



IT355 - WEB SISTEMI 2

Integracija Spring (Boot) i Angular

Lekcija 13

PRIRUČNIK ZA STUDENTE

IT355 - WEB SISTEMI 2

Lekcija 13

INTEGRACIJA SPRING (BOOT) I ANGULAR

- ✓ Integracija Spring (Boot) i Angular
- ✓ Poglavlje 1: Razvoj Spring serverske komponente
- ✓ Poglavlje 2: Kreiranje Spring Maven projekta
- ✓ Poglavlje 3: Kreiranje i podešavanje baze podataka
- ✓ Poglavlje 4: Kreiranje Java konfiguracija
- ✓ Poglavlje 5: Kreiranje modela
- ✓ Poglavlje 6: Kreiranje DAO i servisnog nivoa
- ✓ Poglavlje 7: REST kontoler
- ✓ Poglavlje 8: Razvoj Angular klijentske komponente
- ✓ Poglavlje 9: Pokazna vežba 13
- ✓ Poglavlje 10: Individualna vežba 11
- ✓ Poglavlje 11: Domaći zadatak 13
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Kombinovanje naprednih frontend i backend tehnologija

Lekcija će se baviti čestim pristupom kojeg je moguće sresti u savremenoj industriji veb softvera - kombinovanje naprednih *frontend* i *backend* tehnologija:

- *Angular* - *JavaScript* frontend radni okvir za razvoj veb aplikacija;
- *Spring MVC* - *Java* radni okvir za razvoj veb aplikacija koji će biti korišćen za razvoj serverskog dela aplikacije.

Cilj ove lekcije jeste da nauči studente kako se pristupa razvoju potpuno funkcionalne veb aplikacije, nad bazom podataka, koja će koristiti okvir za podršku skripting programiranju za razvoj klijent strane aplikacije, a takođe i *Spring (Boot)* okvir, koji se oslanja na primenu pravog programskog jezika, za kreiranje servisa nad bazom podataka za obradu zahteva klijenata.

Savladavanjem ove lekcije student će naučiti da kombinuje i koristi tehnologije i alate obrađene i naučene u predmetima *IT255 - Vebsistemi 1* i *IT355 - Veb sistemi 2*.

▼ Poglavlje 1

Razvoj Spring serverske komponente

DEFINISANJE ZAHTEVA

Razvoj jedne potpuno funkcionalne savremene veb aplikacije nad bazom podataka.

U ovom delu lekcije biće demonstrirana primena Spring razvojnog okvira za kreiranje pozadinskih servisa nad bazom podataka aplikacije. Kao osnov za analizu i diskusiju biće upotrebljen konkretan primer sa radnim nazivom: "*Book Management System*". Cilj je da primer bude što realniji i da demonstrira razvoj jedne potpuno funkcionalne savremene veb aplikacije čije su najvažnije funkcionalnosti CRUD operacije nad bazom podataka:

- *CREATE* - kreiranje nove knjige u bazi podataka;
- *READ* - čitanje knjige iz baze podataka;
- *UPDATE* - ažuriranje knjiga iz baze podataka;
- *DELETE* - brisanje knjiga iz baze podataka.

▼ Poglavlje 2

Kreiranje Spring Maven projekta

KREIRANJE KOMPONENTE BOOK API

Neophodno je definisati krajnje tačke REST servisa aplikacije.

Prvi korak u kreiranju željene veb aplikacije jeste kreiranje REST API komponente aplikacije. Ova komponenta će na dalje biti poznata pod nazivom - *Book API*.

U ovom delu izlaganja biće neophodno definisati i priložiti krajnje tačke *REST* servisa aplikacije. Ovi koncepti, kao što je već bilo istaknuto u lekciji Spring REST, koriste URI linkove za pristup kreiranim *REST* servisima.

Aplikacija koja će biti kreirana koristi *HTTP* operacije i krajnje tačke *REST* servisa kao na sledećoj slici:

GET	–	/api/book
GET	–	/api/book/{bookid}
POST	–	/api/book
PUT	–	/api/book/{bookid}
DELETE	–	/api/book/{bookid}

Slika 2.1 Operacije i krajnje tačke REST servisa [izvor: autor]

Sa slike je moguće primetiti da će biti korišćena *GET* operacija za:

- prikaz svih dostupnih knjiga;
- prikaz pojedinačnih knjiga iz baze podataka.

Takođe, biće korišćene *POST* i *PUT* metode za:

- dodavanje novih knjiga u bazu podataka;
- za ažuriranje unetih knjiga u bazu podataka;

Konačno, biće upotrebljena i *DELETE* metoda za:

- brisanje knjiga koje se čuvaju u bazi podataka.

HTTP pozivi, koji odgovaraju navedenim operacijama biće diskutovani i demonstrirani u delu lekcije koja se bavi frontend razvojem primenom razvojnog okvira *Angular*.

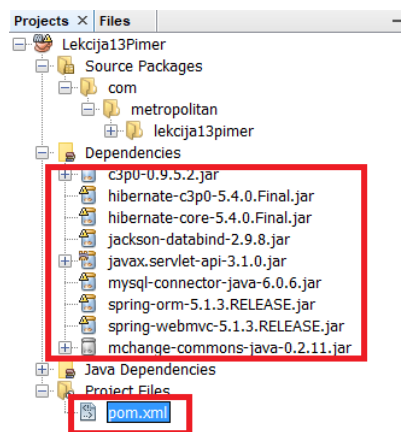
KREIRANJE SPRING MAVEN PROJEKTA

Neophodno je obaviti kreiranje backend Spring projekta.

U nastavku, neophodno je obaviti kreiranje *Spring* projekta po sledećim koracima:

- dodavanje svih neophodnih zavisnosti u projekat;
- kreiranje klasa aplikacije raspoređenih po odgovarajućim paketima;
- kreiranje i podešavanje baze podataka;
- kreiranje *REST API* nivoa.

Za razvoj backend strane projekta biće korišćeno razvojno okruženje *NetBeans IDE* u kombinaciji sa alatom za upravljanje zavisnostima Maven, Naziv projekta će biti *Lekcija13Primer*.



Slika 2.2 Kreirani Maven projekat [izvor: autor]

Kada se pogleda priložena slika, moguće je primetiti da se ističu neophodne zavisnosti za dalji razvoj projekta. Zavisnosti su se pojavile u projektu nakon kompletiranja datoteke *pom.xml* koja je priložena sledećim delimičnim listingom (prikazane su samo dodate zavisnosti).

```
<dependencies>
  <!-- Spring MVC zavisnosti -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.3.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.1.3.RELEASE</version>
  </dependency>
  <!-- Hibernate zavisnosti -->
```

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.4.0.Final</version>
</dependency>
<!-- Hibernate-C3P0 integracija -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-c3p0</artifactId>
  <version>5.4.0.Final</version>
</dependency>
<!-- c3p0 -->
<dependency>
  <groupId>com.mchange</groupId>
  <artifactId>c3p0</artifactId>
  <version>0.9.5.2</version>
</dependency>

<!-- MySQL zavisnosti -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>6.0.6</version>
</dependency>

<!-- Jackson API za JSON -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.8</version>
</dependency>
<!-- Servlet API zavisnosti -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
</dependency>
</dependencies>
```

C3p0 je biblioteka otvorenog koda za **JDBC** **pul konekcije**, sa podrškom za keširanje i višestruku upotrebu **PreparedStatement** upita.

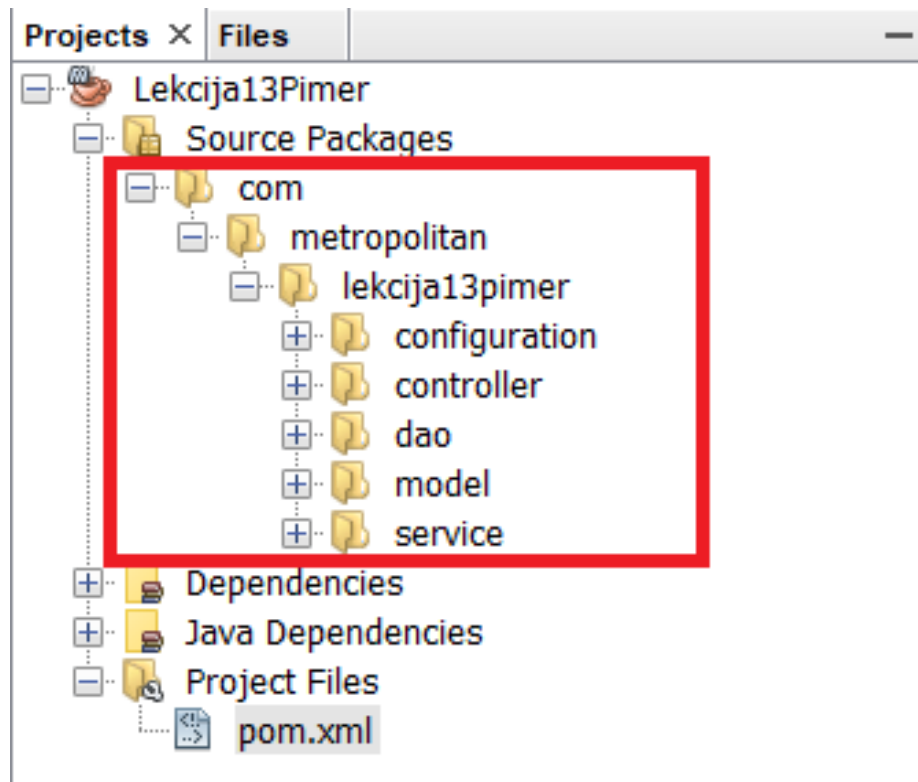
KREIRANJE JAVA PAKETA SPRING MAVEN PROJEKTA

Za početak biće kreirani odgovarajući paketi na lokaciji `src/main/java`.

Sledi dalja priprema za razvoj backend strane aplikacije koja se razvija kao prateći primer tekuće analize i diskusije. Za početak biće kreirani odgovarajući paketi na lokaciji `src/main/java`. Kreiranjem navedenih paketa, serverski deo aplikacije biće podeljen na nivoe:

- konfiguracije;
- modela;
- kontrolera;
- DAO - pristupa podacima;
- servisa.

Kreiranjem navedenih paketa, projekat dobija formu prikazanu sledećom slikom.



Slika 2.3 Paketi backend dela aplikacije [izvor: autor]

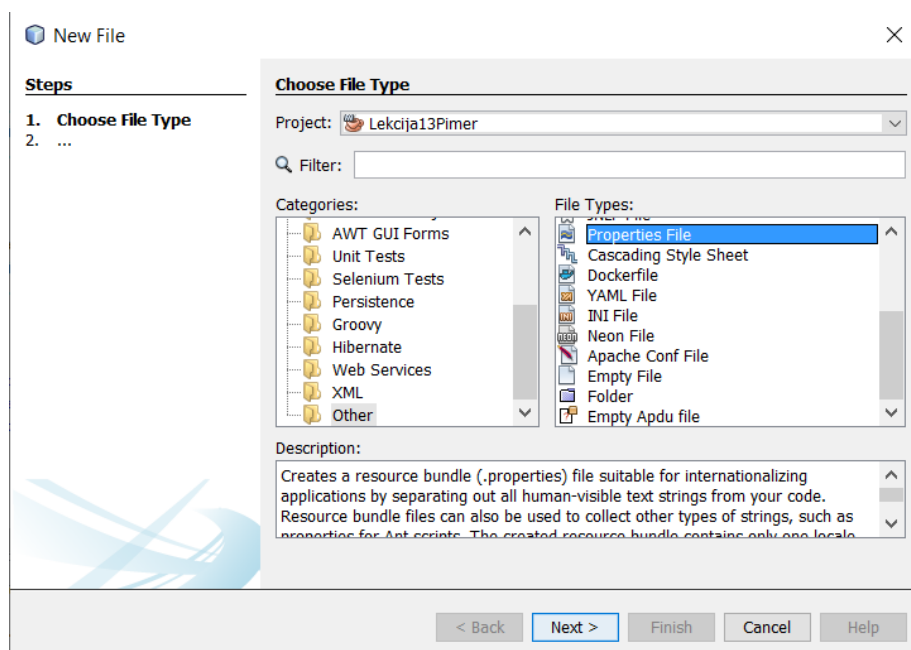
▼ Poglavlje 3

Kreiranje i podešavanje baze podataka

DATOTEKA ZA PODEŠAVANJE KONEKCIJE SA BAZOM PODATAKA

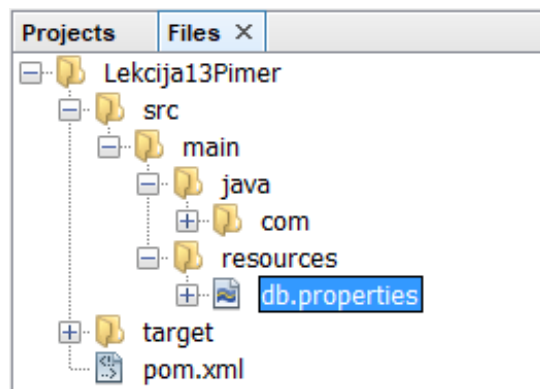
U folderu resursa kreira se datoteka za podešavanje konekcije sa bazom podataka.

Dobra preporuka je korišćenje datoteka osobina za fina podešavanja pristupa bazi podataka unutar Spring aplikacije. Na lokaciji `src/main/resources` biće kreirana datoteka pod nazivom `db.properties`, na način prikazan sledećom slikom.



Slika 3.1 Kreiranje datoteke db.properties iz NetBeans razvojnog okruženja [izvor: autor]

Važno je napomenuti da folder resources nije postojao i da je morao biti kreiran pre nego što je u njega ugrađena pomenuta datoteka. Sledećom slikom je prikazan navedeni obavljeni posao.



Slika 3.2 Kreirana datoteka sa osobinama za konekciju sa bazom podataka [izvor: autor]

Konačno, sledeće osobine su dodate za povezivanje aplikacije sa bazom podataka.

```
# PostgreSQL properties
db.driver=org.postgresql.Driver
db.url=jdbc:postgresql://localhost:5432/bookdb
db.username=postgres
db.password=vlada

# Hibernate properties
hibernate.show_sql=true
hibernate.hbm2ddl.auto=update
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

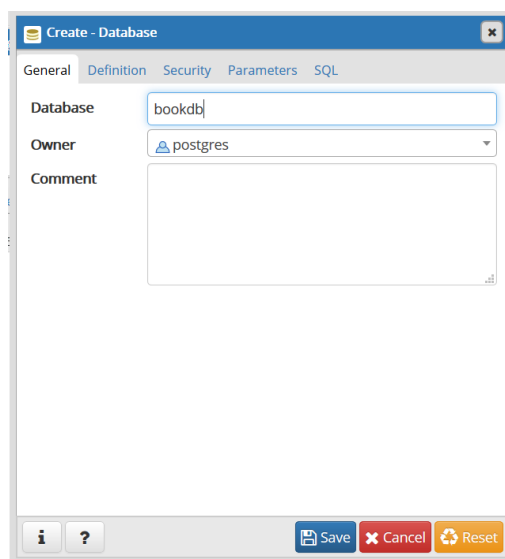
#C3P0 osobine
hibernate.c3p0.min_size=5
hibernate.c3p0.max_size=20
hibernate.c3p0.acquire_increment=1
hibernate.c3p0.timeout=1800
hibernate.c3p0.max_statements=150
```

KREIRANJE BAZE PODATAKA

Kreira se baza podataka bookdb primenom alata za upravljanje MySQL bazom podataka.

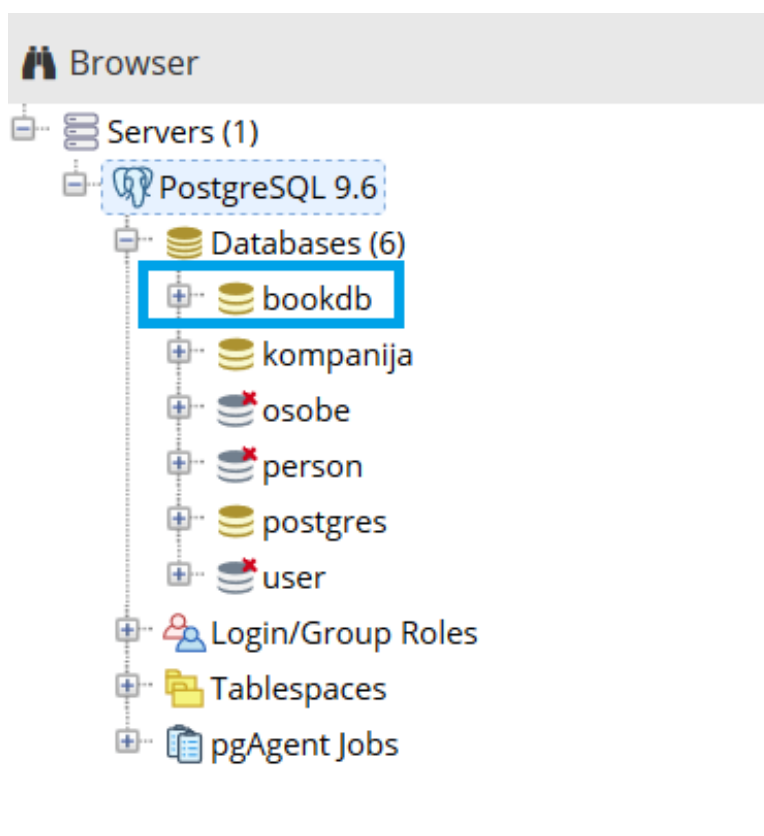
Ako se pogleda detaljnije datoteka `db.properties` moguće je primetiti da je predviđeno povezivanje na bazu podataka pod nazivom `bookdb`. Baza podataka je tipa `PostgreSQL`, a to znači da je neophodno bilo podesiti i odgovarajući drajver, kao i korisničko ime i lozinku za pristup serveru baze podataka.

Zbog svega navedenog, kreira se `baza podataka bookdb` primenom alata za upravljanje `PostgreSQL` bazom podataka (na primer `pgAdmin4`). Navedeno je prikazano slikom 3.



Slika 3.3 Kreiranje nove baze podataka [izvor: autor]

Sledećom slikom je prikazano da je prethodni zadatak uspešno obavljen i da se baza podataka nalazi na listi dostupnih i kreiranih baza podataka.



Slika 3.4 Kreirana baza podataka bookdb [izvor: autor]

HIBERNATE PODEŠAVANJA

Kreiranje tabela ove baze podataka biće prepušteno okviru Hibernate.

Bilo je neophodno kreirati samo bazu podataka *bookdb*. Kreiranje tabela ove baze podataka biće prepušteno okviru Hibernate budući da je u prethodnoj datoteci osobina to omogućeno kroz sledeći izolovani listing.

```
# Hibernate osobine  
hibernate.show_sql=true  
hibernate.hbm2ddl.auto=update
```

Slika 3.5 Hibernate podešavanja [izvor: autor]

Kada se kreiraju JEE aplikacije podrazumeva se da se one oslanjaju na bazu podataka u kojoj čuvaju podatke koje obrađuju. Kreiranje i uspostavljanje konekcije sa bazom podataka može biti složeno iz više razloga: uspostavljanje mrežne konekcije, inicijalizovanje sesije sa bazom podataka, autorizovanje pozadinskih servisa i tako dalje. Iz navedenih razloga, **dobru praksu predstavlja podešavanje i korišćenje pula konekcija**. Na taj način se povećavaju performanse i skalabilnost aplikacije. Navedeno je omogućeno podešavanjem osobina iz biblioteke *C3P0* kao na sledećoj slici.

```
#C3P0 osobine  
hibernate.c3p0.min_size=5  
hibernate.c3p0.max_size=20  
hibernate.c3p0.acquire_increment=1  
hibernate.c3p0.timeout=1800  
hibernate.c3p0.max_statements=150
```

Slika 3.6 Podešavanje i korišćenje pula konekcija [izvor: autor]

▼ Poglavlje 4

Kreiranje Java konfiguracija

APPCONFIG DATOTEKA

Neophodno je pristupiti konfigurisanju same Spring aplikacije.

Nakon kreiranja i podešavanja osobina za povezivanje sa bazom podataka, neophodno je pristupiti konfigurisanju same Spring aplikacije. Prva na redu će biti konfiguraciona datoteka *AppConfig.java* čiji je zadatak da:

- učitati datoteku osobina;
- omogućiti upravljanje transakcijama;
- omogućiti pretragu komponenata;

Da bi kreirana klasa mogla da obavlja navedene zadatke, neophodno je da bude obeležena odgovarajućim anotacijama, kao u sledećem listingu.

```
@Configuration
@PropertySource("classpath:db.properties")
@EnableTransactionManagement
@ComponentScans(value = {
    @ComponentScan("com.metropolitan.lekcija13pimer.dao")
    ,
    @ComponentScan("com.metropolitan.lekcija13pimer.service")})
public class AppConfig {}
```

U nastavku, u kreiranoj klasi, je neophodno izvršiti registrovanje zrna:

- *DataSource* - za preuzimanje parametara konekcije;
- *LocalSessionFactoryBean* - za kreiranje Hibernate sesije konekcije sa bazom podataka;
- *HibernateTransactionManager* - za upravljanje transakcijama.

Prva dva zrna koriste *Environment* zrno za upotrebu podešavanja iz fajla *db.properties* i njihovu implementaciju u sesiju konekcije sa bazom podataka.

```

@Autowired
private Environment env;

@Bean
public DataSource getDataSource() {
    BasicDataSource dataSource = new BasicDataSource();
    dataSource.setDriverClassName(env.getProperty("db.driver"));
    dataSource.setUrl(env.getProperty("db.url"));
    dataSource.setUsername(env.getProperty("db.username"));
    dataSource.setPassword(env.getProperty("db.password"));
    return dataSource;
}

@Bean
public LocalSessionFactoryBean getSessionFactory() {
    LocalSessionFactoryBean factoryBean = new LocalSessionFactoryBean();
    factoryBean.setDataSource(getDataSource());

    Properties props = new Properties();
    props.put("hibernate.show_sql", env.getProperty("hibernate.show_sql"));
    props.put("hibernate.hbm2ddl.auto", env.getProperty("hibernate.hbm2ddl.auto"));

    factoryBean.setHibernateProperties(props);
    factoryBean.setAnnotatedClasses(Book.class);
    return factoryBean;
}

```

umetanje Environment objekta

Slika 4.1 Registrovanje zrna LocalSessionFactoryBean [izvor: autor]

Konačno, registruje se i zrno *HibernateTransactionManager*.

```

@Bean
public HibernateTransactionManager getTransactionManager() {
    HibernateTransactionManager transactionManager = new HibernateTransactionManager();
    transactionManager.setSessionFactory(getSessionFactory().getObject());
    return transactionManager;
}

```

Slika 4.2 Registrovanje zrna HibernateTransactionManager [izvor: autor]

WEBCONFIG DATOTEKA

Kreiranje konfiguracione datoteke za funkcionisanje kreirane aplikacije kao Spring MVC aplikacije

Sledeći korak jeste kreiranje konfiguracione datoteke koja će omogućiti funkcionisanje kreirane aplikacije kao Spring MVC aplikacije:

- dodavanje anotacije koja omogućava *Spring MVC*;
- pronalaženje kontrolera;
- nasleđivanje klase *WebMvcConfigurerAdapter*.

Sledećim listingom su u potpunosti ilustrovane navedene funkcionalnosti.

```

package com.metropolitan.lekcija13pimer.configuration;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

/**
 *

```

```
* @author Vlada
*/
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = { "com.metropolitan.lekcija13pimer.controller" })
public class WebConfig extends WebMvcConfigurerAdapter {

}
```

DISPATCHER SERVLET

Neophodno je kreirati klasu koja podešava Dispatcher Servlet.

Konačno, neophodno je kreirati klasu koja podešava Dispatcher Servlet i omogućava učitavanje svih kreiranih konfiguracionih klasa. Klasa mora da nasledi klasu AbstractAnnotationConfigDispatcherServletInitializer i da implementira tri metode:

- getRootConfigClasses() - učitava AppConfig;
- getServletConfigClasses() - učitava WebConfig;
- getServletMappings() - podešava Servlet mapiranje.

```
package com.metropolitan.lekcija13pimer.configuration;

import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletIni
tializer;

/**
 *
 * @author Vlada
 */
public class MyWebAppInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { AppConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[] { WebConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

ZADATAK ZA SAMOSTALNI RAD 1

Pokušajte sami

Konfigurisanje Spring MVC aplikacije je šablonsko. Utvrdite još jednom svoje znanje i samostalno podesite aplikaciju ispočetka.

▼ Poglavlje 5

Kreiranje modela

MODELSKA KLASA BOOK

Kreiranje klase čiji će objekti biti čuvani u kreiranoj bazi podataka.

Modelska ili entitetska klasa opisuje objekte koji će biti čuvani u bazi podataka i nad kojima će biti izvođene CRUD operacije. U konkretnom slučaju radi se i *Book* objektima. Klasa mora da ispuni sledeće uslove:

- obeležena je anotacijom `@Entity` kojom se definiše veza između klase i odgovarajuće tabele iz baze podataka;
- klasa ima atribut koji je obeležen anotacijom `@Id`. Ovaj atribut odgovara primarnom ključu u odgovarajućoj tabeli.

Koristeći ovu klasu, *Hibernate* okvir će automatski kreirati tabelu u bazi podataka.

```
package com.metropolitan.lekcija13pimer.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

/**
 *
 * @author Vlada
 */
@Entity(name = "Book")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;

    public Long getId() {
        return id;
    }

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

```
public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public String getAuthor() {
    return author;
}

public void setAuthor(String author) {
    this.author = author;
}

@Override
public String toString() {
    return "Book{" + "id=" + id + ", title=" + title + ", author=" + author +
    '}';
}
}
```

PRESLIKAVANJE ENTITETA U TABELU

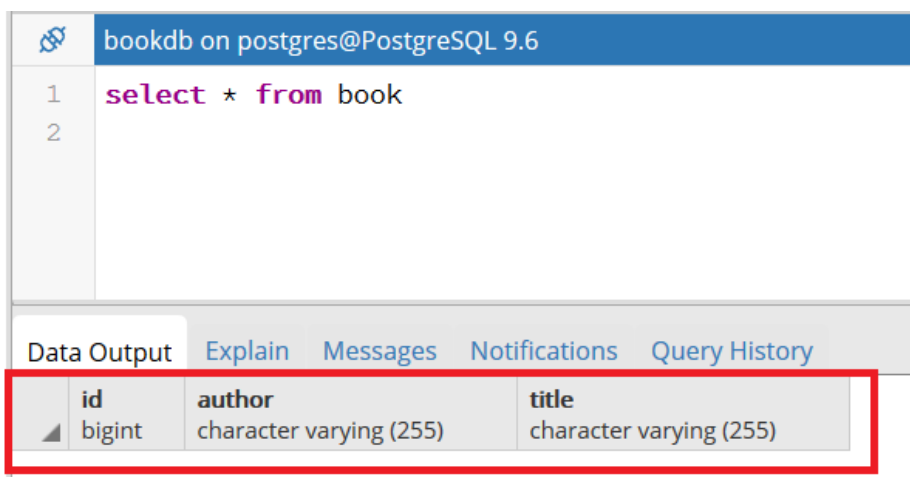
Hibernate okvir će automatski kreirati tabelu u bazi podataka.

Koristeći kreiranu entitetsku klasu i *Hibernate* podešavanja, *Hibernate* okvir će automatski kreirati tabelu u bazi podataka. Dovoljno je da trenutna aplikacija bude prevedena i da bude pokrenuta. Pokreće se aplikativni server *Apache TomCat* i u *Log*-u se brzo dobija informacija da je kreirana tabela *book* u bazi podataka *bookdb*. Navedeno je prikazano sledećom slikom.

```
Hibernate: create table Book (id bigserial not null, author varchar(255) not null, title varchar(255) not null)
19-Dec-2018 19:05:41.591 INFO [http-nio-8084-exec-74] org.springframework.orm.hibernate4.HibernateUtils: Hibernate: create table Book (id bigserial not null, author varchar(255) not null, title varchar(255) not null)
19-Dec-2018 19:05:41.676 INFO [http-nio-8084-exec-74] org.springframework.orm.hibernate4.HibernateUtils: Hibernate: create table Book (id bigserial not null, author varchar(255) not null, title varchar(255) not null)
19-Dec-2018 19:05:41.676 INFO [http-nio-8084-exec-74] org.springframework.orm.hibernate4.HibernateUtils: Hibernate: create table Book (id bigserial not null, author varchar(255) not null, title varchar(255) not null)
19-Dec-2018 19:05:41.676 INFO [http-nio-8084-exec-74] org.springframework.orm.hibernate4.HibernateUtils: Hibernate: create table Book (id bigserial not null, author varchar(255) not null, title varchar(255) not null)
```

Slika 5.1 Trenutak kreiranja tabele Book [izvor: autor]

Da je Hibernate okvir zaista odradio svoj posao moguće je proveriti i u *pgAdmin4* veb konzoli za upravljanje *PostgreSQL* bazama podataka. Navedeno je prikazano sledećom slikom.



The screenshot shows a PostgreSQL query window titled "bookdb on postgres@PostgreSQL 9.6". The query editor contains the SQL statement `select * from book`. Below the query editor, there are tabs for "Data Output", "Explain", "Messages", "Notifications", and "Query History". The "Data Output" tab is selected, and it displays the schema of the "book" table. The schema is shown in a table with three columns: "id" (type "bigint"), "author" (type "character varying (255)"), and "title" (type "character varying (255)"). A red rectangle highlights the table schema output.

	id	author	title
	bigint	character varying (255)	character varying (255)

Slika 5.2 Kreirana tabela book u PostgreSQL bati podataka [izvor: autor]

▼ Poglavlje 6

Kreiranje DAO i servisnog nivoa

KREIRANJE DAO INTERFEJSA I IMPLEMENTACIJE

Implementacija DAO operacija za izvođenje zadataka perzistencije nad entitetskim objektima.

Sledeći zadatak koji se postavlja pred backend programera jeste kreiranje i implementacija DAO operacija za izvođenje zadataka perzistencije nad entitetskim objektima.

U prvom koraku definišu se navedene operacije u odgovarajućem DAO interfejsu na sledeći način:

```
package com.metropolitan.lekcija13primer.dao;

import com.metropolitan.lekcija13primer.model.Book;
import java.util.List;

/**
 *
 * @author Vlada
 */
public interface BookDao {

    long save(Book book);

    Book get(long id);

    List<Book> list();

    void update(long id, Book book);

    void delete(long id);

}
```

Sada je neophodno dati konkretnu implementaciju za kreirane metode kroz odgovarajuću implementacionu klasu čiji listing sledi. Klasa koristi Hibernate SessionFactory zrno za uspostavljanje sesije za bazom podataka i izvršavanje CRUD operacija. Takođe, obeležena je anotacijom @Repository da bi Spring kontejner mogao da je pronade kroz mehanizam skeniranja komponentata.

```
package com.metropolitan.lekcija13primer.dao;

import com.metropolitan.lekcija13primer.model.Book;
```

```
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

/**
 *
 * @author Vlada
 */
@Repository
public class BookDaoImp implements BookDao {

    @Autowired
    private SessionFactory sessionFactory;

    @Override
    public long save(Book book) {
        sessionFactory.getCurrentSession().save(book);
        return book.getId();
    }

    @Override
    public Book get(long id) {
        return sessionFactory.getCurrentSession().get(Book.class, id);
    }

    @Override
    public List<Book> list() {
        List<Book> list = sessionFactory.getCurrentSession().createQuery("from
Book").list();
        return list;
    }

    @Override
    public void update(long id, Book book) {
        Session session = sessionFactory.getCurrentSession();
        Book book2 = session.byId(Book.class).load(id);
        book2.setTitle(book.getTitle());
        book2.setAuthor(book.getAuthor());
        session.flush();
    }

    @Override
    public void delete(long id) {
        Book book = sessionFactory.getCurrentSession().byId(Book.class).load(id);
        sessionFactory.getCurrentSession().delete(book);
    }
}
```

KREIRANJE SERVISNOG INTERFEJSA I IMPLEMENTACIJE

Nakon kreiranja DAO nivoa backend dela veb aplikacije, sledi kreiranje njenog servisnog sloja.

Nakon kreiranja *DAO* nivoa backend dela veb aplikacije, sledi kreiranje njenog servisnog sloja. Servisne metode su definisane sledećim interfejsom.

```
package com.metropolitan.lekcija13primer.service;

import com.metropolitan.lekcija13primer.model.Book;
import java.util.List;

/**
 *
 * @author Vlada
 */
public interface BookService {

    long save(Book book);

    Book get(long id);

    List<Book> list();

    void update(long id, Book book);

    void delete(long id);
}
```

Konačno, kreirana je i implementaciona klasa servisnog nivoa aplikacije. Zadatak klase je da obezbedi transakciono izvršavanje kreiranih metoda. Takođe, klasa je obeležena anotacijom *@Service* da bi mogla da bude pronađena putem mehanizma skeniranja komponenata.

```
package com.metropolitan.lekcija13primer.service;

import com.metropolitan.lekcija13primer.dao.BookDao;
import com.metropolitan.lekcija13primer.model.Book;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

/**
 *
 * @author Vlada
 */
@Service
@Transactional(readOnly = true)
```

```
public class BookServiceImpl implements BookService {

    @Autowired
    private BookDao bookDao;

    @Transactional
    @Override
    public long save(Book book) {
        return bookDao.save(book);
    }

    @Override
    public Book get(long id) {
        return bookDao.get(id);
    }

    @Override
    public List<Book> list() {
        return bookDao.list();
    }

    @Transactional
    @Override
    public void update(long id, Book book) {
        bookDao.update(id, book);
    }

    @Transactional
    @Override
    public void delete(long id) {
        bookDao.delete(id);
    }

    @Autowired(required = true)
    public void setBookDao(BookDao bookDao) {
        this.bookDao = bookDao;
    }
}
```

▼ Poglavlje 7

REST kontroler

KREIRANJE REST KONTROLERA

Kreiranje REST kontrolera je poslednji korak u kreiranju REST API.

Da bi backend razvoj bio uspešno završen neophodno je kreirati REST kontroler koji će manipulirati servisnim metodama razvijenim u prethodnom izlaganju.

Kontroler će biti obeležen anotacijom `@RestController` kao i opštom anotacijom za mapiranje `@RequestMapping`. Njegove metode rukovaoci biće obeležene odgovarajućim anotacijama u zavisnosti od tipa HTTP zahteva kojeg obrađuju.

Listing kontrolera sa detaljnim opisima zaduženja metoda rukovalaca priložen je sledećim listingom.

```
@CrossOrigin(origins = "*")
@RestController
@RequestMapping("/api")
public class BookController {

    @Autowired
    private BookService bookService;

    /*---Dodaje novu knjigu---*/
    @PostMapping("/book")
    public ResponseEntity<?> save(@RequestBody Book book) {
        long id = bookService.save(book);
        return ResponseEntity.ok().body("Nova knjiga je snimljena pod ID:" + id);
    }

    /*---Vraća knjigu po id---*/
    @GetMapping("/book/{id}")
    public ResponseEntity<Book> get(@PathVariable("id") long id) {
        Book book = bookService.get(id);
        return ResponseEntity.ok().body(book);
    }

    /*---vraća sve knjige---*/
    @GetMapping("/book")
    public ResponseEntity<List<Book>> list() {
        List<Book> books = bookService.list();
        return ResponseEntity.ok().body(books);
    }
}
```



```
/*---Ažurira knjigu po id---*/
@PutMapping("/book/{id}")
public ResponseEntity<?> update(@PathVariable("id") long id, @RequestBody Book
book) {
    bookService.update(id, book);
    return ResponseEntity.ok().body("Knjiga je uspesno azurirana.");
}

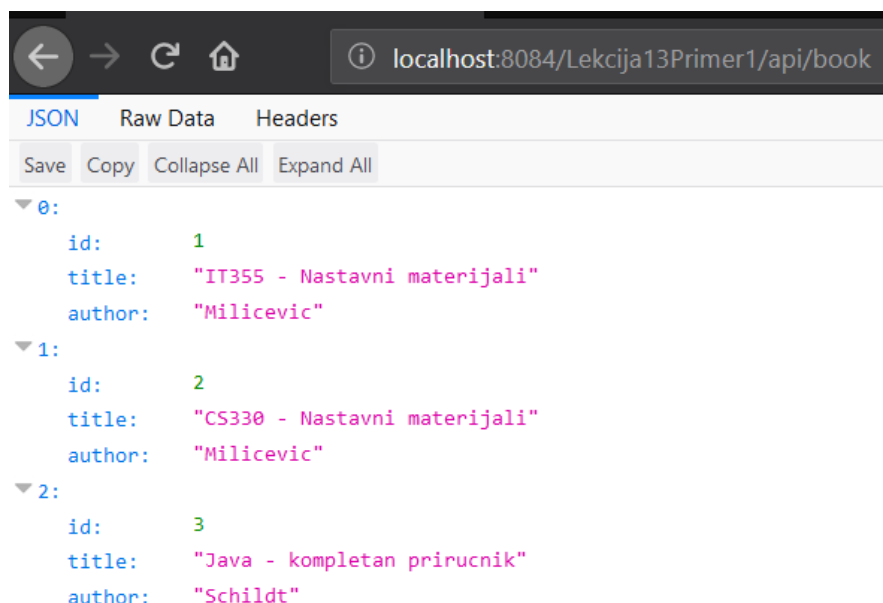
/*---Briše knjigu po id---*/
@DeleteMapping("/book/{id}")
public ResponseEntity<?> delete(@PathVariable("id") long id) {
    bookService.delete(id);
    return ResponseEntity.ok().body("Knjiga je uspesno izbrisana.");
}
}
```

TESTIRANJE BACKEND FUNKCIONALNOSTI - GET ZAHTEVI

Testiranje poziva i funkcionalnosti kontrolera

Kreiranjem *REST* kontrolera završena je definicija backend strane aplikacije koja je predmet analize i diskusije. U daljem radu je neophodno izvršiti prevođenje aplikacije i testirati pozive ka *REST* kontroleru sa ciljem provere ispravnosti njegovog funkcionisanja. Za početak biće testirana *GET* metoda za prikazivanje svih Book objekata sačuvanih u bazi podataka.

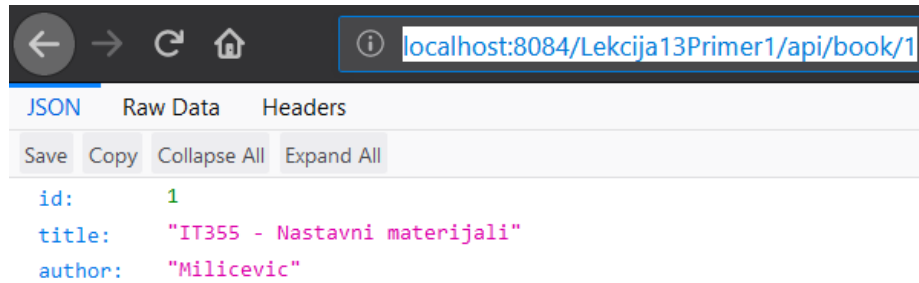
<http://localhost:8084/Lekcija13Primer1/api/book>



Slika 7.1 Prikazivanje svih Book objekata sačuvanih u bazi podataka. [izvor: autor]

Dalje, moguće je proveriti i *GET* zahtev za vraćanje pojedinačnih *Book* objekata.

```
http://localhost:8084/Lekcija13Primer1/api/book/1
```



Slika 7.2 Prikazivanje pojedinačnih Book objekata sačuvanih u bazi podataka. [izvor: autor]

Testiranje GET zahteva je bilo uspešno i kreirani kontroler ove operacije izvodi na način kako je to i očekivano.

TESTIRANJE BACKEND FUNKCIONALNOSTI - DELETE ZAHTEV

Provera da li kontroler briše podatke iz baze podataka

Za zahteve koji menjaju stanje baze podataka: *POST*, *PUT* i *DELETE* moguće je koristiti neki od alata za proveru *REST* servisa poput: *Postman* ili *RestClient* za veb pregledač *Firefox* (oba će biti korišćeni je u ovom slučaju).

Za početak kreiran je *DELETE* zahtev za brisanje knjige iz baze podataka čiji je *ID* jednak 2.

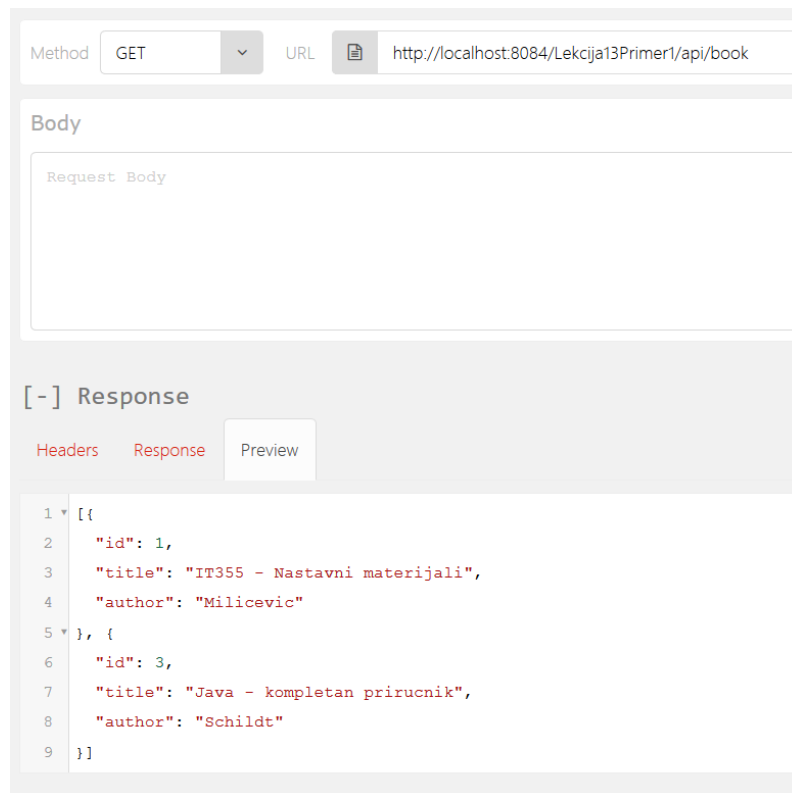


Slika 7.3 DELETE zahtev za brisanje knjige iz baze podataka [izvor: autor]

Iz slike je moguće videti da je obavljen poziv metode kontrolera putem linka:

```
http://localhost:8084/Lekcija13Primer1/api/book/2
```

Nakon obavljene metode brisanja iz baze podataka, neophodno je ponoviti *GET* zahtev za proveru da li stvarno traženi objekat izbrisan iz baze podataka:



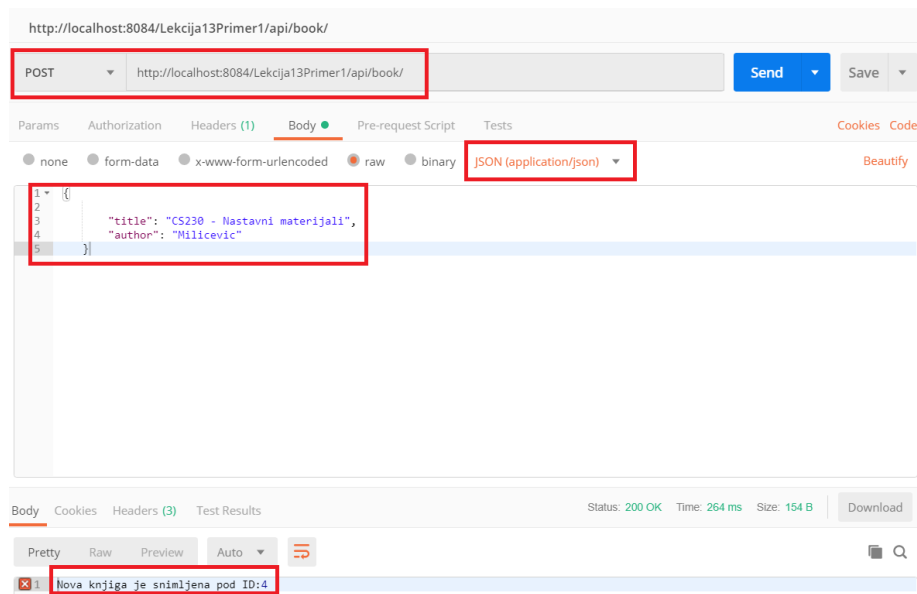
Slika 7.4 Provera zahteva za brisanje knjige iz baze podataka [izvor: autor]

TESTIRANJE BACKEND FUNKCIONALNOSTI - POST ZAHTEV

Provera da li kontroler dodaje nove objekte u bazu podataka

Sada je moguće testirati *POST* zahtev ka kontroleru sa ciljem provere da li ispravno funkcioniše dodavanje novih *Book* objekata u bazu podataka. *POST* zahtev je praćen pozivom:

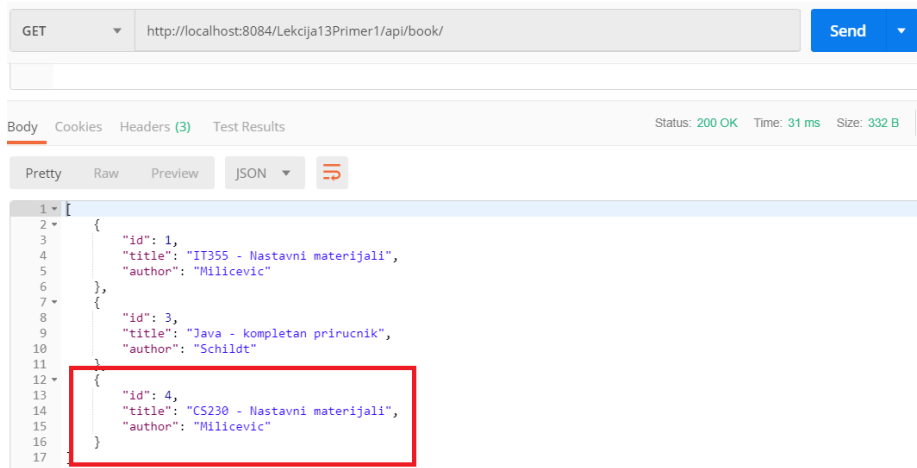
```
http://localhost:8084/Lekcija13Primer1/api/book/
```



Slika 7.5 Zahtev za dodavanjem novog objekta u bazu podataka [izvor: autor]

Nakon obavljene metode dodavanja objekata u bazu podataka, neophodno je ponoviti *GET* zahtev za proveru da li stvarno traženi objekat dodat u bazu podataka.

`http://localhost:8084/Lekcija13Primer1/api/book/`



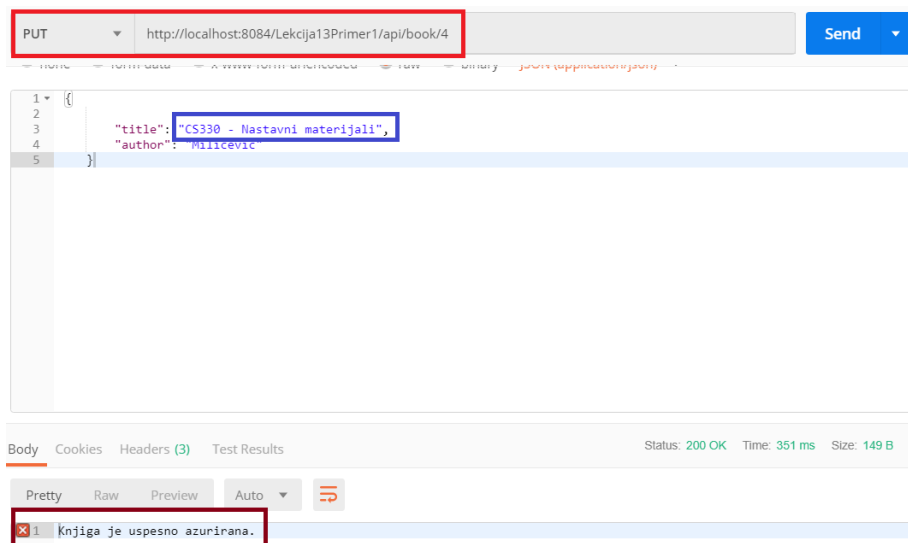
Slika 7.6 Provera dodavanja novog objekta u bazu podataka [izvor: autor]

TESTIRANJE BACKEND FUNKCIONALNOSTI - PUT ZAHTEV

Provera da li kontroler ažurira objekte u bazi podataka

Sada je moguće testirati *PUT* zahtev ka kontroleru sa ciljem provere da li ispravno funkcioniše ažuriranje *Book* objekata u bazi podataka. *PUT* zahtev je praćen pozivom:

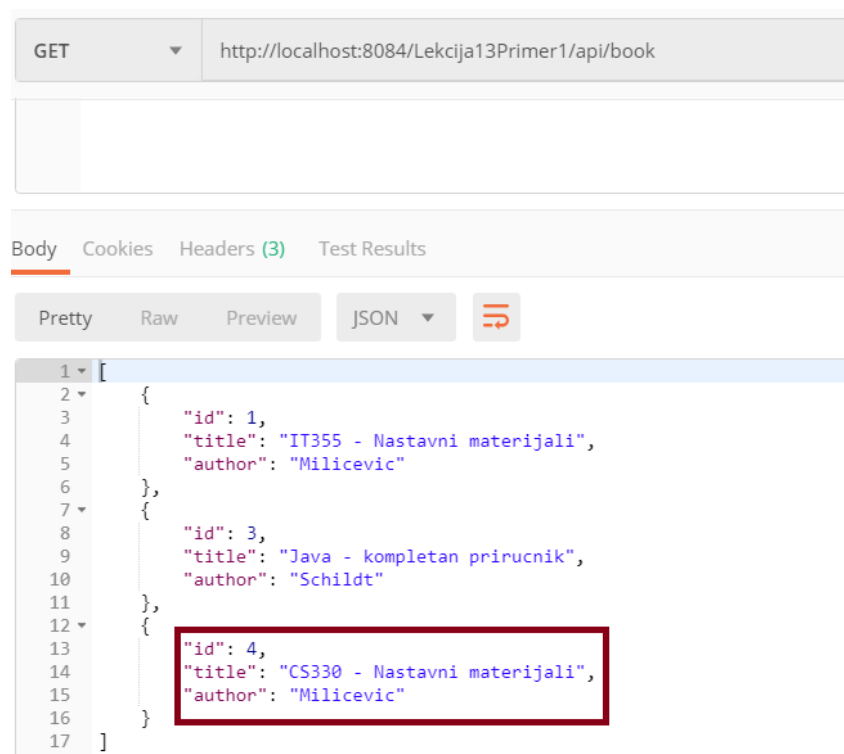
`http://localhost:8084/Lekcija13Primer1/api/book/4`



Slika 7.7 Zahtev za ažuriranjem objekta iz baze podataka [izvor: autor]

Nakon obavljene metode ažuriranja objekata iz baze podataka, neophodno je ponoviti *GET* zahtev za proveru da li stvarno traženi objekat ažuriran.

http://localhost:8084/Lekcija13Primer1/api/book/

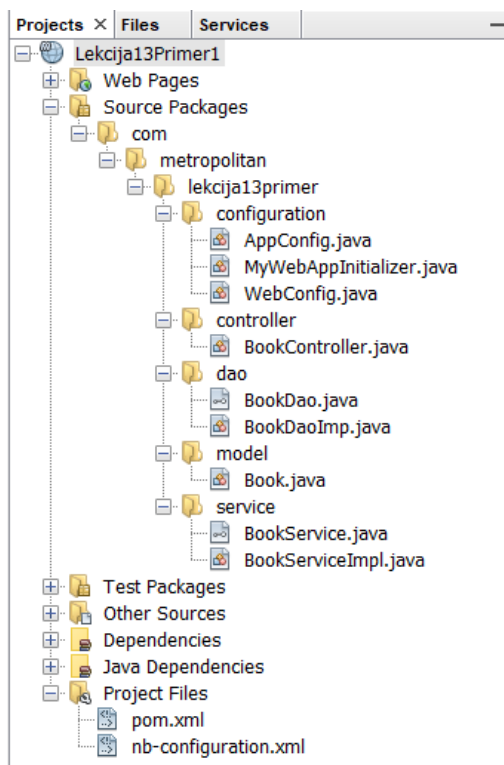


Slika 7.8 Provera ažuriranja podataka [izvor: autor]

KONAČNA STRUKTURA BACKEND PROJEKTA

Provere su pokazale ispravnost kreirane backend aplikacije.

Provere su pokazale ispravnost kreirane *backend* aplikacije. Sve kontrolerove metode za vršenje operacija perzistencije su ispravne i funkcionalne tako da u ovom delu izlaganja može biti priložena konačna struktura ovog projekta.



Slika 7.9 Konačna struktura backend projekta [izvor: autor]

Moguće je izvršiti još jednu proveru. Da li su sve obavljene operacije zaista primenjene na kreiranu bazu podataka? Provera se vrši u *pgAdmin4* konzoli za upravljanje bazom podataka i prikazana je sledećom slikom.

Dashboard Properties SQL Statistics Dependencies Depend

bookdb on postgres@PostgreSQL 9.6

1 `select * from book`

Data Output Explain Messages Notifications Query History

	id bigint	author character varying (255)	title character varying (255)
1	1	Milicevic	IT355 - Nastavni materijali
2	3	Schildt	Java - kompletan prirucnik
3	4	Milicevic	CS330 - Nastavni materijali

Slika 7.10 Provera izmena u bazi podataka [izvor: autor]

Testiran i ispravan projekat možete preuzeti nakon ovog objekta učenja.

VIDEO MATERIJAL

Spring Boot: Making a REST API - 27:12

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 8

Razvoj Angular klijentske komponente

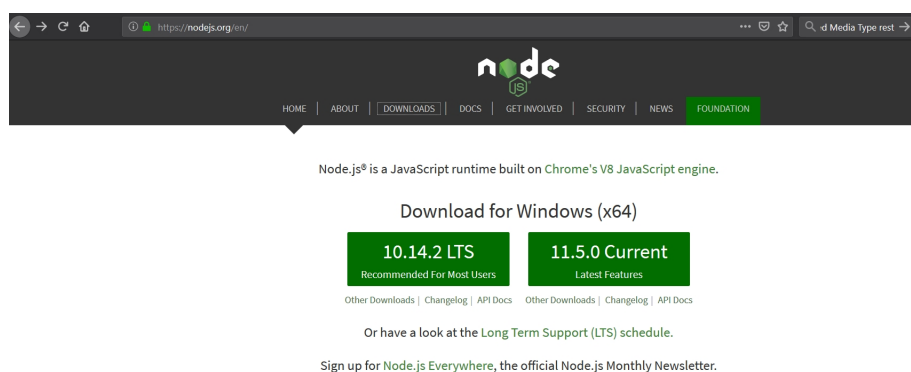
FRONTEND ZAHTEVI

Za razvoj frontend strane biće korišćen radni okvir Angular.

Nakon kreirane *backend* strane veb aplikacije pristupa se analizi i diskusiji koja je u direktnoj vezi sa kreiranjem *frontend* dela aplikacije. U tu svrhu biće korišćen radni okvir *Angular* tako da će u prvom delu ovog izlaganja biti govora o neophodnim alatima za njegovu primenu.

Ovaj radni okvir studenti su savladali na predmetu *IT255 - Veb sistemi 1*, tako da ova lekcija predstavlja zaokruživanje znanja stečenog na ovom i predmetu *IT255*.

Prvi korak koji je neophodno preduzeti jeste preuzimanje i instalacija alata *NodeJS* koji će biti korišćen za razvoj *frontend Angular* aplikacije. Aplikacija *NodeJS* je dostupna za preuzimanje na linku <https://nodejs.org/en/>, a odatle je potrebno preuzeti verziju koja odgovara operativnom sistemu na kojem će biti instalirana.



Slika 8.1.1 Izbor verzije za NodeJS [izvor: autor]

Nakon instalacije, neophodno je proveriti da li je *NodeJS* ispravno instaliran na računaru. Ovo je moguće proveriti u *Command Prompt* aplikaciji kucanjem sledeće naredbe na komandnoj liniji:

```
node -v
```

Ukoliko je rezultat izvršavanja naredbe kao na sledećoj slici, sve je u redu i spremno za dalji rad.



Slika 8.1.2 Uspešno instaliran NodeJS [izvor: autor]

ANGULAR CLI

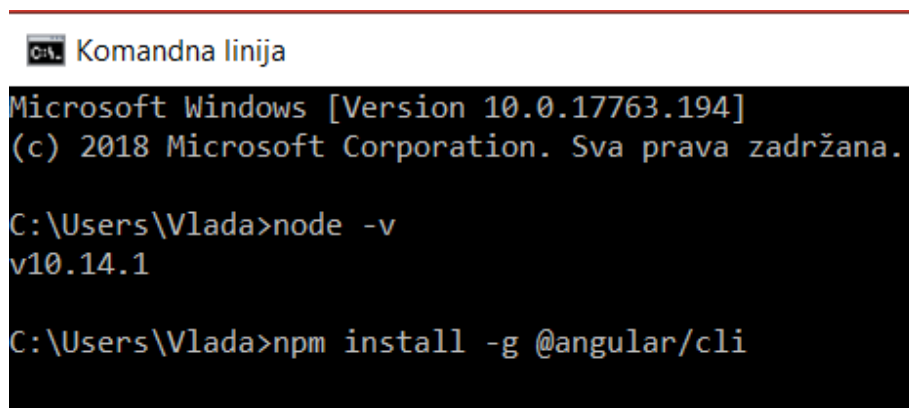
Angular CLI je sledeći neophodni alat za razvoj frontend dela aplikacije

Za nastavak rada na razvoju frontend dela aplikacije biće neophodan i alat [Angular CLI](#) (*Command Line Interface*) kojeg je moguće instalirati nakon što je obavljena prethodna *NodeJS* instalacija.

U *Command Prompt* aplikaciji, na komandnoj liniji, sada je neophodno kucati sledeću naredbu:

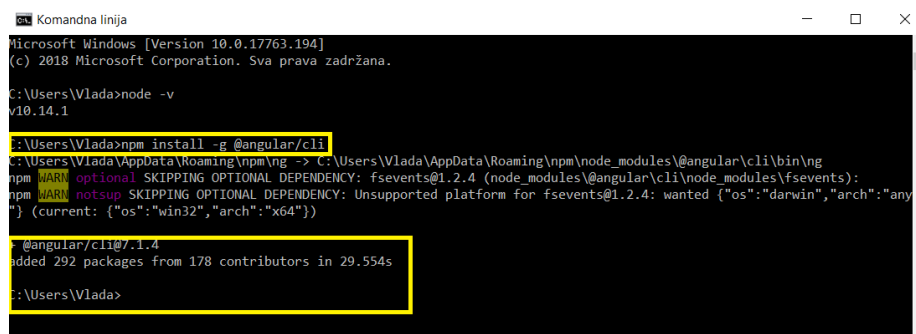
```
npm install -g @angular/cli
```

Navedeno je prikazano sledećom slikom:



Slika 8.1.3 Naredba za Angular CLI instalaciju [izvor: autor]

Pokretanjem navedene naredbe pokreće se instalacija alata *Angular CLI* i veoma brzo će biti uspešno završena. Navedeno je prikazano sledećom slikom:



```
Komandna linija
Microsoft Windows [Version 10.0.17763.194]
(c) 2018 Microsoft Corporation. Sva prava zadržana.

C:\Users\Vlada>node -v
v10.14.1

C:\Users\Vlada>npm install -g @angular/cli
C:\Users\Vlada\AppData\Roaming\npm\ng -> C:\Users\Vlada\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\@angular\cli\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ @angular/cli@7.1.4
added 292 packages from 178 contributors in 29.554s

C:\Users\Vlada>
```

Slika 8.1.4 Instaliran alat Angular CLI [izvor: autor]

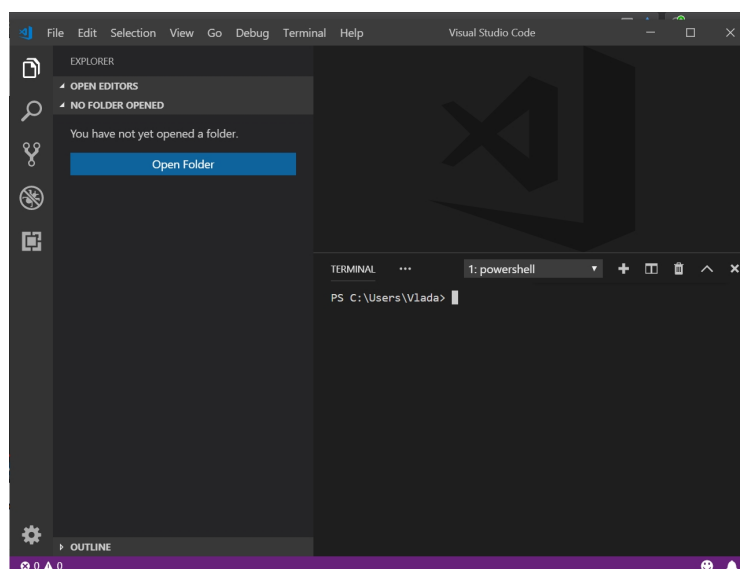
8.1 Kreiranje Angular projekta

ANGULAR FRONTEND PROJEKAT

Nakon instaliranja Angular CLI neophodno je izabrati editor za pisanje frontend programa.

Nakon instaliranja Angular CLI neophodno je izabrati editor za pisanje frontend programa, a to može biti Visual Studio Code. Ovaj alat je moguće preuzeti sa linka <https://code.visualstudio.com/>, a potom i jednostavno instalirati.

Nakon instalacije, pristupa se Visual Studio Code Terminalu izborom tastera ctrl + ~ (taster ispod ESC). Navedeno je prikazano sledećom slikom.



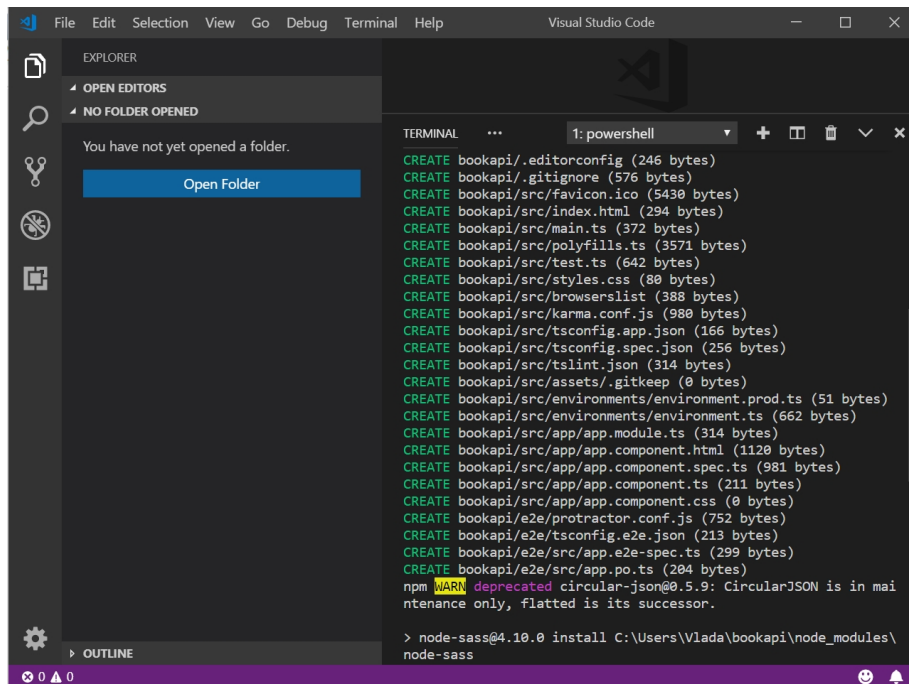
Slika 8.2.1 Visual Studio Code Terminal [izvor: autor]

ng new bookapi

Sada je moguće kreirati novi Angular projekat. U otvorenom terminalu je neophodno otkucati sledeću naredbu:

```
ng new bookapi
```

Izvršavanjem ove naredbe kreira se inicijalni *Angular* frontend projekat.



Slika 8.2.2 Kreiran Angular projekat [izvor: autor]

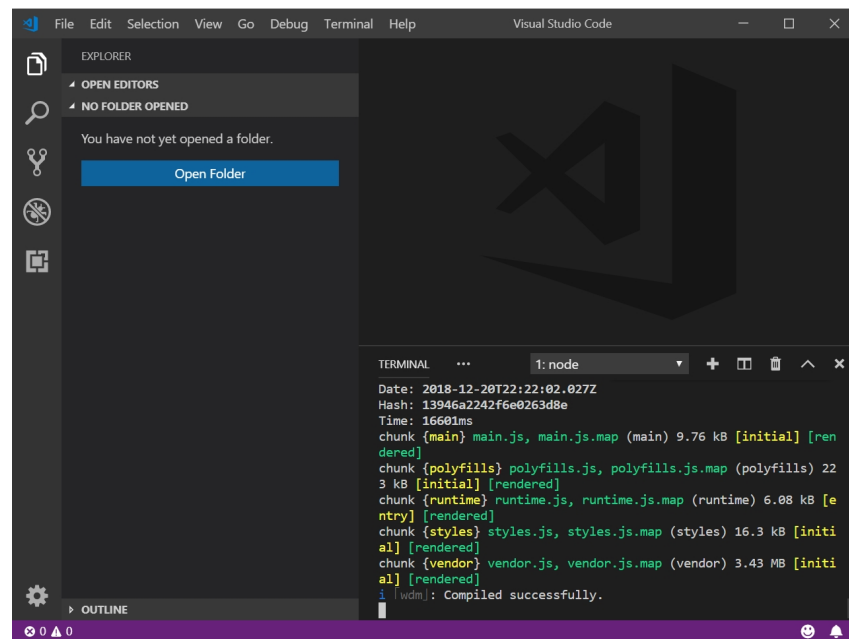
KREIRANJE DIREKTORIJUMA PROJEKTA

Dodatna podešavanja Angula projekta.

Nakon što je kreiran projekat, potrebno je izvršiti sledeće naredbe u terminalu:

```
cd bookapi  
  
ng serve --open
```

Izvršavanjem ovih naredbi u terminalu je dobijena informacija da je priprema za rad na frontend projektu uspešno obavljena.

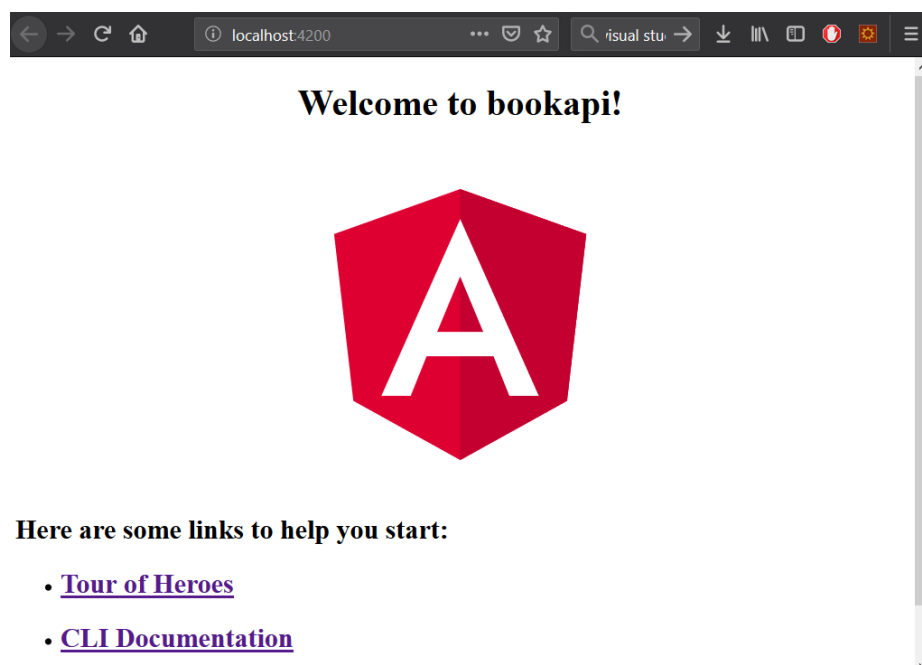


Slika 8.2.3 Priprema za razvoj frontend aplikacije [izvor: autor]

Sada je moguće otići u veb pregledač i na linku:

`http://localhost:4200/`

moguće je pristupiti stranici dobrodošlice za razvoj frontend aplikacije *bookapi*.

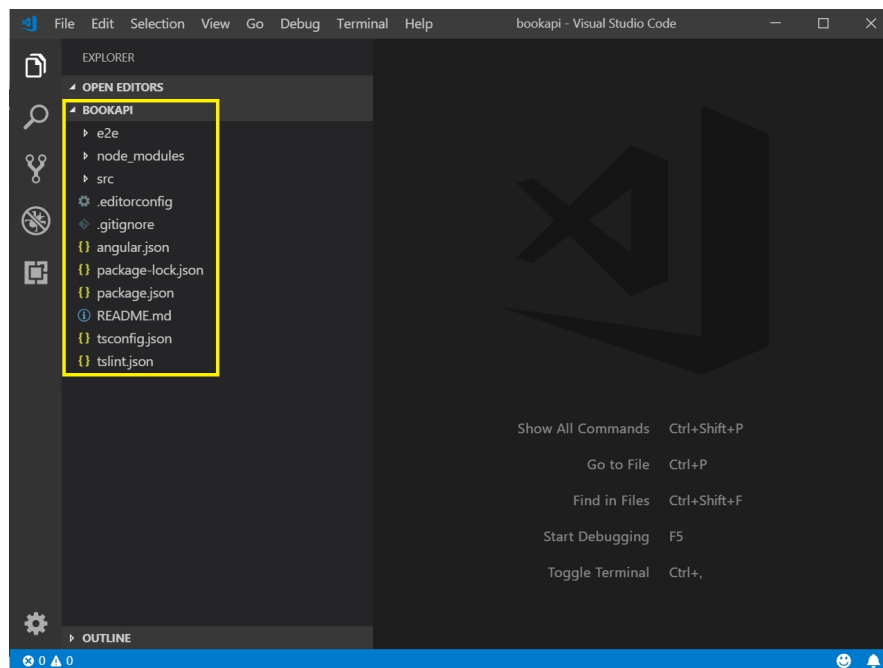


Slika 8.2.4 Dobrodošlica za razvoj frontend aplikacije bookapi. [izvor: autor]

INICIJALNA STRUKTURA KREIRANOG ANGULAR PROJEKTA

*Nakon obavljenih inicijalnih podešavanja, dostupna je početna struktura Angular projekta **bookapi***

Nakon obavljenih inicijalnih podešavanja, sada je dostupna početna struktura **Angular** projekta **bookapi**. Navedeno je prikazano sledećom slikom.

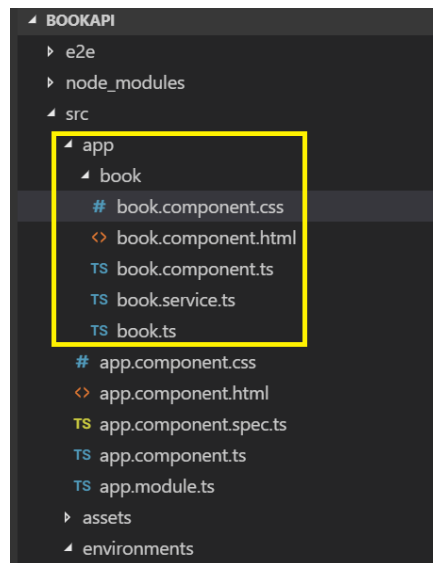


Slika 8.2.5 Početna struktura Angular projekta bookapi [izvor: autor]

Za razvoj **Angular** aplikacije je od posebnog značaj lokacija **src/app** u kreiranoj strukturi projekta. Ovde će, u nastavku, biti kreiran podfolder **book** koji će da čuva pet različitih programskih datoteka:

- **book.ts** - modelska klasa;
- **book.component.ts** - klasa koja vrši ulogu sličnu kontroleru;
- **book.service.ts** - klasa koja sadrži poslovnu logiku, http pozive i manipulaciju krajnjim tačkama REST servisa;
- **book.component.html** - datoteka koja definiše UI;
- **book.component.css** - opcioni fajl sa stilovima.

Sledećom slikom je prikazana lokacija u projektu na kojoj će biti kreirane navedene programske datoteke.



Slika 8.2.6 Folder programskih datoteka [izvor: autor]

▼ 8.2 Programske datoteka Angular projekta

MODELSKA KLASA - BOOK.TS

Prvi korak je kreiranje odgovarajuće modelske klase.

Prvi korak je kreiranje odgovarajuće modelskeklase book sa dobro poznatim osobinama:

- *id*;
- *title*;
- *author*.

Sve osobine će biti tipa *string*, a kao dodatak biće kreiran i konstruktor klase.

Klasa *book.ts* je definisana sledećim listingom.

```
export class Book{
  id: string;
  title: string;
  author: string;
  constructor(){

  }
}
```

DATOTEKA BOOK.COMPONENT.TS

*Sledeća klasa je veoma važna i njena uloga je slična ulozi koju obavlja **Spring** kontroler.*

Sledeća klasa je veoma važna i njena uloga je slična ulozi koju obavlja **Spring** kontroler. Klasa je dobila naziv **book.component.ts** i njena definicija započinje dodavanjem nekoliko **import** poziva za **Angular** biblioteke i klase koje su kreirane za ovaj **klijent** projekat.

```
import {Component, OnInit} from '@angular/core';
import {Router} from '@angular/router';
import {BookService} from './book.service';
import {Book} from './book';
```

U nastavku je ovu klasu neophodno obeležiti anotacijom **@Component** da bi bilo istaknuto da je ova klasa **komponenta**.

```
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
```

Iz priloženog prvog listinga moguće je primetiti da ova klasa zahteva funkcionalnosti definisane u datoteci **book.service**. Kada ove funkcionalnosti budu dostupne biće dovršena i definicija klase **book.component.ts**. Sada je neophodno priložiti njen trenutni izgled.

```
import {Component, OnInit} from '@angular/core';
import {Router} from '@angular/router';
import {BookService} from './book.service';
import {Book} from './book';

@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent {
}
```

REDEFINISANJE DATOTEKA APP.MODULE.TS I APP.COMPONENT.HTML

Redefinisanje podrazumevanih datoteka klijent aplikacije

Da bi Angular klijent aplikacija funkcionisala na željeni način, neophodno je izvršiti redefinisane datoteka koje su automatski generisane kada je kreiran **Angular** projekat.

Prva datoteka koja će biti redefinisana je `app.component.html`. Izvršavanjem ove datoteke prikazuje se dobro poznata stranica sa *Angular* logoom. Neophodno je obrisati u potpunosti njen sadržaj i umesto njega dodati selektor koji je naveden u `@Component` anotaciji prethodno kreirane klase `book.component.ts`. Navedeno je realizovano sledećim kodom:

```
<app-book></app-book>
```

Angular okviru još uvek nije saopšteno da se koristi prikazani selektor. Ovo je moguće učiniti redefinisanjem datoteke `app.module.ts`. U ovom *Angular* korenskom modulu moguće je registrovati selektora na sledeći način.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';

@NgModule({
  declarations: [
    AppComponent, BookComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

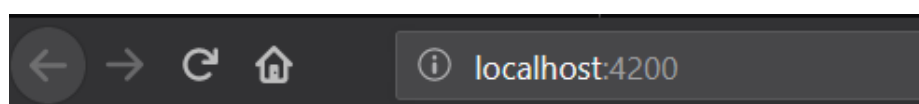
Slika 8.3.1 Redefinisanje app.module.ts [izvor: autor]

Celokupan dosadašnji rad predstavlja uvod u kreiranje ozbiljnih funkcionalnosti Angular klijent aplikacije. Još uvek nije uspostavljena interakcija sa prethodno kreiranom *backend* aplikacijom.

Za proveru da li je dosadašnji posao obavljen na pravi način, u datoteku `book.component.html` biće dodat sledeći kod:

```
<h1>Dobrodošli u BookApi</h1>
```

Pokretanjem aplikacije na `http://localhost:4200/` dobija se izlaz kao na sledećoj slici:



Dobrodošli u BookApi

Slika 8.3.2 Izvršavanje Angular aplikacije [izvor: autor]

Budući da je sve proteklo u najboljem redu, moguće je dalje pristupiti razvoju traženih funkcionalnosti ove klijent *Angular* aplikacije. Ova datoteka će posebno biti proširena u nastavku.

▼ 8.3 Servisi klijenta

KREIRANJE SERVISA KLIJENTA

Kreiranje servisne klase sa metodama koje odgovaraju zahtevima ka kreiranim REST servisima

U prethodnom izlaganju su kreirane modelska i komponentska klasa i u sledećem koraku je neophodno kreirati frontend nivo sa servisima. Datoteka klase je već kreirana i moguće je pristupiti njenom kodiranju. U prvom koraku kreira se servisna klasa i obeležava se anotacijom `@Injectable()` kao u sledećem listingu. Takođe, neophodno je obezbediti konstruktor klase

```
@Injectable()
export class BookService{

    constructor(private _httpService: Http){}

}
```

Putem kreiranog konstruktora je obavljeno umetanje zavisnosti za `Http` objekat. Angular injektor će obaviti ovaj zadatak umesto programera. U nastavku će biti kreirane metode koje odgovaraju pozivima kreiranih REST servisa:

- `getAllBooks()` - vraća listu svih knjiga;
- `getBookById()` - vraća pojedinačne knjige na osnovu ID;
- `addBook()` - dodaje novu knjigu u bazu podataka;
- `deleteBook()` - briše knjigu iz baze podataka.
- `handleError()` - rukuje greškama.

Pre kreiranja navedenih metoda neophodno je navesti `Import` instrukcije koje je neophodno dodati da bi sve funkcionisalo na očekivanom nivou.

```
import { Injectable } from '@angular/core';
import { Http, Response, Headers, RequestOptions } from '@angular/http';
import { Observable } from 'rxjs';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/catch';
import { Book } from './book';
```

SERVIS - GET METODE I UPRAVLJANJE GREŠKAMA

Moguće je opremiti servisnu klasu klijenta odgovarajućim GET metodama

Konačno, moguće je opremiti servisnu klasu klijenta odgovarajućim metodama. Prva metoda odgovara **GET** zahtevu i vraća listu svih dostupnih knjiga iz baze podataka, dok druga metoda, koja takođe odgovara **GET** zahtevu vraća pojedinačne knjige na osnovu traženog identifikatora.

Pre uvođenja ovih metoda biće kreirana metoda `handleError()`. Njen zadatak je da preuzme **Request** objekat i da vrati poruku sa greškom. Sledi njen listing:

```
private handleError(error: Response){  
    return Observable.throw(error);  
}
```

Sada je moguće pristupiti kodiranju metode koja odgovara **GET** zahtevu i koja poziva kreirani **REST** servis za vraćanje liste svih dostupnih knjiga iz baze podataka. Poziv **REST** servisa se obavlja na osnovu poznatih krajnjih tačaka **REST** servisa definisanih u prvom objektu učenja. Nakon obavljenog poziva **REST** servisa, `map()` metoda prihvata **Response** objekat i poziva njegovu `json()` metodu. Ukoliko se desi greška prilikom izvršavanja navedenih metoda, metodom `catch()` se prihvata izuzetak. Sledi listing opisane metode.

```
getAllBooks(): Observable<Book[]>{  
    return this._httpClient.get("http://localhost:8084/Lekcija13Primer1/api/  
book")  
        .map((response: Response) => response.json())  
        .catch(this.handleError);  
}
```

Na sličan način moguće je pristupiti razvoju metode za vraćanje konkretnih knjiga na osnovu zahtevanog identifikatora knjige. Razlika u odnosu na prethodnu metodu vidi se u pozivu odgovarajuće krajnje tačke **REST** servisa i u povratnom tipu. Druga metoda vraća **Book** objekat umesto niza ovih objekata.

```
getBookById(bookId: string): Observable<Book>{  
    return this._httpClient.get("http://localhost:8084/Lekcija13Primer1/api/  
book/"+bookId)  
        .map((response: Response) => response.json())  
        .catch(this.handleError);  
}
```

SERVIS - PUT I POST METODA

Metoda obavlja pozive ka REST servisima za ažuriranje postojećih i dodavanje novih Book objekata

Sledeći korak jeste kreiranje metode koja obavlja pozive ka **REST** servisima za ažuriranje postojećih i dodavanje novih **Book** objekata u bazu podataka. Navedene zadatke je moguće obaviti definisanjem jedne metode `addBook()`.

Metoda kao parametar uzima *Book* objekat i prevodi ga u *JSON* string. Ukoliko njegov identifikator postoji u bazi podataka vrši se ažuriranje objekta, u suprotnom kreira se nov zapis u bazi podataka.

Sledi listing navedene metode.

```
addBook(book: Book){
    let body = JSON.parse(JSON.stringify(book));
    let headers = new Headers({ 'Content-Type': 'application/json' });
    let options = new RequestOptions({ headers: headers });
    if(book.id){
        return this._httpService.put("http://localhost:8084/Lekcija13Primer1/
api/book/"+book.id, body, options);
    }else{
        return this._httpService.post("http://localhost:8084/Lekcija13Primer1/
api/book", body, options);
    }
}
```

SERVIS - DELETE METODA I REGISTROVANJE SERVISA

Metoda za brisanje pojedinačnih zapisa iz baze podataka.

Zaokruživanje definicije servisne klase biće obavljeno kodiranjem metode za brisanje pojedinačnih zapisa iz baze podataka. Metoda se naziva jednostavno *deleteBook()* i po definiciji je najjednostavnija od svih kreiranih servisnih metoda - poziva metodu *Http* objekta *delete()* i predaje joj krajnju taktu *REST* servisa za brisanje objekata iz baze podataka. Sledi listing opisane metode.

```
deleteBook(bookId: string){
    return this._httpService.delete("http://localhost:8084/Lekcija13Primer1/api/
book/"+bookId);
}
```

Za primenu kreiranih servisnih metoda, neophodno je obaviti registrovanje servisa u *Angular* datoteci *app.module*. *Angularu* je neophodno saopštiti ko je provajder servisa, a to je *BookService* i neophodno je uključiti *HttpModule*. Navedeno je prikazano sledećom slikom.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { HttpClientModule } from '@angular/http';
import { BookService } from './book/book.service';

@NgModule({
  declarations: [
    AppComponent, BookComponent
  ],
  imports: [
    BrowserModule, HttpClientModule
  ],
  providers: [BookService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Slika 8.4.1 Registrovanje kreiranih servisa [izvor: autor]

PROVERA GET SERVISA

Biće provereno u konzoli veb pregledača da li kreirani servisi funkcionišu na očekivani način.

Da bi kreirani servisi bili funkcionalni i da bi mogli da budu upotrebljeni, biće redefinisana Angular klasa iz datoteke *book.component.ts*. Za početak biće provereno u konzoli veb pregledača da li kreirani servisi funkcionišu na očekivani način.

Neophodno je u razvojnom okruženju otvoriti datoteku navedene *Angular* klase i u prvom koraku kreirati konstruktor. Putem konstruktora vrši se umetanje *BookService* objekta.

```
constructor(private _bookService: BookService){}
```

Sada je moguće kreirati i metodu *getBooks()* koja će pozvati servisnu metodu *getAllBooks()*. Metoda neće vratiti ništa dok se ne prijavi objekat *bookData* kojem će biti pridružena lista knjiga. Ovo se dešava u metodi *subscribe()*. U slučaju javljanja greške, biće omogućeno njeno prikazivanje u konzoli veb pregledača.

Moguće je dodati još jednu funkcionalnost, koja će potom biti obrisana, za proveru da li metoda vraća listu knjiga u konzoli veb pregledača. Pozivi :

```
console.log(bookData);
console.log(error);
```

prikazuju u konzoli veb pregledača listu vraćenih knjiga i informaciju u vezi sa greškama, respektivno.

Sledećim listingom je prikazana opisana metoda `getBooks()`.

```
getBooks(): void{
    this._bookService.getAllBooks()
        .subscribe((bookData) => {this.books = bookData, console.log(bookData)},
            (error) =>{
                console.log(error);
                this.statusMessage = "Problem sa servisom. Molimo pokušajte
kasnije!";
            }
        );
}
```

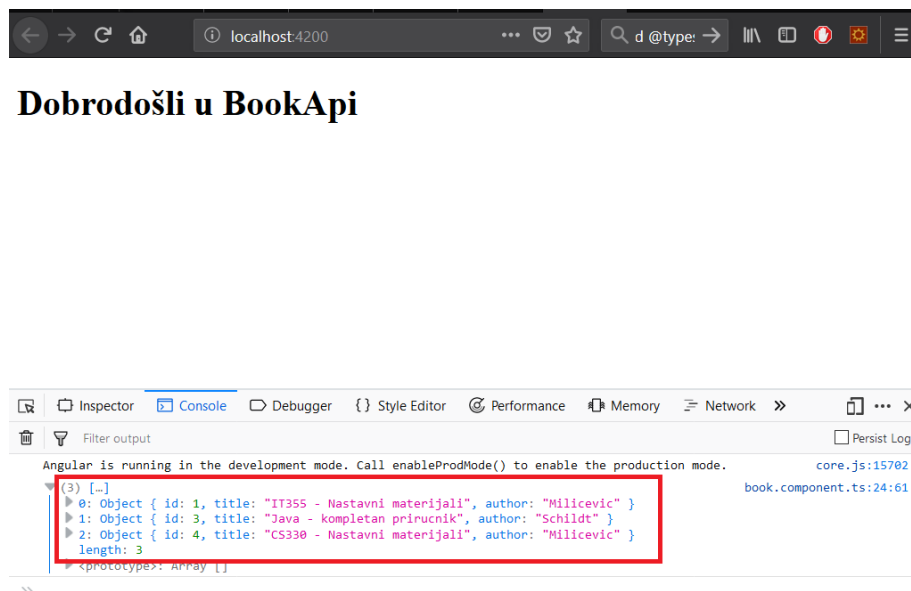
Takođe, klasa `BookComponent` će implementirati interfejs `OnInit` sa ciljem redefinisanja metode `ngOnInit()`. U okviru ove metode poziva se prethodno kreirana metoda `getBooks()`. Inicijalizujuća metoda klase `BookComponent` data je sledećim listingom.

```
ngOnInit(): void {
    this.getBooks();
}
```

PROVERA GET SERVISA - DEMO

Aplikacija se prevodi i u veb pregledaču se prati njeno izvršavanje.

Sada je dovoljno snimiti izvršene izmene, `Angular` aplikacija se sama prevodi i u veb pregledaču se prati njeno izvršavanje. Rezultati su prikazani sledećom slikom.



Slika 8.4.2 Praćenje poziva servisa iz konzole veb pregledača [izvor: autor]

Ukoliko se u konzoli veb pregledača javi greška tipa: "No 'Access-Control-Allow-Origin' header is present on the requested resource" to je iz razloga što REST kontroler, na backend strani, nije obeležen anotacijom `@CrossOrigin(origins = "**")`. Otklanjanjem ove greške dobija se konačno izlaz kao na slici 2.

PROVERA GET SERVISA IZ VEB STRANICE - DEMO

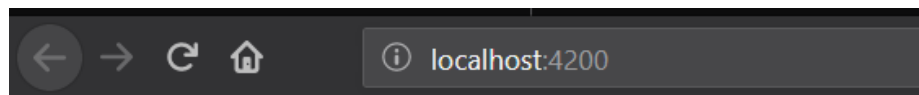
Pozive servisa se proverava u HTML stranici

Pošto je pokazano da kreirani servis funkcioniše na ispravan način, proverom u konzoli veb pregledača, sada je moguće dodati sledeći listing u datoteku `book.component.html`.

```
<h1>Dobrodošli u BookApi</h1>
<table class = "table table-striped table-bordered">
  <tr style = "background: lightblue;">
    <th>ID</th>
    <th>TITLE</th>
    <th>AUTHOR</th>
    <th>ACTIONS</th>
  </tr>
  <tr *ngFor = "let book of books">
    <td>{{book.id}}</td>
    <td>{{book.title}}</td>
    <td>{{book.author}}</td>
    <td>
      EDIT | DELETE
    </td>
  </tr>
  <tr *ngIf = "books && books.length == 0">
    <td colspan = "4">Ne postoje zapisi!</td>
  </tr>
</table>
```

Ovaj `HTML` fragment prikazuje tabelu sa svim knjigama koje postoje u bazi podataka. Ovaj zadatak obavlja Angular petlja `*ngFor`. Opcije `EDIT` i `DELETE` za sada ovde nisu podržane.

Sledećom slikom je prikazana lista svih dostupnih knjiga u bazi podataka putem HTML tabele.



Dobrodošli u BookApi

ID	TITLE	AUTHOR	ACTIONS
1	IT355 - Nastavni materijali	Milicevic	EDIT DELETE
3	Java - kompletan prirucnik	Schildt	EDIT DELETE
4	CS330 - Nastavni materijali	Milicevic	EDIT DELETE

Slika 8.4.3 Provera servisa iz veb stranice - demo [izvor: autor]

▼ 8.4 Pozivi metoda servisnog nivoa klijenta

PROVERA SERVISNIH METODA KLIJENTA

Sada je cilj provera i ostalih metoda koje se obraćaju kreiranim REST servisima

U prethodnom izlaganju je proverena metoda servisnog nivoa klijenta koja vraća sve *Book* objekte koji su sačuvani u bazi podataka. Sada je cilj provera i ostalih metoda koje se obraćaju kreiranim *REST* servisima, a oslanjaju se na *HTTP* pozive tipa: *PUT*, *POST* i *DELETE*:

- dodavanje novih objekata;
- ažuriranje postojećih objekata;
- brisanje objekata iz baze podataka.

PROVERA POST SERVISA

Analiza metode koja podržava poziv POST REST servisa.

Osnov za dalju analizu i diskusiju predstavlja kreirana metoda *addBook()* servisne klase klijenta. Ova metoda podržava poziv *PUT* i *POST REST* servisa.

```
addBook(book: Book){
  let body = JSON.parse(JSON.stringify(book));
  let headers = new Headers({ 'Content-Type': 'application/json' });
  let options = new RequestOptions({ headers: headers });
  if(book.id){
    return this._httpService.put("http://localhost:8084/Lekcija13Primer1/
api/book/"+book.id, body, options);
```

```
    }else{  
        return this._httpService.post("http://localhost:8084/Lekcija13Primer1/  
api/book", body, options);  
    }  
}
```

Kao što je moguće primetiti iz priloženog listinga, metoda uzima Book objekat, parsira ga u *JSON* string. Ukoliko identifikator objekta postoji u bazi podataka, objekat će biti ažuriran. U suprotnom, ako identifikator ne postoji, objekat će biti sačuvan kao nov. Ove dve akcije se obavljaju pozivom odgovarajućih *REST* servisa.

Na sličan način, kao što je bio slučaj sa GET servisom, neophodno je u "klijent kontroleru" *BookComponent* omogućiti pozivanje metode iz servisnog nivoa klijenta *addBook()*. U kodu klase *BookComponent*, neophodno je prvo kreirati nov objekat Book:

```
book = new Book();
```

Nakon toga je neophodno kreirati i *BookComponent* metodu *addBook()*:

```
addBook(): void{  
    this._bookService.addBook(this.book)  
        .subscribe((response) => {console.log(response);  
this.getBooks();this.reset();},  
        (error) =>{  
            console.log(error);  
            this.statusMessage = "Problem sa servisom. Molimo pokusajte  
kasnije!";  
        }  
    );  
}
```

Takođe, u kreiranoj metodi je moguće primetiti i poziv metode *reset()* čiji je zadatak resetovanje polja za unos teksta na formi i ažuriranje liste dostupnih objekata preuzetih iz baze podataka.

```
private reset(){  
    this.book.id = null;  
    this.book.title = null;  
    this.book.author = null;  
}
```

PROVERA POST SERVISA - KREIRANJE FORME

*Neophodno je ponovo izvršiti modifikaciju Angular datoteke *app.module.ts*.*

Budući da će biti korišćena *HTML* forma za dodavanje novog korisnika u bazu podataka, neophodno je ponovo izvršiti modifikaciju Angular datoteke *app.module.ts*. Potrebno je importovati *FormsModule* na način prikazan sledećom slikom.


```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { BookComponent } from './book/book.component';
import { HttpClientModule } from '@angular/http';
import { BookService } from './book/book.service';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    AppComponent, BookComponent
  ],
  imports: [
    BrowserModule, HttpClientModule, FormsModule
  ],
  providers: [BookService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Slika 8.5.1 Dodavanje modula FormsModule [izvor: autor]

Sada je moguće dodati *HTML* formu za dodavanje novog *Book* objekta u bazu podataka. U datoteku *book.component.html*, a iznad prethodno kreirane tabele za prikazivanje liste svih *Book* objekata, neophodno je dodati sledeću *HTML* formu.

```
<h1>Dobrodošli u BookApi</h1>
<form class="form-horizontal">
  <input type = "hidden" [(ngModel)] = "book.id" name = "id" />
  <div class="form-group">
    <label class="control-label col-sm-2">Book Title</label>
    <div class="col-sm-8">
      <input type = "text" [(ngModel)] = "book.title" name = "title"
class="form-control" />
    </div>
  </div>
  <div class="form-group">
    <label class="control-label col-sm-2">Book Author</label>
    <div class="col-sm-8">
      <input type = "text" [(ngModel)] = "book.author" name = "author"
class="form-control" />
    </div>
  </div>
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-8">
      <button (click) = "addBook()" class = "btn btn-primary">Add
Book</button>
    </div>
  </div>
</form>
```

Takođe, moguće je primetiti da su polja za unos teksta forme, preko *ngModel*, povezana sa poljima modelske klase *Book*.

PROVERA POST SERVISA - DEMO

Demonstracija funkcionalnosti poziva PUT REST servisa od strane klijenta.

U nastavku je neophodno obaviti demonstraciju funkcionalnosti poziva *PUT REST* servisa od strane klijenta.

Unosi se nov objekat kao na sledećoj slici.

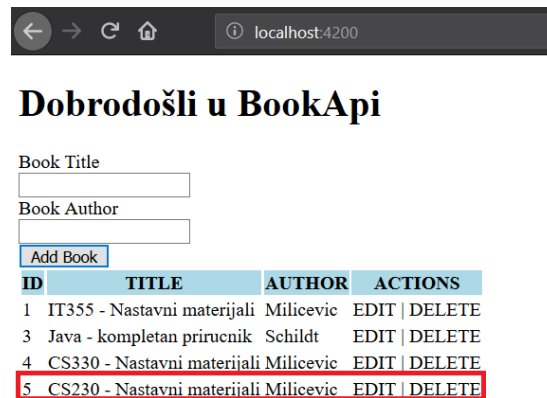


The screenshot shows a web browser at localhost:4200. The page title is 'Dobrodošli u BookApi'. There is a form with two input fields: 'Book Title' (containing '230 - Nastavni materijali') and 'Book Author' (containing 'Milicevic'). Below the form is an 'Add Book' button. Below the form is a table with the following data:

ID	TITLE	AUTHOR	ACTIONS
1	IT355 - Nastavni materijali	Milicevic	EDIT DELETE
3	Java - kompletan prirucnik	Schildt	EDIT DELETE
4	CS330 - Nastavni materijali	Milicevic	EDIT DELETE

Slika 8.5.2 Unos novog Book objekta [izvor: autor]

Klikom na dugme *Add Book* objekat se dodaje u bazu podataka. Uspešno obavljen zadatak je prikazan sledećom slikom.

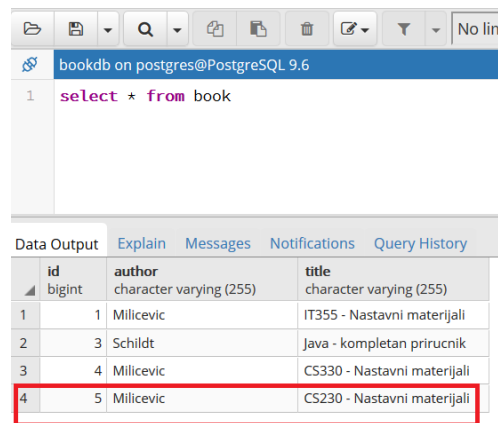


The screenshot shows the same web browser at localhost:4200. The 'Add Book' button is now highlighted in blue. The table now has an additional row, which is highlighted with a red border:

ID	TITLE	AUTHOR	ACTIONS
1	IT355 - Nastavni materijali	Milicevic	EDIT DELETE
3	Java - kompletan prirucnik	Schildt	EDIT DELETE
4	CS330 - Nastavni materijali	Milicevic	EDIT DELETE
5	CS230 - Nastavni materijali	Milicevic	EDIT DELETE

Slika 8.5.3 Sačuvani Book objekat [izvor: autor]

Konačno, proverite da li je stvarno objekat sačuvan u bazi podataka.



id	author	title
1	Milicevic	IT355 - Nastavni materijali
2	Schildt	Java - kompletan prirucnik
3	Milicevic	CS330 - Nastavni materijali
4	Milicevic	CS230 - Nastavni materijali

Slika 8.5.4 Sačuvani Book objekat - baza podataka [izvor: autor]

PROVERA DELETE SERVISIA

Metoda `deleteBook()` podržava poziv `DELETE REST` servisa.

Osnov za dalju analizu i diskusiju predstavlja kreirana metoda `deleteBook()` servisne klase klijenta. Ova metoda podržava poziv `DELETE REST` servisa.

```
deleteBook(bookId: string){
    return this._httpClient.delete("http://localhost:8084/Lekcija13Primer1/api/
book/"+bookId);
}
```

Kao što je moguće primetiti iz listinga, metoda uzima kao parametar identifikator sačuvanog `Book` objekta kojeg ne neophodno izbrisati iz baze podataka.

Na sličan način, kao što je bio slučaj sa `GET` i `POST` servisom, neophodno je u "klijent kontroleru" `BookComponent` omogućiti pozivanje metode iz servisnog nivoa klijenta, u ovom slučaju `deleteBook()`.

```
deleteBook(bookId: string){
    console.log("Pocetak metode deleteBook():::Book id:::"+bookId);
    this._bookService.deleteBook(bookId)
        .subscribe((response) => {console.log(response); this.getBooks();},
        (error) =>{
            console.log(error);
            this.statusMessage = "Problem sa servisom. Molimo pokusajte
kasnije!";
        });
    this.reset();
    console.log("kraj metode deleteBook():::");
}
```

PROVERA DELETE SERVISA - KREIRANJE DUGMETA ZA POZIV SERVISA

U daljem radu je neophodno napraviti novu modifikaciju u datoteci `book.component.html`.

U daljem radu je neophodno napraviti novu modifikaciju u datoteci `book.component.html`. U tabeli u kojoj se prikazuju dostupni `Book` objekti iz baze podataka, u poslednjoj koloni stoje dva stringa `EDIT` i `DELETE`, trenutno bez funkcije. Cilj je da se ovi stringovi zamene dugmadima kojima će biti omogućeno izvođenje odgovarajućih operacija. Da bi navedeno bilo moguće, neophodno je navedeni kod u datoteci `book.component.html` zameniti kodom iz sledećeg listinga:

```
<td>
    <button (click) = "getBookByID(book.id)">Edit</button>
    &nbsp;|&nbsp;
    <button (click) = "deleteBook(book.id)">Delete</button>
</td>
```

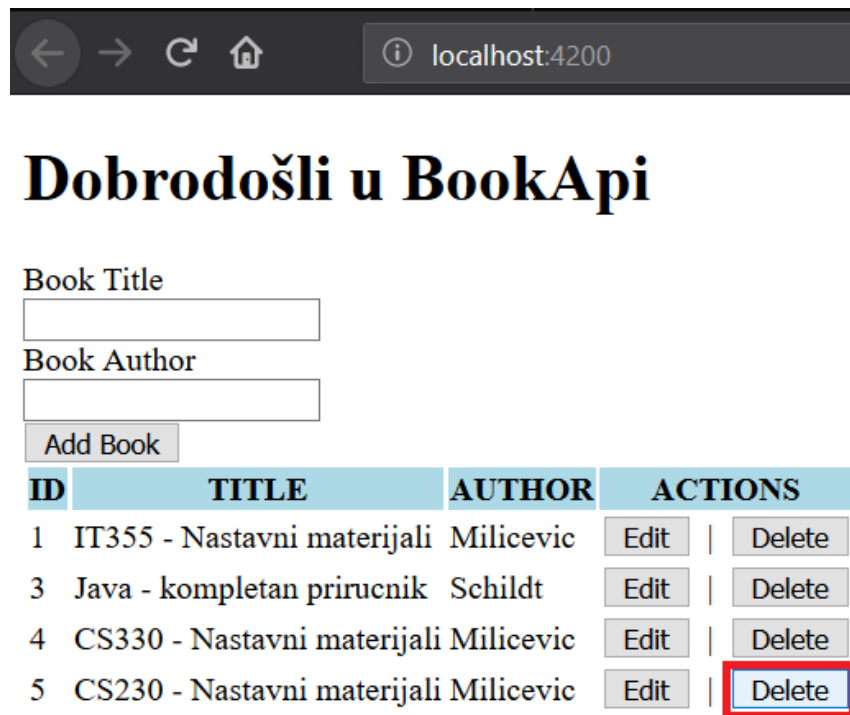
Moguće je primetiti da je kreirano i dugme Edit koje će dobiti funkcionalnost u naknadnoj analizi i diskusiji.

PROVERA DELETE SERVISA - DEMO

Demonstracija funkcionalnosti poziva `DELETE REST` servisa od strane klijenta.

U nastavku je neophodno obaviti demonstraciju funkcionalnosti poziva `DELETE REST` servisa od strane klijenta.

Klikom na dugme `DELETE`, pored odgovarajuće knjige, vrši se brisanje `Book` objekta iz baze podataka.



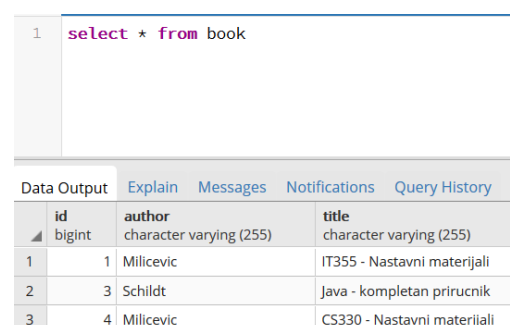
Slika 8.5.5 Brisanje knjige iz dostupne liste [izvor: autor]

Ubrzo se dobija ažurirana tabela sa dostupnim knjigama, kao na sledećoj slici.



Slika 8.5.6 Ažurirana tabela sa dostupnim knjiga [izvor: autor]

Konačno, moguće je proveriti da je knjiga zaista obrisana iz baze podataka.



Slika 8.5.7 Provera brisanja iz baze podataka [izvor: autor]

PROVERA PUT SERVISA

Analiza metode koja podržava poziv PUT servisa.

Osnov za dalju analizu i diskusiju predstavlja kreirana metoda `addBook()` servisne klase klijenta. Ova metoda podržava poziv *PUT* i *POST REST* servisa.

```
addBook(book: Book){
    let body = JSON.parse(JSON.stringify(book));
    let headers = new Headers({ 'Content-Type': 'application/json' });
    let options = new RequestOptions({ headers: headers });
    if(book.id){
        return this._httpService.put("http://localhost:8084/Lekcija13Primer1/
api/book/"+book.id, body, options);
    }else{
        return this._httpService.post("http://localhost:8084/Lekcija13Primer1/
api/book", body, options);
    }
}
```

Slučaj sa *POST* pozivom je diskutovan i sada je na redu analiza i diskusija vezana za poziv *REST* servisa baziran na *PUT* zahtevu. Razlikovaće se dva slučaja:

- klikom na dugme *Edit*, biće izabrana knjiga čije je podatke neophodno ažurirati;
- podaci o knjizi će se pojaviti u poljima za unos teksta kreirane forme; nakon izmene klikom na dugme *AddBook* podaci se ažuriraju u bazi podataka.

Dugme je već kreirano, zajedno sa *Delete* dugmetom i ovde je posao dodatno olakšan.

Ovde je takođe od posebnog značaja servisna metoda `getBookById()` čiji je zadatak da vrati pojedinačni objekat iz baze podataka. Posebno je značajna za vraćanje objekta pored kojeg je izabran klik na dugme *EDIT*.

```
getBookById(bookId: string): Observable<Book>{
    return this._httpService.get("http://localhost:8084/Lekcija13Primer1/api/
book/"+bookId)
        .map((response: Response) => response.json())
        .catch(this.handleError);
}
```

Na sličan način, kao što je bio slučaj sa *GET*, *POST* i *DELETE* servisom, neophodno je u "klijent kontroleru" *BookComponent* omogućiti pozivanje metode iz servisnog nivoa klijenta, u ovom slučaju `GetBookById()`.

```
getBookById(bookId: string){
    this._bookService.getBookById(bookId)
        .subscribe((bookData) => {this.book = bookData; this.getBooks(); }),
        (error) => {
            console.log(error);
            this.statusMessage = "Problem sa servisom. Molimo pokušajte
kasnije!";
        }
}
```

```
}
  this.reset();
}
```

PROVERA PUT SERVISA - DEMO

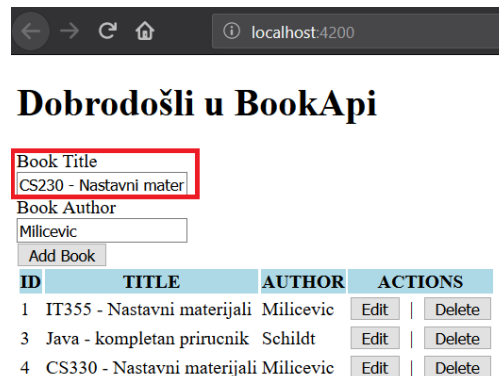
Provera funkcionalnosti ažuriranja objekata iz baze podataka.

Klikom na **EDIT** pored objekta bira se objekat koji se ažurira.



Slika 8.5.8 Izbor objekta za ažuriranje [izvor: autor]

U poljima za unos teksta se menjaju podaci objekta,



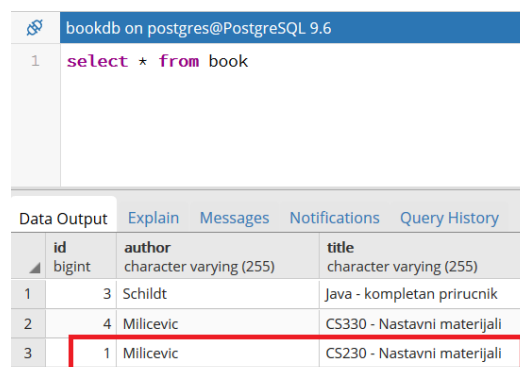
Slika 8.5.9 Izmena podataka objekta [izvor: autor]

Klikom na **AddBook** podaci su ažurirani i prikazani u osveženoj listi.



Slika 8.5.10 Ažuriran objekat u listi objekata [izvor: autor]

Konačno, izvršena je provera korektnosti ažuriranja u bazi podataka.



Slika 8.5.11 Ažuriran objekat u bazi podataka [izvor: autor]

VIDEO MATERIJAL

[Angular 6 + Spring Boot] Introduction to Angular + Spring Boot - serija video materijala

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 9

Pokazna vežba 13

KRIRANJE ANGULAR PROJEKTA (45 MINUTA)

Angular framework nam omogućava da lako pravimo front end aplikacije koje koriste web servise

Kako bismo koristili Angular 2 framework potreban nam je Node JS package manager (NPM). Ako nemate NPM instaliran treba da ga instalirate. Više detalja, kao i link ka download-u možete pronaći ovde: <https://nodejs.org/en/download/>

Po uspešnoj instalaciji NPM/a, neophodno je da instaliramo Angular CLI koristeći sledeću komandu:

```
- npm install -g @angular/cli
```

Potom po uspešnoj instalaciji Angular CLI/a, neophodno je otići u željeni direktorijum i kreirati projekat kroz sledeću komandu:

```
- ng new PROJECT-NAME
```

Time smo uspešno kreirali naš Angular projekat. Nakon toga, ukoliko želimo isti dodati na naš Github repozitorijum, koristićemo sledeće komande:

```
- git init
- git add .
- git commit -m "init commit"
- git remote add origin <adresa repoa>
- git push -u origin master
```

Angular 2 okvir ili framework je jedan od najkorišćenijih i najtraženijih front end frameworka danas. Ovaj framework nam omogućava da lako pravimo front end aplikacije koje koriste web servise. Pošto je danas SOA arhitektura najzastupljenija zbog mobilnih aplikacija, ovo je idealno rešenje koje omogućava da se samo jednom pravi back-end aplikacije dok se frontend radi nativno za mobilne platforme, a upravo preko Angular JS okvira za Web.

Za uspešno pokretanje Angular projekta, koristimo jednu od sledeće 2 komande:

```
- npm start
- ng serve
```

Po uspešnom pokretanju, u pregledaču kada otvorimo <http://localhost:4200> trebalo bi da imamo sledeći rezultat kao što je prikazano na slici 1.

Welcome to My First Angular App!



Slika 9.1 Pokretanje inicijalne Angular aplikacije

OPIS ANGULAR PROJEKTA I FAJLOVA

Glavni fajlovi projekta nalaze se uglavnom u okviru src direktorijuma

Glavni fajlovi koji su nam najbitniji i koje ćemo menjati, dodavati nalaze se uglavnom u okviru "src" direktorijuma. Neki od najbitnijih fajlova su:

- *main.ts* - predstavlja pokretačku klasu aplikacije koja koristi metodu *Bootstrap* da učitava glavni *Angular* modul. U ovom slučaju taj modul je *AppModule* iz fajla *app.module.ts*.
- *app.module.ts* - predstavlja glavni modul za *Angular* koji određuje glavnu komponentu sistema kao i korišćenje paketa unutar sistema kroz importe. U našem slučaju glavna komponenta je *AppComponent* iz fajla *app.component.ts*.
- *app.component.ts* - predstavlja *Angular 2* komponentu a u ovom slučaju i glavnu komponentu.

U *Angularu 2* sve je bazirano na komponentama pa se svaka strana pa čak i web strane aplikacije može nazvati komponentom. Komponenta ima svoju klasu koja može imati svoje parametre. U ovom slučaju klasa ima parametar *name* koji se koristi u templejtu za prikaz. Glavna komponenta mora da ima selector koji predstavlja element po kome se vezuje sa *index.html* fajlom.

POVEZIVANJE SA SPRINGOM

Kako bismo koristili Spring i Angular, neophodno je da podesimo CROS origin policy.

Kako bismo koristili *Spring* i *Angular*, neophodno je da podesimo *CROS* origin policy, tj. da naša *Spring* aplikacija bude dostupna za zahtev sa drugih domena i poddomena. To postizemo tako što dodamo anotaciju *@CrossOrigin* iznad definicije našeg *RestController*. Za potrebe ovog projekta koristićemo rest kontroler koji se nalazi u okviru vežbe br. 9. Isečak tog koda možemo videti ispod.

```
@CrossOrigin
@RestController
public class MyController {
```

Slika 9.2 Dodavanje anotacije @CrossOrigin iznad definicije našeg RestControllera

Takođe, u okviru istog fajla neophodno je da editujemo Post zahtev za proizvod, kako bismo mogli uspešno da ga koristimo kroz *Angular* na sledeći način:

```
@PostMapping("product")
public ResponseEntity<Void> addProduct(@RequestBody Product product) {
    System.out.println("Adding product " + product.toString());
    shoppingCartDao.addProduct(product);

    HttpHeaders headers = new HttpHeaders();
    return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
}
```

Nakon toga, možemo kod preuzeti iz dela deljenog sa vežbi (Link za kod: <https://github.com/markorajevic/it355-ang>).

Koristeći komandu "*ng g s http*" kreiramo *Http* servis koji koristimo za komunikaciju sa našim *Rest Api*-jem.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class HttpService {

  private _apiUrl = 'http://localhost:8080/HibernateCRUD/';

  constructor(private _http: HttpClient) {

  }

  public get(route: String) {
    return this._http.get(this._apiUrl + route);
  }

  public post(route: String, data: any) {
    return this._http.post(this._apiUrl + route, data);
  }

  public delete(id: String) {
```

```
    return this._http.delete(this._apiUrl + '/product/' + id);  
  }  
}
```

KRAJ RADA I POKRETANJE

Kreiranje komponentata za prikaz i dodavanje proizvoda

Nakon toga koristeći komandu "*ng g c pages/add*" i "*ng g c pages/products*" kreiramo dve komponente koje koristimo za prikaz i dodavanje proizvoda:

Modifikovani kod tih komponenti možemo videti ispod.

ADD:

- <https://github.com/markorajevic/it355-ang/blob/master/src/app/pages/add/add.component.html>
- <https://github.com/markorajevic/it355-ang/blob/master/src/app/pages/add/add.component.ts>

PRODUCTS:

- <https://github.com/markorajevic/it355-ang/blob/master/src/app/pages/products/products.component.ts>
- <https://github.com/markorajevic/it355-ang/blob/master/src/app/pages/products/products.component.html>

Sve ovo u cilju ubrzanja rada možete skinuti sa sledećeg linka: <https://github.com/markorajevic/it355-ang>

Nakon uspešnog skidanja, u okviru istog direktorijuma pokrenite sledeće komande:

```
npm install  
npm start
```

Potom će vaša aplikacija biti dostupna na sledećem linku:
<http://localhost:4200>

▼ Poglavlje 10

Individualna vežba 11

INDIVIDUALNA VEŽBA (90 MIN)

Kreiranje Angular frontenda za kontroler iz individualne vežbe Lekcije 9.

Kreirajte Angular frontend za pozive metoda REST kontrolera iz individualne vežbe Lekcije 9. Angular klijent mora da poseduje sve elemente navedene u predavanjima i pokaznoj vežbi.

▼ Poglavlje 11

Domaći zadatak 13

DOMAĆI ZADATAK 13 - ANGULAR KLIJENT NAD SPRING FRONTEND-OM

Samostalno kreiranje Angular klijenta nad Spring frontend-om (135 min)

- Kreirajte *Angular* frontend za pozive metoda *REST* kontrolera iz domaćeg zadatka Lekcije 9.
- *Angular* klijent mora da poseduje sve elemente navedene u predavanjima i pokaznoj vežbi.
- Izradu domaćeg zadatka je neophodno detaljno dokumentovati.
- *Angular* projekat treba izraditi koristeći materijal sa vežbi i predavanja
- Urađeni zadatak u formi zipovanog projekta poslati vašem asistentu na mail.

Nakon obaveznog DZ studenti dobijaju različite zadatke na email od predmetnog asistenta.

▼ Poglavlje 12

Zaključak

ZAKLJUČAK

Lekcija 13 je kombinovala napredne frontend i backend tehnologije

Lekcija 13 se bavila čestim pristupom kojeg je moguće sresti u savremenoj industriji veb softvera - kombinovanje naprednih *frontend* i *backend* tehnologija:

- *Angular* - *JavaScript* frontend radni okvir za razvoj veb aplikacija;
- *Spring MVC* - *Java* radni okvir za razvoj veb aplikacija koji će biti korišćen za razvoj serverskog dela aplikacije.

Cilj ove lekcije je upravo bio da nauči studente kako se pristupa razvoju potpuno funkcionalne veb aplikacije, nad bazom podataka, koja će koristiti okvir za podršku skripting programiranju za razvoj klijent strane aplikacije, a takođe i *Spring (Boot)* okvir, koji se oslanja na primenu pravog programskog jezika, za kreiranje servisa nad bazom podataka za obradu zahteva klijenata.

Savladavanjem ove lekcije studenti su naučili da kombinuju i koriste tehnologije i alate obrađene i naučene u predmetima *IT255 - Vebsistemi 1* i *IT355 - Veb sistemi 2*.

LITERATURA

Za pripremu lekcije 13 korišćena je najnovija štampana i elektronska literatura

1. Marten Deinum, Josh Long, and Daniel Rubio, Spring 5 Recipes Fourth Edition, Apress
2. Nate Murray, Felipe Coury, Ari Lerner, and Carlos Taborda. ng-book - The Complete Book on Angular 6, Fullstack.io
3. Gary Mak, Josh Long, and Daniel Rubio, Spring Recipes Third Edition, Apress
4. Spring Framework Reference Documentation - <https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/>
5. Craig Walls, Spring in Action, Manning
6. Craig Walls, Spring Boot in Action, Manning
7. <https://www.youtube.com/watch?v=Oj28rYNwN98&list=PLVApX3evDwJ2keNFsQ4PhA1yVaXpto2oN>