



Академија струковних
студија Шумадија
Одсек Крагујевац

Студијски програм: Информатика
Предмет: Програмски језици

ПЕКАРА

-Пројектни задатак-

Предметни наставник:
Др Александар Мишковић

Студент:
Магдалена Јакшић 23/2020

Крагујевац 2023.

САДРЖАЈ

1. Увод	3
2. Spring	4
2.1. POJO	4
2.2. Инверзија контроле.....	4
2.3. Инјектовање зависности	4
2.4 RESTful сервис.....	5
3. Поставка пројектног задатка	5
4. Структура пројектног задатка.....	6
5. pom.xml фајл	7
6. application.properties.....	9
7. pekara.sql	10
7.1. entity пакет	10
8. pekaraRepository класа	12
9. PekaraService интерфејс	13
9.1. PekaraServiceImpl класа	13
10. PekaraRest класа.....	15
11. POSTMAN	17
12. Закључак.....	18
Литература	19

1.Увод

Пројекат представља апликацију за праћење стања у бази података пекаре. Апликација прати стање базе по одређеним параметрима и има функционалности као што су преглед стања, чување и брисање производа. Све активности које се дешавају у апликацији складиште се у текстуалну датотеку.

Технологија која је коришћена у развоју апликације је Spring Framework уз помоћ Spring Boot-а ради развијања RESTful сервиса.

2. Spring

Spring је open-source радни оквир који се појавио као једноставнија и лакша за руковање алтернатива JEE(eng. Java Enterprise Edition). Нуди технологије попут POJO(eng. Plain Old Java Object), инверзије контроле, аспектно оријентисаног програмирања и инјектовања зависности.

2.1. POJO

POJO класе могу структурно да се искористе у било ком Јавином развојном окружењу, које није Spring. Spring се базира на „не инвазивном програмирању“. Spring-ове класе су углавном означене анотацијом која говори Spring-у шта класа представља. Нуди лабаву спрегу(eng. loose coupling) која се односи на што мању зависност измеђи повезаних објеката. Често се користи термин зрно (eng. bean) када је реч о компонентима апликације.

2.2. Инверзија контроле

Инверзија контроле је техника која повезује зрна у тренутку извршења, иако веза не постоји у тренутку компајлирања. Омогућава већи степен модуларности у програмирању. Сва контрола се преноси на програм и контејнере, за разлику од традиционалног програмирања где је сва контрола на класи.

2.3. Инјектовање зависности

Тек у тренутку креирања објекта додељује се његова зависност, а те зависности додељује нека друга страна која је задужена за то у систему. Када се примени инјектирање зависности објекти више не прибављају сами референце на објекте од којих су зависни.

2.4 RESTful сервис

REST тип сервиса који карактерише његова лакоћа што се тиче потрошње ресурса, одржавања и скалабилности. За разлику од SOAP сервиса који користи само XML формат за приказ својих објеката, REST подржава велики спектар формата : XML, JSON, YAML, и многе друге. REST може користити само HTTP протокол за комуникацију, а SOAP може користити HTTP или MQ или неки други протокол.

RESTful сервиси се базирају на шест кључних елемената:

1. Ресурси(eng. resources)
2. Глаголи захтева(eng. request verbs)
3. Заглавље захтева(eng. request headers)
4. Тело захтева(eng. request body)
5. Тело одговора(eng. response body)
6. Статус кодови одговора(eng. response status codes)

3. Поставка пројектног задатка

Направити веб апликацију засновану на RESTful сервису за потребе једне пекаре. Апликација треба да омогућити праћења стања у бази података о укупном броју производа: тип производа: хлеб, кроасан, кифла (ЕНУМ тип), цена и количина. У пекари мора да се налази минимум 3 производа од сваке врсте. Ако ови параметри нису задовољени кориснику апликације одмах исписати поруку о грешци и апликацију затворити.

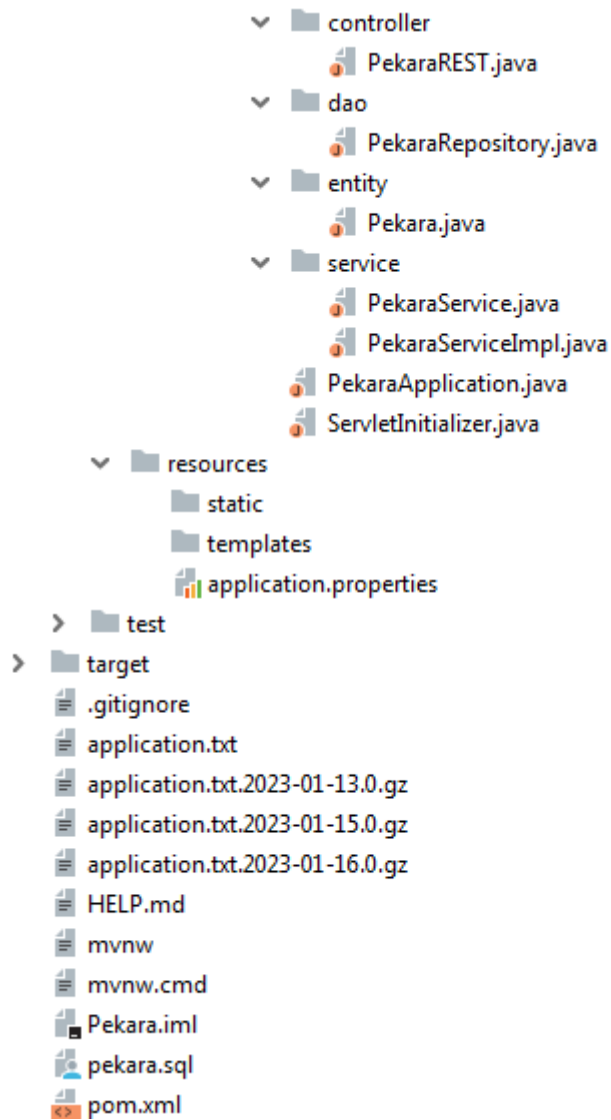
Ваша апликација треба да имплементира следеће функционалности:

1. Приказ свих производа са свим наведеним параметрима.
2. Приказ свих производа у опадајућем редоследу према вредности параметра: цена.
3. Унос новог производа у базу.
4. Измену постојећег производа у бази.
5. Брисање производа из базе.

Помоћу аспеката обезбедити праћење активности над апликацијом тако да се у локалну текстуалну датотеку (која треба да се налази у коренском директоријуму пројекта) уписују и чувају извештаји о покренутим функционалностима.

Ваш код треба да буде написан тако да се задовољи вишеслојност (multi-layer) и то тако да се написани код расподели у онај слој којем по природи припада (као на вежбама).

4. Структура пројектног задатка



5. pom.xml фајл

Пројектни објектни модел је основна датотека неопходна за рад у Мавену. Садржи информације о пројекту и конфигурацији које Мавен користи за изградњу пројекта.

```
<?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>
```

Приказану конфигурацију Мавен сам генерише и у њу спадају верзија пројекта и сајт са ког се врши преузимање зависности.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.4.1</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Конфигурација Spring Parent-а преузима велики део конфигурације Spring-а и помаже око рада са зависностима. Да би грешке у конфигурацији Spring-а биле минималне може се користити сајт за аутоматску конфигурацију spring пројекта(<https://start.spring.io>) где се дефинише име, верзија ,опис пројекта, име главног пакета верзија Јаве и зависности које су потребне.


```
<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>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>
</dependencies>
```

Spring data JPA олакшава рад са базама података користећи Java Persistence API. Spring Web омогућава креирање апликације са RESTful сервисима, користећи SPRING MVC, и Apache Tomcat на коме се подиже сервер. MySQL Drive омогућава повезивање са MySQL базом података.














6. application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/pekara
spring.datasource.username=root
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto= update

logging.file.name=application.txt
logging.level.org.springframework.web= DEBUG
logging.level.org.hibernate= ERROR
```

Application.properties служи за конфигурацију повезивања апликације са базом података. Декларише се директна путања ка бази и корисничко име са којим ће апликација приступити бази података. Дефинише се logger који прати активности у конзоли и уписује у фајл application.txt и бележи DEBUG и ERROR поруке пошто је стандардо дефинисан да само бележи INFO поруке.

7. pekara.sql

<input type="checkbox"/>	1	id 	int(3)	No	None	 Change  Drop  More
<input type="checkbox"/>	2	tipProizvoda	enum("HLEB", "KROASAN", "KIFLA") utf8mb4_general_ci	No	None	 Change  Drop  More
<input type="checkbox"/>	3	cena	int(5)	No	None	 Change  Drop  More
<input type="checkbox"/>	4	kolicina	int(11)	No	None	 Change  Drop  More

База садржи 4 поља: примарни клуч, тип производа са својим енул вредностима, цена и количина. По бази података креира се Entity класа у пројекту која мора да има потпуно исте типове поља као база.

7.1. entity пакет

```
@Entity
@Table(name = "Pekara")
public class Pekara {
    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)
    private int id;
    @Column(name = "tipProizvoda")
    private String tipProizvoda;
    @Column(name = "cena")
    private int cena;
    @Column(name = "kolicina")
    private int kolicina;

    public Pekara() {}

    public Pekara(int id, String tipProizvoda, int cena, int kolicina) {
        this.id = id;
        this.tipProizvoda = tipProizvoda;
        this.cena = cena;
        this.kolicina = kolicina; }

    public int getId() {
        return id;
    }

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

    public String getTipProizvoda() {
        return tipProizvoda;
    }

    public void setTipProizvoda(String tipProizvoda) {
        this.tipProizvoda = tipProizvoda;
    }

    public int getCena() {
        return cena;
    }

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

    public int getKolicina() {
        return kolicina; }

    public void setKolicina(int kolicina) {
        this.kolicina = kolicina;
    }
}
```

```

@Override
public String toString() {
    return "Pekara{" +
        "id=" + id +
        ", tip=" + tipProizvoda + '\'' +
        ", cena=" + cena + '\'' +
        ", kolicina=" + kolicina + '\'' +
        '}';
}

public enum TipProizvoda {
    HLEB("HLEB"),
    KROASAN("KROASAN"),
    KIFLA("KIFLA");

    private String tipProizvoda;

    TipProizvoda(String tipProizvoda) {
        this.tipProizvoda = tipProizvoda;
    }

    public String getTipProizvoda() {
        return tipProizvoda;
    }

    public void setTipProizvoda(String tipProizvoda) {
        this.tipProizvoda = tipProizvoda;
    }
}
}

```

Класа **Pekara** је POJO класа, састоји се само из поља класе, конструктора, гетера и сетера и `toString` методе. У овој класи постоји еnum поткласа **TipProizvoda**.

Анотација **@Entity** представља анотацију која казује Spring-у да је то класа ентитет и да се користи за комуникацију са базом. Анотација **@Table** представља анотацију која казује Spring-у да се ради о класи која се везује за табелу из базе под именом „Пекара“. Анотација **@Id** означава поље које је ID поље у бази података. Анотација **@GeneratedValue** значи да ће њене вредности поља бити аутоматски генерисане. Анотација **@Column** означава поља у класи за која се везују поља из базе података.

8. pekaraRepository класа

```
@Repository
public interface PekaraRepository extends JpaRepository<Pekara,Integer> {
    @Query("FROM Pekara ORDER BY cena DESC")
    List<Pekara> findAllPekarasByCenaDescending();
}
```

Класа `PekaraRepository` је класа која наслеђује класу `JpaRepository`, која омогућава рад са базом података. Класа је означена са аномацијом **@Repository** и представља репозиторијум базе података.

Дефинишу се функционалности које не постоје у `JpaRepository` класи. Означавају се аномацијом **@Query** која садржи SQL захтев који ће се извршити над базом података. Захтевани су производи у опадајућем редоследу по цени. Функционалности које спадају под тражењем свих записа, тражење записа по ID-у, чување и брисање се већ налазе у класи `JpaRepository`.

9. PekaraService интерфејс

```
List<Pekara> findAll();
Pekara findById(int id);

List<Pekara> findAllPekarasByCenaDescending();

Pekara save(Pekara pekara);

String updateById(int id, Pekara pekara);

String deleteById(int id);

String proveraStanja();
```

9.1. PekaraServiceImpl класа

```
@Service
public class PekaraServiceImpl implements PekaraService{
    @Autowired
    private PekaraRepository pekaraRepository;

    @Override
    public List<Pekara> findAll() {
        return pekaraRepository.findAll();
    }

    @Override
    public Pekara findById(int id) {
        return pekaraRepository.findById(id).orElse(null);
    }

    @Override
    public List<Pekara> findAllPekarasByCenaDescending() {
        return pekaraRepository.findAllPekarasByCenaDescending();
    }

    @Override
    public Pekara save(Pekara pekara) {
        return pekaraRepository.save(pekara);
    }

    @Override
    public String updateById(int id, Pekara pekara) {
        Pekara tempPekara = findById(id);
        if (tempPekara != null) {
            tempPekara.setKolicina(pekara.getKolicina());
            tempPekara.setTipProizvoda(pekara.getTipProizvoda());
            tempPekara.setCena(pekara.getCena());
            System.out.println(proveraStanja());
            return "PROIZVOD" + id + " je prepravljen.";
        } else {
            return "PROIZVOD" + id + " ne postoji u bazi.";
        }
    }

    @Override
    public String deleteById(int id) {
        Pekara pekara = findById(id);
        if (pekara != null) {
            pekaraRepository.deleteById(id);
            return "PROIZVOD " + id + " je obrisana.\n" + proveraStanja();
        } else {
            return "PROIZVOD" + id + " ne postoji u bazi.";
        }
    }
}
```

```

    }
}

@Override
public String proveraStanja() {
    List<Pekara> listaPekara = findAll();
    int tempBrojac = 0;

    for (Pekara pekara : listaPekara) {
        if (pekara.getTipProizvoda().equals("KIFLA")) {
            tempBrojac++;
        }
    }
    tempBrojac=0;

    for (Pekara pekara : listaPekara) {
        if (pekara.getTipProizvoda().equals("KROASAN")) {
            tempBrojac++;
        }
    }
    if (tempBrojac < 3) {
        return "Pekara ima manje od 3 kroasana!!!";
    }
    tempBrojac = 0;

    for (Pekara pekara : listaPekara) {
        if (pekara.getTipProizvoda().equals("KIFLA")) {
            tempBrojac++;
        }
    }
    if (tempBrojac < 3) {
        return "Pekara ima manje od 3 kifle!!!";
    }
    return null;
}
}

```

Класа **KnjigaServiceImpl** имплементира интерфејс **KnjigaService** и дефинише све његове методе. Означена је анотацијом **@Service** која означава да је сва бизнис логика у тој класи. Креиран је један репозиторијум типа **PekaraRepository** и аутоматски је ожичен анотацијом **@Autowired**. Ово зрно се користи у свим функцијама ове класе јер је оно директна веза са базом података. Функција **updateById()** тражи да ли постоји производ у бази са датим Id-јем, ако не враћа поруку да производ не постоји у бази, у супротном се тренутно памти у привремену променљиву. Након тога се производ преправља са подацима из добијеног објекта и шаље се на чување уз поруку да је производ успешно преправљен. Функција **deleteById()** функционише на истом принципу само што се производ брише из базе. Током сваке промене стања, преправке или брисања, проверава се стање у бази и враћа се порука којом се обавештава корисник.

10. PekaraRest класа

```

@RestController
public class PekaraREST {
    @Autowired
    private PekaraService pekaraService;
    @GetMapping("/pekara")
    public List<Pekara> findAll(){
        return pekaraService.findAll();
    }
    @GetMapping("/pekara/{id}")
    public Pekara findById(@PathVariable int id) {
        return pekaraService.findById(id);
    }

    @GetMapping("/pekara/cena")
    public List<Pekara> findAllPekarasByCenaDescending() {
        return pekaraService.findAllPekarasByCenaDescending();
    }

    @PostMapping("/pekara")
    public Pekara save(@RequestBody Pekara pekara) {
        return pekaraService.save(pekara);
    }

    @PutMapping("/pekara/{id}")
    public String updateById(@PathVariable int id, @RequestBody Pekara pekara) {
        return pekaraService.updateById(id, pekara);
    }

    @DeleteMapping("/pekara/{id}")
    public String deleteById(@PathVariable int id) {
        return pekaraService.deleteById(id);
    }
}

```

Класа **PekaraRest** је означена са анотацијом **@RestController** која означава да је класа REST контролер. Садржи једно ожичено зрно класе **PekaraService** коју позива у свим својим методима уз одговарајуће параметре и садржи мапирање функција. Свака функција у класи је мапирана на посебну адресу и тип приступа. Типови приступа су **HTTP** типови приступа (get, post, put, delete).

Функције **findAll()**, **findById()** и **findAllPekarasByCenaDescending()** траже податке из базе и означене су са анотацијом **@GetMapping**, приступају бази по get мапирању, траже или добављају информације из базе. Параметар функције **findById()** је означен анотацијом **@PathVariable** која означава да ће се вредност ове променљиве извући из адресе која је унета.

Функција **save()** служи за уписивање нових производа у базу података и означена је анотацијом **@PostMapping**, приступа бази по post мапирању, може само да уписује производе у базу података. Податак који прима ова функција је означен анотацијом **@RequestBody**, што значи да се из захтева за чување производа мора послати „тело“ производа, попуњени објект који може бити у неком формату који се лако преноси (нпр. JSON формат).

Функција **updateById()** преправља податке из базе података и означена је са анотацијом **@PutMapping**, приступа бази у put мапирању, може само мењати производе из базе података. Користи исту функцију као и претходна само у дугачијем мапирању.

Функција **deleteById()** брише одређен производ из базе података и означена је са анотацијом **@DeleteMapping**, приступа бази у delete мапирању, што може само брисати производе из базе података.

11. POSTMAN

http://localhost:8080/pekara/1

Save

DELETE

http://localhost:8080/pekara/1

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```

1
2
3
4
5
6
    {
      "tipProizvoda": "KIFLA",
      "količina": 2,
      "cena": 70
    }

```

ody

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 81 ms

Size: 188 B

Save Response

Pretty

Raw

Preview

Visualize

Text

1

PROIZVOD JE OBRISAN

http://localhost:8080/pekara

Save

GET

http://localhost:8080/pekara

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```

1
2
3
4
5
6
    {
      "tipProizvoda": "KIFLA",
      "količina": 5,
      "cena": 70
    }

```

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 47 ms

Size: 220 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```

1
2
3
4
5
6
7
8
{
  "id": 1,
  "tipProizvoda": "KIFLA",
  "cena": 70,
  "količina": 5
}

```

PUT

http://localhost:8080/pekara/1

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```

1
2
3
4
5
    {
      "id": 1,
      "tipProizvoda": "KIFLA",
      "količina": 2,
      "cena": 70
    }

```

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 21 ms

Size: 185 B

Save Response

Pretty

Raw

Preview

Visualize

Text

1

PROIZVOD JE IZMENJEN.

http://localhost:8080/pekara

Save

POST

http://localhost:8080/pekara

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```

1
2
3
4
5
6
    {
      "tipProizvoda": "KIFLA",
      "količina": 5,
      "cena": 70
    }

```

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 436 ms

Size: 218 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```

1
2
3
4
5
6
{
  "id": 1,
  "tipProizvoda": "KIFLA",
  "cena": 70,
  "količina": 5
}

```

12. Закључак

Spring – ова популарност расте и тако ће наставити јер је прављен од програмера за програмере. Ово окружење је донело иновације у свет програмирања и много олакшао рад у том пољу. Иако изгледа компиковано показало се да је продуктиван и ефикаснији за рад. Spring је програм који се показао ненадмашивим у развоју огромних апликација које имају доста комплексну пословну логику. Он је најбоље развојно окружење за велике апликације које ће бити одржаване и којој ће приступати велики број корисника.

Литература

1. Čolak, S. (2015). Izrada web aplikacije u razvojnem okruženju Spring (Završni rad) Preuzeto sa: https://bib.irb.hr/datoteka/717071.1-Izrada_web_aplikacije_u_razvojnem_okruenju_Spring.pdf
2. Williams, W. C. (2019). *Spring and all*. Dover Publications.
3. Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Risberg, T., ... & Webb, P. (2004). The spring framework–reference documentation. *interface*, 21, 27.