pagina de profil al utilizatorului

**Scenariul de succes:**

* Utilizatorul selectează pagina care conține lista de restaurante
* Utilizatorul selectează restaurantul dorit
* Utilizatorul vizualizează meniul restaurantului
* Utilizatorului ii place meniul
* Utilizatorul dorește să își rezerve o masă
* Masa este liberă la ora și ziua aleasă de către utilizator
* Utilizatorul poate observa rezervarea facută pe pagina de profil în secțiunea de „Booked Tables”

**Scenarii alternative:**

* Utilizatorul nu are o conexiune bună la internet și nu i se încarcă lista cu restaurantele
* Utilizatorului nu ii place meniul restaurantului selectat rezultând în căutarea altui restaurant
* Ora și ziua rezervării nu este liberă ceea ce înseamnă că utilizatorul trebuie să aleagă alta zi, oră sau să schimbe atât ora cât și ziua

A screenshot of a phone

Description automatically generated with medium confidence

Figură 4.4 - Diagramă de flow pentru rezervarea unei mese

### Comanda și plata consumației

Funcționalitate destinată utilizatorilor cu rol de client pentru a plasa o comandă în urma efectuarii unei rezervări la un restaurant dorit, iar la final li se oferă posibilitatea de a alege opțiunea prin care aceștia vor să achite nota de plată.

**Actor:** Clientul

**Descriere:** Clientul dorește să comande din aplicație

**Precondiții:**

* Utilizatorul are o conexiune bună la internet
* Utilizatorul are aplicația instalată și este autentificat
* Utilizatorul este pe pagina de profil în secțiunea de „Booked tables”
* Utilizatorul are o rezervare făcută la restaurantul respectiv
* Utilizatorul confirmă ca a ajuns

**Postcondiții:**

* Utilizatorul a reușit cu succes să selecteze metoda de plată dorită

**Scenariul de succes:**

* Utilizatorul este pe pagina de profil
* Utilizatorul verifică dacă rezervarea facută este în secțiunea de „Booked tables”
* Utilizatorul confirmă sosirea și este redirecționat la ecranul dedicat comenzii
* Utilizatorul adaugă produsele dorite pentru comandă
* Utilizatorul trimite comanda angajaților din restaurant
* Utilizatorul nu mai vrea să comande nimic altceva și dorește să achite nota de plată
* Utilizatorul a dat cel puțin o comandă la restaurant
* Utilizatorul alege metoda prin care dorește să achite nota de plată

**Scenarii alternative:**

* Utilizatorul nu are o conexiune bună la internet și nu i se încarcă lista cu rezervările dorite
* Utilizatorul nu are nicio rezervare la restaurantul respectiv făcându-l să își rezerve o masă
* Utilizatorul nu adaugă niciun produs înainte de a trimite comanda către restaurant

Utilizatorul nu a trimis nicio comandă înainte de a cere nota de plată

A picture containing screenshot, text, graphic design, design

Description automatically generated

Figură 4.5 - Diagramă de flow pentru comanda și plata consumației

### Procesarea comenzilor

Această funcționalitate este destinată numai utilizatorilor cu rol de ospătar și bucătar întrucât un client nu trebuie să știe ce se întâmplă cu o comandă dupa plasarea acesteia.

**Actor:** Angajații restaurantului

**Descriere:** Clientul trimite una sau mai multe comenzi la restaurant și angajații se ocupă de procesarea ei

**Precondiții:**

* Angajații au aplicația instalată
* Angajații trebuie să aibă un cont cu rolul de „Waiter” pentru chelner sau „Cook” pentru bucătar
* Angajații trebuie să aibă o conexiune bună la internet

**Postcondiții:**

* Chelnerii livreaza comanda la masă și confirmă livrarea din aplicație

**Scenariul de succes:**

* Angajatul este înregistrat în aplicație și este redirecționat la pagina de profil
* Chelnerul primește o nouă comandă
* Nu intervine niciun motiv pentru care comanda ar trebui să fie anulată
* Chelnerul mai poate accepta comenzi
* Comanda s-a terminat de preparat, lucru confirmat de către bucătar
* Nu intervine niciun motiv pentru care comanda ar trebui să fie anulată
* Comanda este livrată și confirmată din aplicație

**Scenarii alternative:**

* Angajatul are cont cu rol de „Client”
* Intervine un motiv pentru care comanda ar trebui imediat anulată
* Chelnerul nu mai poate accepta deocamdată comenzi
* Nici alt chelner nu mai poate accepta comenzi

Intervine un motiv de anulare a comenzii dupa ce bucătarul a preparat-o

A screenshot of a computer screen

Description automatically generated with low confidence

Figură 4.6 - Diagramă de flow pentru pentru comenzilor

## Algoritmul de echilibrare a comenzilor per chelner

Aplicația urmărește, de asemenea, să faciliteze procesul de comandă și servire pentru restaurante, astfel încât ar trebui luat în considerare și potențialul volum mare de muncă al angajaților. Un singur chelner nu poate prelua comenzi. Pentru a evita acest caz, a fost implementat un algoritm care reține numărul de comenzi pe care le-a luat fiecare ospătar, iar daca acest număr este depășit, acesta nu mai poate prelua alte comenzi până când comenzile deja acceptate nu sunt servite.

Există o variabilă în memoria telefonului care stochează numărul de comenzi. Crește atunci când o comandă este acceptată și scade când este livrată. Acest lucru permite fiecărui ospătar să primească același număr de comenzi menținând în același timp echilibrul.

Acest lucru este posibil prin dezactivarea unor elemente ce țin de interfața utilizator pentru a-i permite ospătarului doar să confirme servirea unei comenzi. Referitor la subcapitolul precedent, când se atinge limita maximă de acceptare a comenzilor, ospătarului îi este permis să treacă comanda din starea trei în patru și îi este interzis trecerea comenzii din starea unu în starea doi.

De asemenea, la cererea manager-ului restaurantului, această limită de acceptare se poate schimba făcând numărul mai mare sau mai mic, depinzând de preferințe. În figura 4.7 se poate observa o diagramă de flow pentru o vizualizare mai simplă a acestui algoritm.

A screenshot of a cell phone

Description automatically generated with medium confidence

Figură 4.7 - Diagramă de flow pentru gestionarea comenzilor per chelner

# Proiectare de detaliu și implementare

## Proiectarea și implementarea arhitecturii

Aplicația a fost dezvoltata cu scopul de a fi folosita de un public cât mai larg, asadar s-a optat pentru o arhitectura de tipul client-server datorita dezvoltarii usoare a scalabilitatii, a securitatii și a resurselor impartite intre mai multi clienti. Acestea fiind spuse, aplicația iOS propriu-zisa reprezintă partea de client al proiectului, iar Firebase cea de server datorita serviciilor pe care le oferă precum autentificarea și autorizarea utilizatorilor, punerea la dispozitie unei baze de date NoSQL cu actualizari în timp real, reguli de restrictionare asupra bazei de date și multe altele.

Pentru structurarea codului, s-a incercat pe cât mai mult posibil de profitarea folosirii framework-ului SwiftUI cu scopul de a usura utilizarea modelului arhitectural Model-View-View-Model sau MVVM și de a evita folosirea modelului Model-View-Controller sau MVC.

Motivul principal pentru care s-a dorit acest lucru este incarcarea cu foarte multa logica pentru funcționalități a clasei cu rolul de Controller putand să se ajunga intr-un punct foarte greu de dezvoltare sau mentinere a clasei. Asadar, MVVM separa implementarea interfetei de restul aplicației astfel:

* View: reprezintă interfata utilizator. Pot fi un simplu buton, text, o grupare dintre acestea sau o grupare din mai multe View-uri.
* Model: reprezintă structura datelor ce se doresc a fi folosite în aplicație și corespund cu entitățile din baza de date.
* View-Model: clasa care se ocupa de actualizarea interfetei și modelarea clasei de Model, aceasta putand să restrictioneze datele afisate și să arate strict ce este necesar interfetei prezentate la acel moment

Avantajele alegerii de a folosi MVVM în loc de MVC mai pot fi posibilitatea de structurare mai bună a codului, disponibilitatea de a putea refolosi anumite componente de interfate datorita separarii logicii de date fata logica de actualizare a interfetei utilizator, mai multe clasa View putand să aibă acelasi View-Model și oferirea flexibilitatii și extinderii claselor usurand introducerea modificarilor sau adaugarea de noi funcții la aplicație.

Datorita faptului ca SwiftUI are implementata comunicare de tip Publish-Subscribe prin intermediul protocolului ObservableObject, actualizarea interfetei utilizator în urma unei modificari facute în clasa View-Model corespunzatoare ei este una simpla intrucat dezvoltatorul nu mai trebuie să mai adauge schimbari în clasa de View.

De asemenea, din moment ce actualizarile din baza de date a Firebase-ului sunt asincrone, framework-ul Combine este de mare ajutor intrucat ajuta la sincronizarea datelor și actualizarea corecta a acestora în urma interactiunii utilizatorului cu aplicația.

În Figura 5.1 se poate observa arhitectura aplicației QuickPlate.

A picture containing text, screenshot, diagram, design

Description automatically generated

Figură 5.1 - Diagramă aplicației QuickPlate

## Proiectarea și implementarea bazei de date

Baza de date folosita de aplicația este Firestore, de tipul NoSQL și oferita de Firebase. S-a optat pentru aceasta optiune datorita actualizarilor în timp real a datelor în momentul efectuarii unei modificari asupra unui model de date. Interacțiunea cu aceasta este una foarte usoara intrucat dezvoltatorul tot ce a trebuit să faca a fost instalarea unor pachete speciale și adaugarea unui fisier denumit GoogleService-Info.plist ce contine configurari, acestea toate fiind necesare pentru conectarea la aplicația Firebase.

Din moment ce baza de date este una non-relationala, stocarea datelor se face sub forma unor documente de tip JSON. În acest tip de document, valorile sunt stocate sub forma unui dictionar pentru o navigare și scalabilitate mult mai usoara. Firestore are colectii, echivalentul tabelelor a bazelor de date relationale, ce retin o ierarhie de documente. Cu toate acestea, orice document poate să contina o colectie astfel creandu-se un arbore de relatii. Datorita acestui fapt, operatiile CRUD sunt mult mai rapide și eficiente comparativ cu cazul în care pentru aplicația QuickPlate s-ar fi folosit o baza de date relationala.

Aplicația are cinci entități principale care sunt folosite și la crearea unelor DTO-uri: Food, MyUser, Order, Restaurant, Table. Mai departe vor fi prezentate fiecare individual cu scopul crearii lor și proprietile fiecareia. Fiecare entitate are generata automat un UUID() de catre Firebase.

În Figura 5.3 se pot observa relatiile dintre cele cinci entități enumerate anterior.

A screenshot of a computer screen

Description automatically generated with low confidence

Figură 5.2 - Relatiile entităților bazei de date

### Entitatea Food

Aceasta entitate a fost creata cu scopul de a retine date despre un anumit fel de mâncare. Este folositoare pentru a-i arata clientului ce feluri de mâncare are un restaurant inainte de a își rezerva o masă, dar și de putea plasa o comanda când acesta confirma sosirea la local. Este de mentionat ca un meniu este reprezentat sub forma unui tablou unidimensional format din id-urile felurilor de mâncare ce il formează.

Aceasta entitate nu ar referinte la alte entități.

* id: UID ce reprezintă id-ul felului de mâncare
* foodName: String ce reprezintă numele felului de mâncare
* ingredients: Tablou unidimensional de string-uri cu numele ingredientelor ce sunt în felul de mâncare
* price: Număr real cu doua zecimale ce reprezintă pretul felului de mâncare
* stock: Număr intreg ce reprezintă numărul din inventar al acelui fel de mâncare

### Entitatea MyUser

Aceasta este entitatea esentiala aplicației intrucat reprezintă datele fiecarui utilizator al aplicației. Folosind datele acestuia se pot efectua rezervari, comenzi, autentificare, autorizare, deconectare, adaugarea de restaurante preferate, toate operatiile ce tin de utilizatorul unei aplicații. Fara aceasta, aplicația nu ar putea să funcționeze.

Pe de alta parte, aceasta entitate poate să aibă referinta de tipul one-to-many catre entitatea de Restaurant datorita proprietatii de favouriteRestaurants, acelasi tip de referinta catre entitatea Table în cazul în care a facut rezervari sau one-to-one catre entitatea Restaurant în cazul în care utilizatorul este angajat acelui restaurant. Daca acesta este client, nu are restaurante preferate și nu are rezervari atunci el nu va avea nicio referinta catre o alta enitate.

* id: UID ce reprezintă id-ul utilizatorului
* username: String ce reprezintă numele de utilizator pe care să il foloseasca în aplicație
* firstName: String ce reprezintă prenumele utilizatorului
* lastName: String ce reprezintă numele utilizatorului
* role: String ce reprezintă rolul utilizatorului și poate să fie de trei feluri: Client, Waiter sau Cook, primul fiind rolul de client, iar urmatoarele doua de ospătar respectiv bucătar
* restaurantWorking: String ce reprezintă numele restaurantului la care utilizatorul lucreaza în cazul în care este ospătar sau bucătar. Daca este client atunci acesta o să fie gol, iar în caz contrar trebuie obligatoriu completat
* email: String ce reprezintă adresa de email a utilizatorului
* password: String ce reprezintă parola utilizatorului
* favouriteRestaurants: Tablou unidimensional de string-uri cu numele restaurantelor preferate
* bookedTables: Tablou unidimensional de string-uri ce contine id-urile meselor rezervate de catre utilizator

### Entitatea Order

Aceasta entitate este folosita doar în cazul în care un client dorește să plaseze o comanda. În momentul crearii acesteia, sigur o să aibă o referinta catre entitatea MyUser pentru a retine id-ul utilizatorului care a plasat comanda, catre entitatea Table pentru ca ospătarii să stie la ce masă trebuie livrata comanda și nu în ultimul rand catre entitatea Food pentru a stii ce contine comanda.

* id: UID ce reprezintă id-ul comenzii
* resName: String care retine numele restaurantului la care s-a plasat comanda
* tableNr: Număr intreg care reprezintă numărul mesei de la care s-a plasat comanda. Folositoare pentru afisarea în interfata utilizator pentru angajati
* foodIds: Tablou unidimensional de string-uri cu id-urile felurilor de mâncare
* foodQuantity: Tablou unidimensional de numere intregi în care se retine cantitatea fiecarui fel de mâncare
* totalCost: Număr real care retine costul total al comenzii
* userId: String care retine id-ul utilizatorului care a plasat comanda
* tableId: String care retine id-ul mesei la care s-a plasat comanda. Folositoare pentru operatiile cu baza de date
* orderState: String ce retine starea în care se afla comanda

### Entitatea Restaurant

Aceasta entitate este cea mai des intalnita pe parcursul experientei utilizatorului în folosirea aplicației. Aceasta este folosita pe pagina de harta, lista restaurantelor pana și pe pagina de profil este nevoie de ea. Cu ajutorul ei utilizatorul poate observa pe harta restaurantele, în lista, pagina de detalii, iar în cazul rolului de angajat locul unde lucreaza.

Aceasta entitate o să aibă referinta de tipul one-to-many catre entitatea Table pentru a stii ce mese tin de restaurantul respectiv și o referinta de one-to-many catre entitatea Food pentru a crea meniul cu care va interactiona utilizatorul.

* id: UID ce reprezintă id-ul restaurantului
* address: String cu adresa restaurantului
* closeHour: String cu ora inchiderii
* locațion: GeoPoint cu coordonatele geografice ale restaurantului
* name: String cu numele restaurantului
* openDays: Tablou unidimensional de string-uri cu zilele saptamanii când restaurantul este funcțional
* openHour: String cu ora deschiderii
* imageURL: String cu un URL catre o imagine a restaurantului
* rating: Număr real ce reprezintă recenzia restaurantului cuprinsa în intervalul [0, 5]
* tables: Tablou unidimensional de string-uri cu id-urile meselor ce tin de acel restaurant
* menu: Tablou unidimensional de string-uri cu id-urile felurilor de mâncare pentru ce alcatuiesc meniul restaurantului

### Entitatea Table

Cea de a cincea și ultima entitate folosita în aplicația QuickPlate este Table care are rolul de a ajuta clinetul să rezerve o masă la un restaurant dorit, de a putea observa detalii despre rezervarea facuta și să ajute ospătarii cu informatiile necesare pentru servirea comenzii. Aceasta entitate, ca și Food, nu are referinte catre nicio alta entitate ci doar altele au referinta catre ea.

* id: String ce reprezintă id-ul mesei
* booked: Valoare booleana care indica daca masă este sau nu rezervata. True pentru cazul în care este rezervata și false când nu este.
* hourBooked: String care reprezintă ora de la care este facuta rezervarea. Nu are o valoare retinuta daca nu este rezervata.
* nrPersons: Valoare intreaga care indica numărul maxim de persoane care pot sta la masă
* tableNumber: Valoare intreaga pentru a arata numărul mesei. Folositoare pentru opatarii pentru a stii ce masă este rezervata
* day: String pentru a indica ziua în care este rezervata. Daca nu este, atunci nu este nicio valoare.
* userId: String care retine id-ul utilizatorului care rezerva masă. Daca nu este rezervata, atunci ea o să aibă o valoare goala.

## Structura proiectului

Pentru o intelegere cât mai usoara a claselor principale folosite, figura 5.3 prezinta o Diagramă cu cele mai importante View-uri, unele afisandu-se doar în funcție de rolul utilizatorului.

A screenshot of a computer screen

Description automatically generated with low confidence

Figură 5.3 - Diagramă claselor principale

Fiecare clasa prezenta în figura anterioara v-a fi dezvoltata în detaliu în urmatoarele capitole.

### Structura fisierelor

În acest subcapitol voi prezenta structura fisierelor și directoarelor proiectului aplicației QuickPlate. Pe parcursul dezvoltarii proiectului s-a urmarit denumirea cât mai intuitiva a acestora pentru o usoara reutilizare a lor și o intelegere cât mai simplista a legaturilor dintre ele. Directoarele Features, LoginStates și Extras vor fi explicate mai în detaliu în urmatoarele subcapitole.

Fisierul Lozalizable.strings stocheaza mai multe siruri de caractere sub forma de perechi de tipul cheie-valoare pentru fiecare limba acceptata. Aplicația QuickPlate suporta la momentul actual doar limba romana și engleza. Acest fisier ar trebui creat de dezvoltator și configurat cu toate limbile ce se doresc a fi suportate. LocalizedStringKey() este utila pentru folosirea acestui fisier deoarece acest tip reprezintă o punte intre cod și textul afisat. Este de obicei folosit cu componenta Text ca în exemplul urmator: Text(LocalizedStringKey(’’welcome’’)) afiseaza valoarea corespunzatoare cheii „welcome” în funcție de limba setata pe dispozitivul utilizatorului.

În figura 5.4 se poate observa o parte din acest fisier.

A screenshot of a computer

Description automatically generated

Figură 5.4 - Fisierul Lozalizable.strings

Fisierele cu extensiile .plist, Info.plist respective GoogleService-Info.plist, sunt fisiere necesare de configurari, primul fiind pentru aplicația QuickPlate, iar cel de al doilea pentru conectarea la aplicația Firebase. Primul fisier este general de Xcode, IDE-ul în care s-a dezvoltat aplicația, pe când cel de al doilea a fost descarcat când s-a creat proiectul de Firebase. Metadatele din aceste doua fisiere sunt retinute sub aceeasi forma ca și în fisierul Localizable.strings și anume perechi de tipul cheie-valoare.

Assets.xcassets este generat automat de catre Xcode și este un pachet special care actioneaza ca o locație centralizata pentru gestionarea fisierelor de imagine, pictograme și culori. Este folosit pentru a organiza și stoca resursele vizuale ale aplicației. În figura 5.5 se poate observa cum arata acest pachet.

A screenshot of a computer

Description automatically generated

Figură 5.5 - Pachetul Assets.xcassets

Fisierul QuickPlateApp.swift este cel care face posibila rularea aplicației. Ea detine echivalentul metodei de main() din alte limbaje de programare. De asemenea, în aceasta mai este o clasa denumita AppDelegate care se suprascrie metoda didFinishLaunchingWithOptions: care se apeleaza în momentul în care aplicația s-a terminat de pornit. S-a optat pentru aceasta alegere pentru a putea incepe configurarea aplicației Firebase, în metoda mentionata anterior fiind apelata Firebase.configure().

RootView.swift detine rolul ca fiind cea mai importanta clasa a aplicației intrucat în funcție de regulile de autentificare aceasta afiseaza interfata utilizator destinate rolului utilizatorului. La inceput afiseaza o scurta animatie cu un ecran de introducere dupa care se verifica urmatoarele:

* Daca utilizatorul nu este autentificat atunci el va fi redirecționat catre ecranul destinate funcționalitatii de autentificare
* Daca utilizatorul este autentificat și are rol de client atunci o să se afiseze ecranul cu funcționalitatile destinate clientului
* Daca utilizatorul este autentificat și are rol de ospătar atunci o să fie redirecționat catre pagina destinate acestui tip de utilizator
* Daca utilizatorul este autentificat și are rol de bucătar atunci o să ii fie afisat ecranul destinat acestui rol

Aceasta verificare se face cu ajutorul unui View-Model ce va fi prezentat în subcapitolul 5.3.3.

### Modulul Extras

În modulul Extras sunt stocate clasele de ajutor. Structura acestui proiect a fost aleasă în asa fel incat clasele mai putin folosite să fie separate de cele de care dezvoltatorul are nevoie mai mult.

Singurul fisier care nu este cuprins intr-un modul separat este FSCollNames.swift care contine un simplu Enum, ce mosteneste din tipul de data String, cu numele colectiilor din baza de date destinate entităților de date. Acestea pot fi users, restaurants, tables, foods sau orders. Fiecarui caz ii este atribut un sir de caractere ce corespunde cu numele colectiei corespunzatoare din baza de date Firestore. De exemplu, pentru entitatea *Table* este cazul *tables* ce are atribuit un sir de caractere denumit *Tables*. Valorile reprezintă numele entității la plural pentru a nu exista o posibila confuzie intre numele entității și numele colectiei.

Submodulul FirestoreOp continte cinci clase singleton (sunt instantiate o singura data și aceeasi instanta este folosita în intreaga aplicație) pentru a ajuta la interacțiunea cu Firestore. Acestea au fost create cu scopul de a se executa operatii CRUD asupra datelor stocate în Firestore și de a separa aceasta logica de cod fata de restul aplicației. Toate sunt denumite sugestive, fiecare avand un prefix “FS” ce vine de la Firestore, urmata de numele colectiei asupra careia se executa operatiile și un sufix “Coll” care reprezintă o abreviere a cuvantului Collection.

Cu toate acestea, a fost creata o clasa extra, FirebaseEmailAuth, de tipul singleton, dar are alt scop și anume de a se ocupa de partea de autentificare, inregistrare și deconectare a aplicației. Aceasta are implementate funcțiile esentiale pentru a efectua metodele de Sign In/Login, Sign Up/Register și Sign Out/Logout. Pentru primele doua verifica și în plus daca adresa de email a fost confirmata, respectiv daca este deja folosita de alt cont. În figura 5.6 se poate observa un enum denumit StartupError ce mosteneste tipul de data Error și are cinci cazuri de erori:

* signInError: Apare atunci când utilizatorul nu a reusit să se autentifice datorita credențialelor introduse gresit sau folosirea unei adrese de email ce a fost utilizata la crearea unui alt cont
* signUpError: Eroare ce apare la inregsitrare datorita completarii gresite a unor campuri sau evitand completarea unor. De asemenea, reintroducerea unui nume de utilizator deja în folosinta poate să faca ca aceasta eroare să apara
* anonymousUser: Eroare ce apare în momentul în care un utilizator incearca să se autentifice sau să se inregistreze, iar Firebase returneaza un user ce este anonim. În aplicația QuickPlate este tratât ca un caz de eroare din moment ce utilizatorul trebuie să fie obligatoriu autentificat pentru a putea folosi aplicația
* emailExists: Eroare ce apare în momentul în care utilizatorul incearca să se autentifice sau să se inregistreze, iar adresa de email este deja în folosinta de catre alt cont

A screen shot of a computer program

Description automatically generated with low confidence

Figură 5.6 - Funcția de login a clasei FirebaseEmailAuth

De asemenea, toate metodele din aceasta clasa returneaza un closure pentru a putea trata cazurile de succes, respectiv de esec prin completion-urile returnate. În caz de succes se returneaza UID-ul utilizatorului, iar în caz contrar un caz de eroare definit în StartupError.

În submodulul Models sunt toate entitățile definite la subcapitolul 5.2. MyUser, Restaurant, Food, Table și Order. Toate corespund cu exacitate descrierilor facute la subcapitolul mentionat și reprezintă continutul datelor ce sunt stoacate în intreaga aplicație.

În modelul Restaurant exista în plus doua structuri ce reprezintă DTO-uri folosite pe pagina de Sign Up și când se dorește afisarea mai sumara a informatiilor despre un restaurant. Pentru cazul de register se foloseste RestaurantSignUpDTO, iar pentru cel de al doilea RestaurantCardDTO. Acestea permit transferul mai usor a datelor și mai rapid din punct de vedere computational. De asemenea, entitatea Restaurant mai are pe langa proprietatile ce corespund cu cele din baza de date, doua proprietati de tipul computed pentru o usoara prelucrare a proprietatilor de openHour și closeHour.

Entitatea Food, ca și cea de Restaurant, are o proprietate de tipul computed pentru a concatena intr-un singur sir de caractere toate denumirile ingredentelor din felul de mâncare respectiv.

În interiorul fisierului Order.swift este definit un enum ce reprezintă starile unei comenzi mentionate și în capitolele anterioare. Acesta extinde din tipul String al limbajului Swift pentru a li se asigna automat valori ce corespund cu numele acestora (cazul .pending o să aibă atribuit un sir de caractere denumit „pending”, cazul .preparing o să fie echivalent cu „preparing” și asa mai departe).

Ultimul submodul, Extensions, contine clase ce reprezintă extensii a mai multor tipuri de date din limbajul Swift.

* Color+CustomColor.swift: este creata o extensie la structura predefinita Color pentru a adauga culorile personalizate din pachetul Assets.xcassets. Fiecare culoare este definita ca o constanta statica pentru a putea fi accesata de oriunde din aplicație în orice moment.
* LocalizedStringKey+String.swift: sunt doua extensii, una pentru LocalizedStringKey și cealalta pentru String pentru a usura localizarea sirurilor de caractere pe tot parcursul aplicației.
* CLLocaționCoordinate2D+Extension.swift: O extensie la structura CLLocaționCoordinate2D pentru a putea fi encodata, decodata și comparata egalitatea cu o structura de tipul GeoPoint. Pentru aceasta ultima funcționalitate a trebuit crearea unei funcții denumita isEqualTo ce primeste ca parametru o variabila de tipul GeoPoint și returneaza o valoarea booleana în cazul în care longitudinile sunt egale intre ele, respectiv latitudinile.
* GeoPoint+Extensions.swift: O extindere la tipul de date GeoPoint pentru a crea o funcție ce returneaza o structura de tipul CLLocaționCoordinate2D avand aceleasi coordonate cu parametrul de tipul GeoPoint.

În figura 5.7 se poate observa structura modulului prezentat în acest subcapitol.

A screenshot of a computer

Description automatically generated with medium confidence

Figură 5.7 - Structura modulului Extras

### Modulul LoginStates

În acest modul se afla un singur fisier, AuthManager, care contine cea mai importanta clasa a acestui proiect deoarece se ocupa de logica de autentificare și este de asemenea View-Model-ul mentionat la 5.3.1.

În primul rand, este declarat un enum, LoginStateEnum, ce are patru cazuri:

* notSignedIn: caz în care utilizatorul nu este autentificat și ii este prezentat ecranul de Sign In
* clientSignedIn: caz în care utilizatorul este deja autentificat și are rolul de client
* waiterSignedIn: caz în care utilizatorul este deja autentificat și are rolul de ospătar
* cookSignedIn: caz în care utilizatorul este deja autentificat și are rolul de bucătar

Acest enum este folosit sub forma unei proprietati a clasei AuthManager și în funcție de valoarea pe care o stocheaza se decide ce ecran trebuie afisat utilizatorului. Metoda de checkLoginUserDefaultsExist() se apeleaza la initializarea clasei și verifica daca în memoria telefonului este deja retinuta valoarea proprietatii respective. Daca nu exista atunci se adauga cu cazul de baza notSignedIn, iar în caz contrar se apeleaza metoda updateWith(state: LoginStateEnum) care salveaza noua valoare în UserDefaults.

AuthManager este instantiat în clasa RootView, care acesta are în interiorul ei o instructiune switch pentru verificarea valorii proprietatii nextScreen, definita în AuthManager, în care se face schimbarea interfetei utilizator în funcție aceasta. Cazurile sunt:

* .notSignedIn: Se afiseaza view-ul SignInView(). Se poate ajunge dupa instalarea aplicației sau dupa ce utilizatorul se deconecteaza din cont.
* .clientSignedIn: Se afiseaza view-ul QPTabView(). Se poate ajunge doar dupa ce utilizatorul se autentifica și are rolul de client.
* .waiterSignedIn: Se afiseaza view-ul WaiterView(). Se poate ajunge doar dupa ce utilizatorul se autentifica și are rolul de chelner.
* .cookSignedIn: Se afiseaza view-ul CookView(). Se poate ajunge doar dupa ce utilizatorul se autentifica și are rolul de bucătar.

Indiferent de caz, clasa AuthManager o să fie disponibila pentru toate View-urile folosind modificatorul .environmentObject pentru ca fiecare clasa să aibă acces la logica de schimbare a ecranului (folositoare la Sign In și Sign Out). Pentru a avea acces la AuthManager intr-unul dintre View-urile SignInView, QPTabView, WaiterView și CookView, trebuie declarata o proprietate de tipul AuthManager și adnotata cu @EnvironmentObject fara a fi initializata.

Explicat mai sumar, în momentul în care se apeleaza metoda de updateWith(state:) de oriunde din cele patru View-uri enumerate mai sus, valoarea proprietatii nextScreen se schimba, se salveaza în UserDefaults, se reevalueaza valoarea prin intermediul instructiunii switch și se schimba ecranul afisat.

În figura 5.8, în clasa din stanga este View-ul RootView în care se afla instructiunea switch, în mijloc este fisierul AuthManager ce contine clasa AuthManager și enum-ul LoginStateEnum, iar în dreapta, la linia 13, se poate observa declararea proprietatii authManager de tipul AuthManager pentru a avea acces la funcționalitatile ei.

A screenshot of a computer program

Description automatically generated with low confidence

Figură 5.8 - Utilizarea clasei AuthManager

### Modulul Features

Modulul Features este cel cu cele mai multe subdirectoare și clase ce sunt folosite în intreagul proiect. Se poate spune ca acesta este centrul aplicației QuickPlate intrucat majoraitatea funcționalitatilor sunt cuprinse în acest loc.

Primul modul este SpashScreen cu un singur fisier denumit SpashScreenView care este View-ul ce se afiseaza la inceputul aplicației de fiecare data cu o animatie de durata doua secunde prin intermediul folosirii modificatorului onAppear() ce face posibila detectia aparitiei View-ul în ierarhie. Acest View este folosit mereu la initializarea clasei RootView indiferent de starea autentificarii utilizatorului. În aceasta clasa se poate observa folosirea unei culori din fisierul Assets.xcassets folosind structura Color avand ca parametru numele culorii.

Al doilea modul intitulat SignIn contine logica de autentificare și contine doua fisiere SignInView și SignInViewModel, iar pe langa acestea un alt modul „Custom View + Modifiers” care la randul lui este alcatuit din: SecureInputView, un View ce poate fi folosit pentru introducerea textului care nu trebuie securizat (cum ar fi prenumele, numele, adresa de email etc) și parolei (afiseaza cercuri în locul literelor), acest lucru fiind posibil prin folosirea unei instructiuni if pentru a verifica valoarea booleana stocata în variabila isSecured, și SignInTextFieldModifier care este un modificator ce se poate aplica asupra oricarui View pentru a-l stiliza. Un modificator este o funcție care modifica și personalizeaza aspectul sau comportamentul unui View. Acestea sunt folosite în View-ul SignInView cu logica să separata în SignInViewModel.

În SignInViewModel, în funcția signIn, este implementata logia de autentificare a unui utilizator fiind apelata metoda doLogin() din clasa FirebaseEmailAuth(), iar ca rezultatul returnat de catre completion este tratât intr-o instructiune switch. Pentru cazul de esec, se trimite din nou un completion(.failure) indicand ca aceasta acțiune a esuat, iar pentru cazul de succes se seteaza în memoria telefonului id-ul utilizatorului, se aduce din baza de date utilizatorul ce are acel identificator, iar cu ajutorul unei instructiuni if se verifica ce caz din LoginStateEnum se salveaza:

* Utilizatorul este client: trebuie salvat doar starea autentificarii și anume clientSignedIn
* Utilizatorul este ospătar: se salveaza numele restaurantului unde lucreaza și cazul waiterSignedIn
* Utilizatorul este bucătar: se salveaza numele restaurantului la care este angajat și cazul cookSignedIn

În figura 5.9 este prezentata Diagramă de clase a acestui modul.

A picture containing diagram, text, line, plan

Description automatically generated

Figură 5.9 - Diagramă de clase a modulului SignIn

Cel de al treila modul contine funcționalitatea crearii unui cont nou de catre orice utilizator. Are doar doua fisiere, SignUpView și SignUpViewModel, primul reprezentand interfata grafica, iar cel din urma logica codului.

Acest View-Model are multe proprietati @Published pentru a se actualiza în timp real când utilizatorul interactioneaza cu interfata grafica. Cateva exemple sunt username, firstName, lastname, password. În momentul în care utilizatorul apasa pe butonul de creare cont, trebuie mai intai verificat daca s-au completat toate campurile obligatorii, username, firstName, lastName, email și password, logica fiind implemntat în metoda allFieldsAreCompleted() ce returneaza o valoare booleana. Daca pe proprietatea de *role* nu se selecteaza nimic, atunci se va considera ca utilizatorul este client, iar campul de selectie a restaurantului unde este angajat o să fie dezactivat.

Posibilele erori ce pot să apara sunt urmatoarele:

* Utilizatorul nu a completat toate campurile
* Utilizatorul a introdus o adresa de email ce este deja asociata unui alt cont
* Utilizatorul a introdus doua parole diferite
* Utilizatorul nu a respectat formatul parole care este să contina minim sase caractere

În figura 5.10 este prezentata Diagramă de clase a acestui modul.

A picture containing diagram, text, screenshot, line

Description automatically generated

Figură 5.10 - Diagramă de clase a modulului SignUp

Urmatoarele trei directoare, Client, Waiter și Cook, sunt destinate rolurilor de client, ospătar, respectiv bucătar, ultimele doua fiind foarte asemanatoare singura diferenta fiind filtrarea datelor ce se afiseaza. Prima oara o să fie prezentate directoarele ce apartin modulului Client dupa care directoarele Waiter și Cook vor fi detaliate în acelasi timp evidentiand micile diferente intre ele.

* 1. Modulul TabView

Detine clasa QPTabView care cuprinde grupul celor trei ecrane destinate utilizatorului cu rolul de client, format din Harta, Restaurante și Profil. Acest View nu are niciun View-Model asignat intrucat nu se ocupa de nicio logica a vreunei funcționalități al aplicației și se afla în zona de jos al ecranului dispozitivului. Pe metoda de init(), apelata la instantierea clasei, se seteaza culorile de baza ca fiind portocaliu pentru text și poze, iar alb pentru ce se vede în spate.

* 1. Modulul Map

Contine logica și interfata utilizator pentru partea de harta din QPTabView. Cu ajutorul framework-ului MapKit dezvoltat de Apple, integrarea unei harti în aplicație a fost una foarte simpla. În clasa MapView se foloseste un View din acest framework, Map, care ii trebuie trimisi anumiti parametrii pentru a avea acces la locația utilizatorului în timp real și pentru a arata sau nu locația acestuia pe harta. Harta contine în partea de sus un buton interactiv care atunci când este apasat centreaza harta pe coordonatele utilizatorului. În momentul în care se apasa acel buton, se apeleaza metoda de startUpdatingLocațion() din locaționManager pentru a incepe actualizarea locației utilizatorului. Atunci când se detecteaza o noua pozitie, se apeleaza automat metoda didUpdateLocațion din clasa LocaționManager care verifica cu ajutorul unei instructiuni if daca ultima locație (coordonatele fiind retinute în proprietatea lastLocațion) difera de ultima valoare din tabloul unidimensional locațions (parametru al acestei metode ca ultima valoarea coordonatele ultimei locații actualizate). Daca difera, atunci se actualizeaza regiunea cu noile coordonate detectate, se actualizeaza proprietatea lastLocațion și se opreste actualizarea continua a locației utilizatorului prin apelul metode stopUpdatingLocațion().

Mai contine și clasa LocaționManager ce se conformează la protocolul CLLocaționManagerDelegate, iar acesta se ocupa de logica mentionata în paragraful anterior. Delegate este un model de design de baza în aplicațiile iOS intrucat multe funcționalități predefinite trebuie să se conformeze la asemenea protocoale. Acesta are un delegat, un obiect, care contine o metoda sau un set de metode pe care obiectul delegat le poate implementa, și obiectul delegat este cel care implementeaza metodele definite de delegat. Obiectul delegat este inregistrat sau asignat catre delegat, permitandu-i să primeasca și să gestioneze evenimente sau să furnizeze anumite funcționalități. În cazul acesta, atât delegatul cât și obiectul delegat este LocaționManager deoarece este folosit sub tipul unui View-Model, iar toate schimbarile din acesta o să fie publicat în View-ul MapView.

Pe langa acestea, pe harta sunt afisate restaurantele aplicației, fiecare pin fiind interactiv pentru ca, atunci când se apasa pe unul dintre ele, în partea de jos este afisat un View mai mic cu informatii sumare despre acel restaurant. Pinii de pe harta sunt trimisi ca parametru sub forma unui tablou unidimensional ce stocheaza date de tipul MyAnnotationItem la View-ul Map. Acest ultim tip a fost creat avand ca proprietati un id, coordonatele restaurantului și recenzia lui, acestea fiind stocate în annotationItems în MapViewModel. În figura este prezentata Diagramă de clase a modulului Map.

A picture containing diagram, line, screenshot, plan

Description automatically generated

Figură 5.11 - Diagramă de clase a modulului Map

* 1. Modulul Restaurants

Acest modul contine logica și interfata utilizator pentru partea de cautare a restaurantelor și de rezervarea unei mese. Când clientul selecteaza mijlocul celor trei optiuni din partea de jos a ecranului, apare o lista cu toate restaurantele din aplicație, cu o bara de cautare deasupra acesteia, cu mentiunea ca un restaurant se poate căuta doar dupa numele lui.

Fiecare element din lista afiseaza aceleasi informatii cu cele de pe harta, acestea avand în plus o stea interactiva care permite clientului să adauge anumite restaurante la lista de preferate de pe pagina de profil. View-ul ce contine lista se afla în RestaurantsListView avand ca View-Model RestaurantsListViewModel, iar View-ul ce afiseaza informatiile sumare despre restaurant atât pe harta cât și în acest ecran este declarat în RestaurantCardView.

Modificatorul searchable este cel responsabil de adaugare acelei bare de cautare. Ea primeste ca parametrii un sir de caractere care își schimba valoarea pe masura ce utilizatorul tasteaza, plasamentul acesteia pe ecran și un sir de caractere ce se afiseaza inainte de interacțiunea cu clientul pentru a-și sugera rolul. Actualizarea listei în momentul cautarii o face filtrarea restaurantelor din tablou unidimensional denumit ,,restaurants” din View-Model. Momentul în care se initializeaza aceasta clasa și se populeaza tabloul, se face o copie a acestuia pentru a fi folosit la resetarea listei, insemnand ca tabloul ,,restaurants” va avea inapoi toate restaurantele de la momentul initializarii. În figura 5.12 este prezentata Diagramă de clase a modulului Restaurants.

A picture containing text, screenshot, font, design

Description automatically generated

Figură 5.12 - Diagramă de clase a modulului Restaurants

* 1. Modulul RestaurantDetails

Posibilitatea de a naviga intre lista de restaurante și pagina de detalii a acestuia o face View-ul NavigationView care, cu ajutorul NavigationLink, seteaza destinatia fiecarui element din lista ca fiind pagina de detalii corespunzatoare restaurantului. NavigationLink primeste ca parametru destinatia, care în acest caz este RestaurantDetailsView cu parametrul restaurant, iar ca View al elementului din lista este RestaurantCardView care primeste DTO-ul restaurantului respectiv.

Cu toate acestea, în RestaurantDetailsView se defineste și View-ul pentru un fel de mâncare intitulat FoodCard care primeste ca parametru felul de mâncare de tipul Food, iar ca și constructie este una simpla: Este definita o simpla stiva verticala ce contine trei siruri de caractere de afisate:

* Numele mancarii care se afiseaza prin intermediul prorietatii foodName
* Lista de ingrediente ce este returnata de valoarea variabilei ingredientsString
* Pretul ce este returnat și afisat doar cu doua zecimale, stocat în proprietatea price

Toate acestea trei sunt accesat din variabila food al View-ul FoodCard.

RestaurantDetailsViewModel are o singura metoda și anume de a prelua din baza de date toate felurile de mâncare ce apartin de acel restaurant. Cum proprietatea denumita *menu* al entității Restaurant contine toate id-urile ce apartin de meniul sau, în metoda fetchRestaurantMenu se verifica daca id-urile felurilor de mâncare din baza de date se regasesc în *menu.* În caz afirmativ, se adauga în lista foods al View-Model-ului prin care o să se itereze în RestaurantDetailsView pentru a se crea FoodCard-uri și pentru a se afisa meniul.

De asemenea, o metoda importanta în RestaurantDetailsView o reprezintă isOpened() ce returneaza un View depinzand daca restaurantul este sau nu deschis. Mai intai se retin în constante ora de deschidere a restaurantului, de inchidere și ora curenta (când se initializeaza View-ul). Cu ajutorul unei instructiuni if se verifica daca ora curenta este în intervalul programului restaurantului caz în care se afiseaza ora deschidere și inchiderii cu o culoare verde, iar în caz contrar se afiseaza un mesaj de culoare rosie ce sugereaza inchiderea acestuia. În figura 5.13 este prezentata Diagramă de clase a modulului RestaurantsDetails.

A diagram of a restaurant

Description automatically generated with low confidence

Figură 5.13 - Diagramă de clase a modulului RestaurantDetails

* 1. Modulul TablesList

View-ul ce afiseaza lista meselor disponibile pentru rezervare este definit în fisierul TablesView avand View-Model-ul TablesViewViewModel. Daca nu este nicio masă disponibila se va afisa un mesaj în locul listei care sugereaza acest lucru, iar în caz contrar utilizatorul va putea selecta ziua, ora și masă pe care dorește să o rezerve. Fiecare masă afiseaza și numărul maxim de persoane ce pot fi asezati. Daca un fel de mâncare nu este în stock (este entitate Food și valoarea variabilei stock este egala cu 0) atunci ea nu va aparea în meniul restaurantului.

Logica funcționalitatii este una foarte simpla intrucat View-Model-ul are doar trei metode:

* fetchAllTables(forRestaurant: Restaurant): metoda care returneaza toate mesele ce apartin restaurantului trimis ca și parametru apeland metoda de fetchAllTables a clasei singleton FSTableColl și populand tabloul unidimensional tables.
* bookingTable(tableId:String, hour: String, day: String, userId: String): metoda ce actualizeaza o entitate Table cu identificatorul egal cu parametrul tableId, prin schimbarea valorii poprietatilor hour, day, userId și booked cu valorile parametrilor, respectiv true acestea intamplandu-se în metoda tableBooked al clasei FSTableColl. De asemenea, se apeleaza metoda saveBookedTable(withId: String) al clasei FSUserColl pentru actualizarea tabloului unidimensional al utilizatorului ce a rezervat masă. Intr-un final se apeleaza metoda updateTables() din aceeasi clasa
* updateTables(): metoda ce se apeleaza dupa ce o masă a fost rezervata pentru a afisa doar mesele ce sunt disponibile pentru rezervare

În figura urmatoare, 5.14, este prezentata Diagramă de clase a modulului TablesList.

A picture containing diagram, text, line, number

Description automatically generated

Figură 5.14 - Diagramă de clase a modulului TablesList

* 1. Modulul UserProfile

Profilul clientului este creat în acest direcor impreuna cu logica de confirmare a rezervarii. View-ul respectiv este impartit în doua secțiuni: secțiunea de sus în care sunt afisate date despre utilizator precum atât numele de familie cât și cel de utilizator, adresa de email și un buton pentru deconectare, și secțiunea din a doua parte a ecranului ce contine View-ul pentru mesele rezervate și pentru restaurantele preferate.

Lista de restaurante nu este interactiva, ca urmare se face doar afisarea lor. Nici lista de rezervari nu este interactiva, dar fiecare element din aceasta lista are doua butoane interactive: unul pentru a sterge rezervarea, moment în care aceasta nu o să mai apara, și unul pentru a confirma sosirea la restaurant, clientul fiind redirecționat catre ecranul din care poate plasa comanda.

BookedTableView și BookedTableViewModel sunt View-ul, respectiv View-Model-ul pentru fiecare element din lista de mese rezervate, iar UserProfileView și UserProfileViewModel sunt View-ul respectiv View-Model-ul pentru profilul utilizatorului.

BookedTabeViewModel are o singura metoda numita fetchRestaurantName pentru a stoca numele restaurantului în proprietatea restaurantName ce are identificatorul mesei tableId, trimisa ca și parametru. În BookedTableView este o singura proprietate și anume table de tipul Table. Cu ajutorul acesteia se afiseaza informatiile necesare a unei mese.

UserProfileViewModel contine doua metode ce se apeleaza la momentul afisarii ecranului de profil:

* fetchLoggedUser(): metoda care aduce din baza de date entitatea utilizatorului autentificat și il retine în variabila *user* de tipul MyUser. Aceasta metoda apeleaza la final și updateBookedTables(user: MyUser) pentru a aduce din baza de date toate mesele rezervate de acest și se stocheaza în variabila bookedTables
* fetchFavRests(): metoda care aduce din baza de date toti identificatorii restaurantelor favorite a utilizatorului, iar pentru fiecare id se aduce restaurantul corespunzator și se adauga în tabloul unidimensional favouriteRestaurants

Pe langa acestea, mai sunt doua metode ce se apeleaza la apasarea a doua butoane:

* Se apasa butonul de deconectare: se apeleaza metode de signOut() care aceasta apeleaza metoda doLogout al clasei FirebaseEmailAuth pentru deconectarea utilizatorului autentificat
* Se apasa butonul de anulare a rezervari: se apeleaza metoda de cancelBookingForTableWith(tableId: String) care apeleaza metoda deleteBookedTableWith(tableId: String) din clasa FSUserColl care face o stergere a id-ulului din bookedTables a utilizatorului

În figura 5.15 este prezentata Diagramă de clase a modulului UserProfile.

A picture containing diagram, text, line, plan

Description automatically generated

Figură 5.15 - Diagramă de clase a modulului UserProfile

* 1. Modulul OrderView

În acest modul, se formează View-ul pentru o comanda, OrderCardView, cu logica separata în OrderCardViewModel. Acesta este folosit pe ecranele destinate bucătarilor și ospătarilor pentru a afisa statusul comenzii. View-Model-ul are doar doua funcții:

* fetchFoodsFor(order: Order): returneaza felurile de mâncare ce au fost plasate în comanda trimisa ca și parametru, iterandu-se prin tabloul unidimensional al parametrului foodIds, iar pentru fiecare id se adauga în tabloul foods al clasei felul de mâncare ce are acelasi identificator. La final, se apeleaza metoda de orderFoodsAfterIds(order) deoarece datele din baza de date vin ordonate dupa UID-ul generat de Firestore. Astfel, ordinea din foods și order.foodIds ar fi diferita, iar acest lucru nu este de dorita intrucat order.foodQuantity corespunde id-urile felurile de mâncare la plasarea comenzii.
* orderFoodsAfterIds(order: Order): Se face o sortare a tabloului unidimensional foods pentru a corespunde ordinii id-urilor din tabloul foodIds, proprietate a parametrului order.

OrderCardView este folosit doar în ecranul destinat utilizatorilor cu rol de ospătar, respectiv bucătar, în care se afiseaza numărul mesei de la care s-a plasat comanda, fiecare fel de mâncare cu cantitatea comandata și pretul ei, iar la final costul total al comenzii. În figura 5.16 este prezentata Diagramă de clase a acestui modul.

A picture containing text, screenshot, font, design

Description automatically generated

Figură 5.16 - Diagramă de clase a modulului OrderView

* 1. Modulul ClientOrder

Ultimul modul cu funcționalități specifice utilizatorului cu rolul de client este ClientOrder ce contine ecranul de plasare a unei comenzii și a notei de plata. ClientOrderView este fisierul ce contine elementele de interfata utilizator dedicate acestei funcționalități și este alcatuita din doua liste: una în jumatâtea de sus a ecranului în care utilizatorul poate să vada meniul și să își adauge cantitatea de mâncare dorita, și în cealalata jumatâte observa în timp real care este cosul. Daca acesta cere nota fara a plasa cel putin o comanda se va afisa un mesaj de eroare, metoda checkForOrdersStatus() din ClientOrderViewModel tratand acest caz. În caz contrar, i se va afisa ecranul cu o lista a tuturor comenzilor plasate, BillView, (care trebuie să fie în ultima stare, servite) și totalul de plata. Daca vreun fel de mâncare nu este în stock De asemnea, își poate alege metoda prin care dorește să achite nota de plata. În figura 5.17 se poate observa Diagramă de clase a acestui modul.

A screenshot of a computer

Description automatically generated with low confidence

Figură 5.17 - Diagramă de clase a modulului ClientOrder

Directoarele ce se ocupa de funcționalitatile dedicate utilizatorilor care sunt ospătari sau bucătari sunt Waiter respectiv Cook. Fiecare are un View, WaiterView și CookView, cu un View-Model asignat, WaiterViewViewModel și CookViewModel. Interfata este foarte asemanatoare celei de profil a clientului, diferentele fiind:

* În secțiunea de informatii a utilizatorului se mai afiseaza un camp în plus cu numele restaurantului unde acestia lucreaza
* În secțiunea de jos se afiseaza o singura lista cu comenzile, singura diferenta intre acestea este ca pentru ospătar se afieaza când comanda este în prima stare sau a treia pe când bucătarului i se afiseaza comenzile ce sunt în cea de a doua stare

Logica de echilibare a comenzilor per chelner se face în metodele acceptOrder() și canAcceptOrder() din WaiterViewViewModel. Fiecare comanda afisata ospătarului are un buton verde ce trimite comanda ori la bucătar ori pentru a confirma servirea ei. Butonul devine dezactivat în momentul în care chelnerul a acceptat prea multe comenzi. Metoda acceptOrder primeste doi parametrii: id-ul comenzii și starea acesteia. Se preia valoarea din UserDefaults a numarului de comenzi acceptate pana la acel moment și se executa trei instructiuni if: prima verifica daca starea este prima, caz în care numărul de comenzi acceptate creste, a doua verifica daca comanda comanda este în a treia stare ceea ce inseamna ca urmeaza să fie servita și ca urmare numărul de comenzi acceptate scade, iar cea de a treia verifica daca numărul de comenzi acceptate depaseste numărul maxim acceptabil. Daca da, atunci valoarea maxima o să fie salvata inapoi în UserDefaults. Dupa se apeleaza metoda canAcceptOrders() ce actualizeaza valoarea variabilei publicate canTakeOrder cu true sau false depinzand de numărul de comenzi acceptate din UserDefaults. La schimbarea acesteia, butonul de trimitere la bucătarie a comenzii este dezactivat. Aceasta metoda se apeleaza și pe metoda init(), aceasta la randul ei fiind apelata la initializarea View-Model-ului, pentru a seta din start valoarea variabilei canTakeOrder.

În figura 5.18 se pot observa trei simulatoare de iPhone, fiecare fiind autentificat, de la stanga la dreapta, cu un cont de bucătar, unul de ospătar, și unul de client. Clientul poate plasa comenzi, bucătarul poate să modifice starea comenzilor din preparing în ready prin apasarea butonului „Comanda este gata”, iar ospătarul nu mai poate să accepte ultimele doua comenzi din lista lui intrucat a acceptat-o pe care se afla în modul de preparare (cea de pe ecranul bucătarului) și pe cea care este gata de servire (confirmarea trebuie să o faca prin apasarea butonului „Serveste comanda”), lucru evidentiat de textul gri al butoanelor din partea stanga. Pe langa acestea, se pot observa primele trei stari a unei comenzi:

* pending: ospătarul primeste comenzile și textul butonului din partea stanga este „Trimite la bucătarie” (în cazul acesta sunt gri pentru ca ospătarul a depasit limia de acceptare a comenzilor)
* preparing: comanda ajunge la bucătar și acesta poate să o treaca în urmatoarea stare apasand butonul verde
* ready: comanda ajunge de la bucătar inapoi la ospătar, iar textul butonului verde devine „Serveste comanda”
* sent: comanda nu mai este vizibila pe niciun ecran intrucat ea a fost servita și nu trebuie afisata niciunui utilizator

A screenshot of a phone

Description automatically generated with low confidence

Figură 5.18 - Starile unei comenzi

# Testare și validare

În acest capitol se vor prezenta metodele de testare a aplicației QuickPlate. Testarea va fi prin parcurgerea unor actiuni, iar fiecare ca avea un rezultat asteptat. Funcționalitate vor fi inregistrarea de cont, rezervarea unei mese, algoritmul de echilibrare a comenzilor și cererea notei de plata de catre client.

În tabelul 6.1 se va prezenta cazul de test pentru funcționalitatea de inregistrare. Aceasta este folositoare în momentul în care un utilizatorul dorește să beneficieze de funcționalitatile aplicației în funcție de rolul pe care il detine (client, ospătar sau chelner).

**Precondiții**: Utilizatorul trebuie să aibă aplicația instalata

Tabel 6.1 - Cazul de test a funcționalitatii de inregistrare a unui utilizator

|  |  |
| --- | --- |
| **Acțiune** | **Rezultat** |
| Utilizatorul deschide aplicația | Aplicația se deschide fara erori și este prezentat ecranul de autentificare |
| Utilizatorul face clic pe butonul cel mai de jos, „Creeaza un cont nou” | Utilizatorul este redirecționat catre ecranul de inregistrare |
| Uitilizatorul completeaza corect toate campurile prezente și da clic pe butonul de jos | Utilizatorului ii este prezentat un mesaj de informare de a-și verifica adresa de email |
| Utilizatorul apasa pe butonul de ok al mesajului de informare | Utilizatorul este redirecționat inapoi la ecranul de autentificare |

Ca urmare a acestui caz, utilizatorul are pe adresa de email un email nou pentru confirmarea adresei de email introduse la inregistrarea contului.

În urmatorul tabel, 6.2, se va prezenta cazul de test pentru funcționlitatea de rezervare a unei mese. Funcționalitate dedicata doar utilizatorului de tip client și dupa ce acesta s-a hotarat să rezerve o masă la un restaurant ales, în urma interactionarii cu lista de restaurante și pagina de detalii a acestuia.

**Precondiții**: Utilizatorul trebuie să aibă rol de client și să fie deja autentificat în aplicație

Tabel 6.2 - Cazul de test pentru funcționalitatea de rezervare a unei mese

|  |  |
| --- | --- |
| **Acțiune** | **Rezultat** |
| Utilizatorul da clic pe optiunea din mijloc a tabului pozitionat în josul ecranului | Utilizatoul este redirecționat catre ecranul ce contine lista restaurantelor din aplicație |
| Utilizatorul căuta restaurantul dorit la care dorește să rezerve o masă | Restaurantul exista în aplicație |
| Utilizatorul da click pe restaurant evitand zona stelei galbene din dreapta | Utilizatorul este redirecționat catre pagina de detalii a restaurantului |
| Utilizatorul apasa pe butonul plasat în dreapta a barei de sus cu textul „Rezerva o masă” | Utilizatorului ii este prezentat ecranul din care se poate rezerva o masă, iar în mijloc se afla lista meselor disponibile și nu mesajul de eroare care sugereaza indisponibilitatea lor |
| Utilizatorul alege ora din picker-ul din stanga, ziua din picker-ul din dreapta și dupa apasa pe butonul „Rezerva” din dreptul mesei care se dorește a fi rezervata | Se afiseaza un mesaj de confirmare cu doua butoane, „Da” și „Nu” |
| Utilizatorul apasa butonul de „Da” | Apare mesaj de informare în cu un buton de „Ok” |
| Utilizatorul apasa pe butonul de „Ok” | Mesajul de informare dispare |
| Utilizatorul apasa pe butonul „Anuleaza” din partea stanga a barei de sus | Ecranul de rezervare a unei mese dispare și reapare cel de detalii a restaurantaului |
| Utilizatorul apasa pe butonul „Inapoi” din partea stanga a barei de sus | Utilizatorul este redirecționat la lista restaurantelor |
| Utilizatorul apasa pe butonul de profil a tab-ului pozitionat în partea de jos a ecranului | Utilizatorul este redirecționat catre pagina de profil |
| Utilizatorul apasa pe butonul de „Rezervari” dintre cele doua optiuni, iar în caz contrar nu mai apasa niciun buton | Rezervarea este vizibila sub forma unui card în lista de „Rezervari” |

În tabelul 6.3 este prezentat cazul de testare pentru funcționalitatea de echilibrare a comenzilor. Pentru acest caz va fi nevoie de un client, doi ospătari (ospătar 1 și ospătar 2) și un bucătar.

**Precondiții**: Clientul are deja o rezervare facuta la un anumit restaurant și este pe pagina de profil, cei doi chelneri și bucătarul lucreaza la acelasi restaurant unde este și rezervarea și sunt autentificati.

Tabel 6.3 - Cazul de test pentru algoritmul de echilibrare a comenzilor

|  |  |
| --- | --- |
| **Acțiune** | **Rezultat** |
| Clientul apasa butonul verde „Confirma sosirea” | Clientului ii este prezentat ecranul din care poate plasa comenzi |
| Clientul apasa pe butonul de „+” a unui fel de mâncare | Acel fel de mâncare apare în a doua lista ce reprezintă comanda curenta |
| Clientul apasa pe butonul „Trimite” | Lista cu comanda este resetata și comanda este trimisa și apar pe ecranele ospătarilor |
| Clientul repeta ultimii doi pasi pentru a mai trimite trei comenzi | Pe ecranele ospătarilor este un total de patru comenzi |
| Ospătar 1 apasa pe butonul verde „Trimite la bucătarie” la doua comenzi | Cele doua comenzi sunt trimise la bucătar, sterse din lista ospătarilor, ospătar 1 are butonul de „Trimite la bucătarie” cu culoarea gri în timp ce ospătar 2 are cu culoarea verde |
| Bucătarul apasa la o comanda butonul verde „Comanda este gata” | Comanda apare inapoi pe ecranul ospătarilor |
| Ospătar 1 apasa pe butonul verde a comenzii venite de la bucătar cu textul „Serveste comanda” | Comanda este stearsa din lista ospătarilor și butoanele gri ale primului ospătar devin din nou verde |

În tabelul 6.4 este prezentat cazul de testare în care clientul cere nota de plata.

**Precondiții**: Clientul are deja o rezervare facuta și este pe pagina de profil, ospătarul și bucătarul lucreaza la acelasi restaurant la care s-a facut rezervarea.

Tabel 6.4 - Cazul de test a cererii notei de plata

|  |  |
| --- | --- |
| **Acțiune** | **Rezultat** |
| Clientul apasa pe butonul „Confirma sosirea” | Clientului ii este prezentat ecranul din care este posibila acțiunea de plasare a unei comenzi |
| Clientul apasa pe cel putin un buton „+” a unui fel de mâncare | Felul de mâncare apare în lista de jos a ecranului care indica cantiatea aleasa |
| Clientul apasa pe butonul de „Trimite” | Lista de jos este resetata și comanda apare pe ecranul ospătarului |
| Ospătarul apasa butonul verde care are textul „Trimite la bucătarie” | Comanda este stearsa din lista ospătarului și afisata în lista bucătarului |
| Bucătarul apasa pe butonul verde „Comanda este gata” | Comanda este stearsa din lista bucătarului și apare în lista ospătarului |
| Ospătarul apasa butonul verde „Serveste comanda” | Comanda este stearsa din lista ospătarului |
| Clientul apasa pe butonul din partea dreapta a barei de sus „Cere nota” | Clientului ii este prezentat o lista cu toate felurile de mâncare pe care le-a comandat și totalul |
| Clientul apasa butonul „Achitati nota de plata” din partea stanga a barei de sus | I se prezinta un mesaj cu optiuni pentru a alege nota de plata |
| Clientul apasa pe una din urmatoarele optiuni: Cash, Cu cardul, Folosind aplicația | Se afiseaza un mesaj de informare ca un angajat urmeaza să vina la masă |
| Clientul apasa pe butonul de Ok la mesajului informativ | Clientul este redirecționat la pagina de profil |

# Manual de instalare și utilizare

În acest capitol se vor descrie etapele ce trebuie urmate pentru instalarea și rularea cu succes a aplicației QuickPlate pe un dispozitiv. Pe langa acestea, se vor prezenta resursele necesare atât hardware cât și software. Pentru a putea publica aplicația pe AppStore, din moment ce este aplicație iOS, este necesara detinerea unui certificat de dezvoltator, oferit de Apple, în valoare de 100 de dolari. Acestea fiind spuse, s-a preferat pastrarea aplicației intr-o varianta locala. Inainte de toate acestea, utilizatorul va trebui să își creeze un cont Apple, lucru posibil de pe site-ul lor oficial.

## Resursele necesare

### Resurse hardware

* Smartphone cu sistemul de operare iOS sau tableta cu sistemul de operare iPadOS. Este obligatoriu ca ambele versiunea în oricare din cele doua cazuri să fie mai mare decat 16.0.
* Un laptop Macbook, fie cu procesor Intel fie cu procesor M1, acesta fiind necesar pentru rularea sistemului de operare MacOS. În funcție de procesorul ales, trebuie schimbate anumite configurari de Cocoa Pods.
* Un cablu pentru conectarea device-ului cu port Lightning (pentru iPhone sau anumite tipuri de tablete) sau cu port USB-C (pentru anumite tipuri de tablete)

### Resurse software

* Mediul de dezvoltare integrat al Apple, Xcode, cu versiunea minima de 14.2.
* Git pentru a putea clona proiectul din depozitul de pe GitHub
* SourceTree pentru o vizualizare mai bună a evolutiei proiectului, aceasta fiind optionala

## Manual de instalare

În acest subcapitol se vor prezenta pasii ce trebuie urmati în ordinea descrierii lor pentru ca utilizatorul să reuseasca descarcarea și instalarea aplicației:

* Utilizatorul va trebui să își descarce și să își instaleze ultima versiunea de Xcode. Acest lucru se poate face prin intermediul AppStore existent pe MacOS sau de pe site-ul apple developer (<https://developer.apple.com>) , secțiunea Develop, secțiunea Xcode, se apasa butonul de Download, se selecteaza variante de Website, își introduce credențialele contului de Apple, și se va descarca ultima versiune.
* Al doilea pas este descarcarea și instalarea uneltei Git. Acest lucru se poate realiza prin mai multe moduri: se poate căuta pe internet, se poate instala folosind managerul de pachete Homebrew sau se instaleaza automat la instalarea tool-ului SourceTree
* Se va naviga intr-un fisier unde se dorește scanarea proiectului și se va deschide în acel loc un terminal
* Se va introduce urmatoarea comanda: git clone <https://github.com/StanciuOctav/QuickPlate>
* Se va deschide Xcode, fie prin apasarea de doua ori a fisierului QuickPlate.xcodeproj, fie prin a selecta din stanga sus File -> Open -> și se va da dublui clic pe modulul care contine proiectul.
* Pana când proiectul se configureaza și se descarca toate dependintele necesare, se poate conecta dispozitivul la laptop prin intermediul cablului necesar.
* În Xcode, se va alege din lista de simulatoare dispozitivul pe care se dorește rularea și se va apasa butonul de play din stanga sus sau scurtatura CMD + R. În figura 7.1 se poate observa cum arata aceasta lista și cum se poate identifica dispozitivul utilizatorului, acesta a vand alta iconita fata de cele ale simulatorului.

A screenshot of a computer

Description automatically generated with medium confidence

Figură 7.1 - Lista de simulatoare din Xcode

* Pentru a testa aplicația observand în acelasi timp toate funcționalitatile ei, trebuie rulate minimul patru instante dintre care una este rolul de client, a doua cea de bucătar, iar ultimele doua cele de ospătari. Pe dispozitiv se poate rula o singura instanta, astfel se vor alege din lista de simulator inca trei, iar dupa fiecare selectie se va rula proiectul.

## Manual de utilizare

În momentul în care o instanta a aplicației va porni se va afisa pentru doua secunde un ecran de prezentare. Dupa aceasta utilizatorul va fi redirecționat catre ecranul de autentificare. Acestea doua sunt prezentate în figura 7.2

A screenshot of a phone

Description automatically generated

Figură 7.2 - Ecran de prezentare și ecranul de autentificare a aplicației

În ecranul de autentificare utilizatorul își va putea introducere credențialele pentru putea executa acțiunea de SignIn. Daca nu are cont, atunci acesta va fi nevoit să apase cel mai de jos buton, ,,Creeaza un cont nou”, pentru a putea fi redirecționat catre ecranul de inregistrare.

Odata ajuns la acest ecran, utilizatorul este obligat să completeze toate campurile prezente, cu exceptia celui de ,,Restaurant” care este inactiv în cazul în care tipul contului este ,,Client”. Daca rolul utilizatorului va fi ori ,,Chelner” ori ,,Bucătar” atunci campul respectiv va deveni activ și va trebui să se selecteze numele restaurantului la care acesta lucreaza. Bineinteles, pot aparea mesaje de eroare. Pentru cazurile în care parolele nu coincid sau nu se conformează formatului acceptat, se va afisa un mesaj de eroare deasupra butonului ,,Creeaza un cont nou” ca în figura 7.4, iar pentru cazul în care unul dintre campuri nu este completat sau adresa de email este deja folosita atunci se vor afisa mesaje speciale ce ii vor sugera utilizatorului greselile pe care le-a facut pentru a le putea remedia și continua procesur de inregistrare.

Daca toate campurile au fost completate corect și s-a respectat formatul atunci în momentul în care utilizatorul apasa pe butonul de creare a contului, i se va afisa un mesaj de succes similar cu cel din figura 7.4, poza din dreapta, care ii va sugera să își verifice adresa de email pentru a confirma adresa introdusa. Dupa aceasta, o să fie redirecționat inapoi la ecranul de autentificare. În figura 7.3, se poate observa cum arata emailul de confirmare. Dupa ce se acceseaza link-ul de culoare albastra, utilizatorul va putea dupa să se autentifice fara nicio problema.

A screenshot of a email

Description automatically generated with low confidence

Figură 7.3 - Email de confirmare

Odata intors, acesta va trebui să își reintroduca adresa de email și parola ce au fost introduse pe ecranul de inregistrare. În cazul în care, utilizatorul nu a reusit să își faca contul sau nu și-a verificat adresa de email, atunci se va afisa un mesaj de eroare deasupra butonului ,,Intra în cont”.

Odata ce utilizatorul a reusit să se autentifice în aplicație în urma introducerii adresei de email și a parolei corespunzatoare contului creat, acesta va fi redirecționat catre ecranul principal. Acesta difera de la un utilizator la altul, diferenta făcând-o tipul contului ales în ecranul de inregistrare. Daca acesta a ales optiunea de ,,Client” atunci el va fi redirecționat catre un ecran ce contine în partea de jos o bara cu trei secțiuni. Fiecare secțiune va afisa alt ecran în timp ce pentru celelalte doua tipuri de utilizator, se va afisa un singur ecran. În continuare se va prezenta manualul de utilizare pentru client, iar pentru ospătar și bucătar se vor explica în paralel.

### Manual de utilizare pentru client

Odata autentificat în aplicație, clientului i se va prezenta ecranul dedicat rolului lui. Ecranul de harta va fi primul care o să apara, impreuna cu restaurantele ce sunt reprezentate ca și niste pin-uri. Din moment ce pin-urile sunt interactive, clientul poate apasa pe ele, acestea afisand informatii despre restaurantul ce se gaseste la acea locație, lucru ce poate fi observat în figura.

Informatiile restaurantului sunt afisate în partea de jos a ecranului. În partea dreapta de sus a hartii exista un buton ce va recentra harta la locația utilizatorului, dar pentru ca acest lucru să se poate intampla, prima data clientul va trebuie să permita aplicației să acceseze locația acestuia. Daca clientul refuza, acel buton nu va fi de folos. În figura 7.3 se vede rezultatul actiunii apasarii butonului în cazul în care clientul a permis accesul locației, poza din dreapta reprezentand rezultatul.

A screenshot of a map

Description automatically generated

Figură 7.4 - Ecranul de harta

Pe secțiunea din mijloc a barei de jos, daca este selectata, va fi prezentata lista ce contine toate restaurantele din aplicație. Deasupra acesteita este o bara de cautare prin care se poate face filtrarea restaurantlor dupa numele lor. De asemenea, în dreptul fiecarui element este plasat un butonul cu forma unei stele. Daca steaua nu este umpluta cu culoare galbena și butonul este apsat, atunci steaua o să fie umpluta cu culoarea mentionata, insemnand ca restaurantul a fost adaugat în lista de favorite de pe pagina de profil a utilizatorului. În cazul în care steaua este deja umpluta cu culoare și se apasa din nou butonul atunci restaurantul va fi sters din lista de favorite. În figura 7.4, în poza din stanga, se poate vizualiza lista restaurantul cu numele Marty și Trattoria Pineta fiind în lista de favorite în timp ce restul restaurantelor sunt excluse, iar în partea din dreapta se observa funcționalitatea barei de cautare, aceasta filtrand restaurantele ce contin în numele lor sirul de caractere ,,Ma”.

A screenshot of a phone

Description automatically generated with medium confidence

Figură 7.5 - Ecranul cu lista de restaurante

Fiecar element a listei descrise anterior este interactiv fiind de ajutor la navigarea catre pagina de detalii a restaurantului ce contine un număr mai mare de informatii despre restaurantul pe care s-a facut clic. De asemnea, toate acestea sunt puse intr-o lista cu care nu se poate interactiona deoarece are ca scop doar oferirea posibilitatii clientului de a vizualiza meniul corespondent restaurantului. În bara de sus sunt doua butoane:

* Inapoi: la apasarea acestuia clientul o să fie readus pe pagina cu lista restaurantelor
* Rezerva o masă: buton care la apasarea lui va prezenta ecranul din care clientul își poate rezerva o masă

În ecranul de rezervare utilizatorul este nevoie să selecteze ora și ziua prin gestul de derulare a listelor, iar dupa acestea să apese butonul ,,Rezerva” din dreptul mesei ce se dorește a fi rezervata. În ecranul de rezervare, în coltul stanga sus exista un buton ,,Anuleaza” care inchide ecranul de rezervare de mese.

Utlima secțiune a bareri de jos este profilul în care utilizatorul poate vizualiza comenzile respectiv rezervarile efectuate, iar în dreptul numelui de familie este un buton cu titlul ,,Deconectare” ce ajuta utilizatorul să se deconecteze din aplicație. La fiecare rezervare sunt doua butoane: cel verde pentru a confirma sosirea la restaurant și pentru a incepe plasarea comenzilor, iar cel rosu pentru a anula rezervarea. În ecranul de plasare a comenzii, lista felurilor de mâncare este afisata și este interactiva intrucat clientul poate interactiona cu fiecare. De asemenea, în bara de sus sunt trei butoane: unul pentru a plasa comanda, unul pentru a o reseta și unul pentru a cere nota de plata. Dupa alegerea metodei de plata, rezervarea este stearsa și clientul este redirecționat catre pagina de profil.

### Manual de utilizare pentru ospătar și bucătar

Dupa autentificare, angajatul este redirecționat direct catre pagina de profil, singurul ecran care i se afiseaza. În centru, sub secțiunea de detalii a utilizatorului, exista o lista care se populeaza cu comenzile plasate de clienti. Pentru rolul de ospătar, fiecare element are doua butoane: unul verde pentru a trimite comanda la bucătarie și care devine gri în cazul în care acesta nu mai poate accepta comenzi și unul rosu pentru a anula comanda, iar în cazul bucătarului exista doar butonul verde pentru a trimite comanda inapoi la ospătari sugerand ca s-a terminat de preparat. În figura 7.5, se pot observa patru instante ale aplicației ce ruleaza în acelasi timp, cea mai din stanga fiind pentru bucătar, cele doua din mijloc pentru ospătari, iar cea din dreapta pentru client.

A screenshot of a phone

Description automatically generated with low confidence

Figură 7.6 - Exemplu de rulare folosind simluatorul Xcode

# Concluzii

Proiectul urmareste să ajute oamenii în economisirea timpului de a rezerva o masă și de a fluidiza cât mai mult interacțiunea dintre client și angajatul unui restaurant. QuickPlate este o aplicație ce ajuta la rezolvarea acestei probleme datorita funcționalitatilor pe care le detine și prin care se diferentiaza comparativ cu aplicațiile concurente existente pe piata la momentul actual. În tabelul 8.1 este facuta o comparație dintre aplicația QuickPlate și aplicațiile existente pe piata, iar evaluarea s-a facut dupa verificarea de detinere a urmatoarelor funcționalități:

1. O harta integrata pe care utilizatorul poate să observe restaurantele din proximitatea lui.
2. Posibilitatea de a vizualiza meniul restaurantelui și de a comanda din aplicație.
3. Posibilitatea de a plati din aplicație sau de a cere nota de plata să fie adusa la masă
4. Posibilitatea de a folosi aplicația fara a cere un abonament utilizatorilor

Tabel 8.1 - Comparație intre QuickPlate și aplicații existente

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | iaLoc! | Resy | Tock | SevenRooms | OpenTable | QuickPlate |
| 1) | ✅ | ✅ | ✅ | - | - | ✅ |
| 2) | - | - | - | - | - | ✅ |
| 3) | ✅ | ✅ | ✅ | ✅ | ✅ | ✅ |
| 4) | ✅ | - | - | - |  | ✅ |

Pe langa funcționalitatile de baza precum autentificare, deconectare, inregistrare și un ecran de baza, prezente în majoritatea aplicațiilor mobile, atât Android cât și iOS, aplicația dispune și de alte caracteristici precum harta pe care se pot observa restaurantele, posibilitatea de a rezerva o masă și de a comanda direct de pe dispozitiv. Cu toate acestea, interfata utilizator este una foarte sugestiva și minimalista, pe parcursul dezvoltarii proiectului urmarindu-se pastrarea simplitatii și evitarea incarcarii ecranelor cu elemente vizuale.

Desi unele aplicații de pe piata sunt mai bogate în anumite funcționalități de baza cum ar fi detinerea mai multor informatii despre utilizator și restaurant, s-a urmarit implementarea cât mai multor funcționalități cu scopul de a fi cât mai usor scalabila în eventuala dezvoltare prin adaugarea unor funcționalități noi, componentele create să fie reutilizabile, iar clasele să nu fie cuplate strans, lucru realizat datorita folosirii modelului arhitectura Model-View-View-Model.

## Contributiile propria

Proiectul urmareste eficientizarea procedeului de rezervare a unei mese și atât de primire cât și de livrare a unei comenzi la restaurant. Aplicația ce a fost dezvoltata are ca scop indeplinirea acestor sarcini datorita simplitatii și rezultatelor obtinute în urma evoluarii acesteia.

Contributiile proprii aduse au fost stabilirea bazelor aplicației atât din punct de vedere a structurarii codului și componentelor cât și din cel al experientei utilizator și interfetei grafice. Desi anumite aplicații de pe piata oferă mai multe informatii precum zilele speciale în care restaurantele nu sunt deschise, ziua de nastere a utilizatorului, istoricul cautarilor și multe altele, s-a urmarit funcționarea cât mai eficienta și simpla a funcționalitatilor implementate pentru a merge cât mai bine în aplicația proprie. Cateva dintre acestea sunt:

* Integrarea API-ului de la Firebase pentru stocarea și modificarea datelor entităților folosite în aplicație
* Integrarea API-ului de harta oferit de Apple pentru o navigare cât mai usoara a utilizatorului
* Oferirea posibilitatii unui utilizator de a rezerva o masă la un restaurant ales
* Posibilitatea pentru client de a alege metoda de plata
* Echilibrarea comenzilor pentru a eficientiza primirea și livrarea comenzilor

## Dezvoltari ulteriorare

Partea de dezvoltare ulterioara este usoara de realizat datorita arhitecturii simple și structura claselor ce fac posibila refolosirea lor cu usurinta, asa cum am explicat și în capitolele anterioare. Urmatoarea lista enumereaza posibilele optiuni ce se pot adauga intr-o versiune viitoare a aplicației:

* Partea de design: Se poate adauga pe viitor suport pentru modul intunecat (dark theme), aplicația avand la momentul actual culorile setate doar pentru partea luminoasa (light theme), culorile se pot alege mai potrivit pentru acest tip de aplicație.
* Adaugarea de notificari: Se poate adauga funcționalitatea de primire de notificari pentru toti utilizatorii (clientul primeste notificare când s-a plasat comanda, când o comanda a fost servita sau cu un anumit timp inainte de a-i aduce aminte de rezervarea facuta, iar ospătarul primeste notificare când s-a plasat o noua comanda sau cineva dorește să achite nota de plata).
* Adaugarea unui sistem de navigare: Pe pagina de harta se poate adauga un sistem de navigare de la pozitia curent la utilizatorului pana la restaurantul selectat scutind-ul pe acesta de a folosi alta aplicație cu acest scop precum Google Maps, Apple Maps sau Waze.
* Adaugarea unui sistem de recenzii: Implementarea funcționalitatii de a putea lasa recenzii la restaurante de catre clienti, iar în funcție de evaluarea lor să se actualizeze rating-ul restaurantului.
* Adaugarea mai multor filtre de cautare: Posibilitatea de a-i permite utilizatorului cautarea restaurantelor dupa diferite criterii precum rating, locație, fel de mâncare și altele.
* Posibilitatea de a arata o harta a meselor: Integragrea unei funcționalități în care clientului nu ii este prezentat o lista cu restaurantele ci o harta a acestor cu care poate interactiona, iar aceasta să fie configurabila de catre manager-ul restaurantului în cazul unor modificari.
* Posibilitatea de a invita prieteni la o rezervare: Dezvoltarea unei optiuni prin care un client își poate invita prietenii la o rezervare facuta pentru ca toti să primeasca un reminder.
* Posibilitatea de a imparti nota: Impreuna cu funcționalitatea anterioara, toti clientii ce sunt la aceeasi rezervare să poata să comande fiecare separat, iar la final nota să se poata plati ori de un singur om, ori fiecare să plateasca propria consumație.

# Bibliografie

|  |  |
| --- | --- |
| [1] | Apple - https://ro.wikipedia.org/wiki/Apple\_Inc. |
| [2] | John Hughes. ,,iPhone vs Android: Which is better for Web Professionals / Developers?, 1 Noiembrie, 2022. |
| [3] | Petroc Taylor. ,,Mobile operating systems' market share worldwide from 1st quarter 2009 to 4th quarter 2022", 21 Februarie, 2023. |
| [4] | HORECA - <https://ro.wikipedia.org/wiki/HoReCa>. |
| [5] | Lynn Beyrouthy. ,,Online food delivery - statistics & facts", 21 Martie, 2023. |
| [6] | UberEats - https://www.ubereats.com/ |
| [7] | Deliveroo - https://deliveroo.co.uk/ |
| [8] | Loritz Mary. ,,Execute quickly, understand local markets and adapt fast to different needs - Glovo CEO Oscar Pierre", 14 Decembrie, 2018. |
| [9] | Statistica 2023. ,,The number of downloads of leading online food delivery and takeout apps worldwide în 2022", Martie 2023 - https://www.statista.com/statistics/1369501/food-delivery-app-downloads-global/ |
| [10] | Khosla Varuni, Srinivasan Supraja. ,,Zomato co-founder Pankaj Chaddah quits as it shuffles top management", Economic Times, 9 Aprilie 2022. |
| [11] | Yoolim Lee. ,,Delivery Hero's Foodpanda Claims Asia Top Spot, Sees More Growth", 5 Iulie, 2021. |
| [12] | Bottorff Cassie, Haan Kathy. ,,Toast POS Review 2023: Features, Pricing & More”, 5 Mai, 2023. |
| [13] | Fabregas Krista, Main Kelly. ,,Best Restaurant Inventory Management Software 2023”, 5 Aprilir, 2023. |
| [14] | Jolaoso Christiana, Main Kelly, Watts Rob. „Toast Vs. Square (2023 Comparison), 14 Mai, 2023. |

# Anexa 1. Glosar de termeni

|  |  |  |
| --- | --- | --- |
| Abreviere | Denumire | Explicatie |
| API | Application Programming Interface | Orice program software ce are un rol distinct |
| BaaS | Backend-as-a-Service | Un model ce pune la dispozitie dezvoltatorilor de aplicații web și mobile diferite servicii precum o baza de date pe cloud, un sistem de administrare a utilizatorilor, notificari etc. |
| CRUD | Create Read Update Delete | Descriu cele patru operatii esentiale mentru crearea și administrarea datelor persistente în bazele de date |
| DTO | Data Transfer Object | Obiect pentru incapsularea datelor și pentru a reduce cantitatea de date transferate intre multiple sisteme |
| GUI | Graphical User Interface | Interfata prin care utilizatorul interactioneaza cu un sistem |
| IDE | Integrated Development Environment | Aplicație software ce oferă o multitudine de facilitati pentru dezvoltarea software |
| iOS | iPhone Operating System | Sistemul de operare a tuturor telefoanelor iPhone |
| iPadOS | iPad Operating System | Sistemul de operare a tabletelor iPad |
| UID | User Identifier | Un identificator unic asignat utilizatorilor unei aplicații |

# Anexa 2. Lista figurilor și tabelelor

[Figură 4.1 - Design prototipizat în Figma 11](#_Toc137147333)

[Figură 4.2 - Diagramă de stari a unei comenzi 14](#_Toc137147334)

[Figură 4.3 - Diagramă de flow pentru inregistrare 15](#_Toc137147335)

[Figură 4.4 - Diagramă de flow pentru rezervarea unei mese 16](#_Toc137147336)

[Figură 4.5 - Diagramă de flow pentru comanda și plata consumației 18](#_Toc137147337)

[Figură 4.6 - Diagramă de flow pentru pentru comenzilor 19](#_Toc137147338)

[Figură 4.7 - Diagramă de flow pentru gestionarea comenzilor per chelner 20](#_Toc137147339)

[Figură 5.1 - Diagramă aplicației QuickPlate 22](#_Toc137147340)

[Figură 5.2 - Relatiile entităților bazei de date 23](#_Toc137147341)

[Figură 5.3 - Diagramă claselor principale 26](#_Toc137147342)

[Figură 5.4 - Fisierul Lozalizable.strings 27](#_Toc137147343)

[Figură 5.5 - Pachetul Assets.xcassets 28](#_Toc137147344)

[Figură 5.6 - Funcția de login a clasei FirebaseEmailAuth 30](#_Toc137147345)

[Figură 5.7 - Structura modulului Extras 31](#_Toc137147346)

[Figură 5.8 - Utilizarea clasei AuthManager 33](#_Toc137147347)

[Figură 5.9 - Diagramă de clase a modulului SignIn 34](#_Toc137147348)

[Figură 5.10 - Diagramă de clase a modulului SignUp 35](#_Toc137147349)

[Figură 5.11 - Diagramă de clase a modulului Map 36](#_Toc137147350)

[Figură 5.12 - Diagramă de clase a modulului Restaurants 37](#_Toc137147351)

[Figură 5.13 - Diagramă de clase a modulului RestaurantDetails 38](#_Toc137147352)

[Figură 5.14 - Diagramă de clase a modulului TablesList 39](#_Toc137147353)

[Figură 5.15 - Diagramă de clase a modulului UserProfile 40](#_Toc137147354)

[Figură 5.16 - Diagramă de clase a modulului OrderView 41](#_Toc137147355)

[Figură 5.17 - Diagramă de clase a modulului ClientOrder 42](#_Toc137147356)

[Figură 5.18 - Starile unei comenzi 43](#_Toc137147357)

[Figură 7.1 - Lista de simulatoare din Xcode 48](#_Toc137147358)

[Figură 7.2 - Ecran de prezentare și ecranul de autentificare a aplicației 49](#_Toc137147359)

[Figură 7.3 - Email de confirmare 50](#_Toc137147360)

[Figură 7.4 - Ecranul de harta 51](#_Toc137147361)

[Figură 7.5 - Ecranul cu lista de restaurante 52](#_Toc137147362)

[Figură 7.6 - Exemplu de rulare folosind simluatorul Xcode 53](#_Toc137147363)

[Tabel 3.1 - Diferente intre Toast și Square 8](#_Toc137052776)

[Tabel 6.1 - Cazul de test a funcționalitatii de inregistrare a unui utilizator 43](#_Toc137052777)

[Tabel 6.2 - Cazul de test pentru funcționalitatea de rezervare a unei mese 43](#_Toc137052778)

[Tabel 6.3 - Cazul de test pentru algoritmul de echilibrare a comenzilor 44](#_Toc137052779)

[Tabel 6.4 - Cazul de test a cererii notei de plata 45](#_Toc137052780)

[Tabel 8.1 - Comparație intre QuickPlate și aplicații existente 52](#_Toc137052781)

# Anexa 3. Exemple de cod

**Funcționalitatea de autentificare**:  
**func** doLogin(email: String = "", password: String = "", completion: **@escaping** (Result<String, StartupError>) -> Void) {

Auth.auth().signIn(withEmail: email, password: password, completion: { result, error **în**

**guard** result != **nil** **else** {

**if** **let** error {

print(error.localizedDescription)

}

completion(.failure(.signInError))

**return**

}

**guard** **let** user = result?.user **else** {

completion(.failure(.anonymousUser))

**return**

}

completion(.success(user.uid))

**switch** user.isEmailVerified {

**case** **true**:

print("Email is verified")

completion(.success(1))

**case** **false**:

print("Email is not verified")

completion(.failure(.emailExists))

}

})

}

**Funcționalitatea de trimitere a unei comenzi din clasa ClientOrderViewModel:**

**func** sendOrder() {

FSResColl.shared.getResNameThatHas(tableId: tableId) { name **în**

**guard** **let** name = name **else** {

print("ClientOrderVM - Couldn't get restaurant's name that has the table with id \(**self**.tableId)")

**return**

}

**var** ids: [String] = []

**var** quan: [Int] = []

**for** index **în** 0 ..< **self**.numberOrdered.count {

**if** **self**.numberOrdered[index] > 0 {

ids.append(**self**.foods[index].id ?? "")

quan.append(**self**.numberOrdered[index])

FSFoodsColl.shared.updateFoodstockkWith(id: **self**.foods[index].id ?? "", nrOrdered: **self**.numberOrdered[index], addStock: **false**)

}

}

**self**.calculateTotalCost()

**let** order = Order(id: UUID().uuidString,

resName: name,

tableNr: **self**.table.tableNumber,

foodIds: ids,

foodQuantity: quan,

totalCost: **self**.totalCost,

userId: UserDefaults.standard.value(forKey: "userId") **as**? String ?? "",

tableId: **self**.tableId,

orderState: .pending)

FSOrdersColl.shared.saveOrder(order)

**self**.resetOrder()

}

}

**Funcționalitatea de acceptare a unei comenzi de catre un ospătar:**

**func** acceptOrder(id: String, state: OrderState) {

**var** currentNumber = UserDefaults.standard.integer(forKey: "ordersAccepted")

**if** state == .pending {

currentNumber += 1

}

**if** state == .ready {

currentNumber -= 1

}

**if** currentNumber > maxNumberOfOrders {

currentNumber = maxNumberOfOrders

}

UserDefaults.standard.set(currentNumber, forKey: "ordersAccepted")

FSOrdersColl.shared.changeOrderState(id: id)

canAcceptOrders()

}

**private** **func** canAcceptOrders() {

**var** currNumber = 0

**if** (UserDefaults.standard.object(forKey: "ordersAccepted") == **nil**) {

UserDefaults.standard.set(0, forKey: "ordersAccepted")

} **else** {

currNumber = UserDefaults.standard.integer(forKey: "ordersAccepted")

}

canTakeOrder = !(currNumber >= maxNumberOfOrders)

}

**func** changeOrderState(id: String) {

coll.document(id).getDocument { [**weak** **self**] qdSnap, error **în**

**if** **let** error = error {

print("FSOrderColl - Couldn't retrieve order with id \(id)")

print(error.localizedDescription)

**return**

}

**guard** **let** self = **self** **else** { **return** }

**guard** **let** qdSnap = qdSnap **else** {

print("FSOrderColl - There is no order with the id \(id)")

**return**

}

**let** order = **try**? qdSnap.data(as: Order.**self**)

**guard** **let** order = order **else** {

print("FSOrderColl - Couldn't not convert qdSnap to order")

**return**

}

**switch** order.orderState {

**case** .pending:

self.setOrderState(orderId: id, state: .preparing)

**case** .preparing:

self.setOrderState(orderId: id, state: .ready)

**case** .ready:

self.setOrderState(orderId: id, state: .sent)

**case** .sent:

**break**

}

}

}

**Funcționalitatea de rezervare a unei mese**:

**func** bookingTable(tableId: String, hour: String, day: String, userId: String) {

FSTableColl.shared.tableBooked(tableId: tableId, hour: hour, day: day, userId: userId)

FSUserColl.shared.saveBookedTable(withId: tableId)

updateTables()

}

**func** tableBooked(tableId: String, hour: String, day: String, userId: String) {

coll.document(tableId).setData(["booked": **true**,

"hourBooked": hour,

"day": day,

"userId": userId], merge: **true**)

}

**func** saveBookedTable(withId tableId: String) {

**let** userId = UserDefaults.standard.value(forKey: "userId") **as**! String

**let** currUser = coll.document(userId)

currUser.getDocument { qdSnap, error **în**

**if** **let** error = error {

print("FSUserColl - Couldn't assign booked table to user")

print(error.localizedDescription)

**return**

}

**if** **let** qdSnap = qdSnap, **let** document = **try**? qdSnap.data(as: MyUser.**self**) {

**var** newBookedTablesArr = document.bookedTables

newBookedTablesArr.append(tableId)

currUser.updateData(["bookedTables": newBookedTablesArr])

}

}

}