Proiect echivalent examenului scris

Pornind de la o baza de date deja construita in FN3, creati un program care permite utilizatorului sa faca operatii CRUD pe aceasta baza de date (inserare, stergere, editare, vizualizare). Pentru aceasta cerinta se pot folosi adnotari JPA, iar pentru conectarea bazei de date la IntelliJ se va folosi PosgreSQL. La final scrieti pasii de rulare a aplicatiei pentru oricine ar dori sa testeze proiectul.

Documentatie:

* <https://www.jetbrains.com/idea/guide/tutorials/your-first-spring-application/creating-spring-boot-application/> (pentru crearea aplicatiei);
* <https://www.jetbrains.com/help/idea/postgresql.html> (pentru legarea bazei de date de IntelliJ);
* <https://www.bezkoder.com/spring-boot-postgresql-example> (pentru dependente si structura proiectului, respectiv clase repository, controller, model, data to object-DTO).
* <https://medium.com/javarevisited/hibernate-vs-jpa-vs-spring-data-jpa-ff4485aaa780> (hibernate)

Principalele elemente de programare utilizate:

* Am configurat proeictul de tip *Spring* pentru a seta dependentele si a avea acces la *Jakarta Persistence,* dar si pentru alte utilitati (@Autowired-injecteaza instanta portrivita in obiectul care solicita dependenta, @Service-depedentele sunt injectate mai usor, @Repository-furnizeaza metode si operatii pentru a manipula datele, gestionare automata a exceptiilor);
* Am utilizat framework-ul *Hibernate* pentru a scrie mai usor operatii CRUD, interogari asupra bazei de date si pentru a lucra direct cu obiectele din Java.
* Am folosit *Jakarta Persistence (JPA)* pentru stocarea, accesul si gestionarea datelor intre obiectele din Java si baza de date din PostgreSQL (ex: @Entity, @Column, @Id etc);
* Ca si plug-in am utilizat Lombok - genereaza automat getteri, setteri, constructori, metoda *toString()* ca sa afisam informatiile personalizat.

Pentru organizarea proiectului am folosit mai multe clase, asociate unei entitati:

* *‘Entity’*- pentru fiecare entitate din baza de date am creat o clasa unde am implementat si plugin-ul *Lombok*. Astfel se realizeaza accesul la atributele fiecarei entitati;
* *‘Entity’DTO*(data to object) – am separat functionalitatile clientului de cele ale server-ului. In aplicatia curenta, in aceste clase, nu se mai afla cheia primara corespunzatoare fiecarei entitati intrucat aceasta este autoincrementabila, astfel ca am lasat disponibile celelate atribute, execeptand ‘idEntity’.
* *‘Entity’Repository* - pentru fiecare entitate se defineste o interfata care extinde o alta interfata si aume *JPA Repository*. Prin faptul ca *‘Entity’Repository*’ mosteneste aceasta interfata, metodele predefinite pentru operatiuni CRUD sunt deja implementate. Repository-urile interactioneaza direct cu baza de date, astfel ca nu mai este necesar sa fie scrise *query-uri* manual;
* *‘Entity’Service’*- in aceasta clasa sunt create metodele pentru operatiile CRUD pe baza de date, precum si metode de filtrare si sortare.

Deviere de la baza de date initiala:

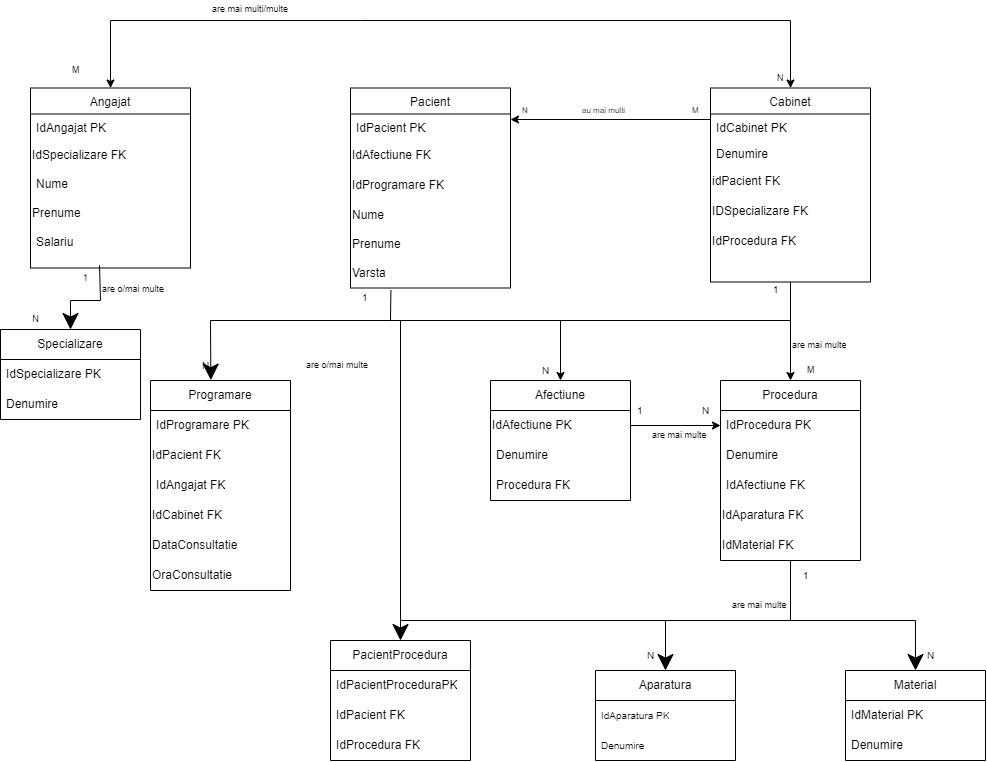


Tabela ‘Cabinet’ va fi stearsa- nu ne mai intereseaza in ce cabinet are loc consultatia, nr de angajati corespunde cu numarul de cabinete si atunci nu se vor suprapune.

Tabela ‘PacientProcedura’ va fi stearsa pentru ca putem gasi imediat print-o filtrare ce tip procedura va efectua pacientul selectat.

Am refacut si legaturile dintre tabele, si sunt prezente urmatoarele constrangeri:

* ADD CONSTRAINT fk\_pacient\_afectiune

FOREIGN KEY (IdAfectiune) REFERENCES Afectiune(IdAfectiune) ON DELETE SET NULL ON UPDATE CASCADE;

* ADD CONSTRAINT fk\_pacient\_programare

FOREIGN KEY (IdProgramare) REFERENCES Programare(IdProgramare) ON DELETE SET NULL ON UPDATE CASCADE;

* ADD CONSTRAINT fk\_programare\_pacient

FOREIGN KEY (IdPacient) REFERENCES Pacient(IdPacient) ON DELETE CASCADE ON UPDATE CASCADE;

* ADD CONSTRAINT fk\_programare\_angajat

FOREIGN KEY (IdAngajat) REFERENCES Angajat(IdAngajat) ON DELETE SET NULL ON UPDATE CASCADE;

* ADD CONSTRAINT fk\_afectiune\_procedura

FOREIGN KEY (IdProcedura) REFERENCES Procedura(IdProcedura) ON DELETE SET NULL ON UPDATE CASCADE;

* ADD CONSTRAINT fk\_procedura\_afectiune

FOREIGN KEY (IdAfectiune) REFERENCES Afectiune(IdAfectiune) ON DELETE SET NULL ON UPDATE CASCADE;

* ADD CONSTRAINT fk\_procedura\_aparatura

FOREIGN KEY (IdAparatura) REFERENCES Aparatura(IdAparatura) ON DELETE SET NULL ON UPDATE CASCADE;

* ADD CONSTRAINT fk\_procedura\_material

FOREIGN KEY (IdMaterial) REFERENCES Material(IdMaterial) ON DELETE SET NULL ON UPDATE CASCADE;

Contributie la proiect:

Amandoua: cautare documentatie, refactorizarea bazei de date si adaugarea noilor constrangeri, arhitectura proiectului, rezolvare bug-uri.

Preda Paula-Maria:

* Adaugarea dependentelor in Spring boot la crearea proiectului;
* Crearea si organizarea (service, repository, entity);
* Adaugarea metodelor pentru operatii CRUD in fiecare clasa;
* Crearea meniului aplicatiei pentru operatii CRUD;
* Adaugarea metodelor de filtrare si sortare;

Rentea Diana-Andreea:

* Realizarea conexiunii bazei de date in application.properties;
* Refactorizare clase;
* Back-up baza de date;
* Refactorizarea claselor cu plugin-ul Lombok si a unor dependente din pom.xml;
* Scrierea documentatiei si a pasilor de rulare ai aplicatiei;