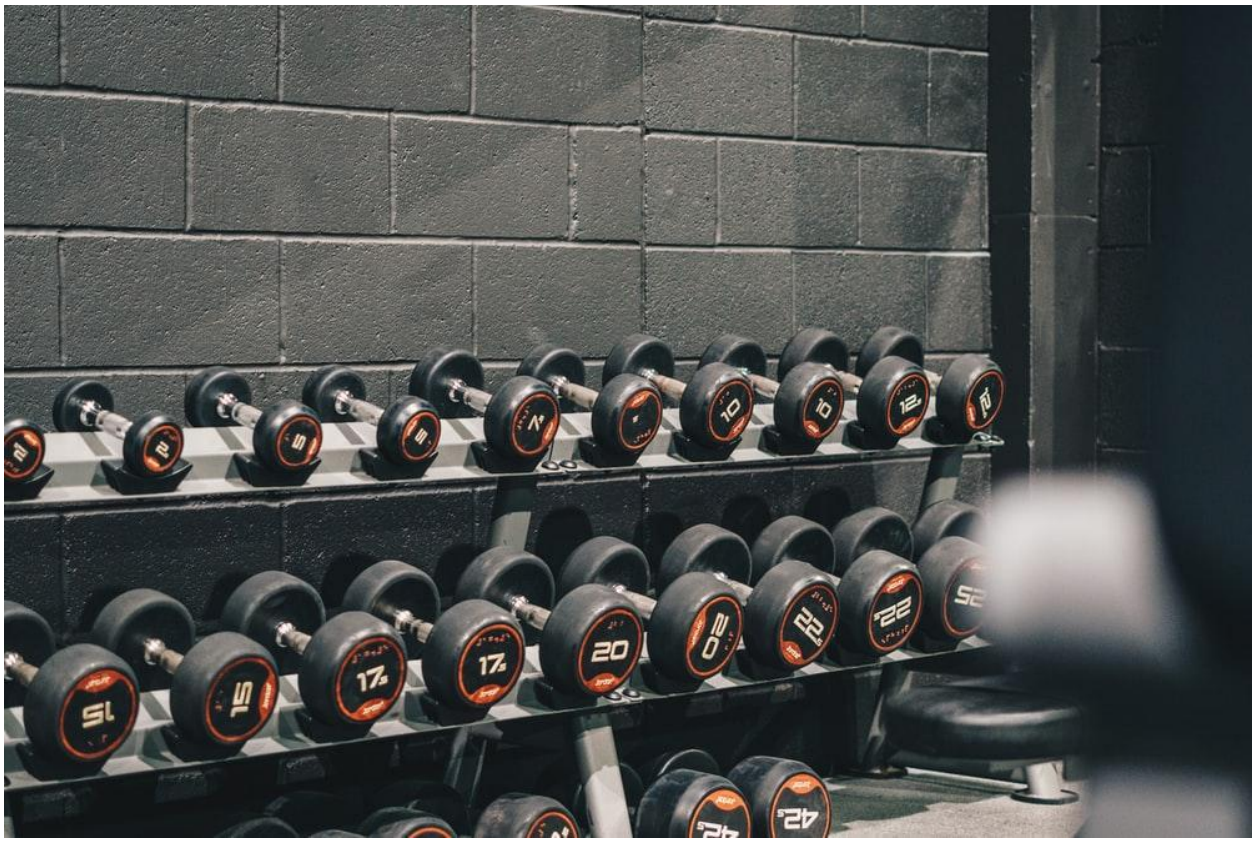
BestFitConfig



Autori:

Dina Andrei

Liță Robert

Bîrsan Mihai

Ene Sebastian

Profesor:

Alin Ștefănescu

CUPRINS:

1. Introducere:

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

Proiectul a fost creat folosind limbajul C++ și oferă o interfață grafică creată cu ajutorul librariei 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”, împreuna cu dorința de a creea o aplicație utilă în viața de zi cu zi. Astfel am ajuns la idea de creare a unei aplicații care gestionează clienții, abonamentele și facilitățile unei săli de fitness.

2. User Stories

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.

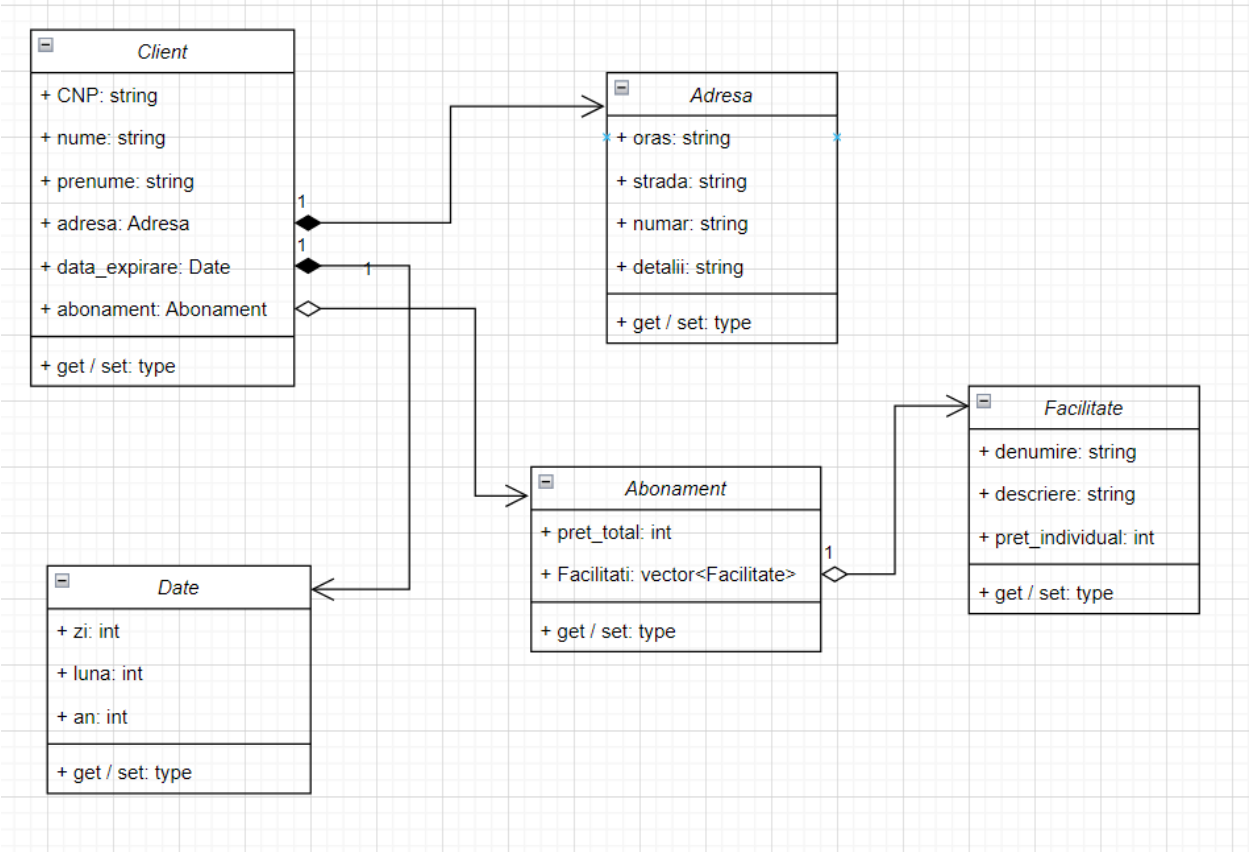
4) As a fitness gym admin, I want to check if a client is able to use a certain facility, so that I can keep track of the number of customers for each facility.

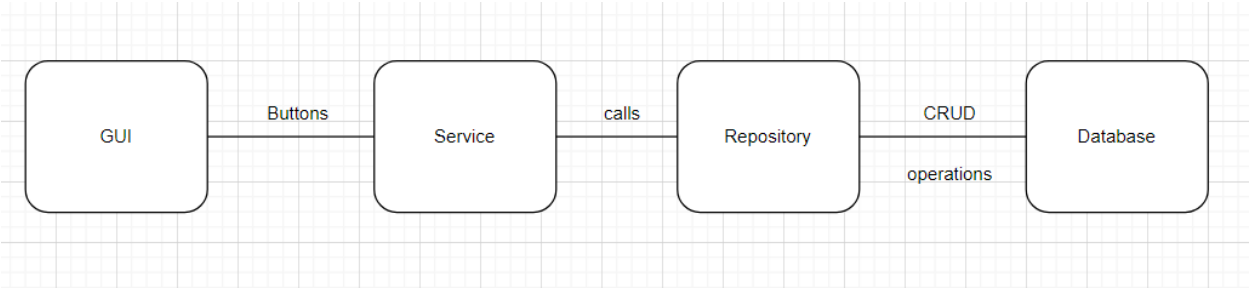
5) As a client, I want to know how much I have to pay for my subscription, so I can manage my money easily.

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





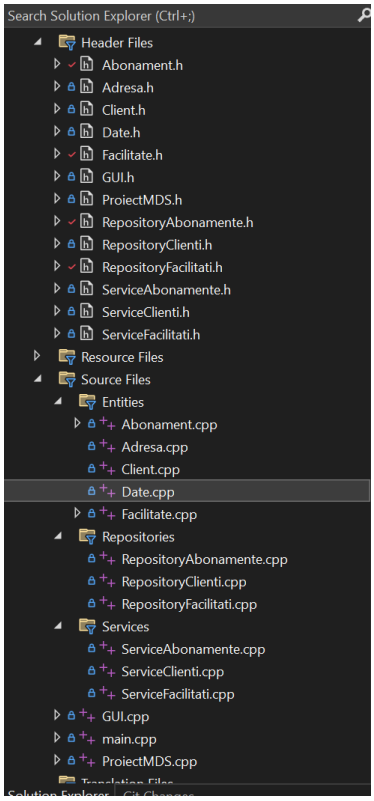
4. Procesul de creație:

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

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

1. Stabilirea obiectivului
2. Scrierea codului (Clase + Interfață Grafică)
3. Corectarea erorilor
4. Testarea funcționalităților și îmbunătățirea lor

Codul este organizat, ușor de urmărit pentru o persoana care este din afara echipei:



5. Prezentarea aplicației:

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

Cand utilizatorul porneste aplicatie poate adauga facilitatile pe care le pune la dispoziție sală respective: denumire, descriere și pret. Dacă utilizatorul a introdus datele greșit sau facilitatea respectivă nu mai exista se poate modifica în aplicație sau șterge.

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

Utilizatorul poate să adauge client noi în aplicație completand următoarele câmpuri cu date personale: nume, prenume, cnp, adresa formată din oraș, stradă, număr și detalii, abonamentul pe care îl dorește clientul, iar data de expirare a

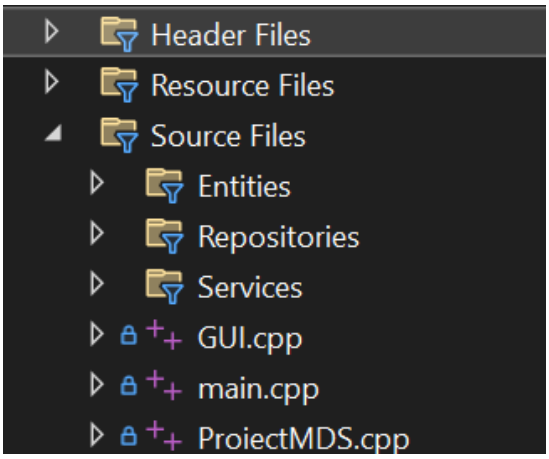
abonamentului este generata automat din ziua în care clientul și-a făcut abonamentul. Dacă utilizatorul a introdus greșit datele unui client, acesta poate să îl modifice sau să îl șteargă.

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

6.Codul sursa:

Am stabilit la nivel de echipa să folosim Visual Studio, deoarece este un IDE-uri care se mula pe necesitățile aplicație noastre.

Codul sursa este consta în 6 pachete, organizate în funcție de ceea ce conțin plus alte fișiere auxiliare.



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;
}
```

7. 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.

- Exemplul 1:

Nu puteam șterge un client sau să îi reînnoim abonamentul deoarece nu verificam într-o manieră corectă dacă data expirării precedă data curentă.

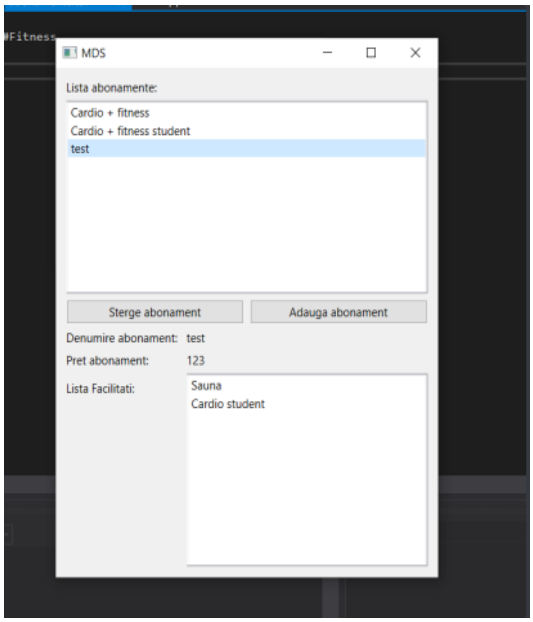
```
bool ok = true;
if (c.getData().getAn() < an)
    ok = false;
else if (c.getData().getAn() == an && c.getData().getLuna() < luna)
    ok = false;
else if (c.getData().getAn() == an && c.getData().getLuna() == luna && c.getData().getZi() < zi)
    ok = false;

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);
```

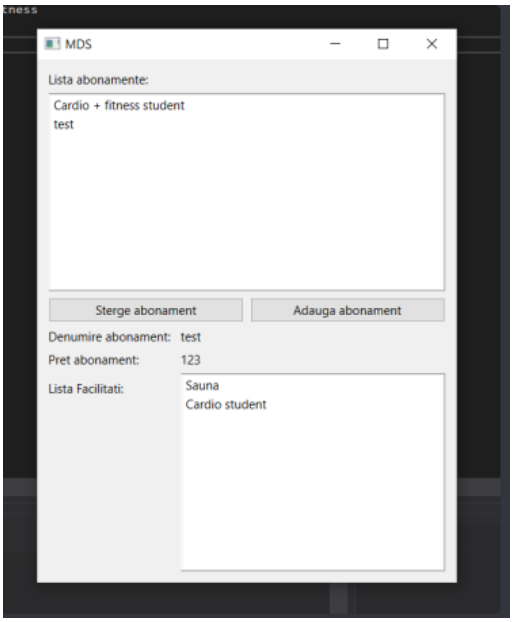
Am rezolvat acest bug prin modificarea funcției esteValid din clasa Date, astfel incat sa putem șterge și reînnoi abonamentul unui client chiar dacă nu a mai fost la aceea sală de mai mulți ani.

```
        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!");
```

- Exemplul 2:

În momentul în care doream să ștergem un abonament, aplicația ștergea mereu primul abonament.





Am depistat 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.

- Exemplul 3:

Clienții adăugați nu erau salvați în fișierul csv, astfel 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);
    }
```

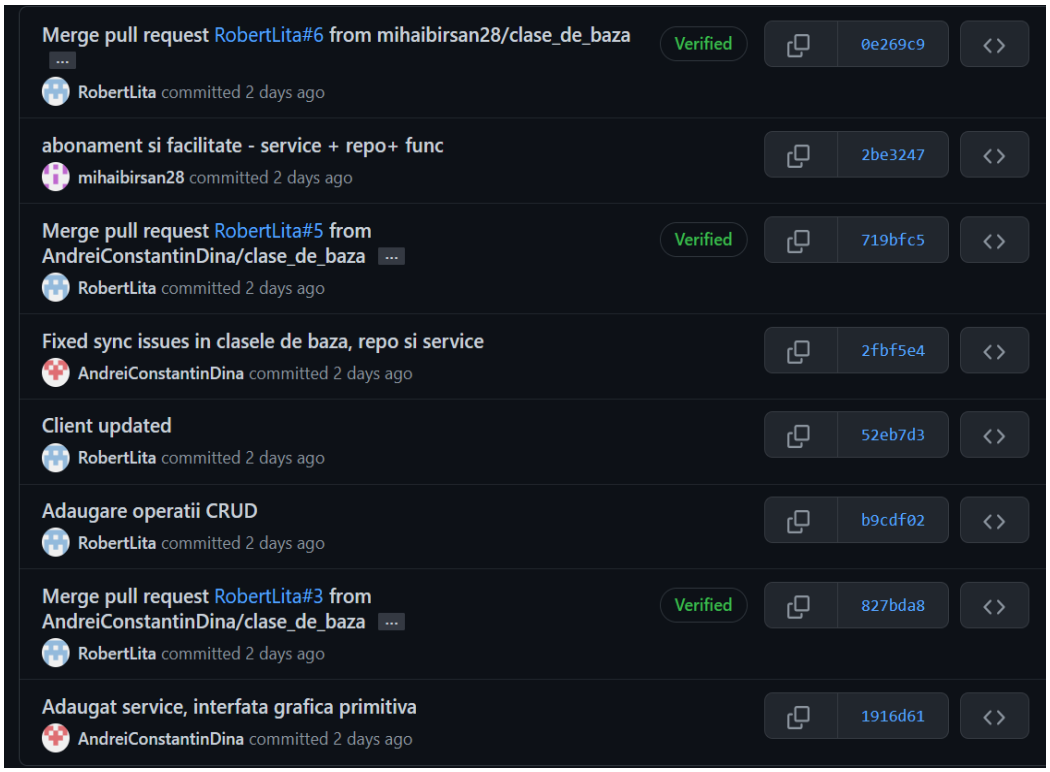
Eroare a fost tratată prin modificarea funcțiilor readClienti și writeClieti.

```
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();
}
```

```
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);
```

8. 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 am sincronizat toate fișierele proiectului.



9.Teste:

Pentru a verifica funcționalitatea aplicației, am testat toate functionalitățile aplicației: adăugare pentru 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 dacă un client are abonamentul valid sau expirat, iar dacă este expirat reînnoirea 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 să fie corecte), dar și din punct nostru de vedere, cel al programatorilor (spre ex. am verificat ca aplicația să nu se blocheze dacă încercăm să ștergem abonamente iar lista este deja goală).

10.Design/arhitectură:

Am organizat clasele in 3 pachete mari: pachetul entities, pachetul repositories si pachetul services. Astfel, când utilizatorul acționa un buton din interfața grafică, apelam funcția respectivă din Service (de ex. adaugăClient(Client c)), iar Service-ul verifica corectitudinea datelor și apela funcția din Repository (adaugăClienti(Client c)) care manipula în mod direct baza de date.

11. Încheiere

Am realizat o aplicație interesantă și funcțională care ne-a învățat lucrul în echipă, tehnologii noi și ne-am folosit și am asimilat tool-uri, concepte și tehnici des utilizate în companiile de IT.

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

Un scurt demo este disponibil aici:

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