Академија струковних
студија Шумадија
Одсек Крагујевац

Студијски програм: Информатика
Предмет: Програмски језици

Програмски језици

- Туристичка агенција –

Предметни наставник:
др Александар Мишковић

Студент:
Дамјан Илић 003/2022

Крагујевац 2024.

Дамјан Илић, 003/2022

Поставка задатка

Циљ пројекта: Развити Јава *RESTful веб* сервис за управљање продајом и сервисом мобилних телефона. Сервис ће омогућити управљање залихама телефона, продајом, сервисним захтевима и евиденцијом клијената.

Функционалности:

а) Преглед путовања: Омогућити клијентима да прегледају доступне туристичке пакете,

укључујући дестинације, датуме путовања, цене и детаљне описе.

б) Резервација путовања: Могућност онлине резервације путовања, са опцијама филтрирања према дестинацији, датуму и цени.

ц) Валидација доступности: Вођење евиденције о клијентима и њиховим претходним путовањима и резервацијама.

д) Управљање клијентима: Слање аутоматских обавештења о променама у вези са испитима (промена датума, отказивање испита).

е) Покретање додатне нити: Имплементација додатне нити која редовно ажурира понуде путовања на основу промена доступности и специјалних понуда.

Технички Захтеви:

а) Бацкенд: Развој бацкенд апликације са Спринг Боот 3.0 и ЈДК 17.

б) База података: Коришћење МуСQЛ-а за складиштење података.

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

д) Мулти-лауер: Имплементација вишеслојне архитектуре.

е) JCOH: Комуникација између клијента и сервера користећи JCOH формат.

ф) Тестирање: Тестирање *API*-ја помоћу *Postman*-а.

Структура Података:

а) Туристички Пакет: ИД пакета, дестинација, опис, датум путовања, цена, статус доступности.

б) Резервација: ИД резервације, ИД клијента, ИД туристичког пакета, датум резервације, статус резервације.

ц) Клијент: Име, презиме, контакт информације, историја резервација.

д) Специјална Понуда: Информације о тренутним специјалним понудама или попустима (накнадно се након покретања програма ручно иницира нека специјална понуда).

е) Остале класе и интерфејси: Пројекат садржи више класа разврстаних духу вишеслојне архитектуре

(Цонтроллер, Цомпонент, Сервице, Репоситору, ДАО, Логгер, … у зависности од реализације

вашег пројекта).

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 веб* сервиса, ефикасно управљање базама података, примену вишенитног програмирања и аспектног програмирања у контексту *Spring Boot* апликација.

Садржај

Поставка задатка....1
Садржај....3
Увод....4
1. Конфигурација....5
1.1. Конфигурација *Maven* фајла....5
1.2. Конфигурисање *Property* фајла....5
2. База података....9
3. Апликација....10
3.1. Model (*Entity*)....10
3.2. Слој података *Repository* (*DAO*) слој....17
3.3. Логички слој (*Service layer*)....18
3.4. Представнички слој (Presentation Layer)....27
3.5. Аспекти....36
Закључак....47
Литература....48

Увод

У овом пројектном задатку сам имао да одрадим апликацију засновану на *RESTful* сервису за потребе једне зубарске ординације.

1. Конфигурација

1.1. 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:

```
2. <?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.asss.pj</groupId>
<artifactId>TuristickaAgencija</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>TuristickaAgencija</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.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

```
</project>
```

2.1. Конфигурисање *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:

```
spring.datasource.url=jdbc:mysql://localhost:3306/turisticka_agencija
spring.datasource.username=root
spring.datasource.password=
```

Прве три линије кода служе за конекцију са базом

1. Прва линија кода служи (**.*.url*) да се лоцира база података (*jdbc:mysql://localhost:3306/turisticka_agencija*).

jdbc – представља скраћеницу "*Java Database Connectivity*", што представља стандардни *API* (*Application Programming Interface*) који омогућава *Java* апликацијама да комуницирају са различитим типовима база података.

mysql - Овај део *URL*-а означава тип базе података са којом се успоставља веза. У овом случају, користи се *MySQL* база података.

localhost – Овде се спецификује име или *IP* адреса рачунара на којем се налази MySQL база података. "*localhost*" означава да се база података налази на истом рачунару где се извршава апликација.

3306 – ово је порт на коме MySQL сервер слуша конекције.

prodaja_i_servis_mobilnih_telefona - Овде се наводи име конкретне базе података са којом желите да успоставите везу.

2. Друга линија кода служи (**.*.username*) за име корисника који ради са базом података.

3. Трећа линија кода служи (**.*.password*) за шифру корисника који ради са базом података. У овом случају је празна, зато што нисам поставио шифру.

4. spring.main.banner-mode=off – служи да се уклони *Spring Boot* лого из конзоле.

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 моје табеле из базе података

Слика 1. Табела из базе података

Нисам урадио релације преко *mySql*-а јер ***Jakarta Persistence*** сам изврши релацију.

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).

Rezervacija класа:

```
package com.asss.pj.TuristickaAgencija.entity;

import jakarta.persistence.*;

import java.util.Date;

@Entity
@Table(name = "rezervacija")
public class Rezervacija {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
```

```
    @OneToOne(cascade = {
            CascadeType.DETACH, CascadeType.MERGE,
            CascadeType.PERSIST, CascadeType.REFRESH})
    @JoinColumn(name = "klijent_id")
    private Klijent klijent;

    @OneToOne(cascade = {
            CascadeType.DETACH, CascadeType.MERGE,
            CascadeType.PERSIST, CascadeType.REFRESH})
    @JoinColumn(name = "turisticki_paket_id")
    private TuristickiPaket turistickiPaket;

    @Column(name = "datum_rezervacije")
    private Date datumRezervacije;

    @Column(name = "status_rezervacije")
    private boolean statusRezervacije;

    public Rezervacija() {

    }

    public Rezervacija(Klijent klijent, TuristickiPaket turistickiPaket, Date
datumRezervacije, boolean statusRezervacije) {
        this.klijent = klijent;
        this.turistickiPaket = turistickiPaket;
        this.datumRezervacije = datumRezervacije;
        this.statusRezervacije = statusRezervacije;
    }


    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public Klijent getKlijent() {
        return klijent;
    }

    public void setKlijent(Klijent klijent) {
        this.klijent = klijent;
    }

    public TuristickiPaket getTuristickiPaket() {
        return turistickiPaket;
    }

    public void setTuristickiPaket(TuristickiPaket turistickiPaket) {
        this.turistickiPaket = turistickiPaket;
    }

    public Date getDatumRezervacije() {
        return datumRezervacije;
    }

    public void setDatumRezervacije(Date datumRezervacije) {
```

```
        this.datumRezervacije = datumRezervacije;
    }

    public boolean isStatusRezervacije() {
        return statusRezervacije;
    }

    public void setStatusRezervacije(boolean statusRezervacije) {
        this.statusRezervacije = statusRezervacije;
    }

    @Override
    public String toString() {
        return "Rezervacija{" +
                "klijent=" + klijent +
                ", turistickiPaket=" + turistickiPaket +
                ", datumRezervacije=" + datumRezervacije +
                ", statusRezervacije=" + statusRezervacije +
                '}';
    }
}
```

Klijent класа:

```
package com.asss.pj.TuristickaAgencija.entity;


import jakarta.persistence.*;

@Entity
@Table(name = "klijent")
public class Klijent {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "ime")
    private String ime;

    @Column(name = "prezime")
    private String prezime;

    @Column(name = "kontakt_informacije")
    private String kontaktInformacije;


    @Column(name = "istorija_rezervacije")
    private String istorijaRezervacije;

    public Klijent() {
    }

    public Klijent(String ime, String prezime, String kontaktInformacije, String
istorijaRezervacije) {
        this.ime = ime;
        this.prezime = prezime;
        this.kontaktInformacije = kontaktInformacije;
        this.istorijaRezervacije = istorijaRezervacije;
    }
```

```
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getIme() {
        return ime;
    }

    public void setIme(String ime) {
        this.ime = ime;
    }

    public String getPrezime() {
        return prezime;
    }

    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }

    public String getKontaktInformacije() {
        return kontaktInformacije;
    }

    public void setKontaktInformacije(String kontaktInformacije) {
        this.kontaktInformacije = kontaktInformacije;
    }

    public String getIstorijaRezervacije() {
        return istorijaRezervacije;
    }

    public void setIstorijaRezervacije(String istorijaRezervacije) {
        this.istorijaRezervacije = istorijaRezervacije;
    }


    @Override
    public String toString() {
        return "Klijent{" +
                "id=" + id +
                ", ime='" + ime + '\'' +
                ", prezime='" + prezime + '\'' +
                ", kontaktInformacije='" + kontaktInformacije + '\'' +
                ", istorijaRezervacije='" + istorijaRezervacije + '\'' +
                '}';
    }
}
```

TuristickiPaket класа:

```
package com.asss.pj.TuristickaAgencija.entity;


import jakarta.persistence.*;

import java.util.Date;
```

```java
@Entity
@Table(name = "turisticki_paket")
public class TuristickiPaket {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "destinacija")
    private String destinacija;

    @Column(name = "opis")
    private String opis;

    @Column(name = "datum_putovanja")
    private Date datumPutovanja;

    @Column(name = "cena")
    private double cena;

    @Column(name = "status_dostupnosti")
    private boolean statusDostupnosti;

    public TuristickiPaket() {
    }

    public TuristickiPaket(String destinacija, String opis, Date datumPutovanja,
double cena, boolean statusDostupnosti) {
        this.destinacija = destinacija;
        this.opis = opis;
        this.datumPutovanja = datumPutovanja;
        this.cena = cena;
        this.statusDostupnosti = statusDostupnosti;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getDestinacija() {
        return destinacija;
    }

    public void setDestinacija(String destinacija) {
        this.destinacija = destinacija;
    }

    public String getOpis() {
        return opis;
    }

    public void setOpis(String opis) {
        this.opis = opis;
    }

    public Date getDatumPutovanja() {
        return datumPutovanja;
```

```
    }

    public void setDatumPutovanja(Date datumPutovanja) {
        this.datumPutovanja = datumPutovanja;
    }

    public double getCena() {
        return cena;
    }

    public void setCena(double cena) {
        this.cena = cena;
    }

    public boolean isStatusDostupnosti() {
        return statusDostupnosti;
    }

    public void setStatusDostupnosti(boolean statusDostupnosti) {
        this.statusDostupnosti = statusDostupnosti;
    }

    @Override
    public String toString() {
        return "TuristickiPaket{" +
                "id=" + id +
                ", destinacija='" + destinacija + '\'' +
                ", opis='" + opis + '\'' +
                ", datumPutovanja=" + datumPutovanja +
                ", cena=" + cena +
                ", statusDostupnosti=" + statusDostupnosti +
                '}';
    }
}
```

SpecijalnaPonuda класа:`

```
package com.asss.pj.TuristickaAgencija.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "specijalna_ponuda")
public class SpecijalnaPonuda {


    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;


    @OneToOne()
    @JoinColumn(name = "turisticki_paket_id") // Foreign key column to establish
the relationship
    private TuristickiPaket turistickiPaket;


    // Integer - popust, TuristickiPaket - objekat
    @Column(name = "opis")
    private String opis;

    @Column(name = "popust")
```

```
    private int popust;

    public SpecijalnaPonuda() {
    }

    public SpecijalnaPonuda(TuristickiPaket turistickiPaket, String opis, int
popust) {
        this.turistickiPaket = turistickiPaket;
        this.opis = opis;
        this.popust = popust;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public TuristickiPaket getTuristickiPaket() {
        return turistickiPaket;
    }

    public void setTuristickiPaket(TuristickiPaket turistickiPaket) {
        this.turistickiPaket = turistickiPaket;
    }

    public String getOpis() {
        return opis;
    }

    public void setOpis(String opis) {
        this.opis = opis;
    }

    public double getPopust() {
        return popust;
    }

    public void setPopust(int popust) {
        this.popust = popust;
    }

    @Override
    public String toString() {
        return "SpecijalnaPonuda{" +
                "id=" + id +
                ", turistickiPaket=" + turistickiPaket +
                ", opis='" + opis + '\'' +
                ", popust=" + popust +
                '}';
    }
}
```

1. Прво сам додао атрибуте у класу:

Анотација *@Entity* означава да је класа *Entity*.

Анотација *@Table* представља да је класа табела са датим именом.

Анотација *@Id* представља да је атрибут примарни кључ.

Анотација *@GeneratedValue* представља на који начин ће се постављати вредност примарног кључа. У мом случају ће расти инкрементално. Сваки пут кад се уноси објекат у базу повећава се вредност за један. Прва вредност примарног кључа је 1, али се може наместити да буде и дугачија.

Анотација *@Column* представља да је атрибут колона у табели са датим именом.

Анотација *@ManyToOne, @ManyToMany* представља релацију између две табеле.

Анотација *@JoinTable* у овом случају користи агрегативну табелу да би досло до релације више у више.

2. Онда сам додао конструкторе. У конструктор нисам хтео да додам “id” јер не желим кад се инстанцира објекат да се прикаже примарни кључ табеле.

3. Getter-e и *seter-e-*

4. И на крају toString методу.

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 можете сами да декларишете и они неће се наследити из хијерархије.

Када радимо са *Spring Data JPA* можемо да поставимо анотацију на наш интерфејс *@Repository* да би *Spring* могао да направи зрно (*bean*) за тај објекат. Наш интерфејс мора да проширу класу *JpaRepository<Var1, Var2>*.

Var1 – је модел (*Entity*)

Var2 – је тип променљиве примарног кључа (*id*)

Само један репозиторијум (интерфејс “ Rezervacija*Repo*”) има додатну методу, остали интерфејси немају додатне методе (интерфејси су празни).

```
package com.asss.pj.TuristickaAgencija.repo;

import com.asss.pj.TuristickaAgencija.entity.Rezervacija;
import org.springframework.data.domain.Sort;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface RezervacijaRepo extends JpaRepository<Rezervacija, Integer> {

    List<Rezervacija> findAllSortedBy(Sort sort);
```

```
}
```

Метода "*findAllSortedBy (Sort sort);*.служи да сортира објекте по датом критеријуму

Spring Data JPA сам генерише упит у овом случају. То чини тако што прочита идентификатор од методе и њене аргументе. Па тако направи упит.

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 конкретној класи, ради имплементирања лабаве спреге (*loose coupling*).

Овако изгеда апстракција сервис класе (унутрашњост сервис интерфејса "Rezervacija*Service*"):

```
package com.asss.pj.TuristickaAgencija.service;

import com.asss.pj.TuristickaAgencija.entity.Rezervacija;

import java.util.Date;
import java.util.List;
import java.util.Optional;

public interface RezervacijaService {
    List<Rezervacija> findAll();

    Optional<Rezervacija> findById(int theId);

    Rezervacija save(Rezervacija rezervacija);

    void deleteById(int theId);

    Rezervacija dodajTuristickiPaket(int rezervacijaId, int turistickiPaketId);

    Rezervacija dodajKlijenta(int rezervacijaId, int klijentId);

    // kod za filtriranje
    List<Rezervacija> findAllSortedBy(String sortBy);

}

       }
```

У њој се налазе мотоде које ће касније конкретна класа (имплементација) да имплементира.

Имплементација интерфејса "*RezervacijaService*" изгледа овако:

```
package com.asss.pj.TuristickaAgencija.service;

import com.asss.pj.TuristickaAgencija.entity.Klijent;
import com.asss.pj.TuristickaAgencija.entity.Rezervacija;
import com.asss.pj.TuristickaAgencija.entity.TuristickiPaket;
import com.asss.pj.TuristickaAgencija.repo.KlijentRepo;
import com.asss.pj.TuristickaAgencija.repo.RezervacijaRepo;
```

```
import com.asss.pj.TuristickaAgencija.repo.TuristickiPaketRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Optional;

@Service
public class RezervacijaServiceImpl implements RezervacijaService {


    // dependency injection
    private final RezervacijaRepo rezervacijaRepo;
    private final TuristickiPaketRepo turistickiPaketRepo;
    private final KlijentRepo klijentRepo;

    @Autowired // umetanje zavisnosti dao sloja
    public RezervacijaServiceImpl(RezervacijaRepo rezervacijaRepo,
TuristickiPaketRepo turistickiPaketRepo, KlijentRepo klijentRepo) {
        this.rezervacijaRepo = rezervacijaRepo;
        this.turistickiPaketRepo = turistickiPaketRepo;
        this.klijentRepo = klijentRepo;
    }

    @Override
    public List<Rezervacija> findAll() {
        return rezervacijaRepo.findAll();
    }

    @Override
    public Optional<Rezervacija> findById(int theId) {
        // 'Optional' koristi se za predstavljanje vrednosti koja moze ili ne
mora biti prisutna

        Optional<Rezervacija> result = rezervacijaRepo.findById(theId);
        if (result.isPresent()) {
            return Optional.of(result.get());
        }
        throw new NullPointerException("Rezervacija not found with id: " +
theId);
    }

    @Override
    public Rezervacija save(Rezervacija rezervacija) {
        return rezervacijaRepo.save(rezervacija);
    }


    @Override
    public void deleteById(int theId) {
        rezervacijaRepo.deleteById(theId);
    }

    @Override
    public Rezervacija dodajTuristickiPaket(int rezervacijaId, int
turistickiPaketId) {
        // Čitanje objekata preko ID-a
        Optional<Rezervacija> rezervacijaOptional =
```

```
rezervacijaRepo.findById(rezervacijaId);
        Optional<TuristickiPaket> turistickiPaketOptional =
turistickiPaketRepo.findById(turistickiPaketId);

        // Provera da li postoje traženi objekti
        if (rezervacijaOptional.isEmpty()) {
            throw new NullPointerException("Rezervacija not found with id: " +
rezervacijaId);
        }

        if (turistickiPaketOptional.isEmpty()) {
            throw new NullPointerException("Turisticki paket not found with id:
" + turistickiPaketId);
        }


        // Pretvaranje objekata u konkretne promenljive
        Rezervacija rezervacija = rezervacijaOptional.get();
        TuristickiPaket turistickiPaket = turistickiPaketOptional.get();

        // Dodavanje turistickog paketa u rezervaciju

        rezervacija.setTuristickiPaket(turistickiPaket);

        // Postavljanje datuma rezervacije
        rezervacija.setDatumRezervacije(new Date());

        // Postavljanje statusa rezervacije na true
        rezervacija.setStatusRezervacije(true);

        // Sačuvaj promene u oba repozitorijuma
        turistickiPaketRepo.save(turistickiPaket); // Sačuvaj ažurirani
turistički paket
        return rezervacijaRepo.save(rezervacija); // Sačuvaj ažuriranu
rezervaciju
    }

    @Override
    public Rezervacija dodajKlijenta(int rezervacijaId, int klijentId) {

        Optional<Rezervacija> rezervacijaOptional =
rezervacijaRepo.findById(rezervacijaId);
        Optional<Klijent> klijentOptional = klijentRepo.findById(klijentId);
        if (rezervacijaOptional.isEmpty()) {
            throw new NullPointerException("Rezervacija not found with id: " +
rezervacijaId);
        }

        if (klijentOptional.isEmpty()) {
            throw new NullPointerException("Klijent not found with id: " +
klijentId);
        }

        Rezervacija rezervacija = rezervacijaOptional.get();
        Klijent klijent = klijentOptional.get();
        rezervacija.setKlijent(klijent);

        String istorijaRezervacije = klijent.getIstorijaRezervacije();


        // kod za formatiranje istorije rezervacije ZA KLIJENTA
```

```
        // formatira se samo u bazi, formatiranje ne radi za json objekte
        if (istorijaRezervacije != null) {
            List<String> istorijaRezervacijeList = new ArrayList<>();

            String sb = istorijaRezervacije + ";\n- " + rezervacija;

            klijent.setIstorijaRezervacije(sb);
        }

        else {
            klijent.setIstorijaRezervacije("- " + istorijaRezervacije);
        }
        return rezervacijaRepo.save(rezervacija);
    }

    // kod za sortiranje
    @Override
    public List<Rezervacija> findAllSortedBy(String sortBy) {
        Sort sort = getSort(sortBy);
        return rezervacijaRepo.findAllSortedBy(sort);
    }

    private Sort getSort(String sortBy) {
        String sort = sortBy.toLowerCase();// postavlja se na mala slova
        switch (sort) {
            case "cena-asc":
                return Sort.by(Sort.Order.asc("turistickiPaket.cena"));
            case "cena-desc":
                return Sort.by(Sort.Order.desc("turistickiPaket.cena"));
            case "destinacija-asc":
                return Sort.by(Sort.Order.asc("turistickiPaket.destinacija"));
            case "destinacija-desc":
                return Sort.by(Sort.Order.desc("turistickiPaket.destinacija"));
            case "datum-asc":
                return Sort.by(Sort.Order.asc("datumRezervacije"));
            case "datum-desc":
                return Sort.by(Sort.Order.desc("datumRezervacije"));
            default:
                throw new IllegalArgumentException("Invalid sort by option: " +
sortBy + "\n" +
                        "Valid options are:\n- cena-asc,\n- cena-desc,\n-
destinacija-asc," +
                        "\n- destinacija-desc,\n- datum-asc,\n- datum-desc");
        }
    }

}
```

Када правимо сервисни слој морамо да ставимо анотацију *@Service* да би *Spring* знао да се ради о сервисном слоју. Анотација се ставља на имплементацију да би се могло направити зрно (*Bean*) тог објекта.

Dependency injection шаблон за пројектовање (*design pattern)* који омогућава да се зависности (у овом случају, *DAO* слој) инјектују у класу (у сервисни слој) уместо да се те зависности креирају унутар класе.

У мом коду, Dependency Injection је имплементиран коришћењем анотације @Autowired над конструктором класе “*RezervacijaServiceImpl*”. Овиме се омогућава Spring-у да аутоматски инјектује потребне зависности приликом прављења објекта

"*RezervacijaServiceImpl*". Конкретно, инјектује се ,, *RezervacijaRepo*" ,,*KlijentRepo"* ,, *TuristickiPaketRepo* зависности која се користи за приступ подацима.

Метода "*findById()*" враћа објекат типа "*Rezervacija()*" по датом примарном кључу. Уколико није пронађен објекат најављује се изузетак "*NullPointerException*'' са датом поруком. U ovoj metodi sam koristio класу "Optional". Ради лакшег руковања уколико дође до вредности која не постоји у бази података.

Метода "*save()"* чува вредност у базу података коју је добио као параметар.

Метода "*deleteById()*" брише вредност из базе података коју је добио као парам

Метода "*dodajTuristickiPaket (int rezervacijaId, int turistickiPaketId)*'' чува turistickiPaket са датим примарним кључем у продају са датим примарним кључем.

Метода "*dodajKlijenta (int rezervacijaId, int klijentId)*'' чува turistickiPaket са датим примарним кључем у продају са датим примарним кључем.

Метода ,, findAllSortedBy" сортира објекте по датом критеријуму

Остале импементације сервисних интерфејса су истих или сличних функционалсности.

Клијент:

Апстракција:

```
package com.asss.pj.TuristickaAgencija.service;

import com.asss.pj.TuristickaAgencija.entity.Klijent;

import java.util.List;
import java.util.Optional;

public interface KlijentService {
    List<Klijent> findAll();

    Optional<Klijent> findById(int theId);

    Klijent save(Klijent klijent);

    void deleteById(int theId);
}
```

Имплементација:

```
package com.asss.pj.TuristickaAgencija.service;

import com.asss.pj.TuristickaAgencija.entity.Klijent;
import com.asss.pj.TuristickaAgencija.repo.KlijentRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import java.util.List;
import java.util.Optional;

@Service
public class KlijentServiceImpl implements KlijentService {


    // dependency injection
    private KlijentRepo klijentRepo;

    @Autowired // umetanje zavisnosti dao sloja
    public KlijentServiceImpl(KlijentRepo klijentRepo) {
        this.klijentRepo = klijentRepo;
    }

    @Override
    public List<Klijent> findAll() {
        return klijentRepo.findAll();
    }

    @Override
    public Optional<Klijent> findById(int theId) {
        // 'Optional' koristi se za predstavljanje vrednosti koja moze ili ne
mora biti prisutna

        Optional<Klijent> result = klijentRepo.findById(theId);
        if (result.isPresent()) {
            return Optional.of(result.get());
        }
        throw new NullPointerException("Klijent not found with id: " + theId);
    }

    @Override
    public Klijent save(Klijent klijent) {
        return klijentRepo.save(klijent);
    }


    @Override
    public void deleteById(int theId) {
        klijentRepo.deleteById(theId);
    }
}
```

СпецијалнаПонуда:

Апстракција:

```
package com.asss.pj.TuristickaAgencija.service;

import com.asss.pj.TuristickaAgencija.entity.Klijent;
import com.asss.pj.TuristickaAgencija.entity.SpecijalnaPonuda;

import java.util.List;
import java.util.Optional;

public interface SpecijalnaPonudaService {
    List<SpecijalnaPonuda> findAll();

    Optional<SpecijalnaPonuda> findById(int theId);

    SpecijalnaPonuda save(SpecijalnaPonuda specijalnaPonuda);
```

```
    void deleteById(int theId);

    SpecijalnaPonuda addTuristickiPaket(int specijalnaPonudaId, int
turistickiPaketId);

    void initializeSpecijalnaPonuda();

    void threadForInitialization();
}

    KlijentDto findByIdDto(int theId);
}
```

Имплементација:

```
package com.asss.pj.TuristickaAgencija.service;

import com.asss.pj.TuristickaAgencija.entity.SpecijalnaPonuda;
import com.asss.pj.TuristickaAgencija.entity.TuristickiPaket;
import com.asss.pj.TuristickaAgencija.repo.SpecijalnaPonudaRepo;
import com.asss.pj.TuristickaAgencija.repo.TuristickiPaketRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class SpecijalnaPonudaServiceImpl implements SpecijalnaPonudaService {

    private final SpecijalnaPonudaRepo specijalnaPonudaRepo;
    private final TuristickiPaketRepo turistickiPaketRepo;

    // dependency injection
    @Autowired // umetanje zavisnosti dao sloja
    public SpecijalnaPonudaServiceImpl(SpecijalnaPonudaRepo
specijalnaPonudaRepo, TuristickiPaketRepo turistickiPaketRepo) {
        this.specijalnaPonudaRepo = specijalnaPonudaRepo;
        this.turistickiPaketRepo = turistickiPaketRepo;
    }

    @Override
    public List<SpecijalnaPonuda> findAll() {
        return specijalnaPonudaRepo.findAll();
    }

    @Override
    public Optional<SpecijalnaPonuda> findById(int theId) {
        // 'Optional' koristi se za predstavljanje vrednosti koja moze ili ne
mora biti prisutna

        Optional<SpecijalnaPonuda> result =
specijalnaPonudaRepo.findById(theId);
        if (result.isPresent()) {
            return Optional.of(result.get());
        }
        throw new NullPointerException("Specijalna ponuda not found with id: " +
theId);
    }
```

```
    @Override
    public SpecijalnaPonuda save(SpecijalnaPonuda specijalnaPonuda) {
        return specijalnaPonudaRepo.save(specijalnaPonuda);
    }


    @Override
    public void deleteById(int theId) {
        specijalnaPonudaRepo.deleteById(theId);
    }


    @Override
    public SpecijalnaPonuda addTuristickiPaket(int specijalnaPonudaId, int
turistickiPaketId) {

        Optional<SpecijalnaPonuda> specijalnaPonudaOptional =
specijalnaPonudaRepo.findById(specijalnaPonudaId);
        Optional<TuristickiPaket> turistickiPaketOptional =
turistickiPaketRepo.findById(turistickiPaketId);
        if (specijalnaPonudaOptional.isEmpty()) {
            throw new NullPointerException("Specijalna ponuda not found with id:
" + specijalnaPonudaId);
        }

        if (turistickiPaketOptional.isEmpty()) {
            throw new NullPointerException("Turisticki paket not found with id:
" + turistickiPaketId);
        }

        SpecijalnaPonuda specijalnaPonuda = specijalnaPonudaOptional.get();
        TuristickiPaket turistickiPaket = turistickiPaketOptional.get();

        specijalnaPonuda.setTuristickiPaket(turistickiPaket);

        return specijalnaPonudaRepo.save(specijalnaPonuda);
    }

    @Override
    public void initializeSpecijalnaPonuda() {
        // citanje svih paketa
        List<TuristickiPaket> turistickiPakets = turistickiPaketRepo.findAll();

        // "uzimanje" nasumicnog paketa
        TuristickiPaket turistickiPaket = turistickiPakets.get((int)
(Math.random() * turistickiPakets.size()));

        SpecijalnaPonuda specijalnaPonuda = new SpecijalnaPonuda();
        specijalnaPonuda.setTuristickiPaket(turistickiPaket);

        // postavljanje nasumicnog popusta
        int discount = (int) (Math.random() * 100);
        specijalnaPonuda.setOpis("Specijalna ponuda sa popustom od " + discount
+ "%");
        specijalnaPonuda.setPopust(discount);


        specijalnaPonudaRepo.save(specijalnaPonuda);
    }

    private double discountCalculation(double price, int discount) {
```

```
        return price * (100 - discount) / 100;
    }

    @Override
    @Async
    public void threadForInitialization() {
        // Zakazuje se tajmer za 2 sekundi
        Thread thread = new Thread(() -> {// inicijalizuje se objekat thread
            try {
                Thread.sleep(1000);
                initializeSpecijalnaPonuda(); // Zove se metoda
initializeSpecijalnaPonuda posle 2 sekundi
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });
        thread.start(); // pokrece se objekat thread
    }


}
```

ТуристицкиПакет:

Апстракција:

```
package com.asss.pj.TuristickaAgencija.service;

import com.asss.pj.TuristickaAgencija.entity.TuristickiPaket;

import java.util.List;
import java.util.Optional;

public interface TuristickiPaketService {
    List<TuristickiPaket> findAll();

    Optional<TuristickiPaket> findById(int theId);

    TuristickiPaket save(TuristickiPaket turistickiPaket);

    void deleteById(int theId);

}
```

Имплементација:

```
package com.asss.pj.TuristickaAgencija.service;

import com.asss.pj.TuristickaAgencija.entity.TuristickiPaket;
import com.asss.pj.TuristickaAgencija.repo.TuristickiPaketRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class TuristickiPaketServiceImpl implements TuristickiPaketService {
```

```
    // dependency injection
    private final TuristickiPaketRepo turistickiPaketRepo;

    @Autowired // umetanje zavisnosti dao sloja
    public TuristickiPaketServiceImpl(TuristickiPaketRepo turistickiPaketRepo) {
        this.turistickiPaketRepo = turistickiPaketRepo;
    }

    @Override
    public List<TuristickiPaket> findAll() {
        return turistickiPaketRepo.findAll();
    }

    @Override
    public Optional<TuristickiPaket> findById(int theId) {
        // 'Optional' koristi se za predstavljanje vrednosti koja moze ili ne
mora biti prisutna

        Optional<TuristickiPaket> result = turistickiPaketRepo.findById(theId);
        if (result.isPresent()) {
            return Optional.of(result.get());
        }
        throw new NullPointerException("Turisticki paket not found with id: " +
theId);
    }

    @Override
    public TuristickiPaket save(TuristickiPaket turistickiPaket) {
        return turistickiPaketRepo.save(turistickiPaket);
    }


    @Override
    public void deleteById(int theId) {
        turistickiPaketRepo.deleteById(theId);
    }

}
```

4.4. Представнички слој (Presentation Layer)

У овом слоју сам писао *Rest Controller*.

У њему сам руковао *HTTP* захтеве.

Овако изгледа класа “*RezervacijaController*”:

```
package com.asss.pj.TuristickaAgencija.contoller;

import com.asss.pj.TuristickaAgencija.entity.Rezervacija;
import com.asss.pj.TuristickaAgencija.service.RezervacijaService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController()
```

```
@RequestMapping("/api")
public class RezervacijaController {

    private final RezervacijaService rezervacijaService;

    @Autowired
    public RezervacijaController(RezervacijaService rezervacijaService) {
        this.rezervacijaService = rezervacijaService;
    }

    @GetMapping("/rezervacije")
    public List<Rezervacija> getRezervacije() {
        return rezervacijaService.findAll();

    }

    @GetMapping("/rezervacija/{rezervacijaId}")
    public Optional<Rezervacija> getRezervacija(@PathVariable int rezervacijaId)
{

        Optional<Rezervacija> rezervacija =
rezervacijaService.findById(rezervacijaId);

        if (rezervacija == null) {
            throw new NullPointerException("Rezervacija not found with id: " +
rezervacijaId);
        }

        return rezervacija;
    }

    @PostMapping("/rezervacija")
    public Rezervacija addRezervacija(@RequestBody Rezervacija rezervacija) {

        // na ovaj nacin izbegavamo da korisnik doda id
        rezervacija.setId(0); // kasnije u bazi ce se postaviti konkretan id
        // u suprotnom bi promenio red sa datim id-em

        Rezervacija dbRezervacija = rezervacijaService.save(rezervacija);

        return dbRezervacija;
    }

    @PutMapping("/rezervacija")
    // koristi se za update ili za insert ukoliko ne postoji objekat u bazi
    public Rezervacija updateRezervacija(@RequestBody Rezervacija rezervacija) {
        // U principu je isto kao i addPacijent metoda
        // Ovim se samo prati pravilan http zahtev (PUT)

        Rezervacija dbRezervacija = rezervacijaService.save(rezervacija);

        return dbRezervacija;
    }


    @DeleteMapping("/rezervacija/{rezervacijaId}")
    public String deleteRezervacija(@PathVariable int rezervacijaId) {

        Optional<Rezervacija> tempRezervacija = null;

        tempRezervacija = rezervacijaService.findById(rezervacijaId);
```

```
        if (tempRezervacija == null) {
            return "Rezervacija id not found: " + rezervacijaId;
        }

        rezervacijaService.deleteById(rezervacijaId);

        return "Deleted rezervacija id: " + rezervacijaId;
    }


    // dodavanje paketa
    @PutMapping("/rezervacija/{rezervacijaId}/turisticki-
paket/{turistickiPaketId}")
    public Rezervacija dodajTuristickiPaket(@PathVariable int rezervacijaId,
@PathVariable int turistickiPaketId) {
        return rezervacijaService.dodajTuristickiPaket(rezervacijaId,
turistickiPaketId);
    }

    // dodavanje klijenta
    @PutMapping("/rezervacija/{rezervacijaId}/klijent/{klijentId}")
    public Rezervacija dodajKlijenta(@PathVariable int rezervacijaId,
@PathVariable int klijentId) {
        return rezervacijaService.dodajKlijenta(rezervacijaId, klijentId);
    }

    // filtriranje
    @GetMapping("/rezervacije/sorted/{sortBy}")
    public List<Rezervacija> getSortedRezervacije(@PathVariable String sortBy) {
        return rezervacijaService.findAllSortedBy(sortBy);
    }

    @ExceptionHandler(NullPointerException.class)
    @ResponseStatus(value = HttpStatus.NOT_FOUND)
    public String handleNullPointerException(NullPointerException e) {
        return e.getMessage();
    }

    @ExceptionHandler(IllegalArgumentException.class)
    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
    public String handleIllegalArgumentException(IllegalArgumentException e) {
        return e.getMessage();
    }


}
```

Класу сам означио анотацијама:

- *@RestController* – прави RESTful веб сервисе коришћењем *Spring*-а *MVC*-а
- @RequestMapping – мапира веб захтев na klasu.

Урадио сам уметање зависности у конструктор класе.

Метода “*findAll*()” враћа листу објеката “*Rezervacija*”. Њу сам означио аннотацијом “@GetMapping”.

Метода “*getRezervacija ()*” објекат типа “*Rezervacija*”. Приступа објекту преко параметра “*rezervacijaId*” што се пренесе преко URL-а као параметар. За то користио анотацију “@PathVariable”. Методу сам означио анотацијом “@GetMapping”.

Метода “*addRezervacija()*” враћа објекат *Prodaja*,који треба да се дода. Њу сам означио аннотацијом “@PostMapping”.Параметар се прима као JSON податак. Параметар сам означио анотацијом “@RequestBody”. Уколико се дефинише примарни кључ као JSON захтеву онда се поставља вредност примарог кључа на 0. Тиме не омогућавамо да корисник уноси примарни кључ, и тако избегавамо преклапање примарних кључева.

Метода “*updateRezervacija()*” враћа објекат *Prodaja*, који треба да се ажурира. Њу сам означио аннотацијом “@PutMapping”.Параметар се прима као JSON податак. Параметар сам означио анотацијом “@RequestBody”. Уколико није дефинисан примарни кључ у HTTP захтеву онда се не врши ажурирање него се прави нови објекат са новим примарним кључем.

Метода “*deleteRezervacija()*” враћа String са исписом да ли је пронађен објекат или не. Метода је означена анотацијом “@DeleteMapping”. Ова метода прима као параметар примарни кључ објекта са анотацијом “@PathVariable”. Користи се метода “findById()” из зависности “*rezervacijaService*” да би проверили да ли објекат постоји и касније га избрисали са методом “deleteById()”.

Методе “*handleNullPointerException()*” служи за руковање изузетака. Користио сам две анотације за те методе “*@ExceptionHandler*” “*@ResponseStatus*”

Анотације:

- “@GetMapping ” - користи се за мапирање HTTP GET захтева

- “@PostMapping ” - користи се за мапирање HTTP POST захтева

- “@PutMapping ” - користи се за мапирање HTTP PUT захтева

- “@DeleteMapping ” - користи се за мапирање HTTP DELETE захтева

- “@PathVariable” – користи се да се параметар добије у виду URL-а

- “@RequestBody” – користи се да мапира “HttpRequest body” за даљи пренос, и тиме омогућава аутоматску десеријализацију долазног *HttpRequest body*-а као *Java* објекат.

- “*@ExceptionHandler*” – означава методу као “Handler method” да би *Spring* знао да се ради о руковању изузетка.

- “*@ResponseStatus*” – означава методу или изузетак да треба да врати “*Status Code*”.

Остале “*Controller*” класе су исте или сличне функционалности.

TuristickiPaketController:

```
package com.asss.pj.TuristickaAgencija.contoller;

import com.asss.pj.TuristickaAgencija.entity.TuristickiPaket;
import com.asss.pj.TuristickaAgencija.service.TuristickiPaketService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController()
@RequestMapping("/api")
public class TuristickiPaketController {

    private final TuristickiPaketService turistickiPaketService;

    @Autowired
    public TuristickiPaketController(TuristickiPaketService
turistickiPaketService) {
        this.turistickiPaketService = turistickiPaketService;
    }

    @GetMapping("/turisticki-paketi")
    public List<TuristickiPaket> getTuristickiPaket() {
        return turistickiPaketService.findAll();

    }

    @GetMapping("/turisticki-paket/{turistickiPaketId}")
    public Optional<TuristickiPaket> getRezervacija(@PathVariable int
turistickiPaketId) {

        Optional<TuristickiPaket> turistickiPaket =
turistickiPaketService.findById(turistickiPaketId);

        if (turistickiPaket == null) {
            throw new NullPointerException("Turisticki paket not found with id:
" + turistickiPaketId);
        }

        return turistickiPaket;
    }

    @PostMapping("/turisticki-paket")
    public TuristickiPaket addTuristickiPaket(@RequestBody TuristickiPaket
turistickiPaket) {

        // na ovaj nacin izbegavamo da korisnik doda id
        turistickiPaket.setId(0); // kasnije u bazi ce se postaviti konkretan id
        // u suprotnom bi promenio red sa datim id-em

        TuristickiPaket dbTuristickiPaket =
turistickiPaketService.save(turistickiPaket);

        return dbTuristickiPaket;
    }

    @PutMapping("/turisticki-paket")
    // koristi se za update ili za insert ukoliko ne postoji objekat u bazi
    public TuristickiPaket updateTuristickiPaket(@RequestBody TuristickiPaket
```

```
turistickiPaket) {
        // U principu je isto kao i addPacijent metoda
        // Ovim se samo prati pravilan http zahtev (PUT)

        TuristickiPaket dbTuristickiPaket =
turistickiPaketService.save(turistickiPaket);

        return dbTuristickiPaket;
    }


    @DeleteMapping("/turisticki-paket/{turistickiPaketId}")
    public String deleteTuristickiPaket(@PathVariable int turistickiPaketId) {

        Optional<TuristickiPaket> tempTuristickiPaket = null;

        tempTuristickiPaket =
turistickiPaketService.findById(turistickiPaketId);

        if (tempTuristickiPaket == null) {
            return "Rezervacija id not found: " + turistickiPaketId;
        }

        turistickiPaketService.deleteById(turistickiPaketId);

        return "Deleted turisticki paket id: " + turistickiPaketId;
    }


    @ExceptionHandler(NullPointerException.class)
    @ResponseStatus(value = HttpStatus.NOT_FOUND)
    public String handleNullPointerException(NullPointerException e) {
        return e.getMessage();
    }

    @ExceptionHandler(IllegalArgumentException.class)
    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
    public String handleIllegalArgumentException(IllegalArgumentException e) {
        return e.getMessage();
    }

}
```

SpecijalnaPonudaController:

```
package com.asss.pj.TuristickaAgencija.contoller;

import com.asss.pj.TuristickaAgencija.entity.SpecijalnaPonuda;
import com.asss.pj.TuristickaAgencija.service.SpecijalnaPonudaService;
import com.asss.pj.TuristickaAgencija.service.TuristickiPaketService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController()
@RequestMapping("/api")
public class SpecijalnaPonudaController {
```

```
    private final SpecijalnaPonudaService specijalnaPonudaService;
    private final TuristickiPaketService turistickiPaketService;

    @Autowired
    public SpecijalnaPonudaController(SpecijalnaPonudaService
specijalnaPonudaService, TuristickiPaketService turistickiPaketService) {
        this.specijalnaPonudaService = specijalnaPonudaService;
        this.turistickiPaketService = turistickiPaketService;
    }

    @GetMapping("/specijalne-ponude")
    public List<SpecijalnaPonuda> getSpecijalnePonude() {
        return specijalnaPonudaService.findAll();

    }

    @GetMapping("/specijalna-ponuda/{specijalnaPonudaId}")
    public Optional<SpecijalnaPonuda> getSpecijalnaPonuda(@PathVariable int
specijalnaPonudaId) {

        Optional<SpecijalnaPonuda> specijalnaPonuda =
specijalnaPonudaService.findById(specijalnaPonudaId);

        if (specijalnaPonuda == null) {
            throw new NullPointerException("Specijalna ponuda not found with id:
" + specijalnaPonudaId);
        }

        return specijalnaPonuda;
    }

    @PostMapping("/specijalna-ponuda")
    public SpecijalnaPonuda addSpecijalnaPonuda(@RequestBody SpecijalnaPonuda
specijalnaPonuda) {

        // na ovaj nacin izbegavamo da korisnik doda id
        specijalnaPonuda.setId(0); // kasnije u bazi ce se postaviti konkretan
id
        // u suprotnom bi promenio red sa datim id-em

        SpecijalnaPonuda dbSpecijalnaPonuda =
specijalnaPonudaService.save(specijalnaPonuda);

        return dbSpecijalnaPonuda;
    }

    @PutMapping("/specijalna-ponuda")
    // koristi se za update ili za insert ukoliko ne postoji objekat u bazi
    public SpecijalnaPonuda updateSpecijalnaPonuda(@RequestBody SpecijalnaPonuda
specijalnaPonuda) {
        // U principu je isto kao i addPacijent metoda
        // Ovim se samo prati pravilan http zahtev (PUT)

        SpecijalnaPonuda dbSpecijalnaPonuda =
specijalnaPonudaService.save(specijalnaPonuda);

        return dbSpecijalnaPonuda;
    }
```

```
    @DeleteMapping("/specijalna-ponuda/{specijalnaPonudaId}")
    public String deleteSpecijalnaPonuda(@PathVariable int specijalnaPonudaId) {

        Optional<SpecijalnaPonuda> tempTuristickiPaket = null;

        tempTuristickiPaket =
specijalnaPonudaService.findById(specijalnaPonudaId);

        if (tempTuristickiPaket == null) {
            return "Specijalna ponuda id not found: " + specijalnaPonudaId;
        }

        specijalnaPonudaService.deleteById(specijalnaPonudaId);

        return "Deleted specijalna ponuda id: " + specijalnaPonudaId;
    }

    @PutMapping("/specijalna-ponuda/{specijalnaPonudaId}/turisticki-
paket/{turistickiPaketId}")
    public SpecijalnaPonuda addTuristickiPaket(@PathVariable int
specijalnaPonudaId, @PathVariable int turistickiPaketId) {
        return specijalnaPonudaService.addTuristickiPaket(specijalnaPonudaId,
turistickiPaketId);
    }

    @ExceptionHandler(NullPointerException.class)
    @ResponseStatus(value = HttpStatus.NOT_FOUND)
    public String handleNullPointerException(NullPointerException e) {
        return e.getMessage();
    }


    @ExceptionHandler(IllegalArgumentException.class)
    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
    public String handleIllegalArgumentException(IllegalArgumentException e) {
        return e.getMessage();
    }

}
```

KlijentController:

```
package com.asss.pj.TuristickaAgencija.contoller;

import com.asss.pj.TuristickaAgencija.entity.Klijent;
import com.asss.pj.TuristickaAgencija.service.KlijentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController()
@RequestMapping("/api")
public class KlijentController {

    private final KlijentService klijentService;

    @Autowired
```

```
    public KlijentController(KlijentService klijentService) {
        this.klijentService = klijentService;
    }

    @GetMapping("/klijenti")
    public List<Klijent> getKlijents() {
        return klijentService.findAll();

    }

    @GetMapping("/klijent/{klijentId}")
    public Optional<Klijent> getKlijent(@PathVariable int klijentId) {

        Optional<Klijent> klijent = klijentService.findById(klijentId);

        if (klijent == null) {
            throw new NullPointerException("Klijent not found with id: " +
klijentId);
        }

        return klijent;
    }

    @PostMapping("/klijent")
    public Klijent addKlijent(@RequestBody Klijent klijent) {

        // na ovaj nacin izbegavamo da korisnik doda id
        klijent.setId(0); // kasnije u bazi ce se postaviti konkretan id
        // u suprotnom bi promenio red sa datim id-em

        Klijent dbKlijent = klijentService.save(klijent);

        return dbKlijent;
    }

    @PutMapping("/klijent")
    // koristi se za update ili za insert ukoliko ne postoji objekat u bazi
    public Klijent updateKlijent(@RequestBody Klijent klijent) {
        // U principu je isto kao i addPacijent metoda
        // Ovim se samo prati pravilan http zahtev (PUT)

        Klijent dbKlijent = klijentService.save(klijent);

        return dbKlijent;
    }


    @DeleteMapping("/klijent/{klijentId}")
    public String deleteKlijent(@PathVariable int klijentId) {

        Optional<Klijent> tempKlijent = null;

        tempKlijent = klijentService.findById(klijentId);

        if (tempKlijent == null) {
            return "Pacijent id not found: " + klijentId;
        }

        klijentService.deleteById(klijentId);

        return "Deleted pacijent id: " + klijentId;
```

```
    }

    @ExceptionHandler(NullPointerException.class)
    @ResponseStatus(value = HttpStatus.NOT_FOUND)
    public String handleNullPointerException(NullPointerException e) {
        return e.getMessage();
    }

    @ExceptionHandler(IllegalArgumentException.class)
    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
    public String handleIllegalArgumentException(IllegalArgumentException e) {
        return e.getMessage();
    }

}
```

4.5. Аспекти

Аспекти и Аспектно оријентисано програмирање (*AOP*) су концепти у развоју софтвера који омогућавају боље управљање аспектима система, као што су логовирање, трансакције и безбедност, који се простиру кроз различите делове апликације. У *Spring* фрејмворку, *AOP* омогућава раздвајање ових аспеката од основног бизнис логике кода.

Аспекти се односе на специфичне активности или функционалности које нису директно повезане са главном бизнис логиком, али су неопходне за функционисање система у целини. *Spring* омогућава дефинисање аспеката и њихову интеграцију у апликацију користећи *AOP*.

AOP у *Spring*-у се омогућава коришћењем анотација или конфигурационих фајлова. Аспекти се дефинишу као савети који се извршавају на одређеним тачкама извршавања програмског кода, као што су пре извршавања метода (*before*), након извршавања метода (*after*), и слично.

Класа “*RezervacijaLogging*”:

```
package com.asss.pj.TuristickaAgencija.aspect;


import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Aspect
@Component
@Order(6)
public class RezervacijaLogging {
```

```
    private Logger myLogger = Logger.getLogger(getClass().getName());

    /*
        () - odgovara metodi bez argumenata
        (*) - odgovara metodi sa jednim argumentom bilo kog tipa
        (..) - odgovara metodi sa 0 ili vise argumenta bilo kog tipa
     */

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.RezervacijaServiceImpl.findAll(..))")
    private void forFindAll() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.RezervacijaServiceImpl.findById(..))")
    private void forFindById() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.RezervacijaServiceImpl.save(..))")
    private void forSave() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.RezervacijaServiceImpl.deleteById(..))")
    private void forDeleteById() {

    }

    @Pointcut("forFindAll() || forFindById() || forSave() || forDeleteById()")
    private void forCrud() {

    }


    @Before("forCrud()")
    public void beforeMethod(JoinPoint theJoinPoint) {// JoinPoint ima
metapodatke o metodi koja je pozvana

        /*
            JoinPoint je koncept koji predstavlja određenu tačku tokom
izvršavanja programa,
            kao što je poziv metode ili pristup polju. Omogućava aspektima da se
zalažu na ovim tačkama i izvrše
            dodatno ponašanje, poznato kao aspect.
         */

        // display method we are calling
        String theMethod = theJoinPoint.getSignature().toShortString();
        myLogger.info("=====>> in @Before: calling method: " + theMethod);

        // display the arguments to the method

        // get the arguments
        Object[] args = theJoinPoint.getArgs();

        // loop through and display args
        for (Object tempArg : args) {
```

```
            myLogger.info("argument: " + tempArg);
        }
    }

    @AfterReturning(
            pointcut = "forCrud()",
            returning = "theResult"
    )
    public void afterReturning(JoinPoint theJoinPoint, Object theResult) {

        // display method we are returning from
        String theMethod = theJoinPoint.getSignature().toShortString();
        myLogger.info("=====>> in @AfterReturning: from method: " + theMethod);

        // display data returned
        myLogger.info("=====>> result: " + theResult);
    }
}
```

SpecijalnaPonudaLogging:

```
package com.asss.pj.TuristickaAgencija.aspect;


import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Aspect
@Component
@Order(4)
public class SpecijalnaPonudaLogging {

    private Logger myLogger = Logger.getLogger(getClass().getName());

        /*
            () - odgovara metodi bez argumenata
            (*) - odgovara metodi sa jednim argumentom bilo kog tipa
            (..) - odgovara metodi sa 0 ili vise argumenta bilo kog tipa
         */

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.SpecijalnaPonudaServiceImpl.findAll(..))"
)
    private void forFindAll() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.SpecijalnaPonudaServiceImpl.findById(..))
")
    private void forFindById() {

    }

    @Pointcut("execution(*
```

```
com.asss.pj.TuristickaAgencija.service.SpecijalnaPonudaServiceImpl.save(..))")
    private void forSave() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.SpecijalnaPonudaServiceImpl.deleteById(..
))")
    private void forDeleteById() {

    }

    @Pointcut("forFindAll() || forFindById() || forSave() || forDeleteById()")
    private void forCrud() {

    }


    @Before("forCrud()")
    public void beforeMethod(JoinPoint theJoinPoint) {// JoinPoint ima
metapodatke o metodi koja je pozvana

            /*
                JoinPoint je koncept koji predstavlja određenu tačku tokom
izvršavanja programa,
                kao što je poziv metode ili pristup polju. Omogućava aspektima
da se zalažu na ovim tačkama i izvrše
                dodatno ponašanje, poznato kao aspect.
             */

        // display method we are calling
        String theMethod = theJoinPoint.getSignature().toShortString();
        myLogger.info("=====>> in @Before: calling method: " + theMethod);

        // display the arguments to the method

        // get the arguments
        Object[] args = theJoinPoint.getArgs();

        // loop through and display args
        for (Object tempArg : args) {
            myLogger.info("argument: " + tempArg);
        }
    }

    @AfterReturning(
            pointcut = "forCrud()",
            returning = "theResult"
    )
    public void afterReturning(JoinPoint theJoinPoint, Object theResult) {

        // display method we are returning from
        String theMethod = theJoinPoint.getSignature().toShortString();
        myLogger.info("=====>> in @AfterReturning: from method: " + theMethod);

        // display data returned
        myLogger.info("=====>> result: " + theResult);
    }
}
```

TuristickiPaketLogging:

```
package com.asss.pj.TuristickaAgencija.aspect;


import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Aspect
@Component
@Order(3)
public class TuristickiPaketLogging {

    private Logger myLogger = Logger.getLogger(getClass().getName());

        /*
            () - odgovara metodi bez argumenata
            (*) - odgovara metodi sa jednim argumentom bilo kog tipa
            (..) - odgovara metodi sa 0 ili vise argumenta bilo kog tipa
         */

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.TuristickiPaketServiceImpl.findAll(..))")
    private void forFindAll() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.TuristickiPaketServiceImpl.findById(..))"
)
    private void forFindById() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.TuristickiPaketServiceImpl.save(..))")
    private void forSave() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.TuristickiPaketServiceImpl.deleteById(..)
)")
    private void forDeleteById() {

    }

    @Pointcut("forFindAll() || forFindById() || forSave() || forDeleteById()")
    private void forCrud() {

    }


    @Before("forCrud()")
```

```
    public void beforeMethod(JoinPoint theJoinPoint) {// JoinPoint ima
metapodatke o metodi koja je pozvana

            /*
                JoinPoint je koncept koji predstavlja određenu tačku tokom
izvršavanja programa,
                kao što je poziv metode ili pristup polju. Omogućava aspektima
da se zalažu na ovim tačkama i izvrše
                dodatno ponašanje, poznato kao aspect.
             */

        // display method we are calling
        String theMethod = theJoinPoint.getSignature().toShortString();
        myLogger.info("=====>> in @Before: calling method: " + theMethod);

        // display the arguments to the method

        // get the arguments
        Object[] args = theJoinPoint.getArgs();

        // loop through and display args
        for (Object tempArg : args) {
            myLogger.info("argument: " + tempArg);
        }
    }

    @AfterReturning(
            pointcut = "forCrud()",
            returning = "theResult"
    )
    public void afterReturning(JoinPoint theJoinPoint, Object theResult) {

        // display method we are returning from
        String theMethod = theJoinPoint.getSignature().toShortString();
        myLogger.info("=====>> in @AfterReturning: from method: " + theMethod);

        // display data returned
        myLogger.info("=====>> result: " + theResult);
    }
}
```

KlijentLogging:

```
package com.asss.pj.TuristickaAgencija.aspect;


import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Aspect
@Component
@Order(3)
public class KlijentLogging {
```

```
    private Logger myLogger = Logger.getLogger(getClass().getName());

    /*
        () - odgovara metodi bez argumenata
        (*) - odgovara metodi sa jednim argumentom bilo kog tipa
        (..) - odgovara metodi sa 0 ili vise argumenta bilo kog tipa
     */

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.KlijentServiceImpl.findAll(..))")
    private void forFindAll() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.KlijentServiceImpl.findById(..))")
    private void forFindById() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.KlijentServiceImpl.save(..))")
    private void forSave() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.KlijentServiceImpl.deleteById(..))")
    private void forDeleteById() {

    }

    @Pointcut("forFindAll() || forFindById() || forSave() || forDeleteById()")
    private void forCrud() {

    }


    @Before("forCrud()")
    public void beforeMethod(JoinPoint theJoinPoint) {// JoinPoint ima
metapodatke o metodi koja je pozvana

        /*
            JoinPoint je koncept koji predstavlja određenu tačku tokom
izvršavanja programa,
            kao što je poziv metode ili pristup polju. Omogućava aspektima da se
zalažu na ovim tačkama i izvrše
            dodatno ponašanje, poznato kao aspect.
         */

        // display method we are calling
        String theMethod = theJoinPoint.getSignature().toShortString();
        myLogger.info("=====>> in @Before: calling method: " + theMethod);

        // display the arguments to the method

        // get the arguments
        Object[] args = theJoinPoint.getArgs();

        // loop through and display args
```

```
        for (Object tempArg : args) {
            myLogger.info("argument: " + tempArg);
        }
    }

    @AfterReturning(
            pointcut = "forCrud()",
            returning = "theResult"
    )
    public void afterReturning(JoinPoint theJoinPoint, Object theResult) {

        // display method we are returning from
        String theMethod = theJoinPoint.getSignature().toShortString();
        myLogger.info("=====>> in @AfterReturning: from method: " + theMethod);

        // display data returned
        myLogger.info("=====>> result: " + theResult);
    }
}
```

TuristickiPaketObavestenje:

```
package com.asss.pj.TuristickaAgencija.aspect;

import com.asss.pj.TuristickaAgencija.entity.TuristickiPaket;
import com.asss.pj.TuristickaAgencija.service.TuristickiPaketService;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import java.util.Optional;
import java.util.logging.Logger;

@Aspect// proxy design patter
@Component//
@Order(2)
public class TuristickiPaketObavestenje {

    private final Logger myLogger = Logger.getLogger(getClass().getName());

    private final TuristickiPaketService turistickiPaketService;

    @Autowired
    public TuristickiPaketObavestenje(TuristickiPaketService
turistickiPaketService) {
        this.turistickiPaketService = turistickiPaketService;
    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.contoller.TuristickiPaketController.updateTuristi
ckiPaket(..))")
    private void forUpdate() {

    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.TuristickiPaketServiceImpl.deleteById(..)
)")
```

```
    private void forDeleteById() {

    }


    @Before("forUpdate() && args(turistickiPaket)")
    public void obavestenjeForUpdate(JoinPoint theJoinPoint, TuristickiPaket
turistickiPaket) {

        myLogger.info("Turisticki paket sa destinacijom " +
turistickiPaket.getDestinacija() + " je promenjen!");
        myLogger.info("\n" + "Posle azuriranja: ");

        myLogger.info("\n" + turistickiPaket.toString() + "");

    }

    @Before("forDeleteById() && args(turistickiPaketId)")
    public void obavestenjeForDeleting(JoinPoint theJoinPoint, int
turistickiPaketId) {
        System.out.println("logg");

        Optional<TuristickiPaket> turistickiPaketOptional =
turistickiPaketService.findById(turistickiPaketId);

        if (turistickiPaketOptional.isPresent()) {
            TuristickiPaket turistickiPaket = turistickiPaketOptional.get();

            myLogger.info("Turisticki paket sa destinacijom " +
turistickiPaket.getDestinacija() + " je obrisan!");
        }
    }

}
```

TuristickiPaketUpdate:

```
package com.asss.pj.TuristickaAgencija.aspect;

import com.asss.pj.TuristickaAgencija.entity.SpecijalnaPonuda;
import com.asss.pj.TuristickaAgencija.entity.TuristickiPaket;
import com.asss.pj.TuristickaAgencija.repo.TuristickiPaketRepo;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import java.util.logging.Logger;

@Aspect// proxy design patter
@Component//
@Order(1)
public class TuristickiPaketUpdating {

    private final Logger myLogger = Logger.getLogger(getClass().getName());

    private final TuristickiPaketRepo turistickiPaketRepo;
```

```
    @Autowired
    public TuristickiPaketUpdating(TuristickiPaketRepo turistickiPaketRepo) {
        this.turistickiPaketRepo = turistickiPaketRepo;
        System.out.println("TuristickiPaketUpdating Aspect Initialized");
    }

    // Define a pointcut that matches the save and update operations for
SpecijalnaPonuda
    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.repo.SpecijalnaPonudaRepo.save(..))")
    public void saveSpecijalnaPonuda() {
    }

    @Pointcut("execution(*
com.asss.pj.TuristickaAgencija.service.SpecijalnaPonudaServiceImpl.initializeSpe
cijalnaPonuda(..))")
    public void initializeSpecijalnaPonuda() {
    }
//
//    @After("execution(*
com.asss.pj.TuristickaAgencija.service.SpecijalnaPonudaServiceImpl.save(..))")
//    public void logAfterSave() {
//        System.out.println("Aspect triggered after saving SpecijalnaPonuda.");
//    }
//
//    // After successfully saving or updating SpecijalnaPonuda, update the
price of TuristickiPaket

    // fixme: ne radi
    @AfterReturning(pointcut = "saveSpecijalnaPonuda() ||
initializeSpecijalnaPonuda()", returning = "result")
    public void updateCenaAfterSavingSpecijalnaPonuda(Object result) {
        myLogger.info("Aspect triggered after saving SpecijalnaPonuda.");

        if (result instanceof SpecijalnaPonuda specijalnaPonuda) {
            TuristickiPaket turistickiPaket =
specijalnaPonuda.getTuristickiPaket();
            if (turistickiPaket != null) {
                double price = turistickiPaket.getCena();
                double discount = specijalnaPonuda.getPopust();
                double updatedPrice = price * (100 - discount) / 100;

                // Update the price of the TuristickiPaket
                turistickiPaket.setCena(updatedPrice);
                turistickiPaketRepo.save(turistickiPaket);

                myLogger.info("Updated cena of TuristickiPaket with ID " +
turistickiPaket.getId() +
                        " to " + updatedPrice + "\nThe procent of discount is "
+ discount + "%");
            }
        }
    }


}
```

Ове класа имају анотације:

- @Aspect – означава одређену класу да је аспект

- @Component - означава одређену класу да може да се направи зрно (bean)

- @Order – означава сортиран редослед аспекта. Којим редоследом ће се који аспект вршити

Направио сам променљиву “logger” да би могао да логујем апликацију.

Методи који имају анотацију “@PointCut” служе као метод показивања који део кода у апликаци би требао да се “пресече”.

Методи са анотацијом “@Before” служи да пресече одговарајући део кода пре његовог извршења.

Са методом “beforeMethod()” пресреће све методе које сам навео у методи “forCrud()”. Параметар “JoinPoint” – је концепт који представља одређену тачку током извршавања програма, као што је позив методе или приступ пољу. Омогућава аспектима да се залажу на овим тачкама и изврше додатно понашање, познато као аспект.

Ја сам преко JoinPoint-а исписао име методе као и његове аргументе. Касни је сам логовао методу и аргументе те методе.

Метода “afterReturning()” се догађа након успешно одрађене методе (све методе које сам навео у методи “forCrud()”). У овој методи логујем име методе на којој се извршио аспект. Као и крајњи резултат методе.

4. Закључак

Пројекат "Продаја и сервис мобилних телефона" представља успешно имплементиран *RESTful API* користећи *Java Spring Boot framework*, са *XAMPP* апликацијом као *servlet*-ом и *MySql* базом података. Кроз употребу *MVC* архитектуре, пројекат обезбеђује јасано раздвајање одговорности између различитих компоненти система, чинећи га модуларним и лако одрживим.

Имплементација *Spring Data JPA* омогућила је ефикасан приступ подацима уз минимално писање *SQL* упита, док су ентитети и *DTO* (*Data Transfer Object*) обезбедили јасно дефинисану структуру података и њихово сигурно преношење кроз различите слојеве апликације. Коришћење сервиса за имплементацију пословне логике и контролера за руковање *HTTP* захтевима додатно је унапредило флексибилност и скалабилност апликације.

Овај пројекат не само да задовољава постављене функционалне захтеве, већ пружа и солидну основу за даље унапређење и проширење функционалности. Интеграција најновијих технологија и придржавање добрих пракси у развоју софтвера чини овај систем спремним за будући раст и прилагођавање специфичним потребама корисника.

Литература

[1]Миодраг Живковић и други. Programski jezici. Univerzitet Singidunum, Beograd, 2018.

[2] Ranga Rao Karanam. Mastering Spring 5.

[3] Bealdung. Persistence with Spring.