**DOCUMENTATIE**

**– Proiect 1 –**

**– Balanici Andrei-Daniel G231/2 –**

**– Berendi Razvan Alexandru G231/2 –**

**– 11.01.2023 –**

1. **Cerinta:**

**Clinica medicala**

O statiune balneara ofera diverse spatii pentru diverse tratamente (acces piscina, masaj, impachetari, etc.) Din cauza existentei unei cereri mari se solicita plata la momentul programarii, existand totusi posibilitatea anularii, caz in care se face si returnarea banilor.

Exista n locatii (S1, S2,... Sn) la care se pot face programari, fiecare locatie oferind acelasi set de tratamente (T(1), T(2), ... T(m)). Locatiile sunt deschise intre orele 10:00-18:00 si fiecare tratament are asociat un cost C(j), un anumita durata de realizare Tdelta(j) si un numar maxim de pacienti care pot fi tratati in acelasi timp N(i,j) 0<i<=n; 0<j<=m.

La prima iteratie de dezvoltare aplicatia de programari nu ofera interfata grafica prin care sa se vizualizeze situatia la zi a programarilor. Un client va incerca sa isi faca o programare in ziua si in intervalul dorit si va primi raspuns „programare reusita” sau „programare nereusita” in functie de disponibilitate. Dupa mesajul de programare reusita clientul trimite cererea de plata.

Pentru planificare se adauga o inregistrare cu urmatoarele informatii: (nume, cnp, data, locatie\_tratament, tip\_tratament, data\_tratament, ora\_tratament).

Pentru plata se adauga o inregistrare cu urmatoarele informatii: (data, cnp, suma).

Pentru anulare se sterge din inregistrarea corespunzatoare planificarii anterioare si pentru returnare bani se adauga o inregistrare cu urmatoarele informatii: (data, cnp, -suma).

Observatie: Salvarea poate fi facuta intr-o baza de date sau in fisiere de tip text.

Limbajul de implementare: Java

1. **Solutie & proiectare:**

**Solutie:** rezolvarea este impartita pe trei pachete *client, common, server*

* Pachetul *client* contine implemetarea claselor *MainClient*, *ClientWorker* si *Services*
  + *MainClient* va initializa o lista de threaduri care vor fi initialiazte cu cate un *ClientWorker* format din servicii de tip Services, in acest fel testand aplicatia cu mai multi clienti.
  + *ClientWorker* primeste in constructor un obiect de tip *Services* pentru a comunica cu serverul*.* Clasa extinde *Thread* si implementeaza metoda *run(),* unde conform cerintei va genera la fiecare 2 secunde o rezervare noua, va efectua rezervarea de tip *Reservation*, daca aceasta este valida o va palti facand un *Paymet,* iar la finalin urma generari aleatoare a unui numar 0(ok) sau 1(cancel) se va anula rezervarea daca este cazul.
  + *Services* va primi in constructor doua stringuri, host si port pentru a se conecta la server prin metoda de initializare a conexiunii. Dupa stabilirea conexiunii cu serverul serviciile vor trimite requesturi (trimise cu ajutorul metodei *sendRequest()*) spre server prin metodele suprascrise de interfata *IHealtCaresServices* si vor astepta response (citite cu ajutorul metodei *readResponse()*), iar daca raspunsul nu este cel asteptat va inchide conexiunea si va arunca exceptie. Dupa ce clientul si-a terminat treaba serviciile se vor deconecta d ela server prin metoda de close connection.
* Pachetul *common* cuprinde clasele din model (*Location, Payment, Reservation, Treatment*) si a DTO-urilor (*DTOReservation*) necesare pentru comunicarea dintre client si server
  + Clasele din model vor fi utilizate atat de client cat si de server. Acestea vor implementa interfata Serializabel pentru a putea realiza comunicarea intre cele doua parti. Campurile claselor pot fi fi observate in diagrama de clase de mai jos, fiecare camp avand getter si setter, dar si o metoata toString().
* Pachetul *server* clasele necesare pentru network, repository si service
  + Network-ul satisfice partea de comunicare dintre client si server. Comunicarea se realizeaza in jurul perechilor request – response. Sunt prezente doua clase ce implementeaza entitatile de **Request** si **Response**. Fiecare clasa are doi membrii private (Object data, Requst/Response(Type)) ce reprezinta informatia ce va fi transmisa pe socket.

**ConccurentServer** initializeaza in constructor o component **TimeOutTask**. Component **TimeOutTask** executa un thread separate ce va fi dat drumu in executie la schedelul specificat. Aceasta componenta inchide cei doi executori( **Executors.newCachedThreadPool(), Executors.newFixedThreadPool(NO\_CLIENTS)** ) folositi de serverul concurrent pentru deservirea clientilor, respective deservirea requesturilor. Odata ce ambele exeutoare au fost inchise, serverul se inchide prin apelarea metodei close().

**Serverul** prezinta o metoda de start prin care asteapta clienti. Odata ce un client s-a conectat, se creeaza un **Worker** ce implementeaza interfata Runnable. Se transmite ca referinta supraserviceul care contine metodele ce implementeaza functionalitatile din interfata IHealthCareServices si executorul pentru deservirea requesturilor.

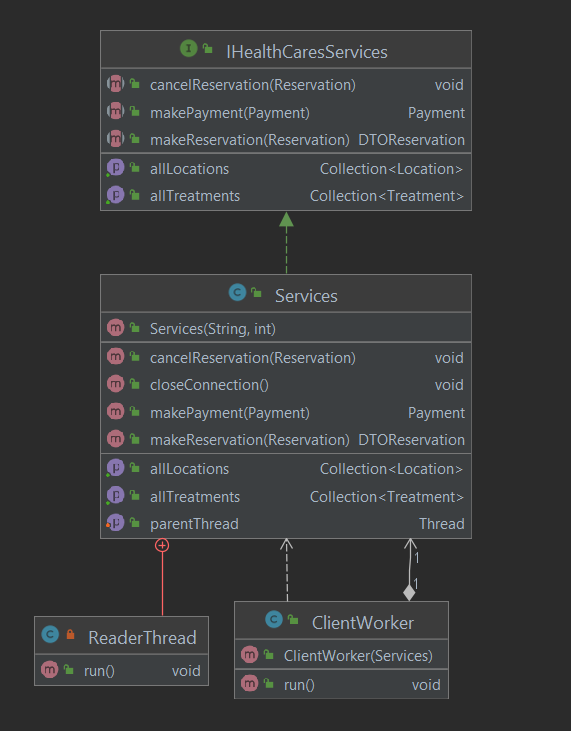
**Worker** in metoda run citeste un request din socket si il da spre procesare metodei handleRequest ce intoarce un future. In acest mod requesturile sunt deservite asincron si vom putea urmari executia lor prin intermediul future lor returnate. In metoda handleRequest() pe baza tipului requestului se determina metoda ce trebe apelata si se creeaza un callable ce va invoka aceasta metoda. Aces callable va fi submis catre executorul primi ca referinta pentru efectuarea requesturilor. Cum request urile se executa pe threaduri difierite in acelasi tip, metode din repositories sunt sincronizate precum si cele de creare si cancel appontment din ReposirtoyServices.

**InspectorWorker** implementeaza interfata Runnable. In metoa run, verifica corectitudinea operatiilor executate pe server. Inainte de a apela metoda de check, se asigura ca nu este realizata nicio operatie de cancel appointment. Realizeaz acest lucru prin intemediul lockului **isCancelledLock**, conditie **isCancelledCondition** si variabilei partajate **isCancelled.** Ele sunt folosite doar in metoda run a InspectorWorker in inmetoda cancelReservation din ReservationService.

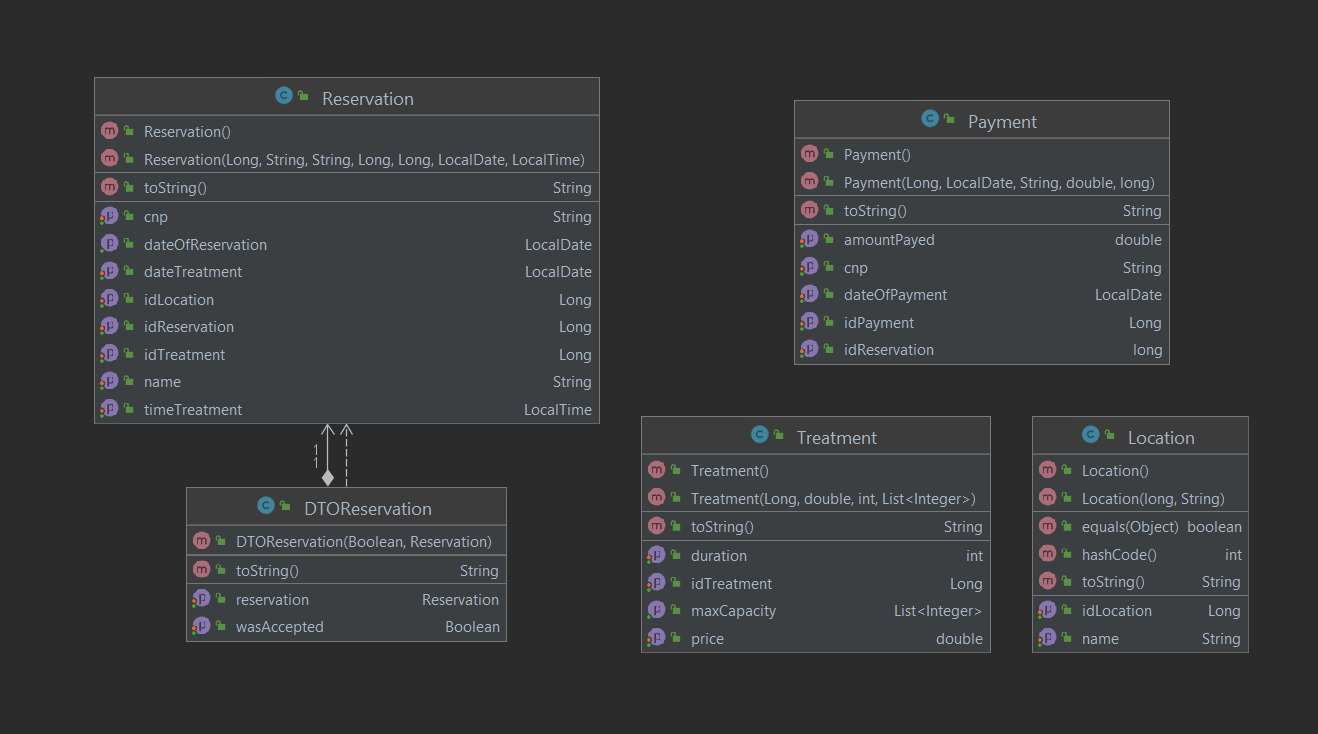
* + Repositoryul contine clase pentru fiecare tip de calsa din common. Avem o interfata Repository comuna pentru fiecare clasa. Fiecare entitate implementeaza o interfata proprie care extinde interfata generala Repository. Datele sunt persistate in fisier.
  + Service-ul implementeaza stratul de business. Exista un service pentru fiecare entitate din domeniu in care sunt implementate functionalitatile relatate in prima sectiune din aceasta documentatie corespunzatoare pentru fiecare entitate

**Proiectare:**

***Client***

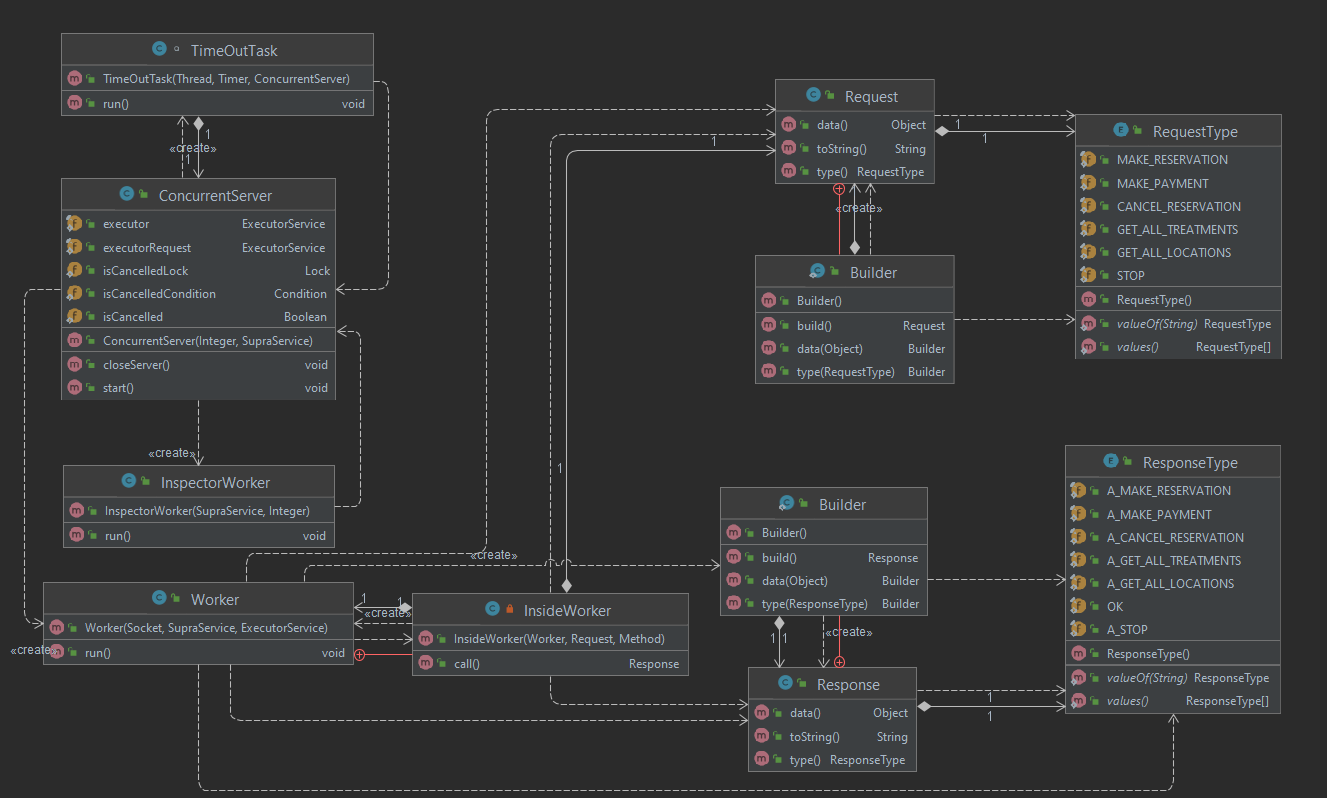
**

***Common (model)***

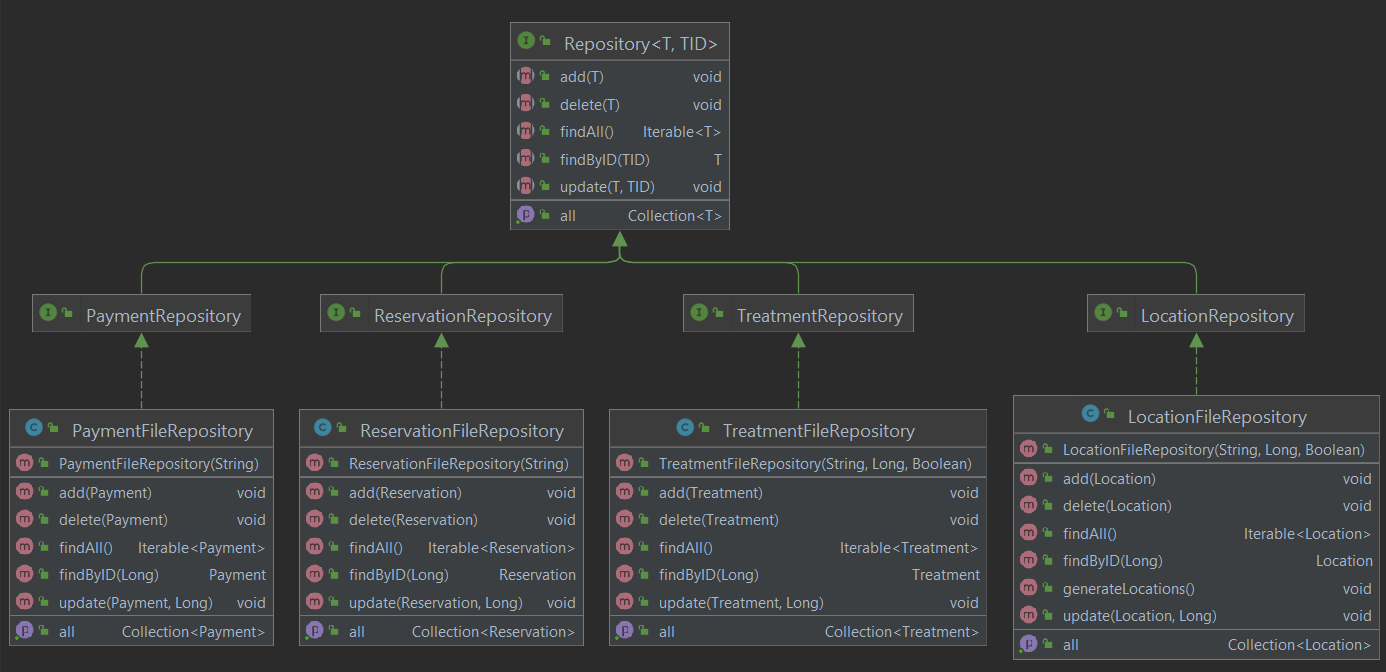
**

***Server***

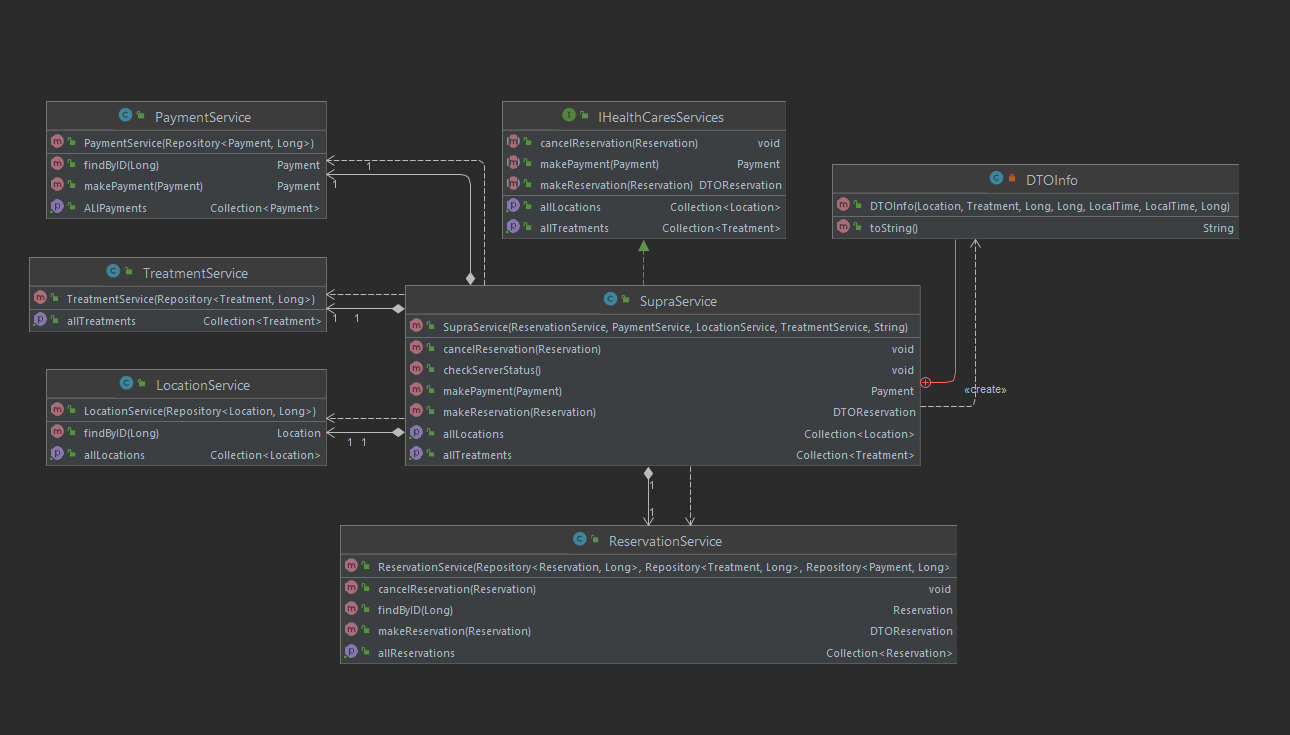
*Network*

******

*Repository*

**

*Service*

**

1. **Testare:**

Pentru testare se va considera ca fiecare client initiaza/creeaza la interval de 2 sec o noua cerere de programare folosind date generate aleatoriu (locatie, tip\_tratament, ora) si se primeste de la server o notificare – ‚programare reusita’ sau ‚programare nereusita’. Nu este necesara interfata grafica!

La generarea aleatorie a datelor pentru cererea de programare se genereaza aleatoriu si o variabila ‘anulare’ cu valori posibile 0(false) sau 1(true). In cazul in care variabila are valoarea 1 si programarea a fost reusita atunci dupa efectuarea platii se va cere anularea.

**Caz testare:**

p=10

Numar clienti=10

Tipuri de tratament n=5; Costuri tratamente: 50, 20, 40, 100, 30

Durate tratamente (minute): 120, 20, 30, 60, 30

Numar maxim de clienti care se pot face simultan un tratament:

N(1,1)=3, N(1,2)=1, N(1,3)=1, N(1,4)=2, N(1,5)=1; N(i,j) =N(1,j)\*(i-1) pentru toti i>1 si 0<j<=5

Aplicatia pune la dispozitie mesaje in consola si un fisier de logger (logs.txt) unde poate fi urmatit faptul ca toate calculele au fost realizate corect. De asemenea periodic sistemul (2 cazuri testare: 5, 10 secunde) face o verificare a programarilor facute si a incasarilor corespunzatoare prin verificarea corespondentei corecte intre numarul programarile facute si sumele incasate dar si calcularea soldului total. Prin aceasta actiune se verifica si faptul ca nu sunt suprapuneri in planificari (numarul total de clienti programati la locatia i, pentru tratamentul T(j) la un anumit timp nu este mai mare decat numarul maxim N(i,j) admis).