

**UNIVERSITATEA ROMÂNO – AMERICANĂ**  
**FACULTATEA DE INFORMATICĂ MANAGERIALĂ**



**LUCRARE DE LICENȚĂ**

Coordonator științific:

Conf.univ.dr Crișan Daniela Alexandra

Absolvent:

Mușeteanu T Corneliu

București 2024

**UNIVERSITATEA ROMÂNO – AMERICANĂ**  
**FACULTATEA DE INFORMATICĂ MANAGERIALĂ**



**REALIZAREA UNEI APLICAȚII INFORMATICE PENTRU  
GESTIUNEA ACTIVITĂȚII UNEI CAFENELE**

Coordonator științific:

Conf.univ.dr Crișan Daniela Alexandra

Absolvent:

Mușeteanu T Corneliu

București 2024

## Cuprins

Introducere.....	4
Oportunitatea și importanța temei propuse.....	4
Cap. 1. Studiul și analiza sistemului existent .....	5
1.1. Prezentarea succinta a unității economico-sociale .....	5
1.2. Principalele activități desfășurate în unitatea economică .....	6
1.3. Studiul sistemului de conducere .....	7
1.4. Studiul sistemului condus .....	9
1.5. Studiul sistemului informațional.....	10
1.5.1. Schema fluxului informațional .....	10
1.5.2. Descrierea circuitului informațional .....	11
1.5.3. Descrierea documentelor utilizate .....	12
1.5.4. Modelul conceptual al prelucrărilor.....	19
1.5.5. Analiza critică a sistemului actual și identificarea neajunsurilor existente în funcționarea sistemului existent .....	21
1.5.6. Direcții de perfecționare a sistemului actual .....	22
Cap. 2. Proiectarea de detaliu a aplicației informative .....	23
2.1. Definirea obiectivelor aplicației informative.....	23
2.2. Proiectarea logică și fizică a ieșirilor.....	24
2.2.1 Lista cu toate documentele de ieșire din aplicație .....	24
2.2.2 Machetele tuturor documentelor de ieșire .....	25
2.3. Proiectarea logică și fizica a intrărilor .....	32
2.3.1 Lista cu toate intrările din aplicație .....	32
2.3.2 Machetele pentru toate video formatele din aplicația informatica .....	34
2.4. Proiectarea sistemului de codificare a datelor .....	41
2.5. Proiectarea bazei de date .....	43
2.5.1 Schema relațională a bazei de date .....	49
2.6. Schema de sistem a aplicației .....	50
2.7. Proiectarea interfeței aplicației .....	51
2.8 Alegerea tehnologiei de prelucrare .....	69
2.9 Estimarea necesarului de resurse și a calendarului de realizare .....	70

2.9.1 Activitățile și necesarul de resurse .....	70
2.9.2. Diagrama Gantt.....	72
Cap. 3. Prezentarea produsului software .....	73
3.1 Cerințele platformei hardware și software ale produsului.....	73
3.2 Descrierea funcțiunilor aplicației.....	74
Cap. 4. Eficiența și utilitatea aplicației informaticе.....	78
4.1 Condiții privind implementarea aplicației .....	78
4.2 Exploatarea curenta a aplicației .....	78
4.3 Considerații privind eficiența aplicației informatice .....	78
5.Bibliografie.....	80
6. Anexe .....	81
6.1. Anexele cu codul din API.....	81
6.2. Anexele cu codul din frontend.....	111

## Introducere

### Oportunitatea și importanța temei propuse

Tehnologia se dezvoltă foarte rapid în ultimii ani, asta a condus la faptul că majoritatea procedurilor fizice din procesele de muncă au fost înlocuite cu procese automatizate prin intermediul unor sisteme informatiche care a permis în primul rând că etapele pe care le-a înlocuit să devină mult mai rapide și să apară mult mai puține erori care puteau fi cauzate în trecut de factorul uman.

Această lucrare are că obiectiv crearea și implementarea unui sistem informatic în activitatea unei cafenele ,SD BAR S.R.L' cu scopul de a automatiza anumite procese pentru a eficientiza procesul de muncă din această societate cu răspundere limitată.

Scopul acestei lucrări este de a crea un sistem informatic cu o bază de date în spate, aplicația informatică va reprezenta un site prin care se vor accesa toate funcționalitățile acestuia. Clienții se vor putea înregistra în aplicația informatică. Motivul pentru care clienții trebuie să se autentifice este pentru a avea acces la istoricul comenzilor, a putea efectua și vizualiza rezervările și pe viitor ca o etapă de perfecționare a sistemului informatic de a beneficia de reduceri în funcție de fidelitatea acestora.

Importanța creării aplicației informaticice este pentru a ușura procesele de muncă din cadrul localului și de a permite gestionarea mai eficientă al acestuia.

---

## Cap. 1. Studiul și analiza sistemului existent

### 1.1. Prezentarea succinta a unității economico-sociale

“SD BAR SRL” este o societate comercială cu răspundere limitată înființată în anul 2007 de către o singură persoană, firma face parte din industria ospitalității având codul CAEN 5630 - „Activitatea în domeniul barurilor și servirii de băuturi”. Firma dată reprezintă un bar care furnizează servicii de vânzarea de băuturi și produse alimentare de complexitate mică, adică în mare parte gustări pentru băuturile oferite.

Obiectivul acestei firme este de a oferi clienților să-i un mediu cât mai relaxant unde ar putea să-și petreacă timpul cu cei apropiati pentru a se bucură de diversele băuturi alcoolice și non-alcoolice cât și de meniul diversificat.

Părțile implicate:

- Personalul de conducere – persoanele care gestionează diferite aspecte legate de funcționarea corectă și maximizarea profitului firmei.
- Angajații – persoanele care lucrează la aceasta societate și care au ca rol principal servirea clienților.
- Furnizorii – întreprinderile de la care se face aprovizionarea necesară activității.
- Clienții – persoanele fizice care beneficiază de serviciile oferite.

## 1.2. Principalele activități desfășurate în unitatea economică

În societatea „SD BAR S.R.L” sunt desfășurate următoarele activități:

1. Preluarea comenziilor de către chelneri – această activitate se efectuează în pași următori. Clientul vine la localul dat, dacă dorește să se rețină pe o perioadă mai îndelungată se poate așeza la o masă, dacă nu, poate face comanda direct la casă. Comanda este preluată de către un chelner care pregătește comanda și o transmite clientului, clientul cere nota de plată sau plătește direct la casa de marcat și primește bonul fiscal.
2. Efectuarea rezervărilor – clienții pot efectua și rezervări la local pentru a se asigura că vor avea o masă liberă pentru ei la ora rezervată. Rezervarea se face telefonic.
3. Aprovizionarea de mărfuri – de partea de aprovizionare se ocupă biroul de aprovizionare, acesta verifică zilnic numărul produselor din gestiune și dacă se observă că un produs din gestiune are stocul foarte mic se efectuează o comanda la furnizor care livrează comanda și biroul de aprovizionare o preia de la acesta.

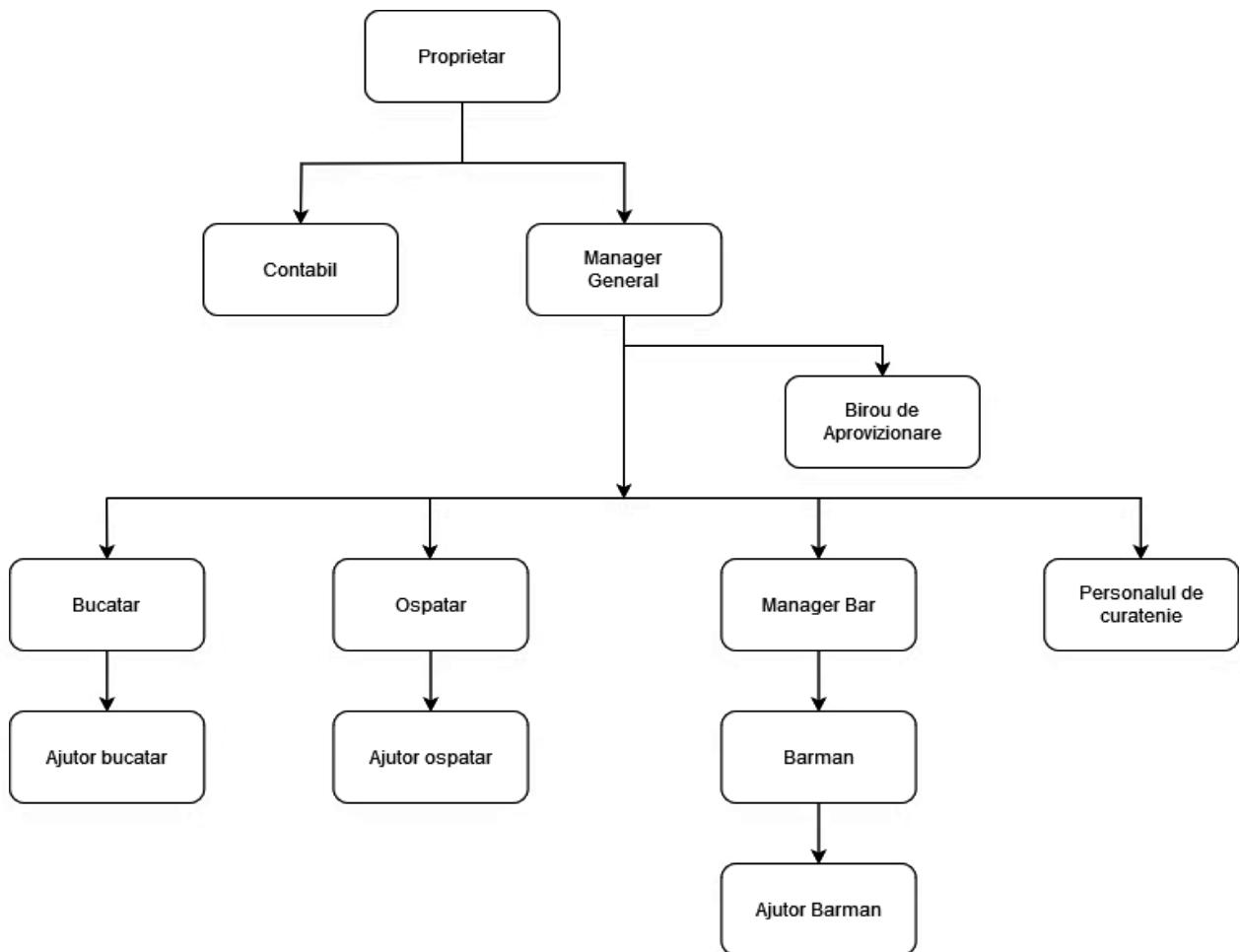
*Figura 1, Indicatorii economici*

AN	CIFRA AFACERI	PROFIT NET	DATORII	ACTIVE IMOBILIZATE	ACTIVE CIRCULANTE	CAPITALURI PROPIII	ANGAJATI (NR. MEDIU)
2022	1,070,059.00	205,917.00	241,497.00	2,545.00	493,580.00	1,200.00	6
2021	685,724.00	77,576.00	187,445.00	3,720.00	232,524.00	1,200.00	3
2020	267,428.00	9,687.00	250,714.00	0.00	221,937.00	1,200.00	5

### 1.3. Studiul sistemului de conducere

Organograma firmei:

*Figura 2, Organograma firmei*



### Studiul sistemului de conducere

#### Proprietar

➤ Responsabilități:

- Detinătorul firmei
- Responsabil pentru obținerea licențelor necesare pentru operarea localului
- Asigurarea securității în local
- Conducerea localului

## **Manager General**

➤ Responsabilități:

- Recrutarea și gestionarea personalului
- Formarea graficului de muncă a personalului
- Implementarea standardelor de serviciu
- Gestionarea relațiilor cu clienții (obținerea de feedback asigurându-se că cererile și plângerile sunt abordate în mod corespunzător)
- Raportarea către proprietar

## **Manager Bar**

➤ Responsabilități:

- Gestionarea operațiunilor zilnice legate de tot ce ține de partea de bar (adică verificarea stocului pentru bar)
- Supravegherea subordonăților
- Menținerea unei calități înalte a tuturor comenziilor trimise către clienți

## 1.4. Studiul sistemului condus

### **Biroul de aprovizionare**

Este reprezentat de o singură persoană care are ca responsabilitate verificarea stocurilor zilnice și aprovizionarea acestora în cazul în care sunt sub numărul minim necesar stabilit, acesta face comenzi către furnizori și se asigură la primirea mărfuii că totul să fie în ordine.

### **Contabil**

Persoana care se ocupă de gestionarea contabilității generale, emiterea salariilor către angajați și asigurarea conformității legale ,asigurarea că toate activitățile financiare ale barului respectă legile și reglementările fiscale locale și naționale.

### **Bucătar**

Este responsabil de gestiunea bucătăriei, prepararea gustărilor mici în avans pentru restul zilei și a prânzurilor în perioada stabilită în care acestea pot fi comandate și asigurarea că toate comenziile trimise în sală sunt la o calitate înaltă.

### **Ajutor bucătar**

Ajutorul bucătarului, se ocupă de pregătirea ingredientelor necesare pentru prepararea bucatelor.

### **Ospătar**

Preia comenziile de la clienți din sală, le duce la bar și duce comenziile către mese. Se asigură că clienții de la mesele pe care le servesc să nu plece fără a achita.

### **Ajutor ospătar (picol)**

Se ocupă de curățarea meselor după plecarea clienților și are grijă că pe mese să fie toate tacâmurile necesare pentru următorii clienți.

### **Barman**

Preia comenziile de la ospătari și prepară băuturile alcoolice și non-alcoolice din comenzi.

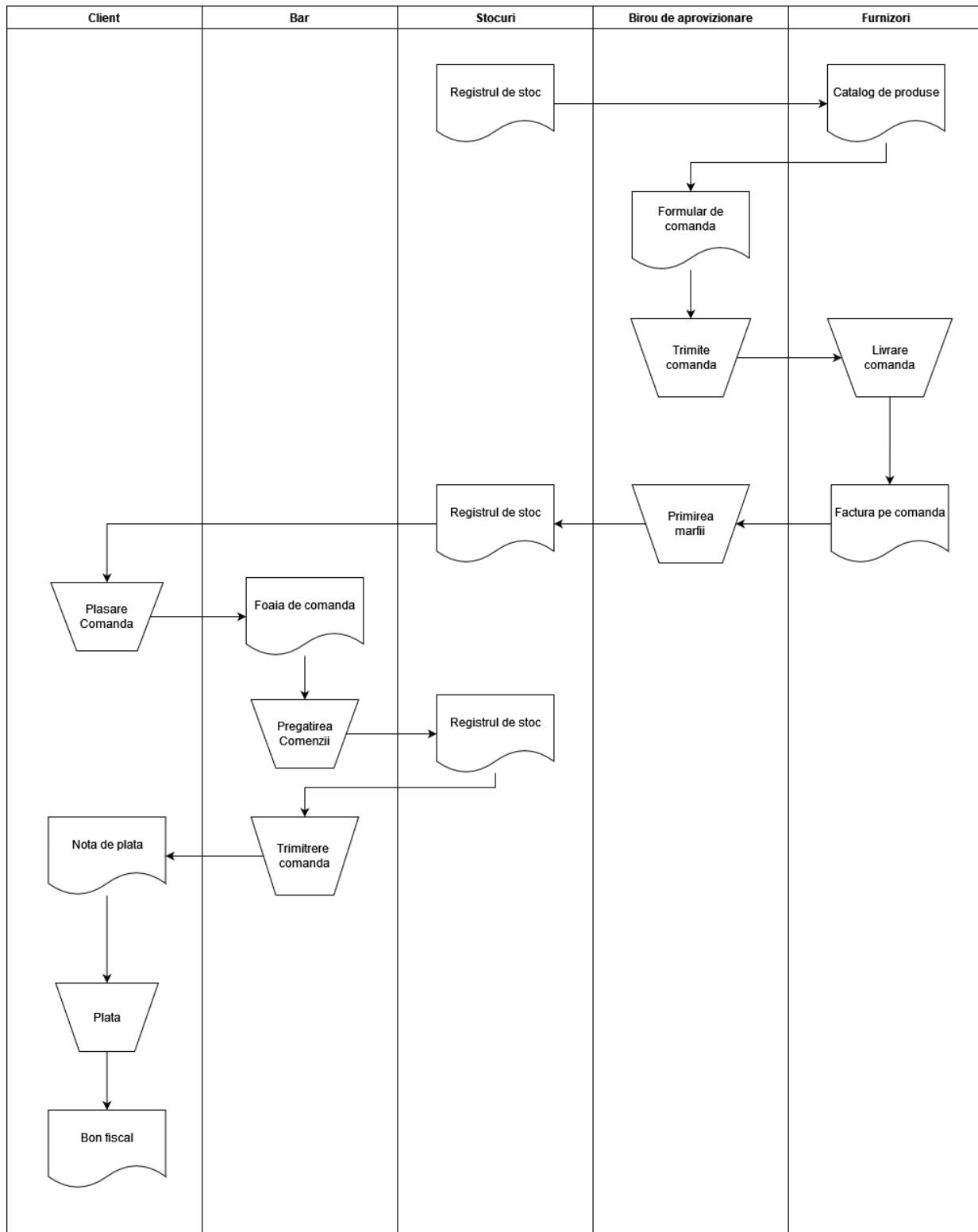
### **Personalul de curătenie**

Se ocupă de menținerea curăteniei generale a localului.

## 1.5. Studiul sistemului informațional

### 1.5.1. Schema fluxului informațional

*Figura 3, Schema fluxului informațional*



### 1.5.2. Descrierea circuitului informațional

1. Verificarea stocului cu ajutorul documentului, regisztr de stoc.
2. Furnizorul vine cu un catalog digital de produse disponibile.
3. Se completează formularul de comandă de la furnizor.
4. Comanda este preluată de către furnizor.
5. Furnizorul livrează comanda.
6. Preluăm factura pe comandă de la furnizor.
7. Se preia marfa de la furnizor.
8. Se adaugă produsele comandate în registrul de stoc.
9. Clientul plasează comanda la local.
10. Angajatul notează comanda pe o foaie de comandă.
11. Comanda se pregătește.
12. Se actualizează registrul de stoc dacă este cazul.
13. Comanda este transmisă clientului la masă.
14. Clientului primește de la ospătar nota de plată.
15. Clientul efectuează plata la casă.
16. Clientul primește bonul fiscal.

### 1.5.3. Descrierea documentelor utilizate

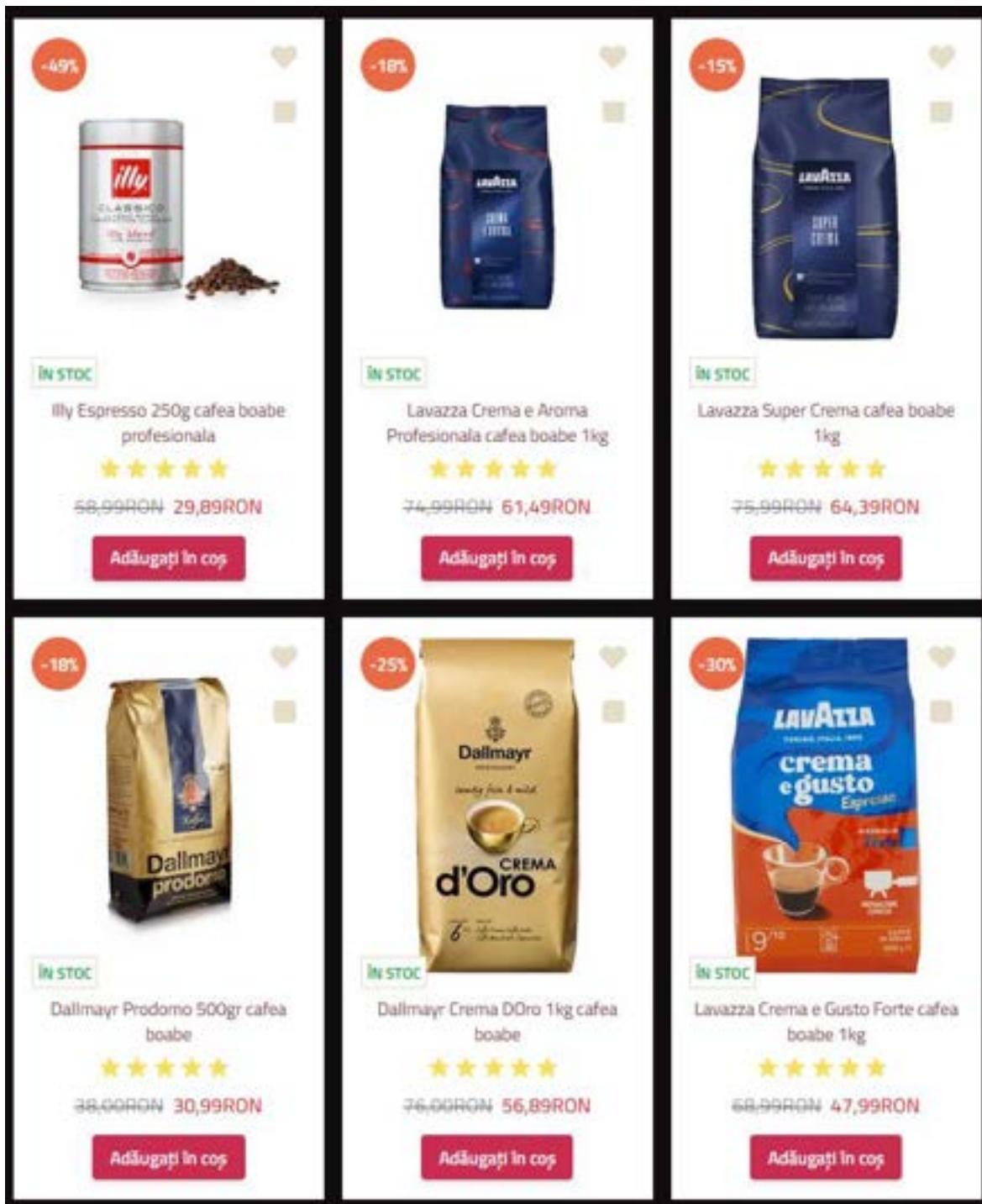
- **Registrul de stoc**

*Figura 4, Registrul de stoc*

Registrul de stoc este folosit pentru gestionarea stocurilor din bar, este actualizat la fiecare aprovizionare și la perioade stabilite, de obicei la sfârșitul zilei pentru a ține la curent stocul din local. Este structurat în forma unui tabel în care putem introduce numele produsului, intrările, ieșirile și stocul produsului dat. Cu ajutorul acestui document se iau deciziile în legătură cu ce cantitate trebuie aprovizionată de la furnizori.

- Catalogul de produse digitalizat

*Figura 5, Catalog de produse digitalizat*



Catalogul de produse este documentul prezentat de către furnizor către biroul de aprovisionare pentru a vedea ce produse sunt disponibile în stocul furnizorului, dacă au introdus produse noi sau dacă sunt oferte în perioada dată. Cu ajutorul acestui document se realizează formularul de comanda care este transmis furnizorului.

#### • Formularul de comanda

*Figura 6, Formularul de comanda*

Documentul reprezentat în figura 6 este folosit pentru a preciza produsele pe care dorim să le comandăm de la furnizor:

- În colțul din stânga sus avem datele companiei care face comanda cu câmpurile: Numele companiei, CUI, Adresa și numărul de înregistrare.
  - În colțul din dreapta sus sunt datele furnizorului.
  - Secțiunea de expediere sunt datele despre expediere care include: adresa codul poștal, strada și informațiile despre plata.
  - Tabelul în care se completează datele produselor care trebuie comandate.
  - În partea de jos se pune semnătura directorului și a contabilului.

- Factura pe comandă

*Figura 7, Factura pe comandă*

Furnizor: XXXXXXXXXX Nr.ord.registru.com./an: JXX/XXXX/XXXX Capital social: xxxx RON C.I.F.: RO XXXXXX Sediul: Str. XXXXXX, Nr. XX, Bl. XXX, Sc. XX, Et. X, Ap. X, Oras XXXX, Jud. XXXXX Contul: XXXX XXXX XXXX XXXX XXXX XXXX XXXX Banca: XXXXXXXXXXXXXXXXXXXXXXX Tel: XXXXXXXXXX	Seria XXXX Nr. XXXXXXXXX								
<b>FACTURĂ</b>									
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">Nr. facturii .....</td> <td style="padding: 2px;">Data (aaa, luna, anul) .....</td> <td style="padding: 2px;">Nr. aviz însoțire a mărfuii .....</td> </tr> </table>							Nr. facturii .....	Data (aaa, luna, anul) .....	Nr. aviz însoțire a mărfuii .....
Nr. facturii .....	Data (aaa, luna, anul) .....	Nr. aviz însoțire a mărfuii .....							
Cota TVA .....									
Nr. crt.	Denumirea produselor sau a serviciilor	U.M.	Cantitatea	Prețul unitar (fără TVA.) - Ieft -	Valoarea - Ieft -	Valoarea T.V.A. - Ieft -			
Semnătura și stempila furnizorului		Date privind expediția Numerele delegatului ..... Buletinul/carteza de identitate seria ..... nr ..... eliberată ..... Mijlocul de transport ..... nr ..... Expedierea a fost efectuat în prezența noastră la data de ..... întra ..... Semnăturile			Total: din care: accize .....  Semnătura de primire	Total de plată (col.5+col.6) .....  X			

Conform Legii nr. 82/1991, Legii nr. 571/2000 și O.M.F.P.R. nr. 2229/2006 – Sistem de înregistrare și numărare a anumitor ai gestiunile de utilizator și atât în responsabilitatea acestuia.

14-4-10/04

Acest document este emis de către furnizor în urma finalizării comenzi, documentul conține în partea de stânga sus datele furnizorului, dreapta sus datele cumpărătorului. Sub denumirea documentului „Factură” avem datele facturii cum ar fi numărul facturii, data și numărul aviz de însoțire a mărfuii. Mai jos este prezent un tabel care conține următoarele secțiuni: Nr. crt. Denumirea produselor, U.M (unitatea de măsurare), Cantitatea, Prețul per unitate, Valoare fără TVA și valoarea cu TVA.

Mai jos găsim date privind expediția, totalul de plată. La primirea comenzi dacă totul este în regulă punem și semnătură de primire în partea dreaptă jos.

- **Foaia de comandă**

*Figura 8, Foaia de comandă*



Foaia de comandă în cazul dat, prezentată printr-o macheta, în prezent foaia de comandă este o foaie dintr-un blocnot în care ospătarul scrie ce a comandat clientul și o duce mai departe la bar pentru a se pregăti comanda. Acest document conține masa care a făcut comanda, produsele alimentare comandate, data și numărul comenzi.

- Nota de plată

*Figura 9, Nota de plată*

Unitatea .....	Localitatea .....	Județul .....		
<b>NOTĂ DE PLATĂ Nr. ....</b>				
Masa nr.	Camera nr.	Ospătar		
Denumire	U.M.	Canti-tatea	P.U.	Valoare -lei-
Data	Semnătura ospătarului		<b>TOTAL DE PLATĂ</b> din care T.V.A.	

Nota de plată este documentul prin care se face plata la casă, acesta este cerut de către client, ospătarul completează nota de plată în funcție de ce a comandat clientul. Sus avem datele localului, după care urmează numărul la nota de plată, acesta de obicei semnifică numărul comenzi din ziua respectivă, mai jos avem numărul mesei și numele ospătarului care a servit masa, după urmează un tabel care este completat cu ce a comandat clientul la final fiind totalul comenzi.

- **Bonul fiscal**

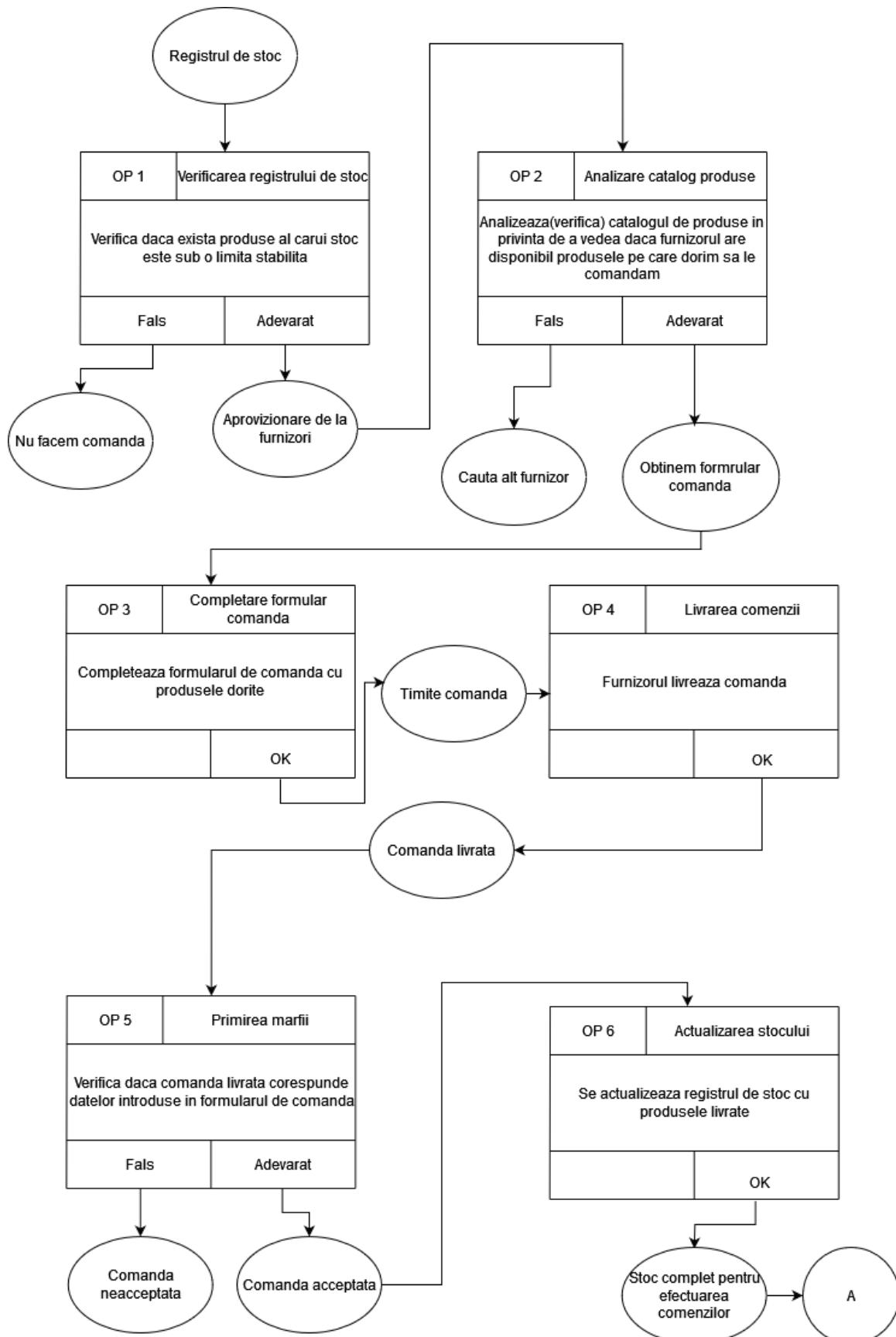
*Figura 10, Bonul fiscal*

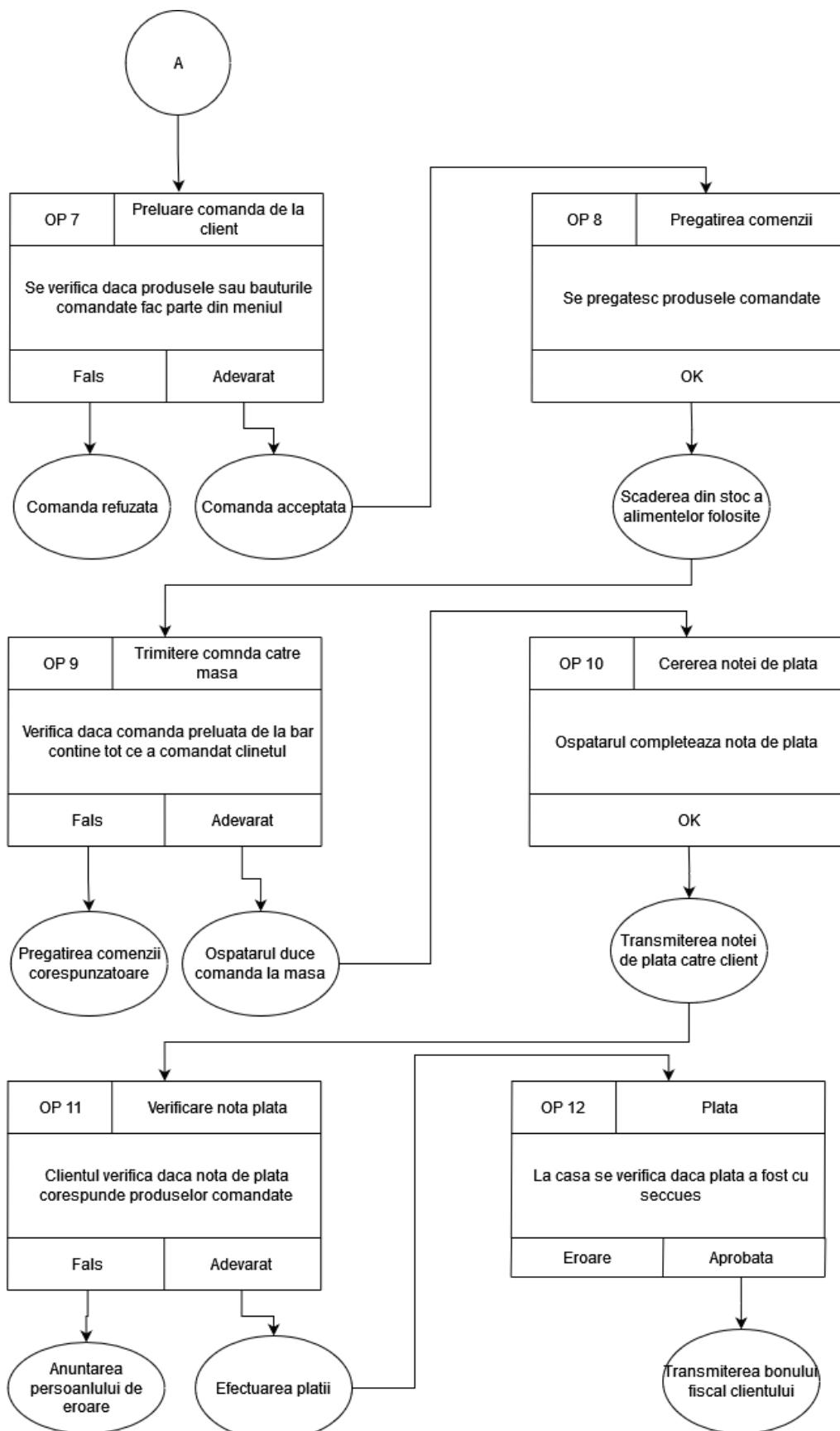


Documentul de mai sus este un exemplu de cum trebuie să arate un bon fiscal emis de casa de marcat, acesta conține în partea de sus datele localului, adresa și codul fiscal, mai jos avem numărul bonului, urmat de produse și cantitatea care a fost comandată și în partea dreapta a acestora avem prețul în funcție de cantitate. Urmează totalul, numele angajatului de la casa și numărul casei. În partea de jos avem total TVA, cotă TVA și data în care a fost emis acest document și ora, urmat la sfârșit de tot Seria și numărul bonului acestea fiind unice.

#### 1.5.4. Modelul conceptual al prelucrărilor

*Figura 11, Modelul conceptual al prelucrărilor*





### 1.5.5. Analiza critică a sistemului actual și identificarea neajunsurilor existente în funcționarea sistemului existent

După analiza sistemului informațional actual al societății comerciale SD BAR S.R.L s-au identificat următoarele neajunsuri în funcționarea sistemului existent:

1. Registrul de stoc se actualizează manual, completarea registrului de stoc manuală este un neajuns semnificativ al sistemului existent deoarece ocupă mult timp, actualizarea acestuia se efectuează doar odată pe zi ceea ce poate duce la epuizarea stocului unui produs fără a se observa din timp acest lucru. Se pot produce erori umane care pot afecta deciziile luate la etapa de aprovizionare, sau în cel mai rău caz poate să fie distrus ceea ce înseamnă că va trebui să se întocmeze de la capăt cu toate produsele din stoc.
2. Documentul foaia de comanda în sistemul actual este reprezentat de un blocnot în care este notată comanda manual de către ospătar, aceasta foaie poate fi pierdută ceea ce duce la creșterea timpului de finalizare a unei comenzi.
3. Documentul nota de plată tot se completează manual, din aceasta cauză pot apărea erori umane ceea ce poate duce la conflicte cu clienții și ocupă mult timp.
4. Plata de către client se efectuează doar la casa de marcat, tot un neajuns al sistemului existent datorită faptului că la orele de vârf există un flux foarte mare de clienți care blochează temporar activitatea de preluare și preparare a băuturilor, deoarece aceste activități sunt făcute de aceeași persoană care se ocupă cu prepararea produselor din comandă.
5. Comenzile nu sunt înregistrate în nici un sistem, aceasta poate fi un neajuns al sistemului atunci când se efectuează schimbul de ture la angajați din cauza faptului că în schimbul turelor de muncă există comenzi nefinalizate, informațiile despre aceste comenzi trebuie transmise următorului angajat. În sistemul existent această procedură se efectuează prin ajutorul unui blocnot în care se scrie masa și ce produse au fost comandate la masa respectiva, din cauza acestei metode există erori umane care pot duce la erori în legătură cu integritatea unor comenzi.

### 1.5.6. Direcții de perfecționare a sistemului actual

Luând în considerare neajunsurile determinate în subcapitolul anterior vom avea următoarele direcții de perfecționare pentru fiecare neajuns determinat:

- Registrul de stoc actualizat manual. O îmbunătățire a acestei etape din fluxul informațional ar fi automatizarea acestui proces prin implementarea unui sistem de gestiune a stocului care va beneficia de o funcționalitate de scădere automată a produsele care permit acest lucru, iar în cazul în care se produc erori la inserarea de date va fi foarte ușor de corectat aceste erori.
- Foaia de comandă, documentul dat poate fi digitalizat printr-un sistem de gestiune a comenziilor. Acest lucru va permite înregistrarea comenziilor în sistem și vizualizarea comenziilor active. Cu ajutorul acestui sistem comenziile se vor putea monitoriza mult mai ușor.
- Nota de plată, o direcție de perfecționare pentru acest neajuns ar fi tot cu un sistem de gestionare al comenziilor care va genera o notă de plată în funcție de comandă.
- Plata de către client se efectuează doar la casa de marcat, o direcție de perfecționare a acestui neajuns ar fi posibilitatea clienților de a accesa o platformă, în care să poată crea o comandă fiind prezent în local și la finalizarea acesteia să existe posibilitatea de a achita online cu cardul prin platforma respectivă.
- Comenziile nu sunt înregistrate în nici un sistem, acest neajuns poate fi îmbunătățit tot cu un sistem de gestionare al comenziilor.

---

## Cap. 2. Proiectarea de detaliu a aplicației informative

### 2.1. Definirea obiectivelor aplicației informative

Aplicațiile informative în prezent sunt prezente aproape în fiecare tip de afaceri, implementarea unei aplicații informative conduce în cele mai multe cazuri către automatizarea unor procese mulțumită cărora se economisesc bani și timp ceea ce duce la creșterea profiturilor.

Aplicația informatică are ca obiective principale automatizarea și digitalizarea proceselor din sistemul informațional prezentat în capitolul 1. Automatizarea și digitalizarea acestor procese vor permite economisirea timpului de muncă și a minimizării erorilor umane.

Aplicația informatică va implementa următoare funcționalități:

- Sistem de gestionare al comenziilor.
- Sistem de gestionare a rezervărilor.
- Panou administrativ cu posibilitatea de generare a rapoartelor.
- Gestionarea stocului

## 2.2. Proiectarea logică și fizică a ieșirilor

### 2.2.1 Lista cu toate documentele de ieșire din aplicație

*Tabelul nr. 1, Lista cu toate documentele de ieșire*

Nr. crt.	Denumire raport	Descriere raport
1	Raport produse stoc	Raportul va fi generat din panoul administrativ pentru a se putea vizualiza datele despre fiecare produs și numărul de bucăți disponibil în stoc.
2	Raport produse pentru aprovizionare	Raportul va putea fi generat din panou administrativ, care va conține toate produsele ce necesită aprovizionare cât mai curând posibil din cauza faptului că stocul acestor produse este sub o limită stabilită
3	Raport închidere casă	Raportul se va gestiona automat la închiderea POS-ului, raportul dat va conține numărul de comenzi finalizate, numărul de comenzi anulate. Mai jos se fa afișa datele fiecarui angajat cu numărul de comenzi preluate și numărul de comenzi livrate. Si după vor fi afișate toate produsele vândute în acea zi, cantitatea și totalul.
4	Raport cu comenziile pe o anumită perioadă	Raportul cu comenzi se va genera din panoul administrativ în cazul în care va fi necesar vizualizarea tuturor comenziilor pe o anumită perioadă de timp
5	Raport cu încasări din comenzi pe perioade de timp	Acest raport se generează din panoul administrativ, pe o perioadă selectată de către administrator, sau pe toată perioada funcționării aplicației. Acest raport va conține încasările din toate comenziile finalizare din perioada aleasă
6	Raport cu datele clientilor înregistrați	Un raport generat în panoul administrativ cu datele clientilor
7	Raport cu toate rezervările	Raportul se generează din panoul administrativ cu toate rezervările pe o anumită perioadă de timp
8	Bonul fiscal	Documentul de ieșire generat la finalizarea unei comenzi
9	Raport cu numărul de comenzi preluate și livrate de către fiecare angajat.	Documentul de ieșire, generat în panoul administrativ pe o perioadă anumită de timp, în care se va genera o tabelă care va conține fiecare angajat, numărul de comenzi preluate și numărul de comenzi livrate către masă.

10	Nota de plată	Documentul de ieșire generat automat la crearea unei cu ajutorul căruia se va pregăti comanda și ulterior va fi predat la masă clientului pentru face plata ulterior cu ajutorul notei de plată.
11	Raport cu produsele vândute pe o anumită perioadă.	Documentul de ieșire cu toate produsele care au fost vândute în acea perioadă, adică cantitatea și totalul încasărilor din acele produse.
12	Raportul pentru închiderea fiscală	Documentul de ieșire generat automat la închiderea POS-ului la sfârșitul zilei după care nu se va mai putea prelua comenzi pentru acea zi.

### 2.2.2 Machetele tuturor documentelor de ieșire

*Figura 12, Raport produse stoc*

*Figura 13, Raport produse aprovizionare*

SDBAR

## Raport

### Produse ce necesită aprovizionare

Creat la data : 21/06/2024 - 15:36:33

Nr.	Nume produs	Cantitate curentă	Limită stabilită
1	Coca Cola 330ml	0	20
2	Pepsi 0.5l	0	15
3	Croissant	1	10
4	Lemonade 0.5l	1	15
5	Whipped cream	1	15
6	Irish whisky 750ml	0	3
7	Heineken330ml	1	20
8	Fanta 330ml	1	20
9	Sprite 330ml	1	20
10	Birra Moretti	1	20

*Figura 14, Raport închidere casă*

SDBAR

### Raport: închidere casă

Creat la data: 21/06/2024 - 15:39:04

Pentru ziua: 18/06/2024

Comenzi finalizate: 4

Comenzi anulate: 1

**Total incasări: 34.41 \$**

**Comenzi agajați**

Nume angajat	Comenzi preluate	Comenzi livrate
first employee1	3	1
second (nonemployee)	0	0
Pos Manager	2	3
third employee	0	0
pos2 managersecond	0	0
Mureșeanu Corneliu	0	0

**Produse vândute**

Nume produs	Cantitate vândută	Total
Coca Cola 330ml	1	2.99
Cappuccino	6	15.00
Croissant	7	13.93
Lemonade 0.5l	1	2.49

*Figura 15, Raport comenzi*

SDBAR

### Raport comenzi

Creat la data: 22/06/2024 - 15:09:45

Pentru perioada: 2024-06-21 - 2024-06-22

Total comenzi: 12

Id.	Data comenzi	Status	Client	Preluata de	Livrata de	Masa	Total
176	21/06/2024 - 11:02:15	Finished	Unknown	Pos Manager	Pos Manager	Bar	1.50\$
177	21/06/2024 - 12:55:04	Finished	Unknown	Pos Manager	first employee1	Bar	8.57\$
178	21/06/2024 - 13:09:28	Finished	Unknown	first employee1	first employee1	Bar	25.25\$
179	21/06/2024 - 15:53:16	Finished	Unknown	Pos Manager	Pos Manager	Bar	37.74\$
180	21/06/2024 - 16:04:26	Finished	Unknown	Pos Manager	Pos Manager	Bar	7.80\$
181	21/06/2024 - 16:04:30	Finished	Unknown	Pos Manager	Pos Manager	Bar	9.59\$
182	21/06/2024 - 16:04:35	Finished	Unknown	Pos Manager	Pos Manager	Bar	45.96\$
183	21/06/2024 - 19:53:42	Canceled	ciobanica radu	Unknown	Unknown	1	10.10\$
184	21/06/2024 - 20:41:53	Finished	Unknown	first employee1	Pos Manager	Bar	8.60\$
185	21/06/2024 - 21:48:11	Finished	Unknown	first employee1	first employee1	Bar	5.49\$
186	21/06/2024 - 21:49:42	Finished	Unknown	first employee1	Pos Manager	Bar	2.99\$
187	22/06/2024 - 14:54:01	Finished	tester client	Pos Manager	Pos Manager	1	61.97\$

*Figura 16, Raport încasări*

SDBAR

### Raport încasări

Creat la data: 21/06/2024 - 15:45:08

Pentru perioada: 02/06/2024 - 21/06/2024

Comenzi finalize: 101

Comenzi anulate: 6

**Total încasări: 7859.12 \$**

*Figura 17, Raport date clienți*

SDBAR

### Raport

#### Date clienti

Creat la data: 22/06/2024 - 15:07:31

Date clienți

<b>Id</b>	<b>Nume</b>	<b>Prenume</b>	<b>Email</b>	<b>Nr telefon</b>	<b>Comenzi finisate</b>	<b>Total</b>
2	radu	clobanica	radu@gmail.com	0786661664	0	0.00 \$
3	sorina	damaschin	sorina@gmail.com	0786661664	0	0.00 \$
7	cian	jackson	cian@gmail.com	+40786661773	0	0.00 \$
8	iulian	ionel	ionel@gmail.com	+40786661668	0	0.00 \$
9	iuliana	museteanu	iuliana@gmail.com	+40756664663	0	0.00 \$
10	adriana	museteanu	adriana@gmail.com	+40786664773	0	0.00 \$
11	jhon	jhonson	jhonny@gmail.com	+40748884883	0	0.00 \$
12	mariana	clobanu	clobanu@gmail.com	+40847773882	0	0.00 \$
13	sorinica	damaschin	sorinica@gmail.com	+40758884388	0	0.00 \$
14	iulian	cheoseaua	cheoseaua@gmail.com	+40798883774	0	0.00 \$
16	client1	client	client1@gmail.com	+40787774332	0	0.00 \$
17	cian123	jackson	cian123@gmail.com	+40786661773	0	0.00 \$
18	iulian	iulianus	dududimmitri37@yahoo.com	+40701010101	0	0.00 \$
19	client	tester	client.tester@gmail.com	0785553773	1	61.97 \$

*Figura 18, Raport rezervări*

SDBAR

### Raport

#### Rezervari

Creat la date: 22/06/2024 - 15:04:32

Type your text

Pentru preioada: 1900-01-01 - 1900-01-01

<b>Id</b>	<b>Data</b>	<b>Nr persoane</b>	<b>Nume</b>	<b>Prenume</b>	<b>Nr telefon</b>	<b>Status</b>	<b>Durata</b>	<b>Masa</b>
65	23/06/2024 - 10:22:00	5	Iulius	Cesar	0786664773	Confirmed	5 hours	2
68	28/06/2024 - 23:53:00	3	Test	Tester	0703591126	Confirmed	5 hours	2
70	28/06/2024 - 09:50:00	3	client	tester	0785553773	Unconfirmed	3 hours	1
71	25/06/2024 - 14:50:00	3	client	tester	0785553773	Unconfirmed	3 hours	1

*Figura 19, Raport bon fiscal*

### Bon fiscal

Nume local: SD Bar  
 Adresa: Strada X nr. 12  
 Orasul: Bucuresti  
 Cif: 86374823

Descriere	TVA	Pret (\$)
Macchiato	9%	1.5 USD
Lemonade 0.5l	9%	2.49 USD
croissant	9%	1.99 USD
Cappuccino	9%	2.5 USD
Espresso	9%	1.3 USD
Fanta 330ml	9%	3.99 USD
Sprite 330ml	9%	3.99 USD
Whisky Cola	9%	9.99 USD
Cocktails1	9%	9.99 USD

**Total: 37.74 \$**  
 din care TVA 9%: 1.60 \$  
 din care TVA 19%: 3.80 \$  
 Bani primiti: - \$.  
 Rest: - \$.

Metoda de plată: card  
 Data: 21/06/2024  
 Ora: 15:53:16  
 Bon fiscal nr: 000179  
 POS nr: 001

Multumim și va mai așteptam!!!

*Figura 20, Raport comenzi angajați*

SDBAR

### Raport : comenzi angajati

Creat la data: 21/06/2024 - 15:57:06  
 Pentru perioada: 02/06/2024 - 21/06/2024

**Comenzi angajati**

Nume prenume	Comenzi preluate	Comenzi livrate
first employee1	73	43
second jhonsonemployee	15	0
Pos Manager	18	17
third employee	0	0
pos2 managersecond	0	0

Figura 21, Raport nota comandă



Data: Fri, Jun 21, 2024, 12:55:04

**Nota comanda**

Masa: bar

Produse:

Nume	Cantitate	Pret	Total
Coca Cola 330ml	1	2.99\$	2.99\$
Pepsi 0.5l	1	1.79\$	1.79\$
Espresso	1	1.30\$	1.30\$
Lemonade 0.5l	1	2.49\$	2.49\$

Nume angajat: Manager Pos  
 Comanda nr: 177  
 Status comanda: Acceptata

**Total: 8.57\$**

Figura 22, Raport produse vândute



**Raport**

**Produse vândute**

Creat la data: 21/06/2024 - 16:09:48  
 Pentru perioada: 02/06/2024 - 21/06/2024

**Produse vândute**

No.	Nume produs	Cantitate vândute	Total
1	Cappuccino	1164	2910.00 \$
2	Irish whisky 750ml	89	4449.11 \$
3	Macchiato	61	91.50 \$
4	Coca Cola 330ml	45	134.55 \$
5	Croissant	42	83.58 \$
6	Lemonade 0.5l	27	67.23 \$
7	Pepsi 0.5l	23	41.17 \$
8	Irish coffee	16	36.80 \$
9	Sprite 330ml	8	31.92 \$
10	Fanta 330ml	6	23.94 \$
11	Espresso	5	6.50 \$
12	Heineken330ml	3	17.97 \$
13	Whisky Cola	2	19.98 \$
14	Cocktails1	2	19.98 \$
15	Birra Moretti	1	5.99 \$
16	Cocktail2	1	19.99 \$

*Figura 23, Raport închidere casă fiscală*

Nume local: SD Bar	
Adresa: Strada X nr. 12	
Orasul: Bucuresti	
CUI: 86374823	
<b>Raport: inchidere fiscală</b>	
Numar bonuri emise: 7 bonuri	
Total incasari: 136.41 \$	
Din care TVA:	
TVA 9%: 5.26 \$	
TVA 19%: 14.80 \$	
Total valoare incasari: 136.41 \$	
Data inchidere fiscală:	
21/06/2024 - 16:04:38	
POS: 001	

## 2.3. Proiectarea logică și fizica a intrărilor

### 2.3.1 Lista cu toate intrările din aplicație

*Tabelul nr. 2, Lista cu toate situațiile de intrare al unui utilizator de tip Client*

Nr. Crt.	Denumire situație intrare	Descriere situație intrare
1	Autentificarea în aplicație	Video-formatul în care se vor introduce datele utilizatorului pentru a putea accesa funcționalitățile aplicației
2	Crearea unui nou cont	Video formatul prin care se introduc datele necesare înregistrării unui nou client în aplicație
3	Creare unei rezervări de către client	Video formatul pentru introducerea datelor necesare creării unei noi rezervări de către client

*Tabelul nr. 3, Lista cu toate situațiile de intrare al unui utilizator de tip Angajat*

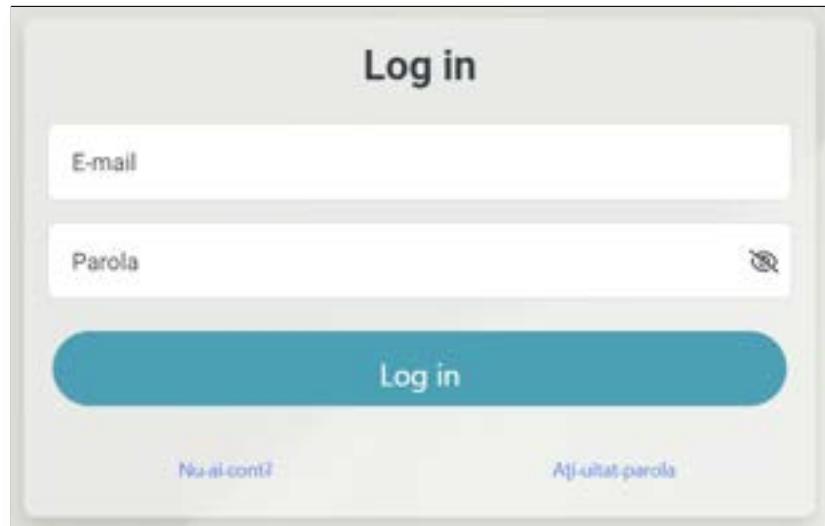
Nr. Crt.	Denumire situație intrare	Descriere situație intrare
3	Echilibrarea de stoc pentru un produs ales	Video-formatul în care se va selecta produsul pentru care se efectuează echilibrarea de stoc, se va selecta și categoria balansării și cantitatea care va fi scăzută.
4	Adăugarea cantității unui produs la efectuarea unei aprovizionări sau la necesitatea de a efectua acest lucru	Video formatul prin care se va alege produsul pentru care se va adăuga o cantitate introdusă de către un angajat!
5	Creare unei rezervări de către angajat	Video formatul pentru introducerea datelor necesare creării unei noi rezervări de către angajat

*Tabelul nr. 4, Lista cu toate situațiile de intrare al unui utilizator de tip Administrator*

Nr. Crt.	Denumire situație intrare	Descriere situație intrare
6	Adăugarea unui nou produs	Video-formatul prin care administratorul va putea înregistra în stoc produse noi prin completarea tuturor câmpurilor din video-format.
7	Adăugarea unui produs complex	Video formatul prin care se va putea adăuga în sistem un nou produs complex prin completarea tuturor câmpurilor din video-format.
8	Adăugarea unei noi categorii pentru produse	Video formatul cu ajutorul căruia se va putea introduce în baza de date o nouă înregistrare pentru categoriile de produse.
9	Adăugarea unei noi categorii de balansare de stoc	Video formatul prin care se va introduce o nouă categorie pentru balansările de stoc.
10	Crearea unui cont pentru un angajat nou	Video formatul prin care se va putea crea un nou cont pentru un nou angajat.
11	Modificarea datelor unui angajat	Video formatul prin care se va putea modifica câmpurile care permit acest lucru pentru un angajat.
12	Adăugarea unei noi mese	Video formatul prin care se va putea adăuga o nouă masă în sistem.
13	Generarea rapoartelor	Video formatul unde se va selecta opțiunile necesare generării unui raport.
14	Schimbarea parolei	Video formatul prin care se va efectua schimbarea parolei contului autentificat în sistem, acest video format este accesibil pentru toți utilizatorii.

### 2.3.2 Machetele pentru toate video formatele din aplicația informatica

*Figura 24, Video-formatul pentru autentificare*



The image shows a login interface titled "Log in". It features two input fields: "E-mail" and "Parola" (Password), both with placeholder text and a visibility icon. Below the fields is a large blue "Log in" button. At the bottom of the screen are two links: "Nu ai cont?" (Don't have an account?) and "Ați uitat parola?" (Forgot password?).

*Figura 25, Video-formatul pentru creare unui nou cont*



The image shows a registration interface titled "Inregistreaza-te". It contains six input fields: "Nume" (Name) and "Prenume" (First name) in the first row; "E-mail" and "Număr de telefon" (Phone number) in the second row; and "Parola" (Password) and "ReParola" (Repeat password) in the third row. Each field has a placeholder text and a visibility icon. Below the fields is a large blue "Creează cont" (Create account) button. At the bottom of the screen is a link: "ai deja un cont?" (Do you already have an account?).

*Figura 26, Video-formatul pentru crearea unei noi rezervări*

**Creează-ți noua rezervare!**

Pentru rezervări mai complexe vă rugăm să ne sunați +4071666273

Selectați numărul de invitați

Selectați numărul tabelului

Câte ore vei sta

Selectați data rezervării  
mm/dd/yyyy

Selectaază-ți ora  
--:--

**Creați rezervare**

*Figura 27, Video-formatul pentru echilibrarea de stoc*

**Echilibrarea stocurilor**

Vă rugăm să trimiteți acest formular numai în cazuri de risipă de produs!

Selectați un produs

Selectați eliminarea categoriei

Cantitate de scos din stoc

**Trimite**

Figura 28, Video-formatul pentru adăugarea cantității de la aprovizionare

## Adăugați cantitățile de produse în stoc

Produs nr. 1:

Selectați produsul

Cantitate adăugată

Adăugați un produs nou

Scoateți produsul

Trimite



Figura 29, Video-formatul pentru crearea unei rezervări de către angajat

## Creați rezervare!

Nume

Nume

Număr de telefon

Selectați numărul de invitat

Selectați numărul tabelului

Durată

Selectați data rezervării  
mm/dd/yyyy

Selectați ora de începere  
--:-- --

Creați rezervare

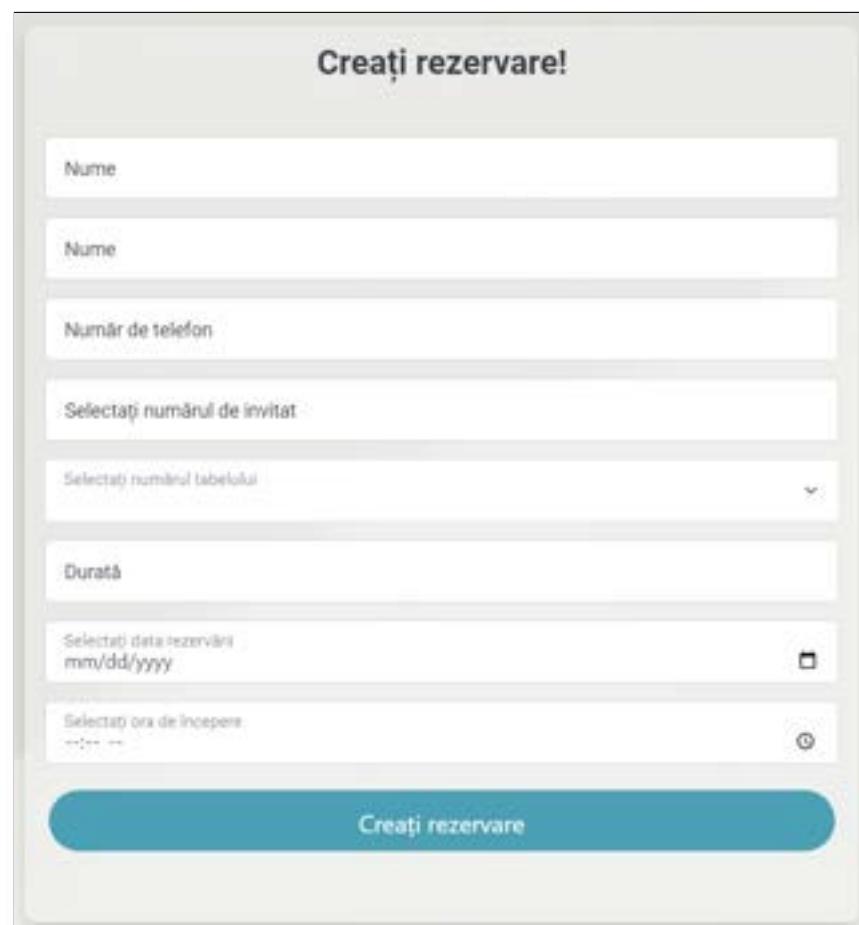


Figura 30, Video-formatul pentru crearea unui produs nou

**Completati formularul va rog!** X

Numele produsului

Preț unitar

Unitatea de măsură

Produsul este disponibil pentru clienți?

da  
 Nu

Selecteaza categoria

Cantitate

Verificarea aprovisionării

Selectați TVA aplicat

**Adăugați un produs nou**

*Figura 31, Video-formatul pentru creare unui produs complex*

**Completati formularul va rog!** ×

Numele produsului

Preț unitar

Unitatea de măsură

Produsul este disponibil pentru clienți?

da  
 Nu

Selecteaza categoria

Selectați TVA aplicat

1.

ID-ul produsului component

Cantitate folosita

**Scoateți produsul**

**Adăugați un produs component nou**

**Creați un nou produs complex**

*Figura 32, Video-formatul pentru creare unei noi categorii*

**Completati formularul va rog!**

Numele categoriei

Această categorie este pentru articolele din meniu?

da  
 Nu

**Adăugați o categorie**



*Figura 33, Video-formatul pentru crearea unei noi categorii pentru balansări*

**Completati formularul va rog!**

Numele categoriei

**Adăugați o categorie de sold!**



*Figura 34, Video-formatul pentru crearea unui nou cont pentru un angajat*

**Înregistrați noul angajat**

Nume

Nume

E-mail

Salariu

Rol  
Angajat

Parola

ReParola

**Creați un nou angajat**



Figura 35, Video-formatul pentru modificarea datelor unui angajat

**Completati formularul va rog!**

Introdu noul salariu  
1500

Schimbați rolul  
Angajat

**Aplica schimbarile**



Figura 36, Video-formatul pentru adăugare unei mese noi

**Completati formularul va rog!**

Capacitatea mesei

**Adăugați o masă nouă!**



Figura 37, Video-formatul pentru generarea rapoartelor

**Generați câștiguri totale între 2 date!**

Data de început  
mm/dd/yyyy

Data de încheiere  
mm/dd/yyyy

**Generează raport**



## 2.4. Proiectarea sistemului de codificare a datelor

Aplicația informatică folosește un sistem de codificare bazat pe valoare cheii primare pentru identificarea unică a unei înregistrări din baza de date, aceasta fiind aplicată pentru fiecare tabel din baza de date.

Un alt sistem de codificare de tip numeric folosit în aplicația informatică este utilizat pentru stabilirea statusului unei comenzi, aceste coduri sunt definite astfel:

*Tabelul nr. 5, Sistemul de codificare pentru comenzi*

Cod	Semnificația
1	Comandă creată
2	Comandă acceptată
3	Comandă predată
4	Comandă finalizată
5	Comandă anulată

Sistemul de crearea și gestionarea a rezervărilor include un sistem de codificare de tip boolean pentru stabilirea statusului unei rezervări, aceste coduri sunt definite astfel:

*Tabelul nr. 6, Sistemul de codificare pentru rezervări*

Cod	Semnificație
True	Rezervare confirmată
False	Verificare rezervare

Un alt sistem de codificare este prezent pentru stabilirea statusului unei mese din local, acest sistem de codificare este de tip boolean fiind definit astfel:

*Tabelul nr. 7, Sistemul de codificare pentru mese*

Cod	Semnificație
True	Masă ocupată
False	Masă liberă

Pentru realizarea închiderii și deschiderii casei de marcat se folosește un sistem de codificare de tip boolean, în tabela organizare din baza de date:

*Tabelul 8 ,Sistem de codificare pentru organizație*

Cod	Semnificație
<b>True</b>	Casă deschisă
<b>False</b>	Casă închisă

Pentru a putea bloca accesul unui utilizator la aplicația informatică din diferite motive de securitate, este un sistem de codificare în tabelele clienți și angajați, de tip boolean prin care se identifică dacă utilizatorul are acces sau nu:

*Tabelul 9, Sistem de codificare pentru accesul utilizatorilor*

Cod	Semnificație
<b>True</b>	Acces blocat
<b>False</b>	Acces permis

## 2.5. Proiectarea bazei de date

Prin aplicația informatică se gestionează mai multe părți al sistemului informațional analizat în capitolul 1, din aceasta cauze este necesar proiectarea unei baze de date în care se vor stoca toate datele necesare pentru funcționarea corectă a tuturor modulelor. Fiecare tabel din baza de date conține doar atributele necesare pentru îndeplinirea funcționalităților stabilite cu clientul.

Baza de date conține următoarele tabele:

- Tabela comenzi

*Tabelul nr. 10, Comenzi*

Denumire coloană	Tip de data
<b>Cod_comandă</b>	Număr întreg
<b>Dată_comandă</b>	Dată
<b>Status_comandă</b>	Număr întreg
<b>Cod_client</b>	Număr întreg
<b>Cod_angajat_prealuare</b>	Număr întreg
<b>Cod_angajat_livrare</b>	Număr întreg
<b>Cod_masă</b>	Număr întreg
<b>Tips</b>	Număr real

Tabela nr. 10 este definită pentru stocarea comenziilor în sistem, tabela conține atributul codul comenzi care reprezintă codul unic al fiecărei comenzi, data la care s-a creat comanda, statusul comenzi, tips-ul pentru angajat, codul clientului care a făcut comanda și codul unic al angajatului care a preluat comanda și a angajatului care a livrat comanda către masă.

- Tabela produse

*Tabelul nr. 11, Produse*

Denumire coloană	Tip de data
<b>Cod_produs</b>	Număr întreg
<b>Nume_produs</b>	Șir de caractere
<b>Pret_unitar</b>	Număr real
<b>Unitate_măsurare</b>	Șir de caractere
<b>Valabil_consumator</b>	Boolean
<b>Produs_complex</b>	Boolean
<b>Cod_categorie</b>	Număr întreg
<b>Cantitate_disponibila</b>	Număr întreg
<b>Cantitate_aprovizionare</b>	Număr întreg
<b>Tva_inclus</b>	Număr întreg

Tabela nr. 11 conține toate datele necesare pentru produsele din stoc, aici se regăsesc atributele pentru numele produsului, prețul unitar al produsului, unitatea de măsurare, valabil consumator este un atribut prin care se face verificarea pentru a ști dacă produsul se poate vinde direct consumatorului sau nu este un produs dedicat vânzării directe, aceste produse care nu sunt dedicate vânzării directe deseori sunt componente ale produselor complexe.

- Tabela produse\_comanda

*Tabelul nr. 12, Produse - comandă*

Denumire coloană	Tip de data
<b>Cod_produs_comandă</b>	Număr întreg
<b>Cod_comandă</b>	Număr întreg
<b>Cod_produs</b>	Număr întreg
<b>Cantitate_comandată</b>	Număr întreg
<b>Pret_unitar</b>	Număr real

Tabelul nr. 12 este un tabel de legătura pentru tabela nr. 10 și tabela nr. 11 pentru a se identifica toate produsele care aparțin unei comenzi, această tabelă conține 2 câmpuri care sunt prețul unitar și cantitatea produselor comandate, s-a adăugat prețul unitar și în această tabel, deși se regăsește și în tabela nr. 11 deoarece în cazul în care se va schimba prețul unor

produse sa nu se afecteze comenziile vechi din sistem sau dacă se va dori de vizualizat evoluția prețului a unui produs.

- Tabela categorii

*Tabelul nr. 13, Categorii*

Denumire coloană	Tip de date
<b>Cod_categorie</b>	Număr întreg
<b>Nume_categorie</b>	Şir de caractere
<b>Valabil_menu</b>	Boolean

Tabela nr. 13 este pentru adăugarea sau modificarea categoriilor pentru produsele din stoc, s-a adăugat aceasta tabelă pentru gestionarea mai ușoara a categoriilor.

- Tabela rezervări

*Tabelul nr. 14, Rezervări*

Denumire coloană	Tip de date
<b>Cod_rezervare</b>	Număr întreg
<b>Dată_rezervare</b>	Data
<b>Număr_oaspeți</b>	Număr întreg
<b>Status_rezervare</b>	Boolean
<b>Ore_rezervate</b>	Număr întreg
<b>Cod_client</b>	Număr întreg
<b>Cod_masă</b>	Număr întreg
<b>Prenume</b>	Şir de caractere
<b>Nume</b>	Şir de caractere
<b>Număr_telefon</b>	Şir de caractere

Tabela nr. 14 este pentru gestionarea rezervărilor din aplicația informatică, această tabelă conține toate datele minime necesare pentru crearea și vizualizarea rezervărilor create de către clienți sau angajați utilizând aplicația informatică, atributele prenume, nume, și număr telefon pot fi adăugate doar de către utilizatorii cu rolul de angajați.

- Tabela organizație

*Tabelul nr. 15, Organizații*

Denumire coloană	Tip de date
<b>Cod_organizație</b>	Număr întreg
<b>Nume</b>	Șir de caractere
<b>Adresă</b>	Șir de caractere
<b>Adresă_logo</b>	Șir de caractere
<b>Status_funcționare</b>	Boolean
<b>CIF</b>	Număr întreg
<b>Oraș</b>	Șir de caractere

Tabela nr. 15 face parte din baza de date pentru a stoca informațiile firmei și pentru a se putea modifica statusul localului, în închis sau deschis, pentru a se putea efectua comenzi în aplicația informatică doar în cazul în care localul este deschis.

- Tabela clienti

*Tabelul nr. 16, Clienti*

Denumire coloană	Tip de data
<b>Cod_client</b>	Număr întreg
<b>Nume</b>	Șir de caractere
<b>Prenume</b>	Șir de caractere
<b>Email</b>	Șir de caractere
<b>Cod_user</b>	Șir de caractere
<b>Număr_telefon</b>	Șir de caractere
<b>Lock_status</b>	Boolean

Tabela nr. 16 este dedicată pentru stocarea datelor clientilor necesare pentru conectarea în aplicația informatică, atributul cod user este pentru stocarea codului care face legătura cu o tabela din baza de date generata pentru funcționalitățile de autentificare și autorizare.

- Tabela angajat

*Tabelul nr. 17, Angajați*

Denumire coloană	Tip de data
<b>Cod_angajat</b>	Număr întreg
<b>Nume</b>	Şir de caractere
<b>Prenume</b>	Şir de caractere
<b>Email</b>	Şir de caractere
<b>Salariu_fix_brut</b>	Număr real
<b>Cod_user</b>	Şir de caractere
<b>Rol</b>	Şir de caractere
<b>Lock_status</b>	Boolean

Tabela nr. 17 conține datele angajaților, tabela dată tot conține codul pentru user care este necesar pentru a face referință la baza de date care efectuează autentificarea și autorizarea în aplicația informatică. S-a mers pe principiul datelor minime necesare.

- Tabela componente\_produs\_complex

*Tabelul nr. 18, CPC*

Denumire coloană	Tip de data
<b>Cod_cpc</b>	Număr întreg
<b>Cod_produs_complex</b>	Număr întreg
<b>Cod_produs_component</b>	Număr întreg
<b>Cantitate_folosită</b>	Număr întreg

Tabelul nr. 18 este un tabel de legătura, doar că face legătura dintre mai multe produse, pentru identificarea produselor componente pentru un produs complex, de exemplu la cafenea sunt multe produse te tip cocktail care au mai multe ingrediente, în aceste ingrediente pot să intre și sticle de 330ml de coca cola sau alte tipuri de sucuri necesare pentru prepararea băuturii și să realizează această tabelă pentru a se putea scădea din stoc automat produsele care fac parte dintr-un produs complex și care permite scăderea automată din stoc.

- Tabela balansare\_stoc

*Tabelul nr. 19, Balansare stoc*

Denumire coloană	Tip de date
<b>Cod_balansare_stoc</b>	Număr întreg
<b>Dată_balansare</b>	Data
<b>Cod_produs</b>	Număr întreg
<b>Cantitate_scăzută</b>	Număr întreg
<b>Categorie_balansare</b>	Număr întreg

Tabelul nr. 19 este pentru stocarea informației necesare balansărilor de stoc, deoarece într-o cafenea sunt foarte multe produse care nu pot fi scăzute automat din stoc, s-a realizat această tabelă prin care se face scăderea manuală a cantității produselor selectate din diferite motive, motivul scăderii fiind reprezentat de o categorie de balansare.

- Tabela categorii\_balansare

*Tabelul nr. 20, Categorii balansare*

Denumire coloană	Tip de dată
<b>Cod_categorie_balansare</b>	Număr întreg
<b>Nume_categorie</b>	Șir de caractere

Tabelul nr. 20 este pentru stocarea categoriilor de balansare pentru a ușura crearea unei balansări de stoc în care trebuie introdusă categoria de balansare.

- Tabela mese

*Tabelul nr. 21, Mese*

Denumire coloană	Tip de dată
<b>Cod_masă</b>	Număr întreg
<b>Capacitate_masă</b>	Număr întreg
<b>Status_masă</b>	Boolean

Tabelul nr. 21 este pentru stocarea informațiilor despre mesele localului, și pentru vizualizarea statusului mesei, care poate fi ocupată sau liberă.

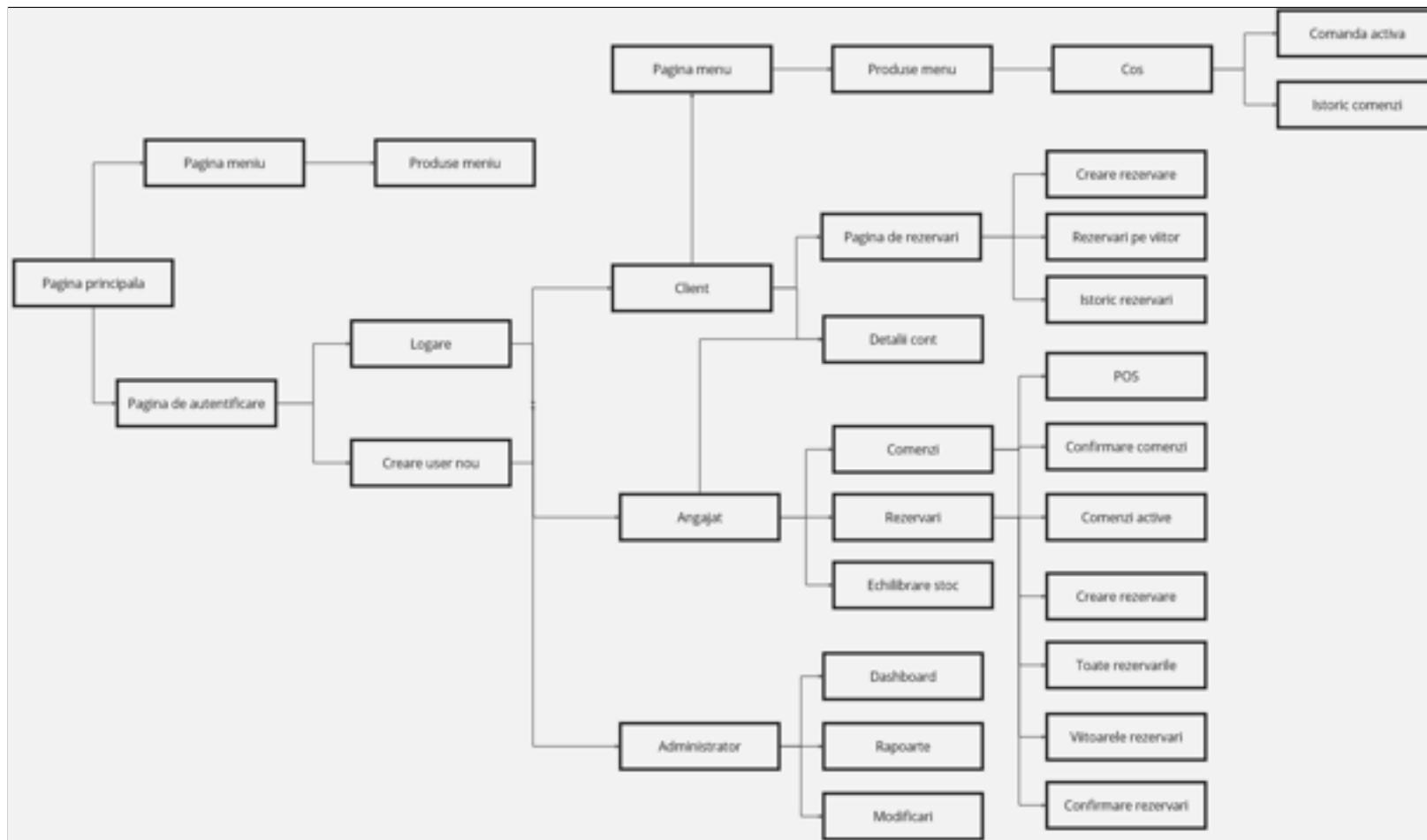
## 2.5.1 Schema relațională a bazei de date

Figura 38, Schema relațională a bazei de date



## 2.6. Schema de sistem a aplicației

*Figura 39, Schema de sistem a aplicației*



## 2.7. Proiectarea interfeței aplicației

Proiectarea interfeței aplicației informaticice a fost gândită pentru a putea fi compatibilă și cu dispozitivele mobile deoarece că va fi folosită în mare parte de către clienți și de către angajați pe dispozitive mobile.

Deoarece în aplicația informatică pot fi 4 tipuri de utilizatori, și anume client, angajat, manager și administrator, pentru fiecare din aceste 4 tipuri de utilizatori sunt proiectate interfețe diferite.

- Pentru utilizatorii de tip client avem următoarele pagini

*Figura 40, Pagina acasă a aplicației*

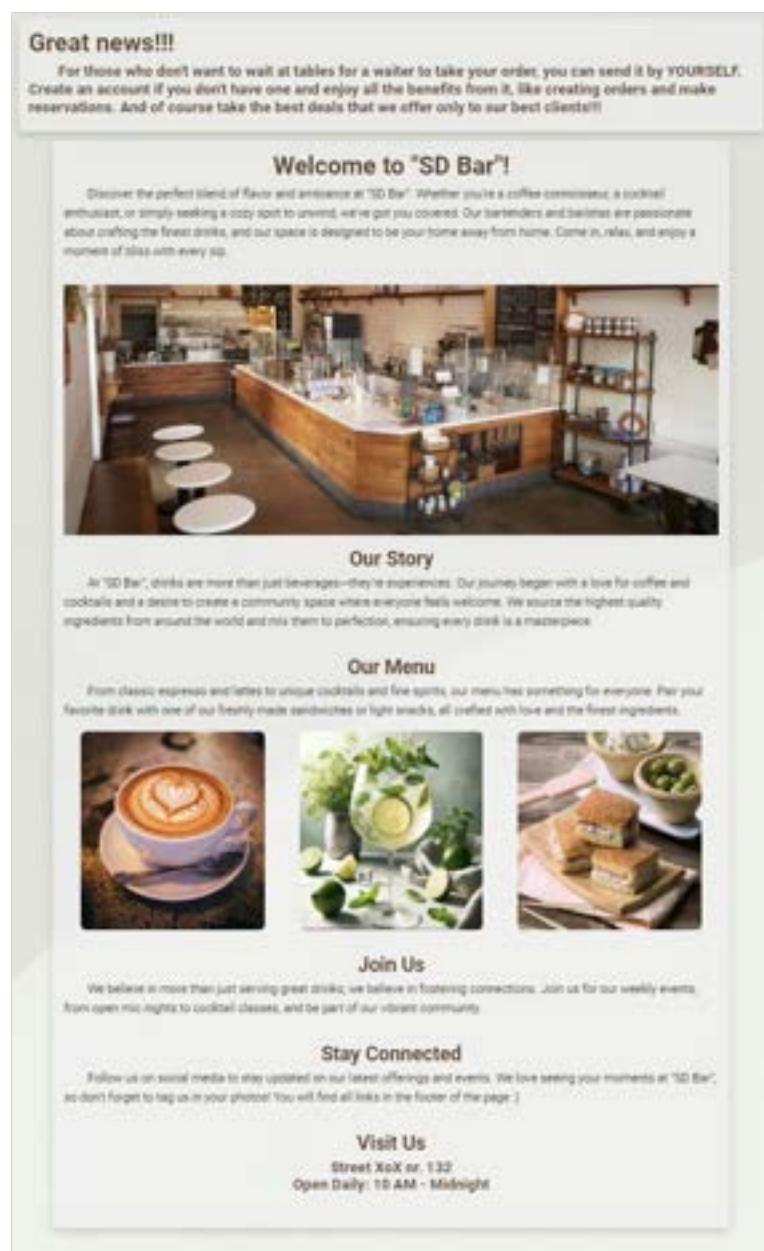


Figura 41, Pagina meniu

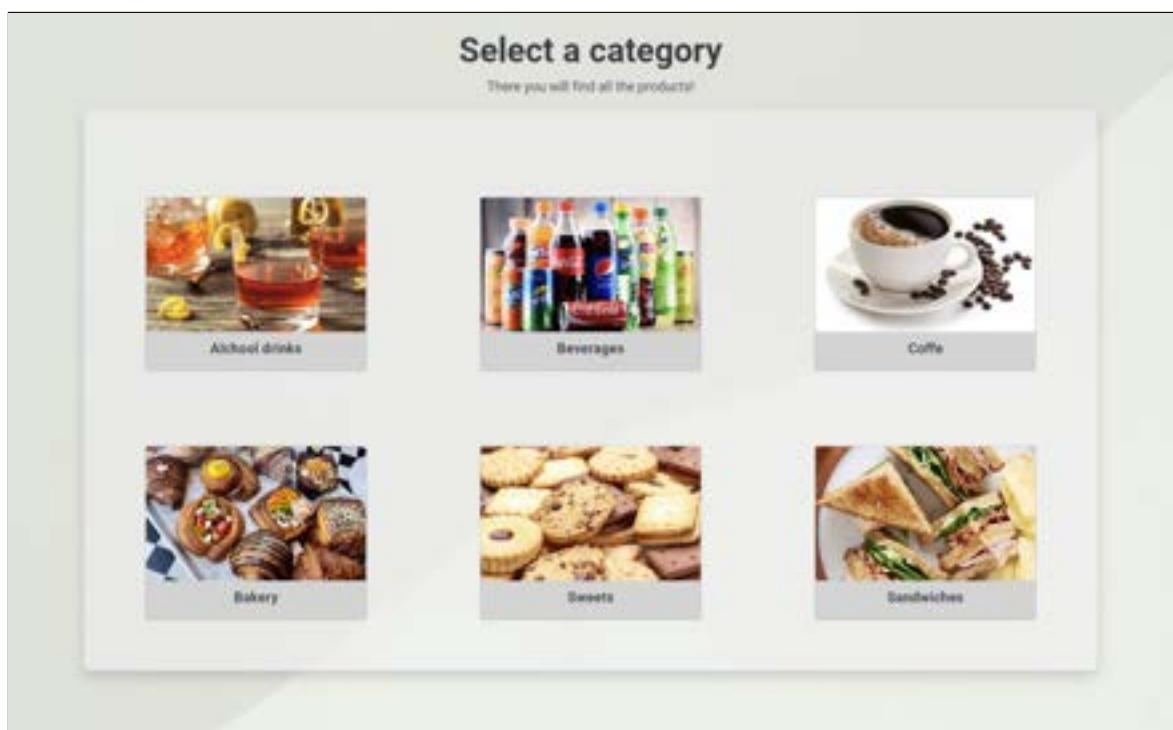


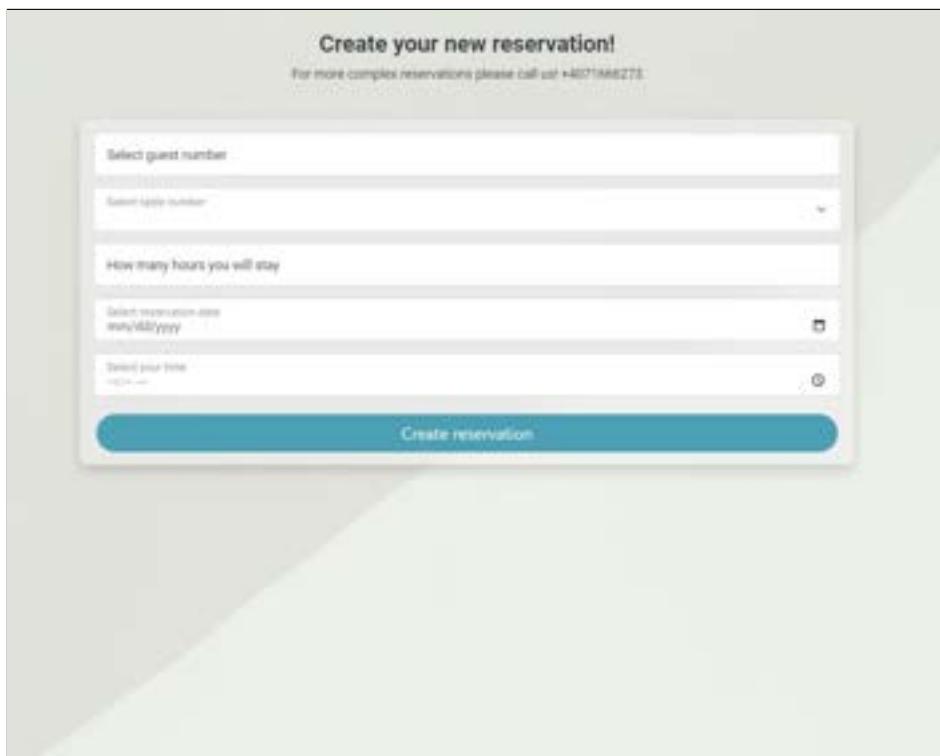
Figura 42, Pagina produse, pentru meniul selectat

The screenshot shows a product selection page for the "Coffee" category. It has a header "Products" and a note "Please select and add the products to the cart! You must be logged in." Below are four items:

Product	Price
Cappuccino	Price: 2.5\$
Macchiato	Price: 1.5\$
Espresso	Price: 1.3\$
Irish coffee	Price: 2.3\$

Each item has a blue "Add to cart" button below its name.

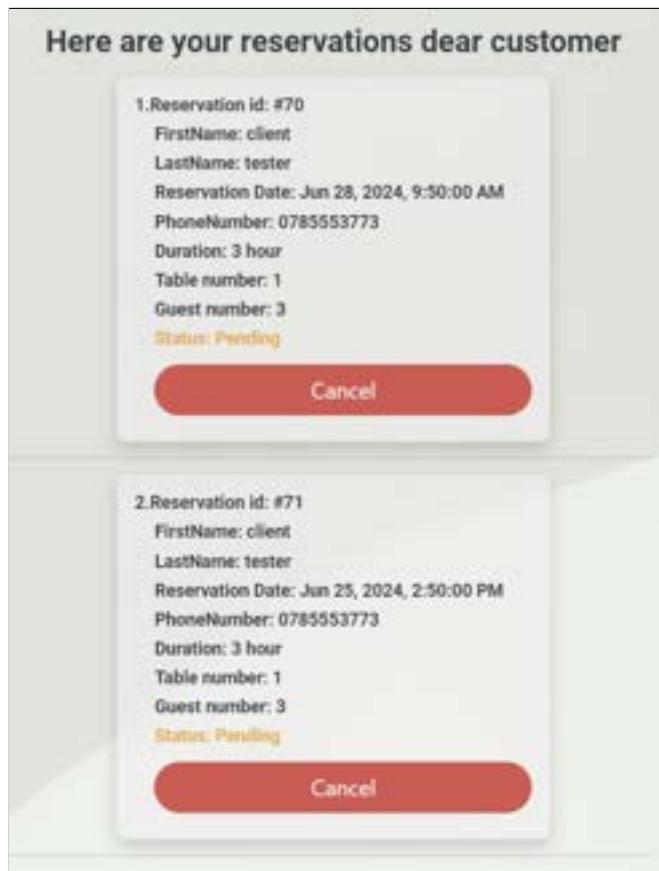
Figura 43, Pagina pentru crearea rezervării de către client



The screenshot shows a web-based reservation form titled "Create your new reservation!". It includes fields for guest count, guest type, stay duration, reservation date, and time. A large blue "Create reservation" button is at the bottom.

Select guest number:  
Guest type number:  
How many hours you will stay:  
Select reservation date:  
Select your time:  
Create reservation

Figura 44, Pagina cu rezervările viitoare ale clientului



The screenshot displays a list of future reservations for a customer. It shows two entries, each with details like reservation ID, first name, last name, reservation date, phone number, duration, table number, guest number, and status. Each entry has a red "Cancel" button below it.

1.Reservation id: #70  
FirstName: client  
LastName: tester  
Reservation Date: Jun 28, 2024, 9:50:00 AM  
PhoneNumber: 0785553773  
Duration: 3 hour  
Table number: 1  
Guest number: 3  
Status: Pending

Cancel

2.Reservation id: #71  
FirstName: client  
LastName: tester  
Reservation Date: Jun 25, 2024, 2:50:00 PM  
PhoneNumber: 0785553773  
Duration: 3 hour  
Table number: 1  
Guest number: 3  
Status: Pending

Cancel

Figura 45, Pagina cu istoricul rezervărilor

### Reservation history

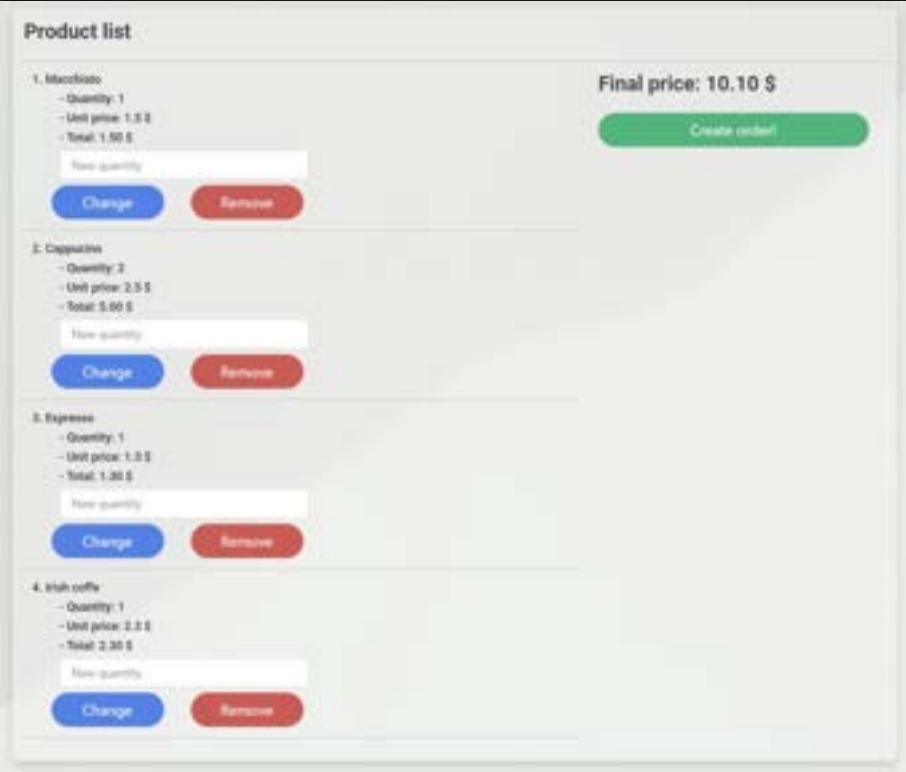
1.Reservation id: #70  
FirstName: client  
LastName: tester  
Reservation Date: Jun 28, 2024, 9:50:00 AM  
PhoneNumber: 0785553773  
Duration: 3 hour  
Table number: 1  
Guest number: 3  
**Status: Pending**

**Cancel**

2.Reservation id: #71  
FirstName: client  
LastName: tester  
Reservation Date: Jun 25, 2024, 2:50:00 PM  
PhoneNumber: 0785553773  
Duration: 3 hour  
Table number: 1  
Guest number: 3  
**Status: Pending**

**Cancel**

Figura 46, Pagina coșul de cumpărături



**Product list**

- 1. Macchiato
  - Quantity: 1
  - Unit price: 1.5 \$
  - Total: 1.50 \$

*Show quantity*

**Change** **Remove**
- 2. Cappuccino
  - Quantity: 2
  - Unit price: 2.5 \$
  - Total: 5.00 \$

*Show quantity*

**Change** **Remove**
- 3. Espresso
  - Quantity: 1
  - Unit price: 1.3 \$
  - Total: 1.30 \$

*Show quantity*

**Change** **Remove**
- 4. Irish coffee
  - Quantity: 1
  - Unit price: 2.3 \$
  - Total: 2.30 \$

*Show quantity*

**Change** **Remove**

**Final price: 10.10 \$**

**Create order**

Figura 47, Pagina comanda activă



**Order summary**

Order id: 183  
 Order date: Jun 21, 2024, 7:53:42 PM  
 Table number: 1  
 Employee name: "Waiting for confirmation"  
 Status: Pending

**Products ordered:**

- 1. Macchiato
  - Quantity: 1
  - Unit price: 1.5 \$
  - Total: 1.50 \$
- 2. Cappuccino
  - Quantity: 2
  - Unit price: 2.5 \$
  - Total: 5.00 \$
- 3. Espresso
  - Quantity: 1
  - Unit price: 1.3 \$
  - Total: 1.30 \$
- 4. Irish coffee
  - Quantity: 1
  - Unit price: 2.3 \$
  - Total: 2.30 \$

**Total to pay: 10.10\$**

**Please reach the pos to pay**

Please refresh the page in order to see if the order status was changed!

To add new products to the order it is the same procedure like creating a new one!

Figura 48, Pagina istoricul comenzilor

**Thank you for your 9 orders at our location!**

1. Order number 175

- Order date 20/06/24 08:15 PM
- Table number 1
- Employee name: "employee1"
- Status: Cancelled

Products:

1.1 Lemonade 0.5l

- Quantity: 1
- Unit price: 2.49 \$
- Total: 2.49 \$

[Get order note](#)

---

2. Order number 162

- Order date 18/06/24 10:28 PM
- Table number 1
- Employee name: "employee1"
- Status: Cancelled

Products:

2.1 Coca Cola 330ml

- Quantity: 4
- Unit price: 2.99 \$
- Total: 11.96 \$

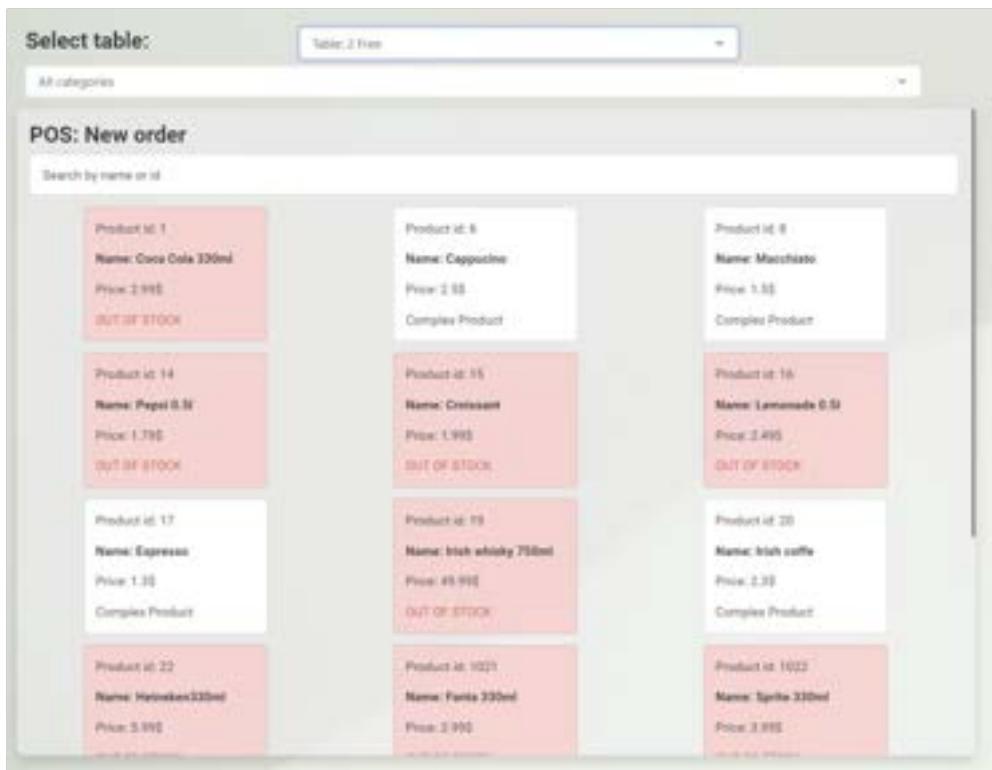
2.2 Irish whisky 750ml

- Quantity: 2
- Unit price: 49.99 \$
- Total: 99.98 \$

[Get order note](#)

- Interfețele pentru utilizatorii de tip angajat

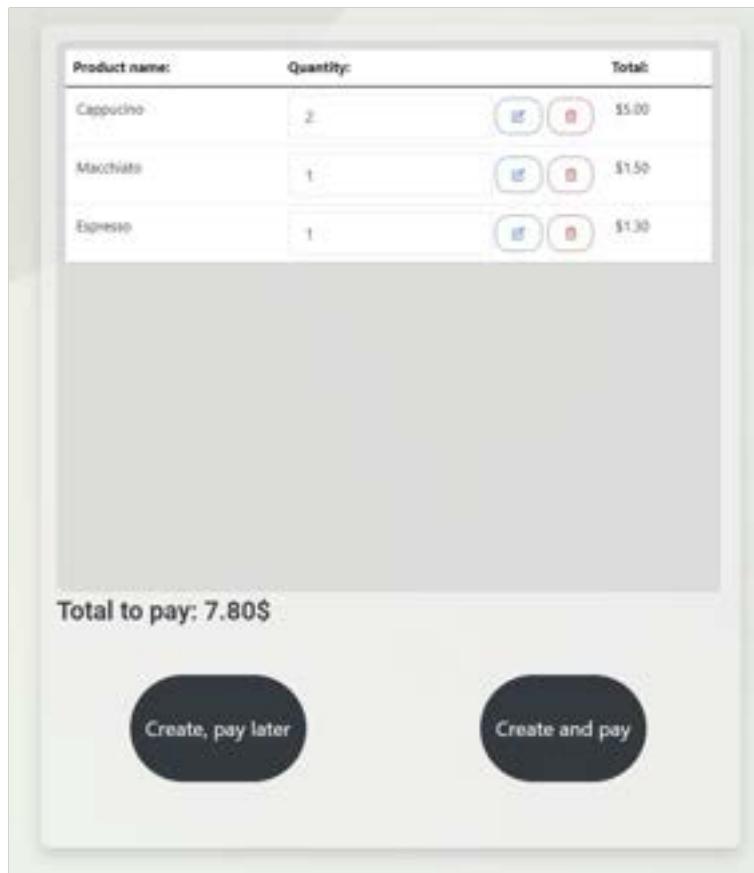
*Figura 49, Pagina POS*



The screenshot shows a POS interface titled "POS: New order". At the top left, there is a "Select table:" dropdown set to "Table 2 Free". Below it is a "All categories" dropdown. The main area is titled "POS: New order" and contains a search bar "Search by name or id". A grid of 12 product cards is displayed in three rows of four:

Product id: 1 Name: Coca Cola 330ml Price: 2.99\$ OUT OF STOCK	Product id: 6 Name: Cappuccino Price: 2.55\$ Complex Product	Product id: 8 Name: Macchiato Price: 1.55\$ Complex Product
Product id: 14 Name: Pepsi 0.5l Price: 1.79\$ OUT OF STOCK	Product id: 15 Name: Croissant Price: 1.99\$ OUT OF STOCK	Product id: 16 Name: Lemonade 0.5l Price: 2.49\$ OUT OF STOCK
Product id: 17 Name: Espresso Price: 1.30\$ Complex Product	Product id: 19 Name: Irish whisky 750ml Price: 49.99\$ OUT OF STOCK	Product id: 20 Name: Irish coffee Price: 2.30\$ Complex Product
Product id: 22 Name: Hellesbier330ml Price: 3.99\$	Product id: 1021 Name: Fanta 330ml Price: 2.99\$	Product id: 1022 Name: Sprite 330ml Price: 2.99\$

*Figura 50, Pagina POS produsele adăugate*



The screenshot shows a summary of items added to the order:

Product name:	Quantity:	Total:
Cappuccino	2	\$5.00
Macchiato	1	\$1.50
Espresso	1	\$1.30

**Total to pay: 7.80\$**

At the bottom, there are two buttons: "Create, pay later" and "Create and pay".

Figura 51, Pagina cu mese



Figura 52, Pagina confirmare comenzilor

### 1 orders waiting for confirmation!

1. Order number #183

- Order date Jun 21, 2024, 7:53:42 PM
- Table number 1
- Employee name: ""
- Status: Pending

Products:

1.1 Macchiato

- Quantity: 1
- Unit price: 1.5 \$
- Total: 1.50 \$

1.2 Cappuccino

- Quantity: 2
- Unit price: 2.5 \$
- Total: 5.00 \$

1.3 Espresso

- Quantity: 1
- Unit price: 1.3 \$
- Total: 1.30 \$

1.4 Irish coffe

- Quantity: 1
- Unit price: 2.3 \$
- Total: 2.30 \$

Figura 53, Pagina comenzi active

# 1 active order/s!

Search by id

**1. Order number #184**

- Date: Jun 21, 2024, 8:41:53 PM
- Table number
- Client name: -
- Created by: "employee1"
- Delivered by : -
- Status: Accepted

---

**Products:**

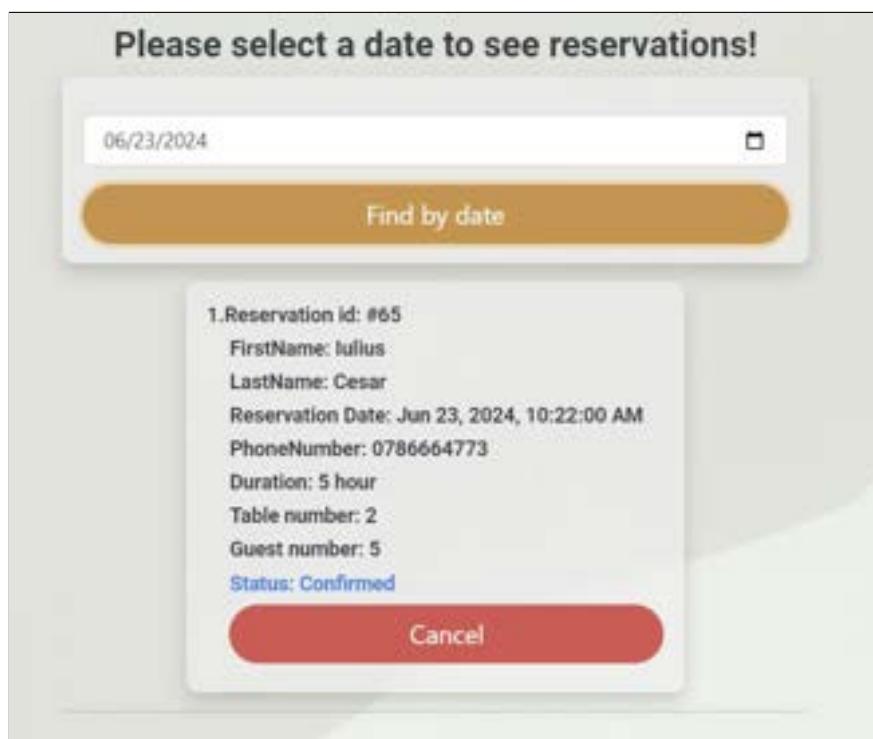
- 1. Macchiato**
  - id: 8
  - Quantity: 1
  - Unit price: \$1.50
  - Tva: 9%
  - Total: \$1.50
- 2. Cappuccino**
  - id: 6
  - Quantity: 1
  - Unit price: \$2.50
  - Tva: 9%
  - Total: \$2.50
- 3. Irish coffe**
  - id: 20
  - Quantity: 2
  - Unit price: \$2.30
  - Tva: 9%
  - Total: \$4.60

**Delivered**

**Finish**

**Add products**

*Figura 54, Pagina cu rezervările viitoare*



*Figura 55, Pagina cu toate rezervările*

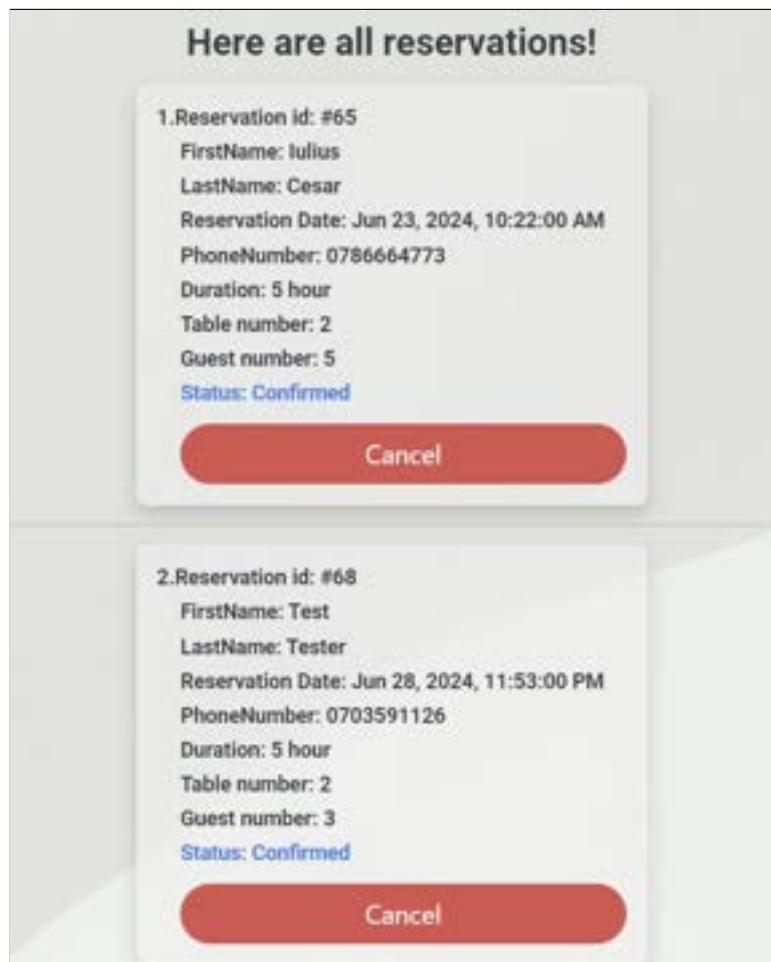


Figura 56, Pagina pentru crearea unei rezervări de către un angajat

The screenshot shows a web-based form titled "Create reservation!". It consists of several input fields: "First name", "Last name", "Phone number", "Select guest number", "Select table number", "Duration", "Select reservation date" (with a date input field showing "2024/06/28"), and "Select start time" (with a time input field showing "09:50 AM"). At the bottom is a large blue "Create reservation" button.

Figura 57, Pagina pentru confirmarea rezervărilor

The screenshot displays a confirmation message: "There are 2 reservations to confirm!". Below this, two separate sections provide details for each reservation:

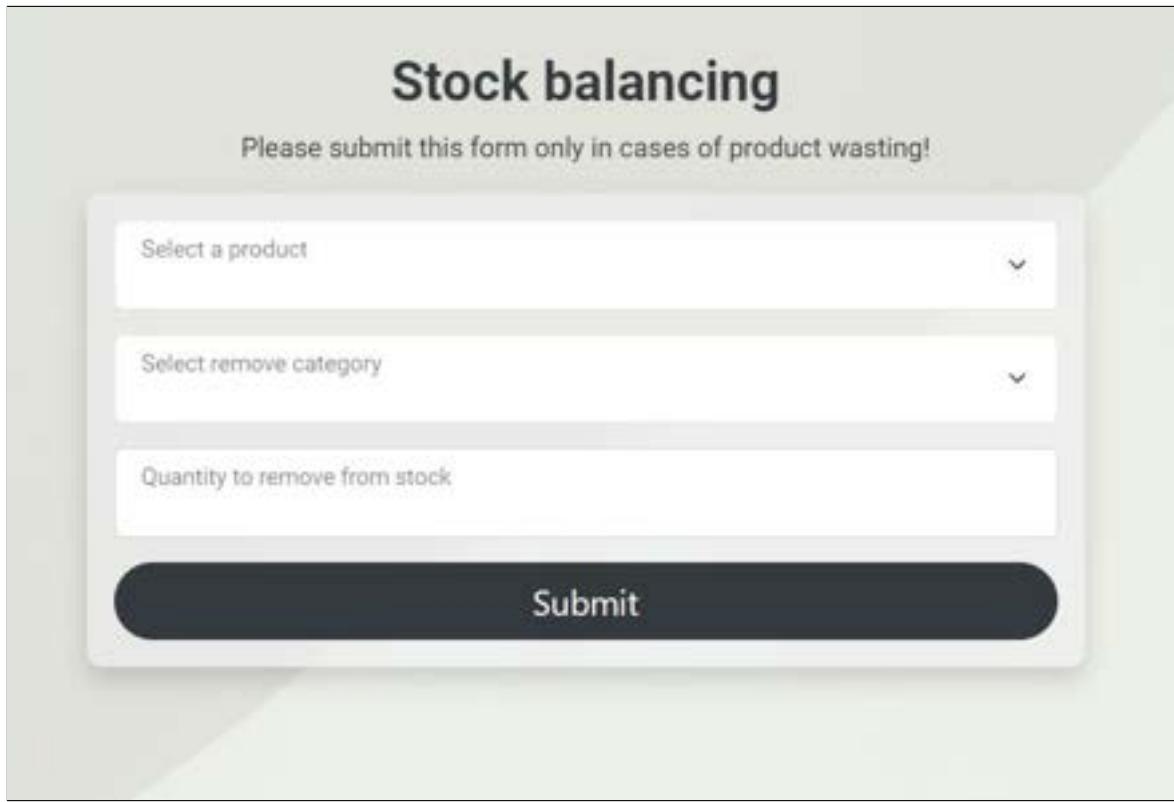
**1.Reservation id: #70**  
FirstName: client  
LastName: tester  
Reservation Date: 28/06/24 09:50 AM  
PhoneNumber: 0785553773  
Duration: 3 hour  
Table number:  
Guest number: 3  
Status: Pending

**2.Reservation id: #71**  
FirstName: client  
LastName: tester  
Reservation Date: 25/06/24 02:50 PM  
PhoneNumber: 0785553773  
Duration: 3 hour  
Table number:  
Guest number: 3  
Status: Pending

Each section contains a blue "Confirm" button and a red "Cancel" button.

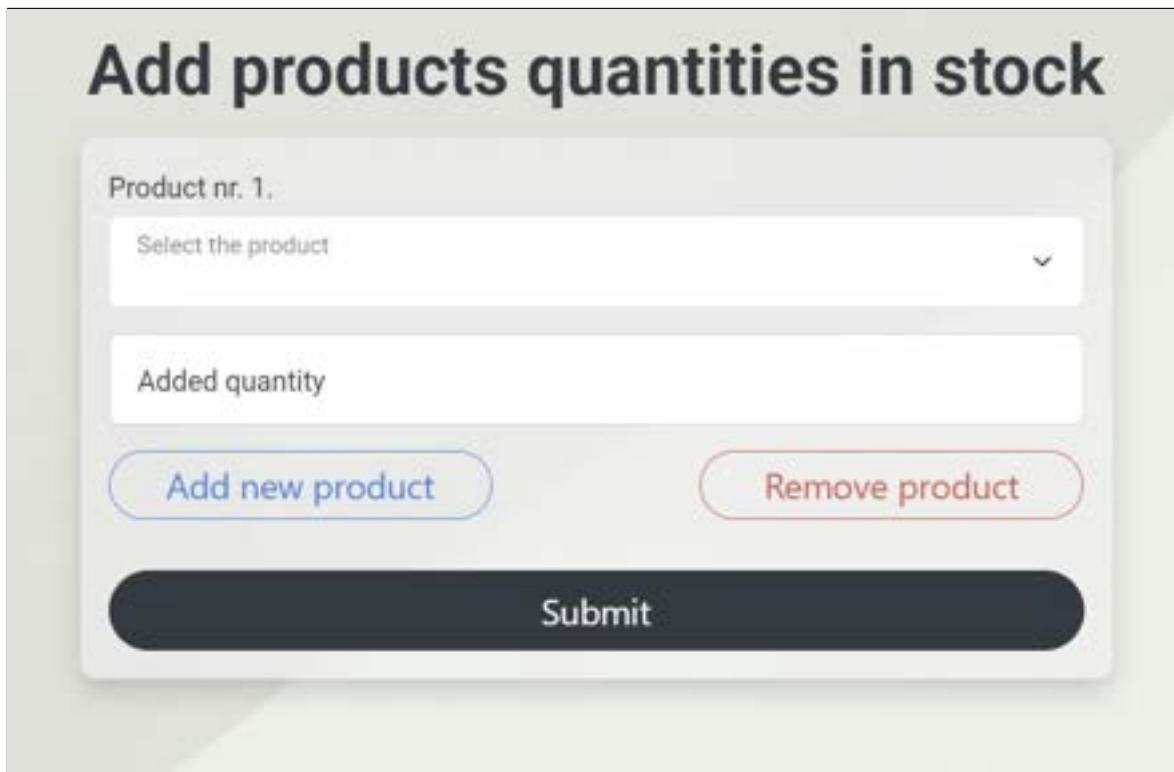
- Utilizatorii de tip manager, au acces la aceleasi interfece ca si utilizatorii de tip angajat, cu diferența de câteva funcționalități la care au acces, plus încă 2 pagini:

*Figura 58, Pagina balansarea de stoc*



The screenshot shows a web form titled "Stock balancing". At the top, a message reads "Please submit this form only in cases of product wasting!". Below this are three input fields: "Select a product", "Select remove category", and "Quantity to remove from stock". A large "Submit" button is positioned at the bottom of the form.

*Figura 59, Pagina aprovizionarea produselor*



The screenshot shows a web form titled "Add products quantities in stock". It starts with a section for "Product nr. 1." which includes a dropdown menu labeled "Select the product". Below this is an input field for "Added quantity". At the bottom, there are two buttons: "Add new product" (in blue) and "Remove product" (in red). A large "Submit" button is located at the very bottom of the form.

- Interfețele pentru utilizatorul de tip administrator

*Figura 60, Pagina statistici*



*Figura 61, Pagina produse*

Stock products												
Id	Product name	Unit Price	unitMeasure	availableForUser	complexProduct	categoryId	quantity	supplyCheck	TVA	Modify		Search:
1	Coca Cola 330ml	\$2.99	piece	true	false	2	99	20	9%	edit		
2	Arabic Coffe	\$0.00	bag	false	false	3	100	2	9%	edit		
3	Milk	\$0.00	pack	false	false	4	106	5	9%	edit		
6	Cappuccino	\$2.50	cup	true	true	5	0	6	9%	edit		
8	Macchiato	\$1.50	cup	true	true	3	0	0	9%	edit		
14	Repar 0.5l	\$1.79	bottle	true	false	2	100	15	9%	edit		
15	Croissant	\$1.99	piece	true	false	5	99	10	9%	edit		
16	Lemonade 0.5l	\$2.49	bottle	true	false	2	98	15	9%	edit		
17	Espresso	\$1.20	cup	true	true	3	0	6	9%	edit		
18	Whipped cream	\$0.00	pack	false	false	4	101	15	9%	edit		

Showing 1 to 10 of 22 entries.

Add NON complex product      Add complex product

Figura 62, Pagina categorii produse

Product categories			
10	entries per page	Search:	
Id	Name	Available menu	Modify
1	Alchool drinks	Yes	
2	Beverages	Yes	
3	Coffee	Yes	
4	Dairy	No	
5	Bakery	Yes	
6	Meat	No	
7	Sweets	Yes	
8	Sandwiches	Yes	

Showing 1 to 8 of 8 entries

[Add a new category](#)

Figura 63, Pagina categorii balansări

Balance categories			
10	entries per page	Search:	
Id	Name	Modify	
1	Automate		
2	Waste		
3	Broke		
4	Droped		

Showing 1 to 4 of 4 entries

[Add a new balance category](#)

Figura 64, Pagina creare cont angajat

### Register new employee

---

First Name

Last Name

Email

Salary

Role:  
Employee

Password

RePassword

**Create new employee**

Figura 65, Pagina angajați

entries per page:									Search:
ID	First name	Last name	Email	Salary	Role	Taken orders	Delivered orders	Edit	Lock
1	employee1	first	first@gmail.com	\$1,500.00	employee	75 orders	43 orders		
2	jhonsonemployee	second	second@gmail.com	\$2,000.23	employee	15 orders	1 orders		
3	Manager	Pos	pos@gmail.com	\$1,500.00	manager	27 orders	28 orders		
4	employee	third	third@gmail.com	\$1,400.00	employee	0 orders	0 orders		
5	managersecond	pos2	pos2@gmail.com	\$2,100.00	manager	0 orders	0 orders		
10	Corneliu	Muresanu	corneliu@gmail.com	\$1,00	admin	0 orders	0 orders		
11	tester	employee	tester.employee@gmail.com	\$1,499.00	employee	0 orders	0 orders		
12	jhone	statham	jhonny1@gmail.com	\$1,999.00	manager	0 orders	0 orders		

Showing 1 to 8 of 8 entries

Figura 66, Pagina clienti

Clients								
Id	First name	Last name	Email	Phone number	Finished orders	Orders value	Lock	
2	radu	cobanica	radu@gmail.com	0766661664	0	\$0.00		
3	sorina	dumitrescu	sorina@gmail.com	0785661664	0	\$0.00		
7	dan	jackson	dan@gmail.com	+40796661773	0	\$0.00		
8	iliana	ionel	ionel@gmail.com	+40796661668	0	\$0.00		
9	iliana	munteanu	iliana@gmail.com	+40756664663	0	\$0.00		
10	adriana	munteanu	adriana@gmail.com	+40786664773	0	\$0.00		
11	john	johnson	johnny@gmail.com	+40743884883	0	\$0.00		
12	mariana	cobanu	cobanu@gmail.com	+4064773882	0	\$0.00		
13	simona	dumitrescu	simonca@gmail.com	+40758884388	0	\$0.00		
14	iliana	cheftea	cheftea@.gmail.com	+40798883774	0	\$0.00		

Showing 1 to 10 of 15 entries

Figura 67, Pagina mese

All tables					
Table Id	Capacity	Table status	Modify	Delete	
1	6	Free			
2	10	Free			
3	4	Free			
4	4	Free			
5	6	Free			
6	6	Free			
7	6	Free			
8	6	Free			
9	4	Free			
10	4	Free			

Showing 1 to 10 of 14 entries

Add a new table

Figura 68, Pagina comenzi

entries per page										Search:
ID	Order date	Status	Client	Taken by	Delivered by	Table	Total	Details		
55	11/06/2024 01:01 PM	Finished	Unknown	second (jhonsonemployee)	Unknown	1	\$8.98			
56	11/06/2024 01:54 PM	Finished	Unknown	second (jhonsonemployee)	Unknown	2	\$6.48			
57	11/06/2024 06:48 PM	Finished	Unknown	first employee1	Unknown	3	\$4.98			
58	11/06/2024 06:51 PM	Finished	Unknown	first employee1	Unknown	Bar	\$1.50			
59	11/06/2024 06:55 PM	Finished	Unknown	first employee1	Unknown	Bar	\$1.50			
60	11/06/2024 07:02 PM	Finished	Unknown	first employee1	Unknown	4	\$2.50			
61	11/06/2024 07:07 PM	Finished	Unknown	first employee1	Unknown	Bar	\$2.50			
62	11/06/2024 07:12 PM	Finished	Unknown	first employee1	Unknown	Bar	\$2.50			
63	11/06/2024 07:30 PM	Finished	Unknown	first employee1	Unknown	Bar	\$2.50			
64	11/06/2024 07:35 PM	Finished	Unknown	first employee1	Unknown	Bar	\$2.50			

Showing 1 to 10 of 509 entries

« ‹ 1 2 3 4 5 … 11 › »

Figura 69, Pagina rezervări

entries per page										Search:
ID	Date	Guest number	First name	Last name	Phone number	Status	Duration	Table number	Delete	
65	23/06/24 10:22 AM	5	Julius	Cesar	0786664773	true	3 hours	2		
66	28/06/24 11:51 PM	3	Test	Tester	0703591126	true	3 hours	2		
70	28/06/24 09:50 AM	3	client	tester	0785553773	false	3 hours	1		
71	28/06/24 02:50 PM	3	client	tester	0785553773	false	3 hours	1		
72	23/06/24 09:00 AM	2	tester	second	0786664416	false	4 hours	1		
73	23/06/24 05:45 PM	6	Cornelia	Museleane	0786667983	true	3 hours	2		

Showing 1 to 6 of 6 entries

« ‹ 1 2 3 › »

Figura 70, Pagina generare rapoarte

**Here you can generate all available reports!**

**Generate current stock report**

Select a category  
All

**Generate report**

**Generate daily POS Closing Report**

Select a day  
mm/dd/yyyy

**Generate report**

**Generate selled products between 2 dates!**

Start date  
mm/dd/yyyy

End date  
mm/dd/yyyy

OrderBy  
Order by selled quantity

**Generate report**

**Generate total earnings between 2 dates!**

Start date  
mm/dd/yyyy

End date  
mm/dd/yyyy

**Generate report**

## 2.8 Alegerea tehnologiei de prelucrare

În urma studiului sistemului informațional și a cerințelor pentru aplicația informatică, s-a decis utilizarea următoarelor tehnologii pentru dezvoltarea aplicații informative:

Pentru realizarea bazei de date sa ales SGBD<sup>1</sup>-ul SQL Server. SQL Server este dezvoltat de compania Microsoft și este folosit pentru stocarea și gestionarea datelor, folosind limbajul SQL<sup>2</sup> pentru manipularea și interogarea de date. Principalul motiv pentru alegerea acestui SGBD este din cauza că pentru realizarea parții de backend<sup>3</sup> se folosește ASP.NET Core WEB API<sup>4</sup>. SQL Server este foarte performant și scalabil, permitând gestionarea volumelor mari de date, deci dacă pe viitor se va dori extinderea aplicației cu noi funcționalități, baza de date nu va fi o problemă. SQL Server nu este gratis pentru utilizarea în producție, în cazul dat se folosește versiunea Express care este gratuită și se utilizează pentru scopurile de dezvoltare.

Aplicația este dezvoltată cu AST.NET Core Web API. Tehnologia .Net este dezvoltată de compania Microsoft, este o tehnologie gratuită care utilizează în mare parte limbajul de programare C#. Limbajul C# este un limbaj de programare OOP<sup>5</sup> și se alfa în top 5 la cele mai utilizate limbi de programare după datele platformei GitHub. În proiect se folosește Entity Framework care ușurează mult interacțiunea din cod cu baza de date. Prin Entity Framework toate interacțiunea cu baza de date pot să fie realizate cu ajutorul unor metode predefinite, minimizând necesitatea de a scrie cod SQL.

Dezvoltarea de frontend<sup>6</sup> a aplicației este realizată în Angular. Angular este framework web, care se folosește pentru crearea aplicațiilor web de tip o singură pagină. Angular folosește limbajul typescript dezvoltat de compania Google. Angular permite crearea aplicațiilor web compatibile pe toate dispozitivele. Această tehnologie a fost aleasă datorită faptului că se bazează pe module și componente, ceea ce înseamnă ca atunci când se trece de la o pagină la alta, nu se încarcă o pagină web nouă, ci doar se încarcă un alt component pe pagina principală care face ca utilizarea aplicației sa fie mult mai plăcută pentru consumatorii acesteia.

---

<sup>1</sup> SGBD – sistem de gestionare a bazelor de date

<sup>2</sup> SQL – limbaj de programare pentru interacțiunea cu baza de date.

<sup>3</sup> Backend – constă în infrastructura aplicației, partea unde se face logica pentru funcționarea tuturor modulelor.

<sup>4</sup> API – Application Programming Interface, reprezintă un set de definiții de sub-programe, protocoale și unelte pentru programarea de aplicații și software.

<sup>5</sup> OOP – object oriented programming, limbaj de programare baza pe clase și obiecte.

<sup>6</sup> Frontend – constă în partea vizuală a aplicației informative, aici se realizează implementarea în aplicație a funcționalităților.

## 2.9 Estimarea necesarului de resurse și a calendarului de realizare

Estimarea necesarului de resurse și a calendarului de realizare este realizat cu ideea în care sunt disponibile toate tehnologiile necesare ca și o echipă completă pentru dezvoltarea aplicației informaticе deși în proiectul dat aplicația informatică este realizată de o singură persoană.

### 2.9.1 Activitățile și necesarul de resurse

#### 1. Studiul sistemului existent

*Tabelul nr. 22, resursele pentru studiul sistemului existent*

Activitate	Personal alocat	Perioada de timp alocata
<b>Analiza fluxului informațional</b>	Analist de sistem	7 zile
<b>Determinarea documentelor necesare</b>	Analist de sistem	4 zile

#### 2. Proiectarea de detaliu

*Tabelul nr. 23, resursele pentru proiectarea de detaliu*

Activitate	Personal alocat	Perioada de timp alocata
<b>Proiectarea intrărilor</b>	Analist de sistem Designer aplicație	7 zile
<b>Proiectarea ieșirilor</b>	Analist de sistem Designer aplicație	7 zile
<b>Proiectarea bazei de date</b>		10 zile
<b>Proiectarea sistemului de codificare</b>	Analist de sistem	2 zile
<b>Proiectarea interfețelor</b>	Designer aplicație	30 zile
<b>Realizarea planului de achiziție software și hardware</b>	Analist de sistem Manager pe proiect	5 zile

### 3. Programarea aplicației

*Tabelul nr. 24, resursele pentru programarea aplicației*

Activitate	Personal alocat	Perioada de timp alocata
<b>Programare backend</b>	Programator 1	45 zile
	Programator 2	
<b>Programare frontend</b>	Programator 3	45 zile
	Programator 1	
<b>Testare module</b>	Tester 1	20 zile
	Tester 2	
<b>Dezvoltarea documentației</b>	Programator 1	9 zile
	Programator 2	
	Programator 3	

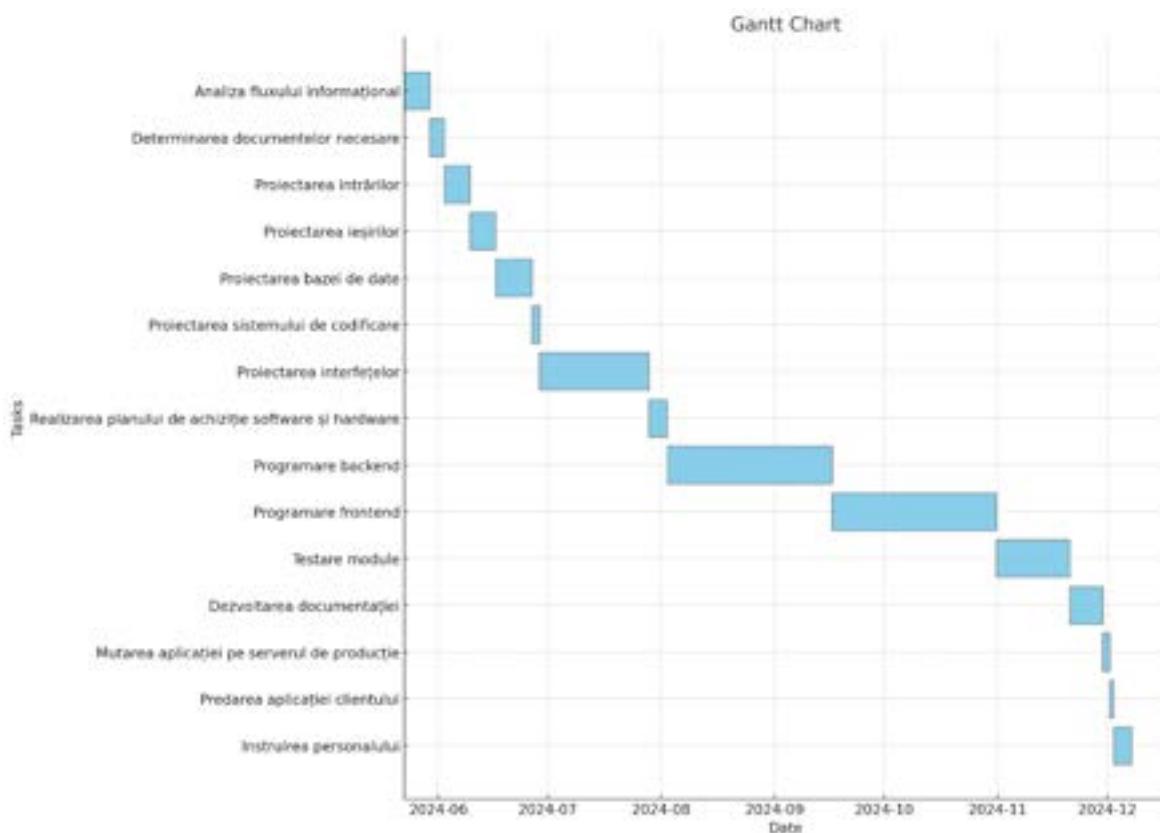
### 4. Implementarea aplicației în sistem

*Tabelul nr. 25, resursele pentru implementarea aplicației în sistem*

Activitate	Personal alocat	Perioada de timp alocata
<b>Mutarea aplicației pe serverul de producție</b>	Inginer baze de date	2 zile
	Programator 1	
<b>Predarea aplicației clientului</b>	Manager pe proiect	1 zile
	Analist de sistem	
<b>Instruirea personalului</b>	Tester 1	5 zile
	Programator 1	

## 2.9.2. Diagrama Gantt

*Figura 71, Diagrama Gantt*



În figura 71 sunt reprezentate toate activitățile pentru finalizarea proiectului și durata de timp pentru îndeplinirea fiecărei activități. Durata totală de realizarea a aplicației informaticice este de 200 de zile.

## Cap. 3. Prezentarea produsului software

### 3.1 Cerințele platformei hardware și software ale produsului

Aplicația informatică va fi de tip web, pentru a putea lansa aplicația în mediul public vom avea nevoie de un server pentru hosting în care vom ține atât API-ul cât și aplicația Angular.

Decizia de a rula ambele proiecte pe același server a fost luată din motivul că aplicația este dezvoltată pentru un singur client, clientul fiind o cafenea nu va avea un trafic foarte mare de utilizatori, deoarece aplicația va fi folosita constant doar de personal, iar clienții vor folosi doar când sunt prezenți în localul dat. Un alt motiv pentru folosirea unui singur server pentru hosting este o latență mai mică pentru afișarea informației necesare în interfața aplicației, deoarece toate cererile din aplicația angular sunt trimise către API pe același server. Totuși decizia de a dezvolta aplicația cu o separare între backend și frontend a fost luată pentru cazul în care numărul utilizatorilor va crește se va putea rula aplicația pe servere de hosting separate pentru backend și frontend ceea ce va crește scalabilitatea aplicației, acest lucru fiind limitat pentru un singur server pentru hosting.

Pentru o performanță optimă a funcționalității aplicației cerințele hardware pentru server-ul de hosting vor fi:

- 8 procesoare virtuale
- 16 GB ram
- 200 GB SSD pentru stocarea datelor necesare
- Conexiune la internet, preferat de 10Gbps

Cerințele software pentru rularea aplicației informaticice sunt următoarele:

- Un sistem de operare instalat pe server, linux server
- Vom avea nevoie de Nginx<sup>7</sup> pentru a putea rula aplicația Angular
- .Net Runtime pentru partea de backend
- SQL Server instalat pe server, pentru baza de date

---

<sup>7</sup> Nginx – un server web dezvoltat pentru o performanță maximă și stabilă pentru gestionarea unui număr mare de conexiuni simultane.

### 3.2 Descrierea funcțiunilor aplicației

Aplicația informatică este împărțita în 4 parti, acestea fiind funcționalitățile pentru utilizatorii de tip client, funcționalitățile pentru utilizatorii de tip angajat, funcționalitățile pentru utilizatorii de tip manager și funcționalitățile pentru administrator. Pe lângă acestea mai avem funcționalitățile pentru autentificarea și autorizarea utilizatorilor și funcționalitățile pentru interacțiunea cu baza de date.

Pentru partea de autentificare și autorizarea a utilizatorilor se folosește Identity din ASP.NET Core. Identity este un sistem de gestionarea a funcționalităților de autentificare și autorizare din cadrul unei aplicații. La implementarea acestui sistem în aplicația se generează o baza de date în SQL Server prin metoda Code-First<sup>8</sup> care folosește migrarea claselor din aplicație în tabele relaționate în baza de date. Prin acest sistem putem gestiona datele utilizatorilor, să setăm rolurile acestora, implementarea sistemul de confirmare prin email, autentificarea în aplicația prin metode externe cum ar fi prin google, facebook, microsoft și multe altele. Baza de date generata de către Identity a fost separată de baza de date a aplicației din motive tehnice pentru a evita conflictele posibile.

Pentru autorizarea utilizatorilor se folosește jwt token care reprezintă un cod unic pentru fiecare utilizator din care se extrag informațiile necesare pentru realizarea autorizării, cum ar fi rolul utilizatorului. Acest cod unic este generat la logarea utilizatorului în sistem. Acest token este necesar pentru accesul la anumite funcționalități a aplicației, fără el utilizatorii nu vor avea acces la funcționalitățile care necesită autorizare.

**Principalele funcționalități pentru utilizatorii cu rolul de client sunt următoarele:**

- Creare unei comenzi noi

Pentru crearea unei comenzi în aplicația informatică, clientul va trebui să efectueze autentificarea în aplicație, după merge la pagina meniu reprezentată în figura 41, acolo vor fi afișate toate categoriile de produse disponibile spre vânzare. Clientul apasă pe categorie dorită și va apărea pagina de produse din categoria dată reprezentată în figura 42. Acolo pentru fiecare produs există butonul de adăugare în coș, după ce clientul a adăugat toate produsele dorite, merge pe pagina cu coșul de cumpărături reprezentată în figura 46, unde va găsi toate produsele adăugate, prețul

---

<sup>8</sup> Code-First – este o metodă de creare și modificare a unei baze de date prin care mai întâi se scriu clasele în aplicație și după aceste clase sunt migrate în baza de date sub forma de tabele relaționate.

---

total și butonul de creare comandă. La apăsarea acestui buton va fi redirecționat către pagina în care va putea scana codul qr de pe masă pentru a putea crea comanda.

După ce comanda a fost creată, poate fi vizualizată în pagina comanda activă reprezentată în figura 47, acolo va avea toate datele comenzi și starea în care se află aceasta. Clientul are posibilitatea de a mai adăuga produse la comanda respectivă repetând pașii de creare a comenzi. Pentru efectuarea plății clientul va merge la casa de marcat sau va efectua plata direct la ospătarul care la servit.

- **Crearea unei rezervări**

Clientul are acces și la funcționalitatea de creare a unei rezervări, formularul pentru crearea unei rezervări este disponibil pe pagina de creare a rezervărilor reprezentată în figura 43. După ce clientul a creat rezervarea, un angajat trebuie să o confirme. Clientului îi mai sunt disponibile două pagini, una pentru vizualizarea rezervărilor viitoare reprezentată în figura 44 și pagina cu istoricul rezervărilor reprezentată în figura 45.

#### **Funcționalitățile principale pentru utilizatorii cu rolul de angajat:**

- **Modulul POS pentru casa de marcat.**

Modulul POS este o interfață grafică reprezentată în figura 49 și figura 50, aici angajații vor putea crea comenzi, prin selectarea produselor necesare, odată selectate produsele vor fi afișate și angajatul va putea crea comanda cu posibilitatea de a efectua plata direct pentru aceasta sau de a o crea și de a efectua plata mai târziu.

- **Confirmarea comenziilor plasate de către clienți.**

Fiecare angajat are acces la pagina de confirmare a comenziilor plasate de către clienți reprezentată în figura 52, unde au posibilitatea de a accepta comanda trimisă de către un client.

- **Confirmarea rezervărilor și crearea rezervărilor.**

Angajații au acces la funcționalitățile pentru crearea unei rezervări, reprezentată în figura 56, unde pot crea rezervările pentru clienții care nu au conturi sau doresc crearea unei rezervări prin telefon. și pentru confirmarea rezervărilor create de către clienți direct pe site, pot merge pe pagina pentru confirmarea rezervărilor reprezentată în figura 57 unde pot accepta sau nu rezervările clienților. Angajații au acces încă la două pagini legate de

rezervări, una pentru a vizualiza rezervările viitoare reprezentată în figura 54 și pagina pentru a vizualiza toate rezervările din sistem reprezentată în figura 55.

### **Funcționalitățile pentru utilizatorii cu rolul de manager:**

Utilizatorii cu rolul de manager au acces la toate funcțiile la care au acces și utilizatorii cu rolul de angajat, plus încă câteva funcționalități adiționale.

- Funcționalitatea balansării de stoc.

Într-un local de tip cafenea, deseori sunt cazuri de risipă a unor produse. De exemplu spargerea unei sticle de alcool sau accidente cu comanda unui client până ca aceasta să fie livrata, din aceasta cauza s-a introdus aceasta funcționalitate pentru a balansa stocul în aceste cazuri, reprezentat în figura 58 unde managerul selectează produsul, categoria balansării și cantitatea care va fi scăzută din stoc.

- Funcționalitatea pentru aprovisionarea produselor.

Când se va face aprovisionarea produselor, managerul sau administratorul vor putea selecta produsele aprovisionate și vor putea introduce cantitatea aprovisionată în stoc. Acest lucru se face efectua pe pagina de aprovisionare, reprezentată în figura 59.

Utilizatorii cu rolul de manager vor putea șterge produse dintr-o comandă, sau să anuleze comenzi, acest lucru nu este permis utilizatorilor cu rolul de angajat din motive de securitate. De exemplu de a nu permite cazuri în care după ce a fost creată o comandă, un angajat să eliminate produse din acea comandă chiar dacă ele au fost livrate către masă, să ceară prețul întreg pe comandă și diferența să nu fie înregistrată în sistem.

### **Funcționalitățile principale pentru administrator:**

- Adăugarea și modificarea datelor din baza de date.

Administratorul va avea acces la funcțiile de modificare, adăugare sau ștergere după caz a înregistrărilor din baza de date. Toate aceste interfețe prin care se vor efectua aceste operațiuni sunt reprezentate în figurile 61-69.

- Generarea rapoartelor.

Administratorul are acces la interfața prin care poate genera toate raporturile disponibile în sistem, interfața fiind reprezentată în figura 70.

- Vizualizarea grafurilor statistice.

Pentru vizualizarea datelor statistice, cum ar fi vânzările lunare sau numărul de comenzi preluate și livrate de către angajați, administratorul poate accesa pagina dată reprezentată în figura 60.

- Blocarea accesului utilizatorilor la aplicație.

Blocarea accesului utilizatorilor în sistem se poate efectua în panoul administrativ de către administrator pentru atât pentru utilizatorii cu rolul de client cât și pentru utilizatorii cu rolul de angajat sau manager.

Deși administratorul are acces la modificarea și adăugare datelor în baza de date, pentru a menține integritatea datelor nu are accese la eliminarea datelor din baza de date, cu excepția unor cazuri care permit eliminarea datelor fără a provoca probleme de integritate la nivelul datelor din baza de date.

## Cap. 4. Eficiența și utilitatea aplicației informaticе

### 4.1 Condiții privind implementarea aplicației

Aplicația informatică fiind de tip web, pentru implementarea acesteia vom avea nevoie de un server, pentru migrarea aplicație din stadiul de dezvoltare în stadiul de producție. Cerințele hardware pentru această migrare sunt descrise în capitolul 3.1. Migrarea aplicației va dura 7 zile lucrătoare, în care se va efectua migrarea propriu zisă, predarea aplicației către client și instruirea personalului pentru utilizarea corectă a acesteia. Numărul de zile pentru fiecare pas poate fi vizualizat în tabelul 25. După ce aplicația a fost migrată cu succes pe serverul de producție, se va crea contul administratorului, iar apoi administratorul va putea crea conturile pentru fiecare angajat în parte și va adăuga toate datele necesare pentru a putea utiliza aplicația, cum ar fi produsele, categoriile, mesele etc.

### 4.2 Exploatarea curentă a aplicației

Aplicația este exploatată cu succes de către angajații localului cât și de către clienții acestuia. Mulțumită interfeței minimalistă a aplicației, utilizatorilor le a fost foarte ușor de a învățat cum să folosească aplicația. După implementarea aplicației gestionarea comenzielor și a rezervărilor a devenit mult mai eficientă. Iar mulțumită gestionării stocurilor aprovisionarea produsele se efectuează la nevoia acestuia ceea ce permite ca stocul produselor mereu să îndeplinească nevoile clienților.

### 4.3 Considerații privind eficiența aplicației informaticе

După utilizarea aplicației pentru o perioadă se poate observa că aplicația rulează toate procesele la o viteză bună și stabilă. Ea este utilizată cu succes de către firma SDBar S.R.L și a beneficiat la optimizarea procesului de muncă din cadrul firmei.

Deși aplicația este deja utilizabilă, nu putem spune că această se află într-o stare finalizată îndeplinind toate funcționalitățile adăugătoare care ar face procesul de muncă și mai eficient.

Pentru viitor se vor implementa funcționalități adăugătoare pentru gestionarea comenzielor, cum ar fi posibilitatea adăugării unor preferințe pentru fiecare produs în parte. Implementarea unui sistem de fidelizare al clienților, de exemplu clienții cu un număr mare de comenzi finalizate să beneficieze de reduceri personale sau de produse gratuită la efectuarea unei comenzi. Fidelizarea clienților prin coduri de invitație și în funcție de câte

persoane noi sau înregistrat în sistemul informatic cu codul dat acesta va beneficia de promoții adiționale. Implementarea unor rapoarte pentru gestionarea mai eficientă al localului. Posibilitatea achitării comenzi online de către clienți la crearea unei comenzi în sistemul informatic. Deci aplicație în prezent este funcțională dar mai sunt posibilități de îmbunătățire a acesteia.

## 5.Bibliografie

### Cărți:

- Aristeidis , B., & Pablo, D. (2023). *Learning Angular: A no-nonsense guide to building web applications with Angular 15*. Packt Publishing.
- Botezatu, M., Botezatu, C., & Căruțășu, G. (2016). *Managementul proiectelor informaticice*. Editura Pro Universitaria.
- Crișan, D. A. (2015). *Programarea aplicațiilor folosind limbajul C# și platforma .NET*. Editura Pro Universitaria.
- Ionel, I., & Botezatu, C. (2007). *Proiectarea sistemelor informaticice*. București: Pro Universitaria.
- Jamie, K., & Brian, W. (2014). *ASP.NET Web API 2: Building a REST Service from Start to Finish*. Apress.
- Price, M. J. (2023). *C# 12 and .NET 8 - Modern Cross-Platform Development Fundamentals*. Packt Publishing.
- Schuster, S. a. (2023). *Building Web APIs with ASP.NET Core*.

### Documentație:

- ASP.NET Core web API documentation*. (2024, 04 25). Preluat de pe Microsoft: <https://learn.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-8.0>
- Entity Framework documentation hub*. (2024). Preluat de pe Microsoft.com: <https://learn.microsoft.com/en-us/ef/>
- Introduction to the Angular docs*. (2023). Preluat de pe Angular: <https://angular.io/docs>
- Microsoft identity platform documentation*. (fără an). Preluat de pe Microsoft.com: <https://learn.microsoft.com/en-us/entra/identity-platform/>
- SQL Server technical documentation*. (fără an). Preluat de pe Microsoft.com: <https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16>
- npm Docs*. (fără an). Preluat de pe npmjs.com: <https://docs.npmjs.com/>

## 6. Anexe

### 6.1. Anexele cu codul din API

*Anexa 1, Model de migrare pentru 2 tabele din baza de date*

```
modelBuilder.UseCollation("SQL_Latin1_General_CI_AS");

modelBuilder.Entity<Category>(entity =>
{
    entity.HasKey(e => e.CategoryId).HasName("PK__Category__D5B1EDCC84AF411B");

    entity.ToTable("Category");

    entity.Property(e => e.CategoryId).HasColumnName("category_Id");
    entity.Property(e => e.AvailableMenu)
        .HasDefaultValue(false)
        .HasColumnName("available_menu");
    entity.Property(e => e.CategoryName)
        .HasMaxLength(50)
        .IsUnicode(false)
        .HasColumnName("category_Name");
});

modelBuilder.Entity<Client>(entity =>
{
    entity.HasKey(e => e.ClientId).HasName("PK__Clients__BF554B6CA16960B9");

    entity.Property(e => e.ClientId).HasColumnName("client_Id");
    entity.Property(e => e.Email)
        .HasMaxLength(100)
        .IsUnicode(false)
        .HasColumnName("email");
    entity.Property(e => e.FirstName)
        .HasMaxLength(50)
        .IsUnicode(false)
        .HasColumnName("firstName");
    entity.Property(e => e.LastName)
        .HasMaxLength(50)
        .IsUnicode(false)
        .HasColumnName("lastName");
    entity.Property(e => e.Lock).HasColumnName("lock");
    entity.Property(e => e.PhoneNumber)
        .HasMaxLength(15)
        .IsUnicode(false)
        .HasColumnName("phone_number");
    entity.Property(e => e.UserId)
        .HasMaxLength(100)
        .IsUnicode(false)
        .HasColumnName("userId");
});
```

## Anexa 2, Autentificarea și autorizarea utilizatorilor

```
[Authorize]
[HttpGet("refresh-user-token")]
public async Task<ActionResult<UserDto>> RefreshUserToken()
{
    var user = await _userManager.FindByNameAsync(User.FindFirst(ClaimTypes.Email).Value);
    if (user.LockoutEnabled == false)
    {
        return Unauthorized(new JsonResult(new { message = "Your account has been blocked!" }));
    }
    return await CreateApplicationUserDto(user);
}

[HttpPost("login")]
public async Task<ActionResult<UserDto>> Login(LoginDto model)
{
    var user = await _userManager.FindByNameAsync(model.Email);
    if (user == null)
    {
        return Unauthorized(new JsonResult(new { title = "Error", message = "Invalid email or password!" }));
    }
    if (user.EmailConfirmed == false) return Unauthorized("Please confirm your email!");
    if (user.LockoutEnabled == false) return Unauthorized(new JsonResult(new { title = "Account blocked!", message = "Your account has been blocked!" }));

    var result = await _signInManager.CheckPasswordSignInAsync(user, model.Password, false);
    if (!result.Succeeded) return Unauthorized(new JsonResult(new { title = "Error", message = "Invalid email or password!" }));

    return await CreateApplicationUserDto(user);
}

[HttpPost("register")]
public async Task<IActionResult> Register(RegisterDto model)
{
    if (!await CheckEmailExistAsync(model.Email))
    {
        return BadRequest(new JsonResult(new { title = "Email registered", message = "This email is already used by another user!" }));
    }

    var userToAdd = new User
    {
        FirstName = model.FirstName.ToLower(),
        LastName = model.LastName.ToLower(),
        UserName = model.Email.ToLower(),
        Email = model.Email.ToLower(),
        EmailConfirmed = true,
    };

    var result = await _userManager.CreateAsync(userToAdd, model.Password);
    var _roleAssigned = await _userManager.AddToRoleAsync(userToAdd, Dependencies.DEFAULT_ROLE);
    if (!result.Succeeded) return BadRequest(result.Errors);

    var userToAddInClient = new Client
    {
        FirstName = model.FirstName.ToLower(),
        LastName = model.LastName.ToLower(),
        Email = model.Email.ToLower(),
        PhoneNumber = model.PhoneNumber.ToLower(),
        UserId = userToAdd.Id,
        Lock = false,
    };

    _applicationContext.Clients.Add(userToAddInClient);
    await _applicationContext.SaveChangesAsync();

    return Ok(new JsonResult(new { title = "Account created", message = "Your account was created!" }));
}
```

Anexa 3, Funcția pentru creare a token-ului JWT

```
public async Task<string> CreateJWT(User user)
{
    // Get user roles from ASP.NET Identity
    var userRoles = await _userManager.GetRolesAsync(user);

    // Create claims
    var claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, user.Id),
        new Claim(ClaimTypes.Email, user.Email),
        new Claim(ClaimTypes.GivenName, user.FirstName),
        new Claim(ClaimTypes.Surname, user.LastName)
    };

    // Add role claims securely
    foreach (var role in userRoles)
    {
        claims.Add(new Claim(ClaimTypes.Role, role));
    }

    var credentials = new SigningCredentials(
        new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["JWT:Key"])),
        SecurityAlgorithms.HmacSha256Signature);

    var tokenDescriptor = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(claims),
        Expires = DateTime.Now.AddDays(int.Parse(_config["JWT:ExpiresInDays"])),
        SigningCredentials = credentials,
        Issuer = _config["JWT:Issuer"]
    };

    var tokenHandler = new JwtSecurityTokenHandler();
    var jwt = tokenHandler.CreateToken(tokenDescriptor);
    return tokenHandler.WriteToken(jwt);
}
```

*Anexa 4, Verificarea statusului casei de marcat*

```

● ● ●

namespace CoffeBarManagement.Policy
{
    public class CheckOpenStatus : AuthorizationRequirement
    {
        public bool Status { get; set; }

        public CheckOpenStatus(bool status)
        {
            Status = status;
        }
    }
}

namespace CoffeBarManagement.Policy
{
    public class CheckOpenStatusHandler : AuthorizationHandler<CheckOpenStatus>
    {
        private readonly ApplicationContext _applicationContext;

        public CheckOpenStatusHandler(ApplicationContext applicationContext, IHttpContextAccessor httpContextAccessor)
        {
            applicationContext = applicationContext;
        }

        protected override async Task HandleRequirementAsync(AuthorizationHandlerContext context, CheckOpenStatus requirement)
        {
            var status = await _applicationContext.Organizations.FindAsync();
            if (status == null || status.OpenStatus != requirement.Status)
            {
                context.Fail();
                return;
            }
            context.Succeed(requirement);
        }
    }
}

```

*Anexa 5, Conexiunea la baza de date și serviciul de autorizare*

```

● ● ●

builder.Services.AddDbContext<Context>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));
});
builder.Services.AddDbContext<ApplicationContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("ApplicationConnection"));
});

builder.Services.AddScoped<ITfService>(); // for injecting our services inside our controllers

// defining our identity service
builder.Services.AddIdentityCore<User>(options =>
{
    options.SignIn.RequireConfirmedEmail = true;
})
    .AddRoles<IdentityRole>()
    .AddRoleManager<RoleManager<IdentityRole>>()
    .AddEntityFrameworkStores<Context>()
    .AddSignInManager<SignInManager<User>>()
    .AddUserManager<UserManager<User>>(); // for creating users

// for adding authentication
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters
    {
        // validating token based on our key
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JWT:Key"])),
        ValidIssuer = builder.Configuration["JWT:Issuer"],
        ValidateIssuer = true,
        ValidateAudience = false
    };
});

builder.Services.AddAuthorization(options => {
    options.AddPolicy("CheckOpenStatus", policy =>
        policy.Requirements.Add(new CheckOpenStatus(true)));
});

builder.Services.AddScoped<AuthorizationHandler, CheckOpenStatusHandler>();

```

### Anexa 6, Creare unui utilizator de tip administrator în cazul în care acesta nu există

```

● ● ●

using (var scope = app.Services.CreateScope())
{
    var context = scope.ServiceProvider.GetRequiredService<ApplicationContext>();
    var userManager = scope.ServiceProvider.GetRequiredService<UserManager<User>>();
    var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<IdentityRole>>();

    var adminCreating = new User
    {
        FirstName = "Corneliu".ToLower(),
        LastName = "Museteanu".ToLower(),
        UserName = "corneliu@gmail.com".ToLower(),
        Email = "corneliu@gmail.com".ToLower(),
        EmailConfirmed = true
    };

    var checkIfExist = await userManager.FindByEmailAsync(adminCreating.Email);
    if(checkIfExist == null)
    {
        await userManager.CreateAsync(adminCreating, "P@ssword1!");
        await userManager.AddToRoleAsync(adminCreating, Dependencis.ADMIN_ROLE);
    }

    var checkAnotherDb = await context.Employees.Where(q => q.UserId == checkIfExist.Id).FirstOrDefaultAsync();
    if(checkAnotherDb == null)
    {
        var admin = new Employee
        {
            FirstName = "Corneliu",
            LastName = "Museteanu",
            UserId = checkIfExist.Id,
            Email = checkIfExist.Email,
            Role = "admin",
            Lock = false,
            Salary = 1,
        };
        await context.AddAsync(admin);
        await context.SaveChangesAsync();
    }
}

```

### Anexa 7, Crearea unei rezervări de către client

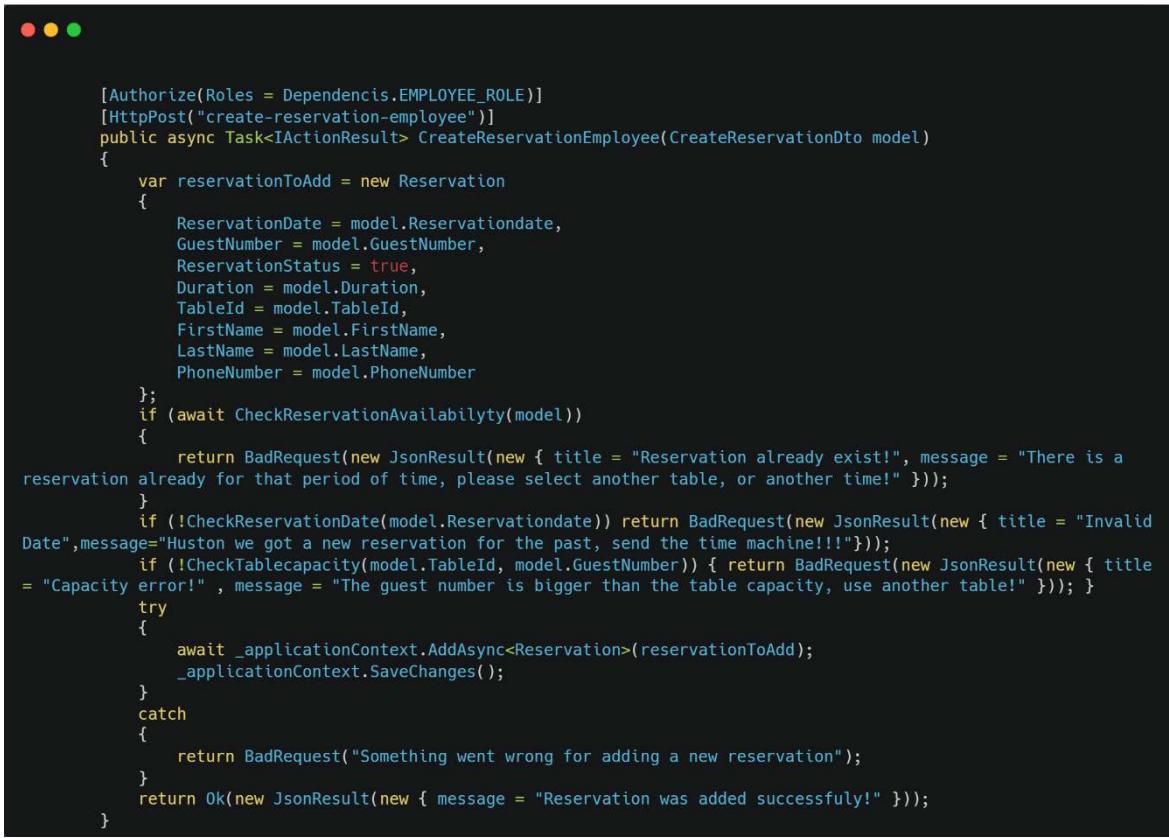
```

● ● ●

[Authorize(Roles = Dependencis.DEFAULT_ROLE)]
[HttpPost("create-reservation-client/{id}")]
public async Task<ActionResult> CreateReservation(CreateReservationDto model, int id)
{
    var clientDetails = await _applicationContext.Clients.FindAsync(id);
    if (clientDetails == null) return NotFound();
    var reservationToAdd = new Reservation
    {
        ReservationDate = model.Reservationdate,
        GuestNumber = model.GuestNumber,
        ReservationStatus = model.ReservationStatus,
        Duration = model.Duration,
        ClientId = clientDetails.ClientId,
        TableId = model.TableId,
        FirstName = clientDetails.FirstName,
        LastName = clientDetails.LastName,
        PhoneNumber = clientDetails.PhoneNumber
    };
    if (model.Reservationdate.Hour > 20 || model.Reservationdate.Hour < 6) return BadRequest(new JsonResult(new { title = "Reservation Error", message = "You can make a reservation between 06:00 - 20:00" }));
    if(await CheckReservationAvailability(model))
        return BadRequest(new JsonResult(new { title = "Reservation already exist!", message = "There is a reservation already for that period of time, please select another table, or another time!" }));
    var checkTableExist = await _applicationContext.Tables.FindAsync(model.TableId);
    if(checkTableExist == null) return NotFound(new JsonResult(new { title = "Inexisting table", message = "Such a table does not exist!" }));
    if (!CheckReservationDate(model.Reservationdate)) return BadRequest(new JsonResult(new { title = "Date error", message = "Mustn't we got a reservation date that is in the past send Time Machine :)" }));
    if (!CheckTableCapacity(model.TableId, model.GuestNumber)) { return BadRequest(new JsonResult(new { title = "Guest Number to big!", message = "The guest number is bigger than the table capacity, use another table!" })); }
    await _applicationContext.AddAsync<Reservation>(reservationToAdd);
    _applicationContext.SaveChanges();
    return Ok(new JsonResult(new { title = "Reservation created!", message = "You can check your reservations in AllReservations!" }));
}

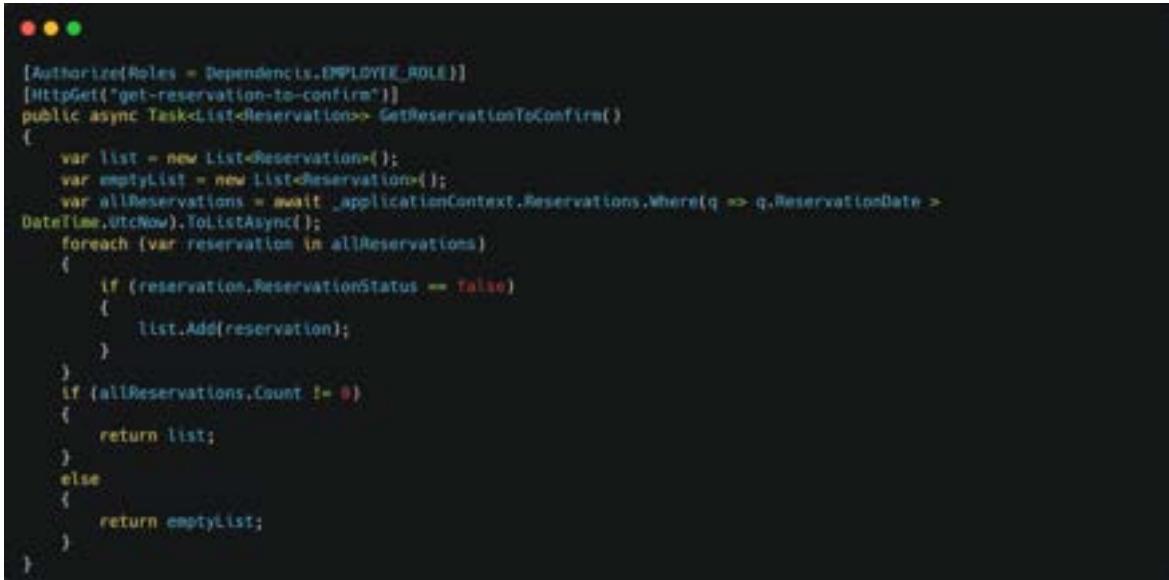
```

### Anexa 8, Crearea unei rezervări de către un angajat



```
[Authorize(Roles = Dependencis.EMPLOYEE_ROLE)]
[HttpPost("create-reservation-employee")]
public async Task<IActionResult> CreateReservationEmployee(CreateReservationDto model)
{
    var reservationToAdd = new Reservation
    {
        ReservationDate = model.Reservationdate,
        GuestNumber = model.GuestNumber,
        ReservationStatus = true,
        Duration = model.Duration,
        TableId = model.TableId,
        FirstName = model.FirstName,
        LastName = model.LastName,
        PhoneNumber = model.PhoneNumber
    };
    if (await CheckReservationAvailabilty(model))
    {
        return BadRequest(new JsonResult(new { title = "Reservation already exist!", message = "There is a reservation already for that period of time, please select another table, or another time!" }));
    }
    if (!CheckReservationDate(model.Reservationdate)) return BadRequest(new JsonResult(new { title = "Invalid Date", message = "Huston we got a new reservation for the past, send the time machine!!!" }));
    if (!CheckTablecapacity(model.TableId, model.GuestNumber)) { return BadRequest(new JsonResult(new { title = "Capacity error!", message = "The guest number is bigger than the table capacity, use another table!" })); }
    try
    {
        await _applicationContext.AddAsync<Reservation>(reservationToAdd);
        _applicationContext.SaveChanges();
    }
    catch
    {
        return BadRequest("Something went wrong for adding a new reservation");
    }
    return Ok(new JsonResult(new { message = "Reservation was added successfully!" }));
}
```

### Anexa 9, Returnarea unei liste cu toate rezervările ce necesită confirmare



```
[Authorize(Roles = Dependencis.EMPLOYEE_ROLE)]
[HttpGet("get-reservation-to-confirm")]
public async Task<List<Reservation>> GetReservationToConfirm()
{
    var list = new List<Reservation>();
    var emptyList = new List<Reservation>();
    var allReservations = await _applicationContext.Reservations.Where(q => q.ReservationDate > DateTime.UtcNow).ToListAsync();
    foreach (var reservation in allReservations)
    {
        if (reservation.ReservationStatus == false)
        {
            list.Add(reservation);
        }
    }
    if (allReservations.Count != 0)
    {
        return list;
    }
    else
    {
        return emptyList;
    }
}
```

Anexa 10, Returnarea datelor pentru rezervările confirmate

```
[Authorize(Roles = Dependencias.EMPLOYEE_ROLE)]
[HttpGet("get-confirmed-reservations")]
public async Task<List<GetReservationDto>> GetConfirmedReservations()
{
    var list = new List<GetReservationDto>();
    var emptyList = new List<GetReservationDto>();
    var allReservations = await _applicationContext.Reservations.ToListAsync();
    foreach (var reservation in allReservations)
    {
        if (reservation.ReservationStatus == true)
        {
            var reservationDto = new GetReservationDto
            {
                ReservationId = reservation.ReservationId,
                Reservationdate = reservation.ReservationDate,
                GuestNumber = reservation.GuestNumber,
                FirstName = reservation.FirstName,
                LastName = reservation.LastName,
                PhoneNumber = reservation.PhoneNumber,
                ReservationStatus = true,
                Duration = reservation.Duration,
                TableNumber = reservation.TableId,
            };
            list.Add(reservationDto);
        }
    }
    if (allReservations.Count != 0)
    {
        return list;
    }
    else
    {
        return emptyList;
    }
}
```

Anexa 11, Confirmarea unei rezervări după id

```
[Authorize(Roles = Dependencis.EMPLOYEE_ROLE)]
[HttpPut("confirm-reservation/{id}")]
public async Task<IActionResult> ConfirmReservation(int id)
{
    var result = await _applicationContext.FindAsync<Reservation>(id);
    if(result != null)
    {
        result.ReservationStatus = true;
        _applicationContext.Update<Reservation>(result);
        await _applicationContext.SaveChangesAsync();
    }
    else
    {
        return BadRequest("Such a reservation does not exist!");
    }
    return Ok(new JsonResult(new {message = $"Reservation number {id} was confirmed!"}));
}
```

Anexa 12, Returnarea datelor rezervărilor pentru fiecare client

```
[Authorize(Roles = Dependencis.DEFAULT_ROLE)]
[HttpGet("reservations-client/{id}")]
public async Task<List<GetReservationDto>> GetConfirmedReservations(int id)
{
    var clientReservations = await _applicationContext.Reservations.Where(q => q.ClientId == id).ToListAsync();
    if(clientReservations == null) { return new List<GetReservationDto>(); }
    var reservationList = new List<GetReservationDto>();
    foreach (var reservation in clientReservations)
    {
        var reservationDto = new GetReservationDto
        {
            ReservationId = reservation.ReservationId,
            Reservationdate = reservation.ReservationDate,
            GuestNumber = reservation.GuestNumber,
            FirstName = reservation.FirstName,
            LastName = reservation.LastName,
            PhoneNumber = reservation.PhoneNumber,
            ReservationStatus = reservation.ReservationStatus,
            Duration = reservation.Duration,
            TableNumber = reservation.TableId
        };
        reservationList.Add(reservationDto);
    }
    return reservationList;
}
```

### Anexa 13, Anularea / ștergerea unei rezervări de către un client

```
[Authorize(Roles = Dependencies.DEFAULT_ROLE)]
[HttpDelete("delete-reservation/{userId}/{reservationId}")]
public async Task<IActionResult> DeleteReservation(int userId, int reservationId)
{
    var result = await _applicationContext.Reservations.FindAsync(reservationId);
    if(result == null) { return BadRequest(new JsonResult(new { message = "Such a reservation does not exist!" })); }
    var checkDate = CheckReservationDate(result.ReservationDate);
    if (!checkDate)
    {
        return BadRequest(new JsonResult(new { message = "You can't cancel reservation from the past!" }));
    }
    try
    {
        _applicationContext.Reservations.Remove(result);
        await _applicationContext.SaveChangesAsync();
    }
    catch
    {
        return BadRequest(new JsonResult(new { message = "Something went wrong!" }));
    }

    return Ok(new JsonResult(new { message = "Reservation was successfully deleted!" }));
}
```

### Anexa 14, Verificarea dății de rezervare dacă este valabilă

```
private async Task<bool> CheckReservationAvailability(CreateReservationDto reservation)
{
    var reservationsToCheck = await _applicationContext.Reservations.Where(r => r.TableId == reservation.TableId).ToListAsync();
    foreach (var rez in reservationsToCheck)
    {
        int checkStart = DateTime.Compare(reservation.Reservationdate, rez.ReservationDate.AddMinutes(-5));
        int checkEnd = DateTime.Compare(reservation.Reservationdate, rez.ReservationDate.AddHours((int)rez.Duration).AddMinutes(5));
        int checkBeforeStart =
            DateTime.Compare(reservation.Reservationdate.AddHours(reservation.Duration), rez.ReservationDate.AddMinutes(-3));
        int checkBeforeEnd =
            DateTime.Compare(reservation.Reservationdate.AddHours(reservation.Duration), rez.ReservationDate.AddHours((int)rez.Duration).AddMinutes(3));

        if (checkStart <= -1 && checkEnd == -1)
        {
            return true;
        }
        if (checkBeforeStart == 1 && checkBeforeEnd == -1)
        {
            return true;
        }
    }
    return false;
}
```

### Anexa 15, Returnarea rezervărilor dintre două dăți

```

[Authorize(Roles = "Employee,Admin,POS")]
[HttpGet("get-all-reservations-employee-between/{startDate}/{endDate}")]
public async Task<List<GetReservationDto>> GetAllReservationBetween(DateTime startDate, DateTime endDate)
{
    var reservations = new List<Reservation>();
    if (startDate.Date == new DateTime(1900, 01, 01) && endDate.Date == new DateTime(1900, 01, 01))
    {
        reservations = await _applicationContext.Reservations.ToListAsync();
    }
    else
    {
        DateTime start = startDate.Date;
        DateTime end = endDate.Date.AddDays(1).AddTicks(-1);
        reservations = await _applicationContext.Reservations.Where(q => q.ReservationDate >= start && q.ReservationDate <= end).ToListAsync();
    }
    var listToReturn = new List<GetReservationDto>();
    if (reservations.Count > 0)
    {
        foreach (var reservation in reservations)
        {
            var reservationToAdd = new GetReservationDto
            {
                ReservationId = reservation.ReservationId,
                Reservationdate = reservation.ReservationDate,
                GuestNumber = reservation.GuestNumber,
                FirstName = reservation.FirstName,
                LastName = reservation.LastName,
                PhoneNumber = reservation.PhoneNumber,
                Reservationstatus = reservation.ReservationStatus,
                Duration = reservation.Duration,
                TableNumber = reservation.TableId,
            };
            listToReturn.Add(reservationToAdd);
        }
        return listToReturn;
    }
    return listToReturn;
}

```

### Anexa 16, Crearea unei mese, modificarea capacitatei acesteia

```

[Authorize(Roles = Dependencis.ADMIN_ROLE)]
[HttpPost("add-table")]
public async Task<IActionResult> AddTable(TabelDto model)
{
    var tableToAdd = new Table
    {
        Capacity = model.Capacity,
        TableStatus = model.Status,
    };
    await _applicationContext.AddAsync<Table>(tableToAdd);
    await _applicationContext.SaveChangesAsync();
    return Ok(new JsonResult(new { message = "Table was successfully added!" }));
}

[Authorize(Roles = Dependencis.ADMIN_ROLE)]
[HttpPut("change-table-capacity/{tableId}")]
public async Task<IActionResult> ChangeTableCapacity(TabelDto model, int tableId)
{
    var result = await _applicationContext.Tables.FindAsync(tableId);
    if (result == null)
    {
        return BadRequest("Such a table does not exist!");
    }
    result.Capacity = model.Capacity;
    await _applicationContext.SaveChangesAsync();
    return Ok(new JsonResult(new { message = $"New capacity for the table {tableId} is {model.Capacity} :D" }));
}

```

### Anexa 17, Ștergerea unei mese după id

```
[Authorize(Roles = Dependencias.ADMIN_ROLE)]
[HttpDelete("delete-table/{tableId}")]
public async Task<IActionResult> DeleteTable(int tableId)
{
    var table = await _applicationContext.Tables.FindAsync(tableId);
    if (table == null) return NotFound(new JsonResult(new { message = "Table was not found" }));
    var orders = await _applicationContext.Orders.Where(q => q.TableId == tableId).ToListAsync();
    if (orders.Count > 0)
    {
        foreach (var order in orders)
        {
            order.TableId = null;
            await _applicationContext.SaveChangesAsync();
        }
        _applicationContext.Remove(table);
        await _applicationContext.SaveChangesAsync();
        return Ok(new JsonResult(new { message = "Table was deleted and all orders with this table modified!" }));
    }
    else
    {
        _applicationContext.Remove(table);
        await _applicationContext.SaveChangesAsync();
        return Ok(new JsonResult(new { message = "Table was deleted and there were no orders with this table!" }));
    }
}
```

### Anexa 18, Crearea unui produs nou de către administrator

```
[Authorize(Roles = "Admin")]
[HttpPost("add-new-product-noncomplex")]
public async Task<IActionResult> AddNewProduct(StockProducts model)
{
    //check if a product with same name already exist. Because we're adding products we can change the name. This
    //is about uniqueness and stuff like this.
    var result = await _applicationContext.Products.Where(p => p.Name == model.Name).FirstOrDefaultAsync();
    if (result != null) { return BadRequest(new JsonResult(new { message = "A product with this name already exist!" })); }
    try
    {
        var productToAdd = new Product
        {
            Name = model.Name,
            UnitPrice = model.UnitPrice,
            UnitMeasure = model.UnitMeasure.ToLower(),
            AvailableForUser = model.AvailableForUser,
            ComplexProduct = false,
            CategoryId = model.CategoryId,
            Quantity = model.Quantity,
            SupplyCheck = model.SupplyCheck,
            Tva = model.TVA,
        };
        await _applicationContext.Products.AddAsync(productToAdd);
        await _applicationContext.SaveChangesAsync();
    }
    catch
    {
        return BadRequest(new JsonResult(new { message = "Something went wrong on adding a new nonComplex product!" }));
    }
    return Ok(new JsonResult(new { message = $"Product {model.Name} was successfully added!" }));
}
```

### Anexa 19, Crearea unui produs complex

```
[AuthorizedRoles = "Admin"]
[HttpPost("add-new-product-complex")]
public async Task<ActionResult> AddNewComplexProduct(ComplexProductDto model)
{
    //check if a product with same name already exist because similar products we can change the name (the
    //constraint called unique)
    var result = await _applicationContext.Products.Where(p => p.Name == model.Name).FirstOrDefaultAsync();
    if (result != null) { return BadRequest(new JsonResult(new { message = "A product with this name already exists!" })); }

    var idArray = model.ProductComponentsId.ToList();
    foreach (var id in idArray)
    {
        var checkExistence = await _applicationContext.Products.FindAsync(id);
        if (checkExistence == null)
        {
            return BadRequest("Components contain an id from an unexistence product!");
        }
    }

    if (idArray.Count == 0) { return BadRequest(new JsonResult(new { message = "A complex product should have some
products to be made of" })); }

    bool areUnique = idArray.Distinct().Count() == idArray.Count;
    if (!areUnique) return BadRequest(new JsonResult(new { message = "You can't duplicate components items!" }));
    try
    {
        var productToAdd = new Product
        {
            Name = model.Name,
            UnitPrice = model.UnitPrice,
            UnitMeasure = model.UnitMeasure.ToString(),
            AvailableForUser = model.AvailableForUser,
            ComplexProduct = true,
            CategoryId = model.CategoryId,
            Quantity = 0,
            SupplyCheck = 0,
            Tva = model.Tva
        };
        await _applicationContext.Products.AddAsync(productToAdd);
        await _applicationContext.SaveChangesAsync();

        foreach (var item in idArray)
        {
            var componentProduct = new ComplexProductsComponent
            {
                TargetProductId = productToAdd.ProductId,
                ComponentProductId = item.id,
                UsageQuantity = item.quantity,
            };
            await _applicationContext.ComplexProductsComponents.AddAsync(componentProduct);
            await _applicationContext.SaveChangesAsync();
        }
    }
    catch
    {
        return BadRequest(new JsonResult(new { message = "Something went wrong on adding a new nonComplex product!" }));
    }
    return Ok(new JsonResult(new { message = $"Product {model.Name} was successfully added!" }));
}
}
```

Anexa 20, Modificarea unui produs complex

```
[Authorize(Roles = "Dependenta,ADMIN_ROLE")]
[HttpPut("modify-complex-product")]
public async Task<ActionResult> ModifyComplexProduct(ComplexProductsDto model)
{
    var productToModify = await _applicationContext.Products.FindAsync(model.ProductId);
    if (productToModify == null) { return NotFound(); }
    productToModify.Quantity = 0;
    productToModify.SupplyCheck = 0;
    productToModify.Name = model.Name;
    productToModify.UnitPrice = model.UnitPrice;
    productToModify.UnitMeasure = model.UnitMeasure;
    productToModify.CategoryId = model.CategoryId;
    productToModify.Tva = model.Tva;
    productToModify.AvailableForUser = model.AvailableForUser;
    await _applicationContext.SaveChangesAsync();
    var componentProducts = await _applicationContext.ComplexProductsComponents.Where(q => q.TargetProductId == productToModify.ProductId).ToListAsync();
    foreach(var item in componentProducts)
    {
        _applicationContext.ComplexProductsComponents.Remove(item);
        await _applicationContext.SaveChangesAsync();
    }

    foreach(var item in model.ProductComponentsId)
    {
        var result = await _applicationContext.ComplexProductsComponents.Where(q => q.TargetProductId == model.ProductId && q.ComponentProductId == item.Id).FirstOrDefaultAsync();
        if (result == null)
        {
            var componentToAdd = new ComplexProductsComponent
            {
                TargetProductId = model.ProductId,
                ComponentProductId = item.Id,
                UsageQuantity = item.Quantity,
            };
            await _applicationContext.ComplexProductsComponents.AddAsync(componentToAdd);
            await _applicationContext.SaveChangesAsync();
            continue;
        }
        if(result.UsageQuantity == item.Quantity)
        {
            continue;
        }
        else
        {
            result.UsageQuantity = item.Quantity;
            await _applicationContext.SaveChangesAsync();
        }
    }
    return Ok(new JsonResult(new { message = "Product was modified successfully" }));
}
```

### Anexa 21, Returnarea tuturor produselor din stoc

```
[Authorize(Roles = Dependencis.ADMIN_ROLE)]
[HttpGet("get-stock")]
public async Task<List<StockProducts>> GetStock()
{
    var returnList = new List<StockProducts>();
    var products = await _applicationContext.Products.ToListAsync();
    if (products.Count == 0) return new List<StockProducts>();

    foreach (var product in products)
    {
        var productToAdd = new StockProducts
        {
            ProductId = product.ProductId,
            Name = product.Name,
            UnitPrice = product.UnitPrice,
            UnitMeasure = product.UnitMeasure,
            AvailableForUser = product.AvailableForUser,
            ComplexProduct = product.ComplexProduct,
            CategoryId = product.CategoryId,
            Quantity = product.Quantity,
            SupplyCheck = product.SupplyCheck,
            TVA = product.Tva,
            ComponentProducts = new List<ComponentProductDto>()
        };
        if (product.ComplexProduct == true)
        {
            var componentProducts = await _applicationContext.ComplexProductsComponents.Where(q => q.TargetProductId == product.ProductId).ToListAsync();
            foreach (var componentProduct in componentProducts)
            {
                var componentProductToAdd = new ComponentProductDto
                {
                    Id = componentProduct.ComponentProductId,
                    Name = componentProduct.ComponentProduct.Name,
                    quantity = componentProduct.UsageQuantity,
                };
                productToAdd.ComponentProducts.Add(componentProductToAdd);
            }
        }
        returnList.Add(productToAdd);
    }
    return returnList;
}
```

### Anexa 22, Returnarea produselor cu cantitatea sub limita stabilită

```
[Authorize(Roles = Dependencis.ADMIN_ROLE)]
[HttpGet("supply-check")]
public async Task<List<Product>> SupplyCheck()
{
    var reportProductList = new List<Product>();
    var productsToCheck = await _applicationContext.Products.ToListAsync();
    if (productsToCheck.Count <= 0)
    {
        return new List<Product>();
    }
    foreach (var product in productsToCheck)
    {
        if (product.Quantity <= product.SupplyCheck & product.ComplexProduct == false)
        {
            reportProductList.Add(product);
        }
    }
    return reportProductList;
}
```

*Anexa 23, Returnarea produselor pentru un meniu specificat după id*

```
[HttpGet("get-menu-products/{categoryId}")]
public async Task<List<GetMenuProductDto>> GetMenuProducts( int categoryId)
{
    var result = await _applicationContext.Products.Where(q => q.CategoryId == categoryId && q.AvailableForUser == true).ToListAsync();
    if (result == null) return new List<GetMenuProductDto>();
    var categoryProducts = new List<GetMenuProductDto>();
    foreach (var item in result)
    {
        var product = new GetMenuProductDto
        {
            ProductId = item.ProductId,
            ProductName = item.Name,
            ProductPrice = item.UnitPrice,
            ProductAvailability = item.Quantity,
            ProductSupplyCheck = item.SupplyCheck,
            ComplexProduct = item.ComplexProduct
        };
        categoryProducts.Add(product);
    }
    return categoryProducts;
}
```

*Anexa 24, Modificarea unui produs non-complex*

```
[Authorize(Roles = Dependencis.ADMIN_ROLE)]
[HttpPut("modify-nonComplexProduct")]
public async Task<IActionResult> ModifyNonComplexProduct(StockProducts model)
{
    var productToModify = await _applicationContext.Products.FindAsync(model.ProductId);
    if (productToModify == null) return NotFound(new JsonResult(new { message = "The product cannot be found!" }));
    productToModify.Name = model.Name;
    productToModify.UnitPrice = model.UnitPrice;
    productToModify.UnitMeasure = model.UnitMeasure;
    productToModify.AvailableForUser = model.AvailableForUser;
    productToModify.ComplexProduct = false;
    productToModify.CategoryId = model.CategoryId;
    productToModify.Quantity = model.Quantity;
    productToModify.SupplyCheck = model.SupplyCheck;
    productToModify.Tva = model.TVA;

    await _applicationContext.SaveChangesAsync();
    return Ok(new JsonResult(new { message = "Product was successfully modified!" }));
}
```

*Anexa 25, Adăugarea unei cantități pentru un produs*

```
[Authorize(Roles = "Admin, POS")]
[HttpPost("add-stock-quantity")]
public async Task<IActionResult> AddStockQuantity(List<AddProductsQuantityDto> model)
{
    if (model.Count > 0)
    {
        foreach (var item in model)
        {
            var productDetails = await _applicationContext.Products.FindAsync(item.ProductId);
            productDetails.Quantity += item.Added_quantity;
            await _applicationContext.SaveChangesAsync();
        }
        return Ok(new JsonResult(new { message = "Products quantities was successfully modified!" }));
    }
    return BadRequest();
}
```

*Anexa 26, Adăugarea unei noi categorii pentru produse*

```
[Authorize(Roles = Dependencias.ADMIN_ROLE)]
[HttpPost("add-category")]
public async Task<ActionResult> AddCategory(CategoryDto model)
{
    var result = await _applicationContext.Categories.Where(q => q.CategoryName == model.Name).FirstOrDefaultAsync();
    if(result != null) return BadRequest(new JsonResult(new { message = "Such a category already exist!" }));
    var categoryToAdd = new Category
    {
        CategoryName = model.Name,
        AvailableMenu = model.AvailableMenu,
    };
    try
    {
        var result = await _applicationContext.Categories.AddAsync(categoryToAdd);
        await _applicationContext.SaveChangesAsync();
    }
    catch
    {
        return BadRequest(new JsonResult(new { message = "Something went wrong for adding a new category!" }));
    }
    return Ok(new JsonResult(new { message = $"A new category was added by name {model.Name}" }));
}
```

*Anexa 27, Modificarea unei categorii de produse*

```
[Authorize(Roles = Dependencias.ADMIN_ROLE)]
[HttpPut("modify-product-category")]
public async Task<ActionResult> ModifyProductCategory(GetCategoryDto model)
{
    var result = await _applicationContext.Categories.FindAsync(model.CategoryId);
    if(result == null) return NotFound(new JsonResult(new { message = "Category was not found!" }));
    result.CategoryName = model.CategoryName;
    result.AvailableMenu = model.AvailableMenu;
    await _applicationContext.SaveChangesAsync();
    return Ok(new JsonResult(new { message = "Modification was applied!" }));
}
```

*Anexa 28, Adăugarea unei noi categorii de balansare*

```
[Authorize(Roles = Dependencias.ADMIN_ROLE)]
[HttpPost("add-balance-category")]
public async Task<ActionResult> AddInBalanceCategory(AddBalanceCategoryDto model)
{
    var result = await _applicationContext.RemoveCategories.Where(q => q.RemoveCategoryName == model.removeCategoryName).FirstOrDefaultAsync();
    if(result != null) return BadRequest(new JsonResult(new { message = "There is a balance category with this name!" }));
    var balanceCategoryToAdd = new RemoveCategory
    {
        RemoveCategoryName = model.removeCategoryName,
    };
    try
    {
        await _applicationContext.RemoveCategories.AddAsync(balanceCategoryToAdd);
        await _applicationContext.SaveChangesAsync();
    }
    catch
    {
        return BadRequest(new JsonResult(new { message = "Well something went to the left somewhere!" }));
    }
    return Ok(new JsonResult(new { message = $"New balance category with the name: {model.removeCategoryName} was successfully added!" }));
}
```

### Anexa 29, Modificarea unei categorii de balansare

```
[Authorize(Roles = Dependencies.ADMIN_ROLE)]
[HttpPost("modify-balance-category")]
public async Task<IActionResult> ModifyBalanceCategory(RemoveCategory model)
{
    var result = await _applicationContext.RemoveCategories.FindAsync(model.RemoveCategoryId);
    if (result == null) return NotFound(new JsonResult(new { message = "Balance category was not found!" }));
    result.RemoveCategoryName = model.RemoveCategoryName;
    await _applicationContext.SaveChangesAsync();
    return Ok(new JsonResult(new { message = "Modification was applied!" }));
}
```

### Anexa 30, Crearea unei comenzi de către client

```
[AuthorizedPolicy = "CheckOpenStatus"]
[Authorize(Roles = Dependencies.DEFAULT_ROLE)]
[HttpPost("new-client-order/{clientId}/{tableId}")]
public async Task<IActionResult> NewClientOrder(ClientOrderDto model, int clientId, int tableId)
{
    var notAddedProducts = new List<string>();
    var checkTableAvailability = await _applicationContext.Tables.FindAsync(tableId);
    if (checkTableAvailability.TableStatus == true) return BadRequest(new JsonResult(new { title = "Table occupied!", message = "This table is not available right now!" }));
    var unfinishedOrders = await _applicationContext.Orders.Where(q => q.ClientId == clientId).ToListAsync();
    foreach (var order in unfinishedOrders)
    {
        if (order.OrderStatus <= 3) return BadRequest(new JsonResult(new { title = "Unfinished order exist!", message = "An unfinished order exist for your account you can add new products to that order or finish the previous one!" }));
    }
    var orderToAdd = new Order
    {
        OrderDate = DateTime.Now,
        OrderStatus = 1,
        ClientId = clientId,
        TakenEmployeeId = null,
        TableId = tableId,
        Tips = 0,
    };
    checkTableAvailability.TableStatus = true;
    await _applicationContext.Orders.AddAsync(orderToAdd);
    await _applicationContext.SaveChangesAsync();
    foreach (var product in model.Products)
    {
        var orderProduct = new OrderProduct
        {
            OrderId = orderToAdd.OrderId,
            ProductId = product.productId,
            UnitPrice = product.unitPrice,
            Quantity = product.quantity,
        };
        var productDetails = await _applicationContext.Products.FindAsync(product.productId);
        if (productDetails == null)
        {
            notAddedProducts.Add(productDetails.Name);
            continue;
        }
        if (productDetails.ComplexProduct == false && productDetails.Quantity < product.quantity)
        {
            notAddedProducts.Add(productDetails.Name);
            continue;
        }
        await _applicationContext.OrderProducts.AddAsync(orderProduct);
        await _applicationContext.SaveChangesAsync();
    }
    if (notAddedProducts.Count > 0)
    {
        return Ok(new JsonResult(new { title = "Order was successfully registered but:", message = $"The next products {string.Join(", ", notAddedProducts)} cannot be added to the order because of the insufficient quantity" }));
    }
    return Ok(new JsonResult(new { title = "Order sent", message = "Order was successfully registered, you can see it in active orders! :D" }));
}
```

Anexa 31, Confirmarea unei comenzi de către un angajat

```
[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = Dependencies.EMPLOYEE_ROLE)]
[HttpPut("confirm-order/{employeeId}/{orderId}")]
public async Task<IActionResult> ConfirmOrder(int employeeId, int orderId)
{
    var notAddedProducts = new List<string>();
    var result = await _applicationContext.Orders.FindAsync(orderId);
    if (result == null) return BadRequest(new JsonResult(new { message = "Such an order does not exist!" }));
    if (result.OrderStatus != 1)
    {
        return BadRequest(new JsonResult(new { message = "This order is already confirmed!" }));
    }

    result.OrderStatus += 1;
    result.TakenEmployeeId = employeeId;
    var orderProducts = await _applicationContext.OrderProducts.Where(q => q.OrderId == orderId).ToListAsync();
    foreach (var product in orderProducts)
    {
        var productDetails = await _applicationContext.Products.FindAsync(product.ProductId);
        if (productDetails == null)
        {
            notAddedProducts.Add(productDetails.Name);
            continue;
        }
        if (productDetails.ComplexProduct == false && productDetails.Quantity < product.Quantity)
        {
            notAddedProducts.Add(productDetails.Name);
            continue;
        }
        if (productDetails.ComplexProduct == false)
        {
            productDetails.Quantity -= product.Quantity;

            var stockBalanceRecord = new StockBalance
            {
                BalanceDate = DateTime.Now,
                ProductId = productDetails.ProductId,
                RemoveQuantity = product.Quantity,
                RemoveCategoryId = 1,
            };

            await _applicationContext.StockBalances.AddAsync(stockBalanceRecord);
            await _applicationContext.SaveChangesAsync();
        }
    }
    if (notAddedProducts.Count > 0)
    {
        await _applicationContext.SaveChangesAsync();
        return Ok(new JsonResult(new { title = "Order was successfully registered but!", message = $"The next products {string.Join(", ", notAddedProducts)} cannot be added to the order because of the insufficient quantity" }));
    }
    await _applicationContext.SaveChangesAsync();
    return Ok(new JsonResult(new { message = "Order is confirmed!" }));
}
```

Anexa 32, Returnarea comenzilor ce trebuie confirmate

```
[Authorize(Roles = Dependencis.EMPLOYEE_ROLE)]
[HttpGet("get-orders-to-confirmed")]
public async Task<List<GetClientOrderDto>> GetClientOrders()
{
    var orderList = new List<GetClientOrderDto>();
    var result = await _applicationContext.Orders.Where(q => q.OrderStatus == 1).ToListAsync();
    foreach (var order in result)
    {
        var employee = await _applicationContext.Employees.FindAsync(order.TakenEmployeeId);
        string employeeName = string.Empty;
        if (employee != null)
        {
            employeeName = employee.FirstName;
        }
        else
        {
            employeeName = "";
        }
        var product = _applicationContext.OrderProducts.Where(q => q.OrderId == order.OrderId);
        var list = new List<OrderProductInformationDto>();
        foreach (var item in product)
        {
            var productName = await _applicationContext.Products.FindAsync(item.ProductId);
            var productOrder = new OrderProductInformationDto
            {
                ProductName = productName.Name,
                UnitPrice = item.UnitPrice,
                Quantity = item.Quantity,
            };
            list.Add(productOrder);
        }

        string statusName = string.Empty;
        switch (order.OrderStatus)
        {
            case 1:
                statusName = "Pending";
                break;
            case 2:
                statusName = "Accepted";
                break;
            case 3:
                statusName = "Delivered";
                break;
            case 4:
                statusName = "Finished";
                break;
            case 5:
                statusName = "Cancelled";
                break;
        }

        var clientOrder = new GetClientOrderDto
        {
            OrderId = order.OrderId,
            OrderDate = order.OrderDate,
            TableId = order.TableId,
            EmployeeName = employeeName,
            Status = statusName,
            products = list,
        };
        orderList.Add(clientOrder);
    }
    return orderList;
}
```

### Anexa 33, Adăugarea produselor noi la o comandă de către client

```

[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = Dependencis.DEFAULT_ROLE)]
[HttpPost("add-new-product-to-order/{clientId}/{orderId}/{tableId}")]
public async Task<IActionResult> AddNewProduct(ClientOrderDto model, int clientId, int orderId, int tableId)
{
    var insufficientProducts = new List<string>();
    var CheckOrderExist = await _applicationContext.Orders.Where(q => q.OrderId == orderId & q.ClientId == clientId &
q.OrderStatus <= 3).FirstOrDefaultAsync();
    if (CheckOrderExist == null) { return BadRequest(new JsonResult(new { message = "Such an order does not exist or
it is not your order!" })); }
    if (CheckOrderExist.TableId != tableId) { return BadRequest(new JsonResult(new { message = "The table id is not
the same!\nMake sure to scan the same qr code as for creating the order!" })); }
    CheckOrderExist.OrderStatus = 1;

    foreach (var product in model.Products)
    {
        var productDetails = await _applicationContext.Products.FindAsync(product.productId);
        if (productDetails == null)
        {
            insufficientProducts.Add(productDetails.Name);
            continue;
        }
        if (productDetails.ComplexProduct == false && productDetails.Quantity < product.quantity)
        {
            insufficientProducts.Add(productDetails.Name);
            continue;
        }
        var checkProductExistInOrder = await _applicationContext.OrderProducts.Where(q => q.OrderId == orderId &&
q.ProductId == product.productId).FirstOrDefaultAsync();
        if (checkProductExistInOrder == null)
        {
            var productToAdd = new OrderProduct
            {
                OrderId = orderId,
                ProductId = product.productId,
                Quantity = product.quantity,
                UnitPrice = product.unitPrice,
            };
            await _applicationContext.OrderProducts.AddAsync(productToAdd);
        }
        else
        {
            checkProductExistInOrder.Quantity += product.quantity;
            await _applicationContext.SaveChangesAsync();
        }
        await _applicationContext.SaveChangesAsync();
    }
    if (insufficientProducts.Count > 0)
    {
        return Ok(new JsonResult(new { title = "Well", message = $"The following products: {string.Join(", ", insufficientProducts)}; cannot be added to your order because there is not enough stock!" }));
    }

    return Ok(new JsonResult(new { title = "Success", message = "The new products was added to your order!" }));
}

```

### Anexa 34, Crearea unei noi comenzi de către un angajat

```

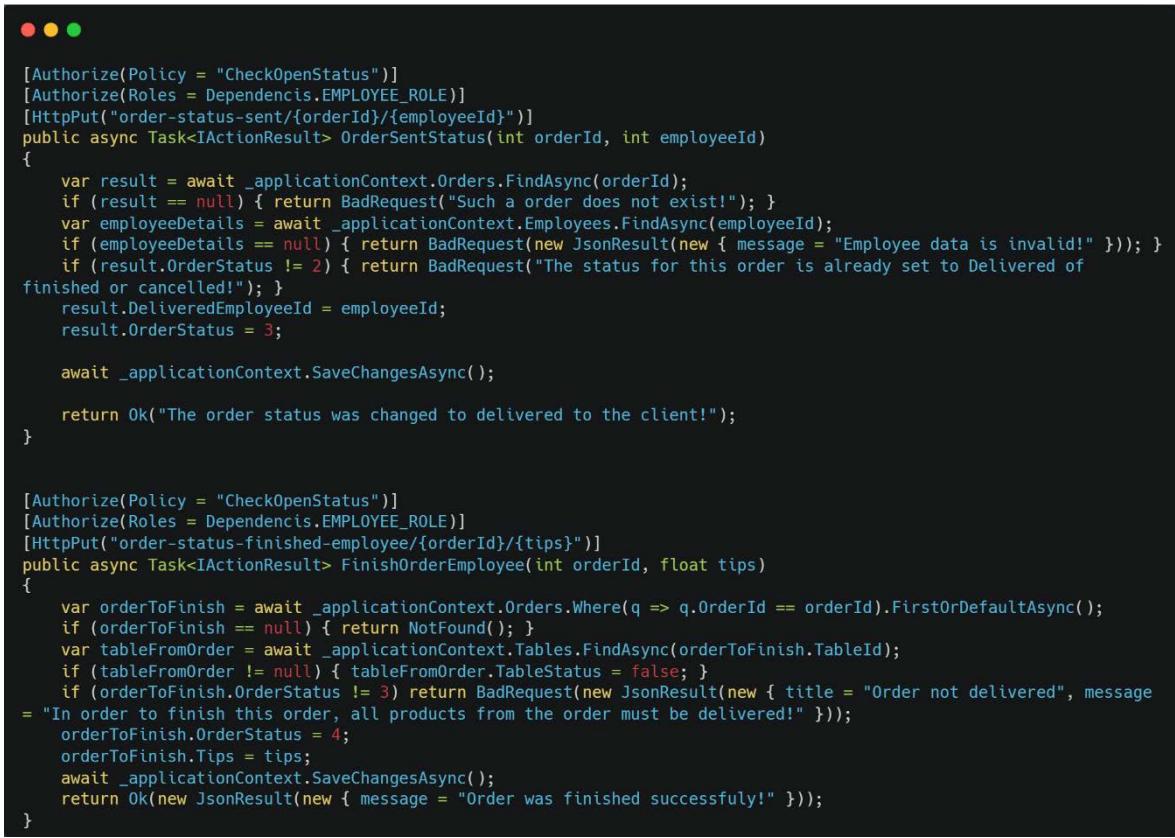
[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = Dependencis.EMPLOYEE_ROLE)]
[HttpPost("employee-create-order/{employeeId}/{payedStatus}")]
public async Task<ActionResult> EmployeeCreateNewOrder(EmployeeOrderDto model, int employeeId, bool payedStatus)
{
    int? finishEmployeeId = 0;
    var status = 2;
    if (payedStatus)
    {
        status = 4;
        finishEmployeeId = employeeId;
    }
    else
    {
        finishEmployeeId = null;
    }
    var notAddedProducts = new List<string>();
    var orderToAdd = new Order();
    var checkTableIdExist = await _applicationContext.Tables.FindAsync(model.TableId);
    if (checkTableIdExist != null)
    {
        if (checkTableIdExist.TableStatus == true) return BadRequest(new JsonResult(new { message = "This table is
occupied right now!" }));
        orderToAdd = new Order
        {
            OrderDate = DateTime.Now,
            OrderStatus = status,
            ClientId = null,
            TakenEmployeeId = employeeId,
            DeliveredEmployeeId = finishEmployeeId,
            TableId = model.TableId,
            Tips = 0,
        };
        if (status == 2) { checkTableIdExist.TableStatus = true; }
        await _applicationContext.Orders.AddAsync(orderToAdd);
        await _applicationContext.SaveChangesAsync();
    }
    else
    {
        orderToAdd = new Order
        {
            OrderDate = DateTime.Now,
            OrderStatus = status,
            ClientId = null,
            TakenEmployeeId = employeeId,
            DeliveredEmployeeId = finishEmployeeId,
            TableId = null,
            Tips = 0,
        };
        await _applicationContext.Orders.AddAsync(orderToAdd);
        await _applicationContext.SaveChangesAsync();
    }
    foreach (var product in model.Products)
    {
        var orderProduct = new OrderProduct
        {
            OrderId = orderToAdd.OrderId,
            ProductId = product.productId,
            UnitPrice = product.unitPrice,
            Quantity = product.quantity,
        };
        var productDetails = await _applicationContext.Products.FindAsync(product.productId);
        if (productDetails == null)
        {
            notAddedProducts.Add(productDetails.Name);
            continue;
        }

        if (productDetails.ComplexProduct == false && productDetails.Quantity < product.quantity)
        {
            notAddedProducts.Add(productDetails.Name);
            continue;
        }
        if (productDetails.ComplexProduct == false)
        {
            productDetails.Quantity -= product.quantity;
            var stockBalanceRecord = new StockBalance
            {
                BalanceDate = DateTime.Now,
                ProductId = productDetails.ProductId,
                RemoveQuantity = product.quantity,
                RemoveCategoryId = 1,
            };
            await _applicationContext.StockBalances.AddAsync(stockBalanceRecord);
        }
        await _applicationContext.OrderProducts.AddAsync(orderProduct);
        await _applicationContext.SaveChangesAsync();
    }
    if (notAddedProducts.Count > 0)
    {
        return Ok(new JsonResult(new { title = "Order was successfully registered but:", message = $"The next products
{string.Join(", ", notAddedProducts)} cannot be added to the order because of the insufficient quantity", orderId =
orderToAdd.OrderId }));
    }
    return Ok(new JsonResult(new { message = "A new order was created successfully!", orderId = orderToAdd.OrderId
}));
}

```

---

Anexa 35, Schimbarea statusului unei comenzi ca livrat și finalizat



```
[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = Dependencis.EMPLOYEE_ROLE)]
[HttpPut("order-status-sent/{orderId}/{employeeId}")]
public async Task<IActionResult> OrderSentStatus(int orderId, int employeeId)
{
    var result = await _applicationContext.Orders.FindAsync(orderId);
    if (result == null) { return BadRequest("Such a order does not exist!"); }
    var employeeDetails = await _applicationContext.Employees.FindAsync(employeeId);
    if (employeeDetails == null) { return BadRequest(new JsonResult(new { message = "Employee data is invalid!" })); }
    if (result.OrderStatus != 2) { return BadRequest("The status for this order is already set to Delivered or finished or cancelled!"); }
    result.DeliveredEmployeeId = employeeId;
    result.OrderStatus = 3;

    await _applicationContext.SaveChangesAsync();

    return Ok("The order status was changed to delivered to the client!");
}

[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = Dependencis.EMPLOYEE_ROLE)]
[HttpPut("order-status-finished-employee/{orderId}/{tips}")]
public async Task<IActionResult> FinishOrderEmployee(int orderId, float tips)
{
    var orderToFinish = await _applicationContext.Orders.Where(q => q.OrderId == orderId).FirstOrDefaultAsync();
    if (orderToFinish == null) { return NotFound(); }
    var tableFromOrder = await _applicationContext.Tables.FindAsync(orderToFinish.TableId);
    if (tableFromOrder != null) { tableFromOrder.TableStatus = false; }
    if (orderToFinish.OrderStatus != 3) return BadRequest(new JsonResult(new { title = "Order not delivered", message = "In order to finish this order, all products from the order must be delivered!" }));
    orderToFinish.OrderStatus = 4;
    orderToFinish.Tips = tips;
    await _applicationContext.SaveChangesAsync();
    return Ok(new JsonResult(new { message = "Order was finished successfully!" }));
}
```

Anexa 36, Adăugarea unor noi produse la o comandă de către un angajat

```

[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = Dependents.EMPLOYEE_ROLE)]
[HttpPost("add-new-product-to-order-employee/{orderId}")]
public async Task<IActionResult> AddNewProductEmployee(ClientOrderDto model, int orderId)
{
    var insufficientProducts = new List<string>();
    var CheckOrderExist = await _applicationContext.Orders.Where(q => q.OrderId == orderId & q.OrderStatus == 3).FirstOrDefaultAsync();
    if (CheckOrderExist == null) { return BadRequest(new JsonResult(new { message = "Such an order does not exist or it is not your order!" })); }
    if (CheckOrderExist.OrderStatus == 1) return BadRequest(new JsonResult(new { message = "Before add some new products order must be confirmed!" }));;
    CheckOrderExist.OrderStatus = 2;

    foreach (var product in model.Products)
    {
        var productDetails = await _applicationContext.Products.FindAsync(product.productId);
        if (productDetails == null)
        {
            insufficientProducts.Add(productDetails.Name);
            continue;
        }
        if (productDetails.ComplexProduct == false && productDetails.Quantity < product.quantity)
        {
            insufficientProducts.Add(productDetails.Name);
            continue;
        }
        var checkProductExistInOrder = await _applicationContext.OrderProducts.Where(q => q.OrderId == orderId && q.ProductId == product.productId).FirstOrDefaultAsync();
        if (checkProductExistInOrder == null)
        {
            var productToAdd = new OrderProduct
            {
                OrderId = orderId,
                ProductId = product.productId,
                Quantity = product.quantity,
                UnitPrice = product.unitPrice,
            };
            await _applicationContext.OrderProducts.AddAsync(productToAdd);
        }
        else
        {
            checkProductExistInOrder.Quantity += product.quantity;
            await _applicationContext.SaveChangesAsync();
        }
        if (productDetails.ComplexProduct == false)
        {
            productDetails.Quantity -= product.quantity;
            var stockBalanceRecord = new StockBalance
            {
                BalanceDate = DateTime.Now,
                ProductId = productDetails.ProductId,
                RemoveQuantity = product.quantity,
                RemoveCategoryId = 1,
            };
            await _applicationContext.StockBalances.AddAsync(stockBalanceRecord);
        }
        await _applicationContext.SaveChangesAsync();
    }
    if (insufficientProducts.Count > 0)
    {
        return Ok(new JsonResult(new { message = $"The following products {string.Join(", ", insufficientProducts)} cannot be added to order because they aren't available in that quantity!" }));
    }
    return Ok(new JsonResult(new { message = "The new products was added to order!", products = model.Products }));
}

```

### Anexa 37, Returnarea datelor pentru nota de comandă

```

[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = "Employee,Client,POS")]
[HttpGet("get-order-note-data/{orderId}")]
public async Task<GetNoteDataDto> GetOrderNoteData(int orderId)
{
    var organization = await _applicationContext.Organizations.FindAsync(1);
    var organizationName = organization.Name;
    var orderDetails = await _applicationContext.Orders.FindAsync(orderId);
    if (orderDetails == null) { return new GetNoteDataDto(); }

    var employeeName = string.Empty;
    var employeeDetails = await _applicationContext.Employees.FindAsync(orderDetails.TakenEmployeeId);
    if (employeeDetails != null)
    {
        employeeName = employeeDetails.FirstName + ' ' + employeeDetails.LastName;
    }

    string statusName = string.Empty;
    switch (orderDetails.OrderStatus)
    {
        case 1:
            statusName = "Pending";
            break;
        case 2:
            statusName = "Accepted";
            break;
        case 3:
            statusName = "Delivered";
            break;
        case 4:
            statusName = "Finished";
            break;
        case 5:
            statusName = "Cancelled";
            break;
    }

    var orderProducts = await _applicationContext.OrderProducts.Where(q => q.OrderId == orderId).ToListAsync();

    double? totalPrice = 0;
    var noteProducts = new List<OrderProductInformationDto>();

    foreach (var item in orderProducts)
    {
        var productDetails = await _applicationContext.Products.FindAsync(item.ProductId);
        if (productDetails == null) { continue; }
        var productName = productDetails.Name;
        var productToAdd = new OrderProductInformationDto
        {
            ProductName = productName,
            UnitPrice = item.UnitPrice,
            Quantity = item.Quantity,
        };
        totalPrice += item.UnitPrice * Convert.ToDouble(item.Quantity);
        noteProducts.Add(productToAdd);
    }

    var NoteData = new GetNoteDataDto
    {
        OrganizationName = organizationName,
        OrderDate = orderDetails.OrderDate,
        TableId = orderDetails.TableId,
        products = noteProducts,
        EmployeeName = employeeName,
        OrderId = orderId,
        OrderStatus = statusName,
        total = totalPrice,
    };
    return NoteData;
}

```

### Anexa 38, Returnarea datelor pentru bonul fiscal

```

● ● ●

[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = Dependencis.EMPLOYEE_ROLE)]
[HttpGet("get-receipt-data/{orderId}")]
public async Task<GetReceiptDataDto> GetReceiptData(int orderId)
{
    var organizationData = await _applicationContext.Organizations.FindAsync(1);
    var orderDetails = await _applicationContext.Orders.FindAsync(orderId);
    var productsFromOrder = await _applicationContext.OrderProducts.Where(q => q.OrderId == orderId).ToListAsync();
    var productsList = new List<OrderProductDto>();
    foreach (var product in productsFromOrder)
    {
        var productNameDetails = await _applicationContext.Products.FindAsync(product.ProductId);
        var item = new OrderProductDto
        {
            productName = productNameDetails.Name,
            productId = product.ProductId,
            unitPrice = product.UnitPrice,
            tva = productNameDetails.Tva,
            quantity = product.Quantity,
        };
        productsList.Add(item);
    }

    var receiptData = new GetReceiptDataDto
    {
        Name = organizationData.Name,
        adress = organizationData.Address,
        CIF = organizationData.Cif,
        City = organizationData.City,
        CreatedDate = DateTime.UtcNow,
        products = productsList
    };
}

return receiptData;
}

```

### Anexa 39, Anularea unei comenzi și ștergerea unui produs din comandă

```

● ● ●

[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = Dependencis.POS_ROLE)]
[HttpPut("order-status-cancel/{orderId}")]
public async Task<IActionResult> CancelOrder(int orderId)
{
    var orderDetails = await _applicationContext.Orders.FindAsync(orderId);
    var tableDetails = await _applicationContext.Tables.FindAsync(orderDetails.TableId);
    if(orderDetails == null) return NotFound();
    orderDetails.OrderStatus = 5;
    if (tableDetails != null) { tableDetails.TableStatus = false; }
    await _applicationContext.SaveChangesAsync();
    return Ok(new JsonResult(new { message = "Order was canceled, please add back the cancelled quantities in stock!" }));
}

[Authorize(Policy = "CheckOpenStatus")]
[Authorize(Roles = Dependencis.POS_ROLE)]
[HttpDelete("delete-order-product/{orderId}/{productId}")]
public async Task<IActionResult> DeleteProductFromOrder(int productId, int orderId)
{
    var productCheck = await _applicationContext.OrderProducts.Where(q => q.OrderId == orderId && q.ProductId == productId).ToListAsync();
    if (productCheck.Count > 0)
    {
        foreach(var product in productCheck)
        {
            _applicationContext.OrderProducts.Remove(product);
        }
        await _applicationContext.SaveChangesAsync();
        return Ok(new JsonResult(new { message = "Product was deleted from order! Please add back product quantity if is needed!" }));
    }
    return BadRequest(new JsonResult(new { message = "Somethin went wrong!" }));
}

```

*Anexa 40, Crearea unei balansări de stoc*

```
[Authorize(Roles = "Admin, POS")]
[HttpPost("new-balance-record/{productID}/{categoryID}/{quantity}")]
public async Task<ActionResult> NewBalanceStockRecords(int productID, int categoryID, int quantity)
{
    var productDetails = await _applicationContext.Products.FindAsync(productID);
    if (productDetails != null)
    {
        if (quantity > productDetails.Quantity) { return BadRequest(new JsonResult(new { message = "Removable quantity is bigger than the stock quantity!" })); }
        productDetails.Quantity -= quantity;
        var stockBalanceToAdd = new StockBalance
        {
            ProductId = productID,
            RemoveQuantity = quantity,
            RemoveCategoryId = categoryID,
            BalanceDate = DateTime.Now,
        };
        await _applicationContext.StockBalances.AddAsync(stockBalanceToAdd);
        await _applicationContext.SaveChangesAsync();
        return Ok(new JsonResult(new { message = "Quantity changed!" }));
    }
    return NotFound(new JsonResult(new { message = "Product not found in db!" }));
}
```

*Anexa 41, Schimbarea statusului casei de marcat închis/deschis*

```
[Authorize(Roles = "Admin,POS")]
[HttpPut("change-status-pos/{status}")]
public async Task<IActionResult> OpenPos(bool status)
{
    var date = DateTime.Now;
    DateTime dateForComarasing = new DateTime(date.Year, date.Month, date.Day);

    var result = await _applicationContext.Organizations.FindAsync(1);
    if (status == true)
    {
        result.OpenStatus = true;
        await _applicationContext.SaveChangesAsync();
        return Ok(new JsonResult(new { message = "Pos is open now!", status = true }));
    }
    else
    {
        var orders = await _applicationContext.Orders.Where(q => q.OrderStatus < 4 && q.OrderDate.HasValue &&
q.OrderDate.Value.Year == dateForComarasing.Year &&
q.OrderDate.Value.Month == dateForComarasing.Month &&
q.OrderDate.Value.Day == dateForComarasing.Day).ToListAsync();
        if (orders.Count > 0)
        {
            return BadRequest(new JsonResult(new { message = "There are still unfinished orders" }));
        }
        result.OpenStatus = false;
        await _applicationContext.SaveChangesAsync();
        return Ok(new JsonResult(new { message = "Pos is closed now!", status = false }));
    }
}
```

*Anexa 42, Returnarea datelor pentru raportul de închidere de casă*

```

[Authorize(Roles = "Admin,PM")]
[HttpGet("pos-closing-report/{date?}")]
public async Task<PosClosingReport> GetPosClosingReport(DateTime date)
{
    DateTime dateForClosing = new DateTime(date.Year, date.Month, date.Day);
    await context.SaveChangesAsync();
    var currentDate = DateTime.UtcNow;
    var organization = await _unitOfWork.Repository.FindByKey(1);
    var totalOrders = await _unitOfWork.Repository.Where(o =>
        o.OrderDate.DateValue.Year == dateForClosing.Year &&
        o.OrderDate.DateValue.Month == dateForClosing.Month &&
        o.OrderDate.DateValue.Day == dateForClosing.Day &&
        o.OrderStatus == 0);
    var cancelOrders = await _unitOfWork.Repository.Where(o =>
        o.OrderDate.DateValue.Year == dateForClosing.Year &&
        o.OrderDate.DateValue.Month == dateForClosing.Month &&
        o.OrderDate.DateValue.Day == dateForClosing.Day &&
        o.OrderStatus == 1);
    var totalRevenue = await _unitOfWork.Repository.Where(o =>
        o.OrderDate.DateValue.Year == dateForClosing.Year &&
        o.OrderDate.DateValue.Month == dateForClosing.Month &&
        o.OrderDate.DateValue.Day == dateForClosing.Day).Sum(o => o.Total);
    if (cancelOrders.Count <= 0) cancelOrders.Count = 0;
    return new PosClosingReport(total);
}
else
{
    foreach (var order in totalOrders)
    {
        var orderProducts = await _unitOfWork.Repository.Where(o => o.OrderId == order.OrderId).ToListAsync();
        if (orderProducts.Count > 0)
        {
            foreach (var product in orderProducts)
            {
                totalRevenue += product.UnitPrice * product.Quantity;
            }
        }
    }
    var employeeList = new List<Employee>();
    var allEmployees = await _unitOfWork.Repository<Employee>.ToListAsync();
    if (allEmployees.Count > 0)
    {
        foreach (var employee in allEmployees)
        {
            var takenOrders = await _unitOfWork.Repository<Orders>.Where(o => o.DeliveryDate.DateValue.Year == dateForClosing.Year &&
            o.DeliveryDate.DateValue.Month == dateForClosing.Month &&
            o.DeliveryDate.DateValue.Day == dateForClosing.Day).Where(o => employee.EmployeeId == o.OrderId).ToListAsync();
            var deliveredOrders = await _unitOfWork.Repository<Orders>.Where(o => o.DeliveryDate.DateValue.Year == dateForClosing.Year &&
            o.DeliveryDate.DateValue.Month == dateForClosing.Month &&
            o.DeliveryDate.DateValue.Day == dateForClosing.Day).Where(o => o.DeliveryEmployee == employee.EmployeeId).ToListAsync();
            employee.EmployeeTotalTakenOrders = takenOrders.Count;
            employee.EmployeeTotalDeliveredOrders = deliveredOrders.Count;
            employee.EmployeeTotalAvailable = employee.EmployeeTotalTakenOrders + employee.EmployeeTotalDeliveredOrders;
        }
    }
    var employeeTable = new EmployeeOrderTable();
    foreach (var employee in allEmployees)
    {
        var name = employee.LastName + " " + employee.FirstName;
        employeeTable.Add(employee, name, employee.EmployeeTotalAvailable);
    }
    var productsTable = new List<ProductOrderTable>();
    var allSelectedProductsPerDay = await _unitOfWork.Repository<OrdersProducts>.Where(o => o.Order.OrderDate.DateValue.Year == dateForClosing.Year &&
    o.Order.OrderDate.DateValue.Month == dateForClosing.Month &&
    o.Order.OrderDate.DateValue.Day == dateForClosing.Day).Where(o => o.Order.OrderStatus == 0).ToListAsync();
    if (allSelectedProductsPerDay.Count > 0)
    {
        var allProducts = await _unitOfWork.Repository<Products>.Where(p => p.Availability > 0).ToListAsync();
        foreach (var item in allProducts)
        {
            foreach (var item2 in allSelectedProductsPerDay)
            {
                if (item2.ProductId == item.ProductId)
                {
                    item2.SelectedQuantity += item2.Quantity;
                    item2.TotalValue += item2.UnitPrice * item2.Quantity;
                }
            }
            var productTotal = new ProductOrderTable();
            if (item2.SelectedQuantity > 0)
            {
                productTotal.Name = item2.ItemName;
                productTotal.SelectedQuantity = item2.SelectedQuantity;
                productTotal.TotalValue = item2.TotalValue;
            }
            if (productTotal.SelectedQuantity > 0)
            {
                productsTable.Add(productTotal);
            }
        }
    }
    var reportData = new PosClosingReport();
    reportData.OrganizationName = organization.Name;
    reportData.Date = dateForClosing;
    reportData.TotalOrdersCounter = totalOrders.Count;
    reportData.CancelOrdersCounter = cancelOrders.Count;
    reportData.TotalRevenueValue = totalRevenue;
    reportData.EmployeeOrders = employeeTable;
    reportData.Products = productsTable;
    return reportData;
}

```

*Anexa 43, Returnarea datelor pentru raportul de închidere fiscală a casei*

```

[Authorize(Roles = "Admin,POS")]
[HttpGet("pos-closing-fiscal-report")]
public async Task<PosClosingFiscalReport> GetFiscalReportData()
{
    var organizationDetails = await _applicationContext.Organizations.FindAsync(1);
    var date = DateTime.Now;
    DateTime dateForComarasing = new DateTime(date.Year, date.Month, date.Day);
    var finishedOrders = await _applicationContext.Orders.Where(q => q.OrderDate.HasValue &&
        q.OrderDate.Value.Year == dateForComarasing.Year &&
        q.OrderDate.Value.Month == dateForComarasing.Month &&
        q.OrderDate.Value.Day == dateForComarasing.Day && q.OrderStatus == 4).ToListAsync();
    if (finishedOrders.Count > 0)
    {
        double? totalOrdersValue9 = 0;
        double? totalOrdersValue19 = 0;
        foreach (var order in finishedOrders)
        {
            var orderProducts = await _applicationContext.OrderProducts.Where(q => q.OrderId == order.OrderId).ToListAsync();
            if (orderProducts.Count > 0)
            {
                foreach (var product in orderProducts)
                {
                    var productDetails = await _applicationContext.Products.Where(q => q.ProductId == product.ProductId).FirstOrDefaultAsync();
                    if (productDetails.Tva == 9)
                    {
                        totalOrdersValue9 += product.UnitPrice * product.Quantity;
                    }
                    if (productDetails.Tva == 19)
                    {
                        totalOrdersValue19 += product.UnitPrice * product.Quantity;
                    }
                }
            }
        }
        var dataToReturn = new PosClosingFiscalReport
        {
            CompanyName = organizationDetails.Name,
            Adress = organizationDetails.Address,
            City = organizationDetails.City,
            CUI = organizationDetails.Cif,
            CurrentDate = date,
            FinishedOrdersCounter = finishedOrders.Count,
            TotalOrdersValue = totalOrdersValue9 + totalOrdersValue19,
            Total19Tva = totalOrdersValue19 * 0.19,
            Total9Tva = totalOrdersValue9 * 0.09,
        };
        return dataToReturn;
    }
    else
    {
        return new PosClosingFiscalReport();
    }
}

```

Anexa 44, Crearea unui cont pentru angajat

```
[HttpPost("register-employee")]
public async Task<ActionResult> RegisterEmployee(RegisterEmployeeDto model)
{
    string role = "";
    if (model.EmployeeRole == "POS") {
        role = "manager";
    }
    else
    {
        role = model.EmployeeRole;
    }
    var checkEmailExist = await _userManager.Users.AnyAsync(x => x.Email == model.Email.ToLower());
    if (checkEmailExist)
    {
        return BadRequest(new JsonResult(new { message = $"An existing employee is using {model.Email}, email address. Please try with another email address!" }));
    }
    var userToAdd = new User
    {
        FirstName = model.FirstName.ToLower(),
        LastName = model.LastName.ToLower(),
        UserName = model.Email.ToLower(),
        Email = model.Email.ToLower(),
        EmailConfirmed = true,
    };
    var result = await _userManager.CreateAsync(userToAdd, model.Password);
    await _userManager.AddToRoleAsync(userToAdd, model.EmployeeRole);
    if (!result.Succeeded) return BadRequest(result.Errors);
    var userToAddInEmployee = new Employee
    {
        FirstName = model.FirstName.ToLower(),
        LastName = model.LastName.ToLower(),
        Email = model.Email.ToLower(),
        Salary = model.Salary,
        UserId = userToAdd.Id,
        Role = role.ToLower(),
        Lock = false
    };
    _applicationContext.Employees.Add(userToAddInEmployee);
    await _applicationContext.SaveChangesAsync();
    return Ok(new JsonResult(new { message = "A new employee account has been created!" }));
}
```

Anexa 45, Blocarea accesului pentru un client sau angajat

```
[HttpPut("lock-unlock-employee/{userId}/{status}")]
public async Task<IActionResult> LockUnlockUser(int userId, bool status)
{
    var employeeDetails = await _applicationContext.Employees.FindAsync(userId);
    if (employeeDetails != null)
    {
        var user = await _userManager.FindByIdAsync(employeeDetails.UserId);
        if (user != null)
        {
            if (status)
            {
                employeeDetails.Lock = true;
                user.LockoutEnabled = false;
                await _applicationContext.SaveChangesAsync();
                await _userManager.UpdateAsync(user);
                return Ok(new JsonResult(new { message = "User access is blocked now!" }));
            }
            else
            {
                employeeDetails.Lock = false;
                user.LockoutEnabled = true;
                await _applicationContext.SaveChangesAsync();
                await _userManager.UpdateAsync(user);
                return Ok(new JsonResult(new { message = "User access is enabled now!" }));
            }
        }
        return NotFound();
    }
    return NotFound();
}
[HttpPut("lock-unlock-client/{userId}/{status}")]
public async Task<IActionResult> LockUnlockClient(int userId, bool status)
{
    var clientDetails = await _applicationContext.Clients.FindAsync(userId);
    if (clientDetails != null)
    {
        var user = await _userManager.FindByIdAsync(clientDetails.UserId);
        if (user != null)
        {
            if (status)
            {
                clientDetails.Lock = true;
                user.LockoutEnabled = false;
                await _applicationContext.SaveChangesAsync();
                await _userManager.UpdateAsync(user);
                return Ok(new JsonResult(new { message = "User access is blocked now!" }));
            }
            else
            {
                clientDetails.Lock = false;
                user.LockoutEnabled = true;
                await _applicationContext.SaveChangesAsync();
                await _userManager.UpdateAsync(user);
                return Ok(new JsonResult(new { message = "User access is enabled now!" }));
            }
        }
        return NotFound();
    }
    return NotFound();
}
```

## 6.2. Anexele cu codul din frontend

*Anexa 46, Generarea raportului de produse vândute pe o perioadă selectată*

```

    . . .
    . . .

    async generateSoldProductsBetweenDatesPdf(report: PosClosedReport) {
        const logoBase64 = await this.imageService.convertImageToBase64('../assets/images/sbar-high-resolution-logo-black-transparent.png');
        const today = new Date();
        const createdAt = this.sharedService.convertDateToYYYYMMDDHHMMSS(today);
        const startDate = this.sharedService.convertDateToDDMMYY(report.createdAt);
        const endDate = this.sharedService.convertDateToDDMMYY(report.endDate);

        const productsTableBody = [
            {
                text: 'No.', style: 'tableHeader' },
            { text: 'Product Name', style: 'tableHeader' },
            { text: 'Sold Quantity', style: 'tableHeader' },
            { text: 'Sold Value', style: 'tableHeader' },
            { text: '' },
        ];
        report.products.forEach((product, index) => {
            productsTableBody.push([
                {
                    text: (index + 1).toString(),
                    style: ''
                },
                {
                    text: product.name || '',
                    style: ''
                },
                {
                    text: product.soldQuantity != null ? product.soldQuantity.toString() : '',
                    style: ''
                },
                {
                    text: `${product.soldValue != null ? product.soldValue.toFixed(2).toString() : ''} $`,
                    style: ''
                },
                { text: '' }
            ]);
        });
        const docDefinition: any = {
            contents: [
                { image: logoBase64, width: 50, height: 30, alignment: 'right' },
                { text: 'REPORT', style: ['header', 'center'] },
                { text: 'Products sold details', style: ['header', 'center'] },
                { text: `Created date: ${createdAt}`, style: 'normal' },
                { text: `Date period: ${startDate} - ${endDate}`, style: 'normal' },
                { text: 'Products Sold', style: 'subheader' },
                { style: 'table',
                    tables: [
                        {
                            widths: [30, '*', '*', '*'],
                            body: productsTableBody,
                        }
                    ]
                }
            ],
            styles: {
                normal: { fontSize: 14, margin: [0, 5, 0, 5] },
                header: { fontSize: 24, bold: true, margin: [0, 5, 0, 20] },
                center: { alignment: 'center', margin: [0, 5, 0, 5] },
                right: { alignment: 'right', margin: [0, 5, 0, 5] },
                subheader: { fontSize: 18, bold: true, margin: [0, 5, 0, 5] },
                tableHeader: { bold: true, fontSize: 12, color: 'black' },
                table: { margin: [0, 5, 0, 15] },
                defaultStyles: {}
            }
        };
        pdfMake.createPdf(docDefinition).open();
    }
}

```

### Anexa 47, Logica pentru componenta navbar

```

● ● ●

export class NavbarComponent implements OnInit {
  constructor(
    public accountService: AccountService,
    public roles: Roles,
    public ordersService: OrdersService,
    public intervalService: IntervalFuntionsService,
  ) {}
  ngOnInit(): void {
    this.ordersService.getCounter();
  }
  faMinimize = faMinimize;
  faMaximize = faMaximize;
  isFullScreen: boolean = false;

  logout() {
    this.accountService.logout();
    this.ordersService.clearCart();
    this.ordersService.getCounter();
    localStorage.removeItem(environment.orderID);
  }

  enterFullscreen() {
    const elem = document.documentElement as HTMLElement & {
      mozRequestFullScreen?: () => Promise<void>;
      webkitRequestFullscreen?: () => Promise<void>;
      msRequestFullscreen?: () => Promise<void>;
    };

    if (elem.requestFullscreen) {
      elem.requestFullscreen();
      this.isFullScreen = true;
    } else if (elem.mozRequestFullScreen) {
      // Firefox
      elem.mozRequestFullScreen();
      this.isFullScreen = true;
    } else if (elem.webkitRequestFullscreen) {
      // Chrome, Safari, and Opera
      elem.webkitRequestFullscreen();
      this.isFullScreen = true;
    } else if (elem.msRequestFullscreen) {
      // IE/Edge
      elem.msRequestFullscreen();
      this.isFullScreen = true;
    }
  }

  // Method to exit fullscreen
  exitFullscreen() {
    const doc = document as Document & {
      mozCancelFullScreen?: () => Promise<void>;
      webkitExitFullscreen?: () => Promise<void>;
      msExitFullscreen?: () => Promise<void>;
    };

    if (doc.exitFullscreen) {
      doc.exitFullscreen();
      this.isFullScreen = false;
    } else if (doc.mozCancelFullScreen) {
      // Firefox
      doc.mozCancelFullScreen();
      this.isFullScreen = false;
    } else if (doc.webkitExitFullscreen) {
      // Chrome, Safari, and Opera
      doc.webkitExitFullscreen();
      this.isFullScreen = false;
    } else if (doc.msExitFullscreen) {
      // IE/Edge
      doc.msExitFullscreen();
      this.isFullScreen = false;
    }
  }
}

```

Anexa 48, Serviciul pentru rezervările clienților

```
export class ReservationsService {
  constructor(private http: HttpClient) {}

  getUserId() {
    const userDetails = localStorage.getItem(environment.userKey);
    if (userDetails) {
      const user: User = JSON.parse(userDetails);
      return user.userId;
    }
    return null;
  }

  cancelReservation(userId:number,reservationId:number){
    return this.http.delete(` ${environment.appUrl}/api/reservations/delete-
reservation/${userId}/${reservationId}`);
  }

  getReservations() {
    const userId = this.getUserId();
    if (userId) {
      return this.http.get<any>(
        `${environment.appUrl}/api/reservations/reservations-client/${userId}`
      );
    }
    return of(null);
  }

  createReservation(model: CreateReservation, userId: number) {
    return this.http.post(
      `${environment.appUrl}/api/reservations/create-reservation-client/${userId}`,
      model
    );
  }

  clearHistory(userId:number){
    return this.http.delete(` ${environment.appUrl}/api/reservations/clear-reservations-
history/${userId}`)
  }
}
```

### Anexa 49, Serviciul pentru comenziile clienților

```

● ● ●

export class OrdersService {
    constructor(private http: HttpClient) {}

    counter: number = 0;

    getCounter(): {
        const counterString = localStorage.getItem(environment.counterKey);
        if (counterString) {
            this.counter = parseInt(counterString, 10);
        } else {
            this.counter = 0;
        }
    }

    getCartItemsToList(): {
        let list: CartProduct[] = [];
        const storageList = localStorage.getItem(environment.cartKey);
        if (storageList) {
            try {
                list = JSON.parse(storageList);
                if (!Array.isArray(list)) {
                    throw new Error('The parsed object is not an array!');
                }
                return list;
            } catch (error) {
                console.error('It was a error trying to parse the localStorage item to the array!', error);
            }
        }
        return [];
    }

    updateCartCounter(list: CartProduct[]): void {
        list = this.getCartItemsToList();
        if (list.length > 0) {
            const cartCounter: number = list.length;
            const cartCounterString = JSON.stringify(cartCounter);
            localStorage.setItem(environment.counterKey, cartCounterString);
        } else {
            localStorage.setItem(environment.counterKey, '0');
        }
    }

    addCartItemsToLocalStorage(list: CartProduct[]): void {
        const listToAdd = JSON.stringify(list);
        localStorage.setItem(environment.cartKey, listToAdd);
    }

    clearCart(): void {
        localStorage.removeItem(environment.counterKey);
        localStorage.removeItem(environment.cartKey);
    }

    getOrderHistory(userId: number): Observable<Order[]> {
        return this.http.get(`${environment.apiUrl}/api/orders/get-my-orders/${userId}`);
    }

    createNewClientOrder(model: NewClientOrder, userId: number, tableId: number): Observable<any> {
        return this.http.post(`${environment.apiUrl}/api/orders/new-client-order/${userId}/${tableId}`, model);
    }

    getActiveOrder(userId: number): Observable<Order> {
        return this.http.get(`${environment.apiUrl}/api/orders/active-order/${userId}`);
    }

    addNewProductsToOrder(model: NewClientOrder, userId: number, orderId: number, tableId: number): Observable<any> {
        return this.http.post(`${environment.apiUrl}/api/orders/add-new-product-to-order/${userId}/${orderId}/${tableId}`, model);
    }

    finishTheOrder(model: FinishOrderDto, userId: number, orderId: number): Observable<any> {
        return this.http.put(`${environment.apiUrl}/api/orders/order-status-finished/${userId}/${orderId}`, model);
    }
}

```

## Anexa 50, Serviciul pentru POS

```

export class PosService {
    constructor(private http: HttpClient, private modalService: BsModalService, private sharedService: SharedService) {}

    @InjectableRef(): BsModalRef;

    getCartProducts(): CartProduct[] {
        let list: CartProduct[] = [];
        const storageList = localStorage.getItem(environment.cartKey);
        if (storageList) {
            list = JSON.parse(storageList);
        }
        return list;
    }

    getModifyCartProducts(list): CartProduct[] {
        let list: CartProduct[] = [];
        const storageList = localStorage.getItem(environment.modifyCartProducts);
        if (storageList) {
            list = JSON.parse(storageList);
        }
        return list;
    }

    addCartItemsLocalStorage(list: CartProduct[]) {
        const listToAdd = JSON.stringify(list);
        localStorage.setItem(environment.cartKey, listToAdd);
    }

    addModifyCartItemsLocalStorage(list: CartProduct[]) {
        const listToAdd = JSON.stringify(list);
        localStorage.setItem(environment.modifyCartProducts, listToAdd);
    }

    getAllProducts(): Observable<Product> {
        return this.http.get(`${environment.apiUrl}/api/products/get-menu-all`);
    }

    changeTableStatus(model: GetTables): void {
        return this.http.put(`${environment.apiUrl}/api/tables/change-table-status`, model);
    }

    createOrderDeployed(model: CreateEmployeeOrder, userId: number, payedStatus: boolean): void {
        return this.http.post(`${environment.apiUrl}/api/orders/employee-create-order/${model.idEmployeeId}/${payedStatus}`);
    }

    getTableOrderDetail(tableId: number): Observable<Order> {
        return this.http.get(`${environment.apiUrl}/api/orders/get-active-order-by-tables/${tableId}`);
    }

    changeStatusToDelivered(orderId: number, employeeId: number): void {
        return this.http.put(`${environment.apiUrl}/api/orders/order-status-const/${orderId}/${employeeId}`, null);
    }

    changeStatusToFinished(employeeId: number, tips: number): void {
        return this.http.put(`${environment.apiUrl}/api/orders/order-status-finished-employee/${employeeId}/${tips}`, null);
    }

    addNewProductToOrder(orderId: number, model: NewClientOrder): void {
        return this.http.post(`${environment.apiUrl}/api/orders/add-new-product-to-order-employee/${orderId}`, model);
    }

    changeOpenStatus(status: boolean): void {
        return this.http.put(`${environment.apiUrl}/api/pos/change-status-pur${status}`);
    }

    initialDeleteProductModel(orderId: number): void {
        const initialState: ModelOptions = {
            initialState: {
                orderId,
            },
        };
        this.initialRef = this.modalService.showDeleteProductModalComponent(initialState);
    }

    cancelOrder(orderId: number): void {
        return this.http.put(`${environment.apiUrl}/api/orders/order-status-cancel/${orderId}`, null);
    }

    deleteProductFromOrder(orderId: number, productId: number): void {
        return this.http.delete(`${environment.apiUrl}/api/orders/delete-order-product/${orderId}/${productId}`);
    }

    getAllProductStockBalance(): Observable<Product> {
        return this.http.get(`${environment.apiUrl}/api/products/get-all-products-stockbalance`);
    }

    addBalanceForOrder(productId: number, categoryId: number, quantity: number): void {
        return this.http.post(`${environment.apiUrl}/api/pos/new-balance-records/${productId}/${categoryId}/${quantity}`);
    }
}

```

### Anexa 51, Serviciul pentru autorizarea utilizatorilor

```


export class AccountService {
    constructor(private http: HttpClient, private router: Router, private intervalService: IntervalFunctionsService) {}

    private userSource = new ReplaySubject<User>({ initialValue: null }); // we are observing the user
    // so if we log in, we will receive the user
    users$ = this.userSource.asObservable();

    refreshUser(jwt: string | null) {
        if (!jwt === null) {
            this.userSource.next(null);
            return of(undefined);
        }
        // set headers + has jwt in header
        // Headers is Headers.create('Authorization': `Bearer ${jwt}`)
        return this.http.get<User>(`${environment.apiUrl}/api/account/refresh-user-token`).pipe(
            map((user: User) => {
                if (user) {
                    this.setUser(user);
                }
            })
        );
    }

    register(model: Register) {
        return this.http.post(`${environment.apiUrl}/api/account/register`, model);
    }

    login(model: Login) {
        return this.http.post<User>(`${environment.apiUrl}/api/account/login`, model).pipe(
            map((user: User) => {
                if (user) {
                    this.setUser(user);
                }
            })
        );
    }

    logout() {
        localStorage.removeItem(environment.userKey);
        localStorage.clear();
        this.userSource.next(null);
        this.router.navigate(['/']);
        this.intervalService.stopIntervalFunction();
    }

    getJWT() {
        const key = localStorage.getItem(environment.userKey);
        if (key) {
            const user: User = JSON.parse(key); // get the object from the local storage
            return user.jwt;
        }
        return null;
    }

    // get the roles from the jwt token
    jwtHelper = new JwtHelperService();

    getRole(): string {
        let JWT: string;
        const token = localStorage.getItem(environment.userKey);
        if (token) {
            const user: User = JSON.parse(token);
            if (user) {
                JWT = user.jwt;
            } else {
                JWT = null;
            }
            const decodedToken = this.jwtHelper.decodeToken(JWT);
            return decodedToken.roles;
        } else {
            return 'null';
        }
    }

    getUserDetails(userId: number, roleString): Observable<User> {
        return this.http.get(`${environment.apiUrl}/api/account/get-user-details/${userId}/${roleString}`);
    }

    private setUser(user: User) {
        localStorage.setItem(environment.userKey, JSON.stringify(user)); // we are storing the user
        // inside local storage
        this.userSource.next(user); // we are store user information inside our angular application
    }

    changePassword(userId: number, model: ChangePassword) {
        return this.http.post(`${environment.apiUrl}/api/account/change-password/${userId}`, model);
    }
}


```

Anexa 52, Serviciul pentru funcțiile de interval

```
export class IntervalFuntionsService {
    private intervalId: any;
    ordersToConfirm: OrderDto[] = [];

    constructor(private employeeOrderService: EmployeeOrderService) {}
    counter: number = 0;

    getCounter() {
        const counterString = localStorage.getItem(environment.ordersToConfirmCounter);
        if (counterString) {
            this.counter = parseInt(counterString, 10);
            //console.log(this.counter);
        } else {
            this.counter = 0;
            //console.log(this.counter);
        }
    }

    startPeriodicFunction(callback: () => void, intervalTime: number = 10000) {
        if (!this.intervalId) {
            this.intervalId = setInterval(callback, intervalTime);
        }
    }

    stopPeriodicFunction() {
        if (this.intervalId) {
            clearInterval(this.intervalId);
            this.intervalId = null;
        }
    }

    isIntervalRunning(): boolean {
        return !!this.intervalId;
    }

    getAllOrdersToConfirm() {
        this.employeeOrderService.getAllOrdersToConfirm().subscribe({
            next: (response: any[]) => {
                this.ordersToConfirm = response.map((order) => {
                    return {
                        orderId: order.orderId,
                        orderDate: new Date(order.orderDate),
                        tableId: order.tableId,
                        employeeName: order.EmployeeName,
                        status: order.status,
                        products: order.products,
                    };
                });
                if (this.ordersToConfirm.length > 0) {
                    localStorage.setItem(environment.ordersToConfirmCounter,
                        this.ordersToConfirm.length.toString());
                    this.getCounter();
                }
            },
            error: (error) => {
                console.log(error);
            },
        });
    }

    startGettingAllOrders(): void {
        this.startPeriodicFunction(() => {
            this.getAllOrdersToConfirm();
        });
    }
}
```

Anexa 53, Generarea graficelor de pe pagina dashboard

```
createOrdersChart(): void {
    const ctx = document.getElementById('myChart1') as HTMLCanvasElement;
    new Chart(ctx, {
        type: 'line',
        data: {
            labels: this.months,
            datasets: [
                {
                    label: 'Orders number',
                    data: this.ordersCounter,
                    backgroundColor: 'rgba(255, 99, 132, 0.2)',
                    borderColor: 'rgba(255, 99, 132, 1)',
                    borderWidth: 1,
                },
                {
                    label: 'Orders value',
                    data: this.ordersValue,
                    backgroundColor: 'rgba(54, 162, 235, 0.2)',
                    borderColor: 'rgba(54, 162, 235, 1)',
                    borderWidth: 1,
                },
            ],
        },
        options: {
            responsive: true,
            scales: {
                y: {
                    beginAtZero: true,
                },
            },
        },
    });
}

createEmployeesChart(): void {
    const ctx = document.getElementById('myChart2') as HTMLCanvasElement;
    new Chart(ctx, {
        type: 'bar',
        data: {
            labels: this.names,
            datasets: [
                {
                    label: 'Taken orders',
                    data: this.takenOrders,
                    backgroundColor: 'rgba(255, 99, 132, 0.2)',
                    borderColor: 'rgba(255, 99, 132, 1)',
                    borderWidth: 1,
                },
                {
                    label: 'Delivered orders',
                    data: this.deliveredOrders,
                    backgroundColor: 'rgba(54, 162, 235, 0.2)',
                    borderColor: 'rgba(54, 162, 235, 1)',
                    borderWidth: 1,
                },
            ],
        },
        options: {
            responsive: true,
            scales: {
                y: {
                    beginAtZero: true,
                },
            },
        },
    });
}
```

*Anexa 54, Logica pentru afișarea datelor de angajați*

```
● ● ●

export class EmployeesDataComponent implements OnInit {
  constructor(public adminService: AdminService, private sharedService:SharedService) {}

  faEdit = faEdit;
  faLock = faLock;
  faUnlock= faUnlock;
  dtOptions: Config = {};
  dtTrigger: Subject<any> = new Subject<any>();
  employeeData : GetEmployeeDataDto [] = [];

  ngOnInit(): void {
    this.getEmployeesData();
    this.dtOptions = {
      pagingType: 'full_numbers',
    };
  }

  getEmployeesData(){
    this.adminService.getEmployeeData().subscribe({
      next: (response:any) => {
        this.employeeData = response;
        this.dtTrigger.next(null);
        console.log(this.employeeData);

      },
      error: e => {
        console.log(e);
      }
    });
  }

  lockUnlockEmployee(employeeId: number,status:boolean){
    this.adminService.lockUnlockEmployee(employeeId,status).subscribe({
      next: (response:any) => {
        this.sharedService.showNotificationAndReload(true,'Lock status
changed',response.value.message,true);
      }
    });
  }
}
```

Anexa 55, Generarea rapoartelor de către administrator

```
getOrdersBetweenDates(start: string, end: string) {
    let dateStart = new Date();
    let endStart = new Date();
    if (start === '' || end === '') {
        dateStart = new Date('1900-01-01');
        endStart = new Date('1900-01-01');
    } else {
        let validation = false;
        const startDate1 = new Date(start);
        const endDate1 = new Date(end);
        if (endDate1 < startDate1) {
            validation = false;
            this.sharedService.showNotification(false, 'Error', 'Start date is bigger than end date!');
        } else {
            if (endDate1 > new Date() || startDate1 > new Date()) {
                validation = false;
                this.sharedService.showNotification(false, 'Error', 'One of the dates is in the future!');
            } else {
                validation = true;
            }
        }
        if (validation) {
            dateStart = new Date(start);
            endStart = new Date(end);
            this.reportService.generateOrdersBetweenTwoDates(dateStart, endStart);
        }
    }
}

getReservationsBetweenDates(start: string, end: string) {
    let dateStart = new Date();
    let endStart = new Date();
    let validation = false;
    if (start === '' || end === '') {
        dateStart = new Date('1900-01-01');
        endStart = new Date('1900-01-01');
        this.reportService.generateReservationReport(dateStart, endStart);
    } else {
        const startDate1 = new Date(start);
        const endDate1 = new Date(end);

        if (endDate1 < startDate1) {
            validation = false;
            this.sharedService.showNotification(false, 'Error', 'Start date is bigger than end date!');
        } else {
            validation = true;
        }
        if (validation) {
            dateStart = new Date(start);
            endStart = new Date(end);
            this.reportService.generateReservationReport(dateStart, endStart);
        }
    }
}
```

*Anexa 56, Rutarea în aplicație și verificarea rolului utilizatorilor*

```

const routes: Routes = [
    //client paths
    { path: '', component: HomeComponent },
    { path: 'account', loadChildren: () => import('./account/account.module').then((module) =>
        module.AccountModule) }, //lazy loading
    { path: 'menu', loadChildren: () => import('./menu/menu.module').then((module) => module.MenuModule) },
    //lazy loading
    {
        path: 'orders',
        loadChildren: () => import('./orders/orders.module').then((module) => module.OrdersModule),
        canActivate: [RoleGuard],
        data: { roles: [Roles.Admin, Roles.Client] },
    },
    //lazy loading
    {
        path: 'reservations',
        loadChildren: () => import('./reservations/reservations.module').then((module) =>
            module.ReservationsModule),
        canActivate: [RoleGuard],
        data: { roles: [Roles.Admin, Roles.Client] },
    },
    //employee paths
    {
        path: 'employees',
        loadChildren: () => import('./employeeModule/employee.module').then((module) =>
            module.EmployeeModule),
        canActivate: [RoleGuard],
        data: { roles: [Roles.Employee,Roles.Pos,Roles.Admin] },
    },
    //admin paths
    {
        path: 'admin',
        loadChildren: () => import('./admin/admin.module').then((module) => module.AdminModule),
        canActivate: [RoleGuard],
        data: { roles: [Roles.Admin] },
    },
    {
        path: 'account-details',
        component: AccountDetailsComponent,
        canActivate: [RoleGuard],
        data: { roles: [Roles.Admin,Roles.Client,Roles.Employee,Roles.Pos] },
    },
    //shared paths
    { path: 'not-found', component: NotFoundComponent },
    { path: '**', component: NotFoundComponent, pathMatch: 'full' }, //in case someone enters an invalid
path
};

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule],
})
export class AppRoutingModule {}

```

Anexa 57, Serviciul de autorizare

```
export class RoleGuard {  
  
    constructor(private accountService : AccountService, private router : Router, private sharedService: SharedService){}  
  
    canActivate(router: ActivatedRouteSnapshot, state: RouterStateSnapshot):boolean {  
        const requiredRoles = router.data['roles'] as Array<string>;  
        const userRole = this.accountService.getUserRole();  
  
        if(requiredRoles && userRole){  
            if(this.checkRoles(userRole,requiredRoles)){  
                return true;  
            }else{  
                this.sharedService.showNotification(false,'Restricted area','Try better if you want access this page!');  
                this.router.navigate([''], {queryParams: {return: state.url}});  
                return false;  
            }  
        }else {  
            // User roles not available, handle accordingly  
            this.sharedService.showNotification(false,'Restricted area','Try better if you want access this page!');  
            this.router.navigate([''], {queryParams: {return: state.url}});  
            return false;  
        }  
    }  
  
    private checkRoles(userRole:string, requiredRoles:string[] ) : boolean{  
        for (const role of requiredRoles) {  
            if (userRole === role) {  
                return true;  
            }  
        }  
        return false; // Return false only if no matching role is found  
    }  
}
```