

Academia de Studii Economice din București
Facultatea de Cibernetică, Statistică și Informatică Economică
Specializarea Informatică Economică

Aplicație web de gestiune a clienților unei firme de telefonie mobilă

lucrare de licență

Coordonator
Prof. Univ. Dr. Cristian-Eugen CIUREA

Absolvent
Cristian-Mihail DUMITRESCU

București 2022

Declarație privind originalitatea conținutului și asumarea răspunderii

Prin prezență declar că rezultatele prezentate în această lucrare sunt în întregime rezultatul propriei mele creații cu excepția cazului în care se fac referiri la rezultatele altor autori. Confirm faptul că orice material folosit din alte surse (reviste, cărți și site-uri de internet) este în mod clar referit în lucrare și este indicat în lista de referințe bibliografice.

Cuprins

Capitolul 1 - Introducere.....	1
Capitolul 2 - Aplicație web de gestiune a clienților unei firme de telefonie mobilă	3
Capitolul 2.1 - Problema abordată	3
Capitolul 2.2 - Utilitatea soluției	4
Capitolul 2.3 - Modul de instalare, rulare și utilizare.....	5
Capitolul 2.4 - Alternativa „Postgresql” pentru baza de date	8
Capitolul 3 - Tehnologii/Metode utilizate	10
Capitolul 3.1 - Tehnologii utilizate	10
Capitolul 3.2 - Metode de securitate utilizate	11
Capitolul 3.3 - Alte metode utilizate	12
Capitolul 4 - Arhitectura soluției.....	15
Capitolul 4.1 - Descrierea componentelor utilizate	15
Capitolul 4.2 - Interacțiunile componentelor	16
Capitolul 4.3 - Tehnologii pentru crearea de grafice	18
Capitolul 5 - Implementarea soluției.....	21
Capitolul 5.1 - Schema bazei de date.....	21
Capitolul 5.2 - Serverului <i>API</i>	26
Capitolul 5.3 - WebGUI.....	27
Capitolul 5.4 - DesktopGUI.....	29
Capitolul 5.5 - Programare prin evenimente.....	30
Capitolul 5.6 - Programare orientată obiect.....	31
Capitolul 5.7 - Baza de date	31
Capitolul 6 - Concluzii	33
Bibliografie	34
Anexa 1 – Lista de figuri	36
Anexa 2 – Lista de acronime	37
Anexa 3 – Cod sursă al aplicației.....	38

Capitolul 1 - Introducere

După cum spune și titlul, obiectivul lucrării este de a realiza o aplicație web de gestiune a clienților unei firme de telefonie mobilă. Această gestiune, însă, implică mai multe elemente, nu doar înregistrarea clienților și permiterea acestora să își modifice detaliile produselor cumpărate.

Unul dintre aceste elemente este, desigur, crearea unui sistem funcțional, eficient și ușor de utilizat.

De asemenea, documentarea aplicației prin comentarii și/sau documente externe, alături de transparenta codului și ușurința de actualizare reprezintă elemente importante.

Importanța este dată de faptul că, în zilele noastre, nu oricine este dispus să meargă în persoană oriunde (mai ales în cea mai recentă perioadă) și să stea la cozi ce par interminabile. Cu cât înaintăm în cel de-al treilea mileniu, nevoia de digitalizare crește. Orice firmă trebuie să se poată adapta la noile cerințe impuse de consumatori.

Nevoia de reclamă a serviciilor pe internet, de asemenea, crește, iar orice firmă ce se respectă dispune de un site de prezentare. Multe firme permit clienților o interacțiune prin aplicații web. Este important, însă, pentru orice firmă să își stocheze datele într-un mod sigur. De asemenea, orice firmă va deriva valoare din statistici create pe astfel de date. Există posibilitatea folosirii datelor despre client în mod constructiv, iar pentru salariații firmei există posibilitatea de a fi utilizate pentru a oferi de exemplu prime sau promovări.

Așadar, importanța unui sistem informatic bine pus la punct și ce permite o interacțiune ușoară cu clientul este un mare plus pentru orice firmă.

O cerință simplă a problemei este: Crearea unui sistem de management al unei firme de telefonie mobilă. Acest sistem trebuie să țină cont atât de clienții din mediul online, cât și de cei ce ajung la ghișee. Astfel o modalitate de separare a celor două situații este necesară. De asemenea, atât administratorii, cât și managerii trebuie să poată monitoriza și modifica date despre clienți, abonamentele lor și extra opțiunile alese. Administratorii au nevoie de acces, de tip modificare și ștergere, la sistemul de conturi, clienți, abonamente și extra opțiuni. Siguranța datelor este de importanță, în acest sistem, reputația firmei fiind afectată în cazul unei breșe. Așadar, metode actualizate de criptare a datelor și metode cât mai sigure de transmisie sunt necesare.

Avantajele unui sistem programat „in-house” pentru o firmă sunt mari, acest sistem permițând o versatilitate și siguranța mult mai mare decât un sistem cumpărat de la o firmă de tip „third-party”. Datele despre clienți trebuie transmise într-un mod sigur, iar datele sensibile (exemplu: parole) trebuie stocate fără drept de vizualizare (prin algoritmi de hashing și modalități de potrivire al hash-ului).

Însă, un astfel de sistem obligă firma să mențină o echipa de tip IT, atât pentru mentenanța hardware-ului pe care acesta rulează, cât și pentru menținerea codului sursă. Acest detaliu este considerat un dezavantaj.

Menținerea codului sursă implică mai multe obiective:

- Se dorește ca acest cod să rămână cât mai bine documentat și cât mai ușor de înțeles. De asemenea, bibliotecile și dependențele folosite trebuie documentate și supravegheate pentru potențiale breșe.
- Se dorește ca toate bibliotecile și dependențele folosite să fie la cea mai nouă versiune, pentru a se asigura securitatea datelor. În cazul unei biblioteci ce nu mai primește actualizări, se recomandă o dezvoltare a unui software similar, ce permite astfel actualizarea în caz de breșe.

Capitolul 2 - Aplicație web de gestiune a clienților unei firme de telefonie mobilă

Întreaga lucrare este delimitată în capitole și subcapitole cu titluri cât mai semnificative și ușor de înțeles.

De asemenea, toate referințele, utilizate și menționate, sunt dispuse în **Bibliografie**, alături de lista de ilustrații din **Anexa 1 – Lista de figuri**. Lucrarea este bazată pe informațiile preluate din documentații, cărți și însușite de către mine în anii de facultate.

Multe dintre informațiile necesare înțelegerii amănunțite a lucrării se regăsesc în referințele din **Bibliografie**, însă, citirea întregilor documentații nu este necesară pentru o înțelegere sumară a acesteia.

Ilustrațiile sunt folosite pentru a-i permite cititorului o înțelegere mai ușoară a conceptelor utilizate și a modului de funcționare a sistemului. Astfel de ilustrații sunt folosite în textul lucrării și indexate în **Anexa 1 – Lista de figuri**.

În decursul lucrării, limbajul folosit este unul de specialitate, tematica lucrării fiind cea a tehnologiei informației, programării web și programării desktop. Mulți termeni sunt definiți în decursul lucrării, în special cei creați pentru a defini modul de lucru folosit sau modul de organizare al lucrării. Acești termeni sunt, de asemenea, menționați și definiți în **Anexa 2 - Lista de acronime** și însemnați la nivelul textului cu format *italic*.

În acest capitol, mai jos, se discută despre subiectul/problema lucrării, utilitatea adusă de soluția implementată cât și despre modul în care cineva are posibilitatea să ruleze proiectul, atât pentru dezvoltare cât și pentru utilizare.

Capitolul 2.1 - Problema abordată

Inițial proiectul se concentra pe modul de procesare al clienților din cadrul web al unei firme de telefonie mobilă. În timp, acesta a evoluat într-un întreg sistem de gestiune, prin observarea unor dependențe a elementului web, de date venite din interiorul firmei.

După începerea proiectului, s-a putut observa că, într-un scenariu real, multe dintre datele de care ar fi nevoie pentru rularea aplicației sunt date, în realitate, de un manager sau un administrator. S-a decis, astfel, să se modifice ideea inițială a proiectului, pentru a crea o aplicație mai interesantă și intuitivă.

Așadar, în final, problema abordată de lucrare este cea a creării unui întreg sistem de gestiune atât al clienților, cât și al angajaților unei firme de telefonie mobilă. Sistemul trebuie să permită

clienților atât înregistrarea și modificarea abonamentelor, atât de la ghișee, cât și de pe internet. Sistemul și codul aferent acestuia trebuie să fie adaptabil și ușor de citit, permițând extinderea firmei în viitor.

De asemenea, atât administratorii, cât și managerii trebuie să poată monitoriza și modifica date despre clienți, abonamentele lor și extra opțiunilor alese. Administratorii au nevoie de acces, de tip modificare și ștergere, la sistemul de conturi, clienți, abonamente și extra opțiuni.

Securitatea datelor și a transmisiei acestora este importantă. Clienții firmei nu doresc facturi mai mari sau, mai rău, furt de identitate sau carduri, din cauza unui sistem conceput greșit sau vulnerabil. Așadar, firma dorește să evite astfel de situații pentru a-și apăra renumele, reputația și a nu fi prinsă în mijlocul unui scandal.

Alegerea temei lucrării nu a fost una ușoară, preferând o tematică ce îmi permite o abordare prin mai multe tehnologii și o necesitate de a studia metode și modalități noi și interesante, în încercarea de a fi cât mai actual cu cele scrise. De asemenea, tema permite un studiu în mai mare detaliu al metodelor și tehnologiilor alese; tehnologii studiate, de asemenea, în timpul anilor de studiu.

Capitolul 2.2 - Utilitatea soluției

În acest subcapitol se argumentează modul în care soluția creată este utilă pentru o firmă, din cât mai multe puncte de vedere.

Necesitatea și utilitatea unei soluții informatice pentru o problemă este dată în cultura vestică de beneficiul financiar al acesteia, însă acesta nu este singurul motiv: există și un beneficiu de îmbunătățire a imaginii firmei, dată de astfel de soluții, mai ales când acestea au legătură cu clientela. În aceste condiții, necesitatea și utilitatea soluției mele este evidentă: aducerea de noi clienți ce doresc o interacțiune cât mai simplă cu firma și îmbunătățirea reputației firmei.

Clienții doresc confortul rezolvării problemelor prin intermediul internetului, în loc de învechitul așteptat la cozi și ghișee. Însă nevoia de ghișee nu este încă zero, generațiile de vârstnici, mai ales, preferând-o.

Firmele ce doresc o competitivitate crescută vor opta pentru a-și integra astfel de servicii online în modelul lor, clienții acestora fiind potențial mai mulțumiți cu serviciile firmei.

Inovația și modul în care firmele își tratează clientela, sunt factori importanți ce influențează succesul acestora. Astfel, un sistem intern ce mulțumește toți clienții este bine venit și extrem de util pentru acestea. Un astfel de sistem, pe termen lung, contribuie la micșorarea costurilor firmei,

sucursalele nemaifiind la fel de utilizate și nemaivând nevoie de spațiu, personal și logistică; toate extrem de costisitoare pentru o firmă.

Actualitatea este dată de modalitatea de rezolvare a problemei, din punct de vedere al securității și al nivelului de simplitate de înțelegere al codului. Nevoia de modalități actualizate de securizare a datelor este dată chiar de cerința problemei și de ideea implementării soluției într-o situație reală, iar nevoia de ușurință a citirii codului este evidentă: pentru a fi ușor de îmbunătățit și modificat.

Capitolul 2.3 - Modul de instalare, rulare și utilizare

În acest subcapitol se explică, în cât mai mare detaliu, modul de instalare, rulare și utilizare a sistemului conceput. Acest capitol este folositor atât pentru dezvoltarea ulterioară a sistemului, cât și pentru folosirea acestuia.

Pentru a nu avea un întreg discurs separat despre modul de utilizare și funcționare a software-ului de control al versiunilor, deoarece nu este scopul lucrării, se presupune că cititorul este, deja, în posesia unei arhive cu proiectul sau are posibilitatea de a accesa într-o formă un-arhivată.

Pentru a fi cât mai exact, voi desparti acest subcapitol în două părți logice:

1. instalare - rulare sistem pentru dezvoltare;

2. instalare - rulare sistem pentru utilizare;

1. Pentru dezvoltare: Este nevoie de o mașină cu un sistem de operare de tip **Windows**. Pe aceasta vor fi necesare câteva programe instalate pentru a fi sigur că putem rula sistemul. De asemenea, specificațiile mașinii trebuie să fie relativ ridicate. Detaliile tehnice legate de specificațiile hardware nu sunt importante, acestea nefiind scopul lucrării; multe cerințe minime/recomandate pentru software se regăsesc pe site-urile programelor menționate mai jos și în resursele menționate în *Bibliografie*.

Este necesar un mediu de dezvoltare pentru limbajul C#. Personal am ales **Visual Studio** [1]. De asemenea, este nevoie de o modalitate ușoară de a instala dependențele necesare rulării a părții web a proiectului. Pentru asta, se va instala utilitarul **Node** [2] și de instalarea unui program ce ne permite modificarea bazei de date folosite are posibilitatea de a fi de ajutor (**Sqlite3** [3], inițial).

Ca editor pentru aplicația web, **WebGUI** și serverul **API**, am utilizat **Visual Studio Code** [4], însă există posibilitatea de a utiliza orice altul, în combinație cu o consolă **Windows** pentru rulare.

Din acest moment, se poate descărca o arhivă cu codul sursă al sistemului și dezarhiva codul, în cazul în care nu există deja o arhivă cu codul.

Ca ultim pas către rulare, se consultă fișierele READ.me de la nivelul aplicației web, *WebGUI* și serverului *API*. Acolo, discut despre instrucțiunea „*npm install*” ce trebuie rulată pentru instalarea dependențelor necesare.

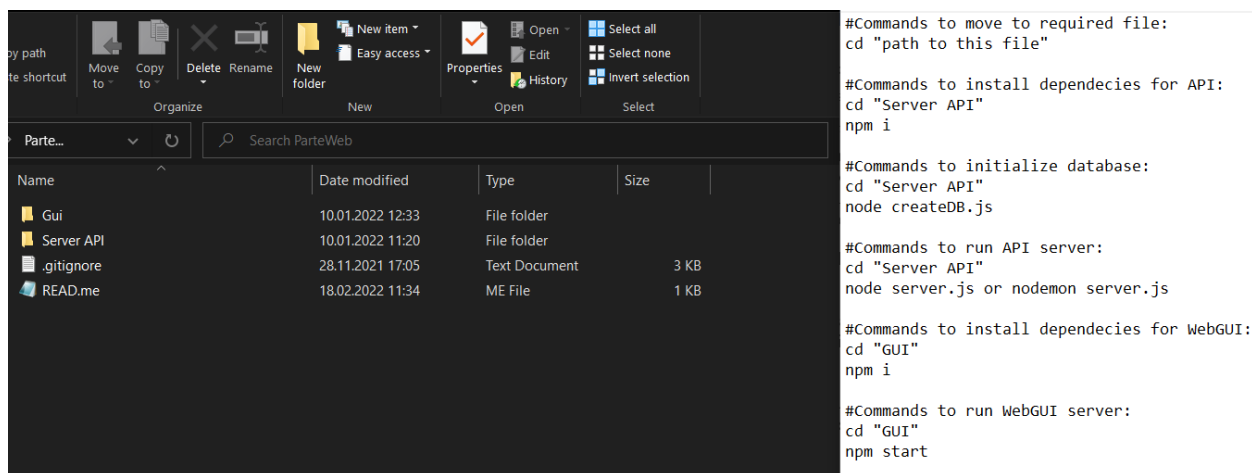


Figura 01. Captură de ecran cu un fișier de tip READ.me.

Mai jos, la punctul **2. instalare - rulare sistem pentru utilizare**; se explică modul de rulare/utilizare, unde există posibilitatea de a folosi sistemul și pe o singură mașină, pentru dezvoltare.

2. Pentru utilizare: Există posibilitatea de a fi nevoie de un număr mai mare de mașini de tip server, unde serverul *API* și cel web (pentru aplicația *WebGUI*) își desfășoară activitatea. De asemenea, ideea aplicației *DesktopGUI* era să fie rulată pe computerele de la front-desk-ul firmei.

Așadar, numărul de mașini ce au nevoie de sistem de operare windows, este direct proporțional cu numărul de sucursale ale firmei, din punct de vedere „**business**”. Acest număr are posibilitatea de a fi însă redus, în timp; mai multe persoane optând pentru a-și gestiona abonamentele online.

Pentru serverele web, există posibilitatea de a fi folosite orice sisteme de operare, atât timp cât există posibilitatea de a instala pe acestea utilitarul *Node* [2] și dependențele proiectelor.

Există posibilitatea de a se folosi soluții de **hosting** web pentru a susține serverele de back-end general și front-end al site-ului web. Acest fel de soluții permițând și o extindere ușoară a capacității serverelor.

Abordarea implementării totale „**in-house**” necesită, de asemenea, dezvoltarea unei rețele interne, ce permite apeluri la serverul *API*. Această abordare este mai greu de abordat și cere mult mai multă implicare directă (probabil chiar crearea unei echipe interne de specialiști vor rezolva atât

probleme ce apar în cadrul rețelei, cât și probleme de securitate), dar este mai sigură ca un „**out-sourceing**” al resurselor hardware necesare serverelor, atunci când este implementată corect.

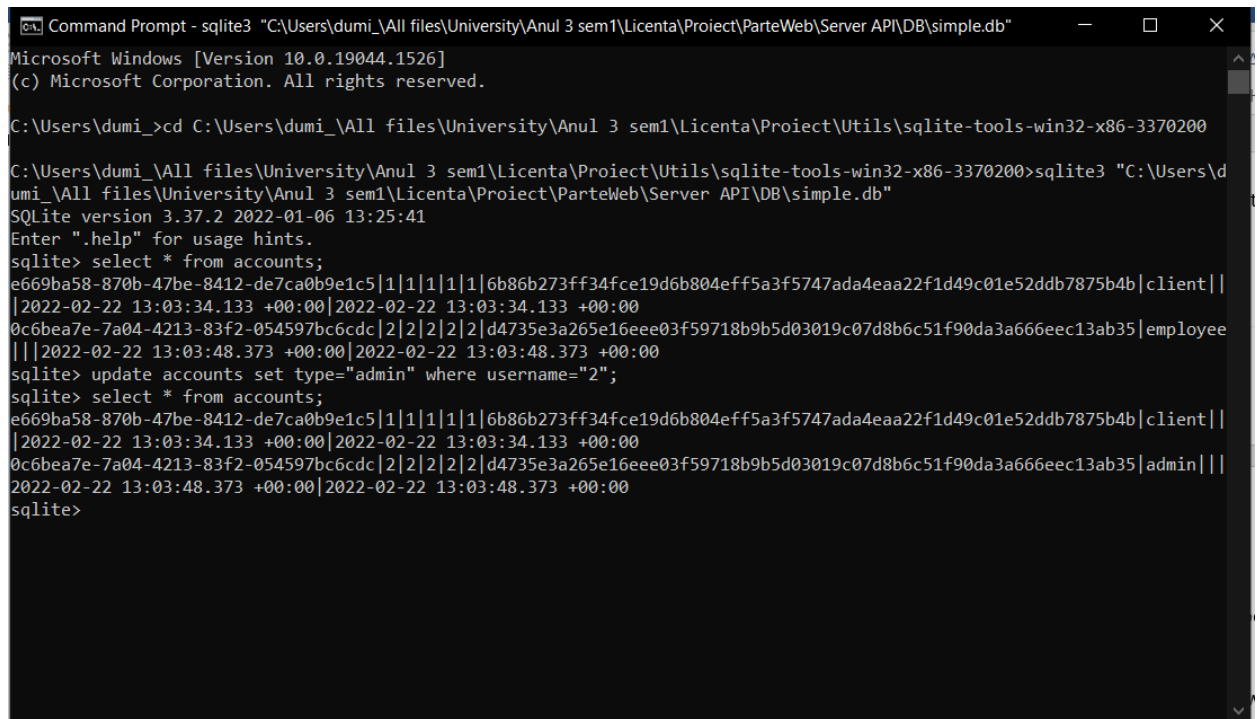
În final, după alegerea unei abordări, rularea concomitentă a programelor și crearea unor utilizatori va fi abordată după cum este descris la punctul: **1. instalare - rulare sistem pentru dezvoltare**; din acest subcapitol.

După realizarea tuturor celor spuse mai sus, este necesară ridicarea unor utilizatori, pentru aplicația *DesktopGUI*, la rang-ul de administratori, pentru a putea administra restul conturilor: pentru clienți, angajați, manageri și chiar alți administratori.

Utilitarul folosit in descriere este găsit în: *%projectRoot%\Utils\sqlite*

De aici se rulează comanda „*Sqlite3*” [3] pe ruta până la baza de date. Ca exemplu de comandă, se va realiza modificarea tipului utilizatorului cu numele de utilizator „2” în „**admin**”.

Mai jos se află comanda rulată:



```
Command Prompt - sqlite3 "C:\Users\dumi_\All files\University\Anul 3 sem1\Licenta\Proiect\ParteWeb\Server API\DB\simple.db"
Microsoft Windows [Version 10.0.19044.1526]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dumi_>cd C:\Users\dumi_\All files\University\Anul 3 sem1\Licenta\Proiect\Utils\sqlite-tools-win32-x86-3370200

C:\Users\dumi_\All files\University\Anul 3 sem1\Licenta\Proiect\Utils\sqlite-tools-win32-x86-3370200>sqlite3 "C:\Users\dumi_\All files\University\Anul 3 sem1\Licenta\Proiect\ParteWeb\Server API\DB\simple.db"
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.
sqlite> select * from accounts;
e669ba58-870b-47be-8412-de7ca0b9e1c5|1|1|1|1|6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b|client||
|2022-02-22 13:03:34.133 +00:00|2022-02-22 13:03:34.133 +00:00
0c6bea7e-7a04-4213-83f2-054597bc6cdc|2|2|2|2|d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35|employee
||2022-02-22 13:03:48.373 +00:00|2022-02-22 13:03:48.373 +00:00
sqlite> update accounts set type="admin" where username="2";
sqlite> select * from accounts;
e669ba58-870b-47be-8412-de7ca0b9e1c5|1|1|1|1|6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b|client||
|2022-02-22 13:03:34.133 +00:00|2022-02-22 13:03:34.133 +00:00
0c6bea7e-7a04-4213-83f2-054597bc6cdc|2|2|2|2|d4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35|admin||
|2022-02-22 13:03:48.373 +00:00|2022-02-22 13:03:48.373 +00:00
sqlite>
```

Figura 02. Modificare tip utilizator direct în baza de date.

Din acest moment, atât pentru rulare cât și pentru dezvoltare, sistemul este funcțional. Atât clienții, cât și angajații au posibilitatea să își realizeze operațiunile fără probleme.

Desigur, din punct de vedere „**business**” și chiar al securității, există posibilitatea de a fi efectuate modificări asupra infrastructurii propuse, însă acestea ar duce la o nevoie de rescriere a unei mari părți a codului sau regândirea întregului sistem.

De asemenea, o documentare mai amănunțită a întregului proiect și probabil, crearea unor reguli scrise de modificare sau îmbunătățire sunt necesare în cazul unei utilizări la scală mai largă.

Capitolul 2.4 - Alternativa „Postgresql” pentru baza de date

O alternativă, pentru producție, la baza de date este o bază de date de tip „*Postgresql*” [5]. În proiect, prin intermediul bibliotecii „*Sequelize*” [6], este banal să se schimbe legătura între serverul *API* și baza de date.

O bază de date de tip „*Postgresql*” [5] se află la distanță și este mai ușor de scalat. La nivelul codului, schimbarea se face în felul următor: la nivelul driverului de baza de date se comentează sau de-comentează codul respectiv și prin instalarea driverul necesar:



```
JS DB_driver.js M ●
Server API > DB > JS DB_driver.js > ...
1  const Sequelize = require("sequelize");
2  //database config
3  // const sequelize = new Sequelize({
4  //   dialect: 'sqlite',
5  //   storage: './DB/simple.db'
6  // });
7
8
9  const sequelize = new Sequelize({
10     host: 'localhost',
11     port: 5432,
12     database: 'licenta',
13     dialect: 'postgres',
14     username: 'postgres',
15     password: 'admin'
16   });
17
18
19   module.exports = sequelize;
20
```

Figura 03. Modificare necesare în „driver-ul” bazei de date.

După rularea comenzii de creare (identică cu cea folosită mai sus la *Capitolul 2.3*), tabelele de mai jos ne vor apărea în interfața grafică a bazei de date:

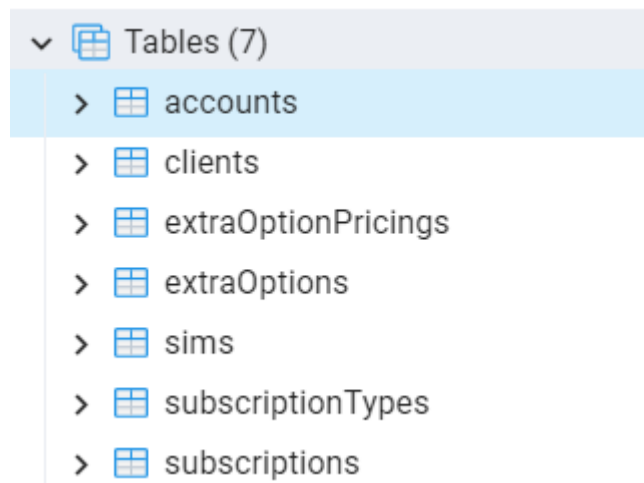


Figura 04. Tabelele in baza de date de tip Postgresql.

De asemenea, modificarea privilegiilor primului utilizator, în „admin”, trebuie făcută prin comanda *SQL*. Aceasta este dată atât în linie de comandă, cât și în interfața grafică a sistemului de gestiune al bazei de date.

Capitolul 3 - Tehnologii/Metode utilizate

În acest capitol au fost detaliate, în linii mari, tehnologiile și metodele utilizate în proiect. Multe dintre acestea sunt detaliate în **Capitolul 5-Implementarea soluției**.

Acestea sunt expuse într-o manieră liniară, sub formă de listă, cu doar mici explicații cu privire la modul de utilizare a acestora, pentru a realiza o imagine de ansamblu asupra proiectului și a permite cititorului să își creeze o idee de ansamblu despre modul de implementare al soluției și de funcționare al întregului sistem. Ambele subiecte sunt tratate, în amănunt, în capitolele următoare: **Capitolul 4** și **Capitolul 5**.

În cazul în care termenii folosiți pentru descrierea componentelor sistemului sunt prea abstracți, se propune consultarea **Capitolului 4.1 - Descrierea componentelor utilizate** și **Anexa 2 - Lista de acronime**.

Capitolul 3.1 - Tehnologii utilizate

Mai jos s-a adăugat o listă cu tehnologiile utilizate în decursul proiectului și locația în care acestea au fost folosite:

1. pentru baza de date, *BD*, s-a utilizat „**Sqlite3**” [3] (ce are posibilitatea de a fi înlocuită în producție) și două drivere aferente. Unul explicit, menționat mai sus în **Capitolul 2.3-Modul de instalare, rulare și utilizare**, în partea a doua (cea legată de modul de utilizare) și unul implicit, instalat prin intermediul comenzilor „**npm**” (utilitar instalat alături de **Node** [2]), la nivelul serverului *API*;

2. pentru partea de back-end, a aplicației web și desktop – **serverul API** și partea de front-end, a aplicației web - **WebGUI**, s-au folosit limbajele: *HTML*, *CSS* și *JS*. Aplicația fiind de tip „**web-app**”, crearea acesteia implică utilizarea limbajelor ce intră în compoziția „**web-stack**”. *HTML* și *JS* nu sunt utilizate în mod direct, independent în partea de front-end a aplicației web, acestea fiind folosite într-un mod cumulat. În partea de back-end a ambelor componente, *JS* este utilizat;

3. pentru partea de back-end, a aplicației web și desktop – **serverul API**, s-au folosit, ca biblioteci:

- „**Express**”; pentru a putea dezvolta o aplicație de forma *API* și pentru abstractizarea interacțiunii *WebGUI*, *DesktopGUI* cu *DB*. Pentru o idee de ansamblu a comunicării aplicațiilor vezi: **Figura 06** și **Capitolul 4-Aritectura**;
- „**Sequelize**” [6]; pentru a abstractiza interacțiunea *API* cu *DB*;
- „**Cors**” [7]; pentru a putea realiza apeluri de la aplicații la *API*;
- „**Crypto-JS**”; pentru a păstra parolele utilizatorilor într-o modalitate sigură;

- „*Sqlite3*” [3]; ca bibliotecă pentru a crea o bază de date specifică, ce are posibilitatea, însă, de a fi înlocuită;
4. pentru partea de front-end, a aplicației web - **WebGUI**, s-au folosit, ca biblioteci: „*React*” [8], „*Redux*” [9] și „*redux-promise-middleware*”; pentru a fi mai ușor de implementat legătura cu *API*-ul și a ține mai ușor sub observație starea părții de front-end, a aplicației web;
5. pentru partea de front-end, a aplicației desktop - **DesktopGUI** s-au folosit, ca limbaje: C#, alături de framework-ul „*winforms*” [10] pentru construirea unui model de aplicație bazată pe evenimente. Acest model este descris pe larg în **Capitolul 5.5-Programare prin evenimente**;
6. pentru partea de front-end, a aplicației desktop - **DesktopGUI**, s-au folosit, ca biblioteci: „*Newtonsoft.Json*” [11]; pentru a realiza o despachetare mai ușoară a datelor, în format *JSON*, ce provin de la *API*. Această bibliotecă a fost utilizată doar în scopul de a ușura utilizarea, chiar documentația **Microsoft** pentru C# spunând că „**System.Text.Json**” nu are încă toate funcționalitățile bibliotecii, fiind nevoie de „workaround-uri” sau pur și simplu lipsesc, nefiind suportate (la momentul scrierii).

Pentru mai multe detalii despre modul în care s-au utilizat aceste tehnologii se va consulta codul scris și **Capitolul 5-Implementarea soluției. Capitolul 2.3-Modul de instalare rulare și utilizare** este, de asemenea, de ajutor. În plus, **Capitolul 4-Arhitectura soluției**, aduce o nouă perspectivă, de nivel superior, asupra întregului proiect.

Capitolul 3.2 - Metode de securitate utilizate

La nivelul proiectelor, s-au implementat mai multe metode de securitate cât mai actuale:

1. În primul rând, deoarece atât clienții cât și angajații au nevoie de conturi, acestea trebuie păstrate în siguranță.

În această privință, trebuie să avem grija la atacuri de tip „*data dump*” din cauza unei „*injecții SQL*”, în care datele acestor conturi sunt furate de către atacatori.

Pentru a preveni acest tip de atacuri, se folosesc framework-uri și biblioteci cât mai actuale. Atât „*React*” [8] cât și „*Sequelize*” [6] nu permit ca astfel de atacuri să fie realizate. „*Sequelize*” [6] în particular, curăță input-urile.

Pe lângă acestea, s-au implementat câteva metode de securitate. Parolele utilizatorilor nu sunt salvate în baza de date sub format text, ci acestea sunt salvate sub formă de „*hash*” (deducerea unui sir de biți unici dintr-un anumit șir de caractere, pentru șirul respectiv), realizat prin algoritmul **sha256**.

Parolele introduse, la intrarea în aplicație, sunt trecute prin algoritmul respectiv și comparate cu valorile din baza de date. Ca o adăugare: o astfel de metodă este de asemenea folosită în „infrastructura de chei publice”. Pentru utilizare, protocolului *HTTPS* [12] și a certificatelor aferente, la o implementare reală a proiectului, este necesară o investiție într-un astfel de certificat de la o firmă ce le distribuie. Fără așa ceva, datele transmise prin intermediul internetului nu sunt criptate și tot sistemul va fi atacat.

Codul aferent acestor metode se află în *Anexa 3 – Cod sursă al aplicației*, la *Codul sursă 1*.

Metodele, menționate mai sus, nu permit indivizilor determinați să preia date sensibile și să le citească ușor, însă datoria fiecărui utilizator va rămâne să își creeze o parola puternică, deoarece în cazul nefericit al unei breșe, cei ce pun mana pe date vor putea folosi metode de spargere de tip dicționar și forța brută. Această metodă de spargere, folosește un dicționar cu parole deja sparte și unul cu parole des utilizate/ușor de creat pe care le trece prin algoritmul de „*hash-ing*”, pentru a afla parolele aferente utilizatorilor.

2. Fiecărui utilizator îi este asociat un „*token*” (și de caractere asociat pentru o perioadă de timp contului respectiv), pentru a nu transmite la fiecare apel către server, numele și parola contului.

Acest „*token*” are un timp, aferent acestuia, în care este activ. Timpul este mai mare sau mai mic, în funcție de cerințele de „*business*”.

De asemenea, există posibilitatea de a implementa o metodă de extindere a timpului, pe care partea de front-să o apeleze la o anumită perioadă.

Capitolul 3.3 - Alte metode utilizate

Mai jos, se descriu alte metode interesante, folosite. Aici se discută metodele abordate pentru a rezolva anumite probleme:

1. Pentru serverul *API*, este nevoie de o modalitate de creare și rulare a acestuia. Aici se apelează la biblioteca „*Express*” [13]. La nivelul acesta, este nevoie de o modalitate de separare a codului, astfel încât modificările să fie ușor de realizat. O astfel de segregare este dată de rute (routes), cu ajutorul cărora, putem separa, în fișiere, fiecare entitate manipulată de cod. Aceste entități sunt separate și la nivelul apelurilor prin rute separate de „/”.

Structura astfel creată, este similară logic cu cea din baza de date, fiecare tabel având un fișier asociat în serverul *API*, ce corespunde acțiunilor ce au posibilitatea de a fi realizate asupra respectivei tabel.

2. Pentru serverul *WebGUI*, s-a folosit biblioteca „*React*” [8], pentru a-l realiza. Pentru o ușurință în transmiterea variabilelor specifice sesiunii, s-a optat de asemenea pentru utilizarea bibliotecii „*Redux*” [9]. Această componentă are o structură de tipul „single page app”, pentru a nu mai complica situația cu ideea de rute pentru aplicația vizuala, în browser.

De asemenea, componentele sunt structurate în directoare pentru a fi mai ușoară accesarea acestora.

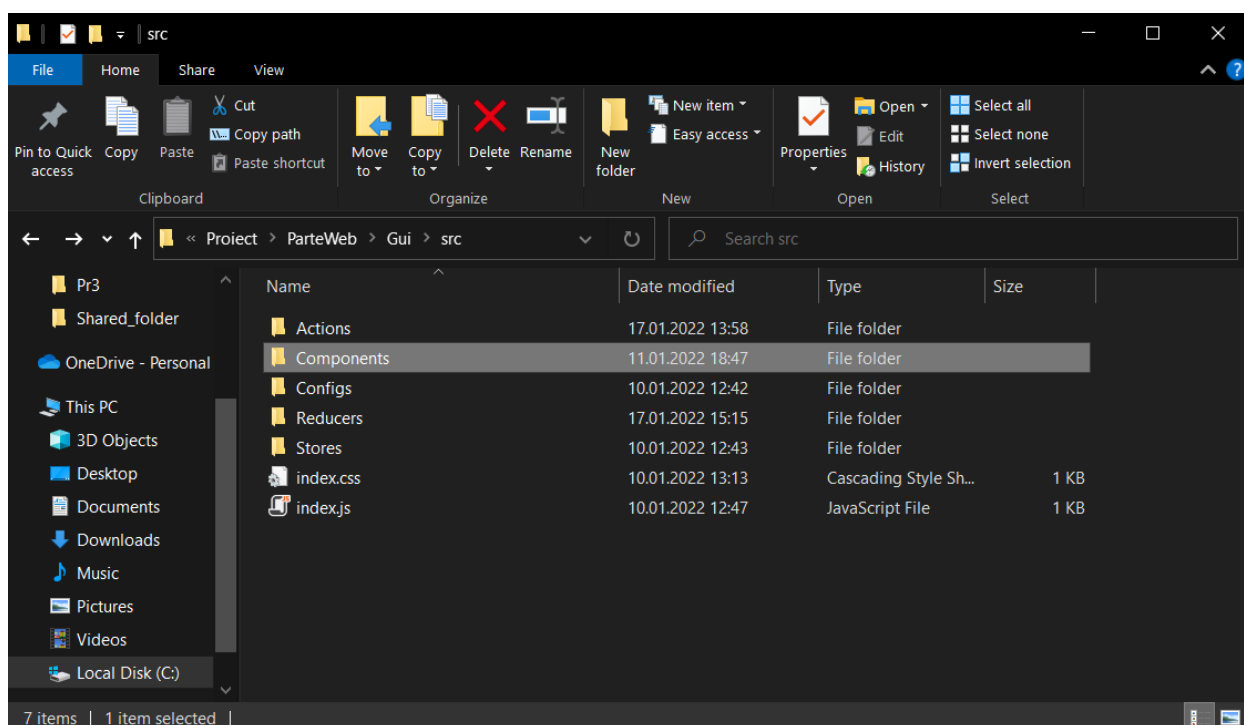


Figura 05. Structura de fișiere a aplicației WebGUI.

Și elementele din directorul „Components” au posibilitatea de a fi structurate astfel, însă pentru acest lucru ar trebui o restructurare a codului aplicației.

3. Pentru aplicația DesktopGUI, s-au folosit de concepte atât ale programării orientate obiecte, detaliate în *Capitolul 5.6-Programare orientată obiect*, cât și ale programării orientată pe evenimente, detaliată în *Capitolul 5.5-Programare prin evenimente*, pentru a

putea realiza o componentă vizuală cu o structură a codului cât mai ușor de urmărit și modificat.

De asemenea, la nivelul componentei, au fost necesare prelucrări de date primite de la serverul *API*, sub format *JSON*. Pentru această prelucrare s-au folosit, după cum am menționat și mai sus, o bibliotecă – *Newtonsoft.Json* [11].

Capitolul 4 - Arhitectura soluției

Soluția realizată este împărțită în mai multe componente. Mai jos se creionează o descriere amănunțită a tot ce s-a folosit și modul de interconectare. Pentru o înțelegere mai ușoară a acesteia, recomand consultarea **Figurii 06**, prezentată mai jos.

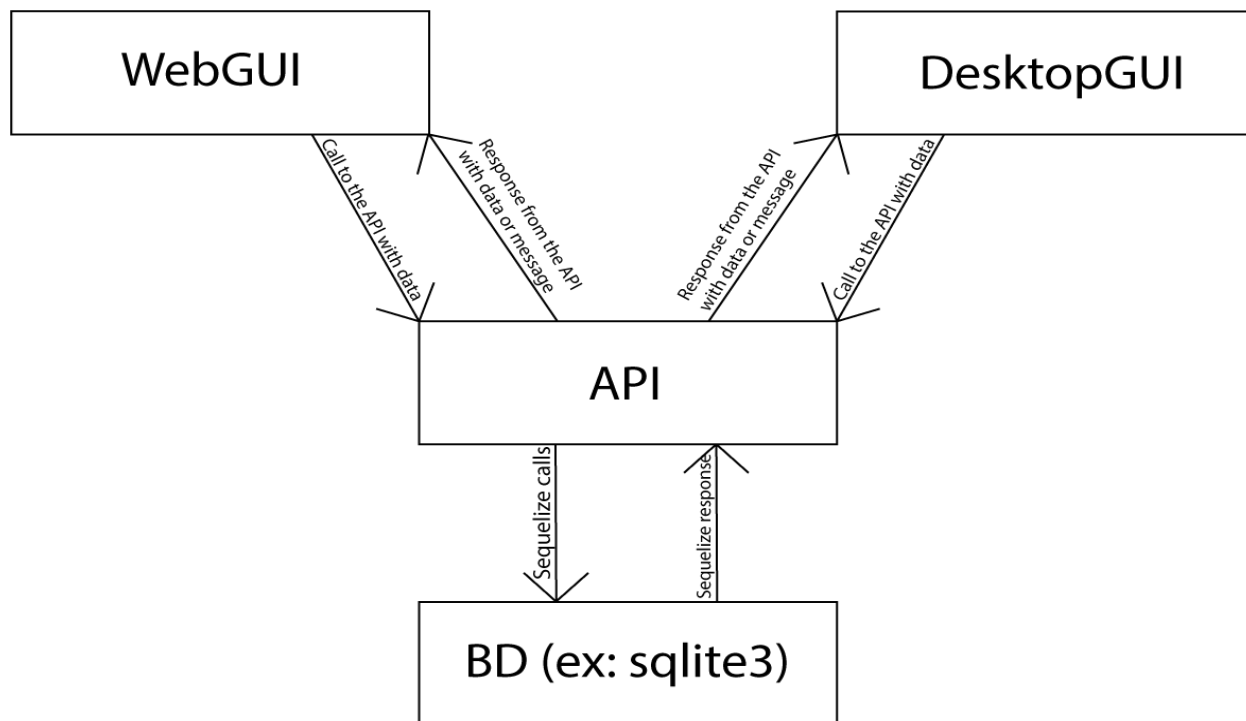


Figura 06. Schema a arhitecturii generală ale proiectului.

De asemenea, se realizează o separare și definire a conceptelor utilizate, atât în capitolele ulterioare, cât și în cele precedente.

Capitolul 4.1 - Descrierea componentelor utilizate

În decursul lucrării s-au folosit anumiți termeni prin care denumesc anumite componente ale arhitecturii soluției folosite în proiect. Mai jos se realizează o delimitare logică a componentelor arhitecturii și se asociază o denumire folosită consistent de-a lungul lucrării (atât înainte cât și după acest capitol). Aceste denumiri sunt, de asemenea referite în **Anexa 2 – Lista de acronime**.

BD (baza de date) – reprezintă baza de date folosită la nivelul arhitecturii. Este o denumire generală, fiind folosită pentru orice bază de date relațională folosită la implementarea proiectului (aceasta putând fi schimbată, datorită abstractizării realizate de biblioteca „**Sequelize**” [6]). La nivelul proiectului realizat, baza de date este una de tipul „**Sqlite3**” [3] și se afla în:

%projectRoot%\ParteWeb\Server API\DB\simple.db; modificată în „*Postgresql*” [5] aflată pe rețea, în „*localhost*”.

Tabelele sunt create de un script de creare aflat la:

%projectRoot%\ParteWeb\Server API\createDB.js.

API (serverul *API*) – reprezintă o aplicație de interfață (software la software), ce face legătura între baza de date și aplicațiile de reprezentare vizuală (*GUI* – prezentate mai jos). Modul în care funcționează aceasta este descris mai jos în **Capitolul 5-Implementarea soluției**.

WebGUI (clientul/serverul de interfața web, cu utilizatorul) – reprezintă aplicația de interfață grafică cu utilizatorul, ce face legătura între *API* și clientul, de acasă, al firmei de telefonie mobilă.

DesktopGUI (clientul de interfață desktop, cu utilizatorul) – reprezintă aplicația de interfață grafică cu utilizatorul, ce face legătura atât între *API* și angajații firmei de la ghișeu, cât și între *API* și managerii și administratorii atribuiți gestiunii întregului sistem de relație cu clienții.

Capitolul 4.2 - Interacțiunile componentelor

După cum a fost descris și în **Figura 06**, în afară de relația server API-Baza de data, componentele comunică prin apeluri de tip *HTTPS* [12] la acest server și răspunsuri la aceste apeluri. Apelurile *HTTP / HTTPS* sunt realizate prin utilizarea protocolului *TCP* [14].

O modalitate diferită de comunicare ar putea fi permiterea transmiterii de date de la serverul *API* la *WebGUI* și *DesktopGUI* în orice moment (nu doar ca răspuns la cereri), însă o astfel de implementare are posibilitatea de a cauza probleme de securitate.

Mai jos sunt câteva exemple de apeluri către serverul *API* din *WebGUI* și *DesktopGUI*:

#1. Apel către *API* de la *WebGUI* și răspunsul primit:

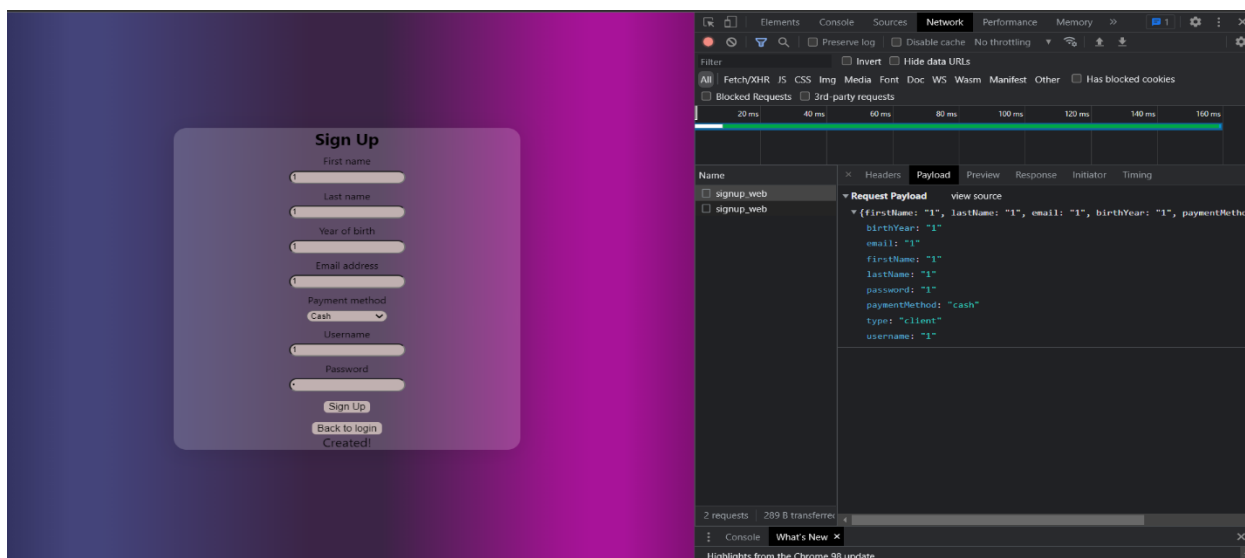


Figura 07. Apel către API din WebGUI.

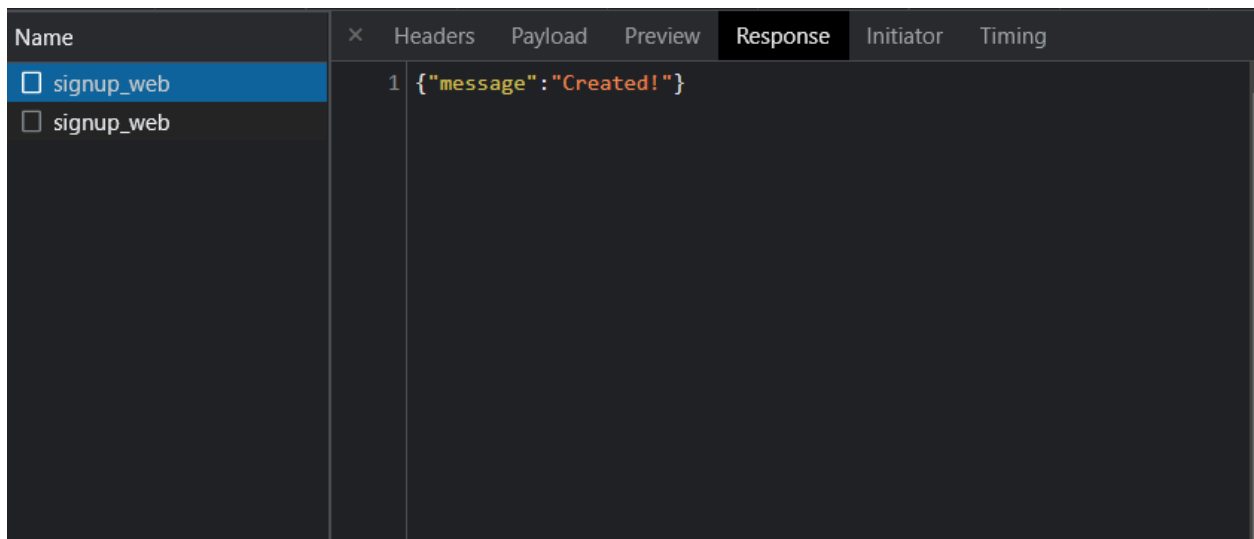


Figura 08. Răspuns de la API la apelul de mai sus.

#2. Apel către *API* de la DesktopGUI și răspunsul primit (vizibil doar în modul de depanare a aplicației):

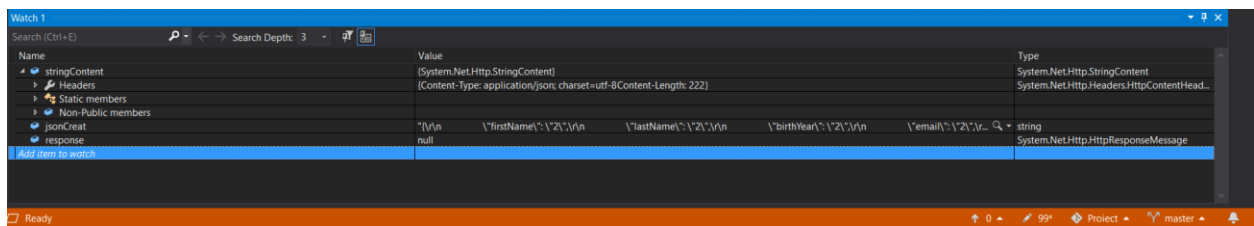


Figura 09. Apel către API din DesktopGUI.

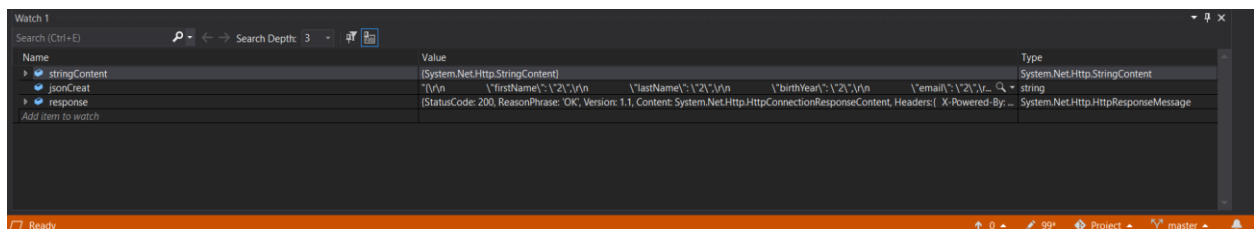


Figura 10. Răspuns de la API la apelul de mai sus.

După cum putem observa, mesajele transmise au format JSON [15] și sunt transmise prin rețea. În ambele aplicații, aceste mesaje sunt analizate, iar datele încapsulate sunt folosite de acestea.

Capitolul 4.3 - Tehnologii pentru crearea de grafice

În decursul proiectului se folosesc, pentru crearea de grafice, atât de *Visual Paradigm Community Edition* [16], cât și de software-uri native **Windows**.

Mai jos, au fost evidențiate câteva grafice ce ajută la înțelegerea lucrării, fiind în format UML, alături de o descrierea ideii reprezentate:

În această figură ne sunt prezentate acțiunile ce au posibilitatea de a fi luate de clienți și angajați, în contextul aplicației *DesktopGUI*.

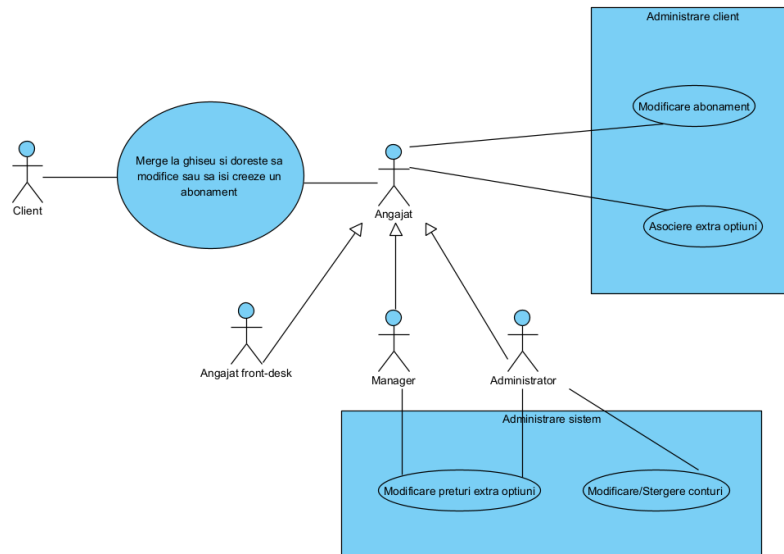


Figura 11. Schema cazurilor de utilizare sistem desktop (DesktopGUI).

În această figură ne sunt prezentate acțiunile ce au posibilitatea de a fi luate de clienți, în contextul aplicației *WebGui*.

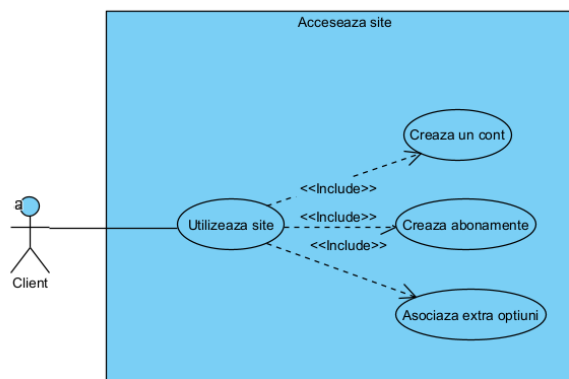


Figura 12. Schema cazurilor de utilizare sistem web (WebGUI).

În această figură sunt reprezentate, în general, modul de comunicare între aplicațiile create.

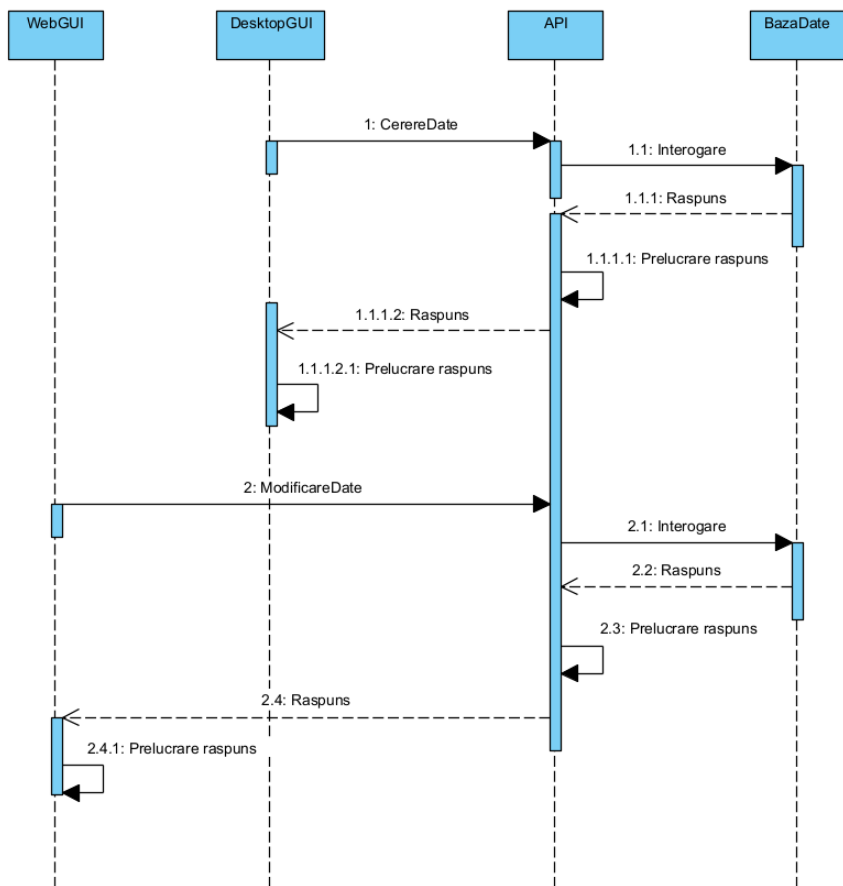


Figura 13. Diagrama de interacțiune a cererilor din întregul sistem.

În figurile următoare sunt reprezentate, modul în care aplicațiile au posibilitatea de a fi utilizate:

- Modul în care se utilizează sistemul de către un angajat de la ghișeu.

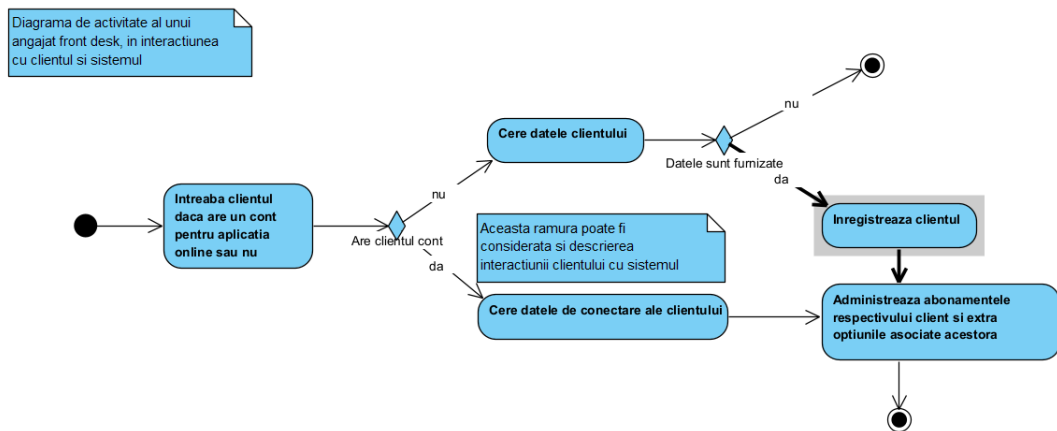


Figura 14. Diagrama de activitate a angajatului de la ghișeu.

- Modul în care se utilizează sistemul de către un administrator.

Diagrama de activitate al unui angajat de tip administrator, în interacțiunea cu sistemul

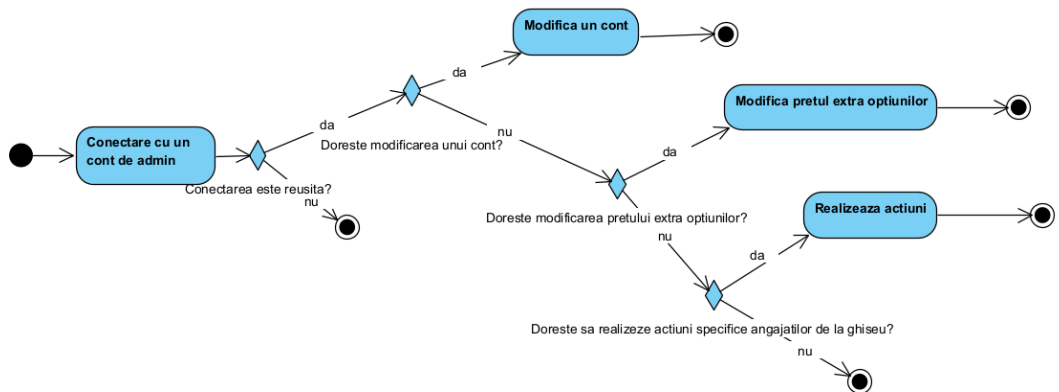


Figura 15. Diagrama de activitate a administratorului.

Capitolul 5 - Implementarea soluției

Mai jos sunt descrise în detaliu elementele definite mai sus la **Capitolul 4.1-Descrierea componentelor create** și modul de utilizare al tehnologiilor definite mai sus la **Capitolul 3-Tehnologii/Metode utilizate**. De asemenea, se intră în detaliu cu privire la modul de funcționare și organizare al codului. Pentru o înțelegere mai ușoară a acestora, se recomandă deschiderea **Figurii 06**, consultarea către **Capitolul 4.2-Interacțiunile componentelor** și consultarea documentațiilor limbajelor folosite (documentațiile se găsesc atât în **Bibliografie**).

Capitolul 5.1 - Schema bazei de date

Baza de date este considerată un punct central al sistemului creat. În cazul nostru, aceasta este de tipul „*Sqlite3*” [3] (modificată în „*Postgresql*” [5]), însă aceasta are posibilitatea de a fi înlocuită cu oricare altă bază de date relațională, cu niște schimbări minimale la driver-ul creat (se află la: %projectRoot %\ParteWeb\Server API\DB\DB_driver.js). Un grafic al acestei baze de date se observă în **Figura 16**, prezentă mai jos. De asemenea, modul de creare al tabelelor bazei de date este derivat din definirea acestora în fișierul „*tableModels*” (se află la ruta: %projectRoot %\ParteWeb\Server API\tableModels);

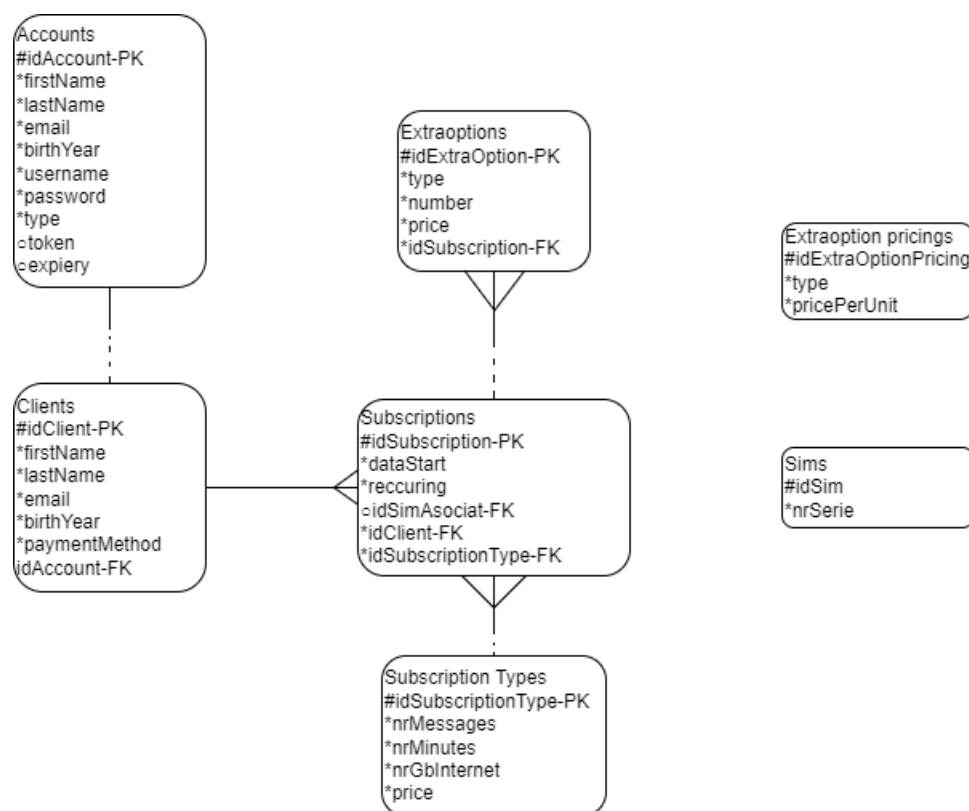


Figura 16. Schema bazei de date.

Mai jos, sunt discutate, în detaliu, modul în care fiecare entitate operează și utilitatea câmpurilor din baza de date, alături de imagini cu ieșiri de consolă pentru interogări pe aceste tabele:

* Fiecare tabela are un câmp, de tip UUID, ce reprezintă un *ID* generat automat. Din documentația „*Sequelize*” [6], ne este specificat ca acest tip de dată este un *CHAR(36) BINARY* pentru „*Mysql*” și *UUID* pentru „*Postgresql*” [5] și „*Sqlite*” [3].

1. **Tabela „accounts”:** Orice angajat ce interacționează cu sistemul are nevoie de un cont în cadrul acestuia. De asemenea, orice client dorește să își administreze activitatea, de acasă, de asemenea, are nevoie de un cont pentru a putea intra în cadrul aplicației web. Codul pentru aceasta se află în *Anexa 3 – Cod sursă aplicație*, fiind *Codul sursă 2*

```
sqlite> .schema accounts
CREATE TABLE `accounts` (`idAccount` UUID NOT NULL PRIMARY KEY, `firstName` VARCHAR(255) NOT NULL, `lastName` VARCHAR(255) NOT NULL, `email` VARCHAR(255) NOT NULL, `birthYear` INTEGER NOT NULL, `username` VARCHAR(255) NOT NULL UNIQUE, `password` VARCHAR(255) NOT NULL, `type` VARCHAR(255) NOT NULL, `token` VARCHAR(255), `expiery` DATETIME, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);
sqlite> pragma table_info('accounts');
0|idAccount|UUID|1||1
1|firstName|VARCHAR(255)|1||0
2|lastName|VARCHAR(255)|1||0
3|email|VARCHAR(255)|1||0
4|birthYear|INTEGER|1||0
5|username|VARCHAR(255)|1||0
6|password|VARCHAR(255)|1||0
7|type|VARCHAR(255)|1||0
8|token|VARCHAR(255)|0||0
9|expiery|DATETIME|0||0
10|createdAt|DATETIME|1||0
11|updatedAt|DATETIME|1||0
```

Figura 17. Output consolă conturi.

2. **Tabela „clients”:** Orice client al companiei trebuie să aibă o intrare în această tabelă. Codul pentru aceasta se află în *Anexa 3 – Cod sursă aplicație*, fiind *Codul sursă 3*.

Cele doua entități, descrise mai sus, au câteva câmpuri ce rămân comune. Acest lucru se întâmplă deoarece, nu toate conturile vor avea un client ce li se asociază, pentru că: angajații firmei nu trebuie să fie și clienții acesteia, iar nu toți clienții doresc să beneficieze de portalul online. În plus, datele sunt mult mai ușor de accesat decât în condițiile în care ar trebui să se interogheze ambele tabele. Astfel câmpurile comune sunt:

- firstName: numele mic al clientului/angajatului, de tip strig;
- lastName: numele de familie al clientului/angajatului, de tip strig;
- email: email-ul clientului/angajatului, de tip strig;
- birthYear: anul de naștere al clientului/angajatului, de tip strig.

Pe lângă aceste câmpuri, tabela „clients” are câmpurile:

- `paymentMethod`: metoda de plată preferată de client, de tip enumerabil, cu valorile: `cash`, `transfer`;

- `idAccount`: identificatorul unic pentru contul acestuia, de tip *UUID* (specific „*Sequelize*” [6], însemnând că acel identificator este universal unic). Acesta are valoare nulă în cazul în care clientul nu are cont și este astfel creat din aplicația desktop.

Pe lângă câmpurile de mai sus, tabela „**accounts**” are câmpurile:

- `username`: numele de utilizator al contului, de tip `String`;
- `password`: parola contului (sub forma securizată prin „hashing”), de tip `String`;
- `type`: tipul de cont, de tip `String` (posibil să fie modificat în tip enumerabil, însă acum este de acest tip pentru a-mi impune implementarea anumitor condiții în cod). Are următoarele valori posibile: `client`, `employee`, `manager`, `admin`;

- `token`: un sir de caractere generat pentru a nu trimite username-ul și parola peste rețea la fiecare cerere, de tip `String`;

- `expiry`: durata până la care token-ul este valabil, de tip date. Este de tip *DATETIME* pentru „*Mysql*” și „*Sqlite*” [3], însă este de tip *TIMESTAMP WITH TIME ZONE* pentru „*Postgresql*” [5].

```
sqlite> .schema clients
CREATE TABLE `clients` (`idClient` UUID NOT NULL PRIMARY KEY, `firstName` VARCHAR(255) NOT NULL, `lastName` VARCHAR(255) NOT NULL, `email` VARCHAR(255) NOT NULL, `birthYear` INTEGER NOT NULL, `paymentMethod` TEXT NOT NULL, `idAccount` UUID, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);
sqlite> pragma table_info('clients');
0|idClient|UUID|1||1
1|firstName|VARCHAR(255)|1||0
2|lastName|VARCHAR(255)|1||0
3|email|VARCHAR(255)|1||0
4|birthYear|INTEGER|1||0
5|paymentMethod|TEXT|1||0
6|idAccount|UUID|0||0
7|createdAt|DATETIME|1||0
8|updatedAt|DATETIME|1||0
```

Figura 18. Output consolă clienți.

3. **Tabela „subscriptions”**: Aici se vor salva detalii despre abonamentele clienților. Codul pentru aceasta se află în *Anexa 3 – Cod sursă aplicație*, fiind *Codul sursă 4*. Această tabelă are următoarele câmpuri:

- `dataStart`: data la care va începe abonamentul, de tip `String` (pentru ușurință de transmitere a datelor de la *GUI*);
- `recurring`: dacă abonamentul va fi reînnoit luna viitoare, de tip `String` (pentru ușurință de transmitere a datelor de la *GUI*);

- idSimAsociat: identificatorul unic pentru *SIM*-ul asociat abonamentului, de tip *UUID* (specific „*Sequelize*” [6], însemnând că acel identificator este universal unic);
- idClient: identificatorul unic pentru clientul asociat abonamentului, de tip *UUID* (specific „*Sequelize*” [6], însemnând că acel identificator este universal unic);
- idSubscriptionType: identificatorul unic pentru tipul de abonament asociat abonamentului, de tip *UUID* (specific „*Sequelize*” [6], însemnând că acel identificator este universal unic).

```
sqlite> .schema subscriptions
CREATE TABLE `subscriptions` (`idSubscription` UUID PRIMARY KEY, `dataStart` VARCHAR(255) NOT NULL, `recurring` VARCHAR(255) NOT NULL, `idSimAsociat` UUID, `idClient` UUID NOT NULL, `idSubscriptionType` UUID NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);
sqlite> pragma table_info('subscriptions');
0|idSubscription|UUID|0||1
1|dataStart|VARCHAR(255)|1||0
2|recurring|VARCHAR(255)|1||0
3|idSimAsociat|UUID|0||0
4|idClient|UUID|1||0
5|idSubscriptionType|UUID|1||0
6|createdAt|DATETIME|1||0
7|updatedAt|DATETIME|1||0
```

Figura 19. Output consolă abonamente.

4. **Tabela „subscriptionTypes”:** În această tabelă se vor salva detalii despre tipurile de abonamente din care clienții firmei au posibilitatea de a alege. Aceste tipuri de abonamente sunt create de administratori sau manageri (vezi câmpul „type” din tabela „accounts”). Codul pentru aceasta se află în *Anexa 3 – Cod sursă aplicație*, fiind *Codul sursă 5*. Această tabelă are următoarele câmpuri:

- nrMessages: numărul de mesaje asociat tipului de abonament, de tip Integer;
- nrMinutes: numărul de minute asociat tipului de abonament, de tip Integer;
- nrGbInternet: numărul de giga-bytes de internet asociat tipului de abonament, de tip Integer;
- price: prețul asociat tipului de abonament, de tip Integer.

```
sqlite> .schema subscriptionTypes
CREATE TABLE `subscriptionTypes` (`idSubscriptionType` UUID PRIMARY KEY, `nrMessages` INTEGER NOT NULL, `nrMinutes` INTEGER NOT NULL, `nrGbInternet` INTEGER NOT NULL, `price` INTEGER NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);
sqlite> pragma table_info('subscriptionTypes');
0|idSubscriptionType|UUID|0||1
1|nrMessages|INTEGER|1||0
2|nrMinutes|INTEGER|1||0
3|nrGbInternet|INTEGER|1||0
4|price|INTEGER|1||0
5|createdAt|DATETIME|1||0
6|updatedAt|DATETIME|1||0
```

Figura 20. Output consolă tipuri de abonamente.

5. **Tabela „sims”:** Această tabelă nu este folosită în aplicația dezvoltată, alături de identificatorul sau asociat din tabela „subscriptions” și există doar în cazul în care se dorește o dezvoltare a aplicației și către aceasta zona operațională. Codul pentru aceasta se află în *Anexa 3 – Cod sursă aplicație*, fiind *Codul sursă 6*. Această tabelă are următoarele câmpuri:

- nrSerie: numărul de serie asociat SIM-ului, de tip Integer.

```
sqlite> .schema sims
CREATE TABLE `sims` (`idSim` UUID PRIMARY KEY, `nrSerie` INTEGER NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);
sqlite> pragma table_info('sims');
0|idSim|UUID|0|1
1|nrSerie|INTEGER|1|0
2|createdAt|DATETIME|1|0
3|updatedAt|DATETIME|1|0
```

Figura 21. Output consolă SIM-uri.

6. **Tabela „extraOptions”:** În cazul în care abonamentele puse la dispoziție nu satisfac nevoile clientului, acesta are posibilitatea de a suplini un abonament prin intermediul unor extra opțiuni. Codul pentru aceasta se află în *Anexa 3 – Cod sursă aplicație*, fiind *Codul sursă 7*. Această tabelă are următoarele câmpuri:

- idSubscription: identificatorul unic pentru abonamentul căreia i se asociază extra opțiunea, de tip *UUID* (specific „*Sequelize*” [6], însemnând că acel identificator este universal unic).
- type: reprezintă tipul extra opțiunii respective, este de tip **String** (posibil să fie modificat în tip enumerabil, însă acum este de acest tip pentru ușurință de transmitere a datelor). Are următoarele valori posibile: nrMinutes, nrMessages, nrGbInternet;
- number: reprezintă numărul de unități de tipul respectiv asociat extra opțiunii curente, de tip Integer.
- price: reprezintă prețul calculat prin intermediul datelor din tabela „extraOptionPricings”, de tip Integer.

```
sqlite> .schema extraOptions
CREATE TABLE `extraOptions` (`idExtraOption` UUID PRIMARY KEY, `idSubscription` UUID NOT NULL, `type` VARCHAR(255) NOT NULL, `number` VARCHAR(255) NOT NULL, `price` INTEGER NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);
sqlite> pragma table_info('extraOptions');
0|idExtraOption|UUID|0|1
1|idSubscription|UUID|1|0
2|type|VARCHAR(255)|1|0
3|number|VARCHAR(255)|1|0
4|price|INTEGER|1|0
5|createdAt|DATETIME|1|0
6|updatedAt|DATETIME|1|0
```

Figura 22. Output consola extra opțiuni.

7. **Tabela „extraOptionPricings”:** Această tabelă ajută în calcularea prețului pentru tipurile de extra opțiuni. După ce sunt create intrările pentru fiecare tip, acestea vor fi doar actualizate cu o noua valoare. Codul pentru aceasta se află în **Anexa 3 – Cod sursă aplicație**, fiind **Codul sursă 8**. Această tabelă are următoarele câmpuri:

- **type:** reprezintă tipul extra opțiunii căreia i se aplica prețul pe unitate, este de tip String (posibil să fie modificat în tip enumerabil, însă acum este de acest tip pentru ușurința de transmitere a datelor). Are următoarele valori posibile: nrMinutes, nrMessages, nrGbInternet;
- **pricePerUnit:** reprezintă prețul pe unitate pentru tipul respectiv, de tip Integer.

```
sqlite> .schema extraOptionPricings
CREATE TABLE `extraOptionPricings` (`idExtraOptionPricing` UUID PRIMARY KEY, `type` VARCHAR(255) NOT NULL, `pricePerUnit` INTEGER NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);
sqlite> pragma table_info('extraOptionPricings');
0|idExtraOptionPricing|UUID|0||1
1|type|VARCHAR(255)|1||0
2|pricePerUnit|INTEGER|1||0
3|createdAt|DATETIME|1||0
4|updatedAt|DATETIME|1||0
```

Figura 23. Output consolă prețuri extra opțiuni.

Capitolul 5.2 - Serverului API

Următorul element prezent în **Figura 06**, este serverul **API**; așa că în continuare se va vorbi despre acesta.

Serverul **API** se află la: %projectRoot%\ParteWeb\Server API\

Aici se pot observa mai multe fișiere și directoare. Mai jos se discută în amănunt ce reprezintă fiecare.

- Directorul **„configuration”**: conține un singur fișier: „index.js” în care s-a salvat portul pe care rulează serverul.
- Directorul **„DB”**: conține două fișiere: „simple.db” și „DB_driver.js”. „simple.db” este fișierul de bază de date de tipul **„Sqlite3”** [3] (baza de date de tipul **„Postgresql”** [5] fiind, teoretic, la distanță, nu are un astfel de fișier) discutat mai sus la **Capitolul 5.1-Schema bazei de date**. „DB_driver.js” este un fișier cu detalii despre modul de conectare la baza de date; de asemenea, acesta duce la crearea bazei de date și este folosit în restul aplicației pentru a păstra referința la instanța actuală a bazei de date. Pentru mai multe detalii se va consulta documentația **„Sequelize”** [6].
- Directorul **„node_modules”**: conține biblioteci externe și dependențe folosite în această parte a proiectului. Fișierele „package-lock.json” și „package.json” au legătură cu

acestea. În „package.json”, se regăsesc clar dependențele instalate la rularea comenzii „npm install” (și alte scripturi de rulare ce nu există, însă, în acest proiect).

- Directorul **„routes”**: conține rutele importate în fișierul „server.js”. Aici, toate operațiile pe tabelele bazei de date sunt împărțite, în mare, pe 4 operații: vizualizare, create, actualizare și ștergere. Majoritatea operațiilor folosesc o metoda de tip „POST”, deoarece denumirile rutelor sunt destul de clare, iar pentru a putea realiza o transmitere de token, avem nevoie ca operațiile să permită transmitere acestuia prin „request.body”.

- Directorul **„tableModels”**: conține modelele tabelelor bazei de date, descrise în amănunt mai sus la *Capitolul 5.1-Schema bazei de date*.

- Fișierul **„createDB.js”**: conține un script de creare al bazei de date. Acesta este folosit în cazul unei erori pentru a recrea baza, însă duce la distrugerea intrărilor. Pentru mai multe detalii se consultă documentația „*Sequelize*” [6].

- Fișierul **„server.js”**: este punctul de intrare în aplicație (al serverului *API*). Aici se observă rutele *API*-ului, importate din fișierele directorului „routes”. Pentru mai multe detalii se consultă documentația de „*Express*”.

Capitolul 5.3 - WebGUI

Unul dintre cele două componente vizuale ale sistemului este componenta *WebGUI* și este un proiect de tip „*React*” [8]. Acest element și cel de după, *DesktopGUI*, implică un concept denumit „Event driven programming”, descris în detaliu în *Capitolul 5.5-Programare prin evenimente*.

Aceasta se află la: %projectRoot%\ParteWeb\GUI\

Mai jos, discut, în amănunt, modul de funcționare al acestei componente și rolul directoarelor și fișierelor:

- a. Directorul **„node_modules”**: similar componentei *Server API*, conține biblioteci externe și dependențe folosite în aceasta parte a proiectului. Fișierele „package-lock.json” și „package.json” au legătură cu acestea. În „package.json”, se regăsesc clar dependențe instalate la rularea comenzii „npm install” (și alte scripturi de rulare ce nu există, însă, în acest proiect).
- b. Directorul **„public”**: conține resurse necesare începerii proiectului (ex: iconițe, un fișier de tip HTML de la care pornește crearea arborelui logic de componente „*React*” [8])

- c. Directorul **„src”**: aici se găsesc cele mai importante elementele ale acestei componente. Acestea sunt descrise mai jos:
- d. Directorul **„Actions”**: element ce exportă prin fișierul „index.js”, de la nivelul acestuia, acțiunile ce se realizează din aplicația *WebGUI*. Aici, acțiunile sunt împărțite pe tabelele din baza de date și *API* ce sunt utilizate.

De asemenea, tipul folosit și valoarea returnată de payload are legătură cu elementele „reducer”; tipul fiind preluat de „redux-promise-middleware” și creând cele 3 cazuri ce trebuie tratate în reducere (Pending, Fulfilled, Rejected).

- e. Directorul **„Components”**: în acest element se găsește codul de afișare al întregii aplicații. „App.js” este inclusă în „index.js”. Aici se realizează o operație de tip switch pentru a afla în ce stadiu se află aplicația. Ecranele de login, signup și cel principal intră în această operație.

La nivelul ecranului principal, sunt împărțite ecranele. Acestea au posibilitatea de a fi separate în componente, de sine stătătoare, însă, pentru ușurință, în versiunea curentă, toate se află în fișierul „Main.js” al directorului „Main”.

- f. Directorul **„Configs”**: element ce are un singur fișier, „globals.js”, ce exportă adresa serverului *API* pentru elementul descris mai sus, directorului „Actions”.
- g. Directorul **„Reducers”**: similar acțiunilor, fișierul „index.js”, aferent „reducerelor”, sunt combinate acestea și trimise către elementul „store”. În fiecare „reducer” este creată o stare ce este modificată de rezultatul provenit de la elementul acțiune. Acestea sunt apoi trimise prin „store”, la componentele aplicației și expuse utilizatorului.
- h. Directorul **„Stores”**: element ce are un singur fișier, „main_store.js”, în care sunt comasați reducerii definiți în elementul descris mai sus, directorului „Reducers”, cu un **„middleware”** ce permite o tratare mai ușoară a promisiunilor în acești reduceri, **„redux-promise-middleware”**. După această comasare, se creează acest „store” ce acționează ca o baza de date locală cu datele preluate din apelurile la serverul *API* și se exportă către a fi folosită de elementul „index.js”
- i. Fișierul **„index.css”**: element ce transmite un element *CSS*, cu reguli generale de stil, pentru elementul descris mai jos, „index.js”, punctul de intrare în aplicație
- j. Fișierul **„index.js”**: este punctul de intrare în aplicație, aici se importă „index.css” și „main_store”-pentru a putea fi accesibil întregii aplicații. De asemenea, pentru a putea utiliza acest „store”, avem nevoie de un element de tip „Provider”.

Capitolul 5.4 - DesktopGUI

Ultima componentă vizuală a sistemului este componenta *DesktopGUI*. Aici se discută versiunea folosită în dezvoltarea aplicației. Versiunea „Release” este discutată mai sus la **Capitolul 2.2-Modul de instalare, rulare și utilizare**. Acest element implică, după cum s-a menționat și mai sus, un concept denumit „Event driven programming”, descris în detaliu în **Capitolul 5.5-Programare prin evenimente**.

DesktopGUI ca versiune de dezvoltare se află la:

%projectRoot%\ParteC#\FrontDeskManagement

De aici, pentru a rula versiunea de dezvoltare, se accesează: FrontDeskManagement-> bin-> Debug-> netcoreapp3.1-> FrontDeskManagement.exe

Accesând: FrontDeskManagement, se pot observa mai multe directoare și fișiere. Mai jos se discută despre acestea și rolul lor:

- Directorul „**bin**”: menționat mai sus, conține executabilul generat la compilare, și dependențele sale (atât pentru varianța „Release” cât și pentru varianta „Debug”).
- Directorul „**Forms**”: în acest folder s-a organizat codul sursă pentru paginile de formular în care angajații trebuie să introducă date.
- Directorul „**obj**”: conține elemente ajutătoare creării conținutului directorului „bin”.
- Directorul „**Pages**”: în acest folder s-a organizat codul sursă pentru paginile de meniu. Acestea sunt în linii mari: pagina de login, pagina de signup, și paginile de selectare de formular pentru rolurile stabilite ale angajaților.
- Directorul „**Properties**”: folder autogenerat de „*Visual Studio*” [1].
- Directorul „**Res**”: conține resurse vizuale folosite la nivelul proiectului (ex: imagini).
- Fișierul „**FrontDeskManagement.csproj**”: fișier autogenerat de „*Visual Studio*” [1].
- Fișierul „**FrontDeskManagement.csproj.user**”: fișier autogenerat de „*Visual Studio*” [1].
- Fișierul „**InstanceGlobals.cs**”: conține o clasă globală, ce reține detalii despre utilizatorul curent.
- Fișierul „**Program.cs**”: punctul de intrare în aplicație.

Capitolul 5.5 - Programare prin evenimente

Acest tip de programare este o paradigmă utilizată în decursul proiectului, pentru a realiza interfețele cu utilizatorii: în speță componentele *WebGUI* și *DesktopGUI*.

Pentru acest subcapitol s-au folosit atât de cunoștințele cumulate la cursul domnului profesor Ion Smeureanu și în cadrul seminarului, cât și de anumite capitole din cartea: „Visual C# .NET” scrisă de Ion Smeureanu, Marian Dârdală și Adriana Reveiu [17]. Cartea se axează pe mai multe aspecte particulare ale limbajului C# și a programării de aplicații de tip „*Winforms*”, folosite în dezvoltarea componentei *DesktopGUI*, însă în acest capitol se dorește a se oferi o explicație pentru modul de funcționare a acestei paradigme și o justificare pentru alegerea acesteia.

În carte, este descris, cât mai practic, modul de funcționare al acestor tipuri de programe. Aici, ne este descris cum în funcția principală a programului interfața este încărcată și există la nivelul acesteia o stivă de evenimente. La fiecare acțiune a utilizatorului, un eveniment este încărcat în această stivă, prin conceptul de delegat și este mai apoi executat (având forma unei funcții).

Ideea de delegat „suplinește conceptul de pointer la funcție din C++”, astfel permițând o „legare întârziată”, la execuție. Această legare întârziată, permite la rândul său să apelăm funcțiile sau metodele în momentul când o acțiune are loc. Pentru a reuși în aplicarea acestui concept, trebuie discutată și conceptul de clasă de tip eveniment, ce încapsulează un delegat și o metodă de tip „fire”, ce permite apelarea delegatului.

Această modalitate este cea abordată în rezolvarea problemei interfețelor cu utilizatorii, deoarece: îmi este cunoscut și acesta este ușor de implementat în mediul de dezvoltare utilizat. Această ușurință de implementare ascunde însă mecanismul de apelare a delegaților evenimentelor în stivă, acesta netrebuind a fi implementat de programator, ci fiind generat automat de mediul de dezvoltare.

O astfel de generare este atât un lucru bun cât și un lucru dăunător; în momentul în care nu se ia în considerare modul său de funcționare, putând genera probleme de compilare. De asemenea, această abordare, eveniment-acțiune, este criticată ca ducând la un cod producător de erori, ce este greu de extins și la aplicații prea complexe.

Ca o replică la aceste critici, s-a răspuns printr-o organizare a codului în clase, în modalitate orientată obiect pentru componenta *DesktopGUI*. Componenta *WebGUI*, însă, nu este organizată în acest fel, iar problemele codului spaghetti încep să fie aparente în acest cadru.

Capitolul 5.6 - Programare orientată obiect

Mai sus s-a discutat în linii mari despre programarea orientată obiect. În acest subcapitol se prezintă în detaliu acest subiect și utilitatea sa în cadrul lucrării.

Pentru acest subcapitol s-au folosit atât de cunoștințele cumulate în anii de facultate cât și de cartea „Programarea în limbajul C/C++” de Ion Smeureanu și Marian Dârdală și „Programarea Orientata Obiect în Limbajul C++”, de Ion Smeureanu și Marian Dârdală. [18]

Deși cartea se axează pe o implementare în C/C++, conceptele de bază sunt folosite într-o aplicație de tip C#. Clasele permit o organizare foarte clară a codului, iar conceptele de baza ale programării descrise în carte au posibilitatea de a fi translate asupra oricărui limbaj.

Programarea orientata obiect în acest context, este o paradigmă impusă pentru a păstra codul într-o stare cât mai clară.

Această implementare este utilizată la nivelul componentei *DesktopGUI*, a proiectului.

Într-un context mai larg, programarea orientată obiect este o modalitate de organizare a codului oferită de limbaje. Obiectele reprezintă atât concepte abstracte cât și entități mai palpabile.

Conceptele de attribute, metode, constructori și suprascrieri sunt folosite în proiect, atât în mod direct, cât și prin cod autogenerat de mediul de programare. Tipologia proiectului ne permite de asemenea utilizarea ideii de clasă statică; Idee inexistentă în C++ și astfel netratată în cărțile scrise de domnii profesori; O clasă statică în limbajul C# este fi conceptualizată ca o clasa în C++ cu toate câmpurile statice. În contextul proiectului, clasa statică are rolul de instanță globală, ce reține date despre utilizatorul curent al aplicației, când acestea sunt primite de la serverul *API*.

În final, peste implementarea propusă, există posibilitatea de a implementa și concepte de „clean code” de tipul „**design patterns**”, însă pentru un proiect de o scală redusă, s-a decis să nu se complice atât de mult situația.

Capitolul 5.7 - Baza de date

La nivelul proiectului nu s-a folosit de un Sistem de gestiune al bazelor de date propriu-zis (pana la adiția bazei de date „*Postgresql*” [5] ce vine instalată cu o interfață grafică), ci de unul mascat printr-o bibliotecă – „*Sequelize*” [6]. O cunoaștere a modului de funcționare al bazelor de date relaționale este însă necesară, pe lângă modul de funcționare a bibliotecii folosite.

Pentru acest subcapitol și implementarea ideilor menționate în proiect, s-a folosit de detalii din cartea „SGBD Oracle. Limbajul SQL”, de Adela Bâra, Iuliana Botha, Anca-Georgiana Fudor, Ion Lungu și Simona Vasilica Oprea. [19]

Ideea de joncțiune, de cheie primară și secundară este implementat, în proiect, în mod direct, în cadrul serverului *API*, pentru a avea un control mai bun asupra datelor ce se regăsesc în baza de date. De asemenea, ca o consecință, codul creat trebuie să țină cont de ștergerea în lanț a tabelelor în cazul unei astfel de necesități.

Acest control ne permite modificarea unor reguli de business, fără a afecta baza de date, nefiind nevoie de o migrare costisitoare și fără a fi nevoie rescrierea unei mari cantități de cod. Problema adusă însă de această implementare este cea a unui cod mai greu de urmărit.

Capitolul 6 - Concluzii

Ca o concluzie la lucrarea prezentată, implementarea unui sistem de gestiune nu este o sarcină ușoară, proiectul descris mai sus implicând o multitudine din cunoștințele acumulate din timpul anilor de studiu.

În capitolele și subcapitolele trecute se evidențiază modul în care deciziile luate, în cadrul proiectării aplicației, au fost influențate de cunoștințele cumulate. Cu toate acestea, nu sunt detaliate limbajele folosite și toate modurile în care funcționează aplicațiile, astfel de amănunte nefiind extrem de interesante sau relevante.

Din momentul actual, aplicația are posibilitatea de a fi dezvoltată în mai multe direcții. O direcție interesantă este administrarea cartelelor *SIM* (câmp asupra căruia este creată și o tabelă în baza de date). O astfel de abordare este interesantă, clienții putând teoretic personaliza abonamentele având în vedere și acest detaliu; mai târziu putând să se intre și în sfera modului de taxare a clienților pe baza acestor detalii.

O altă direcție interesantă este includerea posibilității de asignare și modificare a numărului de telefon, la acest nivel putând fi luate în considerare mai multe probleme de intimitate.

Proiectul prezentat mai sus este format din trei componente. Componenta de server *API* și cel de *DesktopGUI* conțin comentarii ce clarifică modul de scriere a codului cât mai clare. Componenta *WebGUI* nu este la același nivel de calitate al comentariilor, codul putând fi extins printr-o modularizare a acestuia, ce ar necesita o restructurare a comentariilor, de asemenea.

Toate componentele conțin fișiere denumite „*READ.md*” ce descriu atât modul de rulare, cât și modul de organizare al codului.

După cum am specificat și în **Capitolul 2.3 – Modul de instalare, rulare și utilizare**, aplicația este ușor de preluat pentru a fi dezvoltată în continuare într-un context de „business”, însă aceasta va avea nevoie de dezvoltare și, desigur, de o infrastructură construită pentru aceasta.

O direcție în care ar putea evolua, pe lângă adiția cartelelor *SIM*, ar putea fi modul de taxare al clienților: prin adăugarea unei gestionari a de carduri/conturi bancare sau cel puțin un mod automat de a anunța clienții despre scadența facturilor.

O documentație a întregului proiect este, de asemenea, posibilă, însă chiar acest document poate fi considerat o documentație destul de amănunțită atât a modului de organizare cât și de utilizare a aplicației.

Bibliografie

- [1] Microsoft, „Visual Studio Microsoft,” Microsoft, 10 05 2022. [Interactiv]. Available: <https://visualstudio.microsoft.com/>. [Accesat 09 06 2022].
- [2] Node, „NodeJS Documentation,” Node, 30 07 2017. [Interactiv]. Available: <https://nodejs.org/en/docs/>. [Accesat 09 06 2022].
- [3] Sqlite3, „SQLite Documentation,” Sqlite3, 15 09 2016. [Interactiv]. Available: <https://www.sqlite.org/docs.html>. [Accesat 09 06 2022].
- [4] Microsoft, „Visual Studio Code,” VS Code Microsoft, 10 05 2019. [Interactiv]. Available: <https://code.visualstudio.com/learn>. [Accesat 09 06 2022].
- [5] „PostgreSQL: Documentation,” The PostgreSQL Global Development Group, 19 05 2022. [Interactiv]. Available: <https://www.postgresql.org/docs/>. [Accesat 06 07 2022].
- [6] „Sequelize documentation,” present Sequelize contributors, 24 08 2019. [Interactiv]. Available: <https://sequelize.org/v5/>. [Accesat 07 06 2022].
- [7] Mozilla, „Cross-Origin Resource Sharing(CORS),” Mozilla Foundation, 28 05 2022. [Interactiv]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>. [Accesat 12 06 2022].
- [8] Meta Platforms, Inc., „React Documentation,” Meta Platforms, Inc., 04 10 2017. [Interactiv]. Available: <https://reactjs.org/docs/getting-started.html>. [Accesat 12 06 2022].
- [9] D. Abramov, „Redux Usage Guides,” 26 06 2021. [Interactiv]. Available: <https://redux.js.org/>. [Accesat 11 06 2022].
- [10] Microsoft, „Desktop Guide (Windows Forms .NET),” Microsoft, 16 11 2021. [Interactiv]. Available: <https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-6.0>. [Accesat 12 06 2022].
- [11] Microsoft, „Compare Newtonsoft.Json to System.Text.Json, and migrate to System.Text.Json,” Microsoft, 11 03 2022. [Interactiv]. Available: <https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-migrate-from-newtonsoft-how-to?pivots=dotnet-6-0>. [Accesat 11 06 2022].
- [12] Mozilla, „HTTP request methods,” Mozilla Foundation, 13 05 2022. [Interactiv]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>. [Accesat 11 06 2022].
- [13] IBM, „Express Glossary,” OpenJS Foundation, 13 11 2014. [Interactiv]. Available: <https://expressjs.com/en/resources/glossary.html>. [Accesat 11 06 2022].
- [14] University of Southern California, „TRANSMISSION CONTROL PROTOCOL - PROTOCOL SPECIFICATION,” Information Sciences Institute University of Southern California, 01 11 1981. [Interactiv]. Available: <https://datatracker.ietf.org/doc/html/rfc793>. [Accesat 11 06 2022].

- [15] Mozilla, „JSON Documentation,” Mozilla Foundation, 27 03 2022. [Interactiv]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON. [Accesat 11 06 2022].
- [16] Visual Paradigm, „Visual Paradigm Tutorials,” Visual Paradigm, 26 02 2018. [Interactiv]. Available: <https://www.visual-paradigm.com/tutorials/>. [Accesat 11 06 2022].
- [17] I. Smeureanu, M. Dârdală și A. Reveiu, Visual C#.NET, București: Cison, 2002.
- [18] I. Smeureanu și M. Dârdală, Programarea Orientată Obiect în Limbajul C++, Bucuresti: CPLsme, 2002, p. 333.
- [19] I. Botha, A. Bara, A.-G. Fudor, I. Lungu și S. V. Oprea, SGBD Oracle. Limbajul SQL, București: Editura ASE, 2016, p. 166.

Anexa 1 – Lista de figuri

Figura 01.	Captură de ecran cu un fișier de tip READ.me
Figura 02.	Modificare tip utilizator direct în baza de date
Figura 03.	Modificare necesare în „driver-ul” bazei de date
Figura 04.	Tabelele în baza de date de tip Postgresql
Figura 05.	Structura de fișiere a aplicației WebGUI
Figura 06.	Schema a arhitecturii generală ale proiectului
Figura 07.	Apel către API din WebGUI
Figura 08.	Răspuns de la API la apelul de mai sus
Figura 09.	Apel către API din DesktopGUI
Figura 10.	Răspuns de la API la apelul de mai sus
Figura 11.	Schema cazurilor de utilizare sistem desktop (DesktopGUI)
Figura 12.	Schema cazurilor de utilizare sistem web (WebGUI)
Figura 13.	Diagrama de interacțiune a cererilor din întregul sistem
Figura 14.	Diagrama de activitate a angajatului de la ghișeu
Figura 15.	Diagrama de activitate a administratorului
Figura 16.	Schema bazei de date
Figura 17.	Output consolă conturi
Figura 18.	Output consolă clienți
Figura 19.	Output consolă abonamente
Figura 20.	Output consolă tipuri de abonamente
Figura 21.	Output consolă SIM-uri
Figura 22.	Output consola extra opțiuni
Figura 23.	Output consolă prețuri extra opțiuni

Anexa 2 – Lista de acronime

<i>API</i>	Application Programming Interface (interfața programabilă a aplicației)
<i>WebGUI</i>	Aplicația web de interfață grafică cu utilizatorul
<i>DesktopGUI</i>	Aplicația desktop de interfață grafică cu utilizatorul
<i>DB</i>	Data Base (baza de date)
<i>SQL</i>	Structured query language (limbaj structurat de interogări)
<i>HTML</i>	Hyper Text Markup Language
<i>CSS</i>	Cascading style sheets
<i>JS</i>	JavaScript
<i>JSON</i>	JavaScript Object Notation
<i>GUI</i>	Graphical User Interface (interfață grafică cu utilizatorul)
<i>HTTPS</i>	Hypertext Transfer Protocol Secure
<i>HTTP</i>	Hypertext Transfer Protocol
<i>TCP</i>	Transmission Control Protocol
<i>SIM</i>	Subscriber Identity Module
<i>ID</i>	IDentifier (identificator)
<i>UUID</i>	Universal Unique IDentifier (identificator universal unic)

Anexa 3 – Cod sursă al aplicației

```
const hash_sha256=(message)=>{
  return CryptoJS.SHA256
  (CryptoJS.enc.Hex.parse(message)).toString();
}
```

Cod sursă 1. Cod sha256.

```
//account table structure
const Sequelize = require("sequelize");
const sequelize = require('../DB/DB_driver');
const { DataTypes } = require('sequelize');

const Account = sequelize.define('account',{
  idAccount: {
    type: DataTypes.UUID,
    defaultValue: DataTypes.UUIDV4,
    primaryKey: true,
    allowNull:false
  },
  firstName:{
    type: Sequelize.STRING,
    allowNull:false,
  },
  lastName:{
    type: Sequelize.STRING,
    allowNull:false
  },
  email:{
    type: Sequelize.STRING,
    allowNull:false,
  },
  birthYear:{
```

```

        type: Sequelize.INTEGER,
        allowNull:false,
    },
    username:{
        type: Sequelize.STRING,
        allowNull:false,
        unique:true
    },
    password:{
        type: Sequelize.STRING,
        allowNull:false
    },
    type:{
        type: Sequelize.STRING,
        allowNull:false
    },
    token:{
        type: Sequelize.STRING
    },
    expiry:{
        type:DataTypes.DATE
    }
});

```

```

module.exports = Account;

```

Cod sursă 2. Cod creare model conturi.

```

//client table structure
const Sequelize = require("sequelize");
const sequelize = require('../DB/DB_driver');
const { DataTypes } = require('sequelize');

const Client = sequelize.define('client',{

```

```

idClient: {
  type: DataTypes.UUID,
  defaultValue: DataTypes.UUIDV4,
  primaryKey: true,
  allowNull:false
},
firstName:{
  type: Sequelize.STRING,
  allowNull:false,
},
lastName:{
  type: Sequelize.STRING,
  allowNull:false
},
email:{
  type: Sequelize.STRING,
  allowNull:false,
},
birthYear:{
  type: Sequelize.INTEGER,
  allowNull:false,
},
paymentMethod:{
  type: Sequelize.ENUM('cash', 'transfer'),
  allowNull:false,
},
idAccount: {
  type: Sequelize.DataTypes.UUID
}
});

module.exports = Client;

```

Cod sursă 3. Cod creare model clienți.

```

//subscription table structure
const Sequelize = require("sequelize");
const sequelize = require('../DB/DB_driver');
const { DataTypes } = require('sequelize');
const Subscription = sequelize.define('subscription',{
  idSubscription: {
    type: DataTypes.UUID,
    defaultValue: DataTypes.UUIDV4,
    primaryKey: true
  },
  dataStart:{
    type: Sequelize.STRING,
    allowNull:false
  },
  reccuring:{
    type: DataTypes.STRING,
    allowNull:false
  },
  idSimAsociat: {
    type: DataTypes.UUID,
  },
  idClient: {
    type: DataTypes.UUID,
    allowNull:false
  },
  idSubscriptionType: {
    type: DataTypes.UUID,
    allowNull:false
  }
});
module.exports = Subscription;

```

Cod sursă 4. Cod creare model abonamente.

```

//subscriptionType table structure
const Sequelize = require("sequelize");
const sequelize = require('../DB/DB_driver');
const { DataTypes } = require('sequelize');
const SubscriptionType = sequelize.define('subscriptionType',{
  idSubscriptionType: {
    type: DataTypes.UUID,
    defaultValue: DataTypes.UUIDV4,
    primaryKey: true
  },
  nrMessages:{
    type: Sequelize.INTEGER,
    allowNull:false
  },
  nrMinutes:{
    type: Sequelize.INTEGER,
    allowNull:false
  },
  nrGbInternet:{
    type: Sequelize.INTEGER,
    allowNull:false
  },
  price:{
    type: Sequelize.INTEGER,
    allowNull:false
  }
});
module.exports=SubscriptionType;

```

Cod sursă 5. Cod creare model tipuri de abonamente.

```

//subscription table structure
const Sequelize = require("sequelize");

```

```

const sequelize = require('../DB/DB_driver');
const { DataTypes } = require('sequelize');
const Sim = sequelize.define('sim',{
  idSim: {
    type: DataTypes.UUID,
    defaultValue: DataTypes.UUIDV4,
    primaryKey: true
  },
  nrSerie:{
    type: DataTypes.INTEGER,
    allowNull:false
  }
});
module.exports = Sim;

```

Cod sursă 6. Cod creare model *SIM*-uri.

```

//extraOption table structure
const Sequelize = require("sequelize");
const sequelize = require('../DB/DB_driver');
const { DataTypes } = require('sequelize');
const ExtraOption = sequelize.define('extraOption',{
  idExtraOption: {
    type: DataTypes.UUID,
    defaultValue: DataTypes.UUIDV4,
    primaryKey: true
  },
  idSubscription: {
    type: DataTypes.UUID,
    allowNull:false
  },
  type:{
    type: Sequelize.STRING,
    allowNull:false
  }
});

```

```

    },
    number:{
        type: Sequelize.INTEGER,
        allowNull:false
    },
    price:{
        type: Sequelize.INTEGER,
        allowNull:false
    }
});
module.exports = ExtraOption;

```

Cod sursă 7. Cod creare model extra opțiuni.

```

//extraOptionPricing table structure
const Sequelize = require("sequelize");
const sequelize = require('../DB/DB_driver');
const { DataTypes } = require('sequelize');
const ExtraOptionPricing = sequelize.define('extraOptionPricing',{
    idExtraOptionPricing: {
        type: DataTypes.UUID,
        defaultValue: DataTypes.UUIDV4,
        primaryKey: true
    },
    type: {
        type: DataTypes.STRING,
        allowNull:false
    },
    pricePerUnit:{
        type: Sequelize.INTEGER,
        allowNull:false
    }
});
module.exports = ExtraOptionPricing;

```

Cod sursă 8. Cod creare model preturi extra opțiuni.