**QUICKPLATE – APLICATIE MOBILA PENTRU REZERVAREA UNEI MESE LA UN 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: **Prenumele NUMELE**

**TITLUL LUCRĂRII DE LICENŢĂ**

1. **Enunțul temei:** *Scurtă descriere a temei lucrării de licență și datele inițiale*
2. **Conținutul lucrării:** *(enumerarea pârților componente) Exemplu: Pagina de prezentare, aprecierile coordonatorului de lucrare, titlul capitolului 1, titlul capitolului 2,… titlul capitolului n, bibliografie, anexe.*
3. **Locul documentării**: *Exemplu*: 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](#_Toc135678904)

[Capitolul 2. Obiectivele proiectului 2](#_Toc135678905)

[2.1. Obiectivul principal 2](#_Toc135678906)

[2.2. Obiective specific 2](#_Toc135678907)

[Capitolul 3. Studiu bibliografic 4](#_Toc135678908)

[3.1. Studiul utilizarii sistemelor de operare mobile 4](#_Toc135678909)

[3.2. Aplicatii mobile de tipul food-delivery 5](#_Toc135678910)

[3.3. Aplicatii mobile food-delivery din lume 5](#_Toc135678911)

[3.3.1. Aplicatia Zomato 5](#_Toc135678912)

[3.3.2. Aplicatia UberEats 6](#_Toc135678913)

[3.3.3. Aplicatia Foodpanda 6](#_Toc135678914)

[3.3.4. Asemanari si diferente intre Zomato, UberEats si Foodpanda 6](#_Toc135678915)

[3.4. Software de administrare al restaurantelor 7](#_Toc135678916)

[3.4.1. Toast POS 7](#_Toc135678917)

[3.4.2. Square POS 7](#_Toc135678918)

[3.4.3. Toast sau Square? 8](#_Toc135678919)

[Capitolul 4. Analiză și fundamentare Teoretică 9](#_Toc135678920)

[4.1. Cerinte functionale 9](#_Toc135678921)

[4.2. Cerinte non-functionale 9](#_Toc135678922)

[4.3. Cazuri de utilizare 10](#_Toc135678923)

[4.3.1. Inregistrare 10](#_Toc135678924)

[4.3.2. Rezervarea unei mese 11](#_Toc135678925)

[4.3.3. Comanda si plata consumatiei 12](#_Toc135678926)

[4.3.4. Procesarea comenzilor 14](#_Toc135678927)

[4.4. Prezentare la nivel inalt al aplicatiei 16](#_Toc135678928)

[4.4.1. Modelarea problemei 16](#_Toc135678929)

[4.4.2. Modelul de date 18](#_Toc135678930)

[4.4.3. Starile unei comenzi 18](#_Toc135678931)

[4.4.4. Algoritmul de load-balancing pentru chelneri 19](#_Toc135678932)

[Capitolul 5. Proiectare de detaliu și implementare 20](#_Toc135678933)

[5.1. Proiectarea si implementarea arhitecturii 20](#_Toc135678934)

[5.2. Proiectarea si implementarea bazei de date 22](#_Toc135678935)

[5.2.1. Entitatea Food 23](#_Toc135678936)

[5.2.2. Entitatea MyUser 23](#_Toc135678937)

[5.2.3. Entitatea Order 24](#_Toc135678938)

[5.2.4. Entitatea Restaurant 25](#_Toc135678939)

[5.2.5. Entitatea Table 26](#_Toc135678940)

[5.3. Proiectarea si implementarea modulelor 27](#_Toc135678941)

[5.3.1. Structura fisierelor 27](#_Toc135678942)

[5.3.2. Directorul Extras 29](#_Toc135678943)

[5.3.3. Directorul LoginStates 29](#_Toc135678944)

[5.3.4. Directorul Features 29](#_Toc135678945)

[Capitolul 6. Testare și validare 30](#_Toc135678946)

[Capitolul 7. Manual de instalare si utilizare 31](#_Toc135678947)

[Capitolul 8. Concluzii 32](#_Toc135678948)

[Bibliografie 33](#_Toc135678949)

[Anexa 1. Glosar de termeni 34](#_Toc135678950)

[Anexa 2. Lista figurilor si tabelelor din document 34](#_Toc135678951)

[Anexa 3 35](#_Toc135678952)

# Introducere

Ce se scrie aici:

* Contextul proiectului
* Conturarea domeniului exact al temei

Între 2 și 3 pagini. Nu puneți secțiuni

# 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 specific

**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

## Studiul utilizarii sistemelor de operare mobile

In 2007. Apple a lansat primul iPhone, revolutionand modul in care oamenii isi foloses telefoanele. iPhone-ul rula iOS, propriul sistem de operare dezvoltat de Apple. In timp ce iPhone-ul devenea rapid un succes, dezvoltatorii de la Google lucrau la propriul sistem de operare mobil numit Android.

Lansat in 2008, Android a fost conceput ca un siste, de operare open-source care poate fi folosit pe multe dispozitive mobile, nu doar pe smartphone-urile Google. De cand Android a fost dezvoltat ca sistem open-source, a devenit popular in randul producatorilor de telefoane mobile, deoarece dezvoltatorii isi pot adauga propriile personalizari si aplicatii la sistemul de operare.

In anii ce au urmat, Apple si Google au continuat sa isi imbunatateasca sistemele de operare, iar iOS si Android au continuat sa evolueze. In timp ce iOS a fost intotdeauna un sistem de operare inchis, cu control strict asupra hardware-ului si software-ului, Android ramane un sistem de operare open-source si a fost folosit de o gama larga de dispozitive.

Android si iOS sunt cele doua sisteme de operare mobile cele mai utiliate in lume. Prin urmare, conform unui articol [1] de pe site-ul gitnux, sistemul de operare Android detine 87% din piata mondiala a smartphone-urilor pe cand iOS doar 12.4%, acest fapt datorandu-se varietatii telefoanelor pe care Android poate functiona. In Figura 3.1 se poate observa un chart cu aceste procente. De asemenea, pentru a descarca aplicatii pe Android trebuie folosit Google Play Store care detine aproximativ 116 miliarde de aplicatii pe cand pentru iOS utilizatorii folosesc Apple App Store care detine aproximativ 34.5 miliarde de aplicatii.

Chart, pie chart

Description automatically generated

Figura 3.1 Cele mai folosite sisteme de operare pe telefoanele mobile

## Aplicatii mobile de tipul food-delivery

HORECA [2] 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[3], 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 [4], Deliveroo [5] sau Glovo [6] . 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.

## Aplicatii mobile food-delivery din lume

### Aplicatia Zomato

Potrivit unei statistici[7] 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 [8]. 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 [7], 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. [9]

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 [10] 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.

### Toast sau Square?

Conform unui blog [12] 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

## Cerinte functionale

Cerintele functionale ale unui sistem reprezinta specificatiile care defines modul in care sistemul trebuie sa functioneze si sa indeplineasca obiectivele pentru care a fost creat. Aceste cerinte sunt exprimate in termini de functionalitati care trebuie sa fie implementate pentru a asigura un nivel adecvat de calitate si eficienta in utilizarea sistemului. Cerintele functionale pot fi de asemenea utilizate pentru a verifica si valida performantele sistemului si pentru a identifica eventualele probleme sau erori care apar in timpul utilizarii.

Urmatoarele sunt cerintele functionale reprezinte 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 folosita mai tarziu 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.
* **Load balancing pentru chelneri.** 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.
* **Scalabilitatea**. Reprezinta capacitatea de a gestiona cresterea volumului de date sau a numarului de utilizatori fara a afecta performanta. Datorita Firebase, acest lucru nu reprezinta o problema foarte mare pentru dezvoltatorul aplicatiei intrucat acest BaaS se ocupa de majoritatea problemelor ce apar deodata cu scalarea aplicatiei.
* **Performanta.** Performanta aplicatiei consta intr-un timp de raspuns cat mai mic pentru orice actiune desfasurata de utilizator. QuickPlate este destul de performanta intrucat s-a tinut cont de aceasta cerinta pe parcursul dezvoltarii ei, iar daca pe viitor se doreste adaugarea unor functionalitati noi sau numarul de utilizatori v-a creste, diferenta de performanta ar trebui sa fie una foarte mica, aproape nesemnificativa.
* **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. Din fericire, QuickPlate a fost dezvoltata tinand cont de o posibila extensie si mai multe, unele componente au fost deja reutilizate in anumite locuri. Acest lucru este foarte bun pentru ca ajuta la dezvoltarea mai departe a acesteia cu un efort cat mai mic din partea dezvoltatorului.

## Cazuri de utilizare

### Inregistrare

**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 4.1 - Diagrama de flow pentru inregistrare

### Rezervarea unei mese

**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 4.2 - Diagrama de flow pentru rezervarea unei mese

### Comanda si plata consumatiei

**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 4.3 - Diagrama de flow pentru comanda si plata consumatiei

### Procesarea comenzilor

**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 4.4 - Diagrama de flow pentru pentru comenzilor

## Prezentare la nivel inalt al aplicatiei

### Modelarea 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 4.5 - Design prototipizat in Figma

### Modelul de date

Baza de date folosita este Firestore care este flexibila si scalabila datorita faptului ca este NoSQL datorita faptului ca datele sunt retinute sub forma de documente si este furnizata de GCP. Este conceputa pentru a stoca si sincroniza date pentru aplicatii web, mobile si server in timp real.

Principalele modele sunt utilizatorii, restaurantele, comenzile, mancarea si mesele. Utilizatorul are referinte la restaurantele favorite si la mesele rezervate. Acestea doua sunt esentiale pentru pagina de profil din contul unui client.

Restaurantul, pe langa atributele de baza, acesta are o referinta la mai multe mese care ii aparting si la un meniu, acesta din urma fiind compus din mai multe entitati de mancare.

Mancarea si masa sunt de sine statatoare, cu exceptia celei din urma care tine o referinta la user-ul care a facut rezervarea la acea masa. Acestea doua sunt cele mai simple si ocupa cea mai putina memorie comparativ cu celelalte.

Ultimul model de date este comanda, care are referinta la mancarea comandata, la utilizatorul care a plasat comanda si la masa de la care s-a plasat.

Toate acestea au fost gandite in asa fel incat munca dezvoltatorului sa fie una cat mai simpla, fara a fi nevoit sa efectueaza prea multe operatii pe baza de date. Cu toate acestea, pe anumite ecrane se construiesc entitati mai mici pentru a reduce transferul de date intre ecranele telefonului. Un exemplu il reprezinta lista de restaurante deoarece in aceasta nu avem nevoie decat de o poza, nume, rating, adresa si intervalul orar deschis. Toate cele prezentate anterior vor fi dezvoltate in detaliu in urmatorul capitol.

### 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.

* Prima stare - pending: Aceasta este starea initiala a fiecarei comenzi plasate. O comanda se afla in aceasta stare atunci cand utilizatorul plaseaza comanda si este transmisa la angajatii restaurantului
* A doua stare - preparing: Aceasta stare apare in momentul in care chelnerul accepta comanda si este transmisa bucatarului
* A treia stare - ready: Aceasta stare apare atunci cand bucatarul confirma terminarea prepararii comenzii
* A patra stare - sent: 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 4.6 - Diagrama de stari pentru comenzi

### Algoritmul de load-balancing pentru chelneri

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.

# 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.

In Figura 5.1 se poate observa un caz de folosire a modelului MVVM pe pagina ecranul de creare cont.

A screenshot of a computer screen

Description automatically generated with medium confidence

Figură 5.1 - Caz de utilizare a modelului MVVM

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 5.2 se poate observa arhitectura aplicatiei QuickPlate.

A picture containing text, screenshot, diagram, rectangle

Description automatically generated

Figură 5.2 - Arhitectura aplicatiei QuickPlate

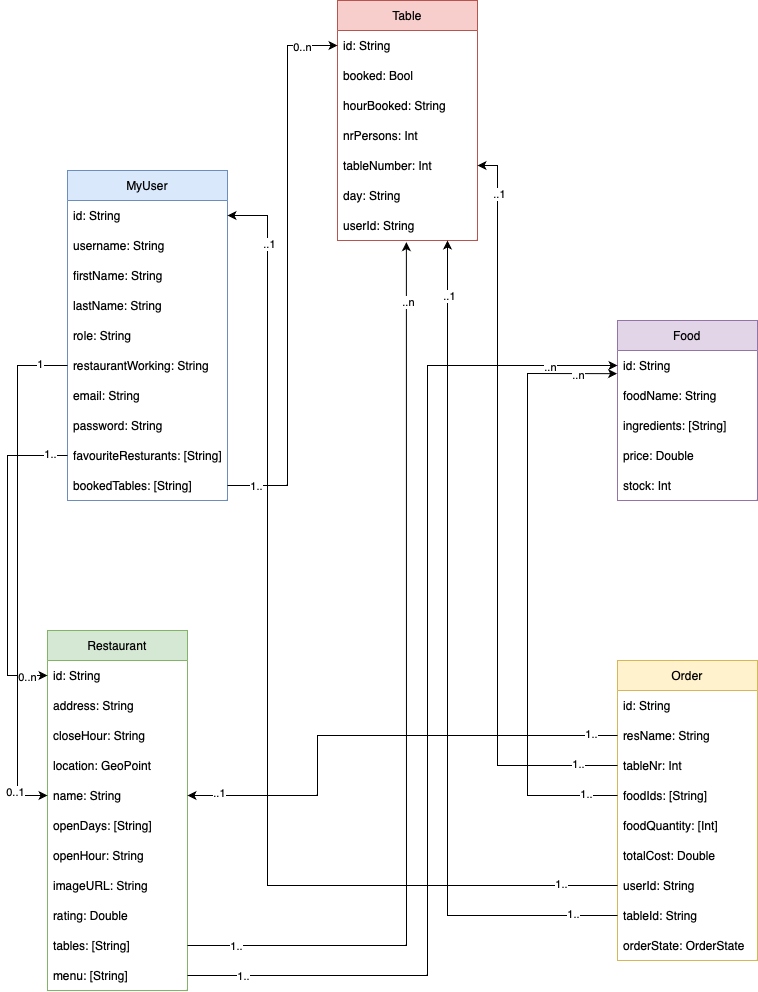
## Proiectarea si implementarea bazei de date

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.



Figură 5.3 - 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

In figura 5.4 se poate observa entitatea Food afisata in interfata utilizator

A screenshot of a phone

Description automatically generated with medium confidence

Figură 5.4 - Entitatea Food in interfata utilizator

### 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
* In figura 5.5 se poate observa utilitatea acestei entitati.

A screenshot of a phone

Description automatically generated with medium confidence

Figură 5.5 - Entitatea Order in interfata utilizator

### 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

In figura 5.6 se pot observa cate folosinte ale entitatii Restaurant

A screenshot of a phone

Description automatically generated with medium confidence

Figură 5.6 - Entitatea Restaurant in interfata utilizator

### 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.

In urmatoarea figura sunt prezentate cateva interfete utilizator in care clientul intalneste folosirea entitatii Table.

Screens screenshot of a phone

Description automatically generated with low confidence

Figură 5.7 - Entitatea Table in interfata utilizator

## Proiectarea si implementarea modulelor

### 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 5.8 se poate observa o parte din acest fisier.

A screenshot of a computer

Description automatically generated

Figură 5.8 - 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 5.9 se poate observa cum arata acest pachet.

A screenshot of a computer

Description automatically generated

Figură 5.9 - 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.

### Directorul Extras

In directorul 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 des.

Singurul fisier care nu este cuprins intr-un director separat este FSCollNames.swift care contine un simplu Enum cu numele colectiilor din baza de date destinate modelelor 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.

Subdirectorul 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 cloud 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 vine de la collection.

Cu toate acestea, clasa din acelasi director cu aceste cinci clase, FirebaseEmailAuth, este si ea de tipul singleton, dar are alt scop si anume de a se ocupa de partea de autentificare, respectiv autorizare 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 5.10 se poate observa un enum denumit StartupError de tipul Error ce are cazuri pentru diferite tipuri de erori si metoda de Sign In denumita doLogin pentru autentificarea utilizatorului.

A screen shot of a computer program

Description automatically generated with low confidence

Figură 5.10 - Functia de login a clasei FirebaseEmailAuth

In subdirectorul 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 subdirector, 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.
* 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
* 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 5.11 se poate observa structura directorului prezentat in acest subcapitol.

A screenshot of a computer

Description automatically generated with medium confidence

Figură 5.11 - Structura directorului Extras

### Directorul LoginStates

In acest director 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 5.12, 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 5.12 - Utilizarea clasei AuthManager

### Directorul Features

Directorul 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 subdirector 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. Acest View este folosit mereu la initializarea clasei RootView indiferent de starea autentificarii utilizatorului.

Al doilea subdirector intitulat SignIn contine logica de autentificare si contine doua fisiere SignInView si SignInViewModel, iar pe langa acestea un alt subdirector „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 modifier ce se poate aplica asupra oricarui View pentru a-l stiliza. Un modifier 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 Figura 5.13 se observa aspectul View-ul SignInView.

A screenshot of a phone

Description automatically generated with medium confidence

Figură 5.13 - Pagina Sign In

Cel de al treila subdirector 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. In Figura 5.14 se vede aspectul acestui ecran.

A screenshot of a phone

Description automatically generated with medium confidence

Figură 5.14 - Pagina de Sign Up

Urmatoarele trei subdirectoare, 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 subdirectoarele din directorul Client dupa care directoarele Waiter si Cook vor fi detaliate in acelasi timp evidentiand micile diferente intre ele.

* 1. Subdirectorul 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, vizibil in figura 5.14.

A close-up of a restaurant

Description automatically generated with medium confidence

Figură 5.15 - View-ul QPTabView

* 1. Subdirectorul 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 usoara intrucat. 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.

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 local. In figura 5.16 se pot observa restaurantele pe harta, iar in urma apasarii pin-ului cu recenzia 4.3 apare View-ul de jos. Pe langa asta, in dreapta se poate vedea efectul apasarii butonului de locatie din partea de sus avand ca efect centrarea hartii la locatia utilizatorului.

A screenshot of a map

Description automatically generated

Figură 5.16 - Ecranul cu continutul hartii

* 1. Subdirectorul Restaurants

Acest subdirector 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. 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 5.17 se poate observa lista restaurantelor.

A screenshot of a phone

Description automatically generated with medium confidence

Figură 5.17 - Ecranul cu lista restaurantelor

* 1. Subdirectorul RestaurantDetails

Pe langa lucrurile descrise anterior, lista restaurantelor este interactiva deoarece facand click pe o intrare din lista, se va ajunge la o pagina cu detalii pentru acel restaurant. Aceasta navigare este posibila datorita structurii NavigationView in care sunt aranjate restul elementelor de interfata utilizator. Fisierul RestaurantDetailsView se ocupa cu prezentarea interfetei utilizator fiind un View si are ca ViewModel fisierul RestaurantDetailsViewModel care are in plus definit View-ul pentru a afisa o entitate de tipul Food. In Figura 5.18 se poate vedea pagina de detalii a restaurantului, iar sub sectiunea „Meniu” este o lista cu felurile de mancare ce tin de meniul restaurantului.

A screenshot of a phone

Description automatically generated with medium confidence

Figură 5.18 - Ecranul de detalii a restaurantului

* 1. Subdirectorul 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.

Logica functionalitatii este una foarte simpla intrucat View-Model-ul are doar trei metode: de a aduce datele din baza de date (la initializarea View-ului), de a salva o masa rezervata (cand utilizatorul apasa pe butonul „Rezerva” din dreptul mesei) si de a actualiza lista lasand afisate doar mesele care nu sunt rezervate (dupa ce utilizatorul rezerva o masa). In figura 5.19 in partea dreapta se observa cazul in care nicio masa nu este disponibila, iar in stanga lista meselor disponibile pentru rezervare.

Screens screenshot of a phone

Description automatically generated with medium confidence

Figură 5.19 - Ecranul de rezervare a unei mese

* 1. Subdirectorul UserProfile

Profilul clientului este creat in acest subdirector 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 buton de deconectare, si sectiunea de dedesubtul acesteia 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. In urmatoarea figura, in ambele imagini, se poate observa setiunea cu informatiile clientului, in timp ce imaginea din stanga arata lista rezervarilor, iar cea din dreapta lista restaurantelor favorite.

Screens screenshot of a phone

Description automatically generated with medium confidence

Figură 5.20 - Ecranul cu profilul clientului

* 1. Subdirectorul OrderView

In acest director, 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 si anume prima este folosita la a lua din baza de date informatii despre felurile de mancare din acea comanda si a doua pentru a le sorta dupa id-uri. Aceasta ultima functie a fost necesara deoarece cand felurile de mancare sunt luate din baza de date sunt ordonate automat dupa UUID-ul asignat de Firestore, iar aceasta ordine poate sa difere de cea din comanda.

* 1. Subdirectorul ClientOrder

Ultimul subdirector 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. De asemnea, isi poate alege metoda prin care doreste sa achite nota de plata. In figura 5.21, in partea stanga se observa ecranul de plasare a unei comenzi, in mijloc ecranul de plata, iar in dreapta metodele prin care clientul poate achita totalul.

Screens screenshot of a phone

Description automatically generated with low confidence

Figură 5.21 – Ecranul de plata a comenzii si ecranul de achitare

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 load-balancing pentru comenzi 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 5.22 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 5.22 - Starile unei comenzi

# Testare și validare

# Manual de instalare si utilizare

# Concluzii

Bibliografie

|  |  |
| --- | --- |
| [1] | Gitnux. ,,The Most Surprising Android vs Iphone Statistics And Trends in 2023, 11 Aprilie 2023 - https://blog.gitnux.com/android-vs-iphone-statistics/ |
| [2] | HORECA - <https://ro.wikipedia.org/wiki/HoReCa> |
| [3] | Lynn Beyrouthy. ,,Online food delivery - statistics & facts", 21 Martie, 2023. |
| [4] | UberEats - https://www.ubereats.com/ |
| [5] | Deliveroo - https://deliveroo.co.uk/ |
| [6] | Loritz Mary. ,,Execute quickly, understand local markets and adapt fast to different needs - Glovo CEO Oscar Pierre", 14 Decembrie, 2018. |
| [7] | 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/ |
| [8] | Khosla Varuni, Srinivasan Supraja. ,,Zomato co-founder Pankaj Chaddah quits as it shuffles top management", Economic Times, 9 Aprilie 2022. |
| [9] | Yoolim Lee. ,,Delivery Hero's Foodpanda Claims Asia Top Spot, Sees More Growth", 5 Iulie, 2021. |
| [10] | Bottorff Cassie, Haan Kathy. ,,Toast POS Review 2023: Features, Pricing & More”, 5 Mai, 2023. |
| [11] | Fabregas Krista, Main Kelly. ,,Best Restaurant Inventory Management Software 2023”, 5 Aprilir, 2023. |
| [12] | Jolaoso Christiana, Main Kelly, Watts Rob. „Toast Vs. Square (2023 Comparison), 14 Mai, 2023. |

# Anexa 1. Glosar de termeni

|  |  |  |
| --- | --- | --- |
| Abreviere | Denumire | Explicatie |
| 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 |
| GCP | Google Cloud Platform | Suita de servicii de cloud computing care ruleaza pe aceeasi infrastructura pe care Google o foloseste intern pentru produsele sale |
| 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 |

# Anexa 2. Lista figurilor si tabelelor din document

[Figură 4.1 - Diagrama de flow pentru inregistrare 11](#_Toc135946599)

[Figură 4.2 - Diagrama de flow pentru rezervarea unei mese 12](#_Toc135946600)

[Figură 4.3 - Diagrama de flow pentru comanda si plata consumatiei 14](#_Toc135946601)

[Figură 4.4 - Diagrama de flow pentru pentru comenzilor 15](#_Toc135946602)

[Figură 4.5 - Design prototipizat in Figma 17](#_Toc135946603)

[Figură 4.6 - Diagrama de stari pentru comenzi 18](#_Toc135946604)

[Figură 5.1 - Caz de utilizare a modelului MVVM 21](#_Toc135946605)

[Figură 5.2 - Arhitectura aplicatiei QuickPlate 21](#_Toc135946606)

[Figură 5.3 - Relatiile entitatilor bazei de date 22](#_Toc135946607)

[Figură 5.4 - Entitatea Food in interfata utilizator 23](#_Toc135946608)

[Figură 5.5 - Entitatea Order in interfata utilizator 25](#_Toc135946609)

[Figură 5.6 - Entitatea Restaurant in interfata utilizator 26](#_Toc135946610)

[Figură 5.7 - Entitatea Table in interfata utilizator 27](#_Toc135946611)

[Figură 5.8 - Fisierul Lozalizable.strings 28](#_Toc135946612)

[Figură 5.9 - Pachetul Assets.xcassets 28](#_Toc135946613)

[Figură 5.10 - Functia de login a clasei FirebaseEmailAuth 30](#_Toc135946614)

[Figură 5.11 - Structura directorului Extras 31](#_Toc135946615)

[Figură 5.12 - Utilizarea clasei AuthManager 32](#_Toc135946616)

[Figură 5.13 - Pagina Sign In 33](#_Toc135946617)

[Figură 5.14 - Pagina de Sign Up 34](#_Toc135946618)

[Figură 5.15 - View-ul QPTabView 34](#_Toc135946619)

[Figură 5.16 - Ecranul cu continutul hartii 35](#_Toc135946620)

[Figură 5.17 - Ecranul cu lista restaurantelor 36](#_Toc135946621)

[Figură 5.18 - Ecranul de detalii a restaurantului 37](#_Toc135946622)

[Figură 5.19 - Ecranul de rezervare a unei mese 38](#_Toc135946623)

[Figură 5.20 - Ecranul cu profilul clientului 39](#_Toc135946624)

[Figură 5.21 – Ecranul de plata a comenzii si ecranul de achitare 40](#_Toc135946625)

[Figură 5.22 - Starile unei comenzi 41](#_Toc135946626)

[Tabel 3.1 - Diferente intre Toast si Square 8](#_Toc135678879)