



**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**  
**DEPARTAMENTUL CALCULATOARE**

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

LUCRARE DE LICENȚĂ

Absolvent: **Ioan-Octavian Stanciu**

Coordonator   **Conf. Dr. Ing. Lia-Anca Hangan**  
științific:

**2023**

## Cuprins

<b>Capitolul 1. Introducere.....</b>	<b>1</b>
<b>Capitolul 2. Obiectivele proiectului .....</b>	<b>3</b>
2.1.    Obiectivul principal .....	3
2.2.    Obiective specifice.....	3
<b>Capitolul 3. Studiu bibliografic .....</b>	<b>5</b>
3.1.    Aplicații mobile de tipul food-delivery .....	5
3.1.1.    Aplicația Zomato .....	5
3.1.2.    Aplicația UberEats.....	6
3.1.3.    Aplicația Foodpanda.....	6
3.1.4.    Asemănări și diferențe între Zomato, UberEats și Foodpanda .....	6
3.2.    Software de administrare al restaurantelor .....	7
3.2.1.    Toast POS .....	7
3.2.2.    Square POS.....	7
3.2.3.    Comparație între aplicațiile Toast și Square.....	8
3.3.    Avantajele iOS comparativ cu Android.....	8
<b>Capitolul 4. Analiză și fundamentare Teoretică .....</b>	<b>10</b>
4.1.    Analiza problemei.....	10
4.2.    Cerințe funcționale.....	11
4.3.    Cerințe non-funcționale .....	12
4.4.    Modelul de date al aplicației.....	12
4.5.    Stările unei comenzi .....	13
4.6.    Cazuri de utilizare.....	14
4.6.1.    Înregistrare .....	14
4.6.2.    Rezervarea unei mese .....	15
4.6.3.    Comanda și plata consumăției .....	16
4.6.4.    Procesarea comenzilor .....	18
4.7.    Algoritmul de echilibrare a comenzilor per chelner .....	19
<b>Capitolul 5. Proiectare de detaliu și implementare .....</b>	<b>21</b>
5.1.    Proiectarea și implementarea arhitecturii .....	21
5.2.    Proiectarea și implementarea bazei de date .....	22
5.2.1.    Entitatea Food.....	23
5.2.2.    Entitatea MyUser.....	23

5.2.3. Entitatea Order.....	24
5.2.4. Entitatea Restaurant.....	24
5.2.5. Entitatea Table.....	25
5.3. Structura proiectului .....	25
5.3.1. Structura fișierelor .....	26
5.3.2. Modulul Extras .....	28
5.3.3. Modulul LoginStates .....	31
5.3.4. Modulul Features.....	33
<b>Capitolul 6. Testare și validare.....</b>	<b>44</b>
<b>Capitolul 7. Manual de instalare și utilizare .....</b>	<b>47</b>
7.1. Resursele necesare .....	47
7.1.1. Resurse hardware.....	47
7.1.2. Resurse software.....	47
7.2. Manual de instalare.....	47
7.3. Manual de utilizare .....	48
7.3.1. Manual de utilizare pentru client .....	50
7.3.2. Manual de utilizare pentru ospătar și bucătar .....	53
<b>Capitolul 8. Concluzii .....</b>	<b>53</b>
8.1. Contribuțiile proprii .....	54
8.2. Dezvoltări ulterioare .....	54
<b>Bibliografie .....</b>	<b>56</b>
<b>Anexa 1. Glosar de termeni .....</b>	<b>57</b>
<b>Anexa 2. Lista figurilor și tabelelor .....</b>	<b>58</b>
<b>Anexa 3. Exemple de cod .....</b>	<b>59</b>

## Capitolul 1. Introducere

Acest proiect își propune dezvoltarea și implementarea unei aplicații mobile de gestionare a rezervărilor și comenziilor 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 săcător 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 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 unui produse care, după 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 atât a clienților cât și a angajaților ce vor beneficia de el de-a lungul săptămânii.

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, poti 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 bibliotecă și să caute prin cărți informația dorită, acum persoana respectivă are acces la mult mai multe surse de informare, toate cuprinse în același dispozitiv. Pe măsură ce tehnologia avansează, dorim ca totul să aibă o aplicație cu scopul de a economisi cât mai mult timp și stres posibil și de a automatiza ceea ce este limitat de oameni pentru a maximiza eficiența și acuratețea.

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 apropiat.

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 aşteptare a unui ospătar, de preluare a comenzi și de cerere în achitarea notei de plată. Din punct de vedere a angajaților, aplicația ajută la fluidizarea stărilor comenziilor, acestea fiind trimise la bucătărie printr-o simplă atingere a ecranului, iar volumul de muncă este egal împărțit între toți chelnerii.

## Capitolul 2. Obiectivele proiectului

### 2.1. 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 vizualiza lista restaurantelor favorite și de a confirma sosirea.

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.

### 2.2. Obiective specifice

**Înregistrarea unor noi utilizatori:** Aplicația va permite înregistrarea noilor utilizatori pentru a îi lăsa să profite de ce are aceasta de oferit. Cu ajutorul acesteia, utilizatorul va fi capabil de a putea naviga între diferite ecrane, 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, aplicația va permite confirmarea adresei de email introdusă de către utilizator pentru a finaliza procesul de înregistrare și de a se asigura că într-adevar el este cel care dorește finalizarea acestui lucru.

**Vizualizarea restaurantelor pe hartă:** Restaurantele vor putea fi vizualizate pe harta integrată din aplicație. Acestea vor fi doar cele din proximitatea locației utilizatorului cu rol de client pentru a aproxima distanța ce trebuie parcursă în cazul în care la acel restaurant se dorește să se servească masa.

**Lista restaurantele:** Restaurantele vor putea fi vizualizate sub forma unei liste interactive pentru o structurare mai bună a interfeței utilizator și cu scopul de a reduce timpul de căutare a unui local. 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 aplicația dispune de afișarea meniului 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:** Aplicația oferă utilizatorului de tip client o listă în care acesta își poate vizualiza într-un mod rapid și ușor restaurantele favorite. Lista este localizată în pagina de profil a acestuia și are rolul de 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:** Aplicația oferă în plus o listă pe pagina de profil pentru a-i permite utilizatorului vizualizarea rezervărilor efectuate, fiecare afișând numărul de persoane, ora și ziua fiecărei.

**Comandă din aplicație:** Când utilizatorul o să confirme sosirea la restaurant, aplicația o să îl redirecționeze 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:** Se vor permite mai multe modalități de plată, acestea putând fi accesate după ce utilizatorul cere nota de plată în urma efectuării a cel puțin unei comenzi. Acest lucru ajută la eficiența atât a angajaților restaurantului cât și a clientului.

## **Capitolul 3. Studiu bibliografic**

### **3.1. Aplicații mobile de tipul food-delivery**

HORECA [1] 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 comenzi 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[2], 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 comenziilor 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 [3], Deliveroo [4] sau Glovo [5]. 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.

#### **3.1.1. Aplicația Zomato**

Potrivit unei statistică[6] 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 fondată de Deepinder Goyal și Pankaj Chaddah în anul 2008 [7]. În anul

2022, compania avea 3800 de angajați, iar veniturile erau approximate 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 comenziilor 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.

### 3.1.2. 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 comenziile și oferă utilizatorului un discount de 5% pentru orice comandă de peste 15 dolari.

Potrivit statisticii menționate și la subcapitolul anterior [8], 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 asemenea, 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 anumiți utilizatori preferă această aplicație este livrarea rapidă. UberEats folosește infrastructura și rețeaua logistică a aplicației Uber, care este cunoscută pentru serviciile sale eficiente de transport. Acest lucru le permite să livreze alimente rapid, adesea în 30 până la 60 de minute. Capacitatea de a primi rapid mancarea este un factor cheie în popularitatea UberEats, în special pentru persoanele cu program încărcat sau cei care preferă să mănânce acasă.

### 3.1.3. 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. [9]

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

### 3.1.4. 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
- **Interfață și experiență 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.

## 3.2. Software de administrare al restaurantelor

### 3.2.1. Toast POS

Toast este un software de administrare a restaurantelor fondat în anul 2012. Cu un număr de aproximativ 3172 de angajați, de 2,4 miliarde de dolari de venituri și 62.000 [10] 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-in-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 măsură 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.

### 3.2.2. 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 plățile rapid și în siguranță. Fie că 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 plășilor, Square POS oferă capabilităști robuste de gestionare a stocurilor. Companiile pot urmări cu ușurință nivelurile stocurilor, pot gestiona variașile 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 dispozișie produsele potrivite pentru a satisface cerinșele clienșilor.

### 3.2.3. Comparașie între aplicașiiile Toast și Square

Conform unui blog [11] publicat de Forbes în 2023, alegerea dintre Toast și Square ar trebui să fie făcută în funcșie de nevoia restaurantului de a folosi POS sau nu. Toast e o alegere excelentă pentru restaurantele cu mai multe locașii, drive-throughs, baruri pe când Square este o opșiune mai bună dacă se dorește un sistem POS pentru a ajuta cu verificarea, statusul de plată, scanarea codului de bare și altele. Acest lucru se datorează deoarece Toast are funcșionalităști 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 indică diferenșele esenșiale intre cele 2:

Tabel 3.1 - Diferenșe între Toast și Square

	<b>Toast</b>	<b>Square</b>
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 plășit	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
Relașii clienti	Suport telefonic la orice oră în orice zi, blog	Chat live, blog
Procesarea plășilor încorporată	Da	Da

## 3.3. Avantajele iOS comparativ cu Android

iOS, sistemul de operare a telefoanelor iPhone de la Apple [12], renumit pentru designul său elegant și interfașa grafică intuitivă a adunat o mulșime 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, interfașa utilizator diferă de la unul la altul substanșial, utilizatorii hotărându-se mai greu în a lua decizia ce telefon să își achizișioneze.

Conform unui blog [13], 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 limitaști la aplicașiiile de pe App Store Apple, lucru ce ajută la prevenirea descărcării de virușii. Pe de altă parte, utilizatorii de Android au acces la mai mulți distribuitorii de aplicașii, dar nu toate aplicașiiile oferite de aceștia sunt verificate.

Din punct de vedere a limbajelor de programare, aplicașiiile 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 atâtore tipuri de dispozitive ce au ecrane diferite, timpii de rulare diferă, rezoluția și multe altele.

Cu toate acestea, conform unei statistici [14] 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 lucru 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.

## **Capitolul 4. Analiză și fundamentare Teoretică**

### **4.1. 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 funcționalităților 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 vârstă 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. Bineîntele, după ce fiecare se autentifică fiecare, o să fie redirecționată 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 să 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:

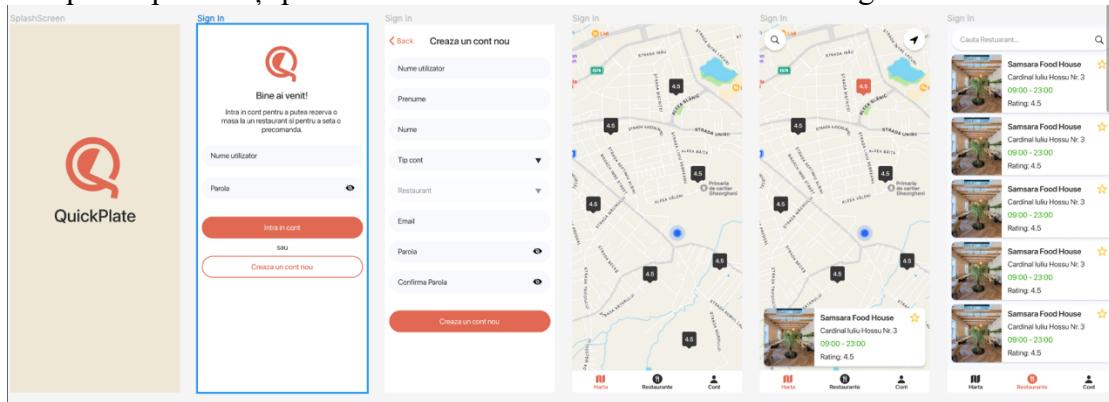
- Secțiunea în care observă datele personale și poate să se deconecteze din aplicație
- Secțiunea în care poate observa toate restaurantele favorite pe care le-a adăugat pe parcursul folosirii aplicației
- Secțiunea în care își poate vizualiza rezervările

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 comenzi

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.



Figură 4.1 - Design prototipizat în Figma

## 4.2. 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, aşadar 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 funcționalitățile 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 acesta 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. Aceasta trebuie să aleagă masă în funcție de numărul de clienți doriti și ora și ziua în care dorește să meargă.
- **Metodele de plată.** Clientului i se oferă posibilitatea de a plăti prin mai multe moduri cum ar fi cash, cu cardul sau folosind direct aplicația. Această funcționalitate ajută la economisirea timpului de aşteptare a chelnerului de a-l atenționa că se dorește plata comenzii.
- **Echilibrarea comenzi 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.

### **4.3. 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.** Această 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 rulează 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.

### **4.4. 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, comenzi 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 importante 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.

- **Comenziile:** Entitate 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 clientilor 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.

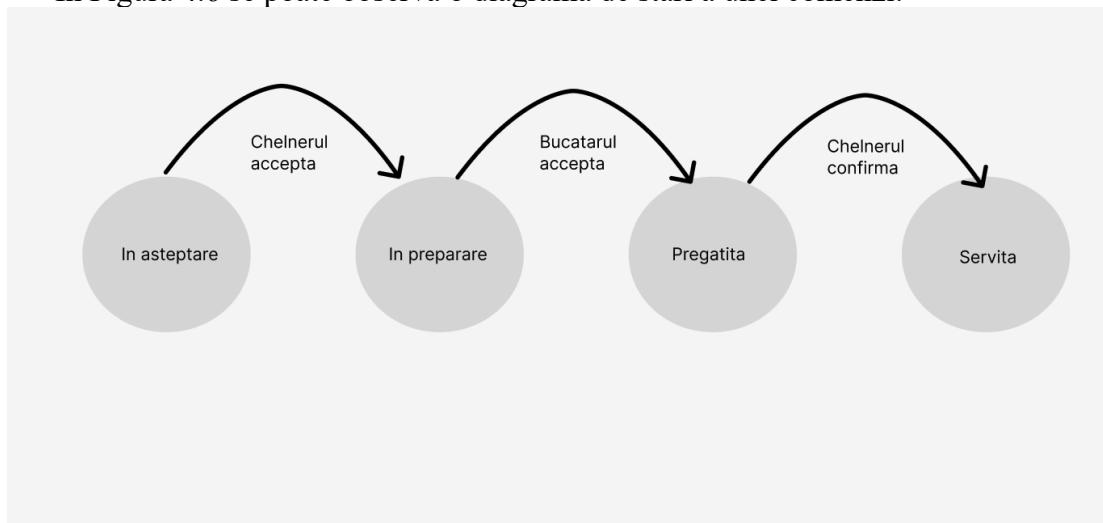
#### 4.5. Stările unei comenzi

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 comenzi
- **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 anulează în starea doi sau patru atunci va fi ștersă din Firestore.

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



Figură 4.2 - Diagramă de stări a unei comenzi

## **4.6. Cazuri de utilizare**

În această secțiune vor fi prezentate patru cazuri de utilizare a aplicației QuickPlate fiecare având o descriere și o diagramă 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 comenziilor, 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 comenziilor este singura acțiune pe care acesta o poate face.

### **4.6.1. Înregistrare**

Această acțiune este efectuată de fiecare utilizator ce are sau nu un cont în aplicația QuickPlate. Dacă 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:**

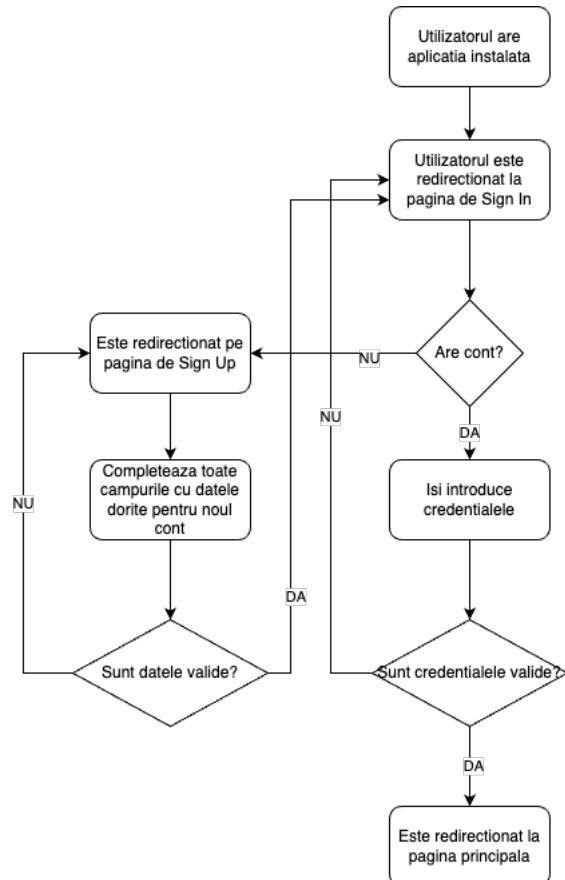
- Utilizatorul o să fie redirecționat la pagina principală a aplicației

**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



Figură 4.3 - Diagramă de flow pentru înregistrare

#### 4.6.2. Rezervarea unei mese

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

**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

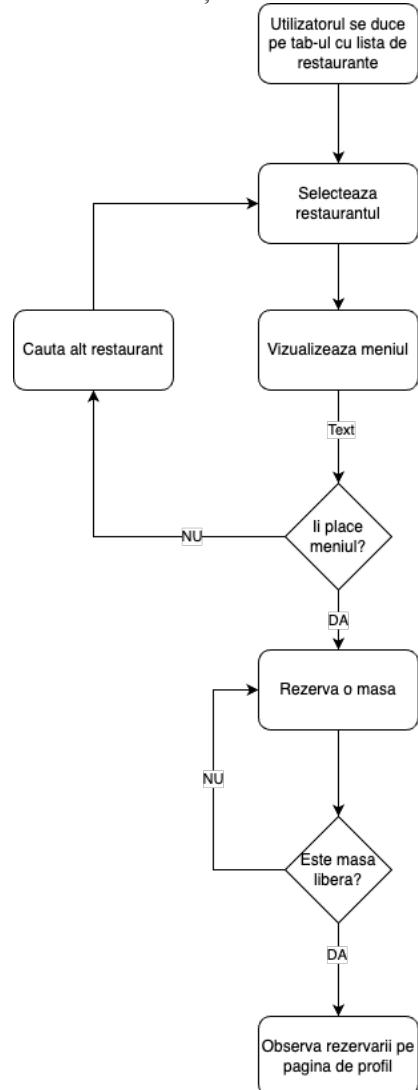
**Scenariul de succes:**

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

**Scenarii alternative:**

- Utilizatorul nu are o conexiune bună la internet și nu 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



Figură 4.4 - Diagramă de flow pentru rezervarea unei mese

#### 4.6.3. Comanda și plata consumației

Funcționalitate destinată utilizatorilor cu rol de client pentru a plasa o comandă în urma efectuării 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:**

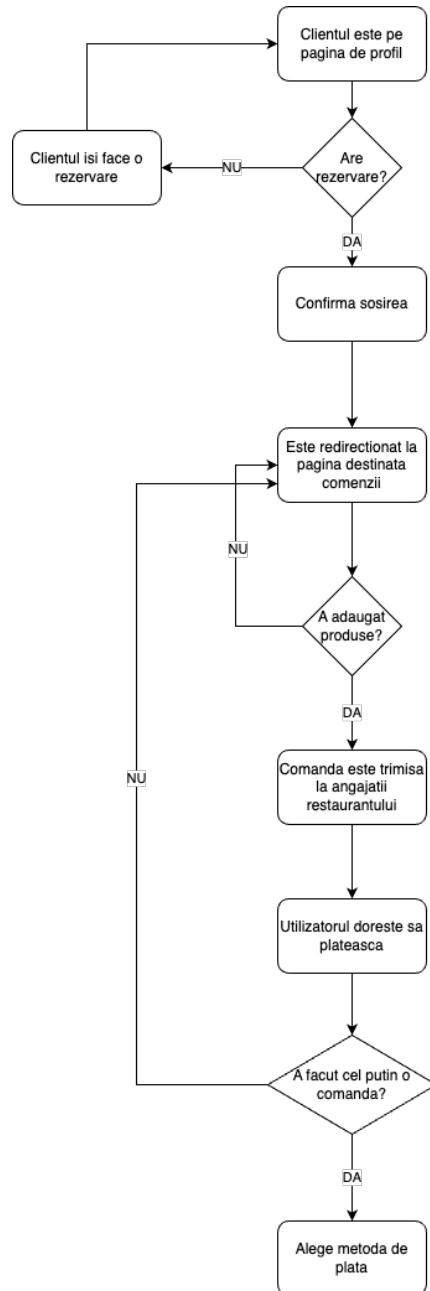
- Utilizatorul a reușit cu succes să selecteze metoda de plată dorită

**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 comenzi
- Utilizatorul adaugă produsele dorite pentru comandă
- Utilizatorul trimitе 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



Figură 4.5 - Diagramă de flow pentru comanda și plata consumației

#### 4.6.4. 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 după plasarea acesteia.

**Actor:** Angajații restaurantului

**Descriere:** Clientul trimită 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 livrează comanda la masă și confirmă livrarea din aplicație

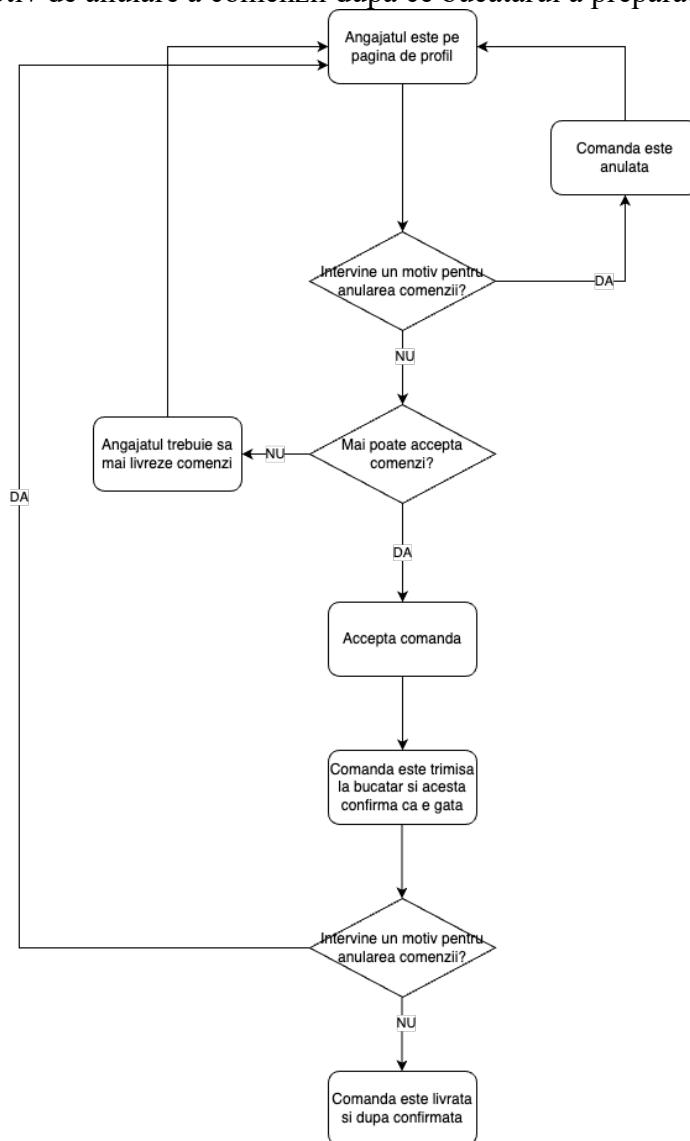
**Scenariul de succes:**

- 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 comenzi după ce bucătarul a preparat-o



Figură 4.6 - Diagramă de flow pentru procesul de comenzi

#### 4.7. Algoritmul de echilibrare a comenziilor per chelner

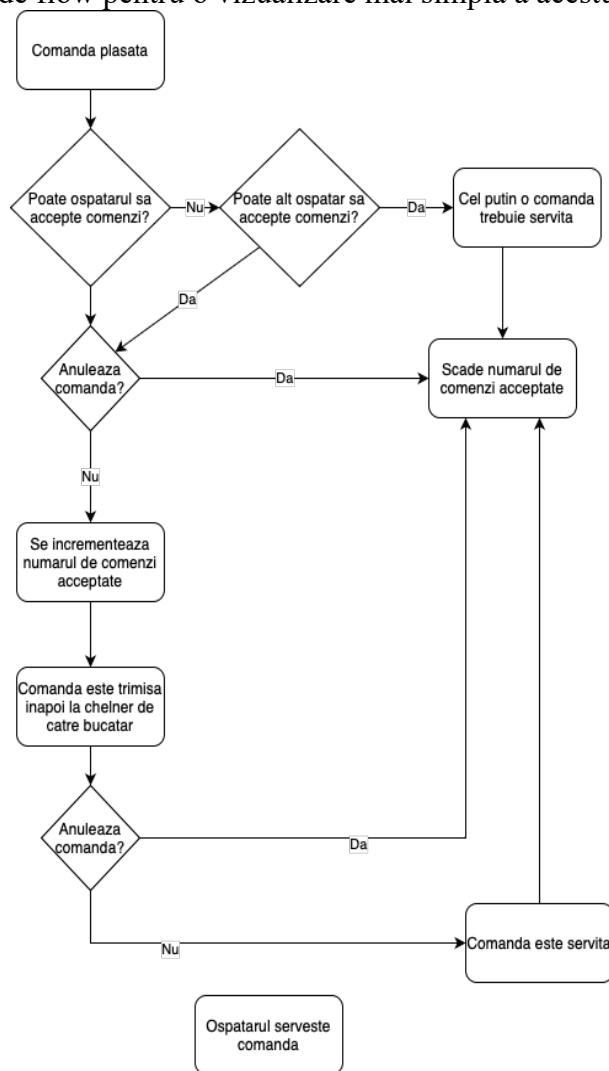
Aplicația urmărește, de asemenea, să faciliteze procesul de comandă și servire pentru restaurante, astfel încât ar trebui luat în considerare și potențialul volum mare de muncă al

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 dacă acest număr este depășit, acesta nu mai poate prelua alte comenzi până când comenziile deja acceptate nu sunt servite.

Există o variabilă în memoria telefonului care stocă 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 comenziilor, ospătarului îi este permis să treacă comanda din starea trei în patru și îi este interzisă trecerea comenzi 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.



Figură 4.7 - Diagramă de flow pentru gestionarea comenziilor per chelner

# Capitolul 5. Proiectare de detaliu și implementare

## 5.1. Proiectarea și implementarea arhitecturii

Aplicația a fost dezvoltată cu scopul de a fi folosită de un public cât mai larg, aşadar s-a optat pentru o arhitectură de tipul client-server datorită dezvoltării ușoare a scalabilității, a securității și a resurselor împărțite între mai mulți clienți. Acestea fiind spuse, aplicația iOS propriu-zisă reprezintă partea de client al proiectului, iar Firebase cea de server datorită serviciilor pe care le oferă precum autentificarea și autorizarea utilizatorilor, punerea la dispoziție unei baze de date NoSQL cu actualizări în timp real, reguli de restricționare asupra bazei de date și multe altele.

Pentru structurarea codului, s-a încercat pe cât mai mult posibil de profitarea folosirii framework-ului SwiftUI cu scopul de a ușura 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 încărcarea cu foarte multă logică pentru funcționalități a clasei cu rolul de Controller putând să se ajungă într-un punct foarte greu de dezvoltare sau menținere a clasei. Așadar, MVVM separă implementarea interfeței de restul aplicației astfel:

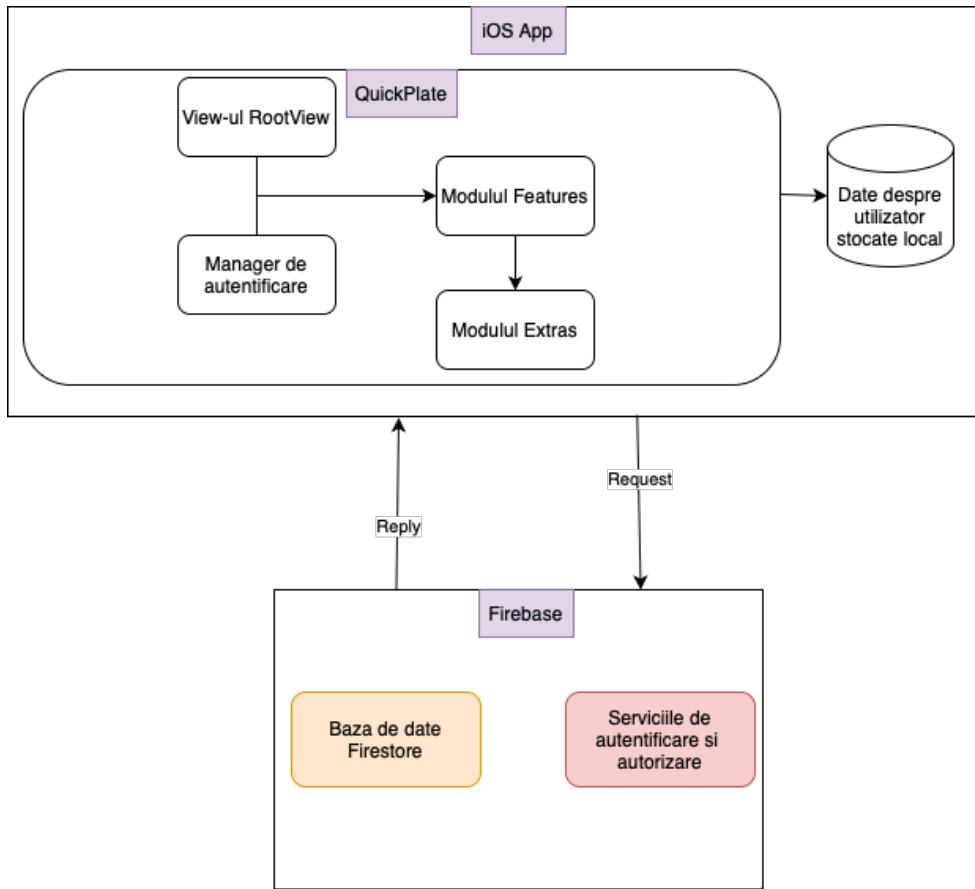
- View: reprezintă interfața utilizator. Poate fi un simplu buton, text, o grupare dintre acestea sau o grupare din mai multe View-uri.
- Model: reprezintă structura datelor ce se doresc a fi folosite în aplicație și corespund cu entitățile din baza de date.
- View-Model: clasa care se ocupă de actualizarea interfeței și modelarea clasei de Model, aceasta putând să restricționeze datele afișate și să arate strict ce este necesar interfeței 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 interfețe datorită separării logicii de date de logica de actualizare a interfeței utilizator, mai multe clase View putând să aibă același View-Model și oferirea flexibilității și extinderii claselor ușurând introducerea modificărilor sau adăugarea de noi funcții la aplicație.

Datorită faptului ca SwiftUI are implementată comunicare de tip Publish-Subscribe prin intermediul protocolului ObservableObject, actualizarea interfeței utilizator în urma unei modificări făcute în clasa View-Model corespunzătoare ei este una simplă întrucât dezvoltatorul nu mai trebuie să mai adauge schimbări în clasa de View.

De asemenea, din moment ce actualizările din baza de date a Firebase-ului sunt asincrone, framework-ul Combine este de mare ajutor întrucât ajută la sincronizarea datelor și actualizarea corectă a acestora în urma interacțiunii utilizatorului cu aplicația.

În Figura 5.1 se poate observa arhitectura aplicației QuickPlate.



Figură 5.1 - Diagrama aplicației QuickPlate

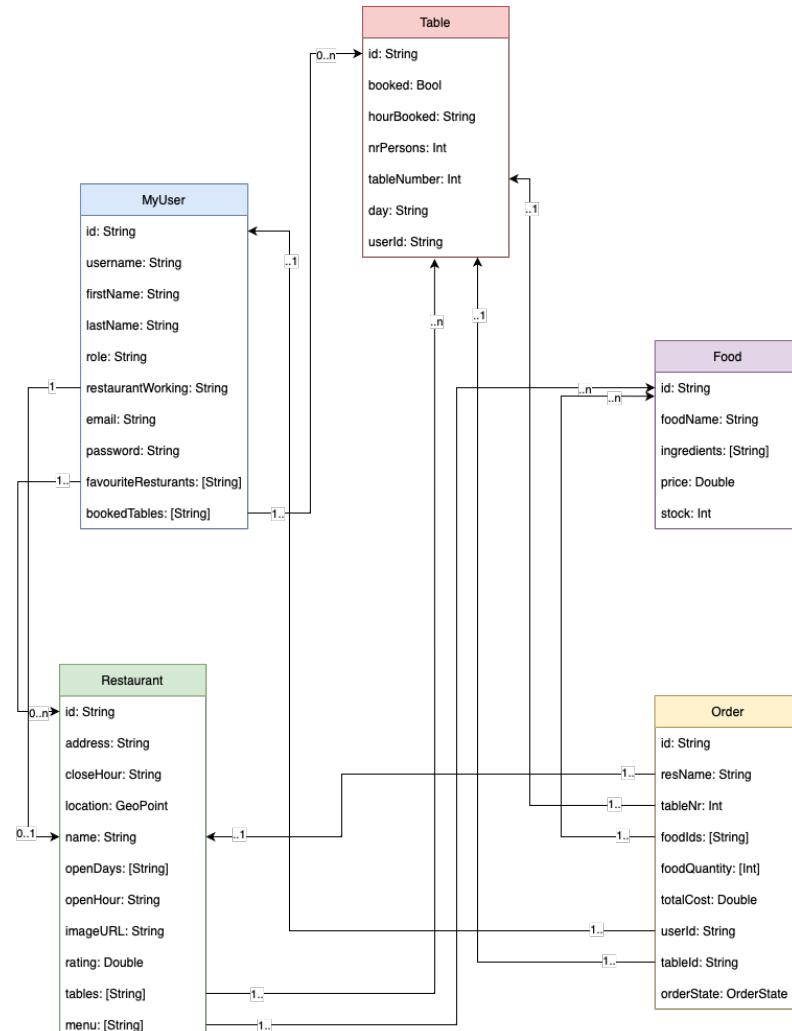
## 5.2. Proiectarea și implementarea bazei de date

Baza de date folosită de aplicație este Firestore, de tipul NoSQL și oferită de Firebase. S-a optat pentru această opțiune datorită actualizărilor în timp real a datelor în momentul efectuării unei modificări asupra unui model de date. Interacțiunea cu aceasta este una foarte ușoară deoarece dezvoltatorul tot ce a trebuit să facă a fost instalarea unor pachete speciale și adăugarea unui fișier denumit GoogleService-Info.plist ce conține configurații, acestea toate fiind necesare pentru conectarea la aplicația Firebase.

Din moment ce baza de date este una non-relațională, stocarea datelor se face sub forma unor documente de tip JSON. În acest tip de document, valorile sunt stocate asemănător unui dicționar pentru o navigare și scalabilitate mult mai ușoară. Firestore stochează datele sub forma unor colecții, echivalentul tabelelor a bazelor de date relaționale, ce rețin o ierarhie de documente. Cu toate acestea, orice document poate să conțină o colecție astfel creându-se un arbore de relații. Datorită acestui fapt, operațiile CRUD sunt mult mai rapide și eficiente comparativ cu cazul în care pentru aplicația QuickPlate s-ar fi folosit o bază de date relațională.

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 creării lor și proprietățile fiecăreia. Fiecare entitate este generată automat un UUID() de către Firebase.

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



Figură 5.2 - Relațiile entităților bazei de date

### 5.2.1. Entitatea Food

Această entitate a fost creată cu scopul de a reține date despre un anumit fel de mâncare. Este folositoare pentru a-i arăta clientului ce feluri de mâncare are un restaurant înainte de a își rezerva o masă, dar și de putea plasa o comandă când acesta confirmă sosirea la local. Este de menționat că un meniu este reprezentat sub forma unui tablou unidimensional format din id-urile felurilor de mâncare ce îl formează.

Această entitate nu are referințe la alte entități.

- **id**: UUID 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 două zecimale ce reprezintă prețul felului de mâncare
- **stock**: Număr întreg ce reprezintă numărul din inventar al acelui fel de mâncare

### 5.2.2. Entitatea MyUser

Aceasta este entitatea esențială aplicației întrucât reprezintă datele fiecărui utilizator al aplicației. Folosind datele acestuia se pot efectua rezervări, comenzi, autentificare, autorizare, deconectare, adăugarea de restaurante preferate, toate operațiile ce țin de utilizatorul unei aplicații. Fără aceasta, aplicația nu ar putea să funcționeze.

Pe de altă parte, această entitate poate să aibă referință de tipul one-to-many către entitatea de Restaurant datorită proprietății de favouriteRestaurants, același tip de referință către entitatea Table în cazul în care s-au făcut rezervări sau one-to-one către entitatea Restaurant în cazul în care utilizatorul este angajat aceluiași restaurant. Dacă utilizatorul este client și nu are restaurante preferate sau rezervări atunci el nu va avea nicio referință către o altă entitate.

- id: UUID ce reprezintă id-ul utilizatorului
- username: String ce reprezintă numele de utilizator pe care să îl folosească 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 următoarele două de ospătar respectiv bucătar
- restaurantWorking: String ce reprezintă numele restaurantului la care utilizatorul lucrează în cazul în care este ospătar sau bucătar, acest câmp fiind obligatoriu de completat la crearea unui nou cont. În cazul în care utilizatorul este client, proprietatea nu o să conțină nicio valoare.
- 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 conține id-urile meselor rezervate de către utilizator

#### 5.2.3. Entitatea Order

Această entitate este folosită doar în cazul în care un client dorește să plaseze o comandă. În momentul creării acesteia, o să aibă o referință către entitatea MyUser pentru a reține id-ul utilizatorului care a plasat comanda, către entitatea Table pentru ca ospătarii să știe la ce masă trebuie livrată comanda și nu în ultimul rând către entitatea Food pentru a știi ce conține comanda.

- id: UUID ce reprezintă id-ul comenzii
- resName: String care reține numele restaurantului la care s-a plasat comanda
- tableNr: Număr întreg care reprezintă numărul mesei de la care s-a plasat comanda. Folositoare pentru afișarea în interfață utilizator pentru angajați
- foodIds: Tablou unidimensional de string-uri cu id-urile felurilor de mâncare
- foodQuantity: Tablou unidimensional de numere întregi în care se reține cantitatea fiecărui fel de mâncare
- totalCost: Număr real care reține costul total al comenzii
- userId: String care reține id-ul utilizatorului care a plasat comanda
- tableId: String care reține id-ul mesei la care s-a plasat comanda. Folositoare pentru operațiile cu baza de date
- orderState: String ce reține starea în care se află comanda

#### 5.2.4. Entitatea Restaurant

Această entitate este cea mai des întâlnită pe parcursul experienței utilizatorului în folosirea aplicației. Aceasta este folosită pe pagina de harta, lista restaurantelor și de asemenea pe pagina de profil. Cu ajutorul ei utilizatorul poate observa restaurantele ori pe harta ori în lista, pagina de detalii a unui local, iar în cazul rolului de angajat locul unde lucrează.

Această entitate o să aibă referință de tipul one-to-many către entitatea Table pentru a știi ce mese țin de restaurantul respectiv și o referință de one-to-many către entitatea Food pentru a crea meniul cu care va interacționa utilizatorul.

- id: UUID ce reprezintă id-ul restaurantului
- address: String cu adresa restaurantului
- closeHour: String cu ora închiderii
- location: GeoPoint cu coordonatele geografice ale restaurantului
- name: String cu numele restaurantului
- openDays: Tablou unidimensional de string-uri cu zilele săptămânii când restaurantul este funcțional
- openHour: String cu ora deschiderii
- imageURL: String cu un URL către o imagine a restaurantului
- rating: Număr real ce reprezintă recenzia restaurantului cuprinsă în intervalul [0, 5]
- tables: Tablou unidimensional de string-uri cu id-urile meselor ce țin de acel restaurant
- menu: Tablou unidimensional de string-uri cu id-urile felurilor de mâncare pentru ce alcătuiesc meniul restaurantului

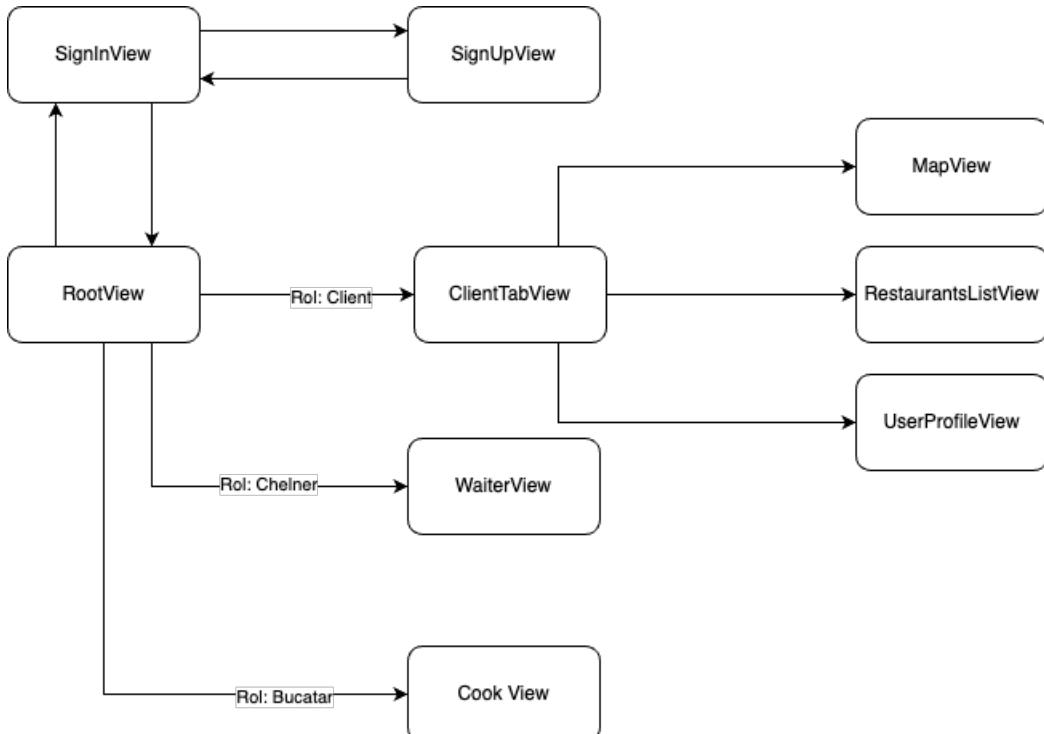
#### 5.2.5. Entitatea Table

Cea de a cincea și ultima entitate folosită în aplicația QuickPlate este Table care are rolul de a ajuta clientul să rezerve o masă la un restaurant dorit, de a putea observa detalii despre rezervarea făcută și să ajute ospătarii cu informațiile necesare pentru servirea comenzi. Aceasta entitate, ca și Food, nu are referințe către nicio alta entitate ci doar altele au referință către ea.

- id: String ce reprezintă id-ul mesei
- booked: Valoare booleană care indică dacă masa este sau nu rezervată. Adevărat pentru cazul în care este rezervată și fals când nu este.
- hourBooked: String care reprezintă ora de la care este făcută rezervarea. Nu are o valoare reținută dacă nu este rezervată.
- nrPersons: Valoare întreagă care indică numărul maxim de persoane care pot sta la masă
- tableNumber: Valoare întreagă pentru a arăta numărul mesei. Folositoare pentru ospătarii pentru a știi ce masă este rezervată
- day: String pentru a indica ziua în care este rezervată. Dacă nu este, atunci nu este nicio valoare.
- userId: String care reține id-ul utilizatorului care rezervă masa. Dacă nu este rezervată, atunci ea nu va stoca nicio valoare.

### 5.3. Structura proiectului

Pentru o înțelegere cât mai ușoară a claselor principale folosite, figura 5.3 prezintă o diagramă cu cele mai importante View-uri, unele afișându-se doar în funcție de rolul utilizatorului.



Figură 5.3 - Diagrama claselor principale

Fiecare clasă prezentă în figura anteroară v-a fi descrisă în detaliu în următoarele capitole.

#### 5.3.1. Structura fișierelor

În acest subcapitol voi prezenta structura fișierelor și directoarelor proiectului aplicației QuickPlate. Pe parcursul dezvoltării proiectului s-a urmărit denumirea cât mai intuitivă a acestora pentru o ușoară reutilizare a lor și o înțelegere cât mai simplă a legăturilor dintre ele. Directoarele Features, LoginStates și Extras vor fi explicate mai în detaliu în următoarele subcapitole.

Fișierul Lozalizable.strings stochează mai multe șiruri de caractere sub formă de perechi de tipul cheie-valoare pentru fiecare limbă acceptată. Aplicația QuickPlate suportă la momentul actual doar limba română și engleză. Acest fișier ar trebui creat de dezvoltator și configurat cu toate limbile ce se doresc să fie suportate. LocalizedStringKey() este utilă pentru folosirea acestui fișier deoarece acest tip reprezintă o punte între cod și textul afișat. Este de obicei folosit cu componenta Text ca în exemplul următor: `Text(LocalizedStringKey(''welcome''))` afișează valoarea corespunzătoare cheii „welcome” în funcție de limba setată pe dispozitivul utilizatorului.

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

```

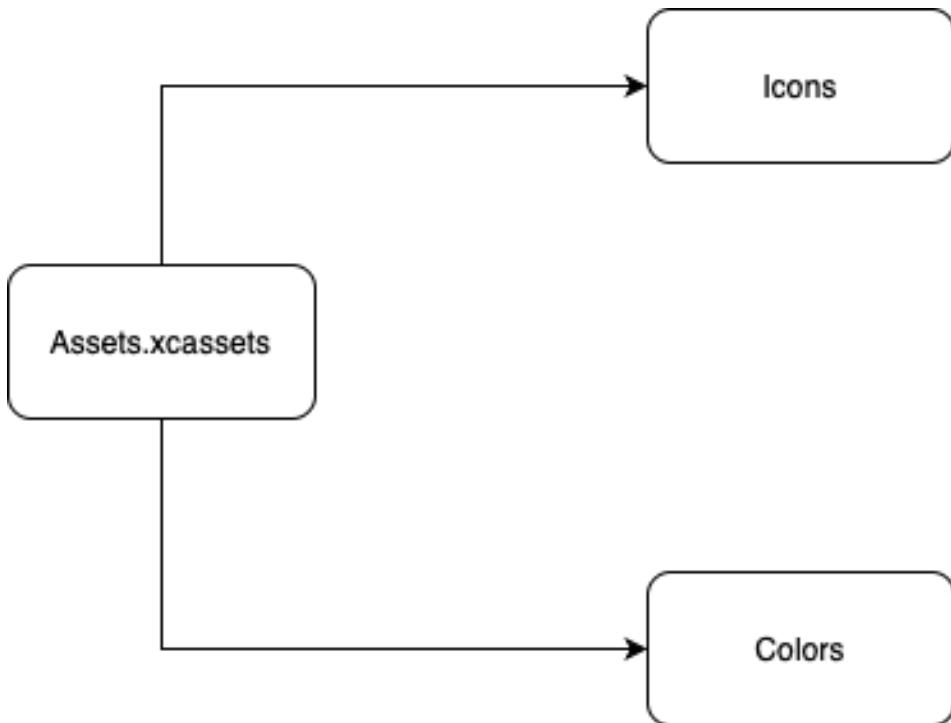
/* Localizable.strings (en)
 *
 * Localizable.strings
 * QuickPlate
 *
 * Created by Ioan-Octavian Stanciu on 02.01.2023.
 */
// MARK: GENERAL
"confirm" = "Confirm?";
"yes" = "Yes";
"no" = "No";
"cancel" = "Cancel";
// MARK: RESTAURANTS LIST
"search-placeholder" = "Search for restaurant...";
// MARK: SIGN IN
"welcome" = "Welcome!";
"signin-message" = "Sign in so you can book a table at your favourite
                    restaurant and to be able to pre-order";
"email" = "Email";
"password" = "Password";
"signin" = "Sign In";
"or" = "or";
"signup" = "Create an account";
"logout" = "Sign Out";
"credentials-error" = "Wrong credentials / Account doesn't exist";
// MARK: SIGN UP
"username" = "Username";
"firstName" = "First Name";
"lastName" = "Last Name";
"accountType" = "Account Type";
"confirm-password" = "Confirm Password";
"password-format-error" = "Password must have at least 6 characters";
"passwords-match-error" = "The passwords don't match";
*/
// MARK: SIGN IN
"welcome" = "Bine ai venit!";
"signin-message" = "Intra in cont pentru a putea rezerva o masa la un
                    restaurant și pentru a seta o precomanda";
"email" = "Email";
"password" = "Parola";
"signin" = "Intra in cont";
"or" = "sau";
"signup" = "Creeaza un cont nou";
"logout" = "Deconectare";
"credentials-error" = "Credentiale gresite / Cont inexistent";
// MARK: SIGN UP
"username" = "Nume utilizator";
"firstName" = "Prenume";
"lastName" = "Nume";
"accountType" = "Tipul contului";
"confirm-password" = "Confirma parola";
"password-format-error" = "Parola trebuie sa contina cel putin 6
                    caractere";
*/

```

Figură 5.4 - Fișierul Localizable.strings

Fișierele cu extensiile .plist, Info.plist respectiv GoogleService-Info.plist, sunt fișiere necesare de configurații, primul fiind pentru aplicația QuickPlate, iar cel de al doilea pentru conectarea la aplicația Firebase. Primul fișier este generat de Xcode, IDE-ul în care s-a dezvoltat aplicația, pe când cel de al doilea a fost descărcat când s-a creat proiectul de Firebase. Metadatele din aceste două fișiere sunt reținute sub aceeași formă ca și în fișierul Localizable.strings și anume perechi de tipul cheie-valoare.

Assets.xcassets este generat automat de către Xcode și este un pachet special care acționează ca o locație centralizată pentru gestionarea fișierelor de imagine, iconițe și culori. Este folosit pentru a organiza și stoca resursele vizuale ale aplicației. În figura 5.5 se poate observa cum arată acest pachet.



Figură 5.5 – Diagrama fișierului Assets.xcassets

Fișierul QuickPlateApp.swift este cel care face posibilă rularea aplicației. Ea deține echivalentul metodei de main() din alte limbaje de programare. De asemenea, în aceasta mai este o clasă denumită AppDelegate în care se suprascrie metoda didFinishLaunchingWithOptions: care se apelează în momentul în care aplicația s-a terminat de pornit. S-a optat pentru această alegere pentru a putea începe configurarea aplicației Firebase, în metoda menționată anterior fiind apelată Firebase.configure().

RootView.swift deține rolul ca fiind cea mai importantă clasă a aplicației întrucât în funcție de regulile de autentificare aceasta afișează interfața utilizator destinate rolului utilizatorului. La început afișează o scurtă animație cu un ecran de introducere după care se verifică urmatoarele:

- Dacă utilizatorul nu este autentificat atunci el va fi redirecționat către ecranul destinat funcționalității de autentificare
- Dacă utilizatorul este autentificat și are rol de client atunci o să se afișeze ecranul cu funcționalitățile destinate clientului
- Dacă utilizatorul este autentificat și are rol de ospătar atunci o să fie redirecționat către pagina destinată acestui tip de utilizator
- Dacă utilizatorul este autentificat și are rol de bucătar atunci o să îi fie afișat ecranul destinat acestui rol

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

### 5.3.2. Modulul Extras

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

Singurul fișier care nu este cuprins într-un modul separat este FSCollNames.swift care conține un simplu Enum, ce moștenește din tipul de dată String, cu numele colecțiilor din baza de date destinate entităților de date. Acestea pot fi users, restaurants, tables sau orders.

Fiecare caz îi este atribut un sir de caractere ce corespunde cu numele colecției corespunzătoare 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 evita confuzia dintre numele entităților și numele colecțiilor.

Submodulul *FirestoreOp* conține cinci clase de tipul singleton (sunt instantiată o singură dată și aceeași instanță este folosită în întreaga aplicație) pentru a ajuta la interacțiunea cu Firestore. Acestea au fost create cu scopul de a se executa operațiile CRUD asupra datelor stocate în Firestore și de a separa această logică de cod față de restul aplicației. Toate au fost denumite sugestiv, fiecare având un prefix "FS", un acronim pentru Firestore, urmată de numele colecției asupra căreia se execută operațiile și un sufix "Coll" care reprezintă o abreviere a cuvântului *Collection*.

Cu toate acestea, a fost creată o clasă extra, *FirebaseEmailAuth*, de tipul singleton, dar are alt scop și anume de a se ocupa de partea de autentificare, înregistrare și deconectare a aplicației. Aceasta conține definirea și implementarea funcțiilor esențiale pentru a efectua metodele de Sign In/Login, Sign Up/Register și Sign Out/Logout. Pentru primele două particularități se verifică și dacă adresa de email a fost confirmată, respectiv dacă este deja în folosință de alt cont. În figura 5.6 se poate observa un enum denumit *StartupError* ce moștenește tipul de date *Error* și are cinci cazuri de erori:

- *signInError*: Apare atunci când utilizatorul nu a reușit să se autentifice datorită credențialelor introduse greșit sau folosirea unei adrese de email ce a fost utilizată la crearea unui alt cont
- *signUpError*: Eroare ce apare la înregistrare datorită completării greșite a unor câmpuri sau la omisiunea în totalitate a acestora. De asemenea, reintroducerea unui nume de utilizator deja în folosință poate să facă această eroare să apară
- *anonymousUser*: Eroare ce apare în momentul în care un utilizator încearcă să se autentifice sau să se înregistreze. În acest caz rezultatul returnat de către Firebase este o entitate de utilizator anonim. În aplicația QuickPlate, acest caz este considerat ca fiind 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 încearcă să se autentifice sau să se înregistreze, iar adresa de email este deja în folosință de către alt cont

```

enum StartupError: Error {
    case signInError
    case signUpError
    case anonymousUser
    case emailExists
}

class FirebaseAuth {
    static let shared = FirebaseAuth()

    // MARK: AUTHENTICATION METHODS

    func doLogin(email: String = "", password: String = "", completion: @escaping (Result<String, StartupError>) -> Void) {
        Auth.auth().signIn(withEmail: email, password: password, completion: { result, error in
            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(user.uid))
            case false:
                print("Email is not verified")
                completion(.failure(.emailExists))
            }
        })
    }
}

```

Figură 5.6 - Funcția de login a clasei FirebaseAuth

De asemenea, toate metodele din această clasă returnează un *closure* (bucată de cod executată la finalul unei acțiuni) pentru a putea trata cazurile de succes, respectiv de eșec prin completion-urile returnate. În caz de succes se returnează UUID-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 în totalitate cu descrierilor făcute la subcapitolul menționat și reprezintă conținutul datelor ce sunt stocate în întreaga aplicație.

În modelul Restaurant există în plus două structuri ce reprezintă DTO-uri folosite pe pagina de Sign Up și când se dorește afișarea mai sumară a informațiilor despre un restaurant. Pentru cazul de register se folosește RestaurantSignUpDTO, iar pentru cel de al doilea RestaurantCardDTO. Acestea permit transferul mai ușor a datelor și mai rapid din punct de vedere computațional. De asemenea, entitatea Restaurant mai are pe lângă proprietățile ce corespund cu cele din baza de date, două proprietăți de tipul *computed* pentru o ușoară prelucrare a proprietăților de openHour și closeHour.

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

În interiorul fișierului Order.swift este definit un enum ce reprezintă stările unei comenzi menționate și în capituloane 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 aşa mai departe).

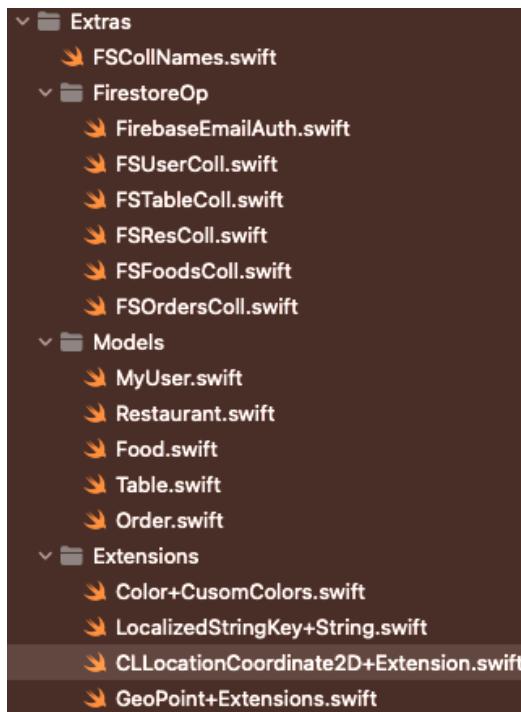
Ultimul submodul, Extensions, conține clase ce reprezintă extensii a mai multor tipuri de date din limbajul Swift.

- Color+CustomColor.swift: este creată o extensie la structura predefinită *Color* pentru a adăuga culorile personalizate din pachetul Assets.xcassets. Fiecare

culoare este definită ca o constantă statică pentru a putea fi accesată de oriunde din aplicație în orice moment.

- LocalizedStringKey+String.swift: sunt două extensii, una pentru LocalizedStringKey și cealaltă pentru String pentru a ușura localizarea sirurilor de caractere pe tot parcursul aplicației.
- CLLocationCoordinate2D+Extension.swift: O extensie la structura CLLocationCoordinate2D pentru a putea fi encodată, decodată și comparată egalitatea cu o structură de tipul GeoPoint. Pentru această ultimă funcționalitate a trebuit crearea unei funcții denumită isEqualTo ce primește ca parametru o variabilă de tipul GeoPoint și returnează o valoarea booleană în cazul în care coordonatele longitudinale sunt egale între ele, respectiv latitudinale.
- GeoPoint+Extensions.swift: O extindere la tipul de date GeoPoint pentru a crea o funcție ce returnează o structură de tipul CLLocationCoordinate2D având aceleași coordonate cu parametrul de tipul GeoPoint.

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



Figură 5.7 - Structura modulului Extras

### 5.3.3. Modulul LoginStates

În acest modul se află un singur fișier, AuthManager, care conține cea mai importantă clasă a acestui proiect deoarece se ocupă de logica de autentificare și este de asemenea View-Model-ul menționat la 5.3.1.

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

- notSignedIn: caz în care utilizatorul nu este autentificat și îi 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 proprietăți a clasei AuthManager și în funcție de valoarea pe care o stochează se decide ce ecran trebuie afișat utilizatorului. Metoda de

checkLoginUserDefaultsExist() se apelează la inițializarea clasei și verifică dacă în memoria telefonului este deja reținută valoarea proprietății respective. Dacă nu există atunci se adaugă cu cazul de baza notSignedIn, iar în caz contrar se apelează metoda updateWith(state: LoginStateEnum) care salvează noua valoare în UserDefaults.

AuthManager este instantiat în clasa RootView, care acesta are în interiorul ei o instrucțiune switch pentru verificarea valorii proprietății nextScreen, definită în AuthManager, în care se face schimbarea interfeței utilizator în funcție de aceasta. Cazurile sunt:

- .notSignedIn: Se afișează view-ul SignInView(). Se poate ajunge după instalarea aplicației sau după ce utilizatorul se deconectează din cont.
- .clientSignedIn: Se afișează view-ul QPTabView(). Se poate ajunge doar după ce utilizatorul se autentifică și are rolul de client.
- .waiterSignedIn: Se afișează view-ul WaiterView(). Se poate ajunge doar după ce utilizatorul se autentifică și are rolul de chelner.
- .cookSignedIn: Se afișează view-ul CookView(). Se poate ajunge doar după ce utilizatorul se autentifică și are rolul de bucătar.

Indiferent de caz, clasa AuthManager o să fie disponibilă pentru toate View-urile folosind modificatorul .environmentObject pentru ca fiecare clasă să aibă acces la logica de schimbare a ecranului (folositoare la Sign In și Sign Out). Pentru a avea acces la AuthManager într-unul dintre View-urile SignInView, QPTabView, WaiterView și CookView, trebuie declarată o proprietate de tipul AuthManager și adnotată cu @EnvironmentObject fără a fi inițializată.

Explicat mai sumar, în momentul în care se apelează metoda de updateWith(state:) de oriunde din cele patru View-uri enumerate mai sus, valoarea proprietății nextScreen se schimbă, se salvează în UserDefaults, se reevaluatează valoarea prin intermediul instrucțiunii switch și se schimbă ecranul afișat.

În figura 5.8, în stânga este View-ul RootView în care se află instrucțiunea switch, în mijloc este fișierul AuthManager ce conține clasa AuthManager și enum-ul LoginStateEnum, iar în dreapta, la linia 13, se poate observa declararea proprietății authManager de tipul AuthManager pentru a avea acces la funcționalitățile ei.

```

// Created by Ioan-Octavian Stanciu on
15.03.2023.

// import SwiftUI

import SwiftUI

struct RootView: View {
    @StateObject var authManager = AuthManager()
    @State private var isShowingSplash = true
    var body: some View {
        ZStack {
            if isShowingSplash {
                SplashScreenView(showing: $isShowingSplash)
            } else {
                switch authManager.nextScreen {
                case .notSignedIn:
                    NavigationStack {
                        SignInView()
                    }
                case .clientSignedIn:
                    QPTabView()
                case .waiterSignedIn:
                    WaiterView()
                case .cookSignedIn:
                    CookView()
                }
            }
        }.environmentObject(authManager)
    }
}

struct RootView_Previews: PreviewProvider {
    static var previews: some View {
        RootView()
    }
}

// 3. The user signed in as a worker => nextScreen is
// .workerSignedIn => will display the WorkerView()

enum LoginStateEnum: String {
    case notSignedIn, clientSignedIn, waiterSignedIn,
        cookSignedIn
}

final class AuthManager: ObservableObject {
    @Published var nextScreen: LoginStateEnum = .notSignedIn

    init() {
        checkLoginUserDefaultsExist()
    }

    func checkLoginUserDefaultsExist() {
        if UserDefaults.standard.object(forKey: "login") == nil {
            UserDefaults.standard.set(LoginStateEnum.notSignedIn.rawValue, forKey: "login")
        } else {
            guard let cachedLoginState =
                UserDefaults.standard.string(forKey: "login") else {
                print("DEBUG - LoginManager - Error in reading the cached state")
                return
            }
            if cachedLoginState != UserDefaults.standard.string(forKey: "login") {
                print("DEBUG - LoginManager - Error in getting the new state conversion")
                return
            }
            guard let newState = LoginStateEnum(rawValue: cachedLoginState) else {
                print("DEBUG - LoginManager - Error in getting the new state conversion")
                return
            }
            updateWith(state: newState)
        }
    }

    func updateWith(state: LoginStateEnum) {
        nextScreen = state
        if (state == .notSignedIn) {
            UserDefaults.standard.set(nextScreen.rawValue, forKey: "login")
        }
    }
}

// Created by Ioan-Octavian Stanciu on 21.11.2022.

import SwiftUI

private var tags: [String] = [LocalizedStringKey("booked-tables").stringValue(),
    LocalizedStringKey("favourite-restaurants").stringValue()]

struct UserProfileView: View {
    @EnvironmentObject var authManager: AuthManager
    @StateObject private var vm = UserProfileViewModel()
    @State private var selectedReservation = Table()

    @State private var isShowingCancelBooking: Bool = false
    @State private var isShowingSignInOutAlert: Bool = false
    @State private var activeTag: String = tags[0]
    @State private var confirmedArrival: Bool = false
    @State private var confirmedBookingTableId: String = ""

    @Namespace private var animation

    var body: some View {
        VStack {
            TopSection()

            TagsView()
                .padding(.vertical, 10)

            switch activeTag {
            case LocalizedStringKey("booked-tables").stringValue():
                BookedTablesView()
            case LocalizedStringKey("favourite-restaurants").stringValue():
                FavouriteRestaurantsView()
            default:
                EmptyView()
            }
        }
        .frame(maxWidth: .infinity)
        .sheet(isPresented: $confirmedArrival) {
            NavigationView {
                ClientOrderView(tableId: self.$confirmedBookingTableId)
            }
        }
        .alert(LocalizedStringKey("cancel-reservation"), isPresented: $isShowingCancelBooking) {
            Button(LocalizedStringKey("yes"), role: .cancel) {
                vm.cancelBookingForTableWith(tableId: self.selectedReservation.id ?? "")
            }
            self.isShowingCancelBooking.toggle()
        }
    }
}

```

Figură 5.8 - Utilizarea clasei AuthManager

#### 5.3.4. Modulul Features

Modulul Features este cel cu cele mai multe subdirectoare și clase ce sunt folosite în întregul proiect. Se poate spune că acesta este centrul aplicației QuickPlate întrucât majoritatea funcționalităților sunt cuprinse în acest loc.

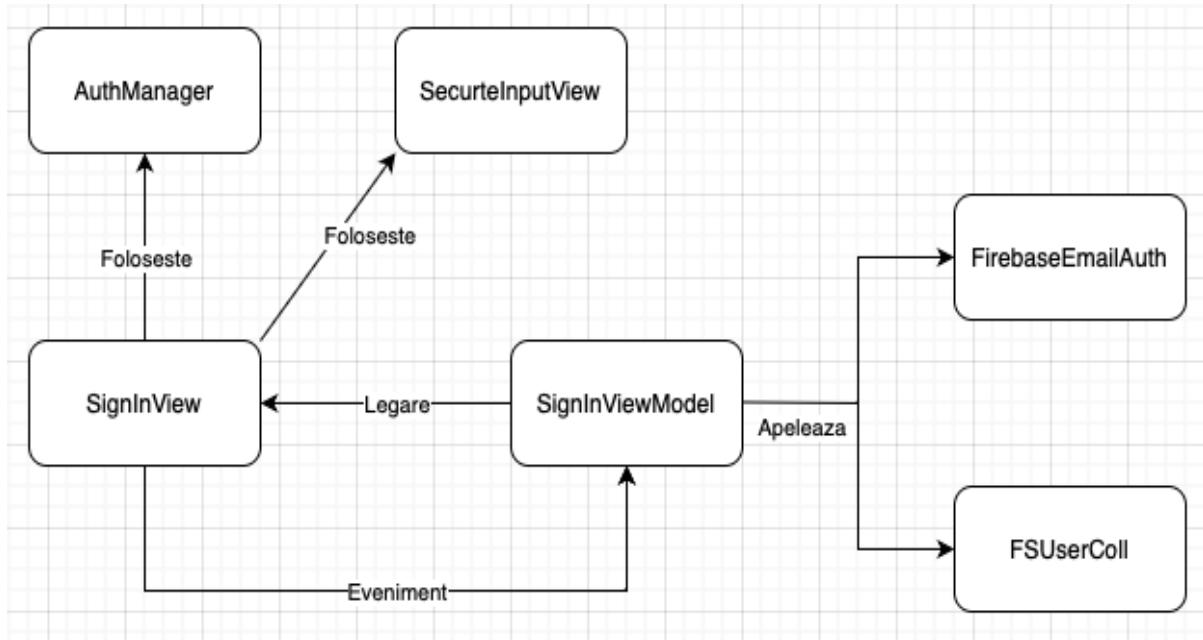
Primul modul este SplashScreen cu un singur fișier denumit SplashScreenView care este View-ul ce se afișează la începutul aplicației de fiecare dată cu o animație de două secunde prin intermediul folosirii modificadorului onAppear() ce face posibilă detecția apariției View-ului în ierarhie. Acest View este folosit mereu la inițializarea clasei RootView indiferent de starea autentificării utilizatorului. În această clasă se poate observa folosirea unei culori din fișierul Assets.xcassets folosind structura Color având ca parametru numele culorii.

Al doilea modul intitulat SignIn conține logica de autentificare și conține două fișiere SignInView și SignInViewModel, iar pe lângă acestea un alt modul „Custom View + Modifiers” care la rândul lui este alcătuit 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 (afișează cercuri în locul literelor), acest lucru fiind posibil prin folosirea unei instrucțiuni if pentru a verifica valoarea booleană stocată în variabila isSecured, și SignInTextFieldModifier care este un modificator ce se poate aplica asupra oricărui View pentru a-l stiliza. Un modificator este o funcție care modifică și personalizează aspectul sau comportamentul unui View. Acestea sunt folosite în View-ul SignInView cu logica sa separată în SignInViewModel.

În SignInViewModel, în funcția signIn, este implementată logica de autentificare a unui utilizator fiind apelată metoda doLogin() din clasa FirebaseAuth(), iar ca rezultatul returnat de către completion este tratat într-o instrucțiune switch. Pentru cazul de eșec, se trimit din nou un completion(.failure) indicând dacă această acțiune a eșuat, iar pentru cazul de succes se setează în memoria telefonului id-ul utilizatorului, se aduce din baza de date utilizatorul ce are acel identificator, iar cu ajutorul unei instrucțiuni if se verifică ce caz din LoginStateEnum se salvează:

- Utilizatorul este client: trebuie salvat doar starea autentificării și anume clientSignedIn
- Utilizatorul este ospătar: se salvează numele restaurantului unde lucrează și cazul waiterSignedIn
- Utilizatorul este bucătar: se salvează numele restaurantului la care este angajat și cazul cookSignedIn

În figura 5.9 este prezentată diagrama de clase a acestui modul.



Figură 5.9 - Diagramă de clase a modulului SignIn

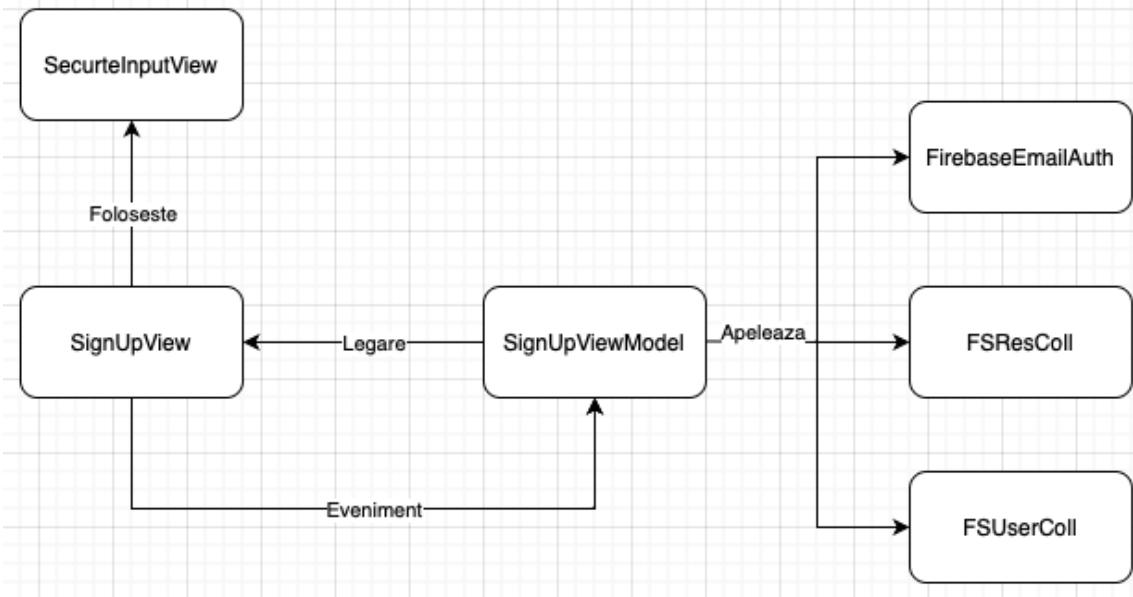
Cel de al treila modul conține funcționalitatea creării unui cont nou de către orice utilizator. Are doar două fișiere, SignUpView și SignUpViewModel, primul reprezentând interfața grafică, iar cel din urmă logica codului.

Acest View-Model are multe proprietăți @Published pentru a se actualiza în timp real când utilizatorul interacționează cu interfața grafică. Câteva exemple sunt username, firstName, lastName, password. În momentul în care utilizatorul apasă pe butonul de creare cont, trebuie mai întâi verificat dacă s-au completat toate câmpurile obligatorii, username, firstName, lastName, email și password, logica fiind implementată în metoda allFieldsAreCompleted() ce returnează o valoare booleană. Dacă pe proprietatea de role nu se selectează nimic, atunci se va considera că utilizatorul este client, iar câmpul de selecție a restaurantului unde este angajat o să fie dezactivat.

Posibilele erori ce pot să apară sunt urmatoarele:

- Utilizatorul nu a completat toate câmpurile
- Utilizatorul a introdus o adresă de email ce este deja asociată unui alt cont
- Utilizatorul a introdus două parole diferite
- Utilizatorul nu a respectat formatul parolei care trebuie să conțină minim șase caractere

În figura 5.10 este prezentată diagrama de clase a acestui modul.



Figură 5.10 - Diagrama de clase a modulului SignUp

Următoarele trei directoare, Client, Waiter și Cook, sunt destinate rolurilor de client, ospătar, respectiv bucătar, ultimele două fiind foarte asemănătoare singura diferență fiind filtrarea datelor ce se afișează. Prima oară o să fie prezentate directoarele ce aparțin modulului Client după care directoarele Waiter și Cook vor fi detaliate în același timp evidențiind miciile diferențe între ele.

### 1. Modulul TabView

Deține 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 însă nu se ocupă de nicio logică a unei funcționalități al aplicației și se află în zona de jos al ecranului dispozitivului. Pe metoda de init(), apelată la instanțierea clasei, se setează culorile de bază ca fiind portocaliu pentru text și poze, iar alb pentru ce se vede în spate.

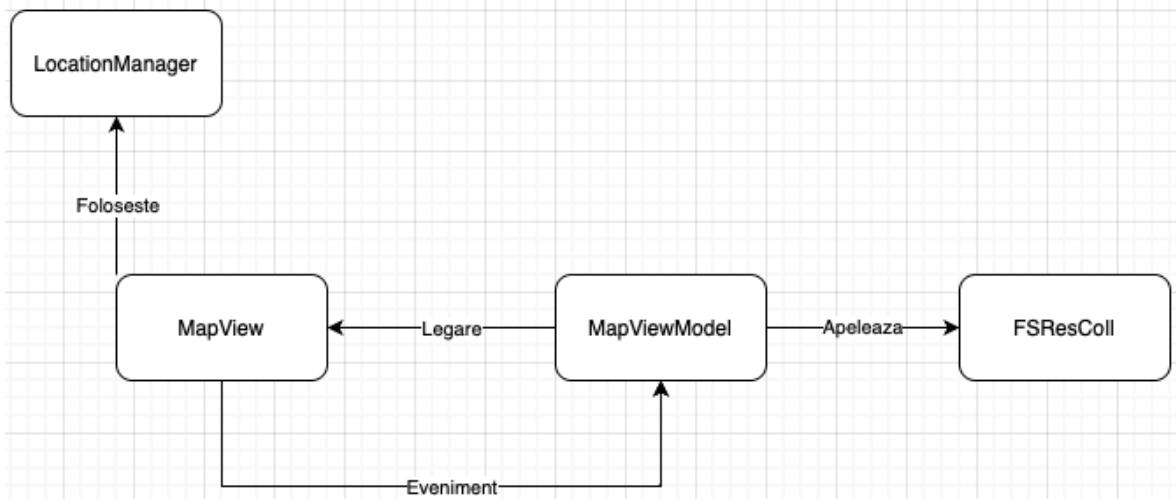
### 2. Modulul Map

Contine logica și interfața utilizator pentru partea de harta din QPTabView. Cu ajutorul framework-ului MapKit dezvoltat de Apple, integrarea unei hărți în aplicație a fost una foarte simplă. În clasa MapView se folosește un View din acest framework, Map, căruia îi trebuie trimiși anumiți parametrii pentru a avea acces la locația utilizatorului în timp real și pentru a arăta sau nu locația acestuia pe hartă. Harta conține în partea de sus un buton interactiv care atunci când este apăsat centrează harta pe coordonatele utilizatorului. În momentul în care se apasă acel buton, se apelează metoda de startUpdatingLocation() din locationManager pentru a începe actualizarea locației utilizatorului. Atunci când se detectează o nouă poziție, se apelează automat metoda didUpdateLocation din clasa LocationManager care verifică cu ajutorul unei instrucțiuni if dacă ultima locație (coordonatele fiind reținute în proprietatea lastLocation) diferă de ultima valoare din tabloul unidimensional locations (parametru al acestei metode care reține coordonatele ultimei locații actualizate). Dacă diferă, atunci se actualizează regiunea cu noile coordonate detectate, se actualizează proprietatea lastLocation și se oprește actualizarea continuă a locației utilizatorului prin apelul metodei stopUpdatingLocation().

Mai conține și clasa LocationManager ce se conformează la protocolul CLLocationManagerDelegate, iar acesta se ocupă de logica menționată în paragraful anterior. Delegate este un model de design de baza în aplicațiile iOS întrucât multe funcționalități predefinite trebuie să se conformeze la asemenea protocole. Acesta are un delegat, un obiect,

care conține o metodă sau un set de metode pe care obiectul delegat le poate implementa, și obiectul delegat este cel care implementează metodele definite de delegat. Obiectul delegat este înregistrat sau asignat către delegat, permitându-i să primească și să gestioneze evenimente sau să furnizeze anumite funcționalități. În cazul acesta, atât delegatul cât și obiectul delegat este LocationManager deoarece este folosit sub tipul unui View-Model, iar toate schimbările din acesta să fie publicat în View-ul MapView.

Pe lângă acestea, pe hartă sunt afișate restaurantele aplicației, fiecare pin fiind interactiv pentru că, atunci când se apasă pe unul dintre ele, în partea de jos este afișat un View mai mic cu informații sumare despre acel restaurant. Pinii de pe hartă sunt trimiși ca parametru sub forma unui tablou unidimensional ce stochează date de tipul MyAnnotationItem la View-ul Map. Acest ultim tip de dată a fost creat având ca proprietăți un id, coordonatele restaurantului și recenzia lui, acestea fiind stocate în annotationItems în MapViewModel. În figură este prezentată diagrama de clase a modulului Map.



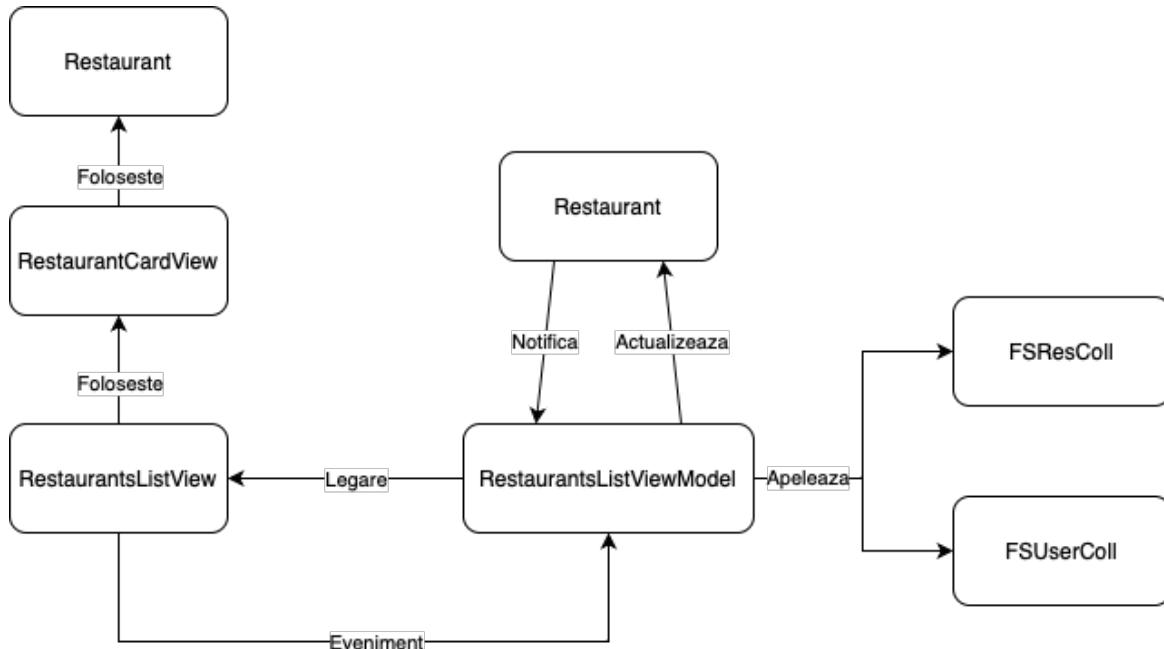
Figură 5.11 - Diagrama de clase a modulului Map

### 3. Modulul Restaurants

Acest modul conține logica și interfața utilizator pentru partea de căutare a restaurantelor și de rezervarea a unei mese. Când clientul selectează mijlocul celor trei opțiuni din partea de jos a ecranului, apare o listă cu toate restaurantele din aplicație, cu o bară de căutare deasupra acesteia, cu mențiunea că un restaurant se poate căuta doar după numele lui.

Fiecare element din listă afișează aceleași informații cu cele de pe hartă, acestea având însă plus o stea interactivă care permite clientului să adauge anumite restaurante la lista de preferate de pe pagina de profil. View-ul ce conține lista se află în RestaurantsListView având ca View-Model RestaurantsListViewModel, iar View-ul ce afișează informațiile sumare despre restaurant atât pe hartă cât și în acest ecran este declarat în RestaurantCardView.

Modifierul searchable este cel responsabil de adăugarea acelei bare de căutare. Ea primește ca parametrii un sir de caractere care își schimbă valoarea pe măsură ce utilizatorul tastează, plasamentul acesta pe ecran și un sir de caractere ce se afișează înainte de interacțiunea cu clientul pentru a-și sugera rolul. Actualizarea listei în momentul căutării o face filtrarea restaurantelor din tablou unidimensional denumit „restaurants” din View-Model. Momentul în care se inițializează această clasă și se populează tabloul, se face o copie a acestuia pentru a fi folosit la resetarea listei, însemnând că tabloul „restaurants” va avea înapoi toate restaurantele de la momentul inițializării. În figura 5.12 este prezentată diagrama de clase a modulului Restaurants.



Figură 5.12 - Diagrama de clase a modulului Restaurants

#### 4. Modulul RestaurantDetails

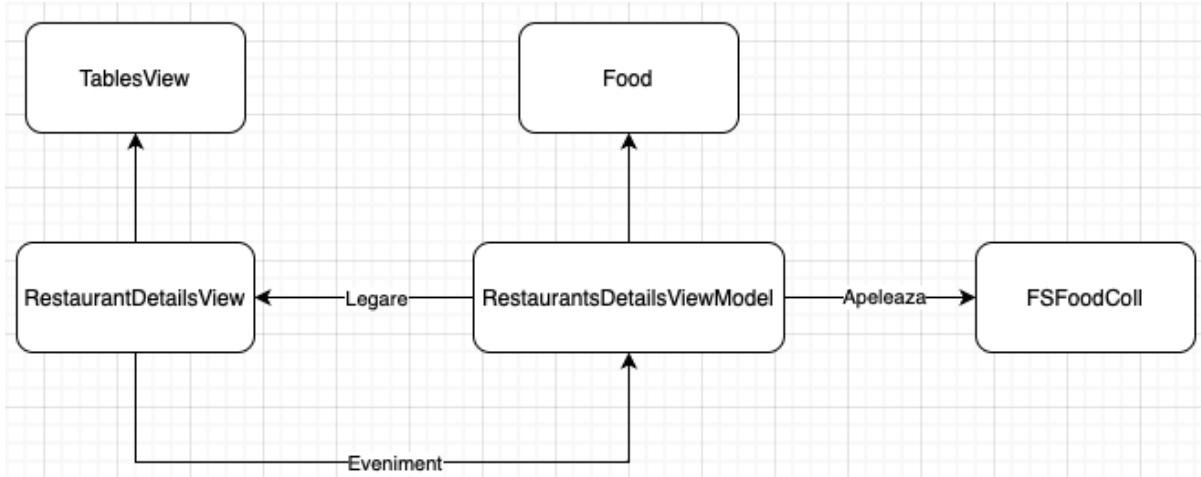
Possibilitatea de a naviga între lista de restaurante și pagina de detalii a acestuia o face View-ul NavigationView care, cu ajutorul NavigationLink, setează destinația fiecărui element din listă ca fiind pagina de detalii corespunzătoare restaurantului. NavigationLink primește ca parametru destinația, care în acest caz este RestaurantDetailsView cu parametrul restaurant, iar ca View al elementului din listă este RestaurantCardView care primește DTO-ul restaurantului respectiv.

Cu toate acestea, în RestaurantDetailsView se definește și View-ul pentru un fel de mâncare intitulat FoodCard care primește ca parametru felul de mâncare de tipul Food, iar ca și construcție este una simplă: Este definită o stivă verticală ce conține trei șiruri de caractere de afișat:

- Numele mâncării care se afișează prin intermediul proprietății foodName
  - Lista de ingrediente ce este returnată de valoarea variabilei ingredientsString
  - Prețul ce este returnat și afișat doar cu două zecimale, stocat în proprietatea price
- Toate acestea pot fi accesate din variabila food al View-ului FoodCard.

RestaurantDetailsViewModel are o singură metodă și anume de a prelua din baza de date toate felurile de mâncare ce aparțin de acel restaurant. Cum proprietatea denumită *menu* al entității Restaurant conține toate id-urile ce aparțin de meniu, în metoda fetchRestaurantMenu se verifică dacă id-urile felurilor de mâncare din baza de date se regăsesc în această proprietate. În caz afirmativ, se adaugă 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 afișa meniul.

De asemenea, o metodă importantă în RestaurantDetailsView o reprezintă isOpened() ce returnează un View depinzând dacă restaurantul este sau nu deschis. Mai întâi se rețin în constante ora de deschidere a restaurantului, de închidere și ora curentă (când se inițializează View-ul). Cu ajutorul unei instrucțiuni if se verifică dacă ora curentă este în intervalul programului restaurantului caz în care se afișează ora deschiderii și închiderii cu o culoare verde, iar în caz contrar se afișează un mesaj de culoare roșie ce sugerează închiderea acestuia. În figura 5.13 este prezentata diagrama de clase a modulului RestaurantsDetails.



Figură 5.13 - Diagrama de clase a modulului RestaurantDetails

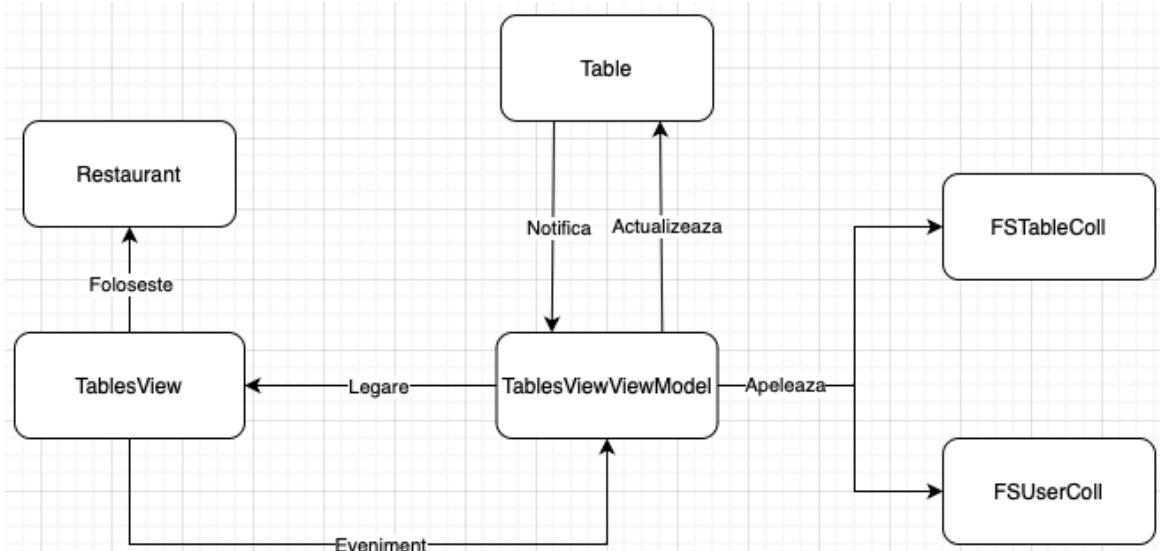
### 5. Modulul TablesList

View-ul ce afișează lista meselor disponibile pentru rezervare este definit în fișierul `TablesView` având View-Model-ul `TablesViewViewModel`. Dacă nu este nicio masă disponibilă se va afișa un mesaj în locul listei care sugerează acest lucru, iar în caz contrar utilizatorul va putea selecta ziua, ora și masa pe care dorește să o rezerve. Fiecare masă afișează și numărul maxim de persoane. Dacă un fel de mâncare nu este în stoc (este entitate `Food` și valoarea variabilei stock este egala cu 0) atunci ea nu va apărea în meniul restaurantului.

Logica funcționalității este una foarte simplă întrucât View-Model-ul are doar trei metode:

- `fetchAllTables(forRestaurant: Restaurant)`: metoda care returnează toate mesele ce aparțin restaurantului trimis ca și parametru apelând metoda de `fetchAllTables` a clasei singleton `FSTableColl` și populând tabloul unidimensional denumit `tables`.
- `bookingTable(tableId: String, hour: String, day: String, userId: String)`: metodă ce actualizează o entitate `Table` cu identificatorul egal cu parametrul `tableId`, prin schimbarea valorii proprietăților `hour`, `day`, `userId` și `booked` cu valorile parametrilor, respectiv `true` acestea întâmplându-se în metoda `tableBooked` al clasei `FSTableColl`. De asemenea, se apelează metoda `saveBookedTable(withId: String)` al clasei `FSUserColl` pentru actualizarea tabloului unidimensional al utilizatorului ce a rezervat masă. Într-un final se apelează metoda `updateTables()` din aceeași clasă
- `updateTables()`: metoda ce se apelează după ce o masă a fost rezervată pentru a afișa doar mesele ce sunt disponibile pentru rezervare

În figura următoare, 5.14, este prezentată diagrama de clase a modulului `TablesList`.



Figură 5.14 - Diagrama de clase a modulului TablesList

#### 6. Modulul UserProfile

Profilul clientului este creat în acest director împreună cu logica de confirmare a rezervării. View-ul respectiv este împărțit în două secțiuni: secțiunea de sus în care sunt afișate 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 conține View-ul pentru mesele rezervate și pentru restaurantele preferate.

Lista de restaurante nu este interactivă, ca urmare se face doar afișarea lor. Nici cu lista de rezervări nu se poate interacționa, dar fiecare element din această listă are două butoane interactive: unul pentru a șterge rezervarea, moment în care aceasta nu o să mai apară, și unul pentru a confirma sosirea la restaurant, clientul fiind redirecționat către 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 singură metodă numită `fetchRestaurantName` pentru a stoca numele restaurantului în proprietatea `restaurantName` ce are identificatorul mesei `tableId`, trimisă ca și parametru. În BookedTableView este o singură proprietate și anume `table` de tipul `Table`. Cu ajutorul acesteia se afișează informațiile necesare a unei mese.

UserProfileViewModel conține două metode ce se apeleză la momentul afișării ecranului de profil:

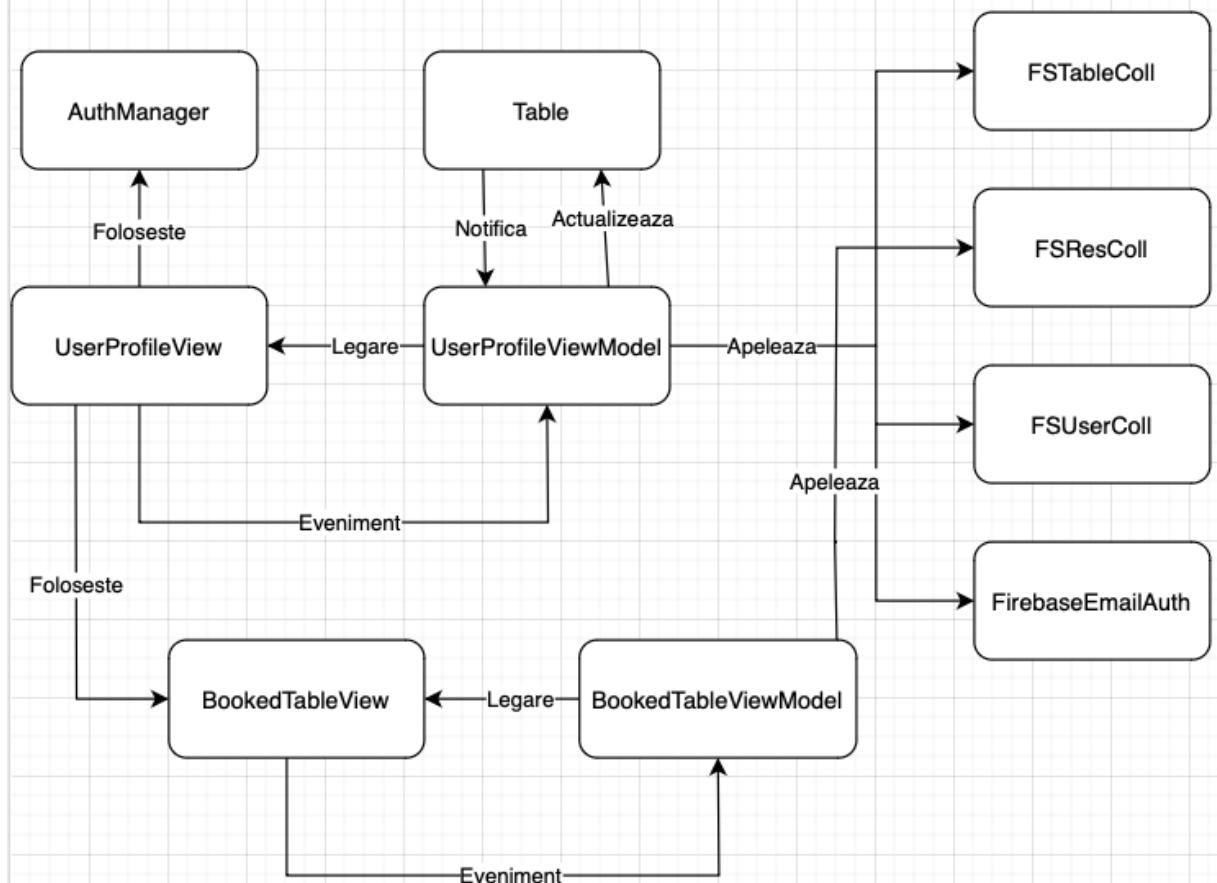
- `fetchLoggedUser()`: metoda care aduce din baza de date entitatea utilizatorului autentificat și îl reține în variabila `user` de tipul `MyUser`. Această metodă apeleză la final și `updateBookedTables(user: MyUser)` pentru a aduce din baza de date toate mesele rezervate de acest și se stochează în variabila `bookedTables`
- `fetchFavRests()`: metoda care aduce din baza de date toți identificatorii restaurantelor favorite a utilizatorului, iar pentru fiecare id se aduce restaurantul corespunzător și se adaugă în tabloul unidimensional `favouriteRestaurants`

Pe lângă acestea, fiecare buton are o metodă particulară care se apeleză cand utilizatorul apasă pe unul dintre ele:

- Se apasă butonul de deconectare: se apeleză metoda de `signOut()` care aceasta apeleză metoda `doLogout` al clasei `FirebaseEmailAuth` pentru deconectarea utilizatorului

- Se apasă butonul de anulare a unei rezervări: se apelează metoda de cancelBookingForTableWith(tableId: String) care apelează metoda deleteBookedTableWith(tableId: String) din clasa FSUserColl care face o ștergere a id-ului din bookedTables a utilizatorului

În figura 5.15 este prezentată diagrama de clase a modulului UserProfile.



Figură 5.15 - Diagrama de clase a modulului UserProfile

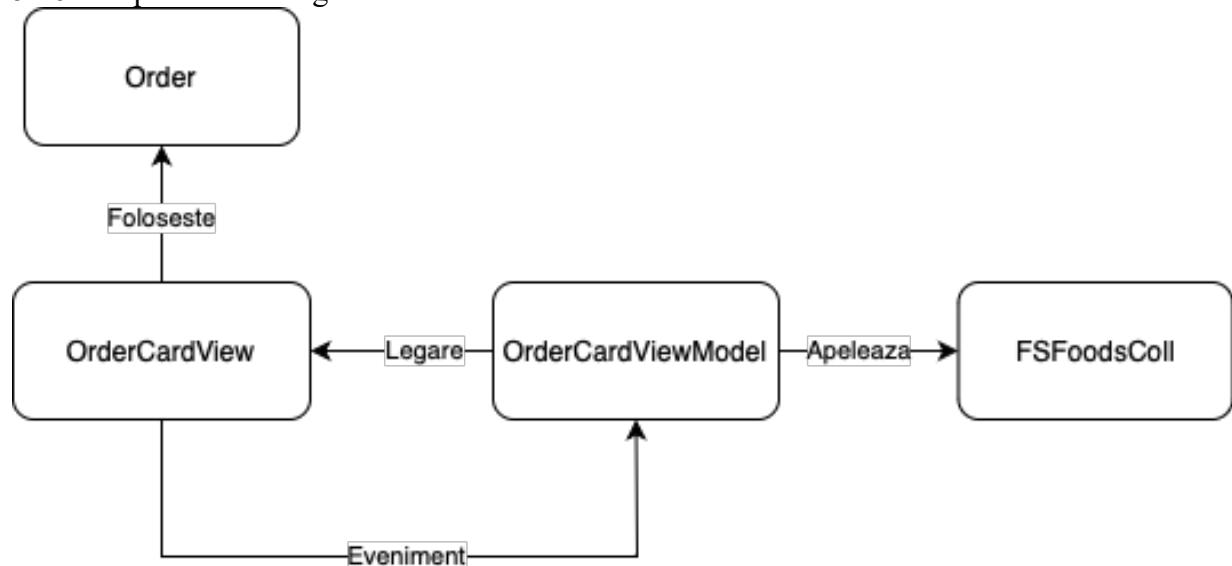
## 7. Modulul OrderView

În acest modul, se formează View-ul pentru o comanda, OrderCardView, cu logica separată în OrderCardViewModel. Aceasta este folosită pe ecranele destinate bucătarilor și ospătarilor pentru a afișa statusul comenzii. View-Model-ul are doar două funcții:

- fetchFoodsFor(order: Order): returnează felurile de mâncare ce au fost plasate în comanda aceasta fiind parametrul metodei. Iterându-se prin tabloul unidimensional foodIds al entității de tip Order, se adaugă în tabloul foods al clasei OrdersCardViewModel entitățile Food a căror id se regăsesc în tabloul de id-uri a comenzii. La final, se apelează metoda de orderFoodsAfterIds(order) deoarece datele din baza de date vin ordonate după UUID-ul generat de Firestore. Astfel, ordinea din foods și order.foodIds ar fi diferită, iar acest lucru nu este de dorit întrucât order.foodQuantity corespunde id-urilor felurilor de mâncare la momentul plasării 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 afișează numărul mesei de la care s-a plasat comanda, fiecare fel

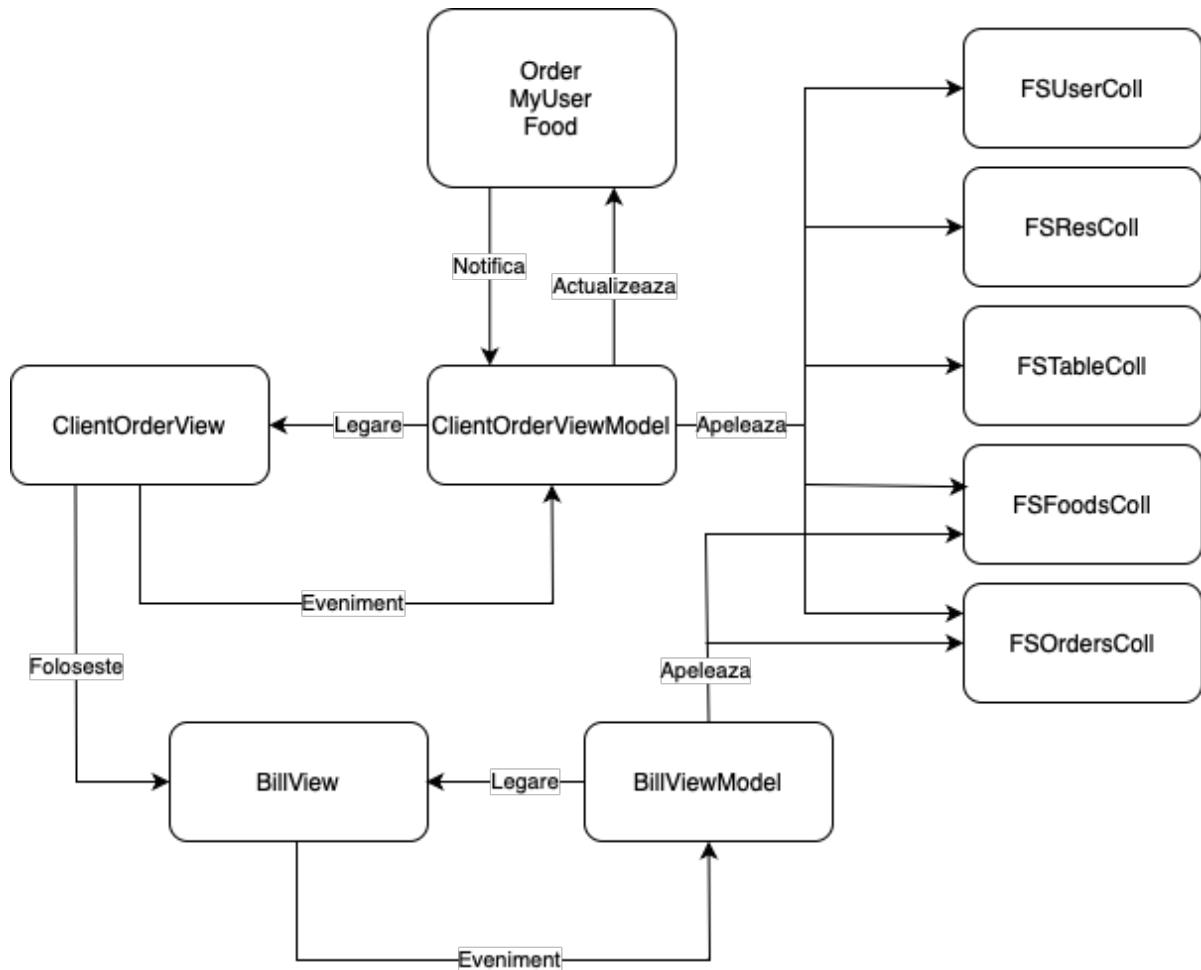
de mâncare cu cantitatea comandata și pretul ei, iar la final costul total al comenzi. În figura 5.16 este prezentată diagrama de clase a acestui modul.



Figură 5.16 - Diagrama de clase a modulului OrderView

#### 8. Modulul ClientOrder

Ultimul modul cu funcționalități specifice utilizatorului cu rolul de client este ClientOrder ce conține ecranul de plasare a unei comenzi și a notei de plată. ClientOrderView este fișierul ce conține elementele de baza a interfeței utilizator dedicate acestei funcționalități și este alcătuită din două liste: una în jumătatea de sus a ecranului în care utilizatorul poate să consulte meniul și să își adauge cantitatea de mâncare dorită, și în cealaltă jumătate observă în timp real care este coșul. Dacă acesta cere nota de plată fără a plasa cel puțin o comandă se va afișa un mesaj de eroare, metoda checkForOrdersStatus() din ClientOrderViewModel tratând acest caz. În caz contrar, i se va afișa pe ecran o listă a tuturor comenziilor plasate, BillView, (care trebuie să fie în ultima stare, servite) și totalul de plată. Dacă vreun fel de mâncare nu este în stoc atunci nu o să apară în meniu. În figura 5.17 se poate observa diagrama de clase a acestui modul.



Figură 5.17 - Diagrama de clase a modulului ClientOrder

Modulele ce se ocupă de funcționalitățile dedicate utilizatorilor care sunt ospătari sau bucătari sunt Waiter respectiv Cook. Fiecare are un View, WaiterView și CookView, cu un View-Model asignat, WaiterViewViewModel respectiv CookViewModel. Interfața este foarte asemănătoare celei de profil a clientului, diferențele fiind:

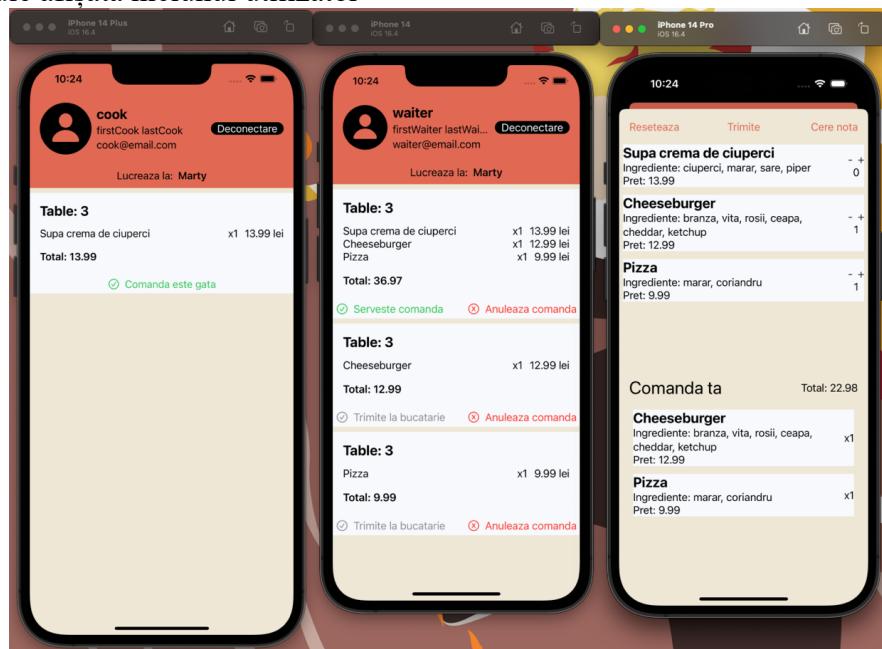
- În secțiunea de informații a utilizatorului se mai afișează un câmp în plus cu numele restaurantului unde aceștia lucrează
- În secțiunea de jos se afișează o singură listă cu comenzi, singura diferență între acestea este că pentru ospătar se afișează când comanda este în prima stare sau a treia pe când bucătarului i se afișează comenzi ce sunt în cea de a doua stare

Logica de echilibare a comenzielor per chelner se face în metodele acceptOrder() și canAcceptOrder() din WaiterViewViewModel. Fiecare comandă afișată ospătarului conține un buton verde ce trimite comanda ori la bucătar ori pentru a confirma servirea ei. Butonul se dezactivează în momentul în care chelnerul a acceptat prea multe comenzi. Metoda acceptOrder primește doi parametrii: id-ul comenzi și starea acesteia. Se preia valoarea din UserDefaults a numărului de comenzi acceptate până la acel moment și se execută trei instrucțiuni if: prima verifică dacă starea este prima, caz în care numărul de comenzi acceptate crește, a doua verifică dacă comanda este în a treia stare ceea ce înseamnă că urmează să fie servită și ca urmare numărul de comenzi acceptate scade, iar cea de a treia verifică dacă numărul de comenzi acceptate depășește numărul maxim acceptabil. Dacă da, atunci valoarea maximă o să fie salvată înapoi în UserDefaults. După se apelează metoda canAcceptOrders() ce actualizează valoarea variabilei publicate canTakeOrder cu true sau false depinzând de numărul de comenzi acceptate din UserDefaults. La schimbarea acesteia, butonul de trimis

la bucătarie a comenzi este dezactivat. Această metodă se apelează și pe metoda init(), aceasta la rândul ei fiind apelată la inițializarea 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 stânga 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 apăsarea butonului „Comanda este gata”, iar ospătarul nu mai poate să accepte ultimele două comenzi din lista lui întrucât a acceptat-o pe care se află în modul de preparare (cea de pe ecranul bucătarului) și pe cea care este gata de servire (confirmarea trebuie să o facă prin apăsarea butonului „Serveste comanda”), lucru evidențiat de textul gri al butoanelor din partea stângă. Pe lângă acestea, se pot observa primele trei stări a unei comenzi:

- pending: ospătarul primește comenziile și textul butonului din partea stângă este „Trimite la bucătarie” (în cazul acesta sunt gri pentru ca ospătarul a depășit limita de acceptare a comenzilor)
- preparing: comanda ajunge la bucătar și acesta poate să o treacă în următoarea stare apăsând butonul verde
- ready: comanda ajunge de la bucătar înapoi la ospătar, iar textul butonului verde devine „Serveste comanda”
- sent: comanda nu mai este vizibilă pe niciun ecran întrucât ea a fost servită și nu trebuie afișată niciunui utilizator



Figură 5.18 - Stările unei comenzi

## Capitolul 6. Testare și validare

În acest capitol se vor prezenta metodele de testare a aplicației QuickPlate. Testarea va fi prin parcurgerea unor acțiuni, iar fiecare va avea un rezultat așteptat. Funcționalitățile vor fi înregistrarea de cont, rezervarea unei mese, algoritmul de echilibrare a comenzi și cererea notei de plată de către client.

În tabelul 6.1 se va prezenta cazul de test pentru funcționalitatea de înregistrare. Aceasta este folositoare în momentul în care un utilizator dorește să beneficieze de funcționalitățile aplicației în funcție de rolul pe care îl deține (client, ospătar sau chelner).

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

Tabel 6.1 - Cazul de test a funcționalității de înregistrare a unui utilizator

Acțiune	Rezultat
Utilizatorul deschide aplicația	Aplicația se deschide fără erori și este prezentat ecranul de autentificare
Utilizatorul face clic pe butonul cel mai de jos, „Creeaza un cont nou”	Utilizatorul este redirecționat către ecranul de înregistrare
Utilizatorul completează corect toate câmpurile prezente și dă clic pe butonul de jos	Utilizatorului îi este prezentat un mesaj de informare de a-și verifica adresa de email
Utilizatorul apasă pe butonul de ok al mesajului de informare	Utilizatorul este redirecționat înapoi la ecranul de autentificare

Ca urmare a acestui caz, utilizatorul va primi pe adresa de email un email nou din care trebuie să confirme adresa introdusă la înregistrarea contului.

În următorul tabel, 6.2, se va prezenta cazul de test pentru funcționalitatea de rezervare a unei mese. Funcționalitate dedicată doar utilizatorului de tip client și după ce acesta s-a hotărât să rezerve o masă la un restaurant ales, în urma interacționării 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 dă clic pe opțiunea din mijlocul bării poziționate în partea de jos a ecranului	Utilizatorul este redirecționat către ecranul ce conține lista restaurantelor din aplicație
Utilizatorul caută restaurantul dorit la care dorește să rezerve o masă	Restaurantul există în aplicație
Utilizatorul dă clic pe restaurant evitând zona stelei galbene din dreapta	Utilizatorul este redirecționat către pagina de detalii a restaurantului
Utilizatorul apasă pe butonul plasat în dreapta a barei de sus cu textul „Rezerva o masă”	Utilizatorului îi este prezentat ecranul din care se poate rezerva o masă, iar în mijloc se află lista meselor disponibile și nu mesajul de eroare care sugerează indisponibilitatea lor

Utilizatorul alege ora din picker-ul din stânga, ziua din picker-ul din dreapta și după apasă pe butonul „Rezerva” din dreptul mesei care se dorește a fi rezervată	Se afișează un mesaj de confirmare cu două butoane, „Da” și „Nu”
Utilizatorul apasă butonul de „Da”	Apare mesaj de informare cu un buton de „Ok”
Utilizatorul apasă pe butonul de „Ok”	Mesajul de informare dispare
Utilizatorul apasă pe butonul „Anuleaza” din partea stângă a barei de sus	Ecranul de rezervare a unei mese dispare și reapare cel de detalii a restaurantului
Utilizatorul apasă pe butonul „Inapoi” din partea stângă a barei de sus	Utilizatorul este redirecționat la lista restaurantelor
Utilizatorul apasă pe butonul de profil a tab-ului poziționat în partea de jos a ecranului	Utilizatorul este redirecționat către pagina de profil
Utilizatorul apasă pe butonul de „Rezervari” dintre cele două opțiuni, iar în caz contrar nu mai apasă niciun buton	Rezervarea este vizibilă sub forma unui card în lista de „Rezervari”

În tabelul 6.3 este prezentat cazul de testare pentru funcționalitatea de echilibrare a comenzi. 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 făcută la un anumit restaurant și îi apare pe pagina de profil, cei doi chelneri și bucătarul lucrează la același restaurant unde este și rezervarea, iar ca o ultimă condiție, angajații sunt autentificați.

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

<b>Acțiune</b>	<b>Rezultat</b>
Clientul apasă butonul verde „Confirma sosirea”	Clientului îi este prezentat ecranul din care poate plasa comenzi
Clientul apasă pe butonul de „+” a unui fel de mâncare	Acel fel de mâncare apare în a doua listă ce reprezintă comanda curentă
Clientul apasă pe butonul „Trimite”	Lista cu comanda este resetată și comanda este trimisă și apar pe ecranele ospătarilor
Clientul repetă ultimii doi pași pentru a mai trimite trei comenzi	Pe ecranele ospătarilor este un total de patru comenzi
Ospătar 1 apasă pe butonul verde „Trimite la bucătarie” la două comenzi	Cele două comenzi sunt trimise la bucătar, șterse din lista ospătarilor, ospătar 1 are butonul de „Trimite la bucătarie” cu culoarea gri în timp ce ospătar 2 are butonul colorat verde
Bucătarul apasă la o comandă butonul verde „Comanda este gata”	Comanda apare înapoi pe ecranul ospătarilor
Ospătar 1 apasă pe butonul verde a comenzi venite de la bucătar cu textul „Serveste comanda”	Comanda este ștearsă din lista ospătarilor și butoanele gri ale primului ospătar devin din nou verzi

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

**Precondiții:** Clientul are deja o rezervare făcută și este pe pagina de profil, ospătarul și bucătarul lucrează la același restaurant la care s-a făcut rezervarea.

Tabel 6.4 - Cazul de test a cererii notei de plată

<b>Acțiune</b>	<b>Rezultat</b>
Clientul apasă pe butonul „Confirma sosirea”	Clientului îi este prezentat ecranul din care este posibilă acțiunea de plasare a unei comenzi
Clientul apasă pe butonul „+” a cel puțin unui fel de mâncare	Felul de mâncare apare în lista de jos a ecranului care indică cantitatea aleasă
Clientul apasă pe butonul de „Trimite”	Lista de jos este resetată și comanda apare pe ecranul ospătarului
Ospătarul apasă butonul verde care are textul „Trimite la bucătarie”	Comanda este ștearsă din lista ospătarului și afișată în lista bucătarului
Bucătarul apasă pe butonul verde „Comanda este gata”	Comanda este ștearsă din lista bucătarului și apare în lista ospătarului
Ospătarul apasă butonul verde „Serveste comanda”	Comanda este ștearsă din lista ospătarului
Clientul apasă pe butonul din partea dreaptă a barei de sus „Cere nota”	Clientului îi este prezentat o listă cu toate felurile de mâncare pe care le-a comandat și totalul
Clientul apasă butonul „Achitati nota de plată” din partea stanga a barei de sus	I se prezintă un mesaj cu opțiuni pentru a alege nota de plată
Clientul apasă pe una din următoarele opțiuni: Cash, Cu cardul, Folosind aplicația	Se afișează un mesaj de informare care anunță că un angajat urmează să vină la masă
Clientul apasă pe butonul de „Ok” la mesajului informativ	Clientul este redirectionat la pagina de profil

## **Capitolul 7. 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 lângă 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 necesară deținerea unui certificat de dezvoltator, oferit de Apple, în valoare de 100 de dolari. Acestea fiind spuse, s-a preferat păstrarea aplicației într-o variantă locală. Înainte de toate acestea, utilizatorul va trebui să își creeze un cont Apple, lucru posibil de pe site-ul lor oficial.

### **7.1. Resursele necesare**

#### **7.1.1. Resurse hardware**

- Smartphone cu sistemul de operare iOS sau tabletă cu sistemul de operare iPadOS. Este obligatoriu ca ambele versiunii în oricare din cele două cazuri să fie mai mare decât 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 configurații 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)

#### **7.1.2. Resurse software**

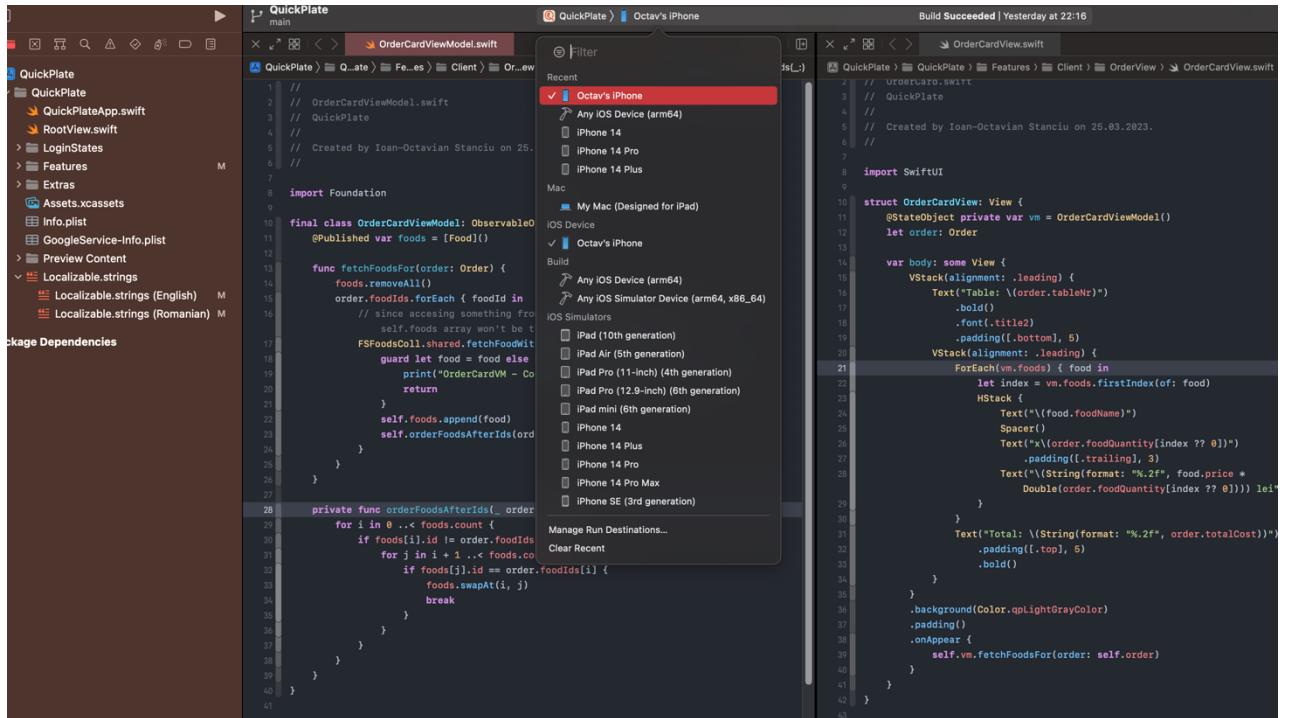
- Mediul de dezvoltare integrat al Apple, Xcode, cu versiunea minimă de 14.2.
- Git pentru a putea clona proiectul din depozitul de pe GitHub
- SourceTree pentru o vizualizare mai bună a evoluției proiectului, aceasta fiind opțională

### **7.2. Manual de instalare**

În acest subcapitol se vor prezenta pașii ce trebuie urmați în ordinea descrierii lor pentru ca utilizatorul să reușească descărcarea și instalarea aplicației:

- Utilizatorul va trebui să își descarce și să își instaleze ultima versiune 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 apasă butonul de Download, se selecteză variante de Website, își introduce credențialele contului de Apple, și se va descarca ultima versiune.
- Al doilea pas este descărcarea și instalarea unei 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 instalează automat la instalarea tool-ului SourceTree
- Se va naviga într-un fișier unde se dorește scanarea proiectului și se va deschide în acel loc un terminal
- Se va introduce următoarea comandă: git clone <https://github.com/StanciuOctav/QuickPlate>

- Se va deschide Xcode, fie prin apăsarea de două ori a fișierului QuickPlate.xcodeproj, fie prin a selecta din stânga sus File -> Open -> și se va da dublu clic pe modulul care conține proiectul.
- Până când proiectul se configurează și se descarcă toate dependințele 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 apăsa butonul de play din stânga sus sau scurtătură CMD + R. În figura 7.1 se poate observa cum arată această listă și cum se poate identifica dispozitivul utilizatorului, acesta având altă iconă față de cele ale simulatorului.

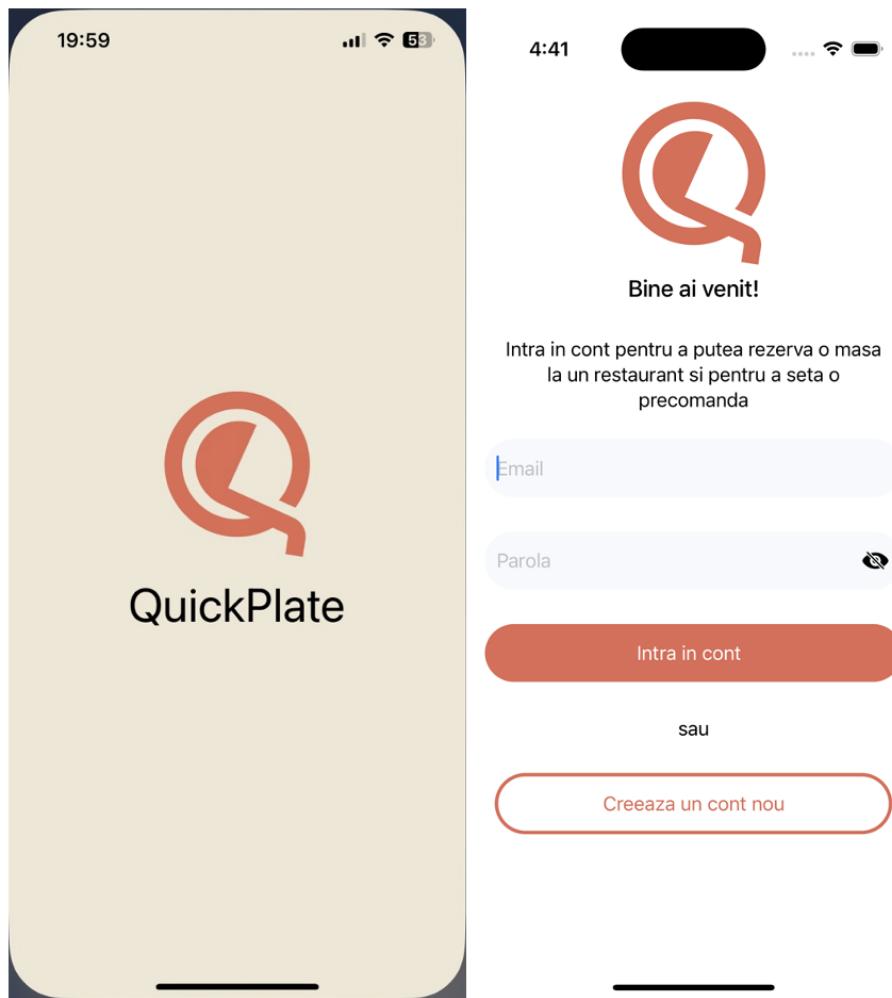


Figură 7.1 - Lista de simulatoare din Xcode

- Pentru a testa aplicația observând în același timp toate funcționalitățile ei, trebuie rulate minimul patru instante dintre care una este rolul de client, a doua cea de bucătar, iar ultimele două cele de ospătari. Pe dispozitiv se poate rula o singura instantă, astfel se vor alege din lista de simulator încă trei, iar după fiecare selecție se va rula proiectul.

### 7.3. Manual de utilizare

În momentul în care o instanță a aplicației va porni se va afișa pentru două secunde un ecran de prezentare. După aceasta, utilizatorul va fi redirectionat către ecranul de autentificare. Aceste două ecrane sunt prezentate în figura 7.2

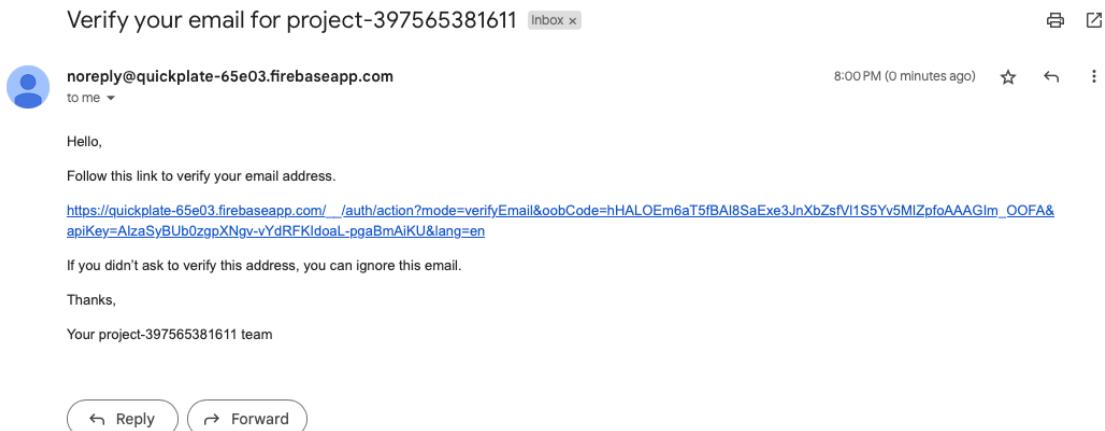


Figură 7.2 - Ecran de prezentare și ecranul de autentificare a aplicației

În ecranul de autentificare utilizatorul își va putea introduce credențialele pentru putea executa acțiunea de Sign In. Dacă nu are cont, atunci acesta va fi nevoie să apese cel mai de jos buton, „Creează un cont nou”, pentru a putea fi redirectionat către ecranul de înregistrare.

Odată ajuns la acest ecran, utilizatorul este obligat să completeze toate câmpurile prezente, cu excepția celui de „Restaurant” care este inactiv în cazul în care tipul contului este „Client”. Dacă rolul utilizatorului va fi ori „Chelner” ori „Bucătar” atunci câmpul respectiv va deveni activ și va trebui să se selecteze numele restaurantului la care acesta lucrează. Bineîntele, pot apărea mesaje de eroare. Pentru cazurile în care parolele nu coincid sau nu se conformează la formatul acceptat, se va afișa un mesaj de eroare deasupra butonului „Creează un cont nou” ca în figura 7.4, iar pentru cazul în care unul dintre câmpuri nu este completat sau adresa de email este deja folosită atunci se vor afișa mesaje speciale ce îi vor sugera utilizatorului greșelile pe care le-a facut pentru a le putea remedia și continua procesul de înregistrare.

Dacă toate câmpurile au fost completeate corect și s-a respectat formatul atunci în momentul în care utilizatorul apasă pe butonul de creare a contului, i se va afișa un mesaj de succes similar cu cel din figura 7.2, poza din dreapta, care îi va sugera să își verifice adresa de email pentru a confirma adresa introdusă. După aceasta, o să fie redirectionat înapoi la ecranul de autentificare. În figura 7.3, se poate observa cum arată emailul de confirmare. După ce se accesează link-ul de culoare albastră, utilizatorul va putea să se autentifice fără nicio problemă.



Figură 7.3 - Email de confirmare

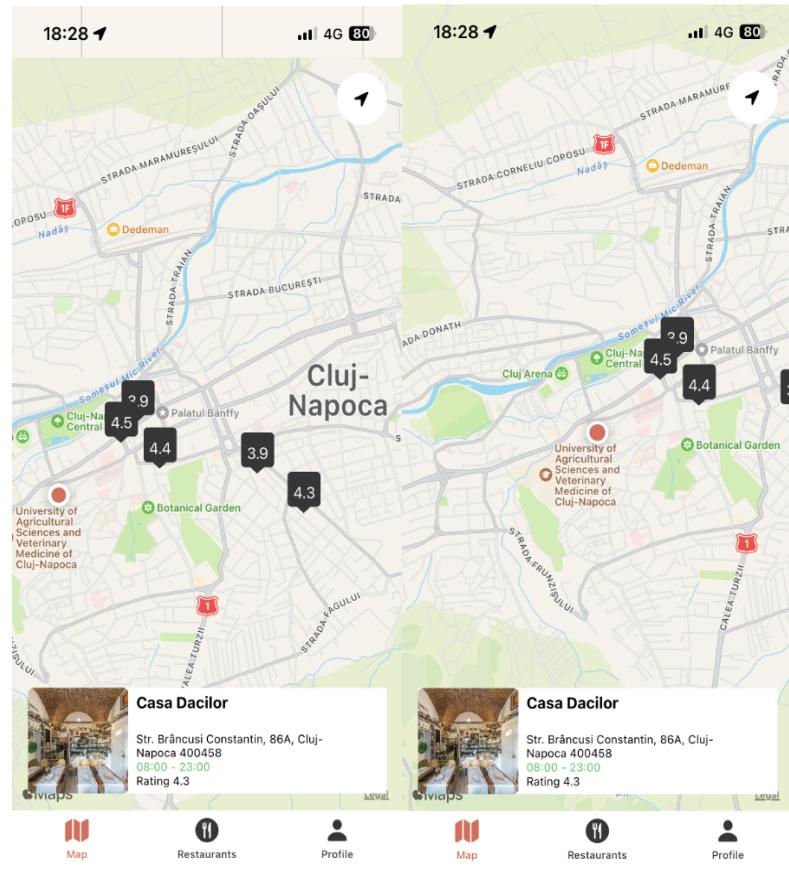
Odată întors, acesta va trebui să își reintroducă adresa de email și parola pe care le-a introdus pe ecranul de înregistrare. În cazul în care, utilizatorul nu a reusit să își facă contul sau nu și-a verificat adresa de email, atunci se va afișa un mesaj de eroare deasupra butonului „Intra în cont”.

Odată ce utilizatorul a reușit să se autentifice în aplicație în urma introducerii adresei de email și a parolei corespunzătoare contului creat, acesta va fi redirecționat către ecranul principal. Acesta diferă de la un utilizator la altul, diferența făcând-o tipul contului ales în ecranul de înregistrare. Dacă acesta a ales opțiunea de „Client” atunci el va fi redirecționat către un ecran ce conține în partea de jos o bară cu trei secțiuni. Fiecare secțiune va afișa alt ecran în timp ce pentru celelalte două tipuri de utilizator, se va afișa un singur ecran. În continuare se va prezenta manualul de utilizare pentru client, iar pentru ospătar și bucătar se vor explica în paralel.

#### 7.3.1. Manual de utilizare pentru client

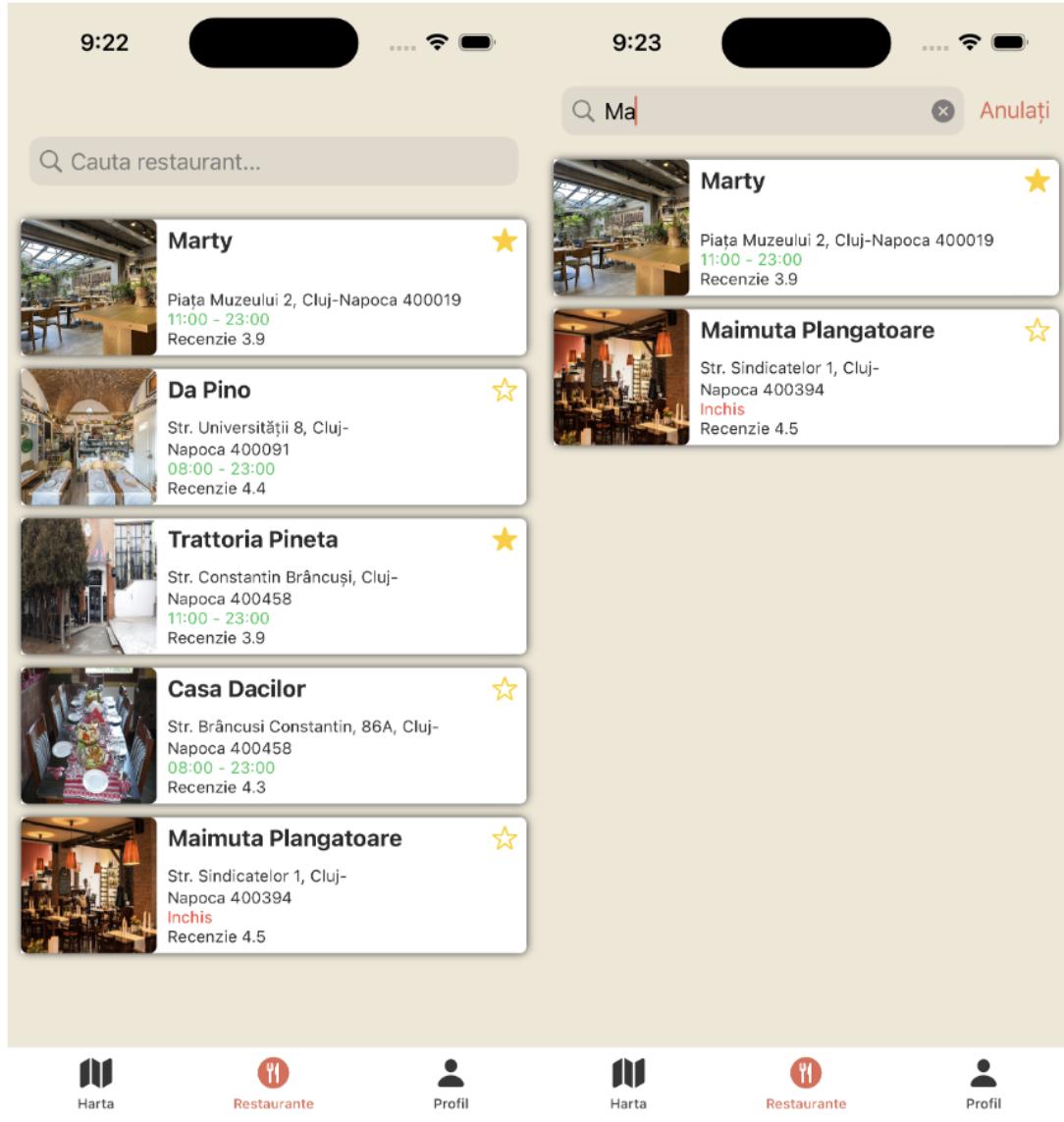
Odată autentificat în aplicație, clientului i se va prezenta ecranul dedicat rolului lui. Ecranul de hartă va fi primul care o să apară, împreună cu restaurantele ce sunt reprezentate ca și niște pin-uri. Din moment ce pinurile sunt interactive, clientul poate apăsa pe ele, acestea afișând informații despre restaurantul ce se găsește la acea locație, lucru ce poate fi observat în figură.

Informațiile restaurantului sunt afișate în partea de jos a ecranului. În partea dreaptă de sus a hărții există un buton ce va recentra harta la locația utilizatorului, dar pentru ca acest lucru să se poate întâmpla, prima dată clientul va trebui să permită aplicației să acceseze locația acestuia. Dacă clientul refuză, acel buton nu va fi de folos. În figura 7.4 se vede rezultatul acțiunii apăsării butonului în cazul în care clientul a permis accesul locației, poza din dreapta reprezentând rezultatul.



Figură 7.4 - Ecranul de hartă

Pe secțiunea din mijloc a barei de jos, dacă este selectată, va fi prezentată lista ce conține toate restaurantele din aplicație. Deasupra acesteia este o bară de căutare cu care se poate face filtrarea restaurantelor după numele lor. De asemenea, în dreptul fiecarui element este plasat un buton în formă de stea. Dacă steaua nu este umplută cu culoarea galbenă și butonul este apăsat, atunci steaua o să fie umplută cu culoarea menționată, însemnând că restaurantul a fost adăugat în lista de favorite de pe pagina de profil a utilizatorului. În cazul în care steaua este deja umplută cu culoare și se apasă din nou butonul atunci restaurantul va fi șters din lista de favorite. În figura 7.5, în poza din stânga, se poate vizualiza în listă restaurantele cu numele Marty și Trattoria Pineta fiind de asemenea în lista de favorite, iar în partea din dreapta se observă funcționalitatea barei de căutare, aceasta filtrând restaurantele ce conțin în numele lor sirul de caractere „Ma”.



Figură 7.5 - Ecranul cu lista de restaurante

Fiecare element a listei descrise anterior este interactiv fiind de ajutor la navigarea către pagina de detalii a restaurantului ce conține un număr mai mare de informații despre restaurantul pe care s-a facut clic. De asemnea, toate acestea sunt puse într-o listă cu care nu se poate interacționa deoarece are ca scop doar oferirea posibilității clientului de a vizualiza meniul corespondent restaurantului. În bara de sus sunt două butoane:

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

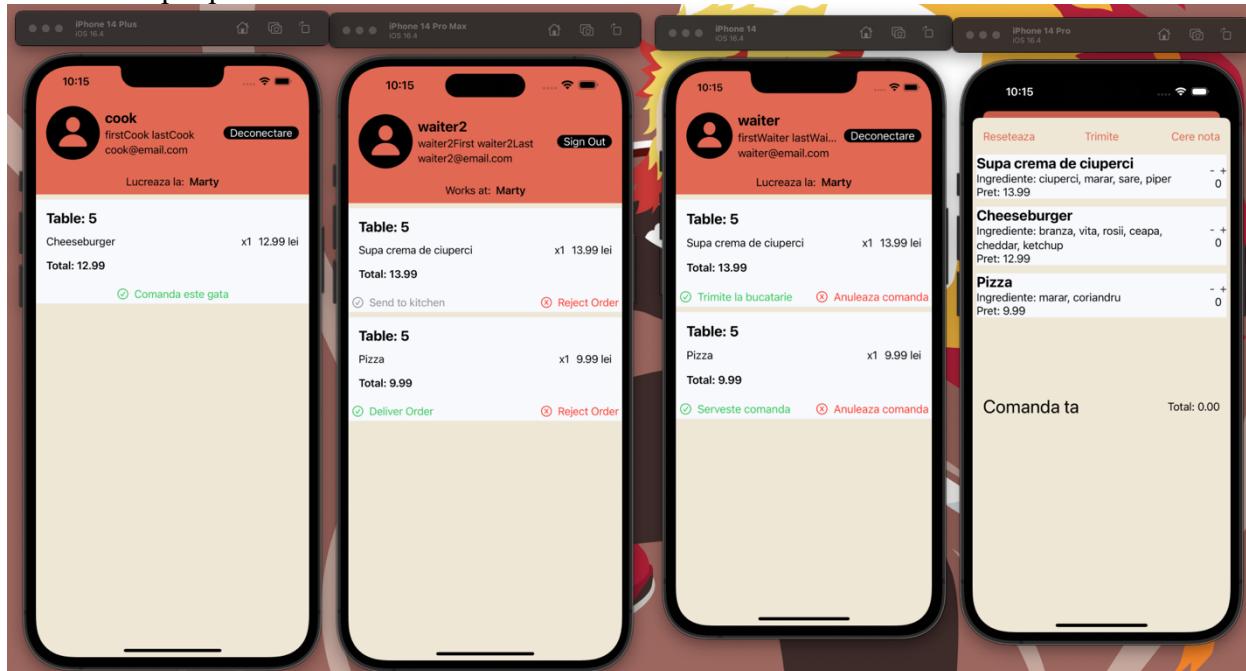
În ecranul de rezervare, utilizatorul este nevoit să selecteze ora și ziua prin gestul de derulare a listelor, iar după acestea să apese butonul „Rezerva” din dreptul mesei ce se dorește a fi rezervată. În același ecran, în colțul stânga sus este plasat un buton „Anuleaza” care închide ecranul respectiv.

Ultima secțiune a barei de jos este profilul în care utilizatorul poate vizualiza comenziile respectiv rezervările efectuate, iar în dreptul numelui de familie este un buton cu titlul „Deconectare” ce ajută utilizatorul să se deconecteze din aplicație. La fiecare rezervare sunt două butoane: cel verde pentru a confirma sosirea la restaurant și pentru a începe plasarea

comenzilor, iar cel roșu pentru a anula rezervarea. În ecranul de plasare a comenzi, lista felurilor de mâncare este afișată și este interactivă încrucât clientul poate interacționa 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 plată. După alegerea metodei de plată, rezervarea este ștearsă și clientul este redirecționat către pagina de profil.

### 7.3.2. Manual de utilizare pentru ospătar și bucătar

După autentificare, angajatul este redirecționat direct către pagina de profil, singurul ecran care i se afișează. În centru, sub secțiunea de detalii a utilizatorului, există o listă care se populează cu comenziile plasate de clienți. Pentru rolul de ospătar, fiecare element are două 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 roșu pentru a anula comanda, iar în cazul bucătarului există doar butonul verde pentru a trimite comanda înapoi la ospătari sugerând că s-a terminat de preparat. În figura 7.5, se pot observa patru instanțe ale aplicației ce rulează în același timp, cea mai din stânga fiind pentru bucătar, cele două din mijloc pentru ospătari, iar cea din dreapta pentru client.



Figură 7.6 - Exemplu de rulare folosind simluatorul Xcode

## Capitolul 8. Concluzii

Proiectul urmărește 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 ajută la rezolvarea acestei probleme datorită funcționalităților pe care le detine și prin care se diferențiază comparativ cu aplicațiile concurente existente pe piață la momentul actual. În tabelul 8.1 este făcută o comparație între aplicația QuickPlate și aplicațiile deja existente, iar evaluarea s-a facut după verificarea de detinere a următoarelor funcționalități:

- 1) O hartă integrată pe care utilizatorul poate să observe restaurantele din proximitatea lui.
- 2) Posibilitatea de a vizualiza meniul restaurantului și de a comanda din aplicație.

- 3) Posibilitatea de a plăti din aplicație sau de a cere nota de plată să fie adusă la masă
- 4) Posibilitatea de a folosi aplicația fără a cere un abonament utilizatorilor

Tabel 8.1 - Comparație între QuickPlate și aplicații existente

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

Pe lângă funcționalitățile de bază precum autentificare, deconectare, înregistrare și un ecran de bază, 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, interfața utilizator este una foarte sugestivă și minimalistă, pe parcursul dezvoltării proiectului urmărindu-se păstrarea simplității și evitarea încărcării ecranelor cu elemente vizuale.

Deși unele aplicații de pe piață sunt mai bogate în anumite funcționalități de bază cum ar fi deținerea mai multor informații despre utilizator și restaurant, s-a urmărit implementarea cât mai multor funcționalități cu scopul de a fi cât mai ușor scalabilă în eventuala dezvoltare prin adăugarea unor funcționalități noi, componentele create să fie reutilizabile, iar clasele să nu fie strâns cuplate, lucru realizat datorită folosirii modelului arhitectural Model-View-View-Model.

## 8.1. Contribuțiiile proprii

Proiectul urmărește eficientizarea procedeului de rezervare a unei mese și atât de primire cât și de livrare a unei comenzi la restaurant. Aplicația ce a fost dezvoltată are ca scop îndeplinirea acestor sarcini datorită simplității și rezultatelor obținute în urma evoluării acesteia.

Contribuțiiile proprii aduse au fost stabilirea bazelor aplicației atât din punct de vedere a structurării codului și componentelor cât și din cel al experienței utilizator și interfeței grafice. Deși anumite aplicații de pe piață oferă mai multe informații precum zilele speciale în care restaurantele nu sunt deschise, ziua de naștere a utilizatorului, istoricul căutărilor și multe altele, s-a urmărit funcționarea cât mai eficientă și simplă a funcționalităților implementate pentru a merge cât mai bine în aplicația proprie. Câteva dintre acestea sunt:

- Integrarea API-ului de la Firebase pentru stocarea și modificarea datelor entităților folosite în aplicație
- Integrarea API-ului de hartă oferit de Apple pentru o navigare cât mai ușoara a utilizatorului
- Oferirea posibilității unui utilizator de a rezerva o masă la un restaurant ales
- Posibilitatea pentru client de a alege metoda de plată
- Echilibrarea comenziilor pentru a eficientiza primirea și livrarea comenziilor

## 8.2. Dezvoltări ulterioare

Partea de dezvoltare ulterioară este ușoară de realizat datorită arhitecturii simple și structura claselor ce fac posibilă refolosirea lor cu ușurință, așa este explicitat și în capituloare anterioare. Următoarea listă enumerează posibilele opțiuni ce se pot adăuga într-o versiune viitoare a aplicației:

- Partea de design: Se poate adăuga pe viitor suport pentru modul întunecat (dark theme), aplicația având la momentul actual culorile setate doar pentru partea luminoasă (light theme), culorile se pot alege mai potrivit pentru acest tip de aplicație.
- Adăugarea de notificări: Se poate adăuga funcționalitatea de primire de notificări pentru toți utilizatorii (clientul primește notificare când s-a plasat comanda, când o comandă a fost servită sau cu un anumit timp înainte de a-i aduce aminte de rezervarea făcută, iar ospătarul primește notificare când s-a plasat o nouă comandă sau cineva dorește să achite nota de plată).
- Adăugarea unui sistem de navigare: Pe pagina de hartă se poate adăuga un sistem de navigare de la poziția curentă a utilizatorului până la restaurantul selectat scutind-ul pe acesta de a folosi altă aplicație cu acest scop precum Google Maps, Apple Maps sau Waze.
- Adăugarea unui sistem de recenzii: Implementarea funcționalității de a putea lăsa recenzii la restaurante de către clienți, iar în funcție de evaluarea lor să se actualizeze rating-ul restaurantului.
- Adăugarea mai multor filtre de căutare: Posibilitatea de a-i permite utilizatorului căutarea restaurantelor după diferite criterii precum rating, locație, fel de mâncare și altele.
- Posibilitatea de a arăta o hartă a meselor: Integrarea unei funcționalități în care clientului nu îi este prezentată lista cu restaurantele ci o hartă a acestora cu care poate interacționa, iar aceasta să fie configurabilă de către manager-ul restaurantului în cazul unor modificări.
- Posibilitatea de a invita prietenii la o rezervare: Dezvoltarea unei opțiuni prin care un client își poate invita prietenii la o rezervare făcută pentru ca toți să primească un reminder.
- Posibilitatea de a împărti nota: Împreună cu funcționalitatea anterioară, toți clienții ce sunt la aceeași rezervare să poată să comande fiecare separat, iar la final nota să se poată achite ori de un singur om, ori fiecare să plătească propria consumație.

## Bibliografie

- [1] Bottorff Cassie, Haan Kathy. „Toast POS Review 2023: Features, Pricing & More”, 5 Mai, 2023.
- [2] Fabregas Krista, Main Kelly. „Best Restaurant Inventory Management Software 2023”, 5 Aprilir, 2023.
- [3] Jolaoso Christiana, Main Kelly, Watts Rob. „Toast Vs. Square (2023 Comparison), 14 Mai, 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 in 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] Apple - [https://ro.wikipedia.org/wiki/Apple\\_Inc](https://ro.wikipedia.org/wiki/Apple_Inc).
- [13] John Hughes. „iPhone vs Android: Which is better for Web Professionals / Developers?”, 1 Noiembrie, 2022.
- [14] Petroc Taylor. „Mobile operating systems' market share worldwide from 1st quarter 2009 to 4th quarter 2022”, 21 Februarie, 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 aplicatii web si 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 pentru crearea si administrarea datelor persistente in bazele de date
DTO	Data Transfer Object	Obiect pentru incapsularea datelor si pentru a reduce cantitatea de date transferate intre multiple sisteme
GUI	Graphical User Interface	Interfața 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
UUID	Unique User Identifier	Un identificator unic asignat utilizatorilor unei aplicatii

## Anexa 2. Lista figurilor și tabelelor

Figură 4.1 - Design prototipizat în Figma.....	11
Figură 4.2 - Diagramă de stări a unei comenzi .....	13
Figură 4.3 - Diagramă de flow pentru înregistrare .....	15
Figură 4.4 - Diagramă de flow pentru rezervarea unei mese .....	16
Figură 4.5 - Diagramă de flow pentru comanda și plata consumăției .....	18
Figură 4.6 - Diagramă de flow pentru gestionarea comenzi per chelner.....	19
Figură 4.7 - Diagramă de flow pentru gestionarea comenzi per chelner.....	20
Figură 5.1 - Diagrama aplicației QuickPlate.....	22
Figură 5.2 - Relațiile entităților bazei de date.....	23
Figură 5.3 - Diagrama claselor principale.....	26
Figură 5.4 - Fișierul Localizable.strings .....	27
Figură 5.5 – Diagrama fișierului Assets.xcassets .....	28
Figură 5.6 - Funcția de login a clasei FirebaseAuth.....	30
Figură 5.7 - Structura modulului Extras .....	31
Figură 5.8 - Utilizarea clasei AuthManager.....	32
Figură 5.9 - Diagramă de clase a modulului SignIn .....	34
Figură 5.10 - Diagrama de clase a modulului SignUp.....	35
Figură 5.11 - Diagrama de clase a modulului Map.....	36
Figură 5.12 - Diagrama de clase a modulului Restaurants .....	37
Figură 5.13 - Diagrama de clase a modulului RestaurantDetails.....	38
Figură 5.14 - Diagrama de clase a modulului TablesList .....	39
Figură 5.15 - Diagrama de clase a modulului UserProfile.....	40
Figură 5.16 - Diagrama de clase a modulului OrderView .....	41
Figură 5.17 - Diagrama de clase a modulului ClientOrder .....	42
Figură 5.18 - Stările unei comenzi .....	43
Figură 7.1 - Lista de simulatoare din Xcode.....	48
Figură 7.2 - Ecran de prezentare și ecranul de autentificare a aplicației .....	49
Figură 7.3 - Email de confirmare .....	50
Figură 7.4 - Ecranul de hartă.....	51
Figură 7.5 - Ecranul cu lista de restaurante.....	52
Figură 7.6 - Exemplu de rulare folosind simulatorul Xcode .....	53
 Tabel 3.1 - Diferențe între Toast și Square .....	8
Tabel 6.1 - Cazul de test a funcționalității de înregistrare a unui utilizator .....	44
Tabel 6.2 - Cazul de test pentru funcționalitatea de rezervare a unei mese .....	44
Tabel 6.3 - Cazul de test pentru algoritmul de echilibrare a comenzi .....	45
Tabel 6.4 - Cazul de test a cererii notei de plată.....	46
Tabel 8.1 - Comparație între QuickPlate și aplicații existente.....	54

### Anexa 3. Exemplu de cod

Functionalitatea de autentificare:

```

func doLogin(email: String = "", password: String = "", completion: @escaping (Result<String, StartupError>) -> Void) {
    Auth.auth().signIn(withEmail: email, password: password, completion: { result,
error in
        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.uuid))
        switch user.isEmailVerified {
            case true:
                print("Email is verified")
                completion(.success(1))
            case false:
                print("Email is not verified")
                completion(.failure(.emailExists))
        }
    })
}

```

Functionalitatea de trimitere a unei comenzi din clasa ClientOrderViewModel:

```

func sendOrder() {
    FSResColl.shared.getResNameThatHas(tableId: tableId) { name in
        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 in 0 ..< self.numberOrdered.count {
            if self.numberOrdered[index] > 0 {
                ids.append(self.foods[index].id ?? "")
                quan.append(self.numberOrdered[index])
                FSFoodsColl.shared.updateFoodstockWith(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()
}
}

```

**Functionalitatea de acceptare a unei comenzi de către 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 in
        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 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
    }
}
```

**Functionalitatea 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 in
        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])
        }
    }
}
```