**QUICKPLATE – APLICATIE MOBILA PENTRU GESTIONAREA REZERVARILOR SI COMENZILOR LA RESTAURANT**

LUCRARE DE LICENŢĂ

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

**2023**

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

Absolvent: **Ioan-Octavian STANCIU**

**QUICKPLATE – APLICATIE MOBILA PENTRU GESTIONAREA REZERVARILOR SI COMENZILOR LA RESTAURANT**

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

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

**Declarație pe propria răspundere privind**

**autenticitatea lucrării de licență**

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

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

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

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

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

Cuprins

[Capitolul 1. Introducere 1](#_Toc137055062)

[Capitolul 2. Obiectivele proiectului 2](#_Toc137055063)

[2.1. Obiectivul principal 2](#_Toc137055064)

[2.2. Obiective specifice 2](#_Toc137055065)

[Capitolul 3. Studiu bibliografic 4](#_Toc137055066)

[3.1. Avantajele iOS comparativ cu Android 4](#_Toc137055067)

[3.2. Aplicatii mobile de tipul food-delivery 4](#_Toc137055068)

[3.2.1. Aplicatia Zomato 5](#_Toc137055069)

[3.2.2. Aplicatia UberEats 5](#_Toc137055070)

[3.2.3. Aplicatia Foodpanda 6](#_Toc137055071)

[3.2.4. Asemanari si diferente intre Zomato, UberEats si Foodpanda 6](#_Toc137055072)

[3.3. Software de administrare al restaurantelor 7](#_Toc137055073)

[3.3.1. Toast POS 7](#_Toc137055074)

[3.3.2. Square POS 7](#_Toc137055075)

[3.3.3. Comparatie intre aplicatiile Toast si Square 7](#_Toc137055076)

[Capitolul 4. Analiză și fundamentare Teoretică 9](#_Toc137055077)

[4.1. Analiza problemei 9](#_Toc137055078)

[4.2. Cerinte functionale 10](#_Toc137055079)

[4.3. Cerinte non-functionale 11](#_Toc137055080)

[4.4. Modelul de date al aplicatiei 12](#_Toc137055081)

[4.5. Starile unei comenzi 12](#_Toc137055082)

[4.6. Cazuri de utilizare 13](#_Toc137055083)

[4.6.1. Inregistrare 13](#_Toc137055084)

[4.6.2. Rezervarea unei mese 14](#_Toc137055085)

[4.6.3. Comanda si plata consumatiei 16](#_Toc137055086)

[4.6.4. Procesarea comenzilor 17](#_Toc137055087)

[4.7. Algoritmul de echilibrare a comenzilor per chelner 19](#_Toc137055088)

[Capitolul 5. Proiectare de detaliu și implementare 20](#_Toc137055089)

[5.1. Proiectarea si implementarea arhitecturii 20](#_Toc137055090)

[5.2. Proiectarea si implementarea bazei de date 21](#_Toc137055091)

[5.2.1. Entitatea Food 22](#_Toc137055092)

[5.2.2. Entitatea MyUser 22](#_Toc137055093)

[5.2.3. Entitatea Order 23](#_Toc137055094)

[5.2.4. Entitatea Restaurant 24](#_Toc137055095)

[5.2.5. Entitatea Table 24](#_Toc137055096)

[5.3. Structura proiectului 24](#_Toc137055097)

[5.3.1. Structura fisierelor 25](#_Toc137055098)

[5.3.2. Modulul Extras 27](#_Toc137055099)

[5.3.3. Modulul LoginStates 30](#_Toc137055100)

[5.3.4. Modulul Features 32](#_Toc137055101)

[Capitolul 6. Testare și validare 43](#_Toc137055102)

[Capitolul 7. Manual de instalare si utilizare 46](#_Toc137055103)

[7.1. Resursele necesare 46](#_Toc137055104)

[7.1.1. Resurse hardware 46](#_Toc137055105)

[7.1.2. Resurse software 46](#_Toc137055106)

[7.2. Manual de instalare 46](#_Toc137055107)

[7.3. Manual de utilizare 47](#_Toc137055108)

[7.3.1. Manual de utilizare pentru client 49](#_Toc137055109)

[7.3.2. Manual de utilizare pentru ospatar si bucatar 51](#_Toc137055110)

[Capitolul 8. Concluzii 52](#_Toc137055111)

[8.1. Contributiile propria 52](#_Toc137055112)

[8.2. Dezvoltari ulteriorare 53](#_Toc137055113)

[Bibliografie 54](#_Toc137055114)

[Anexa 1. Glosar de termeni 55](#_Toc137055115)

[Anexa 2. Lista figurilor si tabelelor 56](#_Toc137055116)

[Anexa 3. Exemple de cod 56](#_Toc137055117)

# Introducere

Ce se scrie aici:

* Contextul proiectului
* Conturarea domeniului exact al temei

Contextul problemei (lipsa/instabilitatea/pret ridicat al fortei de munca in domeniul HORECA….)

Ce problema rezolvi?

Pentru cine este de valoare o aplicatie de genul asta

cs.utluj.ro template sinteza

# Obiectivele proiectului

## Obiectivul principal

Obiectivul principal al acestui proiect este de a dezvolta o aplicatie mobila iOS care are rolul de a ajuta utilizatorii in a rezerva o masa la un restaurant dorit pentru a evita aglomeratia si de a reduce riscul de a nu gasi o masa libera. Acestia pot observa pe harta integrata restaurantele din proximitatea lor pentru a aproxima distanta dintre ei si destinatie, o lista cu toate restaurantele pentru a cauta restaurantul dorit mai repede si pentru a rezerva o masa, si o pagina de profil pentru a confirma sosirea si restaurantele favorite.

Dintotdeauna a existat problema aglomeratiei in localuri care impune o dificultate in rezervarea unei mese in timpul saptamanii. Desi traim in „secolul tehnologiei” nu exista nici pana in ziua de astazi o aplicatie lipsita de probleme care sa rezolve acest lucru. Acest proiect are rolul de a economisi timpul oamenilor in rezervarea unei mese la un restaurant si de a nu depinde de eroare umana care poate sa intervina ca se doreste o rezervare efectuata prin intermediul unui apel telefonic.

## Obiective specifice

**Inregistrarea unor noi utilizatori:** Aplicatia ofera posibilitatea noilor utilizatori de a-si putea face un cont nou pe aplicatie cu scopul de a profita de tot ce ofera aceasta. Cu ajutorul acestei functionalitati, utilizatorul va fi capabil de a putea naviga prin aplicatie, de a-si alege restaurantele preferate, de a rezerva o masa si multe altele. Bineinteles, in functie de rolul ales functionalitatile pot sa difere de la un utilizator la altul.

**Confirmarea adresei de email:** La fiecare creare de cont nou, utilizatorul va primi pe adresa de email introdusa un email de confirmare pentru a finaliza inregistrarea si de a se asigura ca intr-adevar el este cel care doreste acest lucru.

**Vizualizarea restaurantelor pe harta:** Aceast proiect dispune de o harta integrata ce ofera posibilitatea utilizatorilor de a putea vizualiza restaurantele din proximitatea locatiei acestuia pentru a aproxima distanta ce trebuie parcursa in cazul in care la acel restaurant se doreste sa se mearga.

**Lista cu restaurantele:** Utilizatorul va avea acces la o lista interactiva cu toate restaurantele din aplicatie pentru a le vizualiza intr-un mod organizat si de a putea sa caute un restaurant dorit. In plus, fiecare restaurant are un buton interactiv cu care utilizatorul poate interactiona pentru a adauga unul sau mai multe restaurante in lista de favorite.

**Meniul unui restaurant:** In fiecare pagina de detalii a fiecarui restaurant se afla meniul acestuia pentru a lasa utilizatorii sa vizualizeze felurile de mancare si bauturile. Pe langa aceasta, din aceasta pagina se poate rezerva o masa la restaurantul respective prin alegerea unei date.

**Lista de restaurante favorite:** Aceasta lista se afla in pagina de profil pentru a avea acces mai usor si rapid direct la restaurantele preferate adaugate anterior, acest lucru economisind timp utilizatorului in cautarea si gasirea restaurantului pe care l-a accesat in trecut si ar vrea din nou sa reserve o masa la acesta.

**Lista rezervarilor:** Pe pagina de profil se pot gasi toate rezervarile facute de utilizator la restaurantele alese aratand numarul de persoane, ora si ziua fiecareia.

**Comanda din aplicatie:** Cand utilizatorul o sa confirme sosirea la restaurant acesta o sa fie redirectionat la o pagina speciala in care ii este prezentat meniul si poate sa comande folosind aplicatia, reducand timpul de asteptare a unui chelner pentru aducerea unui meniu si de a lua comanda.

**Modalitati de plata:** Dupa ce utilizatorul a terminat de comandat si doreste sa plateasca i se ofera mai multe posiblitati de plata precum, cash, card sau folosind aplicatia. Acest lucru ajuta la eficienta atat angajatilor restaurantului cat si a utilizatorului.

# Studiu bibliografic

## Avantajele iOS comparativ cu Android

iOS, sistemul de operare a telefoanelor iPhone de la Apple [1], renumit pentru designul sau elegant si interfata grafica intuitiva a adunat o multime de oameni curiosi. Unul dintre principalele motive pentru a alege iOS este integrarea sa perfecta cu ecosistemul hardware produs de Apple. Cum Android este folosit de mai multi producatori de telefoane inteligente, interfata utilizator difera de la unul la altul substantial, utilizatorii hotarandu-se mai greu in a lua decizia ce telefon sa isi achizitioneze.

Conform unui blog [2], datorita faptului ca iOS beneficiaza de faptul ca este o platforma cu sursa inchisa, procesul de ecranizare este mai strict la iPhone si utilizatorii ce folosesc acest dispozitiv sunt limitati la aplicatiile de pe App Store Apple ajuta la prevenirea descarcarii de virusi. Pe de alta parte, utilizatorii de Android au acces la mai multi distribuitori de aplicatii, dar nu toate aplicatiile oferite de acestia sunt verificate.

Din punct de vedere a limbajelor de programare, aplicatiile iOS se dezvolta folosind Swift, un limbaj unic creat de Apple, cu scopul de a fi la fel de rapid ca C, usor de citit si sigur de scris in comparatie cu aplicatiile Android ce se dezvolta folosind Java care are o sintaxa lunga si mai dificila de inteles, iar compilatorul nu este la fel de bine optimizat facand construirea si rularea aplicatiilor sa consume mai mult timp.

Cand vine vorba de fragmentare, Apple are un mare avantaj intrucat este o variatie mai mica intre dispozitivele lor in comparatie cu telefoanele ce folosesc Android. O experienta cat mai placuta si consistenta ii este oferita utilizatorului de iOS datorita functiilor de baza pe acest tip de dispozitiv, inclusiv experienta culorilor si animatiilor. In contrast cu acesta, aplicatiile ce ruleaza Android trebuie dezvoltate si testate cu mai multa atentie din cauza atator tipuri de dispozitive ce au ecrane diferite, timpii de rulare difera, rezolutia si asa mai deprate.

Cu toate acestea, conform unei statistici [3] facute in anul 2023, iOS detine din piata globala la finalul anului 2022 doar 27.58% pe cand Android detine 71.72% ceea ce inseamna ca majoritatea utilizatorilor folosesc Android. Acest fapt se poate datora faptului ca, pe langa faptul ca sunt mult mai multi dezvoltatori ce folosesc acest sistem de operare, utilizatorilor li se ofera si mai multe alegeri cand vine vorba de achizitionarea unui telefon nou.

## Aplicatii mobile de tipul food-delivery

HORECA [4] este un acronim care se refera la industria ospitalitatii, formata din hoteluri, restaurante si cafenele. Aceasta industrie este foarte importanta pentru economia globala si reprezinta un sector in continua dezvoltare, cu o varietate de servicii si produse destinate satisfacerii nevoilor turistilor si clientilor. HORECA este responsabila pentru crearea de locuri de munca, pentru promovarea turismului si pentru generarea de venituri. De asemenea, aceasta intdustrie este influentata de factori economici si de stilul de viata al oamenilor, ceea ce o face mereu in schimbare si adaptare.

Industria livrarilor de alimente a evoluat semnificativ in ultimii ani datorita popularitatii tot mai mari a smartphone-urilor si a dezvoltarii aplicatiilor mobile. Pana in prezent, livrarea produselor alimentare s-a limitat in mare masura la comenzile telefonica sau online, ncesitand adesea timp si efor suplimentar pentru a finaliza procesul de comanda si livrare. Odata cu aparitia aplicatiilor de livrare a alimentelor, clientii pot comanda mancare cu un click de pe un telefon mobil si le pot livra chiar la usa, oferind o experienta rapida si convenabila. Aceste aplicatii nu numai ca au simplificat procesul de comanda si livrare, dar au deschis si calea pentru noi afaceri si servicii de livrarede alimente, oferind clientilor acces la o gama mai larga de optiuni de luat masa. Astazi, aplicatiile de livrare a alimentelor sunt folosite in intreaga lume si evolueaza constant.

Potrivit unui raport realizat de Statista, in anul 2020, peste 1,5 miliarde de oameni din intreaga lume au comandat mancare online, iar acest numar este in continuea crestere. In State Unite ale Americii, de exemplu, peste 60% dintre consumatori comanda mancare online cel putin o data pe saptamana.

Conform unui blog[5], in anul 2022 industria de livrare a mancarii era estimata la o valoare de aproximativ de 760 de miliarde de dolari dintre care 300 erau doar din livrarea mancarii, nu a alimentelor. In acelasi timp, numarul de utilizatori a aplicatiilor de livrare a mancarii a crescut enorm cu un numar de peste 3 miliarde.

Aceasta crestere semnificativa a comenzilor online a schimbat modul in care restaurantele isi gestioneaza afacerile. In loc sa foloseasca meniuri traditionale in format fizic, multe restaurante au inceput sa treaca la meniuri virtuale, care sunt disponibile pe site-ul lor sau prin intermediul aplicatiilor de comanda de mancare. Acest lucru a permis oamenilor sa parcurga meniul in mod facil si sa aleaga mancarea preferata fara a fi nevoiti sa se afle fizic la restaurant.

In plus, comanda online de mancare a facut posibila si cresterea popularitatii serviciilor de livrare la domiciliu, cum ar fi Uber Eats [6], Deliveroo [7] sau Glovo [8] . Aceste servicii permit oamenilor sa comande mancare de la o varietate de restaurante diferite si sa o primeasca la usa lor intr-un timp foarte scurt.

### Aplicatia Zomato

Potrivit unei statistici[9] din anul 2022, Zomato a fost cea mai folosita aplicatie de tipul food-delivery din intreaga lume cu un numar de descarcari de peste 54 de milioane si cu un numar activi de utilizatori de 32.1 milioane in fiecare luna.

Zomato este un lant de restaurante indiene cu acoperire internationala si companie de tipul food-delivery fondata de Deepinder Goyal si Pankak Chaddah in anul 2008 [10]. In anul 2022, compania avea 3800 de angajati, iar veniturile erau aproximate la un total de 890 de milioane de dolari in anul 2023.

O functionalitate ce scoate Zamato in evidenta este abilitatea de a-i ajuta pe utilizatori sa descopere restaurante noi. Aplicatia se foloseste de preferintele utilizatorului, cautarile recente si istoricul comenzilor sau rezervarilor facute. Avand toate acestea la dispozitie, sistemul recomanda un nou loc unde utilizatorul ar putea lua cina sau de unde ar putea comanda.

### Aplicatia UberEats

UberEats este o aplicatie de livrare a mancarii lansata in anul 2014. Este folosita in peste 6000 de orase si 45 de tari, iar valoarea acesteia era estimata la undeva in jurul sumei de 8,3 miliarde de dolari in anul 2021 avand in continuare o crestere destul de rapida. Este o extensie a aplicatiei Uber ce ofera utilizatorilor posibilitatea de a comanda un mijloc de transport prin intermediul telefoanelor sau tabletelor doar prin cateva atingeri ale ecranului. UberEats are si un abonament de 10 dolari per luna, numit Eats Pass, care renunta la taxa de livrare pentru toate comenzile si ofera utilizatorului un discount de 5% pentru orice comanda de peste 15 dolari.

Potrivit statisticii mentionate si la subcapitolul anterior [11], aceasta a fost a doua cea mai folosita aplicatie cu 46,8 milioane de descarcari. Majoritatea utilizatorilor folosesc UberEats pentru marea varietate de restaurante din aplicatie, pentru interfata grafica placuta si usor de folosit si pentru posibilitatea de a putea lasa recenzii. De asemnea, aplicatia mai ofera promotii, reduceri sau chiar si oferte exclusive. Aceste lucruri au reusit sa isi mentina cat mai multi din clienti si sa ii atraga pe unii noi pentru a-si mari numarul de utilizatori.

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

### Aplicatia Foodpanda

Foodpanda este o aplicatie de livrare a mancarii si a cumparaturilor detinuta de multinationala germana Delivery Hero. A fost fondata in anul 2012, iar la un an dupa a fost lansata in Bangladesh si Romania, la acel moment compania avand peste 20000 de angajati. Aplicatia a avut cel mai mare succes in Asia, fiind la momentul actual cea mai folosita aplicatie in acea zona, exceptand China. [12]

Aceasta aplicatie, in comparatie cu celelalte, nu iese cu nimic in evidenta intrucat are toate functionalitatile de baza pe care ar trebui sa le aiba o aplicatie din aceasta categorie: interfata usor de inteles si intuitiva, o gama variata mare si diversa de restaurante, disponibilitate la nivel international, mai multe metode prin care poti sa achiti comanda si multe altele.

Foodpanda a avut succes mare din cauza ca fondatorii au ales foarte bine publicul tinta si anume cei din Asia unde, la momentul lansari, nu se stia de o asemenea aplicatie. Pe langa aceasta, in toata lumea in anul 2012 aplicatiile de acest tip nu aveau popularitatea pe care o aveau acum 5 ani.

### Asemanari si diferente intre Zomato, UberEats si Foodpanda

Zomato, UberEats si Foodpanda sunt toate foarte cunoscute ca aplicatii de tipul food-delivery si desi ofera servicii similare, exista mici diferente intre acestea:

* **Restaurantele partenere**: Zomato si Foodpanda au o varietate mai mare de restaurante in toate zonele in care sunt disponibile, pe cand UberEats tinde sa aiba mai multe parteneriate cu lanturile de restaurante mai populare
* **Interfata si experienta utilizator**: Zomato ofera un sistem robust de descoperire si recenzie a restaurantelor, in timp ce UberEats si Foodpanda se concentreaza mai mult pe experienta de livrare a alimentelor
* **Recomandari personalizate:** Zomato si Foodpanda ofera recomandari personalizate bazate pe preferintele utilizatorului, istoricul de comenzi si istoricul cautarilor pe cand UberEats se focuseaza mai mult pe promovarea restaurantelor mai populare sau care sunt in trending
* **Functionalitati unice**: Zomato se remarca prin functiile sale complete de descoperire a restaurantelor, inclusiv informatii detaliate, recenzii, evaluari si posibilitatea de a rezerva mese la restaurantele partenere. UberEats si Foodpanda se concentreaza in primul rand pe livrarea alimentelor, dar pot oferi functii suplimentare, cum ar fi urmarirea in timp real a comenzilor.

## Software de administrare al restaurantelor

### Toast POS

Toast este un software de administrare a restaurantelor fondat in anul 2012. Cu un numar de aproximativ 3172 de angajati, de 2,4 miliarde de dolari de venituri si 62000 [13] de restaurante ce folosesc acest software, face ca acesta sa fie cel mai folosit in categoria sa. Software-ul este folosit doar pe device-urile ce folosesc sistemul de operare Android si ofera un sistem „all-in-one”.

Initial a fost o aplicatie de consum cu focusul pe plati mobile, promotii si un aspect social care mai tarziu toate acestea au fost integrate cu sistemul de POS al restaurantelor.

Acest software se remarca prin mai multe functionalitati:

* **Comanda si checkout la masa**: Toast poate sa accepte comenzi si plati direct la masa prin intermediul ei. Acest lucru poate sa fie rau din moment ce nu poate procesa plati fara o conexiune la internet.
* **Rapoarte**: Programul permite fiecarui restaurant sa genereze rapoarte referitoare la costuri, numarul de alimente din inventar si altele. Cu ajutorul acestora, cel care se ocupa de administrarea restaurantului poate sa ia anumite decizii in functie de rezultatele din rapoarte.
* **Actualizarea meniului**: Fiecare restaurant poate sa isi faca modificari la meniu, iar acest lucru se actualizeaza pentru toti utilizatorii. Poate sa fie un lucru bun mai ales pentru acele restaurante ce isi actualizeaza meniul zilnic sau in fiecare sezon.

Pretul de baza pentru a folosi Toast este de 110 de dolari per luna, dar pe masura ce se adauga servicii, pretul poate sa atinga suma de 165 de dolari cel din urma adaugand Sling scheduling, Payroll & Team Management, Toast Pay Card, Payout si altele.

### Square POS

Square POS este un sistem de puncte de vanzare care da posibilitatea companiilor sa isi eficientizeze operatiunile si sa accepte plati cu usurinta. Dezvoltat de Square Inc., o companie lider de tehnologie financiara, Square POS ofera o gama larga de caracteristici si beneficii concepute pentru a spori eficienta. Companiile pot procesa platile rapid si in siguranta. Fie ca accepta carduri de credit si de debit, plati fara contact sau portofele mobile, Square ofera solutii hardware si software usor de utilizat pentru a facilita tranzactiile fara intreruperi.

Dincolo de procesarea platilor, Square POS ofera capabilitati robuste de gestionare a stocurilor. Companiile pot urmari cu usurinta nivelurile stocurilor, pot gestiona variatiile articolelor si pot primi alerte pentru stocul scazut. Aceste caracteristici permit companiilor sa isi optimizeze inventarul, sa evite epuizarea stocurilor si sa se asigure ca au la dispozitie produsele potrivite pentru a satisface cerintele clientilor.

### Comparatie intre aplicatiile Toast si Square

Conform unui blog [14] publicat de Forbes in 2023, alegerea dintre Toast si Square ar trebui sa fie facuta in functie de nevoia restaurantului de a folosi POS sau nu. Toast e o alegere excelenta pentru restaurantele cu mai multe locatii, drive-throughs, baruri pe cand Square este o optiune mai buna daca se doreste un sistem POS pentru a ajuta cu verificarea, statul de plata, scanarea codului de bare si altele. Acest lucru se datoreaza deoarece Toast are functionalitati care se concentreaza mai mult pe nevoile unui restaurant pe cand Square isi extinde serviciile pentru mai multe tipuri de afaceri.

In Tabelul 3.1 indica diferentele esentiale intre cele 2:

Tabel 3.1 - Diferente intre Toast si Square

|  |  |  |
| --- | --- | --- |
|  | **Toast** | **Square** |
| Taxe lunare | De la $0 pana la $165 | Versiune gratis sau $60 lunar |
| Taxa la plata | 2.99% plus $0.15 cu versiunea gratis  2.49% plus $0.15 cu un plan platit | 2.6% plus $0.10 per tranzactie |
| Cost hardware | Inclus in plata lunara | Primul cititor de carduri este gratis dupa care fiecare cost $10  Terminalele costa incepand cu suma de $299 |
| Relatii clienti | Suport telefonic la orice ora in orice zi, blog | Chat live, blog |
| Procesarea platilor incorporata | Da | Da |

# Analiză și fundamentare Teoretică

## Analiza problemei

Ideea implementarii unui astfel de proiect a plecat de la o problema reala cu care se confrunta tot cei care doresc sa petreaca timp cu prietenii in oras, si anume aglomeratia barurilor si restaurantelor, mai ales in weekend. Aceasta este o problema foarte mare, mai ales in marile orase romanesti precum Cluj-Napoca, Bucuresti, Timisoara, Iasi, Constanta.

Dupa studierea pietei aplicatiilor mobile destinate aducerii functionalitatilor de a rezolva aceasta problema s-a constatat ca nu exista nicio aplicatie mobila care sa contina toate caracteristicile necesare. Asadar, s-a impartit problema in 2 categorii: cum poate fi de folos aplicatia atat clientului care vrea sa rezerve o masa cat si restaurantului pentru a usura aglomeratia?

In primul rand, trebuie sa existe un public tinta, iar acesta este format de utilizatorii cu varsta cuprinsa in intervalul 18-35 de ani. Mai multe statistici arata ca acestia sunt cei mai frecventi utilizatori ai aplicatiilor de tipul food-delivery sau de rezervare a unei mese.

In al doilea rand, nu numai clientii trebuie sa profite de aplicatie, dar si restaurantele la care se pot rezerva mese pentru a simplifica procesul de comanda. In acest fel, chelnerul nu trebuie sa mearga la masa pentru a lua meniul, sa revina pentru a prelua comanda, apoi sa astepte pana cand aceasta se serveste pentru a o servii, iar la final pentru a genera bonul.

In ambele situatii prezentate anterior, fiecare utilizator trebuie sa aiba un cont. In momentul actual, numar aplicatiilor care nu dispun de o astfel de functionalitate este foarte mic. Bineinteles, dupa ce fiecare se autentifica fiecare, o sa fie redirectionati catre interfete utilizator destinate rolurilor lor: una pentru client, iar cealalta pentru chelner si bucatar.

Clientul trebuie mai intai sa vada locatia restaurantului pe harta pentru a stii cat de lung este drumul pana la destinatie. De asemenea, trebuie sa poata alege un restaurant dupa anumite caracteristici, iar cea mai buna este nota de rating. Acest lucru poate sa il determine pe utilizator sa afle mai multe informatii despre restaurant precum meniul, preturile si sa vada anumite poze cu localul respectiv.

S-a luat in considerare si cazul in care pe harta sunt afisate prea multe restaurante si devine foarte aglomerat si greu de urmarit. Ca rezolvare, s-a implementat o lista cu toate restaurantele in care utilizatorul are si optiunea de a cauta dupa nume un restaurant stiut. Aceasta il ajuta pe utilizator sa sa vizualizeze intr-un mod mai organizat si mai usor de interactionat comparativ cu harta. Din aceasta lista, clientul poate intra pe pagina de detalii a restaurantului pentru vizualizarea meniului cu scopul de a-l ajuta in hotararea alegerii locului. In cazul in care acesta este multumit, poate sa rezerve o masa din pagina de detalii. In plus, clientul isi poate adauga restaurante la lista de favorite pentru o navigare extrem de simplificata pe viitor.

Daca fiecare utilizator are cont atunci trebuie sa aiba si o pagina de profil. Clientul are trei sectiuni:

* Sectiunea in care observa datele personale si poate sa se deconecteze din aplicatie
* Sectiunea in care poate observa toate restaurantele favorite pe care le-a adaugat pe parcursul folosirii aplicatiei
* Sectiunea in care isi poate vizualiza rezervarile

In momentul in care clientul isi confirma sosirea la restaurantul respectiv, acesta este redirectionat catre un ecran special din care poate comanda, iar dupa cel putin o comandare poate cere nota de plata.

Pe de alta parte, chelnerul si bucatarul au doar ecranul de profil in care sunt doua sectiuni:

* Sectiunea in care fiecare isi pot vizualiza datele personala si se pot deconecta
* Sectiunea in care vin comenzile

Dupa ce toate acestea au fost decise, s-a inceput un design in Figma, unealta de design folosita pentru proiectare si prototipizare a interfetelor utilizator si a experientelor acestora, bazata pe cloud care foloseste instrumente de editare vectoriale, biblioteci de design, prototipuri interactive si capabilitati de transfer al dezvoltatorilor. In Figura 4.5 Se poate observa prototipizarea si proiectarea ecranelor dezvoltate in tool-ul Figma.

A screenshot of a phone

Description automatically generated with medium confidence

Figură 4.1 - Design prototipizat in Figma

## Cerinte functionale

Fiind o aplicatie ce se ocupa de gestionarea meselor si rezervarilor la restaurante, utilizatorii trebuie sa aiba roluri diferite, atat ca si client cat si ca angajat. Datorita acestui fapt, a trebuit adaugata o metoda de securitate pentru conturile duplicate si pentru a verifica intr-adevar daca utilizatorul respectiv doreste sa isi foloseasca propria adresa cu acest scop.

Ca un client sa poata sa isi faca o rezervare, trebuie sa isi aleaga restaurantul dorit, asadar o lista interactiva prin care isi poate indeplini aceasta dorinta este perfecta. De asemenea, clientului ii trebuie oferita optiunea de a alege metoda de plata prin care doreste sa achite consumatia. Din moment ce aplicatia are ca scop fluidizarea interactiunii client-ospatar, nu se poate ca nota de plata sa fie ceruta in aceeasi modalitate clasica si anume de a chema chelnerul la masa.

Acestea fiind enumerate, urmatoarele cerinte functionale reprezinta functionalitatile aplicatiei QuickPlate:

* **Inregistrarea utilizatorului avand rol de client sau de angajat**. Aplicatia permite noilor utilizatori sa isi aleaga rolul la crearea contului. Acest lucru le ofera functionalitati diferite in functie de alegerea facuta. Cei cu rolul de client vor beneficia de o harta pe care sunt puse restaurantele, o lista cu toate restaurantele si o pagina de profil pe cand cei inregistrati ca si chelner sau bucatar vor avea doar pagina de profil.
* **Confirmarea adresei de email**. Cand un utilizator nou doreste sa isi creeze un cont pe aplicatie, acesta v-a trebui sa introduca o adresa de email ce va fi folosita si pentru autentificare. Acesta v-a primi un mail de confirmare pe adresa de email folosita pentru a verifica daca intr-adevar el este cel care doreste crearea contului. Acest lucru este o metoda simpla de securitate ce inspira incredere unui nou client.
* **Vizualizarea restaurantelor.** Cand utilizatorul a ales rolul de client acest v-a putea sa vizualizeze restaurantele din aplicatie atat pe o harta integrate cat si intr-o lista. Acest lucru il ajuta de a aproxima distanta de la locatia sa pana la restaurantul ales si de a cauta mult mai repede restaurantul in comparatie cu harta.
* **Rezervarea unei mese.** Utilizatorul cu rolul de client poate sa rezerve o masa direct din pagina de detalii a restaurantului. Acesta trebuie sa aleaga masa in functie de numarul de clienti doriti si ora si ziua in care doreste sa mearga.
* **Metodele de plata.** Clientului i se ofera posibilitatea de a plati prin mai multe moduri cum ar fi cash, cu cardul sau folosind direct aplicatia. Acesta ajuta la economisirea timpului de asteptare a chelnerului de a-l antetiona ca se doreste plata comenzii.

**Echilibrarea comenzilor per chelner.** Aplicatia dispune de o functionalitate care vine in ajutorul chelnerilor de a nu-i lasa sa accepte prea multe comenzi, lucru ce poate sa ii incarce prea mult comparativ cu colegii lor. Acesti au un numar limitat de comenzi ce pot fi acceptati de ei si se reseteaza in care comanda a fost livrata cu succes.

## Cerinte non-functionale

In timp ce cerintele functionale ale unui sistem definesc comportamentul si functionalitatea sa, cerintele non-functionale se concentreaza pe calitatile sistemului care nu sunt direct legate de functionalitatea sa. Aceste cerinte sunt la fel de importante ca si cele functionale deoarece acestea. Pot afecta in mod semnificativ experienta utilizatorilor si pot determina gradul de satisfactie al acestora.

Urmatoarele cerinte prezentate sunt cele non-functionale ale aplicatiei QuickPlate:

* **Securitatea**. Aceasta este pe departe cea mai importanta cerinta non-functionala din moment ce aplicatia QuickPlate retine informatii despre utilizatori. Din fericire, Firebase are mecanisme implementate ce se ocupa de acest lucru precum inregistrarea folosind adresa de email si regulile pentru accesul la baza de date.
* **Utilizabilitatea.** Acesta cerinta este prima cu care utilizatorul se intalneste deoarece reprezinta gradul de satisfactie la interactiunea acestuia cu sistemul. Flow-ul trebuie sa fie fluent si cursiv, butoanele sa fie sugestive, fiecare schimbare de ecran sa fie intuitiva, iconitele sa sugereze ce reprezinta, toate acestea ajuta la ridicarea nivelului de satisfactie a utilizatorului pentru a-l determina pe acesta la refolosirea aplicatiei pe viitor.
* **Extensibilitate.** Posibilitatea de a extinde functionalitatile dezvoltate este un lucru crucial in dezvoltarea aplicatiilor. Ce o sa fie implementat o sa stea la baza functionalitatilor mai complicate si o sa ofere cel putin minimul de implementare ce o sa fie necesar pentru o dezvoltare mai complexa.

**Portabilitatea.** Aceasta cerinta este valida datorita faptului ca este destinata dispozitivelor ce ruleaza sistemul de operare iOS sau iPadOS, iar versiunea este cel putin 16.0. De exemplu, daca aplicatia ruleaza pe un iPhone 14 cu versiunea iOS 16.3 atunci poate rula si pe un iPhone X cu versiunea iOS 16.0.

## Modelul de date al aplicatiei

Principalele entitati care formeaza modelul de date sunt:

* Utilizatorii: Cea mai importanta entitate existenta in aplicatie. Aceasta trebuie sa retina date esentiale despre utilizator precum numele, prenumele, parola, adresa de email, dar si alte informatii precum restaurantele preferate, comenzile plasat si altele.
* Restaurantele: Entitate de baza in aplicatie, cu ajutorul careia utilizatorul isi poate alege locul unde vrea sa serveasca masa, sa rezerve o masa sau sa adauga in lista de restaurante preferate. De asemenea, cu ajutorul acesteia clientul poate vizualiza aproxima distanta de la propria locatie pana la un restaurant gasit pe harta. Cele mai importate date pe care trebuie sa le retina sunt numele, orele si zilele in care acesta este deschis , media evaluarilor efectuate de utilizatori si adresa locatiei.
* Comenzile: Entitati folosite de catre toate tipurile de utilizator in momentul in care un client plaseaza o comanda, in urma efectuarii unei rezervari la o masa. Aceasta trebuie sa contina detalii despre pretul total, ce feluri de mancare s-au comandat si cantitatile fiecareia.
* Felul de mancare: Entitate ce contine numele, ingredientele si pretul unui fel de mancare cuprins intr-un meniu aceasta fiind necesara in special pentru cazul in care un client doreste sa plaseze o comanda.
* Meniul: Este format din mai multe feluri de mancare si este creat cu scopul de a corespunde unui restaurant pentru a-i permite utilizatorului sa se hotarasca daca intr-adevar doreste o rezervare sau nu la acel restaurant.
* Mesele: Entitati folositoare clientilor in a-i ajuta de a rezerva o masa la un restaurant dorit. Contin informatii precum numarul de persoane disponibile, daca este sau nu deja rezervata si numele restaurantului de care apartin.

## Starile unei comenzi

Cu scopul de a fluidiza plasarea si servirea unei comenzi a fost implementat ca o comanda sa treaca prin diferite stari. Acest lucru este benefic atat clientului cat si angajatilor.

* **In asteptare**: Aceasta este starea initiala a fiecarei comenzi plasate. O comanda se afla in aceasta stare atunci cand utilizatorul plaseaza comanda si este trimisa la angajatii restaurantului
* **In preparare**: Aceasta stare apare in momentul in care chelnerul accepta comanda si este trimisa bucatarului
* **Pregatita**: Aceasta stare apare atunci cand bucatarul confirma terminarea prepararii comenzii
* **Servita**: omanda se afla in ultima stare dupa ce aceasta a fost servita utilizatorului, confirmata de catre chelner

De asemenea, cand clientul plaseaza comanda, aceasta este salvata in baza de date, iar daca chelnerul o anuleaza in starea doi sau patru atunci va fi stearsa din Firestore.

In Figura 4.6 Se poate observa o diagrama de stari a unei comenzi:

A picture containing text, diagram, circle, screenshot

Description automatically generated

Figură 4.2 - Diagrama de stari a unei comenzi

## Cazuri de utilizare

In aceasta sectiunea vor fi prezentate patru cazuri de utilizare a aplicatiei QuickPlate fiecare avand o descriere si o digrama de desfasurare, dar mai intai se vor prezenta cele trei tipuri de utilizatori ce pot folosi aceasta aplicatie:

* **Clientul**. Acesta o sa fie cel mai important si o sa beneficieze de majoritatea functionalitatilor ce le ofera aplicatia. Acesta o sa poata sa vizualizeze harta cu restaurantele, sa rezerve o masa, sa comande si sa plateasca.
* **Ospatarul**. Acest tip de utilizator o sa se ocupe de gestionarea comenzilor, prin confirmarea si servirea acestora. Din punct de vedere a numarului de functionalitati de care acesta beneficiaza, nu este unul mare intrucat scopul lui nu necesita folosinta majoritatii acestora.
* **Bucatarul**. Ca si rolul de ospatar, functionalitatile nu sunt numeroase, iar gestionarea comenzilor este singura actiune pe care acesta o poate face.

### Inregistrare

Aceasta actiune este efectuata de fiecare utilizator ce are sau nu un cont in aplicatia QuickPlate. Daca are atunci trebuie doar sa se autentifice, iar in caz contrar sa se inregistreze.

**Actor:** Clientul

**Descriere:** Clientul doreste sa profite de functionalitatile aplicatiei QuickPlate

**Preconditii:**

* Utilizatorul are o conexiune buna la internet
* Utilizatorul are aplicatia instalata

**Postconditii:**

* Utilizatorul o sa fie redirectionat la pagina principala a aplicatiei

**Scenariul de succes:**

* Utilizatorul deschide aplicatia si este redirectionat la pagina de Sign In
* Utilizatorul nu are cont, dar doreste sa isi faca fiind redirectionat la pagina de Sign Up
* Utilizatorul completeaza toate campurile cu datele dorite
* Nu exista nicio data gresita sau deja folosita de alt utilizator
* Utilizatorul este redirectionat inapoi la pagina de Sign In
* Utilizatorul isi introduce credentialele
* Credentialele sunt valide
* Utilizatorul este redirectionat la pagina principala

**Scenarii alternative:**

* Utilizatorul are cont, isi introduce credentialele, dar sunt incorecte si este afisat un mesaj de eroare
* Utilizatorul nu are cont si trebuie sa isi creeze unul, iar cel putin o data introdusa este invalida
* Utilizatorul nu are o conexiune buna la internet, iar verificarile datelor nu se pot efectua

A picture containing text, screenshot, font, design

Description automatically generated

Figură 4.3 - Diagrama de flow pentru inregistrare

### Rezervarea unei mese

Aceasta actiune este destinata doar utilizatorului cu rol de client intrucat ospatarul si bucatarul se ocupa doar de gestionarea comenzilor.

**Actor:** Clientul

**Descriere:** Clientul doreste sa rezerve o masa la restaurantul dorit

**Preconditii:**

* Utilizatorul are o conexiune buna la internet
* Utilizatorul are aplicatia instalata
* Utilizatorul are cont cu rolul de „Client”
* Utilizatorul este autentificat cu acel cont in aplicatie

**Postconditii:**

* Rezervarea apare in lista de rezervari de pe pagina de profil al utilizatorului

**Scenariul de succes:**

* Utilizatorul selecteaza pagina care contine lista de restaurante
* Utilizatorul selecteaza restaurantul dorit
* Utilizatorul vizualizeaza meniul restaurantului
* Utilizatorului ii place meniul
* Utilizatorul doreste sa isi rezerve o masa
* Masa este libera la ora si ziua aleasa de catre utilizator
* Utilizatorul poate observa rezervarea facuta pe pagina de profil in sectiunea de „Booked Tables”

**Scenarii alternative:**

* Utilizatorul nu are o conexiune buna la internet si nu i se incarca lista cu restaurantele
* Utilizatorului nu ii place meniul restaurantului selectat rezultand in cautarea altui restaurant
* Ora si ziua rezervarii nu este libera ceea ce inseamna ca utilizatorul trebuie sa aleaga alta zi, ora sau sa schimbe atat ora cat si ziua

A screenshot of a phone

Description automatically generated with medium confidence

Figură 4.4 - Diagrama de flow pentru rezervarea unei mese

### Comanda si plata consumatiei

Functionalitate destinata utilizatorilor cu rol de client pentru a plasa o comanda in urma efectuarii unei rezervari la un restaurant dorit, iar la final li se ofera posibilitatea de a alege optiunea prin care acestia vor sa achite nota de plata.

**Actor:** Clientul

**Descriere:** Clientul doreste sa comande din aplicatie

**Preconditii:**

* Utilizatorul are o conexiune buna la internet
* Utilizatorul are aplicatia instalata si este autentificat
* Utilizatorul este pe pagina de profil in sectiunea de „Booked tables”
* Utilizatorul are o rezervare facuta la restaurantul respectiv
* Utilizatorul confirma ca a ajuns

**Postconditii:**

* Utilizatorul a reusit cu succes sa selecteze metoda de plata dorita

**Scenariul de succes:**

* Utilizatorul este pe pagina de profil
* Utilizatorul verifica daca rezervarea facuta este in sectiunea de „Booked tables”
* Utilizatorul confirma sosirea si este redirectionat la ecranul dedicat comenzii
* Utilizatorul adauga produsele dorite pentru comanda
* Utilizatorul trimite comanda angajatilor din restaurant
* Utilizatorul nu mai vrea sa comande nimic altceva si doreste sa achite nota de plata
* Utilizatorul a dat cel putin o comanda la restaurant
* Utilizatorul alege metoda prin care doreste sa achite nota de plata

**Scenarii alternative:**

* Utilizatorul nu are o conexiune buna la internet si nu i se incarca lista cu rezervarile dorite
* Utilizatorul nu are nicio rezervare la restaurantul respectiv facandu-l sa isi rezerve o masa
* Utilizatorul nu adauga niciun produs inainte de a trimite comanda catre restaurant

Utilizatorul nu a trimis nicio comanda inainte de a cere nota de plata

A picture containing screenshot, text, graphic design, design

Description automatically generated

Figură 4.5 - Diagrama de flow pentru comanda si plata consumatiei

### Procesarea comenzilor

Aceasta functionalitate este destinata numai utilizatorilor cu rol de ospatar si bucatar intrucat un client nu trebuie sa stie ce se intampla cu o comanda dupa plasarea acesteia.

**Actor:** Angajatii restaurantului

**Descriere:** Clientul trimite una sau mai multe comenzi la restaurant si angajatii se ocupa de procesarea ei

**Preconditii:**

* Angajatii au aplicatia instalata
* Angajatii trebuie sa aiba un cont cu rolul de „Waiter” pentru chelner sau „Cook” pentru bucatar
* Angajatii trebuie sa aiba o conexiune buna la internet

**Postconditii:**

* Chelnerii livreaza comanda la masa si confirma livrarea din aplicatie

**Scenariul de succes:**

* Angajatul este inregistrat in aplicatie si este redirectionat la pagina de profil
* Chelnerul primeste o noua comanda
* Nu intervine niciun motiv pentru care comanda ar trebui sa fie anulata
* Chelnerul mai poate accepta comenzi
* Comanda s-a terminat de preparat, lucru confirmat de bucatar
* Nu intervine nici de data asta un motiv pentru care comanda ar trebui sa fie anulata
* Comanda este livrata si confirmata din aplicatie

**Scenarii alternative:**

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

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

A screenshot of a computer screen

Description automatically generated with low confidence

Figură 4.6 - Diagrama de flow pentru pentru comenzilor

## Algoritmul de echilibrare a comenzilor per chelner

Aplicatia urmareste, de asemenea, sa faciliteze procesul de comanda si servire pentru restaurante, astfel incat ar trebui luat in considerare si potentialul volum mare de munca al angajatilor. Un singur chelner nu poate prelua comenzi. Pentru a evita acest caz, a fost implementat un algoritm care retine numarul de comenzi pe care le-a luat fiecare ospatar, iar daca acest numar este depasit, acesta nu poate fi preluat pana cand comenzile pe care le-a luat deja nu sunt servite.

Exista o variabila in memoria cache a telefonului care stocheaza numarul de comenzi. Creste atunci cand o comanda este acceptata si scade cand este livrata. Acest lucru permite fiecarui ospatar sa primeasca acelasi numar de comenzi mentinand in acelasi timp echilibrul.

Acest lucru este posibil prin dezactivarea unor elemente ce tin de interfata utilizator pentru a-i permite ospatarului doar sa confirme servirea unei comenzi. Referitor la subcapitolul precedent, cand se atinge limita maxima de acceptare a comenzilor, ospatarului ii este permis sa treaca comanda din starea trei in patru si ii este interzis trecerea comenzii din starea unu in starea doi.

De asemenea, la cererea manager-ului restaurantului, aceasta limita de acceptare se poate schimba facand numarul mai mare sau mai mic, depinzand de preferinte. In figura 4.7 se poate observa o diagrama de flow pentru o vizualizare mai simpla a acestui algoritm.

A screenshot of a cell phone

Description automatically generated with medium confidence

Figură 4.7 - Diagrama de flow pentru gestionarea comenzilor per chelner

# Proiectare de detaliu și implementare

## Proiectarea si implementarea arhitecturii

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

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

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

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

Avantajele alegerii de a folosi MVVM in loc de MVC mai pot fi posibilitatea de structurare mai buna a codului, disponibilitatea de a putea refolosi anumite componente de interfate datorita separarii logicii de date fata logica de actualizare a interfetei utilizator, mai multe clasa View putand sa aiba acelasi View-Model si oferirea flexibilitatii si extinderii claselor usurand introducerea modificarilor sau adaugarea de noi functii la aplicatie.

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

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

In Figura 5.1 se poate observa arhitectura aplicatiei QuickPlate.

A picture containing text, screenshot, diagram, design

Description automatically generated

Figură 5.1 - Diagrama aplicatiei QuickPlate

## Proiectarea si implementarea bazei de date

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

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

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

In Figura 5.3 se pot observa relatiile dintre cele cinci entitati enumerate anterior.

A screenshot of a computer screen

Description automatically generated with low confidence

Figură 5.2 - Relatiile entitatilor bazei de date

### Entitatea Food

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

Aceasta entitate nu ar referinte la alte entitati.

* id: UID ce reprezinta id-ul felului de mancare
* foodName: String ce reprezinta numele felului de mancare
* ingredients: Tablou unidimensional de string-uri cu numele ingredientelor ce sunt in felul de mancare
* price: Numar real cu doua zecimale ce reprezinta pretul felului de mancare
* stock: Numar intreg ce reprezinta numarul din inventar al acelui fel de mancare

### Entitatea MyUser

Aceasta este entitatea esentiala aplicatiei intrucat reprezinta datele fiecarui utilizator al aplicatiei. Folosind datele acestuia se pot efectua rezervari, comenzi, autentificare, autorizare, deconectare, adaugarea de restaurante preferate, toate operatiile ce tin de utilizatorul unei aplicatii. Fara aceasta, aplicatia nu ar putea sa functioneze.

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

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

### Entitatea Order

Aceasta entitate este folosita doar in cazul in care un client doreste sa plaseze o comanda. In momentul crearii acesteia, sigur o sa aiba o referinta catre entitatea MyUser pentru a retine id-ul utilizatorului care a plasat comanda, catre entitatea Table pentru ca ospatarii sa stie la ce masa trebuie livrata comanda si nu in ultimul rand catre entitatea Food pentru a stii ce contine comanda.

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

### Entitatea Restaurant

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

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

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

### Entitatea Table

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

* id: String ce reprezinta id-ul mesei
* booked: Valoare booleana care indica daca masa este sau nu rezervata. True pentru cazul in care este rezervata si false cand nu este.
* hourBooked: String care reprezinta ora de la care este facuta rezervarea. Nu are o valoare retinuta daca nu este rezervata.
* nrPersons: Valoare intreaga care indica numarul maxim de persoane care pot sta la masa
* tableNumber: Valoare intreaga pentru a arata numarul mesei. Folositoare pentru opatarii pentru a stii ce masa este rezervata
* day: String pentru a indica ziua in care este rezervata. Daca nu este, atunci nu este nicio valoare.
* userId: String care retine id-ul utilizatorului care rezerva masa. Daca nu este rezervata, atunci ea o sa aiba o valoare goala.

## Structura proiectului

Pentru o intelegere cat mai usoara a claselor principale folosite, figura 5.3 prezinta o diagrama cu cele mai importante View-uri, unele afisandu-se doar in functie de rolul utilizatorului.

A screenshot of a computer screen

Description automatically generated with low confidence

Figură 5.3 - Diagrama claselor principale

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

### Structura fisierelor

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

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

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

A screenshot of a computer

Description automatically generated

Figură 5.4 - Fisierul Lozalizable.strings

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

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

A screenshot of a computer

Description automatically generated

Figură 5.5 - Pachetul Assets.xcassets

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

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

* Daca utilizatorul nu este autentificat atunci el va fi redirectionat catre ecranul destinate functionalitatii de autentificare
* Daca utilizatorul este autentificat si are rol de client atunci o sa se afiseze ecranul cu functionalitatile destinate clientului
* Daca utilizatorul este autentificat si are rol de ospatar atunci o sa fie redirectionat catre pagina destinate acestui tip de utilizator
* Daca utilizatorul este autentificat si are rol de bucatar atunci o sa ii fie afisat ecranul destinat acestui rol

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

### Modulul Extras

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

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

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

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

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

A screen shot of a computer program

Description automatically generated with low confidence

Figură 5.6 - Functia de login a clasei FirebaseEmailAuth

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

In submodulul Models sunt toate entitatile definite la subcapitolul 5.2. MyUser, Restaurant, Food, Table si Order. Toate corespund cu exacitate descrierilor facute la subcapitolul mentionat si reprezinta continutul datelor ce sunt stoacate in intreaga aplicatie.

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

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

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

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

* Color+CustomColor.swift: este creata o extensie la structura predefinita Color pentru a adauga culorile personalizate din pachetul Assets.xcassets. Fiecare culoare este definita ca o constanta statica pentru a putea fi accesata de oriunde din aplicatie in orice moment.
* LocalizedStringKey+String.swift: sunt doua extensii, una pentru LocalizedStringKey si cealalta pentru String pentru a usura localizarea sirurilor de caractere pe tot parcursul aplicatiei.
* CLLocationCoordinate2D+Extension.swift: O extensie la structura CLLocationCoordinate2D pentru a putea fi encodata, decodata si comparata egalitatea cu o structura de tipul GeoPoint. Pentru aceasta ultima functionalitate a trebuit crearea unei functii denumita isEqualTo ce primeste ca parametru o variabila de tipul GeoPoint si returneaza o valoarea booleana in cazul in care longitudinile sunt egale intre ele, respectiv latitudinile.
* GeoPoint+Extensions.swift: O extindere la tipul de date GeoPoint pentru a crea o functie ce returneaza o structura de tipul CLLocationCoordinate2D avand aceleasi coordonate cu parametrul de tipul GeoPoint.

In figura 5.7 se poate observa structura modulului prezentat in acest subcapitol.

A screenshot of a computer

Description automatically generated with medium confidence

Figură 5.7 - Structura modulului Extras

### Modulul LoginStates

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

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

* notSignedIn: caz in care utilizatorul nu este autentificat si ii este prezentat ecranul de Sign In
* clientSignedIn: caz in care utilizatorul este deja autentificat si are rolul de client
* waiterSignedIn: caz in care utilizatorul este deja autentificat si are rolul de ospatar
* cookSignedIn: caz in care utilizatorul este deja autentificat si are rolul de bucatar

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

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

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

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

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

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

A screenshot of a computer program

Description automatically generated with low confidence

Figură 5.8 - Utilizarea clasei AuthManager

### Modulul Features

Modulul Features este cel cu cele mai multe subdirectoare si clase ce sunt folosite in intreagul proiect. Se poate spune ca acesta este centrul aplicatiei QuickPlate intrucat majoraitatea functionalitatilor sunt cuprinse in acest loc.

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

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

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

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

In figura 5.9 este prezentata diagrama de clase a acestui modul.

A picture containing diagram, text, line, plan

Description automatically generated

Figură 5.9 - Diagrama de clase a modulului SignIn

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

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

Posibilele erori ce pot sa apara sunt urmatoarele:

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

In figura 5.10 este prezentata diagrama de clase a acestui modul.

A picture containing diagram, text, screenshot, line

Description automatically generated

Figură 5.10 - Diagrama de clase a modulului SignUp

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

* 1. Modulul TabView

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

* 1. Modulul Map

Contine logica si interfata utilizator pentru partea de harta din QPTabView. Cu ajutorul framework-ului MapKit dezvoltat de Apple, integrarea unei harti in aplicatie a fost una foarte simpla. In clasa MapView se foloseste un View din acest framework, Map, care ii trebuie trimisi anumiti parametrii pentru a avea acces la locatia utilizatorului in timp real si pentru a arata sau nu locatia acestuia pe harta. Harta contine in partea de sus un buton interactiv care atunci cand este apasat centreaza harta pe coordonatele utilizatorului. In momentul in care se apasa acel buton, se apeleaza metoda de startUpdatingLocation() din locationManager pentru a incepe actualizarea locatiei utilizatorului. Atunci cand se detecteaza o noua pozitie, se apeleaza automat metoda didUpdateLocation din clasa LocationManager care verifica cu ajutorul unei instructiuni if daca ultima locatie (coordonatele fiind retinute in proprietatea lastLocation) difera de ultima valoare din tabloul unidimensional locations (parametru al acestei metode ca ultima valoarea coordonatele ultimei locatii actualizate). Daca difera, atunci se actualizeaza regiunea cu noile coordonate detectate, se actualizeaza proprietatea lastLocation si se opreste actualizarea continua a locatiei utilizatorului prin apelul metode stopUpdatingLocation().

Mai contine si clasa LocationManager ce se conformeaza la protocolul CLLocationManagerDelegate, iar acesta se ocupa de logica mentionata in paragraful anterior. Delegate este un model de design de baza in aplicatiile iOS intrucat multe functionalitati predefinite trebuie sa se conformeze la asemenea protocoale. Acesta are un delegat, un obiect, care contine o metoda sau un set de metode pe care obiectul delegat le poate implementa, si obiectul delegat este cel care implementeaza metodele definite de delegat. Obiectul delegat este inregistrat sau asignat catre delegat, permitandu-i sa primeasca si sa gestioneze evenimente sau sa furnizeze anumite functionalitati. In cazul acesta, atat delegatul cat si obiectul delegat este LocationManager deoarece este folosit sub tipul unui View-Model, iar toate schimbarile din acesta o sa fie publicat in View-ul MapView.

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

A picture containing diagram, line, screenshot, plan

Description automatically generated

Figură 5.11 - Diagrama de clase a modulului Map

* 1. Modulul Restaurants

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

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

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

A picture containing text, screenshot, font, design

Description automatically generated

Figură 5.12 - Diagrama de clase a modulului Restaurants

* 1. Modulul RestaurantDetails

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

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

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

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

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

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

A diagram of a restaurant

Description automatically generated with low confidence

Figură 5.13 - Diagrama de clase a modulului RestaurantDetails

* 1. Modulul TablesList

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

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

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

In figura urmatoare, 5.14, este prezentata diagrama de clase a modulului TablesList.

A picture containing diagram, text, line, number

Description automatically generated

Figură 5.14 - Diagrama de clase a modulului TablesList

* 1. Modulul UserProfile

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

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

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

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

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

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

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

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

In figura 5.15 este prezentata diagrama de clase a modulului UserProfile.

A picture containing diagram, text, line, plan

Description automatically generated

Figură 5.15 - Diagrama de clase a modulului UserProfile

* 1. Modulul OrderView

In acest modul, se formeaza View-ul pentru o comanda, OrderCardView, cu logica separata in OrderCardViewModel. Acesta este folosit pe ecranele destinate bucatarilor si ospatarilor pentru a afisa statusul comenzii. View-Model-ul are doar doua functii:

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

OrderCardView este folosit doar in ecranul destinat utilizatorilor cu rol de ospatar, respectiv bucatar, in care se afiseaza numarul mesei de la care s-a plasat comanda, fiecare fel de mancare cu cantitatea comandata si pretul ei, iar la final costul total al comenzii. In figura 5.16 este prezentata diagrama de clase a acestui modul.

A picture containing text, screenshot, font, design

Description automatically generated

Figură 5.16 - Diagrama de clase a modulului OrderView

* 1. Modulul ClientOrder

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

A screenshot of a computer

Description automatically generated with low confidence

Figură 5.17 - Diagrama de clase a modulului ClientOrder

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

* In sectiunea de informatii a utilizatorului se mai afiseaza un camp in plus cu numele restaurantului unde acestia lucreaza
* In sectiunea de jos se afiseaza o singura lista cu comenzile, singura diferenta intre acestea este ca pentru ospatar se afieaza cand comanda este in prima stare sau a treia pe cand bucatarului i se afiseaza comenzile ce sunt in cea de a doua stare

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

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

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

A screenshot of a phone

Description automatically generated with low confidence

Figură 5.18 - Starile unei comenzi

# Testare și validare

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

In tabelul 6.1 se va prezenta cazul de test pentru functionalitatea de inregistrare. Aceasta este folositoare in momentul in care un utilizatorul doreste sa beneficieze de functionalitatile aplicatiei in functie de rolul pe care il detine (client, ospatar sau chelner).

**Preconditii**: Utilizatorul trebuie sa aiba aplicatia instalata

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

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

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

In urmatorul tabel, 6.2, se va prezenta cazul de test pentru functionlitatea de rezervare a unei mese. Functionalitate dedicata doar utilizatorului de tip client si dupa ce acesta s-a hotarat sa rezerve o masa la un restaurant ales, in urma interactionarii cu lista de restaurante si pagina de detalii a acestuia.

**Preconditii**: Utilizatorul trebuie sa aiba rol de client si sa fie deja autentificat in aplicatie

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

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

In tabelul 6.3 este prezentat cazul de testare pentru functionalitatea de echilibrare a comenzilor. Pentru acest caz va fi nevoie de un client, doi ospatari (ospatar 1 si ospatar 2) si un bucatar.

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

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

|  |  |
| --- | --- |
| **Actiune** | **Rezultat** |
| Clientul apasa butonul verde „Confirma sosirea” | Clientului ii este prezentat ecranul din care poate plasa comenzi |
| Clientul apasa pe butonul de „+” a unui fel de mancare | Acel fel de mancare apare in a doua lista ce reprezinta comanda curenta |
| Clientul apasa pe butonul „Trimite” | Lista cu comanda este resetata si comanda este trimisa si apar pe ecranele ospatarilor |
| Clientul repeta ultimii doi pasi pentru a mai trimite trei comenzi | Pe ecranele ospatarilor este un total de patru comenzi |
| Ospatar 1 apasa pe butonul verde „Trimite la bucatarie” la doua comenzi | Cele doua comenzi sunt trimise la bucatar, sterse din lista ospatarilor, ospatar 1 are butonul de „Trimite la bucatarie” cu culoarea gri in timp ce ospatar 2 are cu culoarea verde |
| Bucatarul apasa la o comanda butonul verde „Comanda este gata” | Comanda apare inapoi pe ecranul ospatarilor |
| Ospatar 1 apasa pe butonul verde a comenzii venite de la bucatar cu textul „Serveste comanda” | Comanda este stearsa din lista ospatarilor si butoanele gri ale primului ospatar devin din nou verde |

In tabelul 6.4 este prezentat cazul de testare in care clientul cere nota de plata.

**Preconditii**: Clientul are deja o rezervare facuta si este pe pagina de profil, ospatarul si bucatarul lucreaza la acelasi restaurant la care s-a facut rezervarea.

Tabel 6.4 - Cazul de test a cererii notei de plata

|  |  |
| --- | --- |
| **Actiune** | **Rezultat** |
| Clientul apasa pe butonul „Confirma sosirea” | Clientului ii este prezentat ecranul din care este posibila actiunea de plasare a unei comenzi |
| Clientul apasa pe cel putin un buton „+” a unui fel de mancare | Felul de mancare apare in lista de jos a ecranului care indica cantiatea aleasa |
| Clientul apasa pe butonul de „Trimite” | Lista de jos este resetata si comanda apare pe ecranul ospatarului |
| Ospatarul apasa butonul verde care are textul „Trimite la bucatarie” | Comanda este stearsa din lista ospatarului si afisata in lista bucatarului |
| Bucatarul apasa pe butonul verde „Comanda este gata” | Comanda este stearsa din lista bucatarului si apare in lista ospatarului |
| Ospatarul apasa butonul verde „Serveste comanda” | Comanda este stearsa din lista ospatarului |
| Clientul apasa pe butonul din partea dreapta a barei de sus „Cere nota” | Clientului ii este prezentat o lista cu toate felurile de mancare pe care le-a comandat si totalul |
| Clientul apasa butonul „Achitati nota de plata” din partea stanga a barei de sus | I se prezinta un mesaj cu optiuni pentru a alege nota de plata |
| Clientul apasa pe una din urmatoarele optiuni: Cash, Cu cardul, Folosind aplicatia | Se afiseaza un mesaj de informare ca un angajat urmeaza sa vina la masa |
| Clientul apasa pe butonul de Ok la mesajului informativ | Clientul este redirectionat la pagina de profil |

# Manual de instalare si utilizare

In acest capitol se vor descrie etapele ce trebuie urmate pentru instalarea si rularea cu succes a aplicatiei QuickPlate pe un dispozitiv. Pe langa acestea, se vor prezenta resursele necesare atat hardware cat si software. Pentru a putea publica aplicatia pe AppStore, din moment ce este aplicatie iOS, este necesara detinerea unui certificat de dezvoltator, oferit de Apple, in valoare de 100 de dolari. Acestea fiind spuse, s-a preferat pastrarea aplicatiei intr-o varianta locala. Inainte de toate acestea, utilizatorul va trebui sa isi creeze un cont Apple, lucru posibil de pe site-ul lor oficial.

## Resursele necesare

### Resurse hardware

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

### Resurse software

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

## Manual de instalare

In acest subcapitol se vor prezenta pasii ce trebuie urmati in ordinea descrierii lor pentru ca utilizatorul sa reuseasca descarcarea si instalarea aplicatiei:

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

A screenshot of a computer

Description automatically generated with medium confidence

Figură 7.1 - Lista de simulatoare din Xcode

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

## Manual de utilizare

In momentul in care o instanta a aplicatiei va porni se va afisa pentru doua secunde un ecran de prezentare a aplicatiei. Dupa aceasta utilizatorul va fi redirectionat catre ecranul de autentificare. Acestea doua sunt prezentate in figura 7.2

A screenshot of a phone

Description automatically generated

Figură 7.2 - Ecran de prezentare si ecranul de autentificare a aplicatiei

In ecranul de autentificare utilizatorul isi va putea introducere credentialele pentru putea face actiunea de login. Daca nu are cont, atunci acesta va apasa cel mai de jos buton, ,,Creeaza un cont nou”, pentru a putea fi redirectionat catre ecranul de inregistrare.

Odata ajuns la acest ecran, utilizatorul este obligat sa completeze toate campurile prezente, cu exceptia celui de ,,Restaurant” care este inactiv in cazul in care tipul contului este ,,Client”. Daca rolul utilizatorului va fi ori ,,Chelner” ori ,,Bucatar” atunci campul respectiv va deveni activ si va trebui sa se selecteze numele restaurantului la care acesta lucreaza. Bineinteles, pot aparea mesaje de eroare. Pentru cazurile in care parolele nu coincid sau nu corespund formatului acceptat, se va afisa un mesaj de eroare deasupra butonului ,,Creeaza un cont nou” ca in figura 7.4, iar pentru cazul in care unul dintre campuri nu este completat sau adresa de email este deja folosita atunci vor aparea mesaje speciale peste continutul acestui ecran.

Daca toate campurile au fost completate corest si s-a respectat formatul atunci in momentul in care utilizatorul apasa pe butonul de creare a contului, i se va afisa un mesaj de succes similar cu cel din figura 7.4, poza din dreapta, care ii va sugera sa isi verifice adresa de email pentru a confirma adresa introdusa dupa care o sa fie redirectionat inapoi la ecranul de autentificare

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

Odata ce utilizatorul a reusit sa se autentifice in aplicatie in urma introducerii adresei de email si a parolei corespunzatoare contului creat, acesta va fi redirectionat catre ecranul principal. Acesta difera de la un utilizator la altul, diferenta facand-o tipul contului ales in ecranul de inregistrare. Daca acesta a ales optiunea de ,,Client” atunci el va fi redirectionat catre un ecran ce contine in partea de jos o bara cu trei sectiuni. Fiecare sectiune va afisa alt ecran in timp ce pentru celelalte doua tipuri de utilizator, se va afisa un singur ecran. In continuare se va prezenta manualul de utilizare pentru client, iar pentru ospatar si bucatar se vor explica in paralel.

### Manual de utilizare pentru client

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

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

A screenshot of a map

Description automatically generated

Figură 7.3 - Ecranul de harta

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

A screenshot of a phone

Description automatically generated with medium confidence

Figură 7.4 - Ecranul cu lista de restaurante

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

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

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

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

### Manual de utilizare pentru ospatar si bucatar

Dupa autentificare, angajatul este redirectionat direct catre pagina de profil, singurul ecran care i se afiseaza. In centru, sub sectiunea de detalii a utilizatorului, exista o lista care se populeaza cu comenzile plasate de clienti. Pentru rolul de ospatar, fiecare element are doua butoane: unul verde pentru a trimite comanda la bucatarie si care devine gri in cazul in care acesta nu mai poate accepta comenzi si unul rosu pentru a anula comanda, iar in cazul bucatarului exista doar butonul verde pentru a trimite comanda inapoi la ospatari sugerand ca s-a terminat de preparat. In figura 7.5, se pot observa patru instante ale aplicatiei ce ruleaza in acelasi timp, cea mai din stanga fiind pentru bucatar, cele doua din mijloc pentru ospatari, iar cea din dreapta pentru client.

A screenshot of a phone

Description automatically generated with low confidence

Figură 7.5 - Exemplu de rulare folosind simluatorul Xcode

# Concluzii

Proiectul urmareste sa ajute oamenii in economisirea timpului de a rezerva o masa si de a fluidiza cat mai mult interactiunea dintre client si angajatul unui restaurant. QuickPlate este o aplicatie ce ajuta la rezolvarea acestei probleme datorita functionalitatilor pe care le detine si prin care se diferentiaza comparativ cu aplicatiile concurente existente pe piata la momentul actual. In tabelul 8.1 este facuta o comparatie dintre aplicatia QuickPlate si aplicatiile existente pe piata, iar evaluarea s-a facut dupa verificarea de detinere a urmatoarelor functionalitati:

1. O harta integrata pe care utilizatorul poate sa observe restaurantele din proximitatea lui.
2. Posibilitatea de a vizualiza meniul restaurantelui si de a comanda din aplicatie.
3. Posibilitatea de a plati din aplicatie sau de a cere nota de plata sa fie adusa la masa
4. Posibilitatea de a folosi aplicatia fara a cere un abonament utilizatorilor

Tabel 8.1 - Comparatie intre QuickPlate si aplicatii existente

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

Pe langa functionalitatile de baza precum autentificare, deconectare, inregistrare si un ecran de baza, prezente in majoritatea aplicatiilor mobile, atat Android cat si iOS, aplicatia dispune si de alte caracteristici precum harta pe care se pot observa restaurantele, posibilitatea de a rezerva o masa si de a comanda direct de pe dispozitiv. Cu toate acestea, interfata utilizator este una foarte sugestiva si minimalista, pe parcursul dezvoltarii proiectului urmarindu-se pastrarea simplitatii si evitarea incarcarii ecranelor cu elemente vizuale.

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

## Contributiile propria

Proiectul urmareste eficientizarea procedeului de rezervare a unei mese si atat de primire cat si de livrare a unei comenzi la restaurant. Aplicatia ce a fost dezvoltata are ca scop indeplinirea acestor sarcini datorita simplitatii si rezultatelor obtinute in urma evoluarii acesteia.

Contributiile proprii aduse au fost stabilirea bazelor aplicatiei atat din punct de vedere a structurarii codului si componentelor cat si din cel al experientei utilizator si interfetei grafice. Desi anumite aplicatii de pe piata ofera mai multe informatii precum zilele speciale in care restaurantele nu sunt deschise, ziua de nastere a utilizatorului, istoricul cautarilor si multe altele, s-a urmarit functionarea cat mai eficienta si simpla a functionalitatilor implementate pentru a merge cat mai bine in aplicatia proprie. Cateva dintre acestea sunt:

* Integrarea API-ului de la Firebase pentru stocarea si modificarea datelor entitatilor folosite in aplicatie
* Integrarea API-ului de harta oferit de Apple pentru o navigare cat mai usoara a utilizatorului
* Oferirea posibilitatii unui utilizator de a rezerva o masa la un restaurant ales
* Posibilitatea pentru client de a alege metoda de plata
* Echilibrarea comenzilor pentru a eficientiza primirea si livrarea comenzilor

## Dezvoltari ulteriorare

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

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

# Bibliografie

|  |  |
| --- | --- |
| [1] | Apple - https://ro.wikipedia.org/wiki/Apple\_Inc. |
| [2] | John Hughes. ,,iPhone vs Android: Which is better for Web Professionals / Developers?, 1 Noiembrie, 2022. |
| [3] | Petroc Taylor. ,,Mobile operating systems' market share worldwide from 1st quarter 2009 to 4th quarter 2022", 21 Februarie, 2023. |
| [4] | HORECA - <https://ro.wikipedia.org/wiki/HoReCa>. |
| [5] | Lynn Beyrouthy. ,,Online food delivery - statistics & facts", 21 Martie, 2023. |
| [6] | UberEats - https://www.ubereats.com/ |
| [7] | Deliveroo - https://deliveroo.co.uk/ |
| [8] | Loritz Mary. ,,Execute quickly, understand local markets and adapt fast to different needs - Glovo CEO Oscar Pierre", 14 Decembrie, 2018. |
| [9] | Statistica 2023. ,,The number of downloads of leading online food delivery and takeout apps worldwide 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] | Bottorff Cassie, Haan Kathy. ,,Toast POS Review 2023: Features, Pricing & More”, 5 Mai, 2023. |
| [13] | Fabregas Krista, Main Kelly. ,,Best Restaurant Inventory Management Software 2023”, 5 Aprilir, 2023. |
| [14] | Jolaoso Christiana, Main Kelly, Watts Rob. „Toast Vs. Square (2023 Comparison), 14 Mai, 2023. |

# Anexa 1. Glosar de termeni

|  |  |  |
| --- | --- | --- |
| Abreviere | Denumire | Explicatie |
| API | Application Programming Interface | Orice program software ce are un rol distinct |
| BaaS | Backend-as-a-Service | Un model ce pune la dispozitie dezvoltatorilor de 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 mentru 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 | Interfata prin care utilizatorul interactioneaza cu un sistem |
| IDE | Integrated Development Environment | Aplicatie software ce ofera o multitudine de facilitati pentru dezvoltarea software |
| iOS | iPhone Operating System | Sistemul de operare a tuturor telefoanelor iPhone |
| iPadOS | iPad Operating System | Sistemul de operare a tabletelor iPad |
| UID | User Identifier | Un identificator unic asignat utilizatorilor unei aplicatii |

# Anexa 2. Lista figurilor si tabelelor

[Figură 4.1 - Design prototipizat in Figma 10](#_Toc137053064)

[Figură 4.2 - Diagrama de stari a unei comenzi 13](#_Toc137053065)

[Figură 4.3 - Diagrama de flow pentru inregistrare 14](#_Toc137053066)

[Figură 4.4 - Diagrama de flow pentru rezervarea unei mese 15](#_Toc137053067)

[Figură 4.5 - Diagrama de flow pentru comanda si plata consumatiei 17](#_Toc137053068)

[Figură 4.6 - Diagrama de flow pentru pentru comenzilor 18](#_Toc137053069)

[Figură 4.7 - Diagrama de flow pentru gestionarea comenzilor per chelner 19](#_Toc137053070)

[Figură 5.1 - Diagrama aplicatiei QuickPlate 21](#_Toc137053071)

[Figură 5.2 - Relatiile entitatilor bazei de date 22](#_Toc137053072)

[Figură 5.3 - Diagrama claselor principale 25](#_Toc137053073)

[Figură 5.4 - Fisierul Lozalizable.strings 26](#_Toc137053074)

[Figură 5.5 - Pachetul Assets.xcassets 27](#_Toc137053075)

[Figură 5.6 - Functia de login a clasei FirebaseEmailAuth 29](#_Toc137053076)

[Figură 5.7 - Structura modulului Extras 30](#_Toc137053077)

[Figură 5.8 - Utilizarea clasei AuthManager 32](#_Toc137053078)

[Figură 5.9 - Diagrama de clase a modulului SignIn 33](#_Toc137053079)

[Figură 5.10 - Diagrama de clase a modulului SignUp 34](#_Toc137053080)

[Figură 5.11 - Diagrama de clase a modulului Map 35](#_Toc137053081)

[Figură 5.12 - Diagrama de clase a modulului Restaurants 36](#_Toc137053082)

[Figură 5.13 - Diagrama de clase a modulului RestaurantDetails 37](#_Toc137053083)

[Figură 5.14 - Diagrama de clase a modulului TablesList 38](#_Toc137053084)

[Figură 5.15 - Diagrama de clase a modulului UserProfile 39](#_Toc137053085)

[Figură 5.16 - Diagrama de clase a modulului OrderView 40](#_Toc137053086)

[Figură 5.17 - Diagrama de clase a modulului ClientOrder 41](#_Toc137053087)

[Figură 5.18 - Starile unei comenzi 42](#_Toc137053088)

[Figură 7.1 - Lista de simulatoare din Xcode 47](#_Toc137053089)

[Figură 7.2 - Ecran de prezentare si ecranul de autentificare a aplicatiei 48](#_Toc137053090)

[Figură 7.3 - Ecranul de harta 49](#_Toc137053091)

[Figură 7.4 - Ecranul cu lista de restaurante 50](#_Toc137053092)

[Figură 7.5 - Exemplu de rulare folosind simluatorul Xcode 51](#_Toc137053093)

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

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

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

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

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

[Tabel 8.1 - Comparatie intre QuickPlate si aplicatii existente 52](#_Toc137052781)

# Anexa 3. Exemple 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.uid))

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

}

}

**self**.calculateTotalCost()

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

resName: name,

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

foodIds: ids,

foodQuantity: quan,

totalCost: **self**.totalCost,

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

tableId: **self**.tableId,

orderState: .pending)

FSOrdersColl.shared.saveOrder(order)

**self**.resetOrder()

}

}

**Functionalitatea de acceptare a unei comenzi de catre un ospatar:**

**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 not convert qdSnap to order")

**return**

}

**switch** order.orderState {

**case** .pending:

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

**case** .preparing:

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

**case** .ready:

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

**case** .sent:

**break**

}

}

}

**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])

}

}

}