

Студијски програм:Информатика

Предмет: Програмски језици

Програмски језици

- Ветеринарска амбуланта –

|  |  |
| --- | --- |
| Предметни наставник: | Студент: |
| др Александар Мишковић | Лазар Бирташевић, 006/2022 |

Крагујевац 2024.

# Поставка задатка

**Циљ пројекта**: Циљ је развити *RESTful wеб* сервис за управљање активностима зубарске ординације користећи *Spring Boot* 3.0 и *JDK* 17. Сервис ће омогућити управљање пацијентима (животињама) ,ветеринарима, заказивањима, третманима и евиденцијом посета. Подаци ће бити смештени у *MySQL* бази података, а *Maven* ће се користити за управљање зависностима.

**Функционалности**:

а) Управљање пацијентима: Додавање, измена и брисање података о пацијентима, укључујући врсту, расу, имена власника и контакт власника.

б) Управљање ветеринарима: Регистрација нових ветеринара, измена података и преглед распореда и специјализација

ц) Заказивање посета: Праћење и управљање различитим врстама денталних третмана који се пружају.

д) Управљање третманима: Праћење и управљање различитим врстама ветеринарских

третмана који се пружају.

e) Евиденција посета: Праћење извршених посета и услуга које су пружене.

**Технички Захтеви**:

а) Backend: Фокус на развоју бацкенд апликације.

б) Логовање: Имплементација логовања за праћење активности корисника и система.

ц) Multi-layer: Имплементација вишеслојне архитектуре.

д) JSON: Комуникација између клијента и сервера користећи ЈСОН формат. е) Тестирање: Тестирање сервиса коришћењем Постман-а.

ф) Покретање нити: Имплементација нити која ће аутоматски проверавати статус пацијената и слати подсетнике власницима о заказаним терминима након 10 секунди од старта апликације.

**Структура Података**:

а) Пацијент: Врста, раса, име, година рођења, име власника, контакт власника.

б) Ветеринар: Име, презиме, специјализација, контакт информације.

ц) Заказивање: Датум, време, врста третмана, забелешке ветеринара.

д) Третман: Опис третмана, цена, трајање.

е) Посета: Датум посете, ветеринар, пацијент, услуге пружене, забелешке.

Остале класе и интерфејси: Пројекат садржи више класа разврстаних у духу вишеслојне

архитектуре (Controller, Service, Repository, Component, Logger).

**Документација**: Детаљна документација о *API endpoint*-има, примери употребе и логови активности. Писати је према приложеном упутству.

**Завршна напомена**: Пројекат треба да илуструје способност развоја комплексних *RESTful wеб* сервиса, ефикасно управљање базама података, примену вишенитног програмирања и аспектног програмирања у контексту *Spring Boot* апликација.

Садржај

[Поставка задатка 2](#_Toc176305610)

[1.Увод 4](#_Toc176305611)

[2.Конфигурација 5](#_Toc176305612)

[3.База података 1](#_Toc176305613)

[4.Апликација 2](#_Toc176305614)

[1. Пакет entity (Ентитети) 14](#_Toc176305615)

[2. Пакет repository (Репозиторијуми) 14](#_Toc176305616)

[3. Пакет service (Сервиси) 14](#_Toc176305617)

[4. Пакет controller (Контролери) 15](#_Toc176305618)

[Закључак 15](#_Toc176305619)

[5.Logging 16](#_Toc176305620)

[6. Класа ScheduledTasks 18](#_Toc176305633)

[Литература 19](#_Toc176305634)

# 1.Увод

У пројектном задатку имао сам да направим апликацију засновану на

*RESTful* сервису за потребе ветеринарске амбуланте.

# 2.Конфигурација

2.1 **Конфигурација *Maven* фајла**

Mavan је алат за управљање пројектима и аутоматизацију процеса изградње софтверских апликација. Razvijen je od strane *Apache Software Foundation*-a i koristi se u razvoju softvera kako bi se olakšalo upravljanje zavisnostima, kompilacijom, testiranjem i izgradnjom projekata.

Ево неколико кључних карактеристика и функција које Мавен пружа:

**Управљање зависностима**: Мавен омогућава програмерима да једноставно дефинишу зависности својих пројеката преко "*POM*" (*Project Object Model*) датотеке. Мавен аутоматски преузима и управља библиотекама и другим компонентама потребним за пројекат.

* + - 1. **Аутоматизација изградње**: Мавен дефинише стандардне кораке за изградњу пројекта, као што су компилација, тестирање, паковање и издање. Ови кораци се извршавају аутоматски приликом извршавања Мавен циљева.
      2. **Стандардизована структура пројекта**: Мавен промовише одређену структуру директоријума за пројекте, чиме се олакшава организација изворног кода, ресурса и конфигурација.
      3. **Централни репозиторијум**: Мавен има централни репозиторијум где се чувају библиотеке и артефакти који су доступни за поновну употребу у различитим пројектима. То смањује потребу за ручним преузимањем и управљањем библиотека.
      4. **Плагинови**: Мавен подржава различите плагинове који додају функционалности пројекту. На пример, постоје плагинови за извршење тестова, генерисање извештаја, паковање апликација и још много тога.
      5. **Декларативна конфигурација**: Мавен користи XML датотеке за дефинисање конфигурације пројекта, што омогућава декларативан приступ уместо скриптирања корака изградње.

Коришћењем *Maven*-а, програмери могу значајно поједноставити процес изградње, управљања зависностима и управљања пројектима, чиме се повећава ефикасност и олакшава тимски рад.

Моја апликација садржи сависности о “*Spring Boot*”, “*Spring Data JPA*”, “*Spring Boot Web Starter*”, “*Spring Boot devtools*”, “*MySql Connector J*”, “*Spring Starter Test*”

Овако изгледа мој Maven (pom.xml) фајл:

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.3.3</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.example</groupId>  
 <artifactId>demo</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>demo</name>  
 <description>Demo project for Spring Boot</description>  
 <url/>  
 <licenses>  
 <license/>  
 </licenses>  
 <developers>  
 <developer/>  
 </developers>  
 <scm>  
 <connection/>  
 <developerConnection/>  
 <tag/>  
 <url/>  
 </scm>  
 <properties>  
 <java.version>17</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-logging</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter</artifactId>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <excludes>  
 <exclude>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 </exclude>  
 </excludes>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

2.1 **Конфигурација *Property* фајла**

*Property* фајл (такође познат и као *properties* фајл) се користи за дефинисање конфигурационих параметара и вредности које се користе у апликацији.

Ево неколико кључних разлога зашто се користе *property* фајлови у *Spring*-у:

* + - 1. **Конфигурација апликације**: *Property* фајлови омогућавају да се различите конфигурационе опције, као што су путање до ресурса, адресе база података, URL-ови сервиса и други параметри, поставе изван изворног кода. То олакшава прилагођавање понашања апликације без потребе за променама у коду.
      2. **Раздвајање окружења**: Property фајлови омогућавају постављање вредности које се разликују између различитих окружења (нпр. развој, тестирање, продукција). На тај начин, исти код апликације може да се користи у различитим окружењима са различитим конфигурацијама.
      3. **Лакше одржавање**: Када се конфигурациони параметри издвоје у одвојени фајл, промене у конфигурацији могу да се врше без потребе за рекомпилацијом или поновном изградњом апликације. Ово олакшава брзе промене и одржавање система.
      4. Сигурност: Осетљиве информације као што су лозинке и кључеви могу да се чувају у одвојеним property фајловима и да се заштите од јавног увида.
      5. Међународизација (i18n): У *Spring* апликацијама, *property* фајлови често се користе за подршку међународизацији, тј. локализацији апликације на различите језике. Различити *property* фајлови могу садржавати текст на различитим језицима, чиме се омогућава динамичко приказивање одговарајућих текстова корисницима.

Property фајлови у *Spring*-у могу се читати и користити помоћу одговарајућих класа и метода из *Spring* оквира, као што су *PropertySourcesPlaceholderConfigurer* и @*Value* анотација. Најчешће коришћени формат за property фајлове је *.properties*, али се такође може користити и *.yml* формат (*YAML*) за више комплексних конфигурационих опција. Ја сам користио *.property* екстензију.

Овако изгледа мој property фајл:

server.port=8080  
spring.datasource.url=jdbc:mysql://localhost:8889/ambulance  
spring.datasource.username=lazar  
spring.datasource.password=password  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true

Ово је конфигурациони фајл за апликацију. Ево објашњења.

 server.port=8080: Подешава порт на коме ће апликација слушати (у овом случају 8080).

 spring.datasource.url=jdbc:mysql://localhost:8889/ambulance: URL за повезивање на MySQL базу података која се налази на локалном серверу на порту 8889, са именом базе ambulance.

 spring.datasource.username=lazar: Корисничко име за приступ бази података.

 spring.datasource.password=password: Лозинка за приступ бази података.

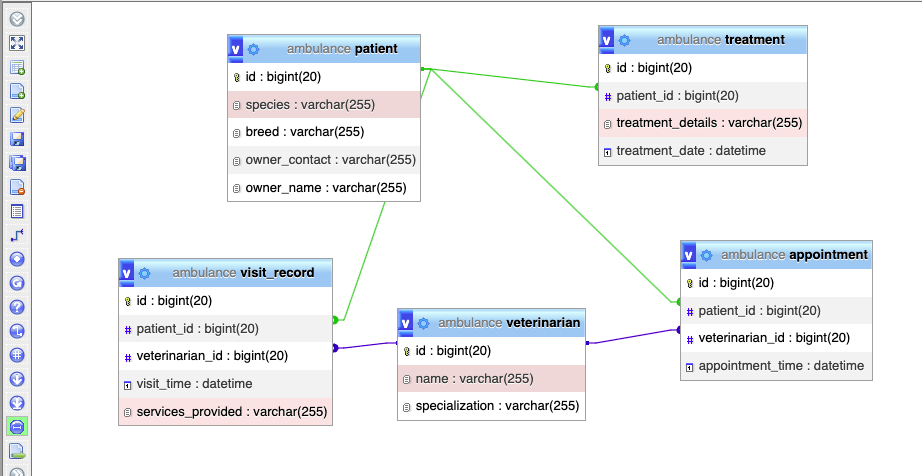
 spring.jpa.hibernate.ddl-auto=update: Подешавање за Hibernate које аутоматски ажурира шему базе података на основу JPA ентитета.

 spring.jpa.show-sql=true: Омогућава приказивање SQL упита у конзоли ради лакшег дебаговања.

# 3.База података

Ја сам радио у mySql-у. Базу покрећем преко Apache сервера и PHPMyAdmin- а. Све то покрећем преко апликације XAMPP.

Овако изгледаju мојe табелe из базе података



# 4.Апликација

Моја апликација је пратила *DAO* шаблон за пројектовање (*DAO design pattern*)

,,*DAO*" (*Data Access Object*) дизајн шаблон је један од шаблона који се користи

у архитектурном облику апликације, посебно у сложеним апликацијама које комуницирају са базама података. Овај шаблон има за циљ изоловати детаље комуникације са базом података од остатка апликације, чиме се подстиче боља одвојеност и прегледност кода.

1. **Представнички слој (*Presentation Layer*)**: Овај слој садржи кориснички интерфејс апликације, као што су веб странице, апликациони интерфејси (*API*-ји) или кориснички интерфејси на графичком корисничком интерфејсу. Овде се обично одвија обрада корисничких захтева и приказивање података.
2. **Логички слој (*Business Logic Layer*)**: Познат као и *Service layer*. Овај слој садржи бизнис логику апликације. Он обрађује податке који се добијају из представничког слоја и представља главну логику апликације.
3. **Слој података (*Data Layer*)**: Овде се налазе "*DAO*" компоненте које су одговорне за комуникацију са базом података. Оне обезбеђују интерфејсе за креирање, читање, ажурирање и брисање података. Изоловане су од детаља базе података и пружају апстракцију која омогућава лаку замену или модификацију података без утицаја на остатак апликације.

Укратко, ,,*DAO*" дизајн шаблон омогућава да апликација има јасно раздвојене слојеве који се баве представљањем корисничког интерфејса, бизнис логиком и комуникацијом са базом података. Ово олакшава одржавање, тестирање и скалабилност апликације.

**4.1 Model (*Entity*)**

*Entity* су објекти из базе података. *Entity* класе се пису у *POJO* (Plain Old Java Object).

**Опис:** Овај ентитет представља пацијента у ветеринарској амбуланти.

 **id**: Јединствени идентификатор пацијента.

 **species**: Врста животиње (нпр. пас, мачка).

 **breed**: Раса животиње.

 **ownerName**: Име власника животиње.

 **ownerContact**: Контакт информације власника.

package com.example.demo.entity;  
  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import lombok.Getter;  
import lombok.Setter;  
import lombok.NoArgsConstructor;  
import lombok.AllArgsConstructor;  
  
@Entity  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class Patient {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String species;  
 private String breed;  
 private String ownerName;  
 private String ownerContact;  
}

**Опис:** Овај ентитет представља преглед заказан за пацијента.

 **id**: Јединствени идентификатор прегледа.

 **patientId**: Идентификатор пацијента који је заказан за преглед.

 **veterinarianId**: Идентификатор ветеринара који ће обавити преглед.

 **appointmentTime**: Време заказаног прегледа.

package com.example.demo.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import lombok.Getter;  
import lombok.Setter;  
import lombok.NoArgsConstructor;  
import lombok.AllArgsConstructor;  
import java.time.LocalDateTime;  
  
@Entity  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class Appointment {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private Long patientId;  
 private Long veterinarianId;  
 private LocalDateTime appointmentTime;  
}

**Опис:** Овај ентитет представља третман који је пацијент примио.

 **id**: Јединствени идентификатор третмана.

 **patientId**: Идентификатор пацијента који је примио третман.

 **treatmentDetails**: Детаљи о третману.

 **treatmentDate**: Датум када је третман обављен.

package com.example.demo.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import lombok.Getter;  
import lombok.Setter;  
import lombok.NoArgsConstructor;  
import lombok.AllArgsConstructor;  
import java.time.LocalDateTime;  
  
@Entity  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class Treatment {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private Long patientId;  
 private String treatmentDetails;  
 private LocalDateTime treatmentDate;  
}

**Опис:** Овај ентитет представља ветеринара у амбуланти.

 **id**: Јединствени идентификатор ветеринара.

 **name**: Име ветеринара.

 **specialization**: Специјализација ветеринара (нпр. хирургија, дерматологија).

package com.example.demo.entity;  
  
import jakarta.persistence.Entity;  
 import jakarta.persistence.GeneratedValue;  
 import jakarta.persistence.GenerationType;  
 import jakarta.persistence.Id;  
 import lombok.Getter;  
 import lombok.Setter;  
 import lombok.NoArgsConstructor;  
 import lombok.AllArgsConstructor;  
  
 @Entity  
 @Getter  
 @Setter  
 @NoArgsConstructor  
 @AllArgsConstructor  
 public class Veterinarian {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private String name;  
 private String specialization;  
}

**Опис:** Овај ентитет представља третман који је пацијент примио.

 **id**: Јединствени идентификатор третмана.

 **patientId**: Идентификатор пацијента који је примио третман.

 **treatmentDetails**: Детаљи о третману.

 **treatmentDate**: Датум када је третман обављен.

package com.example.demo.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import lombok.Getter;  
import lombok.Setter;  
import lombok.NoArgsConstructor;  
import lombok.AllArgsConstructor;  
import java.time.LocalDateTime;  
  
@Entity  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class VisitRecord {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private Long patientId;  
 private Long veterinarianId;  
 private LocalDateTime visitTime;  
 private String servicesProvided;  
}

**4.2 Слој података *Repository* (*DAO*) слој**

У овом слоју сам користио *Spring Data JPA* библиотеку за манипулисање са базом података.

Она пружа све *CRUD* функионалности и још по нешто. Оно што она не пружа межете сами да декларишете и они неће се наследити из хијерархије.

Када радимо са *Spring Data JPA* морамо да поставимо анотацију на наш интерфејс *@Repository* да би *Spring* могао да направи зрно (*bean*) за тај објекат.

Опис**:** Овај репозиторијум омогућава CRUD операције над ентитетом Patient.

 JpaRepository<Patient, Long>: Наслеђује основне CRUD операције за ентитет Patient са идентификатором типа Long.

package com.example.demo.repository;

import com.example.demo.entity.Patient;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface PatientRepository extends JpaRepository<Patient, Long> {

}

Опис**:** Овај репозиторијум омогућава CRUD операције над ентитетом Appointment.

 JpaRepository<Appointment, Long>: Наслеђује основне CRUD операције за ентитет Appointment са идентификатором типа Long.

package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.entity.Appointment;

public interface AppointmentRepository extends JpaRepository<Appointment, Long> {

}

Опис**:** Овај репозиторијум омогућава CRUD операције над ентитетом Treatment.

 JpaRepository<Treatment, Long>: Наслеђује основне CRUD операције за ентитет Treatment са идентификатором типа Long.

package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.entity.Treatment;

public interface TreatmentRepository extends JpaRepository<Treatment, Long> {

}

Опис**:** Овај репозиторијум омогућава CRUD операције над ентитетом Veterinarian.

 JpaRepository<Veterinarian, Long>: Наслеђује основне CRUD операције за ентитет Veterinarian са идентификатором типа Long.

package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.entity.Veterinarian;

public interface VeterinarianRepository extends JpaRepository<Veterinarian, Long> {

}

Опис**:** Овај репозиторијум омогућава CRUD операције над ентитетом VisitRecord.

 JpaRepository<VisitRecord, Long>: Наслеђује основне CRUD операције за ентитет VisitRecord са идентификатором типа Long.

package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.demo.entity.VisitRecord;

public interface VisitRecordRepository extends JpaRepository<VisitRecord, Long> {

}

**4.3 Логички слој (*Service layer*)**

Сервисни слој се састоји од апстракције и имплементације.

У апстракцији се пише код који ће се касније имплементирати у конкректној класи, ради имплементиранја лабаве спреге (*loose coupling*).

**Опис:** Овај сервис пружа методе за управљање пацијентима у ветеринарској амбуланти.

 **getAllPatients()**: Враћа листу свих пацијената.

 **getPatientById(Long id)**: Враћа пацијента по његовом идентификатору или null ако пацијент не постоји.

 **savePatient(Patient patient)**: Чува новог или ажурира постојећег пацијента.

 **deletePatient(Long id)**: Брише пацијента по његовом идентификатору.

 **sendReminders()**: Логика за слање подсетника власницима пацијената.

package com.example.demo.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.example.demo.entity.Patient;

import com.example.demo.repository.PatientRepository;

import java.util.List;

@Service

public class PatientService {

@Autowired

private PatientRepository patientRepository;

public List<Patient> getAllPatients() {

return patientRepository.findAll();

}

public Patient getPatientById(Long id) {

return patientRepository.findById(id).orElse(null);

}

public Patient savePatient(Patient patient) {

return patientRepository.save(patient);

}

public void deletePatient(Long id) {

patientRepository.deleteById(id);

}

public void sendReminders() {

// Логика за слање подсетника

System.out.println("Sending reminders to patient owners...");

}

}

**Опис:** Овај сервис пружа методе за управљање прегледима у ветеринарској амбуланти.

 **getAllAppointments()**: Враћа листу свих прегледа.

 **getAppointmentById(Long id)**: Враћа преглед по његовом идентификатору или null ако преглед не постоји.

 **saveAppointment(Appointment appointment)**: Чува нови или ажурира постојећи преглед.

 **deleteAppointment(Long id)**: Брише преглед по његовом идентификатору.

package com.example.demo.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.example.demo.entity.Appointment;

import com.example.demo.repository.AppointmentRepository;

import java.util.List;

@Service

public class AppointmentService {

@Autowired

private AppointmentRepository appointmentRepository;

public List<Appointment> getAllAppointments() {

return appointmentRepository.findAll();

}

public Appointment getAppointmentById(Long id) {

return appointmentRepository.findById(id).orElse(null);

}

public Appointment saveAppointment(Appointment appointment) {

return appointmentRepository.save(appointment);

}

public void deleteAppointment(Long id) {

appointmentRepository.deleteById(id);

}

}

**4.4 Представнички слој (Presentation** ***layer*)**

Опис**:** Овај контролер пружа REST API за управљање пацијентима у ветеринарској амбуланти.

 @GetMapping: Враћа листу свих пацијената.

 @GetMapping("/{id}"): Враћа пацијента по његовом идентификатору.

 @PostMapping: Креира новог пацијента.

 @DeleteMapping("/{id}"): Брише пацијента по његовом идентификатору.

 @PutMapping("/{id}"): Ажурира податке о пацијенту по његовом идентификатору.

 @GetMapping("/test"): Тест ендпоинт који враћа поруку да је контролер у функцији.

package com.example.demo.controller;

import com.example.demo.entity.Patient;

import com.example.demo.repository.PatientRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/patients")

public class PatientController {

private static final Logger logger = LoggerFactory.getLogger(PatientController.class);

@Autowired

private PatientRepository patientRepository;

@GetMapping

public List<Patient> getAllPatients() {

logger.info("Fetching all patients");

return patientRepository.findAll();

}

@GetMapping("/{id}")

public Patient getPatientById(@PathVariable Long id) {

logger.info("Fetching patient with id: {}", id);

return patientRepository.findById(id).orElse(null);

}

@PostMapping

public Patient createPatient(@RequestBody Patient patient) {

logger.info("Creating new patient: {}", patient);

return patientRepository.save(patient);

}

@DeleteMapping("/{id}")

public void deletePatient(@PathVariable Long id) {

logger.info("Deleting patient with id: {}", id);

patientRepository.deleteById(id);

}

@GetMapping("/test")

public String test() {

logger.info("Test endpoint called");

return "PatientController is working!";

}

@PutMapping("/{id}")

public ResponseEntity<Patient> updatePatient(@PathVariable Long id, @RequestBody Patient patientDetails) {

logger.info("Updating patient with id: {}", id);

Optional<Patient> optionalPatient = patientRepository.findById(id);

if (optionalPatient.isPresent()) {

Patient patient = optionalPatient.get();

patient.setSpecies(patientDetails.getSpecies());

patient.setBreed(patientDetails.getBreed());

patient.setOwnerName(patientDetails.getOwnerName());

patient.setOwnerContact(patientDetails.getOwnerContact());

Patient updatedPatient = patientRepository.save(patient);

logger.info("Updated patient: {}", updatedPatient);

return ResponseEntity.ok(updatedPatient);

} else {

logger.warn("Patient with id: {} not found", id);

return ResponseEntity.notFound().build();

}

}

}

Опис: Овај контролер пружа REST API за управљање прегледима у ветеринарској амбуланти.

 @GetMapping: Враћа листу свих прегледа.

 @GetMapping("/{id}"): Враћа преглед по његовом идентификатору.

 @PostMapping: Креира нови преглед.

 @DeleteMapping("/{id}"): Брише преглед по његовом идентификатору.

 @GetMapping("/test"): Тест ендпоинт који враћа поруку да је контролер у функцији.

package com.example.demo.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import com.example.demo.entity.Appointment;

import com.example.demo.service.AppointmentService;

import java.util.List;

@RestController

@RequestMapping("/appointments")

public class AppointmentController {

@Autowired

private AppointmentService appointmentService;

@GetMapping

public List<Appointment> getAllAppointments() {

return appointmentService.getAllAppointments();

}

@GetMapping("/{id}")

public Appointment getAppointmentById(@PathVariable Long id) {

return appointmentService.getAppointmentById(id);

}

@PostMapping

public Appointment createAppointment(@RequestBody Appointment appointment) {

return appointmentService.saveAppointment(appointment);

}

@DeleteMapping("/{id}")

public void deleteAppointment(@PathVariable Long id) {

appointmentService.deleteAppointment(id);

}

@GetMapping("/test")

public String test() {

return "AppointmentController is working!";

}

}

У овом пројекту имамо четири главна пакета који управљају различитим аспектима ветеринарске амбуланте. Сваки пакет има своју улогу и одговорности, а заједно чине комплетан систем за управљање пацијентима, прегледима, третманима и записима о посети. Ево детаљног објашњења сваког пакета:

# 1. Пакет entity (Ентитети)

Пакет entity садржи JPA ентитете који представљају различите објекте у бази података. Ови ентитети су мапирани на табеле у бази података и користе се за чување и преузимање података. У овом пакету имамо следеће класе:

 Patient (Пацијент): Ова класа представља пацијента у амбуланти. Садржи информације као што су врста животиње, раса, име власника и контакт информације власника.

 Appointment (Преглед): Ова класа представља преглед заказан за пацијента. Садржи информације о времену прегледа, идентификатору пацијента и идентификатору ветеринара који ће обавити преглед.

 Treatment (Третман): Ова класа представља третман који је пацијент примио. Садржи детаље о третману, датум третмана и идентификатор пацијента.

 VisitRecord (Запис о Посети): Ова класа представља запис о посети пацијента ветеринару. Садржи информације о времену посете, пруженим услугама, идентификатору пацијента и идентификатору ветеринара.

# 2. Пакет repository (Репозиторијуми)

Пакет repository садржи интерфејсе који проширују JpaRepository и омогућавају CRUD операције над ентитетима. Ови репозиторијуми пружају основне методе за чување, ажурирање, брисање и преузимање података из базе. У овом пакету имамо следеће интерфејсе:

 PatientRepository (Репозиторијум за Пацијенте): Омогућава CRUD операције над ентитетом Patient.

 AppointmentRepository (Репозиторијум за Прегледе): Омогућава CRUD операције над ентитетом Appointment.

 TreatmentRepository (Репозиторијум за Третмане): Омогућава CRUD операције над ентитетом Treatment.

 VisitRecordRepository (Репозиторијум за Записе о Посети): Омогућава CRUD операције над ентитетом VisitRecord.

# 3. Пакет service (Сервиси)

Пакет service садржи сервисне класе које имплементирају пословну логику апликације. Ове класе користе репозиторијуме за приступ подацима и пружају методе које се користе у контролерима. У овом пакету имамо следеће класе:

 PatientService (Сервис за Пацијенте): Пружа методе за управљање пацијентима, као што су преузимање свих пацијената, преузимање пацијента по идентификатору, чување новог или ажурирање постојећег пацијента и брисање пацијента.

 AppointmentService (Сервис за Прегледе): Пружа методе за управљање прегледима, као што су преузимање свих прегледа, преузимање прегледа по идентификатору, чување новог или ажурирање постојећег прегледа и брисање прегледа.

 TreatmentService (Сервис за Третмане): Пружа методе за управљање третманима, као што су преузимање свих третмана, преузимање третмана по идентификатору, чување новог или ажурирање постојећег третмана и брисање третмана.

 VisitRecordService (Сервис за Записе о Посети): Пружа методе за управљање записима о посети, као што су преузимање свих записа о посети, преузимање записа по идентификатору, чување новог или ажурирање постојећег записа и брисање записа.

# 4. Пакет controller (Контролери)

Пакет controller садржи REST контролере који пружају API ендпоинте за клијенте. Ови контролери користе сервисне класе за обраду захтева и враћање одговора. У овом пакету имамо следеће класе:

 PatientController (Контролер за Пацијенте): Пружа ендпоинте за управљање пацијентима, као што су преузимање свих пацијената, преузимање пацијента по идентификатору, креирање новог пацијента, ажурирање постојећег пацијента и брисање пацијента.

 AppointmentController (Контролер за Прегледе): Пружа ендпоинте за управљање прегледима, као што су преузимање свих прегледа, преузимање прегледа по идентификатору, креирање новог прегледа и брисање прегледа.

# Закључак

Ова четири пакета заједно чине комплетан систем за управљање ветеринарском амбулантом. Ентитети представљају податке у бази, репозиторијуми омогућавају приступ тим подацима, сервиси имплементирају пословну логику, а контролери пружају API ендпоинте за клијенте. Ова структура омогућава лако одржавање и проширење апликације, као и јасну поделу одговорности између различитих делова система.

# 5.Logging

## Објашњење Конфигурације

Ова конфигурација користи Logback за управљање логовањем у Spring Boot апликацији. Ево детаљног објашњења сваког дела конфигурације:

## 1. <configuration>

 Опис: Главни елемент који обухвата целу конфигурацију за Logback.

## 2. <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

 Опис: Овај апендер дефинише логовање у конзолу.

 name: Име апендера је "STDOUT".

 class: Класа апендера је ch.qos.logback.core.ConsoleAppender, што значи да ће логови бити исписани у конзолу.

## <encoder>

 Опис: Енкодер дефинише формат лог порука.

 <pattern>: Патерн дефинише формат лог порука. У овом случају, формат је %d{yyyy-MM-dd HH:mm:ss} - %msg%n, што значи:

 %d{yyyy-MM-dd HH:mm:ss}: Датум и време у формату "година-месец-дан сат:минут:секунда".

 %msg: Лог порука.

 %n: Нова линија.

## 3. <appender name="FILE" class="ch.qos.logback.core.FileAppender">

 Опис: Овај апендер дефинише логовање у фајл.

 name: Име апендера је "FILE".

 class: Класа апендера је ch.qos.logback.core.FileAppender, што значи да ће логови бити исписани у фајл.

## <file>

 Опис: Путања до фајла у који ће логови бити записани.

 Вредност: logs/app.log, што значи да ће логови бити записани у фајл app.log у директоријуму logs.

## <append>

 Опис: Дефинише да ли ће нови логови бити додати на крај постојећег фајла или ће фајл бити преписан.

 Вредност: true, што значи да ће нови логови бити додати на крај постојећег фајла.

## <encoder>

 Опис: Енкодер дефинише формат лог порука.

 <pattern>: Патерн дефинише формат лог порука. У овом случају, формат је исти као и за конзолни апендер: %d{yyyy-MM-dd HH:mm:ss} - %msg%n.

## 4. <root level="info">

 Опис: Дефинише ниво логовања за коренски логер.

 level: Ниво логовања је "info", што значи да ће све поруке нивоа info и вишег (нпр. warn, error) бити логоване.

## <appender-ref ref="STDOUT" />

 Опис: Додаје референцу на апендер "STDOUT" у коренски логер, што значи да ће поруке бити исписане у конзолу.

## <appender-ref ref="FILE" />

 Опис: Додаје референцу на апендер "FILE" у коренски логер, што значи да ће поруке бити записане у фајл.

## Закључак

Ова конфигурација омогућава логовање у конзолу и у фајл истовремено. Формат лог порука је дефинисан тако да укључује датум, време и саму поруку, што олакшава праћење и анализу логова. Ниво логовања је подешен на info, што значи да ће све поруке нивоа info и вишег бити логоване.

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss} - %msg%n</pattern>

</encoder>

</appender>

<appender name="FILE" class="ch.qos.logback.core.FileAppender">

<file>logs/app.log</file>

<append>true</append>

<encoder>

<pattern>%d{yyyy-MM-dd HH:mm:ss} - %msg%n</pattern>

</encoder>

</appender>

<root level="info">

<appender-ref ref="STDOUT" />

<appender-ref ref="FILE" />

</root>

</configuration>

# 6. Класа ScheduledTasks

Ова класа је Spring компонента која користи аннотацију @Scheduled за извршавање задатака у одређеним интервалима.

Основни Елементи:

 Аннотација @Component: Означава да је класа Spring компонента.

2. Логер: Користи се за логовање порука.

3. Метода checkPatientStatus:

 Аннотација @Scheduled: Заказује извршавање методе.

 initialDelay = 10000: Прво извршавање након 10 секунди.

 fixedRate = Long.MAX\_VALUE: Интервал између извршавања је веома дуг.

 Логовање: Логовање поруке о провери статуса пацијената и слању подсетника власницима.

Закључак

Класа ScheduledTasks заказује и извршава методе у одређеним интервалима, користећи логер за бележење активности.

package com.example.demo;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class ScheduledTasks {

private static final Logger logger = LoggerFactory.getLogger(ScheduledTasks.class);

@Scheduled(initialDelay = 10000, fixedRate = Long.MAX\_VALUE)

public void checkPatientStatus() {

logger.info("Provera statusa pacijenata i slanje podsetnika vlasnicima.");

}

}

# Литература

 **"Spring in Action" by Craig Walls**

 **"Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications" by Mark Heckler**