Универзитет у Београду

Математички факултет

Мастер рад

Анализа перформанси *JSF* веб апликација

Студент:

Стефан Татић

Ментор:

др Филип Марић

септембар 2016.

Садржај

Апстракт

JavaServer Faces је један од заступљенијих радних оквира за развој Јава веб-апликација. У овом раду ће бити представљене основе *JSF* радног оквира, најбитније услуге које нуди, као и његове библиотеке. Биће описана структура *JSF* страница и основне компоненте које су заступљене у *JSF*-у, као и стабло компоненти које настаје приликом обраде *JSF* странице од стране веб-сервера. Биће објашњен појам Јава зрна и начин на који је повезан са *JSF* страницом.

У другом делу рада ће бити описана структура *JSF* пројекта чији је главни циљ представљање основа *JSF*-а и његових страница, као и начин организација страница коришћењем шаблона. У пројекту ће посебно бити анализиране перформансе *JSF* страница и њихово понашање на различитим клијентским рачунарима, у зависности од садржаја, функционалности странице и броја компоненти, с обзиром да приликом повећања броја компоненти на страници перформансе опадају. Резултати перформанси ће бити представљени кроз дијаграме. Такође, биће описане технологије које се користе у веб-апликацијама.

1. Увод

Посматрајући некадашње перформансе рачунара, с разлогом се може рећи да је визуелни изглед апликација био у другом плану. Такође, није се могло очекивати да рад у таквим условима буде брз и удобан. Са побољшањем перформанси рачунара, веома је битно да апликација која се извршава на рачунару ради брзо и једноставно, с посебном пажњом посвећеном удобности корисника.

У данашње време се приликом дизајнирања веб-апликација најчешће примењује вишеслојна архитектура коју чине презентациони слој, слој пословне логике и слој података. Презентациони слој приказује податке кориснику кроз кориснички интерфејс и представља највиши ниво апликације. Слој пословне логике имплементира пословну логику апликације која се састоји од пословних правила из стварног света и представља средњи слој апликације. У овом слоју се дефинише који подаци се чувају и на који начин обрађују. Слој података представља најнижи слој апликације и у овом слоју се подаци чувају, најчешће у базама података.

Пре настанка веб-апликација, постојале су апликације које су имале посебан кôд који се налазио на сваком клијентском рачунару и посебан кôд који се налазио на серверу. Апликација је имала свој клијентски програм који је садржао кориснички интерфејс и морао је бити инсталиран на сваком клијентском рачунару. Корисник је кроз клијентску апликацију постављао захтеве серверу, који је вршио обраду података и слао повратне информације (одговоре) кориснику кроз клијентску апликацију.

Данашње веб-апликације имају вишеслојну клијент-сервер архитектуру. Веб-апликације користе веб-документе (*HTML*) који су подржани од стране веб-прегледача. Веб-прегледач представља презентациони слој који комуницира са средњим слојем пословне логике. У веб-апликацијама средњи слој је нека од технологија са динамичким веб-садржајем као што су *ASP*, *ASP.NET*, *JSP/Java*, *JSF/Java*, *PHP*, *Perl*, *Python* или *Ruby on Rails*, док базе података представљају слој података. Веб-прегледач шаље веб-захтев средњем слоју који врши услугу тако што поставља упите бази података, а када се промене подаци у бази, прослеђује обрађене податке назад веб-прегледачу који приказује нови садржај.

Узевши у обзир развој веб-технолгија, сасвим је логична мисао да се једној од њих посебно посвети пажња, што због њених карактеристика и могућности, што због евентуалних проблема које са собом носи. Управо је *JavaServer Faces* (*JSF*) технологија која, по мишљењу аутора, завређује посебан осврт и анализу.

Са првим проблемима у развоју *JSF* апликације аутор овог рада се сусрео у свом пословном окружењу. *JSF* је био изабран за развој апликације која се бави управљањем ресурсима предузећа. У почетку, када су странице биле мањег обима, није било проблема. Међутим, како су странице постајале сложеније, тако су перформансе опадале. Циљ овог рада је да детаљније истражи овај феномен и предложи механизме за његово превазилажење.

2. Основне технологије које су коришћене у раду

У овом поглављу биће укратко описане основне веб-технологије које су коришћене у овом раду (*HTML*, *JavaScript*, *AJAX*, *jQuery*, *CSS*) као и неке од Јава веб-технологија. Биће описане и основне карактеристике *Maven*-а, како се у пројекат укључују екстерне Јава библиотеке (*jar*-ови) и које фазе животног циклуса изградње пројекта постоје.

2.1 Основне веб-технологије

HTML

HTML (енгл. *HyperText Markup Language*) је описни језик за означавање хипертекста намењен опису веб-страница [11]. Настао је осамдесетих година прошлог века. Када је *HTML* био у фази настанка, могао је само да прикаже статички текстуални садржај. У том периоду су и веб-прегледачи имали скромне могућности. Данас су веб-прегледачи веома напредни тако да могу да подрже и најзахтевније веб-апликације, које управљају динамичким саджајем и које врше обраду сложених података.

Битно је да *HTML* документ поштује стандарде да би приказ био исти у свим веб-прегледачима. *HTML* стандарде дефинише *World Wide Web Consortium* (*W3C*). Актуелна стандардизована верзија *HTML*-а је 5.1 представља еволуцију стандарда *HTML 4.01* који му претходи.

Веб-страница се састоји од елемената који су засебне компоненте *HTML* документа. Садржај елемента може бити неки други елемент или текст. *HTML* документ се састоји од ознака (енгл. *tag*) и атрибута. Када се врши парсирање странице, тада се обрађују ознаке и на основу ознака настају *HTML* елементи и генерише се веб-страница. На следећем примеру се може видети ознака *p* од које настаје елемент параграф као и атрибут *style* у ком је дефинисана ширина параграфа.

```
<p style="width:500px">Параграф</p>
```

HTML документ унутар ознака <html>...</html> мора садржати два основна елемента:

- Заглавље (енгл. *head*) - садржи информације које описују документ
- Тело (енгл. *body*) – садржај документа

Унутар заглавља се дефинише наслов документа унутар ознака <title>...</title>. Следећи пример представља минималну структуру *HTML* документа:

```
<!DOCTYPE HTML>
<html>
	<head>
	<title>Naslov</title>
	</head>
	<body>

	</body>
</html>
```

JavaScript

JavaScript је скриптни програмски језик користи се на клијентској страни за дефинисање функционалности веб-страница. Укључује се на веб-страници да бисмо је учинили динамичнијом. *JavaScript* је слабо типизиран и има скромну подршку за објектно оријентисано програмирање. *JavaScript* се најчешће одвија на страни клијента, односно у веб-прегледачу и врши интеракцију са објектним моделом документа. Поред интеракције са објектним моделом документа, *JavaScript* може вршити анимирање елемената на страници тако што елементима може мењати места и величину, може вршити убацивање видео и аудио садржаја, а такође може вршити валидацију унетог садржаја на форми како би серверу били послати исправни подаци.

AJAX

AJAX (енгл. *asynchronous JavaScript and XML*) [12] је група међусобно повезаних техника за развој веба коришћених на клијентској страни за прављење синхроних и асинхроних веб-апликација. Коришћењем *AJAX*-а, веб апликације могу да шаљу и примају податке са сервера асинхроно (у позадини) без мењања тренутног приказа и понашања странице тј. омогућена је асинхрона размена података између веб-прегледача и веб-сервера да би се избегло поновно учитавање целе странице.

jQuery

jQuery [8] је библиотека *JavaScript*-а, чија је сврха да поштујући принцип „пиши мање, уради више“ поједностави рад са *HTML* елементима, омогући управљање анимацијама и *AJAX* апликацијама на веб-страницама. *jQuery* захтева много мање линија кôда од основног *JavaScript*-а, тако што обједињује *JavaScript* кôд у методе које се могу позвати у једној линији кôда. *jQuery* библиотека има следеће карактеристике:

- Управљање *HTML*-ом
- Управљање *CSS*-ом
- *HTML* методе догађаја
- Анимације
- *AJAX*

CSS

CSS (Cascading Style Sheets) [9] је језик форматирања помоћу ког се дефинише изглед елемената веб-странице. На почетку је *HTML* служио да дефинише комплетан изглед, структуру и садржај веб-странице, али је од верзије 4.0 *HTML*-а уведен *CSS* да би дефинисао конкретан изглед, док је *HTML* остао у функцији дефинисања структуре и садржаја.

CSS синтакса се састоји од описа изгледа елемената у документу. Сваки опис се састоји од три елемената:

- Дефиниција циљних елемената
- Својства

- Вредности

```
.headerContent {
    border: none;
    text-align: center;
    position: absolute;
}
```

На пример, *headerContent* је циљни елемент, *text-align* је својство, док је *center* вредност.

Целокупан CSS неке странице можемо чувати у посебној датотеци са екстензијом *.css.*

2.2 Јава

Јава је програмски језик који је постао незаобилазан када су у питању програми који треба да се извршавају на различитим системима. Веома важна карактеристика Јаве је њена објектно оријентисаност. Објектно оријентисани програми су разумљивији и за њихово одржавање је потребно мање времена у односу на програме који су писани без коришћења објеката [6].

Јава технологије које се користе у веб-апликацијама

Поред *JSF* технологије која се користи за развој веб-апликација треба споменути још неколико Јава веб технологија које претходе *JSF*-у.

Сервлет (*Servlet*) класа проширује могућности сервера на коме се покреће апликација и коме се приступа путем програмског модела захтев-одговор (енгл. *request-response*). Иако сервлети могу одговорити на било који тип захтева, они се често користе за проширење веб-апликације која се хостује на веб-серверу. На пример, сервлети се могу користити да би се дохватила вредност са неке веб-форме и да би се та иста вредност вратила назад на *HTML* страницу, или се уместо тога може користити други сервлет да упише податке у базу. Сервлети раде на серверској страни сами за себе, без *HTML* корисничког интерфејса [3].

JavaServer Pages (JSP) технологија омогућава поједностављен, брз начин за креирање динамичног веб садржаја. Она омогућава брз развој апликација базираних на вебу које су независне од сервера и платформе. *JSP* технологија нам дозвољава да додајемо делове сервлетског кôда директно на текстуални документ. *JSP* страница је текстуални документ који садржи два типа текста:

- Статичке податке, који се могу изразити у било ком текстуалном формату као што је *HTML*, *Wireless Markup Language* (*WML*), или *XML*
- Елементе *JSP* технологије који одређују како страница конструише динамички садржај [3]

2.3 Maven

Maven је софтверски алат за управљање и изградњу пројекта. Помоћу овог алата можемо припремити кôд за дистрибуцију. Такође, уз помоћ *Maven*-а можемо на лак начин укључити *Java* библиотеке (*jar*-ове) који су нам потребни у пројекту. Сва правила дефинишемо у датотеци *pom.xml*, која мора садржати групу, предмет за употребу, верзију и назив пројекта. У истој

датотеци дефинишемо и начин паковања, да ли ће бити, на пример, *jar* (*Java Application Arhive*) или *war* (*Web Application Arhive*). Ово је веб-пројекат тако да ће начин паковања бити *war*.

У *dependencies* делу *pom.xml*-а се дефинишу библиотеке које су потребне пројекту. Без *Maven*-а се све библиотеке (*jar*-ови) морају поставити ручно. У *pom.xml*-у се на следећи начин дефинише укључивање *Primefaces* библиотеке:

```
<dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>6.0</version>
</dependency>
```

У *build* делу *pom.xml* се дефинише на који начин се врши изградња пројекта и паковање пројекта у *war* (*jar*), који се компајлер користи, која верзија *Jave*. Приликом паковања *war*-а подразумевано је да у *war* не улази директоријум *src/main/java* пошто нам *Java* датотеке нису потребне на серверу већ искомпајлиране *.class* датотеке. Пошто се датотеке за локализацију апликације налазе у овом директоријуму, потребно је експлицитно навести у *pom.xml*-у унутар ознаке *build* да *war* треба да садржи и ове датотеке. То се постиже следећом командом:

```
<resources>
      <resource>
            <directory>src/main/java/properties</directory>
      </resource>
</resources>
```

Постоји неколико фаза животног циклуса изградње пројекта [2]:

- *validate* (врши се валидација пројекта)
- *compile* (врши се компилација изворног кôда, компајлиран кôд се смешта у директоријум *target* у пројекту)
- *package* (узима се компајлиран кôд из *target* директоријума и пакује се у формат за дистрибуцију, на пример, *war*)
- *install* (направљен пакет се инсталира на локалном репозиторијуму да би пројекат могао да се користи локално у другим пројектима као *dependency*)
- *deploy* (копира се пакет на удаљени репозиторијум и на тај начин пакет постаје доступан осталим развојним инжењерима и пројектима)

Извршавањем команде у командном прозору

```
mvn clean package
```

позива се фаза *package* као и све њене фазе које јој претходе. У случају *package* то су *validate* и *compile*. Командом *clean* се празни *target* директоријум пре извршавања главне *Maven* команде, тј. пре поновне компилације кôда у случају команде *package*.

3. *JavaServer Faces (JSF)*

У овом поглављу биће описане основне карактеристике *JSF* радног оквира као и услуге које *JSF* нуди. Биће представљене библиотеке *JSF* елемената (компоненти) и биће описане карактеристике Јава зрна у *JSF*-у као и веза Јава зрна са *JSF* страницом. Биће описане и фазе животног циклуса обраде веб-захтева као и генерисање стабла компоненти приликом обраде *JSF* странице. Биће представљена основна разлика између редирекције и навигације. На крају ће бити представљен садржај *PrimeFaces* библиотеке *JSF* компоненти.

3.1 Зашто JSF?

JavaServer Faces (JSF) је радни оквир објектно оријентисаног језика Јава и користи се за развој веб-апликација. Он поједностављује развој веб-апликација обезбеђујући приступ заснован на компонентама за развој *Java* веб корисничког интерфејса [3]. Компонента је основни градивни блок за формирање *JSF* корисничког интерфејса (дугмић, поље за унос, лабела, ...) [14]. Комбинацијом компоненти на страници можемо направити сложену веб-страну. Коришћењем компоненти можемо размишљати о корисничком интерфејсу на вишем нивоу у односу на *HTML.*

JSF користи целокупан неопходан кôд за обраду догађаја и организацију компоненти који је део *JSF* имплементације [1]. *JSF* је спецификација, док *JSF* имплементација представља уграђен скуп компоненти и функција неопходних за обраду *JSF* страница. Развојни инжењери не морају размишљати о томе како су компоненте развијане, већ се могу посветити пословној логици апликације. На пример, уколико нам је потребна компонента календара, не морамо је развијати, већ је можемо искористити из неке *JSF* библиотеке.

3.2 *JSF* странице

Постоји неколико библиотека *JSF* елемената. Екстензија *JSF* страница је *.xhtml*. Свака *.xhtml* страница мора имати заглавље који може садржати неке (или све) од наведених библиотека.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:c="http://java.sun.com/jsf/core"
      xmlns:fn="http://java.sun.com/jsp/jstl/functions">
```

HTML

Ова библиотека има следећи именски простор *http://java.sun.com/jsf/html* чији је префикс *h:*. Састоји се од основних *HTML* елемената: поља за унос, поља за испис, дугмиће, *checkbox*, *combobox*, радио дугме, делове *HTML* страница (*head*, *body*...) , линкове, слике, табеле, итд.

JSF CORE

Ова библиотека има следећи именски простор *http://java.sun.com/jsf/core* чији је префикс *f:*. Састоји се од елемената који су независни од *HTML* приказа. Ово су неки од основних *JSF CORE* елемената: *param*, *facet*, *valueChangeListener*, *actionListener*, *ajax*, *view*, *converter*, итд.

JSTL Функције

Ова библиотека има следећи именски простор http://java.sun.com/jsp/jstl/functions чији је префикс *fn:*. Састоји се од елемената који су функције за рад са нискама: *length*, *contains*, *startsWith*, *endsWith*, *split*, *substring*, *toLowerCase*, *toUpperCase*, итд.

JSTL CORE

Ова библиотека има следећи именски простор http://java.sun.com/jsf/core чији је префикс *c:*. Најкоришћенији елементи из ове библиотеке су *if* и *forEach*. У следећем примеру користимо петљу која исписује 20 лабела од којих само прва нема вредност, док остале имају вредност *resources['labela']*.

```
<c:forEach var="i" begin="1" end="20" step="1" varStatus ="status">
	<c:if test="#{i == 1}">
		<p:outputLabel value="" />
	</c:if>
	<c:if test="#{i != 1}">
		<p:outputLabel value="#{resources['labela']}" />
	</c:if>
</c:forEach>
```

Фејслети *(Facelets)*

Фејслети [13] су систем за организацију веб-странице и основна технологија за руковођење *JSF* погледом. Уз помоћ фејслета можемо креирати властите компоненте, можемо креирати странице које су шаблони, на лак начин странице могу постати део друге странице, итд. Више о фејслетима у практичном делу рада, где ће се фејслети објаснити кроз примере.

3.3 Компоненте корисничког интерфејса

Свака компонента на страници је повезана са објекатом класе корисничког интерфејса. *JSF* имплементације обезбеђује скуп класа компоненти корисничког интерфејса у којима је описана функционалност компоненти. У класама се чувају стања компоненти и референце ка другим објектима као и управљање догађајима. Главна апстрактна класа за све компоненте је *javax.faces.component.UIComponent*. *JSF* компоненте наслеђују класу *UIComponentBase* која је потклаcа класе *UIComponent*. У њој је дефинисано основно стање и понашање компоненте. У следећој листи су описане неке од класа компоненти које су укључене у *JSF* имплементацији:

- *UIInput* – Узима податке које је корисник унео
- *UIOutput* – Приказује податке
- *UIForm* – Форма која садржи поља за унос
- *UICommand* – Позива акцију када се изврши неки догађај
- *UIPanel* – Управља распоредом компоненти које се налазе у панелу
- *UIGraphic* – Приказује слику

3.4 Зрна

Јава зрно (енгл. *bean*) је класа која излаже својства и догађаје у радном оквиру. **Својство** (*property*) је именована вредност задатог типа која се може читати и (или) у коју се може уписивати. Својство је било који атрибут зрна који има:

- Име
- Тип
- Методе за читање (*get*) и/или писање (*set*)

Најпростији начин за дефинисање својства је коришћење стандардне конвенције именовања метода читања и писања. Прво слово имена својства се мења у велико слово и додаје на *get* ili *set*.

```
private Integer ajaxSpeed;

public Integer getAjaxSpeed() {
	return ajaxSpeed;
}

public void setAjaxSpeed(Integer ajaxSpeed) {
	if (xhtmlPage.equals("statistic")) {
		ajaxSpeed = Integer.valueOf(0);
		return;
	}
	this.ajaxSpeed = ajaxSpeed;
	addAjaxTime();
}
```

Методе читања и писања могу обављати произвољне радње. Најчешће се користи за читање или постављање вредности неког поља, али такође се у њима могу извршавати нека израчунавања или се може приступати бази података. У претходном примеру се може видети да метода читања само прослеђује вредност својства, док метода писања поред постављања вредности извршава и додатне акције. Називи метода морају почињати са *get* (осим у случају својства са *boolean* типом када може почињати и са *is*) која је без аргумената и *set* која има један аргумент и нема повратну вредност (*void*). Класа зрна може садржати и друге методе поред метода за читање и писање.

У *JSF*-у, зрна чувају стање веб-странице. Прављење зрна и управљање њима је под контролом *JSF* имплементације. Зрна су веза између корисничког интерфејса и пословне логике апликације. *JSF* имплементација ради следеће:

- Ствара и брише зрна по потреби (термин „*managed beans*“)
- Чита својства зрна приликом приказа веб-странице
- Поставља вредност својствима зрна када је форма постављена

На пример, поље за испис чита својство зрна:

```
<p:outputLabel id="ajaxSpeed" value="#{pageSpeedBean.ajaxSpeed} ms" />
```

JSF имплементација је треба да лоцира класу зрна са именом *pageSpeedBean*.

```
@SessionScoped
@ManagedBean(name="pageSpeedBean")
public class PageSpeedBean implements Serializable
```

Можемо изоставити *name* атрибут у *@ManagedBean* анотацији. У том случају назив зрна је изведен од назива класе тако што се прво слово промени у мало слово. Уколико је име класе *PageSpeedBean*, а притом је изостављен атрибут *name*, назив зрна постаје *pageSpeedBean.*

Када *JSF* имплементација обрађује страницу и када први пут прочита израз *#{pageSpeedBean.ajaxSpeed}*, креира објекат класе *PageSpeedBean*. *JSF* развојни инжењер не мора да пише *Java* кôд да би конструисао зрно *pageSpeedBean*. *JSF* имплементација конструише зрно и приступа њему кроз израз у *JSF* страници.

Пре *JSF 2.0* је било потребно да се сва зрна дефинишу у датотеци *faces-config.xml* који је један од конфигурационих датотека апликације. Сада можемо да бирамо да ли ћемо користити анотације или *XML* конфигурацију.

Израз вредности (*value expression*)

Израз вредности дефинише везу вредности компоненте и вредности својства зрна. Ако у *.xhtml* страници имамо израз *#{pageSpeedBean.speed}* и уколико *JSF* имплементација приказује страницу, позваће се метода *getSpeed*() над објектом *pageSpeedBean*. Израз вредности се може користити и за читање и писање, на пример, у компоненти за унос:

```
<p:inputText value="#{pageSpeedBean.speed}" />
```

Када *JSF* имплементација приказује компоненту позваће се метода читања (*get*). Када корисник изврши *submit* форме, позваће се метода писања (*set*) и вредност која је унета у компоненти ће се уписати у својство зрна.

Као у *JavaScrpt*-у можемо користити заграде уместо нотације са тачком. Тако да су следећи изрази еквивалентни.

- *pageSpeedBean.speed*
- *pageSpeedBean['speed']*
- *pageSpeedBean[''speed'']*

Израз за позив методе

Израз за позив методе означава оначава акцију (методу зрна) која се извршава када се изврши неки догађај на форми. На пример, коришћењем следећег дугмета:

```
<p:commandButton value="#{resources['izvrsiajaxakciju']}"
      action="#{messageBean.saveMessage}" update="ajaxSpeed" />
```

извршава се метода *saveMessage* класе *messageBean* када се кликне на дугме. Израз за позив методе је подоган начин да се опише позив методе која треба да се изврши у неком будућем времену. Постоје следећи атрибути за изразе, односно методе зрна које се позивају као и њени параметри:

Р.Бр	Атрибут на *JSF* страници	Метода зрна која се позива
1	`action="#{bean.metoda}"`	`String metoda()`
2	`actionListener="#{bean.metoda}"`	`void metoda(ActionEvent)`
3	`valueChangeListener="#{bean.metoda}"`	`void metoda(ValueChangeEvent)`
4	`validator="#{bean.metoda}"`	`void metoda(FacesContext, UIComponent, Object)`
5	`listener="#{bean.metoda}"`	`void metoda(ComponentSystemEvent)`
6	`listener="#{bean.metoda}"`	`void metoda(AjaxBehaviorEvent)`

Атрибути 1 и 2 се односе на дугмиће и линкове. Атрибут *action* позива методу чија је повратна вредност ниска која се прослеђује руководиоцу навигације. Овај атрибут се најчешће користи да се изврши навигација на другу страницу. *ActionListener* извршава методу зрна и његов циљ је извршавање логике корисничког интерфејса и не учествује у руковођењу навигацијом, а извршава се пре *action*. *ActionListener* понекад ради у сарадњи са *action* атрибутом када су *action*-у потребне информације у вези корисничког интерфејса.

Атрибути 3 и 4 се односе на компоненте за унос (*input*). *ValueChangeListener* извршава методу зрна у случају да се променила вредност поља за унос (*inputText*). *Validator* извршава методу која проверава да ли унета вредност испуњава унапред задате услове. На пример, ако је обавезан унос неког поља, валидатор проверава да ли је попуњена вредност поља, у супротном се исписује грешка на корисничком интерфејсу.

Атрибут 5 се односи на системске догађаје. Синтакса је следећа:

```
<f:event type="postValidate" listener="#{bean.method}" />
```

JSF нема механизам да изврши валидацију групе компонени. На пример, ако на форми имамо дан, месец и годину, нема природног начина да се изврши валидација унесеног датума. У том случају се може користити системски позив *postValidate* који је тип догађаја и извршити потребна валидација у методи зрна која је дефинисана у атрибуту *listener*. Од системских позива поред *postValidate* постоје још и позиви приликом стартовања или гашења апликације, после додавања или пре уклањања компоненте из стабла компоненти, после обнављања стања компоненте, пре валидације компоненте, пре приказа корена погледа, пре приказа компоненте, итд [1].

Атрибут 6 извршава *AJAX* позиве.

Уколико желимо да *actionListener*-у проследимо додатне параметре поред основног параметра *ActionEvent*, то није могуће. Можемо на следећи начин да предефинишемо параметре:

```
<p:commandButton actionListener="#{bean.method()}" />
<p:commandButton actionListener="#{bean.method(argument1)}" />
<p:commandButton actionListener="#{bean.method(argument1, argument2)}" />
```

Следеће методе се односе на *actionListener* из претходних примера. У овом случају се *ActionEvent* не преноси јер су наведене заграде и параметри у изразу за позив методе.

```
	public void method() {}
	public void method(Object argument1) {}
	public void method(Object argument1, Object argument2) {}
```

Зрна подршке (енгл. *backing beans*)

Понекад је потребно да направимо такво зрно који ће садржати неке или све објекте компоненти са веб-форме. Таква зрна се називају **зрна подршке** веб-форме [1].

```
@ViewScoped
@ManagedBean(name="outputBean")
public class OutputBean {

	private UIOutput outputComponent;

	public UIOutput getOutputComponent() {
		return outputComponent;
	}

	public void setOutputComponent(UIOutput outputComponent) {
		this.outputComponent = outputComponent;
	}
}
```

Компоненте за унос припадају *UIInput* класи, док компоненте за приказ припадају *UIOutput* класи. Нека визуелна *JSF* развојна окружења користе зрна подршке. Та окружења аутоматски генеришу својства и методе читања и писања за све компоненте које се довуку на форму.

Када се користе зрна подршке потребно је да се повеже компонента са форме са компонентом у зрну подршке преко *binding* атрибута на *.xhtml* форми.

```
	<p:outputLabel binding="#{outputBean.outputComponent}" />
```

Када се гради стабло компоненти, позива се метода зрна подршке *getOutputComponent*. Као резултат, компонента приказа је конструисана и инсталирана у зрну подршке позивом методе *setOutputComponent*.

Опсег зрна (*bean scope*)

JSF контејнер омогућава неколико опсега који чувају зрна и остале објекте које треба да буду доступни различитим компонентама веб-апликације. Када дефинишемо зрно, неопходно је дефинисати и његов опсег. Ово су опсези *JSF* зрна [1]:

- Опсег сесије (`@SessionScoped`)
- Опсег захтева (`@RequestScoped`)
- Опсег апликације (`@ApplicationScoped`)
- Опсег погледа (`@ViewScoped`)

Опсег сесије

Постоји стална комуникација између веб-прегледача и сервера кроз *HTTP* протокол. Пошто је *HTTP* протокол без стања, ни на серверу ни на клијенту се не чува стање трансакције. Овај начин рада је добар код најпростијих захтева, али уколико је сервер у обавези да зна од ког клијента му је стигао захтев, то није довољно. Уколико се ради о апликацији у којој је обавезно логовање корисника, потребно је негде сачувати корисничко име улогованог корисника.

Сервлет контејнер захтева од HTTP протокола да прати сесију. Сервлет контејнер је контејнер за сервлете, он је део веб-сервера и његова основна функција је интеракција са сервлетима тако што користи Јаву да би динамички генерисао веб-страницу на серверу. Најпростији метод за праћење сесије је коришћење колачића. Колачићи су пар име/вредност који сервер шаље клијенту. У овом случају назив колачића је *JSESSIONID.* Приликом сваког захтева клијента, шаље се и вредност колачића *JSESSIONID* и на тај начин сервер тачно зна који клијент му је послао захтев [1].

Уколико је клијент искључио подршку колачића у веб-прегледачу, податак о сесији се прослеђује као параметар *URL*-а. Праћење сесије је потпуно транспарентно за развојне инжењере. Морамо имати на уму да превелики број зрна са опсегом сесије може негативно утицати на перформансе апликације [1].

Када *JSF* имплементација креира објекат зрна, он остаје жив све док траје сесија, с обзиром да је зрно у опсегу сесије. Ово важи за све захтеве са једног клијента све док сесија не истекне или се експлицитно не прекине. Током сесије зрно се референцира на претходно конструисан објекат који се налази у сесији. Различите сесије припадају различитим клијентима који су активни на серверу. Сваки од њих поседује своју инстанцу објекта класе зрна подршке.

Опсег захтева

Опсег захтева је кратковечан. Он почиње од када се страница поднесе серверу и завршава се када се одговор пошаље назад клијенту. Ако је наше зрно у опсегу захтева, нова инстанца се креира приликом сваког захтева [1].

Опсег захтева је пожељан ако се сви подаци из зрна такође чувају на страници. Уколико нам је потребан неки податак из зрна на неколико страница, тада нам опсег захтева није довољан. Приликом приказивања сложених података, као што је садржај табеле, тада такође опсег захтева није пожељан зато што је потребно поново генерисати податке приликом сваког захтева [1].

Опсег апликације

Опсег апликације траје све док је веб-апликација инсталирана на серверу. Опсег се дели између свих захтева и сесија. Зрно ће бити у опсегу апликације ако се он дели између свих инстанци веб-апликације. Зрно се креира приликом првог захтева било ког корисника апликације и остаје у животу све док се веб-апликација не уклони са сервера [1].

Опсег погледа

Зрно у опсегу погледа траје све док се страница не прикаже поново, тј. све док корисник не напусти *JSF* поглед у прозору или картици веб-прегледача. Чим се корисник преусмери на другу

страницу, зрно излази из овог опсега. Овај опсег је пожељан код *Ajax* апликација, када се подаци на страници мењају кроз *Ajax* позиве [1].

Дакле, ако посматрамо трајање опсега зрна, најкраћи је опсег захтева. Након њега следе опсег погледа, опсег сесије и опсег апликације.

Анотација животног циклуса зрна

Користећи `@PostConstruct` и `@PreDestroy` анотацију, можемо специфицирати методе зрна које ће се аутоматски позвати приликом конструкције зрна и код изласка зрна из опсега.

```
@PostConstruct
public void init() {
	FacesContext context = FacesContext.getCurrentInstance();
	resources = context.getApplication().getResourceBundle(
		FacesContext.getCurrentInstance(), "resources");
	createLineModel();
}
```

Веома је корисно да метода има `@PostConstruct` анотацију ако се у њој генеришу или повлаче подаци из базе података који треба да се иницијално прикажу на страници.

3.5 Услуге радног оквира *JSF*

JSF је одговоран за интеракцију са клијентским уређајима и пружа алате за повезивање презентационог слоја, логике апликације и пословне логике веб-апликације. Опсег *JSF*-а је ограничен на презентационом (поглед) слоју. Приступ бази података, веб-сервиси и остала позадинска (енгл. *backend*) логика су изван оквира *JSF*-а.

Ово су најбитније услуге које нам радни оквир *JSF* нуди [1]:

- **Модел-поглед-контролер** (*model-view-controller*) [13] архитектура – Све софтверске апликације дозвољавају корисницима да управљају подацима. Ти подаци су **модел** који су део позадинске логике. У презентационом слоју (такозваном **погледу**) веб-апликација кориснику се подаци представљају кроз HTML странице. *JSF* повезује ова два слоја тако што компоненту погледа повезује са својством зрна који је објекат модела.

  ```
  <p:chart id="chart" type="line" style="height:600px;"
  model="#{statisticAjaxBean.ajaxSpeedLinearModel}" />
  ```

  JSF имплементација ради као **контролер** који реагује на корисничке акције које мењају модел или поглед. На пример, корисник врши унос новог податка на форми (поглед), *JSF* који је контролер прослеђује серверу податке које је корисник унео, сервер обрађује податке и мења модел (врши унос у базу података) и шаље контролеру обавештење да је податак успешно унет. Контролер мења поглед (освежава *HTML* страницу) на основу одговора од стране сервера.
- **Конверзија података** – Веб-апликација чува различите типове података, али се на веб кориничком интерфејсу приказују искључиво као ниске. На пример, корисник уноси

податке на веб-форми као текст, док објекат пословне логике очекује податак као број или датум. *JSF* нам омогућава да на лак начин дефинишемо правила за конверзију.

- **Валидација и обрада грешака** – У *JSF*-у веома лако дефинишемо правила за валидацију, на пример, да ли је обавезан унос неког поља, да ли унета вредност мора бити број. Уколико је корисник унео погрешне податке, потребно је исписати поруку о грешци.
- **Прављење компоненти** – Софтверски инжењери који се баве развојем компоненти могу направити специфичне компоненте које се на лак начин могу уклопити у програмску логику и на тај начин смањити број линија кôда *.xhtml* странице. На пример, компонента може бити комбинација више компоненти или може бити упрошћена компонента неке од стандардних компоненти из *JSF* библиотеке.
- ***Ajax* подршка** – *JSF* обезбеђује стандардан *Ajax* комуникациони канал за транспарентно извршавање серверских акција и ажурурања клијентских компоненти.

3.6 Стабло компоненти (*component tree*)

JSF имплементација иницијализује *JSF* кôд и чита *.xhtml* страницу. Страница се састоји од елемената чије су ознаке (енгл. *tag*) *h:form* и *h:inputText*. Сваки елемент је повезан са класом компоненти која њиме руководи. Када веб-сервер обрађује страницу, извршавају се методе класа компоненти, оне сарађују међусобно и граде **стабло компоненти** [1]. Стабло компоненти је структура података која садржи Јава објекте за све елементе корисничког интерфејса *JSF* странице. Компонента *p:layoutUnit* садржи у себи четири лабеле (*p:outputLabel*) што се може видети у следећем фрагменту *JSF* кôда.

```
<h:form id="idForm">

	<p:layoutUnit position="center" styleClass="headerContent" size="200">
		<p:outputLabel value="#{resources['matematickiFakultet']}" />
		<br /><br />
		<p:outputLabel value="#{resources['imePrezime']}" />
		<br /><br />
		<p:outputLabel value="#{resources['tema']}" style="font-size:30px" />
		<br /><br/>
		<p:outputLabel value="#{resources['masterRad']}" />
	</p:layoutUnit>

</h:form>
```

На слици 1 је приказан део стабла компоненти који се генерише на основу претходног фрагмента кôда.

```
<HtmlForm enctype="application/x-www-form-urlencoded" id="j_idt8" inView="true" prependId="true"
rendered="true" submitted="true" transient="false">
```

```
<LayoutUnit closable="false" collapseSize="25" collapsed="false" collapsible="false" gutter="6"
id="j_idt14" inView="true" maxSize="0" minSize="50" position="center" rendered="true"
resizable="false" size="200" styleClass="headerContent" transient="false" visible="true">
    <OutputLabel escape="true" id="j_idt15" inView="true" indicateRequired="true"
    rendered="true" transient="false" value="Univerzitet u Beogradu - Matemati?ki fakultet"/>
    <br/><br/>
    <OutputLabel escape="true" id="j_idt17" inView="true" indicateRequired="true"
    rendered="true" transient="false" value="Stefan Tati? 1020/2011"/>
    <br/><br/>
    <OutputLabel escape="true" id="j_idt19" inView="true" indicateRequired="true"
    rendered="true" style="font-size:30px" transient="false" value="Analiza performansi JSF
    Veb aplikacija"/>
    <br/><br/>
    <OutputLabel escape="true" id="j_idt21" inView="true" indicateRequired="true"
    rendered="true" transient="false" value="Master rad"/>
</LayoutUnit>
```

```
</HtmlForm>
```

Слика 1 – Стабло компоненти

Након иницијализације стабла компоненти, страница се приказује. Сав текст који није *JSF* ознака се такође приказује, а све *JSF* ознаке се конвертују у *HTML*. Свака компонента има свог приказивача који генерише *HTML*, одржавајући стање компоненте. На пример, за претходни фрагмент кôда ком одговара стабло компоненти са слике 1, веб-прегледач генерише следећи *HTML*:

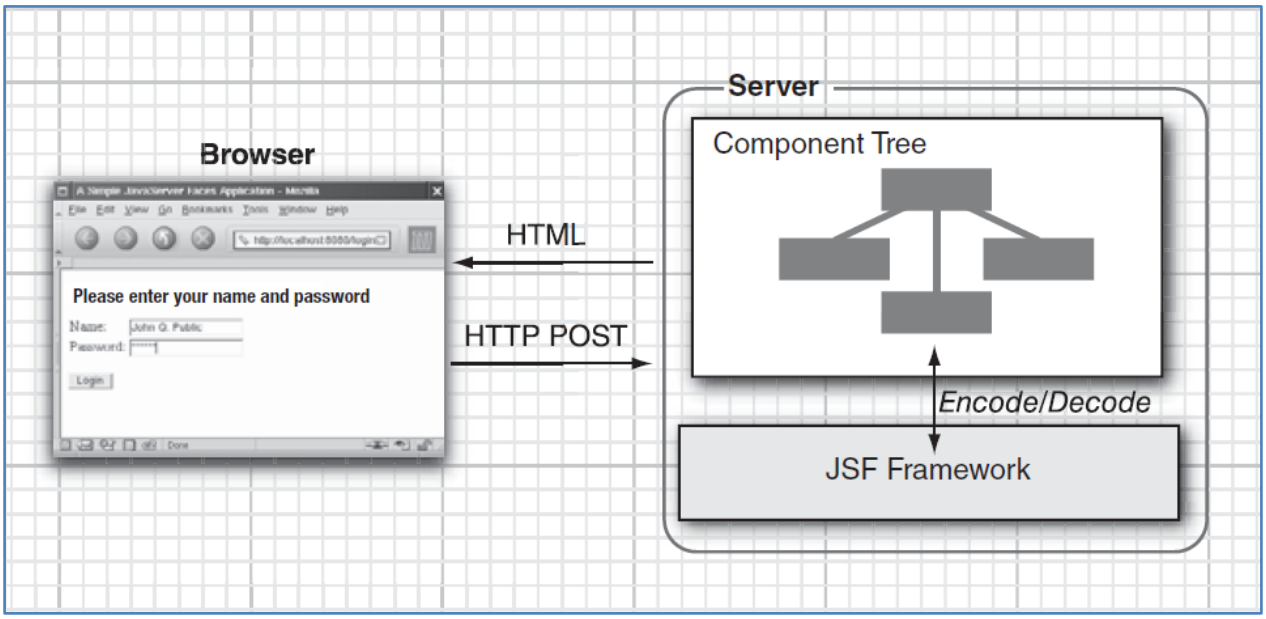
```
<form id="idForm" name="idForm" method="post"
action="/projekat/view/labele/labele1000.xhtml" enctype="application/x-www-
form-urlencoded">

      <div class="ui-layout-unit-content ui-widget-content">
        <label id="idForm:matfLabel" class="ui-outputlabel ui-widget">
                    Univerzitet u Beogradu - Matematički fakultet
                </label>
        <br /><br />
        <label id="idForm:j_idt17" class="ui-outputlabel ui-widget">
                Stefan Tatić 1020/2011
        </label>
        <br /><br />
        <label id="idForm:j_idt19" class="ui-outputlabel ui-widget"
                                              style="font-size:30px">
                Analiza performansi JSF Veb aplikacija
        </label>
        <br /><br />
        <label id="idForm:j_idt21" class="ui-outputlabel ui-widget">
                Master rad
        </label>
      </div>

</form>
```

Овај процес се назива **кодирање**. Приказивач *UIOutput* објекта тражи од *JSF*-а да потражи компоненту по јединственом идентификатору (атрибуту *ID*) и њену вредност. Подразумевано је да су идентификатори ниске одређене од стране *JSF* имплементације, тако што им додељује вредности које могу бити, на пример, `idForm:j_idt17` што се може видети на претходном примеру. Ако компонента има постављен идентификатор, онда *JSF* имплементација неће доделити вредност. На генерисаном *HTML*-у, идентификатор ознаке је пар *idForme:idKomponente*. Ако форма нема постављен идентификатор, *JSF* имплементације ће доделити вредност као и у случају компоненти [1].

Кодирана страница се прослеђује веб-прегледачу коју он приказује на уобичајен начин.

Када након приказа странце у веб-прегледачу, корисник попуни поље на форми и кликне на *submit* дугме тада веб-прегледач враћа податке из форме веб-серверу формирајући *POST* захтев (енгл. *request*). То је специјалан формат, дефинисан као део *HTTP* протокола. *POST* захтев садржи *URL* форме као и њене податке. Као део обраде захтева, подаци са форме са налазе у Хеш табели која је реализована на серверу, тако да јој све компоненте могу приступити [1].

У следећем кораку, *JSF* даје свакој компоненти прилику да приступи Хеш табели, а тај процес се назива **декодирање**. Свака компонента за себе одлучује како ће интерпретирати податке са форме [1]. На слици 2 [1] је представљено кодирање и декодирање *JSF* странице.



Слика 2 – Кодирање и декодирање *JSF* странице

Форма *UIForm* се може састојати од више *UIInput* објеката који одговарају текстуалним пољима на форми и *UICommand* објекат који одговара дугмету *submit* [1].

- *UIInput* компоненте мењају својства зрна референцирајући се *value* атрибутом текстуалног поља. Позива се метода писања (*setter*) са вредношћу коју је корисник унео у текстуалном пољу.
- *UICommand* компонента реагује када се кликне на дугме, тада се окида такозвани акциони догађај (*action event*) који позива методу атрибута *action* на дугмету.

3.7 Животни циклус обраде захтева

JSF спецификација дефинише шест различитих фаза животног циклуса обраде захтева [1]:

- Обнављање погледа (енгл. *Restore view*)
- Примена вредности захтева (енгл. *Apply request values*)
- Валидација процеса (енгл. *Process validations*)
- Освежавање вредности модела (енгл. *Update model values*)
- Позив апликације (енгл. *Invoke application*)
- Пружање одговора (енгл. *Render response*)

Фаза **обнављање погледа** преузима стабло компоненти са странице ако је она већ приказана или конструише ново стабло компоненти ако се страница приказује први пут.

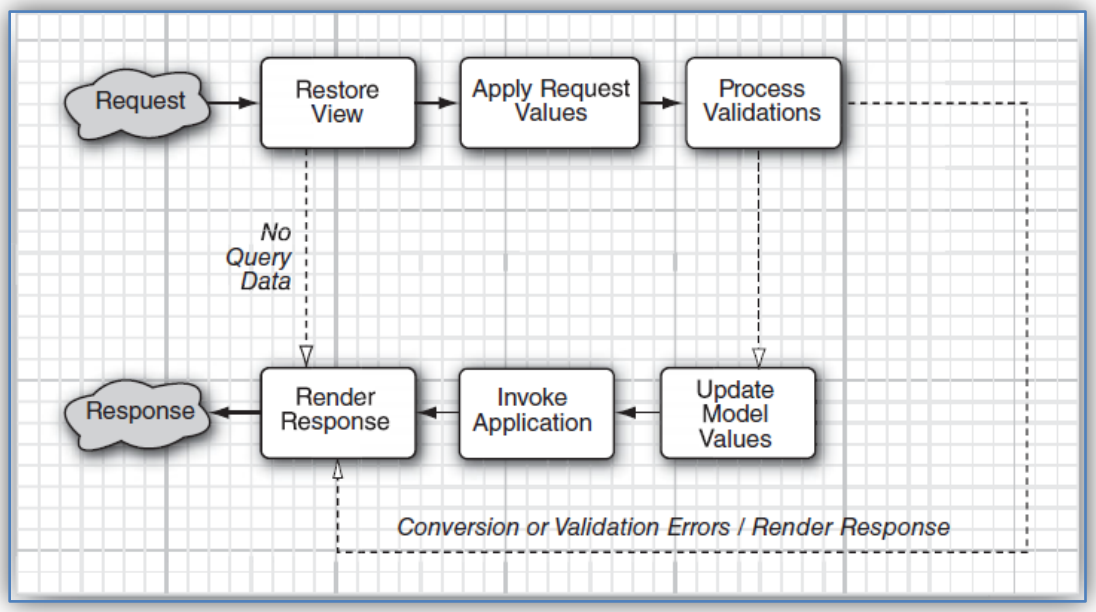
Ако захтев не садржи улазне податке *JSF* имплементација прелази на фазу **пружање одговора**. Ово се дешава када се страница први пут приказује. Иначе, следећа фаза је **примена вредности захтева** у којој JSF имплементација итерира кроз објекте компоненти у стаблу компоненти. Вредности сачуване у компонентама се називају локалне вредности. Сваки објекат проверава која му локална вредност из захтева припада и она му се додељује.

Када се дизајнирају *JSF* странице, за компоненте се могу везати валидатори који врше провере исправности локалних вредности. Валидатори се извршавају у фази **валидација процеса**. Ако се прође валидација, *JSF* циклус се наставља даље. Уколико има грешака приликом валидације, *JSF* имплементација позива директно фазу **пружање одговора**, страница се поново исцртава и даје се кориснику прилика да изврши исправан унос.

Када се валидација заврши, претпоставља се да је безбедно да се промени модел података и то се извршава у фази **освежавање вредности модела**. Локалне вредности се користе да би се промениле вредности својстава зрна које су повезане са компонентом.

У фази **позив апликације**, извршавају се акције дугмета или линка. Метода на акцији дугмета може обављати произвољну акцију. Генерише се излазна ниска која се шаље навигацији, која прелази на следећу страницу.

Фаза **пружање одговора** кодира одговор од стране сервера и шаље га веб-прегледачу. Када корисник кликне на дугмић *submit* или кликне на линк, генерише се нови захтев и започиње нови животни циклус.



Слика 3 – Фазе животног циклуса

3.8 Навигација и редирекција

Корисник на форми попуњава разна поља и све измене форме се дешавају у веб-прегледачу. Када корисник кликне на дугмић тада се подаци прослеђују серверу. Веб-апликација анализира кориснички унос и мора да одлучи која се *JSF* страница користи за приказ одговора од стране сервера. **Руководилац навигације** је одговоран за одабир следеће *JSF* странице [1].

Најпростија навигација је статичка и може се видети на следећем примеру:

```
<p:commandButton value="#{resources['navigacija']}"
                 icon="ui-icon-check" action="dobrodosli" />
```

Кликом на дугмић који се налази на страници *index.xhtml* извршава се акција *dobrodosli* којом се прелази на страницу *dobrodosli.xhtml*. У овом случају, у *action*-у није наведена екстензија, па ће екстензија бити иста као на тренутном погледу (*index.xhtml*). Уколико акција не почиње карактером /, подразумевана путања је путања тренутног погледа.

Поред статичке навигације постоји и динамична навигација. Проток странице не зависи само од дугмета на који се кликне већ зависи и од уноса на форми. На следећем примеру се може видети динамична навигација:

```
<p:commandButton value="#{resources['navigacija']}"
      icon="ui-icon-check" action="#{bean.navigacija}" />
```

Акција кликом на дугмић је позив методе *navigacija* која има следећи садржај:

```
public String navigacija() {
      if (uslov) {
```

```
            return "stranica1";
        } else {
            return "stranica2";
        }
    }
```

У зависности од услова који може бити неки податак са форме, прелази се на страницу *stranica1.xhtml* или на страницу *stranica2.xhtml*.

Од *JSF* имплементације се може затражити да изврши **редирекција** на другу страницу тј. поглед. Тада *JSF* имплеметација шаље *HTTP* редирекцију клијенту. Редирекција говори клијенту који *URL* да користи за следећу страницу. Тада клијент креира *GET* захтев ка том *URL*-у [1].

Редирекција је спора зато што постоји још један повратак ка веб-прегледачу, али редирекција даје веб-прегледачу шансу да промени његово поље за адресу. Без редирекције, оригинални *URL* (http://localhost:8080/projekat/index.xhtml) остаје непромењен када корисник пређе са странице *index.xhtml* на страницу *dobrodosli.xhtml*. Са редирекцијом веб-прегледач учитава нови URL (http://localhost:8080/projekat/dobrodosli.xhtml). Да би се уместо навигације извршила редирекција потребно је акцији додати суфикс *?faces-redirect=true* што се може видети на следећем примеру.

```
<p:commandButton value="#{resources['navigacija']}" icon="ui-icon-check"
    action="dobrodosli?faces-redirect=true" />
```

За разлику од *commandButton*-а који креира *POST* захтев, компонента *button* креира *GET* зехтев тако да су параметри видљиви у *URL*-у.

```
<p:button outcome="dobrodosli" icon="ui-icon-star"
    value="#{resources['redirekcijaSaGETParametrom']}" >
        <f:param name="parametar" value="vrednostParametra" />
</p:button>
```

Кликом на овај дугмић се прелази на следећу страницу која има *URL* параметар (http://localhost:8080/projekat/view/dobrodosli.xhtml?parametar=vrednostParametra).

3.9 PrimeFaces

PrimeFaces је популаран радни оквир отвореног кôда за развој *JSF* апликација који садржи више од 100 компоненти и који нам поред тога нуди и валидацију на страни клијента [4].

PrimeFaces је библиотека *JSF* компоненти која се налази у једној *jar* датотеци која се коришћењем *Maven*-а на једноставан начин укључује у пројекат [5].

У једом од претходних поглавља је објашњена структура заглавља *.xhtml* странице. Укључивање *PrimeFaces* библиотеке на *.xhtml* страници се дефинише на исти начин као и остале *JSF* библиотеке.

```
xmlns:p="http://primefaces.org/ui"
```

PrimeFaces обезбеђује парцијални приказ странице и функцију за обраду погледа која је базирана на *JSF* имплементацији која нуди избор шта урадити у *JSF* животном циклусу и шта треба приказати

на страници по завршетку *AJAX* позива. На серверској страни радни оквир *PrimeFaces AJAX* је заснован на *JSF* имплементацији. На клијентској страни скриптови су засновани на *jQuery JavaScript* библиотеци [5].

PrimeFaces библиотека се састоји од следећих група компоненти:

- Компоненте за извршавање *AJAX* позива
- Компоненте за унос података (поље за унос текста, *combobox*, радио дугме, поље за унос лозинке, тастатура, компонента за унос потписа, текстуални едитор, календар, компонента за избор боје, *slider*, *checkbox*, итд.)
- Дугмићи и линкови (*commandButton*, *commandLink*, *button*, *link*). Ове компоненте су објашњене раније када је било речи о навигацији и редирекцији.
- Компоненте за приказ података (мрежа података, табела података, листа података, распоред, вертикално и хоризонтално стабло, компонента за извоз података у *pdf*, *csv*, *xml,* временска лента, *Google* мапа, дијаграм, итд.)
- Компоненте панела (*css* мрежа, табла за нотификације, приказ у табовима, итд.)
- Прекривајуће компоненте (прозор за потврду са дугмићима да и не, *tooltip*, *lightBox* који је неки вид *Iframe*-а, прекривајући панел, итд.)
- Компоненте менија
- Дијаграми (питица, бар, линија, итд.)
- Компоненте за испис порука
- Мултимедијалне компоненте (баркод, галерија, слика, компонента за учитавање *Youtube* и *QuickTime* видео садржаја, компонента за учитавање *pdf* документа, итд.)
- Компоненте за преузимање и отпремање датотека
- *Drag & Drop* компоненте
- Валидација форми на страни клијента (на пример, провера да ли је унета вредност у пољу коме је обавезан унос, да ли је унета бројчана вредност, провера да ли унос задовољава регуларни израз итд.)
- Радни оквир за рад са дијалозима
- Остале компоненте (*captcha*, сат, компонента која позива штампач, *hotkey* подршка, *Rss feed* читач, руководилац изузетака, итд.)

PrimeFaces Extensions је библиотека отвореног кôда за *JSF.* Пројекат ради над *PrimeFaces* пројектом. Ова библиотека се састоји од проширених компоненти које не постоје у таквом облику у другим *JSF* библиотекама или од побољшаних компоненти које већ постоје у некој другој библиотеци у којој не раде на најфинији начин [7]. На пример, претходне верзије *PrimeFaces*-а нису садржале компоненту *inputNumber* која је постојала у *PrimeFaces Extensions* библиотеци. Практично *inputNumber* компонента је радила над *inputText* компонентом *PrimeFaces* библиотеке у којој је могуће уносити само бројеве и којој се могу дефинисати минимална, максимална вредност, децимални и групни сепаратор. Што се тиче компоненте *remoteCommand* (о којој ће више бити речи у практичном делу), она постоји у једној и другој библиотеци, с тим што је лакше проследити параметре у компоненти из *PrimeFaces Extensions* библиотеке.

Укључивање *PrimeFaces Extensions* библиотеке на *.xhtml* страници се дефинише на исти начин као и остале *JSF* библиотеке.

```
xmlns:pe="http://primefaces.org/ui/extensions"
```

Велики број *PrimeFaces* компоненти има атрибут *icon* у коме се дефинише иконица која се налази са леве стране лабеле компоненте. На пример, иконица *fa fa-home* је једна од многих иконица из скупа иконица *FontAwesome* [10]. Да би се омогућило коришћење ових иконица у пројекту, потребно је укључити ову библиотеку у датотеци *web.xml* следећим кôдом:

```
<context-param>
      <param-name>primefaces.FONT_AWESOME</param-name>
      <param-value>true</param-value>
</context-param>
```

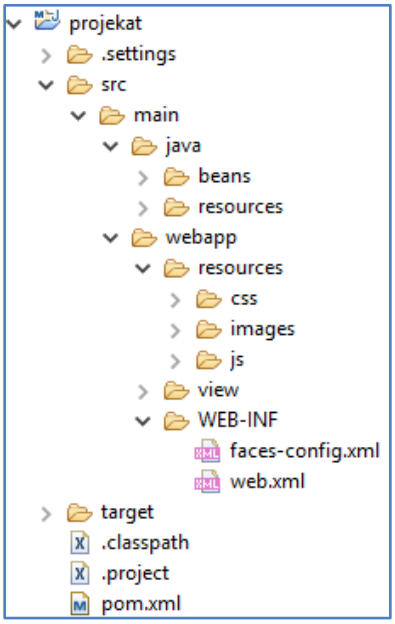
Потребно је навести да поред *PrimeFaces* библиотеке постоје још и следеће *JSF* библиотеке: *RichFaces*, *OmniFaces*, *IceFaces* које неће бити описане у овом раду.

4. Пројекат имплементације веб-апликације за тестирање перформанси *JSF* страница

У овом поглављу биће описан веб-пројекат чији је један од циљева представљање основних фукнција *JSF* радног оквира. Биће представљено коришћење шаблона за организацију *JSF* страница, локализација, неке од компоненти *PrimeFaces* библиотеке. Главни циљ пројекта је анализа перформанси страница, тј. рачунање трајања учитавања странице и трајања *AJAX* позива у односу на број компоненти на страници. Приликом сваког учитавања странице и *AJAX* позива, времена ће се прослеђивати зрну које чува сва трајања. Сва трајања ће бити представљена кроз дијаграме на којима ће се видети на који начин перформансе зависе од броја компоненти на страници.

4.1 Структура пројекта

Направљен је *Maven Java* пројекат чија је структура приказана на слици 4. Пројекат је развијен у развојном окружењу *Spring Tool Suite* верзије *3.8.0.RELEASE* у чијој је позадини *Eclipse Neon* верзије *4.6*. Користи се *Java jdk1.8.0_91* као и *Tomcat 8.0* за апликативни сервер.



Слика 4 – Структура пројекта

Структура пројекта мора бити по *Maven* конвенцији. Структура по директоријумима је следећа:

src/main/java/beans

У овом директоријуму су смештене све *Java* датотеке у којима се налази логика зрна подршке *JSF* апликације.

src/main/java/resources

Када се имплементира веб-апликација, добра је идеја да се на једном месту дефинишу све лабеле и поруке. На овај начин се лако врши локализација апликације, било да се ради о више језика или

о приказу лабела и порука на ћирилици или латиници. У овом директоријуму се налазе две датотеке:

- *resources_sr_cy.properties* – поруке и лабеле на ћирилици
  `matematickiFakultet=Универзитет у Београду - Математички факултет`
- *resources_sr.properties* – поруке и лабеле на латиници
  `matematickiFakultet=Univerzitet u Beogradu - Matematički fakultet`

У датотеци *faces-config.xml* се врши дефинисање ових датотека. Дефинисано је да је подразумевана локализација ћирилица и да је поред ћирилице подржана латиница.

```
<application>
      <locale-config>
         <default-locale>sr_cy</default-locale>
         <supported-locale>sr_cy</supported-locale>
         <supported-locale>sr</supported-locale>
   </locale-config>
   <resource-bundle>
         <base-name>properties.resources</base-name>
         <var>resources</var>
   </resource-bundle>
</application>
```

base-name означава да се датотеке за локализацију називају *resources* (које могу да имају суфиксе из *supported-locale*) и да су смештене у директоријуму *properties*.

var означава варијаблу која се користи у пројекту где нам је потребна вредност на основу кључа из датотеке за локализацију. На пример, уколико на *.xhtml* страници желимо да испишемо лабелу *matematickiFakultet* синтакса је следећа:

```
<p:outputLabel value="#{resources['matematickiFakultet']}" />
```

и на овај начин ће се у страници исписати вредност „Универзитет у Београду - Математички факултет“ на ћирилици или латиници.

Уколико вредност читамо у зрну подршке синтакса је следећа:

```
FacesContext context = FacesContext.getCurrentInstance();
ResourceBundle resources = context.getApplication().getResourceBundle(
   FacesContext.getCurrentInstance(), "resources");
String matematickiFakultet = resources.getString("matematickiFakultet");
```

src/main/webapp/resources/css

У овом директоријуму се налазе све *.css* датотеке у којима се дефинише на који начин *HTML* приказује елементе на екрану. На следећи начин се на *.xhtml* страници у заглављу врши учитавање датотеке *styles.css*:

```
<h:outputStylesheet library="css" name="styles.css"/>
```

src/main/webapp/resources/images

У овом директоријуму се налазе све сличице које се користе у пројекту. Да бисмо учитали слику на *.xhtml* страници, можемо користити компоненту *graphicImage* из *PrimeFaces* библиотеке

```
<p:graphicImage value="/resources/images/matf-grb.gif" styleClass="matf-grb"
	alt="#{resources['matf-grb']}" title="#{resources['matf-grb']}"/>
```

где је *value* путања до сличице *matf-grb.gif*, *styleClass* представља класу из датотеке *styles.css*, док се објашњење чита из датотека за локализацију.

src/main/webapp/resources/js

У овом директоријуму се налазе све датотеке које садрже *jQuery* функције чија је екстензија *.js*. На следећи начин се у заглављу *.xhtml* странице врши учитавање датотеке *functions.js*:

```
<h:outputScript library="js" name="functions.js" />
```

src/main/webapp/view

У овом директоријуму се налазе све *.xhtml* странице.

src/main/webapp/WEB-INF

У овом директоријуму се налазе две датотеке: *faces-config.xml* и *web.xml*. У овом пројекту ће се користити анотације, тако да ће у датотеци *faces-config.xml* бити дефинисана само правила за локализацију апликације која је раније објашњена.

Приликом пребацивања апликације на апликативни сервер потребно је извршити конфигурацију унутар датотеке *web.xml*. Све *JSF* странице се прослеђују *Faces* сервлет-у као део *JSF* имплементационог кôда. Да бисмо били сигурни да је прави сервлет активан када се *JSF* страница захтева од сервера, *JSF URL*-ови имају специјалан формат. У овом пројекту странице имају екстензију *.xhtml*.

```
<servlet-mapping>
	<servlet-name>Faces Servlet</servlet-name>
	<url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

У датотеци *web.xml* се такође може дефинисати која страница се отвара уколико дође до грешке на страни сервера, почетна страница, разна подешавања везана за рад сервера, итд.

У датотеци *web.xml* се почетна страница дефинише на следећи начин:

```
<welcome-file-list>
	<welcome-file>view/index.xhtml</welcome-file>
</welcome-file-list>
```

4.2 Организација страница

Као што је наведено у претходном поглављу, почетна страница пројекта је *index.xhtml* која има следећу структуру:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:c="http://java.sun.com/jsp/jstl/core"
      xmlns:f="http://java.sun.com/jsf/core">
      <head>
            <title></title>
      </head>
      <body>
            <ui:composition template="/view/basicLayout.xhtml">
                  <ui:define name="pageContent">
                        <p:commandButton value="#{resources['navigacija']}"
                              icon="ui-icon-check" action="dobrodosli" />

                        <p:commandButton value="#{resources['redirekcija']}"
                              icon="ui-icon-check"
                              action="dobrodosli?faces-redirect=true" />

                        <p:button outcome="dobrodosli" icon="ui-icon-star"
                              value="#{resources['redirekcijaSaGETParametrom']}" >
                        <f:param name="parametar" value="vrednostParametra" />
                        </p:button>
                  </ui:define>
            </ui:composition>
      </body>
 </html>
```

Страница има комплетну *HTML* структуру тј. унутар елемента *html* се налазе елементи *head* и *body*. У елементу *body* се може видети да се користи фејлсет *ui:composition* чији је шаблон */view/basicLayout.xhtml* и *ui:define* чији је назив *pageContent.* Наслов је дефинисан у шаблону, тако да ће ова страница наследити тај наслов и зато га нећемо наводити (<title></title>). Дугмићи који се налазе на почетној страници су описани у поглављу 3.8 када је било речи о навигацији.

Погледајмо сада структуру странице */view/basicLayout.xhtml.*

Дефиниција *html* елемента је слична као код почетне странице, тј. може се видети у поглављу 3.2.

HTML заглавље има следећу структуру:

```
    <h:head>
        <title>#{resources['title']}</title>
        <h:outputStylesheet library="css" name="styles.css"/>
        <h:outputScript library="js" name="functions.js" />
        <script type="text/javascript">
             pageLoadStart();
        </script>
```

```
    </h:head>
```

Наслов странице се чита из датотека за локализацију и у зависности од изабраног писма ће се исписати на ћирилици или латиници. Учитавају се *CSS* и *jQuery* датотеке и извршава се *jQuery* функција *pageLoadStart()* која је дефинисана у *jQuery* датотеци *functions.js.*

HTML елемент *body* има следећу структуру:

```
<h:body>
    <h:form id="idForm">
        <p:layout>
            <p:layoutUnit position="north" resizable="true"
                    styleClass="noBorder" size="200">
                ...
            </p:layoutUnit>
            <p:layoutUnit position="west" size="200" styleClass="menu-west-east">
                ...
            </p:layoutUnit>
            <p:layoutUnit position="east" size="200" styleClass="menu-west-east">
                ...
            </p:layoutUnit>
                ...
            <p:layoutUnit position="center">
                ...
            </p:layoutUnit>
        </p:layout>

        <pe:remoteCommand id="countPageLoadSpeed" name="countPageLoadSpeed"
           actionListener="#{pageSpeedBean.setPageLoadSpeed}"
           update="pageLoadSpeed ajaxSpeed">
            <pe:methodSignature parameters="java.lang.Integer" />
            <pe:methodParam name="pageSpeed"/>
        </pe:remoteCommand>

        <pe:remoteCommand id="countAjaxSpeed" name="countAjaxSpeed"
           actionListener="#{pageSpeedBean.setAjaxSpeed}" update="ajaxSpeed">
            <pe:methodSignature parameters="java.lang.Integer" />
            <pe:methodParam name="ajaxSpeed"/>
        </pe:remoteCommand>
    </h:form>
    <ui:debug hotkey="n" />
</h:body>
```

Елемент *body* се састоји од форме и фејслета *ui:debug*. Компонента *ui:debug* приликом иницијализације странице узима генерисано стабло компоненти и све варијабле које се користе на страници, тј. објекте зрна и њихове опсеге. Извршавањем команде *ctrl+atl+n* се отвара прозор у коме можемо видети стабло компоненти и опсег варијабли. Стабло компоненти из поглавља 3.6 је извучено из овог прозора.

Форма садржи две компоненте ***remoteCommand*** библиотеке *PrimeFaces Extensions*. Ова компонента је споменута у поглављу 3.9 када је било речи о библиотеци *PrimeFaces Extensions.*

Компонента *remoteCommand* је веома корисна пошто она служи да повеже методу зрна са *JavaSctipt* догађајем. То је једини начин да се на *JavaScript* догађај позове Јава метода.

Компонента садржи неколико атрибута: идентификатор (*id*) који је јединствен на нивоу странице, *name* је назив JavaScript функције, *actionListener* је метода зрна која се позива, док атрибут *update* садржи идентификаторе компоненти чија се вредност треба освежити тј. компоненте које се требају поново исцртати на истом месту. Такође можемо видети и параметре компоненте *remoteCommand*, што значи да треба да се позове *JavaScript* функција са једним параметром и да ће се у случају *remoteCommand*-е

countPageLoadSpeed(loadTime)

позвати метода

```
public void setPageLoadSpeed(Integer pageLoadSpeed) {}
```

зрна *pageSpeedBean. JavaScript* функција *countPageLoadSpeed(loadTime)* се позива унутар датотеке *functions.js*. Компонента *remoteCommand* постоји и у библиотеци *PrimeFaces*, али се у тој библиотеци не прослеђују параметри на овај начин. JavaScript функција се у том случају позива на следећи начин:

countPageLoadSpeed({param1:loadTime});

Позива се метода зрна без параметара, па се унутар методе на следећи начин долази до вредности прослеђеног параметра из *JavaScript*-а:

```
Map<String, String> params = FacesContext.getCurrentInstance().
      getExternalContext().getRequestParameterMap();
Integer param1 = Integer.valueOf(params.get("param1"));
```

Једноставније коришћење компоненте *remoteCommand* из библиотеке *PrimeFaces Extensions* у односу на компоненту из библиотеке *PrimeFaces.*

Форма такође у себи садржи *PrimeFaces* компоненту *layout* која се бави распоредом на страници. Састоји се од компоненти *layoutUnit* од којих свака има своју позицију. Север одговара позицији горе, запад позицији лево, исток позицији десно, а један *layoutUnit* је позициониран у средини странице (*center*). *layoutUnit* на северној позицији има следећу структуру:

```
<p:layoutUnit position="north" resizable="true" styleClass="noBorder" size="200">
    <p:layout>
        <p:layoutUnit position="west" styleClass="noBorder" size="200">
            <p:graphicImage value="/resources/images/matf-grb.gif"
                    styleClass="matf-grb" alt="#{resources['matf-grb']}"
                    title="#{resources['matf-grb']}"/>
        </p:layoutUnit>
        <p:layoutUnit position="center" styleClass="headerContent" size="200">
            <p:outputLabel id="matfLabel"
                    value="#{resources['matematickiFakultet']}" />
            <br /><br />
```

```
            <p:outputLabel value="#{resources['imePrezime']}" />
            <br /><br />
            <p:outputLabel value="#{resources['tema']}" style="font-size:30px" />
            <br /><br />
            <p:outputLabel value="#{resources['masterRad']}" />
        </p:layoutUnit>
    </p:layout>
</p:layoutUnit>
```

layoytUnit на северној позицији странице *basicLayout.xhtml* се састоји од једне компоненте *layout* у којој се на левој страни налази грб Математичког факултета, док се у централном делу налазе четири лабеле. Вредност атрибута *styleClass* једна од класа из *css* датотеке. У атрибуту style директно уносимо вредност *css*-а који желимо на применимо на компоненти. Подсећање да је овај део кôда анализиран у поглављу 3.6 када је описивано стабло компоненти.

На левој (западној) страни странице *basicLayout.xhtml* учитавамо фејслет *ui:insert* у коме учитавамо садржај странице *menu.xhtml* коришћењем фејслета *ui:include*.

```
<p:layoutUnit position="west" size="200" styleClass="menu-west-east">
    <ui:insert name="menu">
        <ui:include src="/view/menu.xhtml"/>
    </ui:insert>
</p:layoutUnit>
```

Датотека *menu.xhtml* се састоји од дугмета који води ка почетној страници (*index.xhtml*) и менија чије ставке садрже линкове ка другим страницама у пројекту. Следећи фрагмент кôда садржи део странице *menu.xhtml*:

```
<p:button icon="fa fa-home" outcome="/view/index" />
<p:menu toggleable="true">
    <p:submenu label="#{resources['statistika']}">
        <p:menuitem value="#{resources['statistika'].concat('')
            .concat(resources['ucitavanje'])}" icon="fa fa-line-chart"
            outcome="/view/statisticPageLoad" />
        <p:menuitem value="#{resources['statistika']} AJAX"
            icon="fa fa-line-chart" outcome="/view/statisticAjax" />
    </p:submenu>
</p:menu>
```

Кликом на ставку подменија, извршава се редирекција на страницу која се налази у атрибуту *outcome*.

На десној (источној) страни странице *basicLayout.xhtml* се учитава страница *pageSpeed.xhtml*, као и два линка који позивају промену писма на ћирилицу или латиницу.

```
<p:layoutUnit position="east" size="200" styleClass="menu-west-east">
    <ui:insert name="pageSpeed">
        <ui:include src="/view/pageSpeed.xhtml"/>
    </ui:insert>
    <p:commandLink id="cy" actionListener="#{languageBean.localeCodeChangedCy}"
        rendered="#{languageBean.localeCode.equals('sr')}" ajax="false">
```

```
        <h:outputText value="#{resources['cirilica']}" />
    </p:commandLink>
    <p:commandLink id="lat" actionListener="#{languageBean.localeCodeChanged}"
        rendered="#{languageBean.localeCode.equals('sr_cy')}" ajax="false">
        <h:outputText value="#{resources['latinica']}" />
    </p:commandLink>
</p:layoutUnit>
```

Од два линка само ће један бити видљив у зависности од тренутног писма које је изабрано. Уколико је на сајту активна ћирилица, биће видљив линк са вредношћу латиница и супротно. Ово се дефинише у атрибуту *rendered*.

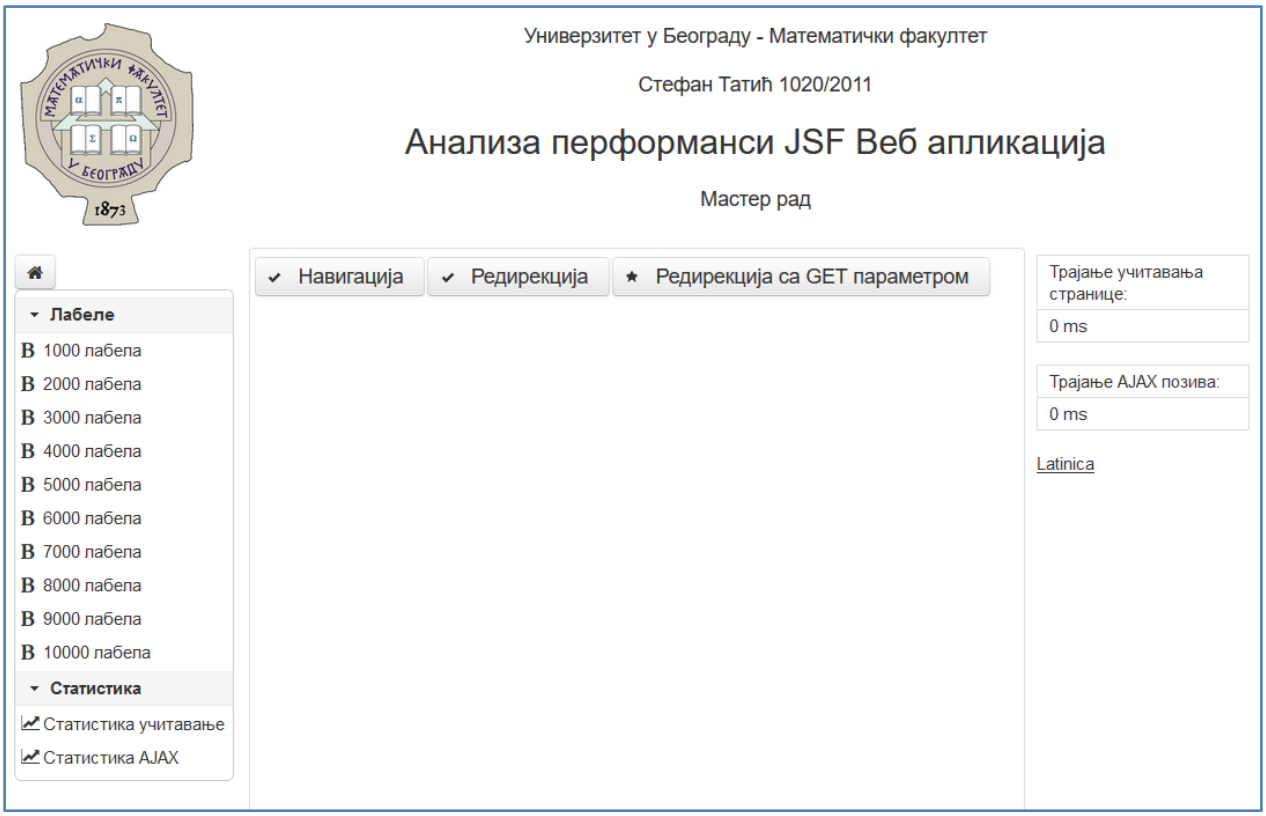
И на крају, на централном делу екрана се исписује главни саджај странице. Само је дефинисан фејслет *ui:insert* без *ui:include* тј. није дефинисано која *.xhtml* страница ће се учитати на том месту.

```
<p:layoutUnit position="center">
       <ui:insert name="pageContent"/>
</p:layoutUnit>
```

Вратимо се сада на дефиницију почетне странице која у себи садржи следећи део кôда:

```
<ui:composition template="/view/basicLayout.xhtml">
    <ui:define name="pageContent">
       ...
    </ui:define>
</ui:composition>
```

То значи да страница *index.xhtml* користи шаблон *basicLayout.xhtml* и да ће се садржај унутар елемента *ui:define* са странице *index.xhtml* учитати на централној позицији странице *basicLayout.xhtml*. На слици 5 се може видети изглед почетне странице.



Слика 5 – Изглед почетне странице

5. Анализа перформанси *JSF* страница

У овом поглављу биће анализиране перформансе *JSF* страница, односно времена која су потребна да се страница учита и времена потребна за реализовање *AJAX* захтева у односу на број компоненти на страници. Биће описан и начин на који се рачунају времена, а добијени резултати ће бити представљени кроз четири дијаграма. Перформансе ће бити тестиране на два рачунара са различитим спецификацијама.

5.1 Значајна зрна и странице

На десној (источној) страни странице *basicLayout.xhtml* учитава се страница *pageSpeed.xhtml* која се састоји од једног податка за трајање учитавања странице и другог податка који означава трајање *AJAX* позива у милисекундама. На истој страници се налази и *ajax-loader.gif* који је у пројекту смештен у директоријуму *webapp/resources/images*, а тај *gif* је кружић који се врти и видљив је само у току трајања *AJAX* позива. Следећи фрагмент кôда је део странице *pageSpeed.xhtml*:

```
<p:panelGrid columns="1" style="width:95%">
    <p:outputLabel value="#{resources['vremeZaUcitavanjeStranice']}:" />
    <p:outputLabel id="pageLoadSpeed" value="#{pageSpeedBean.pageLoadSpeed} ms" />
</p:panelGrid>
<br />
<p:panelGrid columns="1" style="width:95%">
    <p:outputLabel value="#{resources['vremeZaAjax']}:" />
    <p:outputLabel id="ajaxSpeed" value="#{pageSpeedBean.ajaxSpeed} ms" />
</p:panelGrid>
<br />
<span id="ajax-loader" class="ajax-status" style="display:none;"></span>
```

Променљиве које чувају трајање учитавања странице и *AJAX*-а су *pageLoadSpeed* и *ajaxSpeed* су својства зрна *PageSpeedBean* који има опсег сесије.

```
@SessionScoped
@ManagedBean(name="pageSpeedBean")
public class PageSpeedBean {

	private Integer pageLoadSpeed;
	private Integer ajaxSpeed;
	private HashMap<String,ArrayList<Integer>> pageLoadHm;
	private HashMap<String,ArrayList<Integer>> ajaxHm;

	…

}
```

Поред споменутих својстава, *PageSpeedBean* садржи још два својства у којима се чувају трајања свих учитавања страница као и трајања свих *AJAX* позива. Њихов тип је хеш мапа, која за кључ има назив *.xhtml* странице, а вредност је листа целих бројева који представљају трајање у милисекундама.

5.2 Странице на којима се анализирају перформансе

Испод подменија *Лабеле* који се налази са леве стране налази се линкови ка 10 страница. Странице имају назив *labele1000.xhtml*, *labele2000.xhtml*, ... , *labele10000.xhtml* и састоје се од онолико лабела колико стоји у називу странице.

```
<ui:composition template="/view/basicLayout.xhtml">
    <ui:define name="pageContent">
        <h3>1000 #{resources['labela']}</h3>
        <p:commandButton value="#{resources['izvrsiajaxakciju']}" update="ajaxSpeed"
            icon="ui-icon-check" onstart="ajaxStart();" onsuccess="ajaxEnd();" />
        <p:panelGrid columns="10">
            <c:forEach var="i" begin="1" end="1000" step="1" varStatus ="status">
                <p:outputLabel value="#{resources['labela']}" />
            </c:forEach>
        </p:panelGrid>
    </ui:define>
</ui:composition>
```

Странице користе шаблон *basicLayout.xhtml* и садржај страница ће бити исписан у централном делу екрана. Странице поред лабели садрже и *commandButton* који нема никакву акцију у зрну, тј. нема ни *action* ни *actionListener*. Приликом стартовања (*onstart*) позива *jQuery* функцију *ajaxStart* која чува време када је *AJAX* стартован и позива приказ *ajax loader*-а.

```
function ajaxStart() {
	isAjaxStarted = true;
	ajaxStartVar = Date.now();
	$('#ajax-loader').css('display', 'block');
}
```

Када се извршавање *AJAX*-а заврши, позива се *jQuery* функција *ajaxEnd* која склања *ajax loader* и рачуна разлику времена када се *AJAX* завршио и времена када је *AJAX* почео са извршавањем.

```
function ajaxEnd() {
	isAjaxStarted = false;
	countAjaxSpeed(Date.now()-ajaxStartVar);
	$('#ajax-loader').css('display', 'none');
}
```

Када се израчуна трајање извршавања *AJAX* позива, потребно је тај податак сачувати ради праћења статистике и потребно је тај податак исписати у десном делу екрана (*pageSpeed.xhtml*). Ако погледамо део кôда из поглавља 5.1 где је описан саджај *pageSpeed.xhtml* странице, може се приметити да је вредност компоненте *ajaxSpeed* вредност својства *ajaxSpeed* зрна *PageSpeedBean*. То значи да време трајања *AJAX* позива које је израчунато у *jQuery*-ју (*Date.now()-ajaxStartVar*) треба бити прослеђено зрну.

Подсетимо се компоненте *remoteCommand* из поглавља 4.3.

```
<pe:remoteCommand id="countAjaxSpeed" name="countAjaxSpeed"
    actionListener="#{pageSpeedBean.setAjaxSpeed}" update="ajaxSpeed">
    <pe:methodSignature parameters="java.lang.Integer" />
```

```
    <pe:methodParam name="ajaxSpeed"/>
</pe:remoteCommand>
```

У *jQuery* функцији *ajaxEnd* позива се функција *countAjaxSpeed* која је уствари *remoteCommand*. Уз помоћ ове компоненте проследиће се трајање извршавања *AJAX*-а зрну *PageSpeedBean* и освежиће се вредност компоненте са идентификатором *ajaxSpeed* која је назначена у *update* атрибуту. На овај начин ће трајање извршавања *AJAX*-а бити исписано на десном делу екрана.

Следећи пример представља један од могућих одговора (*response*) сервера након извршавања *remoteCommand*-е у ком се од странице захтева да изврши промену вредности компоненте *idForm:ajaxSpeed* на 122 ms. Овај податак се може прочитати у конзоли веб-прегледача.

```
<partial-response>
    <changes>
        <update id="idForm:ajaxSpeed">
            <label id="idForm:ajaxSpeed" class="ui-outputlabel ui-widget">
                122 ms
            </label>
        </update>
    </changes>
</partial-response>
```

На сличан начин се рачуна трајање учитавања странице и прослеђује зрну у ком се сва трајања учитавања памте. Трајање учитавања странице се рачуна у *jQuery*-ју на следећи начин:

```
$(window).load(function() {
    var loadTime = window.performance.timing.domContentLoadedEventEnd-
          window.performance.timing.navigationStart;
    countPageLoadSpeed(loadTime);
});
```

5.3 Странице за приказ статистике

Испод подменија *Статистика* налазе се два линка, један који води ка страници за приказ трајања учитавања страница (*statisticPageLoad.xhtml*), а други који води ка страници за приказ трајања *AJAX* позива (*statisticAjax.xhtml*). Обе странице садрже четири дијаграма који су компоненте *chart* из *PrimeFaces* библиотеке.

```
<p:chart id="chart" type="line" style="height:600px;"
    model="#{statisticAjaxBean.ajaxSpeedLinearModel}" />
<p:chart id="chartAvg" type="line" style="height:600px;"
    model="#{statisticAjaxBean.ajaxSpeedAvgLinearModel}" />
<p:chart id="chartAvgPerComponent" type="line" style="height:600px;"
    model="#{statisticAjaxBean.ajaxSpeedAvgPerComponentLinearModel}" />
<p:chart id="chartLog" type="line" style="height:600px;"
    model="#{statisticAjaxBean.ajaxSpeedLogLinearModel}" />
```

Зрна која одговарају овим страницама су *StatisticPageLoadBean* и *StatisticAjaxBean*. Следећи фрагмент кôда представља део зрна *StatisticAjaxBean*.

```
@ViewScoped
@ManagedBean(name="statisticAjaxBean")
public class StatisticAjaxBean {

	private LineChartModel ajaxSpeedLinearModel;
	private LineChartModel ajaxSpeedAvgLinearModel;
	private LineChartModel ajaxSpeedAvgPerComponentLinearModel;
	private LineChartModel ajaxSpeedLogLinearModel;

	@ManagedProperty(value="#{pageSpeedBean}")
	private PageSpeedBean pageSpeedBean;

}
```

StatisticAjaxBean (*StatisticPageLoadBean*) има опсег погледа и садржи четири својства типа *LinearChartModel* у којима се специфицира изглед дијаграма. Код прва три дијаграма *y*-оса представља трајање у милисекундама, док код четвртог представља логаритам трајања у милисекундама. Код првог дијаграма *x*-оса представља редни број захтева, а код осталих број компоненти.

- *ajaxSpeedLinearModel (pageLoadSpeedLinearModel)* – дијаграм који за сваку страницу *labele1000*, ..., *labele10000* исцртава по једну функцију чије тачке представљају трајање извршавања *AJAX*-а (учитавања странице) у милисекундама за сваки захтев (енгл. *request*).
- *ajaxSpeedAvgLinearModel (pageLoadAvgLinearModel)* – дијаграм који исцртава једну функцију чије тачке представљају просечно трајање *AJAX* позива (учитавања странице) за сваку страницу.
- *ajaxSpeedAvgPerComponentLinearModel (pageLoadAvgPerComponentLinearModel)* – дијаграм који исцртава једну функцију чије тачке представљају однос просечног трајања *AJAX* позива (учитавања странице) и броја компоненти на страници (просечно трајање у милисекундама / број компоненти), тј. време које је потребно за извршавање *AJAX*-а (учитавања странице) за једну компоненту.
- *ajaxSpeedLogLinearModel (pageLoadLogLinearModel)* – дијаграм који је сличан другом дијаграму, уместо просечног трајања приказује логаритам просечног трајања *AJAX* позива (учитавања странице)

Унутар ових зрна учитавамо зрно *PageSpeedBean* у коме се памте трајања свих учитавања страница и *AJAX* позива. Унутар *@PostConstruct* метода зрна се врши иницијализација свих дијаграма на основу података из *PageSpeedBean*.

5.4 Анализа перформанси

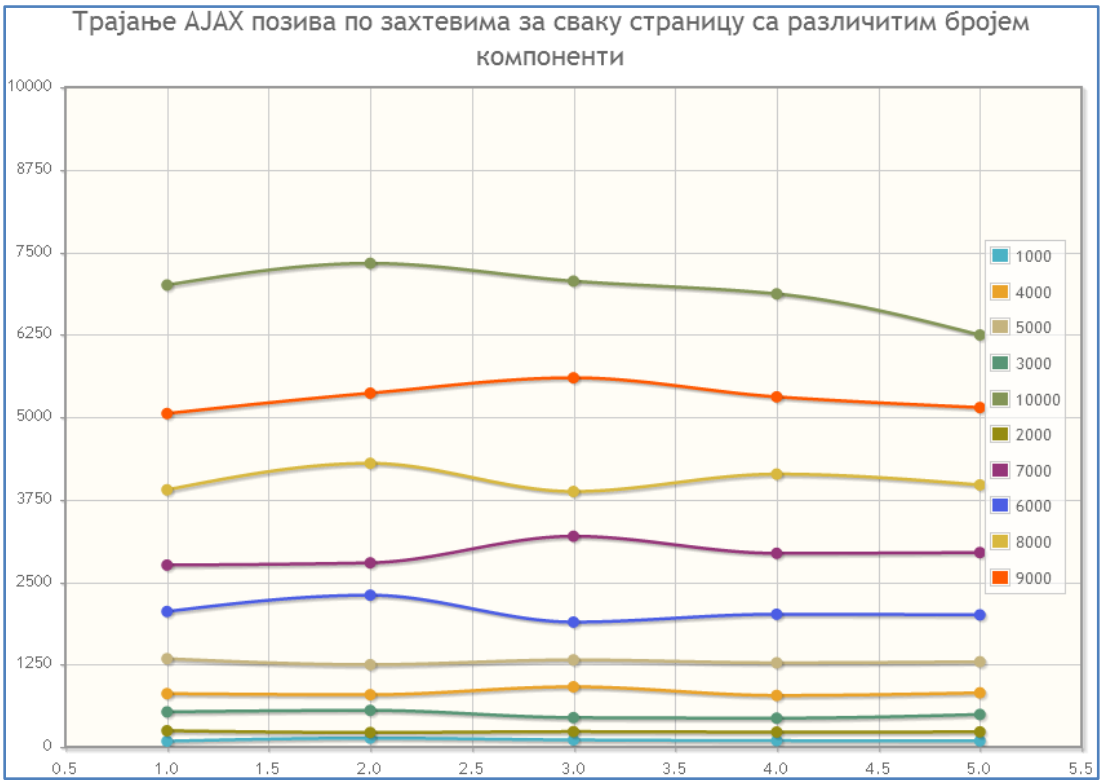
Анализа перформанси је извршена на два рачунара са следећим спецификацијама:

- Рачунар 1 – Intel(R) Core(TM)2 Duo CPU T6400 @ 2.00GHz, 2.00 GB RAM, 32-bit OS Win7
- Рачунар 2 – Intel(R) Core(TM) i5-4460 CPU @ 3.20 GHz, 8.00 GB RAM, 64-bit OS Win10

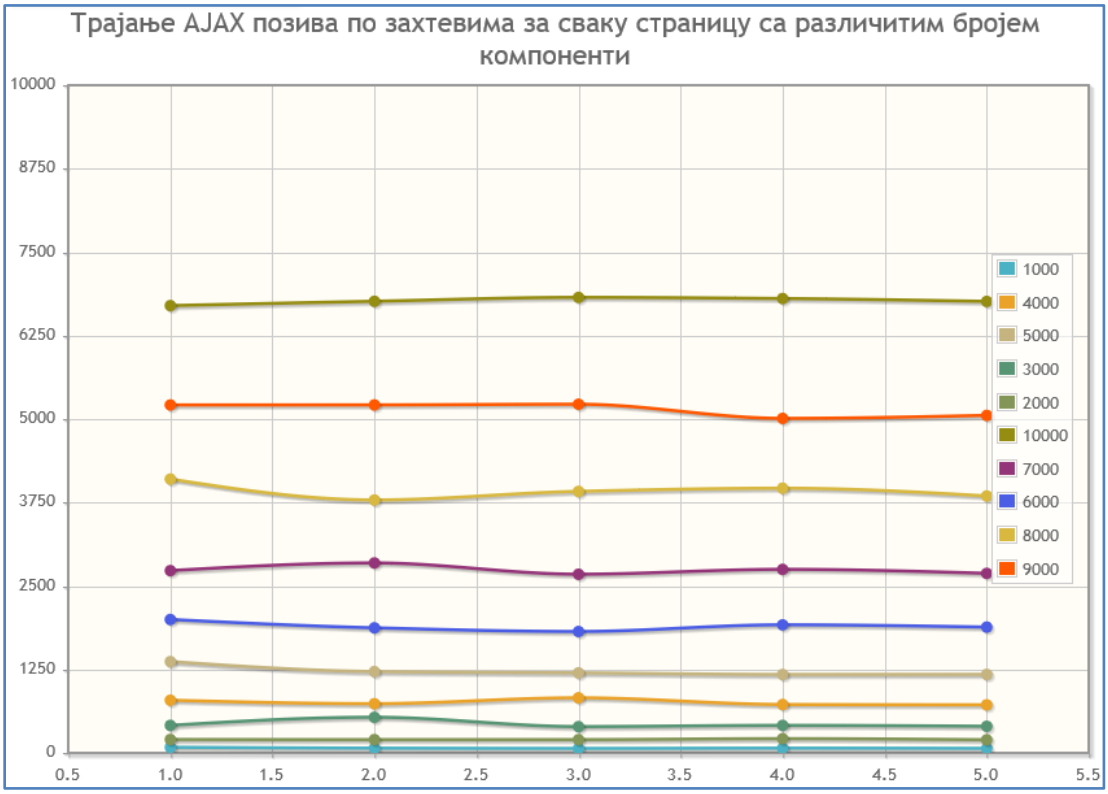
Апликација је покренута у веб-прегледачу *Firefox*, верзије 47.0.1.

Тестирање је обављено тако што је свака страница учитана 5 пута и на свакој страници се 5 пута извршио *AJAX* позив. Слике представљају дијаграме са страница *statisticAjax.xhtml* и *statisticPageLoad.xhtml*:

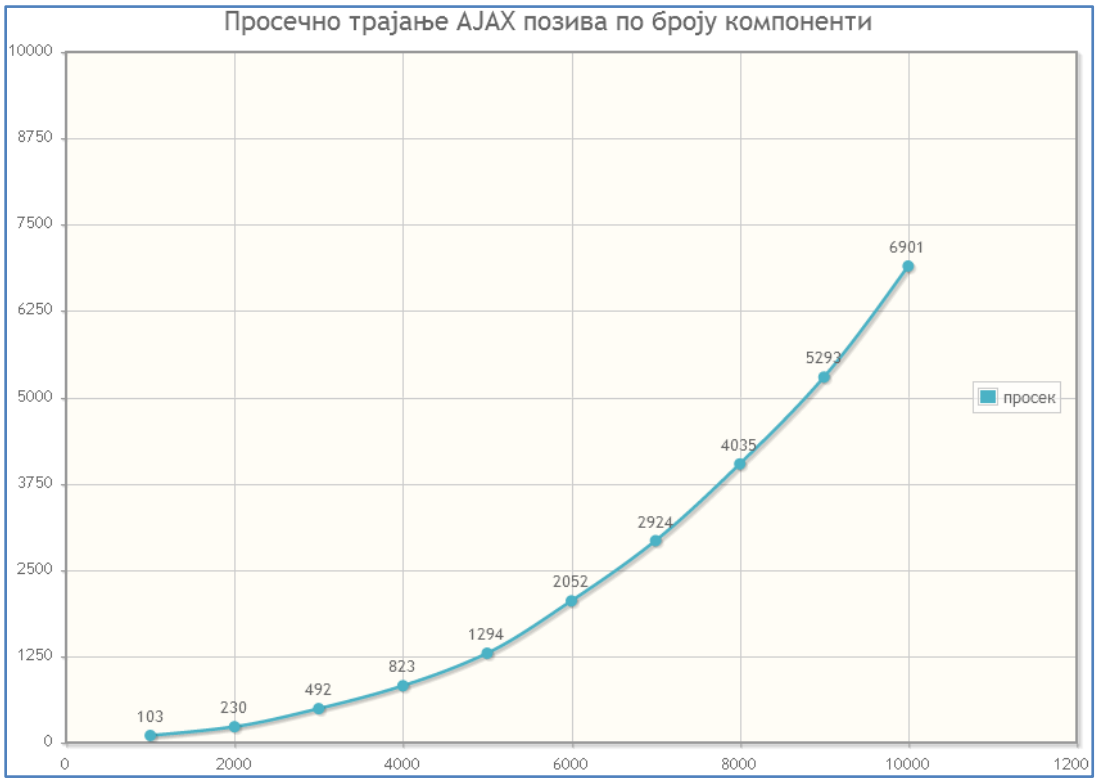
- Слике 6 и 7 представљају дијаграме трајања *AJAX* позива за сваку страницу
- Слике 8 и 9 представљају дијаграме просечног трајања *AJAX* позива за сваку страницу
- Слике 10 и 11 представљају дијаграме трајања *AJAX* позива по једној компоненти тј. просечно трајање подељено бројем компоненти
- Слике 12 и 13 представљају дијаграме логаритма просечног трајања *AJAX* позива
- Слике 14 и 15 представљају дијаграме трајања учитавања сваке странице
- Слике 16 и 17 представљају дијаграме просечног трајања учитавања сваке странице
- Слике 18 и 19 представљају дијаграме трајања учитавања једне компоненте тј. просечно трајање учитавања странице подељено бројем компоненти
- Слике 20 и 21 представљају дијаграме логаритма просечног трајања учитавања странице



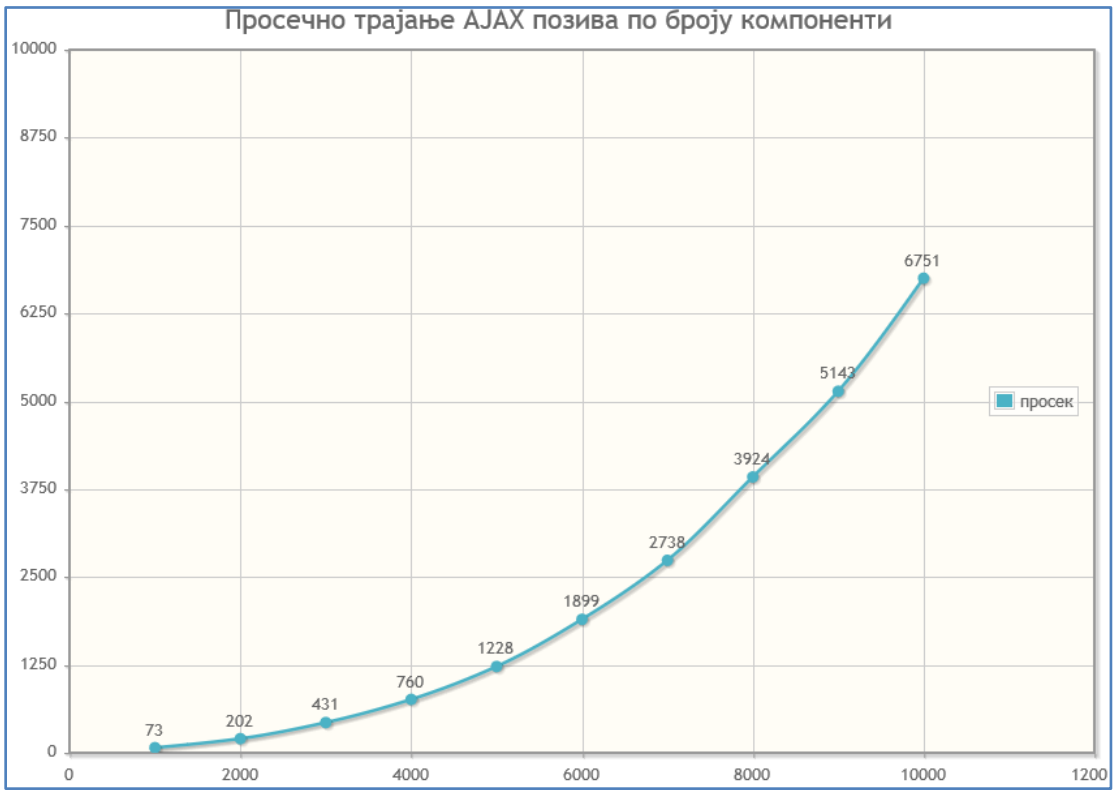
Слика 6 – Трајање AJAX позива по захтевима за сваку страницу (Рачунар 1)



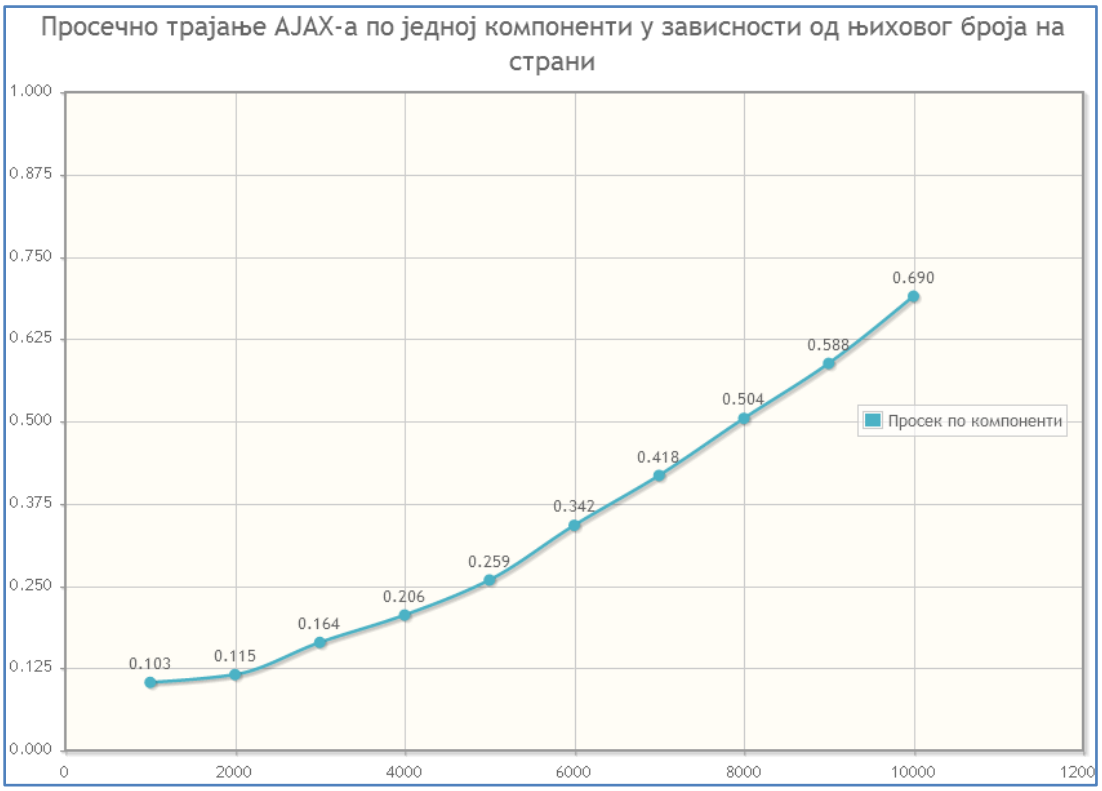
Слика 7 – Трајање AJAX позива по захтевима за сваку страницу (Рачунар 2)



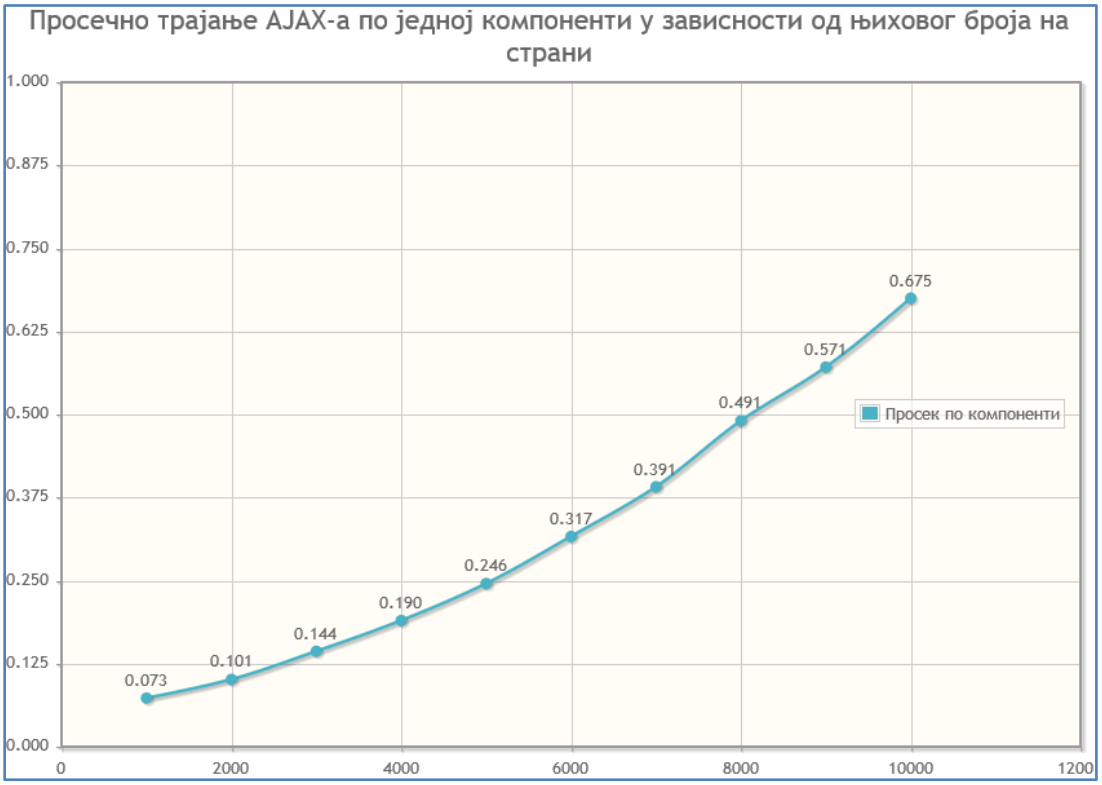
Слика 8 – Просечно трајање AJAX позива (Рачунар 1)



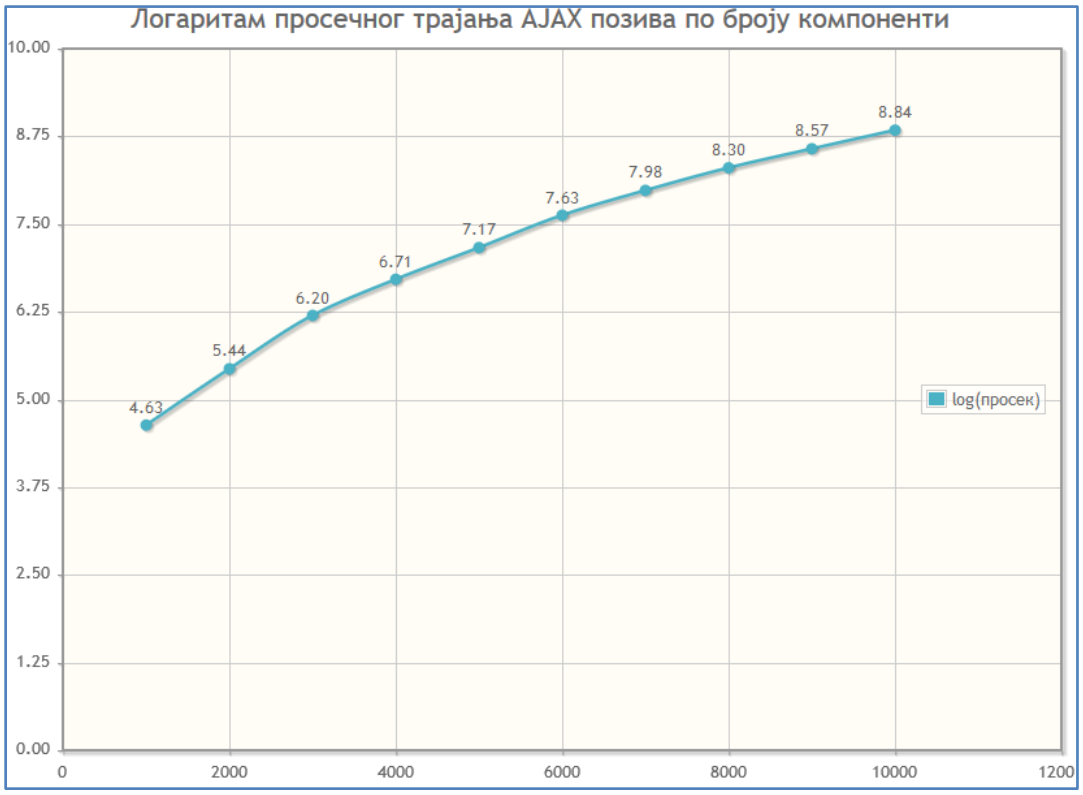
Слика 9 – Просечно трајање AJAX позива (Рачунар 2)



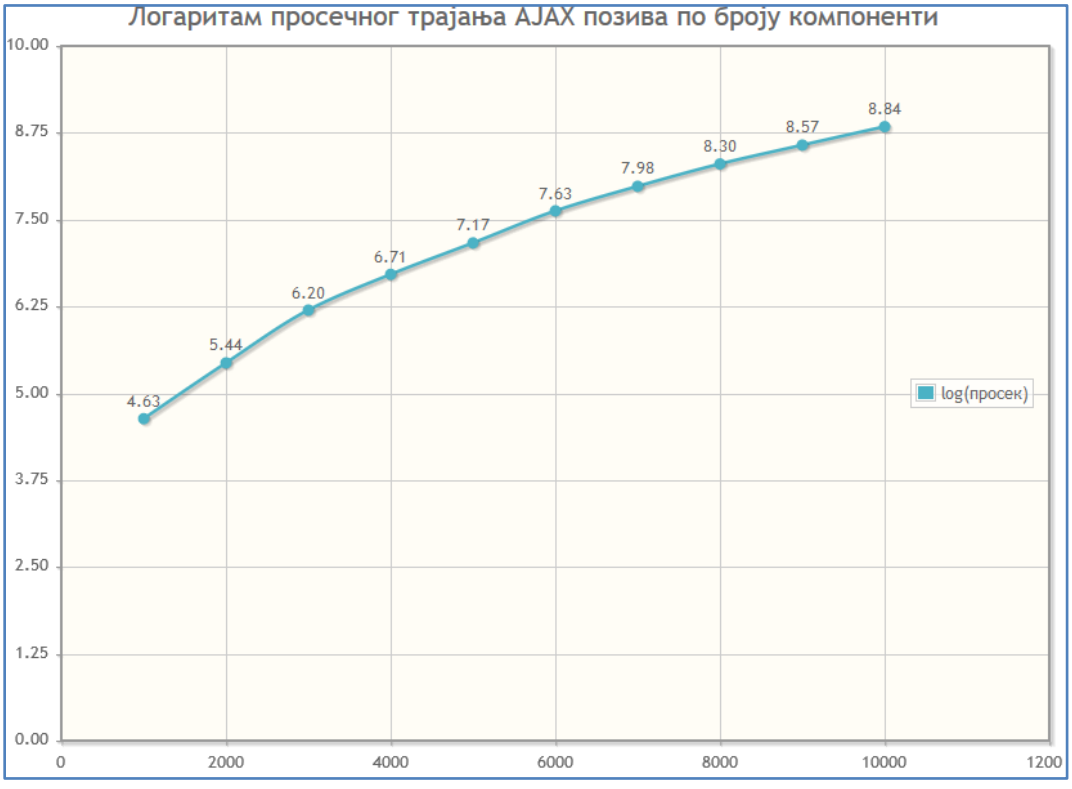
Слика 10 – Просечно трајање AJAX позива подељено бројем компоненти (Рачунар 1)



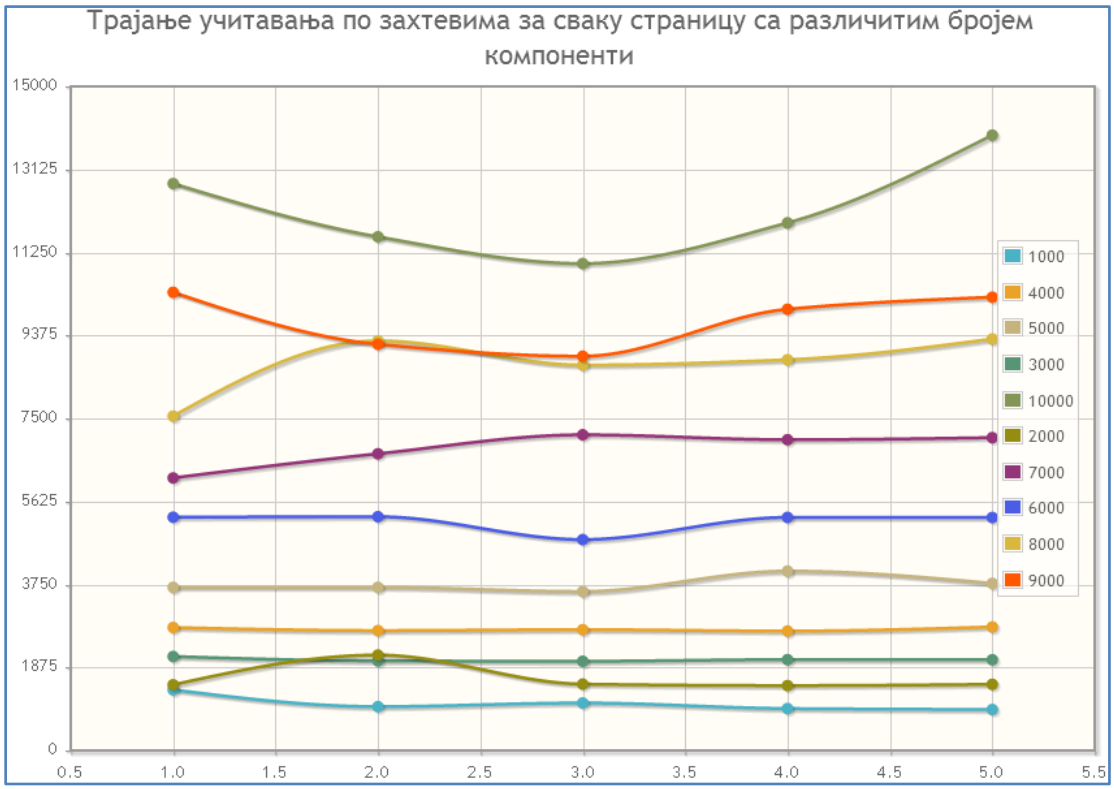
Слика 11 – Просечно трајање AJAX позива подељено бројем компоненти (Рачунар 2)



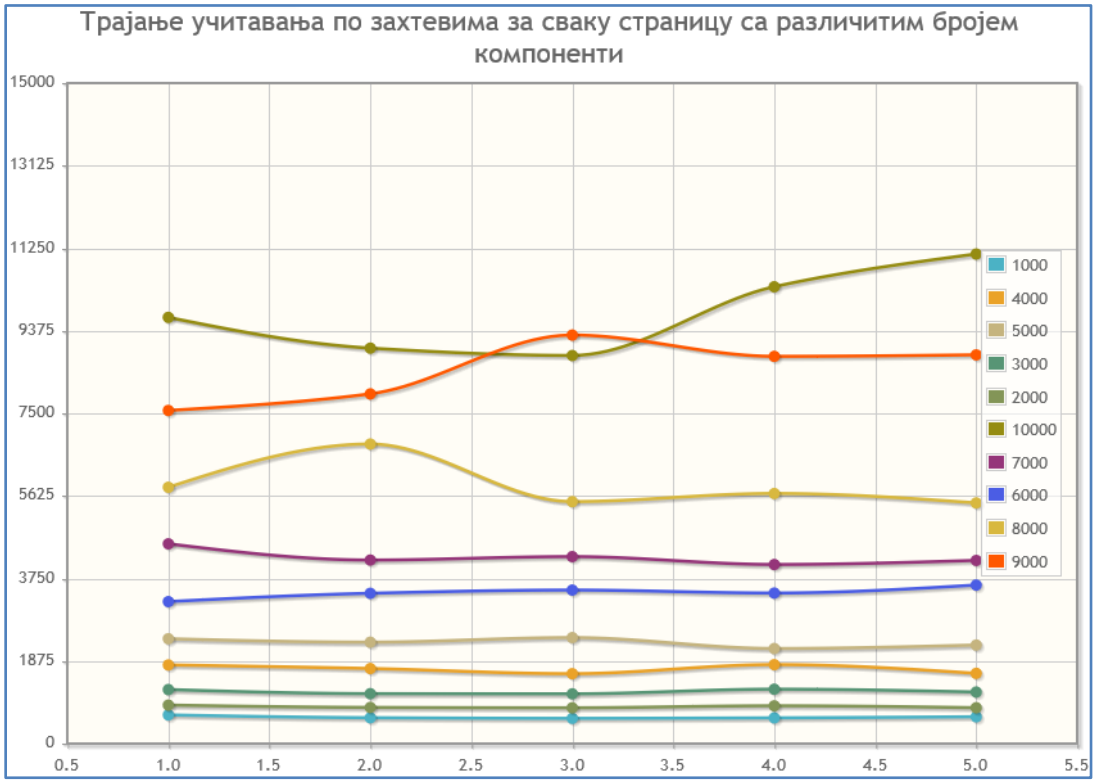
Слика 12 – Логаритам просечног трајања AJAX позива (Рачунар 1)



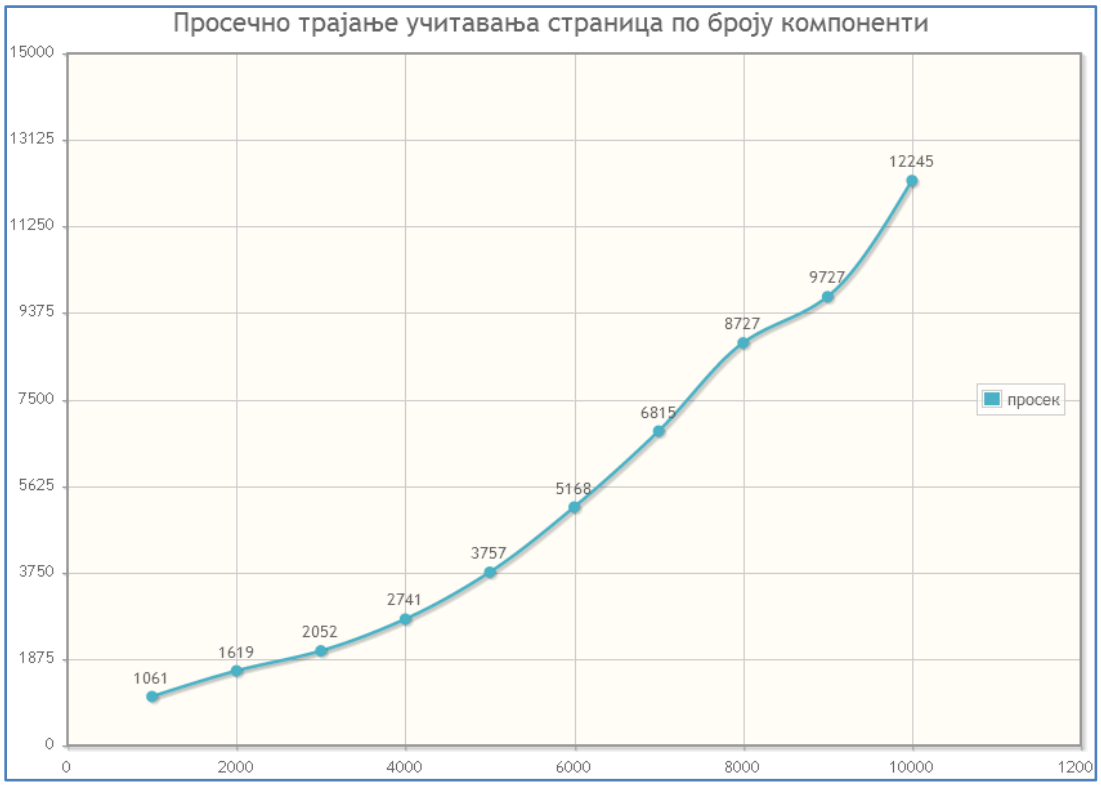
Слика 13 – Логаритам просечног трајања AJAX позива (Рачунар 2)



Слика 14 – Трајање учитавања по захтевима за сваку страницу (Рачунар 1)

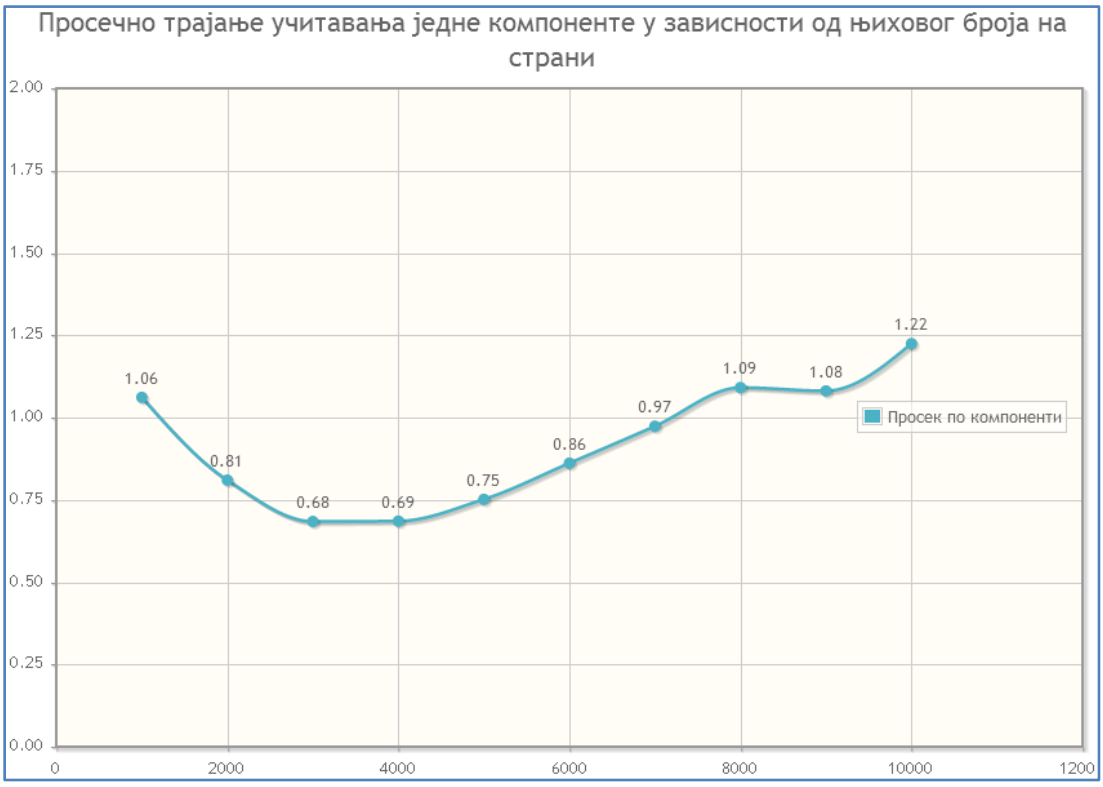


Слика 15 – Трајање учитавања по захтевима за сваку страницу (Рачунар 2)

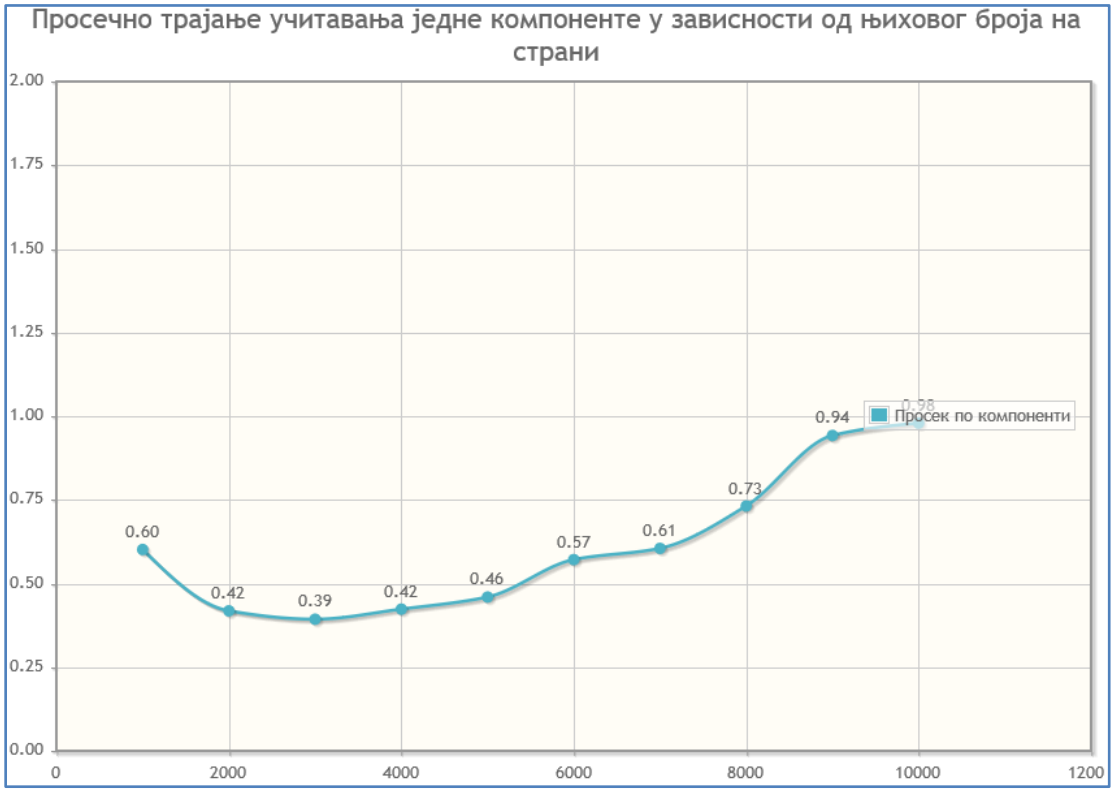


Слика 16 – Просечно трајање учитавања странице (Рачунар 1)

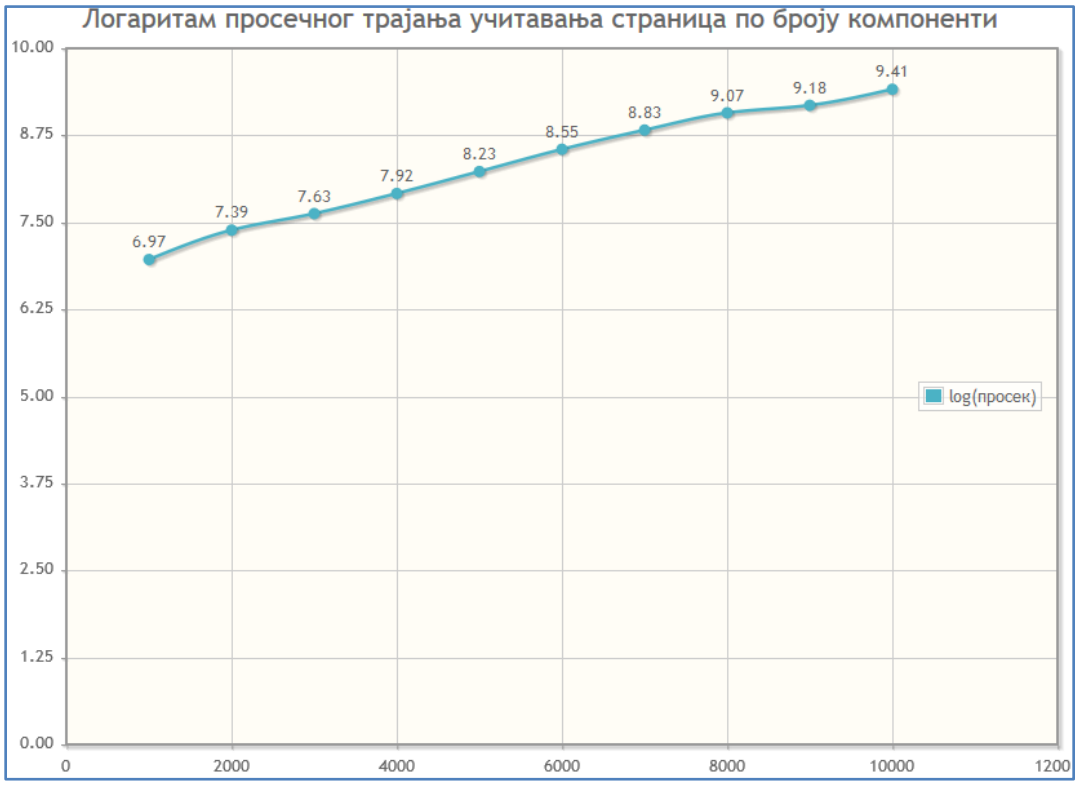
Слика 17 – Просечно трајање учитавања странице (Рачунар 2)



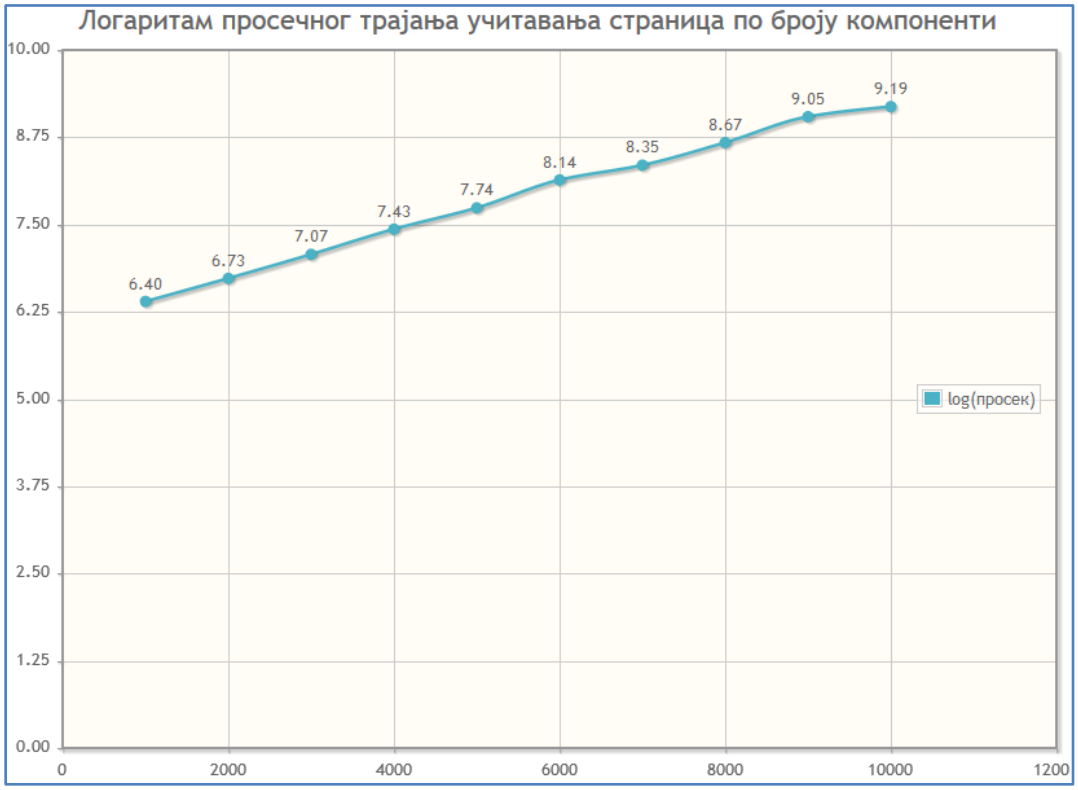
Слика 18 – Просечно трајање учитавања странице подељено бројем компоненти (Рачунар 1)



Слика 19 – Просечно трајање учитавања странице подељено бројем компоненти (Рачунар 2)



Слика 20 – Логаритам просечног трајања учитавања странице (Рачунар 1)



Слика 21 – Логаритам просечног трајања учитавања странице (Рачунар 2)

Анализа трајања AJAX позива

На сликама 6 и 7 посматрајмо, на пример, жуту функцију која одговара страници са 8000 компоненти где можемо видети да су се свих пет *AJAX* позива извршила између 3.75 и 5 секунди. Такође можемо видети да је много већа разлика у трајању *AJAX* позива између страница са 9000 и 10000 компоненти у односу на странице од 1000 и 2000 компоненти.

На сликама 8 и 9 можемо закључити да је просечно трајање *AJAX* позива за страницу од 1000 компоненти око 0.1 секунди, од 5000 компоненти око 1.25 секунди, док је просечно трајање *AJAX* позива за страницу од 10000 компоненти мало мањи од 7 секунди.

На сликама 10 и 11 можемо закључити да је просечно трајање *AJAX* позива по једној компоненти за страницу од 1000 компоненти око 0.1 милисекунди, од 5000 компоненти око 0.3 милисекунди, а од 10000 компоненти око 0.7 милисекунди.

На сликама 12 и 13 се налазе дијаграми који представљају логаритам просечног трајања *AJAX* позива. С обзиром да су разлике просечног трајања *AJAX* позива мале, логаритамска функција је идентична за резултате добијене од оба рачунара.

До 5000 компоненти трајање *AJAX* позива не прелази 1.25 секунди, а разлика трајања *AJAX* позива на страницама од 9000 и 10000 компоненти је већа од 1.5 секунди. Можемо закључити да повећавањем броја компоненти на страници перформансе нагло опадају тј. трајање *AJAX* позива постаје све дуже.

Ако посматрамо перформансе на два рачунара који се веома разликују по спецификацијама, можемо закључити да извршавање *AJAX* позива не зависи од јачине клијентског рачунара пошто смо добили веома сличне перформансе.

Анализа трајања учитавања страница

На сликама 14 и 15 посматрајмо такође жуту функцију која одговара страници која има 8000 компоненти. На слабијем рачунару се страница учитавала између 7.5 и 9.5 секунди, док се на јачем рачунару учитавала између 5.5 и 7 секунди. Можемо приметити да трајања учитавања страница које имају 5000 и мање компоненти нису прелазила 2 секунде, док се трајања учитавања све више повећава како се повећава број компоненти када је њихов број већи од 5000. Сложеније странице се учитавају између 3 и 11 секунди.

На сликама 16 и 17 можемо видети просечна трајања учитавања свих страница. На слабијем рачунару се страница од 8000 компоненти учитавала 8727, док се на јачем рачунару учитавала 5854 милисекунди што је 30% брже. Што се осталих страница тиче, на јачем рачунару је трајање учитавања било брже за 30-45% у односу на слабији рачунар.

На сликама 18 и 19 можемо видети трајање учитавања једне компоненте у односу на њихов број на страници. Трајање учитавања једне компоненте је најбрже код страница које имају 3000-4000 компоненти, док се трајање учитавања повећава како се повећава број компоненти. У односу на перформансе трајања извршавања *AJAX*-а када се трајање само повећавало како се повећавао број компоненти (трајање за страницу од 1000 компоненти је било најкраће), овде то није случај. Време

учитавања једне компоненте на страници од 1000 компоненти је знатно веће од страница са 3000-4000 компоненти, зато што нема наглог пораста трајања учитавања код простијих страница као што је био случај код *AJAX* позива.

На сликама 20 и 21 је представљена логаритамска функција просечног трајања учитавања страница. С обзиром да постоји разлика просечног трајања учитавања страница посматрајући резултате оба рачунара, у овом случају се то одразило на логаритамску функцију.

6. Закључак

JavaServer Faces је веома моћан развојни оквир који нам пружа велике могућности за развој модерних и сложених апликација. Избор компоненти је превелики, тако да у сваком тренутку решење можемо наћи у некој од њих. *JSF* нам нуди могућност да предефинишемо основни начин рада компоненти, али то понекада није могуће у превеликој мери. Можемо предефинисати рад неке од *PrimeFaces* компоненти тако што ћемо извршити измене у *PrimeFaces jQuery* датотекама, али може доћи до великих проблема у случају промене верзије *PrimeFaces*-а.

Уколико користимо *JSF* за развој веб-сајта где је у првом плану сама презентација, а притом перформансе нису претерано битне, *JSF* је одличан избор. Али, у случају развоја модерних веб-апликација у којима је акценат на обради података и брзом уносу и приказу, мора се водити рачуна о сложености страница. Дизајн апликације се у том случају мора заснивати на простијим страницама, односно на страницама које немају превелики број компоненти.

Ако посматрамо резултате из претходног поглавља, можемо закључити да није пожељно дизајнирати странице које имају више од 5000 компоненти. Није пожељно да се страница учитава дуже од 2 секунде као ни трајање *AJAX*-а дуже од 1 секунде. Времена која смо добили представљају само време учитавања страница. Уколико нам је за припрему података потребно извршавање упита који дохвата податке из базе података, време учитавања странице се повећава.

На пример, ако имамо форму у *HTML*-у на којој рачунамо збир два броја, можемо на веома ефикасан начин да израчунамо уз помоћ *JavaSctipt*-а, тако што добијени резултат упишемо у поље предвиђено за збир. У случају *JSF*-а то није могуће урадити у *JavaSctipt*-у, већ морамо позвати *AJAX* који узима вредности са форме и прослеђује серверу на ком се обавља рачунање збира. Добијени збир је потребно приказати на форми тако што се на серверу мења вредност својства зрна који је атрибут *value* на компоненти. Касније се вредност компоненте збира освежи и на тај начин податак постане доступан кориснику. У случају да се форма поред ове три компоненте састоји још од неколико хиљада компоненти, рачунање збира може трајати неколико секунди, што је недопустиво са корисничке тачке гледишта.

Уколико нам изглед апликације није претерано битан и ако је акценат на брзој обради података, можда је потребно упоредити перформансе неке друге технологије са *JSF*-ом и на крају се определити за бржу. Наравно, треба узети у обзир и трајање развоја, што је велика предност *JSF* радног оквира у односу на остале технологије због доступности великог броја компоненти.

На крају, можемо закључити да су перформансе радног оквира *JSF* његова највећа мана.

7. Литература

1. David Geary, Cay Horstmann, *Core JavaServer Faces (3rd edition)*,Prentice Hall, Boston, 2010.
2. Apache Maven [На мрежи, датум последњег приступа 21.09.2016.] https://maven.apache.org/
3. Oracle Technology Network [На мрежи, датум последњег приступа 21.09.2016.] http://www.oracle.com/
4. PrimeFaces Showcase [На мрежи, датум последњег приступа 21.09.2016.] http://www.primefaces.org/showcase/index.xhtml
5. Mert Caliskan, Oleg Varaskin, *PrimeFaces Cookbook*, Packt Publishing, Birmingham, 2013.
6. Ivor Horton, *Ivor Horton's Beginning Java 2 JDK 5*, Wiley Publishing, Indianapolis, 2005.
7. PrimeFaces Extensions Showcase [На мрежи, датум последњег приступа 21.09.2016.] http://www.primefaces.org/showcase-ext/views/home.jsf
8. jQuery Introduction [На мрежи, датум последњег приступа 21.09.2016.] https://jquery.com/
9. CSS (*Cascading Style Sheets*) [На мрежи, датум последњег приступа 21.09.2016.] http://www.csstutorial.net/
10. FontAwesome [На мрежи, датум последњег приступа 21.09.2016.] http://fontawesome.io/icons/
11. HTML [На мрежи, датум последњег приступа 21.09.2016.] https://www.w3.org/TR/html5/
12. Chris Ullman, Lucinda Dykes, *Beginning Ajax*, Wiley Publishing, Indianapolis, 2007.
13. Фејслети [На мрежи, датум последњег приступа 21.09.2016.] http://docs.oracle.com/javaee/6/tutorial/doc/giepx.html
14. Ed Burns, Roger Kitain, *JavaServer Faces Specification*, Sun Microsystems, Santa Clara, 2009.