UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

PROIECT
INGINERIE SOFTWARE

COLTEA ALEXANDRA

HADARAU IOANA

HADARAU ELENA

CUPRINS

1.INTRODUCERE

1.1. Contextul temei

Într-o lume plină de dragoste pentru animale, s-a născut o aplicație revoluționară, avand front- endul implementat in React și Spring pentru back-end, având misiunea de a facilita comunicarea dintre deținătorii de animale și doctorii veterinari.

Cu un cont personalizat, utilizatorii pot centraliza informații esențiale despre ei înșiși, precum și detalii relevante despre fiecare animal de companie. Prin intermediul aplicației, fiecare animal are un medic veterinar dedicat, iar proprietarii pot comunica direct cu aceștia pentru a ține sub control evoluția sănătății animalelor lor.

Fiecare doctor veterinar primește feedback de la animalele pe care le monitorizează, iar aceste pareri sunt vizibile tuturor utilizatorilor în momentul în care caută un medic potrivit. Astfel, utilizatorii pot alege cu încredere un doctor veterinar, având acces la experiențele anterioare ale altor proprietari de animale. Pe lângă această funcționalitate esențială, aplicația oferă o pagina publica, accesibil fără cont, unde utilizatorii pot găsi articole informative despre medicina veterinară, zootehnie și bunăstarea animalelor.

Pagina de home impresionează cu ultimele articole de actualitate, ținând utilizatorii captivați și încurajându-i să exploreze conținutul bogat al aplicației.

Profilurile doctorilor sunt reprezentative, evidențiind specializarea și feedback-ul primit, iar istoricul medical al animalelor este la dispoziția lor pentru a anticipa eventuale pericole.

Toate conturile, fie ele de utilizator sau doctor, sunt gestionate cu grijă de un cont de admin, asigurându-se că înregistrarea este validă și putând face modificări acolo unde este necesar. Astfel, această aplicație inovatoare aduce împreună iubitorii de animale și profesioniștii veterinari într-o comunitate dedicată bunăstării și sănătății animalelor de companie.

1.2. Motivatie

Echipa noastră este motivată de pasiunea și devotamentul față de bunăstarea animalelor. Avem încredere că prin crearea acestei platforme, putem aduce împreună o comunitate care să sprijine și să încurajeze schimbul de informații și experiențe legate de sănătatea și fericirea animalelor de companie. Ne dorim să oferim posesorilor de animale un loc unde pot găsi răspunsuri la întrebările lor, să împărtășească întâmplări și să beneficieze de sfaturi de la profesioniștii veterinari.

În plus, suntem motivate de ideea de a facilita comunicarea între proprietarii de animale și medicii veterinari, creând astfel o legătură mai strânsă și eficientă între comunitatea dedicată animalelor și profesioniștii care le pot oferi îngrijirea adecvată. Prin acest proiect, ne propunem să contribuim la îmbunătățirea vieții animalelor de companie și să construim o comunitate solidă în jurul acestei pasiuni comune.

1.3 Etapele Cheie ale dezvoltarii proiectului

- **Definirea cerințelor**
- **Proiectarea arhitecturii & Alegerea tehnologiilor**
  - Am ales sa structuram proiectul astfel : componenta de frontend e gestionata in React, iar cea de Backend in Spring
  - Comunicarea dintre Spring (backend) și React (frontend) poate fi realizată prin intermediul cererilor HTTP. Una dintre librăriile populare pentru gestionarea cererilor HTTP în React este Axios, pe care am decis sa o folosim si noi
- **Configurarea mediului de dezvoltare**
- **Dezvoltarea backend-ului cu Spring**
- **Dezvoltarea frontend-ului cu React**
- **Gestionarea stării aplicației**
- **Testare**

2. STUDIU BIBLIOGRAFIC

2.1 Tehnica utilizată

Alegerea de a implementa acesta aplicatie folosind Spring pentru backend și React pentru frontend poate fi influențată de mai mulți factori.

Iată câteva motive pentru care această combinație este o alegere bună:

1. **Extensibilitate și modularitate**
2. **Separarea responsabilităților**
3. **Comunitate și suport**
4. **Performanță**

Voi descrie cateva elemente cheie pentru tehnologiile folosite:

Java Spring:

Spring Boot este un framework Java care facilitează crearea și rularea aplicațiilor Java. Acesta simplifică procesul de configurare și instalare, permițând dezvoltatorilor să se concentreze mai mult pe scrierea de cod pentru aplicațiile lor.

Spring boot este dezvoltat de Pivotal Team și oferă o modalitate mai rapidă de a configura și un mod mai ușor, de a configura și de a rula atât aplicații simple, cât și aplicații bazate pe web. Este o combinație între Spring Framework și Embedded Servers. Scopul principal al Spring Boot este de a reduce timpul de dezvoltare, de testare a unității și de testare a integrării, iar în Spring Boot nu este necesară configurația XML.

JPA:

JPA este o specificație Java (Jakarta Persistence API) și gestionează datele relaționale în aplicațiile Java. Pentru a accesa datele între un obiect Java(Plain Old Java object)/clasă și o bază de date relațională, putem utiliza JPA. JPA ofera o platforma pentru a lucra direct cu obiecte in loc sa folosim instructiuni SQL.

React:

React (cunoscut și sub numele de React.js sau ReactJS) este o bibliotecă JavaScript front-end gratuită și open-source pentru construirea de interfețe utilizator bazate pe componente.

React aderă la paradigma de programare declarativă.Dezvoltatorii proiectează vizualizări pentru fiecare stare a unei aplicații, iar React actualizează și redă componentele atunci când datele se schimbă. Acest lucru este în contrast cu programarea imperativă.

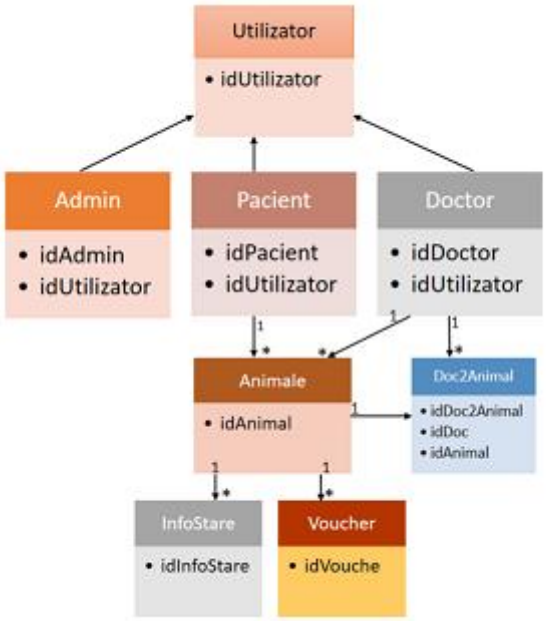
Codul React este alcătuit din entități numite componente. Aceste componente sunt modulare și reutilizabile. Aplicațiile React constau, de obicei, din mai multe straturi de componente. Valorile interne ale unei componente se numesc starea acesteia.

MySQL:

MySQL este un sistem de management al bazelor de date relaționale (RDBMS) opensource. O bază de date relațională organizează datele în unul sau mai multe tabele de date în care datele pot fi legate între ele; aceste relații ajută la structurarea datelor. SQL este un limbaj pe care programatorii îl folosesc pentru a crea, modifica și extrage date din baza de date relațională, precum și pentru a controla accesul utilizatorilor la baza de date. Pe lângă bazele de date relaționale și SQL, un RDBMS precum MySQL lucrează cu un sistem de operare pentru a implementa o bază de date relațională în sistemul de stocare al unui computer, gestionează utilizatorii, permite accesul în rețea și facilitează testarea integrității bazei de date și crearea de copii de rezervă.

3. ANALIZA & DESIGN

3.1 Diagrama de Clase



3.2 Diagrama de UseCase

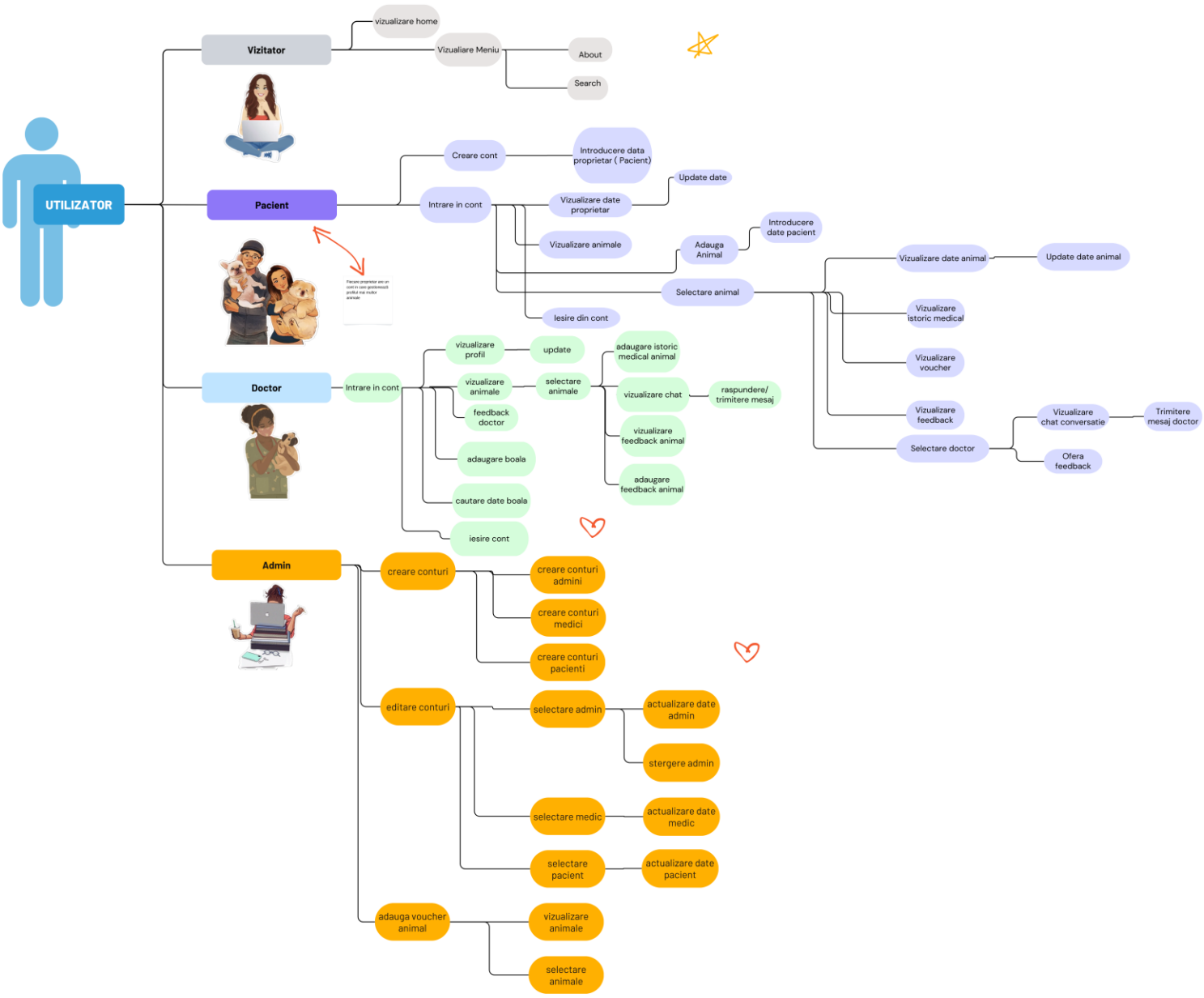
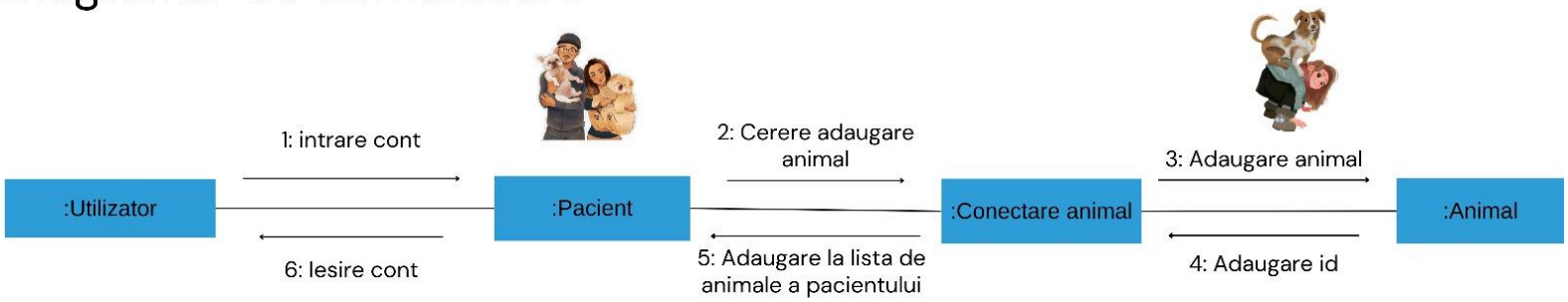


Diagrama de comunicare



(Alexandra)

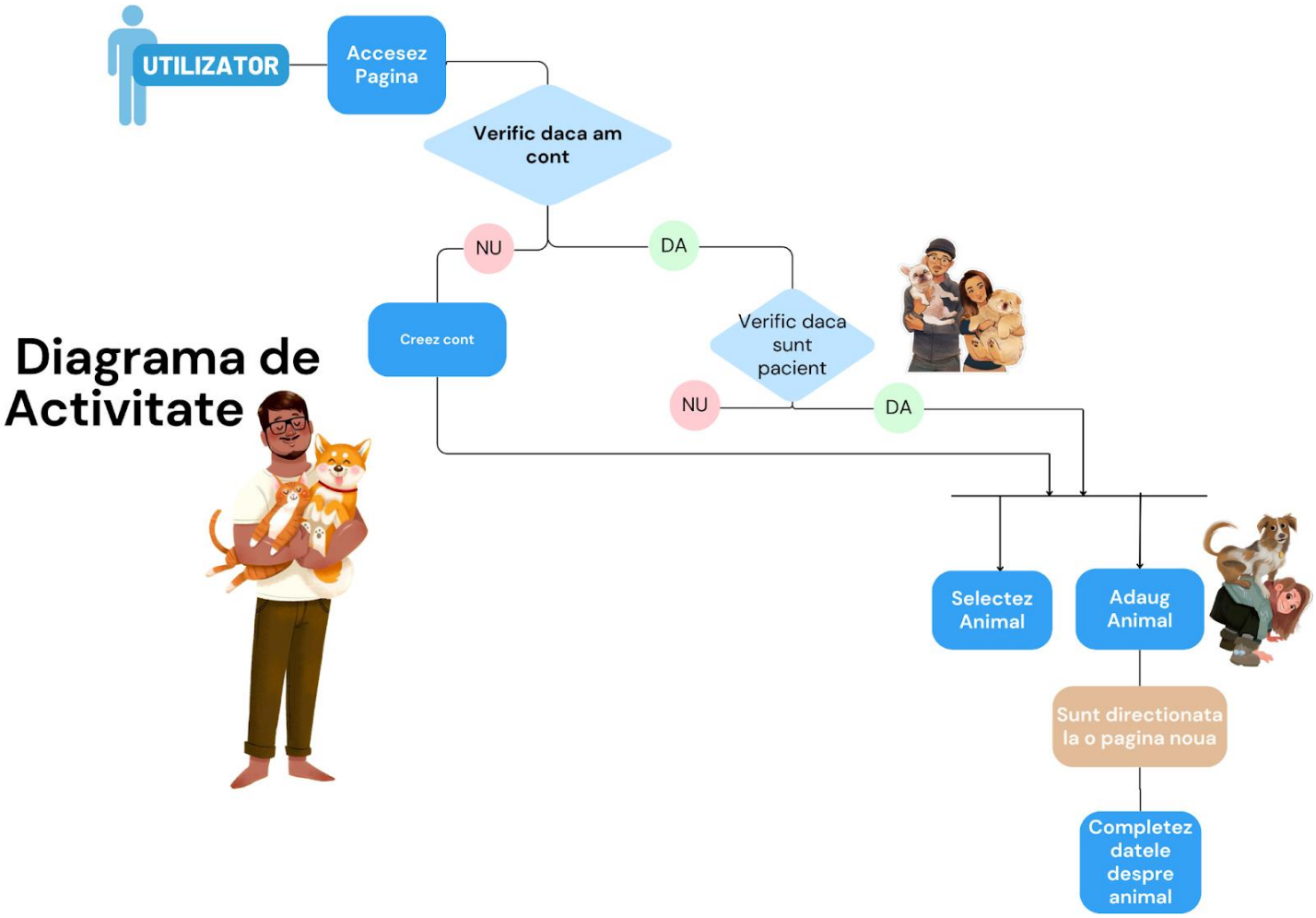


Diagrama de Activitate
UTILIZATOR
Accesez Pagina
Verific daca am cont
NU
DA
Creez cont
Verific daca sunt pacient
NU
DA
Selectez Animal
Adaug Animal
Sunt directionata la o pagina noua
Completez datele despre animal

(Ioana)

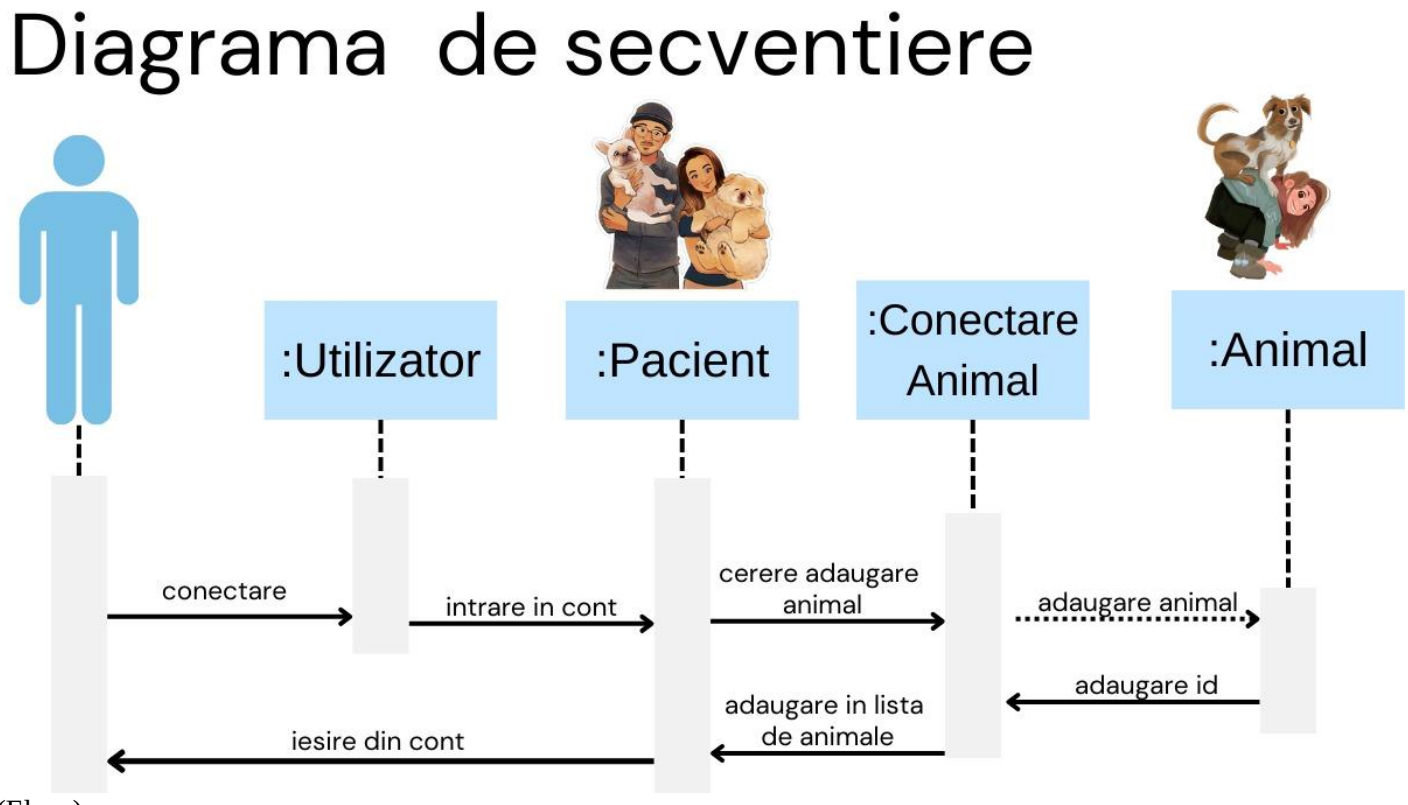


Diagrama de secventiere
:Utilizator
:Pacient
:Conectare Animal
:Animal
conectare
intrare in cont
cerere adaugare animal
adaugare animal
adaugare id
adaugare in lista de animale
iesire din cont

(Elena)

4.IMPLEMENTARE

4.1 DESIGN PATTERNS

(Ioana)

- SINGLE TON:

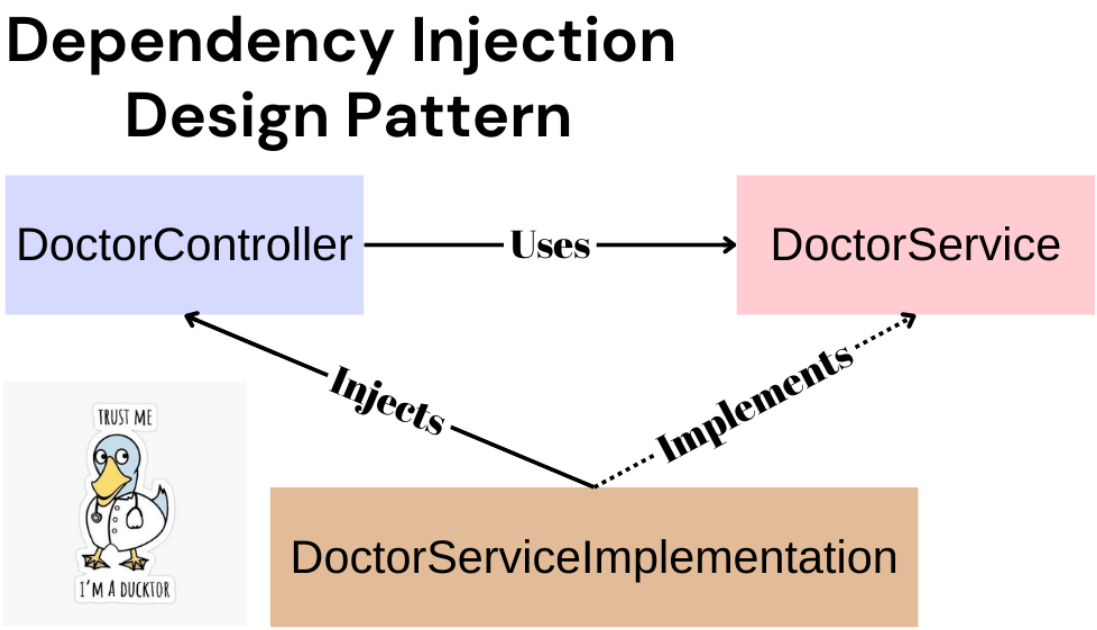
Asigură că o clasă are doar o singură instanță.

Furnizează un punct global de acces la acea instanță. Adică, șabloanele Singleton permit accesul la o instanță de oriunde în program. Aceasta rezolvă problema globală a accesului la obiecte esențiale, fără a le permite să fie suprascrise de alte porțiuni de cod.

```
@Service
@RequiredArgsConstructor
public class AnimalServiceImplementation implements AnimalService
```

- DEPENDENCY INJECTION (DI) este un design pattern în programare orientată pe obiect care vizează inversarea controlului asupra dependențelor. În loc ca o clasă să creeze obiectele de care depinde direct, aceste obiecte sunt furnizate (injectate) de la exterior. Scopul este de a facilita testarea, decuplarea componentelor și gestionarea mai ușoară a dependențelor.

(Elena)



În contextul unei aplicații Spring, Dependency Injection este puternic încorporat în framework-ul Spring. Spring utilizează DI pentru a gestiona și furniza dependențele necesare în aplicație. Aceasta se realizează prin intermediul containerului Spring care se ocupă de crearea și administrarea obiectelor.

Clasa DoctorServiceImplementation: Această clasă este marcată cu **@Service**, ceea ce înseamnă că este recunoscută de Spring ca fiind o componentă gestionată. Dependențele acestei clase sunt injectate prin intermediul

constructorului marcat cu **@RequiredArgsConstructor**. Aceasta se numește constructorul de injecție a dependențelor.

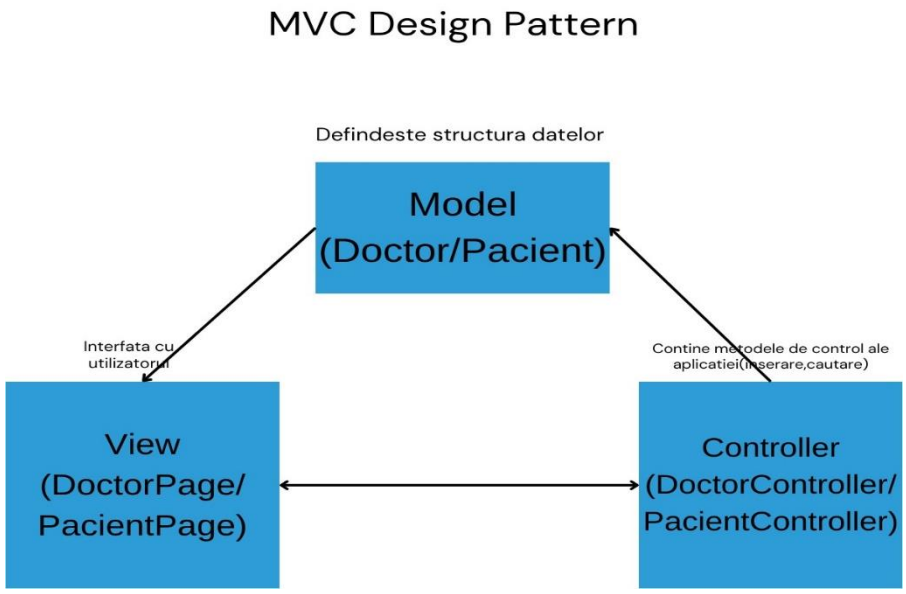
Clasa DoctorController: primește dependențe prin intermediul constructorului, adică prin adnotarea **@RequiredArgsConstructor**. Aceste dependențe sunt serviciile (doctorServiceImplementation etc) necesare pentru a gestiona operațiile specifice controlerului.

Interfața DoctorService:Interfața definește contractul pentru serviciile asociate entității Doctor.

Implementarea concretă (DoctorServiceImplementation) primește dependențele sale prin constructor, în acest caz, repository-urile asociate (DoctorRepo, UtilizatorRepo, Doc2AnimalRepo).

- MVC (Model-View-Controller) este un design pattern utilizat în mod obișnuit pentru a implementa interfețe utilizator, date și logică de control. Acesta pune accentul pe o separare între logica software-ului și afișare. Această "separare a preocupărilor" asigură o mai bună diviziune a muncii și o mai bună întreținere.

(Alexandra)



Cele trei părți ale design patternului MVC pot fi descrise după cum urmează:

Model: Gestionează structura datelor cu care se lucreaza.

View: Gestionează aspectul și afișarea. Partea vazuta de utilizator(interfata).

Controller: Se ocupa de partea logica a aplicatiei. Preia datele de la interfata si le manipuleaza astfel incat sa poate fi intelese de aplicatie.

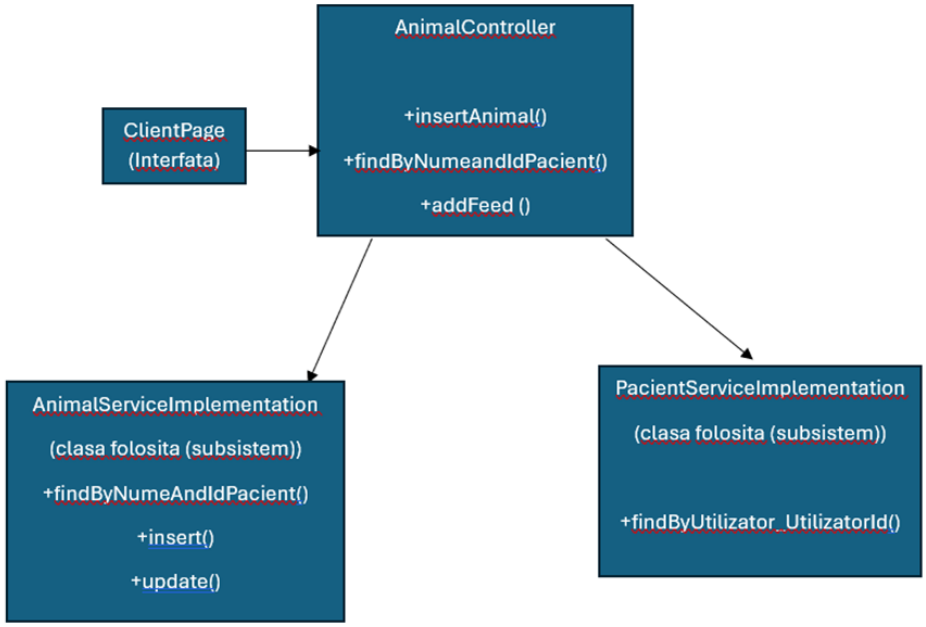
Modelul ne ofera date legate de obiecte precum doctor, pacient, animal, etc. Legatura dintre baza de date si clasele din java se face cu ajutorul JPA (Java Persistence API). View-ul este reprezentat de aplicatia web descrisa cu ajutorul react. Controllerul contine metode care creeaza , actualizeaza si cauta obiecte(conține logica care actualizează modelul și/sau vizualizarea ca răspuns la informațiile primite de la utilizatorii aplicației) Pentru a conecta Controllerul si View-ul am folosit requesturi de tip axios.post .

- FACADE DESIGN PATTERN

(Echipa VetHealth)

> O fațadă este o clasă care oferă o interfață simplă pentru un subsistem complex care conține o mulțime de părți mobile, ea include doar acele caracteristici care interesează cu adevărat clienții.

AnimalController este o clasa fatada deoarece include doar metode de adaugareAnimal, editare si cautare(lucruri de care un utilizator este interesat in mod direct). Aceasta clasa continue in spate un subsistem complex de clase, cum sunt AnimalServiceImplementation, Animal, Pacient, etc. In cadrul subsitemului se realizaeaza actiunile propriu-zise de modificare a obiectelor si bazei de date.



5.READ ME:

Specificatii:

Este necesar sa existe Java Development Kit pe sistemul pe care se doreste rularea aplicatiei. De asemenea este nevoie de un IDE compatibil cu Java(recomandat: IntelliJ).

Rulare aplicatie:

- Se creeaza o baza de date MySQL in care se vor salva informatiile din timpul folosirii aplicatiei
- Se deschid cele doua proiecte (backend si frontend) in IntelliJ sau un alt IDE care suporta JAVA
- Se modifica username(spring.datasource.username) si password(spring.datasource.password) din fisierul application.properties din partea de backend a aplicatiei pentru a coincide cu configurarea bazei de date
- Se ruleaza DemoApplication din proiectul demo(backend)
- Intr-un terminal deschis in IDE unde este proiectul frontend, se ruleaza instructiunea npm run start care deschide aplicatia react in browser pe localhost.

6.Concluzii

Proiectul dezvoltat cu Spring pentru backend și React pentru frontend, concentrat pe îmbunătățirea comunicării între iubitorii de animale și doctorii veterinari, oferă o soluție complexă și personalizată pentru gestionarea sănătății animalelor. Cu patru moduri de utilizare diferite (public, admin, doctor și pacient), fiecare cu funcționalități specializate, aplicația aduce multiple beneficii pentru toți participanții.

Prin furnizarea unui mediu virtual eficient, proiectul facilitează programările, monitorizarea atentă a animalelor și schimbul de informații relevante. Interfața prietenoasă dezvoltată cu React asigură accesibilitatea și confortul utilizatorilor în navigarea și utilizarea aplicației.

Funcționalitățile precum înregistrarea istoricului medical, feedback-ul, și achiziționarea de vouchere contribuie la gestionarea relațiilor cu clienții și îmbunătățesc calitatea serviciilor oferite de doctorii veterinari.

În concluzie, proiectul nu numai că răspunde nevoilor din industrie privind îngrijirea animalelor, dar și oferă o platformă solidă și versatilă, cu potențial de extindere și îmbunătățire continuă pe baza feedback-ului și evoluției necesităților utilizatorilor