6/14/2022

BESTFITCONFIG

PROIECT CURS
METODE DEZVOLTARE SOFTWARE

Studenți: Andrei DINA, Robert LIȚĂ, Mihai BÎRSAN, Sebastian ENE |Prof. Dr. Alin ȘTEFĂNESCU

Cuprins

1. Introducere

Proiectul poate fi găsit la adresa: https://github.com/RobertLita/BestFitConfig

Proiectul a fost creat folosind limbajul C++ și oferă o interfață grafică creată cu ajutorul librăriei Qt. Aplicația poate fi folosită pentru gestionarea unei săli de fitness. Este o aplicație cu un design minimalist, ușor de folosit, cu butoane sugestive.

Motivația ce a stat la baza creării acestui proiect este tema din cadrul cursului "Metode de dezvoltare software", împreună cu dorința de a crea o aplicație utilă în viața de zi cu zi. Astfel am ajuns la ideea de scriere a unei aplicații care gestionează clienții, abonamentele și facilitățile unei săli de fitness.

2. User stories

Am acoperit șapte user-stories, după cum urmează:

1) As a fitness gym admin, I want to identify the clients of this gym, so I can see the clients facilities and maintain the security of the gym.

2) As a fitness gym admin, I want to add/ delete or modify a new client in the database, so I can keep track of all the clients.

3) As a fitness gym admin, I want to check if a subscription of a client is still available, so that we can propose or renew the subscription.

4) As a fitness gym admin, I want to check if a client is able to use a certain facility, so that I can keep track of the number of customers for each facility.

5) As a client, I want to know how much I have to pay for my subscription, so I can manage my money easily.

6) As a fitness gym admin, I want to add a new facility to the database, so the gym clients can benefit of more facilities in a single subscription and in a single place.

7) As a fitness gym admin, I want to add/delete or modify a new subscription in the database, so clients will a bigger variety of subscriptions.

3. Diagrama UML

În cele două figuri de mai jos am redat diagramele UML.

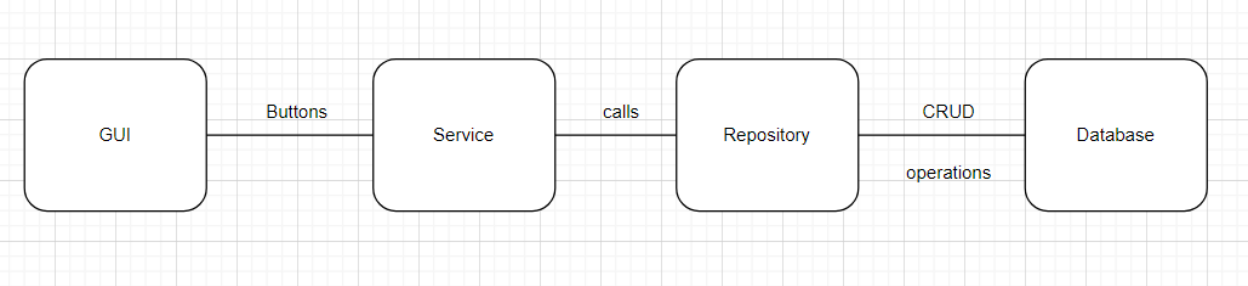

Figura 1 Diagrama UML- abstractizare

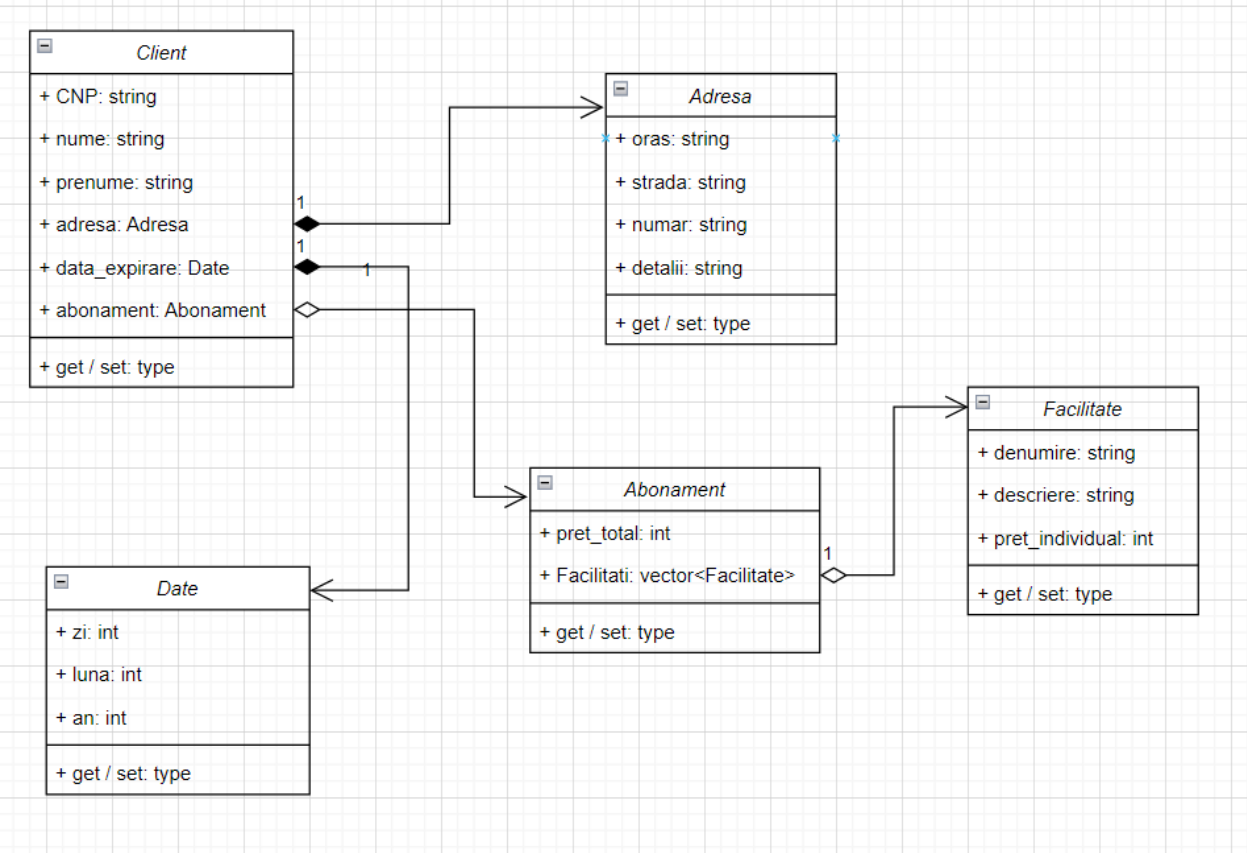

Figura 2 Diagrama UML-detaliu

4. Procesul de *"creație"*

Am urmărit în principal funcționalitatea aplicației și mai puțin aspectul interfeței grafice, care a trecut printr-o lungă lista de adăugiri, îmbunătățiri și reparații.

Crearea aplicației a fost realizată urmărind pașii pe care i-am stabilit încă de la început, anume:

1. Stabilirea obiectivului
2. Scrierea codului (Clase + Interfață Grafică)
3. Corectarea erorilor

5. Structura fișierelor din proiect

Codul este organizat, ușor de urmărit pentru o persoană ce vede proiectul pentru prima dată:

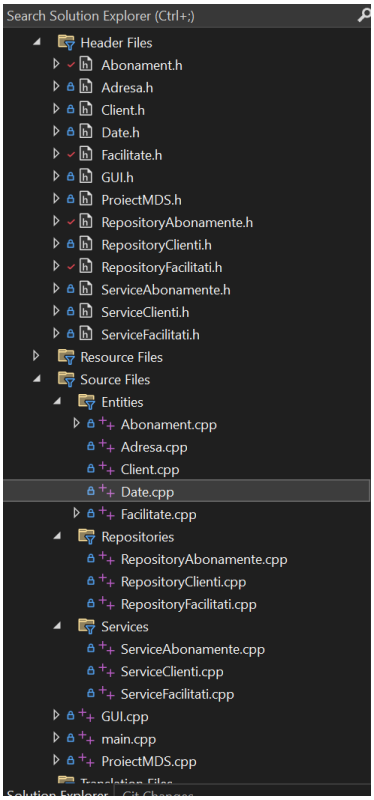

Figura 3 Lista cu fișierele de testat

6. Prezentarea aplicației pe scurt

Cel mai bun mod de a prezenta o aplicație este chiar o simulare de folosire a aplicației pentru o sală de fitness.

Așadar, când utilizatorul pornește aplicația, poate adăuga facilitățile pe care le pune la dispoziție sala de fitness respectivă: denumire, descriere și preț. Dacă utilizatorul a introdus datele greșit sau facilitatea respectivă nu mai există, se poate modifica în aplicație sau șterge.

Ulterior utilizatorul poate să creeze un abonament pentru care selectează facilitățile la care un client o să aibă acces dacă îl cumpără.

Utilizatorul poate să adauge client nou în aplicație completând următoarele câmpuri cu date personale: nume, prenume, cod numeric personal (CNP), adresa (formată din oraș, stradă, număr și detalii), tipul de abonament pe care îl dorește clientul, iar data de expirare a abonamentului este generată automat din ziua în care clientul și-a făcut abonamentul. Dacă utilizatorul a introdus greșit datele unui client, acesta poate să modifice sau să șteargă din câmpuri.

Utilizatorul poate verifica dacă un client mai are abonamentul activ. Abonamentul expirat se poate reînnoi, bineînțeles la cererea clientului.

7. Codul sursă

Am stabilit la nivel de echipă să folosim Visual Studio, deoarece este un IDE (Integrated Development Environment) care se pliază pe necesitățile aplicației noastre.

Codul sursă constă în șase pachete, organizate în funcție de ceea ce conțin acestea plus numeroase fișiere auxiliare. Se poate consulta pe Github, nu a fost copiat în acest fișier într-o anexă.

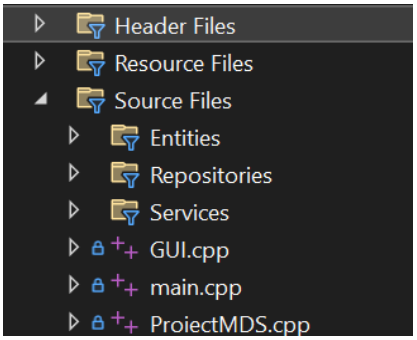

Figura 4 Pachetele ce alcătuiesc proiectul

Codul aplicației este disponibil aici: https://github.com/RobertLita/BestFitConfig

Am introdus metode specifice de tratare a erorilor de fiecare dată când a fost nevoie, astfel încât eroarea să fie semnalată, apoi tratată și aplicația să funcționeze în continuare.

```
bool esteValid() {

    if (this->pretIndividual <= 0) {
        throw runtime_error("Pretul nu poate sa fie negativ sau 0.");
    }

    return true;
}
```

Figura 5 Exemplu de tratare a excepțiilor

8. Principalele probleme întâlnite

Pe parcursul creării aplicației ne-am lovit de multe erori, care au fost rezolvate în cel mai scurt timp după detectarea lor.

8.1 Exemplul 1

Nu puteam șterge un client sau reînnoi abonamentul deoarece nu verificam într-o manieră corectă dacă data expirării precedea data curentă.

```
bool ok = true;
if (c.getData().getAn() < an)
    ok = false;
else if (c.getData().getAn() == an && c.getData().getLuna() < luna)
    ok = false;
else if (c.getData().getAn() == an && c.getData().getLuna() == luna && c.getData().getZi() < zi)
    ok = false;

if (ok)
{
    QString mesaj = QString::fromStdString("Abonament valid!");
    QMessageBox::information(nullptr, "Eroare", mesaj);
}

else {
    QString mesaj = QString::fromStdString("Abonament expirat!");
    QMessageBox::information(nullptr, "Eroare", mesaj);
```

Figura 6 Ilustrarea primului exemplu de problemă

Am rezolvat acest bug prin modificarea funcției **esteValid** din clasa **Date**, astfel încât să putem șterge și reînnoi abonamentul unui client chiar dacă nu a mai fost la acea sală de mai mulți ani.

```
    zi = zi - 28;
}

Client c(serviceClienti.findClient(CNP));
if (c.getData().getAn() >= an && c.getData().getLuna() >= luna && c.getData().getZi() >= zi)
{
    QString mesaj = QString::fromStdString("Abonament valid!");
    QMessageBox::information(nullptr, "Eroare", mesaj);
}

else {
    QString mesaj = QString::fromStdString("Abonament expirat!");
```

Figura 7 Rezolvarea bug-ului (problema 1)

8.2 Exemplul 2

În momentul în care doream să ștergem un abonament, aplicația ștergea de fiecare dată primul abonament.

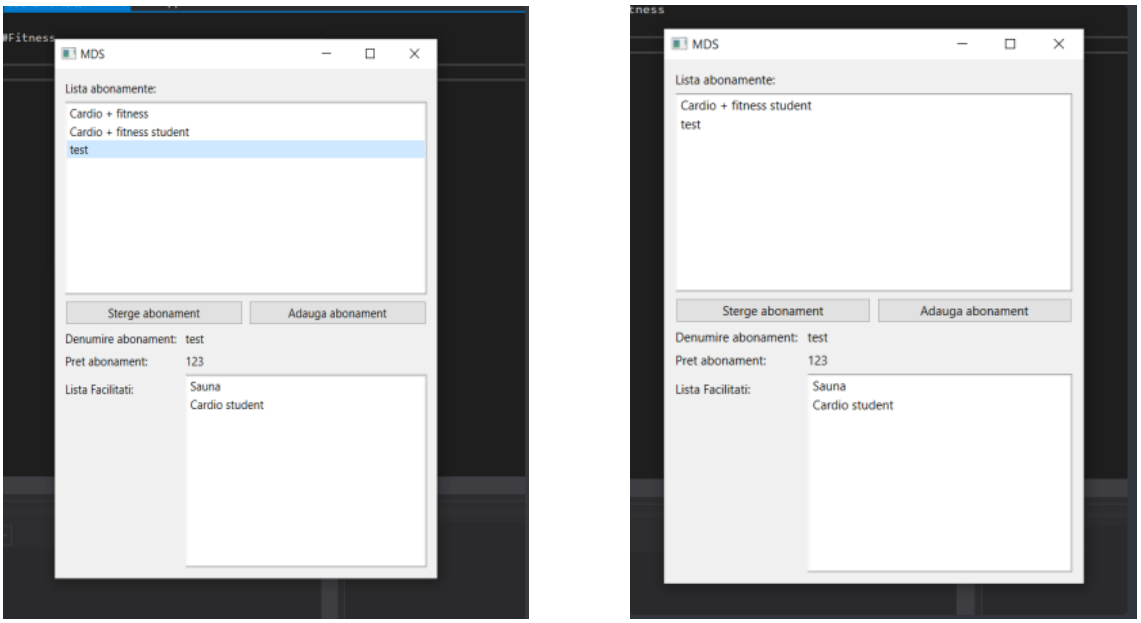

Figura 8 Ilustrarea celei de-a doua probleme

Am identificat problema și ulterior am tratat-o. Am modificat funcția de ștergere a unui abonament, astfel încât să șteargă abonamentul selectat, iar dacă nu este niciun abonament selectat, să-l șteargă pe primul.

8.3 Exemplul 3

Clienții adăugați nu erau salvați în fișierul .CSV, astfel că nu puteau fi vizualizați în momentul în care utilizatorul apăsa butonul <afișează client>.

```
void addClient(Client c) {
    for (auto& Client & client : this->clienti) {
        if (client == c) {
            throw runtime_error("Clientul exista deja!");
        }
    }
    if (c.esteValid()) {
        this->clienti.push_back(_Val: c);
        writeClient(c);
    }
```

Figura 9 Exemplul 3

Eroarea a fost tratată prin modificarea funcțiilor **readClienti** și **writeClient**.

```
void writeClient(Client c) {
    fstream fout;
    fout.open(_Filename: "clienti.csv", _Mode: ios_base::app);
    fout << c.getCNP() << '#' << c.getNume() << '#' << c.getPrenume() << '#' << c.getAbonament().getDenumire() << '#'
        << c.getAdresa().getOras() << '#' << c.getAdresa().getStrada() << '#' << c.getAdresa().getNumar() << '#'
        << c.getAdresa().getDetalii() << '#' << c.getData().getZi() << '#' << c.getData().getLuna() << '#'
        << c.getData().getAn() << '\n';
    fout.close();
}
```

Figura 10 Rezolvarea problemei aferente exemplului 3, writeClient

```
vector<Client> readClienti() {
    vector<Client> clienti;
    fstream fin;
    fin.open(_Filename:"clienti.csv", _Mode:ios::in);
    vector<string> row;
    string line, word, temp;
    while (getline(& _Istr:fin, & _Str:temp)) {
        row.clear();
        stringstream s(temp);
        while (getline(& _Istr:s, & _Str:word, _Delim:'#')) {
            const std::string& str(word);
            row.push_back(_Val:word);
        }
        Client c;
        c.setCNP(cnp:row[0]);
        c.setNume(nume:row[1]);
        c.setPrenume(prenume:row[2]);
        Adresa adresa;
        adresa.setOras(oras:row[4]);
        adresa.setStrada(strada:row[5]);
        adresa.setNumar(numar:row[6]);
        adresa.setDetalii(detalii:row[7]);
        Date d;
        d.setZi(v:atoi(_String:row[8].c_str()));
        d.setLuna(v:atoi(_String:row[9].c_str()));
        d.setAn(v:atoi(_String:row[10].c_str()));
        c.setAdresa(adresa);
        c.setData(d);


        vector <Abonament> abonamente = readAbonamente();
        Abonament abonament = RepositoryAbonamente::getAbonamentByName(name:row[3], abonamente);
        c.setAbonament(ot:abonament);
        clienti.push_back(_Val:c);
```

Figura 11 Rezolvarea problemei aferente exemplului 3, readClient

9. Source control

Încă de la început am folosit Github, pentru o colaborare mai eficientă. Am împărțit toată logica aplicației în mai multe branch-uri, iar prin commit-uri și push-uri am sincronizat toate fișierele proiectului.

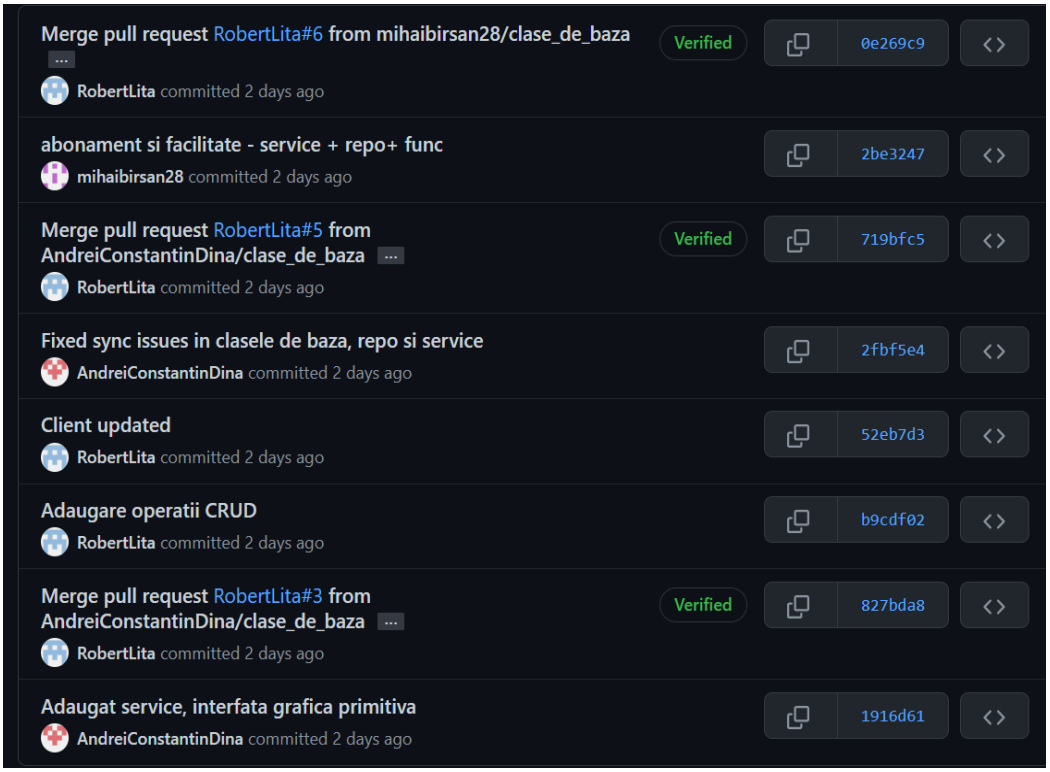

Figura 12 Snapshot din utilizarea Github

10. Teste

Pentru a verifica funcționalitatea aplicației, am testat toate functionalitățile acesteia: adăugare client, facilitate, abonamente; modificare pentru client, facilitate, abonament și ștergere pentru client, facilitate, abonament. Au fost testate ștergerea unei facilități dacă nu exista nici o facilitate în baza de date. Verificarea situației în care un client are abonamentul valid sau expirat, iar dacă este expirat, încercarea reînnoirii lui.

Ne-am asigurat că aplicația rulează conform așteptărilor noastre, verificând-o atât din punct de vedere al unui utilizator obișnuit (ne-am asigurat ca operațiile CRUD --Create/Read/Update/Delete-- să fie corecte), dar și din punct nostru de

vedere, cel al programatorilor (spre exemplu am verificat ca aplicația să nu se blocheze dacă încercăm să ștergem abonamente iar lista este deja goală).

11. Design/arhitectură

Am organizat clasele în trei pachete mari: pachetul **Entities**, pachetul **Repositories** si pachetul **Services**. Astfel, când utilizatorul acționează un buton din interfața grafică, apelăm funcția respectivă din Services (de exemplu **adaugăClient**(*Client c*)), iar **Services** verifică corectitudinea datelor și apelează funcția din **Repositories** (**adaugăClienti**(*Client c*)) care manipulează în mod direct baza de date.

12. Concluzie

Am realizat o aplicație interesantă și funcțională. Am deprins lucrul în echipă, am folosit tehnologii noi (librăria QT) și asimilat tool-uri, concepte și tehnici des utilizate în companiile de IT.

Codul aplicației este disponibil la adresa:
https://github.com/RobertLita/BestFitConfig

Un scurt demo este disponibil aici:

https://www.youtube.com/watch?v=USv6Bf5gPiM

Lista de figuri