Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dejan Sakelšak

# Primerjava spletnih ogrodij Spring MVC, Stripes in Apache Tapestry

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Matjaž B. Jurič

Ljubljana, 2014



*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Analizirajte koncepte razvoja spletni aplikacij. Proučite področje spletnih ogrodij. Podrobno analizirajte ogrodja Spring MVC, Stripes in Apache Tapestry. Ogrodja primerjajte iz vidika funkcionalnosti, hitrosti delovanja, možnosti za razvoj in vzdrževanje spletnih aplikacij ter identificirajte ključne prednosti in pomanjkljivosti. V izbranem ogrodju izdelajte primer spletne aplikacije in pri tem sledite uveljavljenim vzorcem in dobrim praksam.

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Dejan Sakelšak, z vpisno številko **63030049**, sem avtor diplomskega dela z naslovom:

*Primerjava spletnih ogrodij Spring MVC, Stripes in Apache Tapestry*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Matjaža B. Juriča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 28. 4. 2014 Podpis avtorja:

*Na tem mestu bi se zahvalil ženi Taji za vso skrb in podporo, ki mi ju je nudila v času študija, vsej svoji družini pa za podporo in vzpodbudne besede. Zahvaljujem se tudi mentorju, dr. Matjažu B. Juriču, za usmeritve pri izdelavi diplomskega dela.*

Taji in Jerneju.

# Kazalo

# Seznam uporabljenih izrazov in kratic

| kratica | razlaga |
|---|---|
| **API** | Application Programming Interface - vmesnik za programiranje aplikacij. |
| **JSE** | Java Standard Edition - standardna izdaja Jave. |
| **JEE** | Java Enterprise Edition - izdaja Jave za podjetja. |
| **JME** | Java Micro Edition - izdaja Jave za majhne naprave in mobilne telefone. |
| **JCP** | Java Community Process - mehanizem standardizacije tehnologij za platformo Java. |
| **JSR** | Java Specification Request - dokumenti predlagane specifikacije tehnologije. |
| **URL** | Uniform Resource Locator - enolični krajevnik vira. |
| **WWW** | World Wide Web - svetovni širni splet. |
| **Internet** | Omrežje naprav, ki komunicirajo s protokolom TCP/IP. |
| **Web Browser** | Spletni brskalnik je program, ki komunicira s spletnimi strežniki, interpretira in prikazuje vsebino. |
| **Hypertext** | Strukturirano besedilo s povezavami med vozlišči - *hiperpovezavami* - in drugimi strukturnimi lastnostmi. |
| **HTML** | Hypertext Markup Language - hipertekstrovni označevalni jezik. |

| | |
|---|---|
| **CSS** | Cascading Style Sheet - kaskadna lista slogov. |
| **HTTP** | Hypertext Transfer Protocol - prenosni protokol za hipertekst. |
| **XML** | Extensible Markup Language - razširljiv označevalni jezik. |
| **JSON** | JavaScript Object Notation - notacija objektov JavaScript, enostaven opisni jezik, ki ga razume JavaScript. |
| **Web 2.0** | Nova različica spletnega koncepta. |
| **REST** | Representational State Transfer - reprezentativni prenos stanja. |
| **RESTful Web Services** | Spletne storitve REST. |
| **JavaScript** | Skriptni jezik, ki brskalniku določa akcije, s katerimi vpliva na prikaz spletne strani. |
| **AJAX** | Asynchronous JavaScript and XML - asinhroni JavaScript in XML - tehnologija, ki omogoča osveževanje podatkov samo na delu strani. |
| **IPv4** | Internet Protocol version 4 - različica 4 internetnega protokola, specifikacija definira 32 bit internetne naslove oblike 127.0.0.1. |
| **IPv6** | Internet Protocol version 6 - različica 6 internetnega protokola, specifikacija definira 128 bit internetne naslove oblike fe80::224:d7ff:fe73:e014 ter druge izboljšave. |
| **TCP** | Transmission Control Protocol - internetni protokol, ki zagotavlja dostavo podatkov. |
| **TCP/IP** | TCP over Internet Protocol - protokol za nadzor prenosa podatkov nad internetnim protokolom. |
| **LAN** | Local Area Network - omrežje lokalnega območja. |
| **WLAN** | Wireless LAN - brezžično lokalno omrežje. |

| | |
|---|---|
| **WiFi** | Brezžično omrežje WLAN, ki ustreza pravilom konzorcija Wi-Fi. Frekvenca omrežja je 2.4 ali 5 GHz. |
| **DNS** | Domain Name System - sistem za imena domen. Sistem, ki pretvarja človeku razumljiva in berljiva domenska imena v IP-naslove. |
| **Web Services** | Spletne storitve, način za dostop do podatkov, ki je uporaben iz več medijev. |
| **MIME** | Multipurpose Internet Mail Extensions - večnamenska razširitev internetne pošte - definira tip podatka v telesu sporočila npr. *text/plain, text/html, application/json, video/theora itn.*. |
| **NGINX** | Popularen spletni strežnik. |
| **Servlet** | Javanski razred, ki je implementiran po standardu JEE, vsebuje logiko za upravljanje z zahtevki. |
| **Servlet Container** | Javanski program, ki implementira strežniško logiko za servlete po specifikaciji JEE. |
| **DOM** | Document Object Model - model dokumenta in objektov, drevesna predstavitev hierarhične strukture. |
| **CGI** | Common Gateway Interface - splošni prehodni vmesnik, vmesnik med spletnim strežnikom in programom. |
| **JSP** | Java Server Pages - strežniške strani Java so Javanska tehnologija, ki je primerna za implementacijo pogleda. |
| **Scriptlet** | Del Javanske kode, ki je vrinjen med vrstice dokumenta JSP. |
| **JSTL** | JSP Standard Tag Library - standardna knjižnica oznak JSP. |
| **MVC** | Model-View-Cotroller - model-pogled-krmilnik. |
| **BL** | Business Logic - poslovna logika. |

| | |
|---|---|
| **IoC** | Inversion of Control - obrat krmiljenja, skupek konceptov in vzorcev za hitrejši razvoj aplikacij. |

# Povzetek

Diplomsko delo primerja tri ogrodja za razvoj spletnih aplikacij na platformi Java: Spring MVC, Stripes in Apache Tapestry. Primerjava je bila narejena na podlagi izvedbe enake aplikacije z vsakim od ogrodij, in sicer s pomočjo grafov meritev, slik prometa in tabel lastnosti. V okviru analize so bili pregledani podrobnejši opisi posameznih ogrodij, primerjalo se je njihove funkcionalnosti in zmogljivost, ter opozorilo na njihove dobre in slabe lastnosti. Upoštevalo se je tudi težavnost razvoja aplikacij in podprtost z dokumentacijo. Glede na to da je primerjava omenjenih treh ogrodij namenjena v pomoč pri izbiri tehnologij med načrtovanjem projektov, je v sklepnem delu na kratko opisano, katera od ogrodij so najprimernejša za dolgoročne projekte. V pričujočem besedilu je podrobneje predstavljena izvedba aplikacije z ogrodjem Stripes, ki se je v tem primeru izkazalo kot najbolj okretno. V sklepnem delu je na kratko opisano, katera od ogrodij so najprimernejša za dolgoročne projekte.

**Ključne besede:** JavaEE, ogrodje Spring, ogrodje Stripes, ogrodje Apache Tapestry, spletna ogrodja.

# Abstract

The thesis compares three web application development frameworks for the Java platform: Spring MVC, Stripes and Apache Tapestry. The comparison was based on the implementation of the same application with each of the frameworks, and used graphs of measurements, network traffic images and property tables for the analysis. As part of the thesis, detailed descriptions of the frameworks were reviewed, their features and performances were compared as well as their qualities and deficiencies were highlighted. For this comparison we use graphs of measurements, network traffic images and property tables. Some consideration was also given to difficulties of software development and the availability of documentation. Since this comparison is meant to help software developers when selecting technologies for the design of their projects, a brief description on which of the frameworks would be most suitable for long-term projects is given in conclusion. The Stripes framework implementation is described in more detail, as it turned out to be the most agile of the three. In conclusion a brief description is given which of the frameworks is most suited for long-term projects.

**Keywords:** JavaEE, Spring framework, Stripes framework, Apache Tapestry framework, web frameworks.

# Poglavje 1

# Uvod

Za uspešno načrtovanje projekta morajo razvijalci izbrati tehnologije in orodja, s katerimi bodo čim bolje rešili zadano nalogo. Izbira je lažja, če načrtovalec pozna tehnologije, ki so na voljo. To velja tudi v primeru spletnih ogrodij na platformi Java, kjer je poznavanje dobrih in slabih lastnosti ogrodij ključnega pomena. Prava izbira tehnologije je otežena, ker je tržišče preplavljeno s produkti, ki obljubljajo le najboljše.

Napačna izbira tehnologij, predvsem ogrodij, s katerimi ustvarimo celotno aplikacijo, lahko privede do slabe kakovosti končnega produkta, dragega vzdrževanja in nadgradnje ter drugih težav. Pri izbiri so v veliko pomoč primerjave tehnologij, ki so na voljo na raznih spletnih straneh, na portalih skupnosti in pri drugih virih. Pomemben kriterij pri izbiri tehnologije je tudi, kako težavno bo dolgoročno vzdrževanja aplikacije.

Diplomsko delo izmed množice spletnih ogrodij primerja tri ogrodja iz sveta Jave: Spring MVC, Stripes in Apache Tapestry. Primerjava upošteva podatke, pridobljene pri razvoju enake aplikacije z vsemi tremi ogrodji. Pri vseh treh aplikacijah primerjamo naslednje lastnosti: nepredvidene in prikrite težave, ki se pojavijo med razvojem, ter zmogljivost glede na čas odgovora in čas nalaganja. Pomemben kriterij pri primerjavi ogrodij je tudi njihova dokumentacija oziroma stanje dokumentacije, ki je bistveno za razvoj aplikacije.

Cilj diplomskega dela je poenostaviti izbiro najprimernejšega med tremi ogrodji s pregledom in primerjavo lastnosti. Za razvijalca je pomembno, da pozna prednosti in slabosti tehnologij, ki so mu na voljo pri snovanju projektov. Upoštevati je treba več dejavnikov, kot so: vpliv ogrodja na kakovost projektov, njihovo vzdrževanje, nadgradnje in izobraževanje kadra.

Aplikacija, uporabljena za primerjavo, je spletni portal, ki uporabniku omogoča dodajanje in filtriranje konferenc glede na datum začetka ter njihove oznake. Uporabnik z registracijo na portalu ustvari nov račun, se v portal prijavi, dodaja konference in jih pregleduje. Skozi izvedbo vseh treh aplikacij spoznamo potek razvoja z ogrodji, njihovo kompleksnost, značilnosti njihovih funkcionalnosti in stanje dokumentacije. V primerjavo so vključene tudi funkcionalnosti za povezovanje in preverjanje podatkov, za večjezičnost in omogočanje sodobnih prijemov pri razvoju.

# Poglavje 2

# Splet in spletne tehnologije

## 2.1 Splet

Izraz internet pogosto narobe uporabljamo. Internet je omrežje, na katerem so na voljo standardne storitve, kot so svetovni širni splet, elektronska pošta, internetna telefonija in mnoge druge. Kot nestandardne lahko opredelimo storitve, kot je npr. Skype, ki je pod nadzorom enega podjetja in ne deli specifikacije o delovanju z ostalim svetom.

Svetovni širni splet je množica spletnih strani, ki so namenjene prebiranju, in spletnih aplikacij, ki so namenjene interakciji z uporabnikom. V zadnjem desetletju se je pojavil pojem *Web 2.0*, ki klasično predstavo o spletu razširja z dodatnimi tehnologijami. Glavnino teh tehnologij predstavljajo: razširljiv označevalni jezik (ang. Extensible Markup Language - XML), AJAX (Asynchronous JavaScript And XML) in spletne storitve REST (Representational State Transfer Web Services). Splet se je iz medija za iskanje informacij prelevil v tehnologije, ki omogočajo implementacijo naprednih aplikacij za najrazličnejše namene, od spletnih zemljevidov, spletnih klepetalnic in poštnih odjemalcev do spletnih pisarniških paketov, ki omogočajo enostavno sodelovanje, ter multimedijskih portalov. To sta omogočili tehnologiji Adobe Flash in različica 5 hipertekstnega označevalnega jezika (ang. Hypertext Markup Language 5 - HTML5). Spletne storitve REST so tako postale ne-

kaj vsakdanjega, saj omogočajo prikazovanje svežih podatkov in ne zahtevajo osveževanja celotne strani; vse to omogočajo na brskalnikih in aplikacijah za mobilne ter druge naprave.

### 2.1.1 Spletni strežnik

Spletni strežnik je aplikacija, ki čaka na zahtevke in se nanje odzove z odgovorom. Vsebuje lahko zahtevano vsebino ali napako. Po rezultatih spletne strani Netcraft [1] je spletni strežnik Apache na prvem mestu po številu namestitev, sledi mu Microsoftov IIS, vse bolj pa se na tržišču uveljavlja Nginx, ki je v primerjavi z glavnima tekmecema nov.

Na začetku je bil splet najpogosteje poseljen s spletnimi stranmi, ki so bile enostavne datoteke hipertekstnega označevalnega jezika (ang. Hypertext Markup Language - HTML). Te je avtor ustvaril ročno, vanje dodal vsebino in jih objavil na spletnem strežniku. Take spletne strani imenujemo statične, saj mora biti vsaka sprememba vsebine, ki jo vidi uporabnik med obiskom strani, opravljena ročno s spremembo vsebine datoteke. S časom se je vedno bolj začel uporabljati obči prehodniški vmesnik (ang. Common Gateway Interface - CGI), ki omogoča izvajanje skript in programov v ozadju. Na kratko se je skripte v ozadju spletnega strežnika imenovalo skripte CGI. Pogosto je bil jezik za take skripte Perl, lahko pa je to bil kateri koli drug jezik ali program. Na tak način se je uporabnikom omogočila interakcija s strežnikom in izvajanjem logike glede na njihove vnose. Rodile so se dinamične spletne strani in tako imenovane spletne aplikacije.

Skripte CGI se še danes pogosto uporabljajo, čeprav so jih izpodrinile tehnologije, ki so specifične za določene jezike in omogočajo boljšo kontrolo nad izvajanjem. Pri spletnem strežniku Apache imamo module, ki so narejeni za integracijo z določenimi tolmači, npr. `mod_php` za PHP (Personal Home Page) ali `mod_python` za Python.

Dinamične spletne strani omogočajo hranjenje podatkov, procesiranje podatkov, integracijo z drugimi storitvami, izvajanje opravil v ozadju in še mnogo drugih stvari. Tako so se razvili različni sistemi, med katerimi so

najpogostejši sistemi CMS (Content Management System), ki so namenjeni upravljanju vsebine. Avtorji člankov se lahko v sistem prijavijo, vanj vpišejo vsebino, ki se shranjuje v podatkovno bazo, in jo s klikom objavijo.

Java ima nekoliko drugačno zasnovo kot ostale spletne tehnologije, zato spletne strežnike Java imenujemo tudi vsebniki *servletov*. Primer takih spletnih strežnikov so Apache Tomcat in Jetty ali polni aplikacijski strežniki, kot so: JBoss, Oracle WebLogic in IBM WebSphere. Slednji ponujajo tudi vsebnike drugih Javanskih komponent kot so: EJB (Enterprise Java Bean), JSF (Java Server Faces), JMS (Java Message Service) idr.

### 2.1.2 HTML, CSS in JavaScript

Jezik HTML je označevalni jezik, namenjen strukturiranju in opisovanju vsebine, ki se pošlje s spletnega strežnika. V HTML vtkemo zgradbo strani, ki jo brskalnik prikaže. Z oznakami tako opišemo in strukturiramo vsebino ter povežemo vse dodatne datoteke, potrebne za prikaz. Primer izvorne kode 2.1 predstavlja tak dokument.

```
<!DOCTYPE html>
<html>
  <body>
    <h1>To je naslov</h1>
  </body>
</html>
```

Izvorna koda 2.1: Primer enostavnejšega dokumenta HTML

Jezik CSS (Cascading Style Sheets) je opisni jezik in sklop pravil, ki bogatijo videz dokumenta HTML s slogi. Tolmač CSS s selektorji najde pravo vozlišče v dokumentu HTML in na njem izvede spremembe sloga. Spremembe lahko vplivajo na barvo, velikost, senčenje, zaobljenost robov in druge lastnosti. Spletni brskalnik dokumente HTML predstavi z modelom DOM (Document-Object Model). V odrezku izvorne kode 2.2 je uporabljen selektor `article#pageBody`, ki zahteva, da se iz DOM-a izbere oznaka *article* z atributom *id*, nastavljenim na *pageBody*, nad katero se izvedejo slogovne spremembe.

```
article#pageBody {
  background: gray;
  left: 0;
  right: 0;
  margin-left: auto;
  margin-right: auto;
  height: auto;
  min-height: 400px;
  padding-top: 20px;
}
```

Izvorna koda 2.2: Enostavnejši odrezek kode CSS

JavaScript je skriptni jezik, namenjen naknadnim spremembam dokumenta HTML ali proženju akcij. JavaScript se danes uporablja skoraj na vsaki spletni strani. Z njim se implementirajo animacije, 3D-igre v brskalniku, akcije na gumbih, prenos podatkov, telefonija in druge rešitve. Samo z JavaScriptom je mogoče oživeti spletno stran, ko je bila že naložena. Brskalnik potrebuje logiko, ki proži določene akcije na naloženi strani, kar je možno doseči s časovniki ali uporabnikovo interakcijo.

### 2.1.3 Spletni brskalniki

Spletni brskalnik je aplikacija, ki komunicira s spletnim strežnikom na osnovi zahtevkov in odgovorov. Obravnava odgovorov v večini primerov predstavlja rasterizacijo vsebine, ki jo vrne spletni strežnik. Poslana vsebina je najpogosteje v obliki dokumenta HTML. Brskalnik dokument prebere in ga naloži v DOM, nad katerim izvede še spremembe, ki jih zahtevata jezika CSS in JavaScript. Po prikazu slike se sproži zanka, ki aktivira logiko JavaScripta, če je na strani tako definirano.

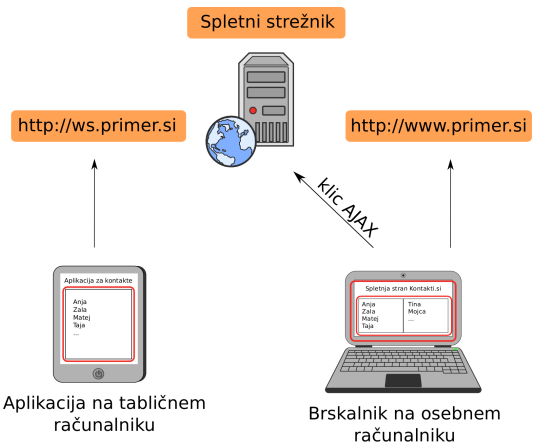
Glavni predstavniki sveta brskalnikov so: Mozilla Firefox, Google Chrome, Apple Safari, Opera in Internet Explorer.

### 2.1.4 Spletne storitve

Pred leti se je začela uporabljati besedna zveza konvergenca medijev, s katero se označuje težnja združevanja storitev tako, da so podatki dostopni preko

več različnih medijev. Podatki naj bi bilo dostopni preko televizije, telefonije, spleta ali mobilnih tehnologij; omogočeni naj bi bili povsod, kjer uporabnik želi dostopati do istih podatkov, gledati slike, poslušati glasbo ... Mobilne aplikacije in splet ponujajo dostop do podatkov in njihovo upravljanje na način, ki je najbolj primeren za določen medij. Sedaj lahko vzpostavljamo videokonference med spletnim brskalnikom, pametnim telefonom, tabličnim računalnikom in celo televizijo. Vsaka od teh naprav naj bi ponujala iste podatke, zato je potreben tudi enoten način za dostopanje do njih.

Za dostopanje do podatkov uporabljamo spletne storitve (ang. Web Services). Poznamo storitve tipa SOAP (Simple Object Access Protocol) in storitve tipa REST. SOAP je primernejši za komunikacijo med strežniškimi aplikacijami, ker zahteva večjo striktnost. Storitve REST so postale popularne zaradi svoje enostavnosti, ker niso nič drugega kot spletne strani, ki namesto dokumentov HTML vračajo XML, JSON (JavaScript Object Notation) ali kateri drug format. JSON je notacija objektov v JavaScriptu, kar poenostavi razvoj skript.




Slika 2.1: Razlaga smisla spletnih storitev

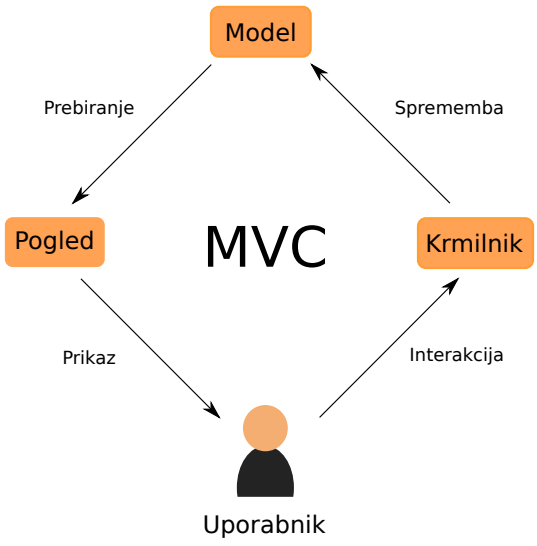
Slika 2.1 prikazuje enega od tipov konvergence podatkov na spletu. Aplikacija na tabličnem računalniku dostopa do podatkov na spletnih storitvah REST. Do teh dostopa tudi JavaScript neposredno iz brskalnika z uporabo tehnike AJAX. Na primer, če na tabličnem računalniku preko vmesnika do-

damo nov kontakt, se bo ta samodejno pojavil tudi v brskalniku, saj bo skripta JavaScript ciklično osveževala podatke preko spletnih storitev.

## 2.2 Koncept model-pogled-krmilnik

Koncept model-pogled-krmilnik (ang. Model View Controller - MVC) je koncept, ko aplikacijo razdelimo na tri dele. Na podatkovni del (ang. Model - M), ki hrani podatke, pogled (ang. View - V), ki skrbi za prikaz vsebine, in na krmilnik (ang. Controller - C). Krmilnik glede na vhodne podatke določa, kateri pogled se bo uporabil.

Koncept MVC je nastal že konec sedemdesetih let [3]. Model hrani podatke, do katerih dostopata krmilnik in pogled (slika 2.2). Obstaja več načinov implementacije koncepta MVC, in sicer glede na končni medij ali strukturo. Poglavitna pri tem konceptu je ločitev logike od pogleda in podatkov, ker tako dosežemo modularnost aplikacije.




Slika 2.2: Prikaz koncepta MVC

MVC je danes prisoten v večini aplikacij, ki imajo interakcijo s svetom. Interakcija je lahko s človekom ali z drugimi aplikacijami in ni omejena na

format. Ta koncept se uporablja pri izvedbi spletnih aplikacij, izvedbi namiznih grafičnih in tekstovnih aplikacij, izvedbi spletnih storitev ter drugih aplikacij.

Zanimive aplikacije koncepta MVC so:

- grafični uporabniški vmesniki za namizne in mobilne aplikacije,
- storitve z glasovnim XML-om (ang. Voice XML - VXML), s katerimi ustvarjamo dinamične glasovne menije na telefonskih centralah, in
- terminalske menijske aplikacije.

## 2.3 Spletni razvoj na strani odjemalca

Spletni razvoj na strani odjemalca (ang. Client-Side Web Developement) predstavlja razvoj dela spletne aplikacije, ki se tolmači v uporabnikovem brskalniku. V to skupino spada tudi razvoj logike v interpretiranih jezikih, kot je JavaScript, ki omogoča animiranje, operacije in osveževanje podatkov. Tudi razvoj aplikacij z uporabo drugih tehnologij, ki se izvajajo v brskalniku in niso privzete, je del tega področja. Taki sta na primer tehnologiji Flash in Javanski appleti. Najpogosteje temu delu pravimo tudi ospredje (ang. Front-End).

## 2.4 Spletni razvoj na strani strežnika

Spletni razvoj na strani strežnika (ang. Server-Side Web Developement) predstavlja razvoj spletne aplikacije s strežniškimi tehnologijami, kot so Javanski servleti, skripte PHP ali kakršnakoli logika, predstavljena s programskim ali skriptnim jezikom, ki se izvaja na strežniku. Logika tolmači zahtevke, ki jih pošlje brskalnik, in se nanje odzove z vračilom zahtevanih podatkov v obliki dokumenta HTML ali drugega formata. Strežniška programska logika skrbi tudi za shranjevanje in branje podatkov iz podatkovne baze ali datoteke. Ta del imenujemo tudi ozadje (ang. Back-End).

## 2.5 Platforma Java

Odkar obstaja, se je splet večkrat spremenil in se pojavil na marsikateri napravi. Za večino je postal nekaj vsakdanjega. Pred vzponom interneta smo za iskanje podatkov o podjetjih uporabljali papirnate telefonske imenike. Danes podatke iščemo drugje. Ko potrebujemo informacije, vpišemo iskalne nize v spletni brskalnik. Samoumevno se nam zdi, da imajo podjetja ali posamezniki svojo domačo spletno stran. Pogosto imajo podjetja tudi spletne aplikacije, ki uporabnikom omogočajo iskanje po katalogih, nakupovanje ali upravljanje z denarjem in naložbami. Splet ni le medij za objave in iskanje informacij, temveč je ekosistem programja, ki uporabnikom omogoča interaktivno delo, povezovanje in komunikacijo.

Skozi čas so se na tržišču pojavile različne platforme, ki so poskušale pokriti določene spletne niše. Ena od omenjenih platform je platforma Java, ki je predmet te diplomske naloge.

Platforma Java je bila definirana kot programski jezik Java, orodja Java, navidezni stroj Java in razredne knjižnice. Kombinacije naštetega nastopajo kot različni produkti: Java Standard Edition (JSE), Java Enterprise Edition (JEE), Java Micro Edition (JME) in Java Embedded. Spletne tehnologije pri Javi ne spadajo v standardno izdajo, temveč so del produkta JEE. Te tehnologije predstavljajo le osnovni sloj za razvoj aplikacij. Ker je na ta način razvoj aplikacij potekal počasi, so se v skupnosti pojavila ogrodja, ki programerjem poenostavijo delo.

V diplomski nalogi smo primerjali tri spletna ogrodja na platformi Java: Spring MVC, Stripes in Apache Tapestry. Področja primerjave so koncept ogrodja, tehnologije pogleda (ang. View), povezovanje in preverjanje podatkov, formatiranje vhodnih in izhodnih podatkov, lokalizacija in večjezičnost, hitrost in enostavnost razvoja, stanje dokumentacije in nazadnje še pogled v podporo dinamičnemu ospredju.

## 2.6 Spletne tehnologije Java

Platforma Java je znana po tem, da močno podpira abstrakcijo. Tudi spletne tehnologije, ki so del standardnega vmesnika JEE, tako imenovani programski vmesnik *servlet* (ang. *Servlet* Application Programming Iterface), abstrahirajo osnovno obliko komunikacije HTTP (Hypertext Transfer Protocol). Parametri, ki se pošljejo v zahtevku, so tako že razčlenjeni v podatkovne strukture znotraj podatkovnega tipa `HttpServletRequest`.

Metoda zahtevka HTTP se odraža s klicem metode na *servletu*. Tako metoda `GET` sproži klic implementirane metode `doGet()`, enako se zgodi za `POST`, ki proži `doPost()`. Seveda `GET` in `POST` nista edini metodi zahtevka HTTP.

Na nivoju *servleta* lahko tako implementiramo vso potrebno logiko od prebiranja podatkov iz podatkovne baze do izpisovanja vsebine telesa odgovora na zahtevek. Na tem mestu je treba omeniti tehnologijo JSP (Java Server Pages), ki je visokonivojska abstrakcija *servleta* in uporablja posebno vrsto označevalnega jezika z možnostjo direktne uporabe programskega jezika Java za sestavljanje logike. Vrinjene dele Java imenujemo *skriptleti* (ang. Scriplttlet). Datoteka JSP se med izvajanjem prevede v *servlet*, ki se nato obnaša enako kot navaden *servlet*. Ker so *skriptleti* delčki kode, postane JSP neberljiv, logika pa težko razumljiva. Zato se je JSP premaknil v območje, ki je namenjeno prikazu, ne pa implementaciji logike. Vsakršna implementacija kompleksnejše logike znotraj JSP-datoteke velja za slabo izbiro.

### 2.6.1 Seznam spletnih tehnologij na platformi Java

Skozi čas so se na platformi Java pojavile različne tehnologije, ki so z naprednimi koncepti razvijalcem olajšale delo. Ker v tehnologiji vlada tekma s časom in konkurenčnimi produkti, je bilo veliko zunanjih vzrokov za razvoj platforme. V tabeli 2.1 so navedene tehnologije, ki se uporabljajo za razvoj spletnih in drugih aplikacij. Standardizirane tehnologije nosijo oznako JSR (Java Specification Request), medtem ko je ostale ne. Tabela vsebuje

tudi tehnologije, ki niso vezane na platformo Java, to so na primer knjižnice JavaScript.

Tabela 2.1: Seznam spletnih tehnologij na platformi Java

| **Tehnologija** | **JSR** | **Kratek opis** |
|---|---|---|
| Servlet | JSR-340 | Standard, ki definira vmesnik za implementacijo logike na strani strežnika [4]. |
| JSP | JSR-245 | Standard, ki abstrahira *servlet* in omogoča implementacijo dinamičnih spletnih strani. Osnovan je na tehnologijah XML [5]. |
| EL | JSR-341 | Standard, ki definira večnamenski izrazni jezik, s katerim se dostopa do podatkov in jih spreminja. |
| JSTL | JSR-245 | Standardna knjižnica oznak za JSP [5]. |
| JSF | JSR-344 | Ogrodje za grajenje aplikacij s komponentami, ki oponaša delovanje grafičnega vmesnika Swing [5]. Definicije se preslikajo tudi na stran odjemalca, kjer torej ni potrebne dodatne logike. |
| Portlet | JSR-286 | Komponenta, analogna *servletu*, ki deluje kot modul v portalskem strežniku. |
| EJB | JSR-345 | Specifikacija, ki definira Javanska zrna, ki opravljajo različne naloge ter omogočajo porazdeljenost in samodejni preklop v primeru izpada [4]. |
| JMS | JSR-343 | Specifikacija sporočilne storitve, ki omogoča uporabo/implementacijo sporočilnih posrednikov, vrst in tem [4]. |

| | | |
|---|---|---|
| JPA | JSR-338 | Specifikacija vmesnika za shranjevanje podatkov v relacijske podatkovne baze [4]. |
| JavaMail | JSR-919 | Vmesnik za pošiljanje in sprejemanje elektronske pošte [4]. |
| Spring MVC | - | Odprtokodno ogrodje za gradnjo spletnih aplikacij, ki omogoča visoko stopnjo modularnosti. |
| Apache Struts | - | Prvo spletno ogrodje, ki je nastalo v svetu tehnologij Java. |
| Stripes | - | Ogrodje, ki poenostavlja koncept ogrodja Struts. |
| Apache Tapestry | - | Sodobno komponentno spletno ogrodje. |
| Apache Wicket | - | Spletno ogrodje, ki preslika celoten postopek razvoja v Javo. Definicije, napisane v Javi, se preslikajo tudi na stran odjemalca, kjer torej ni potrebne dodatne logike. |
| Google Web Toolkit (GWT) | - | Spletno ogrodje za izdelavo bogatih spletnih aplikacij z velikim poudarkom na tehnologiji AJAX. |
| Play framework | - | Moderno ogrodje, ki se odmika od Javanskih standardov in oponaša popularno ogrodje Ruby on Rails. |
| Vaadin | - | Spletno ogrodje za hiter razvoj spletnih aplikacij. Definicije, napisane v Javi, se preslikajo tudi na stran odjemalca, kjer ni potrebne dodatne logike. |
| Apache Click | - | Spletno ogrodje, ki je osnovano na konceptu spletnih strani. |

| | | |
|---|---|---|
| Freemarker | - | Pogon predlog, ki omogoča rabo knjižnic oznak JSP. |
| Apache Velocity | - | Popularen pogon predlog, ki omogoča preprosto integracijo z velikim številom ogrodij in je nadomestek za JSP. |
| Apache Tiles | - | Ogrodje za gradnjo predlog s tehnologijo JSP, ki omogoča modularnost strani JSP. |
| Jangod | - | Pogon predlog, ki oponaša predloge v popularnem ogrodju Django. Predloge Jangod omogočajo dedovanje. |
| HTML5 | - | Najnovejša različica specifikacije hipertekstovnega označevalnega jezika. |
| jQuery | - | Popularna knjižnica JavaScript, ki poenostavlja razvoj spletnih aplikacij na strani odjemalca. |
| Dojo | - | Popularna knjižnica JavaScript, ki omogoča implementacijo bogatih spletnih aplikacij na strani odjemalca. |
| AngularJS | - | Popularna knjižnica JavaScript, ki poenostavlja in standardizira razvoj spletnih aplikacij na strani odjemalca. |
| Bootstrap | - | Popularna knjižnica komponent HTML, CSS in JavaScript za hitro gradnjo spletnih aplikacij na strani odjemalca. |

S pomočjo *servleta* lahko implementiramo krmilnik, del JSP pa predstavlja pogled [7, str. 123], ker je namen tega lažja izvedba prikaza. Podatkovni del (model) se najpogosteje pojavlja v obliki podatkovnih struktur ali razredov, ki opisujejo podatke. Komunikacija med *servletom* in stranjo JSP poteka preko posebne strukture atributov, ki je dosegljiva na trenutnem objektu

`HttpServletRequest`. Podatki iz podatkovne baze se nastavijo kot atribut z določenim ključem. Objekt `HttpServletRequest` se nato z uporabo posebne metode posreduje strani JSP. Izvorna koda 2.3 prikazuje način posredovanja zahtevka na stran JSP.

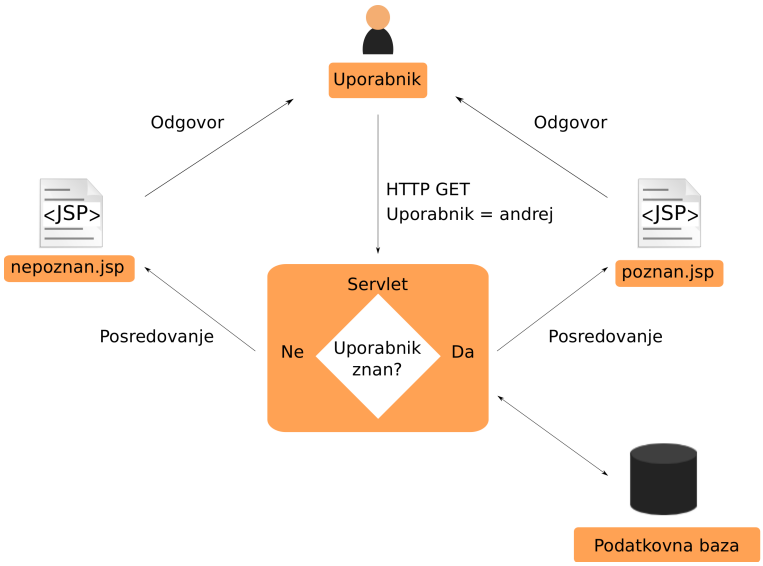
```
public void doGet(HttpServletRequest req, HttpServletResponse resp) {
  String uporabnik = request.getParameter("uporabnik");
  if(uporabnik == null) {
    request.getRequestDispatcher("nepoznan.jsp").forward(request, response);
  } else {
    Map<String, String> podatki = new HashMap<String, String>();
    /*
      Pridobivanje podatkov iz podatkovne baze in nastavljanje v mapo "podatki":
      podatki.put("ime", "Andrej"); ...
    */
    requset.setAttribute("podatki", podatki);
    request.getRequestDispatcher("nepoznan.jsp").forward(request, response);
  }
}
```

Izvorna koda 2.3: Uporaba koncepta MVC s kombinacijo *servletov* in strani JSP

Primer aplikacije, ki implementira koncept MVC s *servleti*, je prikazan na sliki 2.3. Če uporabnik v zahtevku vključi parameter `uporabnik=andrej` in *servlet* ta parameter prepozna, iz podatkovne baze prebere njegove podatke, jih nastavi na `HttpServletRequest` objektu kot atribute in zahtevek preda strani JSP poznan.jsp. Tu se mu prikažejo vsi zbrani podatki iz podatkovne baze, ki so bili nastavljeni kot atribut. Če parameter uporabnik ni nastavljen, je zahtevek posredovan strani JSP nepoznan.jsp, ki uporabniku ponudi, na primer, kratek opis strani.

Za lažje delo na strani pogleda in delo brez *skriptletov* (zaradi njihove neberljivosti) je bila dodana možnost uporabe razširitvenih knjižnic oznak (ang. Tag Libraries). Standardna knjižnica oznak JSTL (Java Standard Tag Library) je prisotna že kot del vsakega Javanskega strežnika. Knjižnice oznak lahko programer tudi sam ustvarja in s tem po lastni želji dodaja pomožne oznake, ki lajšajo delo pri ustvarjanju pogleda.

Celotno preverjanje podatkov ter skrb za pravilnost in varnost sta delo




Slika 2.3: Primer implementacije koncepta MVC s *servleti*

programerjev. Ker so programerji morali sami poskrbeti za veljavnost in pretvorbo podatkov, so začeli ustvarjati lastne knjižnice orodij, ki so skozi čas prerasle v ogrodja. V nadaljevanju si bomo ogledali tri zelo razširjena ogrodja: Spring MVC, Stripes in Apache Tapestry. Vsa tri so zgrajena na specifikaciji *servletov*. Obstajajo tudi ogrodja, ki so ubrala čisto svojo pot ter se ognila tej specifikaciji in strežnikom, ki jo podpirajo. Med te spadata ogrodje Play, ki posnema delovanje ogrodja Ruby on Rails, in ogrodje Groovy on Grails, ki temelji na jeziku Groovy, v ozadju pa uporablja JVM (Java Virtual Machine) za izvajanje.

## 2.7 Inversion of Control (IoC)

IoC je skupek več vzorcev in konceptov, ki skrajšajo čas izvedbe aplikacije, zmanjšajo količino kode, omogočijo višjo stopnjo posplošitve in ponovne uporabe obstoječih razredov. Spodaj so našteti nekateri vzorci [9]:

1. **Vzorec tovarne** (ang. Factory Pattern) - objekt, ki ustvarja druge objekte. Vzorec ločuje ustvarjanje instanc razredov od samih razre-

dov. Možno je zamenjati implementacijo tovarne brez potrebe po spreminjanju ostale programske kode (polimorfizem ustvarjanja objektov). Izvorna koda 2.4 prikazuje izvedbo vzorca tovarne.

```
/* Tovarna osnovnih objektov */
public class SomeObjectFactory {
  public SomeObject makeObject() {
    return new SomeObject();
  }
}
/* Tovarna specificnih objektov */
public class AnotherObjectFactory extends SomeObjectFactory {
  @Override
  public SomeObject makeObject() {
    return new AnotherObject();
  }
}
  /* Primer uporabe tovarn */
  SomeObjectFactory theFactory;
  theFactory = new SomeObjectFactory();
  SomeObject o1 = theFactory.makeObject();
  theFactory = new AnotherObjectFactory();
  o1 = theFactory.makeObject();
```

Izvorna koda 2.4: Primer izvedbe tovarne objektov

2. **Vzorec iskalnika storitev** (ang. Service Locator Pattern) - objekt tipa singleton, ki skrbi za iskanje in povezovanje storitev. Predstavlja enotno dostopno točko do storitev ali objektov z edinstvenim identifikatorjem. V izvorni kodi 2.5 je viden način implementacije tega vzorca.

```
public class ServiceLocator {
  /* Kontekst, iz katerega se pridobiva reference do storitev */
  private Context ctx = null;
  /* Singleton */
  private static ServiceLocator sl = null;
  protected ServiceLocator() {
    this.ctx = new Context();
  }
  public static ServiceLocator getInstance() {
    if(sl == null)
      sl = new ServiceLocator();
    return sl;
  }
  /* Metoda za iskanje storitev */
```

```
  public Service getService(String id) {
    return this.ctx.lookup(id);
  }
}
```

Izvorna koda 2.5: Splošen prikaz iskalca storitev

3. **Injiciranje odvisnosti** (ang. Dependency Injection) - vzorec, ki poenostavlja razvoj aplikacij. Reference objektov storitev v kontekstu se injicirajo v metode ali objekte na naslednje načine:

   - **Injiciranje konstruktorju** - vzorec, ki omogoča pridobivanje instanc objektov iz konteksta kar preko konstruktorja razreda. V izvorni kodi 2.6 je za injiciranje uporabljena anotacija `@Inject`.

```
public SomeObject(@Inject SomeService srv) {
  // ...
}
```

Izvorna koda 2.6: Izgled injiciranja parametrov konstruktorju

   - **Injiciranje parametrov** - vzorec, ki omogoča injiciranje referenc kar v atribute objekta. Izvorna koda 2.7 za ta namen uporablja anotacijo `@Inject`. Praviloma je treba atributu implementirati *setter* metodo.

```
public class SomeObject {
  @Inject
  private SomeService srv;

  public void setSrv(SomeService srv) {
    this.srv = srv;
  }
}
```

Izvorna koda 2.7: Injiciranje referenc v atribute objekta

   - **Injiciranje s setterjem** - vzorec, ki deluje podobno kakor injiciranje parametrov. Anotacija `@Inject` se tokrat uporabi na parametru *setter* metode. Izvorna koda 2.8 prikazuje uporabo tega vzorca.

```
public class SomeObject {
  private SomeService srv;

  public void setSrv(@Inject SomeService serv) {
    this.srv = serv;
  }
}
```

Izvorna koda 2.8: Injiciranje referenc preko *setterja*

- **Injiciranje implementacij vmesnikov** - v kontekstu se poišče instanca implementacije nekega vmesnika, ki se nato injicira v zahtevan atribut. Izvorna koda 2.9 prikazuje konceptualno uporabo tega vzorca. Pogosto je ta način injiciranja kombiniran z imenom spremenljivke.

```
public class SomeObject {
  /* Injiciranje preko vmesnika */
  @Inject
  private SomeServiceInterface srv;
}

/* Implementacija vmesnika */
public class SomeServiceInterfaceImpl implements SomeServiceInterface {}
```

Izvorna koda 2.9: Injiciranje referenc implementacij vmesnikov

4. **Iskanje po kontekstu** (ang. Contextualized Lookup) - s pomočjo konteksta lahko poiščemo objekte, ki ustrezajo določenim kriterijem. Kriterij je lahko identifikator, implementiran vmesnik ali ime razreda. Izvorna koda 2.10 prikazuje primer iskanja po kontekstu.

```
@Inject
private Context ctx;
// ...

public Object someMetod() {
  return this.ctx.lookup(SomeObjectInterface.class);
}
```

Izvorna koda 2.10: Iskanje po kontekstu

5. **Nastavčne metode** (ang. Template Method Design Pattern) - na-

stavčne metode delujejo kakor nastavki, ki jih je programer dolžan izpolniti. Izvorna koda 2.11 prikazuje primer, kjer mora programer poskrbeti za vzpostavitev začetnega stanja z implementacijo metod, kakor je `initSystem()`. Te metode so na primer ob zagonu aplikacije poklicane v določenem vrstnem redu.

```
public final void init() {
  initSystem();
  initDBsubsystem();
  goToReadyMode();
}
/* Nastavki, ki se izvedejo */
protected abstract void initSystem();
protected abstract void initDBsubsystem();
protected abstract void goToReadyMode();
```

Izvorna koda 2.11: Uporaba nastavčnih metod

6. **Vzorec strategij** (ang. Strategy Design Pattern) - strategije omogočajo izbiro logike, ki se bo izvedla. Tako lahko, kakor v primeru izvorne kode 2.12, omogočimo izbiro med tipi storitev.

```
/* Vmesnik strategije */
public interface Cleaning {
  public void clean();
}
/* Implementaciji strategije */
public class DryCleaning implements Cleaning { /* ... */ }
public class WashCleaning implements Cleaning { /* ... */ }
/* Razred, ki upravlja strategije */
public class ApplyCleaningStrategy {
  public static void apply(Cleaning strategy) {
    // ...
    strategy.clean();
    // ...
  }
}
/* Primer uporabe */
public class Main {
  publica static void main(String args) {
    if("dry".equals(args[1]) {
      ApplyCleaningStrategy.apply(new DryCleaning());
    } else {
      ApplyCleaningStrategy.apply(new WashCleaning());
```

```
    }
  }
}
```

Izvorna koda 2.12: Vzorec strategij - primer čistilnice

# Poglavje 3

# Ogrodje Spring MVC

## 3.1 Osnovni podatki

| | |
|---|---|
| Ogrodje: | Spring MVC |
| Spletna stran: | *http://projects.spring.io/spring-framework/* |
| Trenutna različica: | 4.0.4 |
| Leto prve različice: | 2004 |
| Organizacija: | Pivotal Software, Inc. |

## 3.2 Kratek opis

Ogrodje Spring (ang. Spring Framework) je eno od ogrodij za razvoj aplikacij JEE. Spring MVC je samo en od mnogih modulov celotnega ogrodja. Osnovni - jedrni del ogrodja vsebuje module, ki definirajo kontekst, zrna Spring, funkcionalnosti AOP (Aspect Oriented Programming), ORM (Object Relational Mapping), Spring MVC in druge funkcionalnosti. Osnovno ogrodje razširjajo še drugi moduli. Najpomembnejši med njimi so:

- **Spring framework** - osnovni moduli ogrodja (Context, Beans, MVC itd.),

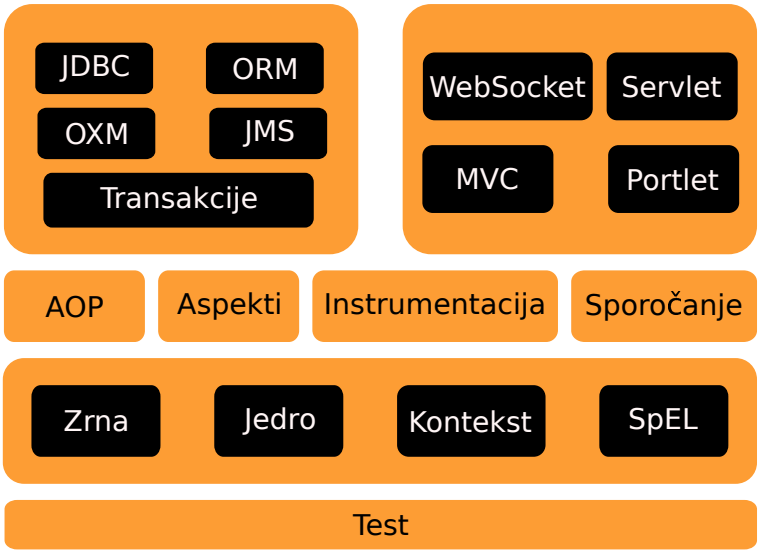
- **Spring Data** - moduli za komunikacijo s podatkovnimi bazami ter ORM (SQL (Structured Query Language), no-SQL, indekserji itd.),
- **Spring Integration** - moduli za lažjo integracijo med aplikacijami,
- **Spring Security** - modul za enostavno implementacijo varnostnih mehanizmov in preverjanja,
- **Spring Social** - modul za enostavno integracijo s socialnimi omrežji,
- **Spring Web Flow** - modul za razvoj čarovnikov,
- **Spring Web services** - modul za enostavno gradnjo in integracijo spletnih storitev in
- **Spring LDAP** - modul za komunikacijo in upravljanjem z direktorijem LDAP.

Moduli se iz različice v različico izboljšujejo, razširjajo in dodajajo. Podjetje Pivotal (hčerinska družba podjetja VMWare) teži k temu, da se proizvodi znotraj lastne palete produktov povezujejo med seboj. Ogrodje Spring slovi po visoki stopnji modularnosti, nastavljivosti in razširljivosti. Vse to je možno, ker arhitekti dosledno sledijo paradigmi IoC (poglavje 2.7). Z IoC in kontekstnim XML-om Spring je mogoče sestaviti velik del aplikacije, ne da bi napisali eno samo vrstico programske kode. Večina zrn je namreč takšnih, da nam zadostuje parametriziranje [11].

Z različico 2.5 je bilo omogočeno, da se aplikacija sestavi z uporabo anotacij. Z njimi lahko nastavimo enake stvari kakor z XML-om. Za določen tip aplikacij je kontekstni XML še vedno boljša izbira, ker omogoča pogled na sestavo aplikacije kot na lastni dlani.

## 3.3 Funkcionalnost

Ogrodje Spring zajema široko paleto orodij, ki ne spadajo v temo tega diplomskega dela. Delo je omejeno na spletne tehnologije, ki so del modula




Slika 3.1: Arhitektura ogrodja Spring [10]

Spring MVC, in module, od katerih je odvisen. Izključen je tudi opis dela ogrodja, ki je namenjen interakciji s podatkovno bazo, ker ta ni neposredno povezan z modulom MVC.

Spring MVC ponuja klasičen programski vmesnik MVC in spada v skupino akcijsko usmerjenih ogrodij, kjer so akcije (npr. miškin klik na gumb) v ospredju. Vsako akcijo moramo povezati z upravljavcem (ang. Handler). Upravljavci so v tem primeru metode na krmilniku, katerih ime ustreza akciji [11]. Možna je tudi uporaba anotacij, ki so del jezika Java [11]. Z njimi definiramo ime akcije, ki jo metoda upravlja. Metode imajo več oblik, ki jih ogrodje razume. Najpogosteje se s posebnimi anotacijami označi povezovanje podatkov iz obrazca na pogledu s podatkovnim modelom (izvorna koda 3.2).

Opisano povezovanje omogoča tudi več načinov preverjanja. V primeru izvorne kode 3.2 je uporabljeno preverjanje po specifikaciji Java JSR-303, kjer v modelu z anotacijami označimo pravila za preverjanje. Omogočena je tudi možnost uporabe naknadnega preverjanja podatkov za bolj vsebinske namene, na primer preverjanje gesla in uporabniškega imena. Poleg dogovora JSR-303 ogrodje Spring omogoča še osnovno preverjanje z uporabo posebnih

razredov (ang. Validators).

Za večjezičnost je poskrbljeno s samodejnim zaznavanjem jezika v brskalniku. Logiko za določanje jezika je možno nadomestiti z lastno logiko. Glede na izbrani jezik se uporabljajo tudi primerni svežnji prevodov oznak, napak in izrazov.

Kot tehnologija pogleda je v priloženi aplikaciji uporabljen standardni JSP/JSTL. Spring namreč omogoča uporabo tudi drugih tehnologij pogleda [12], kot sta na primer Apache Velocity [13] ali Freemarker [14].

Ogrodje Spring je v osnovi zasnovano z IoC, kar pomeni, da lahko interne storitve za dostop do podatkov ali drugih pripomočkov z anotacijami povežemo v krmilnike ali druge storitve [11].

## 3.4 Koncept

Ker je ogrodje Spring zgrajeno okoli standardnega spletnega vmesnika Java in *servletov*, ima Spring pripravljen *servlet*, ki ob zagonu vsebnika *servletov* pripravi celotno aplikacijo. Prebere se kontekstni XML Spring, ki je na *servletu* nastavljen kot zagonski parameter. Nastavitve in definicije se aplicirajo na zagnano okolje, ustvarijo se zrna Spring [11].

Ko je postopek zagona aplikacije uspešno zaključen, je spletna aplikacija na voljo za obdelovanje zahtevkov HTTP. Zahtevki najprej prispejo na *servlet* za razpošiljanje, ki jih usmeri na pravi krmilnik. Vsak krmilnik je možno povezati na določeno kontekstno pot z anotacijo `@RequestMapping` (izvorna koda 3.1).

```
@Controller
@RequestMapping("/webcontest")
public class AppController {
  // ...
}
```

Izvorna koda 3.1: Vezava krmilnika na kontekstno pot

Vsak zahtevek mora imeti svojo upravljalno metodo [16], ki je povezana na določeno kontekstno pot. Pot se enako kot na krmilniku nastavi z anota-

cijo `@RequestMapping` (izvorna koda 3.2). Če kontekstna pot na upravljalni metodi ni nastavljena, postane metoda splošna in zajame vse zahtevke, ki nimajo definiranega svojega upravljavca. Z uporabo parametra `method` lahko še dodatno razdelimo promet glede na metodo zahtevka HTTP. Najpogosteje sta to GET in POST. Prvi se najpogosteje uporablja pri prehajanju med stranmi, drugi pa pri pošiljanju podatkov iz obrazca. Podatki iz obrazca so tisti, ki jih navadno želimo povezati z modelom, preveriti in shraniti. Tako lahko na isto kontekstno pot postavimo prikaz obrazca in pošiljanje podatkov. To se v brskalniku odraža z lepimi in berljivimi naslovi URL.

```
@RequestMapping(value = "/login", method = RequestMethod.POST)
public Object login(@ModelAttribute @Validated UserLogin userLogin, BindingResult
    result) {
  // ...
}
```

Izvorna koda 3.2: Upravljalna metode z anotacijami za povezovanje podatkov

Podatki, poslani na strežnik, gredo do upravljalne metode, ki poskrbi za povezovanje in preverjanje. Za tem se izvede programska logika v upravljalni metodi, ki uporablja storitve za opravilo naloge. Upravljalna metoda lahko odgovori z napako HTTP ali preusmeritvijo zahtevka in modela na pravi pogled (JSP), ki se uporabniku prikaže.

## 3.5 Predloge

Ogrodje Spring samo po sebi ne definira določenega pogona predlog (ang. Template Engine), omogoča pa uporabo svoje knjižnice oznak JSP, ki poenostavi delo programerjem. V kombinaciji z orodjem Spring MVC se pogosto uporablja tudi pogon predlog Apache Velocity, ki ponuja svoj format, jezik in koncept pisanja predlog [16].

V primeru razvite aplikacije se uporablja pogon predlog JSP, ki je najbolj splošen. Spring ponuja lastno knjižnico oznak (ang. Tag Library), ki se dopolnjuje s knjižnico JSTL. Izvorna koda 3.3 prikazuje branje teksta iz svežnja prevodov za trenutni jezik.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<!-- ... -->

<a href="logout"><spring:message code="header.logout" text="Logout"/></a>

<!-- ... -->
```

Izvorna koda 3.3: Uporaba Springove oznake za pridobivanje sporočil iz svežnja prevodov

## 3.6 Povezovanje in preverjanje podatkov

V implementirani aplikaciji smo uporabili način za povezovanje podatkov z anotacijo `@ModelAttribute`, ki v želeni tip poveže istoimenske podatke iz obrazca. Hkrati je v isti vrstici uporabljena še anotacija `@Validate` [11], ki je del specifikacije JSR-303. Za pravilno povezovanje sta pri ogrodju Spring MVC potrebna dva koraka. Prvi je ta, da se ustvari Javanska zrna s spremenljivkami ter njihove setterje in getterje, ki ustrezajo imenu posameznih polj v obrazcu. Drugi je ustvarjanje metode, ki naredi instanco pravilnega tipa atributa (izvorna koda 3.4).

```
@ModelAttribute("userLogin")
public UserLogin userLogin() {
  UserLogin ul = new UserLogin();
  return ul;
}
```

Izvorna koda 3.4: Metoda za pridobivanje instance člana modela

Imeni obrazca in spremenljivke z anotacijo `@ModelAttribute` na upravljalni metodi se morata ujemati z vrednostjo anotacije `@ModelAttribute` na metodi, ki ustvarja objekte [11]. Način povezovanja je nekoliko zamuden zaradi napak, do katerih pogosto pride pri posameznih korakih.

Preverjanje povezanih podatkov v aplikaciji je implementirano z uporabo novejšega koncepta po specifikaciji JSR-303, ki definira način preverjanja Javanskih zrn. V zrna, ki opisujejo obrazec, dodamo anotacije, s katerimi

definiramo omejitve (izvorna koda 3.5).

```
@Size(min=5, max=40, message="{email.length}")
@Pattern(regexp="...", message="{email.invalid}")
private String email;

@Size(min=4, max=20, message="{password.length}")
@NotBlank(message="{password.blank}")
private String password;
```

Izvorna koda 3.5: Uporaba anotacij za preverjanje zrn

JSR-303 pri Springu s seboj prinese težave, na katere naletimo že na začetku. Ogrodje Spring definira zgolj vmesnik JSR-303, implementacije pa ne nudi. Uporabiti moramo katero od obstoječih izvedb preverjanja JSR-303. Najbolj razširjena taka izvedba je del projekta Hibernate, ki je znan po svoji implementaciji ORM. Osnovna integracija obojega je enostavna. V projekt vključimo knjižnico JAR (Java Archive) in preverjanje začne delovati. Problem se pojavi pri lokaliziranih sporočilih napak in samem posredovanju napak na pogled. Voditi moramo dva seznama sporočil za vsak jezik, enega za Spring MVC, drugega za preverjanje. To zmanjša učinkovitost razvoja in vzdrževanja. Veliko dela zahtevata še pravilna izbira jezika in posredovanje informacij o napaki, ki pa v določenih primerih ne deluje. Čas, ki ga privarčujemo pri preverjanju z JSR-303, izgubimo za postavitev okolja. Zaradi vsega naštetega JSR-303 ni najprimernejša izbira. Privzeti način preverjanja je na začetku zamuden, a se obrestuje pri vzdrževanju, saj ne zahteva dodatnih integracij [11].

## 3.7 Podpora za IoC

V uvodu poglavja je omenjeno, da je ogrodje Spring oblikovano po konceptu IoC, kar prinese visoko stopnjo modularnosti, generičnosti in enostavno integracijo.

Ko se preverja uporabnikova identiteta, se navadno poišče v imeniku ali podatkovni bazi, kjer se tudi preveri veljavnost gesla. Za vse to je nujen dostop do podatkovne baze. V ta namen lahko ustvarimo zrno Spring, ki

deluje kot storitev za dostop do podatkovne baze in ga definiramo v kontekstni datoteki XML. Zrno lahko tako z injiciranjem odvisnosti povežemo v krmilnik (izvorna koda 3.6). Atribut XML `id="db"` določi, da se razred `DBManager` v kontekst zapiše pod identifikator *db*, ki ga anotacija `@Autowire` poveže s spremenljivko *db* tipa `DBManager` [11].

```
<!-- applicationContext.xml -->
<!-- ... -->
<bean id="db" class="org.zabica.webcontest.springmvc.beans.DBManager">
   <property name="storeFile" value="/tmp/store" />
</bean>
<!-- ... -->

// AppController.java
@Controller
@RequestMapping("/webcontest")
public class AppController {

  @Autowired
  private DBManager db;

  public DBManager getDb() {
    return this.db;
  }

  public void setDb(DBManager db) {
    this.db = db;
  }
}
```

Izvorna koda 3.6: Uporaba injiciranja odvisnosti IoC

V primeru uporabe polnega aplikacijskega strežnika JEE z vsebnikom EJB je možno integrirati EJB s Springovim kontekstom. Zrna EJB v tem primeru nastopajo kot zrna Spring. Tako lahko lastnosti zrn EJB uporabljamo tudi v aplikacijah z ogrodjem Spring [11].

## 3.8 Lokalizacija

Lokalizacija je v Javanskem svetu privzeta že na nivoju platforme. Na voljo imamo podsisteme, ki nudijo enostavno shranjevanje in branje prevodov. Ogrodje Spring uporablja ravno te konstrukte, in sicer v kombinaciji s preverjanjem in prikazom ter oznako jezika. Oznako jezika nam po privzetem načinu pošlje brskalnik. Privzeto pridobivanje jezika je možno zaobiti z registracijo lastne izvedbe jezikovnega razreševalca (ang. Locale Resolver) [11]. Poleg tega moramo zavesti še lokacije in bazna imena svežnjev sporočil (izvorna koda 3.7).

```
<!-- applicationContext.xml -->
<bean id="messageSource"
      class="org.springframework.context.support
.ResourceBundleMessageSource">
   <property name="basenames">
     <list>
       <value>messages/messages</value>
       <value>messages/validation_errors</value>
     </list>
   </property>
</bean>

<bean id="messageInterpolator"
      class="org.zabica.webcontest.springmvc.validators
.CustomSpringMessageSourceInterpolator">
   <property name="messageSource" ref="messageSource"></property>
</bean>
<bean id="validator"
      class="org.springframework.validation.beanvalidation
.LocalValidatorFactoryBean">
   <property name="validationMessageSource" ref="messageSource"></property>
   <property name="messageInterpolator" ref="messageInterpolator"></property>
</bean>

<bean id="localeResolver"
      class="org.zabica.webcontest.springmvc.validators
.UserSessionLocaleResolver">
</bean>
```

Izvorna koda 3.7: Definicije svežnjev sporočil v kontekstnem XML-ju

V pogledu z uporabo oznake `<spring:message code="header.logout" text="Logout"/>` pridobimo sporočilo ali tekst, ki je za ključ `header.logout` prisoten v trenutnem svežnju. Faza preverjanja za pridobivanje sporočil napak uporabi ključe sporočil, ki so prisotni v anotacijah za preverjanje, kot je `@Size(min=5,max=40,message="{email.length}")`.

## 3.9 Podpora REST

Ogrodje Spring nudi dobro podporo izvedbi spletnih storitev REST. Omogoča namreč neposredno pretvorbo objektov Java v JSON ali XML in obratno. V kontekstni datoteki XML je treba nastaviti le pravilne *serializatorje* za želene tipe MIME (Multipurpose Internet Mail Extensions) [15]. Za samodejno pretvarjanje odgovora v format JSON se na upravljalni metodi nastavi anotacija `@ResponseBody`, na objektu odgovora pa tip MIME `application/json`. Iz metode nato vrnemo instanco objekta, ki ga želimo pretvoriti. Podobno velja za vhodno smer, kjer anotacijo `@RequestBody` nastavimo na objekt, ki ga želimo sprejeti [15]. Postopek velja tudi za druge formate.

## 3.10 Knjižnice ospredja (JS, AJAX)

Ogrodje Spring ne vsebuje ali določa knjižnic, ki se uporabljajo v delu ospredja. To velja tudi za knjižnice JavaScript in klice AJAX. Knjižnice JavaScript lahko izberemo po lastni presoji, na primer jQuery ali Dojo. Klice AJAX moramo prav tako izvesti sami ali pa uporabimo tiste, ki so del izbrane knjižnice JavaScript.

## 3.11 Dokumentacija

Spring je razširjeno ogrodje, ki ga podpirajo močna skupnost in uveljavljena podjetja. Na spletni strani je možno najti JavaDoc, primere in vadnice. Poleg tega je o ogrodju Spring veliko dokumentacije, ki podrobno opisuje

celotno ogrodje. Trenutno je na voljo četrta različica ogrodja Spring in o njem obstaja le ena knjiga. Zanj so še vedno aktualne knjige za predhodne verzije. Stanje dokumentacije je torej dobro.

# Poglavje 4

# Ogrodje Stripes

## 4.1 Osnovni podatki

| | |
|---|---|
| Ogrodje: | Stripes |
| Spletna stran: | *http://www.stripesframework.org* |
| Trenutna različica: | 1.5.7 |
| Leto prve različice: | 2005 |

## 4.2 Kratek opis

Ogrodje Stripes se zgleduje po znanem ogrodju Apache Struts, ki je bilo eno od prvih ogrodij v svetu JEE. Struts ima bogato zgodovino in je še po 15 letih zelo razširjeno ogrodje. Ogrodje Stripes je nastalo kot preprostejša alternativa ogrodju Struts, ki ima zelo kompleksno strukturo. Stripes uporablja anotacije in generike, ki jih je prinesla Java 1.5. Glavna značilnost ogrodja je pravilo, da ima dogovor prednost pred nastavitvami (ang. Convention Over Configuration), kar zmanjšuje količino nastavitev v projektu. Nastavitve so v datoteki web.xml v obliki zagonskih parametrov na *servletu* za razpošiljanje [18].

Stripes velja za akcijsko usmerjeno ogrodje, vendar vsebuje že pripravljene komponente. Koncept MVC je tudi v tem primeru prisoten, a za razliko od

klasičnega koncepta ima to ogrodje združen model in krmilnik v isti razred kot ogrodje Struts.

Stripes je sedaj že zrelo ogrodje. Najnovejša različica 1.5.7 je stara dve leti, kar kaže na njegovo stabilnost.

## 4.3 Funkcionalnost

Ogrodje Stripes delno sledi konceptu MVC, ker združuje model in krmilnik v isti razred - akcijsko zrno (ang. Action Bean). Iz pogleda poteka dostop do modela preko reference akcijskega zrna. Kot tehnologija pogleda je najpogosteje uporabljen JSP zaradi knjižnice oznak, ki jo nudi Stripes. Direktna integracija je možna s tehnologijo Freemarker, ki uporablja drugačen opisni jezik. Freemarker nudi dodatne izboljšave od enostavnega formatiranja izhodnih podatkov do močnejše izvedbe izraznega jezika (ang. Expression Language). IoC sam po sebi ni podprt, je pa možno združevanje z ogrodjem Spring, ki s seboj prinese vse svoje funkcionalnosti. Stripes ima dobro podporo lokalizaciji; preverjanje vnesenih podatkov se izvede v dveh korakih, osnovno z uporabo anotacij in kompleksnejše s posebninimi metodami. Ogrodje Stripes omogoča tudi izvedbo čarovnikov z uporabo anotacije `@Wizard` [18].

## 4.4 Koncept

Kot ogrodje Spring se tudi to ogrodje zažene z uporabo *servleta* za razpošiljanje. Akcijska zrna se z anotacijo `@UrlBinding` preslikajo na kontekstno pot zahtevka, s čimer logično med seboj razdelimo skupke strani. Tudi Stripes uporablja upravljalne metode, ki dobijo ime po dogodkih. Dogodke je možno povezati tudi z anotacijami `@HandlesEvent`. Zanimivo pri ogrodju Stripes je, da ne ločuje metode zahtevka HTTP. Vsi parametri zahtevka so vedno združeni v skupni seznam ne glede na njihov izvor (URL ali obrazec) [18].

Akcije grejo skozi stopnje življenjskega cikla, ki jih definira ogrodje. Vsaka stopnja izvede določene operacije nad zahtevkom in podatki. K življenjskemu

ciklu spadajo stopnja povezovanja, stopnja potrjevanja in stopnja izvajanja dogodka. Videz opisanih funkcionalnosti je viden v primeru izvorne kode 4.1;

```
@UrlBinding("/webcontest/{$event}")
public class WebContestActionBean implements ActionBean, ValidationErrorHandler {
  @Validate(on = { "login" } )
  private String user;

  @DefaultHandler
  public Resolution index() {
    // ...
  }

  @HandlesEvent("login")
  public Resolution login() {
    // ...
  }
}
```

Izvorna koda 4.1: Primer delne implementacije akcijskega zrna

Prejeti podatki se preverijo in povežejo z modelom. V primeru napak pri preverjanju se uporabnik vrne na izvorno stran, kjer so izpisane napake.

## 4.5 Predloge

Kakor pri ogrodju Spring je tudi pri ogrodju Stripes privzeta tehnologija pogleda JSP. Stripes ponuja lastno knjižnico oznak, ki olajša delo na ospredju aplikacije. Stripesova knjižnica oznak ne podpira jezika HTML5, saj ne dovoljuje uporabe atributov, ki so bili dodani v to različico. V ta namen je bila izdana posebna različica knjižnice oznak, ki ne preverja imen lastnosti [18]. Tako lahko z njo uspešno ustvarimo tudi poglede HTML5.

Ogrodje je delno komponentno, ker knjižnica oznak vsebuje komponente, ki programerju poenostavijo delo (izvorna koda 4.2). Dober primer take komponente so izvlečni meniji, ki jih lahko ustvarimo z enumeracijo. Ogrodje je sposobno enumeracije samodejno prevesti v izbrani jezik [18].

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="stripes" uri="http://stripes.sourceforge.net/stripes-dynattr.tld"%>
```

```
<%-- ... --%>
<stripes:text class="form-control" id="email" name="email" placeholder="${loginemail}"/>
<%-- ... --%>
```

Izvorna koda 4.2: Uporaba oznake iz knjižnice Stripes

Stripesova knjižnica oznak se dopolnjuje s knjižnico JSTL, kjer so pripomočki za gradnjo strani. Poleg komponent vsebuje Stripesova knjižnica tudi oznake za združevanje komponent v različne module in razporeditev teh po spletnih straneh.

## 4.6 Povezovanje in preverjanje podatkov

Pri ogrodju Stripes se povezovanje podatkov dogaja sočasno s prvo stopnjo preverjanja, kjer se glede na tip spremenljivke, s katero se povezuje, določita način formatiranja in vrednost. Skupaj s tolmačenjem in pretvorbo se preverja, ali podatki upoštevajo omejitve. Izvorna koda 4.3 prikazuje uporabo povezovanja in preverjanja na gnezdenih podatkovnih strukturah [18].

```
@ValidateNestedProperties({
  @Validate(field="title", required=true, minlength=4 , on = { "doAddConf" } ),
  @Validate(field="description", required=true, on = { "doAddConf" } ),
  @Validate(field="start", required=true, on = { "doAddConf" } ),
  // ...
  @Validate(field="location", required=true, on = { "doAddConf" } ),
  @Validate(field="tags", required=true, converter=TagConverter.class, on = { "doAddConf"
      } ),
})
private Conference conference;
```

Izvorna koda 4.3: Gnezdeno preverjanje in povezovanje vhodnih podatkov

Stripes v osnovi ponuja obširno paleto pretvornikov podatkovnih tipov. Omogoča tudi lastne izvedbe, kar je prikazano v primeru izvorne kode 4.3. Lasten pretvornik tipa lahko z anotacijo `@Validate` nastavimo na polju. Razvoj lastnega pretvornika izvedemo z uporabo vmesnika `TypeConverter<T>`, kjer v metodi `convert()` z logiko opišemo postopek pretvorbe [18].

V primeru napake pri pretvorbi podatkov ali pri preverjanju se iz svežnja sporočil, glede na trenutno izbrani jezik, prebere opis napake, ki se vrne na

pogled za prikaz.

## 4.7 Podpora za IoC

Ogrodje Stripes je bilo osnovano kot preprosta oblika spletnega ogrodja, zato vsebuje samo rešitve, ki so namenjene spletnemu razvoju. Glede uporabe koncepta IoC velja, da je ta prisoten samo v manjši meri za injiciranje parametrov v akcijska zrna. Za izvedbo storitev in drugih pripomočkov je omogočena enostavna integracija z ogrodjem Spring, ki s svojimi moduli doda možnost uporabe internih storitev in komponent za razvoj poslovne logike [19].

## 4.8 Lokalizacija

Problem lokalizacije je pri Stripesu rešen na eleganten način. Jezik se pridobi iz brskalnika, z uporabo vmesnika `LocalePicker` pa lahko izvedemo lastno logiko za izbiro jezika. Polno ime uporabljenega razreda nastavimo v datoteki web.xml kot zagonski parameter. To je prikazano v izvorni kodi 4.4.

```
<!-- web.xml -->

<!-- ... -->

<init-param>
    <param-name>LocalePicker.Class</param-name>
    <param-value>org.zabica.webcontest.stripes.converter
.extensions.SettingsBasedLocalePicker</param-value>
</init-param>

<!-- ... -->
```

Izvorna koda 4.4: Način nastavitve pobiralca jezika

Stripes zahteva uporabo svežnjev lokaliziranih sporočil že od samega začetka razvoja. Vanje moramo vnesti vsaj sporočila za privzeti jezik, ostale jezike lahko dodajamo naknadno. Pri Stripesu je zanimivo to, da omogoča več notacij ključev sporočil in imena vnosnih polj. Tako za napako enakega

tipa napišemo samo eno sporočilo, v katerega se samodejno vključijo še ime polja in podrobnosti o napaki. Primer takega sporočila je prikazan v izvorni kodi 4.5. Če je geslo v polju *register.password* prekratko, stopnja preverjanja vrne sporočilo o napaki oblike "Geslo je prekratko. Zahtevana dolžina je 6". Vrednost šest je pridobljena iz anotacije `@Validate`.

```
################################
### StripesResources.properties
# ...
register.password=Geslo
# ...
valueTooShort={0} je prekratko. Zahtevana dol\u017Eina je {2}
# ...
################################
```

Izvorna koda 4.5: Definicije sporočila napake

Na ta način se bodo vse komponente, ki so del Stripesove knjižnice oznak, prevedle samodejno. Formatiranje izhodnih podatkov je izvedeno z uporabo oznake *stripes:format* oznake [18].

## 4.9 Podpora REST

Podpora storitvam JSON REST v ogrodju Stripes je možna z uprabo odgovora `JavaScriptResolution` in lastne izvedbe pretvornika vsebine. Podobno kot za JSON velja tudi za XML, le da XML nima predstavnika med objekti za odgovor in ga moramo izvesti sami.

## 4.10 Knjižnice ospredja (JS, AJAX)

Stripes kot Spring MVC ne nudi ali zahteva uporabe specifične knjižnice. Omogočena je uporaba katerokoli knjižnice JavaScript in njenega vmesnika AJAX.

## 4.11 Dokumentacija

Ker je ogrodje preprosto, je tudi sama dokumentacija takšna. V večini primerov zadostujejo razlage v dokumentaciji JavaDoc, v vadnicah in primerih na domači strani projekta. Odgovori na vsa vprašanja, na katera spletno gradivo ne odgovarja, so dostopni v redkih izdanih knjigah na to temo [18]. Čeprav je projekt dobro zasnovan, je opaziti, da se je skupnost v zadnjih letih skrčila.

# Poglavje 5

# Ogrodje Apache Tapestry

## 5.1 Osnovni podatki

| | |
|---|---|
| Ogrodje: | Apache Tapestry |
| Spletna stran: | *http://tapestry.apache.org/* |
| Trenutna različica: | 5.3.7 |
| Leto prve različice: | 2000 |

## 5.2 Kratek opis

Ogrodje Tapestry poenostavlja in pospešuje razvoj spletnih strani na platformi Java, in sicer z uvajanjem abstrakcije nad različnimi deli klasičnega razvoja. Privzete komponente omogočajo enostavno izvedbo asinhronih klicev, ki jih je možno parametrizirati po lastni želji. Ogrodje ponuja uporabo prikladnega koncepta razporeditve, ki pospeši razvoj [21].

## 5.3 Funkcionalnost

Tapestry se lahko pohvali z obširno paleto rešitev, ki so uporabne pri razvoju spletnih aplikacij. Ogrodje vsebuje več komponent, lastno implementacijo IoC, integracijo z ogrodjem Spring in integracijo z različnimi tehnologijami

ORM. Omogoča hitro implementacijo preverjanja vsebine na strani odjemalca (izvorna koda 5.1) z opisi kriterijev na komponenti.

## 5.4 Koncept

Tapestry ob zagonu postavi bazen instanc definiranih objektov, iz katerega se ob zahtevkih pridobi instanca. Tak način strežbe pospeši procesiranje in zmanjša vpliv na sistem. Razredi strani vsebujejo spremenljivke, ki so analogne komponentam na strani pogleda, in spremenljivke, ki vsebujejo podatke. Tapestry določa več življenjskih ciklov, ki so vezani na tip zahtevka [25]. V vsakem ciklu se izvajajo določene faze, ki so implicitno določene glede na ime obrazca in akcije. V teh upravljalnih metodah se dostopa do storitev. Faza preverjanja proži klice svojih implicitno poimenovanih metod, v katerih dodatno preverjamo vsebino poslanih podatkov. V ciklu prikaza se predloge sestavijo v prikazano stran. Napake pri preverjanju so po privzetem slogu označene z mehurčki [21].

## 5.5 Predloge

Tapestry 5 uporablja lasten HTML-ju podoben jezik za sestavljanje predlog. To je XML z dodatnimi shemami, ki definirajo komponente oziroma transformirajo oznake HTML [21]. V primeru izvorne kode 5.1 je predstavljena predloga strani.

```
<html t:type="layout" title="${message:login.title}"
       xmlns:t="http://tapestry.apache.org/schema/tapestry_5_3.xsd"
            xmlns:p="tapestry:parameter">
       <div class="col-md-6 col-md-offset-3">
              <t:form method="post" class="form-horizontal" t:id="login">
                     <t:errors />

                     <!-- ... -->
                     <div class="form-group">
                            <t:label class="col-sm-4 control-label" t:for="passwordField">
                                   $${message:login.password}
                            </t:label>
```

```
                        <div class="col-sm-8">
                            <t:passwordfield class="form-control"
                                t:id="passwordField"
                              t:value="password"
                              t:validate="required,
minlength=4"
                              placeholder="${message:login
.password}">
                            </t:passwordfield>
                        </div>
                    </div>
                    <!-- ... -->
                </t:form>
            </div>
</html
```

Izvorna koda 5.1: Primer opisnega jezika Tapestry za predloge

Predloge je možno povezati s posebnim tipom komponente, tj. z razporeditvijo (ang. Layout). Razporeditev je okvir, v katerem posamezne strani prikazujejo vsebino. Razporeditev v strani vključi knjižnice JavaScript in stilske datoteke CSS. Tapestryjeve komponente omogočajo uporabo atributov in s tem spreminjanje njihovega videza, delovanja in drugih lastnosti [21].

## 5.6 Povezovanje in preverjanje podatkov

Privzete komponente pri Tapestryju samodejno izvajajo preverjanje na strani odjemalca in na strani strežnika, ko so te omejitve nastavljene. Za vsak obrazec je mogoče izvesti preverjanje po meri, ki ustreza dogovoru oziroma jo pravilno anotiramo. Dogovor o načinu poimenovanja metod za upravljanje in preverjanje je oblike `onValidateFromImeObrazca`, kjer je *ImeObrazca* ime obrazca v predlogi [21]. V imenu metode *onValidate* določa stopnjo življenjskega cikla. Zamenjamo ga lahko z *onSubmit* in tako dobimo ime upravljalne metode. Z anotacijo `@OnEvent(value=EventConstants.SUBMIT,component="ImeObrazca")` je možno nadomestiti celoten postopek imenovanja. Pri metodi za preverjanje uporabimo *VALIDATE* namesto *SUB-*

*MIT*.

Tapestry ločuje komponento od njene vrednosti, zato med preverjanjem napako dodamo na komponento obrazca. Napaka mora biti povezana še s komponento, na kateri je do napake prišlo. Komponente se lahko injicirajo preko IoC, za kar uporabimo anotaciji `@Component` ali `@InjectComponent`, kjer slednja velja le za bralni dostop [23]. Izvorna koda 5.2 prikazuje javljanje napake.

```
public class TitlePage {
      @Inject
      protected ComponentResources componentResources;

      @Property
      private String title;

      @InjectComponent
      private TextField titleField;

      @Component
      private Form titleform;

      public void onValidateFromTitleForm() {
        if(/* ali naziv obstaja v podatkovni bazi */) {
          titleform.recordError(titleField, componentResources.getMessages()
                  .get("titleField-exists-message"));
        }
      }

      // ...

      public Object onSubmitFromTitleForm() {
        // shrani naziv v podatkovno bazo
        return OtherPage.class;
      }

}
```

Izvorna koda 5.2: Metoda za preverjanje po meri s primerom javljanja napak

Izvorna koda 5.2 prikazuje izvedbo upravljalne metode in uporabo jezikovnih svežnjev. Upravljalna metoda na koncu vrne razred strani, na katero se preusmeri uporabnik. Tapestry ne omogoča uporabe kompleksnih tipov struktur, kot so Javanska zrna za vhodne podatke. Za pisanje v po-

datkovno bazo moramo entitete ročno napolniti. Metoda za preverjanje v primeru napake prebere sporočilo napake, definirano v globalnem svežnju ali v svežnju sporočil, ki je specifičen za trenutno stran. Sporočilo nato pošlje na stran pogleda. Povezovanje podatkov se dogaja preko t. i. prevajalnikov (ang. Translator), ki pretvarjajo podatke iz obrazca v želeno obliko v razredu strani. Tapestry omogoča definicijo podatkovnih prevajalnikov po meri, ki se uporabljajo v predlogah.

## 5.7 Podpora za IoC

Ogrodje Tapestry implementira lasten sklad IoC in omogoča zunanjo integracijo z drugimi tehnologijami za IoC, kot je ogrodje Spring. Izvorna koda 5.2 prikazuje uporabo injiciranja odvisnosti, ki so lahko interne storitve, polja in vrednosti. Z uporabno injiciranja preko vmesnikov je možno implementacije storitev zamenjati brez popravljanja izvorne kode, zato je uporaba tega pristopa priporočena.

## 5.8 Lokalizacija

Tapestry podpira večjezičnost in svežnji sporočil so globalni ali lokalni za vsako stran. Jezik se prevzame po nastavitvah brskalnika. Če aplikacija ne podpira pridobljenega jezika, se uporabi privzeti jezik. Podprte jezike nastavimo kot nastavitve v razredu aplikacijskega modula `AppModule` [21]. Nastavitev je v obliki seznama, kjer prvi jezik v seznamu velja kot privzeti jezik. Formatiranje izhodnih podatkov se doseže z uporabo posebne oznake `t:output` v predlogah. Oznaki se z atributi določi format izpisa in ime spremenljivke v razredu strani.

## 5.9 Podpora REST

Tapestry nima integrirane podpore za spletne storitve REST. Podporo je možno dodati z uporabo zunanje knjižnice RESTEasy in knjižnice za integracijo tapestry-resteasy.

## 5.10 Knjižnice ospredja (JS, AJAX)

Določene komponente, vključene v ogrodju Tapestry, imajo že vključeno podporo za JavaScript in AJAX. Tapestry uporablja lastno knjižnico JavaScript, ki implementira vse potrebne funkcionalnosti. Za uporabo knjižnice jQuery, se doda še integracijsko knjižnico tapestry5-jquery, ki razreši konflikte.

Tapestry omogoča hitro izdelavo strani, ki uporabljajo asinhrone klice AJAX. Komponenta `t:zone` definira območje na strani, ki se lahko asinhrono spreminja ali osvežuje. Komponenta `t:actionLink` sproži klic, ki se odraža na strežniški strani z novo vsebino za območje. To je vidno v primeru izvorne kode 5.3.

```
<!-- ajaxexample.tml -->
<!-- ... -->
<t:actionlink t:id="someLink" zone="myzone">update</t:actionlink>
<!-- ... -->
<t:zone t:id="myZone" id="myzone">
  The current time is ${currentTime}
</t:zone>
<!-- ... -->

// AjaxExample.java
// ...
@Inject
private Request request;

@InjectComponent
private Zone myZone;
// ...
Object onClickFromSomeLink()
{
  return myZone.getBody(); // zahtevek AJAX vrne vsebino oznake t:zone
}
```

```
// ...
```

Izvorna koda 5.3: Uporaba klica AJAX in oznake `t:zone` [22]

## 5.11 Dokumentacija

Dokumentacija ogrodja Tapestry je ena od šibkih točk ogrodja. Obstaja namreč samo ena knjiga, ki opisuje delovanje različice 5.0 [21], in ta je stara že šest let. Spletna dokumentacija je tudi pomanjkljiva [23]; opisi razredov in komponent so površinski, primeri in vadnice preveč enostavni. Obstaja posebna stran s primeri implementacije določenih rešitev [24], ki je v pomoč pri učenju. Na žalost je tudi ta vir površen. Učimo se lahko po metodi poskusov in napak ter z brskanjem po programerskih forumih, kot sta *http://stackoverflow.com/* in *http://apache-tapestry-mailing-list-archives.1045711.n5.nabble.com/*.

# Poglavje 6

# Primerjava ogrodij

## 6.1 Primerjava funkcionalnosti

Opisana ogrodja so si po izbranih funkcionalnostih podobna, saj vsa podpirajo večino od njih. Razlike postanejo očitne šele pri uporabi. Tabela 6.2 prikazuje razlike med ogrodji, ki se jih ne da razbrati iz dokumentacije.

Tabela 6.1: Tabelarični pregled funkcionalnosti

| **Funkcionalnost** | **Spring MVC** | **Stripes** | **Tapestry** |
|---|---|---|---|
| Pogoni predlog | | | |
| Integriran pogon predlog | × | × | ✓ |
| Možnost uporabe različnih pogonov predlog | ✓ | ✓ | × |
| Prisotnost knjižnice komponent in pripomočkov | ✓ | ✓ | ✓ |
| Podpora razporeditvam | × | ✓ | ✓ |
| Povezovanje in preverjanje | | | |
| Preverjanje in konverzija formatov | ✓ | ✓ | ✓ |
| Preverjanje na strani odjemalca | × | × | ✓ |
| Povezovanje podatkov v gnezdena Javanska zrna | ✓ | ✓ | × |
| Povezovanje napake pri preverjanju s poljem in z vrednostmi | × | ✓ | × |

| | | | |
|---|---|---|---|
| Možnost prikaza napak pri preverjanju za vsako polje posebej | ✓ | ✓ | × |
| Sledenje konceptu MVC | | | |
| Striktna ločnica med pogledom in modelom | ✓ | ✓ | × |
| Striktna ločnica med krmilnikom od modelom | ✓ | × | × |
| Striktna ločnica med krmilnikom in pogledom | ✓ | ✓ | × |
| Obveščanje pogleda o spremembi modela | × | × | × |
| IoC | | | |
| Podpora za IoC | ✓ | ✓ | ✓ |
| Integrirana možnost razvoja aplikacijskih storitev | ✓ | × | ✓ |
| Lokalizacija | | | |
| Uporaba svežnjev sporočil | ✓ | ✓ | ✓ |
| Samodejno povezovanje polja s prevedeno obliko | × | ✓ | ✓ |
| Možnost uporabe prevoda za določen tip napake na več poljih | × | ✓ | × |
| Vrivanje vrednosti v sporočilo napake | × | ✓ | ✓ |
| Spletne soritve in asinhroni klici | | | |
| Integrirana podpora storitvam REST | ✓ | ✓ | × |
| Integrirana podpora komponentam AJAX | × | × | ✓ |
| Možnost implementacije vmesnika SOAP | ✓ | × | × |
| Integrirane knjižnice JavaScript | × | × | ✓ |
| Druge funkcionalnosti | | | |
| Podpora implementacije čarovnika | × | ✓ | × |
| Način učenja in dela | | | |
| Odprtokodno ogrodje | ✓ | ✓ | ✓ |
| Dogovor pred konfiguracijo | × | ✓ | ✓ |

| | | | |
|---|---|---|---|
| Uporaba standardnih tehnologij | ✓ | ✓ | × |
| Položna učna krivulja | × | ✓ | × |
| Razumljiva in zadostna dokumentacija | ✓ | ✓ | × |

Ogrodje Tapestry vsebuje lasten pogon za procesiranje predlog, kar onemogoča izbiro druge tehnologije. Spring MVC in Stripes izbire ne omejujeta zaradi drugačne zasnove. Oba podpirata tehnologijo JSP, ki je med razvijalci najbolj razširjena, saj je osnova vseh tehnologij. Razvijalci lahko tako hitro in enostavno prehajajo med različnimi ogrodji, ker je JSP podprt tudi v kombinaciji z ogrodjem JSF in tehnologijo Portal. Razporeditev (ang. Layout) je možno doseči z uporabo navadnih oznak JSP. Ogrodje Stripes definira posebne oznake za določanje razporeditve, ki omogočajo višjo modularnost ospredja in s tem tudi lažje naknadno spreminjanje videza strani. Razporeditve so možne tudi pri ogrodju Tapestry, kjer razporeditev velja za poseben tip komponente po meri.

Vsa tri ogrodja omogočajo preverjanje vhodnih podatkov in povezovanje s polji na modelu. Pri Tapestryju je omogočena preslikava kriterijev za preverjanje tudi na stran odjemalca, kjer JavaScript poskrbi, da se uporabniku prikažejo napake, še preden se podatki pošljejo na strežnik. S tem se izboljša uporabniška izkušnja in zmanjša vpliv na strežnik. Velik pomen ima tudi povezovanje podatkov v Javanska zrna, ki se lahko pošljejo poslovni logiki. Te možnosti Tapestry nima. Pri razvoju ospredja pogosto želimo prikazati sporočila o napaki pri vsakem polju posebej. Pri ogrodju Tapestry se v primeru izklopa klicev JavaScript napake prikažejo le v kupčkih. Ogrodje Stripes omogoča razdelitev prevodov na hierarhičen način. Definira se splošen prevod za neki tip napake, ki vsebuje zastavico `{0}`, v katero se ob napaki samodejno vnese ime polja, kjer je do napake prišlo [18]. Ta funkcionalnost zmanjša količino besedila, ki ga je treba prevesti. Podobno omogoča tudi Spring v ročni obliki, a brez uporabe funkcionalnosti JSR-303.

Stripes, Spring MVC in Tapestry v grobem sledijo konceptu MVC. Ogrodje Tapestry koncept poenostavlja in s tem krši pravilo ločevanja delov MVC

na vseh nivojih. Vmesne spremenljivke morajo biti zato definirane na razredu strani (krmilniku). Tehnologija JSP je samostojna in omogoča pisanje spletnih strani neodvisno od krmilnika. Predloge Tapestryja so zgolj predloge, ker svoj kontekst hranijo na krmilniku. To povečuje kompleksnost kode in zmanjšuje berljivost. Model pri ogrodjih Stripes in Tapestry je dostopen preko krmilnikov (akcijsko zrno pri ogrodju Stripes, stran pri ogrodju Tapestry). Spring model prenese na pogled preko objekta zahtevka. Pri ogrodju Spring moramo vsak objekt ročno dodati na model s klicem metode `addObject(String key, Object obj)` na instanci razreda `ModelAndView`, ki nosi ime strani, na katero se zahtevek interno preusmeri. Podobno delovanje je opisano v 2.6.

Spring in Tapestry omogočata razvoj pripomočkov, kot so interne storitve. Pri ogrodju Stripes se za ta namen uporablja integracija z ogrodjem Spring.

Integriranje implementacije JSR-303, ki omogoča preverjanje Javanskih zrn, povzroči pri razvoju z ogrodjem Spring veliko preglavic. Te so opazne tudi pri lokalizaciji, saj integracija ne omogoča uporabe istega svežnja sporočil kot ostali del ogrodja. Poleg tega določenih napak ni možno prevesti, ker se odražajo kot izjeme/napake. Na področju lokalizacije se je ogrodje Stripes najbolje odrezalo.

Na področju spletnih storitev je Spring najmočnejši v primerjavi z ostalima dvema, ker je to ogrodje najbolj splošno med vsemi tremi in ponuja celo paleto razširitev. Med opisanimi samo Tapestry omogoča uporabo komponent, ki podpirajo AJAX. Razvijalec se mora prilagoditi knjižnici JavaScript, ki jo uporablja Tapestry pri vseh komponentah. Obstaja tudi modul tapestry-jquery, ki omogoča uporabo razširjene knjižnice jQuery.

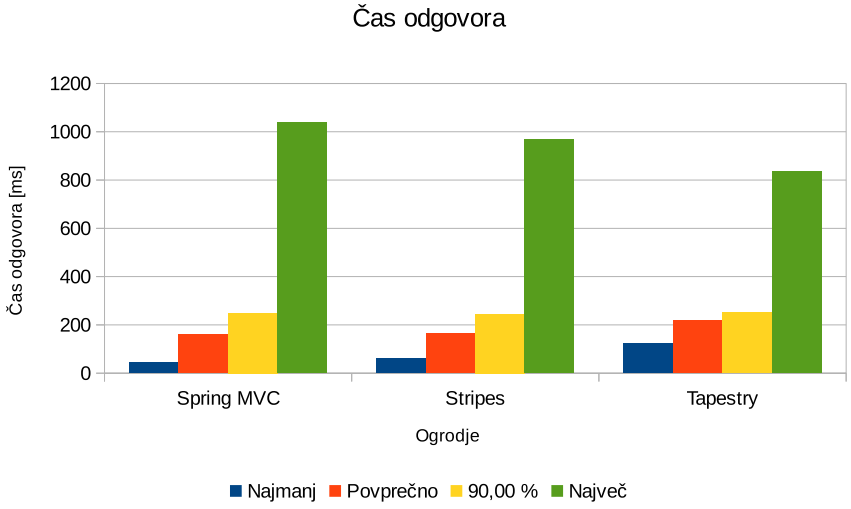
Ogrodje Tapestry omogoča sodobne prijeme z uporabo enega samega ogrodja, kar je podobno strategiji ogrodja JSF. Kompleksnost takih orodij je visoka, ker moramo spletne koncepte preslikati v koncepte navadnih aplikacij. Kompleksnost upočasnjuje razvoj, poveča število napak in zmanjšuje fleksibilnosti, ker so komponente vnaprej definirane.

Pri mnogih podjetjih obstaja ostra ločnica med razvijalci ospredja in raz-

vijalci ozadja. Ogrodji JSF in Tapestry poskušata to ločnico zabrisati ter v enega združiti razvijalca ozadja in ospredja, kar pogosto ni možno. Učna krivulja za ogrodij Spring in Tapestry je strma, a se pri Springu hitreje izravna.

## 6.2 Primerjava hitrosti odgovora

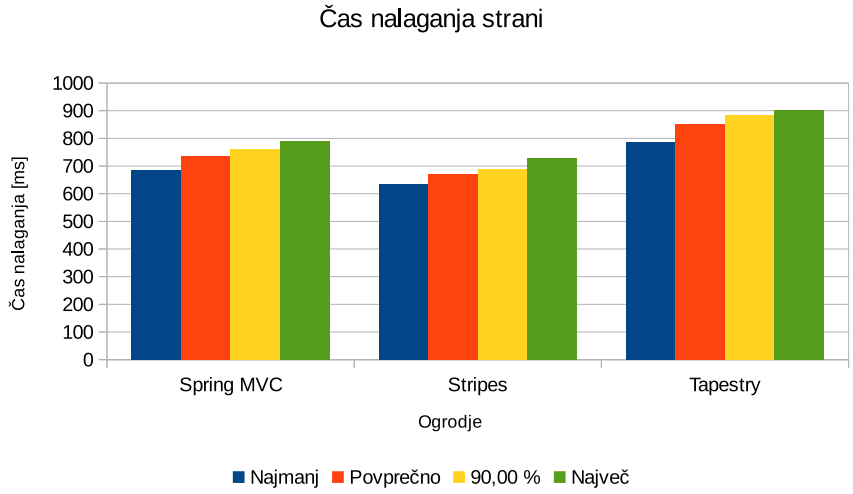
Ogrodje Tapestry velja za odzivno [21, str. 63], ker viri in instance strani čakajo na zahtevek v bazenih, vendar je iz testov razvidno, da so si po hitrosti vsa tri ogrodja zelo blizu. Meritve, predstavljene v grafih 6.1 in 6.2, so bile izvedene z orodjem Apache JMeter [26], ki se uporablja za testiranje zmogljivosti.




Slika 6.1: Primerjava hitrosti odgovora samo za zahtevano stran brez virov

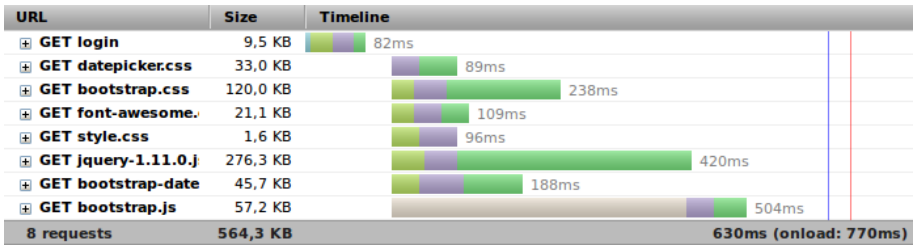
Stolpčni diagram 6.2 prikazuje čase nalaganja strani z vsemi vključenimi viri za vsa tri ogrodja. Razvidno je, da ogrodje Tapestry potrebuje več časa za nalaganje strani, kar je posledica večjega števila virov, ki so vključeni na strani.

Za boljšo primerjavo je bil uporabljen vtičnik Firebug [27] za brskalnik Firefox, s katerim so bili posneti prenosi strani za vsako od treh ogrodij, ki jih



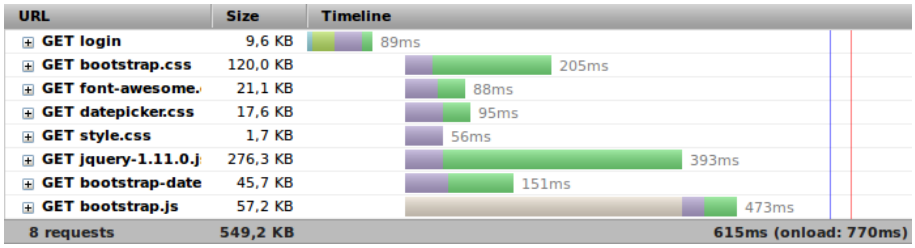

Slika 6.2: Primerjava hitrosti nalaganja enostavne strani

lahko vidimo na slikah 6.3, 6.4 in 6.5. Iz rezultatov poskusa je razvidno, da se pri ogrodju Tapestry prenaša veliko število virov v primerjavi z ostalima dvema ogrodjema.

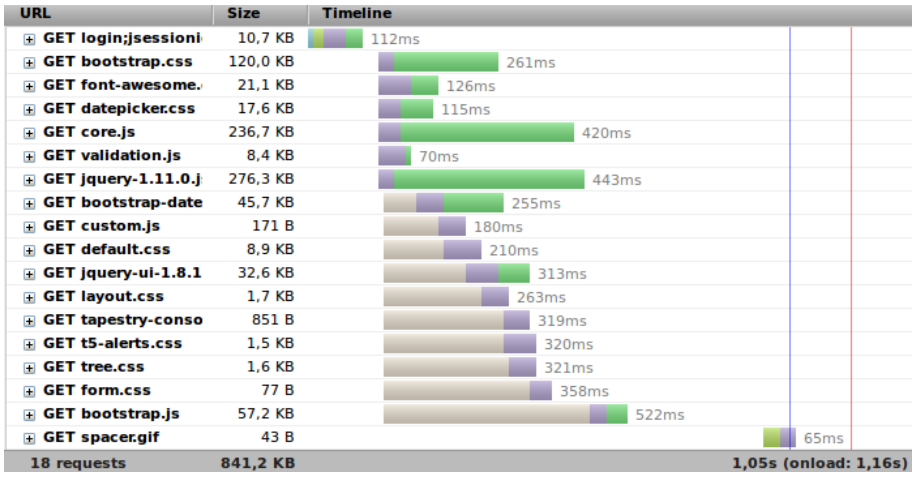



Slika 6.3: Odgovor ogrodja Spring MVC z vsemi viri (630 ms)

Ogrodja so si po hitrosti podobna, vendar je opazno, da je pridobivanje virov pri ogrodju Tapestry nekoliko počasnejše. Razlog za razliko bi lahko bilo programsko upravljanje z viri, ki ga uporablja ogrodje Tapestry. Pri ogrodjih Stripes in Spring za streženje teh virov skrbi kar vsebnik *servletov*.




Slika 6.4: Odgovor ogrodja Stripes z vsemi viri (615 ms)




Slika 6.5: Odgovor ogrodja Tapestry z vsemi viri (1050 ms)

## 6.3 Razvoj in vzdrževanje

S kompleksnostjo ogrodja se veča težavnost razvoja in vzdrževanja. Pri kompleksnih ogrodjih je težko predvideti čas razvoja projekta. Pri enostavnejših ogrodjih je delo nekoliko lažje in od razvijalca zahteva manj učenja ter dela.

Ogrodje Tapestry se ponaša z večjim številom funkcionalnosti, a na žalost mu škodita prevelika kompleksnost in pretirana uporaba pravila, da ima dogovor prednost pred nastavitvami. Tapestry omejuje z veliko količino pravil in s preveč funkcijami, ki jih morajo razvijalci vzdrževati in pregledovati. Razvoj do določene točke poteka brez zapletov, vendar s časom postane zamuden in otežen. Že sam zagon aplikacije med razvojem potrebuje približno 30 sekund, kar je za normalno delo in iskanje napak predolgo. Ogrodje se ne

obnaša na način, ki ga razvijalec pričakuje. K temu pripomore tudi nekonsistenca aplikacijskega programskega vmesnika (ang. Application Programming Interface - API). Poleg nekonsistence je za vzdrževanje problematično tudi pogosto spreminjanje vmesnika API. S podobnimi težavami se sooča standardizirano ogrodje JSF, ker je stopnja kompleksnosti in abstrakcije nad osnovnimi gradniki tako visoka, da je težko odpravljati napake.

Ogrodji Stripes in Spring sta dobro zasnovani in stabilni. Spring MVC na pogled deluje kot primitivno ogrodje, a sta njegovo vzdrževanje in dograjevanje enostavni. Pri razvoju sta zahtevni začetno postavljanje okolja in razširjanje funkcionalnosti, ko pa je osnova postavljena, je nadaljnji razvoj preprost. Omogočeno je vzporedno delo na ozadju, ospredju in na vmesnikih med njima. Ogrodje Stripes ne zahteva veliko dela pri podpornih razredih (implementiran primer vsebuje le 8 razredov - Spring 16, Tapestry 22), moramo pa vnaprej oceniti velikost projekta in upoštevati potrebne sloje abstrakcije. V nasprotnem primeru obstaja nevarnost, da bodo akcijski razredi dolgi nekaj tisoč vrstic, kar oteži vzdrževanje in nadgradnjo.

## 6.4 Prednosti in slabosti

Tabela 6.2 prikazuje dobre in slabe lastnosti opisanih ogrodij. Lastnosti so povzetek

Tabela 6.2: Preglednica prednosti in slabosti

| **Ogrodje** | **Pozitivne (+) in negativne lastnosti (-)** |
|---|---|

| | |
|---|---|
| **Spring MVC** | + Splošnost<br>+ Možnost za širitev<br>+ Veliko obstoječih modulov<br>+ Močno ogrodje za razvoj poslovne logike<br>+ Klasičen in enostaven koncept MVC<br>+ Dobra podpora integraciji z drugimi sistemi<br>+ Velika skupnost<br>+ Veliko dobre dokumentacije |
| | – Strma krivulja učenja<br>– Funkcionalnost JSR-303 ni primerna<br>– Povezovanje podatkov je okorno |
| **Stripes** | + Preprostost<br>+ Omogoča zelo hiter razvoj strežniške logike<br>+ Dobra podpora lokalizaciji<br>+ Poslovna logika predana ogrodju Spring<br>+ Položna krivulja učenja<br>+ Dobra dokumentacija |
| | – Podpora implementacije le spletnega dela aplikacije<br>– Majhna skupnost<br>– Stagnacija razvoja |
| **Tapestry** | + Veliko komponent<br>+ Veliko naprednih funkcionalnosti<br>+ Dober osnovni koncept<br>+ Enostavno osveževanje le dela strani |

| | |
|---|---|
| | – Zelo strma krivulja učenja<br>– Kompleksnost otežuje vzdrževanje in učenje<br>– Omejena parametrizacija komponent<br>– Nestabilnost in napake v API-ju<br>– Pogoste spremembe API-ja<br>– Slaba dokumentacija in pomanjkanje te |

## 6.5 Diskusija

V tem poglavju smo primerjali lastnosti ogrodij Spring MVC, Stripes in Tapestry po različnih kriterijih. Primerjana ogrodja so si podobna in se razlikujejo v le nekaterih točkah. Ogrodje Stripes v zameno za čist in enostaven programski vmesnik ponuja le najnujnejše funkcionalnosti za razvoj strežniškega dela spletne aplikacije. Stripes določene funkcionalnosti prepušča drugim ogrodjem. Ogrodje Spring MVC ponuja klasičen vmesnik MVC in celoten nabor rešitev za razvoj strežniškega dela spletnih ter drugih aplikacij. Tapestry prinaša funkcionalnosti, ki omogočajo razvoj naprednih dinamičnih aplikacij z asinhronimi komponentami, možnostjo zalednih storitev in drugih rešitev, vendar je zaradi tega ogrodje kompleksno. Tapestry je zahteven za razvoj, vzdrževanje in učenje. Problem predstavlja še stabilnost API-ja, ki se pogosto spreminja.

Analiza zmogljivosti je pokazala, da so si ogrodja podobna. Opazna je le razlika pri nalaganju celotne strani ogrodja Tapestry, ki v povprečju zahteva približno 200 ms več od ostalih dveh. Daljši čas nalaganja je verjetno posledica večjega števila statičnih virov, ki jih mora odjemalec prenesti s strežnika, in zakasnitve upravljavca virov.

V primerjavi izstopa razlika v količini dokumentacije na voljo. Ogrodje Tapestry nima zadostne literature, Spring MVC se ponaša z več dokumentacije, ker je tudi bolj prisoten na tržišču [34]. Ogrodje Stripes ima zadovoljivo dokumentacijo, čeprav je ni veliko.

Zaradi navedenih razlogov odsvetujemo uporabo ogrodja Tapestry. Svetujemo uporabo ogrodji Spring in Stripes za vse velikosti projektov, ker sta hitrejši, bolj stabilni in imata boljšo dokumentacijo.

# Poglavje 7

# Razvoj spletne aplikacije z uporabo ogrodja Stripes

## 7.1 Izbrano ogrodje

Ogrodje Stripes je bilo izbrano za podrobnejši pregled, ker je po lastnostih primerljivo z vsemi sodobnimi spletnimi ogrodji. V primerjavi z drugima dvema ogrodjema je Stipes preprostejši, čas razvoja z njim je krajši, količina potrebne programske kode pa je manjša. Polega tega so arhitektura ogrodja in funkcionalnosti premišljene, učenje je hitro in enostavno, dokumentacija zadostna.

## 7.2 Priprava razvojnega okolja

Za razvoj spletne aplikacije je bila uporabljena distribucija JEE integriranega razvojnega okolja IDE (Integrated Development Environment) Eclipse, ki si ga lahko prenesemo s spletnega naslova *http://www.eclipse.org*. V času pisanja je najnovejša različica okolja Eclipse 4.3. Za hitrejši razvoj je treba namestiti še vtičnike Eclipse WTP (dostopno preko integriranih repozitorijev), Spring tools (*http://spring.io/tools*) in m2eclipse (*https://www.eclipse.org/m2e/*) [28].

Poleg razvojnega okolja moramo namestiti še paket Java 7 JDK (Java Development Kit), ki je dostopen na: *http://openjdk.java.net/*, oziroma različico podjetja Oracle *http://java.oracle.com/* in strežnik Apache Tomcat 7 (*http://tomcat.apache.org/*). V Eclipsu dodamo Javansko izvajalno okolje in strežniško okolje Tomcat 7 ter ustvarimo instanco strežnika [28].

Projekt ustvarimo za orodje Apache Maven, ki omogoča modularnost in prenosljivost med razvijalci, ker poskrbi za pripravo celotnega okolja v postopku prevajanja in pakiranja. Apache Maven uporablja opise modulov v obliki datotek POM (Project Object Model). Vsak modul ali podmodul mora vsebovati svojo datoteko pom.xml, ki opisuje način gradnje in pakiranja projekta [31].

Ustvariti moramo enostaven projekt Maven z imenom *webcontest*, v katerem sta vključena dva modula [30]. Projekt kot oznako skupine vsebuje vrednost: *org.zabica.webcontest*, ki jo dedujejo vsi podmoduli. Najprej ustvarimo podmodul: *common* s pomočjo arhetipa: *maven-archetype-quickstart*. V čarovniku tega modula nastavimo Javanski paket na vrednost: *org.zabica.webcontest.common* [30] in drugi modul *stripes*, ki je izpeljan iz arhetipa *maven-archetype-webapp* z Javanskim paketom: *org.zabica.webcontest.stripes*.

Vsebine modula *common* v tem dokumentu se ne komentira. Datoteka pom.xml *modula stripes* mora vsebovati reference na knjižnice, od katerih je odvisen modul. Izvorna koda 7.1 prikazuje definicijo nekaj glavnih odvisnosti. V projektu sta uporabljeni še knjižnici `slf4j` za abstrakcijo dnevniški zapisov in `commons-lang3`, ki prinaša razna priročna orodja. Odvisnosti se dodaja z vmesnikom m2e v okolju Eclipse.

```
<!-- ... -->
 <dependency>
   <groupId>net.sourceforge.stripes</groupId>
   <artifactId>stripes</artifactId>
   <version>1.5.7</version>
 </dependency>

 <dependency>
   <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-webmvc</artifactId>
    <version>4.0.0.RELEASE</version>
  </dependency>

  <dependency>
    <groupId>org.zabica.webcontest</groupId>
    <artifactId>common</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </dependency>
<!-- ... -->
```

Izvorna koda 7.1: Glavne knjižnice, ki so navedene v datoteki pom.xml

Pomembno je, da definiramo različico jezika Java, ki jo želimo uporabiti. To storimo s parametri za vtičnik Maven, ki skrbi za prevajanje izvorne kode. Kako to storimo, si lahko ogledamo na primeru izvorne kode 7.2.

```
<!-- ... -->
 <build>
   <plugins>
     <plugin>
       <artifactId>maven-compiler-plugin</artifactId>
       <version>3.1</version>
       <configuration>
         <source>1.7</source>
         <target>1.7</target>
       </configuration>
     </plugin>
   </plugins>
 </build>
<!-- ... -->
```

Izvorna koda 7.2: Nastavitev prevajalnega vtičnika na različico Jave 7 v datoteki pom.xml

Maven definira posebno drevesno strukturo map za spletne aplikacije, ki je razvidna iz izvorne kode 7.3. Vsebina je v mapi *src/main*, kjer je delitev na izvorno kodo Java (mapa *java*), druge vire (*resources*) in datoteke spletne aplikacije (*webapp*). Maven v ciklu pakiranja datoteke WAR iz lastne strukture sestavi v standardno strukturo paketa WAR (Web Application Archive).

```
stripes
|-- src
```

```
|-- main
   |-- java
   |   |-- org
   |       |-- zabica
   |           |--webcontest
   |              |-- ...
   |-- resources
   |-- webapp
      |-- WEB-INF
```

Izvorna koda 7.3: Videz drevesne strukture map modula Stripes

Izvorno kodo pišemo v mapo *src/main/java*, ki se kaže tudi kot neprevedena lokacija poti razredov (ang. Classpath).

V mapi WEB-INF sta datoteki web.xml in applicationContext.xml. Prva je standardni deskriptor spletne aplikacije, druga pa kontekstni XML ogrodja Spring. V datoteki web.xml se navedejo lokacija kontekstne datoteke Spring, poslušalec konteksta in integracijska zrna `SpringInterceptor` med ogrodjema Stripes in Spring, ki je v obliki interceptorja [32]. Integracijski interceptor poskrbi, da se celoten kontekst ogrodja Spring inicializira preko nastavljene datoteke. Poleg teh nastavitev moramo definirati še *servlet* za razpošiljanje ogrodja Stripes in Javanski paket v katerem se nahajajo akcijska zrna. Vsebina datoteke web.xml je razvidna iz izvorne kode 7.4.

```
<!-- ... -->
<filter>
   <filter-name>DynamicMappingFilter</filter-name>
   <filter-class>net.sourceforge.stripes.controller
.DynamicMappingFilter</filter-class>
   <init-param>
     <param-name>ActionResolver.Packages</param-name>
     <param-value>org.zabica.webcontest.stripes.actions</param-value>
   </init-param>

   <init-param>
     <param-name>LocalePicker.Class</param-name>
     <param-value>org.zabica.webcontest.stripes.converter.extensions
.SettingsBasedLocalePicker</param-value>
   </init-param>

   <init-param>
     <param-name>LocalePicker.Encoding</param-name>
```

```
      <param-value>UTF-8</param-value>
    </init-param>

    <init-param>
      <param-name>ActionBeanContext.Class</param-name>
      <param-value>org.zabica.webcontest.stripes.beans
.SessionActionBeanContext</param-value>
      </init-param>

      <init-param>
        <param-name>Interceptor.Classes</param-name>
        <param-value>net.sourceforge.stripes.integration.spring
.SpringInterceptor,org.zabica.webcontest.stripes.beans.LoginInterceptor</param-value>
      </init-param>
</filter>

<filter-mapping>
  <filter-name>DynamicMappingFilter</filter-name>
  <url-pattern>/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
</filter-mapping>

<servlet>
  <servlet-name>StripesDispatcher</servlet-name>
  <servlet-class>net.sourceforge.stripes.controller
.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

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

<listener>
  <listener-class>org.springframework.web.context
.ContextLoaderListener</listener-class>
</listener>

<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>
<!-- ... -->
```

Izvorna koda 7.4: Vsebina datoteke web.xml

V datoteki web.xml so poleg osnovnih preslikav *servlet* in nastavitev filtra še dodatne nastavitve za filter ogrodja Stripes:

- `ActionResolver.Packages` - javanski paketi, v katerih *servlet* poišče akcijska zrna.
- `LocalePicker.Class` - razred, ki implementira logiko za izbiro jezika.
- `LocalePicker.Encoding` - format kodiranja znakov.
- `ActionBeanContext.Class` - razred, ki razširja osnovni kontekstni razred ogrodja Stripes.
- `Interceptor.Classes` - seznam razredov interceptorjev, ki se izvajajo.

Z ogrodjem Spring lahko ustvarimo instanco zrna, ki skrbi za zapisovanje v podatkovno bazo. Referenca na to zrno se bo preko anotacije `@SpringBean` injicirala v akcijska zrna ogrodja Stripes.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
            xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-4.0.xsd">

       <bean id="dbbean" class="org.zabica.webcontest.stripes.beans.DBBean">
              <property name="storeFile" value="/tmp/store-stripes" />
       </bean>
</beans>
```

Izvorna koda 7.5: Vsebina datoteke applicationContext.xml, ki vsebuje samo zrno Spring za dostop do podatkovne baze

Privzeta mapa datoteke JSP pri ogrodju Stripes je mapa *views*, ki je v korenu paketa WAR; v tem primeru poti ne spreminjamo. Statična vsebina, datoteke JavaScript, ikone, datoteke CSS in ostala statična vsebina so v mapi *static* in v korenu paketa WAR.

Pred začetkom razvoja moramo aplikacijo objaviti na strežniku [28]. Med razvojem se bodo datoteke JSP na strežniku samodejno osveževale. Za osvežitev drugih delov aplikacije moramo objavo sprožiti ročno [28].

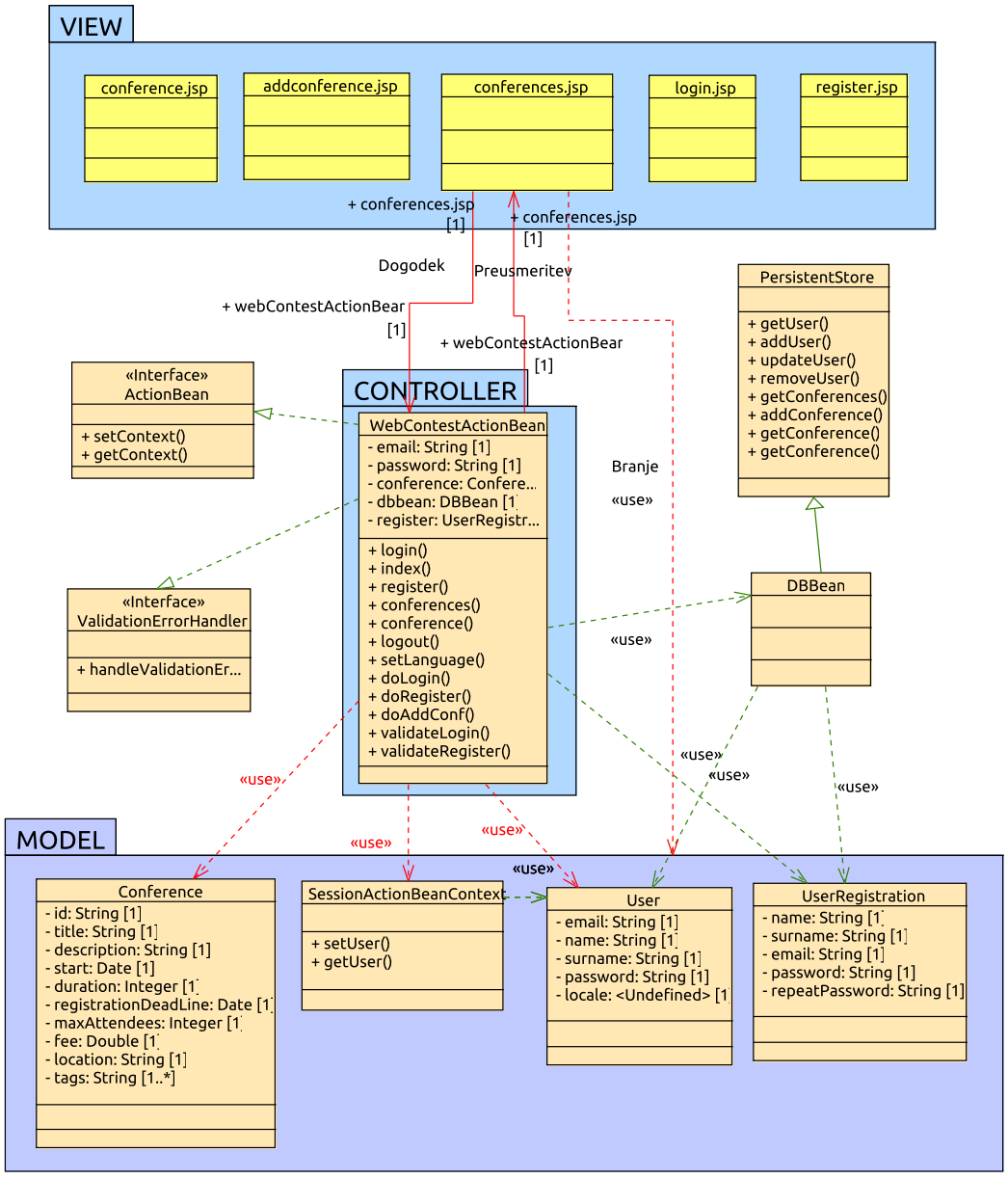
## 7.3 Sestava aplikacije

Modul *common* vsebuje razred `PersistentStore`, ki implementira komunikacijo s podatkovno bazo in podatkovne entitete (`User`, `Conference` itd.). Izbrana podatkovna baza je manjša integrirana baza, ki omogoča shranjevanje Javanskih objektov in iskanje po njih glede na tip. Baza se imenuje HyperGraphDB in je skrita za vmesnikom. Podatkovno bazo je tako možno zamenjati s katerokoli drugo. Razred `DBBean` razširja razred `PersistentStore` z življenjskim ciklom ogrodja Spring, ki inicializira in deinicializira podatkovno bazo ob zagonu ali zaustavitvi aplikacije. Slika 7.1 prikazuje razredni diagram UML (Unified Modeling Language).

Aplikacija je sestavljena iz prijavne strani (sliki 7.2 in 7.3), strani za registracijo (slika 7.4), pregled (slika 7.5), iskanje in dodajanje konferenc (slika 7.6). Preverjanje, ali je uporabnik prijavljen, se izvede z uporabo interceptorskega razreda `LoginInterceptor`, ki za vsak zahtevek preveri, ali je uporabnik prijavljen ali ne. Če uporabnik ni prijavljen, ga preusmeri na stran za prijavo [32].

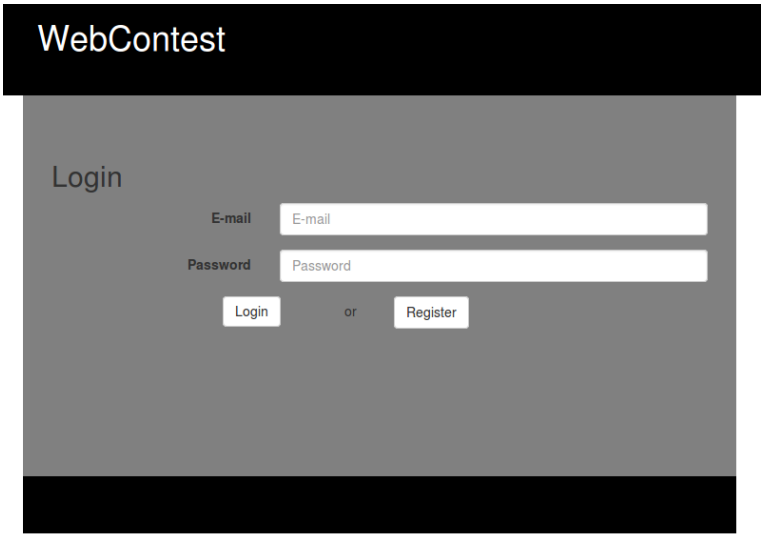
V strukturi projekta obstajajo še drugi razredi, ki so del razširitve osnovnih funkcionalnosti ogrodja Stripes. Prvi od teh je razred `SessionActionBean Context`, ki razširja kontekstni razred Stripes z neposrednim dostopom do sejnega parametra o uporabniku. Interceptorju prijave prisotnost te vrednosti pove, ali je uporabnik prijavljen ali ne. Naslednja sta pretvornika vhodnih podatkov. Prvi pretvarja oznake jezika *locale* v Javanske objekte tipa `Locale`. Drugi pretvarja nize oblike “a,b,c” v seznam tekstovnih nizov glede na vejico.

Razred `SettingsBasedLocalePicker` vsebuje logiko, ki izbere jezik za vsak zahtevek. Najprej preveri obstoj izbranega jezika v seji (naloži se ob




Slika 7.1: Razredni diagram UML projekta. Rdeče relacije predstavljajo ločnico med deli MVC

prijavi uporabnika) in v primeru, da ta ne obstaja, izbere jezik, ki ga posreduje brskalnik. Prijavljen uporabnik z izbiro zastavico države izbere jezik, ki se nato zapiše v podatkovno bazo in ohrani za naslednje obiske.

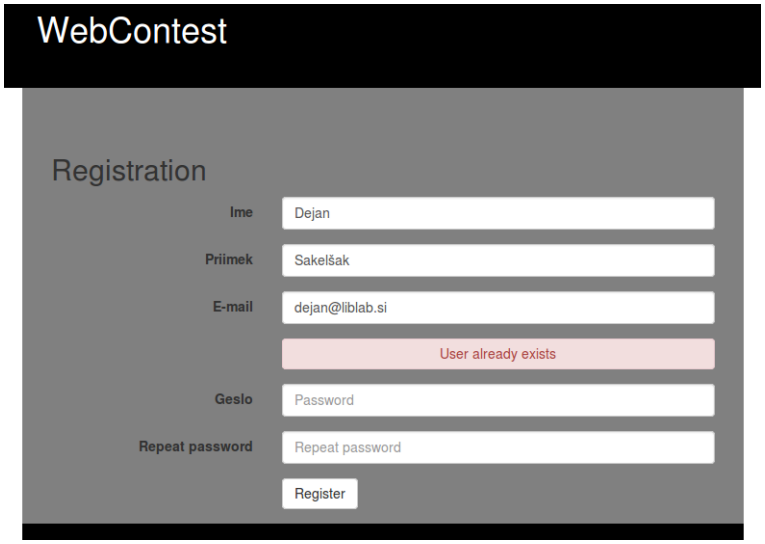



Slika 7.2: Prijavna stran

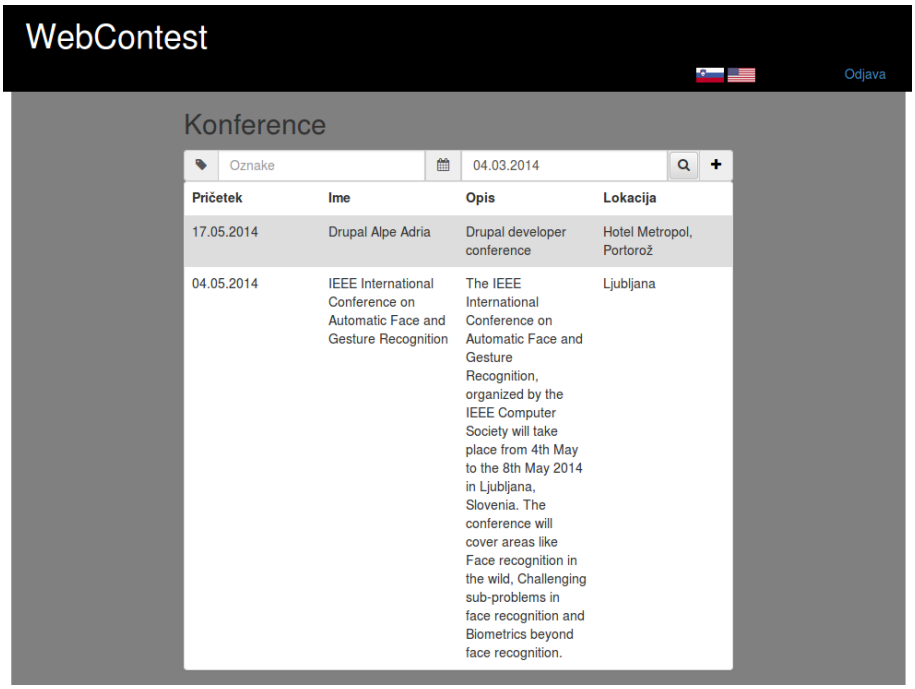

Slika 7.3: Napaka pri prijavi

Uporabniški vmesnik je zasnovan z uporabo popularne knjižnice Bootstrap s paleto komponent, ki se vključijo na strani. Na stran dodamo datoteko CSS, knjižnici JavaScript in iz dokumentacije [33] prepišemo del HTML, ki ga dodatno preuredimo z dodatnimi CSS-razredi.

Razviti sta bili dve strani JSP, ki predstavljata glavo in nogo vsake strani. Druge strani JSP vključijo to glavo in nogo, s čimer se dosežeta višja stopnja




Slika 7.4: Napaka pri registraciji




Slika 7.5: Filtriranje konferenc

modularnosti in lažje vzdrževanje. Izvorna koda 7.6 prikazuje okvir, ki ga uporabi vsaka stran.

Akcijsko zrno v izvorni kodi 7.8 upravlja z obrazcem izvorne kode 7.7, ki je izveden s komponentami Stripes. Komponenta Stripes `stripes:form` iz atributa `beanclass` pridobi razred akcijskega zrna in glede na vrednost anotacije `@UrlBinding` sestavi obliko HTML-oznake `form` s pravim naslovom zahtevka. Ker je akcijsko zrno dogodkovno naravnano, je potrebno na uporabniškem vmesniku s pritiskom na gumb "Prijavašprožiti dogodek. Ogrodje Stripes poenostavlja dogodke na gumbe vrste *submit*, kjer je dogodek ime gumba.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%-- knjiznice oznak JSTL --%>
<%@ taglib prefix="stripes" uri="http://stripes.sourceforge.net/stripes-dynattr.tld"%>

<%@ include file="header.jsp"%>

<article id="pageBody" class="container">
<%-- ... --%>
</article>

<%@ include file="footer.jsp"%>
```

Izvorna koda 7.6: Primer vključevanja glave in noge na strani

Dogodke lahko tudi ročno prožimo s klicem na naslov URL, oblike *http://naslov/stripes/webcontest/dogodek/*, ker Stripes ne ločuje GET- in POST-metod. Oblika naslova URL je takšna zaradi vrednosti, ki je nastavljena v akcijskem zrnu na anotaciji `@UrlBinding("/webcontest/{\$event}")`. To je vidno v obrazcu v vrstici, kjer je uporabljena komponenta `stripes:link`, ki ima nastavljen atribut *event* z vrednostjo *register*, katerega posledica je URL *http://naslov/stripes/webcontest/register*. V primeru napak pri preverjanju se ob poljih, kjer je prišlo do napake, prikažejo sporočila napak, ki jih Stripes avtomatsko prebere iz svežnja sporočil.

Preverjanje pri prijavi poteka v dveh korakih. Najprej se preveri format pri povezovanju (kriteriji v anotaciji `@Validate`). Če je format naslova e-pošte pravilen, se geslo primerja še z geslom, shranjenim v podatkovni bazi. Ko uporabnik s poslanim e-poštnim naslovom ne obstaja in/ali se gesli ne ujemata, se uporabniku vrne sporočilo o napaki, ki pove, da ge-

slo ne ustreza. To naredi metoda za preverjanje z dodajanjem prevedenega sporočila napake na mapo napak. Stripes z oznakama JSP `stripes:errors` in `stripes:individual-error` napake preverjanja prikaže na strani.

```
<%-- ... --%>
<aside id="loginFrame" class="row">
              <div class="col-md-6 col-md-offset-3">
                     <h2 id="formFrameTitle">
                            <fmt:message key="login.title" />
                     </h2>
              </div>
              <div class="col-md-6 col-md-offset-3">
                     <stripes:form beanclass="org.zabica.webcontest.stripes.actions
.WebContestActionBean" method="post" class="form-horizontal">
                            <div class="form-group">
                                   <stripes:label class="col-sm-4 control-label"
                                        for="email">
                                          <fmt:message key="login.email"
                                               var="loginemail"/>
                                          ${loginemail}
                                   </stripes:label>
                                   <div class="col-sm-8">
                                          <stripes:text class="form-control" id="email"
                                               name="email" placeholder="${loginemail}"/>
                                   </div>
                            </div>
                            <c:if test="${stripes:hasErrors(actionBean, 'email')}">
                                   <div class="form-group">
                                          <div class="col-sm-offset-4 col-sm-8">
                                                 <div class="erroralert">
                                                        <stripes:errors field="email">
                                                               <stripes:individual-error
                                                                    />
                                                        </stripes:errors>
                                                 </div>
                                          </div>
                                   </div>
                            </c:if>

<%-- polje gesla --%>

                            <div class="form-group">
                                   <div class="col-sm-offset-3 col-sm-2">
                                          <stripes:submit name="doLogin" class="btn
                                               btn-default">
                                                 <fmt:message key="login.submit" />
                                          </stripes:submit>
```

```
                                    </div>
                                    <div class="col-sm-1 or">
                                           <fmt:message key="login.or" />
                                    </div>
                                    <div class="col-sm-2">
                                           <stripes:link
                                                beanclass="org.zabica.webcontest.stripes
.actions.WebContestActionBean" event="register" class="btn btn-default" ><fmt:message
     key="login.register" /></stripes:link>
                                    </div>
                             </div>
                      </stripes:form>
               </div>
        </aside>
</article>
<%-- ... --%>
```

Izvorna koda 7.7: Stripes komponente na prijavni strani

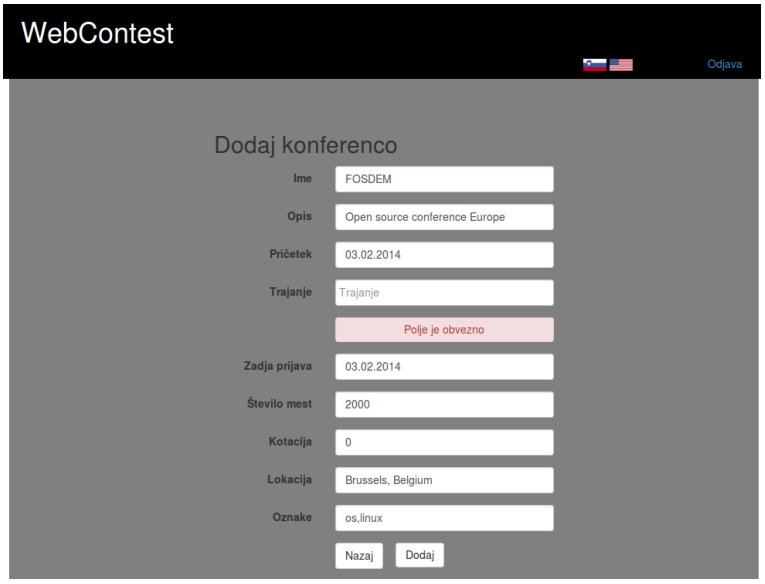
```
@UrlBinding("/webcontest/{$event}")
public class WebContestActionBean implements ActionBean, ValidationErrorHandler {

       private static final String EMAIL_PATTERN =
                     "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)
*(\\.[A-Za-z]{2,})$";

       private static final Logger LOG =
            LoggerFactory.getLogger(WebContestActionBean.class);

       @SpringBean
       private DBBean dbbean;

       private SessionActionBeanContext context;

       @Validate(required = true, mask = EMAIL_PATTERN, on = { "doLogin" })
       private String email;
       @Validate(required = true, on = { "doLogin" })
       private String password;

       // ... atributi razreda

       private Date now = new Date();

       // ... setter konteksta

       @DefaultHandler
       public Resolution index() {
```

```
        String evt = "login";
        if(this.context.getUser() != null) {
                evt = "conferences";
        }
        RedirectResolution rr = new RedirectResolution(WebContestActionBean.class,
             evt);
        LOG.debug("PATH: " + rr.getPath());
        return rr;
}

@HandlesEvent("login")
public Resolution login() {
        return new ForwardResolution("/views/login.jsp");
}

@HandlesEvent("doLogin")
public Resolution doLogin() {
        LOG.debug("User successfully logged in");
        return new RedirectResolution(WebContestActionBean.class, "conferences");
}

// ... druge upravljalne metode

@Override
public Resolution handleValidationErrors(ValidationErrors errors)
                throws Exception {
        for(List<ValidationError> errs : errors.values()) {
                for(ValidationError err : errs) {
                        LOG.debug(err.getFieldName() + " " +
                                err.getMessage(Locale.getDefault()));
                }
        }

        return null;
}

@ValidationMethod(on = { "doLogin" })
public void validateLogin(ValidationErrors errors) {
        User u = this.dbbean.getUser(this.email);
        if(u == null || !u.isPasswordValid(this.password)) {
                errors.add("email", new ScopedLocalizableError("",
                     "userPassInvalid"));
        } else {
                this.context.setUser(u);
        }
}
```

```
        // ... druge metode za preverjanje

        // ... getter in setter metode
}
```

Izvorna koda 7.8: Implementacija akcijskega zrna

Pri ostalih straneh so uporabljeni enaki prijemi in načini, razen pri strani za dodajanje konference, kjer se polje tags pretvori z uporabo razreda `TagsConverter`. Uporaba lastnih pretvornih razredov je prikazana v izvorni kodi 7.9.




Slika 7.6: Napaka pri dodajanju konferenc

```
  // ...
  @ValidateNestedProperties({
    // ...
    @Validate(field="tags", required=true, converter=TagConverter.class, on = { "doAddConf"
        } ),
  })
  private Conference conference;

  // ...
```

Izvorna koda 7.9: Uporaba lastnega pretvornika tipov v razredu WebContestActionBean

# Poglavje 8

# Sklepne ugotovitve

V diplomskem delu so bila primerjana tri ogrodja za platformo Java: Spring MVC, Stripes in Apache Tapestry. Primerjava je potekala na osnovi izvedbe enake aplikacije z uporabo vsakega od ogrodij. Aplikacija je bila zastavljena tako, da je vključevala vse funkcionalnosti, ki so bile predmet primerjave. Tematika aplikacije je bila vodenje koledarja konferenc. Uporabniki si na aplikaciji lahko ustvarijo račun, se vanj prijavijo, pregledujejo in filtrirajo konference ter dodajajo nove.

Skozi implementacijo aplikacij se je izkazalo, da so si ogrodja po večini funkcionalnosti podobna, v določenih primerih pa so se pokazala odstopanja. Pri ogrodju Tapestry je bil poglavitni problem pomanjkanje literature, kar je močno oteževalo razvoj in razumevanje delovanja ogrodja. Velika težava je bila tudi s spreminjanjem vmesnika API, saj je bila večina spletnih virov že zastarela. Pri razvoju z ogrodjem Spring MVC ni bilo drugih posebnosti, razen z integracijo standarda JSR-303. Pri tem se napake pri preverjanju ne povežejo pravilno z ogrodjem Spring MVC. Razvoj z ogrodjem Stripes je potekal tekoče brez zapletov.

Glede na izkušnje z izvedbo aplikacij priporočamo uporabo ogrodje Spring MVC, saj ponuja veliko modulov za implementacijo aplikacij. Tapestry pa odsvetujemo zaradi pomanjkanja dokumentacije in oteženega dolgoročnega vzdrževanja. Uporaba ogrodja Stripes za razvoj spletnih aplikacij je tudi

priporočena, težavo povzroča le izumirajoča skupnost. Pogosto se podjetja odločijo o izbiri ogrodja, ki bo služilo več projektom na dolgi rok, saj je tako lažje izobraziti kader in vzdrževati projekte. Za take primere je Spring MVC najprimernejša odločitev zaradi podpore podjetij, ki stojijo za ogrodjem, saj ta zagotavljajo kakovost.

Med ideje za nadaljnje raziskave bi lahko dali raziskavo vzrokov za zamrtje takšnih odprtokodnih skupnosti. Nadaljnji razvoj te teme bi lahko šel v primerjavo lastnosti ostalih Javanskih ogrodij, kot so Google Web Toolkit, ogrodje Play!, Wicket, Vert.x in druga.

Primerjave ogrodij in orodij so dober vir informacij v fazi snovanja projektov, saj pri začetni izbiri tehnologij pogosto ne poznamo vseh njihovih lastnosti.

# Literatura


[1] Netcraft. (2014, 2. april). *Web server survey*. Dostopno na: http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html

[2] W3C. (2014, 29. april). *HTML5*. Dostopno na: http://www.w3.org/TR/html5/

[3] Martin Fowler. (2014, 6. maj). *GUI Architectures*. Dostopno na: http://martinfowler.com/eaaDev/uiArchs.html

[4] Peter A. Pilgrim, *Java EE 7 Developer Handbook*, Birmingham, VB: Packt Publishing, 2013.

[5] Antonio Goncalves, *Beginning Java EE 7*, New York, ZDA: Apress, 2013.

[6] Arun Gupta, *Java EE 7 Essentials*, Sebastopol, ZDA: O'Reilly Media, 2013.

[7] Joaquin Picon idr., *Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server*, New York, ZDA: IBM Corporation, ITSO, 2000. Dostopno na: http://www.redbooks.ibm.com/abstracts/sg245754.html

[8] Martin Keen idr., *Developing Enterprise JavaBeans Applications*, New York, ZDA: IBM Corporation, ITSO, 2012. Dostopno na: http://www.redbooks.ibm.com/abstracts/REDP4885.html

[9] Wikipedia. (2014, 20. april). *Inversion of control*. Dostopno na: http://en.wikipedia.org/wiki/Inversion_of_control Valida

[10] Spring Framework. (2014, 12. maj). Dostopno na: http://www.spring.io

[11] Rod Johnson idr. (2014. 12. maj). *Spring Framework Reference Documentation*. Dostopno na: http://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf

[12] Spring. (2014. 10. junij). *View technologies*. Dostopno na: http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/view.html

[13] Apache Velocity. (2014. 10. junij). Dostopno na: http://velocity.apache.org/

[14] Freemarker. (2014. 10. junij). Dostopno na: http://freemarker.org/

[15] Spring. (2014. 12. maj). *Guides*. Dostopno na: https://spring.io/guides

[16] Rob Harrop, Clarence Ho, *Pro Spring 3*, New York, ZDA: Apress, 2012.

[17] Stripes Framework. (2014, 12. maj). Dostopno na: http://www.stripesframework.org

[18] Frederic Daoud, *Stripes: ... and Java web development is fun again*. Raleigh, ZDA: Pragmatic Bookshelf, 2008.

[19] Stripes Framework. (2014, 12. maj). *Documentation*. Dostopno na: http://www.stripesframework.org/display/stripes/Documentation

[20] Apache Tapestry. (2014, 19. maj). Dostopno na: http://tapestry.apache.org

[21] Alexander Kolesnikov, *Tapestry 5: Building Web Applications*, Birmingham, VB: Packt Publishing, 2008.

[22] Apache Tapestry 5. (2014, 19. maj). *Ajax and zones*. Dostopno na: http://tapestry.apache.org/ajax-and-zones.html

[23] Apache Tapestry 5. (2014, 19. maj). *Documentation*. Dostopno na: http://tapestry.apache.org/documentation.html

[24] Jumpstart. (2014, 19. maj). *Tapestry 5.3.7 examples*. Dostopno na: http://jumpstart.doublenegative.com.au/jumpstart/examples/

[25] Jumpstart. (2014, 19. maj). *What is Called and When*. Dostopno na: http://jumpstart.doublenegative.com.au/jumpstart/examples /navigation/whatiscalledandwhen

[26] Apache JMeter. (2014, 20. maj). Dostopno na: http://jmeter.apache.org/

[27] Firebug. (2014, 20. maj). Dostopno na: https://getfirebug.com/

[28] Eclipse. (2014, 22. maj). *Documentation*. Dostopno na: http://www.eclipse.org/documentation/

[29] m2eclipse plugin. (2014, 19. maj). *Documentation*. Dostopno na: http://eclipse.org/m2e/documentation/

[30] Bruce Snyder idr. (2014, 19. maj). *Introduction to m2eclipse*. Dostopno na: http://www.theserverside.com/news/1363817/Introduction-to-m2eclipse

[31] Sonatype Company, *Maven: The Definitive Guide*. Sebastopol, ZDA: O'Reilly Media, 2008.

[32] Wikipedia. (2014, 20. april). *Interceptor pattern.* http://en.wikipedia.org/wiki/Interceptor_pattern

[33] Bootstrap. (2014, 20. april), *Getting started.* Dostopno na: http://getbootstrap.com/getting-started/

[34] InfoQ. (2014, 19. maj). *Top 20 Web Frameworks for the JVM.* Dostopno na: http://www.infoq.com/research/jvm-web-frameworks

[35] Play Framework. (2014, 19. maj). Dostopno na: http://www.playframework.com/