UNIVERZITET METROPOLITAN BEOGRAD

IT355 - WEB SISTEMI 2

Uvod u Spring okvir

Lekcija 01

PRIRUČNIK ZA STUDENTE

IT355 - WEB SISTEMI 2

Lekcija 01

UVOD U SPRING OKVIR

- Uvod u Spring okvir
- Poglavlje 1: Umetanje zavisnosti i inverzija kontrole
- Poglavlje 2: Moduli Spring okvira
- Poglavlje 3: Scenariji upotrebe okvira Spring 5
- Poglavlje 4: Verzije okvira - Spring 4
- Poglavlje 5: Verzije okvira - Spring 4.1
- Poglavlje 6: Verzije okvira - Spring 4.2
- Poglavlje 7: Verzije okvira - Spring 4.3
- Poglavlje 8: Verzije okvira - Spring 5
- Poglavlje 9: Pokazni primer - Maven konfiguracioni fajl
- Poglavlje 10: Pokazna vežba 1 - Maven projekat
- Poglavlje 11: Individualna vežba 1
- Poglavlje 12: Domaći zadatak 1
- Zaključak

UVOD

Spring je Java platforma koja obezbeđuje obimnu infrastrukturnu podršku za razvoj Java aplikacija.

Spring okvir (*Spring Framework*) predstavlja Java platformu koja obezbeđuje obimnu infrastrukturnu podršku za razvoj širokog spektra Java aplikacija. Celokupnom infrastrukturom upravlja Spring pa tim za razvoj može da se fokusira isključivo na razvoj konkretnih programa.

Spring omogućava kreiranje aplikacija iz POJO (plain old Java objects) objekata i primenu složenih (enterprise) servisa na POJO objekte. Navedene funkcionalnosti Spring okvira primenjuju se u potpunosti na Java SE (*Java Standard Edition*) model programiranja i u potpunosti, ili parcijalno, na Java EE (*Java Enterprise Edition*) model programiranja.

Programer može da ima brojne benefite kada koristi Spring okvir za kreiranje svojih Java programa. Slede neke očigledne prednosti koje programer može da ima upotrebom ove platforme:

- Izvršavanje Java metoda kroz transakciju sa bazom podataka bez potrebe za rukovanjem konkretnim API - jem za transakcije;
- Kreiranje HTTP krajnje tačke (HTTP endpoint) lokalne Java metode bez potrebe za rukovanjem konkretnim *Servlet API*;
- Kreiranje rukovalaca poruka (message handler) lokalne Java metode bez potrebe za rukovanjem konkretnim *JMS API;*
- Kreiranje upravljačkih operacija (management operation) lokalne Java metode bez potrebe za rukovanjem konkretnim *JMX API*.

Tokom ovog predmeta studenti će steći široku sliku o savremenoj primeni Spring okvira za razvoj Java veb aplikacija. Materijali će se oslanjati na pažljivo birane primere koji će pratiti svako izlaganje koje se odnosi na koncepte i principe Spring okvira.

Poglavlje 1

Umetanje zavisnosti i inverzija kontrole

ŠTA SU UMETANJE ZAVISNOSTI I INVERZIJA KONTROLE?

IoC predstavlja komponentu koja obezbeđuje formalizovane mehanizame uklapanja komponenata u funkcionalnu celinu.

Savremene Java aplikacije mogu se sretati u brojnim oblicima, od prostih ograničenih aplikacija ugrađenih u brojne uređaje do izuzetno kompleksnih, više - nivojskih enterprise serverskih (*server side*) aplikacija. Sve Java aplikacije imaju jednu zajedničku osobinu. Sastoje se iz skupa objekata koji između sebe kolaboriraju (sarađuju). Otuda postoje i međusobne zavisnosti između objekata koji formiraju posmatranu aplikaciju.

Iako Java platforma obezbeđuje ogromnu podršku za razvoj funkcionalnosti Java aplikacija, nedostaju joj mehanizmi za organizovanje blokova koji čine aplikaciju kao koherentnu celinu pa su ovi zadaci prepušteni arhitektima sistema i programerima. Ovo posebno otežava razvoj budući da se članovi razvojnog softverskog tima moraju baviti napornim i dugotrajnim zadacima pre razvoja konkretne programske logike. Iako je moguće koristiti dizajn šablone (design patterns) poput: ***Factory***, ***Abstract Factory***, ***Builder***, ***Decorator***, ***Service Locator*** i drugih, za kombinovanje različitih klasa i odgovarajućih objekata, kao i što su ovi šabloni: najbolja praksa za dodelu naziva u aplikaciji, sa opisom šta su konkretni zadaci šablona, gde se primenjuju šabloni i njihovi zadaci i na koje se probleme odnose i tako dalje, šabloni predstavljaju formalizovanu najbolju praksu koja mora biti ručno urađena u aplikaciju.

Inverzija kontrole Spring okvira (Spring Framework Inversion of Control (IoC)) predstavlja komponentu koja rešava navedeni problem kroz obezbeđivanje formalizovanih mehanizama uklapanja komponenata u funkcionalnu celinu. Spring okvir kodira dizajn šablone kao klase prvog reda koje je moguće integrisati u aplikacije koje se kreiraju.

> Brojne softverske komanije, upravo, koriste Spring okvir za kreiranje robustnih i održivih Java aplikacija.

Poglavlje 2

Moduli Spring okvira

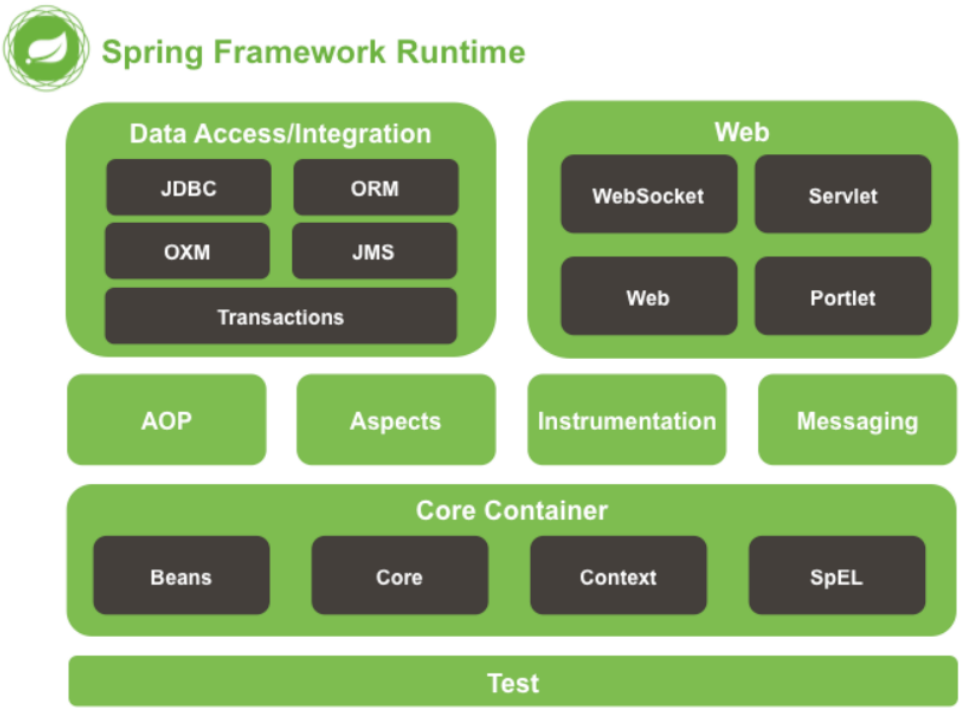
ORGANIZACIJA SPRING OKVIRA

Spring okvir je oranizovan u formi 20 - ak modula.

Spring okvir se sastoji od elemenata koji su organizovani u okviru 20 - ak modula. Navedeni moduli su organizovani u celine pod sledećim nazivima:

- Core Container,
- Data Access/Integration,
- Web,
- AOP (Aspect Oriented Programming),
- Instrumentation,
- Messaging,
- Test.

Organizacija Spring okvira može na elegantan način biti prikazana sledećom šemom



Slika 2.1 Organizaciona šema Spring okvira [izvor: Spring]

CORE CONTAINER

Osnovni kontejner (Core Container) čine osnovni Spring moduli.

Osnovni kontejner (Core Container) čine osnovni Spring moduli: ***spring-core***, ***spring-beans***, ***spring-context***, ***spring-context-support*** i ***spring-expression*** (*Spring Expression Language*).

Moduli ***spring-core*** i ***spring-beans*** su moduli koji obezbeđuju fundamentalne delove Spring okvira, kao što su inverzija kontrole (IoC) i umetanje zavisnosti. Ovde je od posebnog značaja ***BeanFactory*** kontejner koji predstavlja sofisticiranu primenu šablona produkcije (factory pattern). Ovde je uklonjena potreba za programiranim singlton zrnima (***singletone***) i omogućeno je razdvajanje konfiguracije i specifikacije zavisnosti od konkretne programske logike.

Modul konteksta (***spring-context***) čini čvrstu osnovu obezbeđenu modulima ***Core*** i ***Beans***. To podrazumeva pristup objektima na način karakterističan za okvire na sličan način kao kada se primenjuju JNDI (*Java Naming and Directory Interface*). Kontekstni modul nasleđuje vlastite alate iz modula ***Beans*** i uvodi podršku za internacionalizaciju (na primer primenom koncepta resource bundle), rad sa događajima, učitavanje resursa, transparentno kreiranje konteksta upotrebom, na primer, ***Servlet*** kontejnera i tako dalje. Kontekstni modul takođe podržava Java EE (Java Enterprise Edition) alate kao što su: EJB, JMX i rad sa udaljenim resursima. Interfejs ***ApplicationContext*** je centralni deo ***Context*** modula. Posebno, ***spring-context-support*** obezbeđuje podršku za integrisanje biblioteka "treće strane" u kontekst Spring aplikacije za: keširanje (EhCache, Guava, JCache), rad sa porukama (JavaMail), organizovanje (CommonJ, Quartz) i mehanizme rada sa šablonima (FreeMarker, JasperReports, Velocity).

Modul pod nazivom spring-expression obezbeđuje moćan jezik izraza - *Spring Expression Language (SpEL)* za kreiranje upita i manipulisanje šemom objekata tokom vremena izvršavanja aplikacije. Ovaj jezik predstavlja proširenje unificiranog jezika izraza (unified expression language) koji predstavlja deo JSP 2.1 specifikacije (*Java Server Pages*). Jezik podržava podešavanje i preuzimanje vrednosti osobina, dodeljivanje osobina, pozivanje metoda, pristupanje sadržaju nizova i kolekcija, logičke i aritmetičke operacije, primenu varijabli, vraćanje objekata po nazivu iz Spring IoC kontejnera i još mnogo toga. *Spring Expression Language*, takođe, podržava projekciju i selekciju lista kao i spajanje lista.

AOP, INSTRUMENTI I RAD SA PORUKAMA

Spring obezbeđuje implementaciju aspektno - orijentisanog programiranja, instumenata i mehanizama za rad sa porukama.

Modul ***spring-aop*** obezbeđuje implementaciju *aspektno - orijentisanog programiranja* (AOP - aspect-oriented programming) dozvoljavajući definisanje presretača metoda (interceptors) i tačaka prekida (pointcuts) za jasno razdvajanje programskog koda koji implementira funkcionalnosti koje moraju da budu razdvojene (poznato kao razdvajanje funkcionalnosti - separation of concerns (SoC)). Primenom metapodataka za funkcionalnosti iskazane na nivou

koda, moguće je takođe, primenom AOP - a ugraditi informacije o ponašanju u Spring kod, na način sličan atributima u .NET tehnologiji.

Poseban modul ***spring-aspects*** obezbeđuje primenu najpoznatijeg aspektno - orijentisanog pristupa kroz integraciju sa istoimenim aspektno - orijentisanim okvirom *AspectJ*.

Modul ***spring-instrument*** obezbeđuje podršku za instrumentalizaciju klasa i implementaciju čitača klasa (classloader) koji se koriste u izvesnom aplikativnom serveru. Posebno, modul ***spring-instrument-tomcat*** sadrži Spring agenta instrumentalizacije za aplikativni server *Apache Tomcat*.

Spring okvir (Framework) 5 uključuje ***spring-messaging*** modul sa ključnim apstrakcijama definisanim u projektu ***Spring Integration*** za rad sa porukama, poput: ***Message***, ***MessageChannel***, ***MessageHandler***, kao i drugim koje predstavljaju osnovu za kreiranje aplikacija baziranih na porukama (messaging-based applications). Modul, takođe, sadrži skup anotacija za mapiranje (preslikavanje) poruka u metode, slično kao u *Spring MVC* modelu programiranja baziranim na anotacijama.

PRISTUP PODACIMA I INTEGRACIJA

Data Access/Integration nivo je značajan deo Spring 5 okvira za pristup podacima i integraciju.

Data Access/Integration nivo je veoma značajan deo *Spring 5* okvira. Ovaj nivo, za pristup podacima i integraciju, čini nekoliko značajnih modula: ***JDBC***, ***ORM***, ***OXM***, ***JMS*** i ***Transaction***.

Modul pod nazivom ***spring-jdbc*** obezbeđuje JDBC nivo apstrakcije kojim je uklonjena potreba za napornim JDBC programiranjem i parsiranjem specifičnih kodova grešaka karakterističnih za sisteme baza podataka različitih proizvođača.

Modul ***spring-tx*** daje podršku programskom i deklarativnom upravljanju transakcijama za klase koje implementiraju specifične interfejse, kao i za sve POJO (Plain Old Java Objects).

Posebno značajno za rad sa bazama podataka, modul ***spring-orm*** obezbeđuje nivoe integracije za sve popularne API za objektno - relaciono mapiranje (object-relational mapping APIs), uključujući *JPA*, *JDO* i *Hibernate*. Primenom ovog modula moguće je koristiti svaki od navedenih okvira za objektno - relaciono mapiranje u kombinaciji sa svim ponuđenim Spring alatima, poput jednostavnog deklarativnog upravljanja transakcijama, koji je prethodno naveden.

Modul ***spring-oxm*** obezbeđuje nivo apstrakcije za podršku implementacijama *Object/XML* mapiranja, kao što su: *JAXB*, *Castor*, *XMLBeans*, *JiBX* i *XStream*.

Specijalno, modul ***spring-jms*** (*Java Messaging Service*) sadrži alate za kreiranje i potrošnju poruka u Java sistemima. Počevši od verzije Spring 4.1, ovaj modul obezbeđuje integraciju sa modulom ***spring-messaging***.

WEB I TEST MODULI

Web nivo, Spring 5 okvira, sadrži veoma značajne module za kreiranje i održavanje veb aplikacija.

Web nivo, Spring 5 okvira, sadrži veoma značajne module za kreiranje i održavanje veb aplikacija, poput: ***spring-web***, ***spring-webmvc***, ***spring-websocket***.

Modul ***spring-web*** sadrži osnovne veb orijentisane (web-oriented) alate poput funkcionalnosti deljenog postavljanja datoteka (multipart file upload) i inicijalizacije IoC kontejnera pomoću ***Servlet*** osluškivača i veb orijentisanog konteksta aplikacije. Takođe, obaj modul sadrži *HTTP klijent* i Spring veb alate za udaljenu podršku.

Modul ***spring-webmvc*** (takođe poznat kao Web-Servlet modul) sadrži implementaciju Springovih *model - view - controller* (MVC) i *REST veb servira* za kreiranje i održavanje veb aplikacija. *Spring MVC* okvir obezbeđuje čisto razdvajanje između koda modela domena i veb formi i dozvoljava integraciju sa svim ostalim alatima Spring okvira.

Posebno, ***spring-test*** modul obezbeđuje mogućnost jediničnog (unit testing) i integracionog (integration) testiranja Spring komponenata primenom okvira *JUnit* i *TestNG*. Modul obezbeđuje konzistentno učitavanje Spring konteksta aplikacije (***ApplicationContext***) i keširanje ovog konteksta. Modul, takođe, obezbeđuje takozvane *mock objekte* koji mogu biti upotrebljeni za testiranje kreiranog koda u izolaciji.

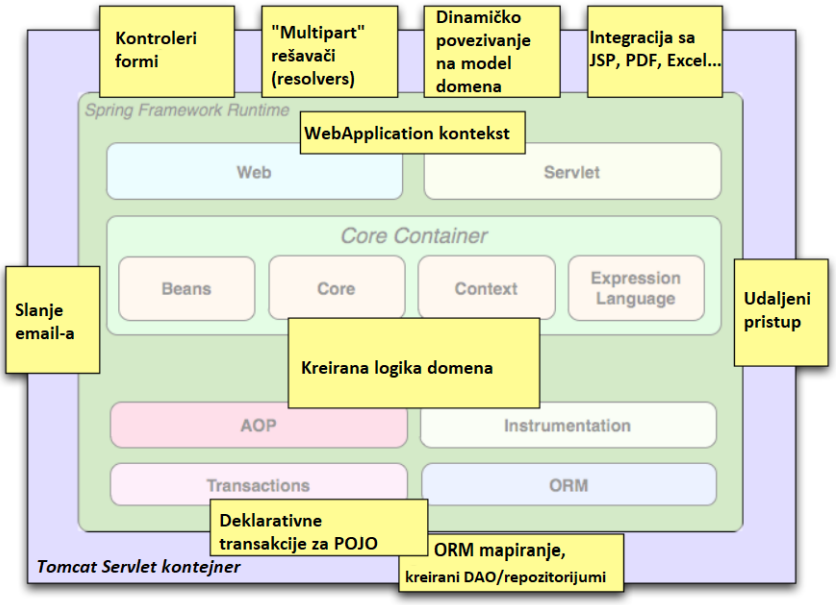
Poglavlje 3

Scenariji upotrebe okvira Spring 5

ELEMENTI SLOŽENE SPRING APLIKACIJE

Spring okvir je logičan izbor za kreiranje velikog broja različitih Java veb aplikacija.

Gradivni blokovi Spring aplikacije, koji su pomenuti u prethodnom izlaganju, čine Spring okvir logičnim izborom za kreiranje velikog broja različitih Java veb aplikacija. Ove aplikacije mogu biti jednostavne aplikacije koje su ugrađene u uređaje sa ograničenim resursima, a mogu biti veoma složene samostalne aplikacije za servisiranje korporacijskog poslovanja koje koriste napredne Spring funkcionalnosti upravljanja transakcijama i integrisanje veb okvira. Sledećom slikom je prikazan opšta organizacija kompleksne Java aplikacije kreirane *Spring5* okvirom.



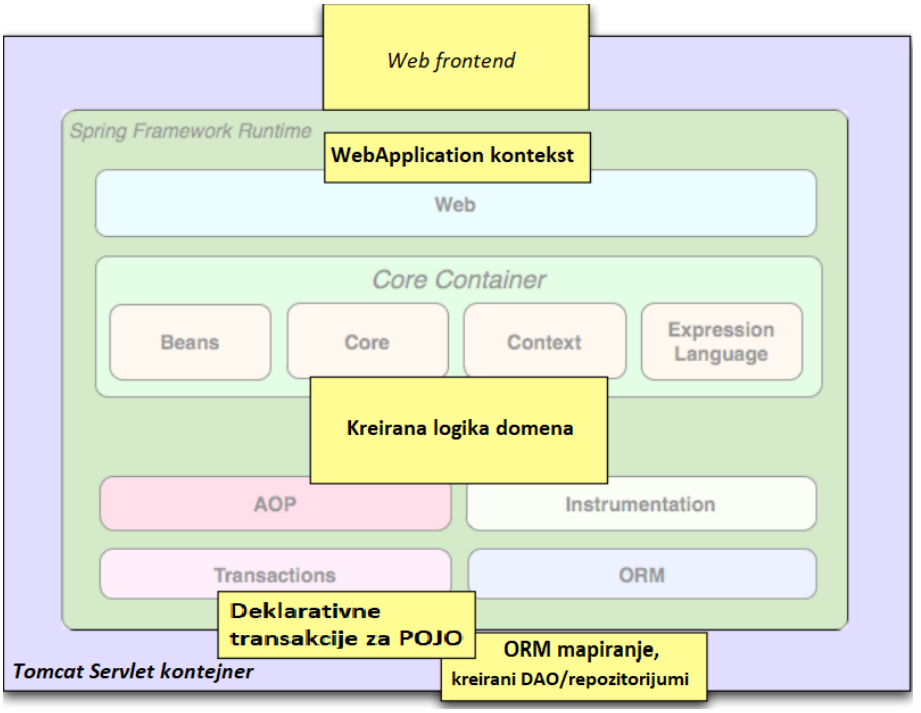
Slika 3.1.1 Složena, sveobuhvatna, Spring 5 aplikacija[izvor: Spring]

Spring alati za deklarativno upravljanje transakcijama čine veb aplikaciju u potpunosti transakcionom kao što bi bila kada bi koristili nativnu JAVA EE platformu i EJB (*Enterprise Java Beans*) kontejner upravljan transakcijama. Kompletna poslovna logika može biti kreirana kao jednostavni POJO objekti i upravljana Spring IoC kontejnerom. Dopunski servisi uključuju mogućnost slanja email poruka i validaciju koja je nezavisna od veb nivoa što omogućava slobodan izbor gde će pravilo validacije biti izvršeno. Spring ORM podrška je integrisana sa JPA (*Java Persistence API*), JDO (*Java Data Objects*) i *Hibernate*. Na primer, kada se koristi *Hibernate* okvir za *ORM* moguće je koristiti postojeće fajlove za mapiranje i standardne

Hibernate ***SessionFactory*** konfiguracije. Kontroleri formi jednostavno integrišu veb nivo i model domena uklanjajući potrebu za ***ActionForms*** i drugim klasama koje transformišu HTTP parametre u vrednosti modela domena.

SPRING 5 SREDNJI SLOJ I VEB OKVIR "TREĆE STRANE"

Spring okvir ne uslovljava programera da koristi sve što nudi ovaj okvir.



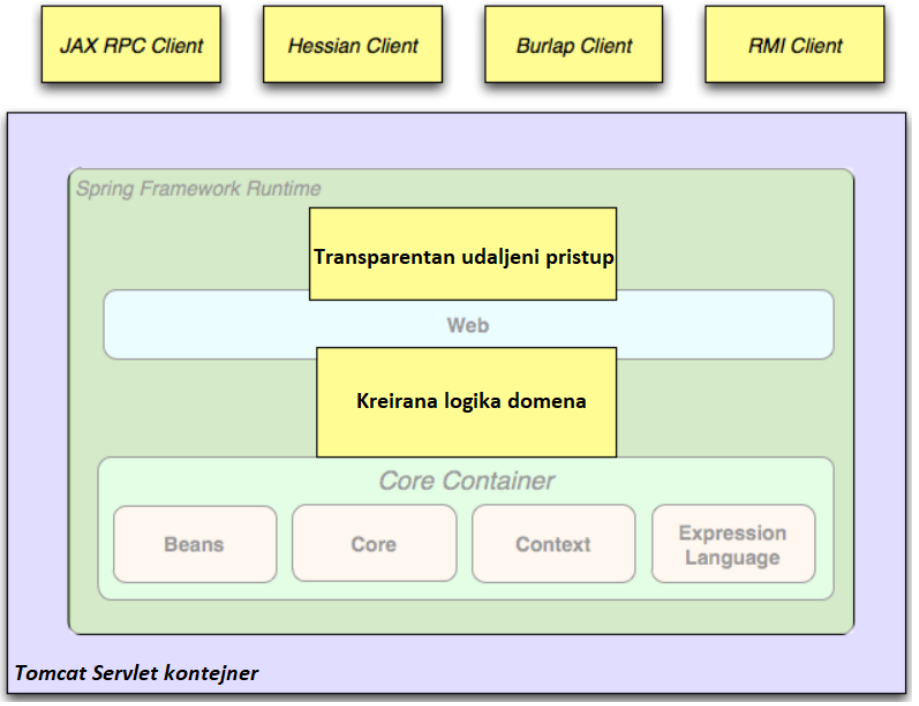
Slika 3.1.2 Spring 5 srednji sloj i veb okvir "treće strane" [izvor: Spring]

U brojnim scenarijima razvoja veb aplikacija nije moguće u potpunosti primeniti alate i tehnike iz drugog razvojnog okvira. Spring okvir ne uslovljava programera da koristi sve što nudi ovaj okvir. Razne klijent stranice (frontend) izgrađene različitim okvirima, poput: *JSF*, *Struts*, *Tapestry* i ostalih UI okvira, mogu biti bez ikakvih problema integrisane u srednji nivo (middle tier) baziran na Springu. Na ovaj način je omogućena upotreba alata Spring transakcija. Neophodno je samo povezati aktuelnu poslovnu logiku primenom ***ApplicationContext*** i upotrebiti ***WebApplicationContext*** za integrisanje vlastitog veb nivoa.

SCENARIO UDALJENE UPOTREBE I EJB ZRNA

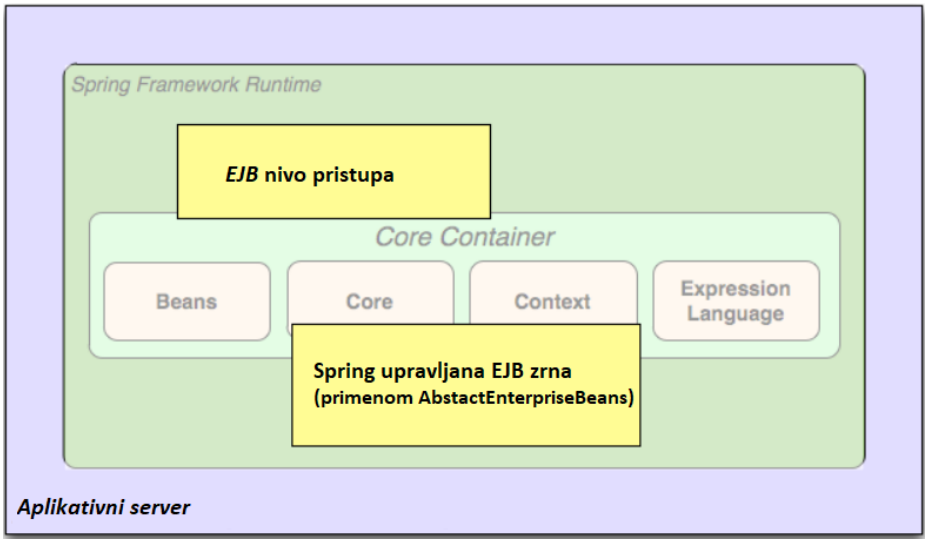
Spring okvir, takođe, obezbeđuje nivo pristupa i apstrakcije za EJB (Enterprise Java Beans).

Prilikom korišćenja mehanizama za pristup postojećem kodu putem veb servisa, moguće je koristiti brojne korisne Spring klase, poput: ***Hessian***-, ***Burlap***-, ***Rmi***- ili ***JaxRpcProxyFactory***. Sam proces omogućavanja udaljenog pristupa za postojeću aplikaciju,u Springu, je veoma jednostavan.



Slika 3.1.3 Scenario udaljenog pristupa [izvor: Spring]

Spring okvir, takođe, obezbeđuje nivo pristupa i apstrakcije (access and abstraction layer) za *EJB* (Enterprise Java Beans). Na ovaj način je omogućena višestruka upotreba postojećih POJO objekata i njihovo umotavanje u *EJB* zrna sesije bez stanja za upotrebu u skalabilnim, stabilnim veb aplikacijama koje mogu da zahtevaju deklarativno bezbednost.



Slika 3.1.4 EJB - umotavanje postojećih POJO [izvor: Spring]

3.1 Upravljanje zavisnošću i konvencija naziva

UPRAVLJANJE ZAVISNOŠĆU I UMETANJE ZAVISNOSTI

Upravljanje zavisnošću i umetanje zavisnosti su dva različita Spring koncepta.

Upravljanje zavisnošću i umetanje zavisnosti su dva različita Spring koncepta. Da bi svi alati, neophodni za razvoj Spring aplikacije, bili dostupni u projektu koji se kreira, potrebno je obezbediti sve neophodne biblioteke (u formi jar datoteka), koje će biti iskorišćene tokom prevođenja aplikacije. Ove zavisnosti nisu virtuelne komponente koje se umeću (inject) već predstavljaju fizičke resurse u sistemu datoteka projekta (file system). Proces upravljanja zavisnostima uključuje lociranje ovih resursa, njihovo čuvanje i dodavanje u putanju klasa (class path) . Zavisnosti mogu biti direktne (na primer aplikacija zavisi od Spring okvira tokom vremena izvršavanja) ili indirektne (na primer aplikacija zavisi od komponente *commons-dbcp* koja zavisi od *commons-pool*). Indirektne zavisnosti su još poznate kao "*tranzitivne*" i ovakve zavisnosti je najteže identifikovati upravljati njima.

Ukoliko se koristi Spring za razvoj neke Java veb aplikacije, neophodno je obezbediti kopije jar biblioteka koje sadrže delove Springa koji je neophodan za razvoj konkretne programske logike. Da bi ovaj proces bio dodatno olakšan, Spring obezbeđuje set modula koji razdvajaju zavisnosti u najvećoj mogućoj meri. Tako, ako se želi da izvrši kreiranje Spring aplikacije koja nije veb aplikacija, nije potrebno koristiti *spring-web* modul. Za isticanje odgovarajuće Spring biblioteke neophodno je poštovati konvenciju naziva. Nazivi biblioteka su dati u opštem obliku spring-* ili spring-*.jar (na primer ***spring-core***, ***spring-webmvc***, ***spring-jms*** i tako dalje). Naziv jar datoteke, koja se koristi, pored naziva obično sadrži i broj verzije (na primer *spring-core-5.1.0.RELEASE.jar*).

Svaka verzija Spring okvira publikuje aritfakte (artifacts) na neki od sledećih načina:

- *Maven Central* - centralni repozitorijum kojeg Maven pretražuje i koji ne zahteva bilo kakvu posebnu konfiguraciju. Većina opštih biblioteka od kojih zavisi Spring je dostupna iz Maven Central repozitorijuma. Otuda, upotreba Maven alata za upravljanje zavisnostima je u širokoj primeni. Nazivi jar datoteka se ovde sreću u formi *spring-*-<version>.jar*, a Maven groupId je *org.springframework*;
- *Javni Maven repozitorijum* postavljen posebno za Spring. Nazivi datoteka u potpunosti odgovaraju nazivima iz Maven Central repozitorijuma. Ovaj repozitorijum, takoše, sadrži i paket sa distribucionim zip fajlom koji sadrži sve Spring jar datoteke okupljene na jednom mestu zbog lakšeg preuzimanja.

PREGLED SPRING ARTIFAKT DATOTEKA

Pod artifaktom se podrazumeva jar fajl koji alat za upravljanje zavisnostima dodaje u projekat.

Pod *artifaktom* se podrazumeva jar datoteka koju alat za upravljanje zavisnostima dodaje u aktuelni projekat jer sadrži alate neophodne za njegovo kreiranje.

Dakle, prva stvar, prilikom razvoja Spring aplikacije, koju je neophodno uraditi jeste izbor načina za upravljanje zavisnostima. Obično se predlaže izbor nekog automatizovanog sistema za upravljanje zavisnostima, poput: *Maven* - a, *Gradle* - a ili *Ivy* - a. Takođe, moguće je manuelno preuzeti sve neophodne jar datoteke i dodati ih u folder biblioteka u aktuelnom projektu.

Sledi lista Spring artifakt datoteka sa odgovarajućim opisom.

GroupId	ArtifactId	Opis
org.springframework	spring-aop	Proxy bazirana AOP podrška
org.springframework	spring-aspects	AspectJ bazirani aspekti
org.springframework	spring-beans	Podrška za zrna
org.springframework	spring-context	Kontekst vremena izvršenja aplikacije uključujući apstrakciju za raspoređivanje i udaljeni pristup
org.springframework	spring-context-support	Klase za podršku integrisanju uobičajenih biblioteka "treće strane" u kontekst Spring aplikacje
org.springframework	spring-core	Alati Spring jezgra
org.springframework	spring-expression	Spring Expression Language (SpEL)
org.springframework	spring-instrument	Agent instrumentalizacije za JVM
org.springframework	spring-instrument-tomcat	Agent instrumentalizacije za Tomcat
org.springframework	spring-jdbc	Paket podrške za JDBC
org.springframework	spring-jms	Paket podrške za JMS
org.springframework	spring-messaging	Paket za arhitekturu sistema poruka i protokola
org.springframework	spring-orm	Paket za ORM mapiranje uključujući JPA i Hibernate podršku
org.springframework	spring-oxm	Object / XML mapiranje
org.springframework	spring-test	Podrška za jedinično i integraciono testiranje Spring komponenata
org.springframework	spring-tx	Infrastruktura transakcija koja uključuje DAO podršku
org.springframework	spring-web	Osnovna podrška za razvoj veb aplikacija
org.springframework	spring-webmvc	Podrška za razvoj MVC aplikacija
org.springframework	spring-webmvc-portlet	MVC implementacija za portlet okruženje
org.springframework	spring-websocket	WebSocket podrška za Spring

Slika 3.2.1 Artifakti Spring okvira [izvor: autor]

UPRAVLJANJE ZAVISNOSTIMA PRIMENOM MAVEN - A

Spring namerno drži neophodne zavisnosti na apsolutnom minimumu.

Iako Spring obezbeđuje integraciju i podršku širokom spektru alata za razvoj Java aplikacija, namerno drži obavezne zavisnosti na apsolutnom minimumu. To praktično znači da nije potrebno lociranje i preuzimanje (čak i automatsko) velikom broja jar biblioteka sa ciljem njihove upotrebe nad jednostavnim Spring primerima. Za osnovno umetanje zavisnosti neophodna je samo jedna obavezna eksterna zavisnost koja se odnosi na logovanje (logging).

U sledećem izlaganju će biti istaknuti osnovni koraci neophodni za podešavanje aplikacije koja zavisi od Springa, upotrebom sistema kao što su Maven, a zatim i Gradle.

Ukoliko se koristi Maven alat za upravljanje zavisnostima, nije potrebno čak ni eksplicitno dodavanje zavisnosti za logovanje. Na primer, za kreiranje konteksta aplikacije i primenu umetanja zavisnosti za podešavanje aplikacije, Maven zavisnosti će izgledati na način koji je identičan priloženom XML kodu:

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.9.RELEASE</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Prethodni kod se odnosi na upravljanje zavisnostima iz *Maven Central* repozitorijuma. Ukoliko je cilj upotreba *Spring Maven* repozitorijuma (na primer za prekretnice (milestones) i snimke koda programera (developer snapshots) neophodno je specificirati lokaciju repozitorijuma u Maven podešavanjima. Za kompletnu Spring podršku navodi se sledeće:

```
<repositories>
    <repository>
        <id>io.spring.repo.maven.release</id>
        <url>http://repo.spring.io/release/</url>
        <snapshots><enabled>false</enabled></snapshots>
    </repository>
</repositories>
```

Za prekretnice:

```
<repositories>
    <repository>
        <id>io.spring.repo.maven.milestone</id>
        <url>http://repo.spring.io/milestone/</url>
        <snapshots><enabled>false</enabled></snapshots>
    </repository>
</repositories>
```

Za snapshot:

```
<repositories>
    <repository>
        <id>io.spring.repo.maven.snapshot</id>
        <url>http://repo.spring.io/snapshot/</url>
        <snapshots><enabled>true</enabled></snapshots>
    </repository>
</repositories>
```

MAVEN "BILL OF MATERIALS" ZAVISNOSTI - BOM

Maven daje punu podršku konceptu "Bill Of Materials" zavisnosti za kontrolu verzije zavisnosti.

Moguće je, slučajno, pomešati različite verzije Spring jar datoteka prilikom korišćenja Maven alata. Na primer, moguće je sresti slučaj da je biblioteka "treće strane", ili neki drugi Spring projekat, uvučen u tranzitivnu zavisnost sa nekom starijom verzijom. Ukoliko je zaboravljeno

eksplicitno deklarisanje direktne zavisnosti, svi tipovi neočekivanih problema mogu se javiti sa većim stepenom verovatnoće.

Za prevazilaženje ovakvih problema Maven daje punu podršku konceptu "*Bill Of Materials*" zavisnosti (*BOM*). U ovom slučaju je moguće uvesti artifakt *spring-framework-bom* u sekciju ***dependencyManagement*** za obezbeđivanje da sve Spring zavisnosti (direktne ili tranzitivne) pripadaju istoj verziji.

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-framework-bom</artifactId>
            <version>4.3.9.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Dobra strana primene BOM koncepta jeste da njegovom primenom više nije potrebno specificirati *<version>* atribut kada aplikacija zavisi od artifakata Spring okvira:

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
    </dependency>
<dependencies>
```

GRADLE UPRAVLJANJE ZAVISNOSTIMA

Upotreba Spring repozitorijuma zahteva navođenje odgovarajućeg URL u sekciji repositories.

Kada se koristi *Gradle* alat za kreiranje aplikacije i upravljanje zavisnostima, upotreba *Spring repozitorijuma* zahteva navođenje odgovarajućeg URL u sekciji repositories. Navedeno je realizovano sledećim listingom:

```
repositories {
    mavenCentral()
    // and optionally...
    maven { url "http://repo.spring.io/release" }
}
```

Kada je repozitorijum podešen, zavisnosti se deklarišu na poznat *Gradle* način. Primer dodavanja zavisnosti je priložen sledećim listingom:

```
dependencies {
    compile("org.springframework:spring-context:5.1.0.RELEASE")
    testCompile("org.springframework:spring-test:5.1.0.RELEASE")
}
```

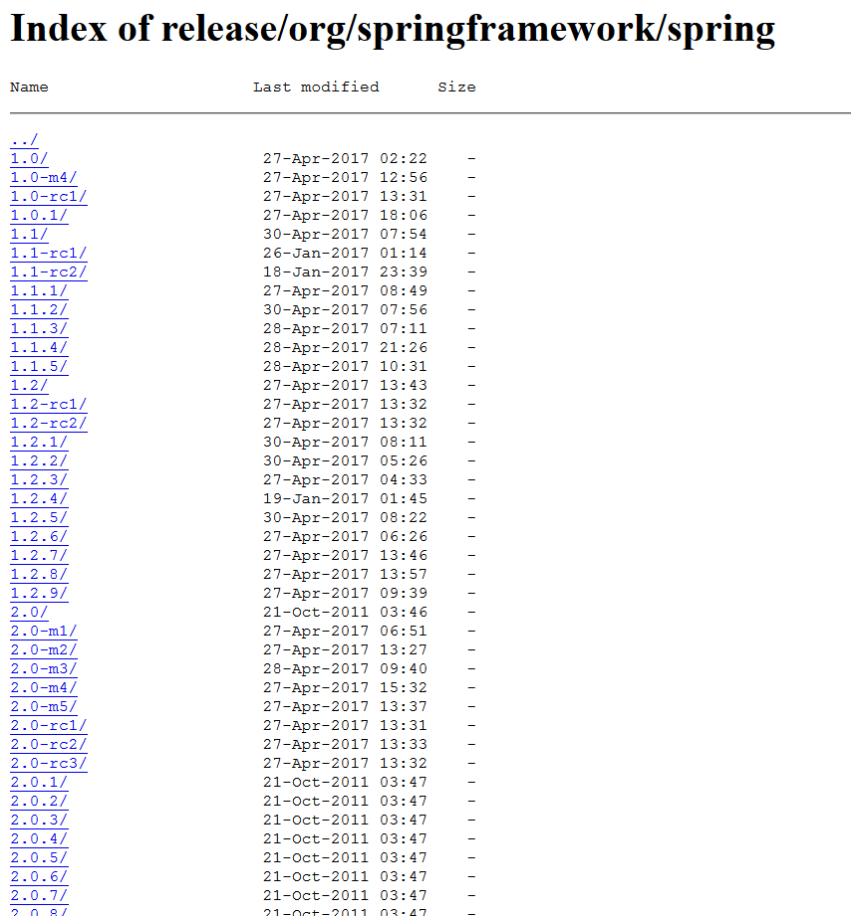
DISTRIBUCIJA ZIP DATOTEKA

Alate Spring okvira je moguće obezbediti i preuzimanjem distribucionog zip fajla.

Iako je primena sistema za kreiranje aplikacija i upravljanje zavisnostima preporučeni način obezbeđivanja alata Spring okvira, moguće je ove alate obezbediti i preuzimanjem distribucionog zip fajla.

Distribucioni zip fajlovi su objavljeni u *Spring Maven Repository* i ne zahtevaju primenu Maven alata, ili bilo kojeg drugog sličnog, za njihovo preuzimanje.

Za preuzimanje distribucione zip datoteke neophodno je otvoriti veb čitač i uneti URL : http://repo.spring.io/release/org/springframework/spring . Nakon toga je neophodno izabrati folder koji odgovara verziji koju je potrebno preuzeti. Nazivi ovih datoteka su određeni tako da se završavaju sa *-dist.zip*, na primer *spring-framework-{spring-version}-RELEASE-dist.zip*. Distribucije su takođe objavljene za kategorije milestones i snapshots.



Index of release/org/springframework/spring

Name	Last modified	Size
../		
1.0/	27-Apr-2017 02:22	-
1.0-m4/	27-Apr-2017 12:56	-
1.0-rc1/	27-Apr-2017 13:31	-
1.0.1/	27-Apr-2017 18:06	-
1.1/	30-Apr-2017 07:54	-
1.1-rc1/	26-Jan-2017 01:14	-
1.1-rc2/	18-Jan-2017 23:39	-
1.1.1/	27-Apr-2017 08:49	-
1.1.2/	30-Apr-2017 07:56	-
1.1.3/	28-Apr-2017 07:11	-
1.1.4/	28-Apr-2017 21:26	-
1.1.5/	28-Apr-2017 10:31	-
1.2/	27-Apr-2017 13:43	-
1.2-rc1/	27-Apr-2017 13:32	-
1.2-rc2/	27-Apr-2017 13:32	-
1.2.1/	30-Apr-2017 08:11	-
1.2.2/	30-Apr-2017 05:26	-
1.2.3/	27-Apr-2017 04:33	-
1.2.4/	19-Jan-2017 01:45	-
1.2.5/	30-Apr-2017 08:22	-
1.2.6/	27-Apr-2017 06:26	-
1.2.7/	27-Apr-2017 13:46	-
1.2.8/	27-Apr-2017 13:57	-
1.2.9/	27-Apr-2017 09:39	-
2.0/	21-Oct-2011 03:46	-
2.0-m1/	27-Apr-2017 06:51	-
2.0-m2/	27-Apr-2017 13:27	-
2.0-m3/	28-Apr-2017 09:40	-
2.0-m4/	27-Apr-2017 15:32	-
2.0-m5/	27-Apr-2017 13:37	-
2.0-rc1/	27-Apr-2017 13:31	-
2.0-rc2/	27-Apr-2017 13:33	-
2.0-rc3/	27-Apr-2017 13:32	-
2.0.1/	21-Oct-2011 03:47	-
2.0.2/	21-Oct-2011 03:47	-
2.0.3/	21-Oct-2011 03:47	-
2.0.4/	21-Oct-2011 03:47	-
2.0.5/	21-Oct-2011 03:47	-
2.0.6/	21-Oct-2011 03:47	-
2.0.7/	21-Oct-2011 03:47	-
2.0.8/	21-Oct-2011 03:47	-

Slika 3.2.2 Lokacija za preuzimanje distribucioni Spring datoteka [izvor: autor]

3.2 Logovanje

LOGGING ZAVISNOST

Logovanje je jedina obavezna spoljna zavisnost

Logovanje (Logging) je veoma važna zavisnost za Spring zbog sledećih razloga:

- to je jedina obavezna spoljna zavisnost;
- prikazuje izlaz iz alata koji se koriste;
- Spring je integrisan sa brojnim alatima od kojih svi mogu da zahtevaju logovanje kao zavisnost.

Jedan od glavnih ciljeva programera, koji razvija Spring aplikaciju, obično može biti potreba za unificiranom konfiguracijom logovanja, na jednom mestu za celu aplikaciju, uključujući i spoljašnje komponente. Ovaj zadatak se dodatno komplikuje budući da postoji veliki izbor okvira (frameworks) za logovanje.

Glavnu zavisnost za logovanje u Springu predstavlja *Jakarta Commons Logging API (JCL)*. Kada se vrši kompajliranje pomoću JCL, JCL Log objekti postaju vidljivi za klase koje proširuju

(extends) Spring okvir. Veoma je važno da korisnici znaju da sve Spring verzije koriste istu biblioteku logovanja. Zbog toga je migracija veoma jednostavna iz prostog razloga što je kompatibilnost unazad (kompatibilnost sa prethodnim verzijama Spring okvira) očuvana čak i kada se radi sa aplikacijama koje proširuju Spring. Navedeno se postiže tako što se jedan od modula u Springu učini eksplicitno zavisnim od ***commons-logging*** (kanonska implementacija JCL), a potom svi ostali moduli se učine zavisnim od navedenog modula tokom vremena prevođenja aplikacije.

Na primer, ukoliko se koristi Maven za upravljanje zavisnostima, i kada se bira koji Spring modul je pogodan za povezivanje sa ***commons-logging***, onda bi to trebalo da bude centralni modul ***spring-co***re.

Dobra stvar u vezi sa commons-logging jeste da ništa više nije potrebno da bi se aplikacija učinila funkcionalnom. Ova zavisnost poseduje algoritam za detekciju, tokom vremena izvršavanja, za prepoznavanje svih ostalih okvira logovanja, na dobro poznatom mestu u putanji klasa (classpath) i koristi upravo onaj za koji smatra da je odgovarajući.

Ukoliko nijedan okvir logovanja nije dostupan, logovanje će biti dostupno iz JDK - a primenom ***java.util.logging*** ili kraće *JUL*.

LOG4J 1.2 ILI 2.X

Log4j 1.2 se napušta i menja se novijim Log4j 2.x izdanjima za platforme Java 7+.

> Log4j 1.2 u međuvremenu je stigao na kraj vlastitog životnog veka. Takođe, Log4j 2.3 predstavlja poslednje izdanje kompatibilno sa Java 6 platformom, sa novijim Log4j 2.xizdanjima za platformeJava 7+.

Dosta programera koristi ***Log4j*** kao okvir logovanja za svrhe podešavanja i upravljanja. Ovaj okvir je efikasan, dobro osmišljen i to je zapravo ono što se koristi tokom vremena izvršavanja (run - time) kada se prevodi Spring. Spring takođe obezbeđuje izvesne alate za podešavanje i inicijalizaciju okvira ***Log4j***, tako da poseduje opcione zavisnosti, tokom vremena prevođenja (compile - time), na ***Log4j*** u pojedinim modulima.

Da bi ***Log4j 1.2*** radio sa podrazumevanim *JCL* zavisnostima (***commons-logging***) sve što je potrebno jeste smestiti ***Log4j*** u putanju klasa (classpath) i obezbediti ga izvesnim konfiguracionim fajlom (*log4j.properties* ili *log4j.xml*) u korenu putanje klasa (classpath root). Za Maven korisnike ovako izgleda deklarisanje zavisnosti:

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.1.0.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
```

```
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
</dependencies>
```

Sledi primer prikaza log4j.properties u konzoli:

```
log4j.rootCategory=INFO, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %t %c{2}:%L - %m%n

log4j.category.org.springframework.beans.factory=DEBUG
```

Slika 3.3.1 Prikaz log4j.properties u konzoli (izvor: autor)

Za upotrebu***Log4j 2.x*** sa *JCL* sve što je potrebno jeste smestiti ***Log4j*** u putanju klasa (classpath) i obezbediti ga izvesnim konfiguracionim fajlom (*log4j2.properties*, *log4j2.xml* ili nekim drugim podržanim formatom za konfigurisanje). Za Maven korisnike, minimum potrebnih zavisnosti izgleda ovako:

```
<dependencies>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-core</artifactId>
        <version>2.6.2</version>
    </dependency>
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-jcl</artifactId>
        <version>2.6.2</version>
    </dependency>
</dependencies>
```

LOG4J 1.2 ILI 2.X - DODATNA RAZMATRANJA

SLF4J služi kao jednostavna fasada - omotač ili apstrakcija za različite okvire logovanja.

The Simple Logging Facade for Java (SLF4J) služi kao jednostavna fasada - omotač ili apstrakcija za različite okvire logovanja (na primer: java.util.logging, logback, log4j) omogućujući krajnjem korisniku priključivanje na željeni okvir logovanja tokom vremena razvoja (videti detaljnije https://www.slf4j.org/).

Ukoliko je cilj da se omogući da *SLF4J* delegira***Log4j***, na primer za druge biblioteke koje koriste *SLF4J*, po osnovnim podešavanjima, sledeća zavisnost je takođe neophodna:

```
<dependencies>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
```

```
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.6.2</version>
  </dependency>
</dependencies>
```

Sada je moguće izvršiti i prilaganje primera fajla *log4j2.xml* za prikazivanje log informacija na konzoli:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Logger name="org.springframework.beans.factory" level="DEBUG"/>
    <Root level="error">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

IZBEGAVANJE COMMONS-LOGGING

Algoritam za detekciju logging okvira je pogodan za krajnjeg korisnika, ali može imati i problematičnu upotrebu u pojedinim situacijama.

Ono što je važno napomenuti u sledećem izlaganju jeste činjenica da je pomenuti algoritam za detekciju, tokom vremena izvršavanja, pripada ***commons-logging*** API paketu. Iako je pogodan za krajnjeg korisnika, u izvesnim situacijama može biti problematičan za upotrebu. Ukoliko je potrebno da se izbegne standardni JCL pristup, moguće je primeniti jedan dobro poznat pristup koji podrazumeva : isključivanje zavisnosti iz spring-core modula (budući da je ovo jedini modul koji eksplicitno zavisi od (commons-logging).

Za isključivanje zavisnosti ***commons-logging*** neophodno je dodati sledeće linije koda u ***dependencyManagement*** sekciju:

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>4.3.9.RELEASE</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
```

```
    </dependency>
</dependencies>
```

U ovom trenutku bi došlo do otkazivanja aplikacije, budući da ne postoji implementacija JCL API u putanji klasa (classpath). Da bi problem bio prevaziđen neophodno je obezbediti alternativnu implementaciju za logovanje. Upravo će to biti zadatak u narednom izlaganju - obezbeđivanje alternativne implementacije za JCL primenom SLF4J.

UPOTREBA SLF4J SA LOG4J ILI LOGBACK

SLF4J je popularan API koji je u širokoj upotrebi od strane drugih biblioteka koje Spring koristi.

The Simple Logging Facade for Java (SLF4J) je popularan API koji je u širokoj upotrebi od strane drugih biblioteka koje se najčešće koriste u Spring okviru. Veoma se često koristi sa *Logback* bibliotekom koja predstavlja prirodnu implementaciju SLF4J API.

SLF4J obezbeđuje mehanizme povezivanja sa brojnim opšte korišćenim okvirima za logovanje, uključujući među njima i ***Log4j***. Takođe, omogućava i obrnuto. *SLF4J* predstavlja vezu između drugih okvira logovanja i samog sebe. Zbog svega navedenog, da bi bilo moguće koristiti *SLF4J*, prilikom razvoja Spring aplikacija, neophodno je zameniti ***commons-logging*** zavisnost za zavisnošću ***SLF4J-JCL***. Kada se ovo jednom uradi, svi log pozivi u okviru Spring aplikacije biće preusmereni ka *SLF4J* API. Ukoliko druge biblioteke, u posmatranoj aplikaciji, koriste navedeni API, omogućeno je jedinstveno mesto za podešavanje i upravljanje procesom logovanja.

U opštem slučaju, moguće je povezati Spring sa *SLF4J* i, nakon toga, obezbediti eksplicitno povezivanje iz *SLF4J* ka ***Log4j***. U tu svrhu je neophodno obezbediti nekoliko zavisnosti (i isključiti postojeću commons-logging) za: JCL premošćavanje (JCL bridge), povezivanje SLF4j sa Log4j i Log4j provajdera.

Primenom Maven alata za upravljanje zavisnostima, moguće je navedeni zadatak obaviti na sledeći način:

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.1.0.RELEASE</version>
        <exclusions>
            <exclusion>
                <groupId>commons-logging</groupId>
                <artifactId>commons-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
```

```
        <artifactId>jcl-over-slf4j</artifactId>
        <version>1.7.21</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-log4j12</artifactId>
        <version>1.7.21</version>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
</dependencies>
```

Često se, među *SLF4J* korisnicima, kao izbor nameće pristup koji uključuje manje koraka i generiše manje zavisnosti. Ovaj pristup podrazumeva direktno povezivanje na *Logback*. Na ovaj način, kao što je i pomenuto, eliminišu se dopunski koraci u povezivanju zavisnosti budući da Logback direktno implementira *SLF4J*. Ovde je potrebno angažovati samo dve zavisnosti: ***jcl-over-slf4j*** i ***logback***.

```
<dependencies>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
        <version>1.7.21</version>
    </dependency>
    <dependency>
        <groupId>ch.qos.logback</groupId>
        <artifactId>logback-classic</artifactId>
        <version>1.1.7</version>
    </dependency>
</dependencies>
```

ŠTA JE SPRING? (VIDEO MATERIJAL)

Sledi prigodan materijal kao kratka definicija Spring okvira.

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

Poglavlje 4

Verzije okvira - Spring 4

NOVE STAVKE I UNAPREĐENJA U SPRING 4.0 OKVIRU

Spring okvir je doživeo nekoliko značajnih glavnih (major) revizija.

Spring okvir je doživeo svoje prvo izdanje u 2004. godini. Od tog datuma Spring okvir je doživeo nekoliko značajnih glavnih (major) revizija:

- Spring 2.0 - obezbeđuje XML prostore naziva (namespaces) i AspectJ podršku;
- Spring 2.5 - uvodi podršku konfiguraciji baziranoj na anotacijama;
- Spring 3.0 - uvodi snažnu Java 5+ osnovu nad bazom koda (codebase) celokupnog okvira i alate poput Java baziranog @Configuration modela;
- Spring 4.0 - predstavlja poslednju glavnu reviziju Spring okvira koja u potpunosti podržava Java 8 alate. Takođe, ovu verziju Spring okvira je moguće koristiti i sa starijim verzijama Jave. Minimum zahteva za korišćenje ove generacije Spring okvira predstavlja instalirana Java SE 6. Primenom ovog Spring okvira dobijena je mogućnost uklanjanja brojnih zastarelih (deprecated) klasa i metoda.

Priručnik za migraciju sa starih verzija Spring okvira na verziju Spring 4.0, detaljno je izložen na stranici Spring Framework GitHub Wiki.

Sledećim video materijalima je opisan istorijski razvoj Spring okvira.

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.

PREGLED NOVITETA I UNAPREĐENJA OKVIRA SPRING 4.0

Spring 4.0 okvir unosi brojne novitete i proširenja.

Spring 4.0 okvir unosi brojne novitete i proširenja koja je moguće globalno sagledati kroz sledeće teze:

- Unapređeno iskustvo učenja za početnike - Potpuno novi spring.io veb sajt obezbeđuje čitav niz vodiča, sa dobro poznatim nazivom "Getting Started" , za početak učenja

Spring okvira. Takođe, ovaj novi veb sajt obezbeđuje značajnu podršku za pregled brojnih dodatih projekata koji su objavljeni pod Spring dokumentacijom;

- Uklonjene zastarele klase i metode - Svi prevaziđeni paketi, brojne prevaziđene klase i metode su uklonjene sa pojavom okvira Spring 4.0. Ukoliko se obavlja ažuriranje iz neke starije verzije Spring okvira, programerima bi trebalo da bude jasno da su upravo izvršene korekcije koje onemogućavaju deprecirane pozive. Za kompletan spisak izmena neophodno je proveriti dokument pod nazivom *Spring 4.0 API Differences Report*. Ovde bi još trebalo dodati da je primena svih zavisnosti "treće strane" određena 2010 / 2011. minimumom. To praktično znači da Spring 4 podržava verzije objavljene s kraja 2010. i kasnije. Na primer, sledeći okviri imaju minimum kompatibilnosti sa Spring 40 okvirom: Hibernate 3.6+, EhCache 2.1+, Quartz 1.8+, Groovy 1.8+ i Joda-Time 2.0+.

- Java 8 (takođe Java 7 i Java 6) - Omogućeni su brojni Java 8 alati, kao što su lambda izrazi i referenciranje metoda pomoću Spring interfejsa za pozivanje (callback interfaces). Takođe postoji i prvoklasna podrška za *java.time* (JSR-310), nekoliko postojećih anotacija naknadno obeleženih kao ***@Repeatable*** i Java 8 upravljanje parametrima (bazirano na -parameters compiler flag). Spring 4.0 zadržava potpunu kompatibilnost sa starijim verzijama Java programskog jezika, zaključno sa Java SE 6 (konkretno, minimum je verzija JDK 6 update 18, objavljena u januaru 2010). Međutim, jasna preporuka je korišćenje verzija Java 7 i Java 8 u kombinaciji sa Spring okvirom;
- Java EE 6 i Java EE 7 - Ove platforme predstavljaju osnovu Spring 4.0 okvira zajedno sa Java 6 EE podrškom za *JPA 2.0* (Java Persistence API) i *Servlet 3.0* i *Java 7 EE* podrškom za *JMS 2.0, JTA 1.2, JPA 2.1, Bean Validation 1.1* i *JSR-236* alate za konkurentno programiranje (Concurrency Utilities).

Hibernate 4.3 je provajder za JPA 2.1 i kao takav predstavlja jedini Hibernate okvir podržan u Springu nad Java 7 EE platformom.

NAPREDNI NOVITETI I UNAPREĐENJA OKVIRA SPRING 4.0

Spring 4.0 okvir unosi, takođe, brojne napredne novitete i proširenja.

Spring 4.0 okvir unosi, takođe i brojne napredne novitete i proširenja koja je moguće globalno sagledati kroz sledeće teze:

- Groovy DSL za definisanje zrna (Domain Specific Language) - Počevši sa Spring 4.0 okvirom moguće je definisati spoljašnju konfiguraciju zrna primenom specifičnog jezika *Groovy DSL*. Ovo je veoma sličan koncept kao definisanje zrna putem XML konfiguracije pri čemu dozvoljava znatno sažetiju sintaksu. Primena *Groovy DSL*, takođe, omogućava da se definicija zrna direktno proširi, na primer, u bootstrap kodu. Navedeno je ilustrovano sledećim listingom:

```
def reader = new GroovyBeanDefinitionReader(myApplicationContext)
reader.beans {
    dataSource(BasicDataSource) {
```

```
        driverClassName = "org.hsqldb.jdbcDriver"
        url = "jdbc:hsqldb:mem:grailsDB"
        username = "sa"
        password = ""
        settings = [mynew:"setting"]
    }
    sessionFactory(SessionFactory) {
        dataSource = dataSource
    }
    myService(MyService) {
        nestedBean = { AnotherBean bean ->
            dataSource = dataSource
        }
    }
}
```

- Unapređenja osnovnog (core) kontejnera - Spring sada tretira generičke tipove u formi kvalifikatora kada vrši umetanje (injecting) zrna. Na primer ukoliko se koristi Spring Data Repository moguće je lako umetnuti specifičnu implementaciju pomoću:

```
@Autowired
Repository<Customer> customerRepository;
```

- Unapređenja osnovnog (core) kontejnera (nastavak) - moguće je razviti i vlastite anotacije za isticanje specifičnih atributa. Takođe, zrna koja su automatski povezana (autowired) mogu biti organizovana pomoću ***@Order*** anotacije ili ***Ordered*** interfejsa u liste ili nizove. Anotacija ***@Lazy*** može biti primenjena na tačku umetanja, kao i na ***@Bean*** definiciju. Anotacija ***@Description*** dostupna je programerima koji koriste Java bazirana podešavanja. Opšti model za uslovno filtriranje zrna dodat je preko anotacije ***@Conditional***. Ovoje slično ***@Profile*** podrški ali dozvoljava programski razvoj korisnički definisanih strategija. *CGLib* (Byte Code Generation Library) bazirane proxy klase više ne zahtevaju podrazumevane konstruktore. Primenom ove strategije, ne vrši se više pozivanje konstruktora za proxy instance. Kao proširene, takođe se javlja podrška upravljanja vremenskim zonama na celokupnom Spring okviru, na primer za kontekst ***LocaleContext***.

VEB I TEST UNAPREĐENJA

Brojna unapređenja nad veb i test modulima su sada dostupna sa Spring 4.0 okvirom.

U narednom izlaganju sledi nastavak isticanja naprednih noviteta i unapređenja koji su stupili na scenu zajedno sa objavljivanjem verzije okvira Spring 4.0:

- Veb unapređenja - Razvoj aplikacija nad Servlet 2.5 serverima je i dalje dostupan, međutim, Spring okvir 4.0 stavlja glavni fokus na *Servlet 3.0 +* okruženja. Ukoliko se koristi Spring MVC Test okvir biće neophodno da se obezbedi JAR datoteka, u putanji klasa, koja je kompatibilna sa *Servlet 3.0*. Kao dodatak ***WebSocket*** podrški, o kojoj je

delimično bilo govora u prethodnom izlaganju, sledeća unapređenja su učinjena nad veb modulima Spring okvira: moguće je, sada, koristiti ***@RestController*** anotaciju u Spring MVC aplikacijama, uklonjena je potreba za dodavanjem anotacije ***@ResponseBody*** za svaku metodu obeleženu anotacijom ***@RequestMapping***. Takođe, dodatat je i klasa AsyncRestTemplate čiji je zadatak omogućavanje asinhrone, neblokirajuće, podrške za razvoj veb klijenata. Dodata je podrška za rad sa vremenskim zonama prilikom razvoja Spring MVC aplikacija.

- WebSocket, SockJS - Najnoviji spring-websocket modul obezbeđuje značajnu, baziranu na Java EE WebSocket, podršku za dvosmetnu komunikaciju između klijenata i servera u veb aplikacijama. Podrška je kompatibilna sa *JSR-356 (Java WebSocket API)* i dodatno obezbeđuje opcije bazirane na *SockJS* (poput: WebSocket emulacije) za veb pregledače koji ne poseduju podršku za WebSocket protokol (npr. InternetExplorer<10).

- STOMP Messaging - Kao unapređenje javlja se i *WebSocket potprotokol*, pod nazivom *Simple (or Streaming) Text Orientated Messaging Protocol (STOMP)* za upotrebu u aplikacijama, zajedno sa modelom programiranja baziranim na anotacijama, za rutiranje i procesiranje *STOMP* poruka klijenata. Kao rezultat anotacija ***@Controller*** sada može da sadrži obe anotacije ***@RequestMapping*** i ***@MessageMapping*** za rukovanje HTTP zahtevima i porukama klijenata povezanih na ***WebSocket.***
- Test unapređenja - Spring 4.0 uvodi nekoliko novih test alata kao zamenu za zastareli kod prisutan u spring-test modulu. Skoro sve anotacije (npr. ***@ContextConfiguration, @WebAppConfiguration, @ContextHierarchy, @ActiveProfiles***) mogu biti kombinovane i primenjivane u okviru korisnički definisanih anotacija. Aktivni profili za definisanje zrna sada mogu biti rešeni programski, implementiranjem korisničkog ActiveProfilesResolver i njegovim registrovanjem pomoću ***resolver*** atributa anotacije ***@ActiveProfiles***. Dalje, nova klasa ***SocketUtils*** je uvedena u modul spring - core za slobodno skeniranje TCP i UDP serverskih portova na lokalnom računaru (localhost). Na kraju, ažurirani su, u svetlu Servlet 3.0 API, nekoliko Servlet API za simulirane objekte (mock objects), na primer ***MockHttpServletRequest, MockServletContext*** i drugi.

ŠTA JE NOVO U OKVIRU SPRING 4.0 VIDEO MATERIJALI.

Detaljan pregled izmena u Spring okviru nakon velike (major) revizije.

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

Poglavlje 5

Verzije okvira - Spring 4.1

PREGLED NOVITETA I UNAPREĐENJA OKVIRA SPRING 4.1

Spring 4.1 predstavlja manje (minor) ažuriranje okvira Spring 4.0 sa daljim unapređenjima.

Spring 4.1 predstavlja manje (minor) ažuriranje okvira Spring 4.0 sa daljim unapređenjima. Spring 4.1 okvir unosi brojne novitete i proširenja koja je moguće globalno sagledati kroz sledeće teze:

- JMS (Java Messaging System) unapređenja;
- Unapređenja keširanja;
- Dalja veb unapređenja;
- Unapređenja za razmenu poruka za WebSocket (WebSocket Messaging);
- Dalja unapređenja iz domena testiranja.

U daljem izlaganju će posebno biti diskutovana unapređenja iz navedenih oblasti za sagledavanje skupa poboljšanja koja su dostupna programerima i razvojnim timovima koji su prešli na korišćenje okvira Spring 4.1.

JMS (JAVA MESSAGING SYSTEM) UNAPREĐENJA

Pojednostavljena je podrška za registovanje JMS osluškivača primenom anotacije @JmsListener.

Sa verzijom okvira Spring 4.1 predstavljena je značajno pojednostavljena infrastruktura za registrovanje krajnjih tačaka (endpoints) JMS osluškivača obeležavanjem metoda zrna pomoću anotacije ***@JmsListener***. XML prostor naziva (XML namespace) je proširen podrškom za nov stil (***jms:annotation-driven***) i takođe ima sposobnost potpunog podešavanja infrastrukture upotrebom Java konfiguracija (@***EnableJms, JmsListenerContainerFactory***). Takođe, moguće je registrovati krajnje tačke osluškivača u programskom kodu primenom ***JmsListenerConfigurer***.

Spring 4.1 takođe ističe vlastitu JMS podršku za omogućavanje pune prednosti primene spring-messaging apstrakcije predstavljene u Spring 4.0. To praktično znači:

- Krajnje tačke osluškivača poruka poseduju fleksibilnije potpise i više koristi od standardnih anotacija poruka kao što su: ***@Payload, @Header, @Headers i @SendTo***.

Takođe je moguće koristiti i standardnu poruku - Message objekat, umesto ***javax.jms.Message***, kao argument metode.
- Novi interfejs ***JmsMessageOperations*** je dostupan i omogućava ***JmsTemplate*** operacije primenom Message apstrakcije;

Takođe, Spring 4.1 donosi i sledeća dodatna poboljšanja:

- Sinhronizovane zahteve - odgovor operacije podržane u JmsTemplate;
- Prioritet osluškivača može biti određen primenom elemenata <jms:listener/> ;
- Oporavak operacija za kontejner osluškivača poruka je moguće podesiti primenom implementacije BackOff;
- JMS 2.0 deljeni potrošači poruka su podržani.

UNAPREĐENJA KEŠIRANJA

Sa okvirom Spring 4.1 dolazi i podrška JCache.

Sa okvirom Spring 4.1 dolazi i podrška JCache (JSR-107) sa anotacijama koje koriste postojeće konfiguracije Spring keša i apstrakciju infrastrukture. Ovde nisu potrebne bilo kakve izmene za korišćenje za upotrebu standardnih anotacija.

Spring 4.1 takođe značajno unapređuje vlastite apstrakcije keširanja:

- Keširanje može biti rešeno tokom vremena izvršavanja primenom ***CacheResolver***. Kao rezultat se dobija da argument ***value*** koji definiše naziv keša, koji bi trebalo da bude iskorišćen, nije više obavezan;
- Podržano je više prilagođavanja na nivou operacija: *resolver, cache manager, key generator*;
- Nova anotacija ***@CacheConfig***, koja pripada nivou klasa, dozvoljava da opšta podešavanja mogu biti deljena na nivou klasa bez potrebe za obezbeđivanjem bilo koje keš operacije;
- Bolje upravljanje izuzecima keširanih metoda primenom ***CacheErrorHandler***.

Spring 4.1 takođe donosi i značajnu izmenu u interfejsu Cache - dodata je metoda *putIfAbsent()*.

Sledećim video materijalom je dat pregled unapređenja iz JMS i keširanja koje donosi revizija Spring 4.1.

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

SPRING 4.1 VEB UNAPREĐENJA

Postojeća Spring 4.0 podrška za veb značajno je proširena uvođenjem Spring 4.1 ažuriranja.

Postojeća podrška za upravljanje resursima, bazirana na ***ResourceHttpRequestHandler***, proširena je novim apstrakcijama ***ResourceResolver***, ***ResourceTransformer*** i ***ResourceUrlProvider***. Brojne ugrađene implementacije obezbeđuju podršku za URL resursa (po verziji) za efektivnije keširanje, lociranje arhiviranih resursa (gzip), generisanje HTML 5 AppCache manifesta i mnogo toga.

Podrška iz JDK 1.8 za java.util.Optional je sada prisutna za ***@RequestParam, @RequestHeader*** i ***@MatrixVariable*** argumente metoda kontrolera.

Metode obeležene anotacijom ***@ModelAttribute*** sada se pozivaju po redosledu koji poštuje unutrašnje zavisnosti (videti specifikaciju SPR-6299).

Podrška za Jackson ***@JsonView*** dostupna je direktno za anotaciju ***@ResponseBody*** i metode kontrolera za ***ResponseEntity*** za serijalizaciju različitih nivoa detalja za isti POJO (na primer zbirni prikaz u odnosu na stranicu sa detaljima (summary vs. detail page)). Ovo je takođe podržano sa kreiranjem pogleda (rendering) kroz dodavanje serijalizacije tipa pogleda kao atributa modela pod specijalnim ključem.

Takođe je potrebno napomenuti da je sada JSONP (JSON with padding) podržan sa Jackson

> Jackson predstavlja popularnu Java biblioteka za serijalizaciju i mapiranje objekata u JSON i obrnuto.

Nove opcije životnog ciklusa su dostupne za presretanje metoda koje su obeležene pomoću anotacija ***@ResponseBody*** i ***ResponseEntity***, upravo nakon što metode kontrolera vrate vrednost i pre nego što je odgovor kreiran. Za postizanje ove aspektno - orijentisane prednosti Springa 4.1 deklariše se zrno ***@ControllerAdvice*** bean za implementiranje ***ResponseBodyAdvice*** podrške. Ove prednosti je moguće koristiti na osnovu ugrađene podrške za ***@JsonView*** i *JSONP*.

Dodate su i tri nove HttpMessageConverter opcije:

- Gson — zauzima manje resursa nego Jackson; već se koristi značajno u okviru Spring Android;
- Google Protocol Buffers - efikasan i efektivan komunikacioni protokol podataka unutrašnjih servisa u okviru složene (enterprise) aplikacije koji mogu biti predstavljeni kao JSON ili XML za veb pregledače.
- XML serijalizacija bazirana na Jackson podršci je dostupna kroz proširenje jackson-dataformat-xml . Kada se koriste ***@EnableWebMvc*** ili *<mvc:annotation-driven/>*, ova podrška je podrazumevana umesto JAXB2, kada se jackson-dataformat-xml nalazi u putanji klasa (claspath).

SPRING 4.1 VEB UNAPREĐENJA - DOPUNSKA RAZMATRANJA

Značajna unapređenja su prisutna i kada su tehnologije pogleda u pitanju.

Tipovi pogleda kao što je *JSP (Java Server Pages)* mogu sada da kreiraju linkove ka kontrolerima ukazujući na mapiranje sa kontrolerima po nazivu. Podrazumevani naziv je pridružen za svaku anotaciju ***@RequestMapping***.

ResponseEntity obezbeđuje API tipa builder-style za upravljanje metodama kontrolera za pripremu odgovora sa strane servera (na primer *ResponseEntity.ok()*).

RequestEntity je nov tip koji obezbeđuje API tipa builder-style za upravljanje REST kodom sa klijent strane za pripremu zahteva sa strane klijenta.

Noviteti kod MVC Java config i XML prostora naziva (namespace):

- rešavači pogleda (View resolvers) su sada podesivi uključivanjem podrške za pregovaranje sadržajem (negotiation);
- kontroleri pogleda imaju sada ugrađenu podršku za preusmeravanje i podešavanje statusa odgovora. Aplikacije mogu da iskoriste navedeno za podešavanje preusmeravajućeg URL, kreiranje 404 odgovora u pogledu, slanje praznog (no content) odgovora i mnogo toga još.
- Prilagođavanje putanje za podudaranje (path matching customizations) koje se često koristi, sada je deo standardne dokumentacije u Springu 4.1.

Na samom kraju ovog izlaganja je moguće istaći da je dodata i podrška za Groovy markup template.

UNAPREĐENJA ZA WEBSOCKET RAZMENU PORUKA

Revizija 4.1 donosti brojna unapređenja za Java WebSocket.

Revizija 4.1 donosi brojna unapređenja za Java ***WebSocket***. Unapređena je moguće, u najkraćim crtama, sagledati kroz sledeće teze:

- *SockJS* (Java) podrška klijent strane;
- Novi događaji konteksta aplikacije ***SessionSubscribeEvent*** i ***SessionUnsubscribeEvent*** koji se objavljuju kada *STOMP* klijent prijavljuje ili odjavljuje, respektivno;
- Dodata je nova ***WebSocket*** oblast (scope) pod nazivom ***"websocket";***
- Anotacija ***@SendToUser*** može da cilja isključivo jednu sesiju i ne zahteva autentifikaciju korisnika;
- ***@MessageMapping*** metode mogu da upotrebljavaju tačku, umesto kose crte "/" kao separator putanje (pogledati SPR-11660).
- *STOMP/WebSocket* monitoring informacija
- Značajno optimizovan i unapređen sistem logovanja, veoma čitljiv i kompaktan čak i na DEBUG nivou;
- Optimizovano kreiranje poruka uključujući podršku za privremeno mutiranje poruka i izbegavanje automatskog identifikatora (id) poruka, kao i kreiranja vremenskih markera (timestamp).
- Zatvaranje *STOMP/WebSocket* konekcije ukoliko nije bilo aktivnosti u intervalu od 60 sekundi nakon uspostavljanja ***WebSocket*** sesije (pogledati SPR-11884),

UNAPREĐENJA IZ OBLASTI TESTIRANJA SPRING APLIKACIJA

Sa Springom 4.1 dolaze i nova unapređenja iz domena testiranja.

Sa Springom 4.1 dolaze i nova unapređenja iz domena testiranja. Najznačajnija unapređenja su istaknuta sledećim tezama:

- Groovy skriptovi sada mogu biti upotrebljeni za podešavanje konteksta aplikacije učitanog za integracione testove u okviru TestContext.
- Transakcije upravljanje testovima mogu da se iz programa pokrenu i završe u okviru metoda transakcionog testa preko novog TestTransaction API.
- Izvršenje SQL skriptova sada može biti podešeno na deklarativan način pomoću novih ***@Sql*** i ***@SqlConfig*** primenjenih na nivou klasa ili metoda.
- Testiranje izvora osobina koji automatski redefinišu sistem i izvor osobina aplikacije, može biti podešeno pomoću nove anotacije ***@TestPropertySource***.
- Podrazumevani osluškivač TestExecutionListeners sada može automatski biti otkriven.
- Korisnički osluškivač TestExecutionListeners može bii automatski spojen sa podrazumevanim osluškivačima.
- Dokumentacija za podršku transakcionom testiranju u okviru TestContext je unapređena sa mnogo novih objašnjenja i dopunskih primera.

- Različita unapređenja za MockServletContext, MockHttpServletRequest i druge Servlet API simulirane objekte (mocks).
- Izmenjena je definicija za AssertThrows koji sad podržava Throwable umesto Exception.
- U Spring MVC Test okviru, JSON odgovori mogu biti istaknuti pomoću JSON Assert kao dodatna opcija korišćenja JSONPath na isti način kao što je to moguće učiniti za XML primenom XMLUnit.
- MockMvcBuilder primeri mogu biti kreirani pomoću MockMvcConfigurer.
- MockRestServiceServer sada podržava AsyncRestTemplate za testiranje klijent strane.

Poglavlje 6

Verzije okvira - Spring 4.2

UNAPREĐENJA OSNOVNOG (CORE) KONTEJNERA

Uvedene su brojne nove anotacije i podrška rada sa pseudonimima atributa anotacija.

Sledi izlaganje koje je u direktnoj povezanosti sa unapređenjima koje sa sobom donosi revizija Spring okvira pod nazivom Spring 4.2:

- Anotacije, poput ***@Bean*** se detektuju i obrađuju standardnim Java 8 metodama, omogućujući povezivanje konfiguracione klase iz interfejsa sa podrazumevanim ***@Bean*** metodama.
- Konfiguracione klase mogu da deklarišu anotaciju ***@Import*** pomoću standardnih komponenata klase dozvoljavajući mešanje uvezenih (imported) konfiguracionih klasa i klasa komponenata.
- Konfiguracione klase mogu da deklarišu vrednost za anotaciju ***@Order***, za obrađivanje po određenom redosledu.
- Tačke umetanja obeležene sa ***@Resource*** podržavaju anotaciju ***@Lazy*** , analogno kao što ***@Autowired*** prima usporeno inicijalizovan proxy (lazy-initializing) za traženo ciljno zrno.
- Upravljanje događajima u aplikacijama takođe se sada zasniva na modelu baziranom na anotacijama.
- Svaka javna metoda upravljanog zrna može biti obeležena anotacijom ***@EventListener*** za obradu događaja.
- Anotacija ***@TransactionalEventListener*** obrađuje podršku za rad sa događajima baziranu na transakcijama.
- Spring okvir 4.2 nudi prvoklasnu podršku za deklarisanje i pretraživanje alternativnih naziva (aliases) atributa anotacija. Nova anotacija ***@AliasFor*** koristi se da deklariše par atributa, čiji su nazivi pseudonimi, u okviru jedinstvene anotacije.

- Sledeće anotacije su naknadno dodate sa ***@AliasFor*** podrškom za značajnu mogućnost pretrage po pseudonimima: ***@Cacheable, @CacheEvict, @CachePut, @ComponentScan, @ComponentScan.Filter, @ImportResource, @Scope, @ManagedResource, @Header, @Payload, @SendToUser, @ActiveProfiles, @ContextConfiguration, @Sql, @TestExecutionListeners, @TestPropertySource, @Transactional, @ControllerAdvice, @CookieValue, @CrossOrigin, @MatrixVariable, @RequestHeader, @RequestMapping, @RequestParam, @RequestPart, @ResponseStatus, @SessionAttributes, @ActionMapping, @RenderMapping, @EventListener, @TransactionalEventListener***.

- Na primer, ***@ContextConfiguration*** iz modula spring-test sada može biti deklarisana na sledeći način:

```
public @interface ContextConfiguration {

    @AliasFor("locations")
    String[] value() default {};

    @AliasFor("value")
    String[] locations() default {};

    // ...
}
```

UNAPREĐENJA OSNOVNOG (CORE) KONTEJNERA - DODATNA RAZMATRANJA

Sada je moguće definisati alias (pseudonim) za vrednosti atributa meta - anotacije.

Nastavlja se isticanje unapređenja osnovnog (core) kontejnera započeto u prethodnoj sekciji:

- Slično, složene anotacije koje redefinišu atribute meta - anotacija mogu da koriste anotaciju ***@AliasFor*** za finiju kontrolu kojom se utvrđuje koji su tačno atributi redefinisani (overridden) u okviru hijerarhije anotacija. U stvari, sada je moguće definisati alias (pseudonim) za vrednosti atributa meta - anotacije.
- Na primer, moguće je razviti složenu anotaciju redefinisanjem atributa na sledeći način:

```
@ContextConfiguration
public @interface MyTestConfig {

    @AliasFor(annotation = ContextConfiguration.class, attribute = "value")
    String[] xmlFiles();

    // ...
}
```

Za detaljnije informacije pogledati https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/annotation-programming-model.html

- Brojna unapređenja su prisutna za Springove algoritme pretrage za pronalaženje meta - anotacija - lokalno deklarisane složene anotacije su sada favorizovane u odnosu na nasleđene anotacije;
- Složene anotacije koje redefinišu atribute meta - anotacija mogu biti otkrivene kako metodama interfejsa, tako i standardnih klasa.

- DefaultConversionService sada obezbeđuje out-of-the-box konvertere za Stream, Charset, Currency i TimeZone.
- DefaultFormattingConversionService dolazi sa podrškom za tipove vrednosti *JSR-354 Money & Currency* (ako je 'javax.money' API prisutan u putanji klasa): ***MonetaryAmount*** i ***CurrencyUnit***. Ovo uključuje podršku za primenu: @NumberFormat.
- ***@NumberFormat*** sada može da se koristi kao meta - anotacija.
- Interfejs ***JavaMailSenderImpl*** poseduje novu metodu testConnection***()*** za proveru konekcije sa serverom..
- Klasa ScheduledTaskRegistrar predstavlja zakazane (scheduled) zadatke.
- Apache***commons-pool2*** paket je sada podržan za grupisanje AOP ***CommonsPool2TargetSource***.
- StandardScriptFactory, kao specifikacijaJSR-223, je osnovni mehanizam za skript zrna, prikazana kao lang:std element u XML. Podržava, na primer, JavaScript i JRuby.

UNAPREĐENJA PRISTUPA PODACIMA I JMS UNAPREĐENJA

Okvir Spring 4.2 donosi značajna unapređenja po pitanju pristupa podacima i JMS sistema.

Okvir Spring 4.2 donosi značajna unapređenja po pitanju pristupa podacima (data access). Najznačajnija su istaknuta sledećim tezama:

- Pomoću aspektno - orijentisanog pristupa AspectJ, dostupna je podrška za javax.transaction.Transact;
- Iterfejs SimpleJdbcCallOperation podržava povezivanje po nazivu.
- Dostupna je puna podrška za Hibernate ORM 5.0: kao JPA provajder ili kroz prirodni API iz paketa org.springframework.orm.hibernate5 package.
- Ugrađenim bazama podataka mogu biti pridruženi jedinstveni nazivi i element <jdbc:embedded-database> podržava nov atribut pod nazivom database-name.

Java Messaging System (JMS) dobija nove funkcionalnosti sa okvirom Spring 4.2. Najznačajnija unapređenja i proširenja mogu biti predstavljeni sledećim tezama:

- Atribut ***autoStartup*** je sada moguće kontrolisati preko interfejsa ***JmsListenerContainerFactory***.
- Tip destinacie odgovora (***Destination)*** moguće je podesiti po osluškivaču kontejnera;
- Vrednost anotacije ***@SendTo*** može sada da koristi SpEL (*Spring Expression Language*).
- Destinacija odgovora može biti određena tokom vremena izvršavanja upotrebom ***JmsResponse***.
- Anotacija ***@JmsListener*** sada može da se ponavlja sa ciljem deklarisanja nekoliko JMS kontejnera za istu metodu.

WEBSOCKET UNAPREĐENJA

Java Messaging System dobija nove funkcionalnosti sa okvirom Spring 4.2.

Za sam početak neophodno je opisati izmene u izlaganju informacija koje su u vezi sa prisustvom korisnika koji su povezani (connected) ili prijavljeni / pretplaćeni (subscribe):

- novi interfejs SimpUserRegistry se predstavlja pomoću zrna "userRegistry".
- moguće je deliti informacije o prisutnosti korisnika preko klastera servera (cluster of servers).

Rešavanje (resolving) korisničkih destinacija preko klastera servera je takođe novina sa ovom revizijom Spring okvira.

Proširenje ***StompSubProtocolErrorHandler*** cilja ka prilagođavanju i kontroli STOMP ERROR okvira za klijente.

Sledeća novina jeste upotreba globalnih ***@MessageExceptionHandler*** metoda preko komponenata ***@ControllerAdvice***.
STOMP klijenti se koriste preko TCP protokola i WebSocket -a.
Anotacijama ***@SendTo*** i ***@SendToUser***, moguće je pridružiti promenljive destinacija.
Takođe, Jackson ***@JsonView*** je podržan za vraćanje vrednosti metoda obeleženih anotacijama ***@MessageMapping*** i ***@SubscribeMapping***.
Dodati su tipovi ***ListenableFuture*** i ***CompletableFuture*** kao povratni tipovi metoda obeleženih anotacijama ***@MessageMapping*** i ***@SubscribeMapping***.
Prisutna je klasa, sa revizijom Spring okvira, ***MarshallingMessageConverter*** za čitanje i pisanje XML sadržaja.

TEST UNAPREĐENJA SA OKVIROM SPRING 4.2

Unapređenja koja donosi Spring 4.2 su najbrojnija iz oblasti testiranja.

Testovi sa JUnit osnovom sada mogu da budu izvršeni preko JUnit pravila umesto ***SpringJUnit4ClassRunner***. Ovo omogućava Spring baziranim testovima integracije da budu izvršeni alternativnim pokretačima (runners) kao što su ***JUnit Parameterized*** ili pokretač "treće strane" kao što je ***MockitoJUnitRunner***.

Okvir Spring MVC Test sada obezbeđuje prvoklasnu podršku za HtmlUnit, uključujući i integraciju sa Selenium WebDriver, omogućavajući testiranje aplikacija, sa značajnim sadržajem veb stranica, bez potrebe za angažovanjem Servlet kontejnera.

Novi alat za testiranje AopTestUtils omogućava programerima dobijanje reference na osnovni ciljni objekat koji je skriven iza jednog ili više Spring poxy - ja.

ReflectionTestUtils sada podržava uzimanje i izmenu (setting and getting) statičkih polja, uključujući i konstante.

Izvorno raspoređivanje (ordering) profila definicija zrna preko anotacije ***@ActiveProfiles*** zadržano je sa ciljem podrške raznim scenarijima primene (use cases), poput ***ConfigFileApplicationListener*** (Spring Boot) koji učitava konfiguracione datoteke na osnovu naziva aktivnih profila.

Anotacija ***@DirtiesContext*** podržava nove režime***BEFORE_METHOD, BEFORE_CLASS*** i ***BEFORE_EACH_TEST_METHOD*** za zatvaranje konteksta aplikacije (ApplicationContext) pre testa — na primer, ukoliko neki test, među velikim brojem testova koji se izvršavaju, ošteti originalnu konfiguraciju za ApplicationContext., moguće je primeniti novu anotaciju ***@Commit*** kao direktnu zamenu za *@Rollback(false)*.

Anotaciju ***@Rollback*** je sada moguće upotrebiti za podešavanje podrazumevane povratne (rollback) semantike na nivou klasa. Kao posledica, primena anotacije ***@TransactionConfiguration*** je sada prevaziđena i biće uklonjena.

Anotacija @Sql sada podržava izvršavanje poređanih (inlined) SQL upita pomoću novog atributa ***statements.***

Modul ***ContextCache***, korišćen za keširanje konteksta aplikacije između testova, sada je javni (***public***) API sa podrazumevanom implementacijom koja može, po potrebi, biti zamenjena u zavisnosti od korisničkih potreba keširanja.

DefaultTestContext, DefaultBootstrapContext i ***DefaultCacheAwareContextLoaderDelegate*** sada su javne klase.

TestContextBootstrapper je sada odgovoran za kreiranje ***TestContext*** - a.

TEST UNAPREĐENJA SA OKVIROM SPRING 4.2 - DOPUNSKA RAZMATRANJA

Brojne izmene u testiranju Spring MVC i JDBC su dodate.

U okviru Spring MVC Test, log detalji MvcResult mogu biti prikazani na nivou DEBUG ili ispisani prilagođenim klasama OutputStream ili Writer.

JDBC XML prostor naziva podržava nov atribut database-name u elementu <jdbc:embedded-database>. Na ovaj način je omogućeno programerima da podese jedinstvene nazive za ugrađene baze podataka. Ovakvim bazama podataka je moguće automatski dodeliti jedinstveni naziv i na taj način omogućiti opštoj konfiguraciji za testiranje baze podataka da bude višestruko upotrebljena u različitim kontekstima aplikacije u okviru skupa testova.

Javne klase MockHttpServletRequest i MockHttpServletResponse sada obezbeđuju bolju podršku za formatiranje zaglavlja (header) podataka pomoću metoda getDateHeader() i setDateHeader().

Unapređenja uvođenjem revizije Spring okvira, pod nazivom Spring 4.2, moguće je delimično pokriti sledećim video materijalom.

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

Poglavlje 7

Verzije okvira - Spring 4.3

UNAPREĐENJA OSNOVNOG KONTEJNERA

Izuzeci osnovnog kontejnera sada obezbeđuju bogatije meta - podatke za programiranje.

Poslednja revizija okvira Spring 4 ima oznaku Spring 4.3. Takođe, donosi značajna unapređenja pri čemu će izlaganje započeti sa unapređenjima koja su u vezi sa osnovnim (core) kontejnerom:

- Izuzeci osnovnog kontejnera sada obezbeđuju bogatije meta - podatke za programiranje;
- Podrazumevane Java 8 metode se javljaju kao setter i getter metode osobina zrna;
- kandidati za usporena zrna (lazy candidate beans) se ne kreiraju u slučaju umetanja primarnog zrna;
- Više nije potrebno specificirati ***@Autowired*** anotaciju ukoliko ciljno zrno definiše samo jedna konstruktor;
- Klase obeležene sa ***@Configuration*** dozvoljavaju umetanje konstruktora;
- Bilo koji SpEL (*Spring Expression Language*) upit iskorišćen za definisanje uslova za ***@EventListener*** može sada da se odnosi na zrno (na primer kao poziv *@beanName.method()*).
- Složene anotacije sada mogu da redefinišu (override) nizove atributa u meta anotacijama pomoću jedinstvenog elementa tipa komponenata niza. Na primer, String[] atribut putanje za anotaciju ***@RequestMapping*** može biti redfinisan pomoću String putanje u složenoj anotaciji;
- Anotacije*@PersistenceContext/@PersistenceUnit* biraju primarno produkciono zrno tipa ***EntityManagerFactory*** ako je tako deklarisano;

- ***@Scheduled*** i ***@Schedules*** mogu sada da se koriste kao meta - anotacije za kreiranje prilagođenih (custom) složenih anotacija sa redefinisanim atributima.
- ***@Scheduled*** je podržana dobro zrnima bilo kojih oblasti.

UNAPREĐENJA PRISTUPA PODACIMA, KEŠIRANJA I JMS SISTEMA

Spring 4.3 dozvoljava da konkurentni pozivi, po datom ključu, budu sinhronizovani.

Prvo unapređenje, koje će biti pomenuto u ovoj sekciji, odnosi se na novine kada je proces pristupa podacima u pitanju. Spring 4.3 omogućava da elementi *jdbc:initialize-database* i

jdbc:embedded-database podržavaju podesiv separator koji je moguće primeniti na bilo koji skript.

Dalje, Spring 4.3 dozvoljava da konkurentni pozivi, po datom ključu, budu sinhronizovani tako da je odgovarajuća vrednost samo jednom obrađena. Ovaj alat je moguće omogućiti pomoću novog ***sync*** atributa anotacije ***@Cacheable***. Ovde je, takođe, učinjena značajna promena u interfejsu Cache, dodavanjem metode *get(Object key, Callable<T> valueLoader).*

Spring 4.3, takođe, unapređuje apstrakciju keširanja na neki od sledećih načina:

- SpEL izrazi u anotacijama koje se odnose na keširanje mogu da ukazuju na zrna (na primer *@beanName.method()*).
- ***ConcurrentMapCacheManager*** i ***ConcurrentMapCache*** podržavaju, sada, serijalizaciju keš unosa preko novog atributa ***storeByValue;***
- Anotacije ***@Cacheable***, ***@CacheEvict***, ***@CachePut*** i ***@Caching*** sada mogu da budu upotrebljene kao meta - anotacije za kreiranje prilagođenih složenih anotacija koje redefinišu atribute;

Spring 4.3 nosi i jedno unapređenje u JMS oblasti:

- anotacije ***@SendTo*** i ***@SendToUser*** mogu biti specificirane na nivou klasa za deljenje opštih destinacija.

UNAPREĐENJA TESTIRANJA

JUnit podrška za Spring TestContext okvir zahteva sada verziju JUnit 4.12 ili noviju.

JUnit podrška za Spring TestContext okvir zahteva sada verziju JUnit 4.12 ili noviju

Anotacije koje su u vezi sa testovima mogu biti deklarisane u okviru interfejsa. Takođe, prazna deklaracija za ***@ContextConfiguration*** sada može biti kompletno izostavljena ukoliko su otkrivene podrazumevane XML datoteke, Groovy skriptovi ili @Configuration klase.

Test metode obeležene sa ***@Transactional*** ne moraju više da budu javne. Takođe, to je slučaj i sa metodama ***@BeforeTransaction*** i ***@AfterTransaction***, koje mogu biti deklarisane podrazumevanim metodama Java 8 baziranog interfejsa.

Keš za ***ApplicationContext*** u Spring ***TestContext*** okviru je sada ograničen na maksimalnu veličinu 32. Novi ContextCustomizer API za podešavanje konteksta aplikacije nakon definisanja zrna učitava se u kontekst pre nego što se kontekst aplikacije osveži (refresh). Prilagođivači (Customizers) mogu biti registrovani globalno, kao "treća strana", zahtevajući prethodno iplementaciju korisničkog (custom) ***ContextLoader.***

Anotacije ***@Sql*** i ***@SqlGroup*** sada mogu da se koriste kao meta - anotacije za kreiranje složenih anotacija sa redefinisanim atributima.

ReflectionTestUtils klasa sada automatski raspakuje (unwraps) proksije prilikom podešavanja (set) ili preuzimanja (get) polja.

Server-side Spring MVC Test podrška podržava očekivanja u zaglavljima odgovora sa višestrukim vrednostima. Takođe, parsira podatke zahteva sa formi i popunjava parametre zahteva. Ova podrška uključuje i prihvatanje simuliranih (mock) objekata za pozvane metode rukovaoce (handler methods).

Client-side REST test podrška dozvoljava das e ukaže koliko puta je zahtev očekivan i da li bi trebalo ignorisati redosled deklarisanja očekivanja. Podržana su očekivanja podataka forme u telu zahteva.

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

PODRŠKA ZA NOVE BIBLIOTEKE I GENERACIJE SERVERA

Dostupne su nove biblioteke i podrška za novije generacije servera.

U Spring 4.3 reviziji, prisutna su sledeća unapređenja i dopune:

- Hibernate ORM 5.2 (podržani su još 4.2/4.3 i 5.0/5.1, 3.6 je odbačen)
- Hibernate Validator 5.3 (minimum ostaje verzija 4.3);
- Jackson 2.8 (minimum je sada Jackson 2.6+);
- OkHttp 3.x ;
- Tomcat 8.5
- Netty 4.1
- Undertow 1.4
- WildFly 10.1.

Savremeni Java dizajn primenom okvira Spring 4.3 detaljno je obrazložen sledećim video materijalom.

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

Poglavlje 8

Verzije okvira - Spring 5

NOVE FUNKCIJE U RADNOM OKVIRU SPRING 5.0

Glavni napredak je evolucija Spring Boot projekta

Radni okvir Spring 5.0 je prva glavna nadgradnja radnog okvira Spring, koja se pojavila skoro četiri godine nakon verzije *Spring 4.0*. U toku te četiri godine glavni napredak bila je evolucija Spring Boot projekta.

Jedna od najvažnijih funkcija radnog okvira *Spring 5.0* je *reaktivno programiranje*. Osnovne funkcije reaktivnog programiranja i podrška *reaktivnih krajnjih tačaka* dostupne su u radnom okviru Spring 5.0. Lista važnih promena uključuje sledeće:

- osnovne nadgradnje
- kompatibilnost pokretanja *JDK-a 9*
- upotreba *JDK* 8 funkcija u kodu radnog okvira Spring
- podrška za reaktivno programiranje
- funkcionalni veb radni okvir
- Java modularnost sa *Jigsawom*
- podrška za *Kotlin*
- otpuštene funkcije

OSNOVNE NADOGRADNJE

Radni okvir Spring 5.0 ima JDK 8 i Java EE 7 osnovu.

Radni okvir Spring 5.0 ima JDK 8 i Java EE 7 osnovu. U suštini,to znači da prethodne verzije JDK-a i Java EE-a više nisu podržane. Neke važne osnove Java EE 7 specifikacija za radni okvir Spring 5.0 su sledeće:

- *Servlet 3.1*
- *JMS 2.0*
- *JPA 2.1*
- *JAX-RS 2.0*
- *Bean Validation 1.1*

Postoji mnogo promena u minimalno podržanim verzijama nekoliko Java radnih okvira. Sledeća lista sadrži neke minimalno podržane verzije istaknutih radnih okvira:

- *Hibernate 5*

- *Jackson 2.6*
- *EhCache 2.10*
- *JUnit 5*
- *Tiles 3*

Podržane su sledeće verzije servera:

- *Tomcat 8.5+*
- *Jetty 9.4+*
- *WildFly 10+*
- *Netty 4.1+* (za veb reaktivno programiranje pomoću *Spring Web Fluxa*)
- *Undertow 1.4+* (za veb reaktivno programiranje pomoću *Spring Web Fluxa*)

> Aplikacije koje koriste ranije verzije bilo kojih navedenih specifikacija/radnih okvira treba da budu nadgrađene najmanje na prethodno izlistane verzije pre nego što mogu da upotrebe radni okvir Spring 5.0.

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

KOMPATIBILNOST POKRETANJA JDK 9

Radni okvir Spring 5.0 je dobio punu kompatibilnost sa novom verzijom Jave.

JDK 9 je izdat kao stabilna verzija u oktobru 2017. Radni okvir Spring 5.0 je u međuvremenu dobio punu kompatibilnost sa novom verzijom Jave.

Sledeći video materijali daju preged kompatibilnosti i upotrebe radnog okvira Spring 5 sa poslednjim JDK izdanjima.

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

JDK 8 I SPRING 5.0

U radnom okviru Spring 5.0 osnovu čini Java 8.

U radnom okviru Spring 5.0 osnovu čini Java 8. Kod radnog okvira Spring je sada nadgrađen da koristi nove funkcije u Javi 8, što će rezultirati čitkijim i bržim kodom radnog okvira. Neke od funkcija Jave 8 koje su upotrebljene su sledeće:

- standardni metodi Java 8 u osnovnim interfejsima Spring 5.0 radnog okvira
- poboljšanja internog koda na osnovu poboljšanja Java 8 refleksije
- upotreba funkcionalnog programiranja u kodu radnog okvira – lambda i tokovi

PODRŠKA ZA REAKTIVNO PROGRAMIRANJE

Reaktivno programiranje je jedna od najvažnijih funkcija radnog okvira Spring 5.0.

Arhitekture mikroservisa su, obično, izgrađene na osnovu komunikacije zasnovane na događajima. Aplikacije su građene tako da reaguju na događaje (ili poruke).
Reaktivno programiranje obezbeđuje alternativni stil programiranja fokusiranog na izgradnju aplikacija koje reaguju na događaje. Iako Java 8 nema ugrađenu podršku za reaktivno programiranje, postoji veliki broj radnih okvira koji obezbeđuju podršku za to programiranje:

- *Reactive Streams* - jezički neutralan pokušaj da se definišu reaktivni API-ji
- *Reactor* - Java implementacija za *Reactive Stream*, koju obezbeđuje *Spring Pivotal* tim
- *Spring WebFlux* - Omogućava razvoj veb aplikacija na osnovu reaktivnog programiranja. Obezbeđuje programski model sličan Spring MVC-u.

FUNKCIONALNI VEB RADNI OKVIR

Spring 5 obezbeđuje funkcionalni veb radni okvir.

Oslanjajući se na reaktivne funkcije, Spring 5 takođe obezbeđuje funkcionalni veb radni okvir. Funkcionalni veb radni okvir obezbeđuje funkcije za definisanje krajnjih tačaka pomoću funkcionalnosti stila programiranja. Jednostavan „hello world“ primer je prikazan ovde:

```
RouterFunction<String> route =
route(GET("/hello-world"),
request -> Response.ok().body(fromObject("Hello World")));
```

Funkcionalni veb radni okvir takođe može da se upotrebi za definisanje složenijih ruta, kao što je prikazano u sledećem primeru:

```
RouterFunction<?> route = route(GET("/todos/{id}"),
request -> {
Mono<Todo> todo = Mono.justOrEmpty(request.
pathVariable("id"))
.map(Integer::valueOf)
.then(repository::getTodo);
return Response.ok().body(fromPublisher(todo, Todo.
class));
})
.and(route(GET("/todos"),
request -> {
Flux<Todo> people = repository.allTodos();
return Response.ok().body(fromPublisher(people, Todo.
class));
}))
.and(route(POST("/todos"),
request -> {
```

```
Mono<Todo> todo = request.body(toMono(Todo.class));
return Response.ok().build(repository.saveTodo(todo));
}));
```

Napomena:

- *RouterFunction* procenjuje odgovarajući uslov za zahteve rutiranja za odgovarajuću funkciju za obradu.
- Definisane su tri krajnje tačke, dve *GET* i jednu *POST*, i mapirane u različite funkcije za obradu.

JAVA MODULARNOST SA JIGSAW-OM

Do verzije Java 8, Java platforma nije bila modularna.

Do verzije Java 8, Java platforma nije bila modularna. Iz toga je proizašlo nekoliko problema:

- prenatrpanost platforme;
- *JAR Hell* - Java *ClassLoader* pronađe klasu,
  neće videti da li za nju postoje druge dostupne definicije. Odmah učitava prvu klasu koju pronađe. Ako dva različita dela aplikacije zahtevaju istu klasu iz različitih arhiva, ne postoji način da specifikuju arhivu iz koje treba da bude učitana klasa.

Svaki modul (koji se zove komplet) definiše sledeće:

- uvoz - drugi kompleti koje modul koristi
- izvoz - paketi koje ovaj paket izvozi

Svaki modul može da ima svoj „životni ciklus“. Može da bude instaliran, pokrenut i zaustavljen nezavisno.

Jigsaw je inicijativa pod projektom *Java Community Process (JCP),* pokrenuta od verzije Java 7, za uvođenje modularnosti u Javu. Jigsaw ima dva glavna cilja:

- definisanje i implementiranje modularne strukture za JDK
- definisanje sistema modula za aplikacije koje su izgrađene na Java platformi

Jigsaw biti deo verzije Java 9 i da će radni okvir *Spring 5.0* uključiti osnovnu podršku za *Jigsaw* module.

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

UKLONJENE FUNKCIJE

Spring 5.0 je takođe uklonio podršku za nekoliko radnih okvira.

Spring 5.0 je takođe uklonio podršku za nekoliko radnih okvira:

- *Portlet*
- *Velocity*
- *JasperReports*
- *XMLBeans*
- *JDO*
- *Guava*

NOVE FUNKCIJE SPRING BOOTA 2.0

Spring Boot okvir je takođe revidiran paralelno sa Spring 5.0 okvirom

Sledi lista nekih važnih ažuriranja koja se očekuju u verziji Spring Boot 2.0:

- Osnovna JDK verzija je *Java 8*.
- Osnovna Spring verzija je *Spring 5.0*.
- *Spring Boot 2.0* ima podršku za reaktivno veb programiranje pomoću *WebFlux*-a.

Minimalna podržana verzija nekih važnih radnih okvira je izlistana ovde:

- *Jetty 9.4*
- *Tomcat 8.5*
- *Hibernate 5.2*
- *Gradle 3.4*

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

Poglavlje 9

Pokazni primer - Maven konfiguracioni fajl

DATOTEKA POM.XML

Kreiranje konfiguracione datoteke za rad u Springu.

Kao što je pomenuto, u nekom od prethodnih izlaganja, za rad na razvoju Spring aplikacija poželjno je upotrebiti neki alat za upravljanje zavisnostima. Dva najpopularnija alata su Maven i Gradle. U ovom primeru akcenat će biti na upotrebi Maven alata za učitavanje zavisnosti u jednostavnu Spring aplikaciju.

Iz priloženog koda se vide sledeće zavisnosti koje se koriste za kreiranje i prevođenje Spring aplikacije:

- Platforma Java 1.8;
- Spring verzija 4.3.0;
- Uključen je modul spring-webmvc;
- Uključen je javax.servlet-api modul - verzija 3.0.1;
- Uključen je jstl (JSP Standard Tag Library) - verzija 1.2;

Za prevođenje aplikacije koriste se sledeći dodaci (plugins):

- maven-compiler-plugin - verzija 2.3.2;
- maven-war-plugin - verzija 2.4.

Naziv aplikacije je "HelloWorld".

```
<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 http://maven.apache.org/
maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>net.viralpatel.spring</groupId>
    <artifactId>HelloWorld</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>HelloWorld Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <properties>
        <java-version>1.8</java-version>
        <org.springframework-version>4.3.0.RELEASE</org.springframework-version>
    </properties>
```

```
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${org.springframework-version}</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.0.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>
    </dependencies>
    <build>
        <finalName>HelloWorld</finalName>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>2.3.2</version>
                    <configuration>
                        <source>${java-version}</source>
                        <target>${java-version}</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-war-plugin</artifactId>
                    <version>2.4</version>
                    <configuration>
                        <warSourceDirectory>src/main/webapp</warSourceDirectory>
                        <warName>helloworld</warName>
                        <failOnMissingWebXml>false</failOnMissingWebXml>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```

ZADATAK ZA SAMOSTALNI RAD

Konfiguracija za jednostavnu Spring aplikaciju u Gradle - u

Kreirati konfiguraciju za jednostavnu Spring aplikaciju u Gradle - u alatu za upravljanje zavisnostima. Aplikacija može biti kao i u prethodnom primeru, a zavisnosti takođe mogu biti identične kao u prethodnom primeru.

Sledećim video materijalima dat je kratak pregled primene Maven i Gradle alata, respektivno.

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 10

Pokazna vežba 1 - Maven projekat

ZADATAK 1 (45 MIN)

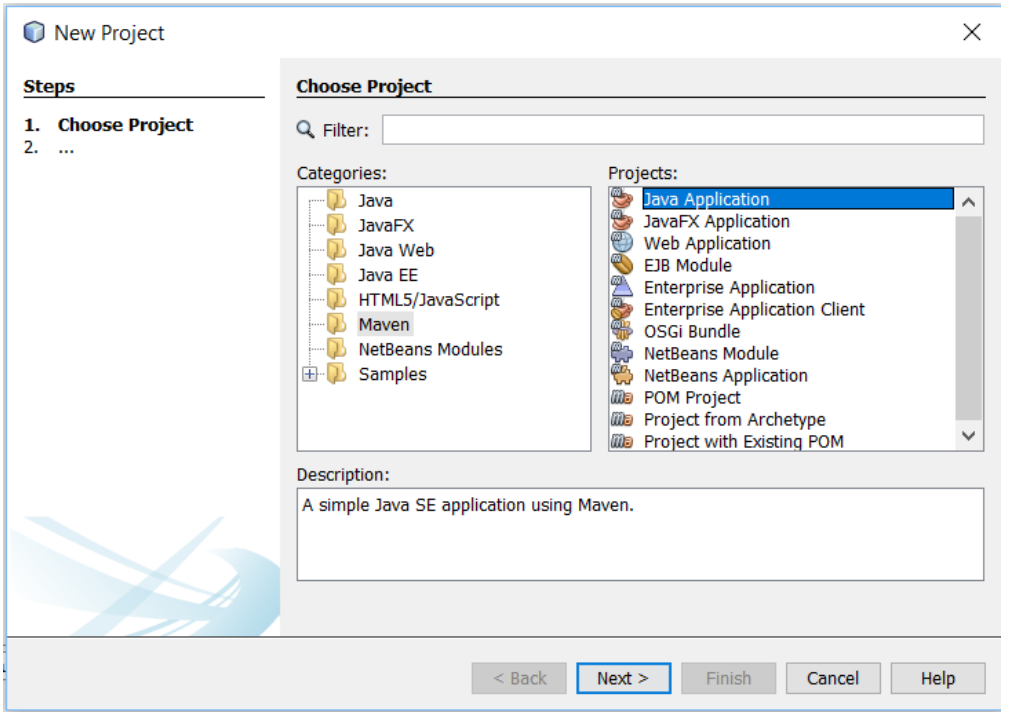
Kreira se prvi Maven projekat u Javi.

U narednim lekcijama će biti značajno korišćen Maven alata za upravljanje zavisnostima. Otuda je prvo zadatak da se pokaže:

- kako se kreira Java Maven projekat,
- kakva mu je struktura,
- kako se prevodi;
- kako se izvršavaju jednostavni testovi, i na kraju
- kako se izvršava aplikacija.

Upravo je navedeno prvi zadatak koji će ovi nastavni materijali praktično pokazati.

Prvo će biti otvoreno razvojno okruženje, neka to bude NetBeans IDE i izborom opcije New Project odabrati Maven Java projekat, baš kao na sledećoj slici:

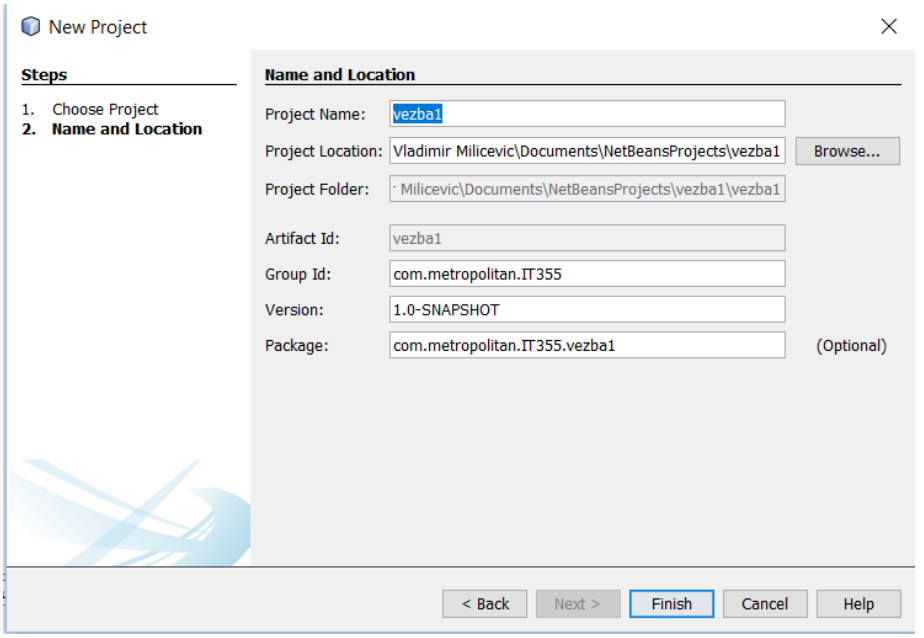


Slika 10.1 Izbor za kreiranje Maven Java projekta - aplikacije [izvor: autor]

POČETNO PODEŠAVANJE PROJEKTA

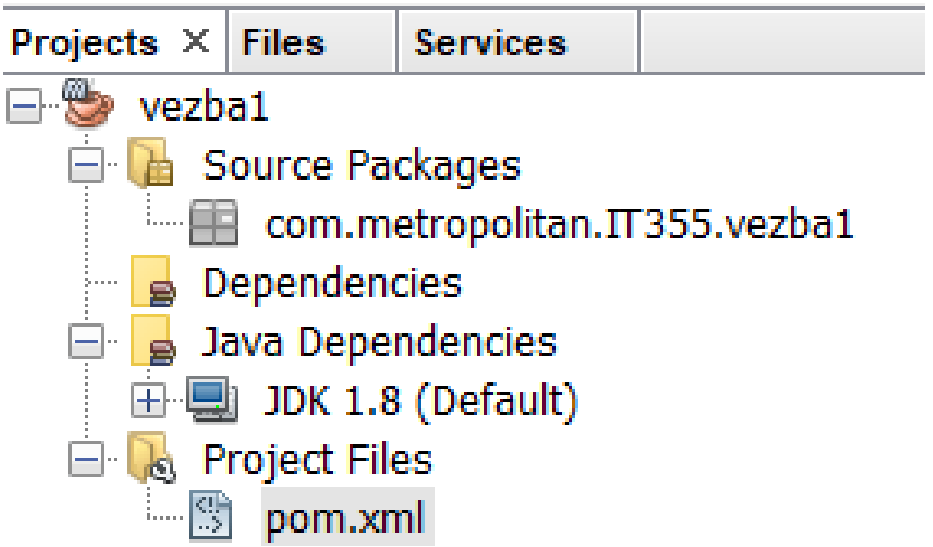
Neophodno je dodeliti naziv i paket projektu

U nastavku je neophodno inicijalno podesiti projekat dodelom odgovarajućeg naziva, kao i paketa kojem projekat pripada. Sledećom slikom je ilustovan ovaj korak. Klikom na dugme Finish, Maven Java projekat je kreiran



Slika 10.2 Podešavanje naziva projekta, paketa i Group Id [izvor: autor]

Ono što je moguće primetiti, jeste, da je kreiran projekat kojem bi trebalo dodati odgovarajuće zavisnosti za kreiranje i prevođenje aktuelne aplikacije, kao i odgovarajuću programsku logiku u formi Java klasa. Inicijalna struktura kreiranog projekta je prikazana sledećom slikom.

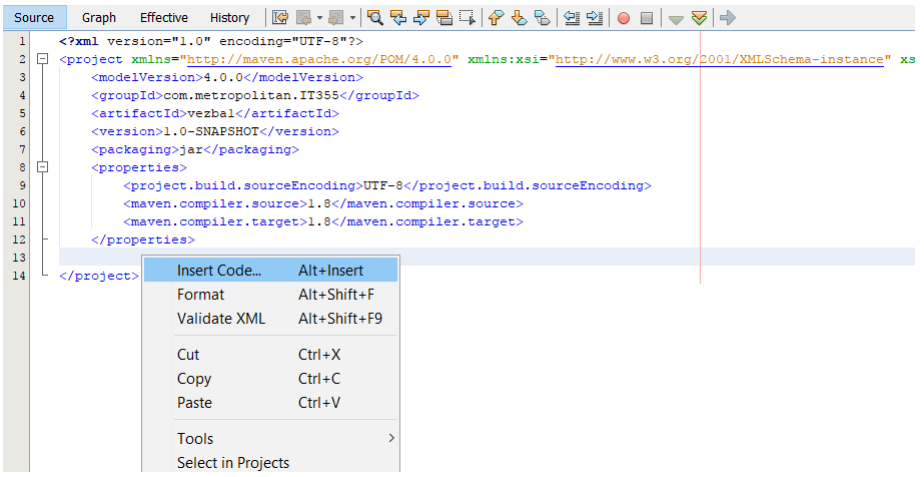


Slika 10.3 Početna struktura kreiranog Maven projekta [izvor: autor]

DODAVANJE ZAVISNOSTI U PROJEKAT

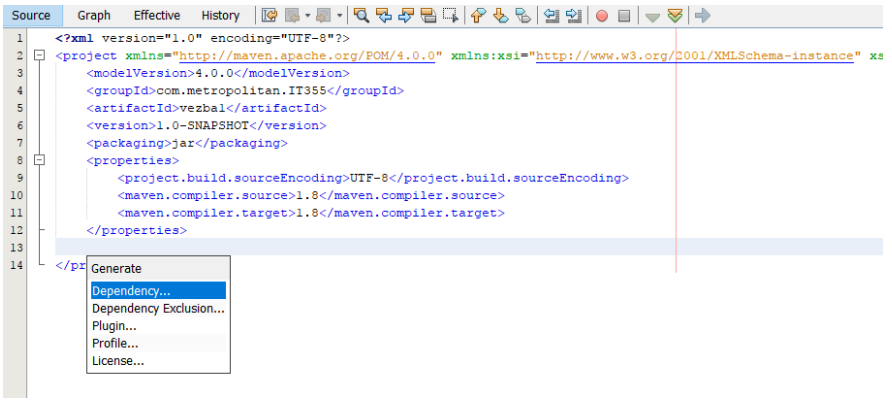
U kreiranom projektu neophodno je modifikovati datoteku pom.xml.

Dodavanje zavisnosti u Maven projekat vrši se modifikovanjem pom.xml datoteke (Project Object Model). Neophodno je otvoriti kod ove datoteke, u razvojnom okruženju NetBeans IDE, i desnim klikom odabrati opciju Insert Code. Navedeno je prikazano sledećom slikom.



Slika 10.4 Izbor opcije Insert Code za pom.xml [izvor: autor]

Izborom navedene opcije, na raspolaganju je sledećih nekoliko opcija. Ovde je od posebnog značaja opcija Dependencies kojom se kreira tag zavisnosti u pom.xml dokumentu. Izbor ove opcije je prikazan sledećom slikom.

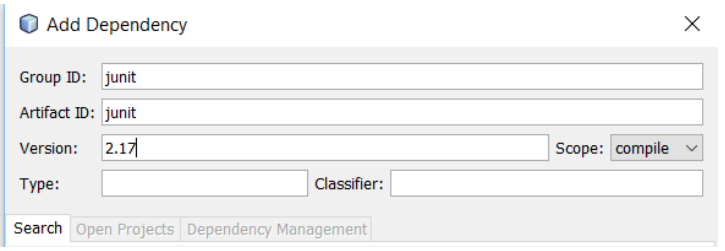


Slika 10.5 Dodavanje taga zavisnosti [izvor: autor]

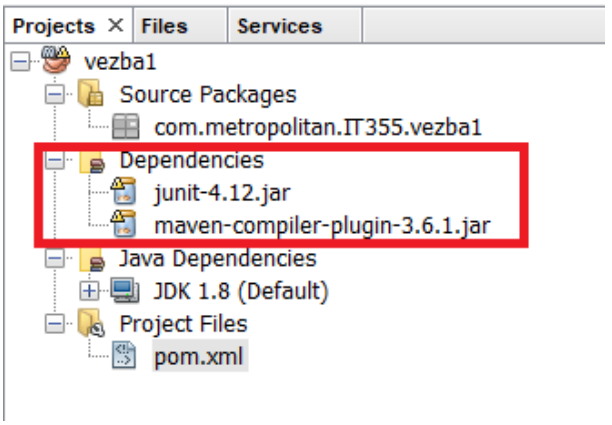
MODIFIKOVANA POM.XML DATOTEKA

Neophodno je dodati sve zavisnosti potrebne za kreiranje i prevođenje aplikacije

Nakon izbora opcije za dodavanje zavisnosti, otvara se nov prozor u kojem je neophodno izabrati grupu kojoj zavisnost pripada (Group ID) kao i naziv same zavisnosti (Artifact ID). Olakšanje je auto - complete opcija i unos početnih slova koje odgovaraju ovim vrednostima biće praćen listom odgovarajućih vrednosti za grupe i artifakte.



Slika 10.6 Izbor grupe i naziva zavisnosti koja se dodaje [izvor: autor]



Slika 10.7 Zavisnosti u ažuriranoj strukturi projekta [izvor: autor]

Iz slike 7 je moguće primetiti da su zavisnosti automatski preuzete i zauzele svoje mesto u folderu Dependencies kreiranog projekta. Sledećim listingom je priložene konačna verzija datoteke pom.xml.

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.metropolitan</groupId>
    <artifactId>IT355vezba1</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <dependencies>
        <dependency>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.6.1</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
```

```
            <version>4.12</version>
        </dependency>
    </dependencies>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

</project>
```

KREIRANJE LOGIKE PROJEKTA I TEST KLASE

Kreiraju se Java klase za izvršavanje tražene logike aplikacije.

Biće kreirana jednostavna Java klasa za generisanje slučajnih kodova sa izvesnim brojem karaktera. Sledećim listingom je priložena navedena klasa čiji će naziv biti App.java.

```
package com.metropolitan;

/**
 *
 * @author Vladimir Milicevic
 */
import java.util.UUID;

/**
 * Generate a unique number
 *
 */
public class App
{

    public static void main( String[] args )
    {
        App obj = new App();
        System.out.println("Unique ID : " + obj.generateUniqueKey());
    }

    public String generateUniqueKey(){

        String id = UUID.randomUUID().toString();
        return id;

    }
}
```

Budući da je dodata zavisnost i za JUnit, biće korišćena dostupna funkcionalnost pa će za kreiranu klasu biti kodirana i odgovarajuća test klasa. Ova klasa proverava da li kreirani

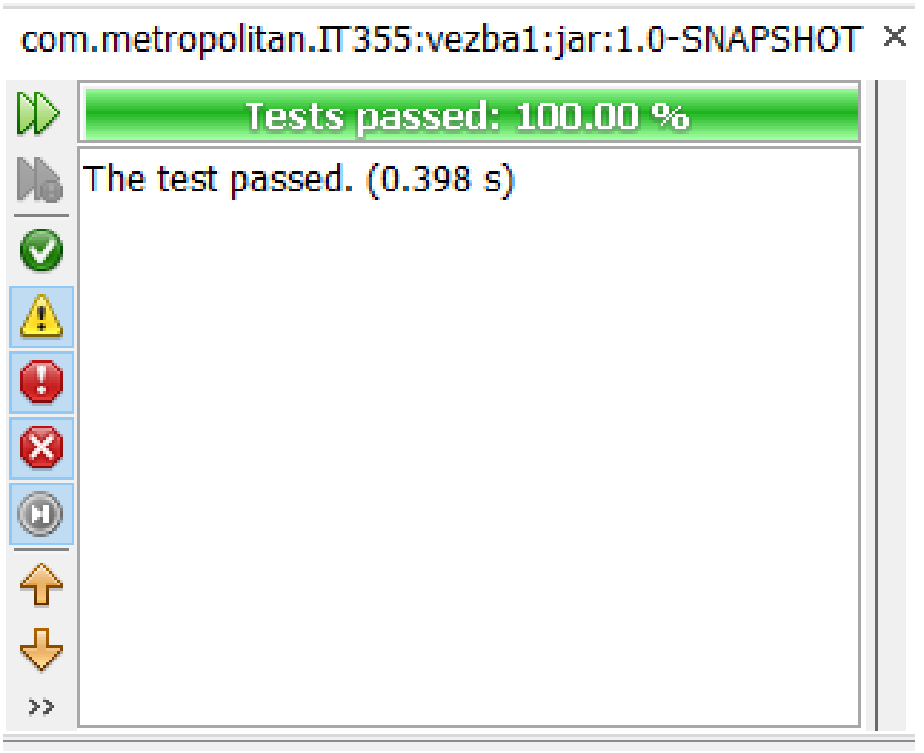
objekat klase App sadrži jedinstveni ključ dužine 36 karaktera. Sledećim listingom je priložena test klasa.

```
package com.metropolitan;

/**
 *
 * @author Vladimir Milicevic
 */
import org.junit.Assert;
import org.junit.Test;

public class AppTest {

    @Test
    public void testLengthOfTheUniqueKey() {

        App obj = new App();
        Assert.assertEquals(36, obj.generateUniqueKey().length());

    }
}
```

TESTIRANJE I IZVRŠAVANJE APLIKACIJE

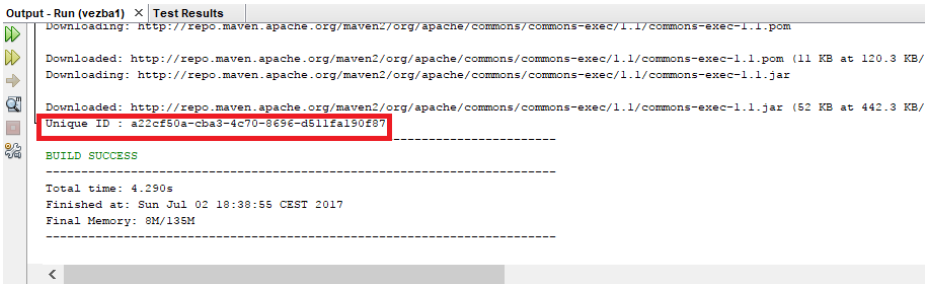
Navedenim klasama je kompletiran projekat

Prvo će biti pokrenuta test klasa AppTest.java. Iz priložene slike se vidi da je test prošao i da je kreirani ključ odgovarjući.



Slika 10.8 Rezultati JUnit testa [izvor: autor]

Na kraju, pokreće se aplikacija iz razvojnog okruženja. Na konzoli je moguće primetiti kako se obrađuju zavisnosti neophodne za prvođenje aplikacije. Ako je sve uspešno, rezultat izvršavanja će, takođe, biti prikazan (videti sliku 9).



Slika 10.9 Izvršavanje logike aplikacije [izvor: autor]

Urađen i testiran primer je priložen kao dodatni materijal uz sekciju Vežba 1 - Maven projekat.

Poglavlje 11

Individualna vežba 1

ZADATAK ZA SAMOSTALNI RAD (90 MIN)

Prethodni zadatak konfigurisati primenom Gradle -a.

Pokušajte u razvojnom okruženju:

- NetBeans,
- Eclipse ili
- IntelliJ

primenom alata za upravljanje zavisnostima da uradite isti zadataka sa pokaznih vežbi:

1. Uradite prvo Maven projekat;
2. Uradite potom i Gradle projekat.

Ukoliko naiđete na poteškoće, kontaktirajte vašeg predmetnog asistenta.

Poglavlje 12

Domaći zadatak 1

ZADATAK 1 (120 MIN)

Izrada domaćeg zadatka.

1. Otvoriti NetBeans IDE razvojno okruženje;
2. Kreirati Maven / Java FX projekat;
3. Projekat može biti urađen po uzoru i na već postojeći primer iz CS102;
4. Modifikovati pom.xml na identičan način kao u vežbama - postoji podrška za JUnit;
5. Napisati bar jednu Test klasu i izvršiti je;
6. Pokrenuti Java FX Maven projekat i demonstrirati njegovu funkcionalnost.

Nakon urađenih obaveznih zadataka, studenti dobijaju od predmetnog asistenta različite primere na email.

Poglavlje 13

Zaključak

ZAKLJUČAK

Lekcija je obradila osnovnu arhitekturu Spring okvira i pregled unapređenja kroz revizije 4.x.

U uvodnim razmatranjima, lekcija je istakla da Spring okvir (*Spring Framework*) predstavlja Java platformu koja obezbeđuje obimnu infrastrukturnu podršku za razvoj širokog spektra Java aplikacija. Celokupnom infrastrukturom upravlja Spring pa tim za razvoj može da se fokusira isključivo na razvoj konkretnih programa.

Takođe je istaknuto kako Spring omogućava kreiranje aplikacija iz POJO (plain old Java objects) objekata i primenu složenih (enterprise) servisa na POJO objekte. Navedene funkcionalnosti Spring okvira primenjuju se u potpunosti na Java SE (*Java Standard Edition*) model programiranja i u potpunosti, ili parcijalno, na Java EE (*Java Enterprise Edition*) model programiranja.

Izlaganje je nastavljeno pokazivanjem i opisom strukture Spring okvira sa posebnim osvrtom na njegove module, njihov sadržaj i namenu.

Lekcija se pobrinula da čitalac ima detaljan pregled unapređenja kroz 4 značajne revizije: Spring 4.0 (glavna - major), kao i Spring 4.1, Spring 4.2 i Spring 4.3 (manje - minor).

Konačno, prikazani su noviteti kroz poslednje revizije okvira *Spring 5.0* i *Spring Boot 2.0.*

LITERATURA

Za pripremu lekcije korišćena je najnovija pisana i elektronska literatura.

Za pripremu lekcije korišćena je najnovija pisana i elektronska literatura:

1. Gary Mak, Josh Long, and Daniel Rubio, Spring Recipes Third Edition, Apress
2. Spring Framework Reference Documentation - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/index.html
3. Craig Walls, Spring in Action, Manning
4. Craig Walls, Spring Boot in Action, Manning
5. Ranga Karanam, Naučite Spring 5, Kompjuter biblioteka

Dopunska literatura:

1. http://www.javacodegeeks.com/tutorials/java-tutorials/enterprise-java-tutorials/spring-

tutorials/
2. http://www.tutorialspoint.com/spring/
3. http://www.javatpoint.com/spring-tutorial