SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Stjepan Čolak

Izrada web aplikacije u razvojnom okruženju Spring

ZAVRŠNI RAD

Varaždin, 2015.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Stjepan Čolak

Matični broj: 39931/11–R

Studij: Poslovni sustavi

Izrada web aplikacije u razvojnom okruženju Spring

ZAVRŠNI RAD

Mentor:

Dr.sc. Tihomir Orehovački

Varaždin, rujan 2015.

Sadržaj

1. Uvod

Razvojem web tehnologija aplikacije postaju sve kompleksnije postaje potreba razviti što učinkovitiju aplikaciju u što kraćem vremenskom periodu, pokriti sve platforme potrebne za razvoj od mobilnih do web platformi. Svaki projekt kreće s izborom ciljane platforme u ovom slučaju je to web. Spring se pokazao kao jedna od najboljih i iz tog razloga jedna od najpopularnijih tehnologija za razvoj teških i kompleksnih aplikacija pisanih za web. Razlog njegove popularnosti leži u njegovoj modularnosti koja mu omogućava da koristi samo one dijelove koji su potrebni.

Cilj ovog rada je pružiti uvod u osnovne koncepte Spring razvojnog okruženja, njihovu primjenu i koristi od korištenja u projektima. Ovaj završni rad se sastoji od teorijskog i praktičnog dijela. U teorijskom dijelu opisuje se Spring razvojno okruženje i njegovi osnovni koncepti i usporedba s ostalim aktualnim razvojnim okruženjima kao što su JSF, Vaadin i Google Web Toolkit, rade se usporedbe i opisuju prednosti odnosno nedostaci Spring-a u odnosu na ostala razvojna okruženja. U prvom dijelu su napisani i nabrojeni moduli Spring razvojnog okruženja, i njihove karakteristike. U drugom poglavlju rada se opisuje Spring MVC (Model – View - Controller) koji omogućuje aplikacijama pisanim korištenjem Springa-a da budu modularne tj. da se odvoji dizajn od funkcionalnosti itd. U praktičnom dijelu se radi na izradi aplikacije te opisu izrade te komponenti same aplikacije. Od same izrade baze podataka do krajnjeg razvoja poslovne logike. Na kraju u zaključku su opisani dojmovi o radu u Spring tehnologijama.

2. Uvod u Spring razvojno okruženje

Spring razvojno okruženje je Java platforma koja pruža široki panel opcija kao podrški razvoju Java aplikacija. Spring rukuje infrastrukturom, tako da programer može usmjeriti svoj fokus na razvoj aplikacije. Važni aspekti Spring-a su Injekcija ovisnosti (Dependency injection) i Inverzija kontrole (Inversion of Control) koji su ključni dio koji omogućuje funkcioniranje Java aplikacije pisane korištenjem tehnologije Spring razvojnog okruženja.

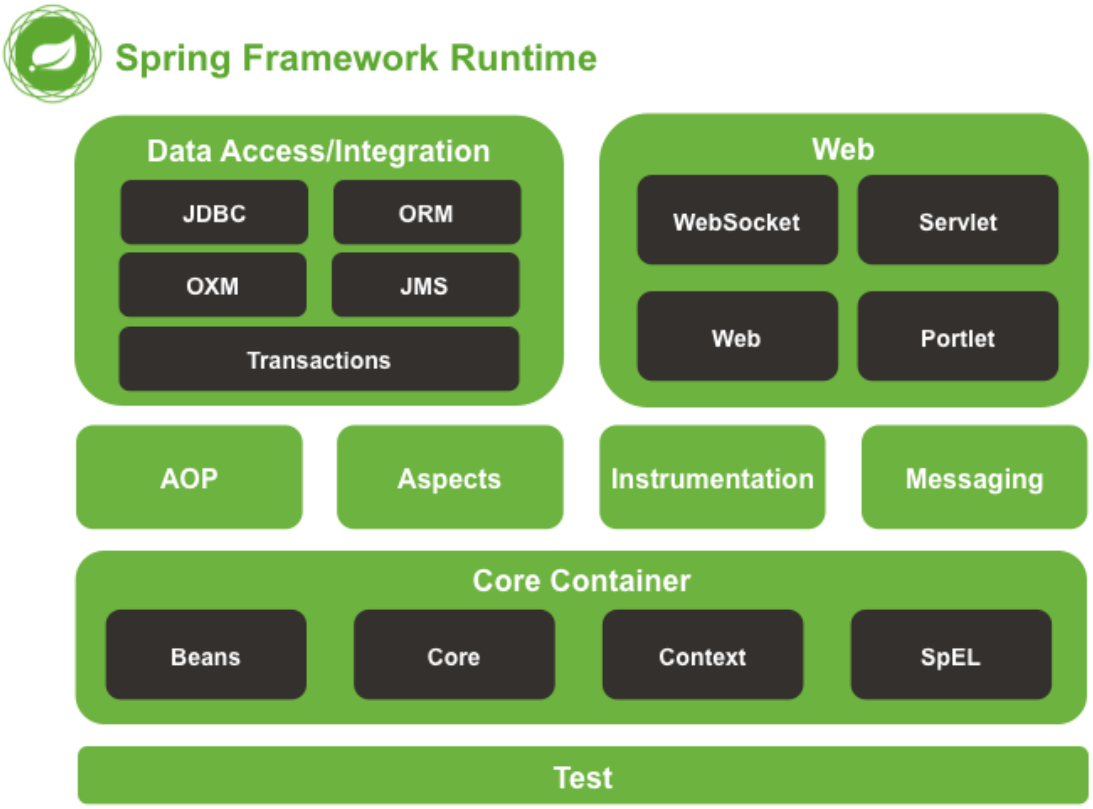
Java aplikacija je širok pojam koji se proteže od osnovnih ugrađenih (embedded) aplikacija do serverskih enterprise aplikacija i obično se sastoji od objekata koji surađuju da bi aplikacija zadovoljavala specifikacije. Što znači da objekti unutar aplikacije imaju ovisnosti (dependencies) jedan o drugome.

Iako Java platforma pruža bogatstvo funkcionalnosti za razvoj aplikacija, manjkava je u smislu organizacije osnovnih građevnih blokova u jedinstvenu i funkcionalnu cjelinu i taj zadatak pada na arhitekte i programere. Iako je moguće koristiti različite uzorke dizajna kao npr. Factory, Abstract Factory, Builder, Decorator itd. kako bi sastavili različite klase i instance objekata koji čine aplikaciju, na kraju krajeva oni su opet samo to uzorci dizajna ili najbolje prakse koje su imenovane, s opisom što rade, gdje ih koristiti i koje probleme rješavaju.

Komponenta IoC (Inversion of Control) Spring razvojnog okruženja pruža formalizirane načine kompozicije razdvojenih komponenti u potpuno funkcionalnu aplikaciju spremnu za korištenje. (prema Spring službenoj dokumentaciji[1])

2.1. Moduli

„Spring razvojno okruženje sastoji se od dijelova organiziranih u oko dvadeset modula. Ti moduli su grupirani u: Core Container, Data Access/Integration, Web, AOP (Aspect oriented programming), Instrumentation, Messaging, Test.“[1]



Slika 1. Pregled arhitekture Spring razvojnog okruženja

2.1.1. Core Container

„Core container se sastoji od više modula a to su redom: *spring-core*, *spring-beans*, *spring-context*, *spring-context-support* i *spring-expression*.“[1]

Spring-core i *spring-beans* moduli su osnovni dijelovi Spring razvojnog okruženja uključujući Inverziju kontrole (IoC) i Injekciju ovisnosti (Dependency Injection) svojstava. *Bean factory* je sofisticirana implementacija Factory uzorka dizajna. Uklanja potrebu za programskim singleton-ovima i dopušta programeru da razdvaja konfiguracije i specifikacije ovisnosti od stvarne programske logike.

Spring-context modul gradi na čvrstoj osnovi koju pružaju *Spring-core* i *spring-beans* moduli što bi značilo pristupanje objektima na način sličan JNDI registrima. Ovaj modul nasljeđuje svojstva od *Spring-beans* modula i dodaje potporu za internalizaciju, učitavanje resursa i transparentno kreiranje konteksta npr. Servlet spremnika. *Spring-context-support* pruža podršku za integraciju čestih programskih biblioteka u Spring aplikacijski kontekst za privremeno spremanje.

Spring-expression modul pruža moćan jezik za upite i manipuliranje grafom objekta u vremenu izvođenja. Ovaj modul je ekstenzija Unified Expression Language specificiranog u JSP 2.1. Jezik podržava postavljanje i čitanje vrijednosti svojstava, dodjeljivanje svojstava, pozivanje metoda, pristupanje sadržaju polja, kolekcija i indeksa, logičke i aritmetičke operacije, imenovane varijable i povrat objekata po imenu iz Spring IoC kontejnera.

2.1.2. AOP i Instrumentation

„*Spring-aop* modul pruža AOP implementaciju dopuštajući programeru da čisto razdvoji kod koji implementira funkcionalnost koja treba biti odvojena.“[1]

Spring-istrument modul pruža podršku instrumentalizaciji klasa i implementaciju učitavanja klasa u određenim aplikacijskim serverima. Poseban *Spring-istrument-tomcat* modul sadrži Spring-ov instrumentacijski agent za Tomcat servere.

2.1.3. Messaging

Verzija 4.0 Spring razvojnog okruženja uključuje *Spring-messaging* modul s ključnim apstrakcijama iz Spring Integration projekta kao što su *Message, MessageChannel, MessageHandler,* i ostali koji služe kao temelj za aplikacije temeljene na razmjeni poruka. Modul uključuje set napomena za mapiranje poruka u metode.

2.1.4. Data Access/Integration

„Data Access/Integration sloj se sastoji od JDBC(Java Database Connectivity), ORM(Object-relational mapping), OXM(Object XML mapping), JMS(Java Messaging Service), i transakcijskih modula.“[1]

Spring-jdbc modul pruža JDBC-apstraktni sloj koji uklanja potrebu za dosadnim JDBC kodiranjem.

Spring-tx modul podržava programsko i deklarativno upravljanje klasama koje implementiraju posebna sučelja i za sve POJO-se (Plain Old Java Objects).

Spring-orm modul pruža integracijske slojeve za popularno objektno relacijsko mapiranje API-ja, uključujući JPA(Java Persistence API), JDO (Java Data Objects) i Hibernate. Korištenjem ovog modula možete koristiti sva ova objektno relacijska razvojna okruženja u kombinaciji sa svime ostalim što Spring pruža.

Spring-oxm modul pruža apstraktni sloj koji podržava Objektno/XML mapirane implementacije kao što su JAXB(Java Architecture for XML Binding), Cactor, XMLBeans, JiBX i XStream.

Spring-jms modul sadržava značajke za slanje i primanje poruka. Od verzije Spring 4.1. pruža integraciju s *Spring-messaging* modulom.

2.1.5. Web

„Web sloj se sastoji od *Spring-web, Spring-webmvc, Spring-websocket i Spring-webmvc-portlet* modula.“[1]

Spring-web modul pruža osnovne web orijentirane integracijske značajke kao višedijelno učitavanje datoteka i inicijaliziranje IoC spremnika koristeći Servlet slušače (listeners) i web orijentirani aplikacijski kontekst. Također koristi HTTP klijenta i dijelove Spring podrške vezane za web.

Spring-webmvc modul (poznat i kao web servlet modul) sadrži Spring-ov MVC (Model-View-Controller) i REST Web servis implementacije za web aplikacije. Spring-ovo MVC razvojno okruženje pruža čisto razdvajanje između koda i formi i integrira se s ostalim značajkama Spring razvojnog okruženja.

Spring-webmvc-portlet modul dopušta MVC implementaciji da se koristi u Portlet okruženju i kopira funkcionalnosti *Spring-webmvc modula.*

2.1.6. Test

„Spring-test modul podržava jedinične testove i integracijsko testiranje Spring komponenti s Junit ili TestNG.“[1] Pruža stalno učitavanje Spring aplikacijskog konteksta i privremeno spremanje tog konteksta. Također pruža lažne (mock) objekte koje je moguće koristiti za testiranje koda u izolaciji.

3. Spring MVC

Spring MVC (Model-View-Controller) je izgrađen na osnovi *DispatcherServlet*-a koji šalje zahtjeve rukovateljima, sa konfigurabilnim mapiranjima rukovatelja, razlučivosti prikaza, vremenskom zonom kao i podrškom za učitavanje datoteka. Zadani rukovatelj se temelji na *@Controller* i *@RequestMapping* anotacijama, nudeći široki raspon fleksibilnih metoda rukovanja. Od verzije Spring 3.0, *@Controller* mehanizam također dopušta kreiranje RESTful web stranica i aplikacija kroz *@PathVariable* anotaciju i ostale značajke.

Ključan princip dizajna u SpringMVC i u Spring-u općenito je „Otvoren za proširivanje, zatvoren za modifikaciju“ princip. Što bi značilo da aplikacije rađene u Spring razvojnom okruženju omogućuje proširivanje raznim dodacima ali ne dozvoljavaju izmjene na samoj aplikaciji. (prema Deinum, M. i suradnicima [2])

3.1. Značajke Spring Web MVC-a

3.1.1. Spring Web Flow

„Spring Web Flow (SWF) cilja na to da postane najbolje rješenje za upravljanje toka stranica web aplikacije. SWF se integrira s postojećim radnim okruženjem kao npr. Spring MVC ili JSF, u Servlet i Portlet okruženjima. Ako postoji poslovni proces ili procesi, koji bi imali koristi od model temeljenog na komunikaciji između procesa više nego modela koji se temelji na zahtjevima između istih u tom slučaju SWF je jedno od mogućih rješenja.“ [2]

SWF dopušta da stranice s logičkim tokovima spremite kao samostalne module koje je moguće višekratno koristiti u različitim situacijama i kao takav je idealan za izradu modula web aplikacija koji vode korisnika kroz kontroliranu navigaciju koja pokreće poslovne procese.

3.1.2. Važne značajke Spring MVC-a[1]

(Prema Deinum, M i suradnicima [2])Spring Web modul uključuje razne i jedinstvene značajke od kojih su neke:

- Čisto razdvajanje uloga, svaka uloga kontrolera, validatora, objekta naredbe, objekta forme, objekta modela itd. se može realizirati kao zaseban objekt
- Moćno i izravo konfiguriranje radnog okruženja i aplikacijskih klasa kao JavaBeans-a, mogućnost ove konfiguracije uključuje lako referenciranje kroz kontekste kao npr. od web kontrolera do poslovnih objekata i validatora
- Prilagodljivost, nenametljivost i fleksibilnost. Mogućnost definiranja potpisa bilo koje metode kontrolera kojeg developer treba, po mogućnosti koristeći neke od anotacija

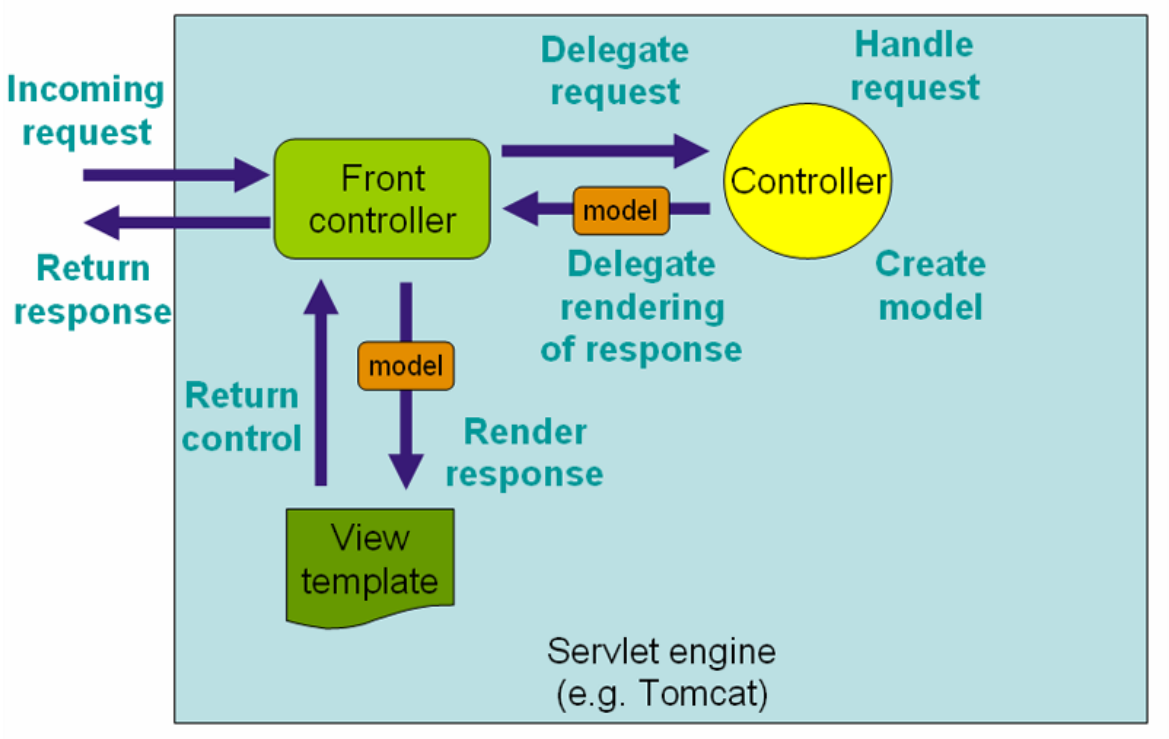
parametara(*@RequestParam, @RequestHeader, @PathVariable* i drugih) za dani slučaj.

- Ponovo iskoristiv poslovni kod nema potrebe za ponovnim pisanjem koda, moguće je koristiti postojeće poslovne objekte kao naredbe ili objekte forme umjesto zrcaljenja istih da bi proširili određenu osnovnu klasu
- Fleksibilan model prijenosa. Model prijenosa sa ime/vrijednost mapom podržava laku integraciju sa bilo kojom tehnologijom pregleda.
- Prilagodljivost lokalne vremenske zone i rezolucije teme, podrška JSP-ovima sa *Spring tag* bibliotekom ili bez nje, podrška za JSTL itd.

3.2. Dispatcher Servlet

„Springovo MVC radno okruženje kao i mnoga druga MVC radna okruženja, temeljena na zahtjevima, dizajnirana su oko centralnog Servleta koji šalje zahtjeve kontrolerima i nudi druge funkcionalnosti koje olakšavaju razvoj web aplikacija. Springov *DispatcherServlet* radi i više od toga, on je u potpunosti integriran s Springovim IoC spremnikom i kao takav dozvoljava programeru da koristi sve druge značajke koje Spring ima.“[1]

Radni tok obrade zahtjeva Spring MVC *DispatcherServlet-a* je ilustriran na sljedećem dijagramu.



Slika 2. Tok obrade zahtjeva DispatcherServleta

Pametni čitač uzoraka će prepoznati da *DispatcherServlet* je produkt „Front Controller“ uzorka dizajna kojeg Spring Web MVC dijeli sa mnogim drugim vodećim web razvojnim okruženjima. *DispatcherServlet* je obični servlet koji nasljeđuje osnovnu HttpServlet klasu i kao takav je deklariran u web.xml datoteci web aplikacije. Developer mora označiti zahtjeve za koje hoće da *DispatcherServlet* rukuje s njima, koristeći URL mapiranje u istoj web.xml datoteci.

```
<web-app>
    <servlet>
        <servlet-name>primjer</servlet-name>
        <servlet-class>
               org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>primjer</servlet-name>
        <url-pattern>/primjer/*</url-pattern>
    </servlet-mapping>

</web-app>
```

Kod 1. Primjer web.xml datoteke

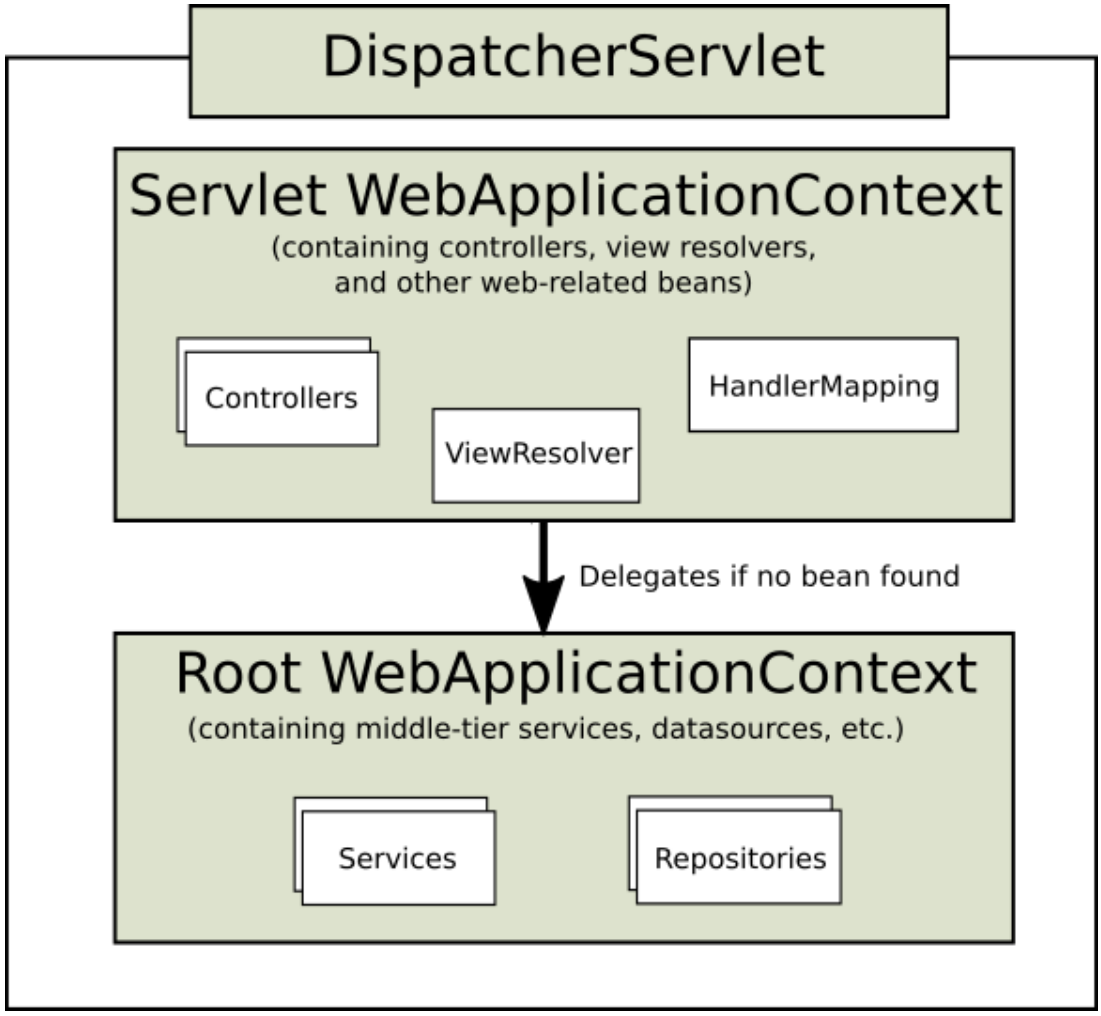
Ovo je Standardna Java EE Servlet konfiguracija, u ovom primjeru pokazuje deklaraciju *DispatcherServleta* i njegovo mapiranje. Svi zahtjevi koji počinju s /primjer bit će rukovani s *DispatcherServlet* instancom nazvanom primjer.

```
public class MyWebApplicationInitializer implements
WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext container) {
        ServletRegistration.Dynamic registration =
container.addServlet("dispatcher", new DispatcherServlet());
        registration.setLoadOnStartup(1);
        registration.addMapping("/primjer/*");
    }
}
```

Kod 2. Primjer programskog konfiguriranja Servlet spremnika

U Servlet 3.0 i višim verzijama okruženja programer ima opciju konfiguriranja Servlet spremnika programski kao u gore navedenom primjeru koji je ekvivalent web.xml datoteke u prethodnom primjeru koda.

WebApplicationInitializer je sučelje koje osigurava Spring MVC koje osigurava da konfiguracije bazirane na kodu je detektirana i automatski korištena da kreira Servlet spremnik. Osnovna klasa ovog sučelja *AbstractDispatcherServletInitializer* olakšava registriranje *DispatcherServlet*-a.



Slika 3. Kontekstna hijerarhija Spring Web MVC-a

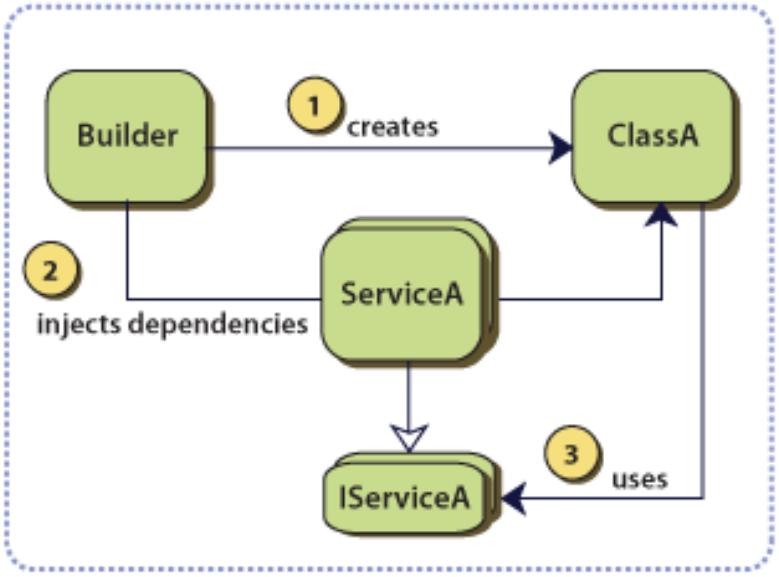
Prilikom inicijalizacije *DispatcherServlet*-a. Spring MVC tražid datoteku pod imenom [naziv-servleta]-servlet.xml u WEB-INF direktoriju web aplikacije i kreira beans-e definirane tamo, zaobilazeći definicije bilo kojeg beansa definiranog s istim imenom unutar aplikacije. Prema web.xml datoteci u kodu 1. u WEB-INF direktoriju bi trebali naći datoteku s putanjom „*/WEB-INF/primjer-servlet.xml*“, koja sadrži sve Spring Web MVC komponente.

4. Injekcija ovisnosti (Dependency injection) uzorak dizajna

Injekcija ovisnosti (Dependency injection) je uzorak dizajna za razvoj softvera koji implementira Inversion of Control (IoC) za rješavanje ovisnosti.

Ovisan je objekt koji može biti korišten (najčešće servis). Injekcija je prosljeđivanje ovisnosti ovisnom objektu (najčešće klijentu) koji koristi taj objekt. Servis je građevni dio klijenta. Prosljeđivanjem servisa klijentu, radije nego dopuštanje klijentu da gradi vlastiti ili traži servis je osnovni zahtjev uzorka dizajna. (prema Prasanna, D. 2009.)

Injekcija ovisnosti dopušta dizajnu programa da prati princip obrnute ovisnosti. Klijent delegira vanjskom kodu odgovornost pružanja njegovih ovisnosti. Klijentu nije dopušteno da poziva kod servisa. Injekcijski kod kreira servise i poziva klijenta da koristi usluge servisa. To znači da klijent ne treba znati ništa o kodu, niti kako kreirati servis, pa niti koje servise koristi. Jedino što klijent treba poznavati je sučelje servisa jer to definira na koji način klijent može koristiti servis.



Slika 4. Konceptualni pogleda na injekciju ovisnosti

Postoje tri osnovna načina kako klijent prihvaća injekciju ovisnosti:

1. Preko setter-a
2. Preko sučelja
3. Preko konstruktora

Setter-ska i konstruktorska injekcija razlikuju se uglavnom po tome kada se mogu koristiti. A kod injekcije preko sučelja klijent ima kontrolu nad injekcijom.

Injekcija ovisnosti odvaja stvaranje ovisnosti klijenta od ponašanja klijenta, što dozvoljava dizajnu aplikacije da prati princip obrnute ovisnosti i princip jedne odgovornosti. Injekcija, osnovna jedinica injekcije ovisnosti, nije novi ili prilagođeni mehanizam. Funkcionira na isti način kao i pružanje parametara.

Injekcija kontrolira pružanje nikad klijent i neovisna ja na koji način se pruža bilo preko reference ili pokazivača. Injekcija ovisnosti uključuje četiri uloge:

1. Servise – objekte koji se koriste
2. Klijente – objekte koji ovise o servisima koje koriste
3. Sučelja – koja definiraju na koji način klijent koristi servise
4. Ubrizgavače (injectors) – koji su odgovorni za kreiranje servisa i injekciju istih u klijenta

Svaki objekt koji je moguće koristiti se smatra servisom, a bilo koji objekt koji koristi neki drugi je klijent. Imena nemaju nikakvu vezu sa svrhom objekata nego sa ulogom koju objekti preuzimaju u jednoj injekciji.

Sučelja su vrata servisa prema klijentima i način na koji klijent zna što očekivati od servisa. Oni mogu biti pravi interface (sučelja) tipovi implementirani od strane servisa pa i apstraktne klase i sami servisi. Jedino zahtijevaju da klijent ne zna što su i da ih ne može konstruirati ili nasljeđivati. (prema Prasanna, D. 2009.g)

Ubrizgavač (injector) prosljeđuje servise klijentu. Često i kreira klijenta. Ubrizgavač (injector) može sačinjavati vrlo kompleksan graf tretirajući neke objekte kao klijente a kasnije kao servise za neke druge klijente. Ubrizgavač (injector) može biti mnogo objekata koji rade zajedno ali nikad ne može biti klijent. Ubrizgavač (injector) je poznatiji pod drugim imenima kao: assembler, pružatelj, factory, container.

4.1. Prednosti i nedostaci

Kao i sve što se koristi u razvoju softvera tako i injekcija ovisnost ima svoje prednosti i nedostatke. Te ovisno o potrebama i o tome što je svrha određenog sustava, arhitekti sustava biraju najbolje komponente i načine za izgradnju. [1]

Prednosti injekcije ovisnosti su:

- Dopušta klijentu da bude konfigurabilan tj. da se jedino ponašanje klijenta popravlja. Klijent može koristiti sve što podržava sučelje koje klijent koristi.

- Injekcija ovisnosti se može koristit za eksternalizaciju sistemskih konfiguracijskih detalja u konfiguracijske datoteke što dopušta sistemu da se rekonfigurira bez ponovne kompilacije. Različite konfiguracije mogu biti pisane za različite implementacije komponenti.
- Iz razloga što Injekcija ovisnosti ne zahtijeva promjene u ponašanju koda, može se primijeniti za refaktoriranje legacy koda. Krajnji rezultat je da su klijenti više nezavisni i imaju lakši pristup za jedinično testiranje u izolaciji koristeći lažne objekte koji simuliraju druge objekte koji nisu pod testom
- Injekcija ovisnosti dopušta klijentu da ukloni svo znanje o konkretnoj implementaciji koju treba koristiti. Ovo pomaže izolirati klijenta od utjecaja u promjeni dizajna i defekata. Iskorištava bolje ponovo iskorištavanje, testiranje i održavanje.
- Smanjenje redundantnog koda u objektima aplikacije iz razloga što sav posao inicijalizacije i postavljanja ovisnosti obavlja pružatelj komponenti
- Injekcija ovisnosti dopušta istovremen i samostalan razvoj. Dva programera mnogu neovisno razvijati klase koje koriste jedna drugu, poznajući samo sučelja preko kojih klase komuniciraju. Dodaci su često razvijeni od treće strane koja nikad nije imala nikakvu komunikaciju s programerima koji su razvijali proizvod koji koristi dodatak.
- Injekcija ovisnosti smanjuje spregu između klasa i njezinih ovisnosti

Nedostatci Injekcije ovisnosti su:

- Injekcija ovisnosti kreira klijente koji zahtijevaju da konfiguracijski detalji budu dostavljeni preko koda. Ovo se može pokazati tegobnim kada su očite zadana konfiguracije dostupne.
- Injekcija ovisnosti može otežati čitanje koda jer razdvaja ponašanje od izgradnje. Što znači da programer mora posvetiti više vremena dokumentaciji kako bi proučio način kako susta funkcionira
- Injekcija ovisnosti zahtijeva više planiranja načina na koji će se pisati kod i implementirati jer klijent ne može pozivati direktno nego mora dati zahtjev servisu i servis mora odobriti i osigurati da je klijent dobio potrebne dijelove
- Injekcija ovisnosti miče kompleksnost iz klasa u veze među klasama što nije uvijek poželjno i lako za održavanje

4.2. Primjeri

U sljedećem primjeru, Klijent klasa sadržava varijablu tipa Servis koja je inicijalizirana u konstruktoru. Klijent kontrolira koja je implementacija servisa korištena i kontrolira kreiranje.

```
 // Primjer bez injekcije ovisnosti
public class Klijent {
    // Unutarnja referenca na servis koji koristi klijent
    private Servis servis;

    // Konstruktor
    Klijent() {
        // Specificira posebnu implementaciju u konstruktoru umjesto da
        //koristi injekciju ovisnosti
        this.servis = new PrimjerServisa();
    }

    // Metoda u kojoj klijent koristi servis
    public String pozdrav() {
        return "Pozdrav " + servis.getName();
    }
}
```

Kod 3. Primjer pozivanja metoda bez injekcije ovisnosti

U ovoj situaciji se kaže da klijent ima hardkodiranu ovisnost o PrimjeruServisa.

Tri tipa injekcije ovisnosti:

- Preko konstruktora – servisi su pruženi preko kostruktora klase
- Preko setter-a – klijent postavlja setter metodu koju pružatelj servisa koristi za injekciju ovisnosti
- Preko sučelja – servis pruža metodu koja će pružiti objekt o kojem je klijent ovisan klijentu kojem je pružena metoda. Klijenti moraju implementirati sučelje koje pruža setter metodu koja prihvaća objekt ovisnosti.

Injekcija preko konstruktora zahtijeva da klijent pruži parametre konstruktoru za ovisni objekt.

```
// Konstruktor
Klijent(Servis servis) {
        this.servis = servis;
}
```

Kod 3. Injekcija preko konstruktora

Injekcija preko setter-a zahtijeva da klijent pruži setter metodu za svaku ovisnost.

```
// Setter metoda
public void setServis(Servis servis) {
    // Sačuva referencu atributu servis unutar ovog klijenta
    this.servis = servis;
}
```

Kod 4. Injekcija preko setter metode

Injekcija preko sučelja gdje klijent daje ulogu sučelja setter metodama ovisnosti klijenta.

```
// Servis setter sučelje
public interface ServisSetter {
    public void postaviServis(Servis servis);
}

// Klijent klasa
public class Klijent implements ServisSetter {
    // Unutarnja referenca na servis
    private Servis servis;

    // Postavlja servis koji klijent koristi.
    @Override
    public void postaviServis(Servis servis) {
        this.servis = servis;
    }
}
```

Kod 5. Injekcija preko sučelja

5. Inversion of Control (IoC) uzorak dizajna

U softverskom inženjerstvu Inversion of Control (IoC) opisuje dizajn u kojem posebno pisani dijelovi programa primaju tok kontrola od generičkih biblioteka. Arhitektura softvera s ovim dizajnom je obrnula kontrolu za razliku od tradicionalnog proceduralnog programiranja. U tradicionalnom programiranju, posebno pisani kod koji izražava svrhu programa poziva biblioteku da se pobrine za generičke zadatke, ali sa inverzijom kontrole, kod koji se iznova koristi poziva se u posebno pisani ili specifičan za zadatak kod.
IoC se koristi da poveća modularnost programa i učini ga proširivim, također ima i primjene u objektno orijentiranom programiranju i drugim programskim paradigmama. IoC je povezan s dependency inversion principom ali je različit od njega po pitanju razdvajanja ovisnosti između slojeva visoke razine i slojeva niske razine kroz dijeljene apstrakcije.(prema Prasana [4])

Na primjer s tradicionalnim programiranjem, glavna funkcija aplikacije može pozvati funkcije koje pozivaju Menu biblioteku da pokaže moguće izbore i napravi upit za korisnika da odabere jedan. Biblioteka bi u tom slučaju vratila odabranu opciju kao vrijednost pozvane funkcije, te nakon toga glavna funkcija koristi ovu vrijednost da izvrši pripadajuću naredbu. Ovaj stil je bio čest u sučeljima temeljnim na tekstu, npr. email klijent prikazuje prozor s naredbama da učita novu poštu, odgovori na postojeću, kreira novi mail itd. i izvršavanje programa bi se prekinulo dok korisnik ne bi unio naredbu.

Sa inverzijom kontrole, s druge strane, program bi bio napisan koristeći razvojno okruženje koje poznaje uobičajena bihevioralne i grafičke elemente, kao Windows sisteme, menije, kontroliranje miša, itd. Programerov kod ispunjava praznine za radno okruženje, kao što su dodavanje stavki tablici menija, i pridruživanje koda za svaku od njih, ali radno okruženje je to koje nadgleda korisnikove radnje i poziva metode kada korisnik odabere određenu stavku kao npr. „Kreiranje novog maila“.

Inverzija kontrole nosi snažnu konotaciju da se ponovo uporabljiv kod i kod vezan za problem mogu razvijati neovisno jedan o drugom iako rade zajedno u aplikaciji.
Prema (Fowler [5])Inverzija kontrole služi za sljedeće svrhe dizajna:

- Da razdvoji izvršenje zadatka od implementacije
- Da fokusira modula na zadatak za koji je razvijen

- Da oslobodi module od brige kako drugi sistemi rade i umjesto toga usredotoče se na svoj zadatak
- Da spriječi popratne probleme prilikom zamjene modula

5.1. Primjeri

```
public class ServerFacade {
   public <K, V> V respondToRequest(K request) {
      if (businessLayer.validateRequest(request)) {
         DAO.getData(request);
         return Aspect.convertData(request);
      }
      return null;
   }
}
```

Kod 6. Primjer koda koji prati IoC metodologiju

Ovaj osnovni Java primjer koda prati IoC metodologiju. Važno je da u *ServerFacade* da su mnoge pretpostavke načinjene o podacima vraćenima od strane *data access object-a (DAO).*

Iako sve ove pretpostavke mogu vrijediti u nekom trenutku, one razdvajaju implementaciju *ServerFacade* na DAO implementaciju. Dizajniranjem aplikacije na način kako bi inverzijom kontrole predao kontrolu u potpunosti DAO objektu. Kod bi izgledao ovako:

```
public class ServerFacade {
   public <K, V> V respondToRequest(K request, DAO
dao) {
      return dao.getData(request);
   }
}
```

Kod 7. Primjer načina konstruiranja metode prema IoC principu

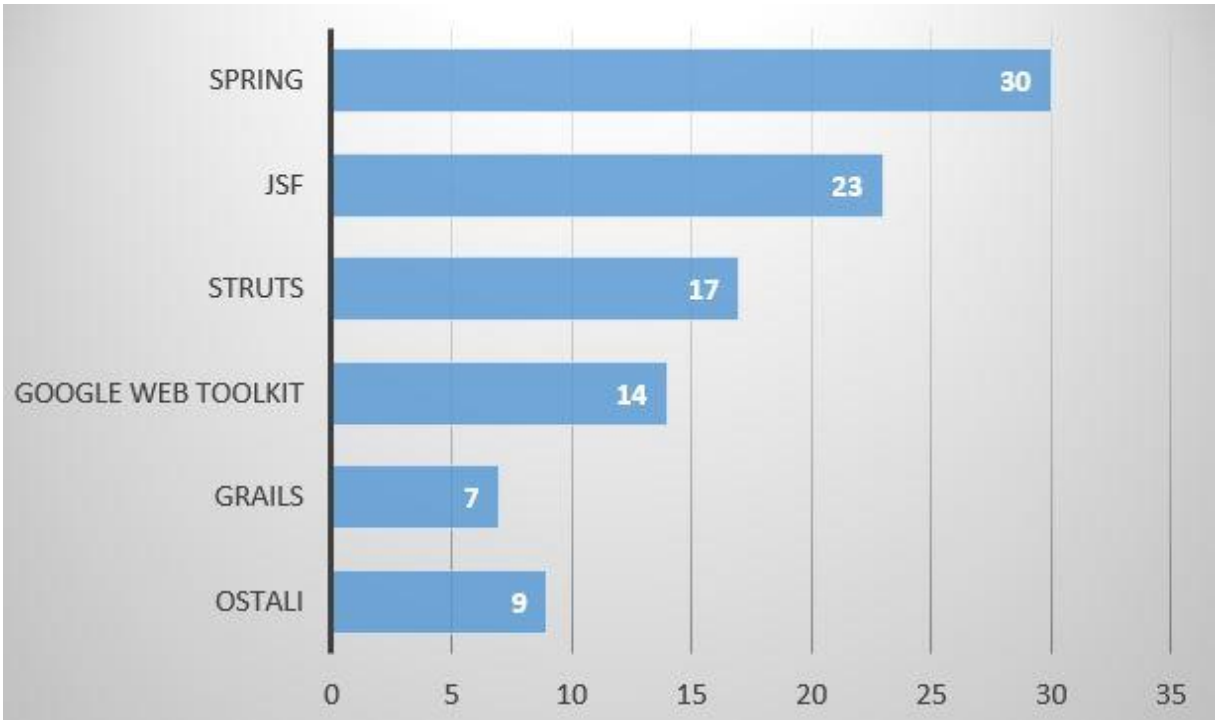
Ovaj primjer pokazuje način na koji je metoda konstruirana određuje je li IoC korišten. Stvar je u načinu na koji su parametri iskorišteni koji definiraju IoC. Ovo je nalik stilu prosljeđivanju poruka koji neki objektno orijentirani programi koriste.

6. Usporedba Spring razvojnog okruženja s ostalim web razvojnim okruženjima

Web razvojna okruženja su jako različita i tipično su kreirana za različite razloga i za postizanje različitih ciljeva. Postoje mnoge značajke koje mogu utjecati na odluku koje razvojno okruženje koristiti na nekom projektu ali na kraju krajeva sve se svodi na tip aplikacije koja se izrađuje.

U izradi modernih aplikacija postavlja se pitanje „Zašto trebamo web razvojna okruženja ?“, a odgovor je jednostavan jer nije jednostavno izraditi aplikaciju s višom razinom kompleksnosti i postići čist kod. Zbog toga web razvojna okruženja su se isturila kao najbolje rješenje i za funkcionalni dio i nefunkcionalni dio aplikacije.

Na uzorku od 1800 programera, web stranica „The curious coders“ je dobila sljedeću statistiku.



Slika 5. Statistika korištenja web razvojnih okruženja

Iz prikazane statistike se vidi da je Spring najpopularnije razvojno okruženje te zatim JSF, Struts i Google web toolkit kao najpopularniji. Što samo po sebi ne govori ništa osim da za Spring razvojnim okruženjem postoji najveća potreba.

Napravljena usporedba se temelji na sljedećim kategorijama:

- Brzo prototipiranje aplikacije
- Kompleksnost razvojnog okruženja
- Lakoća korištenja
- Dokumentacija i zajednica
- Ekosustav razvojnog okruženja
- Propusnost / skalabilnost
- Održavanje koda / ažuriranje

Brzo prototipiranje aplikacije podrazumijeva lakoću izrade prototipa aplikacije bilo za početnika bilo za iskusnog programera i u ovom dijelu se ocjenjuju sposobnost brze izrade sadržaja od nule.

U dijelu kompleksnosti razvojnog okruženja istražuje se arhitektura i izrada samog razvojnog okruženja ta na koji način kompleksnost istih utječe na programera jer što izabere u razvoju mora održavati u radu.

U dijelu o lakoći korištenja se radi o tome koliko je lagano krenuti raditi s određenim radnim okruženjem i koliko je lagano nastaviti raditi tokom izrade aplikacije.

U dijelu o dokumentaciji i zajednici fokus se stavlja na opseg i ažuriranost dokumentacije te na veličinu i mogućnosti interakcije s zajednicom.

Ekosustav razvojnog okruženja pokriva koliko je opširno okruženje u biti, njegove funkcionalnosti i njegovu integraciju s drugima. Jer vrlo često velik ekosistem može značiti veliku kompleksnost ali to nije slučaj uvijek.

Propusnost i skalabilnost su vrlo važni faktori da programer ne bi stvarao uska grla u aplikaciji zbog razvojnog okruženja. Ova kategorija gleda na to što razvojno okruženje može ponuditi da pomogne kada je u pitanju razvoj aplikacije s više korisnika i zahtijeva nego linija koda.

Dio o održavanju koda i ažuriranju aplikacije prvenstveno gleda na lakoću izvršavanja tih funkcija jer planiranje za budućnost čuva programera od bespotrebnih promjena nakon izdavanja aplikacije.

Usporediti ćemo dva najpopularnija razvojna okruženja u statistici, prema svakoj grupi bodujući svako prema mogućim saznanjima o njima.

6.1. Usporedba Spring MVC i Java Server Faces (JSF) razvojnog okruženja

6.1.1. Brzo prototipiranje aplikacije

Prema iskustvima i komentarima programera Spring MVC nije najbolja opcija za kreiranje aplikacije brzo i čisto. Vrlo je opsežno razvojno okruženje i dosta teško naučiti za početnika. Za brzi obrazac uvijek se može preuzeti neki primjer projekta i skinuti bespotrebno, za što također treba određena razina znanja, ali to također uzima vremena i nema uvijek željenog projekta. (prema Spring službenoj dokumentaciji [1])

Spring Roo, projekt Spring zajednice koji podržava Spring razvojno okruženje, Spring sigurnost i Spring web tok, je Spring-ov „convention-over-configuration“ rješenje za brzu izradu aplikacija. Fokusira se na produktivnost Java platforme, upotrebljivost, izbjegavanje zaključavanja i proširivost preko dodataka. Ima mnogo potencijala i trenutno je u fazi razvoja. Krajnji zaključak je da je mnogo predznanja potrebno da bi se krenulo u razvoj prototipa aplikacije i nije lako za početnika izraditi prototip aplikacije u ovoj tehnologiji.

JSF kao ni Spring MVC nije baš najbolje rješenje za brzo prototipiranje jer automatska generacija koda nije ugrađena značajka i prototipi aplikacije zahtijevaju konfiguracije kao i puna aplikacija. Ovo nije isključivo greška razvojnog okruženja jer se dosta oslanja na Java EE specifikacije. Doduše JSF ima neke dosta korisne Maven arhetipove koji pružaju dobru početnu poziciju za osnovnu aplikaciju. Prototipiranje se može postići s širokim rasponom projektnih primjera dostupnih online. Najveća prednost za produktivnost JSF razvojnog okruženja su čarobnjaci koji generiraju većinu ponavljajućeg koda i konfiguracija. (prema E. Burns i suradnicima [4])

Na kraju krajeva malu prednost u ovom dijelu ima JSF razvojno okruženje zbog širokog raspona projektnih primjera koji su dostupni programeru za preuzimanje.

6.1.2. Kompleksnost radnog okruženja

Spring razvojno okruženje je jako kompleksno iako je sama arhitektura vrlo jednostavna ali i dalje ima mnogo slojeva i apstrakcija koje često otežavaju traženje greške ako nešto pođe po krivu. Također jako ovisi o samoj jezgri Spring-a. Na prvi pogled izgleda teže nego što u biti i je, ali ako uzmemo u obzir da je jedno od starijih i zrelijih razvojnih okruženja, koje i ima brojne načine za proširivanje i konfiguraciju, nije tako kompleksno kao što se čini na prvi pogled.

JSF je vrlo kompleksan i to je najveća mana razvojnog okruženja, međutim opet to nije odgovornost isključivo samog razvojnog okruženja nego odgovornost preuzimaju Java EE

specifikacije i nefleksibilnost dostupnog vremena izvršavanja. Postoje implementacije drugih JSF značajki koje dozvoljavaju korištenje spremnika kao što su Tomcat, što smanjuje kompleksnost izrazito za razliko od pokretanja cijelog servera za Java Enterprise aplikaciju.

S obzirom na sve navedeno sama razvojna okruženja su dosta kompleksna i teško je odrediti koje je kompleksnije stoga na kraju krajeva kompleksnost im je jednaka

6.1.3. Lakoća korištenja

Spring je u najmanju ruku opsežan, i kao rezultat toga nije jednostavan za naučiti u jednom danu i koristiti. Kako bi programer iskoristio Spring u potpunosti mora znati kako on funkcionira kao cjelina što je puno teže za savladati. Postoji gomila tečajeva na internetu kao i opširna dokumentacija s primjerima aplikacija koje se mogu koristiti kao temelji za neke druge aplikacije ali općenito Spring je tehnologija za izgradnju ozbiljnih i kompleksnih aplikacija s čvrstim temeljima, bogatim korisničkim sučeljima.

JSF pokušava pružiti razvojno okruženje lako za korištenje i stvaranje web komponenti koje se mogu višekratno koristiti unutar aplikacije bez velike krivulje učenja. JSF je vrlo lagan za početak razvoja s njim jer ne treba nikakva dodatna preuzimanja ili konfiguracije zbog toga što sav potrebni kod je u paketu u bilo kojem Java EE kompatibilnom serveru. JSF ima ugrađenu podršku na serveru aplikacije. Na internetu postoje vrlo dobri tečajevi na Oracle web stranici kao i na nekim stranicama za učenje.

S obzirom na ovo JSF je puno lakši za korištenje od Spring razvojnog okruženja jer ne treba nikakva posebna konfiguracija jer dolazi u paketu sa Java EE serverom, a niti dodatna preuzimanja jer Spring mora dodatno preuzeti JAR(Java Archive) datoteke potrebne za rad.

6.1.4. Dokumentacija i zajednica

Biti najkorištenije razvojno okruženje ima zasigurno i svoje prednosti, što se posebno vidi na količini informacija za Spring razvojno okruženje. Službena stranica sadrži mnogo dijelova s uputama što tekstualnih, što u video formatu i jednako su korisne za početnike i za iskusne programere. Zbog popularnosti i zajednica je jako velika i postoje mnoge Spring grupe koje razgovaraju i u kojima je moguće potražiti bilo kakav savjet vezan za probleme u razvoju. Spring službena stranica se još dodatno potrudila da obavještava sve korisnike putem bloga ili redovitih e-mail obavijesti o svim novinama vezanim uz Spring.

JSF ima podršku kao nijedno razvojno okruženje za Javu zbog toga što je u potpunosti podržano od strane Oracle-a. JSF ima zajednicu hijerarhijski organiziranu tako da su na vrhu Oracle i njegovi zaposlenici kojima je plaćeno da pišu dokumentaciju i kreiraju primjere koji

pomažu pri razvoju. No nažalost većina dokumentacije koju pruža Oracle se temelji na njihovim proizvodima (NetBeans i GlassFish server) što znači ukoliko to nisu izbori programera da mora tražiti izvore s neke druge strane. Ovo nije nužno loša stvar jer postoji velik izbor dokumentacije izvan kruga Oracle dokumentacije ali nažalost nije opširna i ne pokriva sve značajke koje pokriva Oracle dokumentacija. (prema E. Burns [4])

Što se dokumentacije i zajednice tiče jako su slična oba razvojna okruženja no ipak malu prednost ima Spring jer je dokumentacija neovisna o tome kakvo radno okruženje koristi programer za razvoj aplikacije.

6.1.5. Ekosustav razvojnog okruženja

Ekosustav Spring MVC razvojnog okruženja je vrlo dobro razvijen. Temelji se i ovisi o Spring jezgri ali može imati mnogo koristi od alata kao što su Spring Roo i Spring Tool Suite IDE. Nema problema s Maven ovisnostima jer je sve javno dostupno. Postoje i rješenja treće strane kao MyEclipse za Spring koje uključuje konzolne mogućnosti za Spring MVC.

Ekosustav JSF razvojnog okruženja uključuje biblioteke komponenti kao što su Richfaces, OmniFaces, Primefaces, IceFaces no prava vrijednost je to što je dio Java EE specifikacija. JSF se lako može dodati postojećim Java EE okolišima i istog trena primiti sve koristi naslijeđene od Java EE ekosustava. JSF može biti iskoristi spremnike koji nemaju EE standarde ali imaju javno dostupnu podršku kao što su Apache Foundation s svojim MyFaces implementacijom JSF specifikacija.

Nemoguće je odrediti koje razvojno okruženje ima bolji ekosustav jer su dosta slični sa svojim manama i prednostima stoga je rezultat na kraju neriješen i ovisi dosta o tome koliko je koji programer upoznat s kojim razvojnim okruženjem.

6.1.6. Propusnost i skalabilnost

Spring aplikacije su namijenjene skaliranju jer je Spring općenito korišten u aplikacijama velikih razmjera diljem svijet. Sadrži sve potrebne komponente za paralelno procesuiranje i rješenja kao što je EhCache koje se mogu vezati vrlo lako kako bi skalirali memoriju. Spring Batch omogućava izgradnju višedretvenih aplikacija, podijeljenih aplikacija i procesuiranje podataka velikog obujma. Spring aplikacije se mogu modulirati i različiti moduli mogu biti na različitim serverima. Ove komponente mogu komunicirati međusobno zahvaljujući Java servisu za slanje poruka (Java Messaging Service – JMS).

JSF aplikacije mogu imati dobre performanse ali u stvari prednosti u izvođenju dolaze od grupiranja Java EE aplikacijskih servera. JSF sam po sebi ne podržava eksplicitnu potporu

za asinkrone pozive ali oslanja se na Java EE specifikacije da pruže anotacije za poslovnu logiku.

6.1.7. Održavanje koda i ažuriranje

Spring je jako masivno i nabijeno razvojno okruženje bez obziran na to što je usvojilo Model-Pogled-Kontroler paradigmu. Ažuriranje i održavanje koda je izvodivo ako je programer otprije upoznat s zamršenosti razvojnog okružena kao i samog projekta. Ako programer nije iskusan i po prvi puta se susreće s tim zadatkom onda može teći jako sporo. No i dalje anotacije omogućavaju pristup svakakvim vrstama podataka na deklarativan način i ovako nije potrebno mnogo programiranja.

JSF iskorištava postojeću primjer dizajna kao što je MVC da pomogne potaknuti održivost i čitkost. Razvojno okruženje samo po sebi nema neke značajke koje posebno utječu na čitkost ili održavanje koda. Na kraju krajeva može se reći da JSF koristi čistu Java sintaksu u vrlo dobro definiranom primjeru da kreira projekte koje bilo koji iskusan Java EE programer može jednostavno ući u projekt i raditi na njemu.

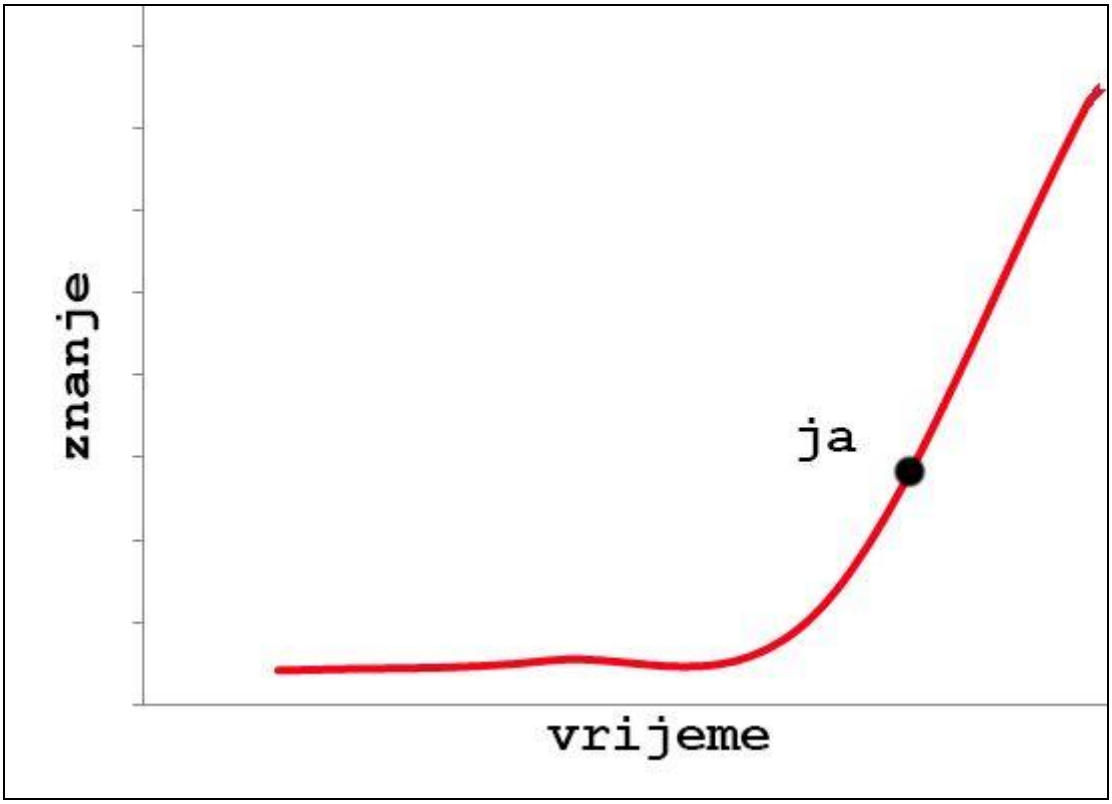
6.2. Zaključak usporedbe

Iako oba razvojna okruženja imaju svoje prednosti i mane Spring u nekoj mjeri zaostaje za JSF-om no ipak ostaje najpopularnije razvojno okruženje prvenstveno zbog svoje dokumentacije koja ne ovisi o radnom okruženju programera što je slučaj kod JSF razvojnog okruženja. No ako programer radi u Oracle radnom okruženju u tom slučaju opet ima prednost zbog toga što ima puno više materijala za početnike. Sve u svemu tijesna pobjeda za JSF razvojno okruženje.

7. Izrada aplikacije u Java Spring razvojnom okruženju

7.1. Uvod

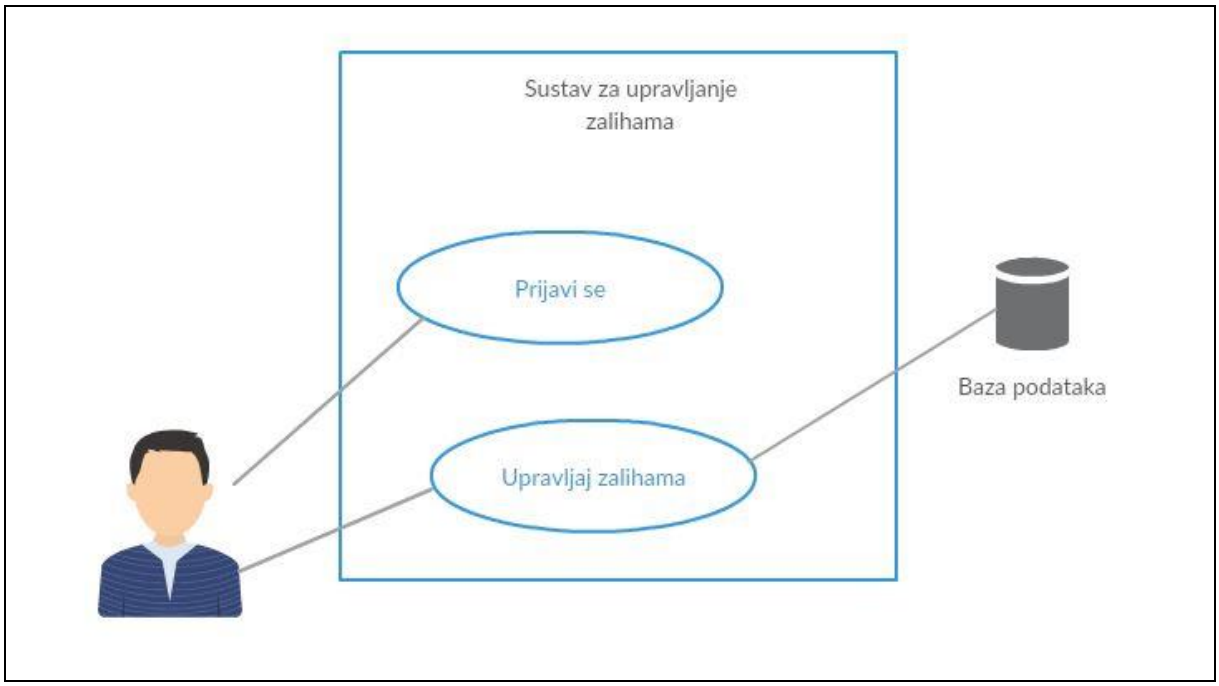
Izrada aplikacije u Java Spring razvojnom okruženju na prvi pogled izgleda jako jednostavno što i vjerojatno stoji za osobu s godinama iskustva u razvoju takvih i sličnih aplikacija. Za valja spomenuti krivulju učenja Spring-a koja većinu bila jako sporo rastuća do trenutka kada svladaju osnovne koncepte i izrade osnovnu aplikaciju pomoću Spring razvojnog okruženja.



Slika 6. Krivulja učenja rada u Java Spring razvojnom okruženju

Za početnika u Java programiranju, a posebice u Java Web programiranju krivulja učenja će izgledati kao ova gore. Dakle jako sporo se usvajaju novi koncepti jer, potrebno je prvo prebaciti se na način funkcioniranja web aplikacija, servera itd. No nakon usvajanja početnih koncepata sve malo po malo ide naprijed tako da krivulja učenja jako brzo raste.

Aplikacija koju sam izradio kao praktični dio završnog je osnovni sustav za upravljanje zalihama. Unos količine, cijene i naziva proizvoda. Dijagram korištenja bi izgledao otprilike ovako.



Slika 7. Dijagram korištenja aplikacije

Aplikacija je vrlo jednostavna ali je ujedno najbolji način za objašnjenje temeljnih koncepata Spring MVC razvojnog okruženja kao što su odvajanje koda prema MVC višeslojnom principu tj. odvajanje formi unosa od funkcionalnosti. Aplikacija se sastoji od baze, te forme za prijavu i upravljanje zalihama. Kao što dijagram opisuje korisnik se prijavljuje za rad u aplikaciji, gdje mu se otvara prozor za upravljanje zalihama, tu korisnik ima opcije uvida u stanje zaliha, unos promjena, brisanje proizvoda i unos novih proizvoda. Kroz izgradnju aplikacije korišteni su neki koncepti koji će biti bolje objašnjeni kroz sljedeća poglavlja.

7.2. Izrada baze podataka

Za izradu baze podataka korišten je WAMP server na kojeg je podignuta MySQL baza podataka, koja je jako jednostavna i sastoji se samo od dvije tablice, a to su tablica proizvoda sa odgovarajućim atributima (id, Ime, Cijena, Opis, Količina) i tablica korisnika s odgovarajućim atributima (id, Ime i prezime, Korisničko ime, Lozinka). Za te tablicu radimo odgovarajući Data Access Object koji u koji prepisujemo atribute i metode za upravljanje spremanjem i čitanjem iz baze.

7.3. Konfiguriranje aplikacije i radnog okruženja

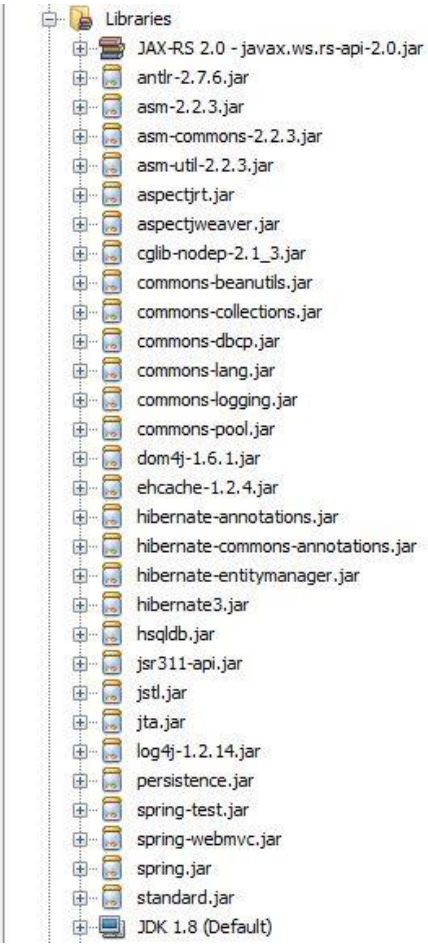
Za izradu aplikacije korišten je NetBeans razvojno okruženje no moguće je koristiti bilo koje razvojno okruženje pa čak i najosnovniji uređivač teksta. S obzirom da je korišten NetBeans bilo je lako postaviti radno okruženje za rad sa Spring-om. Bilo je potrebno skinuti s interneta Spring biblioteke, te postaviti web.xml datoteku da čita DispatcherServlet, nakon čega je datoteka izgledala ovako:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >
    <servlet>
    <servlet-name>springapp</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>
      index.jsp
    </welcome-file>
  </welcome-file-list>
</web-app>
```

Kod 7. web.xml datoteka nakon postavljanja radnog okruženja

Ovo je poznat koncept otprije jer je spomenut ranije kao najobičnije postavljanje putanje da poslužitelj prepozna da se radi o Spring MVC-u. Nakon postavljanja ove datoteke, kreirati će se datoteka čija je putanja „../WEB-INF/springapp-servlet.xml“. Nakon toga dolazimo do učitavanja Spring biblioteka za rad koje je moguće skinuti sa stranica Spring

razvojnog okruženja. One se učitavaju u datoteku pod nazivom „Libraries“. Koja nakon toga izgleda ovako

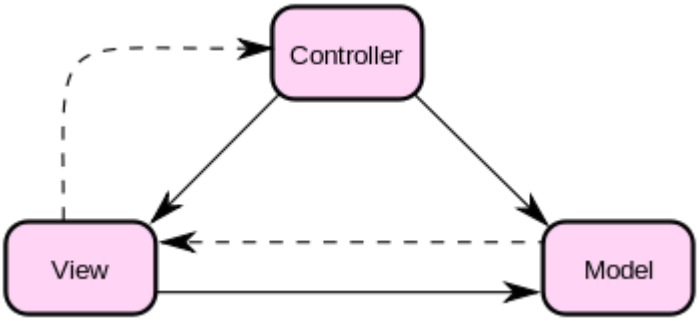


Slika 8. Datoteka Libraries aplikacije

Neke od biblioteka su standardne no neke se koriste prvenstveno za Spring i one sve imaju taj naziv u sebi, možda je još važno spomenuti commons-logging.jar uz koji je lakše raditi debug aplikacije nakon izdavanja aplikacije na server.

7.4. Kreiranje kontrola i pogleda

Model pogledi i kontrole (Model View Controller – MVC) je obrazac softverske arhitekture. Koristi se u softverskom inženjeringu za odvajanje pojedinih dijelova aplikacije u komponente ovisno o namjeni komponenti. Model se sastoji od podatka, što su u našem slučaju proizvodi, poslovnih pravila, logike i funkcija ugrađenih u poslovnu logiku. Pogled je bilo kakav prikaz podataka kao što su forme, tablice ili dijagrami. Moguće je razviti prikaz kroz više pogleda. Kontrole ili upravitelji prihvaćaju ulazne podatke i pretvara i u zahtjeve pogledu ili modelu. --



Slika 9. Prikaz načina komunikacije među komponentama MVC obrasca

Komponente imaju međusobnu interakciju što je vrlo važno naglasiti, ona se odvija na sljedeći način:

- Kontrola ili upravitelj može slati naloge modelu kojima ažurira njegovo stanje. Također može slati naredbe u poglede kojima mijenja prikaz modela
- Model dojavljuje sebi pridruženim pogledima i upraviteljima kada je došlo do promjene u njegovom stanju. Ove dojave omogućuju pogledu da prikaže obnovljeno stanje modela, a upravitelju promjenu dostupnog skup naredbi
- Pogled zahtjeva od modela informacije potrebne za stvaranje prikaza modela korisniku

Pošto je Spring ujedno i MVC (model, pogled, kontrole) razvojno okruženje potrebno je po tom principu i kreirati aplikaciju. Kako aplikacija ima više pogleda za svaki je potrebno napraviti kontrolu koja će upravljati njime. U ovom dijelu biti će prikazan primjer jednog pogleda i jedne kontrole i detaljno opisan što, kako i na koji način radi taj dio aplikacije. Prvenstveno se kreira model koji predstavlja objekt aplikacije. Model sadrži osnovne atribute objekta iz stvarnog svijeta, u ovo slučaju proizvoda, to su jedinstvena identifikacija, naziv, opis te cijena i količina. Model također sadrži i metode za dohvaćanje i postavljanje tih atributa.

```
package springapp.domain;

import java.io.Serializable;

public class Product implements Serializable {

    private int id;
    private String name;
    private Double quantity;
    private String description;
    private Double price;

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

    public int getId() {
        return id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void getName() {
        return this.name
    }

    public void setQuantity(Double quantity) {
        this.quantity = quantity;
    }

    public void getQuantity() {
        return this.quantity;
    }


    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }

    public String toString() {
        StringBuffer buffer = new StringBuffer();
        buffer.append("Description: " + description + ";");
        buffer.append("Price: " + price);
        return buffer.toString();
    }
}
```

Kod 8. Model proizvoda iz izrađene aplikacije

Nakon modela dolazi na red izrada pogleda u ovom slučaju. Na jednom primjeru je prikazano način kreiranja pogleda. Pogledi su obični Java Server Page datoteke, temeljene na HTML-u i Java kodu unutar njih.

```
<html>
  <head>
      <title><fmt:message key="title"/></title>
      <style>
          .error { color: red; }
      </style>
  </head>
  <body>
      <h1><fmt:message key="priceincrease.heading"/></h1>
      <form:form method="post" commandName="addproduct">
          <table>
              <tr>
                  <td>
                      <form:label path="name">Name</form:label>
                  </td>
                  <td>
                      <form:input path="name" />
                  </td>
              </tr>

              <tr>
                  <td>
                       <form:label
                    path="description">Description</form:label>
                  </td>
                  <td>
                      <form:input path="description" />
                  </td>
              </tr>
              <tr>
                  <td>
                      <form:label path="price">Price</form:label>
                  </td>
                  <td>
                      <form:input path="price" />
                  </td>
              </tr>
              <tr>
                  <td>
                      <form:label path="quantity">Quantity</form:label>
                  </td>
                  <td>
                      <form:input path="Quantity" />
                  </td>
              </tr>
              <tr>
                  <td colspan="2">
                      <input type="submit" value="Execute"/>
                  </td>
              </tr>
          </form:form>
          <a href="<c:url value="hello.htm"/>">Home</a>
  </body>
</html>
```

Kod 9. Pogled na unos proizvoda u bazu

```
package springapp.web;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import org.springframework.web.servlet.view.RedirectView;
import springapp.domain.Product;
import springapp.service.AddProduct;
import springapp.service.PriceIncrease;
import springapp.service.ProductManager;

public class AddProducatFormController extends SimpleFormController {

    private ProductManager productManager;

    public ModelAndView onSubmit(Object command) throws ServletException
{

        Product prod = new Product();

        prod.setId(((AddProduct) command).getId());
        prod.setDescription(((AddProduct) command).getDescription());
        prod.setPrice(((AddProduct) command).getPrice());
        logger.info("Product added");

        productManager.addProduct(prod);

        logger.info("returning from PriceIncreaseForm view to " +
getSuccessView());

        return new ModelAndView(new RedirectView(getSuccessView()));
    }

    protected Object formBackingObject(HttpServletRequest request)
throws ServletException {
        AddProduct prod = new AddProduct();
        prod.setId(1);
        prod.setPrice(20);
        prod.setDescription("Ime");
        return prod;
    }


    public void setProductManager(ProductManager productManager) {
        this.productManager = productManager;
    }

    public ProductManager getProductManager() {
        return productManager;
    }
```

Kod 10. Kontrola za dodavanje novih proizvoda

Kontrola vezana za gore navedeni pogled se naziva AddProductFormController i dio je springapp.web paketa. Ovaj kontroler jednostavno odrađuje najjednostavniji posao vraćanja unesenih vrijednosti do našeg DataAccesObject-a, koji je veza između modela i baze podataka,

koji nakon toga upisuje proizvod u bazu podataka. Ujedno ovaj kontroler upisuje početne podatke u formu za korisnika, koje korisnik kasnije može izmijeniti.

Za potrebu spajanja modela s bazom podataka potrebno je kreirati objekt za povezivanje s bazom podataka (Data Access Object - DAO) koji pruža apstraktno sučelje za neke tipove baza podataka. Mapiranjem aplikacijskih poziva persistance sloju baze podataka, DAO pruža operacije s bazom podataka bez otkrivanja detalja o bazi podataka što podržava princip o jednoj odgovornosti za jedan sloj. Tradicionalno je povezan s Java EE aplikacijama pa samim time i Spring MVC razvojnim okruženjem.

```
package springapp.repository;

import java.util.List;

import springapp.domain.Product;

public interface ProductDao {

    public List<Product> getProductList();

    public void saveProduct(Product prod);

    public void addProduct (Product prod);

    public void updateProduct (Product prod);

    public void deleteProduct (Product prod);
}
```

Kod 11. Objekt za povezivanje s bazom podataka (Dana Access Object)

Objekt za povezivanje s bazom podataka je sučelje koje se implementira u druge klase aplikacije sadrži metode za spremanje, brisanje, dodavanje i ažuriranje tablice unutar baze podataka. Ovo sučelje se implementira u klasu JdbcProductDao.java datoteku što je vidljivo u sljedećem primjeru koji opisuje logiku tih metoda za upravljanje podacima o proizvodima.

```
package springapp.repository;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.simple.ParameterizedRowMapper;
import org.springframework.jdbc.core.simple.SimpleJdbcDaoSupport;
import springapp.domain.Product;

public class JdbcProductDao extends SimpleJdbcDaoSupport implements ProductDao {

    protected final Log logger = LogFactory.getLog(getClass());

    public List<Product> getProductList() {
        logger.info("Getting products!");
        List<Product> products = getSimpleJdbcTemplate().query(
                "select id,name,quantity, description, price from products",
                new ProductMapper());
        return products;
    }

    public void addProduct(Product prod)
    {
        logger.info ("Adding product");
        String query = "INSERT INTO products (id,name,quantity,description, price)
VALUES (?,?,?,?,?)";
        getSimpleJdbcTemplate().update(query,
prod.getId(),prod.getName(),prod.getQuantity(),
prod.getDescription(),prod.getPrice());
        logger.info ("Product added");
    }

    public void saveProduct(Product prod) {
        logger.info("Saving product: " + prod.getDescription());
        int count = getSimpleJdbcTemplate().update(
            "update products set name=:name, quantity=:quantity,description =
:description, price = :price where id = :id",
            new MapSqlParameterSource().addValue("description",
prod.getDescription())
                .addValue("price", prod.getPrice())
                .addValue("id", prod.getId())
                .addValue("name", prod.getName()))
                .addValue("quantity", prod.getQuantity())
        );
        logger.info("Rows affected: " + count);
    }

    private static class ProductMapper implements ParameterizedRowMapper<Product>{

        public Product mapRow(ResultSet rs, int rowNum) throws SQLException {
            Product prod = new Product();
            prod.setId(rs.getInt("id"));
            prod.setDescription(rs.getString("description"));
            prod.setPrice(new Double(rs.getDouble("price")));
            return prod;
        }
    }
}
```

Kod 11. Klasa sa implementiranim DAO sučeljem

7.5. Izrada testova za aplikaciju

U računalnom programiranju jedinično testiranje je metoda kojom se testiraju pojedini dijelovi izvornog koda, jedan ili više modula ili aplikacija u cjelini. Intuitivno, jedinica je najmanji mogući dio za testiranje. U proceduralnom programiranju jedinica može biti cijeli modul, ali češće je to pojedina funkcija ili procedura. U objektno orijentiranom programiranju to je najčešće cijelo sučelje, ali može biti i pojedina metoda.

Cilj jediničnog testiranja je izolirati svaki dio programa i pokazati da su pojedini dijelovi ispravni. Jedinični test pruža strog pisani ugovor koji dio koda mora zadovoljiti. Pomaže pronaći probleme rano u još u fazi razvoja, podupire promjene jer programer može kasnije unaprijediti biblioteke i biti siguran da modul i dalje radi, pojednostavljuje integraciju jer testiranjem programa prvo pa sume njegovih dijelova i integracijsko testiranje postaje lakše.

Testovi napisani u ovoj aplikaciji su jako jednostavni kao i aplikacija na kraju krajeva, ali kao cilj svakog testiranja tako i ovoga je dokazat da pogreške postoje a ne obrnuto.

```
package springapp.repository;

import java.util.List;
import
org.springframework.test.AbstractTransactionalDataSourceSpringContextTest
s;
import springapp.domain.Product;

public class JdbcProductDaoTests extends
AbstractTransactionalDataSourceSpringContextTests {

    private ProductDao productDao;


    public void setProductDao(ProductDao productDao) {
        this.productDao = productDao;
    }

    @Override
    protected String[] getConfigLocations() {
        return new String[] {"classpath:test-context.xml"};
    }

    @Override
    protected void onSetUpInTransaction() throws Exception {
        super.deleteFromTables(new String[] {"products"});
        super.executeSqlScript("file:db/load_data.sql", true);
    }

    public void testGetProductList() {

        List<Product> products = productDao.getProductList();

        assertEquals("wrong number of products?", 3, products.size());

    }

    public void testSaveProduct() {

        List<Product> products = productDao.getProductList();

        for (Product p : products) {
            p.setPrice(200.12);
            productDao.saveProduct(p);
        }

        List<Product> updatedProducts = productDao.getProductList();
        for (Product p : updatedProducts) {
            assertEquals("wrong price of product?", 200.12,
p.getPrice());
        }
    }
}
```

8. Zaključak

Spring razvojno okruženje iako jako kompleksno kroz godine se pokazala jako korisno i produktivno, što dokazuje njegova popularnost među programerima gdje je uvjerljivo prvi izbor. Njegove mogućnosti i Model – View – Controler (MVC) princip dizajna aplikacije omogućili su mu razvoj modularnih aplikacija, razdvajanje zadaća po principu jedna funkcionalnost jedan zadatak. Napretkom i razvijanjem alata Spring je ponudio svojim korisnicima i ujedinjen Spring Tool Suite radno okruženje koje je pripremljeno za razvoj Spring aplikacija. Za ubrzani razvoj tu je Spring Roo koji se i sam još razvija ali je tehnologija koja jako obećava.

Dokumentacija za Spring korisnike je jako opširna i nezavisna o radnom okruženju koje programer koristi, dok je zajednica jako velika i spremna pomoći na raznim forumima. Jedini nedostatak Spring razvojnog okruženja je mali broj uputa za čiste početnike što može otjerati dosta potencijalnih korisnika. Potrebno je dosta dobro predznanje iz standard Java programskog jezika kao i Java EE znanje.

Postoje mnoge tehnologije slične Spring razvojnom okruženju, ali Spring jednostavno ima svoju primjenu u razvoju velikih i kompleksnih aplikacija s kompleksnom poslovnom logikom gdje je nenadmašiv u tom segmentu i zadovoljava sve potrebe za razvoj aplikacije. U tome mu pomaže i to što ide ruku pod ruku s uzorkom dizajna Injekcije ovisnosti (Dependency injection) koji se pokazao korisnim u praksi.

Za kraj naglasak na to zašto izabrati Spring za razvoj aplikacije i u kojem slučaju. Definitivno ne za male i srednje aplikacije koje se ne planiraju dalje razvijati i unaprjeđivati jer u tom slučaju postoje puno bolje tehnologije koje će omogućiti puno brže obavljanje posla, Spring je najbolja moguća tehnologija za razvoj aplikacije koja je kompleksna, čiji planirani broj korisnika će prerasti broj linija koda same aplikacije i na kraju krajeva aplikacije koja će biti održavana svakodnevno i kojoj će biti omogućeno proširivanje pomoću dodataka razvijenih od treće strane.

9. Literatura

[1] Spring službena dokumentacija, http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/index.html (dostupno 02.09.2015.g)

[2] Deinum, M., Serneels, K., Yates, C., Ladd, S., Vanfleteren, C. and Vervaet, E. (2012). *Pro Spring MVC*. [S.l.]: Apress.

[3] Prasanna, D. (2009). *Dependency injection*. Greenwich, Conn.: Manning.

[4] Burns, E., Griffin, N. and Schalk, C. (2010). *JavaServer faces 2.0*. New York: McGraw-Hill.

[5] Fowler, M. (2004). Inversion of Control Containers and the Dependency Injection pattern. http://www.martinfowler.com/articles/injection.html dostupno (02.09.2015.g)

[6] Spring upute i vodiči za izradu komponenti aplikacije http://spring.io/guides dostupno (02.09.2015.g)

10. Popis slika i izvori

[1] Slika 1. Pregled arhitekture Spring razvojnog okruženja - (dostupno 2.9.15.) http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/images/spring-overview.png

[2] Slika 2. Tok obrade zahtjeva DispatcherServleta - (dostupno 2.9.15.) http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/images/mvc.png

[3] Slika 3. Kontekstna hijerarhija Spring Web MVC-a - (dostupno 2.9.15.)http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/images/mvc-context-hierarchy.png

[4] Slika 4. Konceptualni pogleda na dependency injection (dostupno 2.9.15) https://i-msdn.sec.s-msft.com/dynimg/IC245731.png

[5] Slika 5. Statistika korištenja web razvojnih okruženja - dostupno (2.9.15) http://zeroturnaround.com/wp-content/uploads/2013/07/Web-frameworks-developer-productivity-report.png

[6] Slika 6. Krivulja učenja rada u Java Spring razvojnom okruženju- rad autora

[7] Slika 7. Dijagram slučajeva korištenja aplikacije - rad autora

[8] Slika 8. Datoteka Libraries aplikacije – rad autora

[9] Slika 9. Prikaz načina komunikacije među komponentama MVC obrasca - (dostupno 2.9.15.) https://hr.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller#/media/File:ModelViewControllerDiagram2.svg